PARADIGMAS DE PROGRAMAÇÃO IV

2º ANO/2º SEMESTRE – LESI e LMCC

EXAME de 2ª CHAMADA – 6 de Julho de 2005
Cotação - 20 valores - Duração - 2h00m

RESPONDA A CADA PARTE EM FOLHAS SEPARADAS

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 **Videoteca**. A classe **Cliente** representa toda a informação necessária sobre um cliente registado, designadamente, o seu código, nome, a lista de produtos já requisitados (sob a forma de códigos) e a lista de produtos actualmente requisitados (cf. códigos). Para cada variável de instância de nome X das classes apresentadas considere que os métodos **daX()** e **mudaX(..)** existem como pré-definidos,.bem como os outros indicados em comentário.

```
public class Cliente implements Serializable, Cloneable {

        private String codigo;
        private String nome;
        private ArrayList alugados; // códigos de todos os produtos já alugados
        private ArrayList requisitados; // códigos dos produtos actualmente requisitados

        public Cliente() {
           codigo = ""; nome = ""; alugados = new ArrayList();
           requisitados = new ArrayList();
        }

        public Cliente(String cod, String nom, ArrayList alug, ArrayList reqs) {
              codigo = cod; nome = nom;
              alugados = (ArrayList) alug.clone();
              requisitados = (ArrayList) reqs.clone();
        }
        // outros métodos cf. daX(), mudaX(..), toString(), clone() estão disponíveis
}
```

A classe **Produto** é definida como:

```
public class Produto implements Serializable, Cloneable {

        private String codigo;
        private String titulo;
        private String autor;
        private int ano;
        private double tempo; // tempo total de duração
        private String requisitante; // se valor for "" então está livre !!

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

Chegamos assim à classe principal que se pretende implementar, a classe **Videoteca**. A classe **Videoteca** é representada por um **HashMap** que associa um *código de produto* a uma instância de **Produto**, e ainda um outro **HashMap** que associa um *código de cliente* à informação completa de um **Cliente**, cf:

```
public class Videoteca implements Serializable, Cloneable {

        private HashMap produtos;
        // Trata-se um HashMap de código de produto (String) -> Produto
        private HashMap clientes;
        // Trata-se um HashMap de código de cliente (String) -> Cliente

        public Videoteca() {
          produtos = new HashMap();
          clientes = new HashMap();
        }

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

Complemente agora a classe **Videoteca** implementando os seguintes métodos:

PARTE I (2.0 +2.0 +2.0 valores)

1.- Método de instância **public Set codigosLivres()** que devolve um conjunto contendo os códigos de todos os produtos actualmente não requisitados, ou seja, livres;

2.- Construtor **public Videoteca(Map prods, Map clientes)** que cria uma instância de Videoteca a partir dos dois Maps, respectivamente de produtos e clientes, passados como parâmetro;

3.- Método **public Cliente requisitou(String codProduto)**, que devolve a informação completa sobre o cliente que requisitou o produto cujo código é dado como parâmetro, caso exista;

PARTE II (2.0 +2.5 +2.0 valores)

4.- Método *robusto* **public void aluguer(String codProduto, String codCliente)**, que regista o aluguer de um dado produto por um dado cliente. Ambos os códigos devem ser correctos, ou seja, existir na Videoteca. Se tal não for verdade o método deverá lançar uma excepção do tipo *AluguerException* (assuma que já foi definida) com um texto adequado;

5.- Método *robusto* **public void registaEntrega(String codProduto, String codCliente)**, que regista a entrega de um dado produto por um dado cliente. Qualquer erro deverá resultar no lançamento de uma excepção adequada, tal como no método anterior (não é necessário definir as classes de excepção);

6.- Método *robusto* **public void actualizaVideoteca(ArrayList produtos, ArrayList clientes)** que tendo como parâmetros 2 ArrayList de novos produtos e novos clientes os insere correctamente na Videoteca;

PARTE III (2.0 + 3.0 + 2.5 valores)

7.- Método **public String toString()** que cria uma representação sob a forma de caracteres do estado actual da Videoteca;

8.- A classe **Videoteca** implementa uma videoteca incapaz de perante pedidos de aluguer de produtos que estão requisitados colocar os clientes numa lista de espera, de tal forma que quanto tal produto for

libertado por quem o requisitou, seja efectuado automaticamente o aluguer de tal produto ao cliente em primeiro lugar em tal fila de espera. A classe Videoteca apresentada apenas é capaz de rejeitar tal pedido.
Assim, pretende-se criar uma subclasse de **Videoteca**, de nome **IntelVideoteca**, que possua métodos mais "inteligentes" para implementar o pedido de aluguer e a entrega de produto, como sendo métodos capazes de gerir a lista de espera de clientes que pretendem alugar um produto que estava requisitado. De notar que as filas de espera estão associadas a produtos requisitados.
Desenvolva a estrutura de dados desta nova classe e o código destes dois métodos explicando claramente como passa a funcionar o sistema, ou seja, o que cada método realiza de facto.

9.- Considere que se pretende generalizar a Videoteca por forma a que em vez de termos apenas a classe rígida **Produto** pudéssemos ter tipos de produtos distintos, cf. filmes em cassetes VHS, filmes em DVD, etc. Como o faria ? Apresente a definição da classe **DVD**, mas apenas na sua estrutura e construtores.

Prof. F. Mário Martins