



PARTE II – FERRAMENTAS DE TESTE

UNIDADE 2: FERRAMENTAS DE TESTE UNITÁRIO

Professor: Fabrício Galende Marques de Carvalho
Curso: Análise e Desenvolvimento de Sistemas
Turno: Manhã
Carga horária: 80h

São José dos Campos - 2019

1



OBJETIVOS DA UNIDADE

- Apresentar o JUNIT como ferramenta de teste unitário aplicável aos sistemas desenvolvidos em JAVA.
- Descrever algumas boas práticas de estruturação de programas objetivando torná-los testáveis.

2

CONTEÚDO DA UNIDADE

1. ARQUITETURA DO JUNIT
 2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE
 3. ANOTAÇÕES DE MÉTODOS DE TESTE
 4. ASSERTIVAS
 5. ASSUNÇÕES
-

3

1. ARQUITETURA DO JUNIT

- ✓ A ferramenta de testes unitários JUNIT, atualmente na sua versão 5, é uma das mais utilizadas para testes de sistemas desenvolvidos em linguagem de programação JAVA.
 - ✓ Há três componentes essenciais à nova plataforma: JUNIT Platform, JUNIT Jupiter, JUNIT Vintage.
-

4

1. ARQUITETURA DO JUNIT

- ✓ **JUNIT Platform:** Permite a execução, independente de plugins, de casos de teste, execução de um console para comando, etc.
 - ✓ **JUNIT Jupiter:** Inclui diversas dependências que permitem que se escreva e se execute os casos de teste (especialmente a API)
 - ✓ **JUNIT Vintage:** Inclui componentes para compatibilidade com versões anteriores do JUNIT.
-

5

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

- ✓ Para se utilizar o JUNIT 5 com o gradle, essencialmente são requeridos componentes do JUNIT Jupiter API e Engine.
 - ✓ No arquivo build.gradle, acrescentar o seguinte repositório, dependências e habilitação do JUNIT:
-

6

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

```
repositories{  
    mavenCentral()  
}  
dependencies{  
    testCompileOnly 'org.junit.jupiter:junit-  
        jupiter-api:5.3.1'  
    testRuntimeOnly 'org.junit.jupiter:junit-  
        jupiter-engine:5.3.1'  
}  
test{  
    useJUnitPlatform()  
}
```

7

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

- ✓ Uma vez configurado, o JUnit está pronto para ser utilizado, bastando que sejam criadas as classes da aplicação e as classes de teste.

OBSERVAÇÃO: É comum, que para cada classe da aplicação Chamada de Classe, tenha-se uma classe de teste denominada de ClasseTest (ou ClasseTeste).

8

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

Exemplo: Projeto de uma calculadora que efetua soma e subtração, somente.

Inicialização de um projeto gradle com layout padrão de aplicação java:

```
gradle init -type java-application
```

9

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

Arquitetura da solução em pacotes (aplicação). Todos os arquivos devem ser criados dentro de src/main/java.

Pacote app (contém a classe aplicação do sistema).

Aplicacao.java

Pacote model (contém a classe modelo da calculadora).

Calculadora.java

10

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

Os pacotes e classes de teste, devem ser criados dentro de src/test/java

Arquitetura da solução em pacotes (teste):

Pacote model (contém a classe que testa o modelo da calculadora);

CalculadoraTest.java

DICA: Ao se criar a estrutura de projeto padrão com o init script, pode-se importar o projeto diretamente no Netbeans, caso o plugin de suporte ao gradle esteja instalado. Atualizar o projeto após criar o build.gradle.

11

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

```
6 package model;
7 public class Calculadora {
8     public int somar(int x, int y) {
9         return (x+y);
10    }
11    public int subtrair(int x, int y) {
12        return (x-y);
13    }
14 }
```

12

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

```
1 package app;
2 import model.Calculadora;
3
4 public class Aplicacao{
5     public static void main(String [] args){
6         Calculadora calc = new Calculadora();
7         System.out.print("1 + 2: ");
8         System.out.println(calc.somar(1,2));
9         System.out.print("1-2: ");
10        System.out.println(calc.subtrair(1, 2));
11    }
12 }
```

13

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

```
package model;
import model.Calculadora;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class CalculadoraTest {
```

14

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE


```
12 public class CalculadoraTest {  
13  
14     private Calculadora calc;  
15     public CalculadoraTest() {  
16         calc = new Calculadora();  
17     }  
18     @Test  
19     public void somarTest() {  
20         assertEquals(1, calc.somar(1,0));  
21     }  
22     @Test  
23     public void subtrairTest() {  
24         assertEquals(-1, calc.subtrair(0,1));  
25     }  
26 }
```

15

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

```
\ex_01>gradlew.bat test  
BUILD SUCCESSFUL in 2s  
3 actionable tasks: 1 executed, 2 up-to-date
```

16

2. UTILIZAÇÃO DA FERRAMENTA COM O GRADLE

CalculadoraTest

all > model > CalculadoraTest

2
tests

0
failures

0
ignored



0.009s
duration

100%
successful

Tests

Test	Duration	Result
somarTest()	0.008s	passed
subtrairTest()	0.001s	passed

17

3. ANOTAÇÕES DOS MÉTODOS DE TESTE

3.1. IDEIA GERAL

✓ O framework JUnit faz uso de diversas anotações (metadados que não alteram a semântica do código) para indicar que métodos e/ou classes devem ser consideradas pelo framework como parte de um determinado conjunto de testes.

18

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.1. IDEIA GERAL

- ✓ O framework gera uma instância de cada classe de teste antes de executar os métodos de teste individuais. Isso permite que os casos de teste sejam executados em isolamento uns dos outros, ou seja, instâncias de objetos criados, por exemplo, em um construtor, não influenciam no resultado da execução dos demais métodos de teste.
-

19

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2. EXEMPLOS DE ANOTAÇÕES

3.2.1. @Test

É uma das mais fundamentais anotações e é utilizada para marcar um método como sendo um método de teste.

Trata-se de uma anotação do tipo marcador, ou seja, não declara parâmetros.

Os métodos marcados não podem ser privados nem retornar valor (devem ser void)

20

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.1. @Test

Exemplo:

```
@Test
public void somarTest() {
    assertEquals(1, calc.somar(1,0));
}
```

21

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.2. @BeforeAll

- ✓ Indica que um determinado método deve ser executado antes de todos os demais métodos de teste, incluindo, por exemplo, os marcados com @Test.
- ✓ Os métodos marcados com @BeforeAll devem ser estáticos por padrão e não devem retornar valor.

22

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.2. @BeforeAll



Caso se deseje que o framework altere o ciclo de vida padrão das instâncias da classe de teste, deve-se anotá-la com `@TestInstance(Lifecycle.PER_CLASS)`. Por padrão o ciclo de vida utiliza `@TestInstance(Lifecycle.PER_METHOD)`

23

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.3. @AfterAll

- ✓ Indica que um determinado método deve ser executado após todos os métodos de teste terem sido executados.
 - ✓ Por padrão, devem possuir retorno void e devem ser estáticos.
-

24

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.4. @BeforeEach

- ✓ Indica que um determinado método deve ser executado antes da execução **de cada método** de teste.
 - ✓ Por padrão, devem possuir retorno void, não podem ser métodos estáticos e nem podem ser privados.
-

25

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.5. @AfterEach

- ✓ Indica que um determinado método deve ser executado após a execução **de cada método** de teste.
 - ✓ Por padrão, devem possuir retorno void, não podem ser métodos estáticos e nem podem ser privados.
-

26

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.6. @Disabled

- ✓ Marca uma classe de teste ou método de teste de modo que o mesmo não seja considerado durante a execução do JUNIT.
 - ✓ Esse tipo de anotação é útil quando se está em um ciclo de desenvolvimento incremental e se parte da especificação de um conjunto de casos de teste que só serão executados a medida que um determinado componente seja desenvolvido.
-

27

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

- ✓ Algumas vezes é interessante que uma mesma funcionalidade seja exercitada utilizando-se diversos valores para um caso de teste que possua a mesma estrutura (ex. partição de equivalência com dados no meio e nos extremos da partição). Nesse caso, podem ser utilizados os chamados testes parametrizados, que fazem uso da anotação ***@ParameterizedTest***.
-

28

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

- ✓ Essa anotação faz parte de um conjunto de *features* experimentais do framework Junit 5.
- ✓ Para que testes parametrizados sejam executados, a seguinte configuração e dependência devem ser utilizadas:

Configuração: `testCompile`

Dependência:

`'org.junit.jupiter:junit-jupiter-params:5.3.1'`

29

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

- ✓ Além da anotação de teste parametrizado, deve ser informada a fonte de dados. Pode-se utilizar dados em um array, informado em um literal por linha, por array de dados de strings CSV ou então através de arquivos de dados CSV. Em todos os casos o framework tenta fazer uma conversão de tipo inferindo a partir dos tipos dos parâmetros que são recebidos no método de teste.

30

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

✓ Exemplo utilizando dados em array literal.

MODELO:

```

1  package model;
2
3  public class Calculadora{
4      public int somar(int x, int y){
5          return (x+y);
6      }
7      public int subtrair(int x, int y){
8          return (x-y);
9      }
10     public int quadrado(int x){
11         return (x*x);
12     }
13 }
  
```

31

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

✓ Exemplo utilizando dados em array literal.

TESTE:

```

5  import static org.junit.jupiter.api.Assertions.assertEquals;
6  import org.junit.jupiter.params.ParameterizedTest;
7  import org.junit.jupiter.params.provider.CsvFileSource;
8  import org.junit.jupiter.params.provider.CsvSource;
9  import org.junit.jupiter.params.provider.ValueSource;
10 public class CalculadoraTest {
11     private Calculadora calc;
12     public CalculadoraTest() {
13         calc = new Calculadora();
14     }
  
```

32

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

✓ Exemplo utilizando dados em array literal.

TESTE:

```
@ParameterizedTest
@ValueSource(ints={1,2,3,4,5})
public void quadradoTest(int par01) {
    int resultado ;
    resultado = calc.quadrado(par01);
    assertEquals(par01*par01, resultado);
}
```

33

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

✓ Exemplo utilizando dados string CSV literal.

TESTE:

```
@ParameterizedTest
//@CsvSource({"1,2,3","4,5,9", "10,20,30"})
@CsvSource(value={"1:2:3","4:5:9", "11:12:23"},
    delimiter=':')
public void somarTest(int x, int y, int resultado){
    assertEquals(resultado, calc.somar(x, y));
}
```

34

3. ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.7. @ParameterizedTest

✓ Exemplo utilizando dados CSV em arquivo.

TESTE:

```
@ParameterizedTest(name="Teste {index} => x={0} y={1}, "  
                    + " resultado={2}")  
//@ParameterizedTest  
@CsvFileSource(resources="/valores_teste.csv",  
               delimiter=',')  
public void somarTestCsv(int x, int y, int resultado){  
    assertEquals(resultado, calc.somar(x, y));  
}
```

35

3. ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.8. @RepeatedTest

✓ Muitas vezes o desenvolvedor está interessado em verificar se um determinado componente, ao ser ativado várias vezes, possui um determinado comportamento (ex. bloqueio de usuário após três tentativas incorretas). Nesse caso, a anotação @RepeatedTest provê um mecanismo adequado a tal fim.

36

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.8. @RepeatedTest

- ✓ Casos de teste com essa anotação podem, adicionalmente, acessar metadados do teste que está sendo repetido (e.g.: número da repetição) afim de que se possa executar um teste mais refinado.
 - ✓ Um exemplo de metadado é o número da repetição executada (atributo `currentRepetition` da classe `RepetitionInfo`)
-

37

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.8. @RepeatedTest

Exemplo:

```
@RepeatedTest(3)
public void
autenticarTestBloqueio(RepetitionInfo repetitionInfo){
    Usuario u = usuarioServiceBlock.autenticar("fabricio",
        "1234");
    assertNull(u);
    if (repetitionInfo.getCurrentRepetition() >= 3){
        assertTrue(usuarioServiceBlock.usuarioBloqueado("fabricio"));
    }
}
```

38

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.9. @DisplayName

Essa anotação possibilita ao desenvolvedor dos casos de teste a exibição de um nome mais intuitivo no relatório de teste.

A vantagem de se utilizar essa anotação está no fato do não comprometimento da convenção de código utilizada nos casos de teste.

39

3.ANOTAÇÕES DOS MÉTODOS DE TESTE

3.2.9. @DisplayName

Exemplo:

```
@Test
@DisplayName("Busca do usuario victor")
public void autenticarTestSucesso2() {
    UsuarioService usuarioService = new UsuarioServiceImpl();
    Usuario u = usuarioService.autenticar("victor", "xyz");
    assertEquals("victor", u.getUserName());
    assertEquals("xyz", u.getSenha());
}
```

40

4. ASSERTIVAS

4.1. NOÇÕES GERAIS

- ✓ O Framework JUnit possui uma classe, denominada de Assertions (Assertivas) que inclui uma série de métodos estáticos que permitem que o desenvolvedor execute testes através de comparações com determinada condição (em geral o valor esperado do caso de teste).
-

41

4. ASSERTIVAS

4.2. EXEMPLOS

4.2.1. assertEquals

Essa assertiva verifica se o valor esperado (primeiro parâmetro) é igual ao valor obtido (segundo parâmetro). Opcionalmente, pode-se acrescentar como argumento uma string gerada em caso de falha.

42

4. ASSERTIVAS

4.2.1. assertEquals

```
24      @Test
25      public void subtrairTest() {
26          assertEquals(-1, calc.subtrair(0,2),
27                      "deveria ser -1");
28      }
```

43

4. ASSERTIVAS

4.2.1. assertEquals



O método `assertEquals`, para efetuar comparação de objetos, procura na classe modelo por um método com a seguinte assinatura:

```
public boolean equals(Object o);
```

44

4. ASSERTIVAS

4.2.1. assertEquals



Há várias versões desse método que suportam diferentes tipos de dados.



Cada versão possui uma adaptação ao tipo (ex. para pontos flutuantes, deve-se acrescentar uma tolerância para igualdade, etc.)

45

4. ASSERTIVAS

4.2.2. assertNotEquals

- ✓ Similar ao assertEquals, entretanto, é utilizado quando não se espera que os objetos passados como argumentos sejam iguais.
 - ✓ Um exemplo de aplicação seria no teste de atualização em um BD ou em um arquivo de dados.
-

46

4. ASSERTIVAS

4.2.3. assertNotNull

- ✓ Utilizado quando se deseja verificar se uma referência não é null (o que tipicamente causa as chamadas NullPointerExceptions).
 - ✓ O seu análogo é o ***assertNull***.
-

47

4. ASSERTIVAS

4.2.4. assertNotSame

- ✓ Utilizado quando se quer verificar se os objetos possuem a mesma referência.
 - ✓ O caso de teste passará em caso de cópias (novas instâncias) e falhará no caso de mesmas referências.
 - ✓ O seu análogo é o ***assertSame***
-

48

4. ASSERTIVAS

4.2.5. assertAll

- ✓ Utilizado quando se quer verificar se todos os executáveis não lançam exceções.
- ✓ É comum a sua utilização com expressões lambda do Java 9.

49

4. ASSERTIVAS

4.2.5. assertAll

```
CalculadoraComEstado c1 = new CalculadoraComEstado(  
CalculadoraComEstado c2 = new CalculadoraComEstado(  
    c1.x = 1;  
    c1.y = 2;  
    c2.x = 1;  
    c2.y = 2;  
    assertAll("Teste de várias assertivas:",  
        () -> assertNotNull(c1),  
        () -> assertNotNull(c2),  
        () -> assertEquals(c1, c2));
```

50

4. ASSERTIVAS

4.2.5. assertAll

```
CalculadoraComEstado c1 = new CalculadoraComEstado(  
CalculadoraComEstado c2 = new CalculadoraComEstado(  
c1.x = 1;  
c1.y = 2;  
c2.x = 1;  
c2.y = 2;  
assertAll("Teste de várias assertivas:",  
    () -> assertNotNull(c1),  
    () -> assertNotNull(c2),  
    () -> assertEquals(c1, c2));
```

51

5. ASSUNÇÕES

5.1. ASSUNÇÕES

- ✓ Diferentemente dos assertivas, a falha em uma assunção não resulta em um estado de falha para o caso de teste, mas sim no aborto da execução.
- ✓ Assunções são tipicamente utilizadas na verificação de configurações de sistema, etc.

52

5. ASSUNÇÕES

5.2. EXEMPLO

5.2.1. `assumeTrue`

Valida uma determinada assunção (hipótese)

```
@Test
public void testAssumption() {
    assumeTrue(System.getenv("PATH") .
contains("Java"));
}
```

53

5. ASSUNÇÕES

5.2. EXEMPLO

5.2.2. PESQUISAR AS SEGUINTESS ASSUNÇÕES

`assumingThat`
`assumingFalse`

54



FIM DA UNIDADE