Universidade Federal de São Carlos
Departamento de Computação

Disciplina de Engenharia de Software 1

Relatório 4 - JUnit 5

Amanda Basso de Oliveira - 727331

Outubro de 2020

Contents

1 Novidades do JUnit 5 **1**
1.1 Diferenças na arquitetura . . . 1
1.2 Diferenças no código . . . 1
1.2.1 *Timeout* . . . 1
1.2.2 *Expected* . . . 1
1.2.3 Repetição de testes . . . 1
1.2.4 Assertions . . . 1
1.2.5 Testes aninhados . . . 1
1.2.6 Testes parametrizados . . . 2
1.2.7 Testes dinâmicos . . . 2
1.2.8 Interfaces para testes . . . 2
1.2.9 *@TestInstance* . . . 2
1.2.10 *@DisplayName* . . . 2
1.2.11 Mudanças de notação . . . 2

1 Novidades do JUnit 5

1.1 Diferenças na arquitetura

Em relação à arquitetura, o JUnit passou a ser constituído como uma junção de diferentes módulos: o JUnit Plataform - descobre e executa os testes na JVM -, o JUnit Jupiter - contém os novos recursos para testes - e o JUnit Vintage - fornece a *TestEngine* para os testes escritos em JUnit 3 ou 4.

1.2 Diferenças no código

Embora o JUnit 5 mantenha a notação *@Test* do JUnit 4 para dizer que o método é um teste, o JUnit não possui mais os dois parâmetros *timeout* e *expected*.

1.2.1 *Timeout*

Nas versões anteriores, o JUnit iniciava a medição do tempo de execução junto com o início do método do teste em vez de verificar apenas o timeout do código que estava sendo testado. A forma de verificar *timeouts* no JUnit 5 é utilizando o método *assertTimeout*.

O método *assertTimeout* recebe dois argumentos. O primeiro deles representa um objeto do tipo *Duration* e representa quantidade de tempo. O segundo é um *Executable*.

Visto que a execução aguardou o bloco terminar - o *Executable* é executado na mesma thread do teste, é possível saber a diferença entre o tempo total da execução e o *timeout* esperado.

Contudo, para casos em que o código leva mais tempo para executar do que o *timeout* estabelecido, existe o método *assertTimeoutPreemptively*. Este método é executado em uma thread separada e a execução é interrompida assim que o *timeout* seja excedido.

1.2.2 *Expected*

Para testar códigos que lançam exceções, agora é utilizado o método *assertThrows*. É necessário especificar o tipo da exceção esperada no primeiro parâmetro e um *Executable* no segundo.

É possível utilizar o mesmo método para capturar a exceção lançada na implementação de alguma validação, ao contrário do JUnit 4, que exigiria o uso da classe *ExpectedException*.

1.2.3 Repetição de testes

O JUnit 5 permite a repetição de testes *n* vezes utilizando o método *RepeatedTest*. Possui, ainda, um parâmetro chamado *name* que é utilizado para gerenciar a saída do teste.

1.2.4 Assertions

A classe *Assert* foi substituída pela classe *Assertion*, embora tenha a mesma finalidade - fornecer os métodos *assert*, que validam as condições que determinam se um teste passou ou não. A única diferença entre os métodos das versões anteriores é que o argumento que permitia a sobrecarga passou a ser o último, em vez de ser o primeiro.

assertAll é um novo método que permite executar um conjunto de testes que serão validados juntamente.

1.2.5 Testes aninhados

O JUnit 5 introduziu a notação *@Nested* para teste de classes que possuem vários métodos que dependem do estado interno do objeto para execução.

1.2.6 Testes parametrizados

Como novidade do JUnit 5, o suporte para testes parametrizados está em módulo separado do JUnit Jupiter e, por isso, é necessário adicionar dependências. Através dele, é possível injetar os parâmetros diretamente no método *factory method*.

1.2.7 Testes dinâmicos

É possível construir testes em tempo de execução no JUnit 5. Os testes dinâmicos são criados em um método chamado *TestFactory*.

Visto que o teste é construído no código, os recursos do Java o parametrizam.

1.2.8 Interfaces para testes

Com JUnit 5, é possível criar interfaces que funcionam como *templates* de testes e inclui-las na assinatura das classes de teste específicas. Isso ocorre porque as anotações podem ser incluídas em métodos default de interfaces.

Ainda é possível implementar espécie de herança no JUnit 5, como métodos de teste na superclasse, porque os métodos são herdados - como nas versões antigas.

1.2.9 *@TestInstance*

Agora é possível executar o teste utilizando instâncias das classes previamente instanciadas através da notação *@TestInstance*, de modo a quebrar o princípio do isolamento.

1.2.10 *@DisplayName*

Essa notação permite descrever mais claramente o teste.

1.2.11 Mudanças de notação

@BeforeEach e *@AfterEach* substituíram as antigas *@Before* e *@After*, respectivamente. De forma análoga, as anotações *@BeforeClass* e *@AfterClass* foram substituídas por *@BeforeAll* e *@AfterAll*.