

# Report

Where you can find Code coverage Report?

The code coverage report can be found in “**Calculator/htmlReport**” in the Calculator Project root directory.

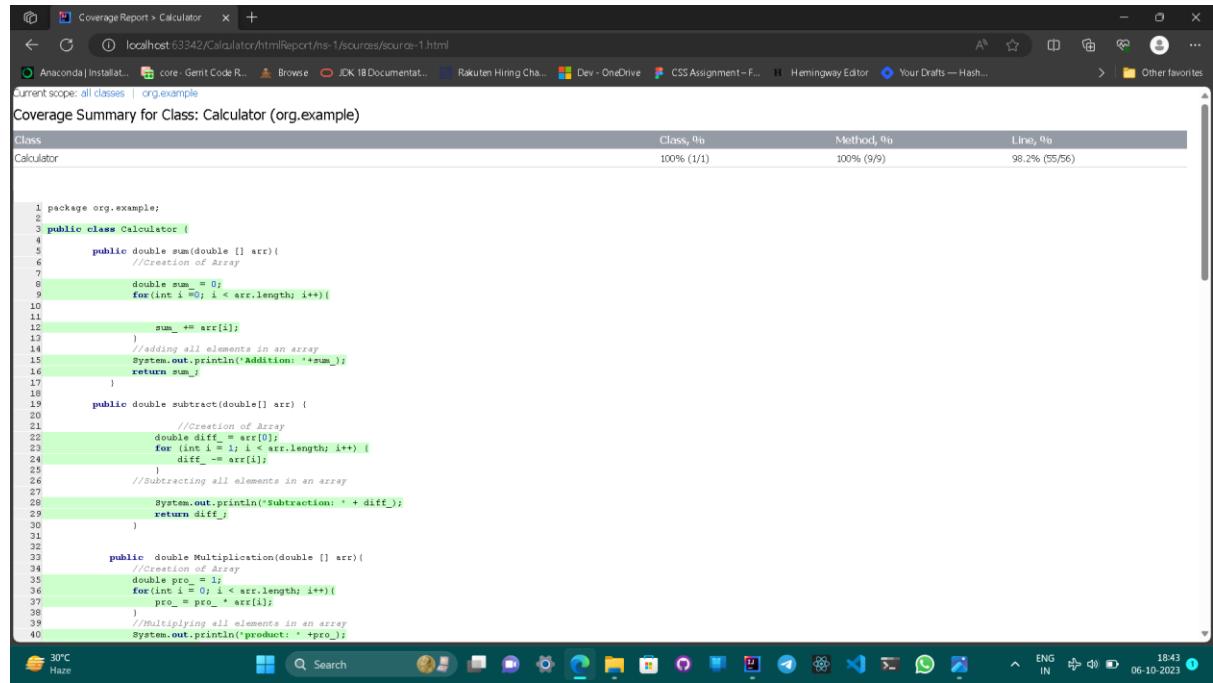
Where You can Find Solutions to the unit test code with test cases passing or not?

You can find the file in the calculator test report of root directory.

The Picture of code coverage report as below.

Class Code Coverage Report with most of the common cases testing all corner cases and all functionalities written in the Calculator.java class in src directory.

Covered addition and subtraction.



Coverage Report > Calculator

localhost:63342/Calculator/htmlReport/hs-1/scurces/source-1.html

Anaconda | Installat... core: Gerrit Code R... Browse JDK 18 Documentat... Rakuten Hiring Cha... Dev - OneDrive CSS Assignment - F... Hemingway Editor Your Drafts — Hash...

Current scope: all classes | org.example

Coverage Summary for Class: Calculator (org.example)

Class	Class, %	Method, %	Line, %
Calculator	100% (1/1)	100% (9/9)	98.2% (55/56)

```
1 package org.example;
2
3 public class Calculator {
4
5     public double sum(double [] arr) {
6         //Creation of Array
7
8         double sum_ = 0;
9         for(int i = 0; i < arr.length; i++){
10
11             sum_ += arr[i];
12
13         }
14         //adding all elements in an array
15         System.out.println("Addition: "+sum_);
16         return sum_;
17     }
18
19     public double subtract(double[] arr) {
20
21         //Creation of Array
22         double diff_ = arr[0];
23         for (int i = 1; i < arr.length; i++) {
24             diff_ -= arr[i];
25         }
26         //Subtracting all elements in an array
27
28         System.out.println("Subtraction: " + diff_);
29         return diff_;
30     }
31
32
33     public double Multiplication(double [] arr) {
34
35         //Creation of Array
36         double pro_ = 1;
37         for (int i = 0; i < arr.length; i++) {
38             pro_ = pro_* arr[i];
39         }
40         //Multiplying all elements in an array
41         System.out.println("product: " +pro_);
42     }
}
```

30°C Haze

Search

ENGLISH IN 18:43 06-10-2023

## Covered Multiplication cases.

A screenshot of a Microsoft Edge browser window displaying Java code for multiplication operations. The code includes methods for subtraction, multiplication, division, and square root calculation. The browser interface shows tabs, a search bar, and a taskbar at the bottom.

```
16     return sum_j;
17   }
18 
19   public double subtract(double[] arr) {
20 
21     //Creation of Array
22     double diff_ = arr[0];
23     for (int i = 1; i < arr.length; i++) {
24       diff_ -= arr[i];
25     }
26     //Subtracting all elements in an array
27     System.out.println("Subtraction: " + diff_);
28     return diff_;
29   }
30 
31 
32 
33   public double multiplication(double [] arr){
34     //Creation of Array
35     double pro_ = 1;
36     for (int i = 0; i < arr.length; i++){
37       pro_ = pro_ * arr[i];
38     }
39     //Multiplying all elements in an array
40     System.out.println("product: " + pro_);
41     return pro_;
42   }
43 
44   public double divide(double a, double b) {
45     if (b == 0) {
46       // If denominator is zero return some minimum value
47       System.out.println("Cannot divide by zero");
48       return Double.MIN_VALUE;
49     }
50     double div_=a/b;
51     System.out.println("division of Given Two Numbers is: "+div_);
52     return div_;
53   }
54 
55 
56   public double squareRoot(double radical){
57     if(radical < 0){
58       System.out.println("Square root of negative number cannot be real number");
59       return Double.MIN_VALUE;
60     }
61     double res = Math.sqrt(radical);
62     System.out.println("Square root: " + res);
63     return res;
64   }
65 }
```

## Covered division cases.

A screenshot of a Microsoft Edge browser window displaying Java code for division and modulo operations. The code includes methods for division, square root calculation, and modulo operation. The browser interface shows tabs, a search bar, and a taskbar at the bottom.

```
41     return pro_j;
42   }
43 
44   public double divide(double a, double b){
45     if (b == 0) {
46       // If denominator is zero return some minimum value
47       System.out.println("Cannot divide by zero");
48       return Double.MIN_VALUE;
49     }
50     double div_=a/b;
51     System.out.println("division of Given Two Numbers is: "+div_);
52     return div_;
53   }
54 
55 
56   public double squareRoot(double radical){
57     if(radical < 0){
58       System.out.println("Square root of negative number cannot be real number");
59       return Double.MIN_VALUE;
60     }
61     double res = Math.sqrt(radical);
62     System.out.println("Square root: " + res);
63     return res;
64   }
65 
66   }
67 
68   public double moduloOfTwoNum(double a, double b) {
69     /*the modulo operation returns the remainder or signed remainder of a division,
70      after one number is divided by another (called the modulus of the operation).*/
71 
72     // Handling negative values
73     if (a < 0) {
74       a = -a;
75     }
76     if (b < 0) {
77       b = -b;
78     }
79     if(b == 0){
80       System.out.println("mod 0 is undefined");
81       return Double.MIN_VALUE;
82     }
83     // Finding mod by repeated subtraction
84     double mod = a;
85     while (mod >= b) {
86       mod = mod - b;
87     }
88     // Sign of result typically depends
89   }
90 }
```

Covered all of modulo by subtraction method except the if( $a < 0$ ) { return -mod;} as the method converts the input a if it is less than 0 to a positive so it can't touch this case if it has to be covered than store the value in another value and return the modulo by comparing it.

The screenshot shows a browser window with the URL [localhost:63342/Calculator/htmlReport/h-/sources/source-1.html](http://localhost:63342/Calculator/htmlReport/h-/sources/source-1.html). The page displays a Java code editor with syntax highlighting for Java code. The code includes methods for calculating square roots and performing modulus operations. A vertical bar on the right indicates code coverage, with green representing covered code and red representing uncovered code. The browser interface includes tabs for other open pages like 'Rakuten Hiring Cha...', 'Dev - OneDrive', and 'CSS Assignment - F...'. The bottom of the screen shows the Windows taskbar with various pinned icons.

```
52     }
53     double res = Math.sqrt(radical);
54     System.out.println("Square root: " + res);
55     return res;
56   }
57
58   public double moduloOfTwoNum(double a, double b) {
59     /*the modulo operation returns the remainder or signed remainder of a division,
60      after one number is divided by another (called the modulus of the operation).*/
61
62     // Handling negative values
63     if (a < 0) {
64       a = -a;
65     }
66     if (b < 0) {
67       b = -b;
68     }
69     if(b == 0){
70       System.out.println("mod 0 is undefined");
71       return Double.MIN_VALUE;
72     }
73
74     // Finding mod by repeated subtraction
75     double mod = a;
76     while (mod >= b) {
77       mod = mod - b;
78     }
79
80     // Sign of result typically depends
81     // on sign of a
82     if (a < 0) {
83       return -mod;
84     }
85
86     System.out.println("Modulo of 2 numbers: " + mod);
87
88     return mod;
89   }
90
91   public double Average(double [] arr){
92
93     if (arr.length == 0) {
94       throw new IllegalArgumentException("Input array cannot be empty");
95     }
96
97     double sum_ = 0;
98     for(int i =0; i < arr.length; i++){
99       sum_ += arr[i];
100    }
101
102    return sum_ / arr.length;
103  }
104
105  public static void main(String[] args) {
106
107   }
108
109 }
```

Covered all cases of Averages

```
Coverage Report > Calculator x +  
localhost:63342/Ccalculator/htmlReport/hs-1/scuras/source-1.html  
Anaconda|Installat... core - Gerrit Code R... Browse JOK 1B Documentat... Rakuten Hiring Cha... Dev - OneDrive CSS Assignment-F... Hemingway Editor Your Drafts — Hash... Other favorites  
89  
90  
91 // Sign of result typically depends  
92 // on sign of a  
93 if (n < 0) {  
94 return -mod;  
95 }  
96  
97 System.out.println("Modulo of 2 numbers: " + mod);  
98  
99 return mod;  
100 }  
101  
102  
103 public double Average(double [] arr){  
104  
105 if (arr.length == 0) {  
106 throw new IllegalArgumentException("Input array cannot be empty");  
107 }  
108  
109 double sum_ = 0;  
110  
111 for(int i = 0; i < arr.length; i++){  
112 sum_ += arr[i];  
113 }  
114 //recalculating average of all elements in an array  
115 double average = sum_ / (arr.length);  
116 System.out.println("Average of numbers: " +average);  
117 return average;  
118 }  
119  
120 public int factorial(int n) {  
121 if (n < 0) {  
122 System.out.println("If the given integer is negative,n! is not defined(n)");  
123 return 0;  
124 } else if (n == 0 || n == 1) {  
125 return 1;  
126 } else {  
127 return n * factorial(n - 1);  
128 }  
129 }  
130  
131  
132  
133 }  
134  
generated on 2023-10-06 18:30  
30°C Haze Search ENG IN 06-10-2023 18:43
```

The Test cases are written using JUnit-4 version to test the code.

## Basic unit test for addition

```
1 package org.example;
2
3 import org.junit.Assert;
4 import org.junit.Before;
5 import org.junit.Test;
6
7 public class CalculatorTest {
8
9     Calculator calculator;
10
11     @Before
12     public void setUp() {
13         calculator = new Calculator();
14     }
15
16     // Addition Unit Testing Coverage
17     /**
18      * test sum of 20 + 30 + 30 = 60
19      * and 100 + 20 + 30 = 150
20      */
21     @Test
22     public void testSumOfTenPlusTwentyPlusThirtyBecomesSixtyAndHundredPlusTwentyPlusThirty() {
23         double result = calculator.sum(new double[]{10,20,30});
24         Assert.assertEquals(60,result,0);
25
26         result = calculator.sum(new double[]{100,20,30});
27         Assert.assertEquals(150,result,0);
28     }
29
30     /**
31      * To test sum of small numbers to be positive
32      */
33     @Test
34     public void testSumOfTwoPlusThreePlusSixEqualsTwelve() {
35         double res = calculator.sum(new double[]{2,4,6});
36         Assert.assertEquals(12,res,0);
37     }
38
39     /**
40      * To test sum of negative and higher positive value result in positive val
41      */
42     @Test
43     public void testSumOfNegativeFivePlusNegativeSixPlusTwelveToBeOne() {
44         double res = calculator.sum(new double[]{-5,-6,12});
45         Assert.assertEquals(1,res,0);
46     }
47
48     // Subtraction Unit Testing Coverage
49 }
```

## Basic unit test for subtraction, multiplication.

```
45
46     // Subtraction Unit Testing Coverage
47     /**
48      * the first digit has to be positive and higher to subtract from others
49      */
50     @Test
51     public void subtract() {
52         double result = calculator.subtract(new double[]{10,20,5,1,6});
53         Assert.assertEquals(-22,result,0);
54     }
55
56     // Multiplication Unit Testing Coverage
57     /**
58      */
59     @Test
60     public void multiplication() {
61         double result = calculator.Multiplication(new double[]{1,2,3,4,5});
62         Assert.assertEquals(120,result,0);
63     }
64
65     // Division Unit Testing Coverage
66     /**
67      * Test division of two positive numbers result in positive value.
68      * 10 / 2 = 5
69      */
70     @Test
71     public void divide() {
72         double result = calculator.divide(10, 2);
73         Assert.assertEquals(5, result, 0);
74     }
75
76     /**
77      * to test division by zero not possible
78      */
79     @Test
80     public void testDivisionByZero() {
81         Assert.assertEquals(Double.MIN_VALUE,calculator.divide(5,0),0);
82     }
83
84     // SquareRoot Unit Testing Coverage
85     /**
86      * To test square root of a 25 to be 5
87      */
88     @Test
89     public void testSquareRootOfTwentyFive() {
90         double result = calculator.squareRoot(25);
91         Assert.assertEquals(5,result,0);
92     }
93
94 }
```

```
Coverage Report > CalculatorTest.html
localhost:6342/Ccalculator/htmlReport/hs-1/scores/source-2.html

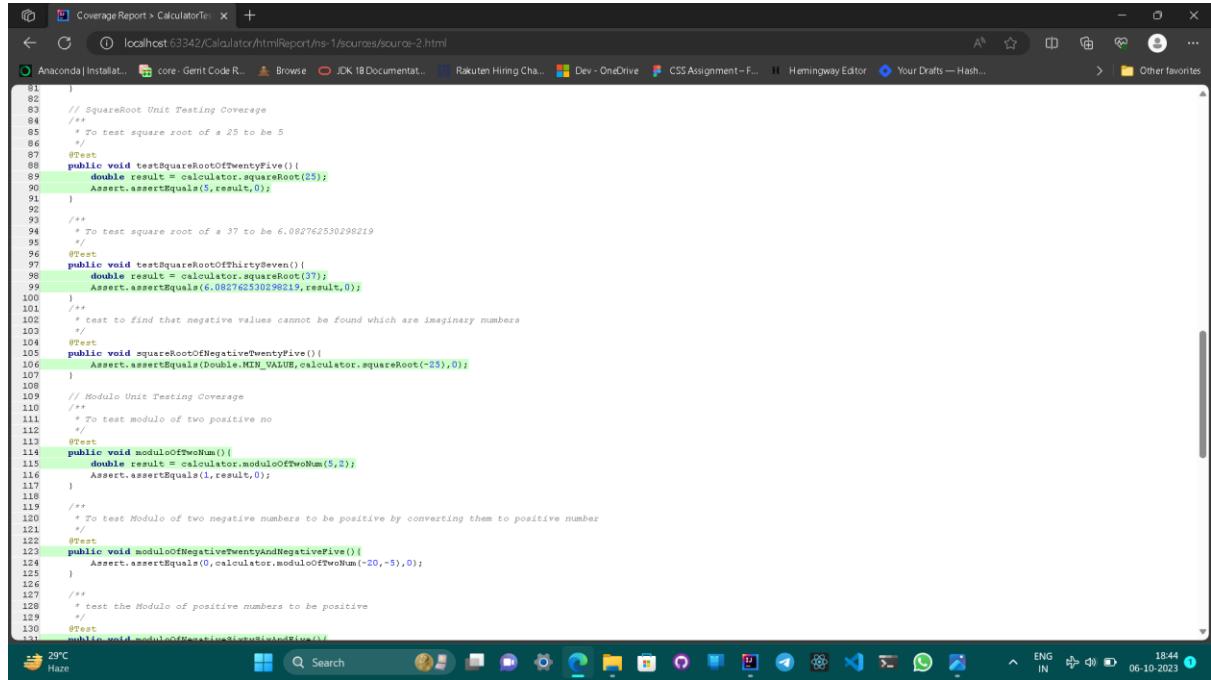
Anacoda | Install... core: Gerrit Code R... Browse JDK 18 Documentat... Rakuten Hiring Cha... Dev - OneDrive CSS Assignment-F... Hemingway Editor Your Drafts — Hash...
ENG IN 18:44 06-10-2023

52     double result = calculator.subtract(new double[]{10, 20, 3, 4, 6});
53     Assert.assertEquals(-22, result, 0);
54 }
55 /**
56 // Multiplication Unit Testing Coverage
57 */
58 /**
59 */
60 @Test
61 public void multiplication(){
62     double result = calculator.Multiplication(new double[]{1, 2, 3, 4, 5});
63     Assert.assertEquals(120, result, 0);
64 }
65 /**
66 */
67 /**
68 * Test division of two positive numbers result in positive value.
69 *      10 / 2 = 5
70 */
71 @Test
72 public void divide() {
73     double result = calculator.divide(10, 2);
74     Assert.assertEquals(5, result, 0);
75 }
76 /**
77 * To test division by zero not possible
78 */
79 @Test
80 public void TestDivisionByZero(){
81     Assert.assertEquals(Double.NaN_VALUE, calculator.divide(5,0), 0);
82 }
83 /**
84 */
85 /**
86 * To test square root of a 25 to be 5
87 */
88 @Test
89 public void testSquareRootOfTwentyFive(){
90     double result = calculator.squareRoot(25);
91     Assert.assertEquals(5,result,0);
92 }
93 /**
94 * To test square root of a 37 to be 6.082762530298219
95 */
96 @Test
97 public void testSquareRootOfThirtySeven(){
98     double result = calculator.squareRoot(37);
99     Assert.assertEquals(6.082762530298219,result,0);
100 }
101 /**
102 * To test square root of a negative number
103 */


```

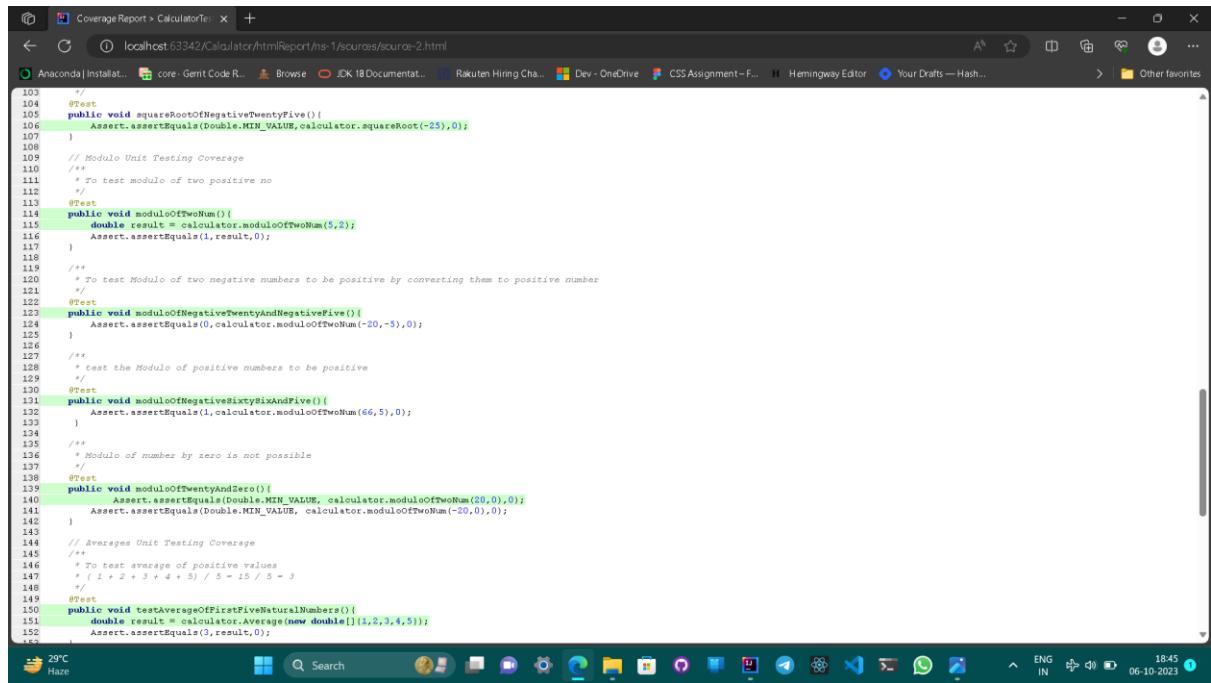
Basic unit test for division.

## Basic unit test for square root of numbers testing all edge cases.



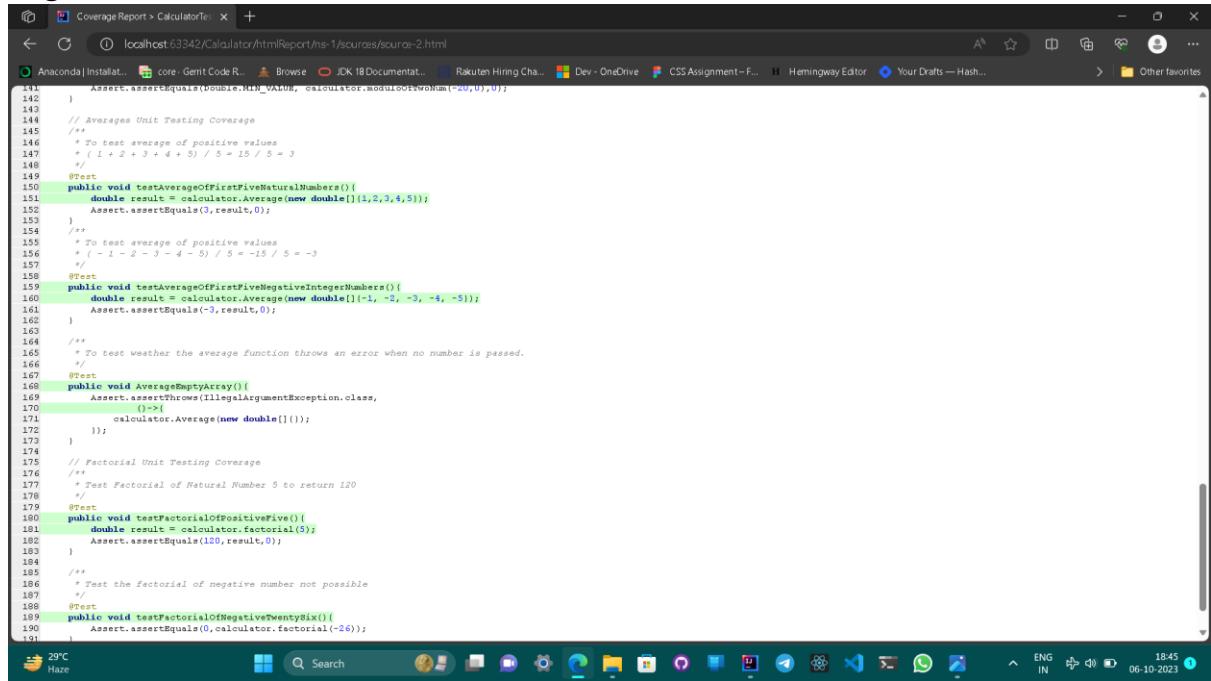
```
81     }
82     // SquareRoot Unit Testing Coverage
83     /**
84      * To test square root of a 25 to be 5
85      */
86     @Test
87     public void testSquareRootOfTwentyFive(){
88         double result = calculator.squareRoot(25);
89         Assert.assertEquals(5,result,0);
90     }
91
92     /**
93      * To test square root of a 37 to be 6.082762530298219
94      */
95     @Test
96     public void testSquareRootOfThirtySeven(){
97         double result = calculator.squareRoot(37);
98         Assert.assertEquals(6.082762530298219,result,0);
99     }
100    /**
101     * test to find that negative values cannot be found which are imaginary numbers
102     */
103    @Test
104    public void squareRootOfNegativeTwentyFive(){
105        Assert.assertEquals(Double.MIN_VALUE,calculator.squareRoot(-25),0);
106    }
107
108    // Modulo Unit Testing Coverage
109    /**
110     * To test modulo of two positive no
111     */
112    @Test
113    public void moduloOfTwoNum(){
114        double result = calculator.moduloOfTwoNum(5,2);
115        Assert.assertEquals(1,result,0);
116    }
117
118    /**
119     * To test Modulo of two negative numbers to be positive by converting them to positive number
120     */
121    @Test
122    public void moduloOfNegativeTwentyAndNegativeFive(){
123        Assert.assertEquals(0,calculator.moduloOfTwoNum(-20,-5),0);
124    }
125
126    /**
127     * test the Modulo of positive numbers to be positive
128     */
129    @Test
130    public void moduloOfPositiveNumbers(){
131        Assert.assertEquals(0,calculator.moduloOfTwoNum(20,5),0);
132    }
133
```

## Basic unit test for Modulo of numbers testing all edge cases.



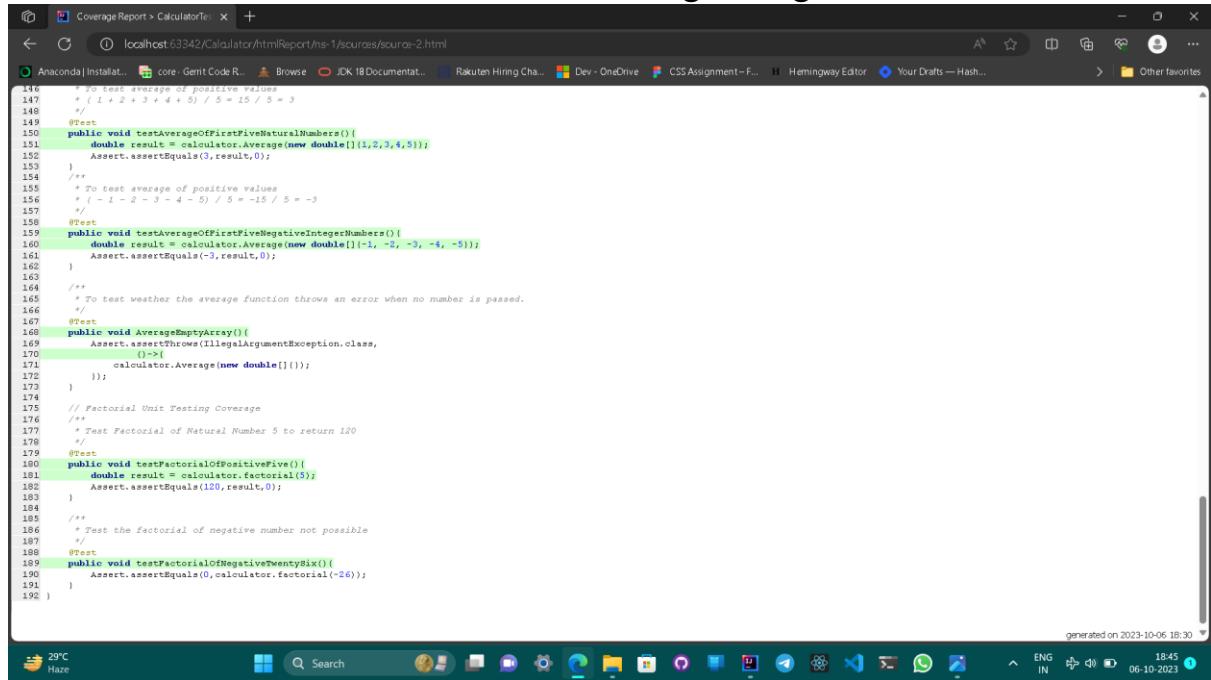
```
103    /**
104     * To test square root of NegativeTwentyFive()
105     */
106     Assert.assertEquals(Double.MIN_VALUE,calculator.squareRoot(-25),0);
107
108    // Modulo Unit Testing Coverage
109    /**
110     * To test modulo of two positive no
111     */
112    @Test
113    public void moduloOfTwoNum(){
114        double result = calculator.moduloOfTwoNum(5,2);
115        Assert.assertEquals(1,result,0);
116    }
117
118    /**
119     * To test Modulo of two negative numbers to be positive by converting them to positive number
120     */
121    @Test
122    public void moduloOfNegativeTwentyAndNegativeFive(){
123        Assert.assertEquals(0,calculator.moduloOfTwoNum(-20,-5),0);
124    }
125
126    /**
127     * test the Modulo of positive numbers to be positive
128     */
129    @Test
130    public void moduloOfPositiveNumbers(){
131        Assert.assertEquals(0,calculator.moduloOfTwoNum(20,5),0);
132    }
133
134    /**
135     * Modulo of number by zero is not possible
136     */
137    @Test
138    public void moduloOfTwentyAndZero(){
139        Assert.assertEquals(Double.MIN_VALUE, calculator.moduloOfTwoNum(20,0),0);
140        Assert.assertEquals(Double.MIN_VALUE, calculator.moduloOfTwoNum(-20,0),0);
141    }
142
143
144    // Averages Unit Testing Coverage
145    /**
146     * To test average of positive values
147     */
148    @Test
149    public void testAverageOfFirstFiveNaturalNumbers(){
150        double result = calculator.Average(new double[]{1,2,3,4,5});
151        Assert.assertEquals(3,result,0);
152    }
153
```

## Basic Unit test of average of no number, and more than two positive and negative numbers.



```
131     Assert.assertEquals(double.MIN_VALUE, calculator.moduloTwoNum(-20,0),0);
142 }
143 /**
144 * Averages Unit Testing Coverage
145 */
146 /**
147 * To test average of positive values
148 * ( 1 + 2 + 3 + 4 + 5 ) / 5 = 15 / 5 = 3
149 */
150 @Test
151 public void testAverageOfFirstFiveNaturalNumbers(){
152     double result = calculator.Average(new double[]{1,2,3,4,5});
153 }
154 /**
155 * To test average of positive values
156 * ( - 1 - 2 - 3 - 4 - 5 ) / 5 = -15 / 5 = -3
157 */
158 @Test
159 public void testAverageOfFirstFiveNegativeIntegerNumbers(){
160     double result = calculator.Average(new double[]{-1,-2,-3,-4,-5});
161 }
162 /**
163 * To test whether the average function throws an error when no number is passed.
164 */
165 /**
166 * To test whether the average function throws an error when no number is passed.
167 */
168 public void AverageEmptyArray(){
169     Assert.assertThrows(IllegalArgumentException.class,
170         ()->{
171             calculator.Average(new double[]{});
172         });
173 }
174 /**
175 * Factorial Unit Testing Coverage
176 */
177 /**
178 * Test Factorial of Natural Number 5 to return 120
179 */
180 @Test
181 public void testFactorialOfPositiveFive(){
182     double result = calculator.factorial(5);
183     Assert.assertEquals(120,result,0);
184 }
185 /**
186 * Test the factorial of negative number not possible
187 */
188 @Test
189 public void testFactorialOfNegativeTwentySix(){
190     Assert.assertEquals(0,calculator.factorial(-26));
191 }
```

## Basic unit test for Factorial of numbers testing all edge cases.



```
146 /**
147 * To test average of positive values
148 * ( 1 + 2 + 3 + 4 + 5 ) / 5 = 15 / 5 = 3
149 */
150 @Test
151 public void testAverageOfFirstFiveNaturalNumbers(){
152     double result = calculator.Average(new double[]{1,2,3,4,5});
153 }
154 /**
155 * To test average of positive values
156 * ( - 1 - 2 - 3 - 4 - 5 ) / 5 = -15 / 5 = -3
157 */
158 @Test
159 public void testAverageOfFirstFiveNegativeIntegerNumbers(){
160     double result = calculator.Average(new double[]{-1,-2,-3,-4,-5});
161 }
162 /**
163 * To test whether the average function throws an error when no number is passed.
164 */
165 /**
166 * To test whether the average function throws an error when no number is passed.
167 */
168 public void AverageEmptyArray(){
169     Assert.assertThrows(IllegalArgumentException.class,
170         ()->{
171             calculator.Average(new double[]{});
172         });
173 }
174 /**
175 * Factorial Unit Testing Coverage
176 */
177 /**
178 * Test Factorial of Natural Number 5 to return 120
179 */
180 @Test
181 public void testFactorialOfPositiveFive(){
182     double result = calculator.factorial(5);
183     Assert.assertEquals(120,result,0);
184 }
185 /**
186 * Test the factorial of negative number not possible
187 */
188 @Test
189 public void testFactorialOfNegativeTwentySix(){
190     Assert.assertEquals(0,calculator.factorial(-26));
191 }
```