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
(GABARITO)

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)
Resposta:

```
public class Carro {
    private double consumo;     // km/l
    private double nivelTanque; // l

    public Carro(double consumo) {
        this.consumo = consumo;
        this.nivelTanque = 0.0;
    }

    public void abastecer(double litros) {
        this.nivelTanque += litros;
    }

    public void dirigir(double km) {
        this.nivelTanque -= km/this.consumo;
    }

    public double obterNivelCombustivel() {
        return this.nivelTanque;
    }

    public String toString() {
        return this.consumo+":"+this.nivelTanque;
    }

    public void print() {
        System.out.println(this.toString());
    }
}
```

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)

Resposta:

```java
public class Estudante {
    private String nome;
    private double total;
    private int numero;
    private double media;

    public Estudante() {
        this.nome = "";
        this.total = 0.0;
        this.numero = 0;
        this.media = 0.0;
    }

    public Estudante(String nome) {
        this.nome = nome;
        this.total = 0.0;
        this.numero = 0;
        this.media = 0.0;
    }

    public String obtenhaNome() {
        return this.nome;
    }

    public void definaNome(String nome) {
        this.nome = nome;
    }

    public void adicioneNota(double nota) {
        if (nota>=0.0 && nota<=10.0) {
            this.total += nota;
            this.numero++;
            this.media = this.total / this.numero;
        }
    }

    public double obtenhaTotalNotas() {
        return this.total;
    }

    public double obtenhaNumNotas() {
        return this.numero;
    }

    public double obtenhaMediaNotas() {
        return this.media;
    }

    public String toString() {
        return this.nome+":"+this.total+":"+this.numero+":"+this.media;
    }

    public void print() {
        System.out.println(this.toString());
    }

}
```

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)

Resposta:

```
public class TestaEstudante {
    public static void main(String[] args) {
        Estudante e = new Estudante();
        e.definaNome("Israel");
        e.print();
        e.adicioneNota(10.0);
        e.print();
        e.adicioneNota(9.0);
        e.print();
        e.adicioneNota(8.0);
        e.print();

        Estudante joaquim = new Estudante("Joaquim");
        joaquim.adicioneNota(7.0);
        joaquim.adicioneNota(8.0);
        joaquim.adicioneNota(9.0);
        joaquim.adicioneNota(10.0);
        System.out.println("Nome        = "+joaquim.obtenhaNome());
        System.out.println("Total Notas = "+joaquim.obtenhaTotalNotas());
        System.out.println("Num. Notas  = "+joaquim.obtenhaNumNotas());
        System.out.println("Media       = "+joaquim.obtenhaMediaNotas());
    }
}
```

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)

Resposta:

```java
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class TurmaDeEstudantes {

    public static void main(String[] args) throws FileNotFoundException {
        Scanner fturma = new Scanner(new File("turma.dados"));
        int numEstudantes = Integer.parseInt(fturma.nextLine());
        Estudante[] turma = new Estudante[numEstudantes];

        // LEITURA
        // Para cada estudante no arquivo...
        for (int i=0;i<numEstudantes;++i) {
            // Le cada uma das linhas com informacoes de cada aluno
            String linha = fturma.nextLine();
            // Separa a linha em campos
            String[] campos = linha.split(";");
            // Cria o objeto Estudante usando o construtor que receb o nome
            turma[i] = new Estudante(campos[0]);
            // As notas estao em campos[1], campos[2], campos[3], ...
            for (int j=1;j<campos.length;++j) {
                turma[i].adicioneNota(Double.parseDouble(campos[j]));
            }
        }
        fturma.close();

        // PROCESSAMENTO
        double maior = turma[0].obtenhaMediaNotas();
        double menor = turma[0].obtenhaMediaNotas();
        double media = turma[0].obtenhaMediaNotas();
        for (int i=1;i<numEstudantes;++i) {
            if (turma[i].obtenhaMediaNotas()>maior)
                maior = turma[i].obtenhaMediaNotas();
            if (turma[i].obtenhaMediaNotas()<menor)
                menor = turma[i].obtenhaMediaNotas();
            media = media + turma[i].obtenhaMediaNotas();
        }
        media = media / numEstudantes;
        System.out.println("MEDIA = "+media);
        System.out.println("MAIOR = "+maior);
        System.out.println("MENOR = "+menor);

        // ORDENACAO
        for (int i=0;i<numEstudantes-1;++i) {
            int menorNome = i;
            for (int j=i+1;j<numEstudantes;++j) {
                if (turma[menorNome].obtenhaNome().compareTo(turma[j].obtenhaNome())>0)
                    menorNome = j;
            }
            if (menorNome != i) {
                Estudante aux = turma[menorNome];
                turma[menorNome] = turma[i];
                turma[i] = aux;
            }
        }

        // IMPRESSAO
        for (int i=0;i<numEstudantes;++i)
            turma[i].print();
    }

}
```

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)

Resposta:

```
public class CPF {
    String cpf;

    public CPF(String cpf) throws IllegalArgumentException {
        if (CPF.valida(cpf)) {
            if (cpf.length()==11)
                this.cpf = cpf;
            else
                this.cpf = cpf.substring(0,3)+cpf.substring(4,7)+cpf.substring(8,11)+cpf.substring(12);
        }
        else
            throw  new IllegalArgumentException("CPF invalido!");
    }

    public void define(String cpf) throws IllegalArgumentException {
        if (CPF.valida(cpf)) {
            if (cpf.length()==11)
                this.cpf = cpf;
            else
                this.cpf = cpf.substring(0,3)+cpf.substring(4,7)+cpf.substring(8,11)+cpf.substring(12);
        }
        else
            throw  new IllegalArgumentException("CPF invalido!");
    }

    public String obtem(boolean format) {
        if (format)
            return this.cpf.substring(0,3)+"."+this.cpf.substring(3,6)+"."+this.cpf.substring(6,9)+"-"+this.cpf.substring(9);
        else
            return this.cpf;
    }

    public static boolean valida(String cpf) {
        int[] digitos = new int[11];
        switch (cpf.length()) {
            case 11:
                 for (int i=0;i<11;++i) {
                     char c = cpf.charAt(i);
                     if (!Character.isDigit(c))
                        return false;
                     digitos[i] = c-'0';
                 }
                 break;
            case 14:
                 int j = 0;
                 for (int i=0;i<14;++i) {
                     char c = cpf.charAt(i);
                     switch (i) {
                         case 3:
                         case 7:
                              if (c != '.')
                                 return false;
                              break;
                         case 11:
                              if (c != '-')
                                 return false;
                              break;
                         default:
                              if (!Character.isDigit(c))
                                 return false;
                              digitos[j++] = c-'0';
                     }
                 }
                 break;
            default:
                 return false;
        }
        int mult1 = 10;
        int dv1 = 0;
        int mult2 = 11;
        int dv2 = 0;
        for (int i=0;i<9;++i) {
            dv1 += digitos[i] * mult1--;
            dv2 += digitos[i] * mult2--;
        }
        dv1 = ( dv1 * 10 ) % 11;
        if (dv1 > 9)
           dv1 = 0;
        dv2 += dv1 * mult2;
        dv2 = ( dv2 * 10 ) % 11;
        if (dv2 > 9)
           dv2 = 0;
        if (dv1 != digitos[9] || dv2 != digitos[10])
           return false;
        return true;
    }

}
```

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)

Resposta:

```
public class ContaCorrente {
    private String agencia;
    private String cc;
    private String titular;
    private CPF cpf;
    private double saldo;

    public ContaCorrente() {
        this.agencia = "";
        this.cc = "";
        this.titular = "";
        this.cpf = null;
        this.saldo = 0.0;
    }

    public ContaCorrente(String ag,String cc,String tit,CPF cpf,double saldo) {
        this.agencia = ag;
        this.cc = cc;
        this.titular = tit;
        this.cpf = cpf;
        this.saldo = saldo;
    }

    public String obtemAgencia() {
        return this.agencia;
    }

    public String obtemCc() {
        return this.cc;
    }
    public String obtemTitular() {
        return this.titular;
    }
    public CPF obtemCpf() {
        return this.cpf;
    }
    public double obtemSaldo() {
        return this.saldo;
    }

    public void defineAgencia(String agencia) {
        this.agencia = agencia;
    }

    public void defineCc(String cc) {
        this.cc = cc;
    }

    public void defineTitular(String titular) {
        this.titular = titular;
    }

    public void defineCpf(CPF cpf) {
        this.cpf = cpf;
    }

    public void defineSaldo(double saldo) {
        this.saldo = saldo;
    }

    public void saque(double valor) {
        this.saldo -= valor;
    }

    public void deposito(double valor) {
        this.saldo += valor;
    }

    public String toString() {
        return "Ag="+this.agencia+";Cc="+this.cc+";Titular="+this.titular+";CPF="+this.cpf.obtem(true)+";Sado="+this.saldo;
    }

    public void print() {
        System.out.println(this.toString());
    }

}
```

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)

Resposta:

```
public class TestaContaCorrente {
    public static void main(String[] args) {
        ContaCorrente cc1 = new ContaCorrente();
        cc1.defineAgencia("1111-1");
        cc1.defineCc("22222-2");
        cc1.defineTitular("Carlos da Silveira");
        cc1.defineCpf(new CPF("242.061.480-15"));
        cc1.defineSaldo(0.00);
        cc1.deposito(1000.00);
        cc1.saque(200.00);
        cc1.print();

        ContaCorrente cc2 = new ContaCorrente("3333-3","44444-4","Fulano de Tal",new CPF("663.254.220-40"),500.00);
        System.out.println(cc2.toString());
        cc2.deposito(100.00);
        cc2.saque(400.00);
        System.out.println("AGENCIA="+cc2.obtemAgencia());
        System.out.println("CC     ="+cc2.obtemCc());
        System.out.println("TITULAR="+cc2.obtemTitular());
        System.out.println("CPF    ="+cc2.obtemCpf().obtem(true));
        System.out.println("SALDO  ="+cc2.obtemSaldo());
    }
}
```

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)
Resposta:

```java
public class NumeroComplexo {
    private double real;
    private double imag;

    public NumeroComplexo(double real, double imag) {
        this.real = real;
        this.imag = imag;
    }

    public void inicializaNumero(double real, double imag) {
        this.real = real;
        this.imag = imag;
    }

    public void imprimeNumero() {
        System.out.printf("%f + %f . i\n",real,imag);
    }

    public double parteReal() {
        return real;
    }

    public double parteImaginaria() {
        return imag;
    }

    public boolean eIgual(NumeroComplexo nc) {
        if (real == nc.parteReal() && imag == nc.parteImaginaria())
            return true;
        return false;
    }

    public void soma(NumeroComplexo nc) {
        real += nc.parteReal();
        imag += nc.parteImaginaria();
    }

    public void subtrai(NumeroComplexo nc) {
        real -= nc.parteReal();
        imag -= nc.parteImaginaria();
    }

    public void multiplica(NumeroComplexo nc) {
        real = real * nc.parteReal() - imag * nc.parteImaginaria();
        imag = real * nc.parteImaginaria() + imag * nc.parteReal();
    }

    public void divide(NumeroComplexo nc) throws ArithmeticException {
        double denominador = Math.pow(nc.parteReal(),2)+Math.pow(nc.parteImaginaria(),2);
        if (denominador==0)
            throw new ArithmeticException("Divisao por 0");
        real = (real * nc.parteReal() + imag * nc.parteImaginaria()) / denominador;
        imag = (imag * nc.parteReal() - real * nc.parteImaginaria()) / denominador;
    }

}
```


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.