Break.{IT}

Home->Tasks->Active

Task: TestSingleton Previous Tasks

Testes de Software

Os testes podem ser classificados de 2 formas: **comportamentais** (***behavior testing***) ou **estado** (***state testing***).

Os **testes comportamentais** permitem determinar se as classes e os métodos estão construídos com os nomes e/ou parâmetros corretos. Estes testes validam as interações da aplicação, mas não validam dados (ex: valor devolvido quando um método é invocado).

Os **testes de estado** avaliam o estado do objeto através da sua interface. Por exemplo, um teste pode verificar se um método recebe um valor *null* num dos parâmetros.

A criação de testes em JUnit é feita recorrendo a um classe para cada Test Case. Um Test Case agrupa um conjunto de testes de uma forma lógica, marcando o papel de cada método com uma anotação. No exemplo seguinte, o teste é marcado por uma anotação **@Test**. As outras anotações são utilizadas para configurar a execução de testes, da seguinte forma:

- a anotação **@BeforeAll** define que o método marcado vai ser **executado antes de todos os testes** (métodos marcados com **@Test**);

- a **@BeforeEach** leva à execução do método marcado **antes da execução de cada teste**;

- a **@AfterEach** e **@AfterAll** leva à execução dos métodos marcados **depois de cada teste** e **depois de todos os testes**, respetivamente.

```
public class TestSingleton {
HashMap h;

@BeforeAll
public static void setUpBeforeAllTests(){
}

@BeforeEach
public void setUp(){
h= new HashMap();
}

@AfterAll
public void tearDown(){
}

@AfterEach
public void setUp(){
}

@Test
public void testNullValue() {
   assertEquals("Path different from null",null,RegistrySingleton.instance().getPath());
}
}
```

Exercício

Considere a seguinte implementação de uma classe Singleton.

```
package com.es2.singleton;

public class Registry {
private static Registry object = null;
private String path;
private String connectionString;

private Registry(){}

public static Registry getInstance(){
   if(object == null) object = new Registry();

   return object;
}

public String getPath() {
   return path;
}

public void setPath(String path) {
   this.path = path;
}

public String getConnectionString() {
   return connectionString;
}

public void setConnectionString(String connectionString) {
   this.connectionString = connectionString;
}
}
```

-Crie um teste para verificar se o método *setPath()* regista o caminho do ficheiro corretamente.
-Crie um teste para verificar se o método *setConnectionString()* regista a string de conexão corretamente.
-Crie um teste para verificar se o método *setPath()* tem o comportamento expectável quanto recebe um valor null.
-Crie um teste para verificar se o método *setConnectionString()* tem o comportamento expectável quanto recebe um valor null.
-Crie um teste para verificar se o construtor da classe *Singleton()* é privado. Pode utilizar a expressão *fail(...)* para fazer o teste falhar, caso o construtor seja público.

- Crie dentro do **método marcado com @AfterAll** um objeto **End** (**objeto sem classes e métodos**). Este objeto vai permitir à plataforma determinar quando todos os testes foram executados.

```
@AfterAll
static void tearDown() {
   End e = new End();
}
```

Submeta a **classe de testes** com o nome **TestSingleton** para validação.

Specification

Deadline
2020-03-29 23:55

Submit for Testing | Submit for Evaluation

Submission Done With Success !!!

```
.
├─ JUnit Jupiter ✓
│  └─ TestSingleton ✓
│     ├─ getInstance() ✓
│     ├─ setNullConnectionString() ✓
│     ├─ getConnectionString() ✓
│     ├─ getPath() ✓
│     ├─ setNullPath() ✓
│     ├─ singletonConstructor() ✓
│     ├─ setPath() ✓
│     └─ setConnectionString() ✓
└─ JUnit Vintage ✓

Test run finished after 172 ms
[ 3 containers found ]
[ 0 containers skipped ]
[ 3 containers started ]
[ 0 containers aborted ]
[ 3 containers successful ]
[ 0 containers failed ]
[ 8 tests found ]
[ 0 tests skipped ]
[ 8 tests started ]
[ 0 tests aborted ]
[ 8 tests successful ]
[ 0 tests failed ]
```