MC 302EF 2. Sem2016 – Primeira Prova (*)

(*) simulada, é claro

1– OO na TV: num documentário que assisti há alguns dias sobre restauração de máquinas antigas, o tema era a restauração de uma máquina de vender refrigerantes na qual o coletor de moedas estava completamente inutilizado. A máquina era antiga e já não havia peças de reposição para a mesma no mercado. O responsável encontrou um coletor de moedas de outra máquina usada para vender cigarros que acabou servindo. Isso aconteceu porque esta se encaixou perfeitamente no lugar da antiga (a furação para os parafusos e os acionadores estavam quase na mesma posição), e os controles (coletar moedas, liberar o produto e devolver o troco) eram os mesmos. Explique os conceitos de orientação a objetos envolvidos na situação acima.

R: Há muito tempo a indústria de máquinas tira proveito da modularidade nos seus projetos: uma máquina de vender refrigerantes é construída a partir de módulos como 'fonte de alimentação', 'dispensador de garrafas/latas', 'mostrador', 'coletor de moedas', etc. Cada uma dessas partes é complexa a ponto de constituir um projeto por si só. É portanto natural que por uma questão de redução de custos ocorra o reuso desses projetos. Na área de desenvolvimento de software não foi diferente: os primeiros sistemas de apoio à programação já previam o uso de funções e bibliotecas. Na história contada no enunciado, aparecem claramente os três conceitos fundamentais da programação orientada a objetos:

- encapsulamento: o coletor de moedas, tanto da máquina de refrigerantes quanto da máquina de vender cigarros encapsulam todos os detalhes da sua construção (nesse caso, numa 'casca' de metal) expondo apenas os elementos que interagem com as demais partes do veículo (esses elementos constituem a 'interface pública' do coletor de moedas).
- polimorfismo: no projeto da máquina de refrigerantes, qualquer objeto que implemente uma interface compatível com o coletor de moedas para a máquina foi projetada pode ser usada no lugar deste.
- herança: olhando um pouco além do contexto da história apresentada, o coletor de moedas da máquina de vender cigarros poderia ser usado em qualquer outra máquina cujo projeto se baseie na mesma interface. Podemos pensar em 'máquina de venda' como a classe mãe da qual derivam classes como 'máquina de vender refrigerantes', '.. cigarros', '.. salgadinhos', '.. adesivos contra o governo', etc.

(e se ainda existir curiosidade: 'o inferno é exotérmico' ?).

2 – Considere o trecho de código mostrado abaixo.

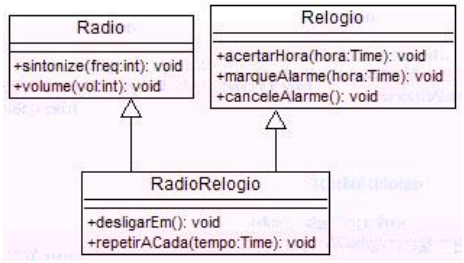
```
...
grafo.verifica(vertice.max(), vertice.min(), vertice.diag());
...
```

Quais as condições a serem atendidas pela variável 'vertice' para que o método 'verifica()' possa ser chamado (em Java) ?

R: O método grafo.verifica() tem 3 parâmetros, cujos tipos T_1, T_2 e T_3 podem ser
- tipos pré-definidos Java
- classes Java

Os tipos retornados pelos métodos max(), min() e diag(), definidos pela classe do objeto vértice, devem ser 'compatíveis para atribuição', respectivamente com T_1, T_2 e T_3 segundo as regras da linguagem.

3 -Considere o diagrama de classes abaixo.



Escreva o código Java correspondente a essas classes (não se preocupe com o 'corpo' dos métodos):

R: Como Java não permite herança múltipla, um rádio relógio será representada por um objeto de uma classe que implementa as interfaces que definem as operações oferecidas por um rádio e por um relógio.

```
interface Radio{
  void sintonize(int freq.);
  void volume(int vol);
  void liga();
  void desliga();
}
Interface Relogio{
  void acertaHora(Time hora);
  void marcaAlarme(Time hora);
  void cancelaAlarme();
  void liga();
  void desliga();
}
public class RadioRelogio{
  void sintonize(int freq){ ... }
  void volume(int vol) { ... }
  void acertaHora(Time hora) { ... }
  void marcaAlarme(Time hora) { ... }
  void cancelaAlarme(){ ... }
  void liga(){ ... }      // atende às 2 interfaces
  void desliga(){ ... }  // atende às 2 interfaces
}
```

4 -Considere a classe Integral, mostrada no quadro abaixo. Sem utilizar a palavra 'extends' de Java, mostre como você usaria essa classe para calcular a integral das funções f(x) = x*x + 2x e f(x) = x*x*x/100.0

```
class Integral {
        static final int N = 1000000;
        double f(double x) { return x; }
        double calc(double x1, double x2){
                double d = (x2 - x1)/N;
                double res = 0;
                for(int i = 0; i < N; i++) { res += f(x1)*d; x1 += d; }
                return res;
        }
}
```

R: como o enunciado proíbe o uso de 'extends', a saída é usar uma classe anônima (*anonymous inner class*), derivada de integral, na qual o método f() é redefinido a cada situação de uso, conforme mostrado no quadro abaixo:

```
...
Integral s1 = new Integral() {
   float f(float x) { return x*x + 2*x; }
};
System.out.println("v:"+s1.calc(-10.0f, 10.0f));
...
Integral s2 = new Integral() {
   float f(float x) { return x*x*x / 100.0f; }
};
System.out.println("v:"+s2.calc(-10.0f, 10.0f));
```

5 – Escreva em Java o código necessário para que o comando a seguir escreva os n primeiros números primos:

```
// neste exemplo os valores impressos seriam 2 3 5 7 11 13
for (int k: primes(5)){ System.out.print(k+' '); }
System.out.println();
```

R: O método primes() deve retornar uma estrutura que possa ser usada nessa forma simplificada do comando for. Dentre as alternativas possíveis estão as estruturas que implementam a interface Collection, como por exemplo LinkedList, ArrayList e HashSet. Essas estruturas foram projetadas para conter objetos e o exemplo supõe uma estrutura contendo inteiros (int). A alternativa que resta é o uso de um vetor de inteiros, que também pode ser utilizado nesse tipo de comando for. Isso é mostrado no exemplo abaixo.

```
 static boolean isprime(int x){
    if(x == 2) return true;
    if((x % 2) == 0) return false;
    int y = 3;
    while(y*y <= x){
        if((x%y) == 0) return false;
        y += 2;
    }
    return true;
 }

static int[] primes(int n){
       int[] res = new int[n];
       int i = 2;
       for(int k = 0; k < n; k++){
              while (! isprime(i)) i++;
              res[k] = i++;
       }
       return res;
}
```