



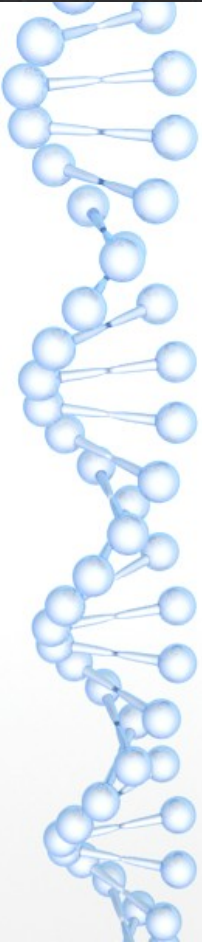
Funções

Modularização

Prof. Dr. Bruno Queiroz Pinto



Definições de Funções

- 
- Divide grandes tarefas da computação em tarefas menores.
 - Evita repetição do código:
 - O trecho de código que faz parte da função pode ser chamado várias vezes.
 - Facilita a manutenção de código:
 - O trecho de código que sofrerá mudanças estará presente em apenas um lugar.
 - Evita códigos longos
 - Um programa bem estruturado deve ser pensado em termos de funções.

Dividir o projeto em partes

Solução não modularizada



```
comando 1
comando 2
comando 3
comando 4
comando 5
comando 6
comando 7
comando 8
comando 9
comando 10
comando 11
comando 12
comando 13
comando 14
comando 15
comando 16
comando 17
comando 18
comando 19
comando 20
comando 21
comando 22
comando 23
comando 24
comando 25
```

Suponha um programa qualquer. Ele pode ser resolvido por meio de um “programão” ou então este “programão” pode ser dividido em partes.

Solução modularizada

```
comando 1
comando 2
comando 3
comando 4
comando 5
comando 6
funcao1();
comando 11
comando 12
comando 13
comando 14
comando 15
funcao2();
comando 23
comando 24
comando 25
```

```
funcao1() {
    comando 7
    comando 8
    comando 9
    comando 10
}

funcao2() {
    comando 16
    comando 17
    comando 18
    comando 19
    comando 20
    comando 21
    comando 22
}
```

Definições de Funções

→ Assinatura de uma função:

```
static tipo_retornado nome_da_função (lista de parâmetros...) {  
    corpo da função  
}
```

Caso não haja retorno (void):

```
nome_da_função(parâmetros);
```

Definições de Funções



→ Exemplo:

```
public class Main {  
    public static void main(String[] args) {  
        mensagemBoasVindas()  
    }  
  
    static void mensagemBoasVindas(){  
        System.out.println("Hello World");  
    }  
}
```

Exemplo de Funções

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         funcaoSemRetorno();
5     }
6
7     static void funcaoSemRetorno(){
8         System.out.println("Hello World");
9     }
10 }
```

Com retorno

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String texto = funcaoComRetorno();
5         System.out.println(texto);
6     }
7
8     static String funcaoComRetorno(){
9         return "Hello World";
10    }
11 }
```

Procedimento

Sem retorno

Exemplo de Funções com retorno

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         System.out.println(funcaoComRetorno());
5     }
6
7     static String funcaoComRetorno(){
8         return "Hello World";
9     }
10 }
```

Retorno utilizado na hora

Recomendado quando o valor é utilizado uma única vez.

Retorno salvo em variável

Recomendado quando o valor é utilizado várias vezes.

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String texto = funcaoComRetorno();
5         texto = "Mensagem : " + texto;
6         System.out.println(texto);
7     }
8
9     static String funcaoComRetorno(){
10         return "Hello World";
11     }
12 }
```



Passagem de Parâmetro

- Por valor:
 - É possível passar um ou mais valores para serem usados para processamentos dentro da função.
 - As funções aceitam parâmetros de diversos tipos:
 - int, double, float, string, Vetor, Matriz, etc.

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         imprimeMensagem("Oi Mundo!!!!");
5     }
6
7     static void imprimeMensagem(String texto){
8         System.out.println("Mensagem : " + texto);
9     }
10 }
```


Passagem de Parâmetro

- Por valor:
 - A modificação do valor na função não alterar o valor da variável passada como parâmetro.

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String original = "Oi Mundo!!!!";
5         imprimeMensagem(original);
6         System.out.println(original);
7     }
8
9     static void imprimeMensagem(String texto){
10         texto = "Mensagem : " + texto;
11         System.out.println(texto);
12     }
13 }
```

Mensagem : Oi mundo!!!!

Oi mundo!!!!

Passagem de Parâmetro

- Por referência:
 - A modificação do valor na função altera o valor da variável passada como parâmetro.
 - Ocorre em vetores e matrizes, por exemplo.

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String original[] = new String [2];
5         original [0] = "Oi Mundo!!!!";
6         imprimeMensagem(original);
7         System.out.println(original[0]);
8     }
9
10    static void imprimeMensagem(String texto[]){
11        texto[0] = "Mensagem : " + texto[0];
12        System.out.println(texto[0]);
13    }
14 }
```

Mensagem : Oi mundo!!!!

Mensagem : Oi mundo!!!!

Outra forma de declarar e inicializar:
`String original[] = new String[] {"Oi Mundo!!!!"};`



Retorno de valores

- A função pode retornar um valor para o local onde ela é chamada
 - Nesse caso, o tipo da função deverá ser o mesmo do tipo de retorno.
 - O tipo de retorno pode ser uma variável primitiva, vetor ou matriz.

Retorno de valores

- Passagem por referência e **retorna uma String**

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String original[] = new String[] {"Oi Mundo!!!!"};
5         System.out.println(retornaMensagem(original));
6         System.out.println(original[0]);
7     }
8
9     static String retornaMensagem(String texto[]){
10         texto[0] = "Mensagem : " + texto[0];
11         return texto[0];
12     }
13 }
```

Retorno de valores

- Passagem por referência e **retorna um novo vetor**

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String original[] = new String[] {"Oi Mundo!!!!"};
5         String nova[] = retornaVetor(original);
6         System.out.println(nova[0]);
7         System.out.println(original[0]);
8     }
9
10    static String[] retornaVetor(String texto[]){
11        String nova[] = new String[texto.length];
12        nova[0] = "mensagem nova : " + texto[0];
13        texto[0] = "mensagem antiga : " + texto[0];
14        return nova;
15    }
16 }
```

Problema

Faça um programa que receba as medidas da base e altura de um retângulo. Em seguida, apresente o menu de opções a seguir:

Menu de opções:

- 1 – Mostrar área
- 2 – Mostrar perímetro
- 3 – Sair

Digite a opção desejada:

Fazer um mecanismo para que o menu fique se repetindo até que o usuário escolha a opção 3.

Trabalhar com menus

Solução não modularizada

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        double base, altura, resposta;  
        int op;  
        System.out.print("Digite o valor da base: ");  
        base = s.nextDouble();  
        System.out.print("Digite o valor da altura: ");  
        altura = s.nextDouble();  
        op = 0;  
        while (op != 3) {  
            System.out.println("Menu de opções:");  
            System.out.println("1 - Mostrar área");  
            System.out.println("2 - Mostrar perímetro");  
            System.out.println("3 - Sair");  
            System.out.print("Digite a opção desejada: ");  
            op = s.nextInt();  
  
            if (op == 1) {  
                resposta = base * altura;  
                System.out.println("Area = " + resposta);  
            } else if (op == 2) {  
                resposta = 2 * (base + altura);  
                System.out.println("Perimetro = " + resposta);  
            } else if (op == 3) {  
                System.out.println("Fim do programa!");  
            } else {  
                System.out.println("Opcao invalida!");  
            }  
        }  
    }  
}
```

Solução não modularizada

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        double base, altura, resposta;  
        int op;  
        System.out.print("Digite o valor da base: ");  
        base = s.nextDouble();  
        System.out.print("Digite o valor da altura: ");  
        altura = s.nextDouble();  
        op = 0;  
        while (op != 3) {  
            System.out.println("Menu de opções:");  
            System.out.println("1 - Mostrar área");  
            System.out.println("2 - Mostrar perímetro");  
            System.out.println("3 - Sair");  
            System.out.print("Digite a opção desejada: ");  
            op = s.nextInt();  
  
            if (op == 1) {  
                resposta = base * altura;  
                System.out.println("Área = " + resposta);  
            } else if (op == 2) {  
                resposta = 2 * (base + altura);  
                System.out.println("Perímetro = " + resposta);  
            } else if (op == 3) {  
                System.out.println("Fim do programa!");  
            } else {  
                System.out.println("Opção inválida!");  
            }  
        }  
    }  
}
```

Solução modularizada

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        double base, altura;  
        int op;  
        System.out.print("Digite o valor da base: ");  
        base = s.nextDouble();  
        System.out.print("Digite o valor da altura: ");  
        altura = s.nextDouble();  
  
        op = 0;  
        while (op != 3) {  
            mostrarMenu();  
            op = s.nextInt();  
            tratarMenu(op, base, altura);  
        }  
    }  
}
```

Perceba que tentamos utilizar nomes sugestivos para as funções.

Solução modularizada

```
public class Main {  
    // AS FUNCOES VEM AQUI  
    (aguarde...)  
  
    // A SEGUIR O PROGRAMA PRINCIPAL  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        double base, altura;  
        int op;  
        System.out.print("Digite o valor da base: ");  
        base = s.nextDouble();  
        System.out.print("Digite o valor da altura: ");  
        altura = s.nextDouble();  
  
        op = 0;  
        while (op != 3) {  
            mostrarMenu();  
  
            op = s.nextInt();  
  
            tratarMenu(op, base, altura);  
        }  
    }  
}
```

Primeira vantagem da modularização

Um programa modularizado fica mais **organizado** e **legível**.

Conseguimos **abstrair** melhor o problema, jogando partes da solução para outro lugar, deixando no programa principal apenas as chamadas das funções.

Implementação das funções

```
public static void mostrarMenu() {
    System.out.println("Menu de opções:");
    System.out.println("1 - Mostrar área");
    System.out.println("2 - Mostrar perímetro");
    System.out.println("3 - Sair");
    System.out.print("Digite a opção desejada: ");
}

public static void tratarMenu(int op, double base, double altura) {
    double resposta;
    if (op == 1) {
        resposta = base * altura;
        System.out.println("Area = " + resposta);
    } else if (op == 2) {
        resposta = 2 * (base + altura);
        System.out.println("Perimetro = " + resposta);
    } else if (op == 3) {
        System.out.println("Fim do programa!");
    } else {
        System.out.println("Opcao invalida!");
    }
}
```

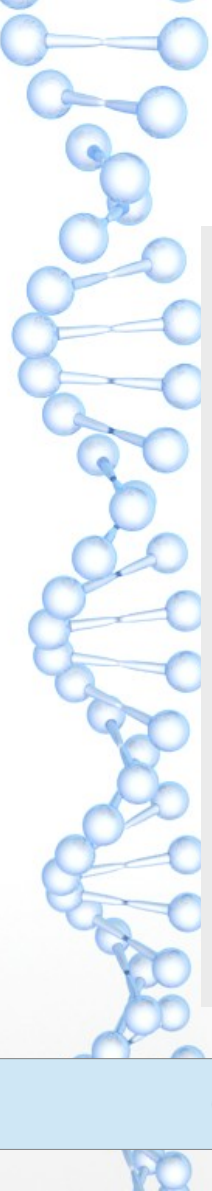


Exercício

OnlineGDB : Funções – Exemplo01

OnlineGDB : Funções – Exemplo02

Manipular matrizes em funções



```
1 public class Main
2 {
3     public static void main(String[] args) {
4         String original[][] = new String[3][2];
5         original[0][0] = "Oi";
6         original[0][1] = "Mundo!!!";
7         String nova[][] = retornaVetor(original);
8         System.out.println(nova[0][0] + " " + nova[0][1]);
9         System.out.println(original[0][0] + " " + original[0][1]);
10    }
11
12    static String[][] retornaVetor(String texto[][]){
13        String nova[][] = new String[texto.length][texto[0].length];
14        nova[0][0] = "Hello";
15        nova[0][1] = "World!";
16        texto[0][0] = "OI";
17        texto[0][1] = "MUNDO!!!";
18        return nova;
19    }
20 }
```

Observe a forma de descobrir o tamanho da matriz... linha 13



Exercício

OnlineGDB : Funções – Exemplo03

OnlineGDB : Funções – Exemplo04



Sobrecarga de funções

- Podem ser declaradas duas funções com o mesmo nome e parâmetros diferentes.
- Exemplo:
 - Função somaVal:
 - Soma dois valores inteiros ou
 - Soma dois valores decimais ou
 - Soma três valores inteiros.