Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação Orientada a Objetos
AP3 2° semestre de 2016.

Nome –

Assinatura –

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

Questão 1) (5.0 pontos)

Suponha que tenhamos que implementar um sistema para controle de gastos de automóveis. Os gastos essenciais são de 2 tipos: abastecimento e manutenção, conforme as classes abaixo.

Altere este programa de forma que estes objetos possam ser manipulados juntos, **numa mesma coleção**, no método main(). As classes *Manutencao* e *Abastecimento* podem ser modificadas, mas devem continuar existindo. Use recursos de OO sempre que possível.

```
import java.util.Arrays;
import java.util.List;

class Manutencao {
	String peca;
	double custo;
	public Manutencao(String peca, double custo) {
		this.peca = peca;
		this.custo = custo;
	}
	public double getCusto() {
		return custo;
	}
}

class Abastecimento {
	float valor;
```

```
	String posto;
	public Abastecimento(float valor, String posto) {
		this.valor = valor;
		this.posto = posto;
	}
	public float getValor() {
		return valor;
	}
}

public class AP2_2014_2_Q3 {
	public static void main(String[] args) {
		double somaReparos = 0, somaManutencao = 0;
		List <Manutencao> reparos = Arrays.asList(new Manutencao("Freio",
150), new Manutencao("Oleo", 200));
		for (Manutencao m : reparos) {
			somaReparos = somaReparos + m.getCusto();
		}
		List <Abastecimento> abastecimentos = Arrays.asList(new
Abastecimento(80, "BR"), new Abastecimento(50, "Shell"));
		for (Abastecimento a : abastecimentos) {
			somaManutencao = somaManutencao + a.getValor();
		}
		System.out.println("A soma dos valores gastos e': " +
(somaManutencao + somaReparos));
	}
}
```

RESPOSTA:

```
/* Enunciado !!!
class Manutencao {
	String peca;
	double custo;
	public Manutencao(String peca, double custo) {
		this.peca = peca;
		this.custo = custo;
	}
	public double getCusto() {
		return custo;
	}
}

class Abastecimento {
	float valor;
	String posto;
	public Abastecimento(float valor, String posto) {
		this.valor = valor;
		this.posto = posto;
	}
	public float getValor() {
		return valor;
```

```
	}
}
*/

// Gabarito !!!!

interface Gasto {
	double getCusto();
}

class Manutencao implements Gasto {
	String peca;
	double custo;
	public Manutencao(String peca, double custo) {
		this.peca = peca;
		this.custo = custo;
	}
	public double getCusto() {
		return custo;
	}
}

class Abastecimento implements Gasto {
	float valor;
	String posto;
	public Abastecimento(float valor, String posto) {
		this.valor = valor;
		this.posto = posto;
	}
	public double getCusto() {
		return valor;
	}
}

public class AP3_2016_2_Q2 {
	public static void main(String[] args) {
		/* Enunciado !!!
		double somaReparos = 0, somaManutencao = 0;

		List <Manutencao> reparos = Arrays.asList(new Manutencao("Freio",
150), new Manutencao("Oleo", 200));
		for (Manutencao m : reparos) {
			somaReparos = somaReparos + m.getCusto();
		}
		List <Abastecimento> abastecimentos = Arrays.asList(new
Abastecimento(80, "BR"), new Abastecimento(50, "Shell"));
		for (Abastecimento a : abastecimentos) {
			somaManutencao = somaManutencao + a.getValor();
		}
		System.out.println("A soma dos valores gastos e': " +
(somaManutencao + somaReparos));
		*/
```

```
            // Gabarito !!!
            double soma = 0;
            List <Manutencao> reparos = Arrays.asList(new Manutencao("Freio",
150), new Manutencao("Oleo", 200));
            List <Abastecimento> abastecimentos = Arrays.asList(new
Abastecimento(80, "BR"), new Abastecimento(50, "Shell"));
            List <Gasto> gastos = new ArrayList<Gasto>();
            gastos.addAll(reparos);
            gastos.addAll(abastecimentos);
            for (Gasto g : gastos) {
                  soma = soma + g.getCusto();
            }
            System.out.println("A soma dos valores gastos e': " + soma);
      }
}
```

Questão 2) (5.0 pontos)
Considere uma estrutura de dados construída sobre uma lista duplamente encadeada, onde cada dado armazenado é associado a um grupo e onde cada grupo é identificado por um valor inteiro. Nessa estrutura, os dados são organizados de modo que, quando pertencem ao mesmo grupo, são colocados de maneira sucessiva no encadeamento e de modo que os grupos se mantenham ordenados de maneira crescente pelo valor identificador. Por exemplo, se o nó da lista encadeada é definido pela classe:

```
class No {
    int grupo;
    Object dado;
    No anterior;
    No proximo;
}
```

então a estrutura encadeada manterá as duplas $(5,E)$, $(4,B)$, $(5,C)$, $(9,D)$ e $(4,A)$, inseridas nesta ordem e onde o primeiro valor indica o grupo e o segundo valor o dado armazenado, conforme a seguinte organização *{{4,B}, {4,A}, {5,E}, {5,C}, {9,D}}*.

A estrutura proposta contém apenas três operações: (i) inserção, (ii) consulta e (iii) verificação de lista vazia. Nessa questão, você deverá implementar tal estrutura na classe MinhaListaMaluca, fazendo uso da classe No e respeitando (implementando) a interface:

```
public interface ListaMaluca {
      public void inserir(int grupo, Object dado);
      public Object consultar(int grupo);
      public boolean estaVazia();
}
```

(a) (2.0 pontos) O método inserir introduz um novo elemento, identificado pelo argumento dado, ao fim do sub-lista encadeada que contém dados pertencentes ao grupo identificado pelo argumento grupo. Por exemplo, se a dupla $(5,F)$ for inserida no

encadeamento ilustrado anteriormente, então o nó que armazena essa dupla entraria entre os nós dos dados *C* e *D*.

(b) (2.0 pontos) O método consultar retorna a referência ao dado associado ao primeiro nó da sub-lista encadeada que contém elementos do grupo identificado pelo argumento grupo. Por exemplo, a invocação do comando `consultar (9)` retorna a referência ao dado *D*. Caso a sub-lista de dados associados ao valor grupo esteja vazia, o método consultar levanta a exceção `EmptyGroupException`, declarada como:

```
public class EmptyGroupException extends Exception {
      public EmptyGroupException() {
            super("O grupo de origem está vazio");
      }
}
```

(c) (1.0 ponto) Finalmente, o método estaVazia retorna **true**, caso a lista esteja vazia, ou **false**, caso não esteja. Por definição, as instâncias de MinhaListaMaluca são inicializadas como listas vazias.

RESPOSTA:

```
class EmptyGroupException extends RuntimeException {
  public EmptyGroupException() {
    super("O grupo de origem está vazio");
  }
}

class No {
  int grupo;
  Object dado;
  No anterior;
  No proximo;

  No(int grupo, Object dado) {
    this.grupo = grupo;
    this.dado = dado;
  }
}


interface ListaMaluca {
  public void inserir(int grupo, Object dado);
  public Object consultar(int grupo);
  public boolean estaVazia();
}

class MinhaListaMaluca implements ListaMaluca {
```

```
  private No primeiro = null;
  private No ultimo = null;

  //Resposta da Questão 2 letra (b)
  private No obterPrimeiro(int grupo) {
    No atual = this.primeiro;
    while ((atual != null) && (atual.grupo < grupo))
      atual = atual.proximo;
    return atual;
  }

  public Object consultar(int grupo) {
    No no = obterPrimeiro(grupo);
    if ((no == null) || (no.grupo != grupo))
      throw new EmptyGroupException();
    return no.dado;
  }

  //Resposta da Questão 2 letra (a)
  private void inserirAntes(No atual, No novo) {
    if (atual != null) {
      novo.anterior = atual.anterior;
      novo.proximo = atual;
      atual.anterior = novo;
      if (novo.anterior != null) novo.anterior.proximo = novo;
      else this.primeiro = novo;
    }
    else {
      novo.anterior = this.ultimo;
      novo.proximo = null;
      if (this.ultimo != null) this.ultimo.proximo = novo;
      else this.primeiro = novo;
      this.ultimo = novo;
    }
  }

  public void inserir(int grupo, Object dado) {
    inserirAntes(obterPrimeiro(grupo + 1), new No(grupo, dado));
  }

  //Resposta da Questão 2 letra (c)
  public boolean estaVazia() { return (this.primeiro == null); }
}
```

```
//Classe para testar a implementação...
public class Q2_AP3_2016_2{
  public static void main (String[] args) throws Exception{
    ListaMaluca l = new MinhaListaMaluca();
    try{
      System.out.println(l.estaVazia());
      l.inserir(5, new String("E"));
      System.out.println(l.estaVazia());
      l.inserir(4, new String("B"));
      l.inserir(5, new String("C"));
      l.inserir(9, new String("D"));
      l.inserir(4, new String("A"));
      l.inserir(5, new String("F"));
      System.out.println(l.consultar(9));
      System.out.println(l.consultar(6));
    } catch (Exception e) { System.out.println(e); }
  }
}
```