

# GESTÃO E QUALIDADE DE SOFTWARE

## QUALIDADE DE CÓDIGO E TESTES EM JAVA OPERAÇÕES COM PESO

**Aluna:** Lara Luísa Ayrolla Abreu

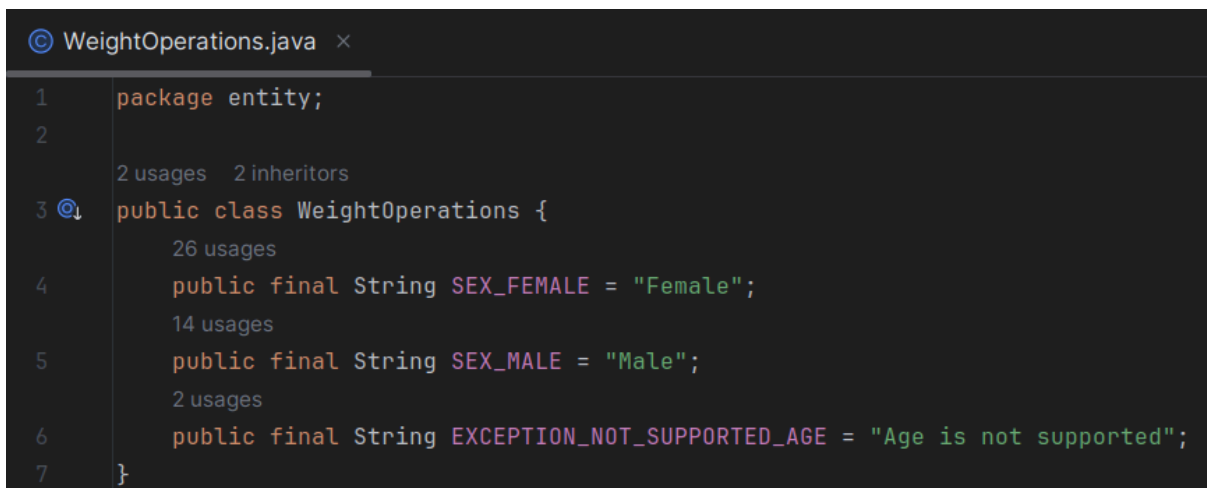
### 1. FUNCIONALIDADES DO CÓDIGO: OPERAÇÕES COM PESO

O código a ser testado foi desenvolvido em Java, no mesmo projeto em que os testes unitários foram desenvolvidos. Para acessar o repositório do código fonte, basta clicar [neste link](#).

No pacote “entity” foram criadas as classes “WeightOperations”, na qual estão contidas variáveis comuns entre as outras classes do mesmo pacote. Também foram criadas as classes “EnergyExpenditureOperations”, que faz cálculos de gasto energético, e “IdealWeightOperations”, que fazem verificações e cálculos de peso ideal, ambas estendendo a classe “WeightOperations”.

#### a. Classe “WeightOperations”

Ao começo da classe, foram criadas variáveis, que serão utilizadas ao longo do código pelos métodos das classes filhas. A classe não possui métodos próprios.



```
© WeightOperations.java x
1 package entity;
2
3 2 usages 2 inheritors
4 @ public class WeightOperations {
5     26 usages
6     public final String SEX_FEMALE = "Female";
7     14 usages
8     public final String SEX_MALE = "Male";
9     2 usages
10    public final String EXCEPTION_NOT_SUPPORTED_AGE = "Age is not supported";
11 }
```

Declaração da classe “WeightOperations” e de variáveis

## b. Classe “IdealWeightOperations”

A classe “IdealWeightOperations” estende a classe “WeightOperations”. Por conta disso, ela herda todas as variáveis. Além disso, ela define variáveis e constantes, ou variáveis do tipo final.

Quanto aos métodos e funções, eles têm as funcionalidades de classificar o peso de acordo com o IMC e calcular o peso ideal para pessoas de diferentes sexos e idades.

```
© IdealWeightOperations.java ×
1  package entity;
2
3  import java.util.Objects;
4
5  import static java.lang.Math.pow;
6
7  public class IdealWeightOperations extends WeightOperations {
8      public final String SITUATION_UNDERWEIGHT = "Underweight";
9      public final String SITUATION_NORMAL_WEIGHT = "Normal weight";
10     public final String SITUATION_OVERWEIGHT = "Overweight";
11     public final String SITUATION_OBESE = "Obese";
12     public final String SITUATION_MORBIDLY_OBESE = "Morbidly obese";
13
14     public double bmi;
15     public double idealBmi;
16 }
```

Declaração da classe “IdealWeightOperations”, constantes e variáveis

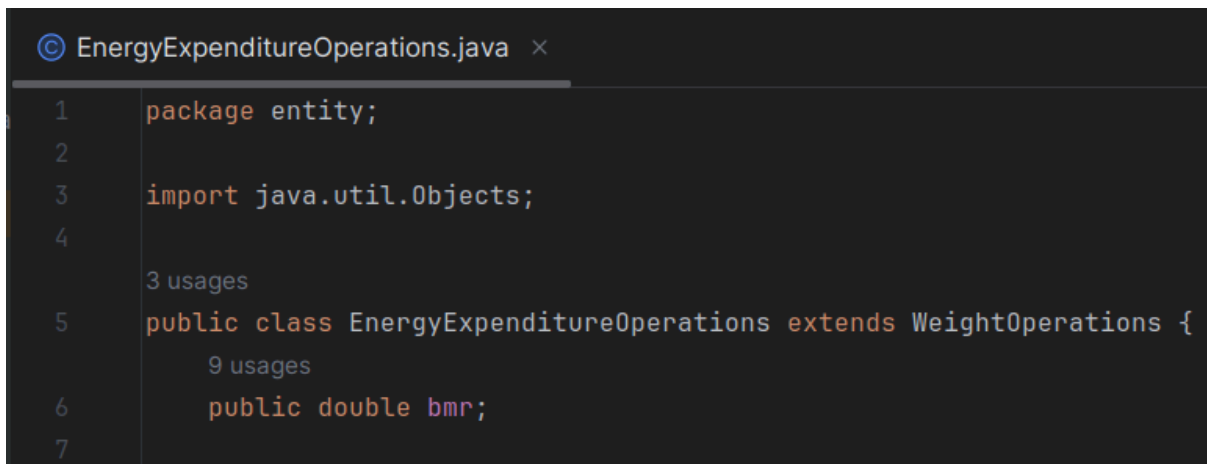
```
17 > public String classifyWeightAccordingToBmi(double weight, double height) {...}
38
39 > public double calculateIdealWeight(int age, String sex, double height) throws Exception {...}
94 }
```

Assinatura das funções da classe “IdealWeightOperations”

### c. Classe “EnergyExpenditureOperations”

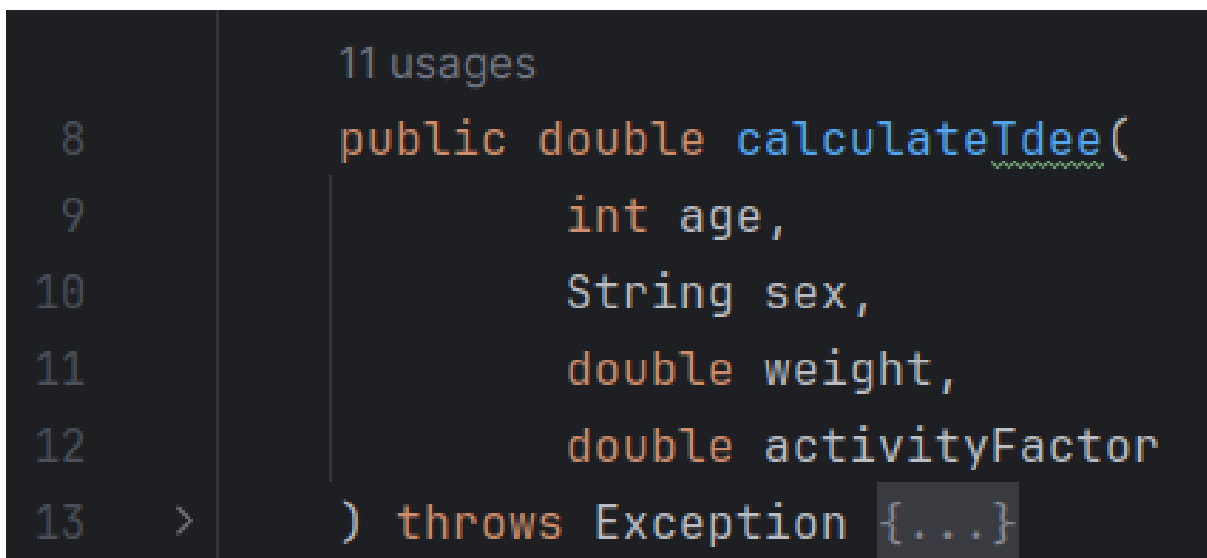
A classe “EnergyExpenditureOperations” estende a classe “WeightOperations”. Por conta disso, ela herda todas as variáveis. Além disso, ela declara a variável bmr, que será utilizada ao longo da classe.

Quanto à única função, ela tem a funcionalidade de calcular o gasto energético de pessoas de diferentes sexos, idades e níveis de atividade física.



```
© EnergyExpenditureOperations.java x
1 package entity;
2
3 import java.util.Objects;
4
5 3 usages
public class EnergyExpenditureOperations extends WeightOperations {
6     9 usages
    public double bmr;
7 }
```

Declaração da classe “IdealWeightOperations”, constantes e variáveis



```
11 usages
8 public double calculateTdee(
9     int age,
10    String sex,
11    double weight,
12    double activityFactor
13 > ) throws Exception {...}
```

Assinatura da função da classe “IdealWeightOperations”

## 2. TESTES DESENVOLVIDOS

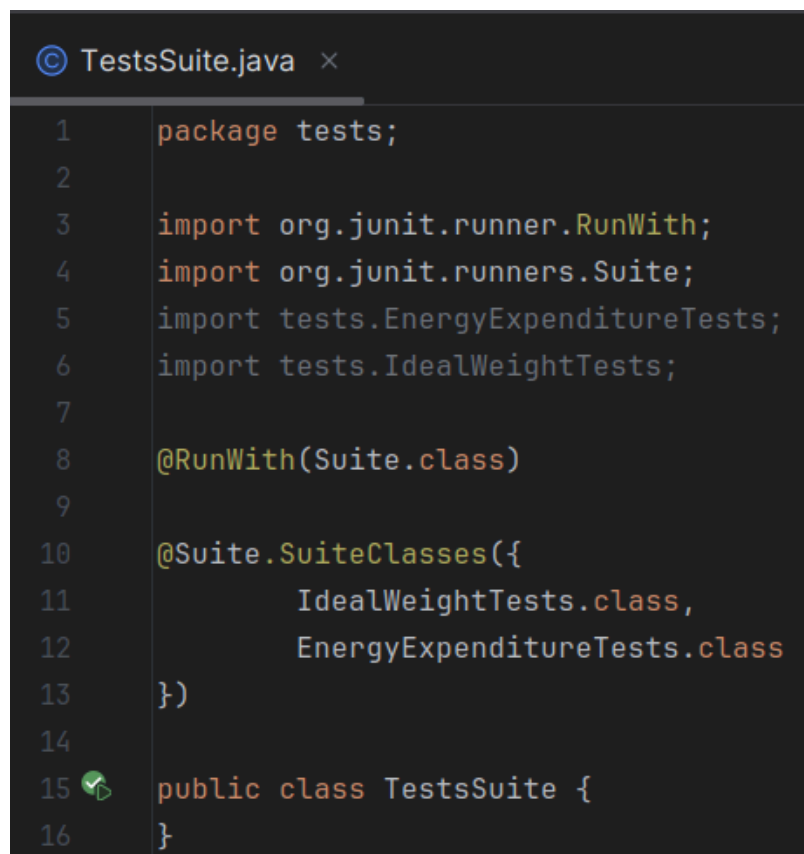
No pacote “tests”, foi criada a classe “TestSuit”, que é a responsável por agrupar e executar os testes de todas as classes do pacote. As classes “IdealWeightTests” e “EnergyExpenditureTests” contêm os métodos utilizados para testar as funções das classes “IdealWeightOperations” e “EnergyExpenditureOperations”.

Ao começo de todas as classes, são feitas as importações das classes que serão testadas e da biblioteca que será utilizada para o desenvolvimento dos testes ("JUnit").

Na classe "IdealWeightOperations" é declarada uma variável de nome "idealWeight" do tipo "IdealWeightOperations", que será utilizada para chamar as funções e acessar as variáveis e constantes da classe testada. Na classe "EnergyExpenditureTests" foi criada a variável "energyExpenditure" do tipo "EnergyExpenditureOperations" com o mesmo objetivo.

Foram desenvolvidos onze métodos, com trinta e oito casos de teste, que verificam se o funcionamento das funções das classes "IdealWeightOperations" e "EnergyExpenditureOperations" está correto. Isso é feito através de comparações dos retornos das funções testadas e das variáveis da classe com valores previamente definidos pelos testes como corretos ou incorretos.

Um teste passa somente se o resultado retornado pela função testada, quando recebe os valores predefinidos como parâmetro, coincide com o resultado esperado. Todos os casos de teste de um método de teste devem passar para que esse método passe.



```
© TestsSuite.java x
1 package tests;
2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5 import tests.EnergyExpenditureTests;
6 import tests.IdealWeightTests;
7
8 @RunWith(Suite.class)
9
10 @Suite.SuiteClasses({
11     IdealWeightTests.class,
12     EnergyExpenditureTests.class
13 })
14
15 public class TestsSuite {
16 }
```

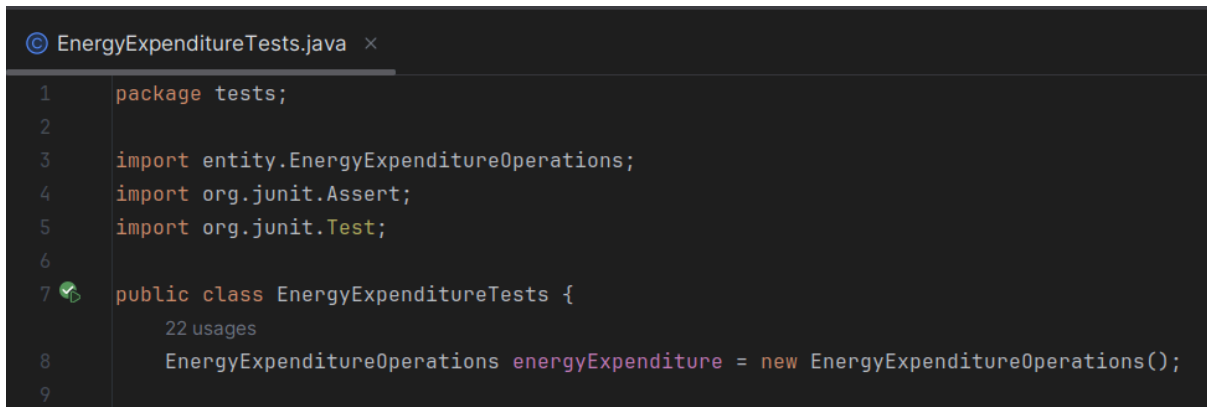
Declaração da classe "TestSuite", importações e chamada das outras classes de teste

```
© IdealWeightTests.java x
1 package tests;
2
3 import entity.IdealWeightOperations;
4 import org.junit.Assert;
5 import org.junit.Test;
6
7 public class IdealWeightTests {
8     IdealWeightOperations idealWeight = new IdealWeightOperations();
9 }
```

Declaração da classe “IdealWeightTests”, importações e criação de variável

```
10 @Test
11 public void shouldCorrectlyCalculateAndClassifyUnderweightBmi() {...}
12
13
14
15 @Test
16 public void shouldCorrectlyCalculateAndClassifyNormalWeightBmi() {...}
17
18
19
20 @Test
21 public void shouldCorrectlyCalculateAndClassifyOverweightBmi() {...}
22
23
24
25 @Test
26 public void shouldCorrectlyCalculateAndClassifyObeseBmi() {...}
27
28
29
30 @Test
31 public void shouldCorrectlyCalculateAndClassifyMorbidlyObeseBmi() {...}
32
33
34
35 @Test(expected = Exception.class)
36 public void shouldCorrectlyThrowExceptionForNotSupportedAges() throws Exception {...}
37
38
39
40 @Test
41 public void shouldCorrectlyCalculateIdealWeightForChildren() throws Exception {...}
42
43
44
45 @Test
46 public void shouldCorrectlyCalculateIdealWeightForAdultsUnder65() throws Exception {...}
47
48
49
50 @Test
51 public void shouldCorrectlyCalculateIdealWeightForAdultsOver65() throws Exception {...}
52
```

Assinatura dos métodos da classe “IdealWeightTests”

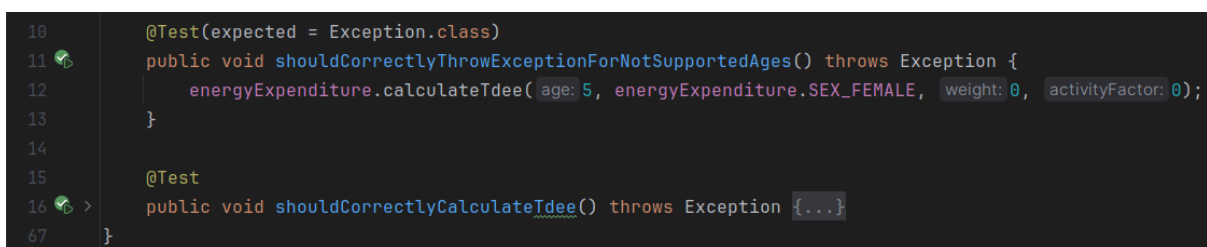


```

1 package tests;
2
3 import entity.EnergyExpenditureOperations;
4 import org.junit.Assert;
5 import org.junit.Test;
6
7 public class EnergyExpenditureTests {
8     22 usages
9     EnergyExpenditureOperations energyExpenditure = new EnergyExpenditureOperations();
10

```

Declaração da classe “EnergyExpenditureTests”, importações e criação de variável



```

10 @Test(expected = Exception.class)
11 public void shouldCorrectlyThrowExceptionForNotSupportedAges() throws Exception {
12     energyExpenditure.calculateTdee(age: 5, energyExpenditure.SEX_FEMALE, weight: 0, activityFactor: 0);
13 }
14
15 @Test
16 public void shouldCorrectlyCalculateTdee() throws Exception {...}
17 }

```

Assinatura dos métodos da classe “EnergyExpenditureTests”

### 3. RESULTADOS E CONCLUSÃO

#### 3.1. Complexidade cognitiva e ciclomática

Utilizando o plugin “MetricsReloaded” na IDE IntelliJ do Jet Brains, calculei as métricas de complexidade do código.

Considerando todo o código fonte do projeto, antes de refatorações, a média da complexidade cognitiva (CogC) foi 4.00 e a média da complexidade ciclomática (v(G)) foi 3.00. É importante ressaltar que as classes de teste, por serem menos complexas ajudam a diminuir a complexidade do projeto em geral.

Após refatorações, a média da complexidade cognitiva (CogC) diminuiu para 2.06 e a média da complexidade ciclomática (v(G)) diminuiu para 2.69.

Method metrics	Class metrics	Package metrics	Module metrics	Project metrics
method ^				
	CogC	ev(G)	iv(G)	v(G)
entity.EnergyExpenditureOperations.calculateTdee(int, String, double, double)	17	2	6	10
entity.IdealWeightOperations.calculateIdealWeight(int, String, double)	35	4	8	16
entity.IdealWeightOperations.classifyWeightAccordingToBmi(double, double)	4	5	1	5
tests.EnergyExpenditureTests.shouldCorrectlyCalculateTdee()	0	1	1	1
tests.EnergyExpenditureTests.shouldCorrectlyThrowExceptionForNotSupportedAges()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyMorbidlyObeseBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyNormalWeightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyObeseBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyOverweightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyUnderweightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForAdultsOver65()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForAdultsUnder65()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForChildren()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyThrowExceptionForNotSupportedAges()	0	1	1	1
<b>Total</b>	<b>56</b>	<b>22</b>	<b>26</b>	<b>42</b>
<b>Average</b>	<b>4.00</b>	<b>1.57</b>	<b>1.86</b>	<b>3.00</b>

Métricas de complexidade para o código inteiro, antes da refatoração

Method metrics	Class metrics	Package metrics	Module metrics	Project metrics
method ^				
	CogC	ev(G)	iv(G)	v(G)
entity.EnergyExpenditureOperations.calculateTdee(int, String, double, double)	13	2	1	9
entity.IdealWeightOperations.calculateIdealWeight(int, String, double)	6	3	2	6
entity.IdealWeightOperations.classifyWeightAccordingToBmi(double, double)	4	5	1	5
entity.IdealWeightOperations.defineFemaleIdealBmi(int)	5	6	1	6
entity.IdealWeightOperations.defineMaleIdealBmi(int)	5	6	1	6
tests.EnergyExpenditureTests.shouldCorrectlyCalculateTdee()	0	1	1	1
tests.EnergyExpenditureTests.shouldCorrectlyThrowExceptionForNotSupportedAges()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyMorbidlyObeseBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyNormalWeightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyObeseBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyOverweightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateAndClassifyUnderweightBmi()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForAdultsOver65()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForAdultsUnder65()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyCalculateIdealWeightForChildren()	0	1	1	1
tests.IdealWeightTests.shouldCorrectlyThrowExceptionForNotSupportedAges()	0	1	1	1
<b>Total</b>	<b>33</b>	<b>33</b>	<b>17</b>	<b>43</b>
<b>Average</b>	<b>2.06</b>	<b>2.06</b>	<b>1.06</b>	<b>2.69</b>

Métricas de complexidade para o código inteiro, após a refatoração

Considerando somente o pacote “entity” no qual está contida a classe “QuadraticEquations”, as médias aumentaram consideravelmente. Antes da refatoração, 18.67 para a complexidade cognitiva (CogC) e 10.30 para a complexidade ciclomática (v(G)).

Após a refatoração, porém, eses números diminuíram para 6.60 para a complexidade cognitiva (CogC) e 6.40 para a complexidade ciclomática (v(G)).

Method metrics	CogC	ev(G)	iv(G)	v(G)
entity.EnergyExpenditureOperations.calculateTdee(int, String, double, double)	13	2	1	9
entity.IdealWeightOperations.calculateIdealWeight(int, String, double)	6	3	2	6
entity.IdealWeightOperations.classifyWeightAccordingToBmi(double, double)	4	5	1	5
entity.IdealWeightOperations.defineFemaleIdealBmi(int)	5	6	1	6
entity.IdealWeightOperations.defineMaleIdealBmi(int)	5	6	1	6
<b>Total</b>	<b>33</b>	<b>22</b>	<b>6</b>	<b>32</b>
<b>Average</b>	<b>6.60</b>	<b>4.40</b>	<b>1.20</b>	<b>6.40</b>

Métricas de complexidade para o pacote “entity”, após da refatoração

É considerada uma boa prática manter a complexidade cognitiva abaixo de 15 e a complexidade ciclomática abaixo de 20, com o ideal sendo ambas abaixo de 10.

Considerando que, após refatorações, as médias foram 6.60 e 6.40 e que as maiores complexidades individuais foram 13 para a cognitiva e 9 para a ciclomática, o código está próximo do cenário ideal e conseguiu atingir bons resultados quanto às métricas de complexidade.

### 3.2. Execução dos testes unitários e coverage

Após a execução dos testes, todos os trinta e oito casos de teste e, consequentemente, todos os onze métodos, passaram.

Test Suite	Duration	Status
TestSuite (tests)	14 ms	Passed
IdealWeightTests	12 ms	Passed
EnergyExpenditureTests	2 ms	Passed
shouldCorrectlyCalculateTdee	1 ms	Passed
shouldCorrectlyThrowExceptionForNotSupportedAges	1 ms	Passed

Tests passed: 11 of 11 tests – 14 ms

Process finished with exit code 0

Resultado da execução dos testes unitários

Além disso, 66% das classes, 100% dos métodos e 100% das linhas do código foram testadas, o que pôde ser comprovado utilizando uma ferramenta da IDE IntelliJ do Jet Brains.



Coverage All in WeightOperations (2) ×			
Element ^	Class, %	Method, %	Line, %
▼ all	66% (4/6)	100% (14/14)	100% (134/134)
▼ entity	66% (2/3)	100% (3/3)	100% (58/58)
Ⓢ EnergyExpenditureOperations	100% (1/1)	100% (1/1)	100% (19/19)
Ⓢ IdealWeightOperations	100% (1/1)	100% (2/2)	100% (39/39)
Ⓢ WeightOperations	0% (0/1)	100% (0/0)	100% (0/0)
▼ tests	66% (2/3)	100% (11/11)	100% (76/76)
Ⓢ EnergyExpenditureTests	100% (1/1)	100% (2/2)	100% (19/19)
Ⓢ IdealWeightTests	100% (1/1)	100% (9/9)	100% (57/57)
Ⓢ TestsSuite	0% (0/1)	100% (0/0)	100% (0/0)

Porcentagem de classes, métodos e linhas testadas em todo o projeto

Os bons resultados foram consequência dos testes foram feitos de maneira cautelosa, utilizando valores adequados para a comparação com os retornos das funções, e as classes testadas cumpriram o objetivo de criar códigos de qualidade que produzem as soluções esperadas. Além disso, o código foi feito tendo em mente que deveria ser testável e que não estava de acordo foi refatorado.

Porém, o ideal, que é sempre o mais próximo possível do 100%, não foi alcançado por conta da impossibilidade de testar a classe “TestSuite”. Entretanto, ela foi utilizada para a execução dos outros testes e está comprovadamente funcionando, então não há uma grande necessidade para a criação de testes especificamente para ela.

É importante ressaltar que, mesmo que todas a maioria dos métodos das classes tenham sido testados e todos os testes tenham passado, não é possível garantir a ausência de *bugs*. Algum caso específico pode não ter sido explorado, além de que a classe “Main” não foi testada.

Em adição, é possível que mudanças futuras no código criem *bugs* ou façam com que os testes deixem de passar. Nesse caso, o *bug* deve ser corrigido ou o teste ajustado para funcionar com a nova versão do código.

Por conta dessas incertezas, é importante criar uma base grande de testes, para identificar erros em todos os trechos do código o mais rápido possível, executar os testes com frequência e dar manutenção no código e nos testes.

## 4. REFERÊNCIAS

DevMedia. **E aí? Como você testa seus códigos?** Disponível em: <https://www.devmedia.com.br/e-ai-como-voce-testa-seus-codigos/39478>. Acesso em: 2 abr. 2024.

Code Climate. **Cognitive Complexity**. Disponível em: <https://docs.codeclimate.com/docs/cognitive-complexity#:~:text=Cognitive%20Complexity%20is%20a%20measure,be%20to%20read%20and%20understand>. Acesso em: 16 abr. 2024.

jUnit 5. Disponível em: <https://junit.org/junit5/>. Acesso em: 2 abr. 2024.

Testing Company. **Testes Automatizados e Unitários: Entenda as suas características e diferenças**. Disponível em: <https://testingcompany.com.br/blog/testes-automatizados-e-unitarios-entenda-as-suas-caracteristicas-e-diferencas#:~:text=Comumente%2C%20testes%20unit%C3%A1rios%20s%C3%A3o%20desenvolvidos,nas%20fases%20iniciais%20do%20projeto>. Acesso em: 2 abr. 2024.

Trimble. **Managing Code Complexity**. Disponível em: <https://devguide.trimble.com/development-practices/managing-code-complexity/#:~:text=Any%20function%20with%20a%20cyclomatic,depending%20on%20the%20application%20domain>. Acesso em: 16 abr. 2024.