Programação Modular (Subprogramas)

A utilização de subprogramas

- Os subprogramas permitem a divisão de um programa em módulos, cada um dos quais pode ser desenvolvido separada e independentemente dos outros, podendo estes módulos, posteriormente, serem integrados num único programa.
- As linguagens de alto nível fornecem dois processos distintos de produzir módulos, dando um deles origem à criação de **funções** e o outro à criação de **procedimentos**.
- Um subprograma para ser executado tem de ser integrado num programa.

Subprogramas (Módulos ou Rotinas)

- Um subprograma
  - é um grupo de acções identificado por um nome.
  - pode executar as mesmas acções que um programa, receber valores, manipular esses valores, e produzir novos valores.
  - recebe informação do programa a que pertence e retorna a informação produzida a esse programa.
  - introduz uma nova estrutura de controlo: a **chamada** ou **invocação** de um subprograma.
  - quando é chamado (invocado), as acções do programa deixam temporariamente de ser executadas e o computador começa a executar as acções do subprograma.
  - quando termina a sua execução, o computador recomeça a execução das acções do programa, a partir da instrução que "chamou" o subprograma.

Vantagens da utilização de subprogramas

- programas mais fáceis de escrever
- programas mais fáceis de ler
- programas, em geral, mais curtos
- programas mais fáceis de modificar
- abstracção
- reutilização

Subprogramas

- Cada subprograma
  - é identificado por um nome
  - pode necessitar de parâmetros (**parâmetros formais**)
- Os parâmetros formais dão forma ao subprograma e permitem a concretização dos valores, feita apenas no momento da chamada
- Depois da definição de um subprograma, as instruções que constituem o corpo desse subprograma são executadas quando o seu nome, seguido pelo número apropriado de argumentos (chamados **parâmetros reais ou parâmetros concretos**) é encontrado.
- A sintaxe da chamada é
  nomeDaClasse.nomeDoMétodo(parâmetrosReais)
  - Se a chamada for na própria classe podemos omitir nomeDaClasse.

Subprogramas

- A execução de uma instrução de chamada de um subprograma resulta numa associação dos parâmetros concretos com os parâmetros formais e na execução do grupo de instruções que corresponde ao corpo (bloco) do subprograma. Quando se atinge o fim do grupo de instruções que constitui o corpo, o subprograma deixa de ser executado e o computador passa a executar a instrução imediatamente a seguir à instrução que chamou o subprograma.
- A associação entre os parâmetros concretos e os parâmetros formais é feita com base na posição que os parâmetros ocupam na lista dos parâmetros (o n-ésimo parâmetro concreto é associado com o n-ésimo parâmetro formal).

Subprogramas

- Existem dois tipos de subprogramas:
  - os que retornam um valor (funções)
  - os que executam acções (procedimentos)
- O tipo que um subprograma retorna pode ser:
  - um tipo de dados primitivo (e.g. `int`, `char`, `double`); ou
  - um tipo classe (e.g. `String`)

O programa principal **main**

- Um programa para resolver um problema é escrito como uma classe com um método (subprograma) main
- A invocação do nome da classe provoca a invocação do método main
- Note-se a estrutura básica:

```
public class NomeDaClasse
{
   public static void main(String[] args)
   {
      ...
   }
}
```

Subprogramas que Não Retornam um Valor (procedimentos)

- Definição:

```
modificadores void nomeDoSubprograma(parâmetrosFormais)
  {
      declaração de variáveis
      instruções
  }
```

- Exemplo:

```
public void escreveSaidas()
{
  System.out.println("Nome = " + nome);
  System.out.println("Nota final = " + nota);
}
```

- Exemplo da invocação de um subprograma **void na própria classe**

```
escreveSaidas();
```


Subprogramas que Retornam um Valor (funções)

- Definição:

```
modificadores tipoDeRetorno nomeDoSubprograma(parâmetrosFormais)
      {
          declaração de variáveis
          instruções
          return expressãoCalculada;
      }
```

- Exemplo:

```
public int media(n1,n2,n3,n4)
    {      double aux;
           aux=(n1+n2+n3+n4)/4.0;
           return (int)aux;
    }
```

- Exemplos da invocação de um subprograma que retorna um valor

```
notaFinal = media(nota1,nota2,nota3,nota4);
calculo = media(a1,a2,a3,a4)+ media(b1,b2,b3,b4);
System.out.println("A média é: "+ media(a1,a2,a3,a4));
numero = Math.random();
```

(Repare que a separação dos parâmetros é feita com virgula)

Passagem de Parâmetros num Subprograma

Passagem por Valor

- Quando um parâmetro é passado **por valor**, o valor do parâmetro concreto é avaliado e esse valor é atribuído ao parâmetro formal correspondente. A única ligação entre os parâmetros concretos e os parâmetros formais é uma associação unidireccional de valores

- Em Java a passagem de parâmetros faz-se **por valor**

  (nota: tipos de dados primitivos vs. tipos classe)

Passagem de Parâmetros num Subprograma

Passagem por Valor

- Quando o parâmetro real é uma constante a passagem por valor não acarreta consequências

- Vejamos um exemplo em que o parâmetro real é uma variável:

```
public class Exemplo
{
      public static void main(String[] args)
      {
              int x = 10;
              muda(x);
              System.out.println(x);
        }
      public static void muda(int valor)
        {
              valor = valor+5;
              System.out.println(valor);
        }
}
```

Passagem de Parâmetros num Subprograma

Passagem por Referência

- Quando um parâmetro é passado **por referência**, o que é fornecido ao parâmetro formal correspondente não é o valor do parâmetro concreto, mas a sua localização na memória do computador (endereço de memória). Utilizando a chamada por referência, os parâmetros formais e os parâmetros concretos vão **partilhar** o mesmo espaço de memória e, consequentemente, qualquer modificação feita aos parâmetros formais reflecte-se nos parâmetros concretos
- Em Java a **passagem por Referência** é simulada quando os parâmetros formais são do **tipo Classe** (e.g. String)

Variáveis locais e Instruções Compostas

- Uma instrução composta (bloco) é um conjunto de instruções entre um par de chavetas { }
- Uma variável declarada dentro de um bloco é conhecida apenas dentro do bloco
  - é local ao bloco e por esse facto é chamada de *variável local*
  - quando o bloco termina a sua execução, a variável local "desaparece"
  - referências à variável fora do bloco causam um erro de compilação

Variáveis locais e Instruções Compostas

- As instruções têm de estar sempre dentro de um subprograma
- A declaração de variáveis pode estar dentro ou fora de um subprograma
- Um subprograma não pode utilizar variáveis declaradas em outros subprogramas
- Variáveis locais declaradas em subprogramas diferentes são variáveis diferentes, mesmo tendo o mesmo nome
- Para além das variáveis locais podemos ter variáveis que estão declaradas fora de qualquer subprograma. São **variáveis globais** que são visíveis a partir de qualquer subprograma.