

# Report

Where you can find Code coverage Report?

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

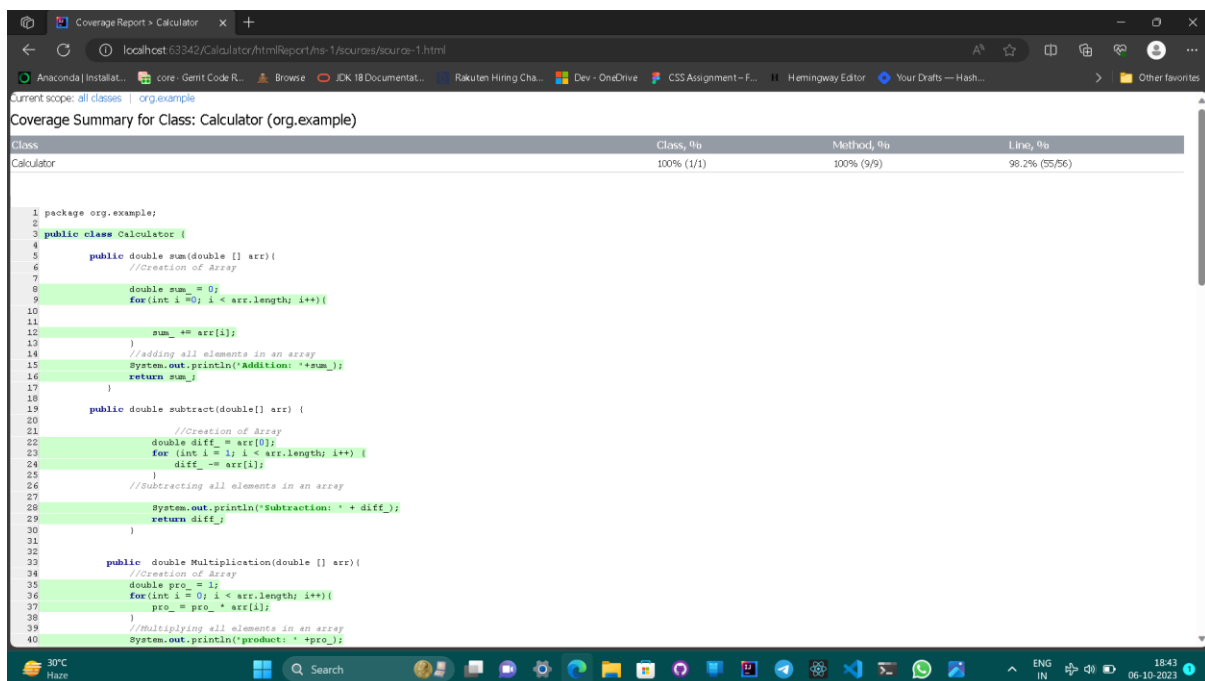
Where You can Find Solutions to the unit test code with test cases passing or not?

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

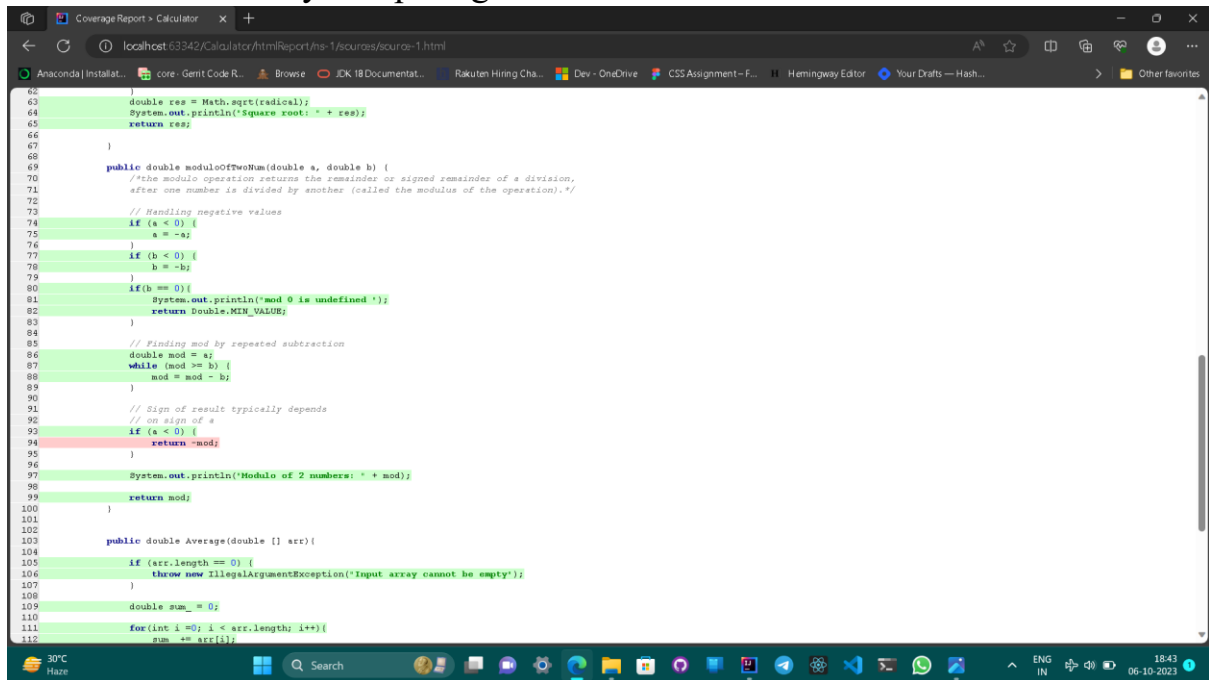
## Covered Multiplication cases.

```
16     return sum;
17 }
18
19 public double subtract(double[] arr) {
20     //Creation of Array
21     double diff = arr[0];
22     for (int i = 1; i < arr.length; i++) {
23         diff -= arr[i];
24     }
25     //Subtracting all elements in an array
26     System.out.println("Subtraction: " + diff);
27     return diff;
28 }
29
30
31
32 public double Multiplication(double [] arr){
33     //Creation of Array
34     double pro = 1;
35     for(int i = 0; i < arr.length; i++){
36         pro = pro * arr[i];
37     }
38     //Multiplying all elements in an array
39     System.out.println("product: " + pro);
40     return pro;
41 }
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 }
```

## Covered division cases.

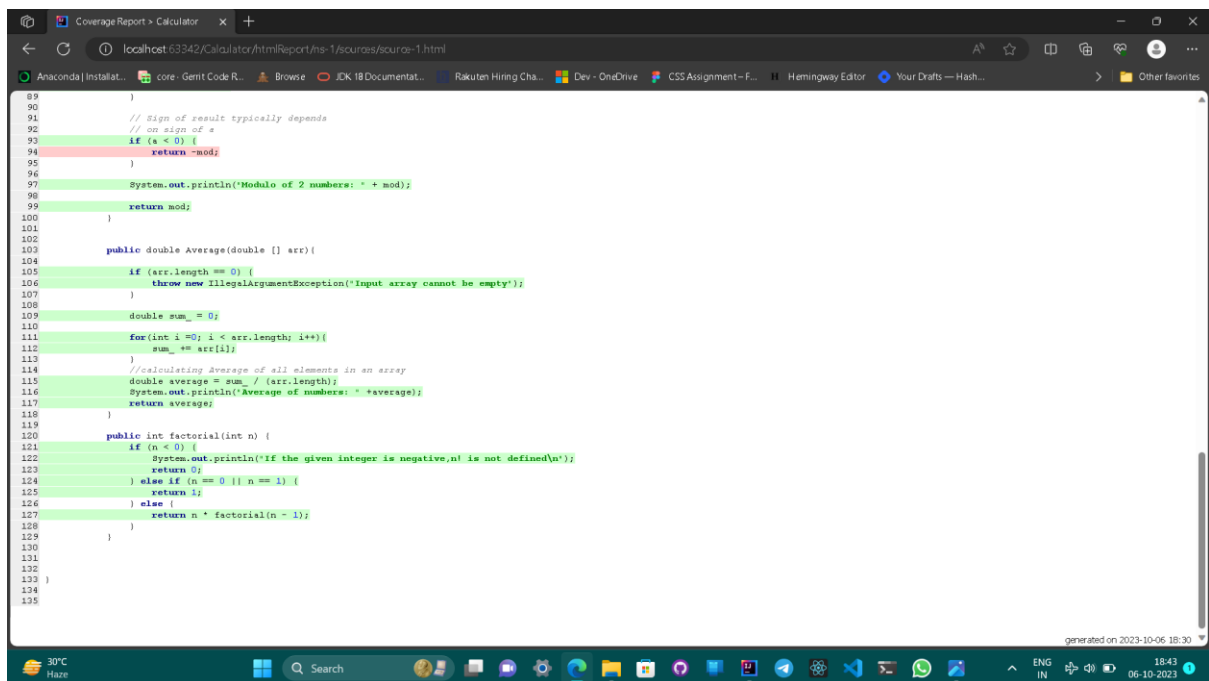
```
42     }
43     return pro;
44 }
45
46 public double divide(double a, double b){
47     if (b == 0) {
48         // if denominator is zero return some minimum value
49         System.out.println("Cannot divide by zero");
50         return Double.MIN_VALUE;
51     }
52     double div = a/b;
53     System.out.println("division of Given Two Numbers is: "+div);
54     return div;
55 }
56
57
58 public double squareRoot(double radical){
59     if(radical < 0){
60         System.out.println("Square root of negative number cannot be real number");
61         return Double.MIN_VALUE;
62     }
63     double res = Math.sqrt(radical);
64     System.out.println("Square root: " + res);
65     return res;
66 }
67
68
69 public double moduloTwoNum(double a, double b) {
70     //this module operation returns the remainder or signed remainder of a division,
71     //after one number is divided by another (called the modulus of the operation).*/
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 }
```

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.



```
62  
63 double res = Math.sqrt(radical);  
64 System.out.println("Square root: " + res);  
65 return res;  
66  
67  
68  
69 public double moduloOffTwoNum(double a, double b) {  
70     //the modulo operation returns the remainder or signed remainder of a division,  
71     //after one number is divided by another (called the modulus of the operation).*/  
72  
73     // Handling negative values  
74     if (a < 0) {  
75         a = -a;  
76     }  
77     if (b < 0) {  
78         b = -b;  
79     }  
80     if (b == 0) {  
81         System.out.println("mod 0 is undefined");  
82         return Double.MIN_VALUE;  
83     }  
84  
85     // Finding mod by repeated subtraction  
86     double mod = a;  
87     while (mod >= b) {  
88         mod = mod - b;  
89     }  
90  
91     // Sign of result typically depends  
92     // on sign of a  
93     if (a < 0) {  
94         return -mod;  
95     }  
96  
97     System.out.println("Modulo of 2 numbers: " + mod);  
98     return mod;  
99 }  
100  
101  
102  
103 public double Average(double [] arr){  
104     if (arr.length == 0) {  
105         throw new IllegalArgumentException("Input array cannot be empty");  
106     }  
107  
108     double sum_ = 0;  
109     for (int i = 0; i < arr.length; i++) {  
110         sum_ += arr[i];  
111     }  
112 }  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135
```

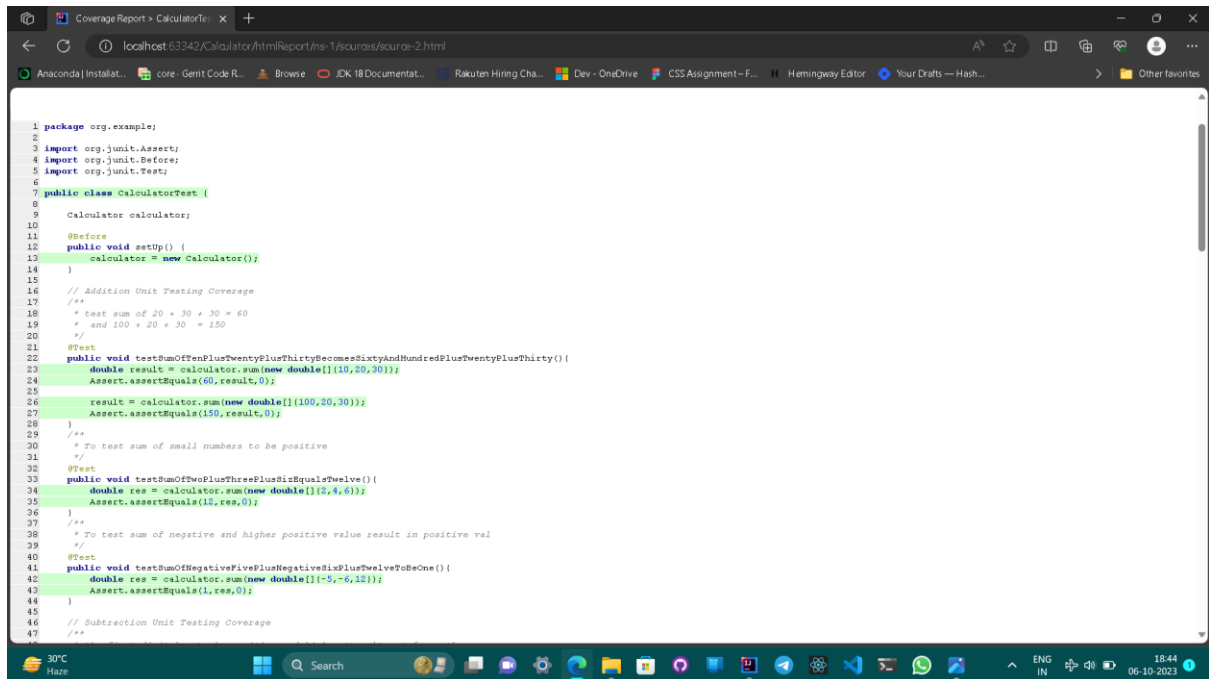
Covered all cases of Averages



```
89  
90  
91     // Sign of result typically depends  
92     // on sign of a  
93     if (a < 0) {  
94         return -mod;  
95     }  
96  
97     System.out.println("Modulo of 2 numbers: " + mod);  
98     return mod;  
99 }  
100  
101  
102  
103 public double Average(double [] arr){  
104     if (arr.length == 0) {  
105         throw new IllegalArgumentException("Input array cannot be empty");  
106     }  
107  
108     double sum_ = 0;  
109     for (int i = 0; i < arr.length; i++) {  
110         sum_ += arr[i];  
111     }  
112  
113     //calculating Average of all elements in an array  
114     double average = sum_ / (arr.length);  
115     System.out.println("Average of numbers: " + average);  
116     return average;  
117 }  
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  
135
```

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

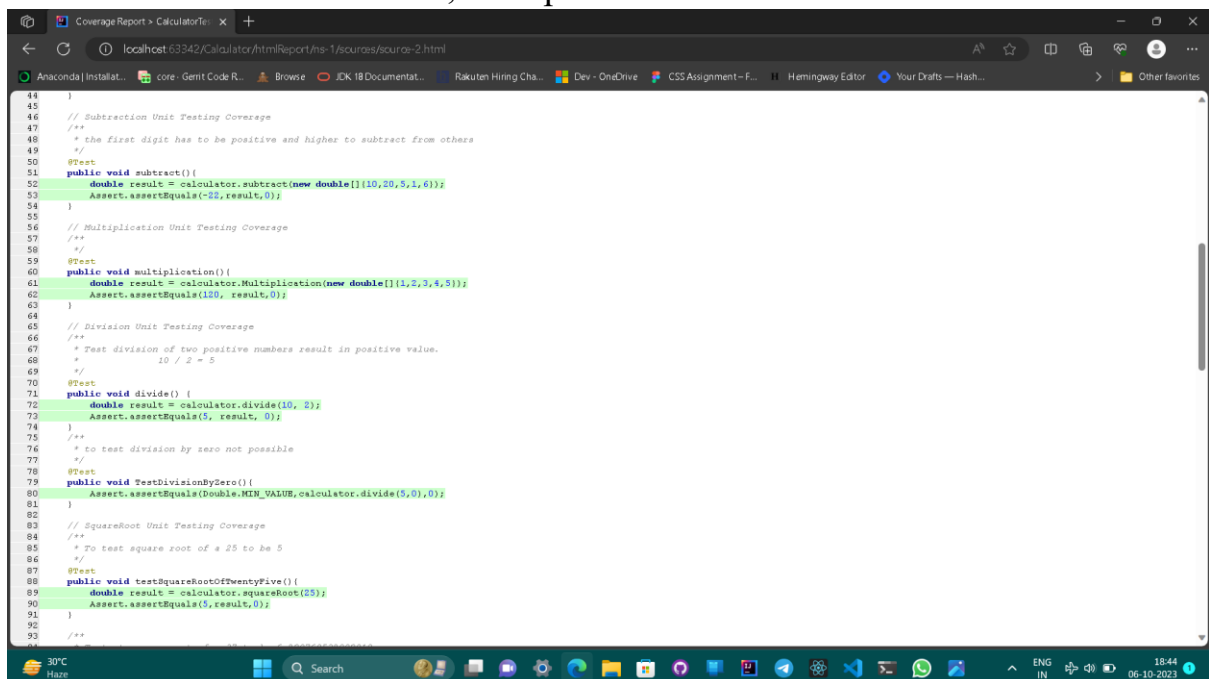
## Basic unit test for addition



The screenshot shows a web browser window displaying a Java code file. The code is for a class named `CalculatorTest` in the package `org.example`. It imports `org.junit.Assert`, `org.junit.Before`, and `org.junit.Test`. The class contains several test methods for addition, including `setUp()`, `testSumOfTenPlusTwentyPlusThirtyBecomesSixtyAndHundredPlusTwentyPlusThirty()`, `testSumOfSmallNumbersToBePositive()`, and `testSumOfNegativeFivePlusNegativeSixPlusTwelveToBeOne()`. The code uses `double` arrays to represent numbers and `Assert.assertEquals` to verify the results. The browser's address bar shows the file path `localhost:63342/Calculator/htmlReport/ms-1/sourc.../source-2.html`. The taskbar at the bottom shows various application icons and the system clock indicating 18:44 on 06-10-2023.

```
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     /**
17      * Addition Unit Testing Coverage
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     /**
49      * Subtraction Unit Testing Coverage
50      */
51 }
```

## Basic unit test for subtraction, multiplication.

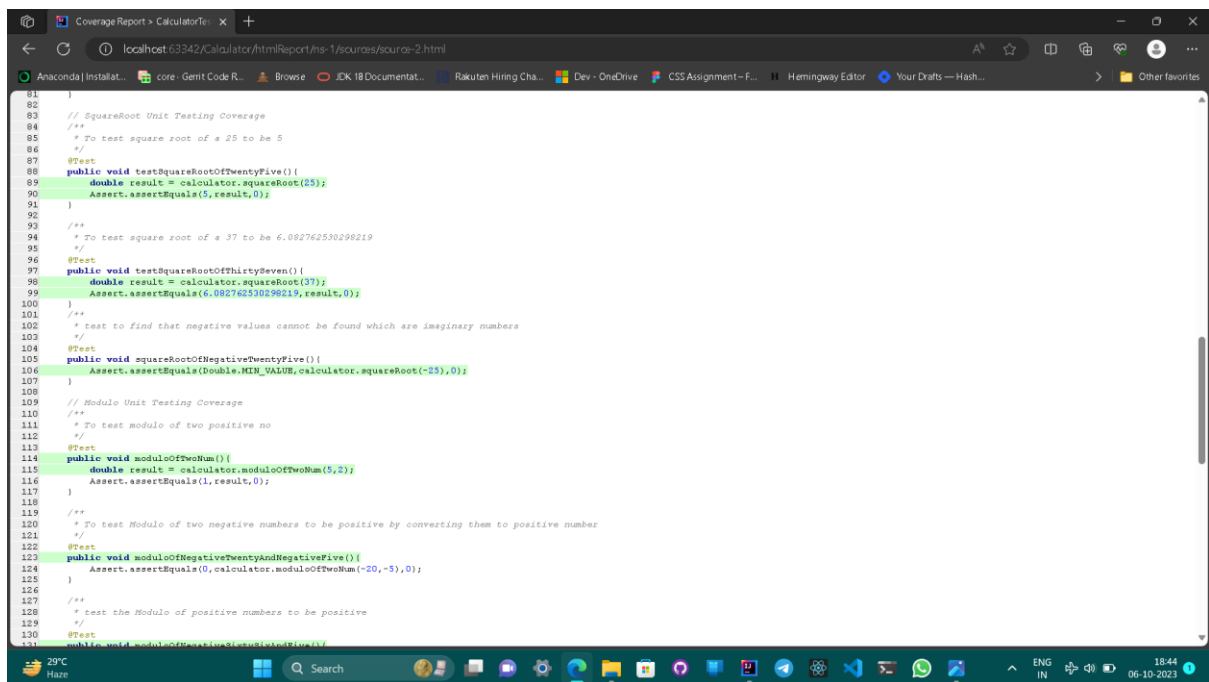


The screenshot shows a web browser window displaying a Java code file. The code continues from the previous one, showing test methods for subtraction, multiplication, division, and square root. The `subtract()` method tests `calculator.subtract(new double[] {10, 20, 5, 1, 6})` and expects `-25`. The `multiplication()` method tests `calculator.Multiplication(new double[] {1, 2, 3, 4, 5})` and expects `120`. The `divide()` method tests `calculator.divide(10, 2)` and expects `5`. There is also a `TestDivisionByZero()` method that expects `Double.MIN_VALUE`. The `squareRoot()` method tests `calculator.squareRoot(25)` and expects `5`. The browser's address bar shows the file path `localhost:63342/Calculator/htmlReport/ms-1/sourc.../source-2.html`. The taskbar at the bottom shows various application icons and the system clock indicating 18:44 on 06-10-2023.

```
44 }
45
46 /**
47      * Subtraction Unit Testing Coverage
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(-25, result, 0);
54     }
55
56     /**
57      * Multiplication Unit Testing Coverage
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     /**
66      * Division Unit Testing Coverage
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     /**
85      * SquareRoot Unit Testing Coverage
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     /**
95      *
96      */
97 }
```

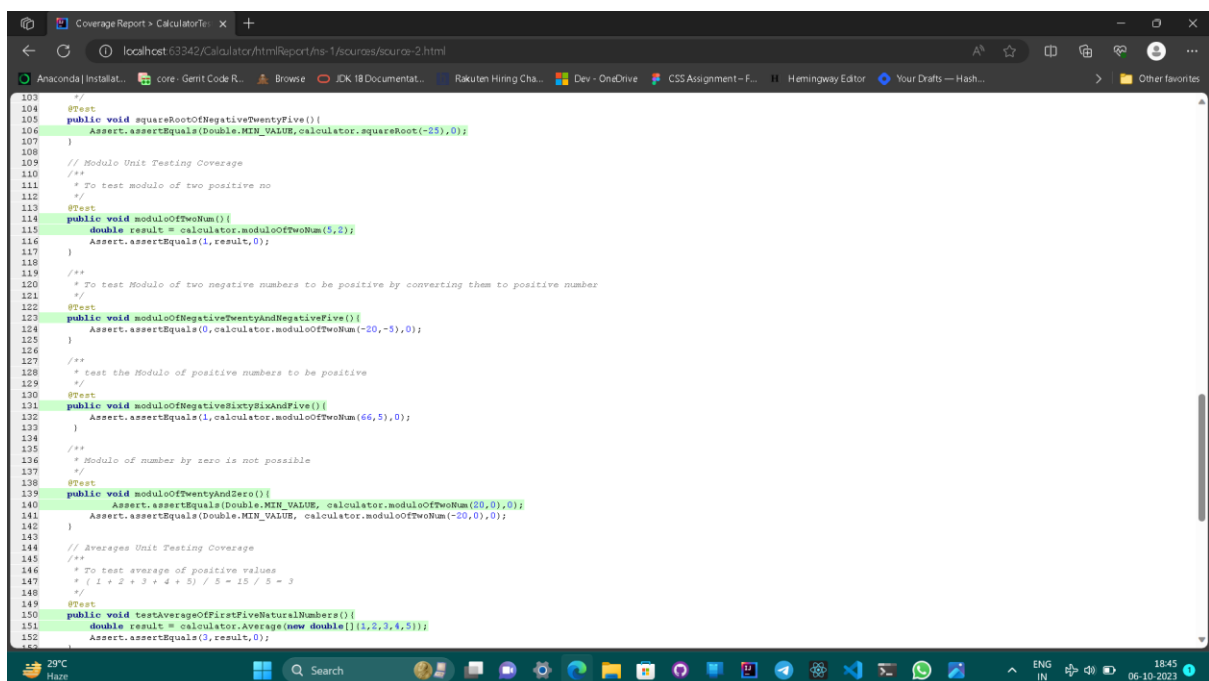


Basic unit test for square root of numbers testing all edge cases.



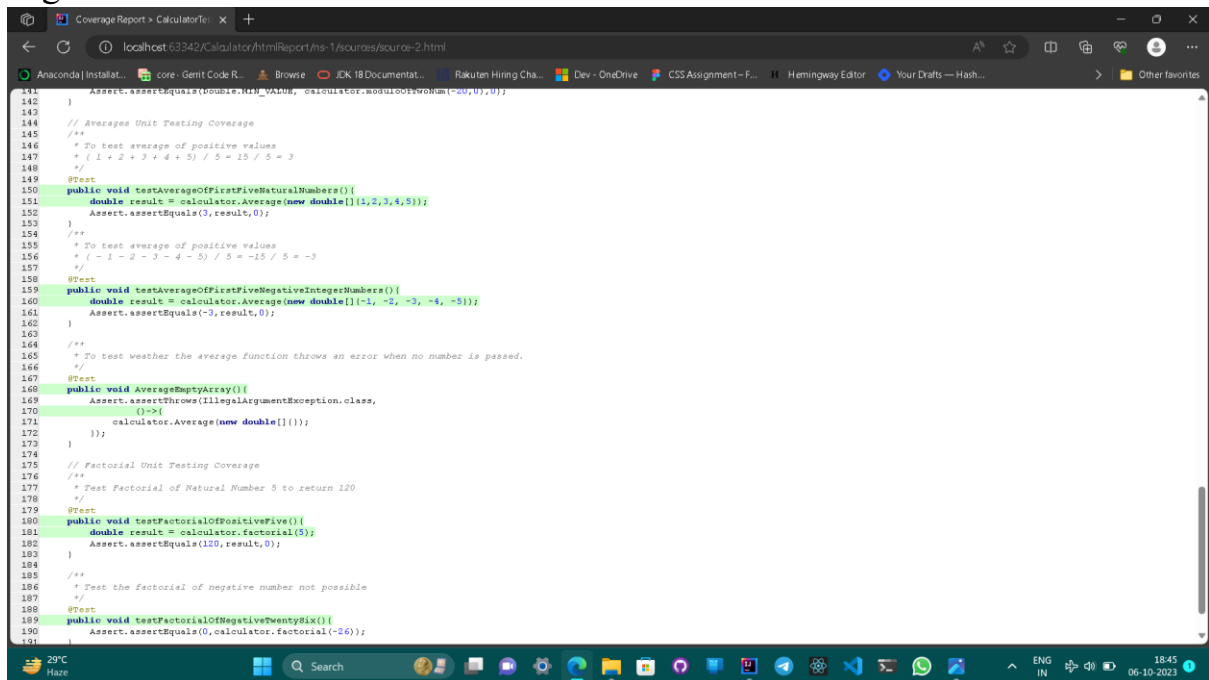
```
81 }
82 // SquareRoot Unit Testing Coverage
83 /**
84  * To test square root of a 25 to be 5
85  */
86 @Test
87 public void testSquareRootOfTwentyFive() {
88     double result = calculator.squareRoot(25);
89     Assert.assertEquals(5, result, 0);
90 }
91 /**
92  * To test square root of a 37 to be 6.082762530298219
93  */
94 @Test
95 public void testSquareRootOfThirtySeven() {
96     double result = calculator.squareRoot(37);
97     Assert.assertEquals(6.082762530298219, result, 0);
98 }
99 /**
100  * test to find that negative values cannot be found which are imaginary numbers
101  */
102 @Test
103 public void squareRootOfNegativeTwentyFive() {
104     Assert.assertEquals(Double.MIN_VALUE, calculator.squareRoot(-25), 0);
105 }
106 // Module Unit Testing Coverage
107 /**
108  * To test modulo of two positive no
109  */
110 @Test
111 public void moduloOfTwoNum() {
112     double result = calculator.moduloOfTwoNum(5, 3);
113     Assert.assertEquals(1, result, 0);
114 }
115 /**
116  * To test Modulo of two negative numbers to be positive by converting them to positive number
117  */
118 @Test
119 public void moduloOfNegativeTwentyAndNegativeFive() {
120     Assert.assertEquals(0, calculator.moduloOfTwoNum(-20, -5), 0);
121 }
122 /**
123  * test the Modulo of positive numbers to be positive
124  */
125 @Test
126 public void moduloOfNegativeSixtyAndFive() {
127     Assert.assertEquals(1, calculator.moduloOfTwoNum(-60, 5), 0);
128 }
```

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



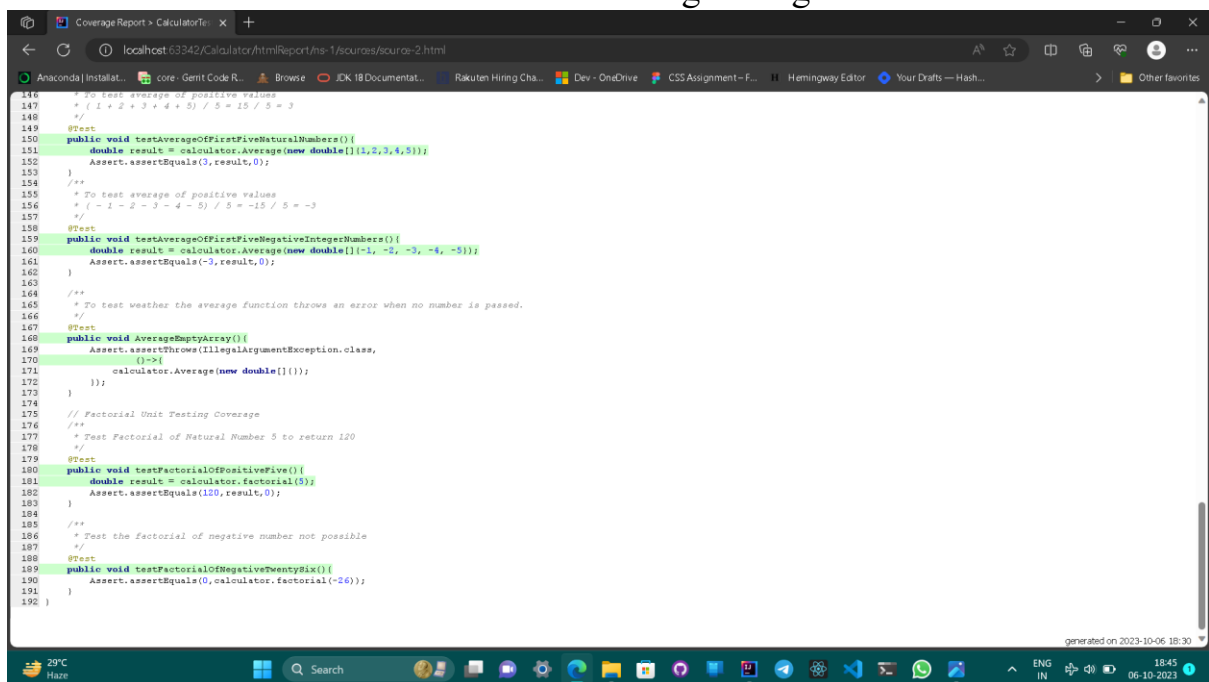
```
103 //
104 @Test
105 public void squareRootOfNegativeTwentyFive() {
106     Assert.assertEquals(Double.MIN_VALUE, calculator.squareRoot(-25), 0);
107 }
108 // Module Unit Testing Coverage
109 /**
110  * To test modulo of two positive no
111  */
112 @Test
113 public void moduloOfTwoNum() {
114     double result = calculator.moduloOfTwoNum(5, 3);
115     Assert.assertEquals(1, result, 0);
116 }
117 /**
118  * To test Modulo of two negative numbers to be positive by converting them to positive number
119  */
120 @Test
121 public void moduloOfNegativeTwentyAndNegativeFive() {
122     Assert.assertEquals(0, calculator.moduloOfTwoNum(-20, -5), 0);
123 }
124 /**
125  * test the Modulo of positive numbers to be positive
126  */
127 @Test
128 public void moduloOfNegativeSixtyAndFive() {
129     Assert.assertEquals(1, calculator.moduloOfTwoNum(-60, 5), 0);
130 }
131 /**
132  * Modulo of number by zero is not possible
133  */
134 @Test
135 public void moduloOfTwentyAndZero() {
136     Assert.assertEquals(Double.MIN_VALUE, calculator.moduloOfTwoNum(20, 0), 0);
137     Assert.assertEquals(Double.MIN_VALUE, calculator.moduloOfTwoNum(-20, 0), 0);
138 }
139 // Averages Unit Testing Coverage
140 /**
141  * To test average of positive values
142  * (1 + 2 + 3 + 4 + 5) / 5 = 15 / 5 = 3
143  */
144 @Test
145 public void testAverageOfFirstFiveNaturalNumbers() {
146     double result = calculator.Average(new double[] {1, 2, 3, 4, 5});
147     Assert.assertEquals(3, result, 0);
148 }
```

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



```
141 }
142 }
143 // Averages Unit Testing Coverage
144 /**
145  * To test average of positive values
146  * ( 1 + 2 + 3 + 4 + 5 ) / 5 = 15 / 5 = 3
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 /**
154  * To test average of positive values
155  * ( - 1 - 2 - 3 - 4 - 5 ) / 5 = -15 / 5 = -3
156  */
157 @Test
158 public void testAverageOfFirstFiveNegativeIntegerNumbers() {
159     double result = calculator.Average(new double[] { -1, -2, -3, -4, -5 });
160     Assert.assertEquals(-3, result, 0);
161 }
162 }
163 /**
164  * To test whether the average function throws an error when no number is passed.
165  */
166 @Test
167 public void AverageEmptyArray() {
168     Assert.assertThrows(IllegalArgumentException.class,
169         () -> {
170             calculator.Average(new double[] {});
171         });
172 }
173 }
174 // Factorial Unit Testing Coverage
175 /**
176  * Test Factorial of Natural Number 5 to return 120
177  */
178 @Test
179 public void testFactorialOfPositiveFive() {
180     double result = calculator.factorial(5);
181     Assert.assertEquals(120, result, 0);
182 }
183 }
184 /**
185  * Test the factorial of negative number not possible
186  */
187 @Test
188 public void testFactorialOfNegativeTwentySix() {
189     Assert.assertEquals(0, calculator.factorial(-26));
190 }
191 }
```

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



```
141 // To test average of positive values
142 * ( 1 + 2 + 3 + 4 + 5 ) / 5 = 15 / 5 = 3
143 */
144 @Test
145 public void testAverageOfFirstFiveNaturalNumbers() {
146     double result = calculator.Average(new double[] {1, 2, 3, 4, 5});
147     Assert.assertEquals(3, result, 0);
148 }
149 /**
150  * To test average of positive values
151  * ( - 1 - 2 - 3 - 4 - 5 ) / 5 = -15 / 5 = -3
152  */
153 @Test
154 public void testAverageOfFirstFiveNegativeIntegerNumbers() {
155     double result = calculator.Average(new double[] { -1, -2, -3, -4, -5 });
156     Assert.assertEquals(-3, result, 0);
157 }
158 }
159 /**
160  * To test whether the average function throws an error when no number is passed.
161  */
162 @Test
163 public void AverageEmptyArray() {
164     Assert.assertThrows(IllegalArgumentException.class,
165         () -> {
166             calculator.Average(new double[] {});
167         });
168 }
169 }
170 // Factorial Unit Testing Coverage
171 /**
172  * Test Factorial of Natural Number 5 to return 120
173  */
174 @Test
175 public void testFactorialOfPositiveFive() {
176     double result = calculator.factorial(5);
177     Assert.assertEquals(120, result, 0);
178 }
179 }
180 /**
181  * Test the factorial of negative number not possible
182  */
183 @Test
184 public void testFactorialOfNegativeTwentySix() {
185     Assert.assertEquals(0, calculator.factorial(-26));
186 }
187 }
188 }
189 }
190 }
191 }
192 }
```