UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

Lista de Exercícios: Teste de Unidade com JUnit

Prof. André Takeshi Endo

(Exercício 1) Considere a classe abaixo:

```
import java.util.Random;

public class Aleatorio {
    /** O metodo deve receber dois inteiros que representam o inicio e o fim de um intervalo e
    * retornar um numero aleatorio que se encontra dentro do intervalo estabelecido,
    * ou seja, [inicio, fim]. Caso o inicio do intervalo ou o fim do intervalo sejam
    * menor que zero, o metodo deve retornar -1. O metodo tambem retorna -1 quando o inteiro
    * representando o inicio do intervalo for maior ou igual ao inteiro representando o
    * fim do intervalo.  **/
    public int gerarNumeroAleatorio(int inicio, int fim) {
        if (inicio < 0 || fim < 0) {
            return -1;
        }

        if (inicio >= fim) {
            return -1;
        }
        int diff = fim - inicio + 1;
        Random random = new Random();

        int tInt = random.nextInt(diff); //esse método retorna um número aleatorio
                                         //entre 0 e diff [ 0, diff [
        return inicio + tInt;
    }
}
```

Implemente casos de teste em JUnit para o método "gerarNumeroAleatorio(..)" da classe anterior que verifique as seguintes situações:

- o início do intervalo é um valor negativo;
- o fim do intervalo é um valor negativo;
- o início do intervalo é igual ao fim do intervalo; e
- um intervalo válido [200, 3000] é fornecido.

(Exercício 2) Considere as classes a seguir; as mesmas não podem ser alteradas.

Utilitario.java

```
public class Utilitario {
    /**
     * @param v
     * @return um objeto da classe Extremos que guarda
     * o maior e o menor numero no vetor e seus indices
     */
    public Extremos acharExtremos(int v[]) throws Exception {
        if(v == null)
            throw new Exception("vetor nao pode ser nulo");
```

Extremos.java

```
public class Extremos {
    int menor, maior, indiceMenor, indiceMaior;

    public Extremos(int menor, int indiceMenor,
                    int maior, int indiceMaior) {
        this.menor = menor;
        this.maior = maior;
        this.indiceMenor = indiceMenor;
        this.indiceMaior = indiceMaior;
```

```
        if(v.length == 0)
            throw new Exception("vetor com zero elementos");

        int menor = v[0], maior = v[0];
        int indiceMenor = 0, indiceMaior = 0;

        for (int i = 1; i < v.length; i++) {
            if(v[i] < menor) {
                menor = v[i];
                indiceMenor = i;
            }
            if(v[i] > maior) {
                maior = v[i];
                indiceMaior = i;
            }
        }
                return new Extremos(menor, indiceMenor, maior,
indiceMaior);
    }
}
```

```
    }
    public int getMenor() {
        return menor;
    }
    public int getMaior() {
        return maior;
    }
    public int getIndiceMenor() {
        return indiceMenor;
    }
    public int getIndiceMaior() {
        return indiceMaior;
    }
}
```

Implemente quatro casos de teste em JUnit para o método “acharExtremos(...)” da classe Utilitario que verifique as seguintes entradas: (i) um vetor {1, 2, 3, 4, 5, 6}; (ii) um vetor {1, 99, 3, -5, 8}; (iii) um vetor vazio; (iv) um vetor nulo.

(Exercício 3) Considere as três classes a seguir; as mesmas não podem ser alteradas.

IMCCalculadora.java

```
public class IMCCalculadora {

    public IMCStatus calcular(Pessoa p) {
        double peso = p.getPeso();
        double altura = p.getAltura();

        if(peso <= 0 || altura <= 0)
            throw new IllegalArgumentException();

        double imc = peso / (altura * altura);
        String classificacao = "";
        if(imc < 18.5)
            classificacao = "abaixo do peso";
        else if(imc < 25)
            classificacao = "normal";
        else if(imc < 30)
            classificacao = "acima do peso";
        else
            classificacao = "obeso";

        return new IMCStatus(imc, classificacao);
    }
}
```

Pessoa.java

```
public class Pessoa {
    String nome;
    double peso, altura;

    public Pessoa(String nome, double peso, double altura) {
        this.nome = nome;
        this.peso = peso;
        this.altura = altura;
    }

    //incluir getters
}
```

IMCStatus.java

```
public class IMCStatus {
    double imc;
    String classificacao;

    public IMCStatus(double imc, String classificacao) {
        this.imc = imc;
        this.classificacao = classificacao;
    }

    //incluir getters
}
```

Implemente quatro casos de teste em JUnit para o método “calcular(...)” da classe IMCCalculadora que verifique as seguintes entradas: o lançamento da exceção e as 4 possíveis classificações. Não esqueça de verificar se o IMC foi calculado corretamente.

(Exercício 4) Considere as classes abaixo:

Classificador.java

```
public class Classificador {
   /** Metodo retorna em qual faixa etaria esta a pessoa;
       lança RuntimeException quando a idade eh invalida */
   public String definirFaixaEtaria(Pessoa p)
                                        throws RuntimeException {
      if(p.getIdade()<0 || p.getIdade() >=110)
         throw new IllegalArgumentException("idade invalida");

      int idade = p.getIdade();
      String tipo = "";
      if(idade <= 11)
         tipo = "crianca";
      else if(idade <= 18)
         tipo = "adolescente";
      else if(idade <= 59)
         tipo = "adulto";
      else
         tipo = "idoso";

      return p.getNome()+" eh "+ tipo;
   }
}
```

Pessoa.java

```
public class Pessoa {
   private String nome;
   private int idade;

   public Pessoa(String pNome, int pIdade) {
      nome = pNome;
      idade = pIdade;
   }
   public int getIdade() {
      return idade;
   }
   public String getNome() {
      return nome;
   }
}
```

Implemente casos de teste em JUnit para o método “definirFaixaEtaria(...)” da classe Classificador que verifique as seguintes situações: (i) um valor de idade inválido; (ii) uma pessoa que é criança; (iii) uma pessoa que é adolescente; (iv) uma pessoa que é adulta; e (ii) uma pessoa que é idosa.

(Exercício 5) Considere as três classes a seguir; as mesmas não podem ser alteradas.

Analisador.java

```
public class Analisador {
   public Estatisticas
         analisar(ArrayList<Candidato> candidatos)
            throws Exception {
      if(candidatos == null || candidatos.isEmpty())
         throw new Exception("lista nula ou vazia");

      int fem = 0, masc = 0;
      float idadeMedia = 0;
      for (Candidato c : candidatos) {
         if(c.getSexo() == 'F')
            fem++;
         else  masc++;

         idadeMedia += c.getIdade();
      }

      String contexto = "normal";
      if(candidatos.size() == 1)
         contexto = "sem concorrencia";
      else if(candidatos.size() == 2)
         contexto = "polarizada";

      return new Estatisticas(fem, masc,
                  idadeMedia / candidatos.size(), contexto);
   }
}
```

Candidato.java

```
public class Candidato {
   char sexo;
   int idade;

   public Candidato(char sexo, int idade) {
      this.sexo = sexo;
      this.idade = idade;
   }
   public char getSexo() {  return sexo;  }

   public int getIdade() {  return idade;  }
}
```

Estatisticas.java

```
public class Estatisticas {
   int mulheres, homens;
   float idadeMedia;
   String contexto;

   public Estatisticas(int m, int h, float i, String c) {
      this.mulheres = m;
      this.homens = h;
      this.idadeMedia = i;
      this.contexto = c;
   }
   // getters
}
```

Implemente cinco casos de teste em JUnit para o método “analisar(...)” da classe Analisador que verifique as seguintes entradas: o lançamento da exceção, os 3 possíveis contextos e um cenário em que há mais candidatos mulheres do que homens. Não esqueça de verificar se todos os campos das Estatisticas foram calculados corretamente.

*****OS EXERCÍCIOS A SEGUIR NÃO FORNECEM A IMPLEMENTAÇÃO DA CLASSE QUE DEVE SER TESTADA. ASSIM, O ALUNO É LIVRE PARA FAZER O DESIGN DAS CLASSES.**

(Exercício 6) Triângulo. Especifique um conjunto de casos de teste para testar o programa a seguir:
O programa lê três valores inteiros que representam os lados de um triângulo. O programa informa se os lados formam um triângulo isósceles, escaleno ou equilatero. Condição: a soma de dois lados tem que ser maior que o terceiro lado.

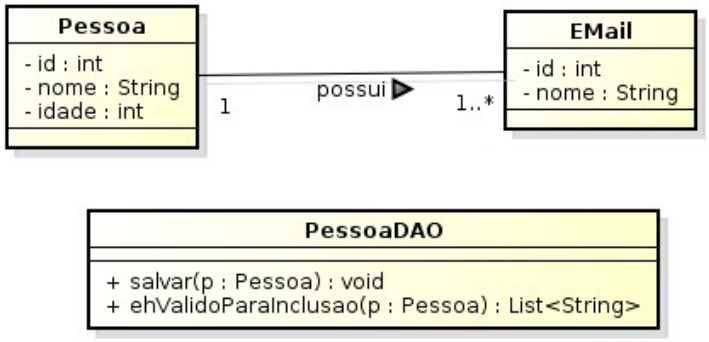
(i) Defina o esqueleto de uma classe Java que resolva o problema acima.

(ii) Escreva casos de teste em JUnit para as seguintes situações:

- Triângulo escaleno válido
- Triângulo isósceles válido
- Triângulo equilatero válido
- Pelo menos 3 casos de teste (CTs) para isósceles válido contendo a permutação dos mesmos valores
- Um valor zero
- Um valor negativo
- A soma de 2 lados é igual ao terceiro lado
- Para o item acima, um CT para cada permutação de valores
- CT em que a soma de 2 lados é menor que o terceiro lado
- Para o item acima, um CT para cada permutação de valores
- Um CT para os três valores iguais a zero

(Exercício 7) Considere o conjunto de classes abaixo e implemente o método “ehValidoParaInclusao()”. Esse método deve retornar uma lista de erros com base no objeto Pessoa passado como parâmetro. Deve ser validado:

- O nome é composto por ao menos 2 partes e deve ser composto de letras
- A idade deve estar no intervalo [1,200]
- A pessoa deve ter pelo menos um objeto da classe e-mail associado
- O e-mail deve estar no formato “___@___.___”, sendo que cada parte deve ter ao menos um caractere



(Exercício 8) Crie uma classe entidade Contato com os atributos: nome, password, e-mail, telefone, idade, peso. Crie uma classe chamada ContatoDAO com um método "salvar()" que recebe como parâmetro um objeto da classe Contato. O método "salvar()" deve lançar exceções, considerando que:

1. Os campos nome, password e e-mail são obrigatórios.
2. A idade se não for vazia (ou seja, igual a zero) deve estar entre 5 e 99 anos.
3. O peso se não for vazio (ou seja, igual a zero) deve estar entre 1 e 200 kg.

Crie uma exceção para cada um dos possíveis 3 erros, lembrando que todas devem ser do tipo "*checked exceptions*". A lógica do método "salvar()" deve ser implementada. Crie casos de teste em JUnit para verificar cada um dos possíveis erros.

(Exercício 9) Escreva uma implementação para a estrutura de dados pilha; tal pilha deve ser capaz de empilhar e desempilhar strings. Considere que:

- Tal pilha possui um tamanho limitado (que deve ser passado como parâmetro no construtor).
- Adicione métodos para empilhar, desempilhar e verificar se a pilha está vazia.
- Crie duas classes de exceção que devem ser do tipo "*checked exception*": PilhaVaziaException e PilhaCheiaException.
  - PilhaVaziaException deve ser lançada caso tente desempilhar a pilha sem elementos.
  - PilhaCheiaException deve ser lançada caso tente empilhar um elemento na pilha cheia.
- Implemente casos de teste em JUnit que ilustrem a utilização dos métodos e as possíveis exceções lançadas.