Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação Orientada a Objetos
AP3 1° semestre de 2018.

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)

Existe uma estrutura de dados do tipo sacola, suportando duas operações:
1 x: joga um elemento **x** na sacola.
2 x: retira um elemento da sacola.
Dada uma sequência de operações que retornam valores, seu programa deve adivinhar a estrutura de dados. É uma pilha (último que entrou é o primeiro a sair), uma fila (primeiro que entrou é o primeiro a sair), uma fila de prioridade (sempre retire os elementos grandes primeiro) ou qualquer outra coisa que você dificilmente consegue imaginar!

Seu progama deve ler de um arquivo de teste as seguintes informações:

- um valor inteiro **n (n > 0)**;
- **n** linhas com comandos do tipo 1, ou do tipo 2, seguido de um número inteiro **x**; e
- o arquivo termina com **n <= 0.**

Para cada caso de teste, mostre uma das seguintes respostas na tela:
stack: é definitivamente uma pilha;
queue: é definitivamente uma fila;
priority queue: é definitivamente uma fila de prioridade;
impossible: não pode ser uma pilha, uma fila ou uma fila de prioridade; e
not sure: pode ser mais de uma das três estruturas mencionadas acima.

A seguir, seguem alguns exemplos de entrada e saída:

Arquivo de entrada:	Resposta na tela:
6	**queue**
1 1	**not sure**
1 2	**impossible**
1 3	**stack**
2 1	**priority queue**
2 2	
2 3	
6	
1 1	
1 2	
1 3	
2 3	
2 2	
2 1	
2	
1 1	
2 2	
4	
1 2	
1 1	
2 1	
2 2	
7	
1 2	
1 5	
1 1	
1 3	
2 5	
1 4	
2 4	
0	

RESPOSTA:

```
import java.io.*;

public class POO_Q1_AP3_2018_1{
  public static void main(String[] args) throws IOException {
    BufferedReader arq_in;
    arq_in = new BufferedReader(new FileReader(args[0]));
    try{
      String linha = arq_in.readLine();
      int n = Integer.parseInt(linha);
      while(n != 0){
        if((n < 1) || (n > 1000)) continue;

        int in[] = new int[n], tam_in = 0;
        int out[] = new int[n], tam_out = 0;
        int acao, elem, i;
        int pilha[] = new int[n], topo = -1;
        int fila[] = new int[n];
        int heap[] = new int[n], ini_h = 0, fim_h = 0;
        int ep = 1, ef = 1, eh = 1;
        String[] partes;

        for(i = 0; i < n; i++){
          linha = arq_in.readLine();
          partes = linha.split(" ");
```

```
            acao = Integer.parseInt(partes[0]);
            elem = Integer.parseInt(partes[1]);
            if(acao == 1){
              heap[fim_h++] = elem;
              pilha[++topo] = in[tam_in++] = elem;
            }
            else if(acao == 2){
              if((topo != -1) && (elem != pilha[topo])) ep = 0;
              else topo--;
              if(Maior(heap, ini_h, fim_h) != elem) eh = 0;
              else{
                Apaga(heap, fim_h, elem);
                fim_h--;
              }
              fila[tam_out] = elem;
              out[tam_out++] = elem;
            }
          }

          for(i = 0; i < tam_out; i++) if(fila[i] != in[i]) ef = 0;

          if((ep + ef + eh) == 0) System.out.println("impossible");
          else if((ep + ef + eh) != 1) System.out.println("not sure");
          else if(ep == 1) System.out.println("stack");
          else if(eh == 1) System.out.println("priority queue");
          else System.out.println("queue");
          linha = arq_in.readLine();
          n = Integer.parseInt(linha);
        }
        arq_in.close();
      }catch(Exception e){ System.out.println(e); }
    }

    static int Maior(int vet[], int ini, int fim){
      int resp = vet[ini], i;
      for(i = ini; i < fim; i++) if(resp < vet[i]) resp = vet[i];
      return resp;
    }

    static void Apaga(int vet[], int n, int elem){
      int i;
      for(i = 0; i < n; i++) if(vet[i] == elem) break;
      if(i != n) vet[i] = vet[n - 1];
    }
}
```

Questão 2) (5.0 pontos)

Considere o código abaixo que representa um valor do IPTU (Imposto Predial e Territorial Urbano) a ser pago por um imóvel.

```
class IPTU implements Imposto {
	double taxa;
	public IPTU (double tx) {
		this.taxa = tx;
	}
	public double calculaValor(Imovel b) {
		return b.getValor() * this.taxa;
	}
}
```

a) (3.0) Defina **Imposto** e **Imovel**, utilizados na definição desta classe. Para **Imovel**, saiba que esta representa um imóvel, a qual possui uma lista de impostos (objetos do tipo Imposto) referentes a este imóvel. **Imposto** pode ser deduzido do código.
b) (1.0) Adicione um novo tipo de imposto para a taxa de bombeiros. Considere que o valor cobrado por este é fixo e vale 200.
c) (1.0) Num método main(), crie um imóvel com valor de 100000 e adicione 2 impostos a este: i) IPTU com taxa de 3% (0.03), e; ii) uma taxa de bombeiro. Imprima o valor do cálculo do imposto para este imóvel. O cálculo deve ser implementado de forma polimórfica.

RESPOSTA:

```
// Classe utilitárias necessárias para podermos
//trabalhar com listas
import java.util.ArrayList;
import java.util.List;

// Interface deduzida pelo fato de termos a definição
//class IPTU implements Imposto. Ou seja, Imposto
//precisa ser uma interface
// Solução de parte do item a)
interface Imposto {
	double calculaValor(Imovel b);
}

// Classe fornecida na questão
class IPTU implements Imposto {
	double taxa;

	public IPTU (double tx) {
		this.taxa = tx;
	}

	public double calculaValor(Imovel b) {
		return b.getValor() * this.taxa;
	}
}

// Classe pedida pelo item b) da questão.
class Bombeiro implements Imposto {
	public double calculaValor(Imovel b) {
		return 200;
	}
}
```

```java
// Classe requerida na questão.
//Apesar da questão só informar explicitamente que imóvel
//possui uma lista impostos, o campo valor também é necessário
//uma vez que o método getValor() da classe dada na questão
//precisa funcionar. Ou seja, a classe Imovel precisar
//armazenar o valor de alguma maneira.
class Imovel {
	private double valor;
	private List<Imposto> impostos;

	public Imovel (double valor) {
		this.valor = valor;
		impostos = new ArrayList<Imposto>();
	}

	public double getValor() {
		return this.valor;
	}

	public void adicionaImposto(Imposto i) {
		impostos.add(i);
	}

	// Neste método, a chamada a calculaValor é polimórfica,
	//uma vez que qualquer objeto do subtipo de Imposto pode
	//ser adicionado à lista
	public double calculaImposto() {
		double temp = 0;
		for (Imposto i : impostos) {
			temp = temp + i.calculaValor(this);
		}
		return temp;
	}
}

// Manipulações objetos pedidos no item c) desta questão
public class AP2_2014_1_Q3 {
	public static void main(String[] args) {
		Imovel apto = new Imovel(100000);
		apto.adicionaImposto(new IPTU(0.03));
		apto.adicionaImposto(new Bombeiro());
		System.out.println(apto.calculaImposto());
	}
}
```