Algoritmo II
Procedimentos e Funções

Prof. Me. Rober Marcone Rosi
Unidade de Engenharia, Computação e Sistemas

Modularização

- É uma técnica utilizada para desenvolver algoritmos por meio de refinamentos sucessivos;
- O refinamento sucessivo nada mais é do que a redução de um problema a um conjunto de tarefas destinadas a solucioná-lo de maneira eficiente;
- Para cada tarefa, desenvolve-se um algoritmo/programa (módulo) que poderá ser utilizado na solução de outros problemas, pois cada módulo é independente;
- O gerenciamento das tarefas é feito pelo algoritmo principal. Esse módulo “chama” ou aciona os outros módulos, que deverão ser escritos por meio de funções ou procedimentos.

Vantagens da Utilização de Modularização/Sub-rotinas

- Partes comuns a vários programas ou que se repetem dentro de um mesmo programa quando modularizamos em uma sub-rotina, são programadas e testadas uma só vez, mesmo que tenham que ser executadas com variáveis diferentes;
- Podem-se constituir bibliotecas de programas, isto é, a coleção de módulos que podem ser usados em diferentes programas sem alteração e mesmo por outros programadores;
- Economia de memória do computador, uma vez que o módulo é armazenado uma única vez, mesmo que utilizado em diferentes partes do programa. Permite que em um determinado instante da execução do programa, estejam na memória principal apenas os módulos necessários à execução deste trecho do programa.

Desvantagem da Utilização de Modularização/Sub-rotinas

- Pode-se citar que existe um acréscimo de tempo de execução dos programas constituídos de módulos, devido ao tratamento adicional de ativação do módulo.

Tipos de Modularização

- São dois tipos de sub-rotinas: PROCEDIMENTOS e FUNÇÕES que são blocos de programa que executam determinada tarefa:
- **Procedimentos**: podem receber valores, mas não retornam outros valores como resultado;
- **Função**: retorna os valores resultantes das operações que realizou.

Também conhecidos como sub-rotina, módulo e em Java método.

Procedimento

- Algoritmos elaborados para resolução de uma tarefa específica;
- Utilizados para modularizar programas;
- Pode receber parâmetros por referência e devolver resultados à rotina de chamada, caso contrário não retorna valores como resultado;
- Um método é criado da mesma maneira que outro algoritmo qualquer, deve ser identificado, possui variáveis, operações e até funções.

Procedimento em Java

<modificador> **void** <nome> (<argumentos>){

  <tipo> <nome_da_variável>;

  <comandos>;

}

Onde:<**modificador**>: caracteriza o método quanto à visibilidade e qualidade. Os métodos, como as classes e as variáveis, podem possuir mais de um modificador, não importando sua ordem.

void: significa que como procedimento, o método não irá retornar nada.

<**nome**>: Deve obedecer às mesmas regras que os identificadores de variáveis. Observe que, em uma classe pode haver mais de um método com o mesmo nome, bastando que os tipos, a ordem ou o número de parâmetros sejam diferentes.

<**argumentos**>: indica a lista de argumentos que serão passados como parâmetros para o método. Eles devem obedecer a seguinte sintaxe: <tipo> par1, <tipo> par2, ..., <tipo> parN.

Modificadores

- public → Indica um método que é visível para qualquer um que enxergue a classe;
- protected → Indica um método que é visível apenas para classes do mesmo pacote ou subclasses;
- private → Indica que o método só pode ser invocado dentro da própria classe;
- final → Indica que o método não pode ser sobrescrito;
- static → Indica que o método pode ser invocado a partir do nome da classe, ou seja, não necessita de objeto.

Exemplo 7.1: Em Java

- *Procedimento para realizar a operação de adição entre dois valores, sem a passagem de parâmetros.*

```
static void calcularAdicao( )
{
        double v1;
        double v2;
        double res;
        Scanner ler = new Scanner(System.in);
        System.out.println(“Digite o  primeiro valor: “);
        v1 = ler.nextDouble();
        System.out.println("Digite o segundo valor“);
        v2 = ler.nextDouble();
       res = v1 + v2;
        System.out.print(“”, "Soma = " + res);
 }
```

Parâmetros

- Parâmetros são variáveis ou valores que podem ser transferidos do algoritmo principal para um módulo que está sendo chamado;
- Funcionam como comunicadores entre os módulos;
- Existem dois tipos de parâmetros:
  - **Formais**: São os parâmetros que são declarados como variáveis no cabeçalho do método. Esses valores serão inicializados com os valores que serão passados pelos métodos chamadores.
  - **Reais**: São variáveis ou valores que são passados pelo programa chamador no instante em que a chamada for feita.

Parâmetros Formais

- São declarados nos módulos e tratados como as variáveis. O algoritmo que chama a função ou o procedimento informa os valores que substituirão esses parâmetros. No exemplo a seguir, as variáveis A e B são parâmetros formais.

Java:

```
1.      static void multiplicarNumeros (int A, int B) {
2.              int res;
3.              res = A * B;
4.              System.out.print("Multiplicação = " + res);
5.      }
```

Parâmetros Reais

- São valores que substituem os parâmetros formais. O algoritmo que chama a função ou o procedimento informa esses valores ou variáveis. No exemplo abaixo, os parâmetros formais A e B do procedimento multiplicar serão substituídos pelos valores fornecidos para as variáveis num1 e num2 do algoritmo principal.

```
1. class MultiplicarNumeros {
2.                 public static void main  (String [ ]  args) {
3.                     Scanner ler = new Scanner(System.in);
4.                     double num1, num2;
5.                     System.out.println(“Digite o  primeiro valor: “);
6.                     num1 = ler.nextDouble();
7.                     System.out.println("Digite o segundo valor“);
8.                     num2 = ler.nextDouble();
9.                     multiplicarNumeros (num1, num2);
10.                    System.exit(0);
11.                }
12.                static void multiplicarNumeros (double A, double B) {
13.                                int res;
14.                                res = A  *  B;
15.                                InOut.MsgDeInformação(“”,“ ”  +  res);
16.                }
17.        }
```

Exemplo 7.2: Java

- *Procedimento para realizar a operação de adição entre dois valores, com a passagem de parâmetros.*

```
static void calcularAdicao (double v1, double v2){
   double res;
    res = v1 + v2;
    System.out.println( "Soma = " + res);
 }
```

Passagem de Parâmetros

- A passagem de parâmetros ocorre por meio da correspondência argumento/parâmetro, em que os argumentos são valores constantes ou variáveis informados no módulo chamador.
- Os argumentos devem ser fornecidos na mesma ordem dos parâmetros. Os parâmetros podem ser passados por valor ou por referência.

Passagem de Parâmetros

- **Passagem de parâmetros por valor**

  Na passagem de parâmetros por valor, o valor do parâmetro real é copiado para o parâmetro formal do módulo, preservando, assim, o valor original do parâmetro. Ex. Tipos primitivos.

- **Passagem de parâmetros por referência**

  Na passagem de parâmetros por referência, toda alteração feita nos parâmetros formais reflete-se nos parâmetros reais; assim, o parâmetro é de entrada e saída. Ex. Tipos construídos.

Exemplo de passagem por valor

```
1.  public class Exercicio {
2.      public static void main(String[] args) {
3.          int num1 = 10, num2 = 30;
4.          troca (num1, num2);
5.          System.out.println("PROGRAMA PRINCIPAL -
    PASSAGEM POR VALOR\nnum1 = "+num1+ " num2 =
    "+num2);
6.          System.exit(0);
7.      }
```

Exemplo de passagem por valor

```
8.  static void troca (int n1, int n2){
9.          int temp;
10.         System.out.println("DENTRO DO MÉTODO ANTES
    DA TROCA\nn1 = "+n1+" n2 = "+n2);
11.         temp = n1;
12.         n1 = n2;
13.         n2 = temp;
14.         System.out.println("DENTRO DO MÉTODO
    DEPOIS DA TROCA\nn1 = "+n1+ " n2 = "+n2);
15.     }
```

Exemplo de passagem por valor

Linha	num1	num2	n1	n2
3	10	30	Não existe	Não existe
4	10	30	Não existe	Não existe
8	10	30	10	30
14	10	30	30	10
5	10	30	Não existe	Não existe

Exemplo de passagem por referência

```
1.  public class PassagemDeObjetos {
2.      public static void main(String[] args) {
3.          int[] vetor = {10,30};
4.          troca (vetor);
5.          System.out.println("PASSAGEM POR VALOR DE UM
    VETOR\nnum1 = "+
6.          vetor[0]+" num2 = "+vetor[1]);
7.          System.exit(0);
8.  }
```

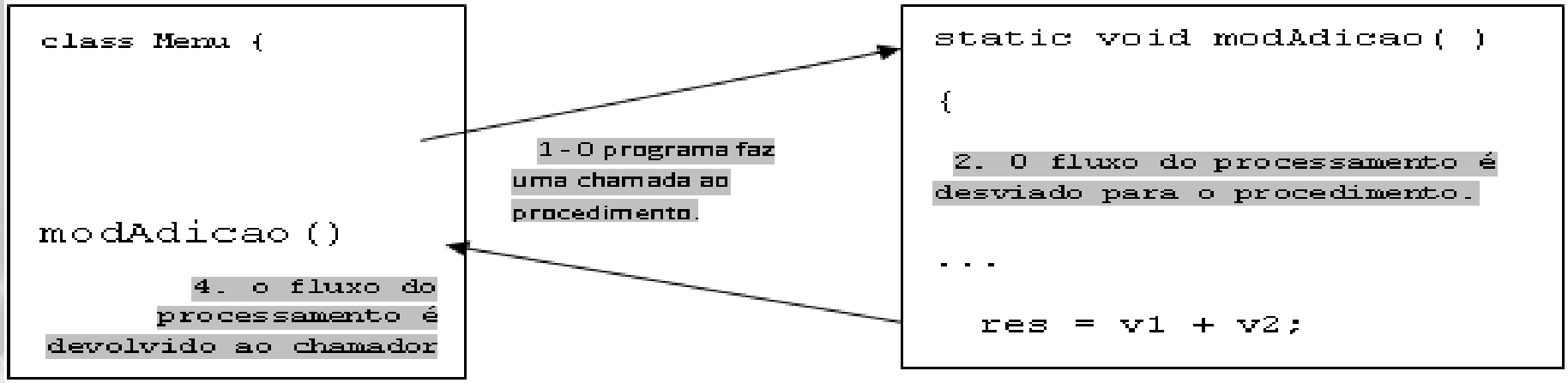
Exemplo de passagem por referência

```
1.  static void troca (int[] vet){
2.      int temp;
3.      System.out.println("DENTRO DA MÉTODO ANTES DA TROCA\nnum1 = "+
4.      vet[0]+" num2 = "+vet[1]);
5.      temp = vet[0];
6.      vet[0] = vet[1];
7.      vet[1] = temp;
8.      System.out.println("DENTRO DA MÉTODO DEPOIS DA TROCA\nnum1 = "+
9.      vet[0]+" num2 = "+vet[1]);
10. }
```

Chamada de um procedimento

- É o momento em que o procedimento é acionado e seu código executado, podendo ocorrer a passagem ou não de parâmetros.
- Quando ocorre a chamada de um procedimento a execução do algoritmo chamador é “interrompida”, o controle é passado para o procedimento até que seu conjunto de instruções seja finalizado, momento que o controle de execução volta para o chamador.
- O algoritmo chamador é o algoritmo que utiliza o procedimento.

Chama de um procedimento

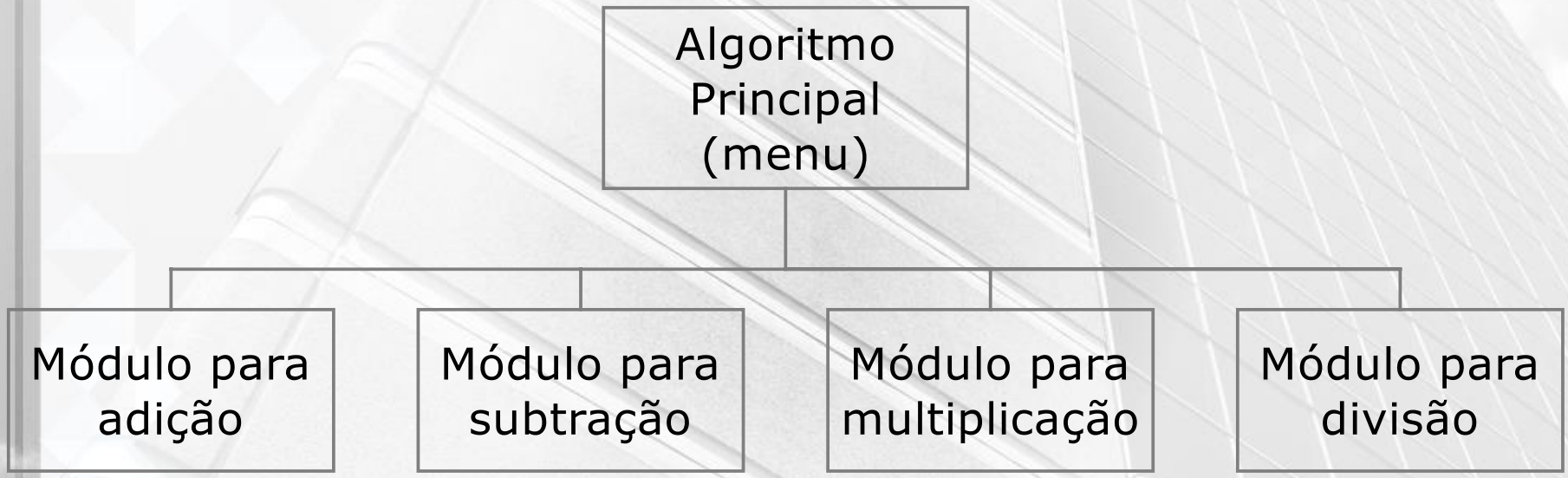


Exemplo 7.8

Elaborar um algoritmo que realize a operação aritmética escolhida pelo usuário, a saber: adição, subtração, multiplicação ou divisão, entre dois valores fornecidos por ele e apresente uma mensagem com o resultado obtido.

Deverá ser criado um *menu* de opções para o usuário no algoritmo principal e módulos com procedimentos para a realização das operações, conforme diagrama a seguir.

Módulo do exemplo



Exemplo 7.8: Java – Principal (main)

```
1. public class Menu {
2.   public static void main (String [] args){
3.     int opcao;
4.     opcao = InOut.leInt(Escolha a sua opção:\n" +
5.         "1 - Adição\n" +
6.         "2 - Subtração\n" +
7.         "3 - Multiplicação\n" +
8.         "4 - Divisão"));
9.     switch (opcao){
10.                 case 1 : calcularAdicao(); break;
11.                 case 2 : calcularSubtr(); break;
12.                 case 3 : calcularMultipl(); break;
13.                 case 4 : calcularDiv(); break;
14.                 default : InOut.MsgDeInformação(“”, "Fim do Programa");
15.               }
16.             }
```

Exemplo 7.8: Java – *calcular*Adicao

```
static void calcularAdicao( ){
        double v1;
        double v2;
        double res;
        Scanner ler = new Scanner(System.in);
        System.out.println(“Digite o  primeiro valor: “);
        v1 = ler.nextDouble();
        System.out.println("Digite o segundo valor“);
        v2 = ler.nextDouble();
        res = v1 + v2;
        System.out.println(“ Soma = " + res);
  }
```

Exemplo 7.8: Java – *calcular*Subtr

```
static void calcularSubtr( ){
        double v1;
        double v2;
        double res;
        Scanner ler = new Scanner(System.in);
        System.out.println(“Digite o  primeiro valor: “);
        v1 = ler.nextDouble();
        System.out.println("Digite o segundo valor“);
        v2 = ler.nextDouble();
        res = v1 - v2;
        System.out.println(" Subtração = " + res);
 }
```

Exemplo 7.8: Java – *calcular*Multipl

```
static void calcularMultipl( ){
   double v1;
   double v2;
   double res;
   Scanner ler = new Scanner(System.in);
   System.out.println(“Digite o  primeiro valor: “);
   v1 = ler.nextDouble();
   System.out.println("Digite o segundo valor“);
   v2 = ler.nextDouble();
   res = v1 * v2;
   System.out.println("Multiplicação = " + res);
}
```

Exemplo 7.8: Java – *calcular*Div

```
static void calcularDiv( ){
    double v1;
    double v2;
    double res;
    Scanner ler = new Scanner(System.in);
    System.out.println(“Digite o  primeiro valor: “);
    v1 = ler.nextDouble();
    System.out.println("Digite o segundo valor“);
    v2 = ler.nextDouble();
    res = v1 / v2;
    System.out.println(“”, "Divisão = " + res);
 }
```

Funções

- São criadas da mesma maneira que os procedimentos;
- A diferença entre eles é que as funções podem ser utilizadas em expressões, como se fossem variáveis, pois as funções retornam valores que são associados ao seu nome e para esses valores se faz necessária a declaração do tipo de dado a ser retornado.

Funções - Java - Sintâxe

```
<modificador> <tipo> <nome> (<argumentos>) {
  <tipo> <nome_da_variável>;
  <comandos>;
  return <variável>;
}
```

Onde:

<**modificador**>, <**nome**>, <**argumentos**>: são idênticos aos de procedimento;

<**tipo**> : é o tipo do valor de retorno da função.

Comparação entre função e procedimento

Procedimento

```
Procedimento
  ModAdicao(v1, v2: real)
Var
  res: real
Início
  res ← v1 + v2
  Mostrar (res)
Fim.
```

Função

```
Função
  ModAdicao(v1, v2: real): real
Var
  res: real
Início
  res ← v1 + v2
  retornar (res)
Fim.
```

Exemplo 7.10 - *Chamada a uma função com parâmetros - Java*

```
public class Exemplo710{
  public static void main (String args []){
    int num1, num2;
    Scanner ler = new Scanner(System.in);
    System.out.println(“Digite o  primeiro valor: “);
    num1 = ler.nextInt();
    System.out.println("Digite o segundo valor“);
    num2 = ler.nextInt();
    System.out.println("Resultado: " + multiplicarNum(num1, num2));
  }
  static int multiplicarNum(int a, int b){
     int res;
     res = a * b;
     return res;
  }
}
```

Exemplo 7.12 - Ler um número fornecido pelo usuário e calcular o fatorial.

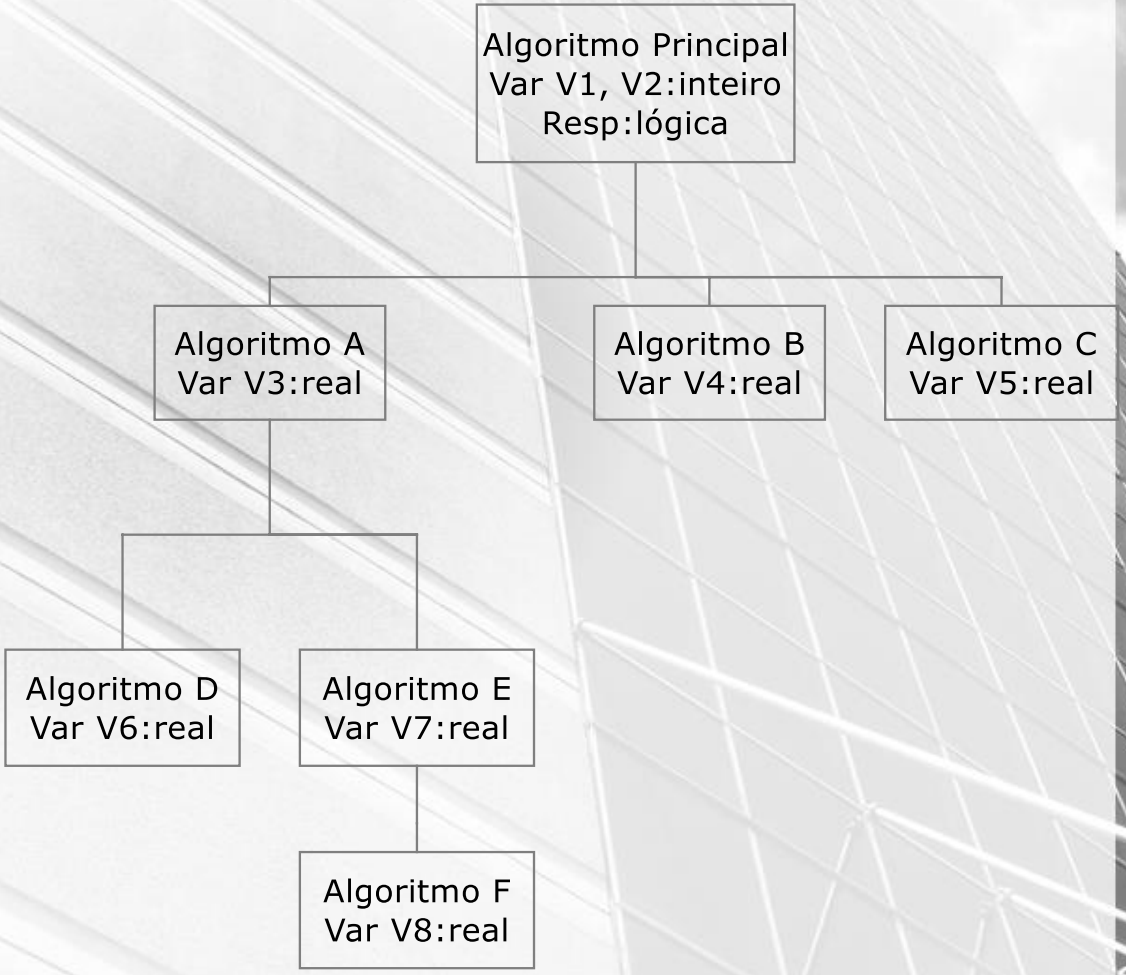
```
public class Exemplo712 {
  public static void main(String args []){
     int numero;
     int fat;
    Scanner ler = new Scanner(System.in);
    System.out.println("Digite o  primeiro valor: ");
    numero = ler.nextInt();
     fat = calcularFatorial(numero);
     System.out.println("O fatorial de " + numero + " é " + fat);
  }
  static int calcularFatorial (int numero){
     int f = 1;
     for (int i = 1; i <= numero; i++){
         f = f * i;
      }
      return f;
  }
}
```

Escopo de variáveis

- Especifica a “visibilidade” da variável
- Uma variável pode ser global ou local
  - As variáveis globais são declaradas no algoritmo principal e podem ser utilizadas por todos os algoritmos hierarquicamente inferiores.
  - As variáveis locais podem ser utilizadas pelo algoritmo em que foram declaradas e nos algoritmos hierarquicamente inferiores
- *A definição adequada das variáveis pode economizar memória e tornar os programas mais eficientes.*

Escopo de variáveis

- as variáveis V1 e V2 foram declaradas no módulo principal e podem ser utilizadas por todos os módulos dos algoritmos;
- a variável V3 foi declarada no algoritmo A e pode ser utilizada pelos algoritmos D, E e F, que são hierarquicamente inferiores a ele;
- as variáveis V4, V5, V6 e V8 podem ser utilizadas somente pelos algoritmos B, C, D e F, respectivamente, pois não possuem algoritmos hierarquicamente inferiores;
- a variável V7 pode ser utilizada pelos algoritmos E e F



Escopo de Variáveis

Algoritmo Principal;
Var V1,V2: real;
Resp: lógica;

Algoritmo A;
Var V3: real;

Algoritmo D;
Var V6: real;

Algoritmo E;
Var V7: real;

Algoritmo F;
Var V8: real;

Algoritmo B;
Var V4: real;

Algoritmo C;
Var V5: real;

Escopo de Variáveis

Exemplo:

```
class NomeDaClasse
{
  /* a variável abaixo é global */
  TipoDaVariahvel variahvel1;
  ...
  TipoDeRetorno nomeDoMehtodo()
  {
       /* a variável abaixo é local; está definida somente dentro deste método */
         TipoDaVariahvel variahvel2;
         for( int i = 0; i < 10; i++ )
         {
       /* a variável i é local, definida só dentro deste bloco */
          ...
         }
    }
}
```