Fundação CECIERJ – Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação Orientada a Objetos
APX3 2° semestre de 2020.

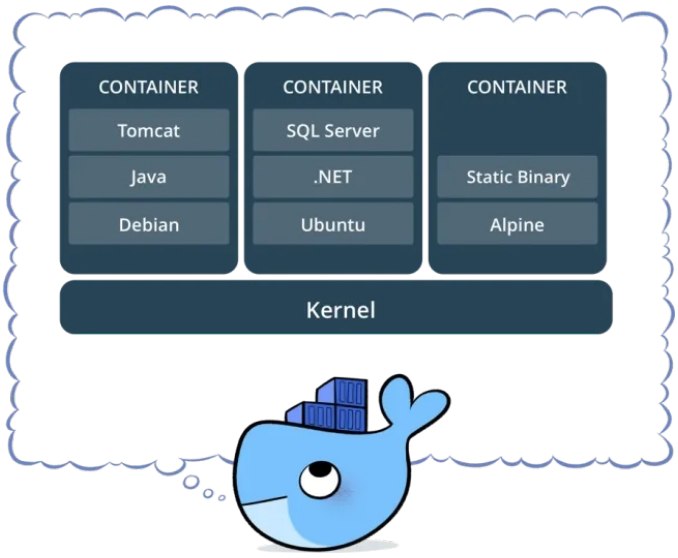
Nome –
Assinatura –

Observações:

1. **As respostas precisam ser entregues de forma manuscrita e a caneta.** A cópia deveria ser realizada após as soluções serem testadas num computador
2. **Utilize qualquer folha em branco para transcrever suas respostas.** Dê preferência a folhas pautadas (com linhas). Assine em cada folha que utilizar.
3. **Estas folhas devem ser escaneadas e enviadas pela plataforma**. Certifique-se de que estejam legíveis. Utilize algum aplicativo para escanear de forma que o(s) arquivo(s) fique(m) pequeno(s).
4. **Não seja o plágio do que ouve, vê, lê por aí. Estilo é plagiar a si mesmo** (Última frase é atribuída a Alfred Hitchcock). Cópias de código de sites também são consideradas plágio.

Questão 1) (5.0 pontos)

Contêineres em TI funcionam como máquinas virtuais leves que permitem, por exemplo, criarmos diferentes ambientes para testes de aplicações. Na figura abaixo, a máquina que hospeda os contêineres (máquina host) contém um kernel, sobre o qual estão instalados 3 contêineres.



Imagine que queiramos simular este ambiente. Considere o código abaixo:

```
1: public class AP3_2020_2_Q1 {
2:     public static void main(String[] args) {
3:             Software sublime = new Software("Sublime", 100);
4:             Software firefox = new Software("Firefox", 1500);
5:             Software chrome = new Software("Chrome", 2500);
6:             Software eclipse = new Software("Eclipse", 500);
7:             Software ubuntu = new Software("Ubuntu", 2000);
8:
9:             Container c1 = new Container();
10:            c1.addSoftware(chrome);
11:            c1.addSoftware(sublime);
12:
13:            Container c2 = new Container();
14:            c2.addSoftware(ubuntu);
15:            c2.addSoftware(firefox);
16:            c2.addSoftware(eclipse);
17:
18:            Compose containers = new Compose();
19:            containers.addContainer(c1);
20:            containers.addContainer(c2);
21:
22:            containers.run();
23:            c1.stop();
24:
25:            System.out.println("Tamanho total: " + containers.getTamanhoTempoReal());
26:    }
27:}
```

Das linhas 3-7 criamos softwares que podem ser instalados em contêineres. Nesta criação informamos os seus nomes e respectivos tamanhos. Das linhas 9-11 temos um exemplo de criação de contêineres e adição de softwares. De forma a facilitar a manipulação de diversos contêineres, na linha 18 temos um novo tipo (classe) sendo usado para facilitar esta manipulação. Na linha 22 vemos um exemplo de manipulação, onde todos os contêineres são executados a partir deste comando (neste nosso cenário, um contêiner, quando criado, possui estado inicial inativo). Apesar da facilidade de manipulação conjunta de contêineres, podemos manipulá-los de forma individual também, como está ocorrendo na linha 23, onde interrompemos a execução de um contêiner. Na linha 25 temos a chamada ao método *getTamanhoTempoReal*, o qual retornará a soma dos tamanhos de todos os softwares de contêineres que estejam ativos.

Implemente os recursos necessários de forma que o código acima funcione. Use OO sempre que possível. **Para testar sua implementação, reescreva a main() criando os contêineres da figura. Execute-os usando o tipo Compose e imprima o tamanho total dos contêineres**. Leve em conta os softwares listados na figura acima, os quais possuem os seguintes tamanhos:

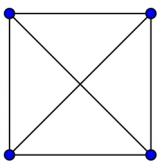
Software	Tomcat	Java	Debian	SQL Server	.NET	Ubuntu	Static Binary	Alpine
Tamanho	300	1000	2000	400	1000	2500	200	800

Questão 2) (5.0 pontos)
Dada a classe Grafo, adaptada da AD2 desse semestre, a seguir:

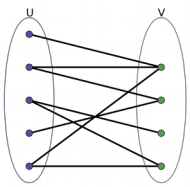
```
class Vizinho{
  int no_viz;
  Vizinho prox;

  Vizinho(int c){
    no_viz = c;
    prox = null;
  }

public String toString(){ return no_viz + " "; }
}

class Lista{
  int no;
  Vizinho prox_viz;
  Lista prox_no;

  Lista(int c){
    no = c;
    prox_viz = null;
    prox_no = null;
  }

  Vizinho pertence(int no){
    Vizinho resp = prox_viz;
    while((resp != null) && (no != resp.no_viz))
      resp = resp.prox;
    return resp;
  }

  void ins_Viz(int c){
    Vizinho v = pertence(c);
    if(v != null) return;
    v = new Vizinho(c);
    v.prox = prox_viz;
    prox_viz = v;
  }

  public String toString(){
    String resp = no + "(" + cor + "): \n";
    Vizinho p = prox_viz;
    while(p != null){
      resp += p.toString();
      p = p.prox;
    }
    return resp + "\n";
  }
}

//CLASSE A SER MINIMAMENTE ALTERADA
class Grafo{
  Lista prim;

  Grafo(){  prim = null; }

  Lista pertence(int no){
    Lista resp = prim;
    while((resp != null) && (no != resp.no)) resp = resp.prox_no;
    return resp;
  }

  void insere(int no){
    Lista p = pertence(no);
```

```
    if(p == null){
      p = new Lista(no);
      Lista q = prim;
      if(q == null){
        prim = p;
        return;
      }
      while(q.prox_no != null) q = q.prox_no;
      q.prox_no = p;
    }
  }

  void insere(int no1, int no2){
    Lista p = pertence(no1);
    p.ins_Viz(no2);
  }

  public String toString(){
    String resp = "";
    Lista p = prim;
    while(p != null){
      resp += p.toString();
      p = p.prox_no;
    }
    return resp;
  }
}
```

Escreva um programa em Java que leia um arquivo passado como parâmetro e classifique um grafo de acordo com as seguintes classes **NÃO EXCLUDENTES**:
→ Completo: grafo em que, para cada vértice do grafo, existe uma aresta conectando este vértice a cada um dos demais;



→ Bipartido: grafo cujos vértices podem ser divididos em dois conjuntos, nos quais não há arestas entre vértices de um mesmo conjunto;



→ Regular: grafo em que o número de vizinhos de cada nó é igual.

→ Simples: grafo que não se encaixa em nenhuma das classes supracitadas.

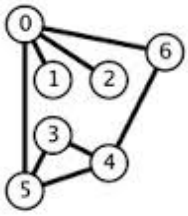
VOCÊ DEVE ALTERAR MINIMAMENTE A CLASSE GRAFO PARA ADOTAR ESSA CLASSIFICAÇÃO MENCIONADA.

ALÉM DISSO, VOCÊ DEVE ESCREVER AS CLASSES SUPRACITADAS USANDO OS CONCEITOS DE OO. NÃO SERÃO ACEITAS IMPLEMENTAÇÕES QUE NÃO LEVEM, EM CONSIDERAÇÃO, ESSES CONCEITOS.

Seu programa Java receberá um arquivo de entrada, passado como PARÂMETRO DE ENTRADA, QUE SÓ PODERÁ SER LIDO UMA ÚNICA VEZ, contendo os grafos a serem analisados. Um exemplo de arquivo de entrada seria:

```
0 1 2 3 4 5 6
0 1
1 0
0 2
2 0
0 5
5 0
0 6
6 0
3 4
4 3
3 5
5 3
4 5
5 4
4 6
6 4
FIM
```

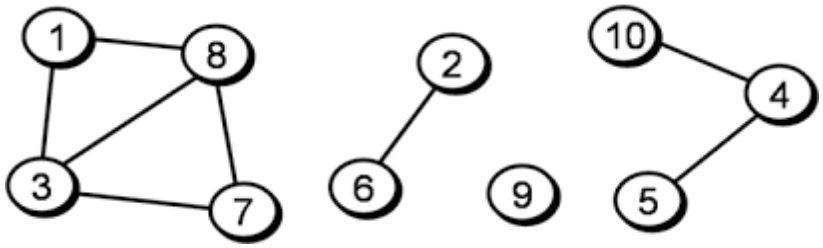
que representa o grafo a seguir:



A resposta seria:

```
Simples
```

Para os quatro grafos a seguir:



O arquivo de entrada seria:

```
1 3 7 8
1 3
3 1
1 8
8 1
3 7
```

```
7 3
3 8
8 3
7 8
8 7
FIM
2 6
2 6
6 2
FIM
9
FIM
4 5 10
4 5
5 4
4 10
10 4
FIM
```

As respostas seriam:

```
Simples
Bipartido – Completo - Regular
Regular
Simples
```

LEMBRE-SE: SEU PROGRAMA DEVE EXECUTAR COM QUAISQUER DADOS INFORMADOS COMO PARÂMETROS DE ENTRADA. SE O SEU PROGRAMA RESOLVER SOMENTE O PROBLEMA DO EXERCÍCIO SUPRACITADO, SUA QUESTÃO SERÁ TOTALMENTE DESCONTADA.