

Programação 1

Aula 4

Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

<http://elearning.ua.pt/>

Revisão da aula anterior

Operadores aritméticos unários

```
int A = 1, B= 2, C = 3, Y; Y = A++ + B++ + C--; // Y = 6; A = 2; B = 3; C = 2
int A = 1, B= 2, C = 3 , Y; Y = ++A + B++ + C--; // Y = 7; A = 2; B = 3; C = 2
int A = 1, B= 2, C = 3 , Y; Y = ++A + ++B + --C; // Y = 7; A = 2; B = 3; C = 2
int A = 1, B= 2, C = 3 , Y; Y = ++A + ++B + ++C; // Y = 9; A = 2; B = 3; C = 4
int A = 1, B= 2, C = 3 , Y; Y = A++ + B++ + C++; // Y = 6; A = 2; B = 3; C = 4
int A = 1, B= 2, C = 3 , Y; Y = --A + --B + --C; // Y = 3; A = 0; B = 1; C = 2
```

Instrução de atribuição com operação

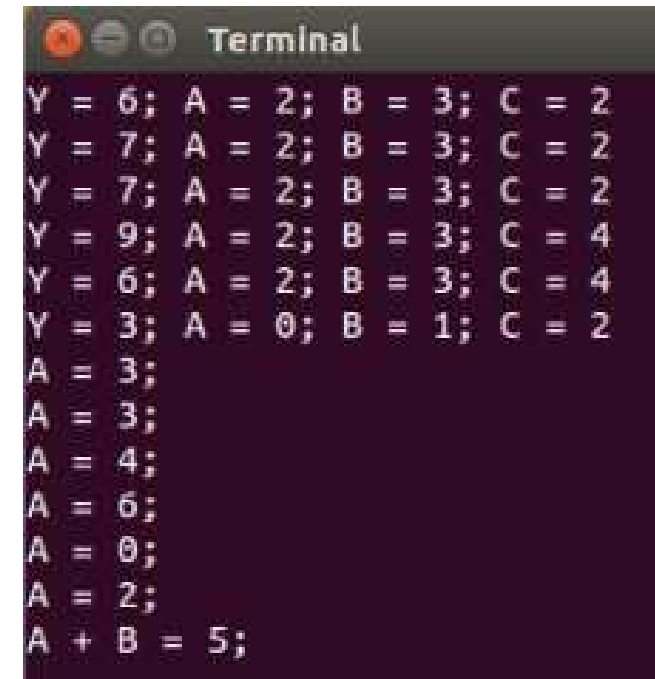
```
int A = 1, B= 2; A += B; // A = A + B = 3
int A = 1, B= 2; A += B++; // A = A + B++ = 3
int A = 1, B= 2; A += ++B; // A = A + ++B = 4
int A = 2, B= 3; A *= B; // A = A * B = 6
int A = 2, B= 3; A /= B; // A = A / B = 0
int A = 2, B= 3; A %= B; // A = A % B = 2
```

```

public class Operations
{
    public static void main (String args[])
    {
        int A = 1, B= 2, C = 3, Y; Y = A++ + B++ + C--;      // Y = 6; A = 2; B = 3; C = 2
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);
        A = 1; B= 2; C = 3; Y = ++A + B++ + C--;           // Y = 7; A = 2; B = 3; C = 2
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);
        A = 1; B= 2; C = 3; Y = ++A + ++B + --C;           // Y = 7; A = 2; B = 3; C = 2
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);
        A = 1; B= 2; C = 3; Y = ++A + ++B + ++C;           // Y = 9; A = 2; B = 3; C = 4
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);
        A = 1; B= 2; C = 3; Y = A++ + B++ + C++;           // Y = 6; A = 2; B = 3; C = 4
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);
        A = 1; B= 2; C = 3; Y = --A + --B + --C;           // Y = 3; A = 0; B = 1; C = 2
        System.out.printf("Y = %d; A = %d; B = %d; C = %d\n",Y,A,B,C);

        A = 1; B= 2; A += B;                                // A = A + B = 3
        System.out.printf("A = %d;\n",A);
        A = 1; B= 2; A += B++;                                // A = A + B++ = 3
        System.out.printf("A = %d;\n",A);
        A = 1; B= 2; A += ++B;                                // A = A + ++B = 4
        System.out.printf("A = %d;\n",A);
        A = 2; B= 3; A *= B;                                  // A = A * B = 6
        System.out.printf("A = %d;\n",A);
        A = 2; B= 3; A /= B;                                  // A = A / B = 0
        System.out.printf("A = %d;\n",A);
        A = 2; B= 3; A %= B;                                  // A = A % B = 2
        System.out.printf("A = %d;\n",A);
    }
}

```



```

Terminal
Y = 6; A = 2; B = 3; C = 2
Y = 7; A = 2; B = 3; C = 2
Y = 7; A = 2; B = 3; C = 2
Y = 9; A = 2; B = 3; C = 4
Y = 6; A = 2; B = 3; C = 4
Y = 3; A = 0; B = 1; C = 2
A = 3;
A = 3;
A = 4;
A = 6;
A = 0;
A = 2;
A + B = 5;

```

Aula 4

- Introdução à programação modular
- Funções
- Tipos primitivos como argumentos
- Visibilidade das variáveis
- Exemplos

Introdução à programação modular

- Na especificação de um problema obtemos um conjunto de tarefas básicas (ex: `ler`, `calcular`, `imprimir`).
- Com o aumento da complexidade dos problemas que queremos resolver, torna-se vantajoso a implementação e teste de cada uma dessas tarefas em separado.
- A linguagem JAVA permite-nos criar **funções** para implementar as várias tarefas básicas de um programa.
- Uma **função** permite realizar um determinado conjunto de operações e, se necessário, devolver um valor.
- As funções desenvolvidas pelo programador são chamadas no programa da mesma forma que as funções criadas por terceiros (por exemplo as funções de leitura ou escrita de dados ou as funções da classe `Math`).

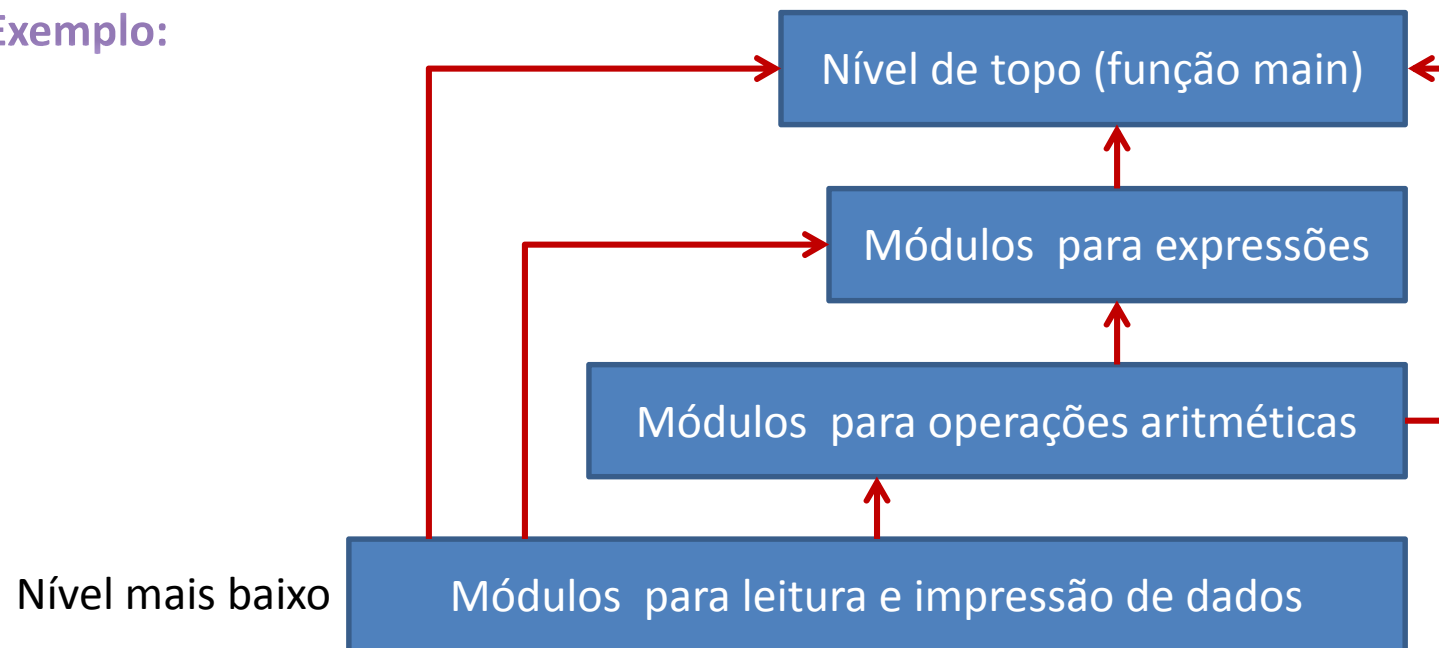
Programação modular permite descrever o código de programa hierarquicamente

1. Descrever módulos do nível mais baixo (ponto 1);
2. Descrever módulos do nível 2 (utilizando módulos do ponto 1);
3. Descrever módulos do nível 3 (utilizando módulos dos pontos 1 e 2);

.....

Descrever módulos do nível de topo (utilizando módulos dos pontos anteriores);

Exemplo:



- Estrutura de um programa (relembrar):

```
//inclusão de classes externas
```

```
public class Programa
```

```
{// declaração de constantes e variáveis visíveis em  
  // classe Programa
```

```
  public static void main (String[] args)
```

```
  {
```

```
    // declaração de constantes e variáveis locais
```

```
    // sequências de instruções
```

```
  }
```

```
    funções desenvolvidas pelo programador
```

```
  }
```

```
// definição de tipos de dados (registos)
```

- As funções são criadas depois **ou antes** da definição da função main.

Definição de uma função

```
// cabeçalho da função  
{  
    // corpo da função  
}
```

No cabeçalho da função indicam-se os qualificadores de acesso (neste momento sempre **public static**), o tipo de resultado, o nome da função e dentro de parênteses curvos a lista de argumentos.

```
public static tipo nome (argumentos)  
{  
    // declaração de variáveis  
    // sequências de instruções  
}
```

Definição de uma função

- Uma função é uma unidade auto-contida que recebe dados do exterior através dos argumentos, se necessário, realiza tarefas e devolve um resultado, se necessário.
- O resultado de saída de uma função pode ser de qualquer tipo primitivo (`int`, `double`, `char`, ...), de qualquer tipo referência (iremos ver mais à frente) ou `void` (no caso de uma função não devolver um valor).
- A lista de argumentos (ou parâmetros) é uma lista de pares de indentificadores separados por vírgula, onde para cada argumento se indica o seu tipo de dados e o seu nome.
- O corpo da função assemelha-se à estrutura de um **módulo**.
- Se a função devolver um valor utiliza-se a palavra reservada **return** para o devolver.
- O valor devolvido na instrução de **return** deve ser compatível com o tipo de saída da função.

Exemplo 1:

```
import java.util.*;
public class Ler
{
    static Scanner kb = new Scanner(System.in);
    public static int lerPositivo()
    {
        int x;
        do {
            System.out.print("Valor inteiro: ");
            x = kb.nextInt();
        } while(x < 0);
        return x;
    }

    public static void main(String[] args)
    {
        System.out.printf("soma = %d\n", soma(lerPositivo(),lerPositivo()));
    }

    public static int soma (int x, int y)
    {
        int soma;
        soma = x + y;
        return soma;
    }
}
```

Visibilidade

The diagram illustrates visibility annotations for the provided Java code. Green arrows point to the following elements: the opening curly brace of the class, the Scanner declaration, the lerPositivo() method signature, the main() method signature, the soma() method signature, and the closing curly brace of the class. A green bracket on the right side groups the entire class structure, with a label 'Visibilidade' (Visibility) pointing to it.

Exemplo 1:

Função soma

Tipo do valor de retorno

Nome da função

Tipo (int) e
nome (x) do
primeiro
argumento

Tipo (int) e
nome (y) do
segundo
argumento

```
public static int soma (int x, int y) {  
    int soma;  
    soma = x + y;  
    return soma;  
}
```

Valor de retorno

Exemplo 1:

Função lerPositivo

Tipo do valor de retorno

Nome da função

Lista de argumentos está vazia

```
public static int lerPositivo() {  
    int x;  
    do {  
        System.out.print("Valor inteiro: ");  
        x = kb.nextInt();  
    } while(x < 0);  
    return x;  
}
```

Valor de retorno

Código alternativo:

```
public static int soma (int x, int y)  {  
    int soma;  
    soma = x + y;  
    return soma;  
}
```

=

```
public static int soma (int x, int y)  {  
    return x + y;  
}
```

```

... main (...){ // Soma de dois números positivos
    int a, b, r;
    a = lerPositivo(); // utilização das funções definidas pelo programador
    b = lerPositivo(); // da mesma forma que utilizamos todas as outras...
    r = soma(a, b); // o valor de a e b são passados á função soma
    printf("%d + %d = %d\n", a, b, r);
}

public static int lerPositivo(){
    ...
    do{
        ...
    }while(x < 0);
    return x; // devolução do valor lido através do teclado, após validação
}

public static int soma (int x, int y){ // neste exemplo x = a e y = b
    int soma;
    soma = x + y;
    return soma;
}

```

```

public static int soma (int x, int y) {
    return x + y;
}

```

Exemplo 2: Escreva um programa que permite calcular o fatorial de N ($1 \leq N \leq 10$) utilizando uma função

Função fact



```
int N, fatorial = 1;
do {
    System.out.print("Introduza um numero: ");
    N = sc.nextInt();
    if (N > 10 || N < 1)
        System.out.println("o número errado");
    } while(N > 10 || N < 1);
for (int i = 1; i <= N; i++)
    fatorial *= i;
System.out.println("fatorial = "+fatorial);
```

```
public static int fact(int N)    {
    int fatorial = 1;
    do {
        System.out.print("Introduza um numero: ");
        N = sc.nextInt();
        if (N > 10 || N < 1)
            System.out.println("o número errado");
        } while(N > 10 || N < 1);
    for (int i = 1; i <= N; i++)
        fatorial *= i;
    return fatorial;
}
```


Exemplo 2: Escreva um programa que permite calcular o fatorial de N ($1 \leq N \leq 10$) utilizando uma função

Função fact

```
public static int fact(int N)    {
    int fatorial = 1;
    do {
        System.out.print("Introduza um numero: ");
        N = sc.nextInt();
        if (N > 10 || N < 1)
            System.out.println("o número errado");
    } while(N > 10 || N < 1);
    for (int i = 1; i <= N; i++)
        fatorial *= i;
    return fatorial;
}
```

O código pode ser melhorado:

```
public static int lerPositivo()  {
    int x;
    do {
        System.out.print("Valor positivo: ");
        x = sc.nextInt();
    } while(x < 0);    return x;
}
```

Ler dados

```
public static int fact(int N)  {
    int fatorial = 1;
    for (int i = 1; i <= N; i++)
        fatorial *= i;
    return fatorial;
}
```

Encontrar fatorial

Agora podemos reutilizar o código

Exemplo 2: Escreva um programa que permite calcular o fatorial de N ($1 \leq N \leq 10$) utilizando uma função

```
import java.util.*;
public class FuncFact
{
    static Scanner sc = new Scanner(System.in);

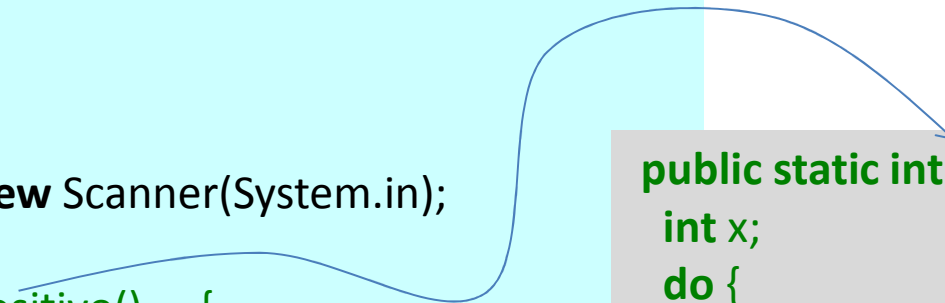
    public static int lerPositivo() {
        // .....

    }

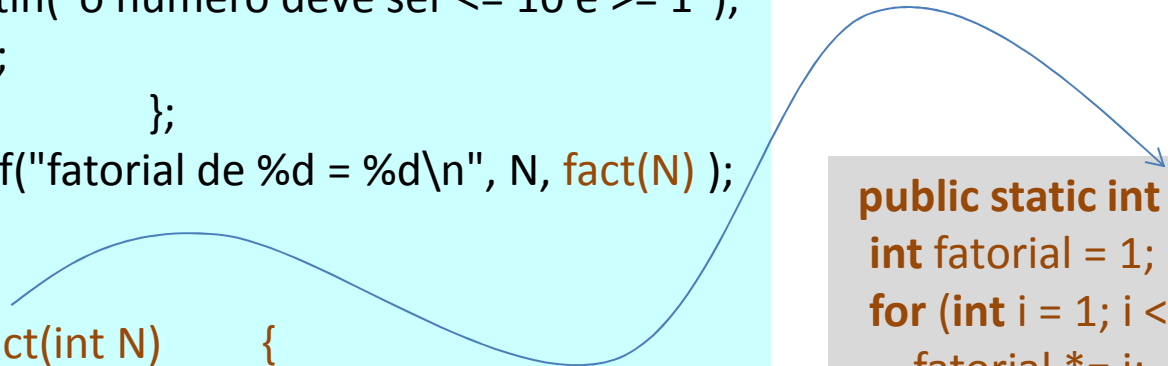
    public static void main(String[] args)
    {
        int N = lerPositivo();
        while(N > 10 || N < 1) {
            System.out.println("o número deve ser <= 10 e >= 1");
            N = lerPositivo();
        };
        System.out.printf("fatorial de %d = %d\n", N, fact(N) );
    }

    public static int fact(int N) {
        // .....

    }
}
```



```
public static int lerPositivo() {
    int x;
    do {
        System.out.print("Valor positivo: ");
        x = sc.nextInt();
    } while(x < 0);    return x;
}
```



```
public static int fact(int N) {
    int fatorial = 1;
    for (int i = 1; i <= N; i++)
        fatorial *= i;
    return fatorial;
}
```

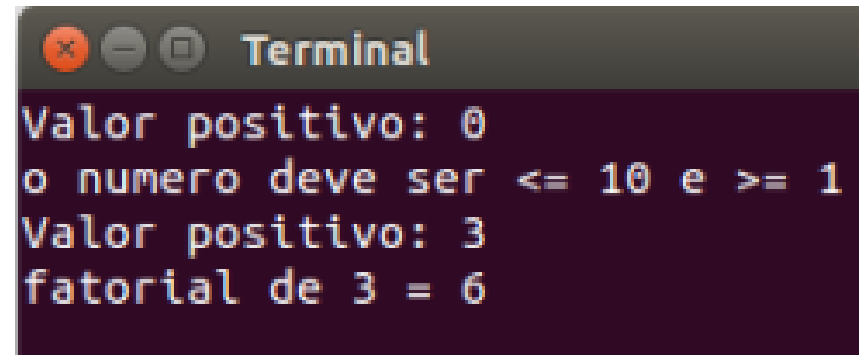
```

import java.util.*;
public class FuncFact
{
    static Scanner sc = new Scanner(System.in);
    public static int lerPositivo() {
        int x;
        do {
            System.out.print("Valor positivo: ");
            x = sc.nextInt();
        } while(x < 0);
        return x;
    }

    public static void main(String[] args)
    {
        int N = lerPositivo();
        while(N > 10 || N < 1) {
            System.out.println("o número deve ser <= 10 e >= 1");
            N = lerPositivo();
        };
        System.out.printf("fatorial de %d = %d\n", N, fact(N) );
    }

    public static int fact(int N) {
        int fatorial = 1;
        for (int i = 1; i <= N; i++)
            fatorial *= i;
        return fatorial;
    }
}

```



```

Terminal
Valor positivo: 0
o numero deve ser <= 10 e >= 1
Valor positivo: 3
fatorial de 3 = 6

```

Uma vantagem de modularidade é a reutilização do código

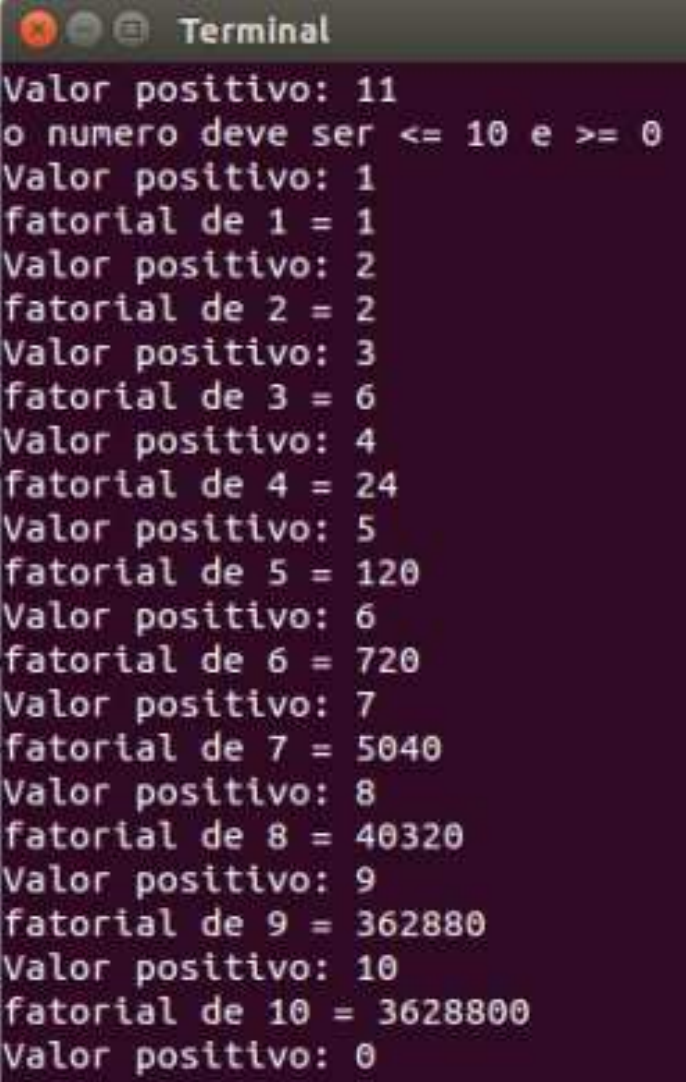
```
import java.util.*;
public class FuncFact
{
    static Scanner sc = new Scanner(System.in);
    public static int lerPositivo() {
        // .....
    }
    public static void main(String[] args)
    {
        int N = lerPositivo();
        while(N != 0) {
            while(N > 10 || N < 0) {
                System.out.println("o número deve ser <= 10 e >= 0");
                N = lerPositivo();
            }
            System.out.printf("fatorial de %d = %d\n", N, fact(N));
            N = lerPositivo();
        }
    }
    public static int fact(int N) {
        // .....
    }
}
```

As funções podem ser chamadas várias vezes com argumentos diferentes

```

import java.util.*;
public class FuncFactReutil
{
    static Scanner sc = new Scanner(System.in);
    public static int lerPositivo() {
        int x;
        do {
            System.out.print("Valor positivo: ");
            x = sc.nextInt();
        } while(x < 0);
        return x;
    }
    public static void main(String[] args)
    {
        int N = lerPositivo();
        while(N != 0) {
            while(N > 10 || N < 0) {
                System.out.println("o número deve ser <= 10 e >= 0");
                N = lerPositivo();
            }
            System.out.printf("fatorial de %d = %d\n", N, fact(N));
            N = lerPositivo();
        }
    };
    public static int fact(int N) {
        int fatorial = 1;
        for (int i = 1; i <= N; i++)
            fatorial *= i;
        return fatorial;
    }
}

```



```

Terminal
Valor positivo: 11
o numero deve ser <= 10 e >= 0
Valor positivo: 1
fatorial de 1 = 1
Valor positivo: 2
fatorial de 2 = 2
Valor positivo: 3
fatorial de 3 = 6
Valor positivo: 4
fatorial de 4 = 24
Valor positivo: 5
fatorial de 5 = 120
Valor positivo: 6
fatorial de 6 = 720
Valor positivo: 7
fatorial de 7 = 5040
Valor positivo: 8
fatorial de 8 = 40320
Valor positivo: 9
fatorial de 9 = 362880
Valor positivo: 10
fatorial de 10 = 3628800
Valor positivo: 0

```

Tipos primitivos como argumentos

- Tomando como exemplo o programa (soma de dois positivos), podemos ver que a função `lerPositivo` não recebe argumentos e devolve um valor inteiro.
- Quando uma função é chamada várias vezes, o seu valor de retorno pode ser armazenado nas variáveis.
- A função `soma` recebe dois argumentos do tipo inteiro e devolve também um valor inteiro.
- Quanto executada, o conteúdo das variáveis `a` e `b` são passados para "dentro" da função `soma` através dos argumentos.
- Sempre que uma função é usada, o programa "salta" para dentro da função, executa o seu código e quando termina continua na instrução que usa a chamada da função.

Visibilidade das variáveis

- Vimos que um programa pode conter várias funções, sendo obrigatoriamente uma delas a função `main`.
- As **variáveis locais** apenas são visíveis no corpo da função onde são declaradas.
- As variáveis declaradas dentro de um bloco (ou seja dentro do conjunto de instruções delimitado por `{ ... }` têm visibilidade limitada ao bloco (por exemplo ciclo **for**).
- Podemos também ter **variáveis globais**, sendo estas declaradas fora da função `main`, dentro da classe que implementa o programa (têm de ser precedidas da palavra reservada **static**).

Exemplo:

A função main pode ser representada como:

```
import java.util.*;
public class FuncFact
{
    static Scanner sc = new Scanner(System.in);
    public static int lerPositivo() {
        // .....
    }
    public static void main(String[] args)
    {
        int N = lerPositivo();
        while(N != 0) {
            while(N > 10 || N < 0) {
                System.out.println("o número deve ser <= 10 e >= 0");
                N = lerPositivo();
            }
            System.out.printf("fatorial de %d = %d\n", N, fact(N) );
            N = lerPositivo();
        }
    }
    // .....
}
```

```
public static void main(String[] args)
{
    for(int N = lerPositivo(); N != 0;) {
        while(N > 10 || N < 0) {
            System.out.println("o número deve ser <= 10 e >= 0");
            N = lerPositivo();
        }
        System.out.printf("fatorial de %d = %d\n", N, fact(N) );
        N = lerPositivo();
    }
}
```


Exemplo:

```
public static void main(String[] args)
{
    for(int N = lerPositivo(); N != 0;) {
        while(N > 10 || N < 0) {
            System.out.println("o número deve ser <= 10 e >= 0");
            N = lerPositivo();
        }
        System.out.printf("fatorial de %d = %d\n", N, fact(N) );
        N = lerPositivo();
    }
    System.out.printf("N = %d\n", N);
}
```

DrJava

Error: cannot find symbol
symbol: variable N
location: class ForGlobal

Geany

ForGlobal.java:26: error: cannot find symbol
System.out.printf("N = %d\n", N);

symbol: variable N
location: class ForGlobal

1 error

Compilation failed.

```
public static void main(String[] args)
{
    int N;
    for(N = lerPositivo(); N != 0;) {
        while(N > 10 || N < 0) {
            System.out.println("o número deve ser <= 10 e >= 0");
            N = lerPositivo();
        }
        System.out.printf("fatorial de %d = %d\n", N, fact(N) );
        N = lerPositivo();
    }
    System.out.printf("N = %d\n", N);
}
```

Ok

```
Terminal
Valor positivo: 4
fatorial de 4 = 24
Valor positivo: 0
N = 0
```

Exemplo:

Mal

```
import java.util.*;
public class Global
{
    static Scanner sc = new Scanner(System.in);
    static int x, N;
    public static int lerPositivo() {
        do {
            System.out.print("Valor positivo: ");
            x = sc.nextInt();
        } while(x < 0);
        return x;
    }
    public static void main(String[] args)
    {
        for(N = lerPositivo(); N != 0;) {
            while(N > 10 || N < 0) {
                System.out.println("o número deve ser <= 10 e >= 0");
                N = lerPositivo();
            }
            System.out.printf("fatorial de %d = %d\n", N, fact());
            N = lerPositivo();
        }
        System.out.printf("N = %d\n", N);
    }

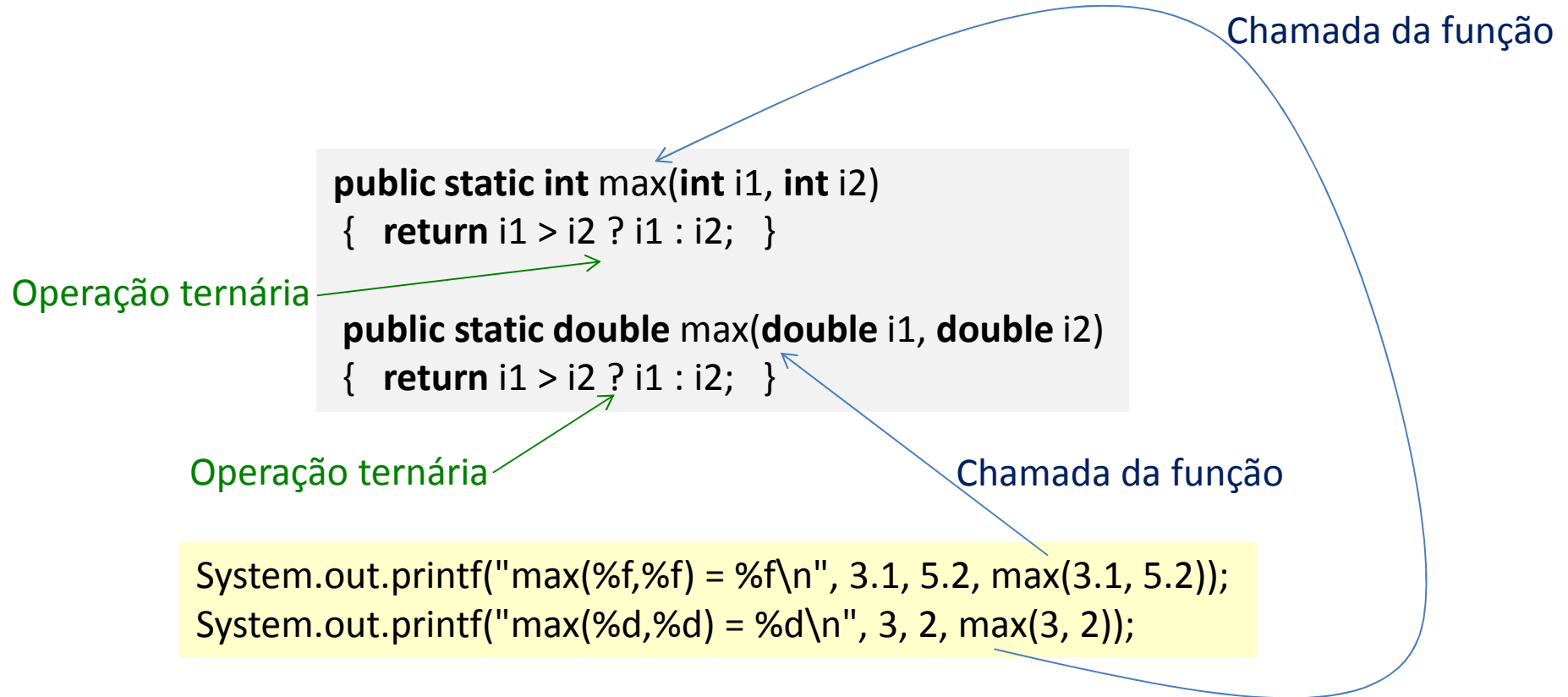
    public static int fact() {
        int fatorial = 1;
        for (int i = 1; i <= N; i++)
            fatorial *= i;
        return fatorial;
    }
}
```

Geralmente, Ok

Deve evitar
variáveis globais

Não use variáveis
(objetos) globais
sem necessidade

Sobrecarga de nomes



Sobrecarga de nomes

```
public static int max(int i1, int i2)  
{ return i1 > i2 ? i1 : i2; }
```

```
public static double max(double i1, double i2)  
{ return i1 > i2 ? i1 : i2; }
```

```
public static int max(int i1, int i2, int i3)  
{ if (i1 > i2 && i1 > i3) return i1;  
  else if (i2 > i1 && i2 > i3) return i2;  
  return i3;  
}
```

Chamada da função

```
System.out.printf("max(%f,%f) = %f\n", 3.1, 5.2, max(3.1, 5.2));  
System.out.printf("max(%d,%d) = %d\n", 3, 2, max(3, 2));  
System.out.printf("max(%d,%d,%d) = %d\n", 3, 2, 6, max(3, 2, 6));
```

Sobrecarga de nomes

```
public static int max(int i1, int i2)
{ return i1 > i2 ? i1 : i2; }
```

```
public static double max(double i1, double i2)
{ return i1 > i2 ? i1 : i2; }
```

```
public static int max(int i1, int i2, int i3)
{ if (i1 > i2 && i1 > i3) return i1;
  else if (i2 > i1 && i2 > i3) return i2;
  return i3;
}
```

```
public static char max(int i1, char c)
{ return i1 > (int)c ? (char)i1 : c; }
```

A própria função é selecionada de acordo com o número e tipo de argumentos

Chamada da função

```
System.out.printf("max(%f,%f) = %f\n", 3.1, 5.2, max(3.1, 5.2));
System.out.printf("max(%d,%d) = %d\n", 3, 2, max(3, 2));
System.out.printf("max(%d,%d,%d) = %d\n", 3, 2, 6, max(3, 2, 6));
System.out.printf("max(%d,%c) = %c\n", 48, 'A', max(48, 'A'));
```

Algumas regras úteis

1. Pode importar a biblioteca `import java.util.*;` em vez de `import java.util.Scanner;` É mais simples para lembrar.
2. Pode declarar um objeto do tipo Scanner `global` e usar este objeto em qualquer função, por exemplo:

```
import java.util.*;  
public class primo  
{  
    static Scanner sc = new Scanner(System.in);  
    public static void main(String[] args)  
    // .....
```

3. Escreva o código mais compacto e compreensível, por exemplo, as funções

```
public static double sqr(double x) {  
    double y;  
    y = x*x;  
    return y;  
}
```

e

```
public static double sqr(double x) { return x*x; }
```

executam a mesma operação mas o último código é melhor porque é mais compacto e não precisa de reservar e libertar memória para a variável y.

Conclusão

As funções permitem implementar hierarquia no código do programa e simplificam reutilização de código

Uma função pode chamar outras funções

As funções podem ser chamadas várias vezes com argumentes diferentes

Os objetos declarados numa função são visíveis só dentro desta função

Os objetos (variáveis) globais são visíveis dentro de qualquer função

Não use variáveis (objetos) globais sem necessidade

Exercício 5.9

Crie uma função booleana `isprime` que indique se um parâmetro inteiro `n` é um número primo. Escreva um programa para listar todos os primos entre 1 e `M`. O valor `M` (positivo) deve ser lido com validação.

```
import java.util.*;
public class ex5_9
{
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    {
        TestPrimo( lerPositivo() );
    }
    public static void TestPrimo(int M)
    {
        for (int x = 1; x <= M; x++) {
            System.out.printf("x = %2d: ", x);
            if (x == 1 || x == 2) { System.out.println("primo"); continue; }
            if (x > 2)
                for(int i=2; i<x; i++)
                {
                    if (x%i == 0) { System.out.println("nao primo"); break; }
                    if (i == x-1) System.out.println("primo"); }
        }

    }

    public static int lerPositivo()
    {
        int x;
        do {
            System.out.print("Valor positivo: ");
            x = sc.nextInt();
        } while(x <= 0);
        return x;
    }
}
```

```
Valor positivo: 0
Valor positivo: -3
Valor positivo: 15
x =  1:  primo
x =  2:  primo
x =  3:  primo
x =  4:  nao primo
x =  5:  primo
x =  6:  nao primo
x =  7:  primo
x =  8:  nao primo
x =  9:  nao primo
x = 10:  nao primo
x = 11:  primo
x = 12:  nao primo
x = 13:  primo
x = 14:  nao primo
x = 15:  nao primo
Press any key to continue . . .
```


Exercício 5.8

Escreva um programa que produza uma tabela com os valores dos polinômios reais $5x^2 + 10x + 3$ e $7x^3 + 3x^2 + 5x + 2$ calculados para uma sequência regular de valores de x . Os valores inicial e final de x e o número de elementos da sequência são introduzidos pelo utilizador. A leitura dos dados deve exigir que o valor final seja superior ao valor inicial, para garantir valores crescentes de x . Use a função de cálculo de polinômios de 3º grau que fez no exercício 5.2. (Note que um polinômio de 2º grau é um caso particular de polinômio de 3º grau no qual o primeiro coeficiente é nulo.) Apresente a tabela com o formato indicado abaixo.

```
-----
|  x   | 5x2+10x+3 | 7x3+3x2+5x+2 |
| ###.# | #####.## | #####.### |
|-----|
| ...   |
|-----|
| ###.# | #####.## | #####.### |
|-----|
```

```

import java.util.*;
public class e5_8 {
    static Scanner sc = new Scanner(System.in);
    public static int lerValido()
    {
        int x;
        do {
            System.out.print("Valor positivo: ");
            x = sc.nextInt();
        } while(x <= 1);
        return x;
    }
    public static double lerReal() {
        double f;
        System.out.print("Valor real: ");
        f = sc.nextDouble();
        return f;
    }
    public static double pol(double a,double b,double c,double d,double x)
    {
        return a*Math.pow(x,3) + b*Math.pow(x,2) + c*x + d;
    }
    public static void main(String[] args)
    {
        int n = lerValido();
        double inicial = lerReal(), fin;
        do fin = lerReal(); while (fin <= inicial);
        double interval = (fin-inicial)/(n-1);
        System.out.printf("intervalo = %.2f\n",interval);
        System.out.printf("-----\n");
        System.out.printf("|   x   | 5x2 + 10x + 3 | 7x3 + 3x2 + 5x + 2 | \n");
        System.out.printf("-----\n");
        for(double x = inicial; x <= fin; x += interval)
            System.out.printf("| %5.1f | %9.2f | %12.3f | \n", x,pol(0,5,10,3,x),pol(7,3,5,2,x));
        System.out.printf("-----\n");
    }
}

```

```

Valor positivo: 1
Valor positivo: -4
Valor positivo: 5
Valor real: 2.8
Valor real: 7.56
intervalo = 1.19
-----
|   x   | 5x2 + 10x + 3 | 7x3 + 3x2 + 5x + 2 |
-----
|  2.8  |    70.20     |    193.184         |
|  4.0  |   122.50     |    514.359         |
|  5.2  |   188.96     |   1081.340         |
|  6.4  |   269.58     |   1964.905         |
|  7.6  |   364.37     |   3235.829         |
-----
Press any key to continue . . . _

```

Exercício 5.7

Implemente uma função que determine o máximo divisor comum (mdc) entre dois números inteiros. Use o algoritmo de Euclides. Este algoritmo consiste em subtrair o menor número ao maior até que os dois números se tornem iguais. (Para maior eficiência, pode usar o resto da divisão em vez de subtração sucessiva.)

```
import java.util.*;
public class e5_7 {
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    { int A,B;
      System.out.print("Introduza A: ");      A = sc.nextInt();
      System.out.print("Introduza B: ");      B = sc.nextInt();
      System.out.printf("Greatest common divisor = %d\n",GCD(A,B));
    }

    public static int GCD(int A, int B)      {
    int tmp;
    while (B>0)
        if (B > A)  { tmp=A; A=B; B=tmp;}
        else       { tmp=B; B=A%B; A=tmp;}
    return A;
    }
}
```

```
Introduza A: 240
Introduza B: 105
Greatest common divisor = 15
Press any key to continue . . .
```