Prova 3

Algoritmos e Programação Orientada a Objetos II — 2016

Instruções para a realização da prova:

1. A prova contém **2 questões práticas**, totalizando **10 pontos**;
2. A prova é **individual** e **sem consulta**;
3. **Não altere a disposição do mobiliário da prova**;
4. **Coloque seu celular** em cima do gabinete, desligado. Caso o professor ouça o celular vibrando ou tocando durante a aula, o aluno receberá **nota zero**;
5. **Faça a prova em silêncio**; não converse durante a prova;
6. **Não tente plagiar a prova do(a) seu(sua) colega**, você pode prejudicar você e seu(sua) colega;
7. **Não é necessário verificar a entrada**; isto é, se seu programa solicita que o(a) usuário(a) informe um número inteiro e o(a) usuário(a) informa uma letra ou qualquer outra coisa diferente de um número, seu programa pode ter qualquer comportamento inesperado;
8. Dentro da pasta *Documentos*, **crie uma pasta com seu nome completo** e coloque todos os arquivos da sua prova dentro dessa pasta.
9. Utilize o editor de texto ou IDE de sua preferência;
10. **Utilize apenas o que foi ensinado em sala de aula**. O uso de qualquer estrutura de programação ou estrutura de dados que não foi ensinada em sala de aula anulará a sua questão.

Questões

1. (2.5) Uma empresa quer manter o registro da vida acadêmica de todos os funcionários. O modelo deve contemplar o registro das seguintes informações:

   - Para o funcionário que não estudou, apenas o nome e o código funcional;
   - Para o funcionário que concluiu o ensino médio, a escola;
   - Para o funcionário que concluiu a graduação, a universidade;
   - Para o funcionário que concluiu a pós-graduação, o título da dissertação de mestrado.

   Além disso, todo funcionário possui uma renda básica de R$1.000,00 e:

   - Com a conclusão do ensino médio, a renda total de um funcionário é a renda básica;
   - Com a conclusão da graduação, a renda total de um funcionário é a renda do nível anterior acrescida de 50%;
   - Com a conclusão da pós-graduação, a renda total de um funcionário é a renda do nível anterior acrescida de 100%;

   Todos os funcionários da empresa possuem pelo menos o ensino médio. Crie um programa que simule uma empresa com 10 funcionários, sendo que 4 possuem ensino médio, 4 possuem graduação e 2 possuem pós-graduação. Mostre os custos da empresa com salários totais e por nível de escolaridade, e imprima um relatório contendo todas as informações de cada funcionário. Em todos os momentos que algum objeto concreto for manipulado, você deve utilizar conceitos de polimorfismo.

2. (7.5) Como visto em sala de aula, a interface genérica *Map* (disponível na biblioteca *java.util*) tem como objetivo disponibilizar um conjunto de métodos que permitem manipular um objeto que mapeia **chaves** em **valores**. Um *map* não pode conter chaves duplicadas e cada chave pode ser mapeada para e no máximo um valor.

   A última página desta prova contém um exemplo de uso de uma das implementações da *Map*: a classe *HashMap*. Nesse exemplo, um objeto *HashMap* é utilizado para contar a quantidade de *tokens* digitados por um usuário, utilizando um espaço como separador.

   O seu objetivo neste exercício é fornecer duas implementações para uma versão minimalista da interface *Map* do Java.

   (a) (1.5) Implemente uma interface genérica chamada *NumberMap<K, V>*. Essa interface tem o mesmo objetivo da interface *Map*, e modela o mapeamento de uma *chave* de tipo K em um *valor* de tipo V. Essa interface deve fornecer a assinatura de um subconjunto de métodos *Map*, que são:

   - `V get(Object key)`

     Retorna o valor para qual a chave especificada está mapeada. Caso não haja um mapeamento para essa chave, lance uma exceção verificada chamada *NullMappingException*;

   - `void put(K key, V value)`

     Associa a chave *key* com o valor *value*. Caso esse mapeamento já exista, ele deve sobrescrito.

   - `V remove(Object key)`

     Remove o mapeamento que tenha a chave *key*. Caso esse mapeamento não exista, lance uma exceção verificada chamada *UnknownMappingException*;

   - `Set<K> keySet()`

     Retorna um *Set* contendo as chaves do *Map*;

   - `List<V> sortedValues()`

     Retorna um *List* ordenado contendo as chaves do *Map*. Para ordenar os valores, utilize o método *sort* da classe *Collections*.

   A interface *NumberMap* deve permitir que apenas valores numéricos (que estendam da classe *Number*) sejam utilizados como chaves.

(b) (5.0) Implemente uma classe chamada *ArrayNumberMap<K, V>* que implementa a interface *NumberMap<K, V>*. Essa classe deve implementar o conceito de mapeamento de chaves em valores utilizando vetores.

P.S.: o foco deste exercício não está no desempenho. Há grandes chances de sua implementação ficar muito mais ineficiente que as implementações nativas do Java

(c) (1.0) Implemente uma classe chamada *TesteNumberMap* que deve solicitar ao usuário, da maneira que você achar mais conveniente, um conjunto de mapeamentos $\{chave \rightarrow valor\}$. O usuário deve informar, de alguma maneira, que não deseja informar mais entradas, e logo após o seu programa deve mostrar os mapeamentos $\{chave \rightarrow valor\}$ criados. Nesta implementação, utilize as classes criadas juntamente com conceitos de polimorfismo.

```
// Exemplo de uso da interface Map
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;

public class WordTypeCount {
    public static void main (String[] args) {
        Map<String,Integer> myMap = new
        HashMap<String,Integer>();

        // Contagem dos tokens
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String[] tokens = input.split(" ");
        for(String token:tokens) {
            String word = token.toLowerCase();

            if(map.containsKey(word)){
                int count = map.get(word);
                map.put(word, count+1);
            }
            else
                map.put(word, 1);
        }

        // Impressão da contagem
        Set<String> keys = map.keySet();
        TreeSet<String> sortedKeys = new
        TreeSet<String>(keys);

        for(String key:sortedKeys)
             System.out.printf("%s -> %s\n", key, map.get(key));
    }
}
```