Prova Optativa

Algoritmos e Programação Orientada a Objetos II — 2016

Instruções para a realização da prova:

1. A prova contém **3 questões práticas**, totalizando **12 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. (4.00) Escreva uma classe *Balanceada*, que contém apenas um método *main* que recebe do teclado uma expressão matemática parentizada e responde se a expressão está balanceada ou não. Uma expressão aritmética está balanceada se todos os parênteses abertos foram fechados corretamente.

   Para resolver esse problema, implemente e use uma classe *Pilha*, utilizando o método de alocação de sua preferência (desde que sua *Pilha* não tenha restrições quanto à quantidade de elementos armazenados).

2. (4.00) O objetivo dessa questão é analisar a sua capacidade de abstrair uma situação do mundo real com os conceitos de orientação a objetos. Para isso, considere o contexto descrito a seguir:

   Os impostos dos Brasil são arrecadados com base no tipo de contribuinte: pessoa física *ou* pessoa jurídica*. Independente do tipo de pessoa (física ou jurídica), todos são contribuintes do imposto de renda e devem ter as seguintes informações armazenadas: nome e renda bruta. Além disso, todo contribuinte deve ter a capacidade de calcular imposto, mas a maneira como esse imposto é calculado depende do tipo de contribuinte. Para pessoa jurídica, o imposto deve corresponder a 10% da renda bruta da empresa. Já para pessoa física, o imposto deve ser calculado de acordo com a seguinte tabela:*

Renda Bruta	**Alíquota**	**Parcela a Deduzir**
R$0,00 a R$ 1.400,00	0%	R$0,00
R$1.400,01 a R$ 2.100,00	10%	R$100,00
R$2.100,01 a R$ 2.800,00	15%	R$270,00
R$2.800,01 a R$ 3.600,00	25%	R$500,00
R$3.600,01 ou mais	30%	R$500,00

   A coluna Parcela a Deduzir *representa o valor que deve ser subtraído do imposto após o cálculo realizado com a alíquota.*

   Você deve armazenar, em alguma classe, o valor total de impostos que já foram calculados de todos os contribuintes.

   Modele o problema acima utilizando o máximo de conceitos de orientação a objetos vistos no decorrer da disciplina. Logo após, crie uma classe chamada *TesteImposto*, que testa todas as classes criadas utilizando polimorfismo.

3. (4.00) 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.

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

   (a) (0.50) 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 eles:

   - `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.

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

   (b) (2.50) 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) (0.50) 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.