PARADIGMAS DE PROGRAMAÇÃO IV

2º ANO/2º SEMESTRE – LESI e LMCC

EXAME de 1ª CHAMADA – 23 de Junho de 2005
Cotação - 20 valores
Duração - 2h00m

Responda a cada parte em folhas separadas

Apresentam-se em seguida as definições principais de classes de JAVA que irão permitir implementar as operações fundamentais de um sistema de controlo de entradas e saídas de barcos de um porto, cf. a classe a desenvolver, de nome **Porto**.
A classe **Barco**, apresentada a seguir, representa toda a informação que é necessário possuir sobre um *barco*, designadamente: matrícula, nome do comandante, comprimento em metros, e lista de nomes da tripulação. Os métodos **daX()** e **mudaX(..)**, devem assumir-se como sendo pré-definidos para cada variável de instância de cada uma das classes apresentadas. Os métodos **toString()** e **clone()** devem ser igualmente assumidos como existentes.

```
public class Barco implements Serializable, Cloneable {

// Variáveis de Instância

private String matricula;         // código inequívoco do barco
private String comandante;        //  nome do comandante
private double comprimento;       //  em metros
private ArrayList tripulacao;     // nomes dos tripulantes como strings

/**
 * Construtores
 */
public Barco() {
  matricula = ""; comandante = ""; comprimento = 0.0;
  tripulacao = new ArrayList();
}

public Barco(String mat, String com,
             double comp, ArrayList trip) {
  matricula = mat; comandante = com;
  comprimento = comp; tripulacao = (ArrayList) trip.clone();
}

 // Métodos de Instância daX(), mudaX(..), toString() e clone()
}
```

A classe **Porto** faz o registo das entradas e saídas de barcos, determina os valores a pagar pela estadia, etc. Para tal regista os barcos entrados no porto (cf. instâncias da classe Barco) e depois regista as saídas dos mesmos adicionando informação adicional cf. classe **RegistoDeBarco** apresentada em seguida.

```
public class RegistoDeBarco {

// Variáveis de Instância

private String matricula;   //  id do barco
private int diasDeEstadia;  // dias de estadia no porto
private double totalPago;   // total a Pagar = precoDia * numDias +
                            //                 precoMetro * comprimento
```

```
    /**
     * Construtores
     **/
    public RegistoDeBarco() {
          barco = new Barco();
          diasDeEstadia = 0; totalPago = 0.0;
    }

    public RegistoDeBarco(Barco b, int dias, double pago) {
          barco = (Barco) b.clone();
          diasDeEstadia = dias; totalPago = pago;
    }

    // Métodos de Instância daX(), mudaX(..), toString() e clone()
}
```

A classe **Porto**, a completar neste teste, possui portanto uma estrutura baseada em dois HashMaps que representam os registos de entrada e os registos de saída dos barcos.

```
public class Porto implements Serializable, Cloneable {

    // Variáveis de Instância

    private HashMap registoEntradas; // regista entrada dos barcos
    // Trata-de um HashMap de Matricula -> Barco
    private HashMap registoSaidas;   // regista saída dos barcos
    // Trata-se de um HashMap de Matricula -> RegistoDeBarco
    private double precoMetro;       //   preco por metro de barco
    private double precoDia;         //    preco por dia de estadia

    /**
     * Construtor por omissão
     **/
    public Porto() {
       registoEntradas = new HashMap();
       registoSaidas = new HashMap();
       precoMetro = 0.0; precoDia = 0.0;
    }

    // Construtores e Métodos de Instância a definir cf. pedido no exame
}
```

PARTE I

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

1.- Método *robusto* **public Barco daInfoBarco(String matricula)**, que dada uma matrícula correcta, ou seja, de um barco efectivamente entrado no porto, devolve o registo completo do respectivo barco. Se tal matrícula não existir deverá ser lançada uma excepção do tipo *EntradaException* (assuma que já foi definida) com um texto adequado.

2.- Método *robusto* **public void registaEntrada(Barco barco)**, que tendo como parâmetro a informação completa de um barco, regista a sua entrada no porto. Se a matrícula do barco for errada deverá ser lançada uma excepção do tipo *EntradaException* (assuma que já foi definida) com um texto adequado.

3.- Método *robusto* **public void registaSaida(String matricula, int numDias)**, que tendo como parâmetro a matrícula de um barco e o número de dias da sua estadia, registe a sua saída do porto, actualizando correctamente o atributo **registoSaidas** da classe **Porto**.

PARTE II

4.- Método **public Map daEntradas()**, que devolve uma cópia completa, sem partilha ("deep"), do HashMap que regista as entradas de barcos no porto;

5.- Método **public Set barcosNoPorto()**, que devolve um conjunto contendo as matrículas de todos os barcos ainda estacionados no porto;

6.- Método **public String barcoDe(String comandante)**, que devolva a matrícula do barco do comandante cujo nome é dado como parâmetro, caso exista;

PARTE III

7.- Método **public void gravaPorto(String fich1, String fich2)** que, simultaneamente, grava em ficheiro de texto (*fich1*) e em ficheiro de objectos (*fich2*) toda a informação actual do porto;

8.- A classe **Porto** implementa um porto de capacidade ilimitada. Todos os barcos entram. Defina a estrutura de uma subclasse de **Porto**, de nome **PortoLimitado**, que possua uma variável lógica que diga se este está ou não cheio e um método accionável pelo operador que muda tal estado cf. **public void estaCheio()**, e redefina nesta classe o método anterior **public void registaEntrada(Barco barco)** de tal forma que se o barco não puder entrar no porto é colocado numa fila de espera.

9.- Considere que se pretende generalizar o sistema de controlo de barcos por forma a que vários tipos de barcos possam ser considerados e reconhecidos, por exemplo: BarcoDeRecreio, Petroleiro, Veleiro, NavioComercial, Rebocador, etc. Um barco de recreio possui uma variável de instância adicional particular do tipo **int capacidade**, um petroleiro também, um veleiro possui uma variável de instância do tipo **int numVelas**, um navio comercial uma variável **String cargaTipo** e um rebocador uma variável **double maxCargaRebocavel**.
Crie uma classe **AbstractBarco** e apresente a definição das subclasses **Petroleiro** e **Rebocador** em termos da sua estrutura de variáveis e de dois construtores, um por omissão (ou seja sem parâmetros) e outro completo, garantindo que quer um Petroleiro quer um Rebocador continuam a possui toda a informação anteriormente representada na classe Barco.

Prof. F. Mário Martins