

## Programação de Sistemas I

Professora Regiane Moreno





# Sumário



INTRODUÇÃO	
MÉTODOS E ARRAYS EM JAVA	3
REFERÊNCIAS	15

## PROGRAMAÇÃO DE SISTEMAS I

## **INTRODUÇÃO**

#### MÉTODOS E ARRAYS EM JAVA

#### 1. Métodos

Uma função, chamada de método em Java, é sempre definida dentro de uma classe e tem como cabeçalho:

```
<modificadores><tipo-retorno><nome-função> (parâmetros recebidos) {
<implementação>
}
```

**Modificadores:** indicam a restrição de uso da função e podem ser public, private ou protected. Inicialmente, usaremos apenas o modificador public (a função pode ser chamada de qualquer lugar do programa, inclusive fora da classe que a criou/definiu). Mais a frente teremos uma aula só para tratar de modificadores de acesso, e aí esse assunto ficará mais claro.

função, além de executar uma tarefa, devolve algum resultado (de um cálculo, por exemplo). Deve-se informar, no cabeçalho da função, qual é o tipo do retorno da função:

Valor	Retorno
void	não retorna nada
tipo(int, float, char, entre outros)	define qual é o tipo do dado retornado pela rotina

Nome da função – por convenção, uma função deve iniciar com letra minúscula e, se tiver mais de uma palavra, estas, deverão começar com letra maiúscula (Exemplos: calcular, verSaldo, exibirNome).

Imagine que existe uma classe chamada Televisao que tem uma função chamada mudarCanal. Para que a função funcione adequadamente, é preciso informar, para essa função, qual é o novo número de canal desejado. A intenção da função seria:

"Me informe qual é o número do canal que você quer assistir, que eu mudarei para ele".

```
O cabeçalho desta função poderia ser:

public void mudarCanal (int nCanal) {

<comandos para mudar o canal>
}
```

#### Características da função:

public	acessível em qualquer local do programa/classe
void	não faz nenhum tipo de retorno
mudarCanal	nome da função
(int nCanal)	número do canal que se deseja assistir parâmetro para a função mudarCanal

Para fazer a correta chamada desta função, é necessário chamá-la pelo seu nome e passar o valor que será copiado para seu parâmetro:

#### mudarCanal(nro);

A variável **nro** contém o número do canal desejado (parâmetro passado à função). Lembre-se de que deve existir coerência entre o tipo do parâmetro passado para a função (nro) e o tipo informado no cabeçalho (int nCanal). Bem como deve existir coerência na quantidade de parâmetros passados e recebidos.

É importante observar que as atualizações feitas pelas funções, nos valores de parâmetros recebidos, não têm efeito nenhum no restante do programa.

Por exemplo, se uma variável x tem conteúdo 14 e é passado para uma função, que modifica este parâmetro, no retorno da função, esta variável continua valendo 14:

Por exemplo, se uma variável x tem conteúdo 14 e é passado para uma função, que modifica este parâmetro, no retorno da função, esta variável continua valendo 14:

```
x=14;
rot1(x);
System.out.println(x);
```

O valor de x que é 14 será copiado para o parâmetro z, porém, localmente, a variável x continua com o valor 14.

public void rotl (int z){

System.out.println(z);

z=42;

System.out.println(z);

}

z receberá 14, portanto, localmente, no método rot1, o valor de z será 14.

Dentro do método, o valor 14 de z será substituído por 42, devido à instrução z=42 e, no método, será impresso o valor 42.

A sequência de exibição será: 14 42 14

Na volta de rot1, será exibido o número 14, mesmo com a atribuição de 42 para o parâmetro z.

Veja mais alguns exemplos de cabeçalhos de função e analise as características de cada exemplo:

Exemplos de assinatura de método
public static void jogar ( ){
<comandos></comandos>

Chamada das funções públicas:

As chamadas de funções públicas podem ser feitas dentro e fora da classe que a criou. Se a função jogar() do exemplo anterior foi criada dentro de uma classe chamada JogoDaVelha, sua chamada DENTRO desta classe seria:

jogar();

Note que não existem parâmetros passados à função porque, em seu cabeçalho, não existem parâmetros informados. Caso a chamada fosse feita fora da classe JogoDaVelha, ela seria:

JogoDaVelha.jogar();

Nesse caso, é preciso informar qual é a classe que contém a função jogar().

Para se chamar um método corretamente, deve-se saber: seu nome, a quantidade e o tipo de cada parâmetro enviado. A esse conjunto de informações, atribui-se o nome de assinatura da rotina ou método. A linguagem JAVA permite mais de um método com o mesmo nome, porém, com assinaturas diferentes.

**Exemplo:** método, sem retorno, para calcular e exibir todos os divisores de um inteiro (>0)

/\* Exibe todos os divisores de N

\* inteiro maior que zero informado

```
*/
```

A chamada dessa função deve ser: exibeDivisores(N);

Retorno de funções:

É muito comum que métodos recebam parâmetros, executem processamentos com esses parâmetros e devolvam algum resultado. Nesse caso, a função deixa de ser do tipo void e passa a ser do tipo do retorno. Isto é, se o retorno é int, a função é do tipo int e assim por diante. O comando que retorna algum resultado é:

```
return <expressão>;
```

Lembre-se de que o comando return encerra a execução da função. Veja dois exemplos de funções que retornam valores:

#### Exemplo 1:

```
public static int proxN (int nAtual) {

if (nAtual \% 2 == 1)
```

```
return 3 * nAtual + 1;
else
return nAtual / 2;
```

Alguns programadores preferem ter um único comando return na rotina. Nesse caso, a rotina acima ficaria:

```
public static int proxN (int nAtual) {
  int resp;

if (nAtual % 2 == 1)
    resp = 3 * nAtual + 1;

else
    resp = nAtual / 2;

return resp; }
```

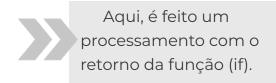
Nos dois exemplos, a chamada da função precisa prever o local de armazenamento do retorno ou de que forma esse retorno será tratado. Veja alguns exemplos para isso:

```
numero = proxN (valor);
System.out.println("O resultado é " +
```

proxN(valor));

Aqui, o retorno é armazenado em uma variável do tipo int chamada numero (a função retorna um valor inteiro – veja o cabeçalho da função).

Aqui, o retorno é exibido diretamente na tela.



Exemplo 2:

O programa, a seguir, faz a soma de dois números inteiros.

Observe que foi criado um método somatoria para retornar esta soma. Depois; esse método foi invocado no método main.

```
import java.util.Scanner;
public class Soma_2_Numeros {
    /*Exemplo de um método que soma dois números inteiros*/
  public static int somatoria(int a, int b) {
    return a + b;
  public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    int x, y, soma;
System.out.print("Digite o valor do lo número: ");
    x = entrada.nextInt();
```

```
System.out.print("Digite o valor do 20 número: ");

y = entrada.nextInt();

soma = somatoria(x, y);

System.out.println("A soma é: " + soma);

}
```

import java.util.Scanner

public class Soma\_2\_Numero

#### /\*Exemplo de um método que soma 2 números inteiros\*/

public static int

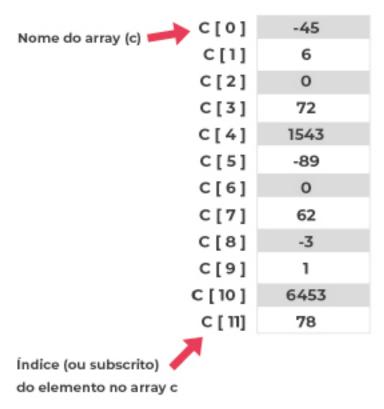
```
import java.util.Scan
                     Retorna um valor inteiro
public class Soma-2
/*Exemplo de m método que soma 2 números inteiros* /
public static int somatoria (int a, int b) {
   return a +b;
                                        Dois parâmetros
public static loid ma/String{} args {
Scanner entrada = w_Scanner(System in):
int x, y, soma;
                     Retorna a soma de
                      2 números inteiros
System. out.print("Di
x=entrada.nextInt()
System.out.print ("Digne
y=entrada.nextInt();
soma = somatoria(x,y);
System.out.printin("Asoma é:" +soma);
```

#### 2.Arrays

#### Conceitos teóricos:

- Arrays são estruturas de dados compostas por itens de dados do mesmo tipo.
- Arrays tornam-se conveniente para processar grupos relacionados de valores.
  - · O tamanho dos **arrays** permanece o mesmo depois de serem criados.

#### Exemplo de um vetor de inteiros chamado c, com 12 elementos:



Fonte: Deitel (2017, pág.193)

#### Declarando e criando arrays

Os objetos de **array** ocupam espaço na memória. Como os outros objetos, os **arrays** são criados com a palavra-chave **new**.

Para criar um objeto de **array**, especifique o tipo dos elementos do **array** e o número de elementos.

#### Exemplo:

```
int[] c: // declara a variável de array
  c = new int [12]; // cria o array: atribui à variável de array
ou
  int[] c = new int [12];
```

#### Criando e inicializando um array

Ao criar um array numérico, o padrão de inicialização dele é zero.

#### Exemplo:

```
// Figura 7.2: InitArray.java
1
2
     // Inicializando os elementos de um array como valores padrão de zero.
3
4
     public class IntArray
5
     public static void main(Stirng[] args)
6
7
8
     // declara array variável e o iniciava com um objeto array
     int [] array = new int [10]; // cria o objeto array
9
10
    System.out.printf("%s%8s%n", "Index". Value): // títulos de coluna
11
12
      // gera saída do valor de cada elemento do array
13
     for (int counter = 0: counter array, length; counter ++)
14
         SystemCrint( "%5d%8d%n", counter, array, [counter]):
15
16
       }
17 } // fim da classe InitArray
```

ı	Index	Value
١	0	0
١	1	0
١	2	0
١	3	0
١	4	0
١	5	0
١	6	0
	7	0

Fonte: Deitel (2017, pág.196).

A estrutura for aprimorada

A instrução for aprimorada itera pelos elementos de um array sem usar um contador, evitando, assim, a possibilidade de ultrapassar o limite do array.

```
Sintaxe:

for (parâmetro : nomeDoArray){

     <comandos>
}
```

Exemplo para somar os valores de um array usando o for convencional:

```
for ( int counter = 0; counter < array, length; counter ++)
total += array[counter];</pre>
```

A instrução for aprimorada só pode ser utilizada para obter elementos de array – ela não pode ser usada para modificar elementos. Se seu programa precisar modificar elementos, utilize a tradicional instrução for controlada por contador.

A instrução for aprimorada é indicada sempre que não for necessário o acesso ao índice do elemento do array.

Exemplo para somar os valores de um array usando o for aprimorado:

```
// Figura 7.12: IEnhacedForTest.java
1
2
      // Utilizando a instrução for aprimorada para somar inteiros em um array.
3
4
      public class EnhancedForTest
5
      public static void main(Stirng[] args)
6
7
      int [] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 }; // cria o objeto array
8
9
     int total = 0;
10
     // adiciona o valor de cada elemento ao total
11
12
    for (int number: array)
13
    total += number:
```

Fonte: Deitel (2017, pág.206).

### **REFERÊNCIAS**

DEITEL, H.; DEITEL, *P. Java – Como Programar.* 10. ed. São Paulo: Pearson Prentice Hall, 2017. ISBN 9788543004792

HORSTMANN, C. S.; *CORNELL, G. Core Java*. 8. ed. São Paulo: Pearson Prentice Hall, 2010. ISBN 978857603576