IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

PA 2021/22

Programação Avançada

Enunciado de Laboratório

0.2

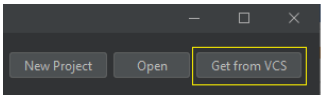
Laboratório de Introdução ao ambiente IntelliJ e JUnit (Parte 2/2)

Objetivos:

- Criar um clone de um projeto que está no GitHub;
- Criar testes em JUnit no IDE.

01 – Criar um clone de um projeto que está no GitHub

1. Execute o IntelliJ e *clone* o projeto que está no seguinte endereço:
   https://github.com/estsetubal-pa-geral/lab0_02.git



   Esta nova versão do projeto, que foi apresentado na fase 1 do laboratório, são apresentadas algumas alterações, das quais podemos destacar:
   - O nome do retângulo é colocado em maiúsculas no construtor;
   - Nova classe para o tratamento da exceção **InvalidSizeException**;
   - Deteção das exceções **InvalidSizeException** e **IllegalArgumentException**.

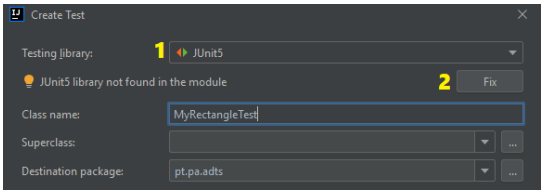
2. Verifique que o projeto executa o método **main** sem problemas.

02 – Criar a pasta de testes

3. Crie a pasta **test** na raiz do seu projeto (mesmo nível da pasta src).
4. Configure a pasta como pasta de testes no projeto. Botão direito do rato em cima da pasta/**Mark Directory as**/**Test Sources Root** (a pasta deverá aparecer a verde).
5. Crie o mesmo **package** que está associado à pasta src.

03 – Criar o primeiro teste em JUnit no IDE

6. Abra a classe **MyRectangle** no IDE.
7. Coloque o cursor na linha onde a classe é definida e botão direito do rato/**Show Content Actions**/**Create Test**.
8. Caso o módulo não esteja presente escolha a JUnit5 [Passo 1] e prima o botão **Fix** [Passo 2].



9. Em seguida selecione a versão junit-jupiter:5.*.
10. Selecione (*check*) o método isPerfect()/OK.
11. Escreva e execute o seguinte teste.

```
class MyRectangleTest {

    @Test
    void isPerfect() {
        assert(true);
    }
}
```

12. Altere o teste anterior e execute-o com o seguinte código. Qual o resultado?

```
class MyRectangleTest {

    @Test
    void isPerfect() {
        assertTrue( condition: false);
    }
}
```

13. Pretende-se escrever testes cujo nome seja autoexplicativo daquilo que o teste verifica. Vamos aumentar a legibilidade do teste atribuindo uma designação explicita àquilo que está a ser testado, indicando também o resultado esperado.

```
@Test
void isPerfect_returnsTrue_whenRectangleIsPerfect() {
    MyRectangle rect = new MyRectangle( name: "One",  x: 10.0,  y: 20.0,  width: 30.0,  height: 40.0);
    assertTrue(rect.isPerfect());
    rect = new MyRectangle( name: "Two",  width: 20.0,  height: 40.0);
    assertTrue(rect.isPerfect());
}

@Test
void isPerfect_returnsFalse_whenRectangleIsNotPerfect() {
    MyRectangle rect1 = new MyRectangle( name: "One",  x: 10.0,  y: 20.0,  width: 10.0,  height: 40.0);
    MyRectangle rect2 = new MyRectangle( name: "Two",  width: 80.0,  height: 10.0);
    assertAll(
            () -> assertFalse(rect1.isPerfect()),
            () -> assertFalse(rect2.isPerfect())
    );
}
```

14. Crie o teste que verifica se a hipotenusa do retângulo com o nome "Zip Rect', x=10.0, y=20.0, width=1.0, height=4.0 tem uma hipotenusa igual a 4.1231 (*assertEquals*).
15. Altere o teste para usar uma precisão de apenas 0.0001 (ver 3º argumento de assertEquals) e compare o resultado.
16. Crie o teste **void name_isUpperCase_afterCreate()**.
17. Crie o teste **void name_isUpperCase_afterSettingName ()**.

Será que os métodos da classe estão bem implementados? Faça a correção de algum método que possa estar incorretamente implementado.

18. Crie o teste que permite validar o lançamento da exceção **InvalidSizeException** sempre que uma das dimensões é inválida.

19. Crie o teste que permite validar o lançamento da excepção **IllegalArgumentException**.
20. Altere a forma como o nome do teste anterior aparece ao utilizador colocando uma anotação (@...) que apresente ***Checks if name cannot be null*** em caso de erro.
21. Execute de uma só vez todos os testes da classe MyRectangle.