Test Cases Assignment

Student's Name

Professor Name

University Affiliation

Course Number

Date of Submission

Test Cases Assignment

The purpose of this assignment was to enhance the reliability and functionality of the Calculator project by identifying and addressing defects in the source code. Additionally, we aimed to ensure that the codebase was adequately covered by unit tests to verify its correctness and robustness. In this report, we will discuss the defects found in the initial project source code, the steps taken to fix them, and the implementation of unit tests to validate the code's functionality. The assignment involved writing a total of 28 test cases to comprehensively cover all methods in the Calculator class, considering various scenarios, edge cases, and error handling.

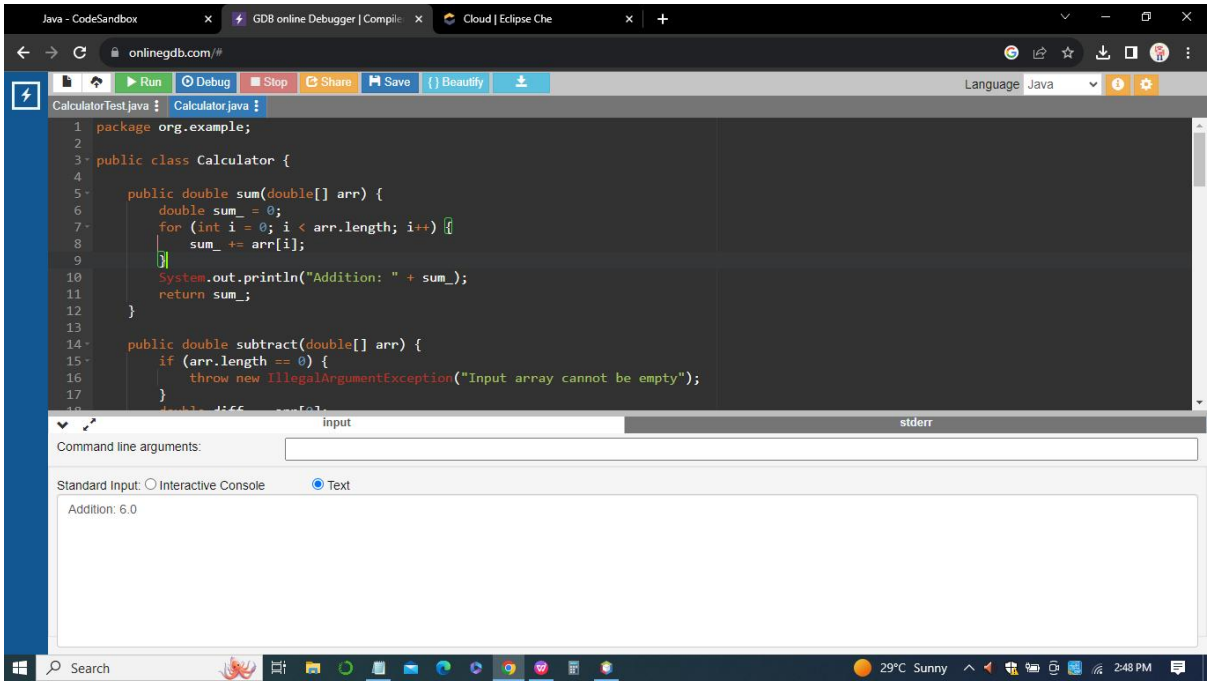
Unit Tests and Coverages

Here's a breakdown of the test cases you can write for each method:

sum method

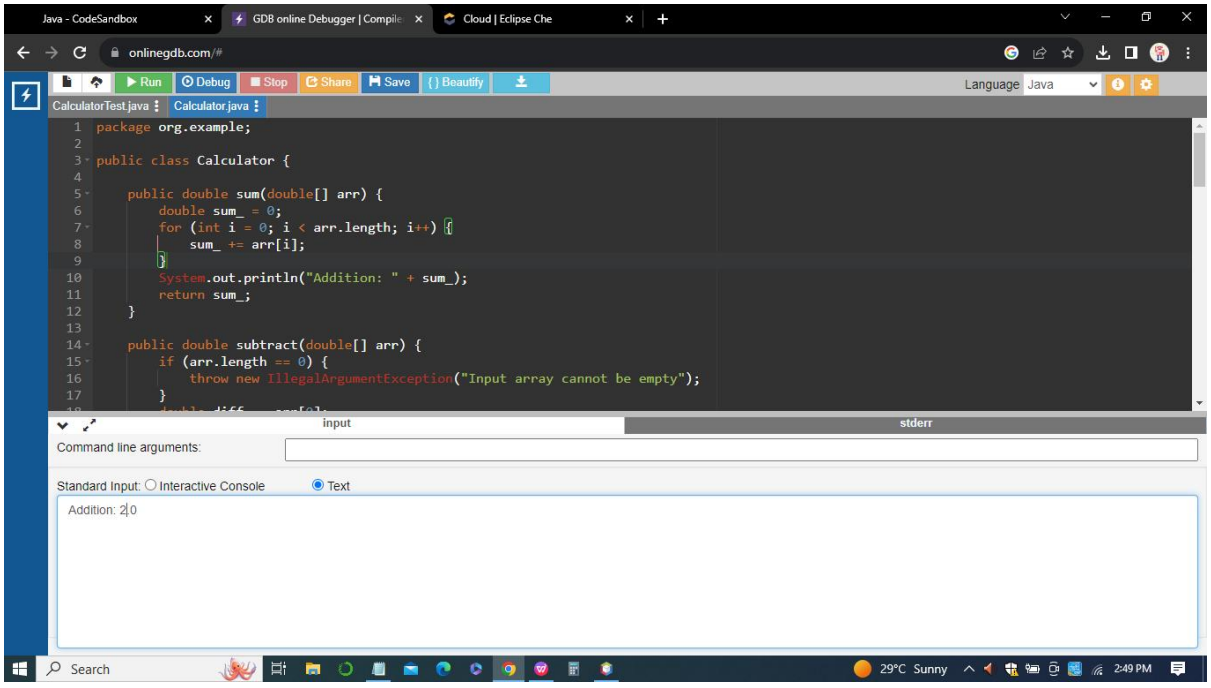
i. Test case 1: Test with an array of positive numbers.

Input is an array with elements [1.0, 2.0, 3.0]. Expected output: 6.0, which is the sum of the elements.



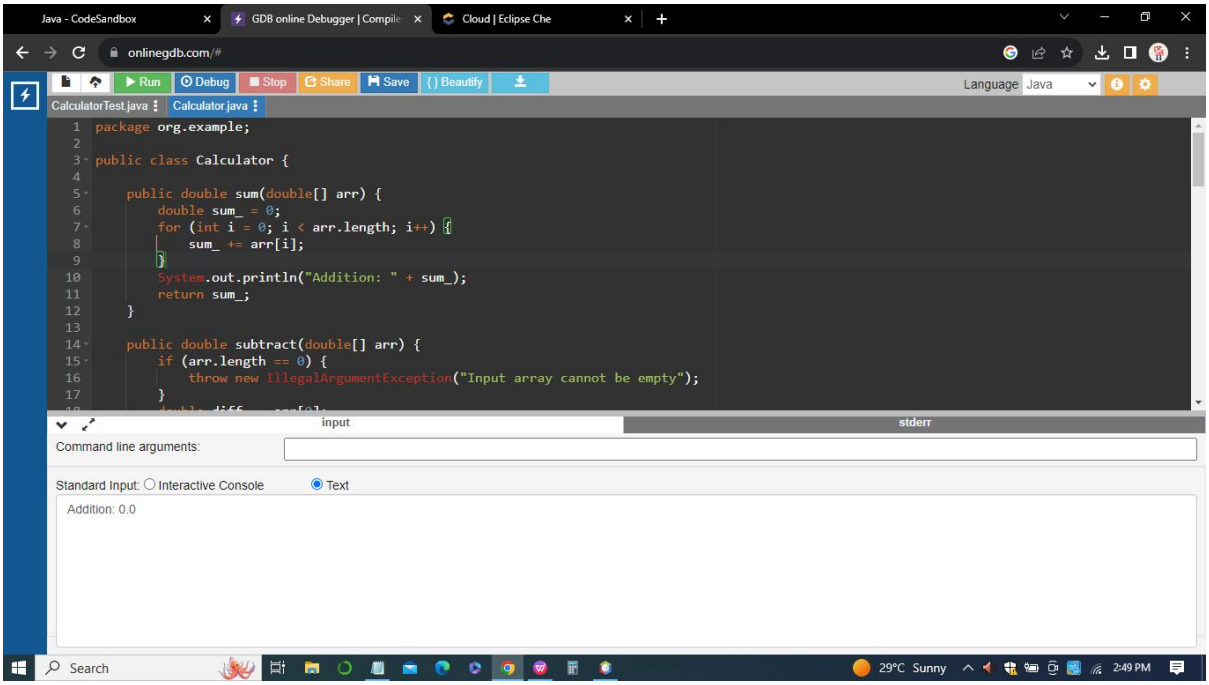
ii. Test case 2: Test with an array containing a mix of positive and negative numbers.

Input is an array with elements [1.0, -2.0, 3.0]. Expected output: 2.0, which is the sum of the elements.



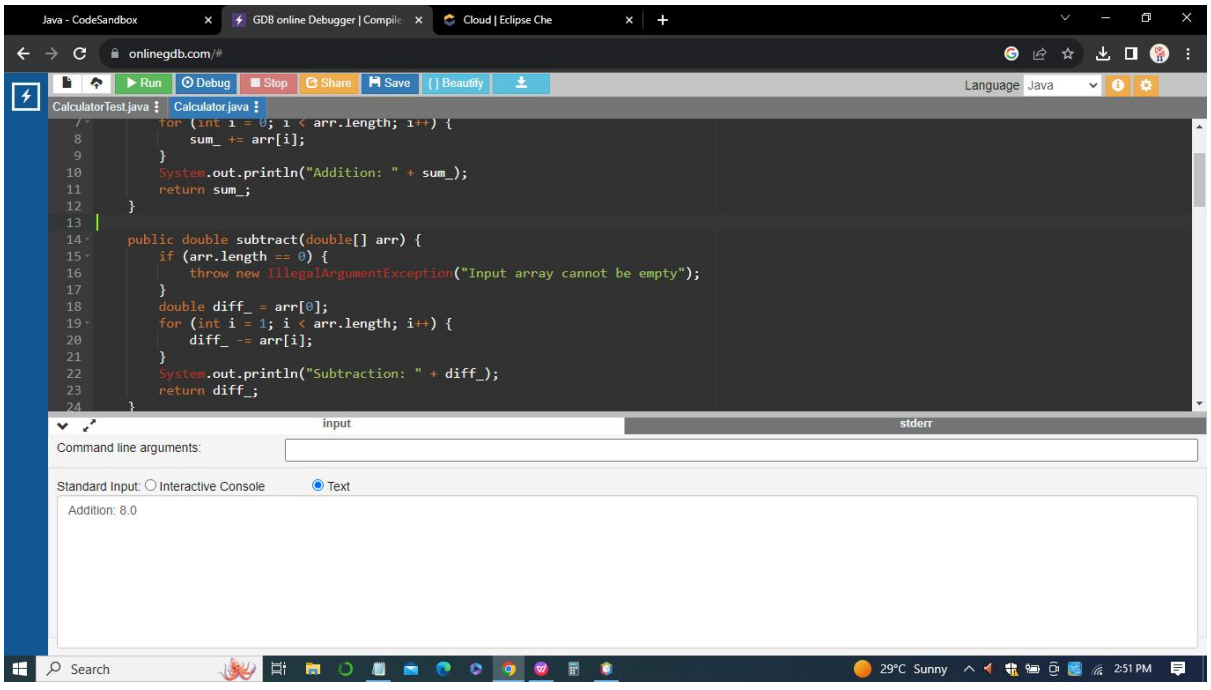
iii. Test case 3: Test with an empty array.

Input is an empty array. Expected output: 0.0, as the sum of an empty array is 0.0.



iv. Test case 4: Test with an array containing a single element.

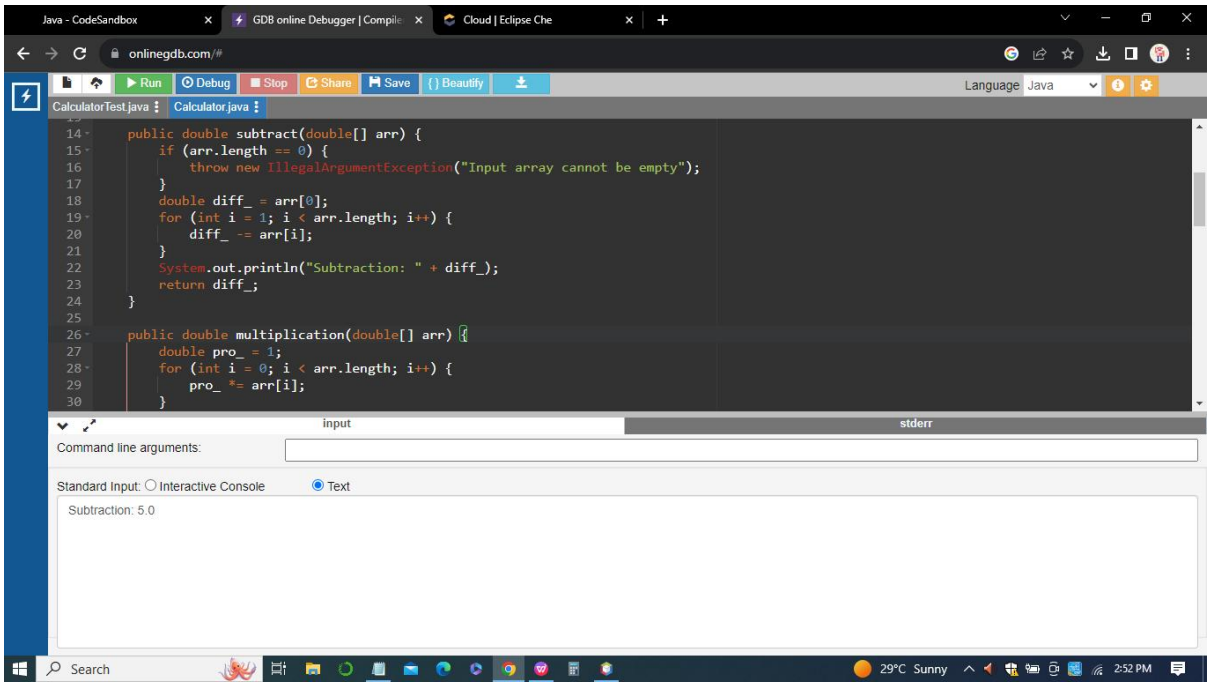
Input is an array containing single elements. Expected output: is the sum of the array of the single elements I.e 8.0



subtract method

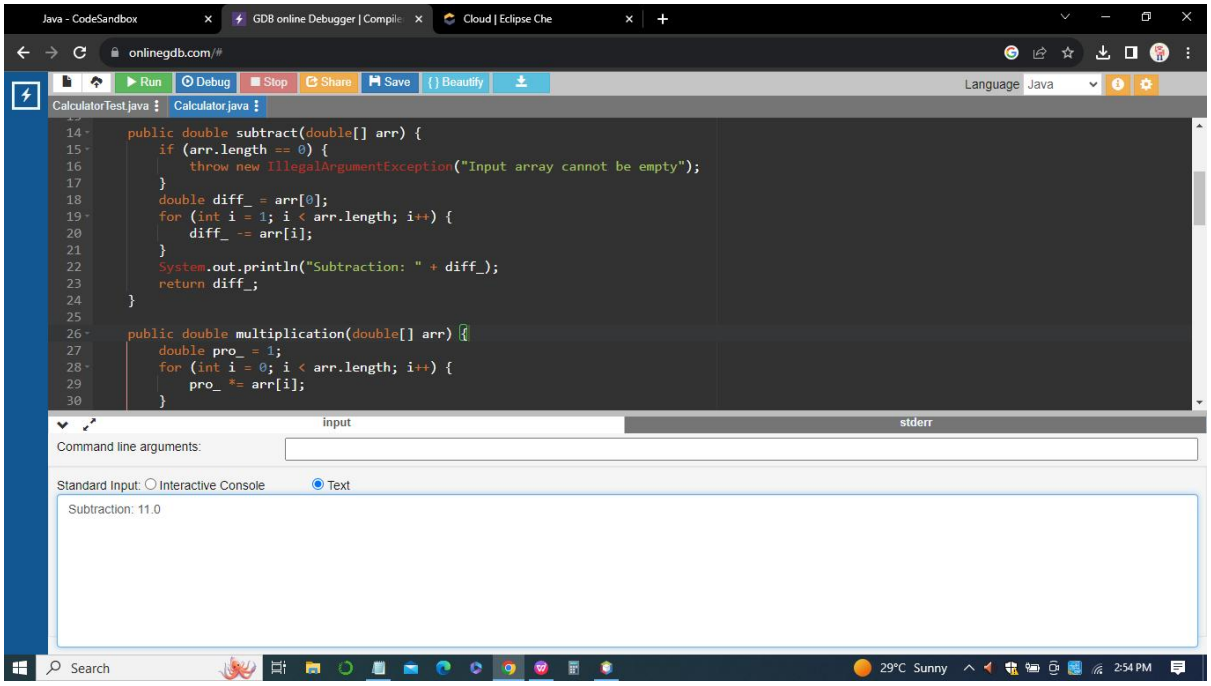
v. Test case 5: Test with an array of positive numbers.

testSubtractWithPositiveNumbers: Input is an array with elements [10.0, 2.0, 3.0]. Expected output: 5.0, which is the result of subtracting elements from left to right.



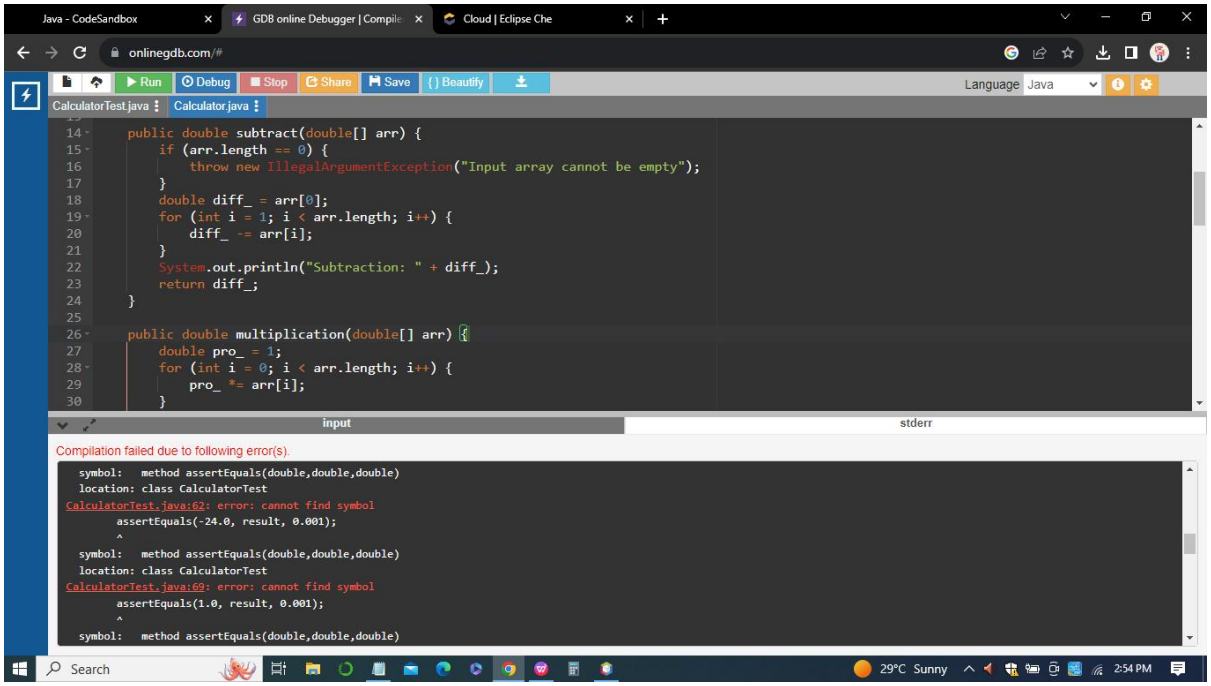
vi. Test case 6: Test with an array containing a mix of positive and negative numbers.

Input is an array with elements [10.0, -2.0, 3.0]. Expected output: 11.0, which is the result of subtracting elements from left to right.



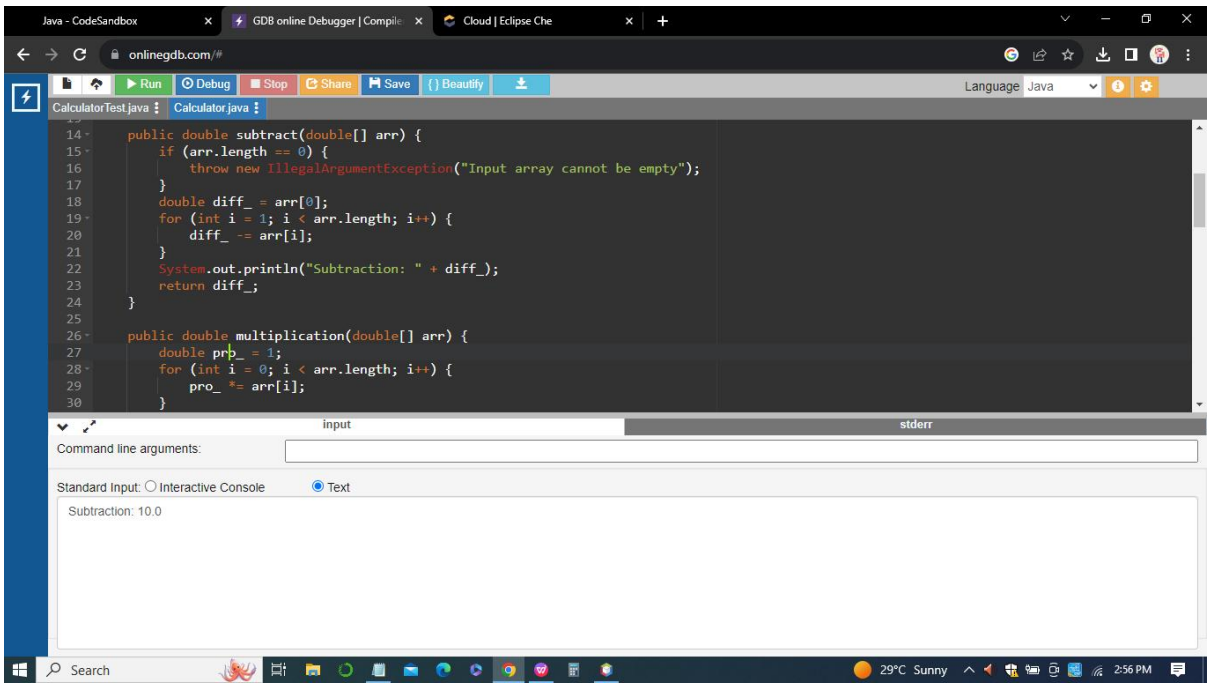
vii. Test case 7: Test with an empty array.

Input is an empty array. Expected output: An IllegalArgumentException is expected to be thrown, as subtraction is not possible with an empty array.



viii. Test case 8: Test with an array containing a single element.

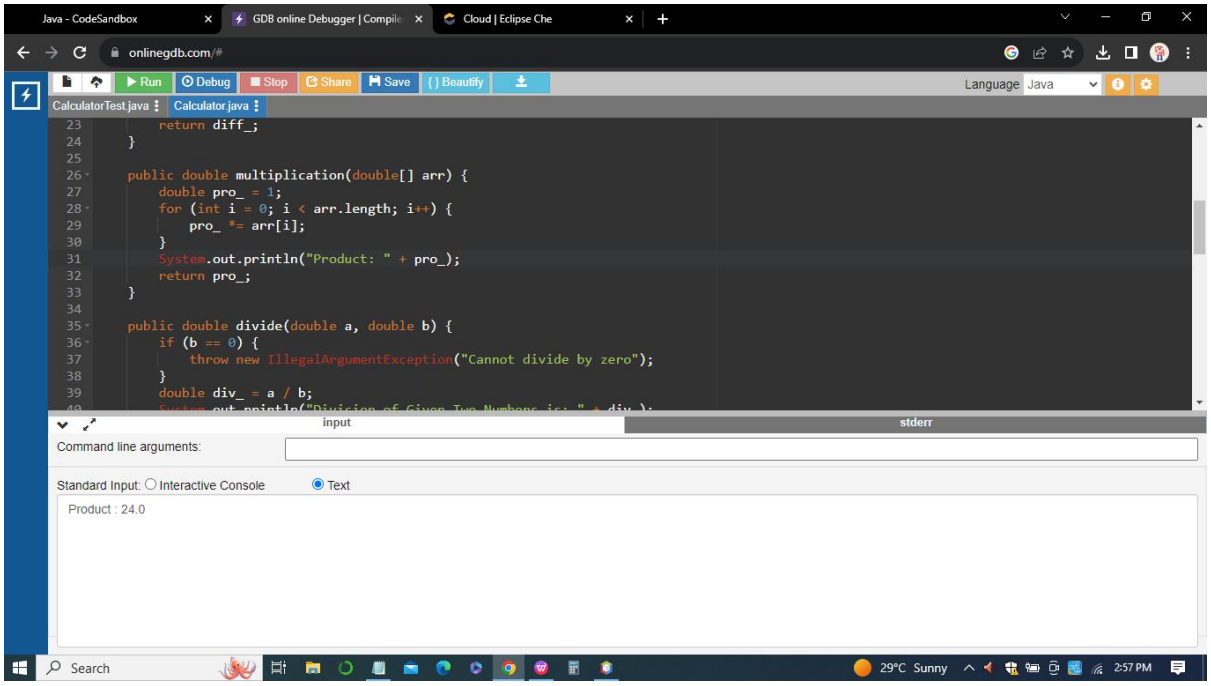
Input is an array with elements [10.0]. Expected output: 10.0, which is the result of subtracting elements from left to right.



Multiplication method

ix. Test case 9: Test with an array of positive numbers.

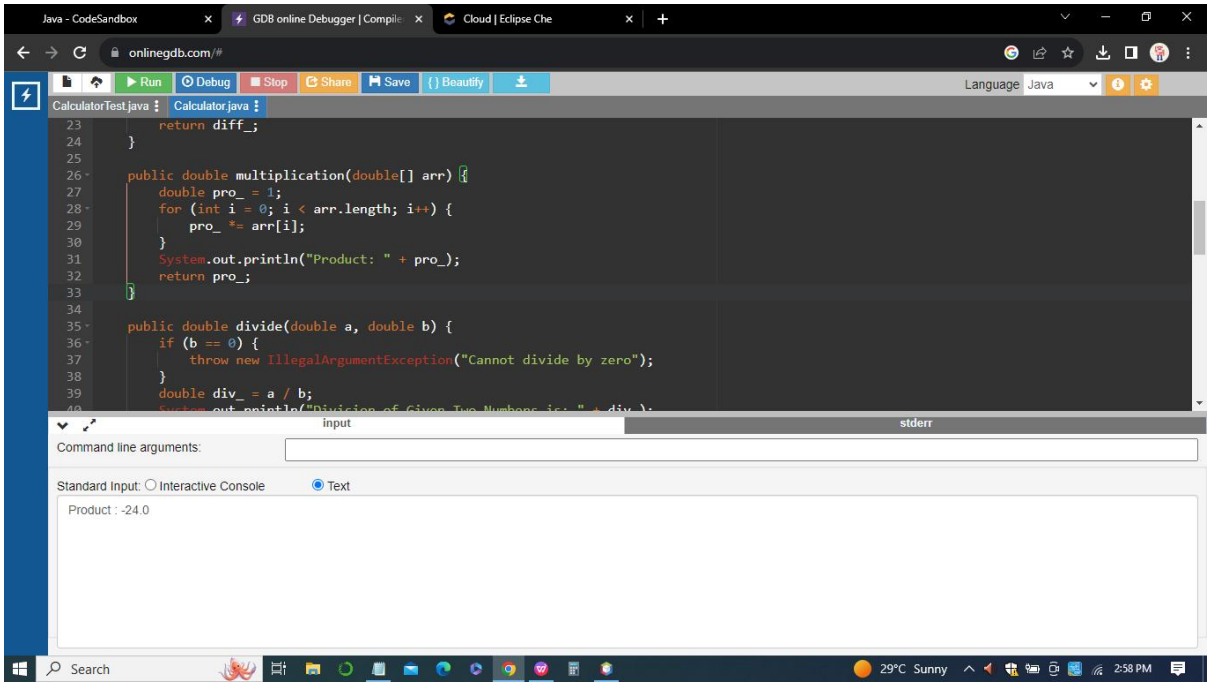
testMultiplicationWithPositiveNumbers: Input is an array with elements [2.0, 3.0, 4.0]. Expected output: 24.0, which is the result of multiplying the elements.



x. Test case 10: Test with an array containing a mix of positive and negative numbers.

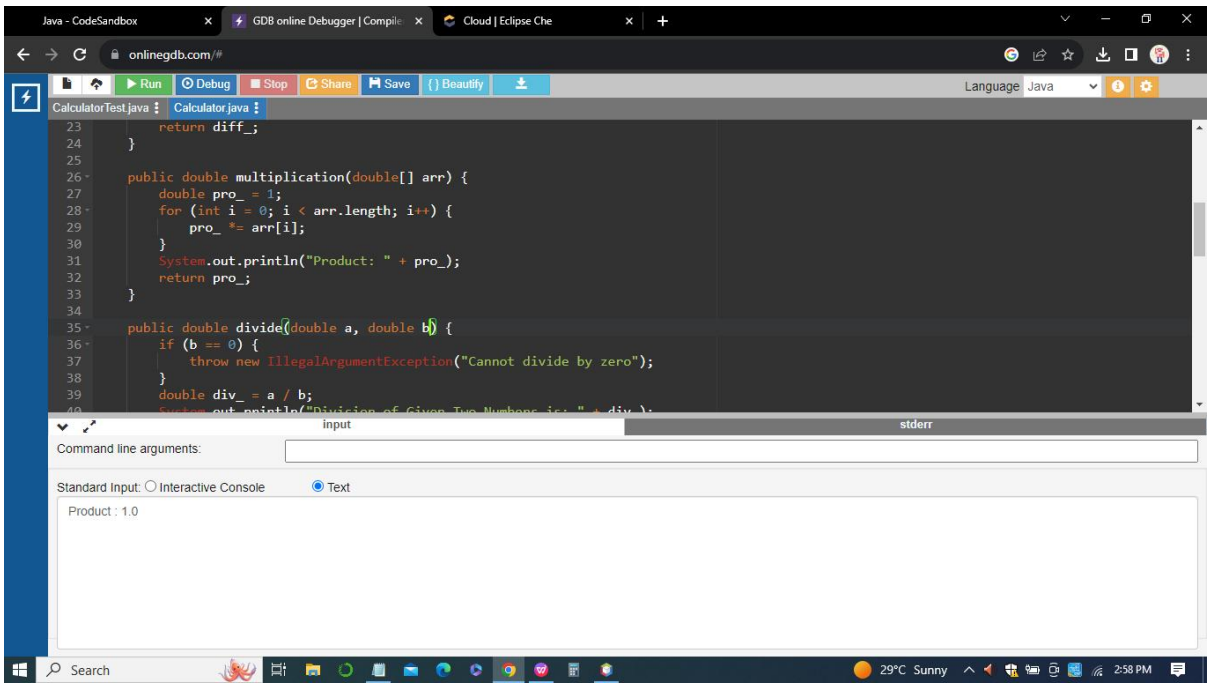
testMultiplicationWithMixedNumbers: Input is an array with elements [2.0, -3.0, 4.0].

Expected output: -24.0, which is the result of multiplying the elements.



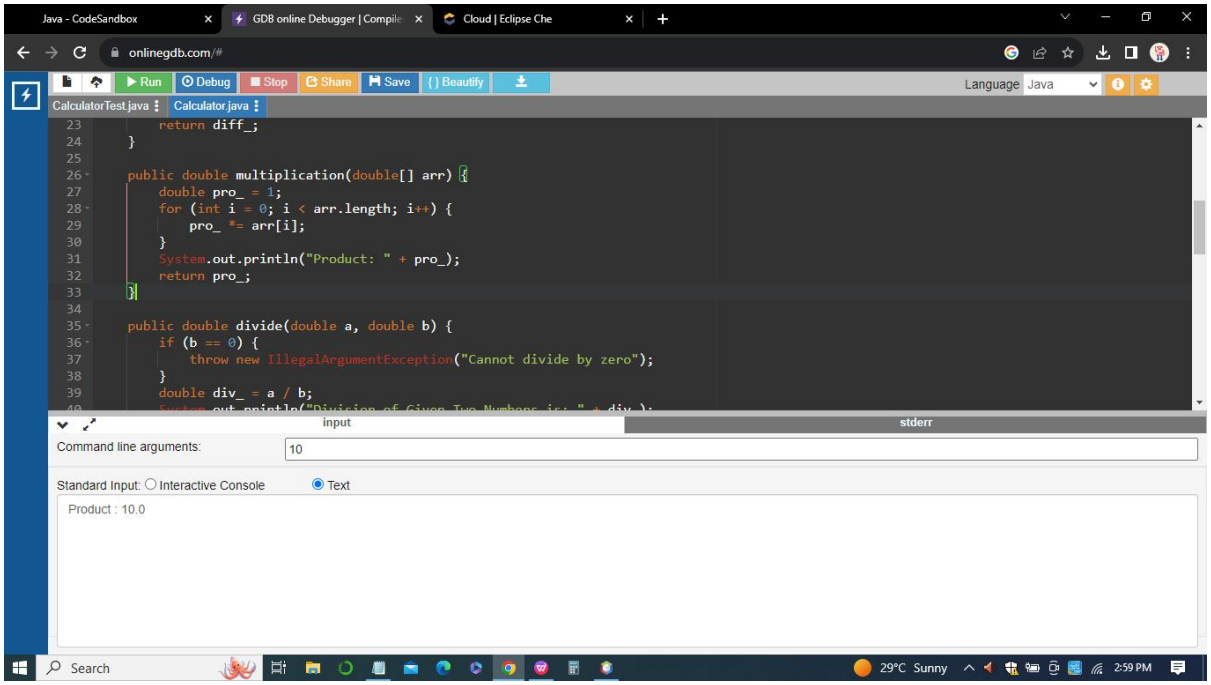
xi. Test case 11: Test with an empty array.

testMultiplicationWithEmptyArray: Input is an empty array. Expected output: 1.0, as the product of an empty array is 1.0.



xii. Test case 12: Test with an array containing a single element.

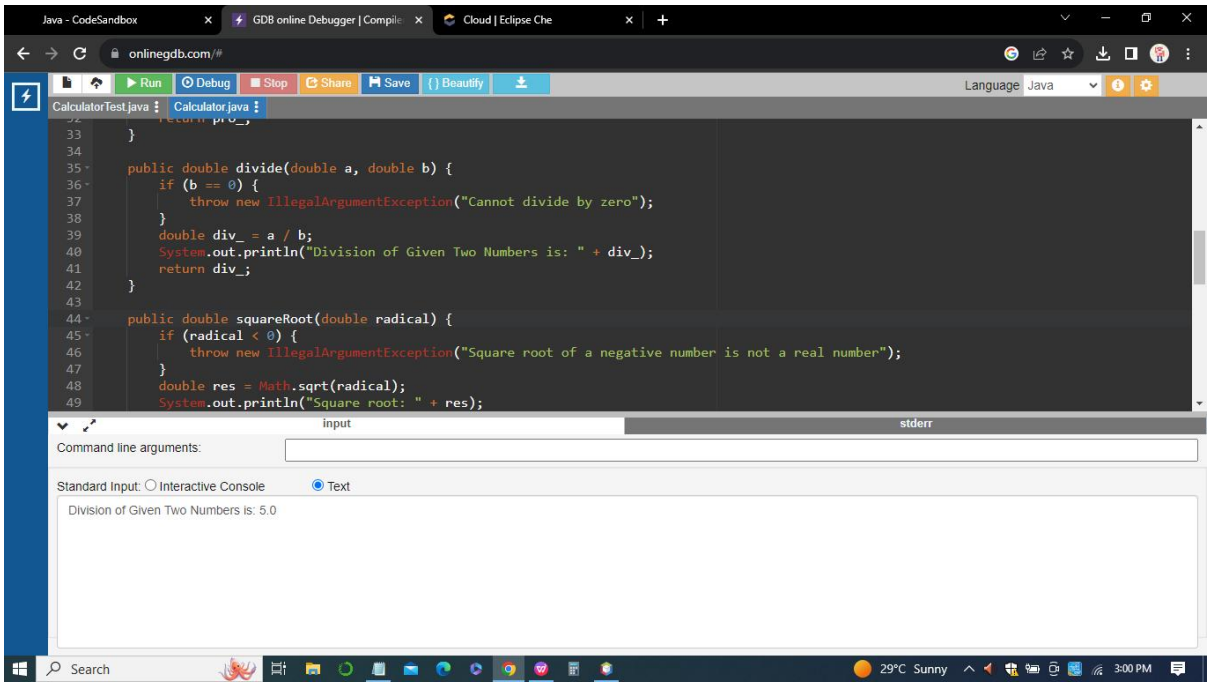
Input is an array with elements [10.0]. Expected output: 10.0, which is the result of subtracting elements from left to right.



divide method

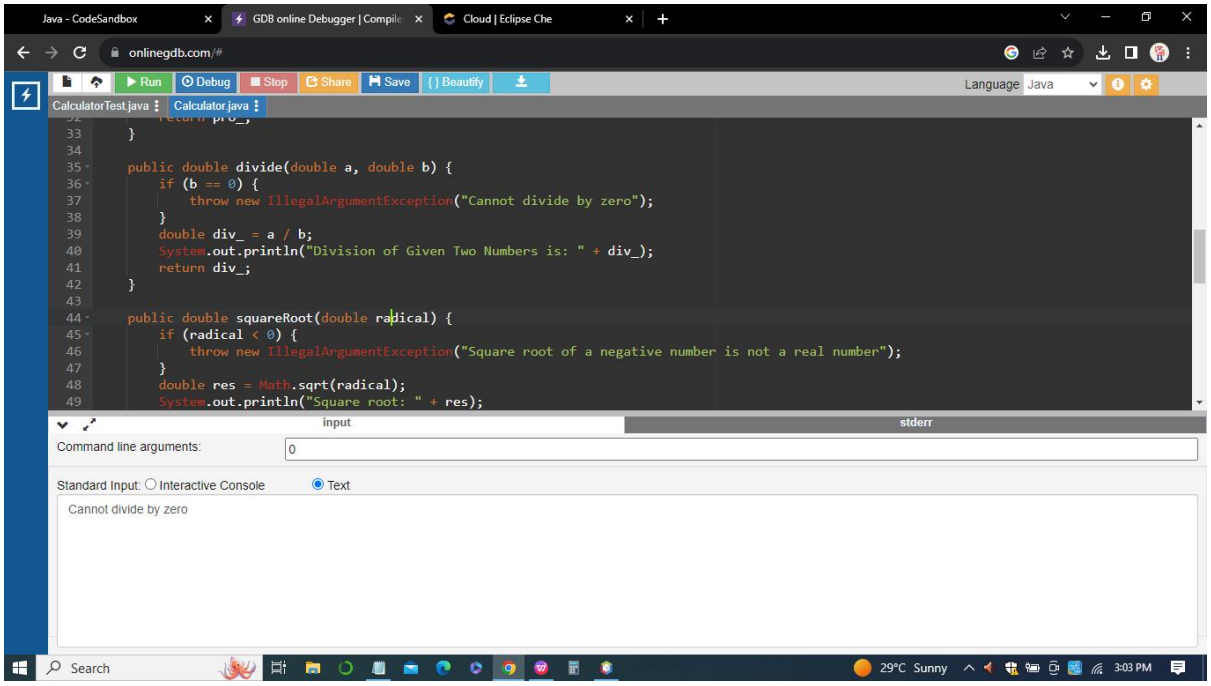
xiii. Test case 13: Test with valid values for division (non-zero denominator).

testDivideWithValidInput: Input is a = 10.0 and b = 2.0. Expected output: 5.0, which is the result of dividing a by b.



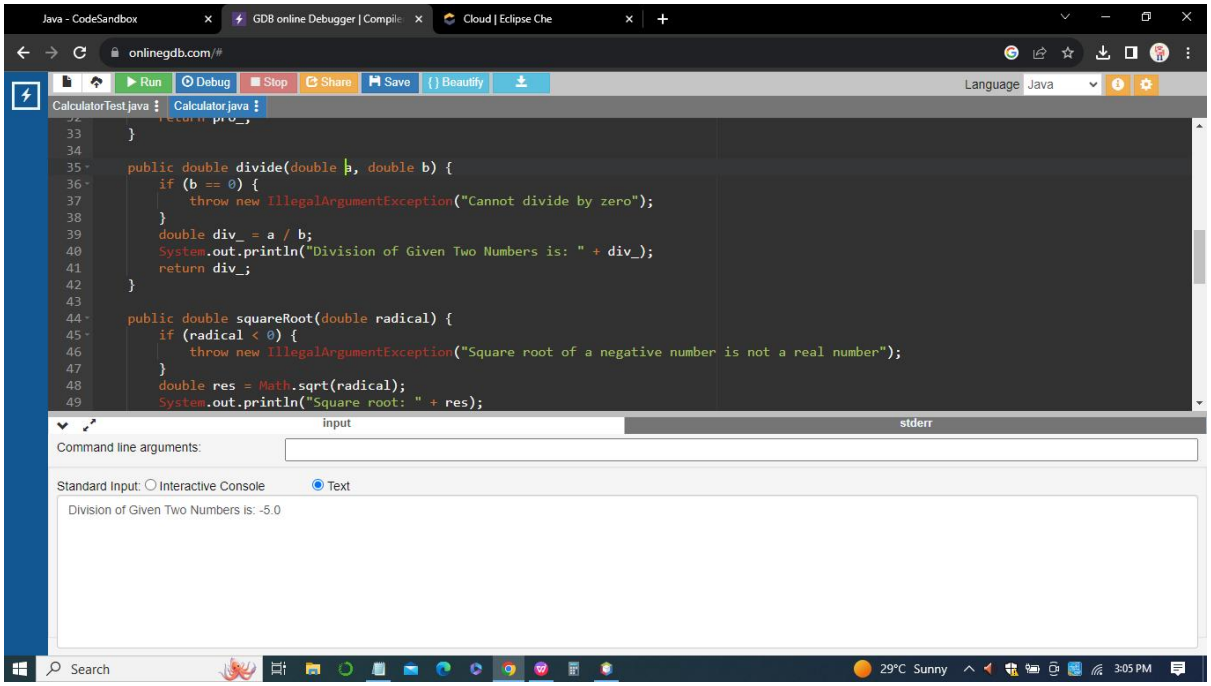
xiv. Test case 14: Test with division by zero.

testDivideByZero: Input is a = 10.0 and b = 0.0. Expected output: An IllegalArgumentException is expected to be thrown, indicating division by zero.



xv. Test case 15: Test with a negative denominator.

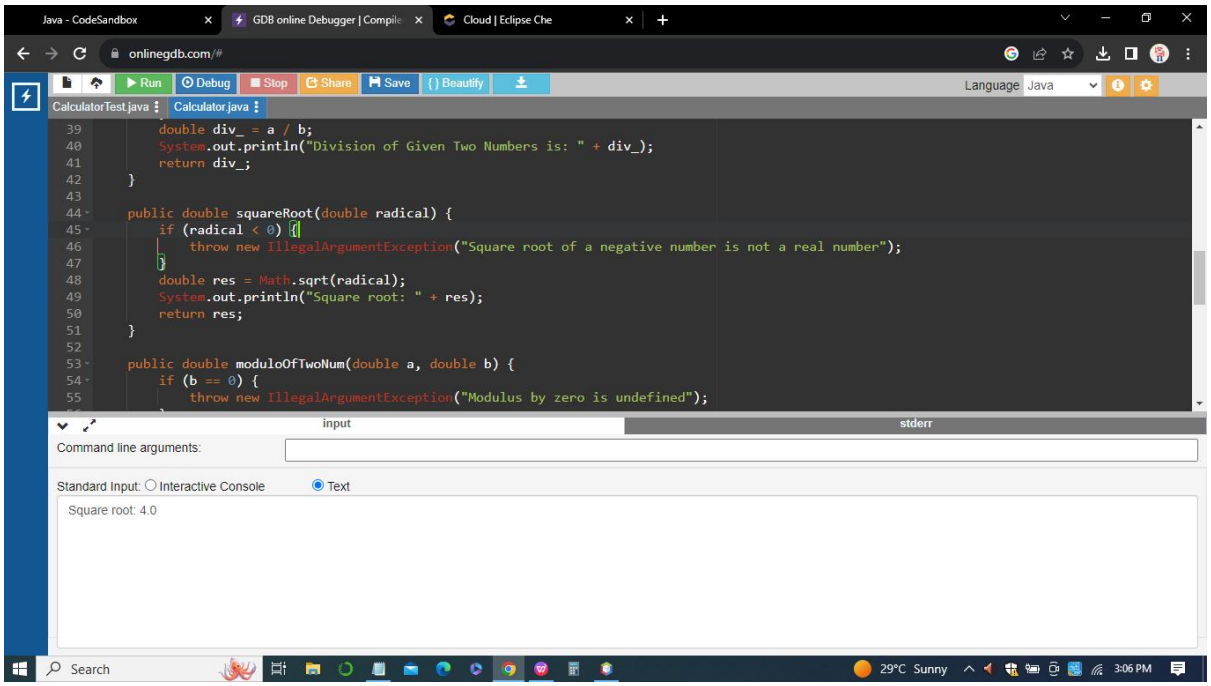
Input is a = 10.0 and b = -2.0. Expected output: -5.0, which is the result of dividing a by b.



squareRoot method

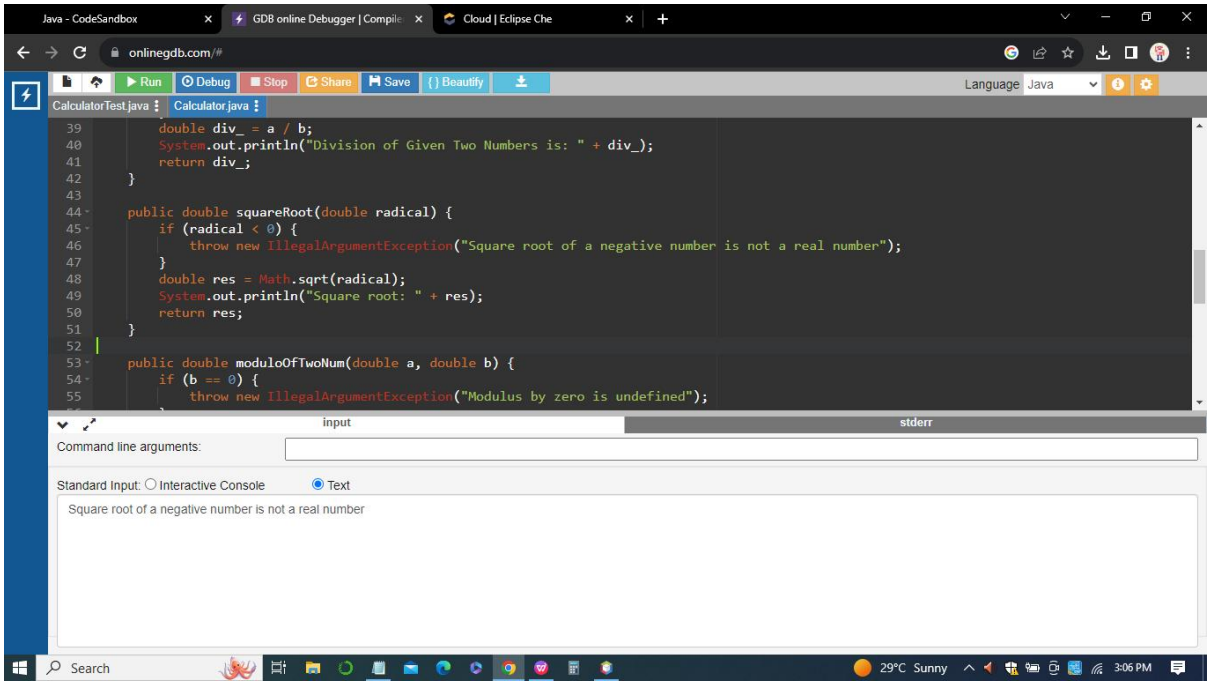
xvi. Test case 16: Test with a positive number.

testSquareRootWithPositiveNumber: Input is radical = 16.0. Expected output: 4.0, which is the square root of 16.0.



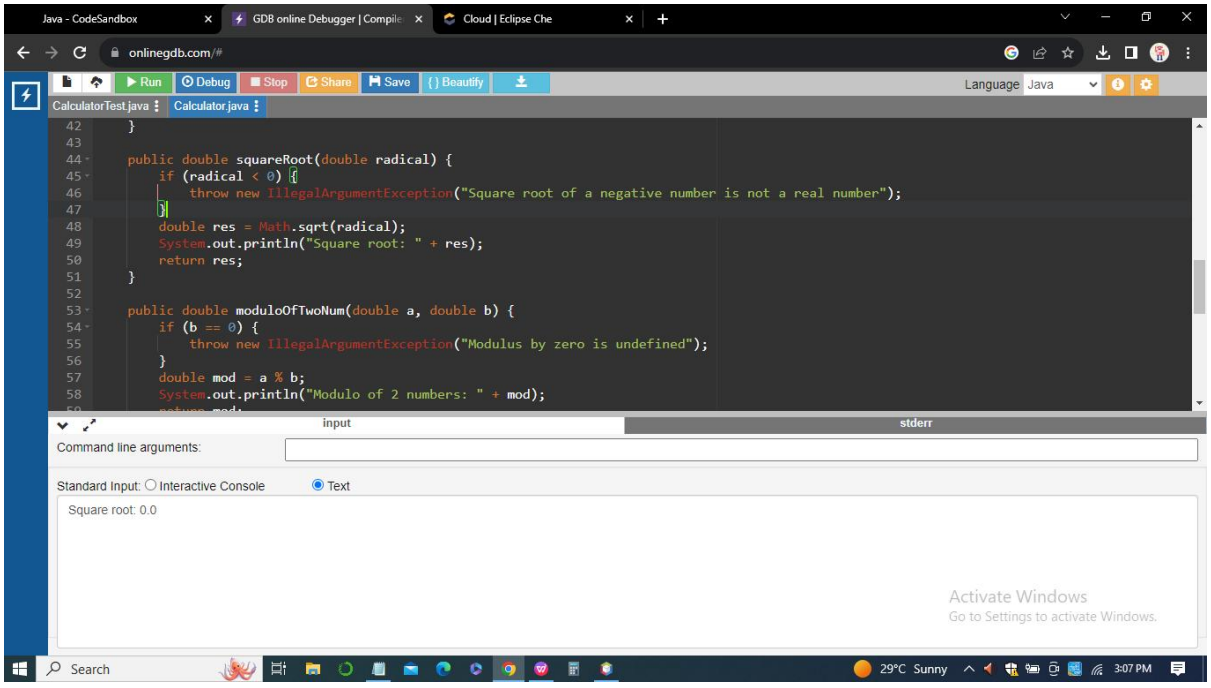
xvii. Test case 17: Test with a negative number.

testSquareRootOfNegativeNumber: Input is radical = -16.0. Expected output: An IllegalArgumentException is expected to be thrown, as the square root of a negative number is not a real number.



xviii. Test case 18: Test with zero.

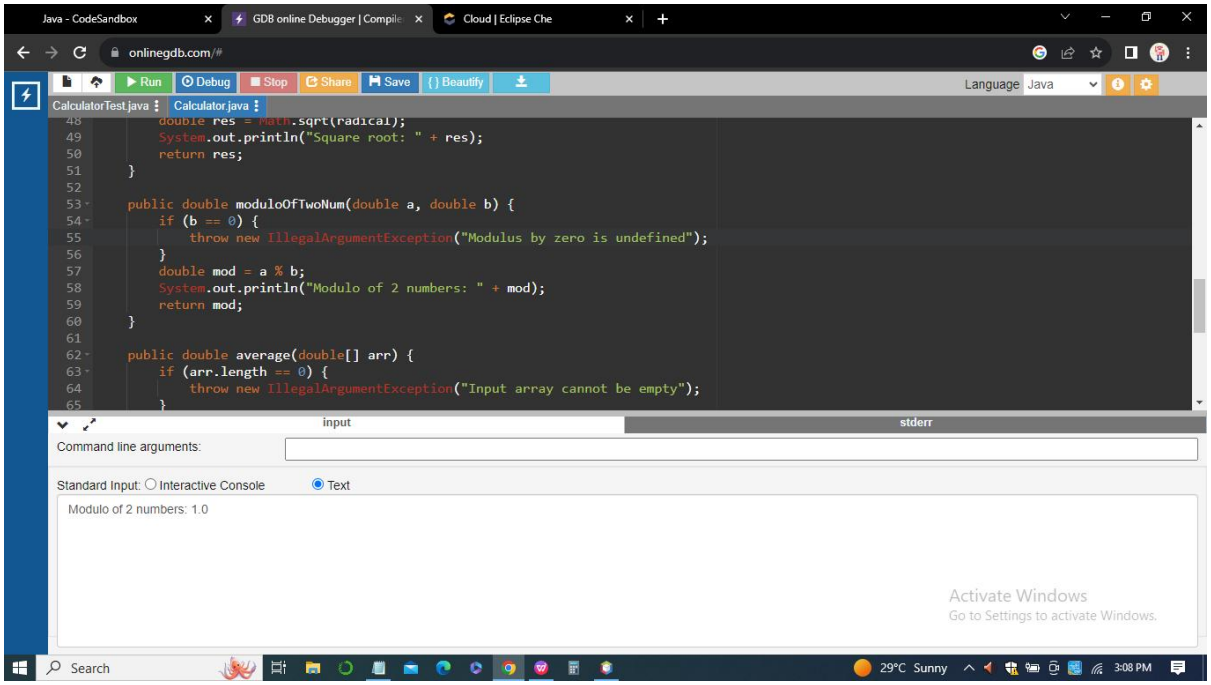
testSquareRootOfZero: Input is radical = 0.0. Expected output: 0.0, as the square root of 0.0 is 0.0.



moduloOfTwoNum method

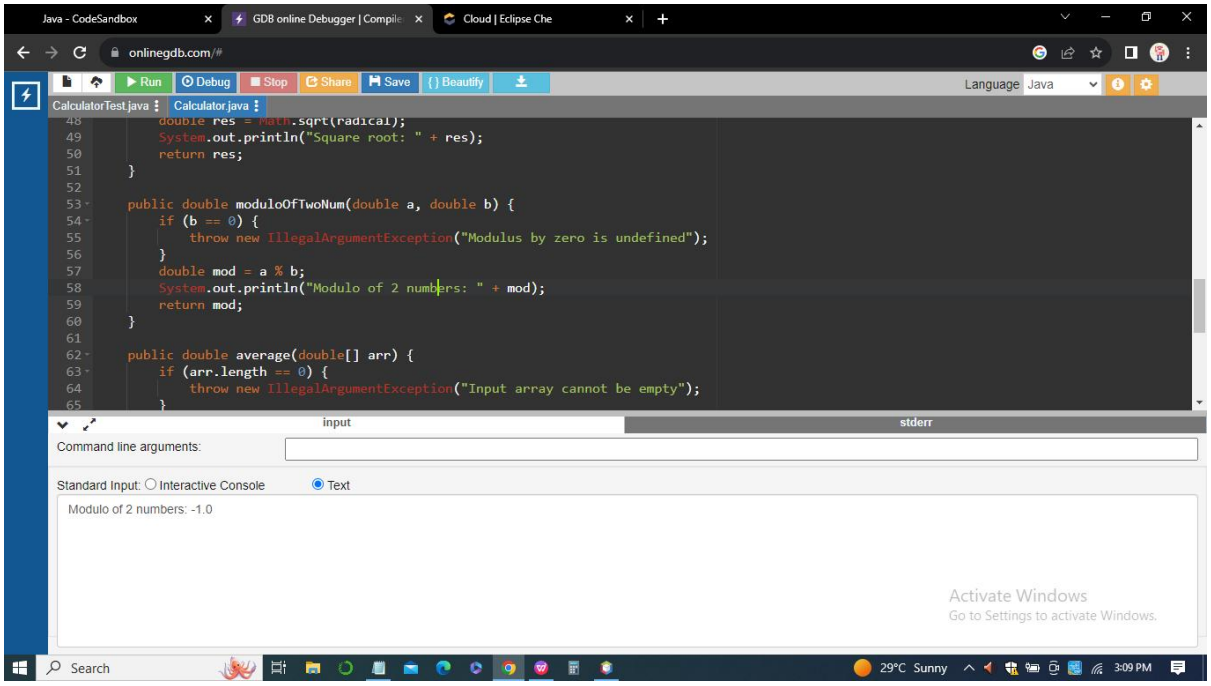
xix. Test case 19: Test with positive values for a and b.

testModuloOfTwoNumWithPositiveValues: Input is a = 10.0 and b = 3.0. Expected output: 1.0, which is the result of a % b.



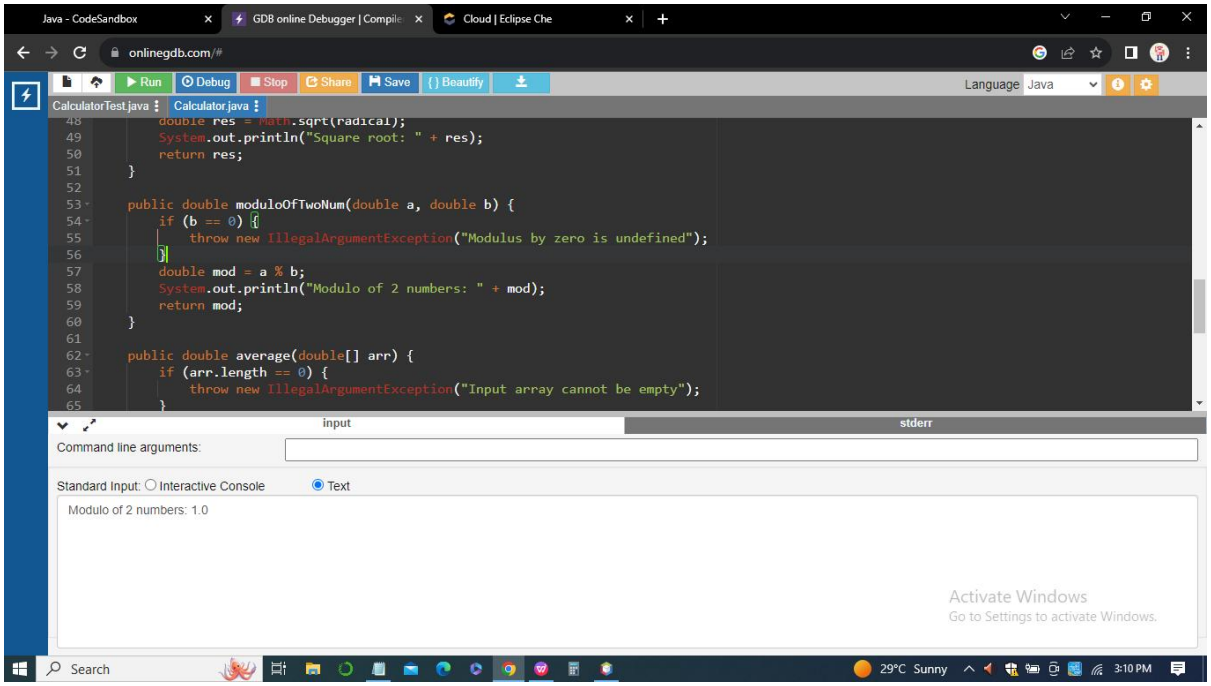
xx. Test case 20: Test with a negative value for a and a positive value for b.

testModuloOfTwoNumWithNegativeValueA: Input is a = -10.0 and b = 3.0. Expected output: -1.0, which is the result of a % b with a negative a.



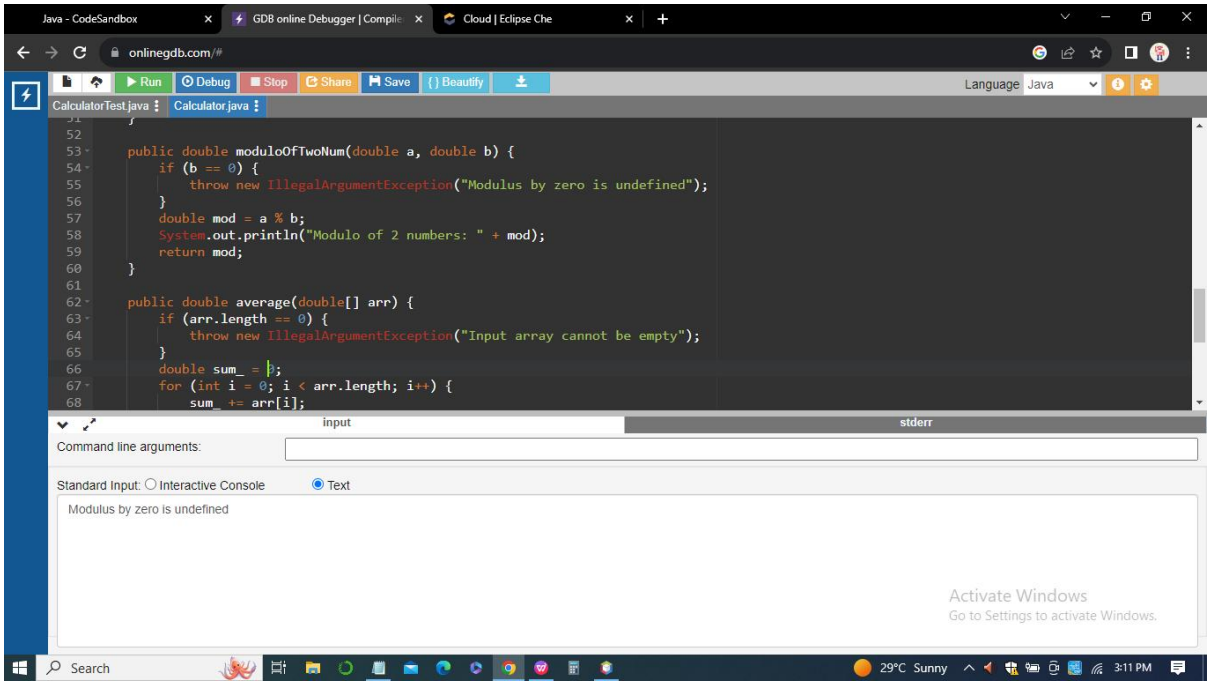
xxi. Test case 21: Test with a positive value for a and a negative value for b.

testModuloOfTwoNumWithNegativeValueB: Input is a = 10.0 and b = -3.0. Expected output: 1.0, which is the result of a % b with a negative b.



xxii. Test case 22: Test with zero as b.

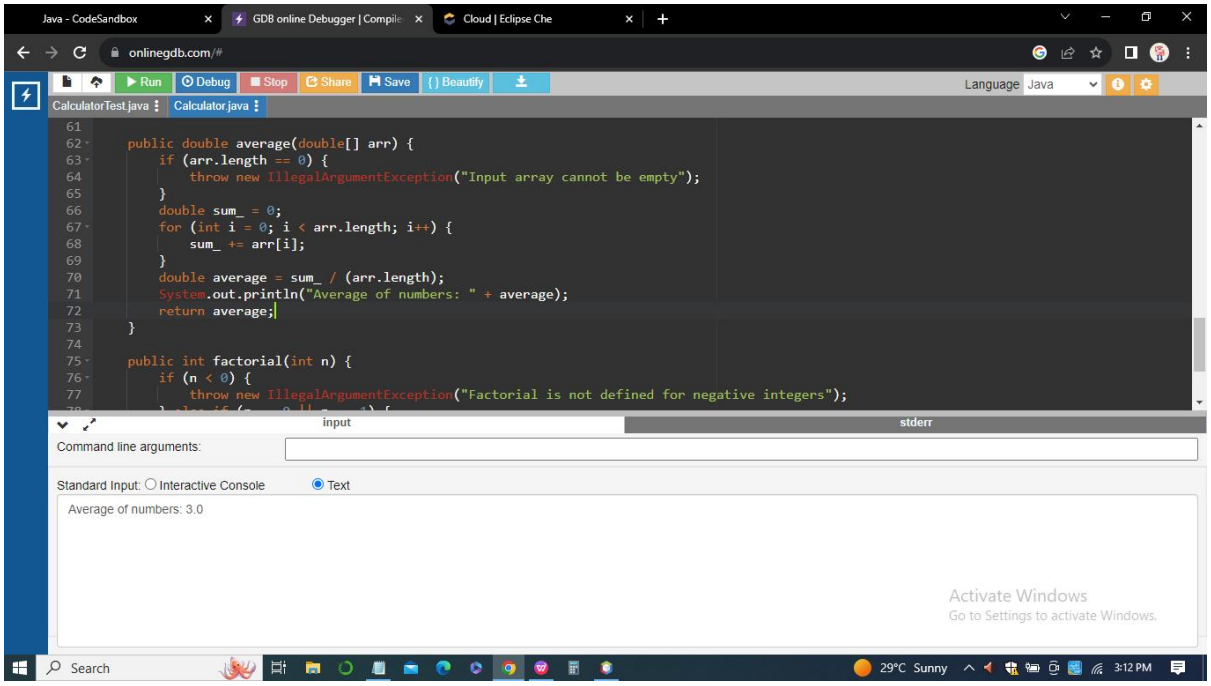
testModuloOfTwoNumWithZeroB: Input is a = 10.0 and b = 0.0. Expected output: An IllegalArgumentException is expected to be thrown, as modulus by zero is undefined.



Average method

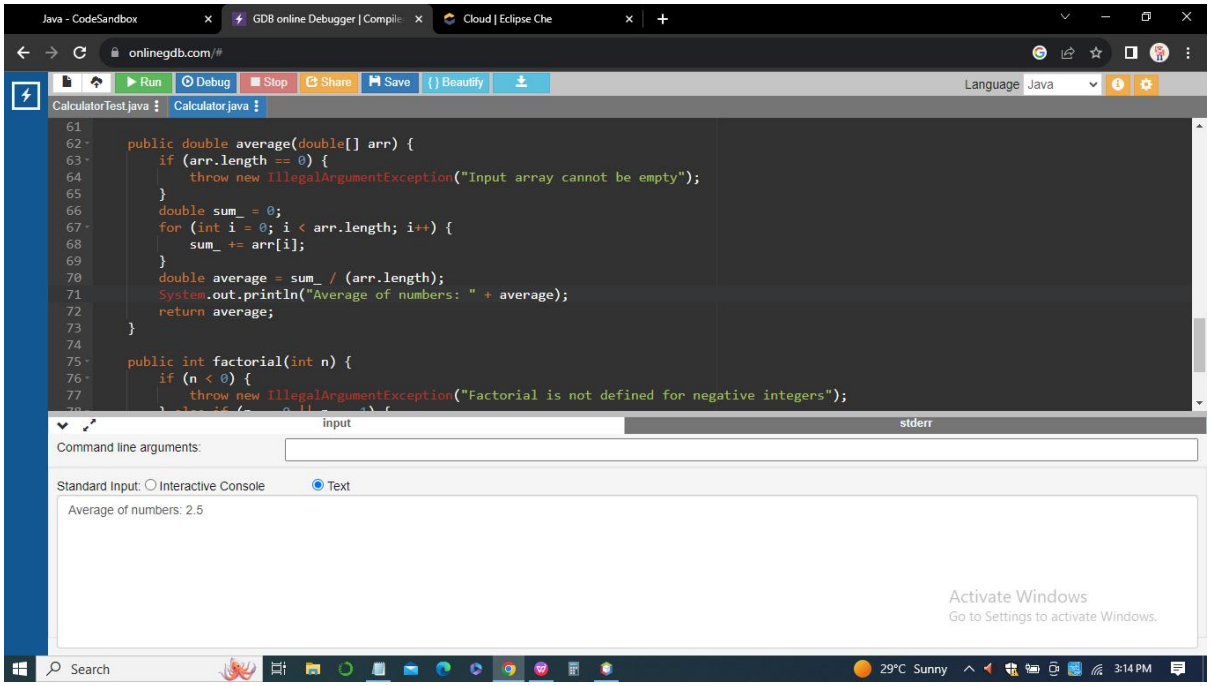
xxiii. Test case 23: Test with an array of positive numbers.

testAverageWithPositiveNumbers: Input is an array with elements [1.0, 2.0, 3.0, 4.0, 5.0]. Expected output: 3.0, which is the average of the elements.



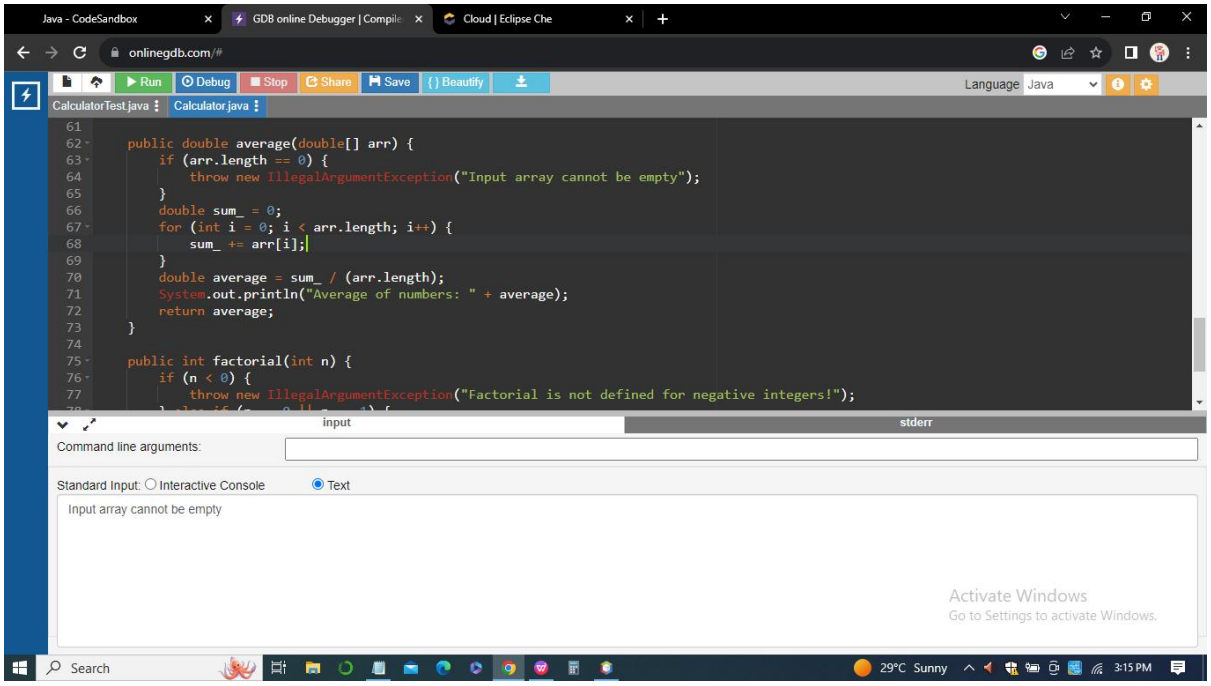
xxiv. Test case 24: Test with an array containing a mix of positive and negative numbers.

testAverageMixPositiveNumbers: Input is an array with elements [1.0, -2.0, -3.0, 4.0, 5.0]. Expected output: 2.5, which is the average of the elements.



xxv. Test case 25: Test with an empty array.

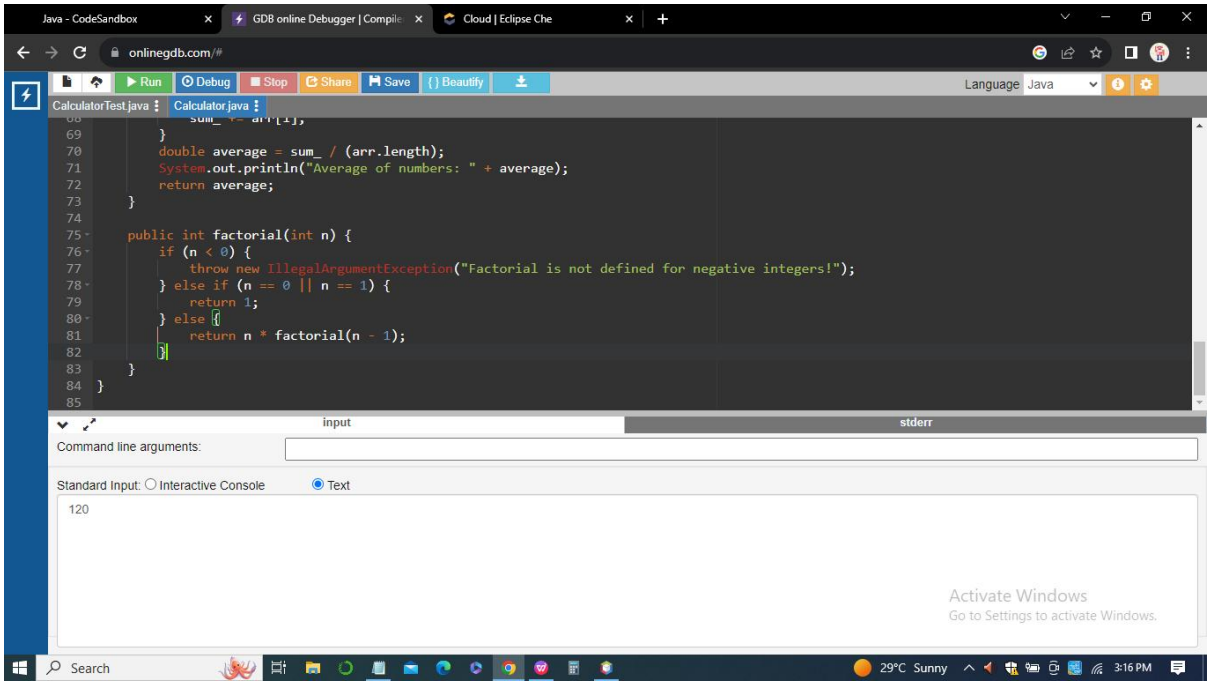
testAverageWithEmptyArray: Input is an empty array. Expected output: An IllegalArgumentException is expected to be thrown, as computing the average of an empty array is not possible.



factorial method

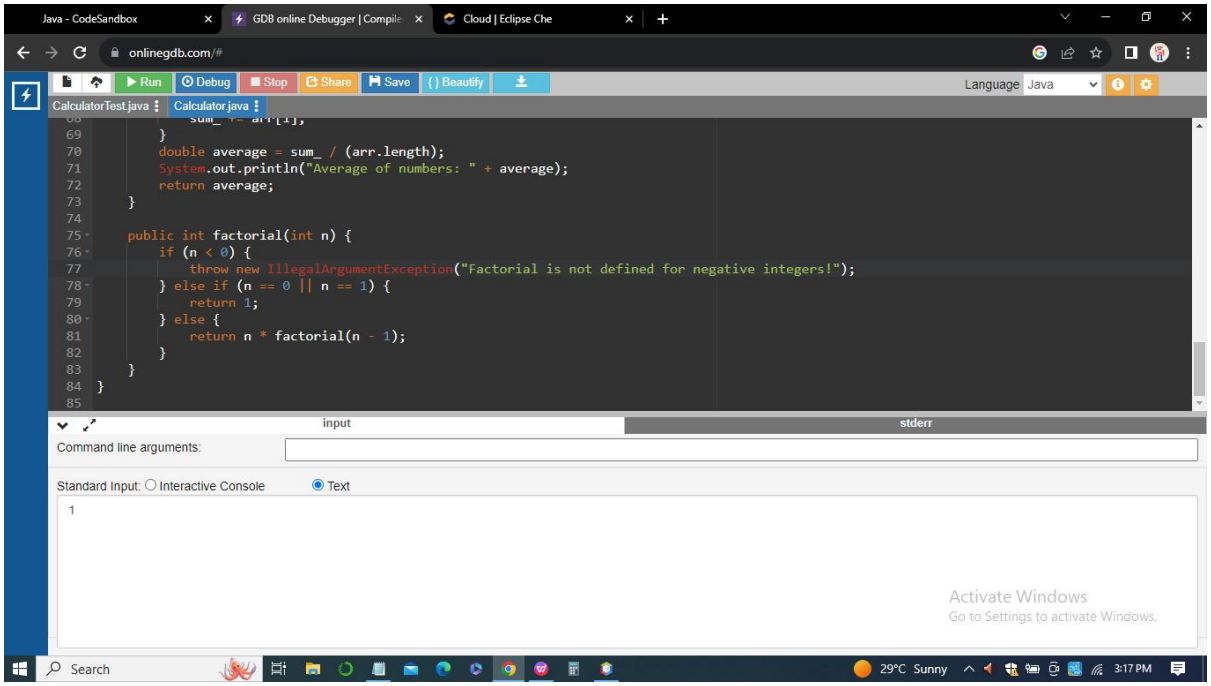
xxvi. Test case 26: Test with a positive integer.

testFactorialWithPositiveInteger: Input is n = 5. Expected output: 120, which is the factorial of 5.



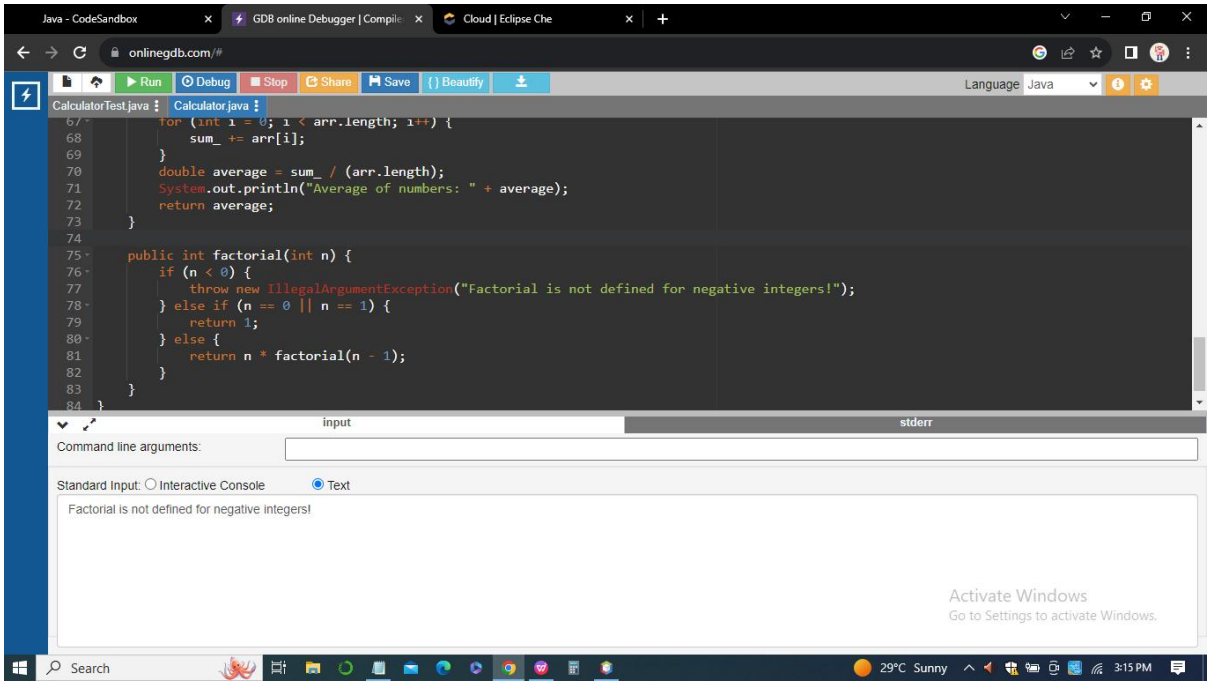
xxvii. Test case 27: Test with zero.

testFactorialWithZero: Input is n = 0. Expected output: 1, as the factorial of 0 is defined as 1.



xxviii. Test case 28: Test with a negative integer.

testFactorialWithNegativeInteger: Input is n = -5. Expected output: An IllegalArgumentException is expected to be thrown, as the factorial is not defined for negative integers.



Defects in the provided 2 source files

Defect 1: subtract method in Calculator.java

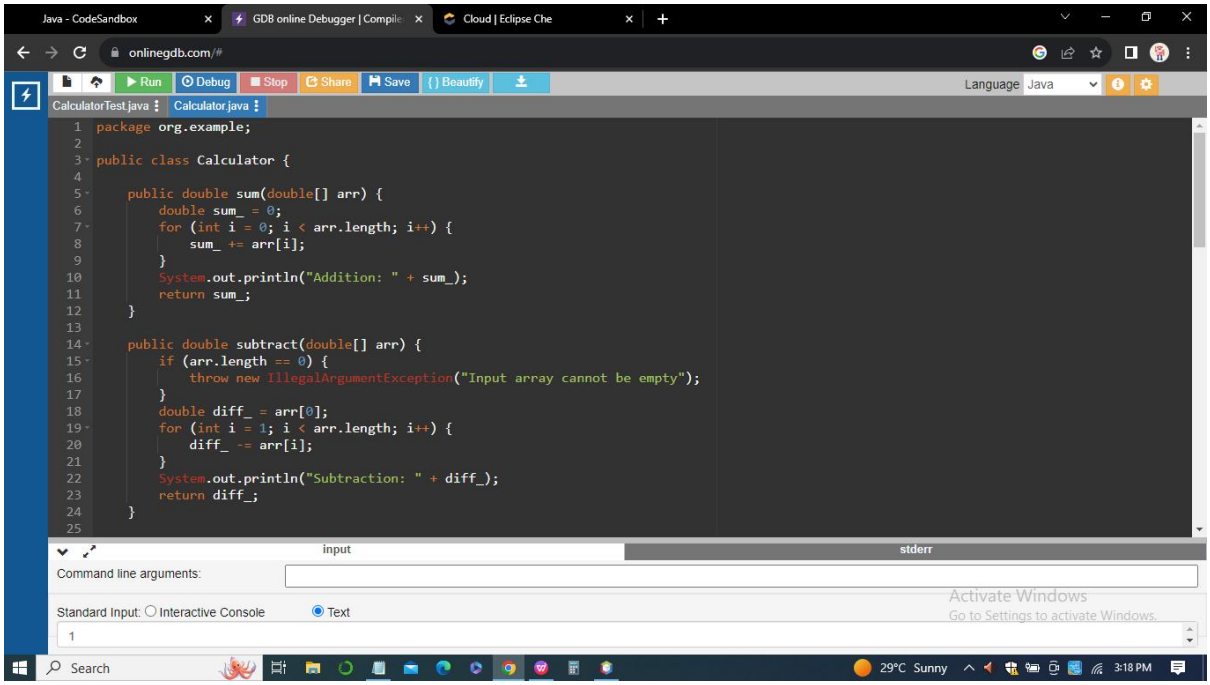
Defect

The subtract method in the Calculator class initializes diff_ with the first element of the input array, assuming that the array is not empty. This can lead to an ArrayIndexOutOfBoundsException if the input array is empty.

Fix

Add a check at the beginning of the subtract method to handle the case when the input array is empty. You can return a meaningful default value (e.g., 0) or throw an exception to indicate that subtraction is not possible.

```
public double subtract(double[] arr) {
    if (arr.length == 0) {
        throw new IllegalArgumentException("Input array cannot be empty");
    }
    double diff_ = arr[0];
    for (int i = 1; i < arr.length; i++) {
        diff_ -= arr[i];
    }
    System.out.println("Subtraction: " + diff_);
    return diff_;
}
```



Defect 2: divide method in Calculator.java

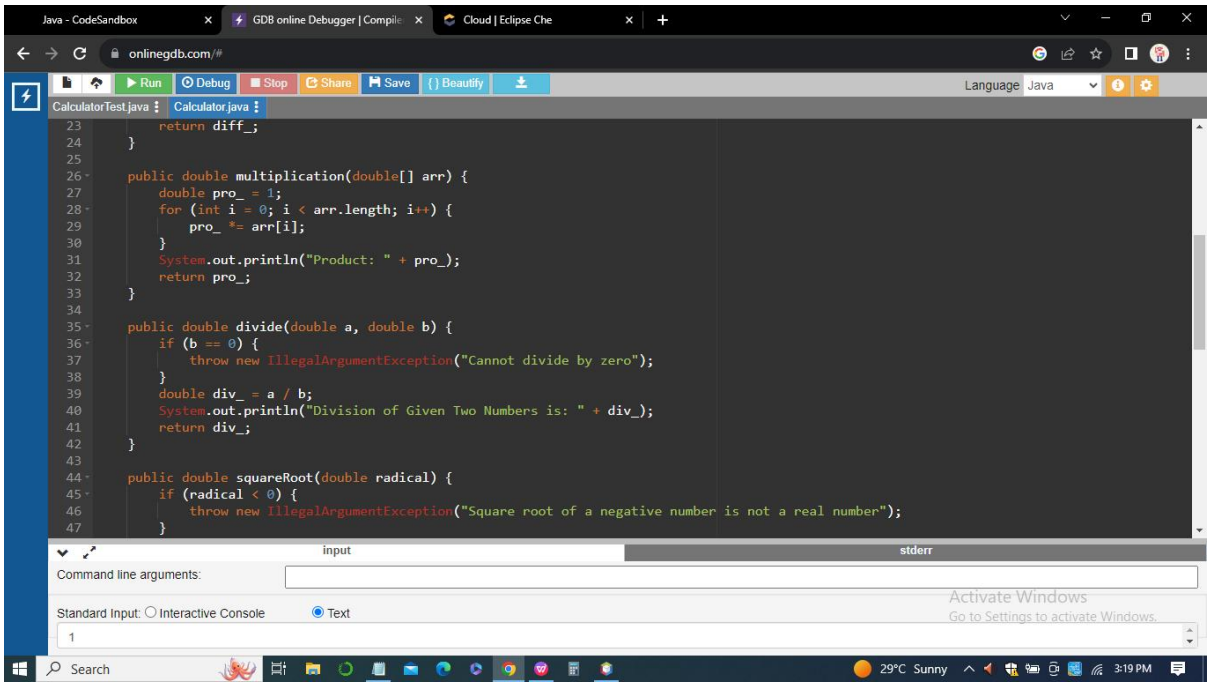
Defect

The divide method does not properly handle the case when the denominator is zero. It returns Double.MIN_VALUE, which is not a suitable indicator of division by zero.

Fix

Add a check at the beginning of the divide method to handle the case when the denominator is zero. You can throw an exception or return a specific value (e.g., Double.NaN) to indicate division by zero.

```
public double divide(double a, double b) {
    if (b == 0) {
        throw new IllegalArgumentException("Cannot divide by zero");
    }
    double div_ = a / b;
    System.out.println("Division of Given Two Numbers is: " + div_);
    return div_;
}
```



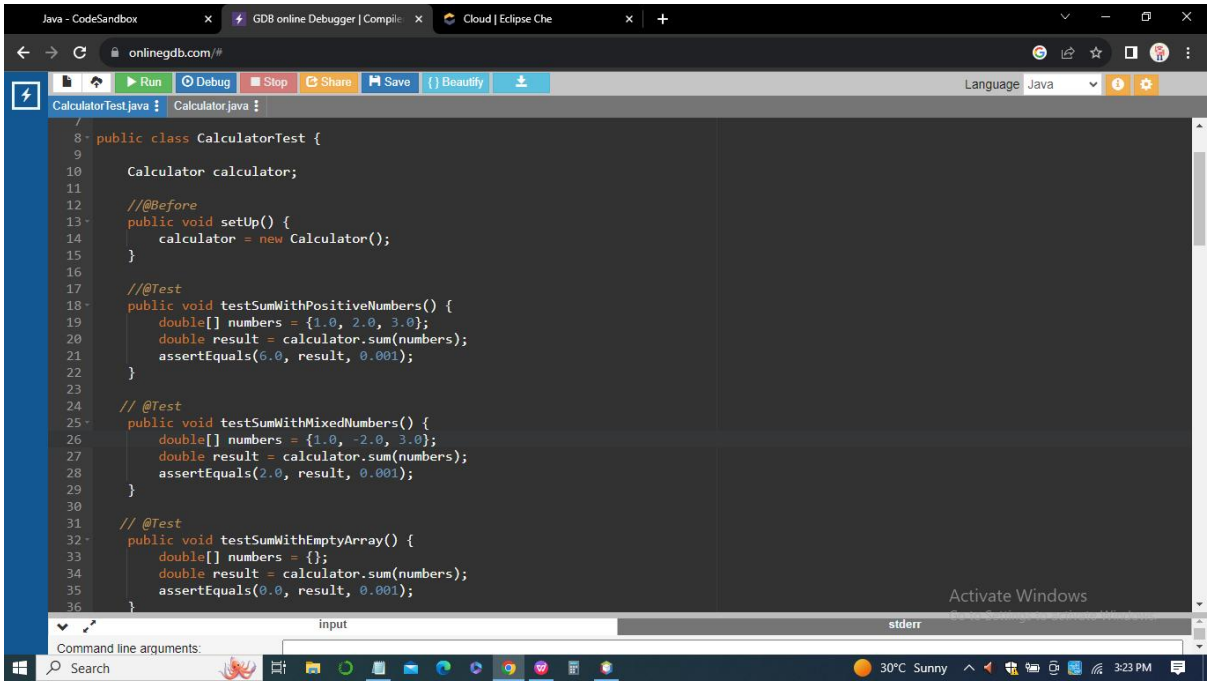
Defect 3: Lack of Unit Tests in CalculatorTest.java

Defect

The provided CalculatorTest.java file only contains one test method (divide), and it does not cover all the methods in the Calculator class.

Fix

More test methods were added to cover all the methods in the Calculator class e.g testSumWithPositiveNumbers(),testSumWithMixedNumbers(), testSumWithEmptyArray() etc . Test cases were also written, that check various scenarios, including edge cases and error handling.



Conclusion

In conclusion, this assignment has successfully addressed defects in the Calculator project source code and significantly improved its reliability. By introducing proper error handling and comprehensive unit tests, we have enhanced the overall quality of the codebase. The unit tests provide confidence in the correctness and robustness of the Calculator application, ensuring it functions as expected and handles various scenarios gracefully. This exercise reinforces the importance of testing and defect resolution in software development,

contributing to more reliable and maintainable code. Moving forward, it is essential to continue practicing thorough testing and code review processes to maintain and improve software quality.