Universidade Federal do Rio Grande do Sul
Instituto de Informática
Técnicas de Construção de Programas

Luís Eduardo Pereira Mendes
00333936

Estudo e aplicação de frameworks de teste unitário automatizado

Porto Alegre, Rio Grande do Sul, Brasil
Março de 2023

Definição da classe e dos testes

A classe `Triangle` deve ser capaz de:

- Inicializar (por meio de construtor) os 3 lados que compõem o triângulo;
- Atualizar os lados do triângulo, sempre alterando os 3 lados simultaneamente;
- Devolver verdadeiro ou falso corretamente para a pergunta “é um triângulo?”;
- Devolver uma *String* informando se o triângulo atual é Equilátero, Isósceles, Escaleno ou não é um triângulo.

Os testes serão realizados em 3 classes distintas, cada uma irá testar:

- Casos de triângulos válidos;
- Casos de triângulos inválidos por valores inválidos (menores ou iguais a zero);
- Casos de triângulos inválidos pela definição da soma.

O primeiro conjunto de testes irá testar cada um dos 3 tipos de triângulos, além da permutação dos valores para o caso do triângulo isósceles.

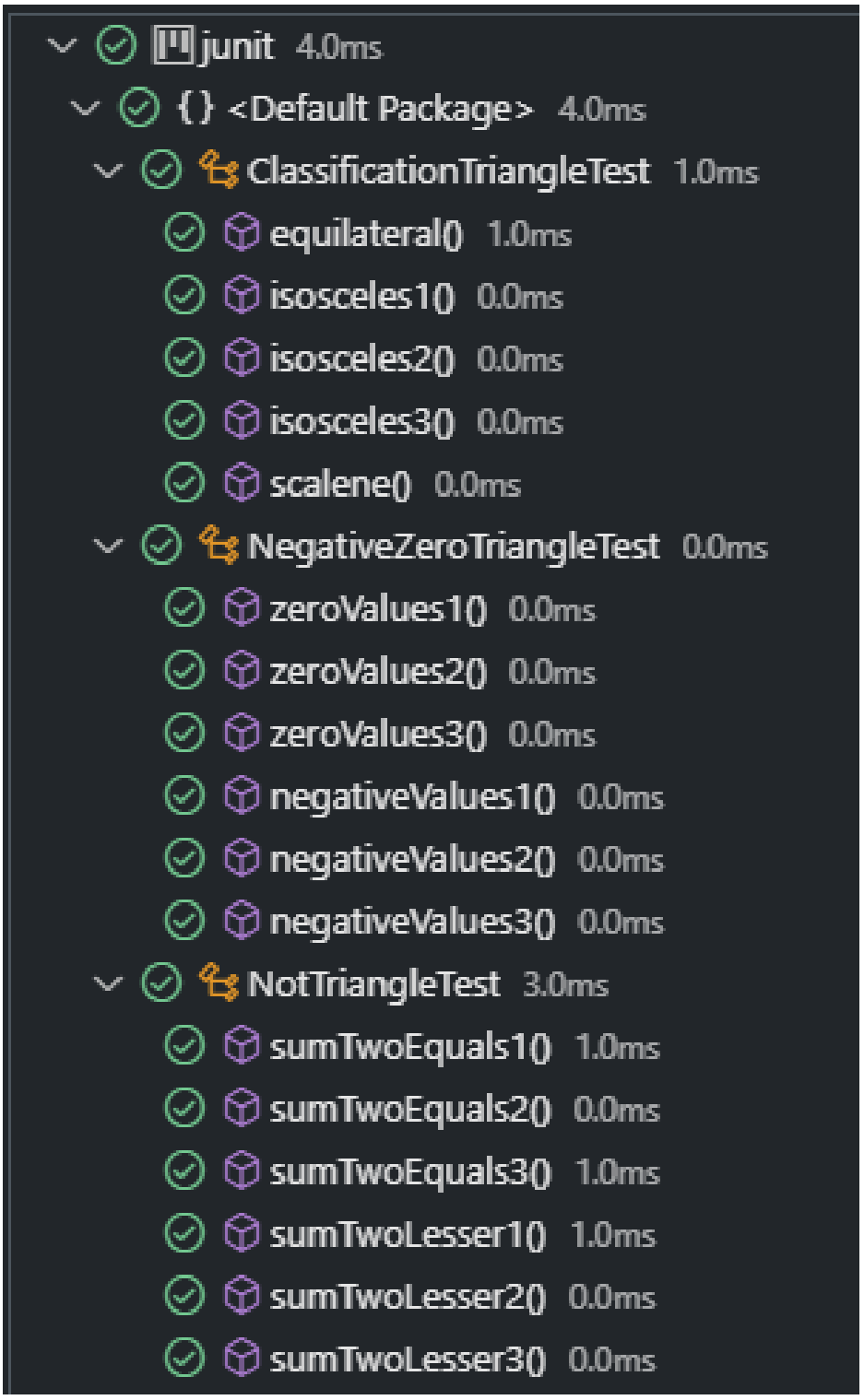
O segundo conjunto de testes irá verificar permutações de valores zero e valores negativos.

O terceiro conjunto de testes irá verificar triângulos cuja soma de dois lados é menor ou igual a um dos lados, realizando permutações entre esses valores.

Classe *Triangle*

```java
public class Triangle {
    private double a, b, c;

    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public void setEdgesLength(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public boolean isTriangle() {
        if (a <= 0 || b <= 0 || c <= 0){
            return false;
        }
        else if (a + b > c && a + c > b && b + c > a){
            return true;
        }
        else{
            return false;
        }
    }

    public String classification(){
        if (isTriangle()){
            if (a == b && a == c){
                return "Equilateral";
            } else if (a == b || a == c || b == c){
                return "Isosceles";
            } else{
                return "Scalene";
            }
        }
        else{
            return "NotTriangle";
        }
    }
}
```

Resultado geral dos testes



Testes de Classificação de triângulos válidos

```java
import org.junit.*;

public class ClassificationTriangleTest {
    Triangle triangle = new Triangle(a: 0, b: 0, c: 0);

    @Test
    public void equilateral(){
        triangle.setEdgesLength(a: 10,b: 10,c: 10);
        Assert.assertEquals(triangle.classification(), "Equilateral");
    }
    @Test
    public void isosceles1(){
        triangle.setEdgesLength(a: 5,b: 10,c: 10);
        Assert.assertEquals(triangle.classification(), "Isosceles");
    }
    @Test
    public void isosceles2(){
        triangle.setEdgesLength(a: 10,b: 5,c: 10);
        Assert.assertEquals(triangle.classification(), "Isosceles");
    }
    @Test
    public void isosceles3(){
        triangle.setEdgesLength(a: 10,b: 10,c: 5);
        Assert.assertEquals(triangle.classification(), "Isosceles");
    }
    @Test
    public void scalene(){
        triangle.setEdgesLength(a: 8,b: 10,c: 5);
        Assert.assertEquals(triangle.classification(), "Scalene");
    }


}

```

Casos de teste de valores de entrada inválidos para arestas de triângulos (negativos e zero)

```java
import org.junit.*;

public class NegativeZeroTriangleTest {
    Triangle triangle = new Triangle(a: 0, b: 0, c: 0);

    @Test
    public void zeroValues1(){
        triangle.setEdgesLength(a: 4,b: 10,c: 0);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void zeroValues2(){
        triangle.setEdgesLength(a: 4,b: 0,c: 0);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void zeroValues3(){
        triangle.setEdgesLength(a: 0,b: 0,c: 0);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void negativeValues1(){
        triangle.setEdgesLength(-4,b: 10,c: 10);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void negativeValues2(){
        triangle.setEdgesLength(a: 4,-10,c: 10);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void negativeValues3(){
        triangle.setEdgesLength(a: 4,b: 10,-10);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
}
```

Casos de teste para valores que não formam triângulos válidos

```java
import org.junit.*;

public class NotTriangleTest {
    Triangle triangle = new Triangle(a: 0, b: 0, c: 0);

    @Test
    public void sumTwoEquals1(){
        triangle.setEdgesLength(a: 10,b: 5,c: 5);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void sumTwoEquals2(){
        triangle.setEdgesLength(a: 5,b: 10,c: 5);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void sumTwoEquals3(){
        triangle.setEdgesLength(a: 5,b: 5,c: 10);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void sumTwoLesser1(){
        triangle.setEdgesLength(a: 5,b: 4,c: 10);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void sumTwoLesser2(){
        triangle.setEdgesLength(a: 4,b: 10,c: 5);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
    @Test
    public void sumTwoLesser3(){
        triangle.setEdgesLength(a: 10,b: 5,c: 4);
        Assert.assertEquals(triangle.classification(), "NotTriangle");
    }
}
```

Relatório do trabalho de desenvolvimento de testes

Durante o desenvolvimento pude perceber alguns conceitos que não tinha visto durante as leituras realizadas. Uma delas foi que, a medida que os casos de teste iam sendo desenvolvidos, se tornou necessário, para continuar entendendo cada caso de teste, separar os testes que se relacionavam entre si, como os testes de valores negativos ou iguais a zero, que deveriam ser separados, por motivos de compreensão, dos testes que verificam se a classificação de um dado triângulo tinha sido conforme o esperado. Além disso, também notei que, diferentemente da minha primeira abordagem, não valeria a pena realizar um laço de repetição com vários testes, uma vez que dificultaria a compreensão do que está sendo testado no momento que um erro foi encontrado, o que dificultaria o diagnóstico. Para o caso de ser necessário (ou desejado) um laço de repetição de uma bateria de testes, os testes ainda deveriam mostrar o estado do sistema, para tornar possível o diagnóstico de um erro, o que entraria, na maioria das vezes, em casos de testes randomizados, que foi estudado durante a disciplina e trabalhada nas leituras obrigatórias. Pude perceber também como pensar nos testes anteriormente ao desenvolvimento do código auxilia na forma que o software é desenvolvido, com base no TDD (Test Driven Development). Por fim, notei que, ao executar os testes, a ferramenta utilizada é extremamente poderosa para encontrar falhas e que, uma vez feitos os testes, estes poderiam ser reutilizados posteriormente, no caso de uma possível refatoração, a fim de assegurar a ausência de alteração no software.