

GESTÃO E QUALIDADE DE SOFTWARE

MÉTRICAS DE COMPLEXIDADE EM JAVA EQUAÇÃO QUADRÁTICA

Aluna: Lara Luísa Ayrolla Abreu

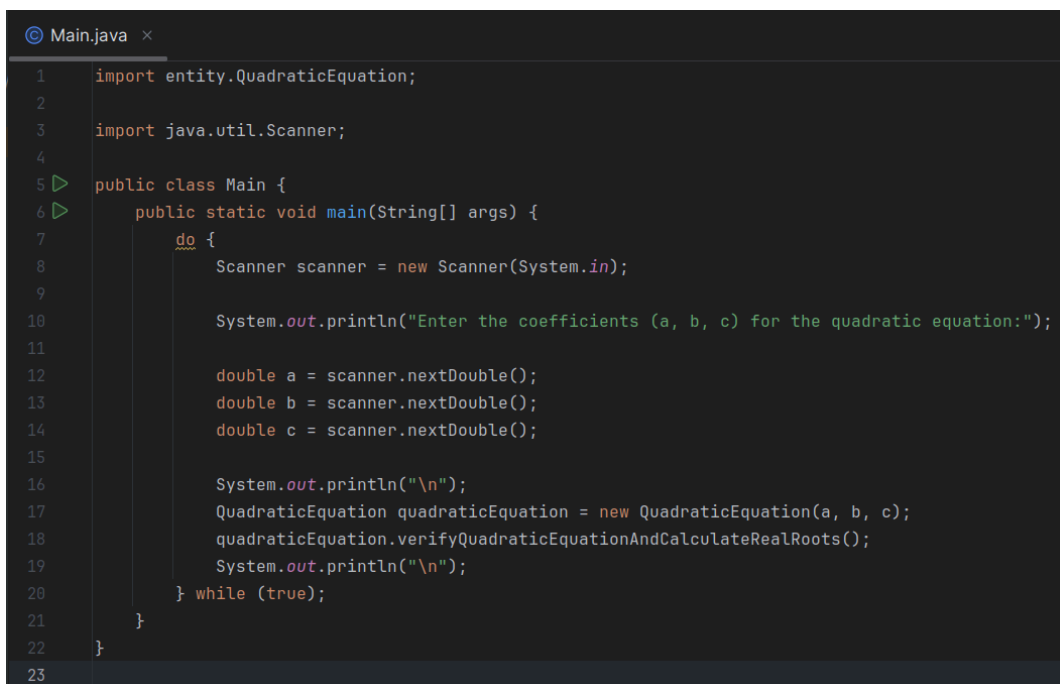
1. FUNCIONALIDADES DO CÓDIGO: OPERAÇÕES COM EQUAÇÕES QUADRÁTICAS

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](#).

Na pasta de “src”, foi criada a classe “Main”, responsável pela execução principal do programa, e o pacote “entity”, contendo a classe “QuadraticEquation”, que faz verificações e cálculos de equação quadrática.

a. Classe “Main”

Ao começo da classe estão as importações necessárias para o funcionamento do código. Há somente um método, que é responsável por interagir com o usuário e coletar a entrada de dados do usuário e chamar a classe “QuadraticEquation” e seus métodos.



```
1  import entity.QuadraticEquation;
2
3  import java.util.Scanner;
4
5  public class Main {
6      public static void main(String[] args) {
7          do {
8              Scanner scanner = new Scanner(System.in);
9
10             System.out.println("Enter the coefficients (a, b, c) for the quadratic equation:");
11
12             double a = scanner.nextDouble();
13             double b = scanner.nextDouble();
14             double c = scanner.nextDouble();
15
16             System.out.println("\n");
17             QuadraticEquation quadraticEquation = new QuadraticEquation(a, b, c);
18             quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
19             System.out.println("\n");
20         } while (true);
21     }
22 }
23
```

Declaração da classe “Main” e seu único método

b. Classe “QuadraticEquation”

Na classe “QuadraticEquation” estão contidas constantes, ou variáveis do tipo final, com diferentes tipos de mensagens e variáveis para guardar os coeficientes (a, b e c), o discriminante, as raízes reais, a validação de equação quadrática e uma mensagem. Há um construtor para preencher os valores dessas variáveis através de parâmetros e de padrões predefinidos.

Quanto aos outros métodos e funções, eles têm as funcionalidades de verificar se os coeficientes fazem parte de uma equação quadrática, calcular o valor do discriminante, determinar quantas raízes reais a equação possui, calcular seus valores, inclusive para os casos em que os coeficientes façam parte de uma equação linear, e imprimir mensagens para informar o usuário do resultado dessas operações.

```
QuadraticEquation.java x
6  public class QuadraticEquation {
    2 usages
7      public final String MESSAGE_INCORRECT_COEFFICIENTS = "Incorrectly informed coefficients";
    2 usages
8      public final String MESSAGE_EQUALITY_CONFIRMED = "Equality confirmed: 0 = 0";
    2 usages
9      public final String MESSAGE_LINEAR_EQUATION = "This is a linear equation";
    2 usages
10     public final String MESSAGE_NO_REAL_ROOTS = "This equation has no real roots";
    2 usages
11     public final String MESSAGE_TWO_EQUAL_REAL_ROOTS = "This equation has two equal real roots";
    2 usages
12     public final String MESSAGE_TWO_DIFFERENT_REAL_ROOTS = "This equation has two different real roots";
13
    5 usages
14     public double coefficientA;
    6 usages
15     public double coefficientB;
    4 usages
16     public double coefficientC;
    6 usages
17     public double discriminant;
    7 usages
18     public double realRootOne;
    6 usages
19     public double realRootTwo;
    2 usages
20     public boolean isQuadraticEquation;
    18 usages
21     public String message;
```

Declaração da classe “QuadraticEquations”, constantes e variáveis

```

7 usages
23 public QuadraticEquation(double a, double b, double c)
24 {
25     this.coefficientA = a;
26     this.coefficientB = b;
27     this.coefficientC = c;
28     this.discriminant = pow(this.coefficientB,2) - (4*this.coefficientA*this.coefficientC);
29     this.message = "";
30 }

```

Construtor da classe “QuadraticEquations”

```

7 usages
32 public void verifyQuadraticEquationAndCalculateRealRoots() {
33     this.isQuadraticEquation = this.isQuadraticEquation();
34
35     if (!this.isQuadraticEquation) {
36         return;
37     }
38
39     if (this.discriminant < 0) {
40         this.message = this.MESSAGE_NO_REAL_ROOTS;
41         System.out.println(this.message);
42     } else {
43         if (this.discriminant == 0) {
44             this.message = this.MESSAGE_TWO_EQUAL_REAL_ROOTS;
45             System.out.println(this.message);
46         } else if (this.discriminant > 0) {
47             this.message = this.MESSAGE_TWO_DIFFERENT_REAL_ROOTS;
48             System.out.println(this.message);
49         }
50         this.calculateQuadraticEquationRealRoots();
51         System.out.println(STR."Real roots: \{this.realRootOne} and \{this.realRootTwo}");
52     }
53 }

```

Método público responsável por organizar as verificações e cálculos

```

1 usage
55     private boolean isQuadraticEquation() {
56         if (this.coefficientA == 0) {
57             if (this.coefficientB == 0) {
58                 if (this.coefficientC != 0) {
59                     this.message = this.MESSAGE_INCORRECT_COEFFICIENTS;
60                 } else {
61                     this.message = this.MESSAGE_EQUALITY_CONFIRMED;
62                 }
63                 System.out.println(this.message);
64             } else {
65                 this.message = this.MESSAGE_LINEAR_EQUATION;
66                 System.out.println(this.message);
67                 this.calculateLinearEquationRealRoot();
68                 System.out.println(STR."Real root: \{this.realRootOne}");
69             }
70             return false;
71         } else {
72             return true;
73         }
74     }

```

Função que verifica se os coeficientes formam uma equação quadrática ou não

```

1 usage
76     private void calculateQuadraticEquationRealRoots ()
77     {
78         this.realRootOne = (this.coefficientB*-1 + sqrt(this.discriminant)) / (2*this.coefficientA);
79         this.realRootTwo = (this.coefficientB*-1 - sqrt(this.discriminant)) / (2*this.coefficientA);
80     }

```

Método que calcula as raízes reais da equação quadrática

```

1 usage
82     private void calculateLinearEquationRealRoot ()
83     {
84         this.realRootOne = this.realRootTwo = (this.coefficientC*-1) / this.coefficientB;
85     }

```

Método que calcula a raiz real da equação linear

2. TESTES DESENVOLVIDOS

2.1. Testes Manuais

Durante o desenvolvimento, foram realizados diversos testes manuais, executando o programa e inserindo dados como um usuário faria. Isso foi feito com o objetivo de adaptar o código no caso de alguma resposta inesperada, até que resultados satisfatórios foram obtidos.

Abaixo, seguem imagens dos últimos testes manuais feitos, para os quais todas as saídas de dados estão corretas de acordo com o esperado.

```
Enter the coefficients (a, b, c) for the quadratic equation:  
0  
0  
0  
  
Equality confirmed:  $0 = 0$ 
```

```
Enter the coefficients (a, b, c) for the quadratic equation:  
0  
0  
8  
  
Incorrectly informed coefficients
```

```
Enter the coefficients (a, b, c) for the quadratic equation:  
0  
2  
-8  
  
This is a linear equation  
Real root: 4.0
```

```
Enter the coefficients (a, b, c) for the quadratic equation:  
10  
-4  
12  
  
This equation has no real roots
```

```
Enter the coefficients (a, b, c) for the quadratic equation:  
4  
-4  
1
```

```
This equation has two equal real roots  
Real roots: 0.5 and 0.5
```

```
Enter the coefficients (a, b, c) for the quadratic equation:  
1  
6  
7
```

```
This equation has two different real roots  
Real roots: -1.5857864376269049 and -4.414213562373095
```

2.2. Testes automatizados

No pacote “tests”, foi criada a classe “EquationTests”, na qual estão contidos os métodos utilizados para testar as funções da classe “QuadraticEquation”.

Ao começo da classe, são feitas as importações da classe que será testada, da biblioteca que será utilizada para o desenvolvimento dos testes (“JUnit”) e é declarada uma variável de nome “quadraticEquation” do tipo “QuadraticEquation”, que será utilizada para chamar as funções e acessar as variáveis e constantes da classe testada.

Foram desenvolvidos seis métodos, com doze casos de teste, que verificam se o funcionamento das funções da classe “QuadraticEquation” 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.

```

© EquationTests.java x
1 package tests;
2
3 import entity.QuadraticEquation;
4 import org.junit.Assert;
5 import org.junit.Test;
6
7 public class EquationTests {
8     QuadraticEquation quadraticEquation;
9

```

Declaração da classe “EquationTests”, importações e declaração de variável para teste

```

10 @Test
11 public void shouldCorrectlyConfirmZeroEquality ()
12 {
13     quadraticEquation = new QuadraticEquation(a: 0, b: 0, c: 0);
14     quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
15
16     Assert.assertEquals(
17         quadraticEquation.MESSAGE_EQUALITY_CONFIRMED,
18         quadraticEquation.message
19     );
20 }

```

Método que testa o comportamento com todos os coeficientes igual a 0

```

22 @Test
23 public void shouldCorrectlyIdentifyIncorrectCoefficients ()
24 {
25     quadraticEquation = new QuadraticEquation(a: 0, b: 0, c: 8);
26     quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
27
28     Assert.assertEquals(
29         quadraticEquation.MESSAGE_INCORRECT_COEFFICIENTS,
30         quadraticEquation.message
31     );
32 }

```

Método que testa o comportamento com coeficientes incorretos

```

44     @Test
45     public void shouldCorrectlyCalculateSellerOvertime ()
46     {
47         Assert.assertEquals( expected: 2000, sellerOne.calculateOvertime( numberOfHours: 100), delta: 0);
48         Assert.assertEquals( expected: 400, sellerTwo.calculateOvertime( numberOfHours: 20), delta: 0);
49         Assert.assertEquals( expected: 200, sellerThree.calculateOvertime( numberOfHours: 10), delta: 0);
50
51         Assert.assertEquals( expected: 2000, sellerOne.calculateOvertime( numberOfHours: 100), delta: 0);
52         Assert.assertEquals( expected: 400, sellerTwo.calculateOvertime( numberOfHours: 20), delta: 0);
53         Assert.assertEquals( expected: 200, sellerThree.calculateOvertime( numberOfHours: 10), delta: 0);
54
55         Assert.assertEquals( expected: 4000, sellerOne.overtime, delta: 0);
56         Assert.assertEquals( expected: 800, sellerTwo.overtime, delta: 0);
57         Assert.assertEquals( expected: 400, sellerThree.overtime, delta: 0);
58     }

```

Método que testa o cálculo do valor das horas extras de vendedores

```

34     @Test
35     public void shouldCorrectlyIdentifyAndCalculateLinearEquation ()
36     {
37         quadraticEquation = new QuadraticEquation( a: 0, b: 2, c: -8);
38         quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
39
40         Assert.assertEquals(
41             quadraticEquation.MESSAGE_LINEAR_EQUATION,
42             quadraticEquation.message
43         );
44
45         Assert.assertEquals( expected: 4, quadraticEquation.realRootOne, delta: 0);
46         Assert.assertEquals( expected: 4, quadraticEquation.realRootTwo, delta: 0);
47     }

```

Método que testa a verificação e o cálculo da equação linear

```

49     @Test
50     public void shouldCorrectlyIdentifyQuadraticEquationWithNoRealRoots ()
51     {
52         quadraticEquation = new QuadraticEquation( a: 10, b: -4, c: 12);
53         quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
54
55         Assert.assertEquals(
56             quadraticEquation.MESSAGE_NO_REAL_ROOTS,
57             quadraticEquation.message
58         );
59     }

```

Método que testa a verificação de equação quadrática sem raízes reais


```

61     @Test
62     public void shouldCorrectlyIdentifyAndCalculateEquationWithEqualRealRoots ()
63     {
64         quadraticEquation = new QuadraticEquation( a: 4, b: -4, c: 1);
65         quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
66
67         Assert.assertEquals(
68             quadraticEquation.MESSAGE_TWO_EQUAL_REAL_ROOTS,
69             quadraticEquation.message
70         );
71
72         Assert.assertEquals( expected: 0.5, quadraticEquation.realRootOne, delta: 0);
73         Assert.assertEquals( expected: 0.5, quadraticEquation.realRootTwo, delta: 0);
74     }

```

Método que testa a verificação e cálculo de equação quadrática com duas raízes reais iguais

```

76     @Test
77     public void shouldCorrectlyIdentifyAndCalculateEquationWithDifferentRealRoots ()
78     {
79         quadraticEquation = new QuadraticEquation( a: 1, b: 6, c: 7);
80         quadraticEquation.verifyQuadraticEquationAndCalculateRealRoots();
81
82         Assert.assertEquals(
83             quadraticEquation.MESSAGE_TWO_DIFFERENT_REAL_ROOTS,
84             quadraticEquation.message
85         );
86
87         Assert.assertEquals( expected: -1.58, quadraticEquation.realRootOne, delta: 0.1);
88         Assert.assertEquals( expected: -4.4, quadraticEquation.realRootTwo, delta: 0.1);
89     }
90 }

```

Método que testa a verificação e cálculo de equação quadrática com duas raízes reais diferentes

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, a média da complexidade cognitiva (CogC) foi 1.33 e a média da complexidade ciclomática (v(G)) foi 1.67. É importante ressaltar que as classes de teste, por serem menos complexas ajudam a diminuir a complexidade do projeto em geral.

Metrics Complexity metrics for Project 'QuadraticEquation' fro... x					
Method metrics Class metrics Package metrics Module metrics Project metrics					
method ^		CogC	ev(G)	iv(G)	v(G)
entity.QuadraticEquation.calculateLinearEquationRealRoot()		0	1	1	1
entity.QuadraticEquation.calculateQuadraticEquationRealRoots()		0	1	1	1
entity.QuadraticEquation.isQuadraticEquation()		9	2	3	4
entity.QuadraticEquation.QuadraticEquation(double, double, double)		0	1	1	1
entity.QuadraticEquation.verifyQuadraticEquationAndCalculateRealRoots()		6	2	4	5
Main.main(String[])		1	1	2	2
tests.EquationTests.shouldCorrectlyConfirmZeroEquality()		0	1	1	1
tests.EquationTests.shouldCorrectlyIdentifyAndCalculateEquationWithDifferentRe		0	1	1	1
tests.EquationTests.shouldCorrectlyIdentifyAndCalculateEquationWithEqualRealR		0	1	1	1
tests.EquationTests.shouldCorrectlyIdentifyAndCalculateLinearEquation()		0	1	1	1
tests.EquationTests.shouldCorrectlyIdentifyIncorrectCoefficients()		0	1	1	1
tests.EquationTests.shouldCorrectlyIdentifyQuadraticEquationWithNoRealRoots()		0	1	1	1
Total		16	14	18	20
Average		1.33	1.17	1.50	1.67

Métricas de complexidade para o código inteiro

Considerando somente o pacote “entity” no qual está contida a classe “QuadraticEquations”, as médias aumentaram consideravelmente. 3.00 para a complexidade cognitiva (CogC) e 2.40 para a complexidade ciclomática (v(G)).

Metrics Complexity metrics for Project 'QuadraticEquation' fro... x Complexity metrics for Directory '.../src/entity' from Tu... x					
Method metrics Class metrics Package metrics Module metrics Project metrics					
method ^		CogC	ev(G)	iv(G)	v(G)
entity.QuadraticEquation.calculateLinearEquationRealRoot()		0	1	1	1
entity.QuadraticEquation.calculateQuadraticEquationRealRoots()		0	1	1	1
entity.QuadraticEquation.isQuadraticEquation()		9	2	3	4
entity.QuadraticEquation.QuadraticEquation(double, double, double)		0	1	1	1
entity.QuadraticEquation.verifyQuadraticEquationAndCalculateRealRoots()		6	2	4	5
Total		15	7	10	12
Average		3.00	1.40	2.00	2.40

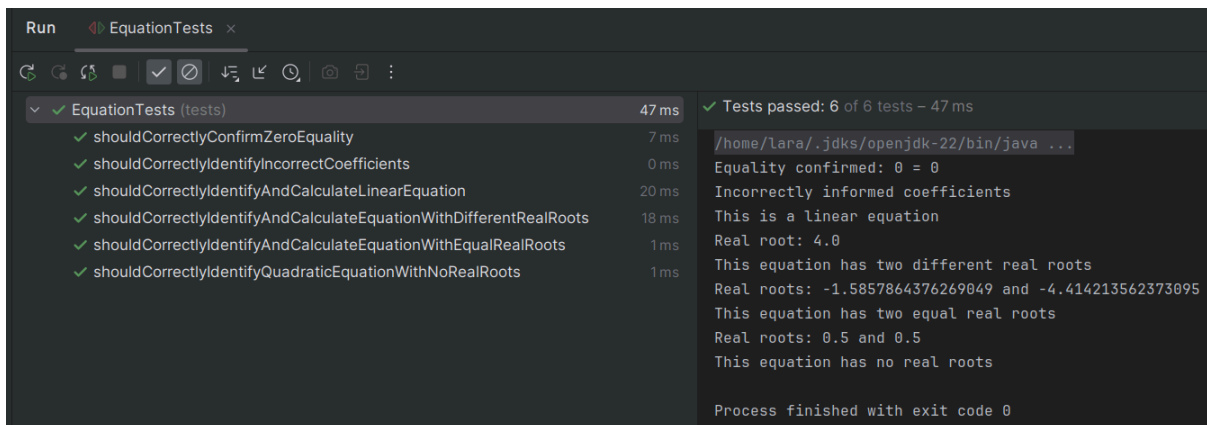
Métricas de complexidade para o pacote “entity”

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 as médias foram 3 e 2.4 e que as maiores complexidades individuais foram 9 para a cognitiva e 4 para a ciclomática, o código está no cenário ideal e conseguiu atingir bons resultados quanto às metrcas de complexidade.

3.2. Execução dos testes unitários e coverage

Após a execução dos testes da classe “QuadraticEquationTests”, todos os doze casos de teste e, conseqüentemente, todos os seis métodos, passaram.

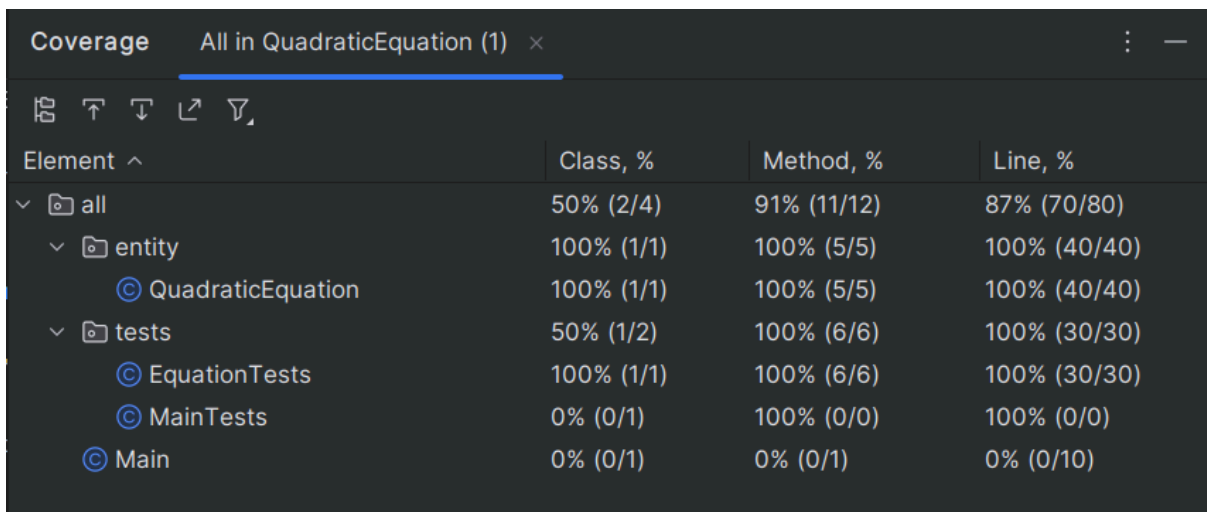


```
Run EquationTests x
✓ EquationTests (tests) 47 ms
  ✓ shouldCorrectlyConfirmZeroEquality 7 ms
  ✓ shouldCorrectlyIdentifyIncorrectCoefficients 0 ms
  ✓ shouldCorrectlyIdentifyAndCalculateLinearEquation 20 ms
  ✓ shouldCorrectlyIdentifyAndCalculateEquationWithDifferentRealRoots 18 ms
  ✓ shouldCorrectlyIdentifyAndCalculateEquationWithEqualRealRoots 1 ms
  ✓ shouldCorrectlyIdentifyQuadraticEquationWithNoRealRoots 1 ms

✓ Tests passed: 6 of 6 tests - 47 ms
/home/lara/.jdk/openjdk-22/bin/java ...
Equality confirmed: 0 = 0
Incorrectly informed coefficients
This is a linear equation
Real root: 4.0
This equation has two different real roots
Real roots: -1.5857864376269049 and -4.414213562373095
This equation has two equal real roots
Real roots: 0.5 and 0.5
This equation has no real roots
Process finished with exit code 0
```

Resultado da execução dos testes unitários

Além disso, 50% das classes, 91% dos métodos e 87% das linhas do código foram testadas, o que pôde ser comprovado utilizando uma ferramenta da IDE IntelliJ do Jet Brains.



Element ^	Class, %	Method, %	Line, %
all	50% (2/4)	91% (11/12)	87% (70/80)
entity	100% (1/1)	100% (5/5)	100% (40/40)
QuadraticEquation	100% (1/1)	100% (5/5)	100% (40/40)
tests	50% (1/2)	100% (6/6)	100% (30/30)
EquationTests	100% (1/1)	100% (6/6)	100% (30/30)
MainTests	0% (0/1)	100% (0/0)	100% (0/0)
Main	0% (0/1)	0% (0/1)	0% (0/10)

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 “Main”. Isso poderia ser resolvido com ainda mais refatorações e possivelmente uma nova classe de testes com o objetivo de testar somente o método “main”.

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.