Qualidade Software
Licenciatura em Engenharia Informática
1º Semestre 2022/2023

Ficha de trabalho 5

Objetivos:

- Introdução ao JUnit

Exercícios

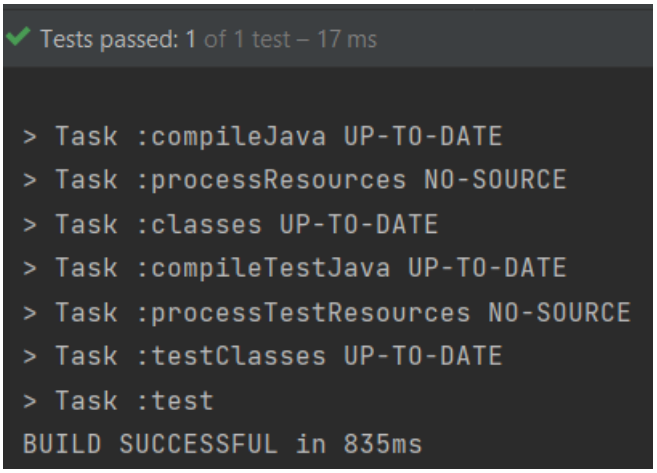
1- Primeiro exemplo de uso do JUnit para criar um método de teste

a. Cria um novo projeto Gradle. Faz o build do projeto para que no src do projeto tenhas as pastas main e test
   i. Relembra que no main tens o código desenvolvido
   ii. O test vai incluir as classes de teste que vamos desenvolver com o JUnit
   iii. Nota que por defeito ao criar o projeto Gradle, é acrescentada automaticamente a dependência JUnit (ver o ficheiro build.gradle)
b. Cria na pasta main uma nova classe chamada Calculadora
   i. Implementa um método que permita somar dois valores passados por parâmetro
c. Cria na pasta test uma nova classe chamada CalculadoraTest
   i. As classes de teste devem estar de acordo com os testes a implementar
   ii. De seguida implementa o seguinte código

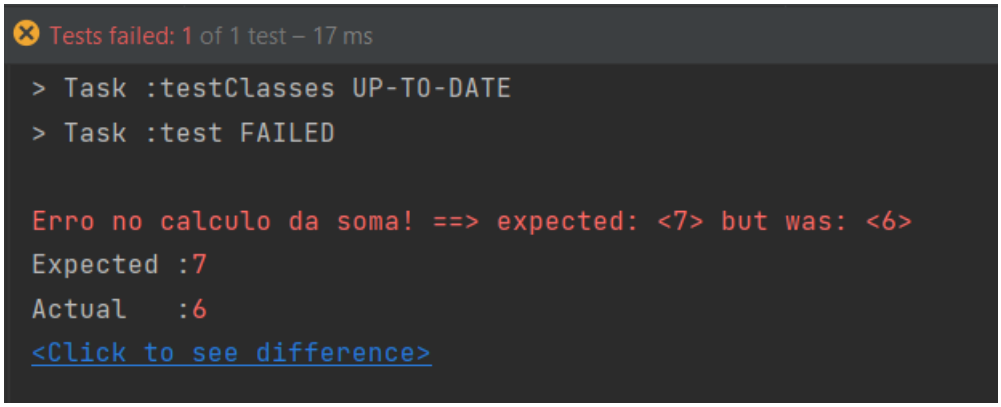
```
public class CalculadoraTest {

    @Test
    public void testSoma() {
       Calculadora calculo = new Calculadora();
       int soma = calculo.soma(2,5);
       int testSoma = 7;
       assertEquals (soma, testSoma, "Erro no calculo da
soma!");
    }
}
```

iii. Faz run do teste que implementaste. Repara esta classe não tem método main. A anotação @Test permite que o método de teste seja executável
iv. Em alternativa podes correr o teste utilizando o comando `./gradlew clean test` no terminal

Em caso de sucesso dos testes, é obtido um build successful.



v. Modifica o valor da variável testSoma para 6 forçando a falha no teste. Analisa o resultado.



2- Uso dos diferentes métodos assert.

a. A biblioteca JUnit incorpora diferentes assert como descreve a tabela seguinte. Para mais métodos assert consulta a API do JUnit.

assertion method	description
`assertEquals()`	Uses the `equals()` method to verify that objects **are identical**.
`assertTrue()`	Checks if the condition **is true**.
`assertFalse()`	Checks if the condition **is false**.
`assertNull()`	Checks if the object **is null**.
`assertNotNull()`	Checks if the object **is not null**.
`assertNotEquals()`	Uses the `equals()` method to verify that objects **are not identical**.
`assertArrayEquals()`	Checks if two arrays **contain the same objects in the same order**.
`assertSame()`	Uses `==` to verify that objects **are the same**.
`assertNotSame()`	Uses `==` to verify that objects **are not the same**.
`assertThat()`	An entry point to the **matchers**-based assertions which we will discuss in Section 6.6.

b. Cria uma nova class na pasta test para testarmos os diferentes métodos assert da biblioteca JUnit

c. Executa o seguinte código e força a falha do teste.

```
public class Ex2_Asserts {

    @Test
    public void AssertMethodstest() {
        String obj1 = "junit";
        String obj2 = "junit";
        String obj3 = "test";
        String obj4 = "test";
        String obj5 = null;
        int var1 = 1;
        int var2 = 1;
        int var3 = 5;
        int[] a1 = {1, 2, 3};
        int[] a2 = {1, 2, 3};
        assertEquals(obj1, obj2);
        assertSame(obj3, obj4);
        assertNotSame(obj2, obj4);
        assertNotNull(obj1);
        assertNull(obj5);
        assertTrue(var1 == var2);
        assertFalse(var1 == var3);
        assertArrayEquals(a1, a2);
    }
}
```

3- **Pretende-se escrever e criar uma classe que permita calcular a área e o perímetro de um retângulo.**

a. Escreve o código da classe e um conjunto de testes que possam ser usados para testar a classe.

b. Os métodos da classe a testar têm a seguinte assinatura:

```
public int area(int base, int altura)

public int perimetro(int base, int altura)
```

c. Utilizando os métodos do JUnit que aprendeste, implementa um teste para cada uma das seguintes situações:

i. Caso 1: comp=3, larg = 5

ii. Caso 2: comp=5, larg = 8

iii. Caso 3: comp=2, larg = 4

d. Adiciona um método à classe que permita verificar se um triângulo é possível e implementa o teste com o JUnit

```
public int isTriangle(int a, int b, int c)
```

4- **Exemplo de algumas anotações JUnit descritas na tabela seguinte**

Annotation	Description
`@Test public void method()`	The `Test` annotation indicates that the public void method to which it is attached can be run as a test case.
`@Before public void method()` **@BeforeEach (V.5)**	The `Before` annotation indicates that this method must be executed before each test in the class, so as to execute some preconditions necessary for the test.
`@BeforeClass public static void method()` **@BeforeAll (V.5)**	The `BeforeClass` annotation indicates that the static method to which is attached must be executed once and before all tests in the class. That happens when the test methods share computationally expensive setup (e.g. connect to database).
`@After public void method()` **@AfterEach (V.5)**	The `After` annotation indicates that this method gets executed after execution of each test (e.g. reset some variables after execution of every test, delete temporary variables etc)
`@AfterClass public static void method()` **@AfterAll (V.5)**	The `AfterClass` annotation can be used when a method needs to be executed after executing all the tests in a JUnit Test Case class so as to clean-up the expensive set-up (e.g disconnect from a database). Attention: The method attached with this annotation (similar to `BeforeClass`) must be defined as static.

a. Cria uma nova classe de teste na pasta testes.

b. Executa o seguinte código e tenta entender cada anotação.

```java
public class AnnotationTest {

    private ArrayList<String> testList;

    @BeforeAll
    public static void onceExecutedBeforeAll() {
        System.out.println("@BeforeClass:
onceExecutedBeforeAll");
    }

    @BeforeEach
    public void executedBeforeEach() {
        testList = new ArrayList<String>();
        System.out.println("@Before: executedBeforeEach");
    }

    @AfterAll
    public static void onceExecutedAfterAll() {
        System.out.println("@AfterClass: onceExecutedAfterAll");
    }

    @AfterEach
    public void executedAfterEach() {
        testList.clear();
        System.out.println("@After: executedAfterEach");
    }

    @Test
    public void OneItemCollection() {
        testList.add("oneItem");
        assertEquals(1, testList.size());
        System.out.println("@Test: OneItemArrayList");
    }

    @Test
    public void EmptyCollection() {
        assertTrue(testList.isEmpty());
        System.out.println("@Test: EmptyArrayList");
    }

    }
```

5- Executa as seguintes classes e analisa o uso das anotações JUnit

```java
public class Student {

    private long id;
    private String name;
    private ArrayList<Integer> grades;

    public Student(long id, String name) {
        this.id = id;
        this.name = name;
        this.grades = new ArrayList<>();
    }

    public long getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }

    public ArrayList<Integer> getGrades() {
        return this.grades;
    }

    public void addGrade(int grade) {
        this.grades.add(grade);
    }

    public double getGradeAverage() {
        double sum = 0;
        for (int grade: this.grades) {
            sum += grade;
        }
        return sum / this.grades.size() ;
    }


}
```

```java
public class StudentTest {

    private Student fer;
    private Student ryan;

    @BeforeAll
    public static void onceExecutedBeforeAll() {
        System.out.println("Test Student Class");
    }

    @BeforeEach
    public void setup() {
        fer = new Student(1L, "fer");
        fer.addGrade(100);
        fer.addGrade(80);
    }

    @DisplayName("Check data in Student object")
    @Test
    public void testCreateStudent() {
        assertNull(ryan);
        assertNotNull(fer);
    }

    @Disabled
    @Test
    public void testStudentFields() {
        assertSame(1L, fer.getId());
        assertSame("fer", fer.getName());
        assertSame(2, fer.getGrades().size());
    }


    @Test
    public void testAddGrade() {
        assertSame(100, fer.getGrades().get(0));
        assertSame(80, fer.getGrades().get(1));
    }

    @Test
    public void testAverageGrade() {
        assertEquals(90, fer.getGradeAverage(), 0);
    }


}
```

6- Para consolidar o conhecimento.

a. Cria uma nova Classe Java e implementa um método que determine a quantidade de dígitos de um número dado como parâmetro. O método deverá retornar a média dos dígitos.

b. Implementa a classe que permitirá testar o método anterior. Faz uso das anotações @Test e @BeforeEach.

ATENÇÃO

No final da ficha de trabalho cria um repositório no GitHub com o nome JUnitExercices<numero do aluno>. Coloca lá a resolução do exercício 3 e 6.

Acrescenta o URL do repositório AQUI.

Bom trabalho!