

# APROG – Algoritmia e Programação



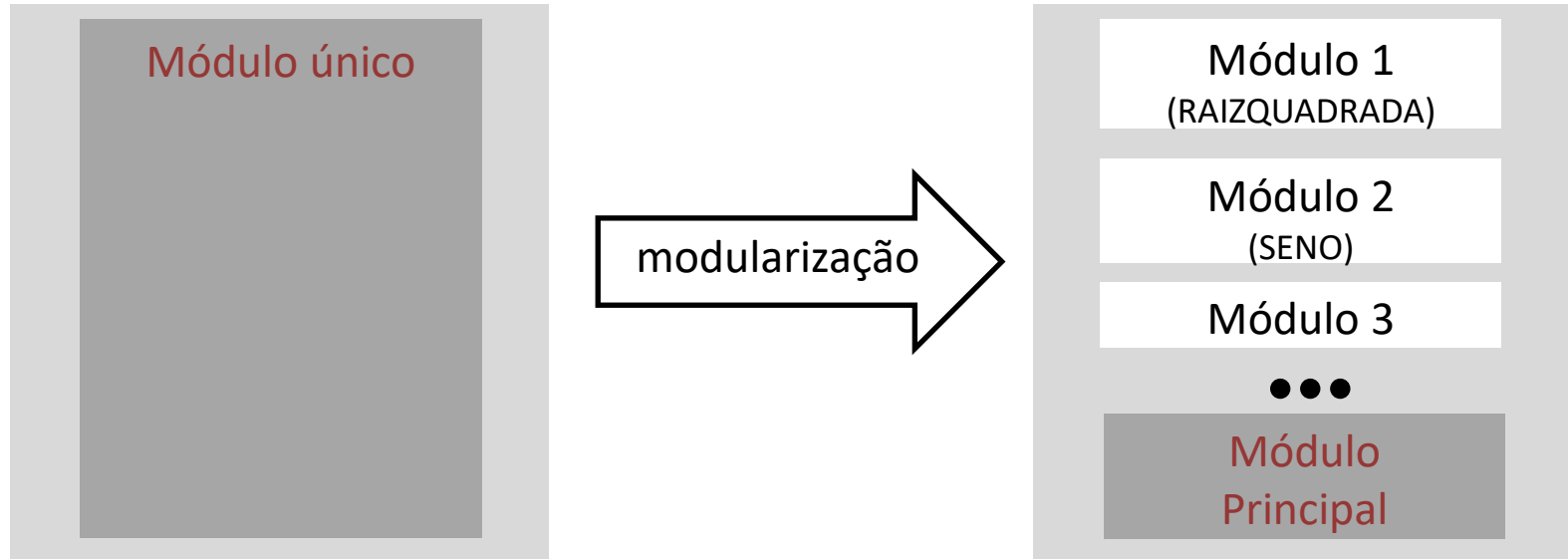
**Modularização**

Emanuel Cunha Silva

[ecs@isep.ipp.pt](mailto:ecs@isep.ipp.pt)

# Modularização

Mecanismo fornecido pelas Linguagens de Programação que permite decomposição dum programa em módulos



## Módulo

- Sequência de instruções que executa uma tarefa específica
- Exemplos
  - RAIZQUADRADA() e SENO()

# Modularização

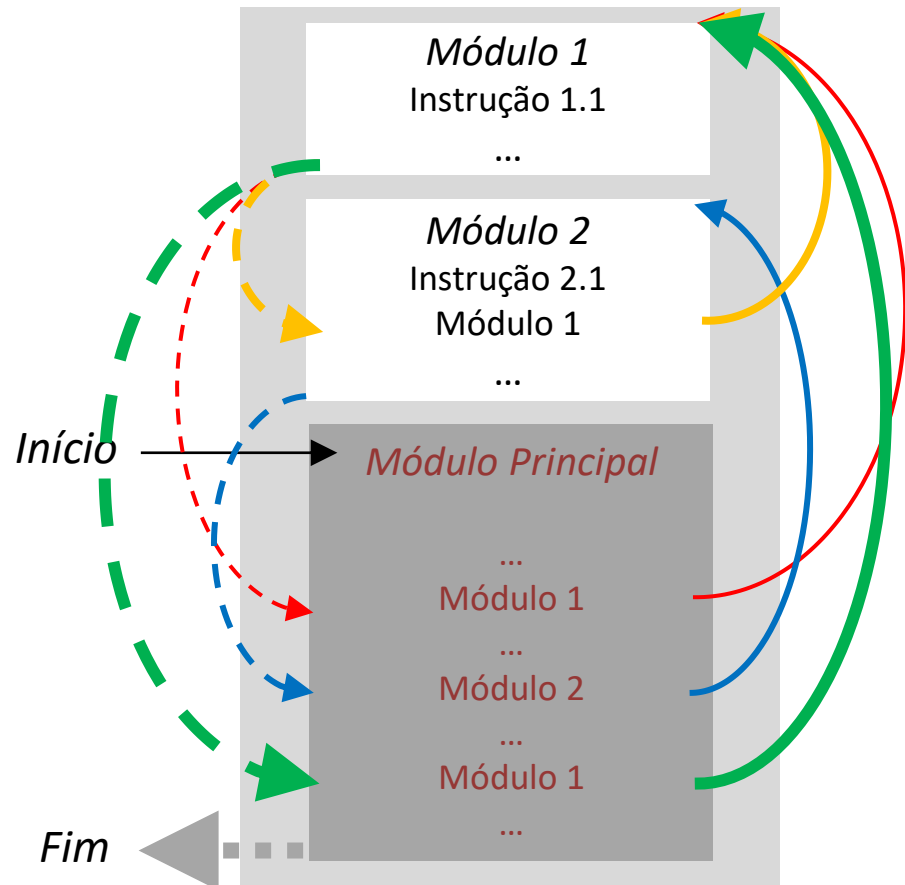
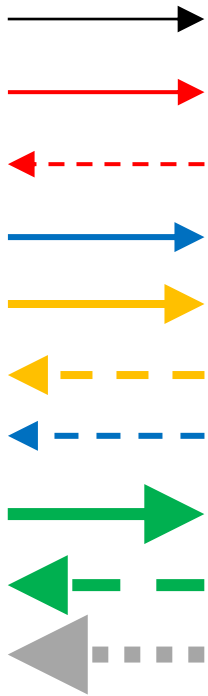
O programa é composto por um **Módulo Principal** e outros módulos

- **Módulo Principal**
  - É sempre o primeiro módulo a ser executado
  - Controla execução do programa
- Na chamada de um módulo, o controlo passa temporariamente para o módulo chamado
- Após a execução de um módulo, o controlo retorna para o módulo que o chamou e para a instrução seguinte à que originou a chamada

# Modularização

- Na chamada de um módulo, o controlo passa temporariamente para o módulo chamado
- Após a execução de um módulo, o controlo retorna para módulo que o chamou

Sequência de execução:



# Modularização - Interesse

- Decomposição de programa em subprogramas mais pequenos
  - Reduzir a complexidade do desenvolvimento
    - Aplicação direta da estratégia dividir-para-conquistar
  - Tornar mais rápido o desenvolvimento
    - Módulos independentes podem ser desenvolvidos em paralelo
- Reutilização de módulos
  - Tornar mais rápido desenvolvimento (evitar redundância)
  - Tornar o programa mais pequeno e simples de ler/compreender
- Abstração da implementação de tarefas
  - Facilitar o desenvolvimento
    - Programador de um módulo abstrai-se dos detalhes da implementação dos outros módulos
  - Exemplo : *RAIZQUADRADA(x)*

# Modularização

**Exemplo:** Calcular combinações

$$C_{(n,r)} = \frac{n!}{r! (n-r)!}$$

INICIO

ED: n, r, resultado, fn, fr, fnr, i INTEIRO

LER (n, r)

//n!

resultado ← 1

REPETIR PARA i ← 2 ATE n PASSO 1

    resultado ← resultado \* i

FIMREPETIR

fn ← resultado

//r!

resultado ← 1

REPETIR PARA i ← 2 ATE r PASSO 1

    resultado ← resultado \* i

FIMREPETIR

fr ← resultado

//(n-r)!

resultado ← 1

REPETIR PARA i ← 2 ATE (n-r) PASSO 1

    resultado ← resultado \* i

FIMREPETIR

fnr ← resultado

ESCREVER (fn / (fr \* fnr))

FIM

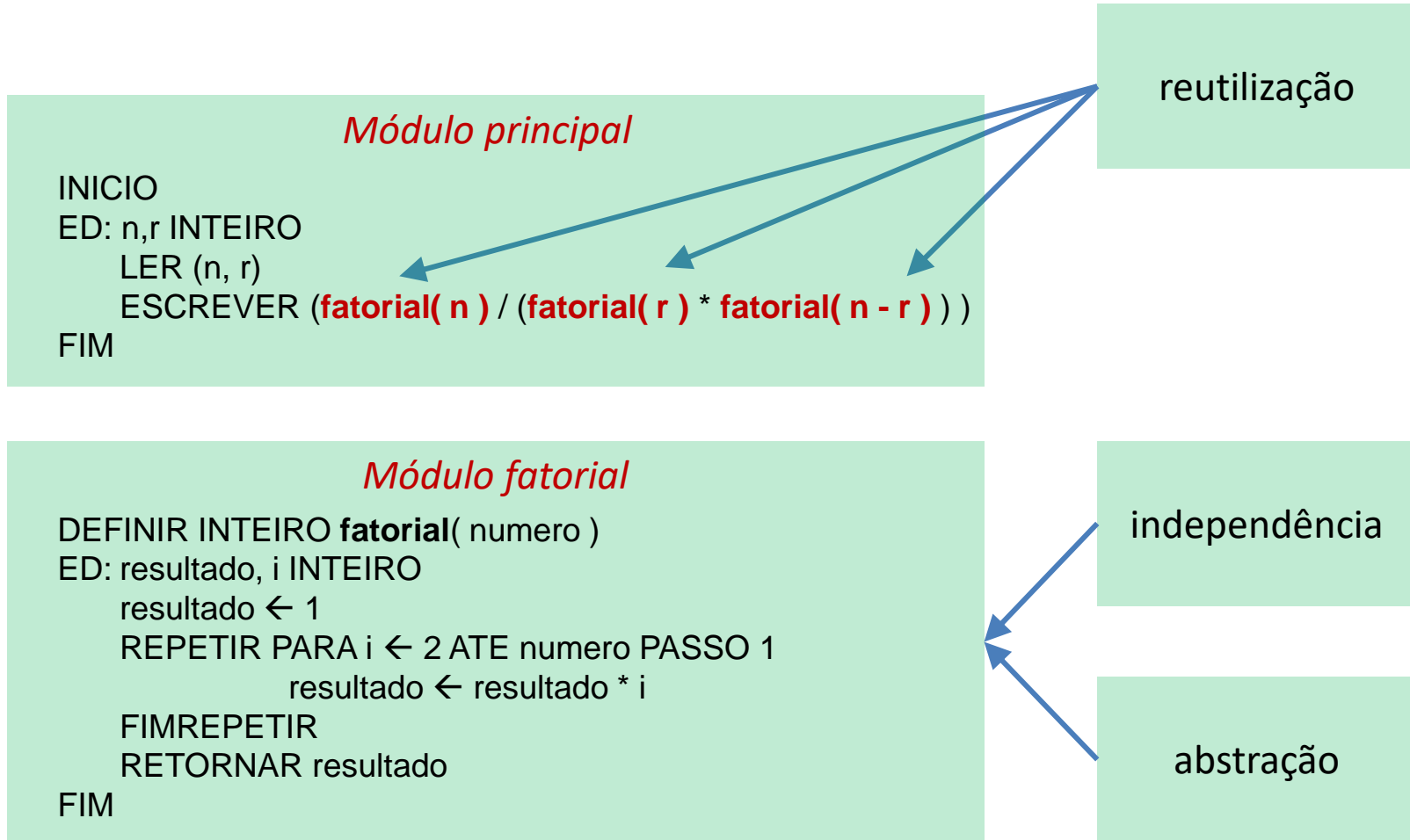
redundância



# Modularização

**Exemplo:** Calcular combinações

$$C_{(n,r)} = \frac{n!}{r! (n-r)!}$$



# Modularização – tipos

## Procedimento

- Módulo que executa uma sequência de instruções
- Exemplo:
  - ESCREVER (“Olá, bom dia”)

## Função

- Módulo que executa uma sequência de instruções e, no final, **retorna um valor** ao ponto da sua chamada
- Exemplo da raiz quadrada de um número
  - resultado  $\leftarrow$  RAIZQUADRADA (25)





# Modularização – âmbito das variáveis

## Variáveis Globais

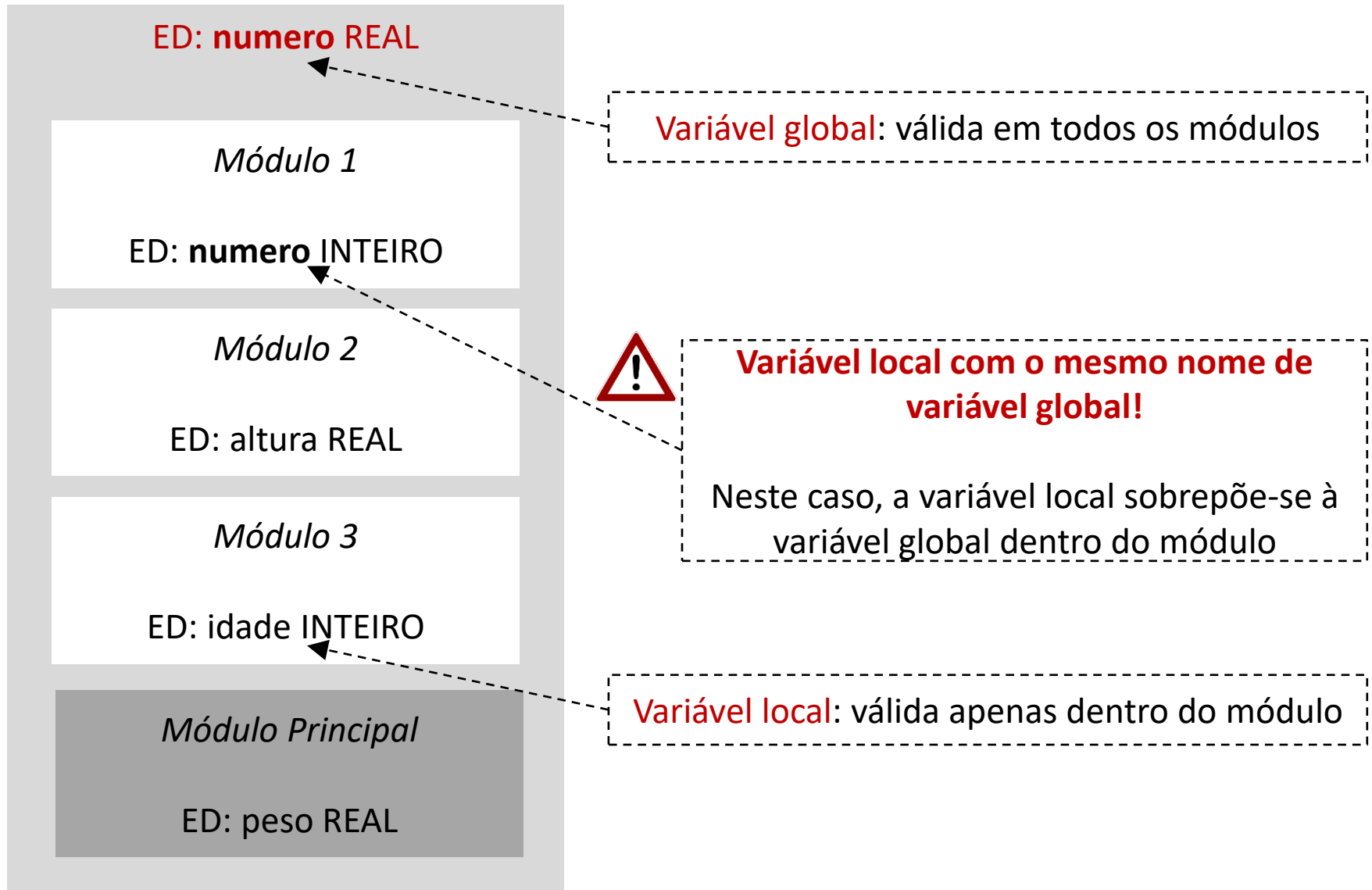
- Acessíveis em todo o programa, dentro de qualquer módulo
- Declaradas fora dos módulos
- **Desvantagens**
  - Dificultam a compreensão do algoritmo porque é necessário analisar em todo o programa a forma como são utilizadas
  - Maior dificuldade em detetar eventuais erros
  - Dificultam a reutilização dos módulos noutros algoritmos que usem as mesmas variáveis globais
- **Recomendação**
  - Evitar o uso de variáveis globais
  - Usar variáveis locais
  - Comunicar entre módulos com passagem de parâmetros

# Modularização – âmbito das variáveis

## Variáveis Locais

- Declaradas dentro dos módulos
- Acessíveis apenas no módulo onde são declaradas
- Só existem durante a execução do respetivo módulo
- **Vantagens**
  - Independência dos módulos
  - Facilitam o desenvolvimento dos programas

# Modularização – âmbito das variáveis



# Modularização – passagem de parâmetros

## Passagem de parâmetros

- Meio de Comunicação entre Módulos Independentes
- Suportam a transferência de informação entre módulos
  - Entrada de Dados
  - Saída de Resultados
- Os parâmetros funcionam como variáveis locais, só são visíveis dentro do módulo e são inicializadas com os valores dos argumentos de chamada
- Tipos de Parâmetros
  - **Atuais**
    - valores passados para o módulo chamado para inicializar os seus parâmetros formais (normalmente, designados por argumentos)
  - **Formais**
    - identificadores que recebem os valores passados para esse módulo (normalmente, designados apenas por parâmetros)

# Modularização – passagem de parâmetros

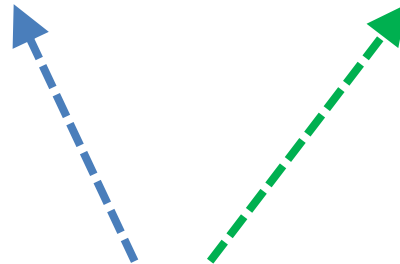
## Correspondência entre parâmetros atuais e parâmetros formais

- A correspondência é feita pela ordem dos parâmetros formais
- Primeiro parâmetro formal liga-se ao primeiro parâmetro atual e assim por diante
- Exemplo:

**maior**(*INTEIRO* **numero1**, *INTEIRO* **numero2** )

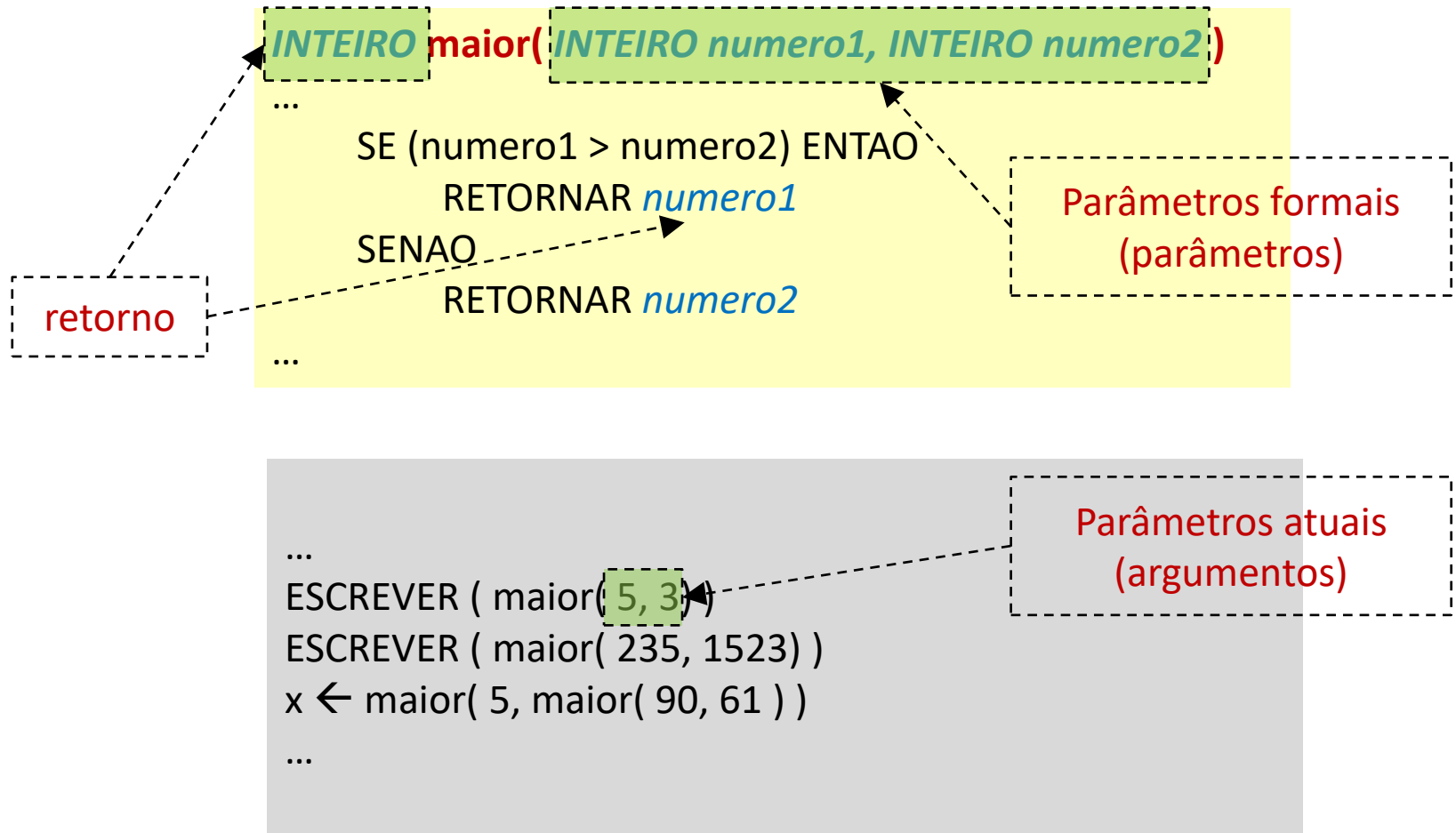
...

**maior** ( **5**, **12** )



# Modularização – passagem de parâmetros

## Exemplo:



# Modularização

## ▪ Função

### ▪ Sintaxe

```
DEFINIR tipo_retorno nome (tipo pf1, ..., tipo pfn)           // cabeçalho
ED:      // variáveis e constantes locais
INÍCIO
    // corpo da função
    RETORNAR valor_do_tipo_retorno
FIMDEFINIR
```

### ▪ Exemplo

```
DEFINIR INTEIRO maior(INTEIRO x, INTEIRO y)
ED: INTEIRO resultado
INÍCIO
    SE (x > y) ENTÃO
        resultado ← x
    SENÃO
        resultado ← y
    FIMSE
    RETORNAR resultado           // resultado tem de ser INTEIRO (tipo_retorno)
FIMDEFINIR
```

# Modularização

## ▪ Procedimento

### ▪ Sintaxe

```
DEFINIR nome (tipo pf1, ..., tipo pfn)           // cabeçalho
ED:      // variáveis e constantes locais
INÍCIO
    // corpo do procedimento
FIMDEFINIR                                     // não tem retorno
```

### ▪ Exemplo

```
DEFINIR mostrarNumerosDoIntervalo(INTEIRO limiteInf, INTEIRO limiteSup)
ED: INTEIRO x;
INÍCIO
    REPETIR PARA x ← limiteInf ATE limiteSup PASSO 1
        ESCREVER ( x )
    FIMREPETIR
FIMDEFINIR
```



# Modularização

## Problema:

Leia um número inteiro e mostre os seus algarismos linha a linha. Mostre também quantos algarismos possui esse número.

Desenvolva um programa adotando uma estrutura modular. Para tal crie um procedimento para mostrar os algarismos e uma função para calcular e retornar a quantidade de algarismos de um número inteiro.

```
ED: INTEIRO numero
INÍCIO
    LER ( numero )
    mostrarAlgarismos ( numero )
    ESCREVER ( contarAlgarismos ( numero ) )
FIM
```

```
DEFINIR mostrarAlgarismos(INTEIRO numero)
ED: INTEIRO algarismo
INÍCIO
    REPETIR ENQUANTO ( numero > 9 )
        algarismo ← numero % 10
        ESCREVER ( algarismo )
        numero ← numero / 10
    FIMREPETIR
    ESCREVER ( numero )
FIMDEFINIR
```

Procedimento

```
DEFINIR INTEIRO contarAlgarismos(INTEIRO numero)
ED: INTEIRO qtd
INÍCIO
    qtd ← 1
    REPETIR ENQUANTO ( numero > 9 )
        numero ← numero / 10
        qtd ← qtd + 1
    FIMREPETIR
    RETORNAR ( qtd )
FIMDEFINIR
```

Função

# <Java métodos>

# Java - métodos

Um método é uma sequência de instruções que realiza uma tarefa específica e no final pode retornar:

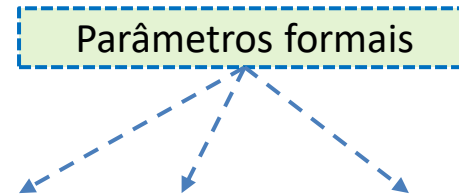
- Um valor de um determinado tipo (int, double, String, ...)
- Nada (void)
- Tipos de Métodos
  - Métodos de **instância**
  - Métodos de **classe**
- **Métodos de Instância**
  - Aplicam-se aos objetos da classe ( Ex: Scanner **ler** = new Scanner(System.in);  
String s = **ler.next()**; )
- **Métodos de Classe**
  - Aplicam-se à classe ( Ex: double a = **Math.sqrt**(25.0) )
  - Declarados como métodos estáticos (static) ( Ex: public **static** double sqrt(double x) )

# Java - métodos de classe

## ▪ Declaração

### ▪ Tipo Procedimento

```
modificador_acesso static void nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
}
```



### ▪ Tipo Função

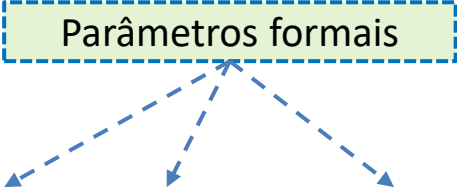
```
modificador_acesso static tipo_retorno nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
    return expressão;  
}
```

Obs: O cabeçalho dos métodos de classe **inclui** o prefixo **static**

# Java - métodos de instância

## ▪ Declaração

Parâmetros formais



### ▪ Tipo Procedimento

```
modificador_acesso void nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
}
```

### ▪ Tipo Função

```
modificador_acesso tipo_retorno nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
    return expressão;  
}
```

Obs: O cabeçalho dos métodos de instância **não inclui** o prefixo **static**

# Java - métodos

## ▪ Declaração

### ▪ Tipo Procedimento

```
modificador_acesso void nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
}
```

### ▪ Tipo Função

```
modificador_acesso tipo_retorno nome(tipo pf1, tipo pf2, ..., tipo pfn) {  
    ...  
    return expressão;  
}
```

## ▪ Modificadores de Acesso

- **private** // método só pode ser chamado por métodos da **própria** classe
- **public** // método pode ser chamado por métodos de **qualquer** classe
- Sem modificador // método pode ser chamado por métodos de classes do mesmo package

# Java - métodos

- Parâmetro Formal de **Tipo Primitivo**
  - Tipos primitivos: **byte, short, int, long, float, double, boolean, char**
  - É passada a **cópia de um valor**
  - Método chamado **não tem acesso** ao parâmetro atual
  - Alterações posteriores do parâmetro formal **não afetam** parâmetro actual

- **Exemplo:**

```
public class Exemplo{
    public static void main(String[] args){
        int c=0;
        metodo(c);
        System.out.println("C=" + c);
    }

    private static void metodo(int c){
        c=1;
    }
} // Programa escreve C=0
```

# Java - métodos

- Parâmetro Formal de **Tipo Não Primitivo**
  - Tipos não primitivos
    - Tipos criados pelo programador
    - Estruturas indexadas (arrays)
      - O identificador é seguido de []
      - Exemplo: int[], float[], double[], String[], ...
  - É passada a **cópia de uma referência**
  - Método chamado **tem acesso** ao parâmetro atual
  - Alterações posteriores do parâmetro formal **afetam** parâmetro actual
  - String é exceção – é imutável



# Java - métodos

## Problema:

Leia um número inteiro e mostre os seus algarismos linha a linha. Mostre também quantos algarismos possui esse número.

Desenvolva um programa adotando uma estrutura modular. Para tal crie um procedimento para mostrar os algarismos e uma função para calcular e retornar a quantidade de algarismos de um número inteiro.

```
public class Algarismos {  
  
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
        int numero;  
  
        numero = ler.nextInt();  
        mostrarAlgarismos(numero);  
        System.out.println(contarAlgarismos(numero));  
    }  
}
```

```
public static void mostrarAlgarismos(int numero) {  
    int algarismo;  
    while (numero > 9) {  
        algarismo = numero % 10;  
        System.out.println(algarismo);  
        numero = numero / 10;  
    }  
    System.out.println(numero);  
}
```

Procedimento

```
public static int contarAlgarismos(int numero) {  
    int qtd = 1;  
    while (numero > 9) {  
        numero = numero / 10;  
        qtd = qtd + 1;  
    }  
    return (qtd);  
}
```

Função

```
}
```

**</Java métodos>**

# <Java Strings>

# Java - Strings

## String

- Cadeia de caracteres = texto
- **Exemplos**
  - "ISEP"
  - "Algoritmia e Programação"
  - "JUPITER"

- **Declaração**

```
String v = "";                // String vazia
String s = "ISEP";
String nome = "Ana Berta Carla Dionísio";
```

# Java - Strings

## Carateres especiais

- Strings podem conter caracteres especiais com funções específicas (sequência de escape)
- Cada sequência de escape é um único carácter embora seja escrito com dois símbolos.

<code>\t</code>	Inserir um <i>tab</i> no texto
<code>\b</code>	Inserir um <i>backspace</i> no texto
<code>\n</code>	Inserir uma mudança de linha no texto
<code>\r</code>	Inserir um <i>carriage return</i> no texto
<code>\'</code>	Inserir uma plica no texto.
<code>\"</code>	Inserir uma aspa no texto.
<code>\\</code>	Inserir uma barra para trás ( <i>backslash</i> ) no texto

## ■ Exemplos

```
String s = "Aprog\t, é \tfixe\t e fácil";
```

```
//s="Aprog   , é "fixe" e fácil"
```

# Java - Strings

## String

```
String s = "Bom dia, Rui";
```

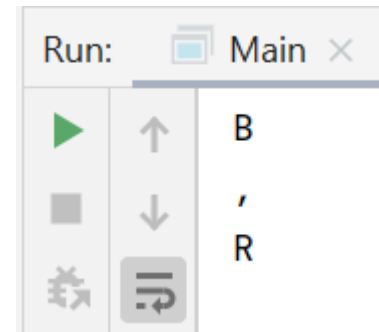
B	o	m		d	i	a	,		R	u	i
0	1	2	3	4	5	6	7	8	9	10	11

conteúdo

posições

a posição inicial é zero (0) e não um (1)

```
System.out.println( s.charAt(0) );  
System.out.println( s.charAt(7) );  
System.out.println( s.charAt(9) );
```



# Java - Strings

## Métodos

- int **length()**
- String **toLowerCase()**
- String **toUpperCase()**
- int **compareTo**(String outraString)
- boolean **equals**(String outraString)
- boolean **equalsIgnoreCase**(String outraString)
- String **trim()**
- char **charAt**( int índice )
- ...

# Java - Strings

- **Exemplos** // considerando: `String s = "Aprog";`

- `int` **length()**

- Retorna comprimento da string (quantidade de caracteres)

```
int comprimento = s.length();           // comprimento = 5
```

- `String` **toLowerCase()**

- Retorna a string com todas as letras minúsculas

```
String s2 = s.toLowerCase();           // s2 = "aprog"
```

- `String` **toUpperCase()**

- Retorna a string com todas as letras maiúsculas

```
String s2 = s.toUpperCase();           // s2 = "APROG"
```



# Java - Strings

- **Comparação de Strings**

**Não se pode usar os operadores relacionais para comparar Strings**

É necessário verificar se o conjunto de caracteres é o mesmo e pela mesma ordem

- **Exemplos** // considerando: String s = "Aprog";
- int **compareTo**(String outraString)
  - Compara duas strings alfabeticamente e retorna um número inteiro
    - Negativo Ex: s.compareTo("PPROG") s é anterior a "PPROG"
    - Positivo Ex: s.compareTo("ALIN") s é posterior a "ALIN"
    - Zero Ex: s.compareTo("Aprog") s e "Aprog" são iguais

```
if ( s.compareTo("APROG")==0 )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

# Java - Strings

- **Exemplos** // considerando: String s = "Aprog";

- boolean **equals**(String outraString)

- Distingue maiúsculas de minúsculas

s ≠ "APROG"

```
if ( s.equals("APROG") )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

- boolean **equalsIgnoreCase**(String outraString)

- Compara duas strings alfabeticamente sem distinguir maiúsculas de minúsculas e retorna:

- true            caso sejam iguais
- false          caso sejam diferentes

s = "APROG"

```
if ( s.equalsIgnoreCase("APROG") )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

# Java - Strings

- **Exemplos** // considerando: `String s = "Aprog";`

- `char charAt( int índice )`

- Retorna o caráter que se encontra na posição “índice”

```
char c = s.charAt(0);           // c = 'A'
```

```
char c = s.charAt(1);           // c = 'p'
```

- `String trim()`

- Retorna cópia da string sem espaços brancos iniciais e finais

```
String s1 = "    APROG    ";
```

```
String s2 = s1.trim();           // s2 = "APROG"
```

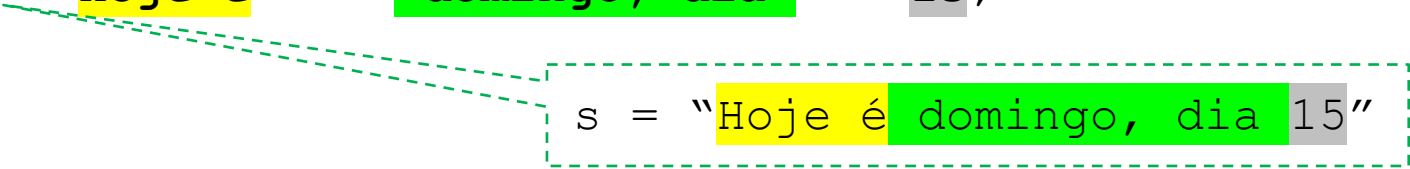
# Java - Strings

- **Concatenar Strings** - Usa-se o operador **+**

Permite criar uma nova string a partir da junção dos valores de várias strings ou números. O primeiro elemento da junção tem de ser uma string.

- **Exemplos**

```
String s = "Hoje é" + "domingo, dia" + 15;
```



s = "Hoje é domingo, dia 15"

```
String s1 = "D";
```

```
String s2 = "ia";
```

```
int dia = 15;
```

```
String s = s1.trim() + s2 + dia;
```



s = "Dia15"

**</Java Strings>**

## <modularização – casos de uso>

# Java - métodos

**Problema1(v1):** Ler uma frase e mostrar quantas vogais existem nessa frase.

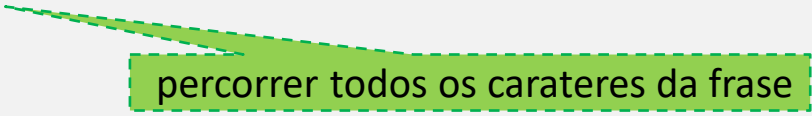
```
public class VogaisMetodoUnico {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        String frase;
        int qtdVogais=0;

        frase=ler.nextLine();

        frase = frase.toLowerCase();           //passar tudo para minúsculas para não repetir

        for(int i=0; i<frase.length(); i++){
            switch(frase.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais++;
                        break;
            }
        }
        System.out.println("Vogais=" + qtdVogais);
    }
}
```



percorrer todos os caracteres da frase

# Java - métodos

**Problema2(v1):** Ler duas frases e mostrar quantas vogais tem cada uma.

```
public class VogaisMetodoUnico {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        String frase1;
        int qtdVogais1=0;
        frase1=ler.nextLine();
        frase1 = frase1.toLowerCase(); //passar tudo para minúsculas para não repetir
        for(int i=0; i<frase1.length(); i++){
            switch(frase1.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais1++; break;
            }
        }
        System.out.println("Vogais1="+qtdVogais1);
        //-----
        String frase2;
        int qtdVogais2=0;
        frase2=ler.nextLine();
        frase2 = frase2.toLowerCase();
        for(int i=0; i<frase1.length(); i++){
            switch(frase2.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais2++; break;
            }
        }
        System.out.println("Vogais2="+qtdVogais2);
    }
}
```

[Copy + Paste] (duplicar o código)  
+ alterar o nome das variáveis

possível ocorrência de erros:  
Depois da cópia, escapou à alteração



# Java - métodos

**Problema2(v1):** Ler duas frases e mostrar quantas vogais tem cada uma.

```
public class VogaisMetodoUnico {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        String frase1;
        int qtdVogais1=0;
        frase1=ler.nextLine();
        frase1 = frase1.toLowerCase(); //passar tudo para minúsculas para não repetir
        for(int i=0; i<frase1.length(); i++){
            switch(frase1.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais1++; break;
            }
        }
        System.out.println("Vogais1="+qtdVogais1);
        //-----
        String frase2;
        int qtdVogais2=0;
        frase2=ler.nextLine();
        frase2 = frase2.toLowerCase();
        for(int i=0; i<frase1.length(); i++){
            switch(frase2.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais2++; break;
            }
        }
        System.out.println("Vogais2="+qtdVogais2);
    }
}
```

## Desvantagens:

- Código duplicado
- Código extenso e difícil de ler
- Maior possibilidade de erros
- Mais difícil manter o código

# Java - métodos

Problema2.1: Ler **quatro** frases e mostrar quantas vogais tem cada uma.

```
public class VogaisMetodoUnico {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        String frase1;
        int qtdVogais1=0;
        frase1=ler.nextLine();
        frase1 = frase1.toLowerCase(); //passar tudo para minúsculas para não repetir
        for(int i=0; i<frase1.length(); i++){
            switch (frase1.charAt(i)){
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': qtdVogais1++; break;
            }
        }
        System.out.println("Vogais1="+qtdVogais1);
    }
    //-----
    String frase2;
    int qtdVogais2=0;
    frase2=ler.nextLine();
    frase2 = frase2.toLowerCase();
    for(int i=0; i<frase1.length(); i++){
        switch (frase2.charAt(i)){
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': qtdVogais2++; break;
        }
    }
    System.out.println("Vogais2="+qtdVogais2);
}
//-----
String frase2;
int qtdVogais2=0;
frase2=ler.nextLine();
frase2 = frase2.toLowerCase();
for(int i=0; i<frase1.length(); i++){
    switch (frase2.charAt(i)){
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': qtdVogais2++; break;
    }
}
System.out.println("Vogais2="+qtdVogais2);
}
//-----
String frase2;
int qtdVogais2=0;
frase2=ler.nextLine();
frase2 = frase2.toLowerCase();
for(int i=0; i<frase1.length(); i++){
    switch (frase2.charAt(i)){
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': qtdVogais2++; break;
    }
}
System.out.println("Vogais2="+qtdVogais2);
}
}
```

Parece razoável ???



# Java - métodos

**Problema1(v2):** Ler uma frase e mostrar quantas vogais existem nessa frase.

```
public class VogaisMetodoSeparado {
```

```
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
  
        String frase;  
        frase=ler.nextLine();  
  
        System.out.println("Vogais=" + qtdVogaisNaFrase(frase));  
    }
```

Abstração da implementação

```
    public static int qtdVogaisNaFrase(String frase){  
        int qtdVogais=0;  
        frase = frase.toLowerCase();        //passar tudo para minúsculas para não repetir  
  
        for(int i=0; i<frase.length(); i++){  
            switch(frase.charAt(i)){  
                case 'a':  
                case 'e':  
                case 'i':  
                case 'o':  
                case 'u': qtdVogais++;  
                    break;  
            }  
        }  
        return qtdVogais;  
    }  
}
```

# Java - métodos

**Problema2(v2):** Ler duas frases e mostrar quantas vogais tem cada uma.

```
public class VogaisMetodoSeparado {
```

```
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
  
        String frase1 = ler.nextLine();  
        String frase2 = ler.nextLine();  
        System.out.println("Vogais1=" + qtdVogaisNaFrase(frase1));  
        System.out.println("Vogais2=" + qtdVogaisNaFrase(frase2));  
    }
```

Não há [Copy + Paste] do código  
Há reutilização do código

```
    public static int qtdVogaisNaFrase(String frase){  
        int qtdVogais=0;  
        frase = frase.toLowerCase(); //passar tudo para minúsculas para não repetir  
  
        for(int i=0; i<frase.length(); i++){  
            switch(frase.charAt(i)){  
                case 'a':  
                case 'e':  
                case 'i':  
                case 'o':  
                case 'u': qtdVogais++;  
                        break;  
            }  
        }  
        return qtdVogais;  
    }
```

O algoritmo está num local único e não  
tem de ser alterado fora daqui

# Java - métodos

**Problema1(v3):** Ler uma frase e mostrar quantas vogais existem nessa frase.

```
public class VogaisMetodoSeparadoAbstrato {  
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
  
        String frase;  
        frase=ler.nextLine();  
        System.out.println("Vogais=" + qtdVogaisNaFrase(frase));  
    }  
}
```

```
public static int qtdVogaisNaFrase(String frase){  
    return  qtdOcorrenciasDoSimbolo(frase, 'a')+  
           qtdOcorrenciasDoSimbolo(frase, 'e')+  
           qtdOcorrenciasDoSimbolo(frase, 'i')+  
           qtdOcorrenciasDoSimbolo(frase, 'o')+  
           qtdOcorrenciasDoSimbolo(frase, 'u');  
}
```

Há reutilização do código

A lógica de cada invocação está num local único

```
public static int qtdOcorrenciasDoSimbolo(String frase, char simbolo){  
    int qtd=0;  
  
    for(int i=0; i<frase.length(); i++){  
        if(frase.charAt(i) == simbolo)  
            qtd++;  
    }  
    return qtd;  
}
```

Criar um método mais genérico

# Java - métodos

**Problema3:** Ler uma frase e mostrar quantas vogais, vírgulas e pontos de interrogação existem nessa frase.

```
public class VogaisMetodoSeparadoAbstrato {  
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
  
        String frase = ler.nextLine();  
        System.out.println("Vogais=" + qtdVogaisNaFrase(frase));  
        System.out.println("Vírgulas=" + qtdOcorrenciasDoSimbolo(frase, ','));  
        System.out.println("Pontos de interrogação=" + qtdOcorrenciasDoSimbolo(frase, '?'));  
    }  
  
    public static int qtdVogaisNaFrase(String frase){  
        return qtdOcorrenciasDoSimbolo(frase, 'a')+  
               qtdOcorrenciasDoSimbolo(frase, 'e')+  
               qtdOcorrenciasDoSimbolo(frase, 'i')+  
               qtdOcorrenciasDoSimbolo(frase, 'o')+  
               qtdOcorrenciasDoSimbolo(frase, 'u');  
    }  
  
    public static int qtdOcorrenciasDoSimbolo(String frase, char simbolo){  
        int qtd=0;  
  
        for(int i=0; i<frase.length(); i++){  
            if(frase.charAt(i) == simbolo)  
                qtd++;  
        }  
        return qtd;  
    }  
}
```

# Java - métodos

**Problema 4:** Ler a idade de uma pessoa. Considerar apenas o intervalo [0,120];

```
public class LerIdadeValida {  
    public static void main(String[] args) {  
  
        int idade = lerInteiroNoIntervalo(0,120);  
        System.out.println("Idade="+idade);  
    }  
  
    public static int lerInteiroNoIntervalo(int lim1, int lim2){  
        Scanner ler = new Scanner(System.in);  
        int numero;  
  
        System.out.println("Numero:["+lim1+", "+lim2+"] :");  
        numero=ler.nextInt();  
        while(numero < lim1 || numero > lim2){  
            System.out.println("Numero:["+lim1+", "+lim2+"] :");  
            numero=ler.nextInt();  
        }  
        return numero;  
    }  
}
```

# Java - métodos

**Problema 5:** Ler a idade, o peso e a altura de uma pessoa. Considerar apenas válidos valores dentro dos intervalos: idade=[0,120]; peso=[100g,300000g]; altura=[30cm,250cm];

```
public class LerIdadeValida {  
    public static void main(String[] args) {  
        int idade = lerInteiroNoIntervalo(0,120);  
        int peso = lerInteiroNoIntervalo(100,300000);  
        int altura = lerInteiroNoIntervalo(30,250);  
    }  
}
```

Há reutilização do código

Parâmetros distintos para usos diferentes

```
public static int lerInteiroNoIntervalo(int lim1, int lim2){  
    Scanner ler = new Scanner(System.in);  
    int numero;  
  
    System.out.println("Numero: ["+lim1+", "+lim2+"] :");  
    numero=ler.nextInt();  
    while(numero < lim1 || numero > lim2){  
        System.out.println("Numero: ["+lim1+", "+lim2+"] :");  
        numero=ler.nextInt();  
    }  
    return numero;  
}
```

método já existente



# Java - métodos

**Problema 6:** Ler uma string e verificar se é elegível para password. Uma password tem de conter no mínimo 8 símbolos e pelo menos um elemento de cada um dos seguintes conjuntos: letras maiúsculas, minúsculas e algarismos.

```
public class ValidarPassword {  
  
    public static void main(String[] args) {  
  
        Scanner ler = new Scanner(System.in);  
  
        String password = ler.nextLine();  
  
        if( passwordValida(password))  
            System.out.println("password válida");  
        else  
            System.out.println("password NAO É válida");  
    }  
  
    ...  
}
```

Abstração da implementação



# Java - métodos

**Problema 6:** Ler uma string e verificar se é elegível para password. Uma password tem de conter no mínimo 8 símbolos e pelo menos um elemento de cada um dos seguintes conjuntos: letras maiúsculas, minúsculas e algarismos.

```
...
public static boolean passwordValida(String pwd) {
    boolean existeMaiuscula = false;
    boolean existeMinuscula = false;
    boolean existeNumero = false;
    char simbolo;

    if(pwd.length() < 8)
        return false;

    for(int i=0; i < pwd.length(); i++) {
        simbolo = pwd.charAt(i);
        if( Character.isDigit(simbolo)) {
            existeNumero = true;
        } else if (Character.isUpperCase(simbolo)) {
            existeMaiuscula = true;
        } else if (Character.isLowerCase(simbolo)) {
            existeMinuscula = true;
        }
    }

    if(existeNumero && existeMaiuscula && existeMinuscula)
        return true;
    else
        return false;
}
```

A lógica da validação está num local único e é completamente transparente para o resto do programa

# Java - métodos

**Problema 6.1:** Ler uma string e verificar se é elegível para password. Uma password tem de conter no mínimo **12** símbolos e pelo menos **dois** elementos de cada um dos seguintes conjuntos: letras maiúsculas, minúsculas e algarismos.

```
...
public static boolean passwordValida(String pwd) {
    int qtdMaiuscula = 0;
    int qtdMinuscula = 0;
    int qtdNumero = 0;
    char simbolo;

    if(pwd.length() < 12)
        return false;

    for(int i=0;i < pwd.length();i++) {
        simbolo = pwd.charAt(i);
        if( Character.isDigit(simbolo)) {
            qtdNumero++;
        } else if (Character.isUpperCase(simbolo)) {
            qtdMaiuscula++;
        } else if (Character.isLowerCase(simbolo)) {
            qtdMinuscula++;
        }
    }
    if(qtdNumero>1 && qtdMaiuscula>1 && qtdMinuscula>1)
        return true;
    else
        return false;
}
```

Se mudar a lógica da validação, as alterações serão efetuadas apenas neste módulo.

Não tem qualquer impacto no código dos outros módulos

# Java - métodos

**Problema 7 (v1):** Quantas vezes surge o algarismo 3 entre os números 100 e 1000?

```
public class OcorrenciasDoAlgarismo3 {  
  
    public static void main(String[] args) {  
        int qtd = 0;  
        for (int numero = 100; numero <= 1000; numero++) {  
            int num = numero;  
            while (num > 9) {  
                int resto = num % 10;  
                num = num / 10;  
                if (resto == 3) {  
                    qtd++;  
                }  
            }  
            if (num == 3) {  
                qtd++;  
            }  
        }  
        System.out.println(qtd);  
    }  
}
```

# Java - métodos

**Problema 7 (v2):** Quantas vezes surge o algoritmo 3 entre os números 100 e 1000?

```
public class OcorrenciasDoAlgarismo3NoIntervalo {  
  
    public static void main(String[] args) {  
        System.out.println(qtdDe3NoIntervalo(100, 1000));  
    }  
  
    public static int qtdDe3NoIntervalo(int lim1, int lim2) {  
        int qtd = 0;  
        for (int numero = lim1; numero <= lim2; numero++) {  
            int num = numero;  
            while (num > 9) {  
                int resto = num % 10;  
                num = num / 10;  
                if (resto == 3) {  
                    qtd++;  
                }  
            }  
            if (num == 3) {  
                qtd++;  
            }  
        }  
        return qtd;  
    }  
}
```

Intervalo de pesquisa definido pelos parâmetros

Algoritmo a comparar é fixo !

# Java - métodos

**Problema 7 (v3):** Quantas vezes surge o algarismo 3 entre os números 100 e 1000?

```
public class OcorrenciasDeAlgarismoNoIntervalo {  
    public static void main(String[] args) {  
        System.out.println(qtdDoAlgarismoNoIntervalo(3, 100, 140));  
    }  
  
    public static int qtdDoAlgarismoNoIntervalo(int algarismo, int lim1, int lim2) {  
        int qtd = 0;  
        for (int numero = lim1; numero <= lim2; numero++) {  
            int num = numero;  
            while (num > 9) {  
                int resto = num % 10;  
                num = num / 10;  
                if (resto == algarismo) {  
                    qtd++;  
                }  
            }  
            if (num == algarismo) {  
                qtd++;  
            }  
        }  
        return qtd;  
    }  
}
```

Intervalo de pesquisa definido pelos parâmetros

Algarismo a comparar definido por parâmetro

# Java - métodos

**Problema 8:** Quantas vezes surge o algarismo 3, 1 e 5 entre os números 100 e 1000 e o 2 e 4 entre os números 11 e 64?

```
public class OcorrenciasDeAlgarismoNoIntervalo {
    public static void main(String[] args) {
        System.out.println(qtdDoAlgarismoNoIntervalo(3, 100, 1000));
        System.out.println(qtdDoAlgarismoNoIntervalo(1, 100, 1000));
        System.out.println(qtdDoAlgarismoNoIntervalo(5, 100, 1000));
        System.out.println(qtdDoAlgarismoNoIntervalo(2, 11, 64));
        System.out.println(qtdDoAlgarismoNoIntervalo(4, 11, 64));
    }

    public static int qtdDoAlgarismoNoIntervalo(int algarismo, int lim1, int lim2) {
        int qtd = 0;
        for (int numero = lim1; numero <= lim2; numero++) {
            int num = numero;
            while (num > 9) {
                int resto = num % 10;
                num = num / 10;
                if (resto == algarismo) {
                    qtd++;
                }
            }
            if (num == algarismo) {
                qtd++;
            }
        }
        return qtd;
    }
}
```

Há abstração da implementação.  
Há reutilização do código

**</modularização – casos de uso>**