PARADIGMAS DE PROGRAMAÇÃO IV
2º ANO/2º SEMESTRE – LESI e LMCC

EXAME de RECURSO – 22 de Julho de 2005

Cotação - 20 valores - Duração - 2h00m

RESPONDA A CADA PARTE EM FOLHAS SEPARADAS

PARTE I (1.5 + 2.0 + 2.0)

A classe **Dicionário** apresentada em seguida implementa, estruturalmente, uma correspondência entre um *termo* (String) e um ArrayList de Strings contendo os *significados* de tal termo (cf. amar -> [gostar, adorar, …]).

```
public class Dicionario implements Serializable, Cloneable {
        private HashMap dicio;  // termo -> lista de significados
        // métodos a definir
}
```

São as seguintes as funcionalidades da classe **Dicionario** que devem ser implementadas:

- Método que devolve o número total de significados existentes no dicionário;
- Método que dado um *significado* devolve um ArrayList com todos os *termos* de que tal palavra é um significado;
- Método que recebe um dicionário como parâmetro e o junta ao dicionário receptor cf. as seguintes regras: para cada termo do dicionário parâmetro que não exista no dicionário receptor, tal termo e seus significados são inseridos; se o termo existir no dicionário receptor então apenas os significados novos devem ser acrescentados à lista de significados desse termo;

PARTE II (1.5 + 2.0 + 1.5 + 2.0)

Apresentam-se em seguida as definições principais de 3 classes de JAVA que irão permitir implementar as operações fundamentais de um sistema de gestão de uma Biblioteca. A classe **Leitor** representa toda a informação necessária sobre um leitor registado na biblioteca, designadamente, o seu código, nome e lista de livros já consultados (sob a forma de códigos dos respectivos livros) e lista de livros actualmente requisitados (cf. códigos). Para cada variável X das classes apresentadas considere que os métodos **daX()** e **mudaX(..)** existem como pré-definidos, bem como os outros indicados em comentário.

```
public class Leitor implements Serializable, Cloneable {

        private String codigo; private String nome;
        private ArrayList consultados; // códigos dos Livros consultados
        private ArrayList requisitados; // códigos dos Livros requisitados

        public Leitor() {
           codigo = ""; nome = "";
           consultados = new ArrayList();
           requisitados = new ArrayList();
        }
        public Leitor(String cod, String nom, ArrayList consults, ArrayList reqs) {
              codigo = cod; nome = nom;
              consultados = (ArrayList) consults.clone();
              requisitados = (ArrayList) reqs.clone();
        }
        // outros métodos cf. daX(), mudaX(..), toString(), clone() estão disponíveis
}
```

A classe **Livro** é definida como:

```
public class Livro implements Serializable, Cloneable {
        private String codigo; private String titulo;
        private String autor; private int ano;
        private String requisitante; // se valor for "" então está livre !!

        public Livro() {
             // inicializações das variáveis
        }
        public Livro(.....) {
            // recebe todos os parâmetros e inicializa a instância de Livro
        }
        // e daX(), mudaX(..), toString() e clone(), etc.
}
```

Chegamos assim à classe principal que se pretende implementar, a classe **Biblioteca**. A classe **Biblioteca** é representada por um **HashMap** que associa um *código de livro* a uma instância de **Livro**, e ainda um outro **HashMap** que associa um *código de leitor* à informação completa de um **Leitor**, cf:

```
public class Biblioteca implements Serializable, Cloneable {

        private HashMap livros;
        // Trata-se um HashMap de código de livro (String) -> Livro

        private HashMap leitores;
        // Trata-se um HashMap de código de leitor (String) -> Leitor

        public Biblioteca() {
          livros = new HashMap();
          leitores = new HashMap();
        }

        // métodos de instância a definir neste teste
}
```

A classe **Biblioteca** deverá agora ser completada com um conjunto de métodos que permitam realizar as diversas operações que se pretendem ver disponíveis relativamente à gestão de uma biblioteca. Estes métodos devem ser, idealmente, **robustos**, ou seja, lançando excepções caso não possam por qualquer razão ser executados correctamente. Caso não consiga implementar tal requisito, apresente o código que implementaria sem usar excepções. São as seguintes as funcionalidades da classe **Biblioteca** que devem ser implementadas nesta PARTE II:

- Método que dado o código de um leitor registado devolve o número de livros por ele requisitados;
- Método que devolve o número total de livros consultados mais os actualmente requisitados;
- Método que devolve um registo completo de um livro dado o seu código;
- Método que determina o conjunto dos códigos dos livros actualmente requisitados;

PARTE III (2.0 + 2.5 + 2.0)

- Método que determina o código do leitor com mais livros requisitados;
- Método robusto que regista a devolução à biblioteca de um livro requisitado por um dado leitor;
- Método toString() da classe Biblioteca;

Prof. F. Mário Martins