Programação modular/ Funções

- Introdução à programação modular
  (Livro, pág. 199-217)
- Funções
- Tipos primitivos como argumentos
- Visibilidade das variáveis
- Exemplos

Introdução

- 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`).

Estrutura de um programa - funções

- Estrutura de um programa (relembrar):

```
inclusão de classes externas

public class Programa
{
  public static void main (String[] args)
  {
    declaração de constantes e variáveis
    sequências de instruções
  }
  funções desenvolvidas pelo programador
}

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

- As funções são criadas depois da definição da função `main`.

Definição de uma função (1)

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

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

```
public static tipo nome (argumentos)
{
  declaração de variáveis
  sequências de instruções
  return valor // devolve valor do tipo da função
}
```

Definição de uma função (2)

- 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 programa.
- 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 - funções

```
public static int lerPositivo(){
  Scanner kb = new Scanner(System.in);
  int x;
  do{
    System.out.print("Valor inteiro: ");
    x = kb.nextInt();
  }while(x < 0);
  return x; // devolução do valor lido através do
  teclado, após validação
}
public static int soma (int x, int y){
  int soma;
  soma = x + y;
  return soma;
}
```

Exemplo – uso das funções

```
// Soma de dois números positivos
public static void main(String[] args) {
   int a, b, r;
   a = lerPositivo(); // utilização das funções definidas
   b = lerPositivo();
   r = soma(a, b);  // o valor de a e b são passados à
   função soma
   printf("%d + %d = %d\n", a, b, r);
}
public static int lerPositivo(){
   …
}
public static int soma (int x, int y){
// neste exemplo x = a e y = b
   …
}
```

Outro exemplo

```
... main(...){
  int num = lerInteiroNoIntervalo(1, 10);
  imprimeTabela(num);
}
public static int lerInteiroNoIntervalo(int linf, int lsup){
  Scanner kb = new Scanner(System.in);
  int n;
  do{
    System.out.print("Valor inteiro: ");
    n = kb.nextInt();
  }while(n < linf || n > lsup);
  return n;
}
public static void imprimeTabela(int n){
  for(int i = 0 ; i <= 10 ; i++)
    printf("%2d X %2d = %3d\n", n, i, n*i);
}
```

Tipos primitivos como argumentos

- Tomando como exemplo o programa do slide 8 (soma de dois positivos), podemos ver que a função `lerPositivo` não recebe argumentos e devolve um valor inteiro.
- Quando chamada duas vezes dentro da função `main`, o seu valor de retorno é armazenado nas variáveis `a` e `b`.
- 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 parâmetros `x` e `y` respetivamente (`x = a`, `y = b`).
- Sempre que uma função é usada, o computador "salta" para dentro da função, executa o seu código e quando termina continua na instrução seguinte à 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:
  - no exemplo anterior, `a`, `b` e `r` apenas são visíveis na função `main`;
  - os argumentos `x` e `y` apenas são visíveis na função `soma`.
- 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 – visibilidade das variáveis

```
public class visibilidade{
   static int a = 1; // variável global
   public static void main(String[] args){
      int x, y; // variáveis locais à função main
      ...
   }
   public static int f1(int y){// variável local à função f1
      int x; // variável local à função f1
      ...
      for (int i = 0 ; i < x ; i++){ // i é visível no ciclo for
         ...
      }
      ...
   }
}
```