PARADIGMAS DE PROGRAMAÇÃO IV – LESI
e
PROGRAMAÇÃO ORIENTADA AOS OBJECTOS - LCC
2º ANO/2º SEMESTRE – 2006/2007

EXAME da 2ª CHAMADA – 30 de Junho de 2007
Cotação - 20 valores *Duração - 2h00m*

(ATENÇÃO: RESPONDA A CADA PARTE EM FOLHAS SEPARADAS)

PARTE I (12 valores)

Numa dada exploração agrícola um sistema de informação regista todas as medições de temperatura e humidade realizadas nas várias estufas. Apresentam-se em seguida as definições das principais classes de JAVA que constituirão tal sistema de informação.
A classe **FichaReg** (ficha de registo) representa o tipo de informação que é obtida num dado momento a partir do sensor de uma estufa, designadamente: o código da estufa, a temperatura e a humidade. Não são apresentados os métodos de instância desta classe, mas admite-se a existência de métodos comuns.

```
public class FichaReg {
    private String estufa; private double temp; private double humidade;
     ............
    // considere que construtores, getX(), setX(..), toString() e clone() estão disponíveis
}
```

A classe **Tempo** é definida, estruturalmente, como:

```
public class Tempo {
    private int hora; private int minuto; private int seg;
    ..........
    // considere que construtores, getX(), setX(..), toString(), equal() e clone() estão disponíveis
}
```

Chegamos assim à classe principal que se pretende implementar, a classe **RegistoEstufas**. A classe **RegistoEstufas** é representada por um **TreeMap** que associa um ***tempo*** a cada ***ficha de registo***, não existindo nunca dois registos feitos ao mesmo tempo.

```
public class RegistoEstufas {
        private TreeMap<Tempo, FichaReg> registos;
        // Construtores
        public RegistoEstufas() { registos = new TreeMap<Tempo, FichaReg>(); }
        public RegistoEstufas(TreeMap<Tempo, FichaReg> regs) { .......... }
        public RegistoEstufas(RegistoEstufas reg) { ............... }
        // Métodos de instância
        ..................
}
```

As classes anteriores deverão ser agora completadas com um conjunto de métodos que permitam realizar as diversas operações que se pretendem ver disponíveis relativamente a cada uma delas.
Relativamente à classe **FichaReg**, implemente os seguintes métodos de instância:

1) Método *equals(Object o)* que verifica se dois registos são iguais;

São os seguintes os métodos (de instância) de **RegistoEstufas** que devem ser implementados:

2) Método que determina o número de registos efectuados numa dada hora e minuto;
3) Método que dada uma FichaReg a insira no *treemap* com a hora, minuto e segundo actuais;
4) Método que dá como resultado uma lista contendo todos os registos feitos a dada hora (ex. 11);
5) Método que devolva uma lista de todas as humidades registadas numa dada estufa;
6) Método que determine se alguma estufa ultrapassou a temperatura dada como parâmetro;
7) Método *toString()*;
8) Método que devolve uma cópia dos registos efectuados – ie. o código efectivo de **getRegistos()**.

PARTE II (8 valores)

Considere agora a classe **Estufa** que contém toda a informação comum a qualquer estufa agrícola, designadamente: código, área em m^2, e temperaturas mínima e máxima. Há porém 3 tipos de estufa em particular: de fruta, de legumes e de flores. Existe um valor comercial fixo para cada tipo de estufa que é definido em Euros/m^2. As estufas de flores possuem o nome da flor e lista de cores. As estufas de fruta uma lista dos nomes dos frutos. As de legumes o nome do legume e a altura média atingida.

A classe **GereEstufas** representa o sistema de informação global de gestão das estufas e possui o registo das medições das estufas num dado momento e uma tabela onde a cada código de estufa se associa a sua informação particular, cf. se representa na declaração seguinte:

```
public GereEstufas implements Serializable {
    private RegistoEstufas regEstufas;
    private TreeMap<String, Estufa> infoEstufas;
    ....
}
```

a) Apresente as declarações completas da estrutura das classes **Estufa** e **EstufaDeFlores**, do método **valorTotal()**, que determina o valor comercial total de cada estufa, e dos construtores de cópia (considere definidos todos os outros métodos usuais);
b) Defina um método que calcule a área total das estufas da exploração agrícola;
c) Considerando as temperaturas registadas nas estufas a uma hora dada como parâmetro, e sabendo que caso a temperatura não se encontre entre os limites mínimo e máximo a produção de uma estufa pode estar destruída, escreva um método **emRisco(int hora)** que devolva o conjunto dos códigos das estufas que estavam em risco a essa hora;
d) Sabendo que se uma estufa está em risco tal pode implicar um prejuízo equivalente ao seu valor, escreva um método **perdaActualFlores(int hora)** que determine o valor total de prejuízo em estufas de flores à hora dada (Nota: mesmo que não tenha implementado o método **valorTotal()** pode usá-lo);

Prof. F. Mário Martins

CORRECÇÃO – PARTE I

------ Métodos de FichaReg ----------

```
public boolean equals(Object obj) {
   if (this == obj) return true;
   if ((obj == null) || (this.getClass() != obj.getClass()))
      return false;
   // É de certeza uma FichaReg
   FichaReg f = (FichaReg) obj;   // casting para tipo
   return estufa.equals(f.getEstufa()) && temp == f.getTemp() &&
          humidade == f.getHumid();
}
```

1.5

------ Métodos de RegistoEstufas ----------

```
public int numRegHM(int h, int m) {
   int total = 0;
   for(Tempo t : registos.keySet())
     if(t.getHora() == h && t.getMin() == m) total++;
   return total;
}
```

1.0

```
public void insereReg(FichaReg f) {
   GregorianCalendar tempo = new GregorianCalendar();
   Tempo t = new Tempo(tempo.get(Calendar, HOUR_OF_DAY),
                       tempo.get(Calendar.MINUTE), tempo.get(Calendar.SECOND));
   registos.put(t, f.clone());
}
```

1.5

```
public HashSet<FichaReg> registosDaHora(int h) {
   HashSet<FichaReg> regs = new HashSet<FichaReg>();
   for(Tempo t : registos.keySet())
     if(t.getHora() == h) regs.add(registos.get(t).clone());
   return regs;
}
```

1.5

```
public HashSet<Double> humidEstufa(String codEst) {
   HashSet<Double> humids = new HashSet<FichaReg>();
   for(FichaReg f : registos.values())
     if(f.getEstufa().equals(codEst)) humids.add(f.getHumid());
   return humids;
}
```

1.5

```
public boolean maisQueT(double temp) {
   boolean encontrei = false;
   Iterator<FichaReg> itFichas = registos.values().iterator();
   while(itFichas.hasNext() && !encontrei) {
    if(itFichas.next().getTemp() > temp) encontrei = true
    }
   return encontrei;
}
```

2.0

```
public String toString() {
   StringBuilder sb = new StringBuilder("-----  REGISTOS DAS ESTUFAS ----\n");
   for(Tempo t : registos.keySet())
    sb.append("Hora: " + t.toString() + "\n" + registos.get(t).toString() + "\n");
   return sb.toString();
}
```

1.5

```
public TreeMap<Tempo, FichaReg> getRegistos() {
   TreeMap<Tempo, FichaReg> copia = new TreeMap<Tempo, FichaReg>();
   for(Tempo t : registos.keySet()) copia.put(t, registos.get(t).clone());
   return copia;
}
```

1.5

CORRECÇÃO – PARTE II

```
public abstract class Estufa {
        private String cod; private double area; private double tempMin, tempMax;
        //
        public Estufa(Estufa est) {
          cod = est.getCodigo(); area = est.getArea(); tempMin = est.getTempMin();
          tempMax = est.getTempMax();
        }
        // Métodos de instância
        public abstract double valorTotal();
        ..................
}
```

1.0

```
public class EstufaDeFlores {
        private static VALOR_AREA = 0.0;
        public static void setValorArea(double val) { VALOR_AREA = val; }
        private String nome;
        private ArrayList<String> cores = new ArrayList<String>();
        //
        public EstufaDeFlores(EstufaDeFlores estF) {
          super((Estufa) estF);
          nome = estF.getNome();
          cores = estf.getCores().clone();  // é um ArrayList<String> !!
        }
        // Métodos de instância
        public double valorTotal() { return this.getArea() * VALOR_AREA; }
        ..................
}
```

2.0

```
public double areaTotal() {
   double area = 0.0;
   for(Estufa est : infoEstufas.values())
       area += est.getArea();
   return total;
}
```

1.0

```
public TreeSet<String> emRisco(int hora) {
   TreeSet<String> codigos = new TreeSet<String>();
   TreeMap<Tempo, FichaReg> regs = regEstufas.getRegistos();
   TreeSet<FichaReg> regsDaHora = new TreeSet<FichaReg>();
   for(Tempo t : regs.keySet())
       if(t.getHora() == h) regsDaHora.add(regs.get(t));
    Estufa estf = null; double tempReg = 0.0;
    for(FichaReg f : regsDaHora) {
       tempReg = f.getTemp(); estf = infoEstufas.get(f.getCodigo());
       if(estf.getTempMax() >= tempReg || estf.getTempMin() <= tempReg)
              codigos.add(estf.getCodigo());
    return codigos;
}
```

2.5

```
public double perdaActualFlores(int hora) {
   double prej = 0.0; Estufa estf = null;
   TreeSet<String> codigos = this.emRisco(hora);
   for(String codEst : codigos) {
    estf = infoEstufas.get(codEst);
   if(estf.getClass().getSimpleName().equals("EstufaDeFlores")
   // ou ainda if(estf instanceof EstufaDeFlores)
     prej += estf.valorTotal();
   return prej;
}
```

1.5