PROGRAMAÇÃO ORIENTADA AOS OBJECTOS
(em JAVA)

LEI - LCC
2º ANO/2º SEMESTRE – 2010/2011

Teste Teórico – 21 de Junho de 2011
Cotação: 20 valores Duração: 120 m

Nota 1: Responda a cada parte em folhas separadas.
Nota 2: Cada questão tem a respectiva cotação indicada no fim da mesma.
Nota 3: Pode sempre usar métodos anteriormente definidos (mas correctos).

O SISTEMA GEO

A tecnologia actual permite que qualquer pessoa se possa localizar no planeta com uma precisão nunca imaginada por navegantes e aventureiros até há bem pouco tempo. O sofisticado sistema que tornou tal possível designa-se GPS, de *Global Positioning System*, e foi concebido pelo Departamento de Defesa dos EUA no início da década de l960, tendo por base informações obtidas por satélites.
Aqui, consideraremos um sistema que funciona por unidades lineares reais (km, e não angulares, como graus ou grados), e que localizam um ponto em função da sua distância real relativamente aos 3 eixos fundamentais de referência – equador (latitude 0), o meridiano de Greenwich (longitude 0) e o nível do mar (altitude 0). Será o nosso referencial para as questões que se seguem.
Chamaremos a este sistema **GEO**, e a estes pontos, com as 3 coordenadas reais, **ponto GEO**.

PARTE I (12,5 valores)

Relembre a classe **Ponto2D** estudada e utilizada nas aulas. Considere a classe **PontoGeo**, que representa, de forma simplificada, as coordenadas GEO de um dado ponto, designadamente, latitude, longitude e altura, classe parcialmente definida como:

```
public class PontoGeo extends Ponto2D implements Serializable {
    // x e y são as variáveis de instância de Ponto2D; são a latitude e a longitude.
    private double alt;
     ..............
    // métodos equals() e clone() estão implementados e podem ser usados, bem como os herdados
}
```

1) Escreva o código do construtor de cópia e do método **`toString()`** desta classe (1.5);
2) Escreva o código do método **`double getLongitude()`** que devolve a longitude do **PontoGeo** (1);
3) Escreva o código do método **`double distancia(PontoGeo p)`** que determina a distância euclidiana (no plano X-Y, 2D, cf. teorema de Pitágoras) entre o receptor e o parâmetro (1.5).

Considere agora a classe **InfoGeo** que representa o conjunto de informação que se pretende associar a um dado ponto GEO, designadamente, o continente, país e cidade, e ainda uma tabela que associa a uma dada *string* identificativa de um *tipo de serviço* (cf. "farmácia", "hospital", "restaurante", "hotel", etc.) um conjunto contendo as localizações (*nome* e *endereço* do tipo String guardadas em instâncias da classe **LocalServ**) de serviços desse tipo.

```
public class InfoGeo implements Serializable {
   // variáveis de instância
   private String cont, pais, cidade;
   private TreeMap<String, TreeSet<LocalServ>> infos;
   ……….
   // considere que os construtores e os métodos getX() "deep" e setX(..) estão definidos. Use-os.
}
```

Chegamos assim à classe principal que se pretende implementar, a classe **MapaGeo**. A classe **MapaGeo** é representada por um **TreeMap** que associa a um ***ponto GEO*** uma ***ficha de informação sobre esse ponto***, cf.:

```
public class MapaGeo implements Serializable {
   private TreeMap<PontoGeo, InfoGeo> mapa;

   // considere que construtores e métodos setX(..) estão disponíveis
}
```

A classe **MapaGeo** deverá ser agora completada desenvolvendo o seguinte código JAVA:

4) Método **`int numPontosEntreAlturas(double alt1, double alt2)`** que determina o número total de pontos do mapa que se situam entre 2 alturas dadas como parâmetro, sendo a primeira garantidamente <= à segunda (1.0);
5) Método **`getMapa()`** que devolve uma "deep copy" da variável de instância *mapa* (1.5);
6) Método que determina o conjunto completo de países referenciados pelos pontos GEO do mapa (1.5);
7) Método **`Set<String> tiposDeServico()`** que devolve o conjunto dos tipos de serviço ("farmácia, "hospital", etc.) que existem no mapa (1.5);
8) Método **`String paisComMaiorAltitude()`** que devolva o nome do país de maior altitude (1.5);
9) Método que devolve uma tabela onde a cada país do mapa se associa o conjunto das respectivas cidades que estão representadas em pontos GEO (1.5);

PARTE II (7,5 valores)

10) Crie uma classe que implemente a interface **`Comparator<PontoGeo>`** (1.5);
11) Existem vários tipos de *coordenadas GEO*, designadamente militares, civis e ainda diferenciais. As coordenadas GEO militares e civis têm a si associado adicionalmente um *tempo de precisão* que é a data mais actual da sua obtenção ou leitura. As diferenciais são relativas a um ponto GEO fixo, pelo que cada ponto diferencial apenas guarda as suas diferenças para este ponto referencial, pré-definido e que é igual para todas as instâncias da classe. Tomando em consideração a classe PontoGeo apresentada, defina agora as classes PontoGeoMilitar e PontoGeoDiferencial, mas apresentando apenas a sua estrutura (variáveis) e os dois construtores não vazios. (3.0)
12) Considerando apenas a classe **MapaGeo** escreva os seguintes métodos de instância:

    a) Método que recebe por parâmetro um conjunto de PontoGeo e um nome de um ficheiro, e grava numa `ObjectStream` cada um desses PontoGeo e a sua respectiva InfoGeo tal como estão representados em **`mapa;`** (1)
    b) Método que devolve um `TreeSet<PontoGeo>` contendo todos os PontoGeo lidos de uma `ObjectStream` com a estrutura da criada pelo método anterior. (2)

Prof. F. Mário Martins (Responsável de POO – LEI)
Prof. António Nestor (Responsável de POO – LCC)