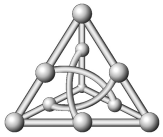


UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

Prova 2

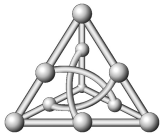
Linguagem de Programação Orientada a Objetos — Turma P01 — 2020/02

Instruções para a realização da prova:

1. A prova é **INDIVIDUAL** e contém **1 questão**, totalizando **10 pontos**;
2. Após finalizar a prova, compacte todos os arquivos e pastas em um único arquivo com extensão **.ZIP** e **envie via AVA** na tarefa Prova Optativa;
3. **Não tente plagiar a prova do(a) seu(sua) colega**, você pode prejudicar você e seu(sua) colega;
4. **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. Note que isso não significa que você não deve aplicar os princípios de encapsulamento;
5. **Utilize apenas o que foi ensinado em aula**. O uso de qualquer estrutura de programação ou estrutura de dados que não foi ensinada em sala de aula anulará a sua questão.
6. A prova tem duração de **24 horas**. Fiquem tranquilos que, a princípio, essa avaliação foi **dimensionada para 4 horas**. As horas a mais correspondem a um tempo extra para evitar qualquer tipo de contratempo de envio, bem como para permitir que vocês a realizem com mais tranquilidade;
7. O AVA ficará aberto para recebimento da avaliação até às 17h25 do dia 18/12/20. Após esse horário, **NÃO SERÁ POSSÍVEL ENVIAR ARQUIVOS**. Portanto, se programe para enviar os arquivos com pelo menos 15 minutos de antecedência;
8. Você pode submeter a prova quantas vezes achar necessário, apenas o último envio contará;
9. Qualquer dúvida em relação à prova deve ser escrita no grupo público do WhatsApp da turma, cujo link é o seguinte:
   https://chat.whatsapp.com/LuVDvCXKSM4CkGA8l5RaMM
10. Apenas mensagens públicas direcionadas a esse grupo serão respondidas durante a prova. Ou seja, o professor não responderá e-mails ou mensagens privadas durante o horário da prova.

Boa prova!

"O amor é paciente, o amor é bondoso. Não inveja, não se vangloria, não se orgulha. Não maltrata, não procura seus interesses, não se ira facilmente, não guarda rancor. O amor não se alegra com a injustiça, mas se alegra com a verdade. Tudo sofre, tudo crê, tudo espera, tudo suporta."1 Coríntios 13:4-7.



Questões

1. **[10,00 pontos]** Por conta de uma série de fatores, infelizmente não tivemos tempo hábil para trabalhar com orientação a objetos aplicada em algumas estruturas de dados específicas, como por exemplo os *dicionários*. Enquanto pensava em uma questão para a Prova Optativa da disciplina, pensei: porque não aproveitarmos esse último momento da disciplina para ao menos termos uma noção do que se tratam essas estruturas? *Voilà!*

   O objetivo dessa questão é implementar uma importante estrutura de dados utilizando a orientação a objetos: os *dicionários*, também conhecidos como *mapas*. Um *dicionário* é uma estrutura de dados com um objetivo bastante simples: mapear uma *chave* em um *valor*, sendo que *chave* e *valor* são de tipos pré-estabelecidos.

   Por exemplo, suponha que você está construindo um sistema para a receita federal cujo objetivo é receber como entrada o cpf de um contribuinte e retornar o seu nome. É possível modelar a principal estrutura de dados desse problema através de um *dicionário*, onde a chave seria o *cpf* do contribuinte (*String*) e o valor seria o nome do contribuinte (*String*). Utilizando esse dicionário, você poderia adicionar um novo mapeamento (*cpf->nome*), procurar um mapeamento (a partir de um *cpf*) e remover um mapeamento (a partir de um *cpf*). Note que os *cpfs* são únicos, ou seja, só pode existir um mapeamento para um *cpf*. Esse comportamento reflete a principal característica de um *dicionário*: as chaves são únicas.

   A página seguinte contém um código que demonstra um exemplo de inserção, remoção e busca de mapeamentos em um dicionário.

```
00. Dicionario dic = new Dicionario();

01. dic.adicionaMapeamento("737.044.800-63", "Samuel Ferraz");
02. dic.adicionaMapeamento("123.123.123-12", "Raul Adler");
03. dic.adicionaMapeamento("234.234.234-23", "Iago Akio");
04. dic.adicionaMapeamento("345.345.345-34", "Julia Alves");

05. if(dic.existeMapeamento("123.123.123-12")) {
06.        print("Esse cpf existe!");
07.        print("Dono do cpf: " + dic.retornaValor("123.123.123-12");
08. }

09. dic.adicionaMapeamento("234.234.234-23", "Raul Adler");

10. if(dic.existeMapeamento("234.234.234-23")) {
11.        print("Esse cpf existe!");
12.        print("Dono do cpf: " + dic.retornaValor("234.234.234-23");
13. }

14. dic.removeMapeamento("345.345.345-34");
15. print(dic.retornaValor("345.345.345-34"));

16. String chaves[] = dic.retornaChaves();
17. for(int i = 0 ; i < dic.retornaChaves() ; i++) {
18.       print(dic.retornaValor(chaves[i]));
19. }
```

Algumas considerações importante sobre o código acima:

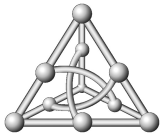
(a) As linhas 1 -> 4 adicionam mapeamentos no dicionário;

(b) As linhas 5 -> 8 e 10 -> 13 procuram mapeamentos já adicionados anteriormente. Ou seja, a busca de um mapeamento é feita apenas com base na *chave*, que no caso desse exemplo, é o *cpf*, mas que em termos gerais é uma *String* qualquer;

(c) A linha 9 sobrescreve um mapeamento já existente. Ou seja, o cpf "234.234.234-23"deixou de ser mapeado no "Iago Akio"e, a partir dessa linha, passou a ser mapeado no "Raul Adler". Note que já existia uma pessoa com o nome "Raul Adler"na listagem de mapeamentos, mas isso não é um problema. O que não pode estar duplicado são as chaves, mas os valores podem se repetir. Pensando em termos práticos, é possível que existam duas pessoas com o mesmo nome (no caso do exemplo, dois "Raul Adler") com cpfs diferentes;

(d) A linha 14 removeu o mapeamento "345.345.345-34". Ou seja, quando a linha 15 tenta retornar o mapeamento desse cpf, será retornado *null*.

(e) As linhas 16 -> 19 imprimem todos os mapeamentos *chave->valor* existentes no dicionário criado.



O objetivo desse exercício é implementar uma classe *Dicionário* capaz de criar, remover e exibir mapeamentos *chave -> valor*, onde *chave* sempre será do tipo *String* e *valor* sempre será do tipo *String*. Sua classe deverá, pelo menos, implementar todos os métodos exibidos no exemplo fornecido anteriormente. Não se esqueça de usar os princípios de encapsulamento nessa classe.

Crie também um programa principal que forneça um menu com as seguintes opções:

(a) Cadastrar um novo mapeamento;

(b) Buscar um mapeamento;

(c) Remover um mapeamento;

(d) Listar todos os mapeamentos.

Dica: você pode utilizar vetores na classe Dicionário para modelar as chaves e os valores.

BOA PROVA MEU POVO!
Foi um prazer estar com vocês nesse semestre.
Deixo aqui uma célebre frase de um poeta Japonês para reflexão.

```
Com o seu sorriso, por favor
Faz me moer, moer, moer, moer
Vem cá, meu bem
Ou eu vou aí do seu lado, é?
```

Y-No, 2010.