Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Curso de Bacharelado em Ciência da Computação
Disciplina: Fundamentos de Programação - Professor: Roland Teodorowitsch
1 de setembro de 2018

Lista de Exercícios - Unidade 8: Objetos e Classes

1. Implemente uma classe `Carro` com as propriedades descritas a seguir. Um carro apresenta certo consumo de combustível (medido em quilômetros por litro) e tem certa quantidade de combustível no seu tanque de gasolina. O consumo é especificado no construtor, e o nível inicial de combustível é 0. Forneça um método `dirigir()` que simula o uso do carro até determinada distância, reduzindo o nível de gasolina no tanque, e métodos `obterNivelCombustivel()`, para obter o nível atual de combustível, e `abastecer()` para adicionar combustível ao carro. Exemplo de uso:

```
Carro meuCarro = new Carro(10.0);    // 10 km por litro
meuCarro.abastecer(40.0);            // Abastece 40 litros
meuCarro.dirigir(100);               // Dirige o carro por 100 km
System.out.println(meuCarro.obterNivelCombustivel()); // Combustivel restante
```

Adaptado de: Horstmann (2013, p. 406-407)

2. Implemente uma classe chamada `Estudante` que armazene o nome e controle a nota dos alunos em uma série de avaliações. Crie o construtor apropriado e os métodos `obtenhaNome()`, `definaNome()`, `adicioneNota()`, `obtenhaTotalNotas()`, `obtenhaNumNotas()` e `obtenhaMediaNotas()`. Os nomes dos métodos são autoexplicativos, portanto, declare variáveis paramétricas adequadas para cada método e também variáveis de instância suficientes para implementar o comportamento esperado em cada método. Não tente armazenar cada uma das notas adicionadas aos objetos desta classe. Sugestão: implemente também as classes `toString()` e `print()` para esta classe.

Adaptado de: Horstmann (2013, p. 407)

3. Implemente uma classe chamada `TestaEstudante` que faz a verificação (teste unitário) da classe `Estudante` implementada na questão anterior. Todos os construtores e métodos implementados devem ser invocados de forma que seja possível verificar o seu correto funcionamento.

Autor: Roland Teodorowitsch (11 nov. 2016)

4. Ainda com a classe `Estudante`, implemente uma classe `TurmaDeEstudantes` que leia de um arquivo chamado `turma.dados` os dados de um conjunto de alunos e armazene-os em um vetor de objetos da classe `Estudante`. A primeira linha do arquivo `turma.dados` contém o número de estudantes e cada uma das linhas restantes do arquivo contém os dados dos estudantes, cada um em uma linha. As linhas com os dados dos estudantes contém o nome e as notas obtidas pelo estudante, usando ponto-e-vírgula como separador. Este arquivo poderia conter, por exemplo:

```
5
Claudio;9.0;8.0;7.0
Janaina;8.0;9.0;10.0
Augusto;10.0;9.0;8.0
Fernanda;5.0;6.0;7.0
Paulo;4.0;6.0;8.0
```

   Depois de realizar a leitura e o processamento das notas dos alunos usando os métodos da classe `Estudante`, seu programa deverá:

   - Calcular a média da turma;
   - Encontrar a nota mais alta;
   - Encontrar a nota mais baixa;
   - Ordenar a turma pelo nome;
   - Imprimir o vetor de estudantes ordenado, bem como os dados calculados e procurados nos itens anteriores.

Autor: Roland Teodorowitsch (11 nov. 2016)

5. Implemente uma classe para armazenar o CPF (Cadastro de Pessoa Física) de uma pessoa. Esta classe deverá aceitar CPFs válidos nos formatos "DDD.DDD.DDD-DD"ou "DDDDDDDDDDD"(onde D é um dígito de zero a nove). Nenhum outro caractere ou formato deverá ser aceito. Sua classe deverá ser formada por:

   - construtor que recebe um *string* com o CPF, validando-o e armazenando-o;
   - método `void define(String cpf)` que define o novo conteúdo para objetos da classe CPF, validando-o antes do armazenamento;
   - método `String obtem(boolean format)` que retorna o CPF formatado (`format` igual a `true`) ou não (`format` igual a `false`);
   - método `boolean valida(String cpf)` que recebe um CPF em qualquer um dos dois formatos e verifica se trata-se de um CPF válido ou não (este método pode ser público e estático, não acessando nenhuma variável de instância, e podendo ser usado tanto pelos outros métodos quanto por outras aplicações).

   O construtor e o método `void define(String cpf)`, depois de usarem o método `boolean valida(String cpf)`, devem lançar a exceção `IllegalArgumentException`, caso não se trate de um CPF válido.

Autor: Roland Teodorowitsch (11 nov. 2016)

6. Implemente uma classe em Java chamada ContaCorrente. Esta classe deverá ser capaz de controlar o saldo de diversas contas-correntes, armazenando: agência (String), número da conta-corrente (String), nome do titular (String), CPF do titula (em um objeto da classe CPF definida na questão anterior) e saldo (double). Para esta classe implemente os seguintes métodos:

   - para construir objetos desta classe;
   - para obter cada uma das variáveis de instância de objetos desta classe;
   - para definir cada uma das variáveis de instância de objetos desta classe;
   - `saque()` que debita determinado valor do saldo da conta-corrente;
   - `deposito()` que credita determinado valor no saldo da conta-corrente;
   - `toString()` que gera uma cadeia de caracteres com os dados da conta-corrente;
   - `print()` que exibe todos os dados da conta-corrente.

Autor: Roland Teodorowitsch (11 nov. 2016)

7. Implemente uma classe chamada `TestaContaCorrente` que faz a verificação (teste unitário) da classe `ContaCorrente` implementada na questão anterior. Todos os construtores e métodos implementados devem ser invocados de forma que seja possível verificar o seu correto funcionamento.

Autor: Roland Teodorowitsch (11 nov. 2016)

8. Escreva em Java a classe `NumeroComplexo` que represente um número complexo. A classe deverá ter os seguintes métodos:

   - `inicializaNumero`, que recebe dois valores como argumentos para inicializar os campos da classe (parte real e imaginária);
   - `imprimeNumero`, que deve imprimir o número complexo encapsulado usando a notação a + bi onde a é a parte real e b a imaginária;
   - `eIgual`, que recebe outra instância da classe `NumeroComplexo` e retorna `true` se os valores dos campos encapsulados forem iguais aos da instância passada como argumento;
   - `soma`, que recebe outra instância da classe `NumeroComplexo` e soma este número complexo com o encapsulado usando a fórmula $(a + bi) + (c + di) = (a + c) + (b + d)i$;
   - `subtrai`, que recebe outra instância da classe `NumeroComplexo` e subtrai o argumento do número complexo encapsulado usando a fórmula $(a + bi) - (c + di) = (a - c) + (b - d)i$;
   - `multiplica`, que recebe outra instância da classe `NumeroComplexo` e multiplica este número complexo com o encapsulado usando a fórmula $(a + bi).(c + di) = (ac - bd) + (ad + bc)i$;
   - `divide`, que recebe outra instância da classe NumeroComplexo e divide o número encapsulado pelo passado como argumento usando a fórmula $\frac{(a+bi)}{(c+di)} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$.

Fonte: Santos (2013)

REFERÊNCIAS

HORSTMANN, C. **Java for Everyone – Late Objetct**. 2. ed. Hoboken: Wiley, 2013. xxxiv, 589 p.
SANTOS, Rafael. **Introdução à Programação Orientada a Objetos usando Java**. 2. ed. 2013.