



# Desenvolvimento de sistemas

---

## Linguagem de programação

### A linguagem Java

A linguagem Java tem tido muito sucesso no mercado e diversas ferramentas estão surgindo para manipular ou gerar código Java. Um dos motivos da grande aceitação da linguagem Java é que a tornou tão atraente é o fato de que programas podem ser executados virtualmente em qualquer plataforma e ser aceitos em qualquer tipo de computador (ou outros aparelhos).

Os programadores não precisam se preocupar em saber em qual máquina o programa será executado, uma vez que um mesmo programa pode ser usado em um PC (computador pessoal), em um Mac ou em um computador de grande porte. Considerando isso, a maioria das linguagens de programação desenvolvidas recentemente também são multiplataforma.

A linguagem Java contém diversas características e, entre elas, destacam-se as seguintes:

### Orientação a objetos

É um paradigma de programação já sólido no mercado e a maioria das linguagens de programação atuais permite trabalhar dessa forma (para compreender melhor, leia ou releia o conteúdo sobre orientação a objeto desta unidade curricular).

### Portabilidade

Java é uma linguagem multiplataforma, isto é, uma mesma aplicação pode ser executada em diferentes tipos de plataforma sem a necessidade de adaptação de código.

## Suporte

Uma das vantagens do Java é fornecer um grande conjunto de classes com funcionalidades específicas, ou seja, muitos detalhes de programação são encapsulados em classes já prontas.

A seguir, conheça conceitos e estruturas básicas da linguagem Java.

Para melhor compreender este conteúdo, é essencial que você acompanhe o texto praticando os exemplos por meio de projetos no NetBeans. Execute-os e reflita sobre como o resultado obtido foi alcançado a partir do código proposto.

Primeiro contato com Java → (?page=2)

## Primeiro contato com Java

Inicialmente, para fornecer o primeiro contato com a linguagem, será apresentada uma classe em Java que escreve uma mensagem na tela. Apesar de o exemplo ser simples, ele contém os itens fundamentais para a criação de qualquer aplicação em Java: elaboração do código, compilação e execução. Esses itens serão seguidos durante o processo de elaboração das aplicações.

Confira agora o primeiro exemplo, cujos itens estão detalhados em seguida. Depois disso, você poderá testar esse exemplo, usando o IDE (*integrated development environment*, ou ambiente de desenvolvimento integrado) NetBeans.

```
public class PrimeiroPrograma {  
  
    public static void main(String[] args) {  
        System.out.println("Bem-vindo ao mundo JAVA");  
    }  
}
```

### public class PrimeiroPrograma

É a definição da classe. Como o Java é totalmente orientado a objeto, sua programação é feita dentro de classes. Você pode conferir a criação dessas classes em Java no conteúdo sobre a linguagem de programação orientada a objetos desta unidade curricular. O importante, a princípio, é entender que, para que o programa funcione, ele deve ter pelo menos uma classe com o método principal (**main**).

### public static void main

É o método principal do código. Quando se executa um programa, é esse método que é chamado. Em outras palavras, sem ele, não há programa.

(String[] args)

É o argumento passado ao método **main**. Qualquer programa pode ser iniciado ou não com argumentos que modificam sua execução. Argumentos com programas é um assunto que será tratado futuramente.

{ }

As chaves são usadas para marcar o bloco de comandos – ou escopo. No exemplo, elas marcam o código que será executado na classe **PrimeiroPrograma** e também todo o código que será executado quando o método **main()** for chamado. Chaves serão usadas em estruturas como condicionais (*if*), laços (*while*, *for*, *do-while*), métodos e outras que definam um escopo de código.

System.out.println("Bem vindo ao mundo Java!")

É o método de saída em texto de Java. Nesse caso, imprimirá “Bem-vindo ao mundo Java!” no *console*.

Toda linha de comando em Java termina com um ponto e vírgula (;).

Agora, codifique o exemplo demonstrado antes.

1. No IDE NetBeans, clique no menu **File**, no canto superior esquerdo e depois em **New Project**.
2. Selecione, à esquerda, na área **Categories**, a opção **Java with Ant**, e, à direita, selecione **Java Applicaton**. Feito isso, clique em **Next**.

3. Em **Project Name**, dê o nome ao projeto de “OlaMundo”.



4. **Project Location**: é a pasta na qual os seus projetos serão armazenados. O local padrão é em **Documentos > NetBeans Projects**, mas você pode alterar, caso queira.

5. Deixe marcadas as demais opções e clique em **Finish**.

6. A estrutura básica foi criada. Abra o arquivo “OlaMundo.java” (a classe principal) e copie o código que está descrito em seguida.

7. Antes de executar o código, limpe-o e o construa, para verificar se há ou não algum erro. Clique no menu **Run** e escolha a opção **Clean and Build Project**, depois novamente **Run** e a opção **Run Project**.

```
public class OlaMundo {  
  
    public static void main(String[] args) {  
        System.out.println("Bem-vindo ao mundo JAVA");  
    }  
}
```

A saída na tela, na aba **output** na região inferior da tela, deverá ser:



Depois dessa primeira experiência, explore estruturas e características da linguagem. Você provavelmente já estará familiarizado com vários dos conceitos que serão mencionados em seguida, pois eles já foram vistos em algoritmos (com a

linguagem Portugol). Agora, esses conceitos serão aplicados ao contexto da linguagem Java, utilizando, neste caso, apenas projetos do tipo *console*, ou seja, com entrada e saída textuais.

← A linguagem Java (?page=1) Variáveis → (?page=3)

## Variáveis

Qualquer informação a ser processada em um computador por um programa precisa ser previamente armazenada na memória. É possível executar essa ação quando se usa variáveis. Uma variável é uma região de memória, previamente identificada por um rótulo (nome), que tem por finalidade armazenar os dados de um programa temporariamente. Cada variável armazena apenas um valor por vez, e esse valor é um elemento qualificado a partir de um dos tipos de dados da linguagem.

O nome de uma variável é usado para sua identificação e posterior manipulação dentro do programa. Para usar variáveis, é preciso considerar algumas regras quanto aos nomes que podem ser definidos:

- ◆ O nome pode conter um ou mais caracteres, sendo que o primeiro caractere do nome de uma variável não pode ser um número. Assim, “1variavel” ou “578” seriam nomes não aceitos para variáveis em Java, mas sim “abc12”, por exemplo.
- ◆ O nome de uma variável não pode ter espaços em branco. O nome “minha variável”, por exemplo, geraria erro em Java.
- ◆ O nome não pode ser uma palavra reservada (conceito que você estudará logo em seguida).
- ◆ Além de letra e número, o único símbolo aceito é “\_”, que pode ser utilizado para simular a separação de duas palavras, como Nome\_Aluno (que também podem estar escritas como NomeAluno).

Em Java, para qualquer dado a ser armazenado em variável, primeiro é necessário saber o seu tipo para depois fazer o seu armazenamento. Depois de armazenado, o dado pode ser utilizado e manipulado a qualquer momento durante a execução do programa.

A definição de variáveis na linguagem Java obedece à seguinte sintaxe:

```
<tipo> NOME [= <valor>];
```

Nessa sintaxe, **tipo** é a definição do tipo de dado primitivo que uma variável pode assumir, **NOME** é o nome de identificação a ser definido para a variável e a definição de **valor** é uma operação opcional de atribuição, que poderá ou não definir um valor inicial para a variável.

Afirma-se que Java é uma linguagem **fortemente tipada** por exigir um tipo específico para suas variáveis. Assim, uma vez declarada a variável, só poderá receber dados do tipo especificado. Outras linguagens, como JavaScript, são chamadas de fracamente tipadas, pois suas variáveis podem receber qualquer tipo de valor a qualquer momento.

## Exemplo 1

Declare as variáveis para um programa que realizará a média aritmética de um aluno, com base em três notas.

Nesse exemplo, será necessário declarar variáveis para notas, para soma (caso queira, pois é possível calcular a média realizando diretamente a soma) e para média.

Portanto, você criará o seguinte:

double nota1;	Variáveis para obter a nota do aluno
double nota2;	
double nota3;	
double soma;	Variável que armazenará a soma, a qual será: $\text{nota1} + \text{nota2} + \text{nota3}$
double media;	Variável que armazenará a média, a qual será: $\text{soma} / 3$



### Quadro 1 – Declarando variáveis para calcular média de aluno

Fonte: Senac EAD (2022)

Tenha atenção ao declarar variáveis que dependam do valor de outras, como, por exemplo, a soma. Caso você inicialize a variável, sem os valores das notas, seu código dará erro, pois o compilador não identificou as notas.

Uma prática bem comum na programação é inicializar uma variável com valor 0, pois, como o nome já diz, é **variável**, então seu valor mudará dentro do programa.

## Exemplo 2

Declare as variáveis para descobrir a idade do usuário.

Nesse exemplo, precisa-se calcular a idade do usuário, com base no ano em que ele nasceu e no ano atual. Para isso, você declarará o seguinte:


<code>int anoAtual = 2022;</code>	Variável do tipo inteiro, com valor atribuído, que é o ano atual
<code>int anoNascimento</code>	Variável para obter o ano em que o usuário nasceu
<code>int idade</code>	Variável usada para calcular a idade do usuário, que será <code>anoAtual – anoNascimento</code>

### Quadro 2 – Declarando variáveis para calcular idade do usuário

Fonte: Senac EAD (2022)

Agora, codifique o exemplo mostrado. Crie um novo projeto no NetBeans chamado “Idade” e edite o arquivo “Idade.java”:

```
public class Idade {  
  
    public static void main(String[] args) {  
        int anoAtual = 2022;  
        int anoNasc = 1991;  
        int idade = anoAtual - anoNasc;  
  
        System.out.println("Você tem: " + idade);  
    }  
}
```

Execute usando o ícone  ou o menu **Run > Run Project**. O resultado exposto na aba **output**, na região inferior do NetBeans, deverá ser algo como o seguinte:

Você tem 31

Observe que, no código, foi utilizado um ano fixo (2022), ano de publicação deste material, para a variável “anoAtual”. Posteriormente, seria preciso alterar o código para o ano correspondente. É possível modificar esse código para que se torne flexível, capturando, por meio de classes de Java, o ano atual, independentemente de quando o programa está executando.

Modifique “Idade.java” para o seguinte:

```
import java.util.Calendar;
import java.util.GregorianCalendar;

public class Idade {

    public static void main(String[] args) {
        //trecho de código que fornecerá o ano atual
        Calendar cal = GregorianCalendar.getInstance();

        //o objeto "cal" fornecerá o ano atual por meio do método get(Calendar.YEA
R)
        int anoAtual = cal.get(Calendar.YEAR);

        //imagine que o usuário nasceu em 1991
        int anoNasc= 1991;

        int idade = anoAtual - anoNasc;

        System.out.println("Você tem: " + idade);
    }
}
```

Se você executar esse projeto em 2022, o resultado também será “Você tem 31”, mas, se executar em outros anos, a idade será outra. Isso porque se utiliza a classe **GregorianCalendar** para recuperar qual é o ano atual.

← Primeiro contato com Java (?page=2) Constantes → (?page=4)

# Constantes

Em Java, é possível nomear um **valor imutável** declarando uma constante. Como exemplo prático, pode-se considerar a constante matemática “pi”, que equivale ao valor aproximado 3.14159265. Assim, ao invés de usar diretamente o valor em várias partes do código, pode-se nomeá-lo e referenciar o valor por esse nome. É como uma variável preenchida que não pode ter valor alterado.

Para definir uma constante em Java, utilize esta sintaxe:

```
final <tipo> NOME = <valor>;
```

Uma constante é definida em Java com o qualificador **final**. **Tipo** é a definição do tipo de dado primitivo associado; a constante é identificada por **NOME**; e **valor** é o valor que se manterá constante.

Veja um exemplo de como declarar variáveis e constantes.

```
public class VariaveisEConstantes {  
    public static void main(String[] args) {  
  
        final float PI = 3.146f;  
        final double E = 1.234e2;  
        String nome = "João Pedro";  
        int lote = 75825;  
        boolean aprovado = true;  
  
        System.out.println(PI);  
        System.out.println(E);  
        System.out.println(nome);  
        System.out.println(lote);  
        System.out.println(aprovado);  
    }  
}
```

Depois de digitar o exemplo apresentado e de ele mostrar os valores declarados, faça o seguinte teste para verificar o erro ao tentar mudar o valor de uma constante:

- ◆ Abaixo da linha “final *double* E = 1.234e2;”, faça **E = 1.48;**
- ◆ Será mostrado o erro à esquerda: “não pode atribuir valor à variável final E”.

As constantes em Java podem ser de qualquer tipo primitivo.

← Variáveis (?page=3) Tipos de dados → (?page=5)

## Tipos de dados

Os tipos de dados são o que define a quantidade de memória do computador que será utilizado para guardar tal dado. Cada tipo de dado tem um tamanho diferente e, por consequência, seu alcance também. Isso significa que, se você declarar uma variável para guardar um número inteiro, jamais poderá guardar um número decimal, porque um número decimal ocupa mais espaço de memória do que um inteiro.

Para declarar qualquer variável, como visto anteriormente, utiliza-se primeiro o tipo de dado e depois o nome da variável.

## Tipos numéricos

Você iniciará com os tipos de dados **inteiros**, que são os mais simples. Inteiros são os números negativos ou positivos que não contêm casas decimais. Em Java, os números inteiros são divididos em quatro tipos: ***byte***, ***short***, ***int*** e ***long***. Esses quatro guardam o mesmo tipo de dado, mas ocupam espaços de memória diferentes, o que afeta seu alcance.

### *Byte*

O tipo *byte* é o de menor alcance entre os inteiros. Como o próprio nome sugere, ele consome apenas um *byte* (8 *bits*) e pode guardar valores entre -128 e 127. Exemplo:

```
byte variavelbyte;
```

### *Short*

O tipo *short* guarda inteiros de 2 *bytes* (16 *bits*) e pode guardar números entre -32.768 a 32.767. Exemplo:

```
short variavelShort;
```



## *Int*

O tipo *int* é o tipo de dado mais comum. Ele consome 4 *bytes* (32 *bits*) e guarda valores entre -2.147.483.648 e 2.147.483.647. Exemplo:

```
int numeroInteiro;
```

## *Long*

É o tipo de dado com maior alcance entre os inteiros. Consequentemente, também é o que ocupa mais espaço (8 *bytes* ou 64 *bits*). Tem um grande alcance que fica entre -9,22E+18 (exatos -9.223.372.036.854.775.808) e 9,22E+18 (exatos 9.223.372.036.854.775.807). Exemplo:

```
long numeroGrande;
```

Para melhor desempenho da aplicação, é importante usar o tipo com a precisão adequada. Por exemplo, imagine que você precise de uma variável para armazenar a idade de uma pessoa. Muito provavelmente, o tipo *byte* será suficiente, já que é (quase) impossível um ser humano ter mais de 127 anos.

Ao contrário de algumas linguagens de programação, Java não conta com tipos *unsigned*, ou seja, tipos que armazenam apenas números positivos.

Para **números decimais**, utilizam-se dois tipos de dados, dependendo da necessidade de seu alcance e da precisão (números após a vírgula).

## *Double*

É o tipo de dado mais complexo que há em Java e o maior valor possível de se armazenar é 1,797.693.134.862.315.7E+308, que é muito mais do que qualquer programador precisa. Exemplo:

```
double variavelDouble = 1.5;
```

## *Float*

É um ponto flutuante de 32 *bits* de precisão única. Recomenda-se usar um *float* (em vez de *double*) se você precisar economizar memória em grandes *arrays* (matrizes) de número de ponto flutuante.

Seu valor padrão é 0.0f. Ao expressar um valor *float*, é necessário sempre incluir um algarismo “f” à frente; caso contrário, o Java interpretará como sendo *double*. Exemplo:

```
float variavelFloat = 1.5f;  
float variavelFloat2 = 1.5; //esta declaração gera erro
```

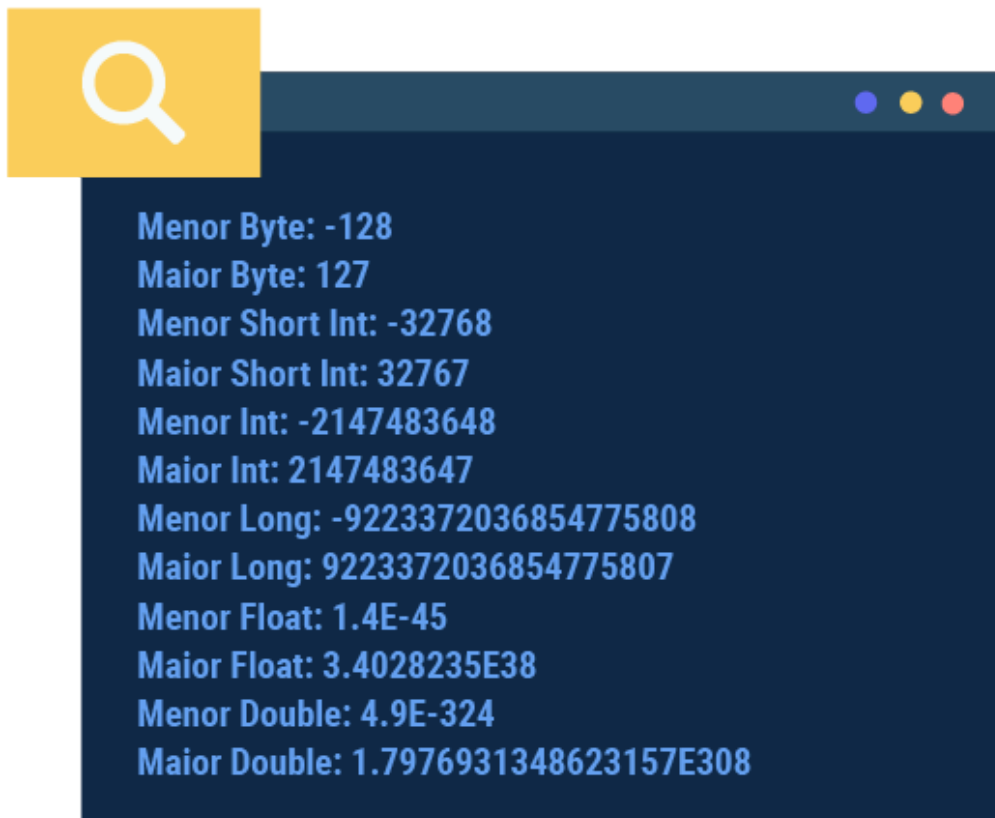
Para exemplificar melhor os tipos primitivos numéricos, será criado um projeto que mostrará o maior e o menor valor de cada tipo de dados.

Que tal codificar agora? Crie um novo projeto Java NetBeans chamado “TiposDeDados” e, na classe principal, digite o seguinte código:



```
public class TiposDeDados {  
  
    public static void main(String[] args) {  
        // "\n" é usado para realizar uma quebra de linha  
        System.out.println("Tipos de dados em Java: \n"  
            + "\nMenor Byte: " + Byte.MIN_VALUE+ "\nMaior Byte: " + Byte.MAX_VALUE  
            + "\nMenor Short Int: " + Short.MIN_VALUE + "\nMaior Short Int: " + Short.MAX  
            + "\nMenor Int: " + Integer.MIN_VALUE + "\nMaior Int: " + Integer.MAX_VALUE  
            + "\nMenor Long: " + Long.MIN_VALUE + "\nMaior Long: " + Long.MAX_VALUE  
            + "\nMenor Float: " + Float.MIN_VALUE + "\nMaior Float: " + Float.MAX_VALUE  
            + "\nMenor Double: " + Double.MIN_VALUE + "\nMaior Double: " + Double.MAX_VAL  
            UE);  
    }  
}
```

O resultado na aba **output** do NetBeans, depois dessa execução, será o seguinte:

A screenshot of the NetBeans IDE's Output window. It features a yellow tab with a magnifying glass icon. The window has a dark blue background with light blue text. The output shows the minimum and maximum values for various Java data types, with line breaks after each pair of values.

```
Menor Byte: -128  
Maior Byte: 127  
Menor Short Int: -32768  
Maior Short Int: 32767  
Menor Int: -2147483648  
Maior Int: 2147483647  
Menor Long: -9223372036854775808  
Maior Long: 9223372036854775807  
Menor Float: 1.4E-45  
Maior Float: 3.4028235E38  
Menor Double: 4.9E-324  
Maior Double: 1.7976931348623157E308
```

# Operadores em Java



O Java conta com operadores matemáticos, lógicos e de manipulação de *bits*. No contexto de numerais, será mantido o foco por enquanto nos seguintes operadores matemáticos:

## Operadores aditivos

### ◆ + (soma)

```
short a = 10, b = 10, c, d;  
c = a + b;  
c = 10 + 5;  
d = c + 8;
```

### ◆ - (subtração)

```
float x = 1.45f, y = 1f, z;  
z = x - y;  
System.out.println(z - 0.4f);
```

## Operadores multiplicativos

### ◆ \* (multiplicação)



```
int horasSemana, minutosSemana;  
horasSemana = 7 * 24;  
minutosSemana = horasSemana * 60;
```

### ◆ / (divisão)

```
double km = 200, litros = 30, desempenho;  
desempenho = km / litros;  
System.out.println(desempenho + "kms por litro");
```

### ◆ % (resto de divisão)

```
int dividendo = 5, divisor = 2;  
int resto = dividendo % divisor;  
System.out.println(resto);
```

## Operadores unários

### ◆ ++ (incrementa em um o valor da variável)



- ◆ Pós-fixado: primeiro usa o valor e depois o incrementa

```
int contador = 10;  
System.out.println(contador); //mostra 10  
System.out.println(contador++); //mostra 10 e incrementará depois  
de mostrar  
System.out.println(contador); //mostra 11
```

- ◆ Prefixado: primeiro incrementa e depois usa o valor

```
int contador = 10;  
System.out.println(contador); //mostra 10  
System.out.println(++contador); //incrementa e mostra 11  
System.out.println(contador); //mostra 11
```

### ◆ -- (decrementa em um o valor da variável; pós-fixado e prefixado)

```
int contador = 10;  
System.out.println(contador); //mostra 10  
System.out.println(contador--); //mostra 10, depois decrementa  
System.out.println(contador); //mostra 9  
System.out.println(--contador); //decrementa e mostra 8
```

## Operação com atribuição

- ◆ `+= -= *= /= %=` (operam o valor da variável à esquerda com o valor à direita, atribuindo o resultado à variável)



```
int valor = 5, valor2 = 5;
valor += 5; //corresponde a valor = valor + 5
System.out.println(valor); //10
valor *= 2; //corresponde a valor = valor * 2
System.out.println(valor); //20
valor /= valor2; //corresponde a valor = valor / valor2
System.out.println(valor); //4
valor -= 2; //corresponde a valor = valor - 2
System.out.println(valor); //2
```

Os operadores contêm **precedência**, isto é, quando estão em uma mesma expressão, são avaliados em uma prioridade específica. Por exemplo, considere a expressão a seguir para calcular a média de três notas:

```
float media = 9.5f + 7f + 9f / 3f;
System.out.println(media); //19.5
```

Esperava-se um resultado de 8,5 para essa média. No entanto, o que se obterá, na variável **media**, será o valor 3. Isso acontece porque **há precedência entre divisão e adição**, ou seja, os operadores multiplicadores são avaliados antes dos operadores aditivos. Assim, o Java fará primeiro a divisão  $9 / 3$  para depois somar o resultado (3) com 9.5 (12.5) e 7 (19.5).

A precedência entre os operadores matemáticos é a seguinte:

1. Operadores unários
2. Operadores multiplicativos
3. Operadores aditivos
4. Atribuições (com ou sem operação)

No exemplo da expressão para a média aritmética das notas, a solução seria aplicar parênteses, para priorizar uma operação sobre a outra, desta maneira:

```
Float media = (9.5f + 7f + 9f) / 3f;  
System.out.println(media); //19.5
```

Os parênteses garantem que toda a soma será realizada antes da divisão.

## Casting

Em alguns casos em Java, é possível converter um valor de um tipo para outro. Isso se chama *cast* ou *casting*. A sintaxe dessa conversão é a seguinte:

```
variavel_TipoA = (TipoA) variavel_TipoB;
```

Por exemplo, considere uma variável *double* que precisa ser convertida para inteiro. Pode-se, então, aplicar a seguinte conversão:

```
double var_dbl = 10.5;  
int var_int = (int) var_dbl; //10 (perde o .5)
```

Nesse caso, haverá a perda da informação dos valores decimais que *int* não suporta. É importante notar que alguns tipos não suportam *cast* direto entre si, como é o caso de se converter uma *string* para *int*.

## Tipo lógico *boolean*

A linguagem Java conta com o tipo ***boolean*** para armazenar os valores lógicos ***true*** (verdadeiro) e ***false*** (falso). Uma variável *boolean* não pode armazenar nenhum valor diferente desses. Ela será útil em seguida, quando forem abordados os condicionais e as expressões lógicas.

```
boolean estouAprendendoJava = true;  
boolean sereiBomSeNaoPraticar = false;
```

```
System.out.println(estouAprendendoJava); // Mostra true
System.out.println(sereiBomSeNaoPraticar); // Mostra false
boolean resultado = 10 > 5;
System.out.println(resultado); //Mostra true
```

## Caracteres e o tipo *string*

Para textos, é possível trabalhar com caracteres individuais ou cadeias de caracteres.

Existe o tipo de dados próprio para **caracteres**, que é o ***char***. O tipo *char* ocupa 2 *bytes*, permitindo o uso do formato de caracteres Unicode, padrão que permite uma variedade de caracteres especiais e com acentuação. Note, entretanto, que uma variável *char* armazena exatamente um caractere. Exemplo:

```
char varCaractere = 'A';
```

Para palavras ou textos inteiros, o Java conta com o tipo ***string***, que na verdade é uma classe. Um valor *string* deve ser sempre envolto em aspas duplas ("). Exemplo:

```
String umaFrase = "O rato roeu a roupa do rei de Roma";
String umaPalavra = "Programação!";
```

Nos exemplos anteriores, foi utilizada *string* como argumento ao método **System.out.println()** para mostrar mensagens na tela.

O operador + pode ser aplicado à *string* para concatenar (juntar) um texto a outro. Veja o exemplo:

```
String mensagem = "Aprendendo ";
mensagem = mensagem + "a programar "; //variavel mensagem terá "Aprendendo a programar "
System.out.println(mensagem + "com Java"); //mostrará na tela "Aprendendo a programar com Java"
```

*String* é uma classe e, por isso, conta com métodos, ou seja, pode-se operar e manipular um texto utilizando comportamentos de *string*. Os principais métodos são:

Método	Uso	Exemplo
<b>length()</b>	Retorna a quantidade de caracteres em um texto <i>string</i> .	<pre>String mensagem = "Aprendendo "; int tamanho = <b>mensagem.length()</b>; System.out.println(tamanho); // 11</pre>
<b>contains()</b>	Verifica se a <i>string</i> contém um texto informado.	<pre>String mensagem = "Aprendendo "; System.out.println(mensagem.contains("Aprende"));</pre>
<b>indexOf()</b>	Retorna o índice do caractere no qual inicia um texto informado (ou -1, se não existe).	<pre>String mensagem = "Aprendendo "; System.out.println(mensagem.indexOf("prende")); //1</pre>
<b>toLowerCase()</b>	Retorna a <i>string</i> com caracteres minúsculos.	<pre>String mensagem = "Aprendendo "; mensagem = mensagem.toLowerCase(); System.out.println(mensagem); //aprendendo</pre>
<b>toUpperCase()</b>	Retorna a <i>string</i> com caracteres maiúsculos.	<pre>String mensagem = "Aprendendo "; mensagem = mensagem.toUpperCase(); System.out.println(mensagem);//APRENDENDO</pre>
<b>isEmpty()</b>	Verifica se a <i>string</i> está vazia.	<pre>String mensagem = "Aprendendo "; System.out.println(mensagem.isEmpty()); //false</pre>
<b>substring()</b>	Retorna uma porção de uma <i>string</i> . Deve-se informar o índice de início e de fim da extração.	<pre>String mensagem = "Aprendendo "; System.out.println(mensagem.substring(1, 7)); //prende</pre>



### Quadro 3 – Principais métodos de *string*

Fonte: Senac EAD (2022)

*String* e *char* contam com alguns caracteres especiais chamados “sequências de escape”. As principais são as seguintes: `\t` (tab), `\n` (quebra de linha), `\'` (aspa simples no texto), `\"` (aspa dupla no texto), `\\` (barra inversa no texto).

Exemplo:

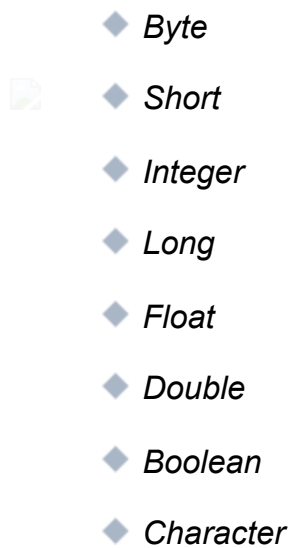
```
System.out.println("linha 1 de texto \n linha 2 de texto");
```

Uma característica do Java é a disponibilização de classes correspondentes aos tipos primitivos, as quais podem ser utilizadas da mesma maneira que os tipos primitivos. Elas são chamadas de **classes wrapper**.

## Tipo primitivo

- ◆ *byte*
- ◆ *short*
- ◆ *int*
- ◆ *long*
- ◆ *float*
- ◆ *double*
- ◆ *boolean*
- ◆ *char*

## Classe wrapper



O uso das classe **wrapper** é similar ao uso do tipo primitivo. Por exemplo, é possível declarar uma variável inteira da seguinte forma:

```
Integer variavel = 10;  
System.out.println(variavel + 5);
```

Essas classes podem ser necessárias no uso de *generics* (para saber mais, consulte o conteúdo sobre a estrutura de dados, desta unidade curricular). Também podem ser usadas em situações muito específicas, nas quais é necessário atribuir valor nulo à variável:

```
int num1 = null; //erro  
Integer num2 = null; //válido
```

## Praticando com variáveis

Confira a seguir alguns exemplos para praticar o uso de variáveis, tipos e operações. Crie um projeto NetBeans para cada exemplo ou substitua o código do método **main()** de um mesmo projeto. Por simplicidade, será incluído apenas o código do método **main()**. Depois de transcrever os códigos, execute-os e verifique seu resultado.

### Exemplo 1

Calcule o número de meses que uma pessoa de 30 anos viveu.

```
public static void main(String[] args) {  
    int idade = 30, meses;  
    meses = idade * 12;  
    System.out.println("A pessoa viveu " + meses + " meses");  
}
```

Nesse código muito simples, você está usando variáveis de número inteiro para armazenar a idade e a quantidade de meses. Note que a concatenação de *string* (com `+`) permite o uso de variáveis de qualquer tipo primitivo.

## Exemplo 2

Sabendo que a conversão de uma temperatura (T) de Celsius para Farenheit é calculada por  $(T^{\circ}\text{C} \times 1,8) + 32$  e que a conversão para Kelvin se dá por  $T^{\circ}\text{C} + 273,15$ , monte a aplicação Java que converta 20 graus Celsius em Farenheit e Kelvin.

```
public static void main(String[] args) {  
    float celsius = 20, fahrenheit, kelvin;  
  
    fahrenheit = celsius * 1.8f + 32;  
    kelvin = celsius + 273.15f;  
    System.out.println(celsius + "°C correspondem a " + fahrenheit + "F e " + kelvin + "K.");  
}
```

Você poderia usar aqui variáveis *double*, mas, como serão trabalhados neste momento valores baixos, o mais recomendável é *float*. Note ainda que, apesar de o enunciado apresentar a fórmula de Farenheit com parênteses, no código isso não é necessário, por causa da precedência da operação de multiplicação.

## Exemplo 3



Calcule a área de um círculo ( $A=\pi r^2$ ) com raio de 10 cm.

```
public static void main(String[] args) {  
    final float pi = 3.1416f;  
    float raio = 10;  
    float area = pi * raio * raio;  
  
    System.out.println("A área do círculo é " + area + "cm");  
}
```

No exemplo, foi aplicada uma constante para o valor de pi. O restante é composto de cálculos simples.

## Exemplo 4

A partir do nome de uma pessoa, verifique o tamanho do nome completo e extraia o primeiro nome, ignorando os sobrenomes.

```
public static void main(String[] args) {  
    String nome = "João da Silva", primeiroNome;  
    int fimNome, tamanhoNome;  
  
    //tamanho do nome completo  
    tamanhoNome = nome.length();  
    //encontra o índice no qual ocorre o primeiro caractere de espaço  
    fimNome = nome.indexOf(" ");  
    //extraí o primeiro nome  
    primeiroNome = nome.substring(0, fimNome);  
  
    //mostra na tela o resultado  
    System.out.println("Tamanho total: " + tamanhoNome + " caracteres;\nPrimeiro nome: " + primeiroNome);  
}
```

Foram aplicadas aqui as funções **length()** (número de caracteres) para o tamanho do nome completo, **indexOf()** (localização de um texto) e **substring()** (extração de trecho da *string*) para extrair o primeiro nome – o subtexto que permeia do início da *string* até a posição na qual está o primeiro caractere de espaço.

← Constantes (?page=4) Classe scanner → (?page=6)

## Classe scanner

Até agora, nas aplicações do tipo *console* em Java, aplicou-se diversas vezes a saída de dados, usando o método **System.out.println()**, mas até agora nenhuma entrada de dados. Ao se trabalhar com esse tipo de programa, em modo de texto, a alternativa mais adequada para entradas é o uso da classe **Scanner**, pois tem justamente a finalidade de facilitar a **entrada de dados** no modo *console*. Essa classe apareceu a partir do Java 5 e será muito útil, pois, nos exemplos a seguir, será possível digitar o dado que você quiser, e não somente atribuir valor a uma variável.

Antes de tudo, é necessário conhecer algumas funções e alguns aspectos que essa classe tem para exercer o funcionamento dentro do esperado. Quando invocada a classe **Scanner**, o compilador pedirá para fazer a seguinte importação:

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

*Import* significa, basicamente, que o código necessita de uma classe externa, que está em outro pacote ou na biblioteca.

Em resumo, essa classe ajuda na leitura dos dados informados. Para realizar essa ação na prática, é necessário criar um objeto do tipo “Scanner”, que passa como argumento o objeto **System.in** dentro construtor, por exemplo:

```
Scanner entrada = new Scanner (System.in);
```

Realize um exemplo prático agora. No NetBeans, crie um projeto chamado “EntradaDeDados” e, na classe principal (**EntradaDeDados.java**), digite o código a seguir:

```
import java.util.Scanner;

public class EntradaDeDados {

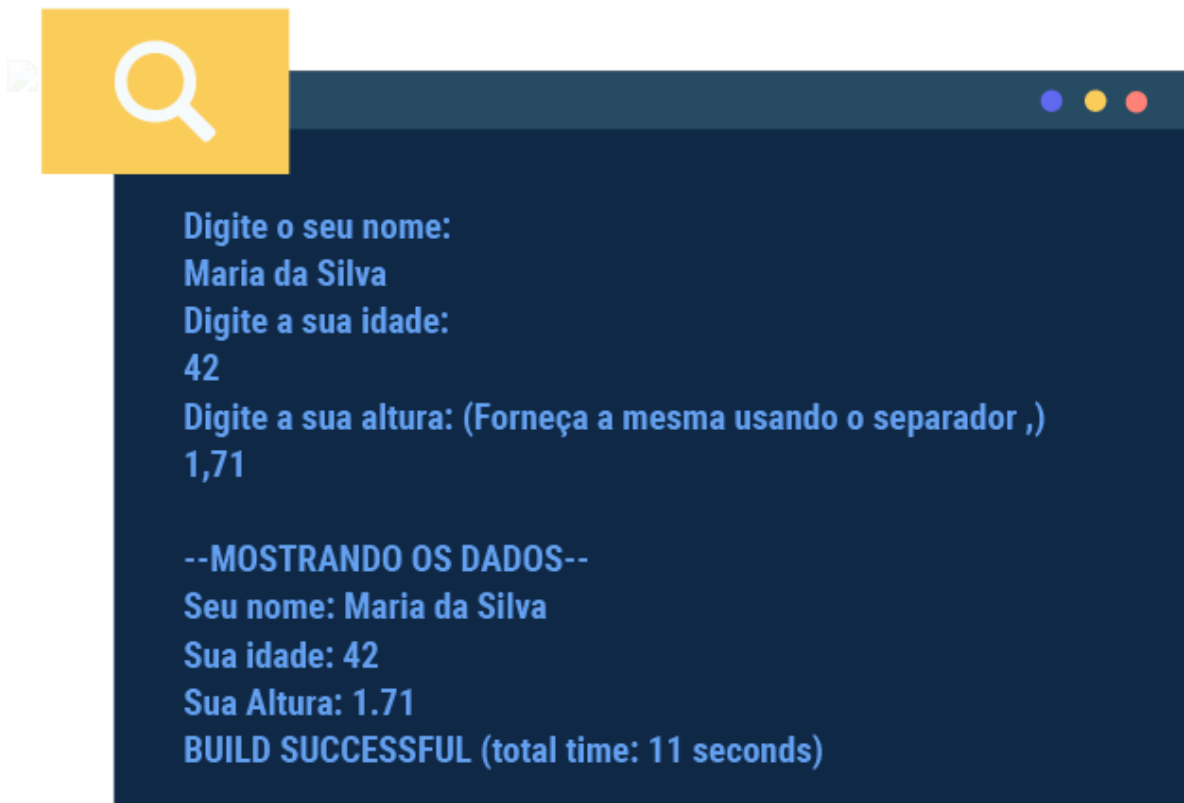
    public static void main(String[] args) {
        String nome;
        int idade;
        float altura;

        //declaração do scanner
        //A palavra "entrada" será o objeto criado, fique à vontade para trocá-la por outra
        //Não se esqueça de importar o pacote java.util.Scanner, clicando no símbolo de erro na linha
        Scanner entrada = new Scanner (System.in);

        //solicitando os dados para o usuário e armazenando nas variáveis criadas
        System.out.println("Digite o seu nome:");
        nome = entrada.nextLine();
        System.out.println("Digite a sua idade:");
        idade = entrada.nextInt();
        System.out.println("Digite a sua altura: (Forneça a mesma usando o separador ,)");
        altura = entrada.nextFloat();
        entrada.close(); //fechando o buffer de entrada

        //mostrando os dados na tela
        System.out.println("\n--MOSTRANDO OS DADOS--");
        System.out.println("Seu nome: " + nome);
        System.out.println("Sua idade: " + idade);
        System.out.println("Sua Altura: " + altura);
    }
}
```

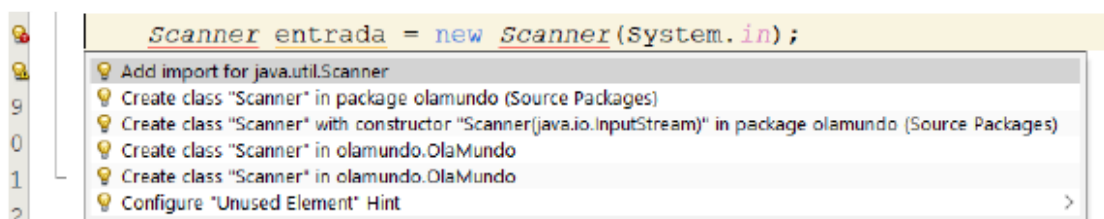
O resultado será algo como o descrito a seguir (você pode preencher com os dados que desejar):



A seguir, você aplicará **Scanner** para a entrada de dados aos exemplos expostos na seção anterior. Não se esqueça de que, em todos eles, será necessário incluir **import java.util.Scanner;** no topo do arquivo.

O NetBeans conta com uma variedade de ferramentas de produtividade. Uma delas é sugerir a importação de bibliotecas de acordo com o código digitado. Ao aplicar **Scanner**, por exemplo, pode surgir o ícone , à esquerda. Ele indica que o IDE tem sugestões para algum problema no código – no caso, a ausência do *import* para **Scanner**.

Ao clicar no ícone, pode-se selecionar uma das opções.





## Figura 1 – Sugestões do NetBeans

Fonte: Senac EAD (2022)

Ao clicar na primeira opção (“adicionar import para java.util.Scanner”), o IDE já incluirá o *import* automaticamente no topo do arquivo.

## Exemplo 1

Dada uma idade, calcule o número de meses que a pessoa viveu.

```
public static void main(String[] args) {  
    Scanner entrada = new Scanner(System.in);  
    int idade, meses;  
  
    System.out.print("Digite a idade: ");  
    //lê um número inteiro do console  
    idade = entrada.nextInt();  
  
    meses = idade * 12;  
    System.out.println("A pessoa viveu " + meses + " meses");  
}
```

## Exemplo 2

Considerando que a conversão de uma temperatura (T) de Celsius para Farenheit é calculada por  $(T^{\circ}\text{C} \times 1,8) + 32$  e que a conversão para Kelvin se dá por  $T^{\circ}\text{C} + 273,15$ , monte a aplicação Java que converta graus Celsius em Farenheit e Kelvin. O usuário informará a temperatura em Celsius.

```
public static void main(String[] args) {  
    Scanner entrada = new Scanner(System.in);  
    float celsius, fahrenheit, kelvin;  
  
    System.out.print("Informe a temperatura em °C: ");  
    celsius = entrada.nextFloat();  
  
    fahrenheit = celsius * 1.8f + 32;  
    kelvin = celsius + 273.15f;  
    System.out.println(celsius + "°C correspondem a " + fahrenheit + "F e " + kelvin  
+ "K.");  
}
```

É importante ressaltar que, apesar de, no código, ser preciso expressar números com casas decimais sempre separando-os com ponto, ao informar via *console* pode ser necessário separar com vírgula a porção decimal. Isso ocorre de acordo com a configuração de localização da máquina.

## Exemplo 3

Dado o raio de um círculo, calcule sua área ( $A=\pi r^2$ ).

```
public static void main(String[] args) {  
    Scanner entrada = new Scanner(System.in);  
    final float pi = 3.1416f;  
    float raio;  
  
    raio = entrada.nextFloat();  
  
    float area = pi * raio * raio;  
  
    System.out.println("A área do círculo é " + area + "cm");  
}
```

## Exemplo 4

A partir do nome de uma pessoa informado pelo usuário, verifique o tamanho do nome completo e extraia o primeiro nome, ignorando os sobrenomes.

```
public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in, "latin1");
    String nome, primeiroNome;
    int fimNome, tamanhoNome;

    System.out.print("Informe nome e sobrenome: ");
    nome = entrada.nextLine();

    //tamanho do nome completo
    tamanhoNome = nome.length();
    //encontra o índice no qual ocorre o primeiro caractere de espaço
    fimNome = nome.indexOf(" ");
    //extraí o primeiro nome
    primeiroNome = nome.substring(0, fimNome);

    //mostra na tela o resultado
    System.out.println("Tamanho total: " + tamanhoNome + " caracteres;\nPrimeiro nome: " + primeiroNome);
}
```

Há alguns pontos importantes nesse exemplo. O primeiro deles é o uso de um argumento novo em **new Scanner()**.

```
new Scanner(System.in, "latin1");
```

O segundo argumento indica a codificação de caracteres que Java deve considerar na entrada de dados. Nesse caso, o valor **“latin1”** permite que o usuário informe livremente **caracteres com acentuação**.

Outro ponto importante no exemplo é que, para entrada de dados de texto, utiliza-se o método **nextLine()**. Esse é o método padrão para capturar textos e, basicamente, obtém todos os caracteres digitados até que o usuário tecla **Enter**.

Veja ainda que, caso você informe um nome sem o caractere de espaço, o programa resultará em erro. Isso ocorre porque **indexOf()** retornará -1 (não encontrado) e **substring()** usará -1 como limite para a extração do nome, o que é inválido. Esse erro

pode ser facilmente contornado utilizando uma estrutura condicional, o que será visto em seguida.

Escreva um programa que receba dois números e ao final mostre a soma, subtração, multiplicação e divisão dos números lidos.

← Tipos de dados (?page=5) Palavras reservadas em Java → (?page=7)

# Palavras reservadas em Java

Em Java, existem algumas palavras reservadas, mas isso significa que elas não devem ser usadas?

Palavras reservadas da linguagem são palavras que não podem ser usadas como identificadores, ou seja, não podem ser usadas para representar variáveis, classes ou nomes de métodos.

Veja então quais são essas palavras reservadas, organizadas dentro de sua categoria, conforme Barros (2016):

Muitas dessas palavras serão usadas posteriormente no decorrer deste estudo de Java. Não se preocupe em memorizar a funcionalidade de todas elas.

## Modificadores de acesso

- ◆ *private*: concede acesso apenas dentro da classe.
- ◆ *protected*: concede acesso por classes no mesmo pacote e na subclasse.
- ◆ *public*: concede acesso de qualquer classe.

## Modificadores de classes, variáveis ou métodos




- ◆ *abstract*: classe que não pode ser instanciada ou método que precisa ser implementado por uma subclasse não abstrata.
- ◆ *class*: especifica uma classe.
- ◆ *extends*: indica a superclasse que a subclasse está estendendo.
- ◆ *final*: impossibilita que uma classe seja estendida, que um método seja sobrescrito ou que uma variável seja reinicializada.
- ◆ *implements*: indica as interfaces que uma classe implementará.
- ◆ *interface*: especifica uma interface.
- ◆ *native*: indica que um método está escrito em uma linguagem dependente de plataforma, como o C.
- ◆ *new*: instancia um novo objeto, chamando seu construtor.
- ◆ *static*: faz um método ou variável pertencer à classe ao invés de às instâncias.
- ◆ *strictfp*: é usada em frente a um método ou uma classe para indicar que os números de ponto flutuante seguirão as regras de ponto flutuante em todas as expressões.
- ◆ *synchronized*: indica que um método só pode ser acessado por uma *thread* de cada vez.
- ◆ *transiente*: impede a serialização de campos.
- ◆ *volatile*: indica que uma variável pode ser alterada durante o uso de *threads*.

## Controle de fluxo dentro de um bloco de código

- ◆ *break*: sai do bloco de código em que ele está.
- ◆ *case*: executa um bloco de código, dependendo do teste do *switch*.
- ◆ *continue*: pula a execução do código que viria após essa linha e vai para a próxima passagem do *loop*.
- ◆ *default*: executa esse bloco de código caso nenhum dos testes de *switch-case* seja verdadeiro.
- ◆ *do*: executa um bloco de código uma vez, e então realiza um teste em conjunto com o *while* para determinar se o bloco deverá ser executado novamente.
- ◆ *else*: executa um bloco de código alternativo, caso o teste *if* seja falso.
- ◆ *for*: é usada para realizar um *loop* condicional de um bloco de código.
- ◆ *if*: é usada para realizar um teste lógico de verdadeiro ou falso.
- ◆ *instanceof*: determina se um objeto é uma instância de determinada classe, superclasse ou interface.
- ◆ *return*: retorna de um método sem executar qualquer código que venha depois desta linha (também pode retornar uma variável).
- ◆ *switch*: indica a variável a ser comparada nas expressões *case*.
- ◆ *while*: executa um bloco de código repetidamente enquanto a condição for verdadeira

## Tratamento de erros

- 
- ◆ *assert*: testa uma expressão condicional para verificar uma suposição do programador.
  - ◆ *catch*: declara o bloco de código usado para tratar uma exceção.
  - ◆ *finally*: bloco de código, após um *try-catch*, que é executado independentemente do fluxo de programa seguido ao lidar com uma exceção.
  - ◆ *throw*: é usada para passar uma exceção para o método que o chamou.
  - ◆ *throws*: indica que um método pode passar uma exceção para o método que o chamou.
  - ◆ *try*: bloco de código que tentará ser executado, mas que pode causar uma exceção.

## Controle de pacotes

- ◆ *import*: importa pacotes ou classes para dentro do código.
- ◆ *package*: especifica a que pacote todas as classes de um arquivo pertencem.

## Primitivos

- ◆ *boolean*: um valor indicando verdadeiro ou falso.
- ◆ *byte*: um inteiro de 8 *bits*.
- ◆ *char*: um caractere Unicode.
- ◆ *double*: um número de ponto flutuante de 64 *bits*.
- ◆ *float*: um número de ponto flutuante de 32 *bits*.
- ◆ *int*: um inteiro de 32 *bits*.
- ◆ *long*: um inteiro de 64 *bits*.
- ◆ *short*: um inteiro de 16 *bits*.



## Variáveis de referência



- ◆ *super*: refere-se à superclasse imediata.
- ◆ *this*: refere-se à instância atual do objeto.

## Retorno de um método

- ◆ *void*: indica que o método não tem retorno.

← Classe scanner (?page=6) Estruturas condicionais → (?page=8)

# Estruturas condicionais

Para um programa de computador tomar decisões, é necessário criar para ele uma condição, que pode produzir uma de duas respostas: pode ser uma resposta verdadeira (*true*) ou uma resposta falsa (*false*).

Os elementos de uma condição podem ser variáveis ou valores constantes e a relação lógica existente entre esses elementos é conseguida com a utilização dos operadores relacionais que, em Java, são os seguintes:

Operador	Operador em Java	Significado
=	==	Igual
≠	!=	Diferente
<	<	Menor
≤	<=	Menor ou igual
>	>	Maior
≥	>=	Maior ou igual
E	&&	E
OU		OU
NÃO	!	Negação

Quadro 4 – Operadores relacionais em Java  
Fonte: Senac EAD (2022)

Esses operadores funcionam de maneira bastante análoga ao que se aprendeu em “Algoritmos”. A seguir, confira alguns exemplos de expressões em Java.

```
int num1 = 10, num2 = 15, num3 = 20;
System.out.println( num1 > num2 ); //false
System.out.println( num1 != num2 ); //true
//operador NÃO
System.out.println(!(num1 < num2)); //false
//operador E
System.out.println( num1 < num2 && num2 < num3); //true
//operador OU
System.out.println( num1 >= num2 || num1 >= num3); //false
//expressão extensa
System.out.println( (num1 > num2 && num2 > num3) || (num2 +10 > num3 ) ); //true
```

Um ponto importante a se notar é com relação à classe *string*. Por ser um objeto e não um tipo primitivo, **a comparação entre dois valores *string* não se dá pelo operador ==, mas sim por meio do método equals().**

Veja a seguir um exemplo:

```
String valorA = "Este é um texto";
String valorB = "Este é outro texto";
String valorC = new String("Este é outro texto");
//comparando valorA e valorB
System.out.println(valorA.equals(valorB)); //false
//comparando valorA com um texto equivalente
System.out.println(valorA.equals("Este é um texto")); //true
//comparando valores dos objetos
System.out.println(valorB.equals(valorC)); //true
//comparando objetos, e não valores
System.out.println(valorB == valorC); //false
```

Note que a última comparação retorna *false* mesmo tendo valores idênticos. Isso acontece porque com == estão sendo comparadas referências de objetos e não valores. Por isso, o resultado é *false*, uma vez que, para “valorC”, um novo objeto *string* está sendo instanciado explicitamente. Apesar de, em alguns casos, o operador == retornar o resultado esperado, isso acontece em casos específicos e, por isso, de modo geral, recomenda-se que, ao comparar dois valores *string*, use-se **equals()** para evitar problemas.

## Desvio condicional simples



Um desvio condicional será simples quando se tiver uma condição que desvia a execução do programa, caso o resultado lógico avaliado de certa condição seja verdadeiro. Se o resultado lógico avaliado for falso, nada acontecerá, e o programa simplesmente seguirá o seu fluxo de execução (as instruções após o condicional).

Neste caso, pode-se fazer uma comparação: no Portugol, utiliza-se “se”; em Java, será utilizado *if*, que contém a seguinte sintaxe:

```
if <(condição)> {  
    <instrução 1 executada quando condição for verdadeira>;  
    <instrução 2 executada quando condição for verdadeira>;  
    <instrução N executada quando condição for verdadeira>;  
}  
<instrução executada após condição ser verdadeira ou falsa>;
```

## Exemplo 1

Elabore um programa que leia dois valores numéricos inteiros, some esses valores e apresente o resultado, caso o valor somado seja maior que 10.

## Portugol

```
funcao inicio()
{
    inteiro a, b, resultado

    escreva("Entre com o valor <A>:")
    leia(a)
    escreva("Entre com o valor <B>:")
    leia(b)

    resultado = a + b

    se(resultado > 10){
        escreva("Resultado = ", resultado)
    }

    escreva("\nFim do programa")
}
```

## Java

```
import java.util.Scanner;

public class Exemplo01 {

    public static void main(String[] args) {

        int a, b, resultado;
        Scanner input = new Scanner(System.in);

        System.out.print("Entre o valor <A>: ");
        a = input.nextInt();
        System.out.print("Entre o valor <B>: ");
        b = input.nextInt();
        resultado = a + b;

        if (resultado > 10) {
            System.out.println("Resultado = " + resultado);
        }
        System.out.println("Fim do programa");
    }
}
```

Só serão mostrados na tela os valores somados que sejam maior que 10, pois a cláusula do *if* diz: se o valor do resultado ( $a + b$ ) for maior que 10, imprima na tela "Resultado = " + resultado. A instrução final **System.out.println("Fim do programa");** é executada independentemente de o programa entrar no *if* ou não, pois ele está fora da estrutura condicional.

## Exemplo 2

Neste exemplo, será criado um programa para verificar se uma pessoa é maior de idade, usando somente o *if*, então não haverá resultado caso o usuário seja menor de idade.

## Portugol

```
funcao inicio() {  
    inteiro idade  
    cadeia nome  
  
    escreva("\nDigite o seu nome: ")  
    leia(nome)  
  
    escreva("\nDigite a sua idade: ")  
    leia(idade)  
  
    limpa()  
  
    se (idade >= 18)  
    {  
        escreva(nome, ", sua idade é ", idade, " anos e você é maior de idade! ")  
    }  
}
```

## Java

```
import java.util.Scanner;

public class ExemploIf {

    public static void main(String[] args) {
        int idade;
        String nome;

        Scanner entrada = new Scanner (System.in);
        System.out.println("Digite o seu nome:");
        nome = entrada.nextLine();
        System.out.println("Digite sua idade:");
        idade = entrada.nextInt();

        if (idade >= 18)
        {
            System.out.println(nome + ", sua idade é " + idade + " anos e você é maior de idade!");
        }
    }
}
```

## Desvio condicional composto

Você aprendeu anteriormente como usar a instrução *if* simples para avaliar determinada condição (desvio condicional simples). Agora, será explicado como usar a instrução *if... else*. Em uma instrução *if... else*, se a condição for verdadeira, serão executados os comandos posicionados entre as instruções *if* e *else*. Se a condição for falsa, será executada a instrução posicionada logo após *else*, como é indicado na sintaxe a seguir:

```
if <(condição)> {
    <instrução 1 executada quando condição for verdadeira>;
    <instrução 2 executada quando condição for verdadeira>;
    <instrução N executada quando condição for verdadeira>;
}
else {
    <instrução 1 executada quando condição for falsa>;
    <instrução 2 executada quando condição for falsa>;
    <instrução N executada quando condição for falsa>;
}
```

```
}  
<instrução executada após condição ser verdadeira ou falsa>;
```



Neste momento é possível fazer outra comparação: no Portugol, utilizava-se “**senao**”; em Java, utiliza-se “**else**”.

## Exemplo 1

Elabore um programa que faça a leitura de dois valores numéricos do tipo inteiro e efetue a adição desses valores. Caso o resultado obtido seja maior que 10, o programa apresentará a mensagem “Resultado ultrapassa o valor dez.”. Caso o resultado do valor somado não seja maior que 10, o programa deverá apresentar a mensagem “Resultado não ultrapassa o valor dez.”.

## Portugol

```
funcao inicio()  
{  
    inteiro a, b, resultado  
  
    escreva("Entre com o valor <A>:")  
    leia(a)  
    escreva("Entre com o valor <B>:")  
    leia(b)  
  
    resultado = a + b  
  
    se(resultado > 10){  
        escreva("A soma dos números é ", resultado, "que ultrapassa o valor dez")  
    }  
    senao{  
        escreva("A soma dos números é ", resultado, "que não ultrapassa o valor dez")  
    }  
  
    escreva("\nFim do programa")  
}
```

## Java



```
import java.util.Scanner;

public class Exemplo02 {

    public static void main(String[] args) {

        int a, b, resultado;
        Scanner entrada = new Scanner(System.in);
        System.out.println();
        System.out.print("Entre o valor <A>: ");
        a = entrada.nextInt();
        System.out.print("Entre o valor <B>: ");
        b = entrada.nextInt();
        resultado = a + b;
        System.out.println();
        if (resultado > 10) {
            System.out.println(" A soma dos números é: " + resultado + " ,que ultrapassa o
valor dez.");
        } else {
            System.out.println(" A soma dos números é: " + resultado + " ,que não ultrapasa o valor dez.");
        }
        System.out.println("Fim do programa");
    }
}
```

Observe que esse exemplo é um complemento do exemplo 1, do desvio condicional simples. Porém, agora, se o resultado da soma for menor que 10, o programa lerá a cláusula *else* e mostrará na tela a mensagem “Resultado não ultrapassa o valor dez”.

A chave (*{*) pode estar à frente de *if* ou *else* ou na linha seguinte. Isso dependerá da preferência do programador.

## Exemplo 2

Crie um programa que realiza uma autenticação de um usuário, permitindo acesso caso a senha seja o mês de aniversário ou a idade dele.

# Portugol



```
funcao inicio() {
    inteiro idade, senha, mes
    cadeia nome

    escreva ("----SOLICITANDO DADOS DO USUÁRIO----\n")
    escreva ("Digite o seu nome: ")
    leia (nome)

    escreva ("Digite a sua idade: ")
    leia (idade)

    escreva ("Digite o número correspondente ao mês que você nasceu: ")
    leia (mes)
    limpa()

    escreva ("--- Acesso ao sistema ---\n")
    escreva ("Digite a sua senha: \n")
    leia (senha)

    limpa()

    se ( senha == idade ou senha == mes)
    {
        escreva ("Olá ", nome, ", acesso permitido!\n")
    }
    senao
    {
        escreva ("Acesso negado!\n")
    }
}
```

# Java

```
import java.util.Scanner;

public class AutenticacaoUsuario {

    public static void main(String[] args) {

        int idade, senha, mes;
        String nome;

        Scanner entrada = new Scanner (System.in);

        System.out.println("--Solicitando dados ao usuário--");
        System.out.println("Digite seu nome:");
        nome = entrada.nextLine();
        System.out.println("Digite a sua idade:");
        idade = entrada.nextInt();
        System.out.println("Digite o número correspondente ao mês que você nasceu:");
        mes = entrada.nextInt();

        System.out.println("\n---Acesso ao sistema---");
        System.out.println("Digite a sua senha:");
        senha = entrada.nextInt();

        if (senha == idade || senha == mes) {
            System.out.println("Olá " + nome + ", acesso permitido!");
        } else {
            System.out.println("Acesso negado!!");
        }
    }
}
```

Assim como em Portugol e em outras linguagens de programação, com o Java, é possível aninhar condicionais, ou seja, incluir um *if* no escopo de outro *if*.

## Exemplo 3

Receber duas notas e calcular a média. Se ela for superior a 7, mostrar “Aprovado”; se estiver entre 5 e 7, mostrar “Recuperação”, e se estiver abaixo de 5, mostrar “Reprovado”.

## Portugol

```
funcao inicio()
{
    real nota1, nota2, media

    escreva("Digite a primeira nota: ")
    leia(nota1)

    escreva("Digite a segunda nota: ")
    leia(nota2)

    media = (nota1 + nota2) / 2

    se (media >= 7)
    {
        escreva("Aprovado")
    }
    senao
    {
        se ((media >= 5) e (media < 7))
        {
            escreva("Recuperação")
        }
        senao
        {
            escreva("Reprovado")
        }
    }
}
```

## Java

```
import java.util.Scanner;

public class IfAninhado {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        float nota1, nota2, media;

        System.out.print("Digite a primeira nota: ");
        nota1 = entrada.nextFloat();
        System.out.print("Digite a segunda nota: ");
        nota2 = entrada.nextFloat();
        media = (nota1 + nota2)/2;

        if(media >= 7)//num1 é maior de todos
        {
            System.out.println("Aprovado");
        }
        else
        {
            if(media >= 5 && media < 7)
            {
                System.out.println("Recuperação");
            }
            else
            {
                System.out.println("Reprovado");
            }
        }
    }
}
```

Evidentemente, pode haver mais de um *if* dentro de outro. Também, naturalmente, o *if* aninhado pode estar dentro da porção *if* ou *else* do outro condicional.

- ◆ Escreva um programa que receba dois números diferentes e imprima o menor número.
- ◆ Escreva um programa que receba o nome e a idade do usuário. Ao final, mostre a qual faixa etária ele pertence, considerando o seguinte:



Até 17	Jovem
Até 59	Adulto
Acima de 60	Idoso

## Operador condicional ternário

A linguagem Java, bem como outras linguagens, conta com um operador condicional que funciona como um *if* simples, em uma única linha. É o operador ternário

```
<condicao> ? <valor se condicao verdadeira> : <valor se condicao falsa>;
```

Confira no exemplo a seguir um trecho de código Java para preencher a variável “resultado” com o valor “par”, se um número inteiro for par, e “ímpar”, caso contrário.

```
Scanner entrada = new Scanner(System.in);  
int valor = entrada.nextInt();  
String resultado = (valor % 2 == 0) ? "par" : "ímpar";  
System.out.println(resultado)
```

Caso “valor” seja divisível por 2, então a expressão retornará o valor “par”; caso contrário, “ímpar”.

Atente-se ao uso do operador ternário, pois ele pode deixar o código confuso.

## Desvio condicional seletivo

Trabalhar com grande quantidade de desvios condicionais pode resultar em um programa de difícil interpretação. Para auxiliar e facilitar esse tipo de necessidade, existe a estrutura de controle com múltipla escolha, denominada *switch*, cuja sintaxe é:

```
switch <variável>
{
    case <opção 1>:
        <operação 1>;
        break;

    case <opção 2>:
        <operação 2>;
        break;

    case <opção N>:
        <operação N>;
        break;

    default:
        <operação default>;
        break;
}
```

A essa sintaxe aplicam-se as seguintes definições:

<variável> é o nome da variável a ser controlada na decisão. <opção> é conteúdo da variável a ser verificado. <operação> é a execução de alguma ação específica.

A instrução *default* da estrutura de seleção *switch* e qualquer operação a ela associada são opcionais, não caracterizando obrigatoriedade de uso. A instrução *break* tem a finalidade de desviar o processamento para fora do comando *switch*. Isso é necessário porque, após o processamento de cada *case*, ocorre retorno do fluxo de ação do comando para a primeira instrução após sua chamada, que nesse caso será *break*.

A variável utilizada em uma estrutura *switch* para controlar a ação desse tipo de desvio pode ser do tipo inteiro, *char* ou *string*.

A palavra *break* é necessária para encerrar a linha de execução de um caso em *switch*. Experimente retirar o primeiro *break* do exemplo 1. (destacado a seguir).

## Exemplo 1

Crie um programa que solicite a entrada de um valor numérico inteiro entre 1 e 7 e apresente o nome do dia da semana por extenso.

```
import java.util.Scanner;

public class Exemplo02 {

    public static void main(String[] args) {
        int dia;
        Scanner entrada = new Scanner(System.in);
        System.out.print("Entre um valor referente a um dia da semana: ");
        dia = entrada.nextInt();

        switch (dia) {
            case 1:
                System.out.println("Domingo");
                break;
            case 2:
                System.out.println("Segunda-feira");
                break;
            case 3:
                System.out.println("Terça-feira");
                break;
            case 4:
                System.out.println("Quarta-feira");
                break;
            case 5:
                System.out.println("Quinta-feira");
                break;
            case 6:
                System.out.println("Sexta-feira");
                break;
            case 7:
                System.out.println("Sábado");
                break;
            default:
                System.out.println("Dia inválido");
                break;
        }
    }
}
```

É muito importante considerar que, do ponto de vista operacional de funcionalidade de um computador, o uso da instrução *switch* prevê apenas condições com operador relacional do tipo “igual a”. Em situações operacionais que envolvam outras formas de relações lógicas, deve-se fazer uso da instrução *if* nas formas sequencial ou encadeada.



## Exemplo 2



Crie um programa Java que recebe o nome de uma forma geométrica (retângulo, triângulo ou círculo) e, de acordo com a forma, solicita dados para cálculo da área e realiza esse cálculo.

Portugal

```
funcao inicio() {  
  
    inteiro forma  
    real raio, base, altura, area  
  
    escreva ("-- Escolha a forma geométrica para realizar o cálculo de sua área --\n")  
    escreva ("1 - Retângulo\n")  
    escreva ("2 - Triângulo\n")  
    escreva ("3 - Círculo\n")  
  
    escreva("Digite o número correspondente a opção desejada: ")  
    leia(forma)  
  
    escolha (forma)  
    {  
        caso 1:  
  
            escreva ("Digite a base: \n")  
            leia(base)  
            escreva ("Digite a altura: \n")  
            leia(altura)  
            area = base * altura  
            escreva ("Área do retângulo: " , area)  
            pare  
        caso 2:  
            escreva ("Digite a base: \n")  
            leia(base)  
            escreva ("Digite a altura: \n")  
            leia(altura)  
            area = base * altura / 2  
            escreva ("Área do triângulo: " , area)  
            pare  
        caso 3:  
            escreva ("Digite o raio:\n")  
            leia (raio)  
            area = 3.1416 * raio * raio  
            escreva ("Área do círculo: " , area)  
            pare  
  
        caso contrario:  
            escreva ("Forma não reconhecida")  
    }  
}
```

## Java

```
import java.util.Scanner;

public class Areas {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in, "latin1");//permite acentos

        String forma;
        float raio, base, altura, area;

        System.out.println("Digite a forma da qual quer calcular área (retangulo, triangulo ou circulo):");
        forma = entrada.nextLine();

        switch(forma)
        {
            case "retangulo":
                System.out.println("Digite primeiro a base depois a altura:");
                base = entrada.nextFloat();
                altura = entrada.nextFloat();
                area = base * altura;
                System.out.println("Área do retângulo: " + area);
                break;

            case "triangulo":
                System.out.println("Digite primeiro a base depois a altura:");
                base = entrada.nextFloat();
                altura = entrada.nextFloat();
                area = base * altura / 2;
                System.out.println("Área do triângulo: " + area);
                break;

            case "circulo":
                System.out.println("Digite o raio:");
                raio = entrada.nextFloat();
                area = 3.1416f * raio * raio;
                System.out.println("Área do círculo: " + area);
                break;

            default:
                System.out.println("Forma não reconhecida");
                break;
        }
    }
}
```

Nesse exemplo, foi utilizada uma variável do tipo *string* para a seleção.

```
switch(forma)
{
    case "retangulo":
        System.out.println("Digite primeiro a base depois a altura:");
        base = entrada.nextFloat();
        altura = entrada.nextFloat();
        area = base * altura;
        System.out.println("Área do retângulo: " + area);
        //break;

    case "triangulo":
        System.out.println("Digite primeiro a base depois a altura:");
        base = entrada.nextFloat();
        altura = entrada.nextFloat();
        area = base * altura / 2;
        System.out.println("Área do triângulo: " + area);
        break;

    case "circulo":
        System.out.println("Digite o raio:");
        raio = entrada.nextFloat();
        area = 3.1416f * raio * raio;
        System.out.println("Área do círculo: " + area);
        break;

    default:
        System.out.println("forma não reconhecida");
        break;
}
```

Ao executar, escolha a opção “retangulo” e note que ele solicitará tanto os dados de retângulo quanto de triângulo. Isso porque, a partir do *case* alvo, o programa executará até encontrar um *break* ou o fim do escopo de *switch* – ou seja, a chave *{}*. Em alguns casos, isso pode ser benéfico. Por exemplo, se você quisesse permitir que o usuário digite “retangulo” ou “retângulo”, com acento (e assim também com “triângulo” e “círculo”), você poderia aplicar a seguinte alteração no código do *switch*:

```
switch(forma)
{
    case "retângulo":
    case "retangulo":
        System.out.println("Digite primeiro a base depois a altura:");
        base = entrada.nextFloat();
        altura = entrada.nextFloat();
        area = base * altura;
        System.out.println("Área do retângulo: " + area);
        break;

    case "triângulo":
    case "triangulo":
        System.out.println("Digite primeiro a base depois a altura:");
        base = entrada.nextFloat();
        altura = entrada.nextFloat();
        area = base * altura / 2;
        System.out.println("Área do triângulo: " + area);
        break;

    case "círculo":
    case "circulo":
        System.out.println("Digite o raio:");
        raio = entrada.nextFloat();
        area = 3.1416f * raio * raio;
        System.out.println("Área do círculo: " + area);
        break;

    default:
        System.out.println("forma não reconhecida");
        break;
}
```

Escreva um programa que mostrará duas opções para o usuário e, em seguida, solicitará ao usuário que escolha uma opção.

◆ Java

◆ JDK

Caso o usuário escolha 1, será mostrada a mensagem: “Java é uma linguagem de programação e plataforma computacional”. Se o usuário escolher 2, será mostrada a mensagem: “É um pacote de software que você pode usar para desenvolver aplicativos

baseados em Java”. Qualquer outra opção mostrará a mensagem: “Opção inválida”.



## If Else

Assista agora um vídeo que explica passo a passo o uso do If Else



← [Palavras reservadas em Java \(?page=7\)](#) Estruturas de repetição → [\(?page=9\)](#)

## Estruturas de repetição

Sabe-se que a capacidade operacional de um computador é executar programas e que os programas (obtidos a partir de algoritmos) são sequências de instruções que fornecem certa ordem de execução a um computador. Algumas dessas ordens devem ser repetidas algumas vezes e, para realizar esse tipo de ação, existe a técnica de uso dos laços de repetição.

Os laços de repetição (*looping*) formam uma importante estrutura nas linguagens de programação, pois possibilitam a repetição da execução de um bloco de instruções em um programa. Eles determinam que um certo bloco seja executado repetidamente até que uma condição específica ocorra. A repetição é uma das estruturas mais usadas em programação, possibilitando a criação de contadores, temporizadores, rotinas para classificação, obtenção e recuperação de dados. A criação de laços de repetição em Java é feita a partir das estruturas *for*, *while* e *do-while*. Em comparação com o Portugol, tem-se:

*for* = para; *while* = enquanto; *do-while* = faça-enquanto

### Laço *for*

A instrução ***for*** (ou “***para***”) é um tipo de contador finito, isto é, ela realiza a contagem de um valor inicial conhecido até um valor final também conhecido. Ela contém uma variável de controle do tipo contador, que pode ser crescente ou decrescente, e tem a seguinte sintaxe:

```
for (inicialização; condição; incremento ou decremento)
{
    <instrução 1>;
    <instrução 2>;
    ...
    <instrução N>;
}
```

### Inicialização

É o valor inicial da variável de controle do laço.



## Condição

Contém uma expressão booleana (ou booliana), que será utilizada para controlar a continuidade do laço. Deve conter o valor final que a variável de controle pode assumir dentro do laço.

## Incremento ou decremento

É o passo com que a variável de controle será acrescida (++) ou decrescida (--). Esse incremento pode ser realizado por meio de uma variável inteira ou com ponto flutuante, permitindo pequenos incrementos decimais.

Confira agora um exemplo prático:

## Exemplo 1

Crie um programa que realizará uma contagem, na ordem crescente, de 1 a 10.

No NetBeans, crie um novo projeto Java chamado “LacoFor” e modifique a classe principal, conforme apresentado a seguir:



```
public class LacoFor {  
  
    public static void main(String[] args) {  
        int contador;  
  
        // INICIALIZAÇÃO: atribua o número 1 à variável contador  
        // CONDIÇÃO: atribua a condição menor ou igual a 10  
        // INCREMENTO ou DECREMENTO: use o incremento, pois quer-se fazer uma contagem na ordem crescente  
        for ( contador = 1; contador <= 10; contador++) {  
            System.out.println(contador);  
        }  
    }  
}
```

O resultado será:

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

## Exemplo 2

Crie um programa Java que recebe um número e mostra na tela a tabuada desse número.

### Portugol

```
programa
{
    funcao inicio()
    {
        inteiro numero, resultado, contador

        escreva("Informe o número: ")
        leia(numero)

        limpa()

        para (contador = 1; contador <= 10; contador++)
        {
            resultado = numero * contador
            escreva (numero, " X ", contador, " = ", resultado , "\n")
        }
    }
}
```

## Java

```
import java.util.Scanner;

public class Tabuada {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int numero, resultado;

        System.out.print("Informe o número: ");
        numero = entrada.nextInt();

        for(int i = 1; i <= 10; i++)
        {
            resultado = numero * i;
            System.out.println(numero + " X " + i + " = " + resultado);
        }
    }
}
```

Observe, pelo exemplo anterior, que, em Java, **é possível declarar e inicializar a variável de contagem** diretamente na instrução *for* (no caso, *int i = 1*). Essa variável existe apenas no escopo do laço *for*; fora do laço, ela não será reconhecida.

Em *for*, pode-se decrementar a variável ao invés de incrementar. Por exemplo, o código a seguir recebe um número e um laço *for* para imprimir desde o número informado até 1. Observe o trecho destacado em azul:

```
numero = entrada.nextInt();
for(int i = numero; i > 0; i--)
{
    System.out.println(i);
}
```

Também é possível incrementar ou decrementar em quantidades maiores que 1, como no exemplo a seguir, que mostra todos os números pares entre 0 e 50.

Observe o incremento destacado em azul:

```
for(int i=0; i <= 50; i += 2)
{
    System.out.println(i);
}
```

Existe uma estrutura de *for* específica para listas, chamada *for-each* (“para cada”). Ela é abordada no conteúdo “Estruturas de Dados”, desta unidade curricular.

## Laços *while* e *do-while*

O *while* é outro laço condicional, isto é, um conjunto de instruções que são repetidas enquanto o resultado de uma expressão lógica (uma condição) é avaliado como verdadeiro. Sua sintaxe é:

```
while (<condição>){  
    <conjunto de instruções>  
}
```

A instrução *while* avalia o resultado da expressão (condição) antes de executar as instruções do bloco { }. Assim, é possível que as instruções nunca sejam executadas, caso a condição seja inicialmente falsa. Um problema típico, relacionado à avaliação da condição *while*, é o **laço infinito**: caso a condição nunca se torne falsa, o laço será repetido infinitamente.

Há outro tipo de laço condicional, o chamado *do-while*, que é muito parecido com o *while*, porém o conjunto de instruções é executado antes da avaliação da expressão lógica. Isso faz com que essas instruções sejam executadas pelo menos uma vez. Veja como a sintaxe do *do-while* é bem parecida com a do *while*:

```
do{  
    <conjunto de instruções>  
}  
while (<condição>;
```

## Exemplo 1

Crie um programa que realizará uma contagem de 1 a 10, conforme critério do laço *while*.

### Portugol

```
funcao inicio()  
{  
    inteiro contador = 1  
    enquanto(contador <= 10)  
    {  
        escreva("\nEste é o número: ", contador)  
        contador++  
    }  
}
```

## Java

```
public class ExemploWhile {  
  
    public static void main(String[] args) {  
  
        int contador = 1;  
  
        while (contador <= 10) {  
            System.out.println("Este é o número: " + contador);  
            contador++;  
        }  
    }  
}
```

Repare que existe a variável `contador`, que é iniciada valendo 1 e, a cada *loop* executado (repetição), é somado 1 ao contador. Perceba que o *while* manterá a repetição enquanto a variável `contador` for menor ou igual ao número 10.

Outro ponto importante é que a variável `contador` é inicializada antes de chegar ao *while*, porque o *while* comparará a sentença e só depois permitirá a execução do bloco.

## Exemplo 2

Crie um programa que realizará uma contagem de 1 até 10, conforme critério do laço *do-while*.

## Portugol

```
funcao inicio()
{
  inteiro i = 1
  faca
  {
    escreva("\nNúmero: ", i)
    i++
  } enquanto (i <= 10)
}
```

## Java

```
public class ExemploDowhile {

    public static void main(String[] args) {
        int i = 1;

        do {
            System.out.println("Número: " + i);
            i++;
        } while (i <= 10);
    }
}
```

Note que o resultado desse programa será o mesmo do exemplo 1, mas, utilizando o *do-while*, primeiro são colocadas as instruções (*do*) e depois a condição (*while*).

## Exemplo 3

Crie um programa Java para ler os tempos de volta de um velocista e calcular qual foi o menor tempo.

## Portugol

```
funcao inicio() {  
  
    real tempoVolta  
    real menorTempo = 100000 //tempo máximo por volta  
    cadeia continuar = "s"  
  
    enquanto (continuar == "s")  
    {  
  
        escreva("Informe o tempo (s) da volta do velocista: ")  
        leia (tempoVolta)  
  
        se (menorTempo > tempoVolta)  
        {  
            menorTempo = tempoVolta  
        }  
  
        escreva ("Deseja informar novo tempo(s/n)?\n")  
        leia (continuar)  
  
    }  
    escreva ("Menor volta: ", menorTempo)  
  
}
```

## Java

```
import java.util.Scanner;

public class Velocista
{

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        float tempoVolta, menorTempo;
        String continuar = "S";

        //inicializando o menor tempo com o maior valor possível
        // (qualquer primeiro valor informado será menor que este)
        menorTempo = Float.MAX_VALUE;

        while(continuar.equals("S"))
        {
            System.out.print("Informe o tempo (s) da volta do velocista:");
            tempoVolta = entrada.nextFloat();

            //este nextLine é necessário para que o próximo nextLine, ao fim do laço, não
            falhe
            entrada.nextLine(); //consome o "enter" digitado pelo usuário

            if(menorTempo > tempoVolta)
            {
                menorTempo = tempoVolta;
            }

            System.out.print("Deseja informar novo tempo? S/N ");
            continuar = entrada.nextLine();
        }

        System.out.println("A menor volta do velocista foi de " + menorTempo + " segundo
s.");
    }
}
```

*While* e *do-while* são ideais quando não se sabe quantas vezes o laço terá que executar, como no exemplo mostrado (o usuário pode informar “N” após a primeira entrada ou após ter informado centenas de valores). Quando se sabe exatamente quantas iterações o laço tem que executar, o mais indicado é usar *for*.



## Exemplo 4



Crie um programa Java que valida a entrada do usuário. Ele deve informar uma idade válida, que deve estar entre 0 e 150.

### Portugol

```
programa
{
funcao inicio()
{
    inteiro idade

    faca
    {
        escreva ("Informe uma idade válida (entre 0 e 150)")
        leia (idade)
    }
    enquanto (idade < 0 ou idade > 150)

    escreva ("\nObrigado por informar uma idade válida\n")
}
}
```

### Java

```
import java.util.Scanner;

public class Idade
{

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int idade;

        do{
            System.out.println("Informe uma idade válida (entre 0 e 150)");
            idade = entrada.nextInt();
        }while(idade < 0 || idade > 150);

        System.out.println("obrigado por informar uma idade válida");
    }

}
```

Validações de entrada de usuário são bastante importantes para que não seja gerado erro no programa. No caso apresentado, seria possível, evidentemente, utilizar o laço *while* ao invés de *do-while*; no entanto, como sempre há ao menos uma entrada de dados do usuário, o *do-while* torna-se um pouco mais lógico.

- ◆ Escreva um programa no qual o usuário digite um número entre 1 e 10 e seja mostrado o fatorial desse número (fatorial de  $n = n * (n-1) * (n-2) \dots * 1$ . Exemplos: fatorial de 4 =  $4 * 3 * 2 * 1$ ; fatorial de 3 =  $3 * 2 * 1$ ).
- ◆ Escreva um programa que receba um valor  $n$ , leia  $n$  notas de um aluno e, ao final, mostre a média (soma das notas dividida por  $n$ ) e a mensagem “Aprovado” ou “Reprovado”, considerando que, para aprovação, o aluno deve ter média maior ou igual a 6.

## If Else, For e Do While

Assista agora um vídeo que explica o uso do If Else, For e Do While em um projeto.



← Estruturas condicionais (?page=8) Estrutura de dados: vetor e matriz → (?page=10)

## Estrutura de dados: vetor e matriz

Suponha que seja necessário armazenar e manipular dezenas de nomes de pessoas em um programa de computador. De acordo com o que foi estudado até aqui, seriam necessárias dezenas de variáveis, cada uma armazenando um nome diferente, como, por exemplo, **nome1="Lucas"**, **nome2="Daniel"** e assim por diante. Em vez disso, é possível a declaração de apenas uma variável indexada, chamada **array**.

Em outras palavras, pode-se definir uma variável cujos elementos são referenciados por um índice no seguinte formato: **nome[0]="Lucas"**, **nome[1]="Daniel"** etc. O número entre colchetes se refere ao índice que diferencia os conteúdos da variável. Talvez você não tenha notado, mas está sendo usado um *array* desde o primeiro exemplo em Java, pois **String[] args** nada mais é do que um *array* de elementos do tipo *string*. Os *arrays* podem ser unidimensionais (com uma única dimensão, conhecido como vetor) ou bidimensionais (com duas dimensões, conhecido por matriz).

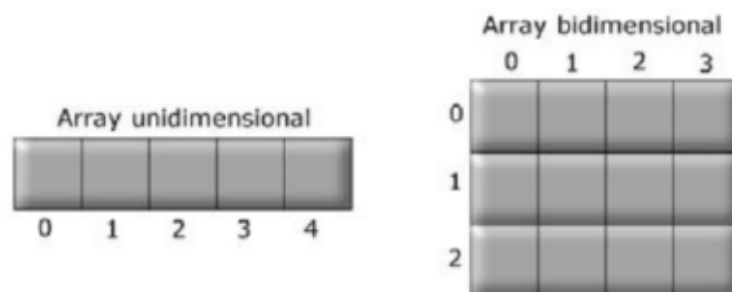


Figura 2 – Array unidimensional e bidimensional

Fonte: Senac EAD (2022)

Repare que, em Java, o primeiro elemento do **array** sempre possui índice 0 (zero).

### Arrays unidimensionais: vetores

Os *arrays* unidimensionais são os que contêm apenas um índice para acessar seu conteúdo. Eles são declarados da seguinte maneira:

```
Tipo[] nome-do-array = new Tipo[quantidade]
```

Essa declaração constitui-se dos seguintes conceitos:

- ◆ **Tipo:** pode ser qualquer tipo de variável
- ◆ **Nome-do-array:** um nome válido (aqui são seguidas as mesmas regras para nomes de variáveis)
- ◆ **quantidade:** a quantidade de elementos que o *array* pode manipular

Veja dois exemplos de vetores:

- ◆ **int [ ] numeros = new int[10];**: cria um *array* com o nome **numeros**, que contém **10 elementos do tipo *int*** e seu **índice varia de 0 a 9**.
- ◆ **String [ ] meses = new String[12];**: cria um *array* com o nome **meses**, que contém **12 elementos do tipo *string*** e seu **índice varia de 0 a 11**.

Os *arrays* podem ser criados e inicializados de outra maneira. Em vez de usar o operador *new* para criar um *array*, é possível definir os elementos do *array* entre chaves e separados por vírgula. Esses elementos devem ser do mesmo tipo da variável definida como *array*. A sintaxe é:

```
Tipo[] nome-do-array = { valores separados por vírgulas }
```

Confira dois exemplos práticos agora, criando *arrays* dos dois métodos:

## Exemplo 1

Crie um vetor de 10 números inteiros, inicializando-o com valores aleatórios.

```
public class ArrayNumeros {  
  
    public static void main(String[] args) {  
        int [] numeros = new int [10];  
        for (int i = 0; i < 10; i++) {  
            numeros[i] = (int) (Math.random() * 100);  
            System.out.println("Número: " + numeros[i]);  
        }  
    }  
}
```

O seu resultado será semelhante a este (os números mudarão, pois *random* gera números aleatórios):

Número: 17

Número: 91

Número: 66

Número: 32

Número: 43

Número: 56

Número: 57

Número: 69

Número: 82

Número: 21

Explicação detalhada do código:

- ◆ **int [] numeros = new int [10]:** declara o *array* unidimensional chamado **numeros**, contendo dez elementos (índices de 0 a 9).
- ◆ **for (int i = 0; i < 10; i++):** contém um laço de repetição com a instrução *for*, que faz com que as linhas 6 a 8 sejam executadas dez vezes. O valor da variável "i" inicia em zero é incrementado em 1 até o limite estabelecido pelo laço.
- ◆ **numeros[i] = (int) (Math.random() \* 100):** armazena no *array* **numeros** um valor gerado aleatoriamente por meio do método *random*. A cada ciclo de execução do laço, o número gerado é armazenado em um elemento diferente do *array*. Na primeira vez, o número é armazenado em **numeros[0]**, na segunda, em **numeros[1]**, e assim sucessivamente.
- ◆ **System.out.println("Número: " + numeros[i]):** imprime em tela o número gerado que foi armazenado no elemento do *array*.

## Exemplo 2

Crie e inicialize um vetor de *string* com os dias da semana.

```
public class ArraySemana {  
  
    public static void main(String[] args) {  
  
        String[] semana = { "Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado" };  
        System.out.println("Dias da Semana:");  
        for (int i = 0; i < semana.length; i++) {  
            System.out.println(semana[i]);  
        }  
    }  
}
```

O resultado será:

Dias da Semana:

Domingo

Segunda

Terça

Quarta

Quinta

Sexta

Sabádo

Explicação do código:

- ◆ **String[] semana = { "Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sabádo" };** declara o *array* unidimensional chamado **semana**, contendo os sete dias da semana por extenso (índices de 0 a 6).
- ◆ **for (int i = 0; i < semana.length; i++):** contém um laço de repetição com a instrução *for*, que faz com que a linha 6 seja executada sete vezes, imprimindo o conteúdo dos elementos do *array*.

## Exemplo 3

Crie um programa Java para ler notas de uma turma, calcular a média e determinar quantas e quais notas ficaram acima da média.

### Portugol



```
inclua biblioteca Util --> u

funcao inicio()
{
    inteiro qtdeAlunos, qtdeAcimaMedia
    real soma, media

    /*escreva("Quantos alunos serão informados?")
    leia(qtdeAlunos)*/

    //Portugol não suporta tamanho dinâmico na declaração de vetor,
    // apenas valor constante
    real notas[5]
    soma = 0

    para(inteiro i = 0; i < u.numero_elementos(notas); i++)
    {
        escreva("\nDigite a nota ", (i+1), ": ")
        leia(notas[i])
        soma = soma + notas[i]
    }

    media = soma / 5
    escreva("\nMédia ", media)

    qtdeAcimaMedia = 0
    para(inteiro i = 0; i < u.numero_elementos(notas); i++)
    {
        se(notas[i] > media)
        {
            escreva("\nNota ", notas[i], " acima da média")
            qtdeAcimaMedia++
        }
    }

    escreva("\nForam no total ", qtdeAcimaMedia, " notas acima da media")
}
```

## Java

```
import java.util.Scanner;

public class AcimaDaMedia
{
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        int qtdeAlunos, qtdeAcimaMedia;
        float soma, media;

        System.out.println("Quantos alunos serão informados?");
        qtdeAlunos = entrada.nextInt();

        //declarando o vetor usando o valor da variável qtdeAlunos para a quantidade
de elementos
        float[] notas = new float[qtdeAlunos];
        soma = 0;

        //vetor.length é uma propriedade que traz o número de elementos de um vetor
for(int i = 0; i < notas.length; i++)
        {
            System.out.println("Digite a nota " + (i+1) + ": ");
            notas[i] = entrada.nextFloat();
            soma += notas[i];
        }

        //calculando a média a partir do somatório obtido no laço anterior
media = soma / qtdeAlunos;
System.out.println("Média " + media);

        qtdeAcimaMedia = 0;

        for(int i = 0; i < notas.length; i++)
        {
            if(notas[i] > media)
            {
                System.out.println("Nota " + notas[i] + " acima da média.");
                qtdeAcimaMedia++;
            }
        }

        System.out.println("Foram no total " + qtdeAcimaMedia + " notas acima da méd
ia");
    }
}
```

Pelo exemplo anterior, note que a propriedade **length** pode ser usada para descobrir o tamanho de um vetor. Além disso, o tamanho de um vetor pode ser determinado dinamicamente, por meio de variável, ao invés de um número fixo.

Note, a partir dos exemplos, que a leitura e a atribuição de valores ao vetor são semelhantes às praticadas em Portugal.

```
vetor[índice] = valor;  
umaVariavel = vetor[índice]
```

É possível que você acesse um índice maior que o estipulado para o vetor, como neste trecho de código:

```
int[] numeros = new int[10];  
numeros[10] = 50;
```

Neste caso, o compilador e o NetBeans emitirão um erro com mensagem semelhante à seguinte:

```
“Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out of  
bounds for length 10”  
Traduzindo: Exceção na linha de execução “main”  
java.lang.ArrayIndexOutOfBoundsException() (algo como “exceção array índice fora dos  
limites”): índice 10 fora do limite para tamanho 10.
```

Como se sabe, o vetor inicia com índice 0 e termina com índice tamanho-1 (ou seja, o último índice nesse caso seria 9).

É importante lembrar ainda que vetores são imutáveis e, caso você queira incluir um elemento a mais nele, será preciso criar um novo vetor com mais posições e transferir os dados do antigo para o novo. Analogamente, não se pode diminuir o número de posições de um vetor.

## Arrays bidimensionais: matrizes

Um *array* bidimensional contém dois índices e possibilita que os valores sejam armazenados na forma de matrizes – ou seja, com linhas e colunas. Java trata matriz como um vetor de vetores, ou seja, cada posição de um vetor é um vetor de valores. Na prática, a sua manipulação funciona como se lidássemos diretamente com linhas e colunas. As mais usuais são as matrizes de duas dimensões, mas é possível criar *arrays* com quantas dimensões forem necessárias. Esses *arrays* devem ser declarados da seguinte maneira:

```
Tipo nome-matriz[][] = new Tipo[dimensão linha][dimensão coluna];
```

A essa declaração aplicam-se as seguintes definições:

- ◆ **<tipo>** é o tipo de dado primitivo ou a classe.
- ◆ **<nome-vetor>** é o nome atribuído ao arranjo.
- ◆ **<dimensão linha>** é o tamanho do arranjo em número de linhas.
- ◆ **<dimensão coluna>** é o tamanho do arranjo em número de colunas.

Confira os seguintes exemplos práticos:

## Exemplo 1

Crie um programa Java para preencher uma matriz de três linhas e quatro colunas com números aleatórios.

**Portugol**

```
inclua biblioteca Util --> g

funcao inicio()
{
    // Cria a matriz
    inteiro matriz[3][4]

    para (inteiro linha = 0; linha < 3; linha++)
    {
        para (inteiro coluna = 0; coluna < 4; coluna++)
        {
            matriz[linha][coluna] = g.sorteia(0, 10) // Atribui um valor aleatório d
e 0 até 10 à posição da matriz

            escreva("[", matriz[linha][coluna], "]") // Exibe o valor contido na pos
ição da matriz
        }

        escreva ("\n")
    }
}
```

## Java

```
import java.util.Random;
public class PrimeiraMatriz
{
    public static void main(String[] args) {
        int[][] numAleatorio = new int[3][4];
        Random numeroRandom = new Random();

        for (int i = 0; i < numAleatorio.length; i++) {
            for (int j = 0; j < numAleatorio[i].length; j++) {
                numAleatorio[i][j] = numeroRandom.nextInt(10); //gera número aleatório de 0
a 10
            }
        }

        System.out.print("\t\tMatriz\n");
        for (int i = 0; i < numAleatorio.length; i++) {
            for (int j = 0; j < numAleatorio[i].length; j++) {
                System.out.print("\t" + numAleatorio[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Nesse código, foi utilizado um laço controlado pela variável “i” para percorrer linha a linha e, dentro desse laço, um controle pela variável “j”, indo coluna a coluna. O acesso aos valores ocorre de maneira muito semelhante à de Portugal.

## Exemplo 2

Crie um programa que efetue a entrada de três notas escolares de quatro alunos de uma sala e apresente os dados fornecidos em seguida.

```
import java.text.DecimalFormat;
import java.util.Scanner;

public class ArrayBidimensional {

    public static void main(String[] args) {
        float NOTA[][] = new float[4][3];
        byte I, J;
        DecimalFormat df = new DecimalFormat("0.00");
        Scanner s = new Scanner(System.in);
        for (I = 0; I <= 3; I++) {
            System.out.println();
            System.out.println();
            System.out.println("Aluno ---> " + (I + 1));
            System.out.println();
            for (J = 0; J <= 2; J++) {
                System.out.print((J + 1) + "ª Nota = ");
                NOTA[I][J] = s.nextFloat();
            }
        }
        System.out.println();
        System.out.println("Aluno Nota1 Nota2 Nota3 Nota4");
        System.out.println("-----");
        for (I = 0; I <= 3; I++) {
            System.out.print((I + 1) + " --> ");
            for (J = 0; J <= 2; J++) {
                System.out.print(df.format(NOTA[I][J]) + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

O programa desenvolvido usa um arranjo de duas dimensões e, para sua execução, foram necessários dois laços para controlar o acesso às posições de índice.

A instrução `float NOTA[][] = new float[4][3];` usa um arranjo bidimensional com quatro linhas e três colunas, que será manipulado pelo trecho de código do programa. Observe também o uso da referência `NOTA[I][J]`, em que a **variável I** está controlando o acesso à linha da tabela (do arranjo) e a **variável J** está controlando o acesso à coluna na tabela.

## Array e For



Assista agora um vídeo que explica o uso do vetor Array e For.



← Estruturas de repetição (?page=9) Destrutores: coletor de lixo → (?page=11)



## Destrutores: coletor de lixo

Cada variável e cada objeto em Java é um espaço em memória que fica reservado para receber valores. Quando essas variáveis e esses objetos não são mais necessários, o espaço em memória reservado para eles deve ser liberado, para que novas variáveis e novos objetos sejam criados. O trecho a seguir, por mais simples que pareça, afetaria a memória de uma máquina que está rodando a aplicação Java.

```
int numero1 = 5, numero2 = 10;  
String informacao1 = "Lorem Ipsum";  
String informacao2 = "dolor sit amet";  
String informacao3 = new String("consectetur adipiscing elit");  
float[] vetorFloat1 = new float[10];  
char[] vetorChar = {'a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', 'U'};
```

Cada linha corresponde a uma declaração de variável ou vetor e, portanto, ocupa espaço em memória. É necessária uma atenção especial às variáveis *string*, que na verdade são objetos implícitos, e aos vetores. Se todos esses itens fossem criados e nunca excluídos, rapidamente a memória atingiria seu limite e geraria erro no sistema.

Apesar de essa característica ser um problema, ela não representa uma ameaça, pois a linguagem Java tem um processo automático para limpeza de objetos não utilizados depois de um certo tempo. Isso é feito por meio de um processo nomeado **coleta automática de lixo (*automatic garbage collection*)**. Cada vez que um objeto deixa de ser referenciado pelo programa, ficando perdido na memória e, portanto, não tendo mais utilidade, ele recebe uma marcação para eliminação futura. Assim que o programa ficar ocioso, o gerenciador automático de memória destrói os objetos perdidos, liberando os recursos ao sistema. Com esse processo, torna-se desnecessário liberar a memória explicitamente em um programa em Java, uma vez que isso é feito automaticamente. Isso ocorre de maneira diferente de linguagens como C e C++, que permitem a destruição explícita e imediata de objetos.

← Estrutura de dados: vetor e matriz (?page=10) Encerramento → (?page=12)

## Encerramento

Neste conteúdo, você teve o primeiro contato com a linguagem Java, aprendendo sobre o escopo padrão de código, que é criado sempre quando se inicia um projeto, e o que significa cada linha. Foram abordadas as estruturas condicionais e de repetição sempre com foco na prática, pois elas são constantemente utilizadas no dia a dia do programador. Você aprendeu também sobre a definição de variáveis e como declará-las. É muito importante que você tenha esse fundamento bem definido, pois sempre o utilizará.

Todos esses conceitos são essenciais na linguagem Java (ou em qualquer outra linguagem de programação orientada a objetos), porém, o importante não é decorar cada nome de função, cada sintaxe, e sim saber que elas existem e como aplicá-las no contexto desejado.

A partir dessa experiência inicial, você estará preparado para trabalhar com orientação a objetos, paradigma sobre o qual Java é construído.

← [Início \(?page=1\)](#)