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

EXAME de 1ª CHAMADA – 15 de Junho de 2007

CORRECÇÃO

PARTE I:

```
public int aulasParaDar() {
   return previstas.size() - livroPres.size();
}
```

0.5

```
public double perctPres() {
   int numAulas = folhasPres.size();
   int freqTotal = numAulas * inscritos.size();
   int freqReal = 0;
   for(TreeSet<String> folha : livroPres.values())
      freqReal += folha.size();
   return ((double) freqReal) / ((double) freqTotal);
}
```

1.5

```
public TreeSet<GregorianCalendar> aulasFreqAluno(String num) {
   TreeSet<GregorianCalendar> datas = new TreeSet<GregorianCalendar>();
   for(GregorianCalendar data : livroPres.keySet())
     if(livroPres.get(data).contains(num))
        datas.add((GregorianCalendar) data.clone());
   return datas;
}
```

1.5

```
public boolean nuncaFoi(String num) {
   boolean naoFoi = true;
   Iterator<TreeSet<String>> folhas = folhasPres.values().iterator();
   while(folhas.hasNext() && naoFoi) {
     if(folhas.next().contains(num)) naoFoi = false;
   }
   return naoFoi;
}
```

1.5

```
public void insereFolha(GregorianCalendar data, TreeSet<String> fp) {
   TreeSet<String> faux = new TreeSet<String>();
   for(String num : fp) faux.add(num);
   livroPres.put((GregorianCalendar) data.clone(), faux);
}
```

1.5

```
public TreeMap<GregorianCalendar, TreeSet<String>> getLivroPres() {
   TreeMap<GregorianCalendar, TreeSet<String>> copia =
      new TreeMap<GregorianCalendar, TreeSet<String>>();
   TreeSet<String>> folhaAux = new TreeSet<String>();
   for(GregorianCalendar data : livroPres.keySet()) {
      for(String num : livroPres.get(data)) folhaAux.add(num);
      copia.put((GregorianCalendar) data.clone(), folhaAux);
      folhaAux = new TreeSet<String>();
   }
   return copia;
}
```

ou ainda:

```
public TreeMap<GregorianCalendar, TreeSet<String>> getLivroPres() {
   TreeMap<GregorianCalendar, TreeSet<String>> copia =
      new TreeMap<GregorianCalendar, TreeSet<String>>();
   for(GregorianCalendar data : livroPres.keySet())
      copia.put((GregorianCalendar) data.clone(),
                livroPres.get(data).clone());
   return copia;
}
```

1.5

RESOLUÇÃO PARTE II:

```
public abstract class Veiculo implements Serializable {
   // Variáveis de Instância
   private String matric;
   private String marca;
   private String cor;
   private double preco;
   // Construtores
   public Veiculo(String mat, String mc, String cl, double pr) {
     matric = mat; marca = mc; cor = cl; preco = pr;
   }
   public Veiculo(Veiculo v) {
     matric = v.getMatric();   }
   // Gets e Sets
   public String getMatricula() { return matric; }
   …
   public String toString() {
     StringBuilder s = new StringBuilder("—Veiculo --\n");
     s.append(this.getClass().getName() + this.getMatric());
     s.append("Cor: " + this.getCor() + "\n");
     …
     return s.toString();
   }
   // Métodos abstractos
   public abstract double custoFinal();
   public abstract Veiculo clone();
   public abstract boolean equals(Object o);
}
```

```
public class Automovel extends Veiculo {
   // Membros de Classe
   public static double TAXA_CIL;
   public static void mudaTaxa(double nvTaxa) { TAXA_CIL = nvTaxa; }
   // Variáveis de Instância
   private double consumo;
   private int cilindrada;
   private boolean gasolina;
   // Construtores
   public Automovel(String mat, String mc, String cl, double pr,
                    double cons, int cil) {
     super(mat, mc, cl, pr); consumo = cons; cilindrada = cil;
   }
   public Automovel(Automovel auto) {
     super(auto.getMatric(), auto.getCor(), …);
     consumo = autogetCons(); …
   }

   // Gets e Sets e outros concretos
   …
   public String toString() {
     StringBuilder s = new StringBuilder(super.toString());
     s.append("Cilindrada: " + this.getCil() + "\n");
     …
     return s.toString();
   }

   public double custoFinal() {
     return this.getPreco() + TAXA_CIL*cilindrada;
   }
   public Automovel clone() { return new Automovel(this); };

   public boolean equals(Object o) { … }
}
```

2 + 2 = 4

```
public void insere(Veiculo v) {
   veículos.put(v.getMatric(), v.clone());
}
```

0.5

```
public Stand(Stand st) {
   nome = st.getNome();
   HashMap<String, Cliente> clientes = new HashMap<String, Cliente>();
   TreeMap<String, Veiculo> veiculos = new TreeMap<String, Veiculo>();
   for(Cliente c : st.getClientes())
        clientes.put(c.getCodigo(), c.clone());
   for(Veiculo v : st.getVeiculos())
        veiculos.put(v.getMatric(), v.clone());
}
```

ou simplesmente, sabendo-se que getClientes() é um "deep clone":

```
public Stand(Stand st) {
   nome = st.getNome();
   clientes = st.getClientes(); veiculos = st.getVeiculos();
}
```

1.5

```
public void insereCompra(String codCli, String matric) {
   clientes.get(codCli).getCompras().add(matric);
}
```

1.0

```
public double valorVendasA(String cli) {
   for(Cliente c : clientes.get(cli))
      for(String codV : c.getCompras())
         total += veiculos.get(codV).custoFinal();
}
```

1.5

```
public int doTipo(String tipo) {
   int total = 0;
   for(Veiculo v : veiculos.values())
     if(v.getClass().getName().equals(tipo)) total++;
   return total;
}
```

1.5

```
public String telfCliente(String nome) {
   String telf = “”; boolean enc = false;
   Cliente c = new Cliente();
   Iterator<Cliente> cli = clientes.values().iterator();
   while(cli.hasNext() && ! enc) {
     c = cli.next();
     if(c.getNome().equals(nome)) {
        telf = c.getTelf(); enc = true;
     }
   return telf;
}
```

1.5