PROGRAMAÇÃO ORIENTADA AOS OBJECTOS
(em JAVA5/6)

LEI - LCC
2º ANO/2º SEMESTRE – 2007/2008

Teste Teórico – 20 de Junho de 2008
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* (Sistema de Posicionamento Global), 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 (4 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 = latitude, y = longitude são vars de instância de Ponto2D
    private double alt;
     ..............
    // métodos toString() e clone() estão implementados e podem ser usados, bem como os herdados
}
```

1) Escreva o código do construtor de cópia e do método `equals(Object o)` desta classe (1.5);
2) Escreva o código do método `double getLatitude()` que devolve a latitude de um **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).

PARTE II (9.5 valores)

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 cf. classe **LocalServ**) de serviços desse tipo.

```
public class InfoGeo implements Serializable {
    // 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 um ***ponto GEO*** a 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 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 de países referenciados pelos pontos GEO do mapa (1.0);
7) Método `boolean` que verifique se existe algum ponto de altura superior a uma altura dada (1.5);
8) Método que devolva o nome do país de maior altitude (2.0);
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 (2.5);

PARTE III (6.5 valores)

10) Crie uma classe que implemente a interface **Comparator<PontoGeo>**, considerando, para simplificação, que a classe **Ponto2D** implementa já a interface **Comparator<Ponto2D>** (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 igual para todas as instâncias da classe. Tomando em consideração a classe PontoGeo apresentada, defina agora as classes PontoGeoMilitar e PontoGeoDiferencial, usando reutilização, e apresentando a sua estrutura e os construtores completos. (3.0)
12) Admitindo que o método `toString()` de **PontoGeo** gera para cada ponto uma "string" da forma "23.45/34.57/-0.87\n", ou seja, uma "string" terminada por "newline", e sabendo que uma `PrintWriter` permite escrever objectos em ficheiro de texto usando os respectivos métodos `toString()`, escreva o código de um método `TreeSet<PontoGeo> copiaPorFich(String fich)` que realize o seguinte (2.0):

    a) O método escreve no ficheiro de nome `fich` cada uma das chaves do **TreeMap** ***mapa***;
    b) O método lê em seguida do ficheiro `fich` todos os pontos para um `TreeSet<PontoGeo>` que devolve como sendo o seu resultado.

Prof. F. Mário Martins

CORRECÇÃO COMPLETA

(A verde o código dos métodos e classes pedidos no teste)

PARTE I

```
/**
 * Pontos descritos como duas coordenadas reais.
 *
 * @author F. Mário Martins
 * @version 2005
 */
import static java.lang.Math.abs;
import java.util.Comparator;
import java.io.Serializable;
public class Ponto2D implements Comparator<Ponto2D>, Serializable {

  // Variáveis de Instância
   private double x, y;

   // Construtores usuais
   public Ponto2D(double cx, double cy) { x = cx; y = cy; }
   public Ponto2D(){ this(0, 0); }
   public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }

   // Métodos de Instância
   public double getX() { return x; }
   public double getY() { return y; }

   /** incremento das coordenadas */
   public void incCoord(double dx, double dy) {
     x += dx; y += dy;
   }
   /** decremento das coordenadas */
   public void decCoord(double dx, double dy) {
     x -= dx; y -= dy;
   }
   /** soma as coordenadas do ponto parametro ao ponto receptor */
   public void somaPonto(Ponto2D p) {
     x += p.getX(); y += p.getY();
   }
   /** soma os valores par?metro e devolve um novo ponto */
   public Ponto2D somaPonto(double dx, double dy) {
     return new Ponto2D(x += dx, y+= dy);
   }
   /** determina se um ponto é simétrico (dista do eixo dos XX o
   mesmo que do eixo dos YY */
   public boolean simetrico() {
     return abs(x) == abs(y);
   }
   /** verifica se ambas as coordenadas sao positivas */
   public boolean coordPos() { return x > 0 && y > 0; }

   // Métodos complementares usuais

   /** Verifica se os pontos sao iguais */
   public boolean igual(Ponto2D p) {
     if (p != null) return x == p.getX() && y == p.getY();
     else return false;
   }
```

```
  /** outra versao de igual(Ponto2D p) */
  public boolean igual1(Ponto2D p) {
    return (p == null) ? false : x == p.getX() && y == p.getY();
  }

  /** método equals() standard - recebe Object como parâmetro */
  public boolean equals(Object obj) {
     if(this == obj) return true;  // ? o pr?prio
     if((obj == null) || (this.getClass() != obj.getClass())) return false;
     Ponto2D p = (Ponto2D) obj;
     return x == p.getX() && y == p.getY();
  }

  /** Converte para uma representação textual  */
  public String toString() { return "Pt2D = " + x + ", " + y + "\n"; }

  /** Cria e devolve uma copia "deep" do ponto receptor */
  public Ponto2D clone() { return new Ponto2D(this); }

  // implementação de Comparator<Ponto2D>
  public int compare(Ponto2D p1, Ponto2D p2) {
    if( p1.getX() < p2.getX() ) return -1;
    if( p1.getX() > p2.getX() ) return 1;
    if( p1.getY() < p2.getY() ) return -1;
    if( p1.getY() > p2.getY() ) return 1; else return 0;
  }
}


/**
 * PontoGeo: Um ponto com latitude, longitude e altura, a partir de um Ponto2D
 *
 * @author Mário Martins
 * @version 1/2008
 */
public class PontoGeo extends Ponto2D {

   // variáveis de instância
   // x de Ponto2D é latitude
   // y de Ponto2D é longitude

   private double alt;

   // Construtores
   public PontoGeo() { super(0.0, 0.0); alt = 0.0; }
   public PontoGeo(double lat, double longi, double alt) {
      super(lat, longi); this.alt = alt;
   }
   public PontoGeo(PontoGeo pg) {
      super(pg.getX(), pg.getY()); alt = pg.getAlt();
   }

   // métodos de instância
   public double getAlt() { return alt; }
   public double getLatitude() { return this.getX(); } // ou super.getX()

   public boolean equals(Object o) {
         if (this == o) return true;
         if ( (o == null) || this.getClass() != o.getClass() ) return false;
         PontoGeo pg = (PontoGeo) o;
         return super.equals(o) && this.getAlt() == pg.getAlt();
   }
```

```
    // calcula a distância entre dois pontos
    public double distancia(PontoGeo pg) {
            double deltaX = this.getX() - pg.getX();   // ou getLatitude()
            double deltaY = this.getY() - pg.getY();
            return Math.sqrt(Math.pow(deltaX,2) + Math.pow(deltaY,2));
    }

    public PontoGeo clone() { return new PontoGeo(this); }

    public String toString() {
      return "PontoGeo: (" + this.getX() + "," + this.getY() + "," + this.getAlt() + ")\n";
    }
}
```

PARTE II

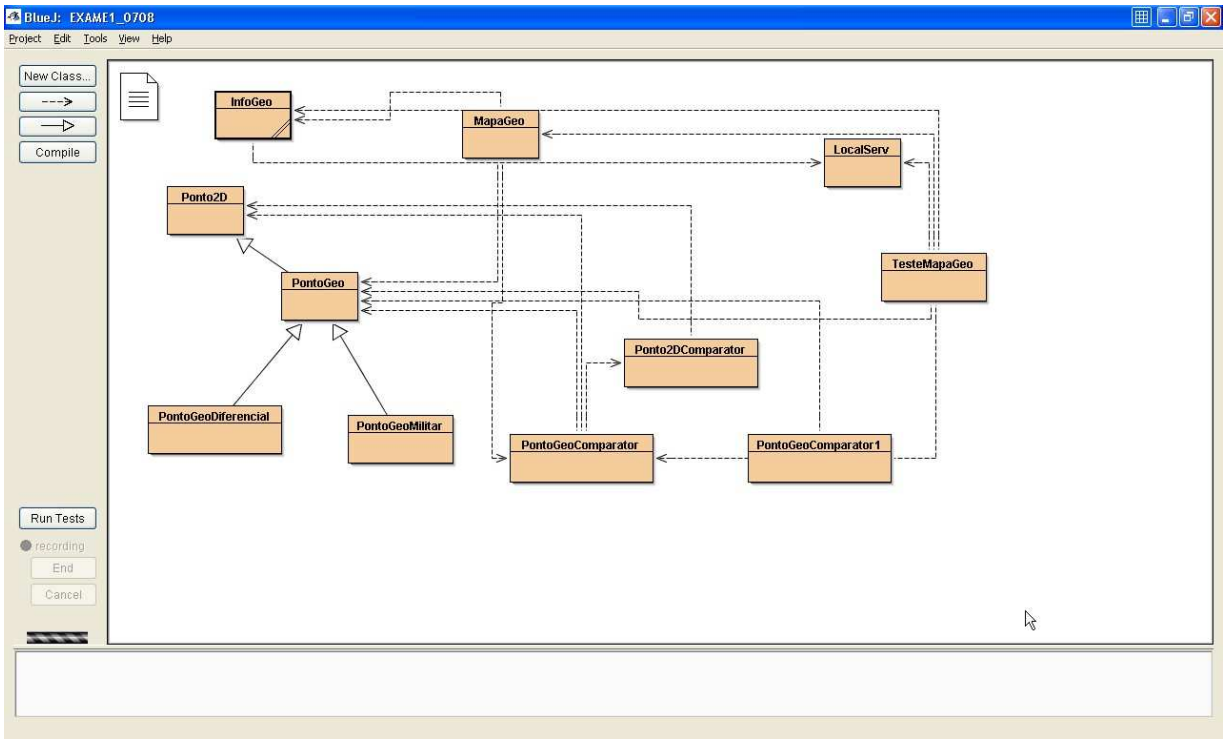
```
/**
 * InfoGeo: todas as informações a associar a um PontoGeo
 *
 * @author Mário Martins
 * @version 1/2008
 */
import java.io.Serializable;
import java.util.*;
public class InfoGeo implements Serializable {

    // variáveis de instância
    private String cont, pais, cidade;
    private TreeMap<String, TreeSet<LocalServ>> infos =
        new TreeMap<String, TreeSet<LocalServ>>();

    // Construtores
    public InfoGeo(String ct, String ps, String cid) {
       cont = ct; pais = ps; cidade = cid;
    }

    public InfoGeo(String ct, String ps, String cid,
                   TreeMap<String, TreeSet<LocalServ>> infpar) {
       cont = ct; pais = ps; cidade = cid;
       TreeSet<LocalServ> copia = new TreeSet<LocalServ>(new LocalServ());
       for(String serv : infpar.keySet()) {
          for(LocalServ nomeMora : infpar.get(serv)) copia.add(nomeMora.clone());
          infos.put(serv, copia); copia.clear();
       }
    }

    // métodos de instância
    public String getPais() { return pais; }
    public String getContinente() { return cont; }
    public String getCidade() { return cidade; }

    // devolve uma cópia do TreeMap
    public TreeMap<String, TreeSet<LocalServ>> getInfos() {
       TreeMap<String, TreeSet<LocalServ>> infaux =
               new TreeMap<String, TreeSet<LocalServ>>();
       TreeSet<LocalServ> copia = new TreeSet<LocalServ>(new LocalServ());
       for(String serv : infos.keySet()) {
          infaux.put(serv, new TreeSet<LocalServ>(new LocalServ()));
          for(LocalServ nomeMora : infos.get(serv)) copia.add(nomeMora);
          infaux.put(serv, copia); copia.clear();
       }
       return infaux;
    }
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if ( (o == null) || this.getClass() != o.getClass() ) return false;
        InfoGeo ig = (InfoGeo) o;
        return cont.equals(ig.getContinente()) && pais.equals(ig.getPais()) &&
                cidade.equals(ig.getCidade());
    }

    public InfoGeo clone() {
       return new InfoGeo(cont, pais, cidade, this.getInfos());
    }

}


/**
 * LocalServ: ficha de localização de um serviço
 *
 * @author Mário Martins
 * @version 1/2008
 */
import java.util.Comparator;
public class LocalServ implements Comparator<LocalServ> {
      // variáveis de instância
      private String nome;
      private String morada;

      // construtores
      public LocalServ() { }
      public LocalServ(String nom, String mora) {
          nome = nom; morada = mora;
      }

      public LocalServ(LocalServ ls) {
          nome = ls.getNome(); morada = ls.getMorada();
      }

      // métodos de instância
      public String getNome() { return nome; }
      public String getMorada() { return morada; }
      public String toString() {
         return "Nome: " + nome + " Morada: " + morada + " \n";
      }
      public LocalServ clone() { return new LocalServ(this); }

      public boolean equals(Object obj) {
         if(this == obj) return true;
         if((obj == null) || (this.getClass() != obj.getClass())) return false;
         LocalServ ls = (LocalServ) obj;
         return nome.equals(ls.getNome()) && morada.equals(ls.getMorada());
      }

      // implementação de compare para usar LocalServ em TreeSet<LocalServ>
      public int compare(LocalServ ls1, LocalServ ls2) {
        return ls1.getNome().compareTo(ls2.getNome());
      }

}
```

```
/**
 * Ponto2DComparator: Um comparador de Ponto2D
 *
 * @author F. Mário Martins
 * @version 1/2006
 */
import java.util.Comparator;
public class Ponto2DComparator implements Comparator<Ponto2D> {
  public int compare(Ponto2D p1, Ponto2D p2) {
    if( p1.getX() < p2.getX() ) return -1;
    if( p1.getX() > p2.getX() ) return 1;
    if( p1.getY() < p2.getY() ) return -1;
    if( p1.getY() > p2.getY() ) return 1; else return 0;
  }
}
```

```
/**
 * PontoComparator: Um comparador de PontoGeo
 *
 * @author F. Mário Martins
 * @version 1/2008
 */
import java.util.Comparator;
public class PontoGeoComparator implements Comparator<PontoGeo> {
  public int compare(PontoGeo pg1, PontoGeo pg2) {
    Comparator<Ponto2D> comp2D = new Ponto2DComparator();
    int resultado = comp2D.compare( ((Ponto2D) pg1), ((Ponto2D) pg2));
    if(resultado == 0) { // são iguais em latitude e longitude
      if(pg1.getAlt() == pg2.getAlt()) return 0;
      else if(pg1.getAlt() < pg2.getAlt()) resultado = -1;
          else resultado = 1;
    }
    return resultado;
  }
}
```

```
/**
 * PontoGeoComparator1 alternativa a PontoGeoComparator sem usar
 * Ponto2DComparator.
 *
 * @author F. Mário Martins
 * @version 1/2008
 */
import java.util.Comparator;
public class PontoGeoComparator1 implements Comparator<PontoGeo> {
  public int compare(PontoGeo pg1, PontoGeo pg2) {
    if( pg1.getX() < pg2.getX() ) return -1;
    if( pg1.getX() > pg2.getX() ) return 1;
    if( pg1.getY() < pg2.getY() ) return -1;
    if( pg1.getY() > pg2.getY() ) return 1;
    else return comparaAlt(pg1, pg2);
  }

  private int comparaAlt(PontoGeo pg1, PontoGeo pg2) {
    if(pg1.getAlt() < pg2.getAlt()) return -1;
    if(pg1.getAlt() > pg2.getAlt()) return 1; else return 0;
  }
}
```



CLASSE MapaGeo

```
/**
 * MapaGeo: uma implementação de um sistema de GPS especial.
 *
 * @author Mário Martins
 * @version 1/2008
 */
import java.util.*;
import java.io.*;
public class MapaGeo implements Serializable {
   // Variáveis de Instância
   private TreeMap<PontoGeo, InfoGeo> mapa =
       new TreeMap<PontoGeo, InfoGeo>(new PontoGeoComparator());

   // Construtores
   public MapaGeo() { }

   public MapaGeo(MapaGeo mg) {
     TreeMap<PontoGeo, InfoGeo> aux = mg.getMapa();
     for(PontoGeo pga : aux.keySet())
       mapa.put(pga.clone(), aux.get(pga).clone());
   }

   // Métodos de Instância

   public int numPontosGeo() { return mapa.keySet().size(); }

   public int numPontosEntreAlt(double alt1, double alt2) { // alt1 <= alt2
      int conta = 0;
      for(PontoGeo pg : mapa.keySet())
         if(pg.getAlt() >= alt1 && pg.getAlt() <= alt2) conta++;
      return conta;
   }
```

```
public InfoGeo consultaInfo(PontoGeo pg) { return mapa.get(pg).clone(); }

public TreeMap<PontoGeo, InfoGeo> getMapa() {
   TreeMap<PontoGeo, InfoGeo> aux =
      new TreeMap<PontoGeo, InfoGeo>(new PontoGeoComparator());
   for(PontoGeo pg : mapa.keySet())
      aux.put(pg.clone(), mapa.get(pg).clone());
   return aux;
}

public String pontosGeo() {
   StringBuilder sb = new StringBuilder();
   for(PontoGeo pg : mapa.keySet()) sb.append(pg.toString());
   return sb.toString();
}

public void inserePontoGeo(PontoGeo pg, InfoGeo ig) {
   mapa.put(pg.clone(), ig.clone());
}

public boolean existePontoGeo(PontoGeo pg) {
   return mapa.containsKey(pg);
}

public void removePontoGeo(PontoGeo pg) {
   mapa.remove(pg);
}

public TreeSet<String> paisesNoMapa() {
   TreeSet<String> paises = new TreeSet<String>();
   for(InfoGeo infg : mapa.values()) paises.add(infg.getPais());
   return paises;
}

public boolean haMaisAltoQue(double altRef) {
   boolean achei = false;
   Iterator<PontoGeo> it = mapa.keySet().iterator();
   while(it.hasNext() && !achei) {
       if (it.next().getAlt() > altRef) achei = true;
   }
   return achei;
}

public String paisMaisAlto() {
   double maxAltura = Double.MIN_VALUE;
   String pais = "";
   for(PontoGeo pg : mapa.keySet())
      if(pg.getAlt() > maxAltura) {
         maxAltura = pg.getAlt();
         pais = mapa.get(pg).getPais();
      }
   return pais;
}

public TreeMap<String, TreeSet<String>> tabelaPaisCidades() {
   TreeMap<String, TreeSet<String>> tabela =
            new TreeMap<String, TreeSet<String>>();
   TreeSet<String> paises = new TreeSet<String>(this.paisesNoMapa());
   // ou ainda TreeSet<String> paises = new TreeSet<String>();
   // ou paises.addAll(this.paisesNoMapa());   // são strings !!

   // Solução 1: Mais simples
   // Para cada país, percorrer cada uma das InfoGeo e verificar a que país
   // diz respeito. Se for o que está neste momento em causa, se for novo,
   // temos primeiro que inicializar o conjunto das suas cidades
   // na tabela e, só depois, juntar a cidade registada nesta InfoGeo.
```

```
    // Se não for novo, temos apenas que adicionar a respectiva cidade associada.
    // Se não é uma ficha de tal país, não se faz nada !!
    // E de forma igual para todos os outros. No final estará certo !!
    for(String pais : paises)   // para cada país do mapa fazer …
      for(InfoGeo infg : mapa.values())
        if(infg.getPais().equals(pais)) {
          if(tabela.get(pais) == null)
              tabela.put(pais, new TreeSet<String>());   // inicializa
          tabela.get(pais).add(infg.getCidade());          // adiciona a cidade
        }
    return tabela;
}

// Solução 2: Mais difícil mas, aparentemente, mais "esperta", porém …
// Em vez de não se fazer nada, inicializaríamos logo o conjunto das suas
// cidades na tabela (Map) para que não esteja a null.
// Porém, como nunca sabemos o que se passou antes teríamos que a cada
// momento (iteração nas InfoGeo) perguntar "já foi inicializado ou não", e,
// ainda, distinguir o caso do primeiro país. Não serve portanto !!

// Solução 3: Menos óbvia, mas muito simples.
// Para cada país possível, inicializar a tabela e percorrer todas as
// InfoGeo, adicionando à tabela as de tal país.
public TreeMap<String, TreeSet<String>> tabelaPaisCidades1() {
    TreeMap<String, TreeSet<String>> tabela =
            new TreeMap<String, TreeSet<String>>();
    // Inicialização da tabela
    for(String pais : this.paisesNoMapa()) {
       tabela.put(pais, new TreeSet<String>());
       // Varrimento das fichas e criação da tabela
       for(InfoGeo infg : mapa.values())
         if(infg.getPais().equals(pais))                  // é do país fixado
             tabela.get(pais).add(infg.getCidade());  // adiciona a cidade
    }
    return tabela;
}
```

PARTE III

```
public TreeSet<PontoGeo> copiaPorFicheiro(String fich) throws IOException {
    // escreve todos os pontos no formato indicado, sem usar toString()
    TreeSet<PontoGeo> lidoDeFich =
            new TreeSet<PontoGeo>(new PontoGeoComparator());
    // escreve os pontos GEO num ficheiro de nome fich
    PrintWriter pw = new PrintWriter(fich);
    for(PontoGeo pg : mapa.keySet()) {
        pw.print(pg.getX() + "/");
        pw.print(pg.getY() + "/");
        pw.println(pg.getAlt() + "/");
    }
    pw.close(); pw.flush();
    // lê do ficheiro fich cada um dos pontos GEO, 1 por linha, e reconstrói o TreeSet
    PontoGeo p = null; String linha = null;
    Scanner scan = new Scanner(new FileReader(fich));
    // deve usar-se o separador de linhas default da JVM
    scan.useDelimiter(System.getProperty("line.separator"));
    // Leitura das linhas a partir do ficheiro e obtenção dos PontoGeo
    while(scan.hasNext()) {
      linha = scan.next();
      p = this.parseLinha(linha);
      lidoDeFich.add(p); p = null;
    }
    scan.close();
    return lidoDeFich;
 }
```

```
    // faz a "desmontagem" da linha por forma a devolver um PontoGeo
    private PontoGeo parseLinha(String linha) {
      double lat, longi, alt;
      Scanner lineScan = new Scanner(linha);
      lineScan.useDelimiter("/");
      lat = Double.valueOf(lineScan.next()).doubleValue();
      longi = Double.valueOf(lineScan.next()).doubleValue();
      alt = Double.valueOf(lineScan.next()).doubleValue();
      lineScan.close();
      return new PontoGeo(lat, longi, alt);
    }
}
```

```
/**
 * PontoComparator: Um comparador de PontoGeo
 *
 * @author F. Mário Martins
 * @version 1/2008
 */
import java.util.Comparator;
public class PontoGeoComparator implements Comparator<PontoGeo> {
  public int compare(PontoGeo pg1, PontoGeo pg2) {
    Comparator<Ponto2D> comp2D = new Ponto2DComparator();
    int resultado = comp2D.compare( ((Ponto2D) pg1), ((Ponto2D) pg2));
    if(resultado == 0) { // são iguais em latitude e longitude
      if(pg1.getAlt() == pg2.getAlt()) return 0;
      else if(pg1.getAlt() < pg2.getAlt()) resultado = -1;
          else resultado = 1;
    }
    return resultado;
  }
}
```

```
/**
 * PontoGeoMilitar: um ponto GEO com registo de tempo de leitura
 *
 * @author Mário Martins
 * @version 1/2008
 */
import java.util.GregorianCalendar;
public class PontoGeoMilitar extends PontoGeo {
      // variáveis de instância
      private GregorianCalendar tempo;

      // Construtores
      public PontoGeoMilitar(double lat, double longi, double alt, GregorianCalendar time) {
         super(lat, longi, alt);
         tempo = (GregorianCalendar) time.clone();
      }
}
```

```
/**
 * PontoGeoDiferencial:
 *
 * @author Mário Martins
 * @version 1/2008
 */
public class PontoGeoDiferencial extends PontoGeo {
      // variáveis de classe - o Ponto Referencial e seus métodos
      private static PontoGeo pontoRef = new PontoGeo();
      private static void definePontoRef(double lat, double longi, double alt) {
         pontoRef = new PontoGeo(lat, longi, alt);
      }
      private static PontoGeo getPontoRef() {
         return new PontoGeo(pontoRef.getX(), pontoRef.getY(), pontoRef.getAlt());
      }

      // Construtores
      public PontoGeoDiferencial(double lat, double longi, double alt) {
          super(lat, longi, alt);
      }

      public PontoGeoDiferencial(PontoGeoDiferencial pgd) {
          super(pgd.getX(), pgd.getY(), pgd.getAlt());
      }

}
```

CLASSE DE TESTE

```
/**
 * TesteMapaGeo: Classe de teste do projecto MapaGeo
 *
 * @author Mário Martins
 * @version 1/2008
 */
import java.util.*;
import java.io.*;
import static java.lang.System.out;
public class TesteMapaGeo {
   public static MapaGeo main() {

       MapaGeo mapaGeo = new MapaGeo();

       // pontos
       PontoGeo pg1 = new PontoGeo(10.0, 10.0, 300.0);
       PontoGeo pg2 = new PontoGeo(11.0, 10.0, 130.0);
       PontoGeo pg3 = new PontoGeo(-10.0, -30.0, 230.0);

       // infos
       TreeMap<String, TreeSet<LocalServ>> info1 =
                  new TreeMap<String, TreeSet<LocalServ>>();

       TreeSet<LocalServ> moradas = new TreeSet<LocalServ>(new LocalServ());
       moradas.add(new LocalServ("S. Marcos", "Rua de S. Lázaro"));
       moradas.add(new LocalServ("Universitário", "Gualtar"));

       info1.put("HOSPITAL", moradas);
       moradas.clear();
       moradas.add(new LocalServ("Peixoto", "Quinta da Capela"));
       moradas.add(new LocalServ("Pinto", "Sra. Branca"));
       moradas.add(new LocalServ("Central", "Avenida Central"));
       info1.put("FARMÁCIA", moradas);
       moradas.clear();
       moradas.add(new LocalServ("Sardinha Biba", "Carandá"));
       moradas.add(new LocalServ("Sabão Rosa", "Rua Nova"));
       moradas.add(new LocalServ("Insólito", "Avenida Central"));
```

```
        info1.put("BAR", moradas);

        mapaGeo.inserePontoGeo(pg1, new InfoGeo("Europa", "Portugal", "Braga", info1));

        info1.clear(); moradas.clear();

        moradas.add(new LocalServ("Sto. António", "Rua do Hospital"));
        moradas.add(new LocalServ("S. João", "Estrada do Freixo"));
        info1.put("HOSPITAL", moradas);
        moradas.clear();
        moradas.add(new LocalServ("Martins", "Amial"));
        moradas.add(new LocalServ("Pinto", "Cedofeita"));
        moradas.add(new LocalServ("S. João", "Avenida Aliados"));
        info1.put("FARMÁCIA", moradas);
        moradas.clear();
        moradas.add(new LocalServ("TWINS", "Jardim da Foz"));
        moradas.add(new LocalServ("TripleX", "Amial"));
        moradas.add(new LocalServ("FENIX", "Avenida Boavista"));
        info1.put("BAR", moradas);
        moradas.clear();
        moradas.add(new LocalServ("DRAGÃO", "Avenida das Antas"));
        moradas.add(new LocalServ("BOAVISTA", "Av. da Boavista"));
        info1.put("ESTÁDIO", moradas);

        mapaGeo.inserePontoGeo(pg2, new InfoGeo("Europa", "Portugal", "Porto", info1));

        info1.clear(); moradas.clear();

        moradas.add(new LocalServ("Buenos Aires", "Avenida de la Plata"));
        moradas.add(new LocalServ("Iberostar BA", "Avenida del Brasil"));
        moradas.add(new LocalServ("Rui Argentina", "Avenida Colombo"));
        info1.put("HOTEL", moradas); moradas.clear();

        moradas.add(new LocalServ("Espanha", "Avenida Amial"));
        moradas.add(new LocalServ("Brasil", "Avenida del Sol"));
        moradas.add(new LocalServ("Portugal", "Avenida de los Aliados"));
        info1.put("CONSULADO", moradas); moradas.clear();

        moradas.add(new LocalServ("ARGENTA", "Jardim da Foz"));
        moradas.add(new LocalServ("BUENAS NOCHES", "Avenida del ALmirante"));
        moradas.add(new LocalServ("COLOMBO", "Avenida de las Pampas"));
        info1.put("BAR", moradas);
        moradas.clear();
        moradas.add(new LocalServ("BOCA JUNIORS", "Avenida de la Libertad"));
        moradas.add(new LocalServ("RIVER PLATE", "Avenida Rio de la Plata"));
        info1.put("ESTÁDIO", moradas);

        mapaGeo.inserePontoGeo(pg3,
                        new InfoGeo("América do Sul", "Argentina", "Buenos Aires", info1));

        TreeSet<PontoGeo> pontos = new TreeSet<PontoGeo>(new PontoGeoComparator());
        try {
            pontos = mapaGeo.copiaPorFicheiro("mapaGeo5.txt");
        }
        catch(IOException e) { out.println("Erro na leitura de ficheiro !!"); }
        out.println("Pontos Lidos: " + pontos.size());
       // teste: imprimir pontos na Terminal Window
        out.println(mapaGeo.pontosGeo());
        return mapaGeo;
    }
}
```

EXECUÇÃO DE main()