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

Ficha de trabalho 6

Objetivos:

- JUnit 5
- Testes paramétricos, Nested classes, AssertThrows

Exercício 1 - Testar um programa para gerir contactos

Cria um projeto MAVEN e adiciona as duas classes que estão no MOODLE à pasta main.
Analisa as classes.
Um contacto tem um nome, um apelido e um nº de telefone
Algumas validações que deverão ser realizadas ao criar um contato:

- O primeiro nome não deve ser nulo
- O apelido não deve ser nulo
- O número de telefone deve: Ter exatamente 10 dígitos de comprimento, contém apenas dígitos, começa com 0

Cria uma classe de teste, de acordo com o código seguinte, que permite;

a) Testar a criação de um contacto

b) Testar que não é criado um contacto com o 1º nome nulo

c) Testar que não é criado um contacto com o apelido nulo

d) Testar que não é criado um contacto sem nº de telefone

e) Testar que não é criado um contacto com um nº de telefone que não comece em 0

f) Repetir um teste de criação de contacto 5 vezes

g) Criar testes com parâmetros, obtidos quer usando @ValueSource, quer usando @MethodSource

Depois de criar os testes, executa-os; experimenta mudar os valores atribuídos aos dados do contacto e analise os resultados de novos testes.

- Para que são utilizadas as Nested classes?
- Verifica e entendo o uso @ValueSource e @MethodSource
- Entende as exceptions testadas através do AssertThrows

```java
public class ContactManagerTest {
     private ContactManager contactManager;

    @BeforeAll
    public static void setupAll() {
        System.out.println("Should Print Before All Tests");
    }

    @BeforeEach
    public void setup() {
        System.out.println("Instantiating Contact Manager");
        contactManager = new ContactManager();
    }

    @Test
    @DisplayName("Deve criar um Contact")
    public void shouldCreateContact() {
        contactManager.addContact("John", "Doe", "0123456789");
        assertFalse(contactManager.getAllContacts().isEmpty());
        assertEquals(1, contactManager.getAllContacts().size());
    }

    @Test
    @DisplayName("Should Not Create Contact When First Name is
Null")
    public void shouldThrowRuntimeExceptionWhenFirstNameIsNull()
{
        Assertions.assertThrows(RuntimeException.class, () -> {
            contactManager.addContact(null, "Doe",
"0123456789");
        });
    }

    @Test
    @DisplayName("Should Not Create Contact When Last Name is
Null")
    public void shouldThrowRuntimeExceptionWhenLastNameIsNull()
{
        Assertions.assertThrows(RuntimeException.class, () -> {
            contactManager.addContact("John", null,
"0123456789");
        });
    }

    @Test
    @DisplayName("Should Not Create Contact When Phone Number is
Null")
    public void
shouldThrowRuntimeExceptionWhenPhoneNumberIsNull() {
```

```
        Assertions.assertThrows(RuntimeException.class, () -> {
            contactManager.addContact("John", "Doe", null);
        });
    }

    @Test
    @DisplayName("Phone Number should start with 0")
    public void shouldTestPhoneNumberFormat() {
      contactManager.addContact("John", "Doe", "0123456789");
        assertEquals(1, contactManager.getAllContacts().size());
    }

    @Nested
    class RepeatedTests {
        @DisplayName("Repeat Contact Creation Test 5 Times")
       @RepeatedTest(value = 5)
        public void shouldTestContactCreationRepeatedly() {
            contactManager.addContact("John", "Doe",
"0123456789");

assertFalse(contactManager.getAllContacts().isEmpty());
            assertEquals(1,
contactManager.getAllContacts().size());
        }
    }

    @Nested
    class ParameterizedTests {
        @DisplayName("Phone Number should match the required
Format")
        @ParameterizedTest
        @ValueSource(strings = {"0123456789", "0123456798",
"0123456897"})
        public void
shouldTestPhoneNumberFormatUsingValueSource(String phoneNumber)
{
            contactManager.addContact("John", "Doe",
phoneNumber);

assertFalse(contactManager.getAllContacts().isEmpty());
            assertEquals(1,
contactManager.getAllContacts().size());
        }

    }

    @DisplayName("Method Source Case - Phone Number should match
the required Format")
    @ParameterizedTest
```

```
    @MethodSource("phoneNumberList")
    public void
shouldTestPhoneNumberFormatUsingMethodSource(String phoneNumber)
{
        contactManager.addContact("John", "Doe", phoneNumber);
        assertFalse(contactManager.getAllContacts().isEmpty());
        assertEquals(1, contactManager.getAllContacts().size());
    }

    private static List<String> phoneNumberList() {
        return Arrays.asList("0123456789", "0123456798",
"0123456897");
    }

    @Test
    @DisplayName("Test Should Be Disabled")
    @Disabled
    public void shouldBeDisabled() {
        throw new RuntimeException("Test Should Not be
executed");
    }


    @AfterEach
    public void tearDown() {
        System.out.println("Should Execute After Each Test");
    }

    @AfterAll
    public static void tearDownAll() {
        System.out.println("Should be executed at the end of the
Test");
}
}
```

Exercício 2

1. Cria uma class na pasta main do teu projeto e implementa os seguintes métodos:
    a. **isPrime(int number)** que verifica se um número é primo. Um número é primo se for apenas divisível por si próprio e pela unidade, por exemplo: 11 é número primo (visto que é apenas divisível por 11 e por 1), enquanto 21 não é primo, pois tem os seguintes divisores: 1, 3, 7 e 21.
    b. **isEven(int number)** que verifica se um número é par
    c. **isMultiple (int number, int divisor)** que verifica se um número é múltiplo de outro
2. Cria a class de teste correspondente.
    a. Utilizando testes com parâmetros, implementa os métodos de teste que permita testar vários valores.

i. Números primos: 3, 23, 311, 487, 653, 947
ii. Número pares: 32, 64, 2, 20, 30, 26
iii. Múltiplos de 23: 23, 46, 115, 184, 207, 230
iv. Testa com outros valores

Exercício 3
Pretende-se criar um sistema para gerir notas de Unidades curriculares; neste sistema existem 2 classes Nota e UC.

A classe Nota tem as seguintes variáveis de instância:
int numAluno
double nota

A classe UC com as seguintes variáveis de instância:

- **String nomeUC** // nome da unidade curricular
- **Nota [] notas** // array com as notas de cada aluno (pode ser um ArrayList)
- **int totalAlunos** // nº máximo de alunos na UC

E os seguintes métodos:

- **insereNotaUC(int numAluno, double nota)**
- **pesquisaAluno (int numAluno)** – pesquisa e devolve a nota de um dado aluno, ou -1 se o aluno não existe nessa UC
- **media ()** – devolve a nota média da UC
- **aprovado(int numAluno)** – verifica se um dado aluno teve aprovação na UC (nota >=9.5), devolvendo true em caso afirmativo e false se reprovou

Escreva o código das classes Nota e UC e um conjunto de testes que possam ser usados para testar as classes. Utiliza as notações que aprendeste nomeadamente testes paramétricos.
Explora também notações como @CsvSource

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 2 e 3.

Acrescenta o URL do repositório AQUI.

Bom trabalho!