Exercício prático 1

Aluno: Luiz Henrique Oliveira de Assis

Questão 1

Encontre a média dos valores de uma lista.
Nome dado a classe: Media.java

```java
public double calculaMedia(ArrayList<Integer> listaInteiros) {

        double totalValorList = 0;
        int tamanhoList = listaInteiros.size();

        for (int i = 0; i < tamanhoList; i++) {

            totalValorList += listaInteiros.get(i);

        }

        return totalValorList / tamanhoList;

}
```

O caso de teste 2 testa se a List recebida e vazia ou não, retornando "true" caso seja vazia e "false" caso não seja.

```java
    // caso de teste 1
    @Test
    void testArrayVazio() {

        ArrayList<Integer> arrayInteiros = new ArrayList<>();
        assertTrue(arrayInteiros.isEmpty());

    }
```

O caso de teste 2 testa se a média do maior número inteiro possível está correta.

```java
    // caso de teste 2
    @Test
    void testValorMaximo() {
        ArrayList<Integer> arrayInteiros = new ArrayList<>();
        arrayInteiros.add(Integer.MAX_VALUE);

        assertEquals(Integer.MAX_VALUE, media.calculaMedia(arrayInteiros));
    }
```

O caso de teste 3 testa se a média de números positivos está correta.

```java
    // caso de teste 3
    @Test
    void testMediaPositiva() {

        ArrayList<Integer> arrayInteiros = new ArrayList<>();
        arrayInteiros.add(2);
        arrayInteiros.add(2);
        arrayInteiros.add(2);

        assertEquals(2, media.calculaMedia(arrayInteiros));

    }
```

O caso de teste 4 testa se a média de números negativo está correta.

```
    // caso de teste 4
    @Test
    void testMediaNegativa() {

        ArrayList<Integer> arrayInteiros = new ArrayList<>();
        arrayInteiros.add(-2);
        arrayInteiros.add(-2);
        arrayInteiros.add(-2);
        assertEquals(-2, media.calculaMedia(arrayInteiros));

    }
```

Questão 2

Verifique se uma lista está ordenada.
Nome dado a classe: ListaOrdenada.java

```
public class ListaOrdenada {
    public boolean estaOrdenada(ArrayList<Integer> listaEntrada){

        boolean ordenacao = false;

        if(listaEntrada.stream().sorted().toList().equals(listaEntrada)){
            ordenacao = true;
        }

        return ordenacao;
    }

}
```

O caso de teste 1 testa como a classe ira comportar recebendo uma List com o menor valor possível e maior valor possível ordenados.

```
//caso de teste 1
@Test
void testValorMaximoValorMinimo(){

        ArrayList<Integer> listInt = new ArrayList<>();
        listInt.add(Integer.MIN_VALUE);
        listInt.add(Integer.MAX_VALUE);

        assertEquals(true, listaOrdenada.estaOrdenada(listInt));

}
```

O caso de teste 2 testa como a classe ira comportar recebendo uma List vazia.

```
//caso de teste 2
@Test
void testListVazia() {

        ArrayList<Integer> listInt = new ArrayList<>();

        assertEquals(true, listInt.isEmpty());

}
```

O caso de teste 3 testa como a classe ira comportar recebendo uma List ordenada com valor negativo.

```
//caso de teste 3
@Test
void testEstaOrdenada() {
```

```
94
95          ArrayList<Integer> listInt = new ArrayList<>();
96          listInt.add(-11);
97          listInt.add(1);
98          listInt.add(1);
99          listInt.add(2);
100         listInt.add(3);
101
102         assertEquals(true, listaOrdenada.estaOrdenada(listInt));
103
104 }
```

O caso de teste 4 testa como a classe ira comportar recebendo uma List com valores não ordenados.

```
105 //caso de teste 4
106 @Test
107 void naoEstaOrdenada() {
108
109         ArrayList<Integer> listInt = new ArrayList<>();
110         listInt.add(1);
111         listInt.add(3);
112         listInt.add(2);
113
114         assertNotEquals(true, listaOrdenada.estaOrdenada(listInt));
115
116 }
```

Questão 3

Calcule o fatorial de um número.
Nome dado a classe: Fatorial.java

```
117 public class Fatorial {
118
119     public int calcularFatorial(int x) {
120
121         if (x <= 0) {
122             return 1;
123         }
124
125         if (x == 0) {
126             return 1;
127         }
128
129         return x * calcularFatorial(x - 1);
130
131     }
132
133 }
```

O caso de teste 1 testa como a classe ira comportar recebendo um valor inteiro mínimo.

```
134 //caso de teste 1
135 @Test
136 void valorMinimo(){
137         assertEquals(1, fac.calcularFatorial(Integer.MIN_VALUE));
138 }
```

O caso de teste 2 testa como a classe ira comportar recebendo um valor inteiro valido.

```
139 //caso de teste 2
140 @Test
141 void testFatorialCorreto(){
142         assertEquals(120, fac.calcularFatorial(5));
143 }
```

O caso de teste 3 testa como a classe ira comportar recebendo um valor inteiro invalido.

```
144 //caso de teste 3
145 @Test
146 void testFatorialValoresEntradaInvalidos() {
147 
148         assertEquals(1, fac.calcularFatorial(1));
149         assertEquals(1, fac.calcularFatorial(0));
150         assertEquals(1, fac.calcularFatorial(-1));
151 
152 }
```

O caso de teste 4 testa como a classe ira comportar recebendo um valor inteiro valido mas com o resultado invalido.

```
153 //caso de teste 4
154 @Test
155 void testFatorialIncorreto(){
156         assertNotEquals(119, fac.calcularFatorial(5));
157         assertNotEquals(121, fac.calcularFatorial(5));
158 }
```

Questão 4

Crie um conversor de temperatura.
Nome dado a classe: Conversor.java

```
159 public class Conversor {
160 
161     public double converteTemperatura(double celsius){
162         return (celsius * 1.8) + 32;
163     }
164 
165 }
```

O caso de teste 1 testa como a classe ira comportar ao receber um número valido positivo.

```
166 //caso de teste 1
167 @Test
168 void testConveterTemperaturaValidaPositiva() {
169         assertEquals(32, conversor.converteTemperatura(0));
170 }
```

O caso de teste 2 testa como a classe ira comportar ao receber um número valido negativo.

```
171 //caso de teste 2
172 @Test
173 void testConveterTemperaturaValidaNegativa() {
174         assertEquals(30.2, conversor.converteTemperatura(-1));
175 }
```

O caso de teste 3 testa como a classe ira comportar ao receber um número valido positivo com um resultado errado.

```
176 //caso de teste 3
177 @Test
178 void testConverterTemperaturaErrada() {
179         assertNotEquals(121, conversor.converteTemperatura(50));
180         assertNotEquals(123, conversor.converteTemperatura(50));
181     }
```

O caso de teste 4 testa como a classe ira comportar ao receber o maior número inteiro número valido positivo e negativo.

```
182 //caso de teste 4
183 @Test
184 void testConverterTemperaturaMaximaEMinima() {
185         assertEquals(3.8654705966E9, conversor.converteTemperatura(Integer.MAX_VALUE)
            );
186         assertEquals(-3.8654705344E9, conversor.converteTemperatura(Integer.MIN_VALUE
            ));
187 }
```

Questão 5

Verifique se um número é primo.
Nome dado a classe: Primo.java

```
188 public class Primo {
189
190 public boolean ePrimo(int numero) {
191
192         if(numero <= 1){
193             return false;
194         }
195
196         for (int j = 2; j < numero; j++) {
197             if (numero % j == 0)
198                 return false;
199         }
200         return true;
201     }
202
203 }
```

O caso de teste 1 testa como a classe ira comportar ao receber o maior e menor número inteiro possível.

```
204 //caso de teste 1
205 @Test
206 void valorLimite() {
207         assertEquals(true, primo.ePrimo(Integer.MAX_VALUE));
208         assertEquals(false, primo.ePrimo(Integer.MIN_VALUE));
209 }
```

O caso de teste 2 testa como a classe ira comportar ao receber números de entrada inválidos.

```
210 //caso de teste 2
211 @Test
212 void testEntradaInvalida() {
213
214         assertEquals(false, primo.ePrimo(1));
215         assertEquals(false, primo.ePrimo(0));
216         assertEquals(false, primo.ePrimo(-1));
217
218 }
```

O caso de teste 3 testa como a classe ira comportar ao receber números de entrada validos e não primo.

```
219 //caso de teste 3
220 @Test
221 void testNaoEPrimo() {
222
223         assertEquals(false, primo.ePrimo(10));
224 }
```

O caso de teste 4 testa como a classe ira comportar ao receber números de entrada validos e primo.

```java
//caso de teste 4
@Test
void testEPrimo() {

        assertEquals(true, primo.ePrimo(5));

}
```

Questão 6

Verifique se uma lista está ordenada.
Nome dado a classe: ListaOrdenada.java

```java
public class ListaOrdenada {

    public boolean listaOrdenada(ArrayList<Double> lista) {

        boolean crescente = true;
        boolean decrescente = true;

        for (int i = 0; i < lista.size() - 1; i++) {
            if (lista.get(i) > lista.get(i+i)) {
                crescente = false;
            }
            if (lista.get(i) < lista.get(i+1)) {
                decrescente = false;
            }
        }

        return crescente || decrescente;

    }

}
```

O caso de teste 1 testa como a classe ira comportar ao receber uma lista não ordenada.

```java
//caso de teste 1
@Test
void testListaOrdenadaSemOrdem() {

        ArrayList<Double> listinha = new ArrayList<>();
        listinha.add(2D);
        listinha.add(3D);
        listinha.add(1D);

        assertEquals(false, ordena.listaOrdenada(listinha));

}
```

O caso de teste 2 testa como a classe ira comportar ao receber uma lista ordenada e crescente.

```java
//caso de teste 2
@Test
void testListaOrdenadaCrescente() {

        ArrayList<Double> listinha = new ArrayList<>();
        listinha.add(1D);
        listinha.add(2D);
        listinha.add(3D);

        assertEquals(true, ordena.listaOrdenada(listinha));

```

```
}
```

O caso de teste 3 testa como a classe ira comportar ao receber uma lista ordenada e decrescente.

```
//caso de teste 3
@Test
void testListaOrdenadaDecrescente() {

        ArrayList<Double> listinha = new ArrayList<>();
        listinha.add(3D);
        listinha.add(2D);
        listinha.add(1D);

        assertEquals(true, ordena.listaOrdenada(listinha));

}
```

O caso de teste 4 testa como a classe ira comportar ao receber uma lista ordenada e crescente com o menor número possível e maior número possível.

```
//caso de teste 4
@Test
void testListaOrdenadaValorMaximoEValorMinimo() {

        ArrayList<Double> listinha = new ArrayList<>();
        listinha.add(Double.MIN_VALUE);
        listinha.add(Double.MAX_VALUE);

        assertEquals(true, ordena.listaOrdenada(listinha));

}
```