Exame de Paradigmas da Programação IV
1ª chamada

2003.06.26
Duração: **2h30min**

Leia o teste com muita atenção antes de começar. Deve tentar resolver os exercícios procurando, o mais possível, reutilizar definições já existentes.
RESPONDA A CADA PARTE EM FOLHAS SEPARADAS.

PARTE I

1. Considere as seguintes definições JAVA:

```
public class EntradaDicionario {
  private String termo;
  private String definicao; //definicao do termo

  public EntradaDicionario(String t, String d) {
    this.termo = t;
    this.definicao = d;
  }

  public String getTermo() { return this.termo;}
  public String getDefinicao() {return this.definicao;}

  //...
}
```

sendo que o Dicionário é implementado como sendo uma sequência de entradas do tipo `EntradaDicionario`.

```
public class Dicionario {
  private String nomeDic;   //nome do dicionário
  private Vector entradas;  //Vector com as entradas

  public Dicionario(String nomeDic) {
    this.nomeDic = nomeDic;
    this.entradas = new Vector();
  }
 //...
}
```

Tendo em conta esta modelação implemente os seguintes métodos de `Dicionario`:

   (a) `void inserirEntrada(EntradaDicionario ed) throws ExistingEntryException`

   (b) `EntradaDicionario getEntrada(String termo) throws EntryDoesNotExistException`

   (c) `boolean existeTermo(String termo)`

   (d) `Collection getTermos()`

   (e) `boolean equals(Dicionario d)`

2. Considere agora que se pretendem criar Dicionários em que a informação associada a cada termo, para além da definição, contém também uma lista de *sinónimos*. Crie as classes necessárias para o efeito, sabendo que instâncias da classe `DicionarioComSinonimos` deverão responder aos métodos de `Dicionario` (embora os tipos dos parâmetros possam mudar!) bem assim como aos métodos:

   (a) `Vector getSinonimos(String termo) throws EntryDoesNotExistException`

   (b) `void addSinonimo(String termo, String sinonimo) throws EntryDoesNotExistException`

   (c) `boolean saoSinonimos(String termoA, String termoB) throws EntryDoesNotExistException`

3. Relembre o trabalho prático. Considere que foi desenvolvida a classe `Parque` que implementa a interface `IParque`:

```
public inteface IParque {
  public void entra(Cartao c, Veiculo a) throws SemPermissaoException;
  public void sai(String matricula) throws VeiculoNaoExisteException;
  public String getRelatorioSimples();
  public String getRelatorioCompleto();
}
```

   Desenvolva a classe `ParqueComRecusados` que, para além de implementar IParque mantém o registo de todas as tentativas de entrada recusadas, permitindo obter esse registo com o método `Collection getRecusas()`. (Nota: no registo são guardados os cartões a que foi recusada a entrada)

4. Recorde a matéria leccionada nas aulas teóricas. Diga por palavras suas o que entende serem as principais diferenças entre os conceitos de *classe abstracta* e *interface*. Descreva em que situações de modelação de um problema é que consideraria mais vantajoso utilizar um ou outro.

Parte II

5. Considere que se pretende modelar um *Stand* de automóveis. O stand que queremos construir comercializa, de momento, três categorias de carros: os *Utilitários*, os *Comerciais* e os *Desportivos*. É no entanto de esperar que em qualquer altura o stand possa vender outro tipo de veículos, pelo que a solução a desenvolver deve estar preparada para tal.

   O Sistema de Informação deve guardar para todas as categorias de carros a seguinte informação:

   - Matrícula
   - Cor
   - Peso
   - Preço
   - Cilindrada

   Além desta informação para os carros Utilitários acrescenta-se o consumo e o número de portas, para os Comerciais a tara e o volume de carga e os Desportivos guardam informação acerca do tempo de aceleração 0-100km/h e o número de cilindros do motor.

   O Stand além da informação sobre os carros de que actualmente dispõe, deve conter informação relativa ao seu nome e localização.

   Tendo em conta estas premissas, responda às seguintes questões:

   (a) Modele o sistema desenhando as classes que o constituem.

   (b) Escreva os métodos necessários para efectuar as seguintes operações sobre um Stand:

      i. Criação de um stand;
      ii. Inserção de um carro no stand;
      iii. Venda (remoção) de um carro;
      iv. Número total de carros existentes no stand;
      v. Número de carros por tipo (Comercial, Utilitário, Desportivo);
      vi. Visualização de um stand sobre a forma de String;
      vii. Gravação do stand para ficheiro utilizando uma ObjectStream;
      viii. Leitura de um stand a partir de um ficheiro utilizando ObjectStream.

   Deve utilizar excepções para tornar a sua solução o mais completa possível. Será valorizada a capacidade de modularidade que as soluções apresentem.