Desenvolvimento Modular de Procedimentos

Em Java os Procedimentos

implementam-se com

Métodos

Maria da Conceição Neves

Desenho Modular de Procedimentos

- A melhor metodologia para o desenvolvimento de procedimentos com alguma dimensão é modularizá-los, isto é, decompô-los em procedimentos mais simples.
- Nas linguagens orientadas a objectos como o Java os procedimentos são chamados **métodos**.
- Em C estes procedimentos são chamados **funções**.

Maria da Conceição Neves

Tipos de Procedimentos

- Um procedimento é um bloco de instruções que executa uma tarefa específica.
- Existem dois tipos de procedimentos:
  - **Função** – que executa um bloco de instruções e **retorna um valor** para a linha de código que a chamou através da instrucão **return** valor.
  - **Subrotina (Procedure)** – apenas executa um bloco de instruções.
- Na programação OO os procedimentos designam-se por métodos.
  - **Há métodos que retornam valor**

    EX:

    ```
    public long factorial ( long numero ) { … }
    ```
  - **Há métodos que não retornam valor**

    EX:

    ```
    public void mostrar ( int valor ) { … }
    ```

Maria da Conceição Neves

Dividir, Reutilizar e Abstrair

- A utilização da modularização cumpre 3 objectivos fundamentais da programação
  - **Dividir** em partes mais pequenas e mais fáceis de codificar e testar
    - Pemite divisão do programa por uma equipa de programadores, cada um concentrado um objectivo mais específico.
  - Permitir a **reutilização** do código
    - Torna o desenvolvimento de aplicações mais rápido
  - Implementar o conceito de **abstracção**
    - Um método comporta-se como uma "caixa negra" não interessa como funciona só interessa qual o resultado esperado. Liberta dos detalhes.

Maria da Conceição Neves

Analise os dois Algoritmos

```
INICIO
ler(nota)

SE(nota<10)
   ENTÃO
     escrever ("Reprovado")
   SENÃO
        SE(nota<16)
          ENTÃO
          escrever (
            "Admitido a exame")
          SENÃO
          escrever (
            "Dispensado a exame")
        FIMSE
 FIMSE
FIM
```

```
INICIO
ler(nota1, nota2, nota3)
media<-(nota1+nota2+nota3)/3
SE(media<10)
   ENTÃO
     escrever ("Reprovado")
   SENÃO
        SE(media<16)
          ENTÃO
          escrever (
            "Admitido a exame")
          SENÃO
          escrever (
            "Dispensado a exame")
        FIMSE
 FIMSE
FIM
```

Maria da Conceição Neves

Modularizar - Definir Procedimento e Função

```
DEFINIR PResultado (val)
     SE(val<10)
       ENTÃO
         escrever ("Reprovado")
       SENÃO
         SE(val<16)
            ENTÃO
               escrever (
               "Admitido a exame")
            SENÃO
               escrever (
              "Dispensado a exame")
         FIMSE
     FIMSE
FIMDEFINIR
```

```
DEFINIR FResultado (nota)
     SE(nota<10)
       ENTÃO
            result= "Reprovado"
       SENÃO
         SE(nota<16)
            ENTÃO
               result= "Admitido a exame"

            SENÃO
              result= "Dispensado a exame"

         FIMSE
     FIMSE
   Retorna result
FIMDEFINIR
```

Maria da Conceição Neves

Algoritmos

```
INICIO
  ler(nota)
  PResultado(nota)
FIM
------------ Ou --------
INICIO
  ler(nota)
  res= FResultado(nota)
  escrever( res)
FIM
------------ Ou --------
INICIO
  ler(nota)
  Escrever( FResultado(nota))
FIM
```

```
INICIO
  ler(nota1, nota2, nota3)
  media<-(nota1+nota2+nota3)/3
  PResultado(media)
FIM
--------- Ou ----------------
INICIO
  ler(nota1, nota2, nota3)
  media<-(nota1+nota2+nota3)/3
  res= FResultado(media)
  escrever( res)
FIM
--------- Ou ----------------
INICIO
  ler(nota1, nota2, nota3)
  media<-(nota1+nota2+nota3)/3
  Escrever( FResultado(nota))
FIM
```

Maria da Conceição Neves

Definir Procedimento e Função resultado

```
DEFINIRPResultado(n1, v1 ,v2)
    SE(n1<v1)
      ENTÃO
        escrever ("Reprovado")
      SENÃO
        SE(n1<v2)
            ENTÃO
               escrever (
               "Admitido a exame")
            SENÃO
               escrever (
               "Dispensado a exame")
        FIMSE
    FIMSE
FIMDEFINIR
```

```
DEFINIR FResultado (nota , val1 , val2)
    SE(nota<val1)
      ENTÃO
          result= "Reprovado"
      SENÃO
        SE(nota<val2)
          ENTÃO
             result= "Admitido a exame"

          SENÃO
            result= "Dispensado a exame"

        FIMSE
    FIMSE
    Retorna result
FIMDEFINIR
```

Maria da Conceição Neves

Algoritmos

```
INICIO
ler(nota)
PResultado(nota, 10,16)
FIM
------------ Ou --------
INICIO
ler(nota)
res= FResultado(nota, 8, 14)
escrever( res)
FIM
------------ Ou --------
INICIO
ler(nota)
Escrever(FResultado(nota,9,12)
   )
FIM
```

```
INICIO
ler(nota1, nota2, nota3)
media<-(nota1+nota2+nota3)/3
PResultado( media, 10,16)
FIM
--------- Ou ----------------
INICIO
ler(nota1, nota2, nota3)
media<-(nota1+nota2+nota3)/3
res= FResultado(media, 8, 12)
escrever( res)
FIM
--------- Ou ----------------
INICIO
ler(nota1, nota2, nota3)
media<-(nota1+nota2+nota3)/3
Escrever( FResultado(nota,9,12))
FIM
```

Maria da Conceição Neves

Métodos

Definição de método

public
private
protected
static

Assinatura do método

```
modificadores tipo_retorno nome (list_params)
{
    variaveis locais

      instruções
}
```

Os parâmetros são os meios de comunicação com o método

Os parâmetros funcionam como variáveis locais inicializadas com os valores dos argumentos de chamada

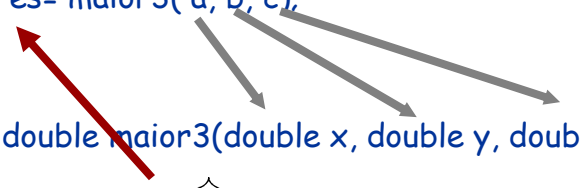
Maria da Conceição Neves

Correspondência Argumentos - Parâmetros

- Os métodos podem ser chamados permitindo que o programa execute as suas instruções sempre que necessário.

```
double a=12.8, b=7, c=3.5;
double res= maior3( a, b, c);

private double maior3(double x, double y, double  z)
{
return (Math.max(Math.max(x,y),z)
}
```



Maria da Conceição Neves

Passagem de Parâmetros para os métodos

Exemplos:
Para chamar métodos estáticos de uma outra classe

```
String res= JOptionPane.showInputDialog (null,"Qual o número");
int n = Integer.parseInt("124");
```

Nome Classe

Nome Método

Lista de argumentos

Na chamada de métodos estáticos da própria classe pode-se omitir o nome da classe

Cabeçalho do método public static **float area_r(float l1,float l2){ }**

Chamada do método float res= **area_r(lado1,lado2);**

Em Java os argumentos são passados por valor.

Maria da Conceição Neves

Definir função em Pseudo-Código

```
DEFINIR factorial (int num)
ED  int i, fact
   INÍCIO
      fact<- 1;
      PARA i=num ATÉ i=1 PASSO -1 FAZER
         fact=fact*i;
      FIMPARA
      RETORNA fact
FIMDEFINIR
```

Maria da Conceição Neves

Definir procedimento em Pseudo-Código

```
DEFINIR  mostrarDigitosInversamente (int numero)
ED
   ENQUANTO (numero >=10) FAZER
        escrever(numero MOD 10)
        numero <- numero  DIV  10
   FIMENQUANTO
   escrever(numero)
FIMDEFINIR
```

Maria da Conceição Neves

Documentar métodos

EXEMPLO

```
/**
 * Calcular de factorial
 *
 * @param  y   o número inteiro não negativo
 * @return     o valor do factorial do y
 */
public long factorial(long y)
{
    ...
}
```

Maria da Conceição Neves

Cálculo Combinatório

Definir a classe CalCombinatorio com os seguintes métodos da classe (static).

Definir uma classe Teste com método main que evoca os vários métodos da CalCombinatorio

- Método auxiliar factorial

  n!

- Método permutacoes

  $P_n = {}^nA_n = n!$

- Método arranjosSemRepeticao

  ${}^nA_p = n! / (n-p)!$ $n>=p$

- Método combinacoes

  ${}^nC_p = n! / (p!* (n-p)!$ $p<=n$

- ...

Maria da Conceição Neves

```
package  calculocombinatorio;
/**
*   Disponibiliza métodos associados ao Cálculo Combinatório
*/
public  class CalCombinatorio {
    /**
     * Calcular o factorial de um número
     *
     * @param  num    número inteiro não negativo
     * @return   o valor do factorial do num
     */
   private static long factorial(long num)
   {
      long fact=1;
      for (long i=num; i>1;i=i-1)
      {
         fact=fact*i;
      }
       return fact;
   }
```

Maria da Conceição Neves

```
    /**
     * Calcula o combinações de n p a p
     *
     * @param  n    número inteiro não negativo
     * @param  p    número inteiro não negativo
     * @return   o valor combinações  de n p a p ou -1 caso n<p
     */
public static long combinacoes(long n, long p )
   {
      if(n<p)
         return -1;
       return factorial(n)/(factorial(p)*factorial(n-p));
   }
}
```

Maria da Conceição Neves

```
import java.util.Formatter;
import java.util.Scanner;
/**
*  Cálculos vários utilizando  métodos da classe CalCombinatorio
*/
public class Teste {
  public static void main(String[ ] args)
     long n1,n2,res;
     Formatter obj1=new Formatter(System.out);
     Scanner obj2=new Scanner(System.in);

     obj1.format("%s%n","Escreva n");
     n1= obj2.nextLong();
     obj1.format("%s%n","Escreva p");
     n2= obj2.nextLong();

     res=CalCombinatorio.combinacoes(n1,n2);
     if(res !=-1)
          obj1.format("%s%d%n","Resultado=",res);
     else
        obj1.format("ERRO!")
  }
}
```

Maria da Conceição Neves