**Funções**

As funções também são conhecidas como métodos. Elas são blocos de código que realizam tarefas específicas dentro de um programa em Java. Esses métodos são reutilizáveis e podem ser chamados em diferentes partes do código, facilitando a organização e modularização. A estrutura básica de uma função inclui a definição do tipo de retorno, nome do método, parâmetros e o corpo do método.

*tipoDeRetorno nomeDoMetodo(tipoParametro1 nomeParametro1, tipoParametro2 nomeParametro2, …) {  
    // Corpo do método  
    return valorRetornado; // se tipoDeRetorno não for void  
}*

**Procedimentos e Funções**

Em Java, a distinção entre procedimentos e funções se dá principalmente pela presença ou ausência de um valor de retorno.

**Função:**É um método que devolve um valor ao final da sua execução, como no exemplo abaixo:

*int soma(int a, int b) {  
    return a + b;  
}*

**Procedimento:**Também conhecido como função void, não retorna valor e é usado para realizar uma tarefa sem necessidade de devolver uma resposta.

*void imprimirMensagem(String mensagem) {  
    System.out.println(mensagem);  
}*

**Passagem de Parâmetros**

Os parâmetros podem ser passados para os métodos em Java de duas formas:

• Passagem por Valor: Quando se trabalha com tipos primitivos (int, float, char etc.), o valor é copiado e passado ao método. Qualquer modificação no parâmetro dentro do método não afeta o valor original fora do método.

• Passagem por Referência: Ao passar objetos, Java passa uma referência ao objeto. Isso significa que as mudanças no estado do objeto dentro do método afetam o objeto original, mas a referência não pode ser alterada dentro do método.

Abaixo temos um exemplo com dois métodos:

1. modificarValorPrimitivo: Recebe um número do tipo int (primitivo). Como a passagem é feita por valor, o método recebe uma cópia do número e qualquer alteração feita dentro do método não afeta o valor original fora dele.

2. modificarValorObjeto: Recebe um objeto do tipo Pessoa. Como a passagem de objetos em Java é feita por referência, o método pode modificar o estado do objeto (neste caso, o nome da pessoa). A alteração será refletida fora do método, já que a referência aponta para o mesmo objeto.

A saída do programa irá ilustrar em detalhes a diferença entre os dois tipos de passagem.

A passagem de parâmetros em Java varia dependendo do tipo de dado que está sendo passado. Entender essa diferença é importante para escrever um código que evite erros inesperados ao manipular dados em métodos.

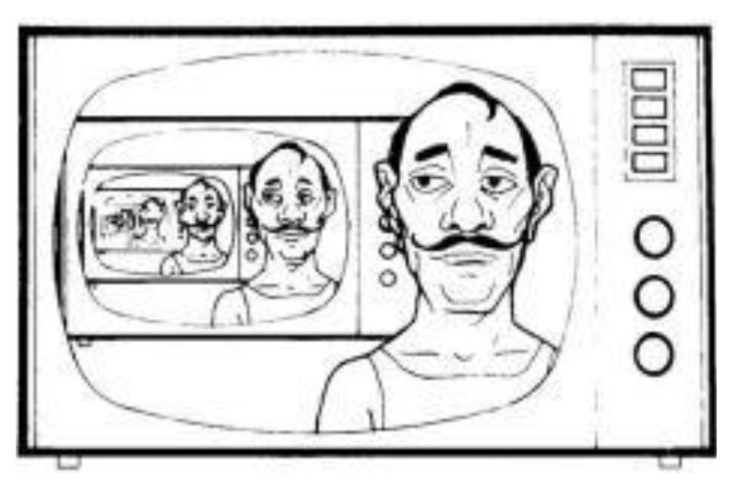
*class ExemploPassagemParametros {  
    // Método que tenta modificar um valor primitivo (passagem por valor)  
    public void modificarValorPrimitivo(int numero) {  
        numero = numero + 10;  
        System.out.println("Dentro do método (valor primitivo): " + numero);  
    }  
    // Método que modifica o estado de um objeto (passagem por referência)  
    public void modificarValorObjeto(Pessoa pessoa) {  
        pessoa.nome = “Maria”;  
        System.out.println("Dentro do método (objeto): " + pessoa.nome);  
    }  
    public static void main(String[] args) {  
        ExemploPassagemParametros exemplo = new ExemploPassagemParametros();*

*// Passagem por valor (primitivo)  
        int numero = 5;  
        System.out.println("Antes do método (valor primitivo): " + numero);  
        exemplo.modificarValorPrimitivo(numero);  
        System.out.println("Depois do método (valor primitivo): " + numero);*

*// Passagem por referência (objeto)  
        Pessoa pessoa = new Pessoa();  
        pessoa.nome = “João”;  
        System.out.println("Antes do método (objeto): " + pessoa.nome);  
        exemplo.modificarValorObjeto(pessoa);  
        System.out.println("Depois do método (objeto): " + pessoa.nome);  
    }  
}*

*class Pessoa {  
    String nome;  
}*

**Funções Recursivas**



**Fonte:** Computador de Papel, 2022.

**Recursão**

A recursão é uma técnica poderosa que pode ser utilizada para resolver problemas que podem ser divididos em subproblemas menores. E consiste em uma técnica onde uma função chama a si mesma para resolver problemas que podem ser divididos em subproblemas menores. Um exemplo clássico de função recursiva é o cálculo de fatoriais.

Exemplo de função recursiva para calcular o fatorial de um número:

*int fatorial(int n) {  
    if (n == 0 || n == 1) {  
        return 1;  
    } else {  
        return n \* fatorial(n - 1);  
    }  
}*

A recursão é especialmente útil para resolver problemas como cálculo de Fibonacci, operações com árvores e grafos e qualquer outro requisito que precise ser decomposto em subproblemas menores. Esse tipo de implementação deixa seu código mais elegante, mas aumenta a complexidade e exige uma maturidade grande de entendimento do código.

**Funções Lambda**

As funções lambda, introduzidas no Java 8, oferecem uma maneira concisa de implementar interfaces funcionais, ou seja, interfaces que possuem apenas um método abstrato. Elas simplificam a passagem de comportamento como argumento para métodos e tornam o código mais enxuto.

A sintaxe de uma função lambda é:

(parâmetros) -> { corpo }

**Exemplo de uma função lambda com um único parâmetro:**

x -> x \* 2

As funções lambda são amplamente utilizadas em expressões funcionais e operações sobre coleções, tornando o código mais legível e expressivo.

O uso de funções, procedimentos, recursão e lambdas em Java oferece grande flexibilidade e eficiência na construção de programas. Essas técnicas ajudam a organizar o código e a resolver problemas complexos de forma modular e eficiente.

**Conteúdo Bônus**

**Título:** Curso de Java - Recursividade - Aula 20

**Canal:** Rodrigo Freitas

**Plataforma**: YouTube

**Descrição:** O método recursivo é muito útil quando você quer que um método chame a si mesmo em Java. Neste vídeo, são apresentados os cuidados que devem ser tomados ao utilizar a recursividade. O curso de Java possui vários vídeos além da recursividade.

**Referências Bibliográficas**

ASCENCIA, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação**: algoritmos, Pascal, C/C++ e Java. 2. ed. Pearson, 2007.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. **Sistemas operacionais**. 3. ed. Pearson, 2005.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. Pearson, 2005.

GUEDES, S. (Org.). **Lógica de programação algorítmica**. Pearson, 2014.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações móveis**: arquitetura, projetos e desenvolvimento. Pearson, 2005.

MELO, A. C. V. de; SILVA, F. S. C. da.**Princípios de linguagens de programação**. Blucher, 2014.

MENEZES, A. M. de. **Os paradigmas de aprendizagem de algoritmo computacional**. Blucher, 2015.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados, com aplicações em Java**. Pearson, 2016.