Linguagem Java

Arrays bidimensionais

Programação Orientada a Objetos em Java

Classe Object

Membros estáticos

Exercício

- Resolução do exercício 7 da ficha
  - *Arrays* bidimensionais
  - Encapsulamento
  - Métodos estáticos
  - Introdução à classe `Object`

7. Escreva uma aplicação que some matrizes retangulares. Uma matriz deverá ser representada através de uma classe adequada. A soma deverá ser realizada através de duas formas distintas:

a. Função membro que acumulará aos valores de uma matriz os valores de outra matriz.

b. Função estática que recebe duas matrizes e retornará uma nova matriz com o resultado da soma, ou seja, não se pretende que sejam alteradas as matrizes originais.

Métodos utilitários

- Existem várias classes e bibliotecas que disponibilizam métodos úteis em situações diversas

- No caso de operações sobre *arrays* podem-se salientar:
  - `System.arraycopy(src,src_pos,dst,dst_pos,length)`
    - Permite copiar elementos de um *array* para outro, criado previamente
  - A classe `java.utils.Arrays`
    - Disponibiliza um conjunto de métodos para trabalhar com *arrays*
      - `copyOf, copyOfRange`
      - `fill`
      - `compare, equals, mismatch`
      - `binarySearch, sort, stream, toString`

Programação orientada a objetos

- Sendo a linguagem Java totalmente orientada a objetos, inclui todos os princípios associados a este tipo de programação, nomeadamente:
  - Abstração e Encapsulamento
    - Esconder detalhes de implementação
    - Controlar acesso à informação
    - Generalizar utilização
  - Herança
    - Definição de um novo tipo de objeto como especialização de outro tipo de objeto

      ```
      class <nova_classe> extends <classe_base> { ... }
      ```
    - São herdadas as características do objeto base
    - Podem ser definidas novas características e comportamentos adequados à especialização em causa
  - Polimorfismo
    - Redefinição (`@Override`) dos comportamentos declarados e/ou definidos na classe base, permitindo a sua execução de forma genérica

Hierarquia única

- Em Java todas as classes derivam direta ou indiretamente da classe `Object`
  - Mesmo quando não é explicitada essa dependência, ela é assumida de forma implícita

- A classe `Object` disponibiliza um conjunto de funcionalidades (métodos) onde se incluem
  - **`toString`**
  - `clone`
  - `equals`
  - `hashCode`
  - `getClass`

this

- No contexto de qualquer instância de um determinado tipo de objeto, poder-se-á ter acesso à sua referência (autorreferência) através da palavra-chave `this`

```
class Point {
   int x, y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   ...
}
```

Membros estáticos

- Na definição de uma classe os diversos membros, variáveis ou funções, podem ser definidos como sendo estáticos
  - Utilização da etiqueta `static`
- Os membros estáticos...
  - Podem ser acedidos/usados sem que existam instâncias desse tipo de objeto
  - Os valores das variáveis são acedidos e pode ser alvo de alteração, sendo partilhados pelas eventuais instâncias existentes desse tipo de objeto
    - As alterações realizadas a variáveis estáticas no contexto de uma instância são visíveis para todas as outras instâncias

String

- A classe `String` representa uma cadeia de carateres **imutável**
  - A alteração de uma `String` (por exemplo, concatenando através do operador '+') origina sempre a criação de uma nova `String`

- Disponibiliza métodos utilitários para trabalhar com os caracteres presentes na `String`
  - **`equals`**, `equalsIgnoreCase, matches, compareTo, compareToIgnoreCase, startsWith, contains, endsWith, indexOf`
  - `isBlank, isEmpty`
  - `concat, replace, replaceAll, repeat, trim`
  - `split`
  - `toUpperCase, toLowerCase`
  - `...`

String

- Existem alguns cuidados a ter em atenção quando se usam instâncias de objectos *String*, por exemplo:

```
String s1 = "DEIS-ISEC";
String s2 = new String("DEIS-ISEC");
String s3 = s2;
String s4 = "Deis-Isec";

System.out.println(s1 == s2); // false
System.out.println(s2 == s3); // true
s3 = s1;
System.out.println(s2 == s3); // false
System.out.println(s1.equals(s2)); // true
System.out.println(s2.equals(s4)); // false
System.out.println(s2.equalsIgnoreCase(s4)); // true
```

StringBuffer e StringBuilder

- As classes `StringBuffer` e `StringBuilder` permitem gerir *strings* **mutáveis**
- Ambas as classes disponibilizam métodos diversos para trabalhar com *strings*, incluindo métodos para adicionar novos caracteres ou *strings*, modificar caracteres pontuais ou *substrings*, ...
- Diferenças
  - `StringBuffer` – *thread-safe*
  - `StringBuilder` – mais rápida

Exercício

- Resolução do exercício 9 da ficha
  - *Arrays* bidimensionais com número variável de elementos

9. Escreva uma aplicação que calcule e imprima o triângulo de Pascal. O triângulo de Pascal deverá ser representado através de uma classe própria, que deverá incluir funções para: gerar triângulo com uma determinada profundidade, gerar uma *String* representativa do triângulo (`toString()`) e mostrar o triângulo na consola. Quando o objeto for criado poderá ser indicado através do *construtor* a profundidade pretendida.