



Introdução ao Framework Mockito

Prof. Evandro César Freiburger

Instituto Federal de Mato Grosso
Departamento da Área de Informática
evandro.freiberger@ifmt.edu.br

2023

Sumário

- 1 Introdução aos Teste Unitário
- 2 Introdução ao Framework Mockito
- 3 Mockito para Simular uma Dependência
- 4 Mockito para Simular uma Implementação Futura
- 5 Exemplos de Testes Unitários
- 6 Mocks de Métodos com Retorno void
- 7 Teste de Classe de Negócio com Mocks de DAO

Teste Unitário / Unidade

É o nível de teste que tem como objetivo realizar testes nas menores unidades de código de um software (módulo, método de classe, classe, componente).

A automação de testes unitários é fortemente incentivada, visto que, trata-se de testes em porções de código consideradas primitivas dentro de um contexto de sistema, sendo mais fáceis de serem testadas pelos próprios programadores que as produzem.

Outro fato que justifica a automação de testes unitário, é que esses testes devem ser repetidos todas as vezes que um software for alterado, ou quando uma nova versão for distribuída, garantindo a integridade do código, pela facilidade de repetir sua execução.

Isolamento da Unidade de Teste

Quando estamos trabalhando com testes de Unidade ou testes Unitário, uma das maiores preocupações e as vezes um dos maiores esforços se concentra no isolamento da unidade a ser testada.

O objetivo é ter certeza que o teste diz respeito apenas à unidade em teste no momento, e não tenha interferência com funcionalidades que pertençam a outras unidades.

Se o teste está sendo produzido para uma unidade (método ou classe), que essa unidade não sofra interferência de outros recursos externos usados na sua execução, visto que, se o teste falhar não será possível saber facilmente se a falha é da unidade em questão ou se é de algum recurso externo.

Outra questão é que se algo mudar no sistema e provocar uma quebra em testes previamente estabelecidos, o ideal é que a quebra se dê apenas na unidade que foi feita a alteração, não provocando efeitos colaterais.

O nível de testes que envolvem a integração/dependência entre unidade de um software é o Teste de Integração.

Testes Unitários

Crie um Projeto Maven (maven-archetype-quickstar), com os seguintes dados:

- **groupId** = br.ifmt.cba
- **artifactId** = junit02
- **version** = 1.0-SNAPSHOT
- **pasta do projeto**: JUnit02

Classe de negócio Circulo

```
1 package ifmt.cba.negocio;
2
3 public class Circulo {
4
5     private double raio;
6
7     public Circulo(double raio) {
8         this.raio = raio;
9     }
10
11     public double getRaio() {
12         return raio;
13     }
14
15     public void setRaio(double raio) {
16         this.raio = raio;
17     }
18
19     public double getArea() throws RuntimeException {
20         if (this.raio > 0) {
21             return Math.PI * Math.pow(this.raio, 3); //provoca erro
22             //return Math.PI * Math.pow(this.raio, 2);
23         } else {
24             throw new RuntimeException("Raio com valor inconsistente");
25         }
26     }
27
28 }
```

Testes Unitários

A classe de negócio **Circulo** pode ser considerada uma **Unidade** a ser testada.

O que testar da classe **Circulo**?

Poderíamos testar todos os métodos.

Contudo, métodos construtores e Getters/Setters normalmente são gerados automaticamente pelas IDE e seu teste acaba não trazendo grandes benefícios.

Considerando essa estratégia, **o método getArea() é um bom candidato a automação de teste**, já que é um método cuja lógica é escrita totalmente pelo programador, e que envolve um cálculo baseado em valores e fórmula.

Testes Unitários

Classe de teste CirculoTest

```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 public class CirculoTest {
8
9     @Test
10     public void quandoTodosDadosCorretos() {
11         Circulo circulo = new Circulo(2);
12         assertEquals(12.56, circulo.getArea(), 0.01);
13     }
14
15     @Test
16     public void quandoRaioMenorOuIgualZero() {
17         Circulo circulo = new Circulo(-1);
18         RuntimeException thrown = assertThrows(RuntimeException.class,
19             () -> circulo.getArea(), "Esperava-se a producao de uma execucao, mas nao ocorreu");
20
21         assertTrue(thrown.getMessage().contains("Raio com valor inconsistente"));
22     }
23 }
```

O primeiro cenário testa o método `getArea()` quando os dados estão corretos e um valor de área válido será retornado.

O segundo cenário testa o método `getArea()` com dado inconsistente, cujo retorno é uma exceção

Classe de negócio Cilindro (1)

```
1 package ifmt.cba.negocio;
2
3 public class Cilindro {
4
5     private Circulo circulo;
6     private double altura;
7
8     public Cilindro(Circulo circulo, double altura) {
9         this.circulo = circulo;
10        this.altura = altura;
11    }
12
13    public Circulo getCirculo() {
14        return circulo;
15    }
16
17    public void setCirculo(Circulo circulo) {
18        this.circulo = circulo;
19    }
20
21    public double getAltura() {
22        return altura;
23    }
24    public void setAltura(double altura) {
25        this.altura = altura;
26    }
27
28    public double getArea() throws RuntimeException {
29
30        if (this.circulo != null && this.circulo.getRaio() > 0 && this.altura > 0){
31            //return circulo.getArea() + 2 * Math.PI * this.circulo.getRaio() * this.altura;
```

Classe de negócio Cilindro (2)

```
32         return 2 * circulo.getArea() + 2 * Math.PI * this.circulo.getRaio() * this.altura;  
33     } else {  
34         throw new RuntimeException("Dados inconsistentes");  
35     }  
36 }  
37 }
```

Testes Unitários

A classe de negócio **Cilindro** **pode ser considerada uma Unidade** a ser testada.

Considerando o mesmo raciocínio da classe Circulo, podemos decidir testar cenários que envolvam o método `getArea()`.

O primeiro cenário testa o método `getArea()` quando os dados estão corretos e um valor de área válido será retornado.

O segundo cenário testa o método `getArea()` com dado inconsistente, cujo retorno é uma exceção.

Testes Unitários (1)

Tabela de decisão para os teste de Cilindro

Cenário: Teste de cálculo de área do cilindro (getArea())

Tipo	Variáveis	Partições	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Entradas	Objeto Circulo	Raio > 0	X	X	X									
		Raio = 0				X	X	X						
		Raio < 0							X	X	X			
		Nulo										X	X	X
	Altura	> 0	X			X			X			X		
		0		X			X			X			X	
		< 0			X			X			X			X
Saídas	Área		X											
	Exceção			X	X	X	X	X	X	X	X	X	X	X

Testes Unitários (1)

Classe de Teste CilindroTest

```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 public class CilindroTest {
8
9     @Test
10     public void quandoRaioEAlturaPositivo() {
11         Cilindro cilindro = new Cilindro(new Circulo(2), 4);
12         assertEquals(75.39, cilindro.getArea(), 0.01);
13     }
14
15     @Test
16     public void quandoRaioPositivoEAlturaNegativa() {
17
18         Cilindro cilindro = new Cilindro(new Circulo(2), -4);
19
20         RuntimeException thrown = assertThrows(RuntimeException.class,
21             () -> cilindro.getArea(), "Esperava-se a producao de uma execucao, mas nao ocorreu");
22
23         assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
24     }
25
26     @Test
27     public void quandoRaioNegativoEAlturaPositiva() {
28         Cilindro cilindro = new Cilindro(new Circulo(-1), 4);
29     }
```

Testes Unitários (2)

```
30     RuntimeException thrown = assertThrows(RuntimeException.class,
31         () -> cilindro.getArea(), "Esperava-se a producao de uma excecao, mas nao ocorreu");
32
33     assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
34 }
35
36 @Test
37 public void quandoRaioNegativoEAlturaNegativa() {
38     Cilindro cilindro = new Cilindro(new Circulo(-1), -4);
39     RuntimeException thrown = assertThrows(RuntimeException.class,
40         () -> cilindro.getArea(), "Esperava-se a producao de uma excecao, mas nao ocorreu");
41
42     assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
43 }
44
45 @Test
46 public void quandoCirculoNullAlturaPositiva() {
47     Cilindro cilindro = new Cilindro(null, 4);
48     RuntimeException thrown = assertThrows(RuntimeException.class,
49         () -> cilindro.getArea(), "Esperava-se a producao de uma excecao, mas nao ocorreu");
50
51     assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
52 }
53
54 @Test
55 public void quandoCirculoNullAlturaNegativa() {
56     Cilindro cilindro = new Cilindro(null, -4);
57     RuntimeException thrown = assertThrows(RuntimeException.class,
58         () -> cilindro.getArea(), "Esperava-se a producao de uma excecao, mas nao ocorreu");
59
60     assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
61 }
```

Testes Unitários (3)

62 }

Testes Unitários (1)

Execução do Teste CilindroTest

```
J CilindroTest.java × J Cilindro.java
src > test > java > ifmt > cba > negocio > J CilindroTest.java > CilindroTest > quandoRaioNegativoEAltura
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 public class CilindroTest {
8
9     @Test
10     public void quandoRaioEAlturaPositivo() {
11         Cilindro cilindro = new Cilindro(new Circulo(raio:2), altura:4);
12         assertEquals(expected:75.39, cilindro.getArea(), delta:0.01);
13     }
14
15     @Test
16     public void quandoRaioPositivoEAlturaNegativa() {
```

PROBLEMS OUTPUT TEST RESULTS TERMINAL PORTS DEBUG CONSOLE

```
%TESTC 6 v2
%TSTTREE2,ifmt.cba.negocio.CilindroTest,,[engine:junit-jupiter]/[class:ifmt.cba.negocio.CilindroTest]
%TSTTREE3,quandoRaioEAlturaPositivo(ifmt.cba.negocio.CilindroTest),false,1,false,2,quandoRaioEAlturaPositivo(),,[engine:junit-jupiter]/[class:ifmt.cba.negocio.CilindroTest]/[method:quandoRaioEAlturaPositivo()]
%TSTTREE4,quandoRaioPositivoEAlturaNegativa(ifmt.cba.negocio.CilindroTest),false,1
```

Test run at 11/22/2023, 8:25:43 AM

- quandoCirculoNull(AlturaNegativa())
- quandoCirculoNull(AlturaPositiva())
- quandoRaioEAlturaPositivo()
- quandoRaioNegativoEAlturaNegativa()
- quandoRaioPositivoEAlturaNegativa()
- quandoRaioNegativoEAlturaPositiva()

Test run at 11/22/2023, 8:23:17 AM

Testes Unitários (1)

Provocando Erro na classe Cilindro (OK - erro na unidade em teste)

```
28 public double getArea() throws RuntimeException {
29
30     if(this.circulo != null && this.circulo.getRaio() > 0 && this.altura > 0){
31         return circulo.getArea() + 2 * Math.PI * this.circulo.getRaio() * this.altura;
32         //return 2 * circulo.getArea() + 2 * Math.PI * this.circulo.getRaio() * this.altura;
33     } else{
34         throw new RuntimeException(message:"Dados inconsistentes");
35     }
36 }
```

The screenshot displays the 'TEST RESULTS' tab in an IDE. The left pane lists several test cases, including '%TESTE 6, quandoCirculoNullAlturaNegativa' and '%TESTES 7, quandoRaioNegativoEAlturaPositiva'. The right pane shows a detailed view of a failed test run at 11/22/2023, 8:35:44 AM. The test 'quandoRaioEAlturaPositivo()' is highlighted, showing an 'AssertionFailedError' where the expected value [75.39] did not match the actual value [62.83185307179586]. Below this, a list of other tests is shown with green checkmarks, indicating they passed.

PROBLEMS OUTPUT TEST RESULTS TERMINAL PORTS DEBUG CONSOLE

%TESTE 6, quandoCirculoNullAlturaNegativa (ifmt.cba.negocio.CilindroTest)

%TESTES 7, quandoRaioNegativoEAlturaPositiva (ifmt.cba.negocio.CilindroTest)

%TESTE 7, quandoRaioNegativoEAlturaPositiva (ifmt.cba.negocio.CilindroTest)

%TESTES 8, quandoRaioNegativoEAlturaNegativa (ifmt.cba.negocio.CilindroTest)

%TESTE 8, quandoRaioNegativoEAlturaNegativa (ifmt.cba.negocio.CilindroTest)

Test run at 11/22/2023, 8:35:44 AM

quandoRaioEAlturaPositivo()

Expected [75.39] but was [62.83185307179586]

org.opentest4j.AssertionFailedError: expected: [75.39] but was: [62.83185307179586]

quandoCirculoNullAlturaNegativa()

quandoCirculoNullAlturaPositiva()

quandoRaioNegativoEAlturaNegativa()

quandoRaioPositivoEAlturaNegativa()

quandoRaioNegativoEAlturaPositiva()

Test run at 11/22/2023, 8:25:43 AM

Test run at 11/22/2023, 8:23:17 AM

Object Mocks

Object Mock foi o termo usado para objetos que simulam o comportamento de outros objetos.

Esse termo foi traduzido para **Objeto Dublê**, que são objetos que fingem ser outros objetos.

Usaremos o **Framework Mockito para isolar dependências** entre as unidades em procedimentos de testes.

Também pode ser usado para simular uma implementação que ainda não existe, apoiando o TDD (Desenvolvimento Orientado a Testes).

Exemplos de Implementações de Object Mock:

- Java: JMockit, Mockito, EasyMock, JMock, MockCreator, MockLib e HibernateMock.
- .NET: NMockLib, Rhino Mocks, NMock e NMock 2 TypeMock.
- Ruby: Mocha, RSpec e FlexMock.
- PHP: SimpleTest, Yay! Mock, SnapTest e PHPUnit.

Dependência do Mockito

Essa distribuição é a recomendada para o uso com o framework JUnit 5.

```
1 <!-- https://mvnrepository.com/artifact/org.mockito/mockito-junit-jupiter -->
2 <dependency>
3   <groupId>org.mockito</groupId>
4   <artifactId>mockito-junit-jupiter</artifactId>
5   <version>5.6.0</version>
6   <scope>test</scope>
7 </dependency>
```

Código 1: Dependência Maven Mockito

Usando o Mockito para Isolar uma Dependência (1)

Provocando Erro na classe Circulo (Erro na unidade dependente)

```
public double getArea() throws RuntimeException {  
    if (this.raio > 0) {  
        return Math.PI * Math.pow(this.raio, 3); //provoca erro  
        //return Math.PI * Math.pow(this.raio, 2);  
    } else {  
        throw new RuntimeException(message:"Raio com valor inconsistente");  
    }  
}
```

PROBLEMS	OUTPUT	TEST RESULTS	TERMINAL	PORTS	DEBUG CONSOLE
					✓ Test run at 11/22/2023, 8:49:13 AM
					✓ quandoRaioEAlturaPositivo()
					Expected [75.39] but was [100.53096491487338] org.opentest4j.AssertionFailedError: expected: [75.39] but was: [100.53096491487338]
					✓ quandoCirculoNullAlturaNegativa()
					✓ quandoCirculoNullAlturaPositiva()
					✓ quandoRaioNegativoEAlturaNegativa()
					✓ quandoRaioPositivoEAlturaNegativa()
					✓ quandoRaioNegativoEAlturaPositiva()
					> ✓ Test run at 11/22/2023, 8:46:43 AM

Terminal output:

```
%TESTE 6,quandoCirculoNullAlturaNegativa  
(ifmt.cba.negocio.CilindroTest)  
  
%TESTS 7,quandoRaioNegativoEAlturaPositi  
va(ifmt.cba.negocio.CilindroTest)  
  
%TESTE 7,quandoRaioNegativoEAlturaPositi  
va(ifmt.cba.negocio.CilindroTest)  
  
%TESTS 8,quandoRaioNegativoEAlturaNegati  
va(ifmt.cba.negocio.CilindroTest)  
  
%TESTE 8,quandoRaioNegativoEAlturaNegati  
va(ifmt.cba.negocio.CilindroTest)
```

Usando o Mockito para Isolar uma Dependência (1)

Classe de teste CilindroComMockitoTest

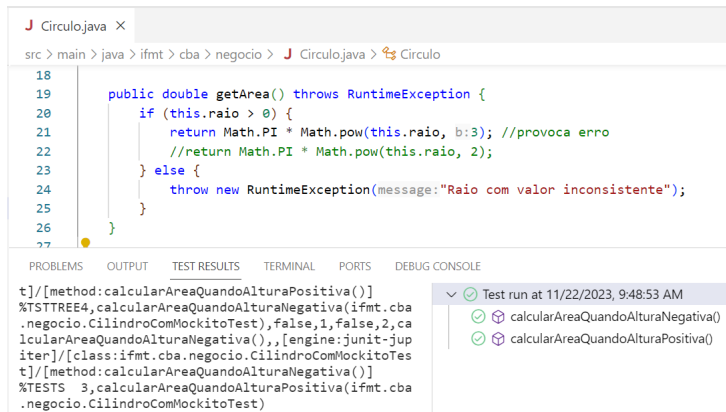
```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.*;
4 import static org.mockito.Mockito.mock;
5 import static org.mockito.Mockito.when;
6 import org.junit.jupiter.api.BeforeAll;
7 import org.junit.jupiter.api.Test;
8
9 public class CilindroComMockitoTest {
10
11     private static Circulo circuloFalso;
12
13     @BeforeAll
14     public static void inicializacao() {
15         // criando o objeto duble
16         circuloFalso = mock(Circulo.class);
17         // ensinando ele a responder os metodos que serao usados
18         when(circuloFalso.getRaio()).thenReturn(2.0);
19         when(circuloFalso.getArea()).thenReturn(12.56);
20     }
21
22     @Test
23     public void calcularAreaQuandoAlturaPositiva() {
24         Cilindro cilindro = new Cilindro(circuloFalso, 4);
25         assertEquals(75.39, cilindro.getArea(), 0.01);
26     }
27
28     @Test
29     public void calcularAreaQuandoAlturaNegativa() {
```

Usando o Mockito para Isolar uma Dependência (2)

```
30     Cilindro cilindro = new Cilindro(circuloFalso, -4);
31     RuntimeException thrown = assertThrows(RuntimeException.class,
32         () -> cilindro.getArea(), "Esperava-se a producao de uma excecacao, mas nao ocorreu");
33     assertTrue(thrown.getMessage().contains("Dados inconsistentes"));
34 }
35 }
```

Usando o Mockito para Isolar uma Dependência

Execução do Teste CilindroComMockitoTest



The screenshot shows an IDE window with a file named `Circulo.java`. The code defines a `getArea()` method that throws a `RuntimeException` if the radius is non-positive, otherwise it calculates the area using `Math.PI * Math.pow(this.raio, 2)`. Below the code, the `TEST RESULTS` tab is active, showing the output of a test run. The output includes the test class `CilindroComMockitoTest` and the methods `calcularAreaQuandoAlturaPositiva()` and `calcularAreaQuandoAlturaNegativa()`, both of which passed. The test run was completed at 11/22/2023, 9:48:53 AM.

```
18
19 public double getArea() throws RuntimeException {
20     if (this.raio > 0) {
21         return Math.PI * Math.pow(this.raio, 2); //provoca erro
22         //return Math.PI * Math.pow(this.raio, 2);
23     } else {
24         throw new RuntimeException(message:"Raio com valor inconsistente");
25     }
26 }
27
```

PROBLEMS OUTPUT TEST RESULTS TERMINAL PORTS DEBUG CONSOLE

t]/[method:calcularAreaQuandoAlturaPositiva()]
%TSTTREE4,calcularAreaQuandoAlturaNegativa(ifmt.cba
.negocio.CilindroComMockitoTest),false,1,false,2,ca
lcularAreaQuandoAlturaNegativa(),,[engine:junit-jup
iter]/[class:ifmt.cba.negocio.CilindroComMockitoTes
t]/[method:calcularAreaQuandoAlturaNegativa()]
%TESTS 3,calcularAreaQuandoAlturaPositiva(ifmt.cba
.negocio.CilindroComMockitoTest)

Test run at 11/22/2023, 9:48:53 AM

- calcularAreaQuandoAlturaNegativa()
- calcularAreaQuandoAlturaPositiva()

Usando o Mockito para Simular uma Implementação

Ao invés de criar um Mock para uma classe, será criado um Mock para uma Interface que representa a implementação futura.

```
1 package ifmt.cba.negocio;  
2  
3 public interface ICalculadora {  
4  
5     public double somar(double valor1, double valor2);  
6     public double subtrair(double valor1, double valor2);  
7     public double multiplicar(double valor1, double valor2);  
8     public double dividir(double valor1, double valor2);  
9 }
```


Usando o Mockito para Simular uma Implementação (1)

Mockito como dublê de uma Classe que ainda não existe.

```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.assertThrows;
5 import static org.junit.jupiter.api.Assertions.assertTrue;
6 import static org.mockito.Mockito.mock;
7 import static org.mockito.Mockito.when;
8
9 import org.junit.jupiter.api.BeforeAll;
10 import org.junit.jupiter.api.Test;
11
12 public class CalculadoraTest1 {
13
14     private static ICalculadora calculadora;
15
16     @BeforeAll
17     public static void inicializacao() {
18         calculadora = mock(ICalculadora.class);
19         when(calculadora.somar(10d, 20d)).thenReturn(30d);
20         when(calculadora.subtrair(30d, 10d)).thenReturn(20d);
21         when(calculadora.multiplicar(10d, 20d)).thenReturn(200d);
22         when(calculadora.dividir(40d, 20d)).thenReturn(2d);
23         when(calculadora.dividir(40d, 0d)).thenThrow(new ArithmeticException("Divisao por zero"));
24     }
25
26     @Test
27     public void testSomar() {
28         double resultado = calculadora.somar(10d, 20d);
29         assertEquals(30d, resultado);
30     }
31 }
```

Usando o Mockito para Simular uma Implementação (2)

```
30 }
31
32 @Test
33 public void testSubtracao() {
34     double resultado = calculadora.subtrair(30d, 10d);
35     assertEquals(20d, resultado);
36 }
37
38 @Test
39 public void testMultiplicacao() {
40     double resultado = calculadora.multiplicar(10d, 20d);
41     assertEquals(200d, resultado);
42 }
43
44 @Test
45 public void testDivisao() {
46     double resultado = calculadora.dividir(40d, 20d);
47     assertEquals(2d, resultado);
48 }
49
50 @Test
51 public void testDivisaoPorZero() {
52     ArithmeticException execucao = assertThrows(ArithmeticException.class,
53         () -> calculadora.dividir(40d, 0d), "Espera-se a producao de uma execucao");
54     assertTrue(execucao.getMessage().contains("Divisao por zero"));
55 }
56 }
```

Usando o Mockito para Mocks Parciais

Implementando parcialmente a interface.

Usando o Mockito como dublê para uma parte da classe.

```
1 package ifmt.cba.negocio;
2
3 public class CalculadoraImpl implements ICalculadora{
4
5     @Override
6     public double somar(double valor1, double valor2) {
7         return valor1 + valor2;
8     }
9
10    @Override
11    public double subtrair(double valor1, double valor2) {
12        return valor1 - valor2;
13    }
14
15    @Override
16    public double multiplicar(double valor1, double valor2) {
17        return valor1 * valor2;
18    }
19
20    @Override
21    public double dividir(double valor1, double valor2) {
22        return 0d;
23    }
24
25 }
```

Usando o Mockito para Mocks Parciais (1)

```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.assertThrows;
5 import static org.junit.jupiter.api.Assertions.assertTrue;
6 import static org.mockito.Mockito.*;
7 import static org.mockito.Mockito.when;
8
9 import org.junit.jupiter.api.BeforeAll;
10 import org.junit.jupiter.api.Test;
11
12 public class CalculadoraTest2 {
13
14     private static ICalculadora calculadora;
15
16     @BeforeAll
17     public static void inicializacao(){
18         calculadora = mock(CalculadoraImpl.class);
19         //when(calculadora.somar(10d, 20d)).thenReturn(30d);
20         //when(calculadora.subtrair(30d, 10d)).thenReturn(20d);
21         //when(calculadora.multiplicar(10d, 20d)).thenReturn(200d);
22         when(calculadora.dividir(40d, 20d)).thenReturn(2d);
23         when(calculadora.dividir(40d, 0d)).thenThrow(new ArithmeticException("Divisao por zero"));
24     }
25
26     @Test
27     public void testSomar(){
28         double resultado = calculadora.somar(20, 20);
29         assertEquals(40d, resultado);
30     }
31 }
```

Usando o Mockito para Mocks Parciais (2)

```
32 @Test
33 public void testSubtracao(){
34     double resultado = calculadora.subtrair(40d, 10d);
35     assertEquals(30d, resultado);
36 }
37
38 @Test
39 public void testMultiplicacao(){
40     double resultado = calculadora.multiplicar(20d, 20d);
41     assertEquals(400d, resultado);
42 }
43
44 @Test
45 public void testDivisao(){
46     double resultado = calculadora.dividir(40d, 20d);
47     assertEquals(2d, resultado);
48 }
49
50 @Test
51 public void testDivisaoPorZero(){
52     ArithmeticException excecao = assertThrows(ArithmeticException.class,
53         ()-> calculadora.dividir(40d, 0d), "Espera-se a producao de uma excecao");
54     assertTrue(excecao.getMessage().contains("Divisao por zero"));
55 }
56 }
```

Usando o Mockito para Mocks Parciais (1)

The screenshot shows an IDE window with a Java file named `CalculadoraTest2.java`. The code defines a test method `testSomar()` that calls `calculadora.somar(20, 20)` and asserts the result is 40.0. The test fails because the actual result is 0.0. Below the code, a red banner displays the error message: "Expected [40.0] but was [0.0] testSomar()". A table below the banner compares the expected value (40.0) with the actual value (0.0). At the bottom, the "TEST RESULTS" tab shows a stack trace of the failure, and the "TEST RESULTS" sidebar lists the test outcomes: `testMultiplicacao()` failed, `testSomar()` failed, `testSubtracao()` failed, `testDivisao()` passed, and `testDivisaoPorZero()` passed.

```
src > test > java > ifmt > cba > negocio > CalculadoraTest2.java > CalculadoraTest2 > testSomar()

24     }
25
26     @Test
27     public void testSomar(){
28         double resultado = calculadora.somar(valor1:20, valor2:20);
29         assertEquals(expected:40d, resultado); Expected [40.0] but was [0.0]

Expected [40.0] but was [0.0] testSomar()

Expected: 40.0
Actual: 0.0

PROBLEMS OUTPUT TEST RESULTS TERMINAL PORTS DEBUG CONSOLE

tEquals.java:65)
    at org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:885)
    at ifmt.cba.negocio.CalculadoraTest2.testMultiplicacao(CalculadoraTest2.java:41)
    at java.base/java.lang.reflect.Method.invoke(Method.java:578)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

%TRACEE

%TESTE 7,testMultiplicacao(ifmt.cba.negocio.CalculadoraTest2)

%DI INTIME121E

Test run at 11/7/2023, 12:12:07 PM
  ✖ testMultiplicacao()
    Expected [400.0] but was [0.0]
    org.opentest4j.AssertionFailedError: expected: [400.0]
  ✖ testSomar()
    Expected [40.0] but was [0.0]
    org.opentest4j.AssertionFailedError: expected: [40.0]
  ✖ testSubtracao()
    Expected [30.0] but was [0.0]
    org.opentest4j.AssertionFailedError: expected: [30.0]
  ✔ testDivisao()
  ✔ testDivisaoPorZero()
```

Usando o Mockito para Mocks Parciais (1)

```
1 package ifmt.cba.negocio;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.assertThrows;
5 import static org.junit.jupiter.api.Assertions.assertTrue;
6 import static org.mockito.Mockito.spy;
7 import static org.mockito.Mockito.when;
8
9 import org.junit.jupiter.api.BeforeAll;
10 import org.junit.jupiter.api.Test;
11
12 public class CalculadoraTest3 {
13     private static ICalculadora calculadora;
14
15     @BeforeAll
16     public static void inicializacao(){
17         // calculadora = mock(CalculadoraImpl.class);
18         calculadora = spy(CalculadoraImpl.class);
19         //when(calculadora.somar(10d, 20d)).thenReturn(30d);
20         //when(calculadora.subtrair(30d, 10d)).thenReturn(20d);
21         //when(calculadora.multiplicar(10d, 20d)).thenReturn(200d);
22         when(calculadora.dividir(40d, 20d)).thenReturn(2d);
23         when(calculadora.dividir(40d, 0d)).thenThrow(new ArithmeticException("Divisao por zero"));
24     }
25
26     @Test
27     public void testSomar(){
28         double resultado = calculadora.somar(20, 20);
29         assertEquals(40d, resultado);
30     }
31 }
```

Usando o Mockito para Mocks Parciais (2)

```
32 @Test
33 public void testSubtracao(){
34     double resultado = calculadora.subtrair(40d, 10d);
35     assertEquals(30d, resultado);
36 }
37
38 @Test
39 public void testMultiplicacao(){
40     double resultado = calculadora.multiplicar(20d, 20d);
41     assertEquals(400d, resultado);
42 }
43
44 @Test
45 public void testDivisao(){
46     double resultado = calculadora.dividir(40d, 20d);
47     assertEquals(2d, resultado);
48 }
49
50 @Test
51 public void testDivisaoPorZero(){
52     ArithmeticException excecao = assertThrows(ArithmeticException.class,
53         ()-> calculadora.dividir(40d, 0d), "Espera-se a producao de uma excecao");
54     assertTrue(excecao.getMessage().contains("Divisao por zero"));
55 }
56 }
```


Teste Unitário - Classe GrupoAlimentar (1)

Exemplo de unidade independente

```
1 package ifmt.cba.entity;  
2  
3 import org.apache.commons.lang3.builder.ToStringBuilder;  
4 import org.apache.commons.lang3.builder.ToStringStyle;  
5  
6 import jakarta.persistence.Column;  
7 import jakarta.persistence.Entity;  
8 import jakarta.persistence.GeneratedValue;  
9 import jakarta.persistence.GenerationType;  
10 import jakarta.persistence.Id;  
11 import jakarta.persistence.Table;  
12  
13 @Entity  
14 @Table (name = "grupo_alimentar")  
15 public class GrupoAlimentar {  
16  
17     @Id  
18     @GeneratedValue(strategy = GenerationType.IDENTITY)  
19     private int codigo;  
20  
21     @Column(name = "nome", length = 40)  
22     private String nome;  
23  
24     public int getCodigo() {  
25         return codigo;  
26     }  
27  
28     public void setCodigo(int codigo) {  
29         this.codigo = codigo;  
30     }  
31 }
```

Teste Unitário - Classe GrupoAlimentar (2)

```
30     }
31
32     public String getNome() {
33         return nome;
34     }
35
36     public void setNome(String nome) {
37         this.nome = nome;
38     }
39
40     @Override
41     public String toString() {
42         return ToStringBuilder.reflectionToString(this, ToStringStyle.JSON_STYLE);
43     }
44
45     @Override
46     public int hashCode() {
47         final int prime = 31;
48         int result = 1;
49         result = prime * result + codigo;
50         return result;
51     }
52
53     @Override
54     public boolean equals(Object obj) {
55         if (this == obj)
56             return true;
57         if (obj == null)
58             return false;
59         if (getClass() != obj.getClass())
60             return false;
61         GrupoAlimentar other = (GrupoAlimentar) obj;
```

Teste Unitário - Classe GrupoAlimentar (3)

```
62         if (codigo != other.codigo)
63             return false;
64         return true;
65     }
66
67     public String validar(){
68         String retorno = "";
69
70         if (this.nome == null || this.nome.length() < 3){
71             retorno += "Nome nao valido";
72         }
73
74         return retorno;
75     }
76 }
```

Teste Unitário - Classe GrupoAlimentar (1)

Exemplo de unidade independente

```
1 package ifmt.cba;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.Test;
5 import ifmt.cba.entity.GrupoAlimentar;
6
7 public class GrupoAlimentarTest {
8
9     @Test
10     public void validarDadosValidos() {
11         GrupoAlimentar grupoAlimentar = new GrupoAlimentar();
12         grupoAlimentar.setNome("Carboidratos");
13         String resultadoValidacao = grupoAlimentar.validar();
14         Assertions.assertTrue(resultadoValidacao.isEmpty());
15     }
16
17     @Test
18     public void validarDadosInvalidos() {
19         GrupoAlimentar grupoAlimentar = new GrupoAlimentar();
20         grupoAlimentar.setNome("C");
21         String resultadoValidacao = grupoAlimentar.validar();
22         Assertions.assertFalse(resultadoValidacao.isEmpty());
23     }
24 }
```

Teste Unitário - Classe Produto (1)

Exemplo de unidades com relação de dependência

```
1 package ifmt.cba.entity;  
2  
3 import org.apache.commons.lang3.builder.ToStringBuilder;  
4 import org.apache.commons.lang3.builder.ToStringStyle;  
5  
6 import jakarta.persistence.Column;  
7 import jakarta.persistence.Entity;  
8 import jakarta.persistence.FetchType;  
9 import jakarta.persistence.GeneratedValue;  
10 import jakarta.persistence.GenerationType;  
11 import jakarta.persistence.Id;  
12 import jakarta.persistence.JoinColumn;  
13 import jakarta.persistence.ManyToOne;  
14 import jakarta.persistence.Table;  
15  
16 @Entity  
17 @Table(name = "produto")  
18 public class Produto {  
19  
20     @Id  
21     @GeneratedValue(strategy = GenerationType.IDENTITY)  
22     private int codigo;  
23  
24     @Column(name = "nome")  
25     private String nome;  
26  
27     @Column(name = "custo_unidade")  
28     private float custoUnidade;  
29
```

Teste Unitário - Classe Produto (2)

```
30 @Column(name = "valor_energetico")
31 private int valorEnergetico;
32
33 @Column(name = "estoque")
34 private int estoque;
35
36 @Column(name = "estoque_minimo")
37 private int estoqueMinimo;
38
39 @ManyToOne(fetch = FetchType.EAGER)
40 @JoinColumn(name = "id_grupo")
41 private GrupoAlimentar grupoAlimentar;
42
43 public int getCodigo() {
44     return codigo;
45 }
46
47 public void setCodigo(int codigo) {
48     this.codigo = codigo;
49 }
50
51 public String getNome() {
52     return nome;
53 }
54
55 public void setNome(String nome) {
56     this.nome = nome;
57 }
58
59 public float getCustoUnidade() {
60     return custoUnidade;
61 }
```

Teste Unitário - Classe Produto (3)

```
62  
63 public void setCustoUnidade(float custoUnidade) {  
64     this.custoUnidade = custoUnidade;  
65 }  
66  
67 public int getValorEnergetico() {  
68     return valorEnergetico;  
69 }  
70  
71 public void setValorEnergetico(int valorEnergetico) {  
72     this.valorEnergetico = valorEnergetico;  
73 }  
74  
75 public GrupoAlimentar getGrupoAlimentar() {  
76     return grupoAlimentar;  
77 }  
78  
79 public void setGrupoAlimentar(GrupoAlimentar grupoAlimentar) {  
80     this.grupoAlimentar = grupoAlimentar;  
81 }  
82  
83 public int getEstoque() {  
84     return estoque;  
85 }  
86  
87 public void setEstoque(int estoque) {  
88     this.estoque = estoque;  
89 }  
90  
91 public int getEstoqueMinimo() {  
92     return estoqueMinimo;  
93 }
```

Teste Unitário - Classe Produto (4)

```
94 public void setEstoqueMinimo(int estoqueMinimo) {
95     this.estoqueMinimo = estoqueMinimo;
96 }
97
98 @Override
99 public String toString() {
100     return ToStringBuilder.reflectionToString(this, ToStringStyle.JSON_STYLE);
101 }
102
103 @Override
104 public int hashCode() {
105     final int prime = 31;
106     int result = 1;
107     result = prime * result + codigo;
108     return result;
109 }
110
111 @Override
112 public boolean equals(Object obj) {
113     if (this == obj)
114         return true;
115     if (obj == null)
116         return false;
117     if (getClass() != obj.getClass())
118         return false;
119     Produto other = (Produto) obj;
120     if (codigo != other.codigo)
121         return false;
122     return true;
123 }
124
125
```


Teste Unitário - Classe Produto (5)

```
26 public String validar() {
27     String retorno = "";
28
29     if (this.nome == null || this.nome.length() < 3) {
30         retorno += "Nome invalido";
31     }
32
33     if (this.custoUnidade <= 0) {
34         retorno += "Custo por unidade invalido";
35     }
36
37     if (this.valorEnergetico < 0) {
38         retorno += "Valor energetico invalido";
39     }
40
41     if (estoque < 0) {
42         retorno += "Estoque invalido";
43     }
44
45     if (this.grupoAlimentar == null || !this.grupoAlimentar.validar().isEmpty()) {
46         retorno += "Grupo alimentar invalido";
47     }
48
49     return retorno;
50 }
51 }
```

Teste Unitário - Classe GrupoAlimentar (1)

Exemplo de unidades com relação de dependência

```
1 package ifmt.cba;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.Test;
5 import org.mockito.Mockito;
6
7 import ifmt.cba.entity.GrupoAlimentar;
8 import ifmt.cba.entity.Produto;
9
10 public class ProdutoTest {
11
12     @Test
13     public void validarDadosValidos() {
14         GrupoAlimentar grupoAlimentar = new GrupoAlimentar();
15         grupoAlimentar.setCodigo(15);
16         grupoAlimentar.setNome("Proteínas");
17
18         Produto produto = new Produto();
19         produto.setCodigo(10);
20         produto.setNome("Produto Teste");
21         produto.setEstoque(100);
22         produto.setEstoqueMinimo(10);
23         produto.setValorEnergetico(50);
24         produto.setCustoUnidade(2.0f);
25         produto.setGrupoAlimentar(grupoAlimentar);
26
27         Assertions.assertTrue(produto.validar().isEmpty());
28     }
29 }
```

Teste Unitário - Classe GrupoAlimentar (2)

```
30 @Test
31 public void validarDadosGrupoAlimentarInvalido() {
32     GrupoAlimentar grupoAlimentar = new GrupoAlimentar();
33     grupoAlimentar.setCodigo(15);
34     grupoAlimentar.setNome("Pr"); //falha na validacao de Produto
35
36     Produto produto = new Produto();
37     produto.setCodigo(10);
38     produto.setNome("Produto Teste");
39     produto.setEstoque(100);
40     produto.setEstoqueMinimo(10);
41     produto.setValorEnergetico(50);
42     produto.setCustoUnidade(2.0f);
43     produto.setGrupoAlimentar(grupoAlimentar);
44
45     Assertions.assertFalse(produto.validar().isEmpty());
46 }
47
48 @Test
49 public void validarMockandoGrupoAlimentar() {
50
51     GrupoAlimentar grupoAlimentar = Mockito.mock(GrupoAlimentar.class);
52     Mockito.when(grupoAlimentar.validar()).thenReturn("");
53
54     Produto produto = new Produto();
55     produto.setCodigo(10);
56     produto.setNome("Produto Teste");
57     produto.setEstoque(100);
58     produto.setEstoqueMinimo(10);
59     produto.setValorEnergetico(50);
60     produto.setCustoUnidade(2.0f);
61     produto.setGrupoAlimentar(grupoAlimentar);
```

Teste Unitário - Classe GrupoAlimentar (3)

```
62  
63     Assertions.assertTrue(produtos.validar().isEmpty());  
64 }  
65 }
```

Mocks de Métodos com Retorno void

Classe de negócio Contador

```
1 package ifmt.cba.negocio;  
2  
3 public class Contador {  
4  
5     private int contagem;  
6  
7     public Contador(){  
8         this.contagem = 0;  
9     }  
10  
11     public void contar(){  
12         this.contagem++;  
13     }  
14  
15     public int getContagem(){  
16         return this.contagem;  
17     }  
18 }
```

Mocks de Métodos com Retorno void (1)

Teste da Classe de negócio Contador

```
1 package ifmt.cba.negocio;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.Test;
5 import org.mockito.Mockito;
6
7 public class ContadorTest {
8
9     @Test
10     public void validarContador1() {
11         Contador contador = Mockito.mock(Contador.class);
12         Mockito.when(contador.getContagem()).thenReturn(1);
13         // Mockito.when(contador.contar()).thenReturn(?); //nao aceita metodo com retorno void
14
15         contador.contar();
16         Mockito.verify(contador).contar();
17         Assertions.assertEquals(1, contador.getContagem());
18     }
19
20     @Test
21     public void validarContador2() {
22         Contador contador = Mockito.mock(Contador.class);
23         Mockito.when(contador.getContagem()).thenReturn(3);
24         contador.contar();
25         contador.contar();
26         contador.contar();
27         Mockito.verify(contador, Mockito.times(3)).contar();
28
29         Assertions.assertEquals(3, contador.getContagem());
30     }
31 }
```

Mocks de Métodos com Retorno void (2)

```
30     }  
31 }
```

Mocks de Métodos com Retorno void

Classe de negócio Estatística

```
1 package ifmt.cba.negocio;
2
3 public class Estatistica {
4
5     private Contador numMenor18;
6     private Contador numMaior18;
7
8     public Estatistica(Contador numMenor18, Contador numMaior18) {
9         this.numMenor18 = numMenor18;
10        this.numMaior18 = numMaior18;
11    }
12
13    public int incrementaMenor18() {
14        this.numMenor18.contar();
15        return this.numMenor18.getContagem();
16    }
17
18    public int incrementaMaior18() {
19        this.numMaior18.contar();
20        return this.numMaior18.getContagem();
21    }
22
23    public float getPercentualMenor18() {
24        float soma = this.numMenor18.getContagem() + this.numMaior18.getContagem();
25        return this.numMenor18.getContagem() / soma * 100;
26    }
27
28    public float getPercentualMaior18() {
29        float soma = this.numMenor18.getContagem() + this.numMaior18.getContagem();
30        return this.numMaior18.getContagem() / soma * 100;
31    }
32 }
```


Mocks de Métodos com Retorno void (1)

Teste da Classe de negócio Estatística

```
1 package ifmt.cba.negocio;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.Test;
5 import org.mockito.Mockito;
6
7 public class EstatisticaTest {
8
9     @Test
10     public void testValidarEstatisticaSemMockDependencia() {
11         Contador menor18 = new Contador();
12         Contador maior18 = new Contador();
13
14         Estatistica estatistica = new Estatistica(menor18, maior18);
15
16         estatistica.incrementaMenor18();
17         estatistica.incrementaMenor18();
18         estatistica.incrementaMenor18();
19         estatistica.incrementaMenor18();
20
21         estatistica.incrementaMaior18();
22         estatistica.incrementaMaior18();
23         estatistica.incrementaMaior18();
24         estatistica.incrementaMaior18();
25         estatistica.incrementaMaior18();
26         estatistica.incrementaMaior18();
27
28         Assertions.assertEquals(40, estatistica.getPercentualMenor18(), 001);
29         Assertions.assertEquals(60, estatistica.getPercentualMaior18(), 001);
```

Mocks de Métodos com Retorno void (2)

```
30 }
31
32 @Test
33 public void testValidarEstatisticaComMockDependencia () {
34
35     Contador menor18 = Mockito.mock(Contador.class);
36     Mockito.when(menor18.getContagem()).thenReturn(4);
37
38     Contador maior18 = Mockito.mock(Contador.class);
39     Mockito.when(maior18.getContagem()).thenReturn(6);
40
41     // Mockito.when(contador.contar()).thenReturn(?); //nao aceita metodo com retorno void
42
43     Estatistica estatistica = new Estatistica(menor18, maior18);
44
45     estatistica.incrementaMenor18();
46     estatistica.incrementaMenor18();
47     estatistica.incrementaMenor18();
48     estatistica.incrementaMenor18();
49     Mockito.verify(menor18, Mockito.times(4)).contar();
50
51     estatistica.incrementaMaior18();
52     estatistica.incrementaMaior18();
53     estatistica.incrementaMaior18();
54     estatistica.incrementaMaior18();
55     estatistica.incrementaMaior18();
56     estatistica.incrementaMaior18();
57     Mockito.verify(maior18, Mockito.times(6)).contar();
58
59     Assertions.assertEquals(40, estatistica.getPercentualMenor18(), 001);
60     Assertions.assertEquals(60, estatistica.getPercentualMaior18(), 001);
61 }
```

Mocks de Métodos com Retorno void (3)

62 }

Teste Unitário - Classe GrupoAlimentarNegocio (1)

Exemplo de unidades com relação de dependência (GrupoAlimentarDAO)

```
1 package ifmt.cba.negocio ;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.modelmapper.ModelMapper;
7
8 import ifmt.cba.dto.GrupoAlimentarDTO;
9 import ifmt.cba.entity.GrupoAlimentar;
10 import ifmt.cba.persistencia.GrupoAlimentarDAO;
11 import ifmt.cba.persistencia.PersistenciaException;
12 import ifmt.cba.persistencia.ProdutoDAO;
13
14 public class GrupoAlimentarNegocio {
15
16     private ModelMapper modelMapper;
17     private GrupoAlimentarDAO grupoAlimentarDAO;
18     private ProdutoDAO produtoDAO;
19
20     public GrupoAlimentarNegocio(GrupoAlimentarDAO grupoAlimentarDAO, ProdutoDAO produtoDAO) throws NegocioException {
21
22         this.grupoAlimentarDAO = grupoAlimentarDAO;
23         this.produtoDAO = produtoDAO;
24
25         this.modelMapper = new ModelMapper();
26     }
27
28     public void inserir(GrupoAlimentarDTO grupoAlimentarDTO) throws NegocioException {
29
```

Teste Unitário - Classe GrupoAlimentarNegocio (2)

```
30 GrupoAlimentar grupoAlimentar = this.toEntity(grupoAlimentarDTO);
31 String mensagemErros = grupoAlimentar.validar();
32
33 if (!mensagemErros.isEmpty()) {
34     throw new NegocioException(mensagemErros);
35 }
36
37 try {
38     // nao pode existir outro com o mesmo nome
39     if (!grupoAlimentarDAO.buscarPorParteNome(grupoAlimentar.getNome()).isEmpty()) {
40         throw new NegocioException("Ja existe esse grupo alimentar");
41     }
42     grupoAlimentarDAO.beginTransaction();
43     grupoAlimentarDAO.incluir(grupoAlimentar);
44     grupoAlimentarDAO.commitTransaction();
45 } catch (PersistenciaException ex) {
46     grupoAlimentarDAO.rollbackTransaction();
47     throw new NegocioException("Erro ao incluir o grupo alimentar - " + ex.getMessage());
48 }
49 }
```

Teste Unitário - Classe GrupoAlimentarNegocio (1)

Dependência - DAO (superclasse)

```
1 package ifmt.cba.persistencia;  
2  
3 import jakarta.persistence.EntityManager;  
4  
5 public class DAO<VO> {  
6  
7     protected EntityManager entityManager;  
8  
9     public DAO(EntityManager entityManager) throws PersistenciaException {  
10         this.entityManager = entityManager;  
11     }  
12  
13     public void incluir(VO vo) throws PersistenciaException {  
14         try {  
15             this.entityManager.persist(vo);  
16         } catch (Exception e) {  
17             throw new PersistenciaException("Erro ao incluir " + vo.getClass() + " - " + e.getMessage());  
18         }  
19     }
```

Teste Unitário - Classe GrupoAlimentarNegocio (1)

Dependência - GrupoAlimentarDAO

```
1 package ifmt.cba.persistencia;
2
3 import java.util.List;
4 import ifmt.cba.entity.GrupoAlimentar;
5 import jakarta.persistence.EntityManager;
6 import jakarta.persistence.Query;
7
8
9 public class GrupoAlimentarDAO extends DAO<GrupoAlimentar> {
10
11     public GrupoAlimentarDAO(EntityManager entityManager) throws PersistenciaException {
12         super(entityManager);
13     }
14
15     public GrupoAlimentar buscarPorCodigo(int codigo) throws PersistenciaException {
16
17         GrupoAlimentar grupoAlimentar = null;
18
19         try {
20             grupoAlimentar = this.entityManager.find(GrupoAlimentar.class, codigo);
21         } catch (Exception ex) {
22             throw new PersistenciaException("Erro na selecao por codigo - " + ex.getMessage());
23         }
24         return grupoAlimentar;
25     }
26
27     @SuppressWarnings("unchecked")
28     public List<GrupoAlimentar> buscarPorParteNome(String nome) throws PersistenciaException {
29         List<GrupoAlimentar> listaGrupoAlimentar;
```

Teste Unitário - Classe GrupoAlimentarNegocio (2)

```
30     try {
31         Query query = this.entityManager
32             .createQuery("SELECT ga FROM GrupoAlimentar ga WHERE UPPER(ga.nome) LIKE :pNome ORDER BY ga.nome");
33         query.setParameter("pNome", "%" + nome.toUpperCase().trim() + "%");
34         listaGrupoAlimentar = query.getResultList();
35     } catch (Exception ex) {
36         throw new PersistenciaException("Erro na selecao por parte do nome - " + ex.getMessage());
37     }
38     return listaGrupoAlimentar;
39 }
40 }
```


Teste Unitário - Classe GrupoAlimentarNegocio (1)

```
1 package ifmt.cba;
2
3 import java.util.ArrayList;
4
5 import org.junit.jupiter.api.Assertions;
6 import org.junit.jupiter.api.Test;
7 import org.mockito.Mockito;
8
9 import ifmt.cba.dto.GrupoAlimentarDTO;
10 import ifmt.cba.entity.GrupoAlimentar;
11 import ifmt.cba.negocio.GrupoAlimentarNegocio;
12 import ifmt.cba.negocio.NegocioException;
13 import ifmt.cba.persistencia.FabricaEntityManager;
14 import ifmt.cba.persistencia.GrupoAlimentarDAO;
15 import ifmt.cba.persistencia.ProdutoDAO;
16
17 public class GrupoAlimentarNegocioTest {
18
19     @Test
20     public void validarOperacaoInclusaoSemMockDAO1() {
21
22         GrupoAlimentarDAO grupoAlimentarDAO;
23         ProdutoDAO produtoDAO;
24         GrupoAlimentarNegocio grupoAlimentarNegocio;
25
26         grupoAlimentarDAO = Assertions.assertDoesNotThrow(
27             () -> new GrupoAlimentarDAO( FabricaEntityManager.getEntityManagerProducao() ) );
28         produtoDAO = Assertions.assertDoesNotThrow(
29             () -> new ProdutoDAO( FabricaEntityManager.getEntityManagerProducao() ) );
30
31         grupoAlimentarNegocio = Assertions.assertDoesNotThrow(
```

Teste Unitário - Classe GrupoAlimentarNegocio (2)

```
32         ()-> new GrupoAlimentarNegocio(grupoAlimentarDAO, produtoDAO));
33
34     GrupoAlimentarDTO grupoDTO = new GrupoAlimentarDTO();
35     grupoDTO.setNome("Teste de Inclusao"); //GrupoAlimentar valido, vai ate o banco
36
37     Assertions.assertDoesNotThrow(()-> grupoAlimentarNegocio.inserir(grupoDTO));
38 }
39
40 @Test
41 public void validarOperacaoInclusaoSemMockDAO2() {
42
43     GrupoAlimentarDAO grupoAlimentarDAO;
44     ProdutoDAO produtoDAO;
45     GrupoAlimentarNegocio grupoAlimentarNegocio;
46
47     grupoAlimentarDAO = Assertions.assertDoesNotThrow(
48         ()-> new GrupoAlimentarDAO(FabricaEntityManager.getEntityManagerProducao()));
49     produtoDAO = Assertions.assertDoesNotThrow(
50         ()-> new ProdutoDAO(FabricaEntityManager.getEntityManagerProducao()));
51
52     grupoAlimentarNegocio = Assertions.assertDoesNotThrow(
53         ()-> new GrupoAlimentarNegocio(grupoAlimentarDAO, produtoDAO));
54
55     GrupoAlimentarDTO grupoDTO = new GrupoAlimentarDTO();
56     grupoDTO.setNome("Te"); //Grupo alimentar invalido, para na regra de negocio e nao chega no banco
57
58     //recebe uma excecao de validacao, nao inclui
59     Assertions.assertThrows(NegocioException.class, ()-> grupoAlimentarNegocio.inserir(grupoDTO));
60 }
61
62 @Test
63 public void validarOperacaoInclusaoComMockDAO1() {
```

Teste Unitário - Classe GrupoAlimentarNegocio (3)

```
64 GrupoAlimentarDAO grupoAlimentarDAO ;
65 ProdutoDAO produtoDAO;
66 GrupoAlimentarNegocio grupoAlimentarNegocio;
67
68
69 //GrupoAlimentar valido , mas nao chega no banco pois DAO foi mockado
70 GrupoAlimentarDTO grupoDTO = new GrupoAlimentarDTO();
71 grupoDTO.setNome("Teste de Inclusao2");
72
73 grupoAlimentarDAO = Mockito.mock(GruoAlimentarDAO.class); //mocka do DAO
74 produtoDAO = Mockito.mock(ProdutoDAO.class); //mocka do DAO
75
76 //Mockito.when(grupoAlimentarDAO.incluir(grupo)).thenReturn(void); //nao mocka metodo void
77
78 Assertions.assertDoesNotThrow(
79     ()-> Mockito.when(grupoAlimentarDAO.buscarPorParteNome(grupoDTO.getNome()))
80     .thenReturn(new ArrayList<GrupoAlimentar>()));
81
82 grupoAlimentarNegocio = Assertions.assertDoesNotThrow(
83     ()-> new GrupoAlimentarNegocio(grupoAlimentarDAO, produtoDAO));
84
85 Assertions.assertDoesNotThrow(()-> grupoAlimentarNegocio.inserir(grupoDTO));
86
87 //verifica se o metodo void foi chamado uma vez
88 Assertions.assertDoesNotThrow(
89     ()->Mockito.verify(grupoAlimentarDAO).incluir(new GrupoAlimentar()));
90 }
91
92 @Test
93 public void validarOperacaoInclusaoComMockDAO2() {
94
95     GrupoAlimentarDAO grupoAlimentarDAO ;
```

Teste Unitário - Classe GrupoAlimentarNegocio (4)

```
96 ProdutoDAO produtoDAO;  
97 GrupoAlimentarNegocio grupoAlimentarNegocio;  
98  
99 //GrupoAlimentar invalido, nao chega a invocar o metodo incluir do DAO  
00 GrupoAlimentarDTO grupoDTO = new GrupoAlimentarDTO();  
01 grupoDTO.setNome("Te");  
02  
03 grupoAlimentarDAO = Mockito.mock(GruoAlimentarDAO.class); //mocka do DAO  
04 produtoDAO = Mockito.mock(ProdutoDAO.class); //mocka do DAO  
05  
06 //Mockito.when(grupoAlimentarDAO.incluir(grupo)).thenReturn(void); //nao mocka metodo void  
07  
08 Assertions.assertDoesNotThrow(  
09     () -> Mockito.when(grupoAlimentarDAO.buscarPorParteNome(grupoDTO.getNome()))  
10     .thenReturn(new ArrayList<GrupoAlimentar>()));  
11  
12 grupoAlimentarNegocio = Assertions.assertDoesNotThrow(  
13     () -> new GrupoAlimentarNegocio(grupoAlimentarDAO, produtoDAO));  
14  
15 //espera uma execucao de validacao dos dados  
16 Assertions.assertThrows(NegocioException.class, () -> grupoAlimentarNegocio.inserir(grupoDTO));  
17  
18 //verifica se o metodo void NAO foi chamado  
19 Assertions.assertDoesNotThrow(  
20     () -> Mockito.verify(grupoAlimentarDAO, Mockito.never()).incluir(new GrupoAlimentar()));  
21 }  
22 }
```