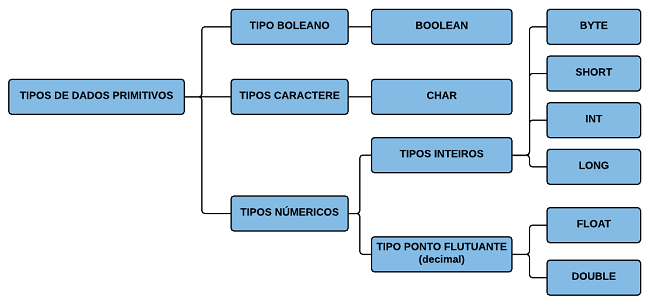
## **Tipos de dados básicos (primitivos)**

**O que são tipos primitivos de dados?**

As tarefas executadas por computadores, tais como escrever mensagens na tela, resolver problemas matemáticos ou até mesmo desenhar arquivos tridimensionais, são baseadas em manipulação de dados. Esses dados podem ser de vários tipos: números, caracteres, textos etc. Chamamos de tipos primitivos os tipos de dados básicos de uma linguagem de programação a partir dos quais é possível criar tipos mais complexos. Java, assim como outras linguagens de programação, também possui seus tipos primitivos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Valores Possíveis** | |  |  |  |
| **Tipos** | **Primitivo** | **Menor** | **Maior** | **Valor Padrão** | **Tamanho** | **Exemplo** |
| Inteiro | byte | -128 | 127 | 0 | 8 bits | byte ex1 = (byte)1; |
| short | -32768 | 32767 | 0 | 16 bits | short ex2 = (short)1; |
| int | -2.147.483.648 | 2.147.483.647 | 0 | 32 bits | int ex3 = 1; |
| long | -9.223.372.036.854.770.000 | 9.223.372.036.854.770.000 | 0 | 64 bits | long ex4 = 1; |
| Ponto Flutuante | float | -1,4024E-37 | 3.40282347E + 38 |  | 32 bits | float ex5 = 5.50f; |
| double | 4.94E-307 | 1.79769313486231570E + 308 | 0 | 64 bits | double ex6 = 10.20d;  ou  double ex6 = 10.20; |
| Caractere | char | 0 | 65535 | /0 | 16 bits | char ex7 = 194;  ou  char ex8 = ‘a’; |
| Booleano | boolean | false | true | false | 1 bit | boolean ex9 = ‘true’; |

Conforme você pôde perceber, Java possui oito tipos primitivos de dados. Para facilitar o entendimento sobre eles, confira o diagrama a seguir, que apresenta os tipos primitivos da Linguagem Java divididos por categoria.



Operadores

|  |  |
| --- | --- |
| **OPERADOR** | **DESCRIÇÃO** |
| [Operadores de atribuição] | |
| =, +=, -=, \*=, /=, %= | Operadores de atribuição de valores. |
| [Operadores Lógicos] | |
| && , || , ^ | Operadores “E”, “OU” e “OU EXCLUSIVO” |
| ! | Operador lógico de negação. |
| [Operadores Relacionais] | |
| <, >, <=, >=, ==, != | Operadores que comparam dois valores e desenvolvem uma resposta booleana. |
| [Operadores Aritméticos] | |
| +, -, \*, /, % | Operadores de funções matemáticas básicas. |
| ++, -- | Operadores de incremento e decremento de uma unidade. |

Existem também os operadores de manipulação bit-a-bit, como >>, << e >>>. Esses operadores podem ser aplicados a variáveis que contenham valores numéricos, pois modificam diretamente suas representações binárias salvas na memória. Para determinados casos, esses operadores são mais eficientes para efetuar cálculos como multiplicação e divisão que os operadores tradicionais.

Por exemplo, suponha que temos uma variável do tipo inteiro contendo o valor 16. Sua representação interna em memória é 10000. Se aplicarmos o operador de deslocamento >> em 1 bit para a direita, transformaremos o valor em 01000, que equivale ao valor 8, ou seja, à metade de 16. Da mesma forma, o operador deslocamento para a esquerda, <<, desloca os bits para a esquerda, equivalendo à operação de multiplicação.

Obs: Um detalhe importante é que operadores de atribuição não geram retorno quando utilizados. Isso significa que eles apenas alteram uma das variáveis envolvidas, não podendo ser usados como referência em uma condição, por exemplo.

Conversão de Tipos de Dados

Quando há a necessidade de se converter o tipo de dado de uma variável para outro tipo, por exemplo, quando precisamos arredondar valores, executamos uma ação conhecida como [casting](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=casting) ou conversão de tipo. É importante notar que nem todo tipo de conversão pode ser feita com sucesso, podendo ocorrer erros ou haver perda de dados. A título de exemplo, em Java, não é possível converter um dado para o tipo booleano, qualquer que seja ele. Existem duas formas de [casting](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=casting) em Java:

**Implícito**

No [casting](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=casting) implícito, a variável de destino deve ser capaz de converter automaticamente o valor da variável de origem sem perda de dados, caso contrário, será gerada uma exceção (estudaremos exceções mais adiante). Um exemplo deste tipo de [casting](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=casting) é o que ocorre quando tentamos colocar um valor inteiro em uma variável de ponto flutuante, como no exemplo a seguir.

int a = 10;

float b;

b = a;

Nesse caso, a variável "a", que é do tipo int (inteiro), simplesmente é convertida para um valor do tipo float (ponto flutuante). O valor inteiro da variável “a” foi armazenado na variável "b" em um formato com casas decimais. Perceba que assim nenhum dado da variável "a" foi perdido.

**Explícito**

O [casting](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=casting) explícito é permitido entre quase todos os tipos de dados (exceto para o tipo booleano). Poderá ocorrer a perda de dados ou a geração de uma exceção caso haja incompatibilidade entre os valores. Para realizar uma conversão explícita, deve-se colocar o tipo de dado para o qual se deseja converter a variável de origem, como no exemplo a seguir.

int x;

float y =79.5f;

x = (int) y;

Perceba que explicitamos o tipo para o qual queremos converter o valor da variável "y", no caso, int. Assim, estamos forçando que o valor de "y" seja convertido para inteiro e colocado em "x". Essa conversão implica em perda de informação, pois as casas decimais armazenadas em "y" não podem ser mantidas em "x", já que "x" é do tipo inteiro.

**Como utilizo variáveis do tipo Texto em Java?**

Em Java, utilizamos objetos da classe String para armazenar uma sequência de caracteres. É importante entender que, conforme afirma Xavier (2009), String não é um tipo primitivo da linguagem Java, mas sim uma classe. Como estudaremos sobre classes e objetos apenas mais adiante, vamos considerar que variáveis que armazenam texto são "variáveis do tipo String". Posteriormente, constataremos que a criação de variáveis do tipo String é idêntica à criação de variáveis de tipos primitivos em Java.

## [**Métodos**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=M%C3%A9todos)

**Mas o que são** [**métodos**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todos)**?**

[Métodos](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=M%C3%A9todos) são blocos organizados de código que podem ser programaticamente executados. Sua utilidade principal é agrupar trechos de código que trabalham conjuntamente para alcançar um determinado objetivo. Por exemplo, um método que efetua a soma de dois números deve conter todas as instruções para executar com sucesso a soma de dois números dados. Assim, se precisarmos somar dois números diversas vezes em nosso programa, não é obrigatório repetir todas as instruções que são necessárias para efetuar essa soma, basta executar nosso método de soma. Embora métodos também sejam conhecidos como funções em outras linguagens como C e C++, na literatura relacionada à Linguagem Java, você sempre os encontrará com o nome de "método".

A estrutura que compõe um método é denominada “assinatura” e possui, pelo menos, três partes. A primeira é um nome, a segunda é o tipo de dado do que é gerado como resultado da sua execução e a terceira parte é uma lista de parâmetros. Um método pode gerar um resultado que precise ser "devolvido", ou seja, repassado à linha de código fonte onde ele foi requisitado. Nesse caso, devemos definir o tipo do resultado que será devolvido pelo método após a sua execução (como int, float, boolean, String etc). Quando não é necessário devolver este resultado, define-se o retorno do método como void (vazio). Além disso, um método pode receber uma lista de parâmetros, que é uma lista de variáveis que ele precisa para trabalhar. Essa lista pode ser vazia, o que significa que o método não precisa de informações de fora dele para trabalhar.

Existem mais dois elementos que podem compor a assinatura de um método: modificador de visibilidade e modificador de contexto. O modificador de visibilidade diz se o método é público, protegido ou privado. Esse conceito tem efeito sobre quem pode acessar o método em questão. Já o modificador de contexto indica se o método precisa de um objeto da classe à qual pertence para ser acessado ou não. Estudaremos mais sobre esses dois parâmetros nas próximas aulas. Podemos resumir a assinatura mais completa de um método como sendo:

< modificador de visibilidade > < modificador de contexto >

< tipo de retorno > < nome do método > ( < lista de parâmetros.. .> )

Exemplo de um método contendo todos os elementos descritos anteriormente:

public [static](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=static) void soma(int valor\_1, int valor\_2) { ... }

Você deve ter percebido o uso da palavra [static](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=static) na assinatura de todos os [métodos](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todos) do nosso exemplo. Essa palavra-chave é um modificador de contexto e tem relação com a possibilidade de execução de um método sem a presença de um objeto do mesmo tipo da classe que o contém. Estudaremos o significado exato desta palavra-chave e suas aplicações em outras aulas.

## **Arrays**

Arrays são estruturas de dados que agrupam itens de um mesmo tipo, que são ordenados sequencialmente e de tamanho fixo.  
  
 Vetores são estruturas de dados de tamanho fixo que agrupam elementos do mesmo tipo. Em Java, chamamos vetores e matrizes de arrays. Você encontrará conteúdos sobre vetores e matrizes com o nome de [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) em todos os materiais de Java escritos em inglês e algumas classes da biblioteca padrão do Java também terão esse nome. ***Vetores são chamados de arrays unidimensionais*** e ***matrizes são arrays multidimensionais***.  
  
 Para criação de arrays em Java, precisamos das seguintes especificações: um nome (identificador), o tipo de dado e o tamanho de cada dimensão.

01 public void Exemplo1(){

02

03 /\* [Array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Array) (vetor) de caracteres, com tamanho igual a 4\*/

04 char letras[];

05 letras = new char[4];

06

07 /\* Adicionando elementos ao [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array)\*/

08 letras[0] = 'J';

09 letras[1] = 'a';

10 letras[2] = 'v';

11 letras[3] = 'a';

12

13 /\* Exibição do conteúdo do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) \*/

14 int indice = 0;

15 while(indice < 4){

16 System.out.println(letras[indice]);

17 indice++;

18 }

19 }

No exemplo, estamos declarando um [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) de caracteres na linha 4 e atribuímos a ele o tamanho na linha 5, sendo que a inicialização com valores é feita nas linhas 8 a 11. Portanto, perceba que o processo de inicializar um [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) é diferente do modo de inicialização de uma variável de tipo primitivo comum. Primeiro, é necessário declarar o [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) e, em seguida, utilizar a palavra-chave “new” para alocar o tamanho do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) no seu respectivo tipo. Assim, a Máquina Virtual Java (JVM) saberá quanto espaço em memória será necessário alocar para esse [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array). Nas linhas 8 a 11, atribuímos o valor de cada elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array), que são as letras 'J','a','v' e 'a'. Declarada a variável indice do tipo inteiro na linha 14, utilizamos um laço while (linha 15) para executar 4 vezes as instruções das linhas 16 e 17, que exibem um elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) e incrementam uma variável de controle indice, respectivamente. Em outras palavras, acessamos cada elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) letras através da variável inteira indice. Se esse método for executado, a palavra "Java" deve aparecer no console do usuário.

01 public void Exemplo2(){

02 /\* [Array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Array) (vetor) de inteiros, com tamanho igual a 50 \*/

03 int numeros[] = new int[50];

04 /\* Laço 'for' que preenche o vetor com valores de 0 a 50

05 \* sempre aumentando de 10 em 10 unidades

06 \*/

07 for(int i = 0; i <= numeros.length; i++){

08 numeros[i] = i \* 10;

09 }

10 }

No exemplo, estamos declarando um [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) de caracteres na linha 4 e atribuímos a ele o tamanho na linha 5, sendo que a inicialização com valores é feita nas linhas 8 a 11. Portanto, perceba que o processo de inicializar um [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) é diferente do modo de inicialização de uma variável de tipo primitivo comum. Primeiro, é necessário declarar o [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) e, em seguida, utilizar a palavra-chave “new” para alocar o tamanho do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) no seu respectivo tipo. Assim, a Máquina Virtual Java (JVM) saberá quanto espaço em memória será necessário alocar para esse [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array). Nas linhas 8 a 11, atribuímos o valor de cada elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array), que são as letras 'J','a','v' e 'a'. Declarada a variável indice do tipo inteiro na linha 14, utilizamos um laço while (linha 15) para executar 4 vezes as instruções das linhas 16 e 17, que exibem um elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) e incrementam uma variável de controle indice, respectivamente. Em outras palavras, acessamos cada elemento do [array](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=array) letras através da variável inteira indice. Se esse método for executado, a palavra "Java" deve aparecer no console do usuário.

## **Arrays Multidimensionais**

public class JavaMultidimensionais {

public void exemplo3(int quantidadeAlunos, int notasPorAluno) {

float notas[][] = new float[quantidadeAlunos][notasPorAluno];

for(int i = 0; i < quantidadeAlunos; i++) {

System.out.println("Notas do Aluno " + (i + 1));

for(int j = 0; j < notasPorAluno; j++) {

float temp = 0;

System.out.println("informe a nota " +(j+1) + " do Aluno " +(i+1) + ": ");

/\* Considere que o usuário irá formecer um valor para a variável temp aqui: \*/

/\* Atribuição da nota j para aluno i \*/

notas[i][j] = temp;

}

}

}

}

No código do método exemplo3, temos a criação de uma matriz de duas dimensões. Essa matriz armazenará as notas de vários alunos. A quantidade de alunos e de notas que cada um deve possuir é passada para o método como parâmetros, através das variáveis quantidadeAlunos e notasPorAluno, respectivamente. Para percorrer todas as posições dessa matriz, utilizamos duas variáveis, i e j, como contadores. Vamos considerar que a cada iteração do laço for interno (baseado na variável j), será coletado um valor de nota dado pelo usuário que será armazenado na variável temp, na linha 15. Na linha 18, atribuímos ao elemento da linha i e da coluna j da matriz o valor da variável temp. Esse exemplo está incompleto porque não explicitamos como requisitar dados ao usuário. Abordaremos sobre entrada de dados no próximo tópico.  
  
 Perceba que a criação e o acesso a elementos é muito similar para arrays unidimensionais (vetores) e multidimensionais (matrizes). Tenha sempre cuidado com a quantidade de dimensões quando utilizar matrizes.

A matriz do exemplo anterior pode ser representada pela figura a seguir. Supondo que as variáveis quantidadeAlunos e notasPorAluno tivessem os valores 6 e 4, respectivamente, cada linha da figura vai representar a lista de notas de um aluno, e cada coluna vai representar uma nota específica.

A imagem mostra uma representação gráfica de uma matriz de 6 linhas, numeradas de 0 a 5 e 4 colunas, numeradas 0 a 3. Nela, as colunas representam a quantidade de alunos e as linhas representam as notas por alunos.

