UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
INSTITUTO METRÓPOLE DIGITAL
DISCIPLINA: TESTE DE SOFTWARE I

LISTA DE EXERCÍCIOS

1. [ROBERTA] Construa uma classe chamada Triangulo ela deve ter: (a) um construtor que recebe os 3 lados de um triângulo, e lança a exceção NaoTrianguloException caso os 3 lados nao formem um triângulo; (b) 3 métodos que não recebem parâmetros e retornam um booleano e que diz se o triângulo é isósceles, escaleno ou equilátero; (c) uma classe de testes que checa o comportamento dos métodos desta classe. Quando criar os testes lembre-se de testar o lançamento das exceções.

2. [ROBERTA] Construa classes de testes JUnit para a classe abaixo:

```
public class SmartWatch {

    /* Se a quantidade de batimentos por minuto for superior a 100 e se a
    distância percorrida for maior que 10km o método retorna a string
    "ALTO_RISCO", caso contrário retorna "NAO_HA_RISCO" ou a
    Exceção ArgumentoInvalido se algum dos valores for negativo */

     public String calcRiscoOverTraining (int bpm, double distancia)
                                        throws   ArgumentoInvalidoException {
        ….
     }

}
```

3. [JOSÉ VITOR F. CUNHA] De acordo com a classe Filme abaixo:

```java
public class Filme {
    String titulo;
 double duracaoMin;
String diretor;

 public Filme(String ti, double du, String di) throws FilmeNaoValido {
        if (du < 45 || ti.isBlank() || di.isBlank()) {
            throw new FilmeNaoValido();
} else {
            titulo = ti;
duracaoMin = du;
diretor = di;
}
    }

    public String getFilmeMenorDuracao(Filme filme2) throws FilmeNaoValido {
    }

    public boolean isMesmoDiretor(Filme filme2) throws FilmeNaoValido {
    }
```

Implemente:

a) 1 método que retorna o filme com menor duração entre os dois e lança uma exceção FilmeNaoValidoException caso sejam filmes iguais;

b) 1 método que retorna true caso os filmes forem do mesmo diretor e false caso contrário, além disso lança uma exceção FilmeNaoValidoException caso sejam iguais;

c) Por último, crie uma classe FilmeTest e crie testes de unidade para checar os cenários.

12. [FRANK] Escreva testes parametrizados para o exemplo abaixo.

```
public class Average {

    private Average() {
    // Non default constructor
    }

    public static double semester(int num1, int num2, int num3) {
```

```
        return (num1 + num2 + num3) / 3;
        }

    }
```

13. [LETICIA] Considere a classe abaixo.

```java
public class AlunoAprovado {

public static String calculaMediaFinal(int primeiraNota, int segundaNota) {
      int mediaAprovado = 7;
      int mediaRecuperacao = 5;

      int media = (primeiraNota + segundaNota) / 2;

      String situacao = "Aprovado";

      if (media >= mediaAprovado) {
            return situacao;
      } else if (media < mediaAprovado && media >= mediaRecuperacao){
            situacao = "Recuperação";
            return situacao;
      } else {
            situacao = "Reprovado";
            return situacao;
      }
   }
}
```

Refatore a classe para lançar uma exceção MediaInvalidaException caso uma das notas seja negativa. b. Após o reparo, realize testes parametrizados utilizando JUnit, e que trabalhe as 3 situações existentes retornadas pela String situacao.

14. [LETICIA] Construa uma classe chamada IngressoNaUFRNPeloSisu que deve conter:

a) Construtor com 7 valores que indique o valor das seguintes situações:
   - Ano do último enem realizado;
   - Nota tirada na área de Linguagens;
   - Nota tirada na área de Matemática;

- Nota tirada na área de Humanas;
- Nota tirada na área de Natureza;
- Nota tirada na prova de redação;
- Área do curso que pretende ingressar.

b) Lance uma exceção InscricaoIrregularException caso o ano do último enem realizado não seja o ano de 2024.

c) De acordo com a imagem abaixo, crie um método booleano caso o estudante esteja apto a tentativa de ingresso na área que pretende ingressar.

Área	Linguagens		Matemática		C. Humanas		C. Natureza		Redação	
	Peso	Corte	Peso	Corte	Peso	Corte	Peso	Corte	Peso	Corte
Biomédica	1,5	450	1,0	450	1,5	450	3,0	450	1,5	500
Humanística I	2,0	450	2,0	450	2,0	450	1,0	450	1,5	500
Humanística II	2,5	450	1,0	450	2,5	450	1,0	450	1,5	500
Tecnológica I	1,0	450	2,0	450	2,0	450	2,0	450	1,5	500
Tecnológica II	1,0	450	3,0	450	1,0	450	2,0	450	1,5	500

d) Criar uma classe de testes parametrizados que verifique os métodos desta classe, inclusive o lançamento das exceções quando lançadas.

16. [GABRIEL MARTINS] Sobre o Testes JUnit responda:

a) Qual é a diferença entre o uso de assertThrows() e @Test (expected=...) utilizados no JUnit4 e JUnit5 respectivamente?

b) Como eles podem ser utilizados para testar exceções em métodos de uma classe? Explique usando exemplos.

c) Quais são os principais benefícios de utilizar testes unitários com JUnit em um projeto de software? Além disso, explique como esses testes podem ajudar a ***garantir*** a qualidade do código e a ***facilitar*** a manutenção do sistema.

17. [THIAGO JOSÉ] A respeito do JUnit, analise as afirmativas abaixo:

I - Na versão 4 do JUnit, quando se utiliza o método assertEquals() do JUnit para comparar duas variáveis do tipo double, é possível passar um terceiro parâmetro que corresponde ao delta, que corresponde à diferença máxima que será tolerada entre os dois números comparados.

II - Um dos métodos pertencentes ao framework JUnit é o método assertObject(), que compara quaisquer duas variáveis do tipo Object.

III - A anotação @Before pode ser associada a um método de testes JUnit e garante que este será o primeiro método de teste a ser executado.

IV - A versão 4 do JUnit oferece o método assertThat(), que traz maior flexibilidade às comparações que podem ser realizadas no corpo de um método de testes.

Estão corretas as afirmativas
A) II e III.
B) I e III.
C) I e IV.
D) II e IV.

OBS: O assertThat é uma forma de comparação de valores que utiliza o conceito de matchers. Este assert foi removido no JUnit5, e para os casos onde este tipo de comparação mais complexa necessário o JUnit indica que sejam utilizadas libs de terceiros como AssertJ, Hamcrest, Truth, etc. https://junit.org/junit5/docs/snapshot/user-guide/

18. [THIAGO JOSÉ] Considere a classe Java abaixo:

```
public class Prova {
    static int vetor(int n, int vet[]) {
        int val;
        val = vet[0];
        for (int j = 1; j < n; j += 1) {
            if (val < vet[j]) {
                val = vet[j];
            }
        }
        return val;
    }
}
```

Para a classe Prova, foi criada uma classe de teste utilizando o JUnit, contendo o método de teste abaixo:

```
@Test
public void testVetor() {
    int[] vet = {89, 90, 84, 91};
    int r = Prova.vetor(vet.length, vet);
    I ;
}
```

Para que o teste seja aprovado, a lacuna I deve ser corretamente preenchida com

A) assertTrue(91, r)

B) assertValue(84, r)

C) assertEquals(91, r)

D) assertEquals(90, r, true)

E) assertEquals(84, r)

19. [CLAUBER] No Código a seguir, porque o camando "buffRead.close()" está no bloco do finally? O que aconteceria se ele estivesse fora do bloco try/catch?

static void readFile(String path, Charset encoding) throws IOException
{

```
  BufferedReader buffRead = new BufferedReader(new FileReader(path));
  try {
    String linha = "";
    while (true) {
      if (linha != null) {
        System.out.println(linha);
      } else
        break;
     linha = buffRead.readLine();
     }
  }catch(IOException ex) {
    System.out.println("Erro de Entrada e saida");
    throw ex;
  }catch(Throwable ex) {
    throw ex;
  }finally {
    buffRead.close();
  }
}
```

21. [GUILHERME EULLER] Considere a classe para calcular a média ponderada abaixo:

```
import java.util.List;

public class CalculaMediaPonderada {

public static double calculaMediaPonderada( List<Double> valores, List<Integer> pesos ) throws Exception {

        if(valores.size() != pesos.size()) {
                throw new Exception();
        }
        double resultado = 0;

        for(int i = 0; i < valores.size() ;i++)
```

```
            resultado += valores.get(i)*pesos.get(i);

        double somaPesos = 0;

        for(int i = 0; i < pesos.size() ;i++)
            somaPesos += pesos.get(i);

        return resultado/somaPesos;
}                                                   }
```

A função calculaMediaPonderada recebe uma lista de valores e uma lista de pesos, crie testes parametrizados utilizando JUnit4 para essa classe.

22. [RAQUEL] (Senado federal - 2022) Você foi contratado para liderar uma equipe de DevOps. Um dos objetivos da sua liderança é aumentar a velocidade das entregas e a qualidade de novos recursos das aplicações utilizando o desenvolvimento orientado a testes.

Assinale a opção que indica a ordem que descreve corretamente o ciclo de desenvolvimento orientado a testes.

A) Refatorar - > Escrever um código funcional
B) Escrever um caso de teste -> Refatorar
C) Refatorar - > Escrever um código funcional - > Escrever um caso de teste
D) Escrever um caso de teste -> Escrever um código funcional -> Refatorar
E) Escrever um código funcional -> Escrever um caso de teste -> Refatorar

24. [GABRIEL ROCHA] Considere a seguinte classe ConversorMoeda que realiza conversões entre diferentes moedas:

```
public class ConversorMoeda {

    private static final double TAXA_CONVERSAO_USD_BRL = 5.20;
    private static final double TAXA_CONVERSAO_EUR_BRL = 6.15;

    public static double converterUSDBRL(double valor) {
```

```
            if (valor < 0) {
                throw new IllegalArgumentException("O valor não pode ser
            negativo!");
             }

            return valor * TAXA_CONVERSAO_USD_BRL;
        }

        public static double converterEURBRL(double valor) {
            if (valor < 0) {
                throw new IllegalArgumentException("O valor não pode ser
            negativo!");
            }
            return valor * TAXA_CONVERSAO_EUR_BRL;
        }

}
```

Implemente testes de unidade para a classe ConversorMoeda usando JUnit, com os seguintes requisitos:

- Teste a conversão de um valor zero.
- Teste a conversão de um valor negativo, onde uma exceção IllegalArgumentException é esperada.
- Use o método assertEquals do JUnit para verificar os resultados esperados.

25. [ANA MARIA] Considere as seguintes afirmativas a respeito do framework JUnit e marque V ou F para cada uma delas.

( ) I. Até a versão 3.8.1 do JUnit, todas as classes de testes precisavam herdar da classe TestCase do framework JUnit.

( ) II. A partir da versão 4 do JUnit, para se construir uma classe de teste, precisa-se apenas associar a anotação @Test à declaração de qualquer classe pública.

( ) III. Com a anotação @Test(timeout=), é possível definir o tempo de duração do teste em milissegundos. Se a execução ultrapassar o tempo definido, o teste irá acusar a falha.

( ) IV. Para que um determinado objeto seja compartilhado entre vários métodos de testes JUnit, deve-se colocar a inicialização do objeto no construtor da classe.

26. [ILANA DUARTE] Considerando as classes TvShow e TvShowServiceImpl abaixo, crite testes de unidade para a função (buildTvShowFromInput) que constrói o objeto (TvShow) de uma entrada (string) (ex: `"BR" "Jornal Cultura" 19:00 00:00`).

```
public class TvShow {

	private String region;
	private String name;
	private LocalTime beginning;
	private LocalTime end;

	public TvShow(String region, String name, LocalTime beginning, LocalTime end) {
		super();
		 this.region = region;
		 this.name = name;
		 this.beginning = beginning;
		 this.end = end;
	}
}



public class TvShowServiceImpl {

	public TvShow buildTvShowFromInput(String input) {
		 String lines[] = input.split("\"");
		 TvShow tvShow = new TvShow(lines[1],lines[3]);
		 lines = input.split("");
		 tvShow.setBeginning(LocalTime.parse(lines[lines.length-2]));
		 tvShow.setEnd(LocalTime.parse(lines[lines.length-1]));
		 return tvShow;
	}
}
```

27. [ISAAC LOPES] Considere a seguinte classe EquivalenciaModular que contém o método calcularEquivalencia que retorna **true** se dois números são equivalentes mod, e **false** se não forem equivalentes.

```
public class EquivalenciaModular {

		public static boolean CalcularEquivalencia(int a, int b, int m){
			return (a % m) == (b % m);
		}

}
```

Crie testes de unidade para verificar o funcionamento do método calcularEquivalencia.

28. [JOSE FILHO] Complete o teste MonstroTestem JUnit para verificar se o método "receberGolpe" da classe "Monstro" está diminuindo corretamente a vida do monstro.

Classe Monstro:

```
public class Monstro {
		private int vida;
		public Monstro(int vida) {
			this.vida = vida;
		}
		public void receberGolpe(int forcaGolpe) {
			this.vida -= forcaGolpe;
			if (vida <= 0)
				throw new MorteException();
		}

		public void antidoto(int restaura) {
			this.vida += restaura;
		}

		public int getVida() { return vida; }
```

}

O teste o comportamento do Monstro.

Exemplo de estrutura de um teste:

```
import org.junit.Test;
import static org.junit.Assert.*;
public class MonstroTest
{

@Test
public void testReceberGolpe() { }

}
```