CIÊNCIAS DA COMPUTAÇÃO

FACULDADE DE CIÊNCIAS | UNIVERSIDADE AGOSTINHO NETO

FUNDAMENTOS DE PROGRAMAÇÃO

Docente:

- ✓ Lufialuiso Sampaio Velho, MSc.

Monitor

- ✓ João Pedro

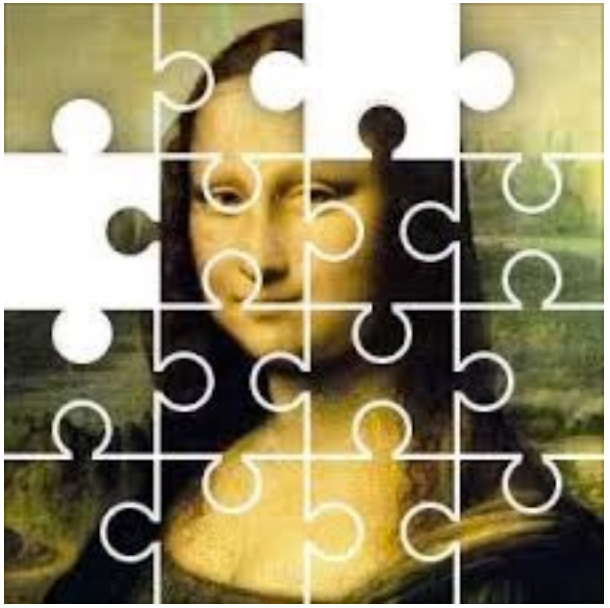
Conteúdo

Cap. VI

Modularização em Java

CAP. VI - Modularização

Modularizar consiste em repartir um problema em sub-problemas, onde a solução de todos os sub-problemas depois de associados, resolvem o problema geral.



CAP. VI - Modularização

Em programação, a modularização é aplicada utilizando métodos (procedimento ou função). Cada módulo é também denominado sub-rotina ou método e pode ser desenvolvido de forma independente.

Estes métodos podem ser:

Com retorno (Função) ou **sem retorno (void)** (procedimento).

CAP. VI - Modularização

Desvantagens de não modularizar

- São construídos programas bastante extensos que podem dificultar a compreensão do código.
- A alteração das variáveis é feita directamente
- Um único programador é sacrificado para a implementação da solução
- Não há reutilização do código
- Dificulta a localização e correcção de erros (depuração)
- Programas pouco eficientes

CAP. VI - Modularização

Vantagens da modularização

- Facilidade na solução de problemas complexos;
- Reutilização de Código (codificar uma única vez e utilizar quantas vezes desejar);
- Facilidade na compreensão do código;
- Permite que o trabalho seja feito em equipa (solução de pequenos problemas separadamente por cada membro da equipa).

Vantagens da modularização (cont.)

- Facilidade na depuração do código;
- Utilização de parâmetros para permitir trabalhar com a cópia das variáveis;
- Para o utilizador do módulo, interessa apenas o COMO utilizar e não o COMO foi implementado (caso das API).

CAP. VI - Modularização

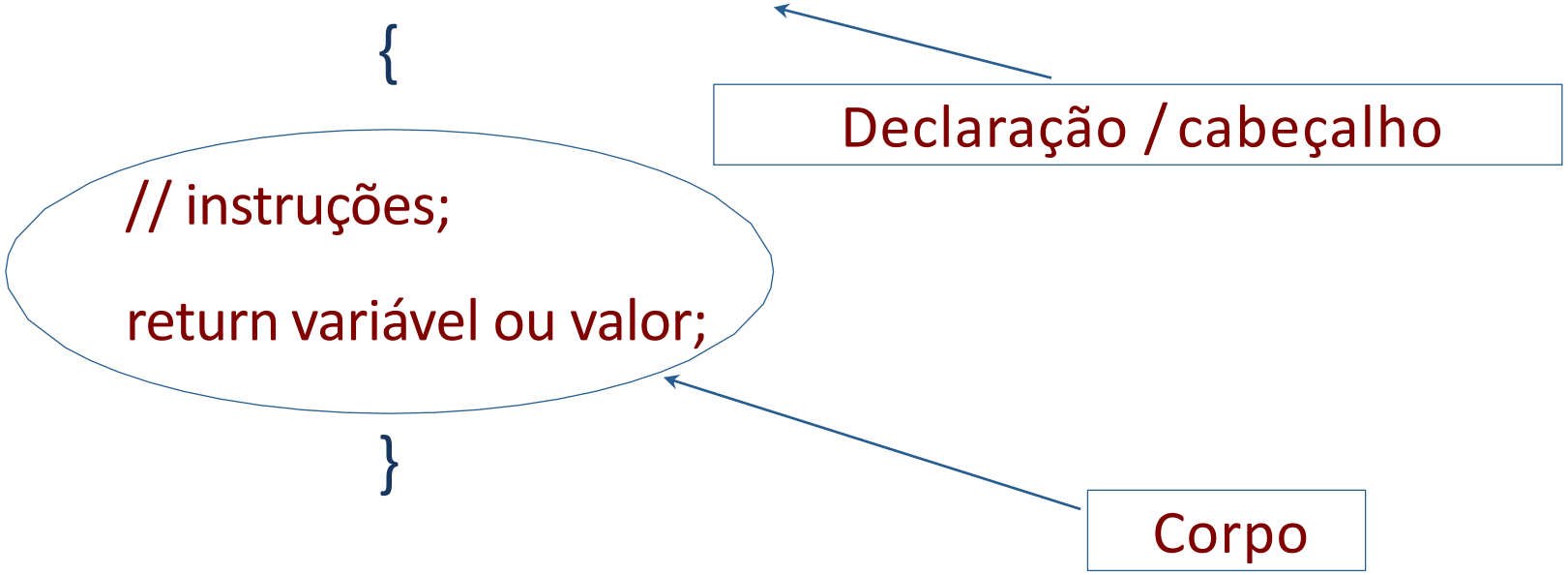
Funções (Definição e Declaração)

É sequência de instruções bem definidas que efectuam determinado cálculo ou operação e no final retorna explicitamente um valor ao exterior. Este valor deve ser **obrigatoriamente** do tipo declarado na função.

Ex: Cálculo do factorial, média de um estudante, soma de números, etc.

Sintaxe:

```
public static tipo nome(parâmetros)
{
// instruções;
return variável ou valor;
}
```



CAP. VI – Modularização

Exemplo de uma função (método com retorno)

```
import java.util.Scanner;
public class P_programa{
publicstatic int factorial(int num){
int fact=1;
if(num==0)
return 1;
for(int i=1;i<=num;i++){
fact*=I; }
return fact; }
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
System.out.println("Digite um inteiro não negativo");
int n=teclado.nextInt();
if(n>=0){
System.out.print(factorial(n));
}else
System.out.print("Não existe factorial de números negativos"); }}
```

Parâmetro

Este bloco representa a criação de uma função.

Os métodos (função ou procedimento) devem ser declarados após a classe, e podem ou não ter parâmetros.

Chamada da função

Argumento

Programa principal

Procedimentos (Definição e Declaração)

Define-se como uma sequência de instruções bem definidas que executam instruções sem o retorno de um valor para o exterior. Ex: menu de opções, leitura de dados, impressão de um resultado, etc.

CAP. VI - Modularização

Sintaxe:

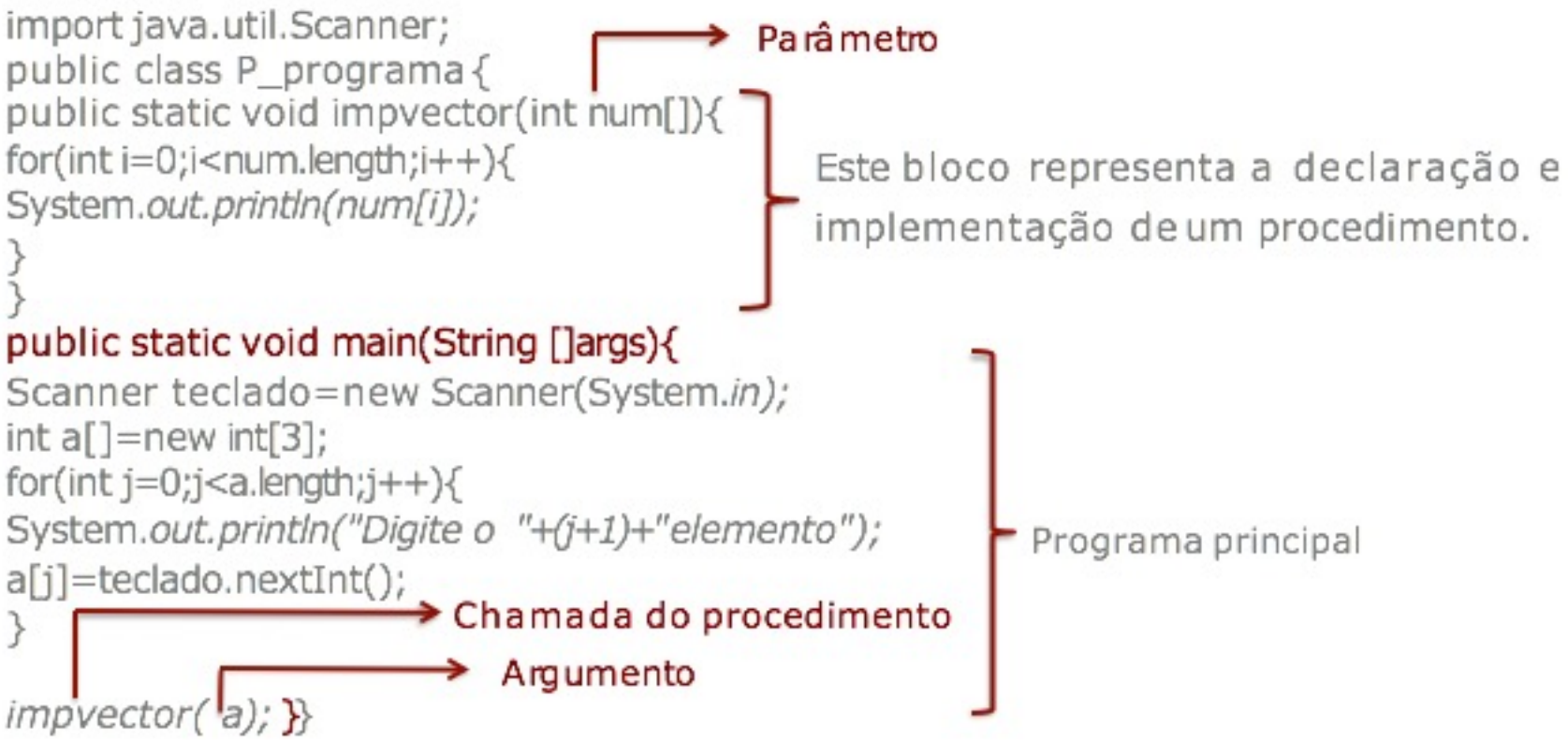
```
public static void nome (parâmetros) {

    Instruções;

}
```

Declaração / cabeçalho

Corpo

CAP. VI - Modularização

Exemplo de um procedimento (método sem retorno)




```
import java.util.Scanner;
public class P_programa{
public static void impvector(int num[]){
for(int i=0;i<num.length;i++){
System.out.println(num[i]);
}
}
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
int a[]=new int[3];
for(int j=0;j<a.length;j++){
System.out.println("Digite o  "+(j+1)+"elemento");
a[j]=teclado.nextInt();
}
impvector(a); }}
```

Parâmetro e argumento

Parâmetros e argumentos, definem o meio pelo qual os dados podem ser introduzidos dentro de um método. Importa não confundir com as variáveis locais ou globais (abordado mais adiante).

Parâmetro: é definido no instante em que declaramos ou construímos o protótipo do método. Este possui um tipo e um identificador.

CAP. VI - Modularização

Exemplo:

Parâmetros

```
public static int contNumero(int x, int y){
    ...
    }
```

Estes podem ser simples variáveis, vectores, matrizes, objectos, etc.

Parâmetro e argumento

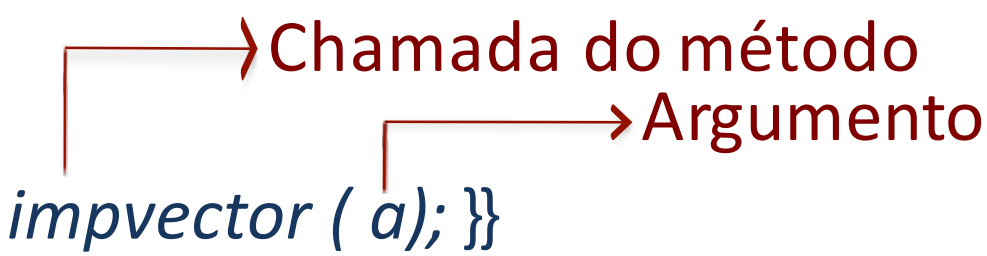
Argumento: define o valor concreto do parâmetro declarado no método. Diferente do parâmetro que é definido na criação do método, este (argumento) existe no momento em que invocamos o método no programa principal ou no interior de outro método e deve ser do mesmo tipo que o parâmetro.

Um argumento pode ser uma constante, variável do tipo primitivo ou uma referência.

CAP. VI - Modularização

Exemplo:

```
// foram ocultadas as instruções iniciais
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
int a[]=new int[3];
for(int j=0;j<a.length;j++){
System.out.println("Digite o "+(j+1)+" elemento");
 a[j]=teclado.nextInt();}
```



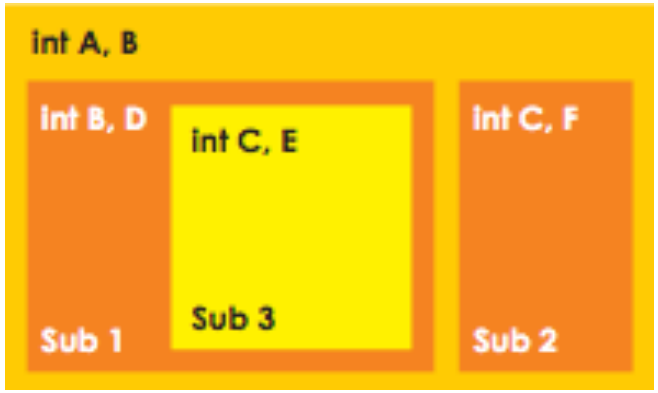
CAP. VI - Modularização

Escopo de uma Variável

Representa a visibilidade ou o raio de acção que uma variável ocupa dentro do programa ou método. Uma variável pode ser Local ou Global.

Variável Local: possui visibilidade limitada, são declaradas normalmente no interior de uma estrutura de selecção, repetição, método, ou ainda como parâmetro de um método. Isto implica que após o bloco em que ela foi declarada, esta variável não será reconhecida.

Ex:



Suponha as variáveis declaradas no interior do rectângulo à esquerda.

A,B são variáveis globais comparadas às do bloco sub1, sub2 e sub3.

C,E são variáveis locais do bloco sub 3.

CAP. VI - Modularização

Escopo de uma Variável

Variável Global: são declaradas após a criação da classe, e podem ser acedidas em toda a parte do programa. Isto implica que podemos acedê-las em qualquer parte do programa.

Elas são geralmente associadas ao modificador static (será abordado mais adiante) durante a sua declaração para indicar que é uma variável da classe.

Ex: Suponha o programa abaixo

```
public class Exercicio2 {
(1*) static int k;
k=0;
(2*) for(int i=1; i<=10 ; i++){
if (i%2== 0){
k=k+1;}}
System.out.print(k);}}
```

(1*) Declaração de uma variável global.

public static void main(String[]args){

(2*) A variável que controla o ciclo for, tem visibilidade apenas dentro do ciclo. Isto implica dizer que ela não é permitida fora do ciclo sem que seja novamente declarada.

Passagem de parâmetro

É uma operação efectuada na chamada de um procedimento ou função onde os parâmetros são substituídos por valores reais em forma de variáveis ou constantes para permitir a execução do subprograma. Existem duas espécies de passagem de parâmetros: por valor e por referência.

Passagem por valor: ocorre quando na chamada de um subprograma é passada uma cópia da variável ou do valor como argumento, não causando alteração ao conteúdo da variável original.

Analise o exemplo com atenção

CAP. VI - Modularização

Passagem de parâmetro por valor (exemplo)

```
import java.util.Scanner;
public class P_programa {
public static void testeValor (int num){
num=num*2;
System.out.println("Resultado do procedimento "+num);
}
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
System.out.println("Digite um inteiro");
int a=teclado.nextInt();
 testeValor(a);
System.out.print("Valor da variável é: "+a+" repare, não houve alteração");
}}
```

CAP. VI - Modularização

Passagem por referência: ocorre quando na chamada de um subprograma é passada a localização (endereço) na memória de uma variável, podendo assim alterar por completo o conteúdo antes armazenado nela.

Em java **não existe** o conceito de passagem por referência. Todas as passagens de parâmetro são efectuadas apenas por valor.

CAP. VII. Métodos- Exercícios

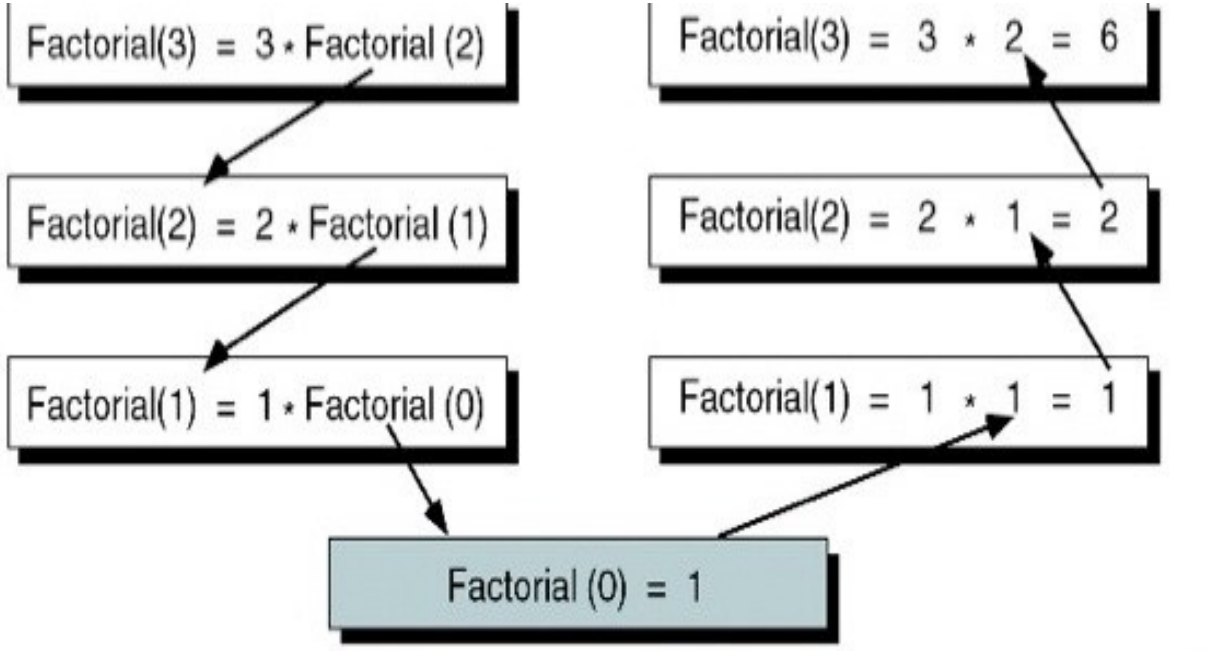
1. Crie um programa (utilizando métodos) que recebe um vector de inteiros e imprime na tela todos os números impares múltiplos de três existentes no vector.

CAP. VII. Métodos Recursivos

Uma função é considerada recursiva quando esta é capaz de invocar a si mesma afim de solucionar um determinado problema. A solução parte da decomposição de um problema por vários do mesmo tipo, e no final constrói-se a solução geral por intermédio das pequenas soluções.

Isto é, a decomposição do problema é feita do topo para a base e a solução obtém-se da base para o topo.

Um dos grandes exemplos do uso da recursão é o cálculo do factorial. Vejamos:



Regras para a implementação de uma função recursiva

1. Primeiramente determina o caso Base (caso por meio do qual a solução é construída)
2. Determina o caso genérico (o modo como o problema deve ser resolvido)
3. Combine o 1º e o 2º ponto num só algoritmo

Cada chamada recursiva reduz o tamanho do problema e tenderá ao caso base.

O caso base deve terminar o algoritmo, retornando no final a solução do problema.

Exemplo: Crie uma função em Java que retorna a soma dos números de 1 à n.

Função sem recursividade

```
public static int recfuncao (int num){
int soma=0;
if(num==1)
return 1;
for(int i=1;i<=num;i++){
soma=soma+i;
}
return soma;
}
```

Função com recursividade

```
public static int recfuncao (int num){
if(num==1)
return 1;
return num+recfuncao(num-1);
}
```

Exemplo2: Crie uma função em Java que imprime todos os números no intervalo de A à B inclusive.

Função sem recursividade

```
public static void imprime(int a,int b)
{
for(int i=a;i<=b;i++){
System.out.println(i);
}
}
```

Função com recursividade

```
public static void imprime(int a,int b)
{
if(a<=b){
System.out.println(a);
a++;
imprime(a,b);
}
}
```

Próxima Aula – Classes