

:INCUBYTE TDD ASSESSMENT:

-:String Calculator:-

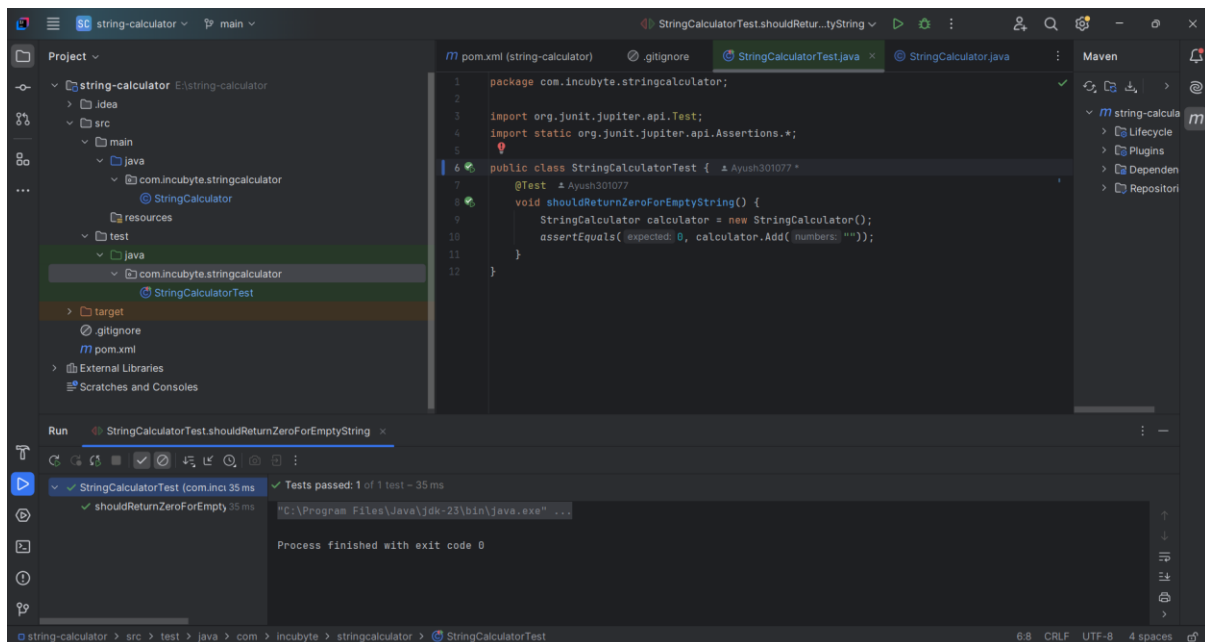
Name: Soni Ayush Vimalkumar

Branch: Information Technology

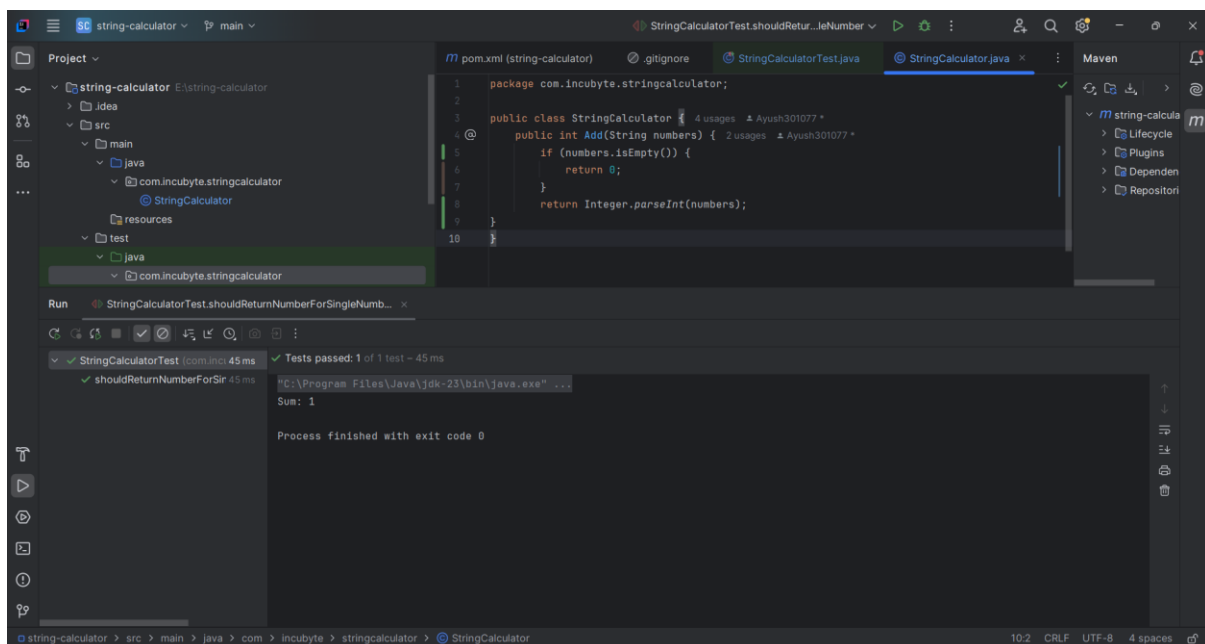
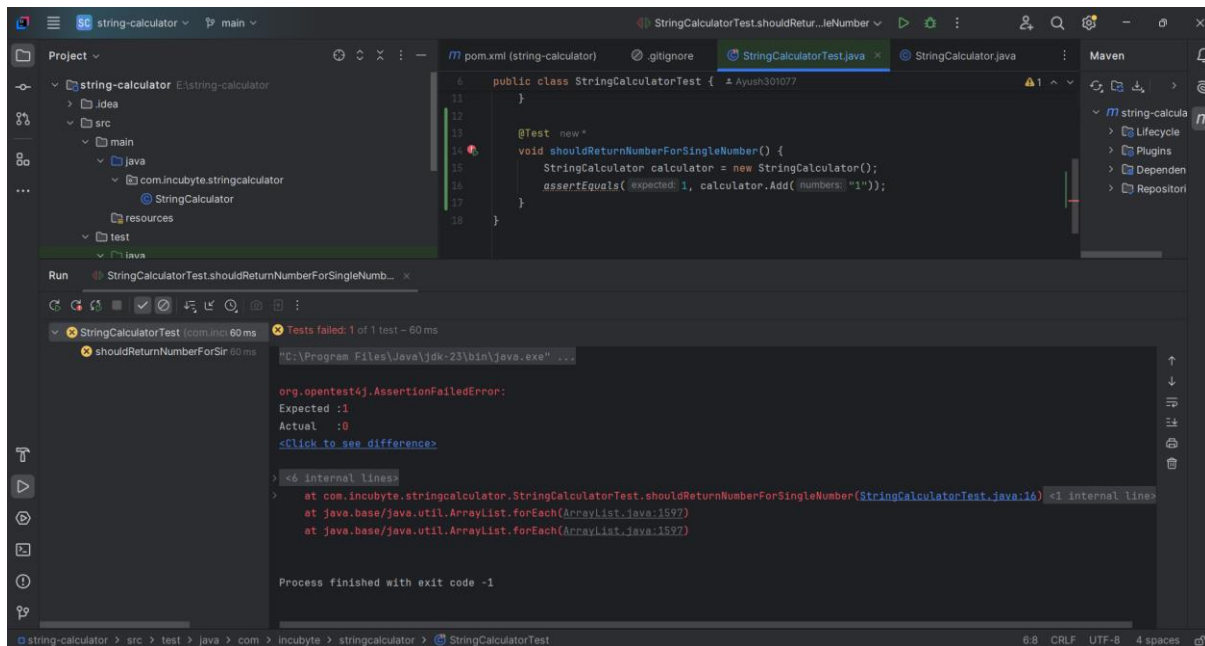
Email: ayushvsoni493@gmail.com

Step 1: Test for Empty String Input:

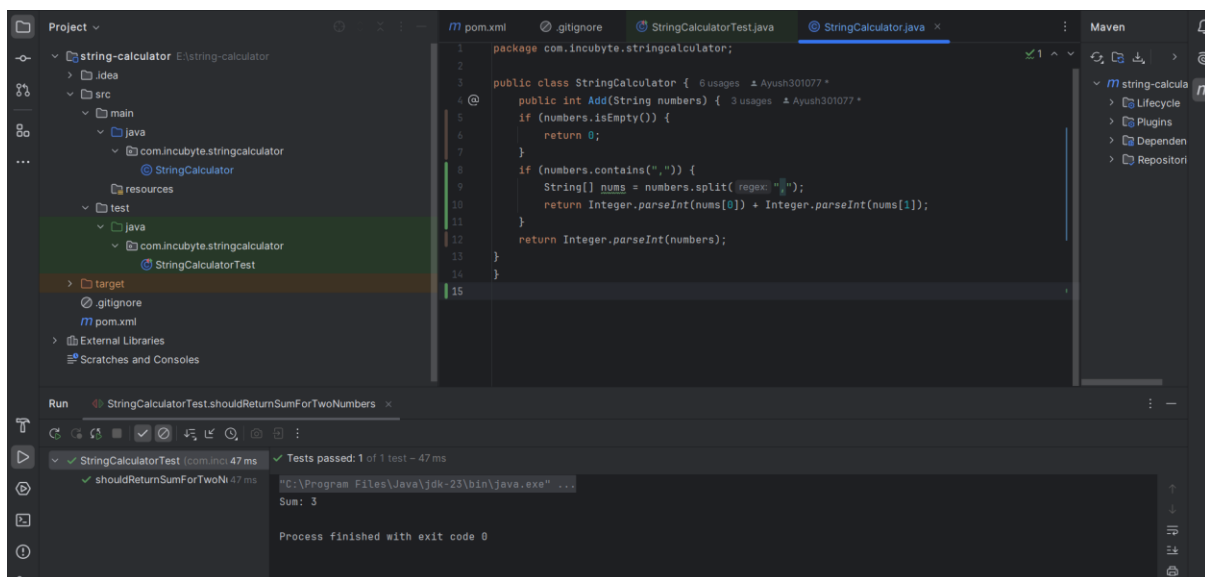
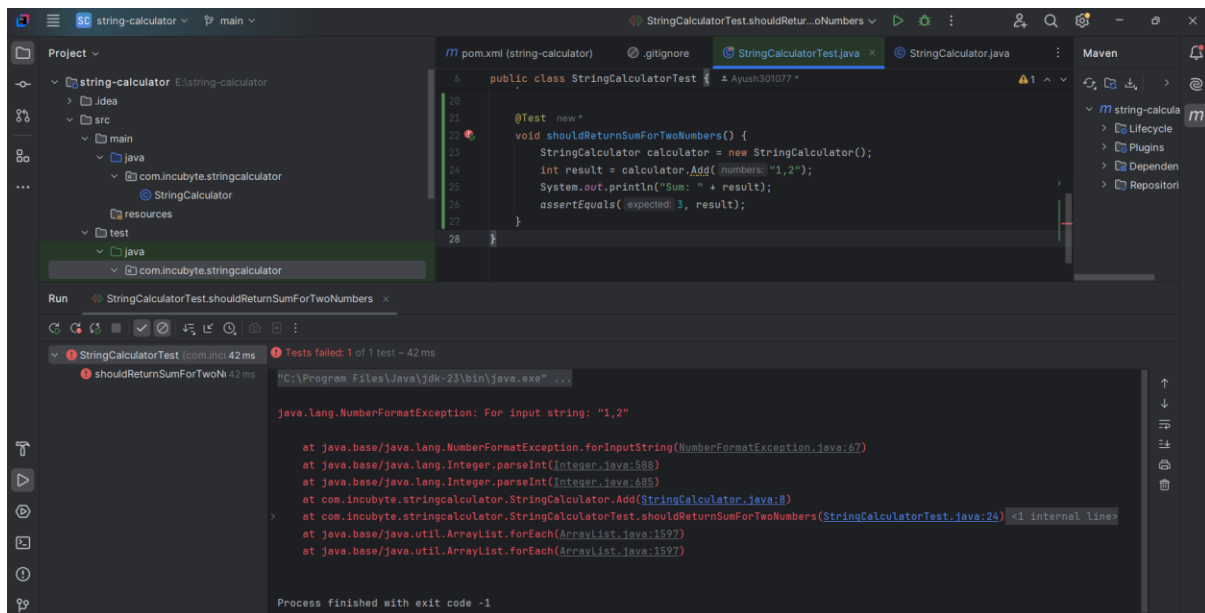
This screenshot shows the result of running the first unit test for the StringCalculator. The test checks that calling Add("") returns 0, as required by the kata. The green checkmark and "Tests passed" message confirm that the implementation is correct for this case.



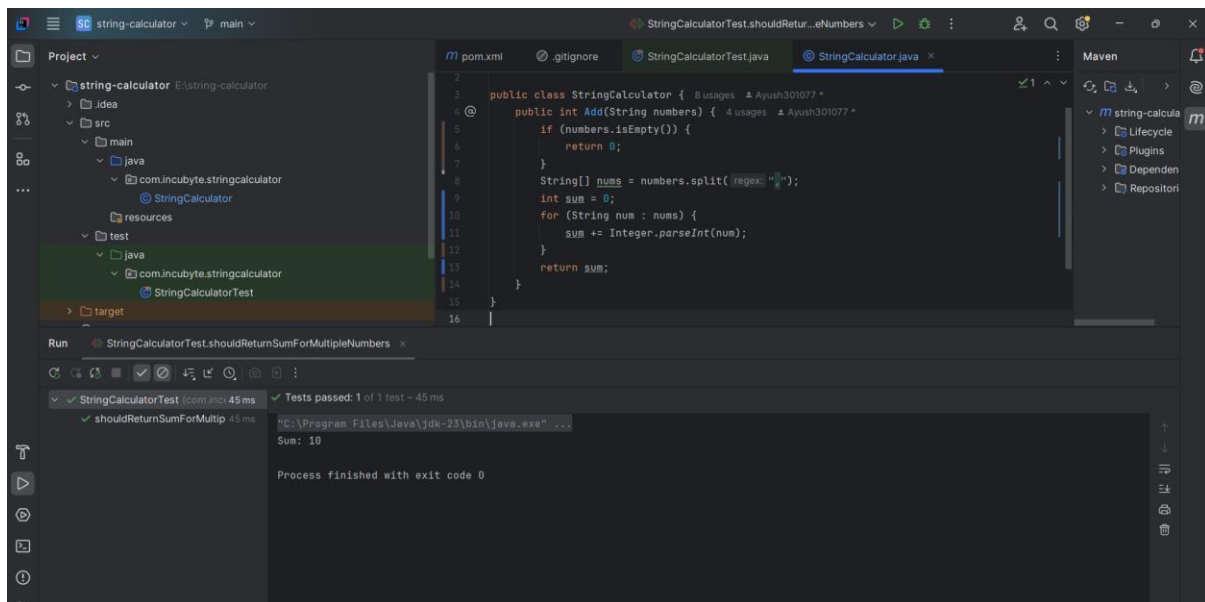
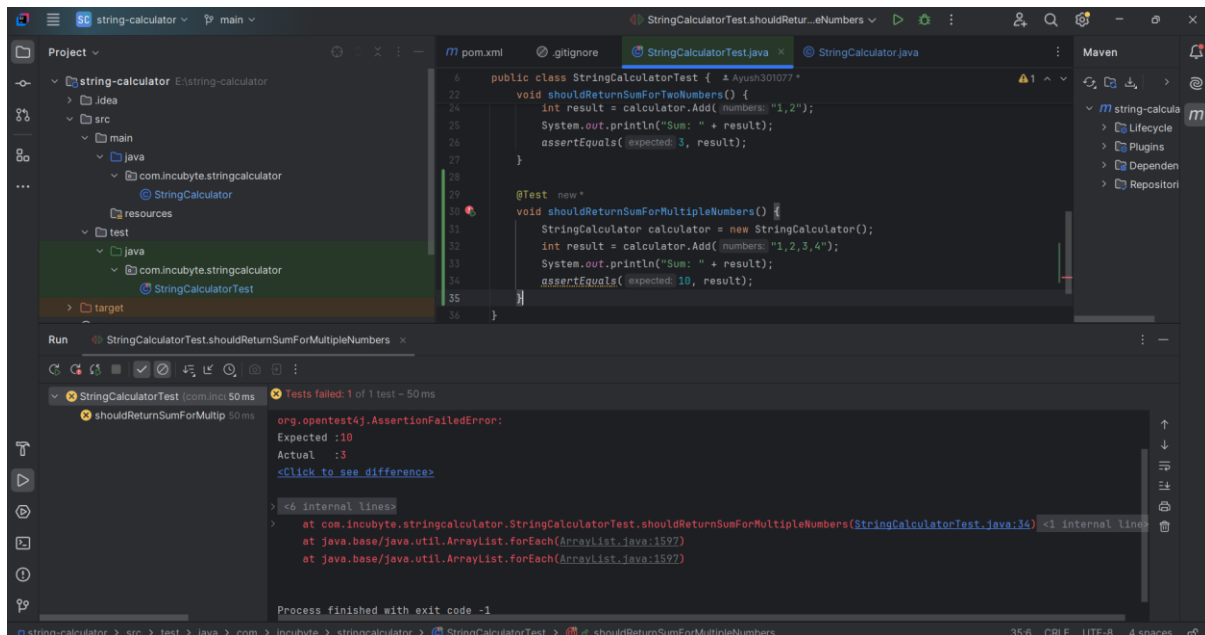
Step 2: Test for Single Number Input This screenshot shows the result of running the second unit test for the StringCalculator. The test checks that calling Add("1") returns 1, as required by the kata. The green checkmark and "Sum: 1" message confirm that the implementation is correct for this case.



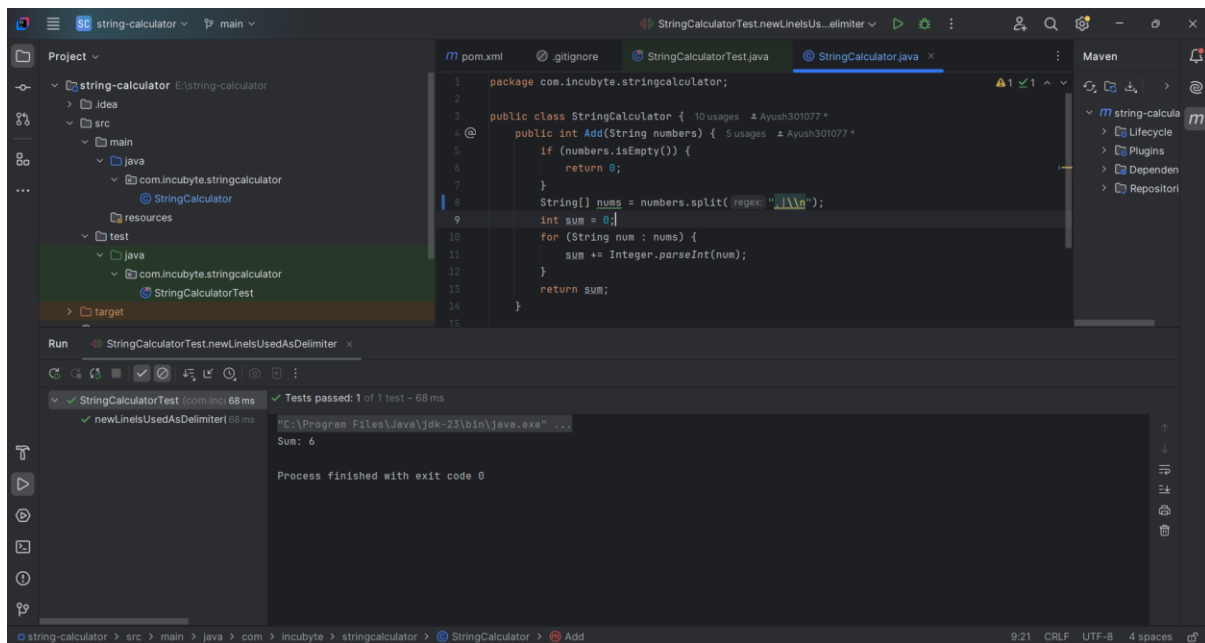
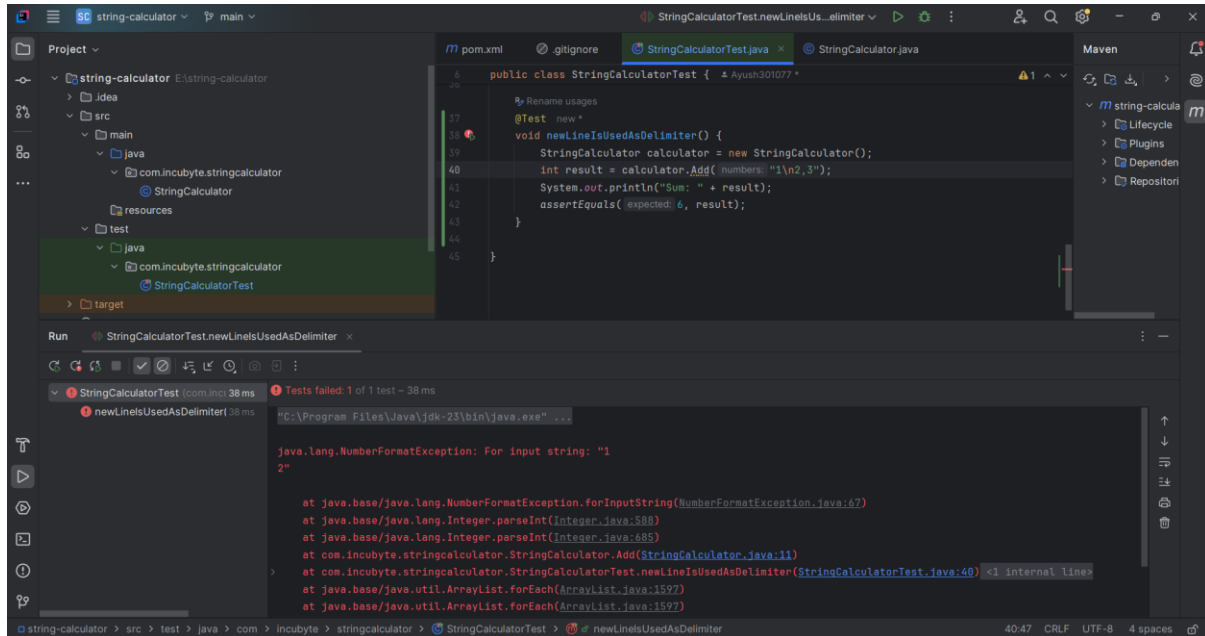
Step 3: Test for Two Numbers Input This screenshot shows the result of running the unit test for the case where the input string contains two numbers, such as "1,2". The test checks that calling `Add("1,2")` returns 3, as required by the kata. The green checkmark and "Tests passed" message confirm that the implementation is correct for this case.



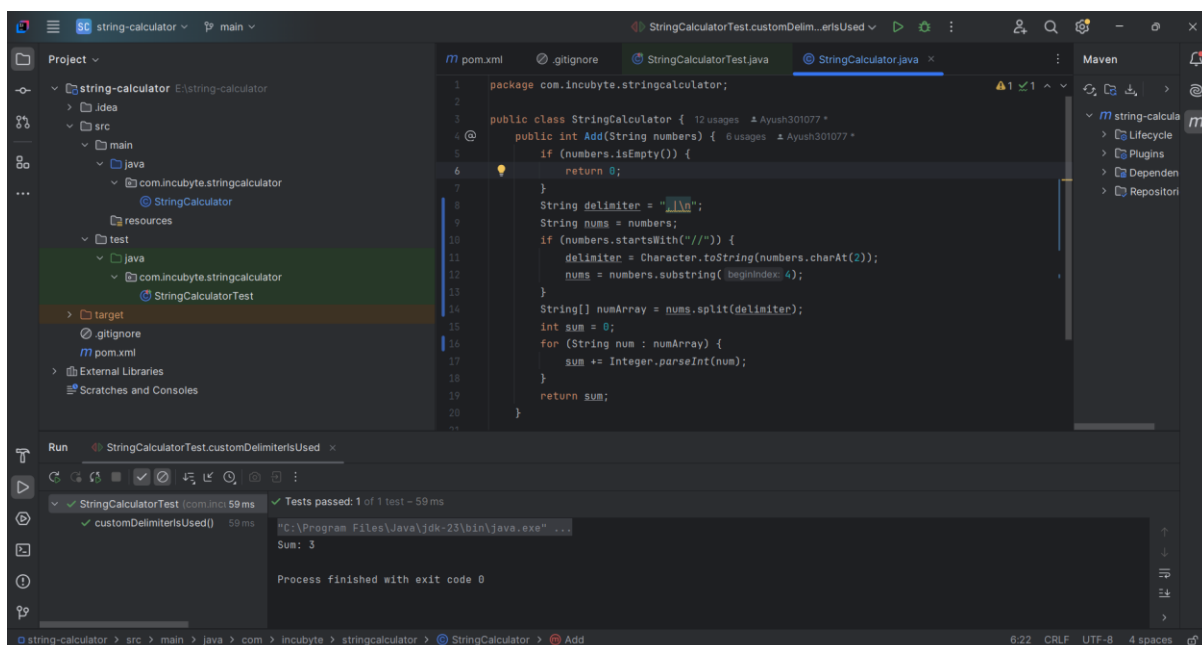
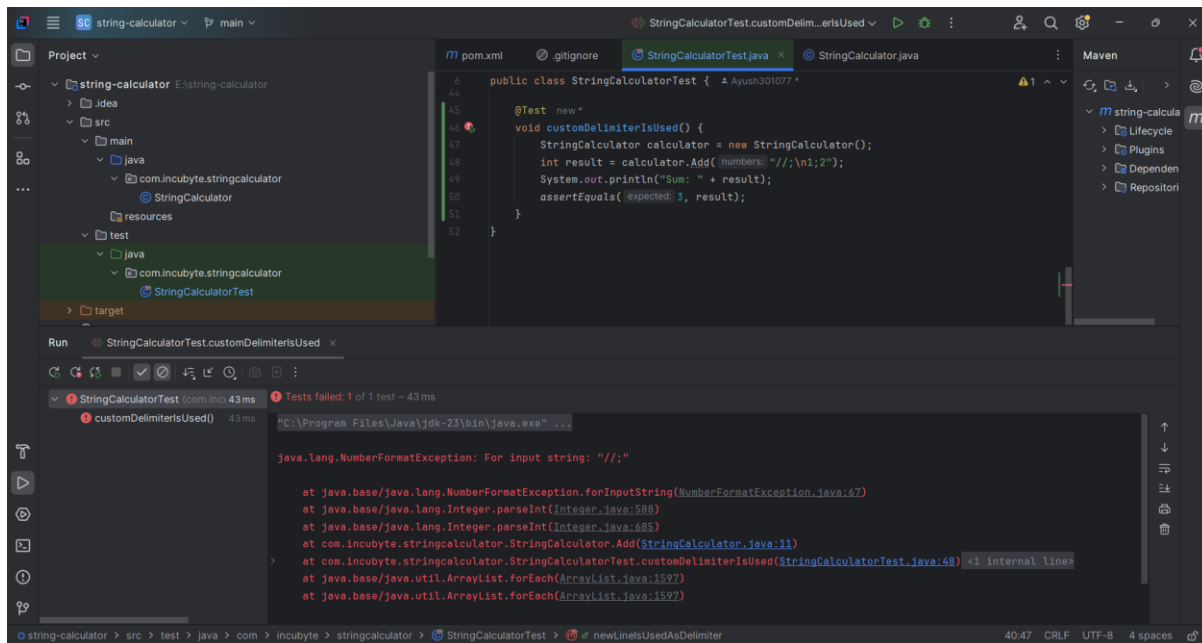
Step 4: Test for Multiple Numbers Input This step verifies that the StringCalculator can handle and correctly sum an unknown number of comma-separated numbers, such as "1,2,3,4". The test checks that calling Add("1,2,3,4") returns 10, as required by the kata. The green checkmark and "Sum: 10" message confirms that the implementation is correct for this case.



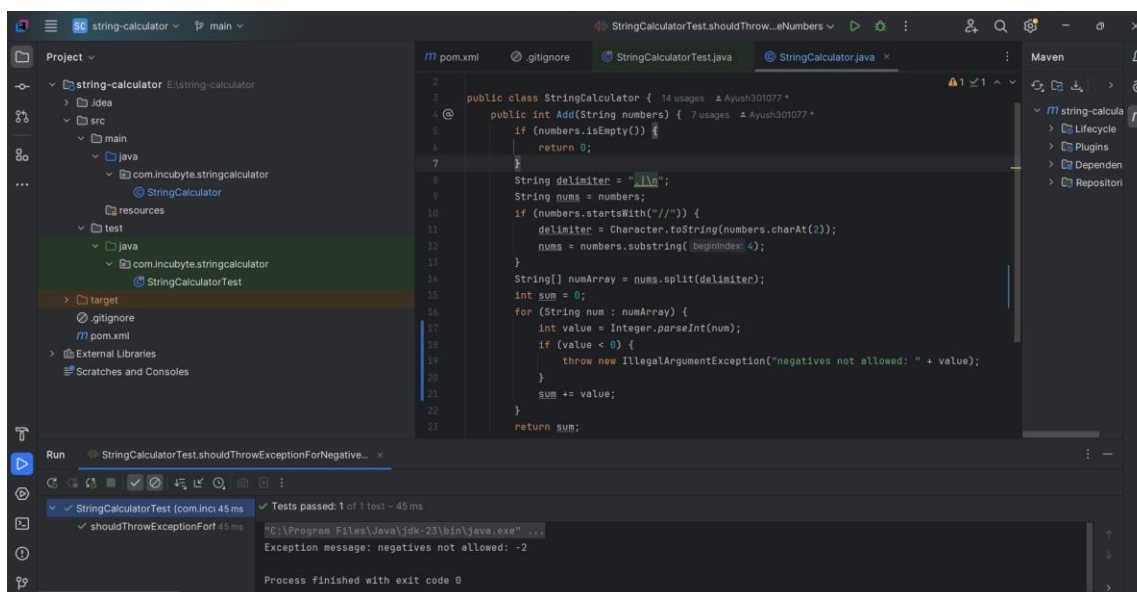
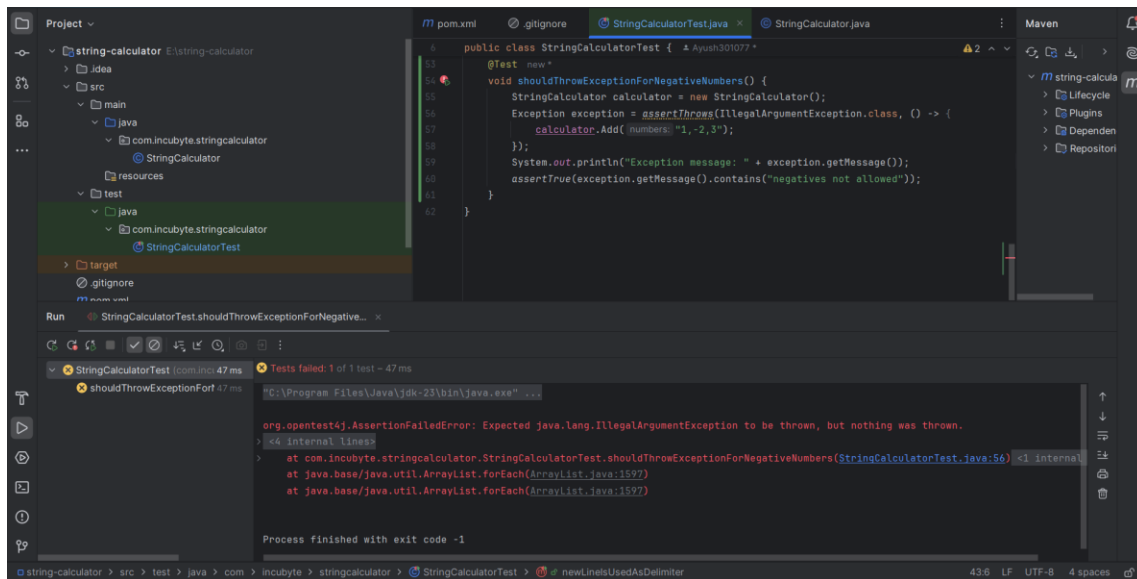
Step 5: Test for New Line as Delimiter This step verifies that the StringCalculator can handle new lines as delimiters in addition to commas. The test checks that calling Add("1\n2,3") returns 6, as required by the kata. The green checkmark and "Sum: 6" message will confirm that the implementation is correct for this case.



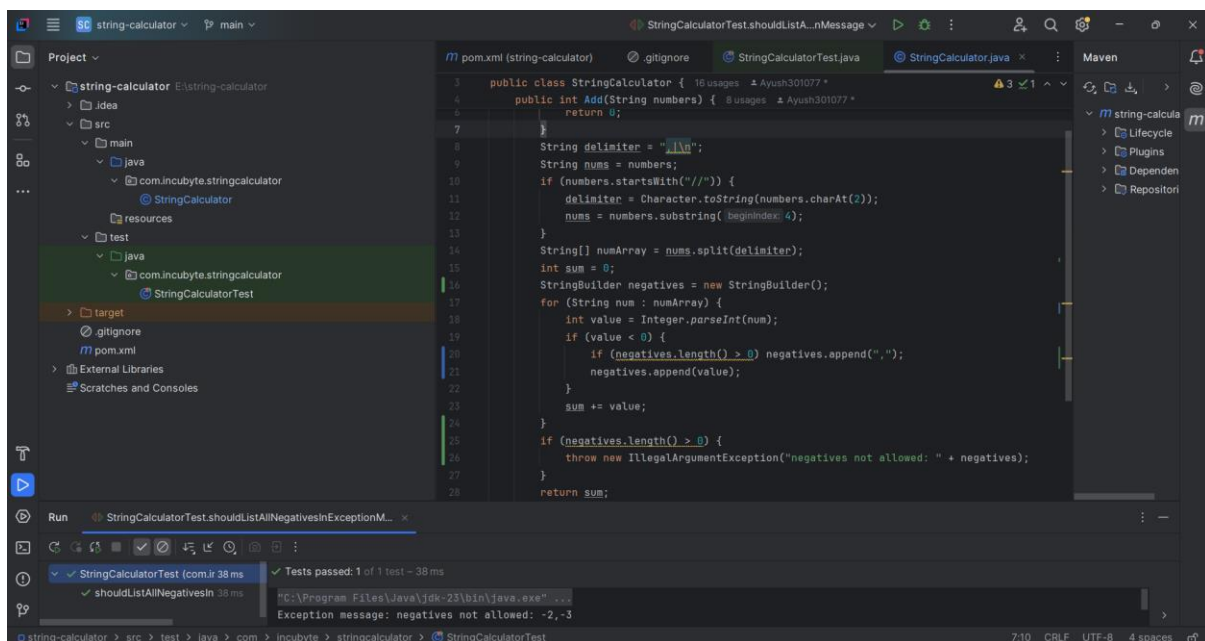
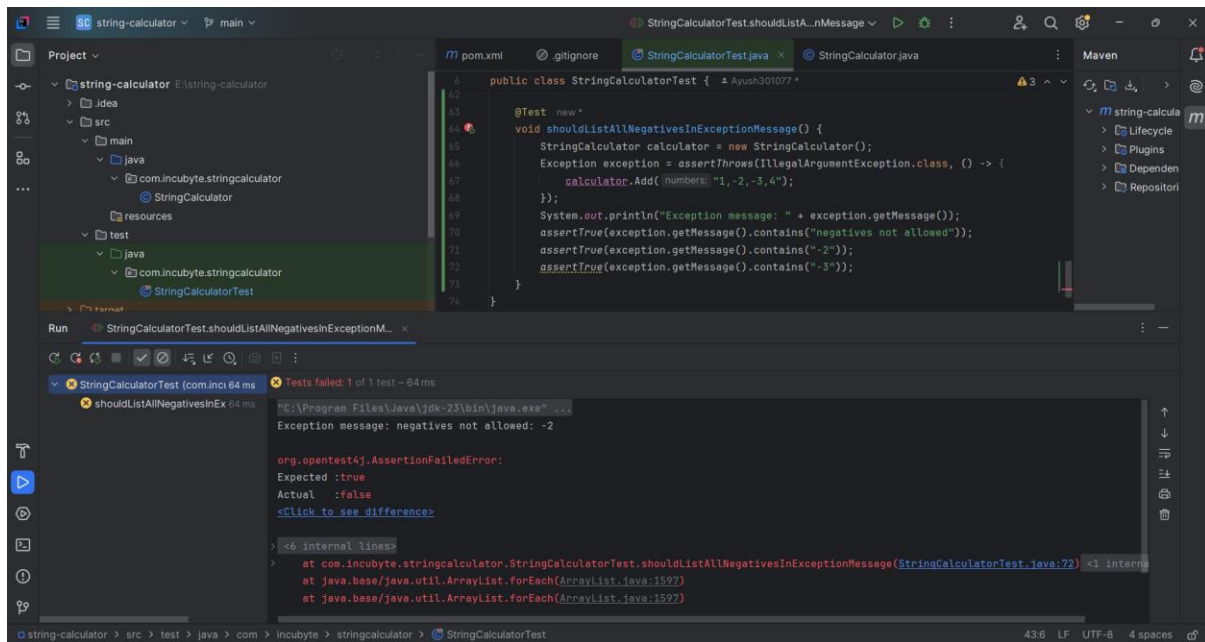
Step 6: Test for Custom Delimiter This step verifies that the StringCalculator can handle a custom single-character delimiter specified at the start of the input string (e.g., "///



Step 7: Test for Negative Numbers This step verifies that the StringCalculator throws an exception with the message "negatives not allowed" when a negative number is present in the input. The test checks that calling Add("1,-2,3") throws the correct exception, as required by the kata. The green checkmark and "Exception message: negatives not allowed: -2" message confirms that the implementation is correct for this case.



Step 8: Test for Multiple Negatives in Exception This step verifies that the StringCalculator throws an exception listing all negative numbers present in the input. The test checks that calling Add("1,-2,-3,4") throws an exception with a message containing both -2 and -3, as required by the kata. The green checkmark and "Exception message: negatives not allowed: -2,-3" message will confirm that the implementation is correct for this case.



The screenshot shows an IDE with the following components:

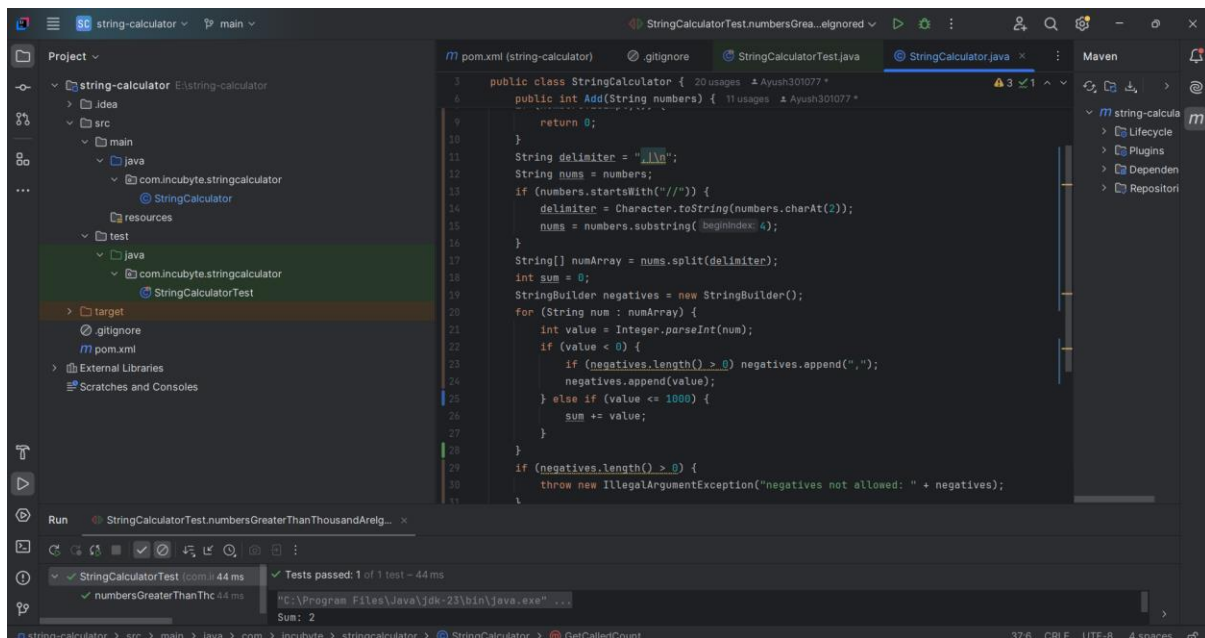
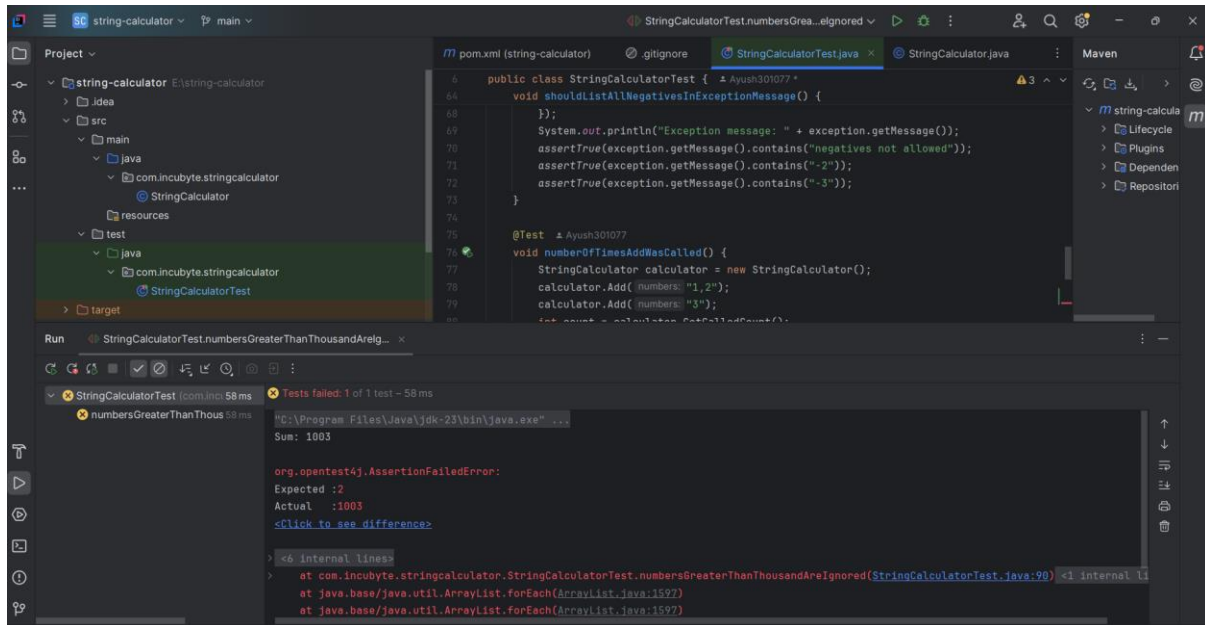
- Project Explorer (Left):** Displays the project structure for 'string-calculator'. It includes folders for 'idea', 'src', and 'main'. Under 'src', there is a 'main' folder containing 'StringCalculator.java'. Under 'test', there is a 'test' folder containing 'StringCalculatorTest.java'. The 'target' folder is also visible.
- Source Editor (Center):** Displays the source code of 'StringCalculatorTest.java'. The code includes:


```

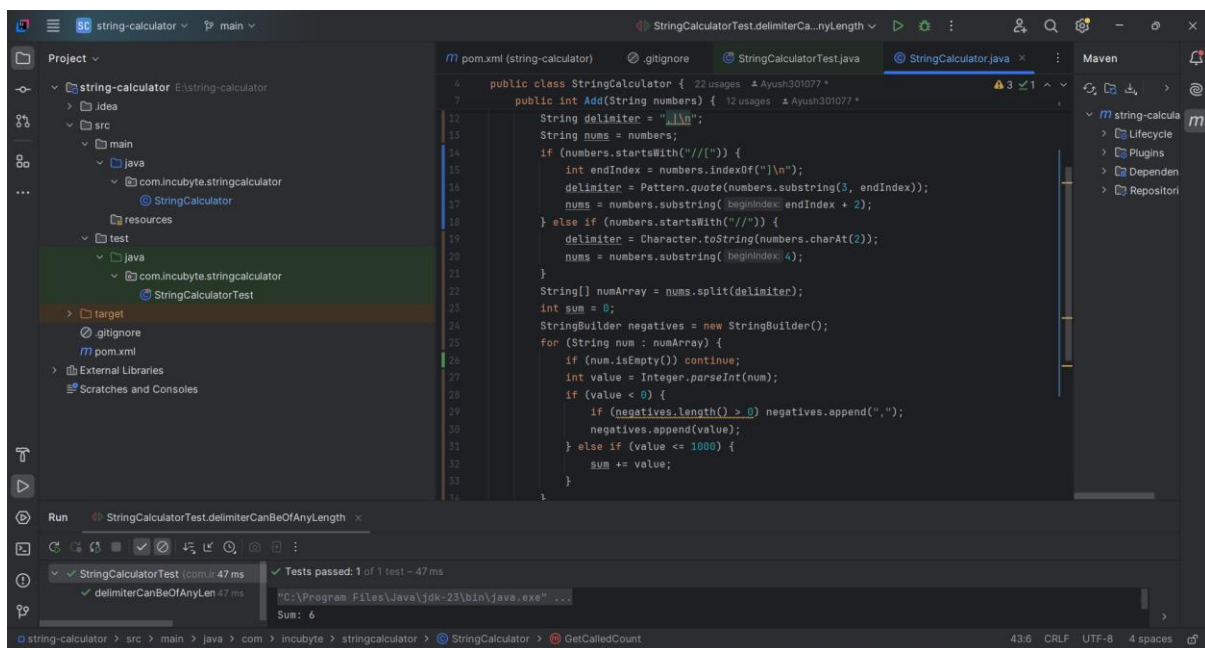
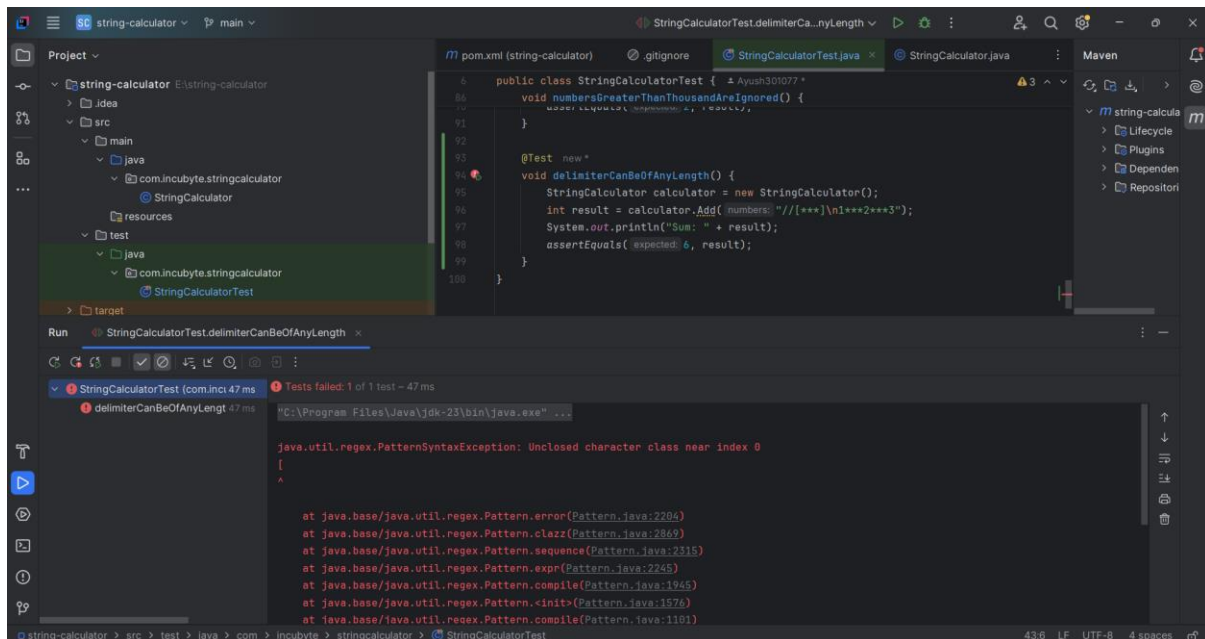
      37 @Test
      38 void newLineIsUsedAsDelimiter() {
      39     // ...
      64 public class StringCalculatorTest {
      65     // ...
      66     void shouldListAllNegativesInExceptionMessage() {
      67         // ...
      68         System.out.println("Exception message: " + exception.getMessage());
      69         assertTrue(exception.getMessage().contains("negatives not allowed"));
      70         assertTrue(exception.getMessage().contains("-2"));
      71         assertTrue(exception.getMessage().contains("-3"));
      72     }
      73 }
      74
      75 @Test
      76 void numberOfTimesAddWasCalled() {
      77     StringCalculator calculator = new StringCalculator();
      78     calculator.Add( numbers: "1,2");
      79     calculator.Add( numbers: "3");
      80     int count = calculator.GetCalledCount();
      81     System.out.println("Add called: " + count + " times");
      82     assertEquals( expected: 2, count);
      83 }
      84 }
      
```
- Run Bar (Bottom):** Shows the command 'Run StringCalculatorTest.numberOfTimesAddWasCalled'.

Made By: Soni Ayush Vimalkumar

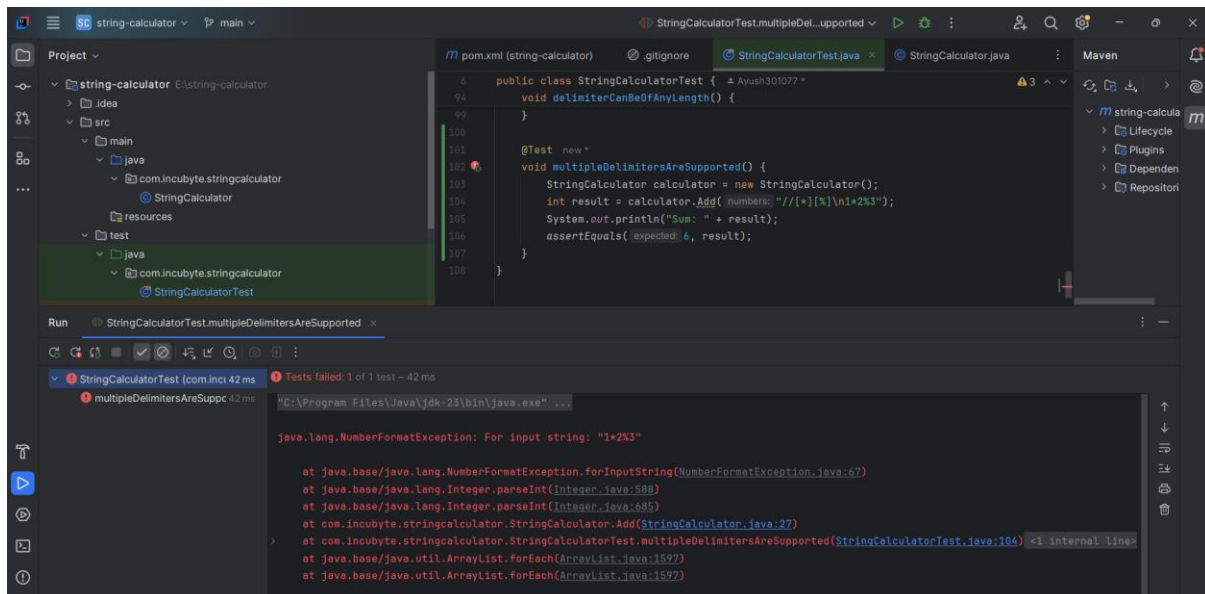
Step 10: Test for Ignoring Numbers Greater Than 1000 This step verifies that the StringCalculator ignores numbers greater than 1000 when calculating the sum. The test checks that calling Add("2,1001") returns 2, as required by the kata. The green checkmark and "Sum: 2" message will confirm that the implementation is correct for this case.



Step 11: Test for Delimiter of Any Length This step verifies that the StringCalculator can handle delimiters of any length specified in the format "[delimiter]\n". The test checks that calling Add("[***]\n1***2***3") returns 6, as required by the kata. The green checkmark and "Sum: 6" message will confirm that the implementation is correct for this case.



Step 12: Test for Multiple Delimiters This step verifies that the StringCalculator can handle multiple delimiters of any length specified in the format "[delim1][delim2]\n". The test checks that calling Add("[*][%]\n1*2%3") returns 6, as required by the kata. The green checkmark and "Sum: 6" message will confirm that the implementation is correct for this case.



```

public class StringCalculatorTest {
    void delimiterCanBeOfAnyLength() {
    }

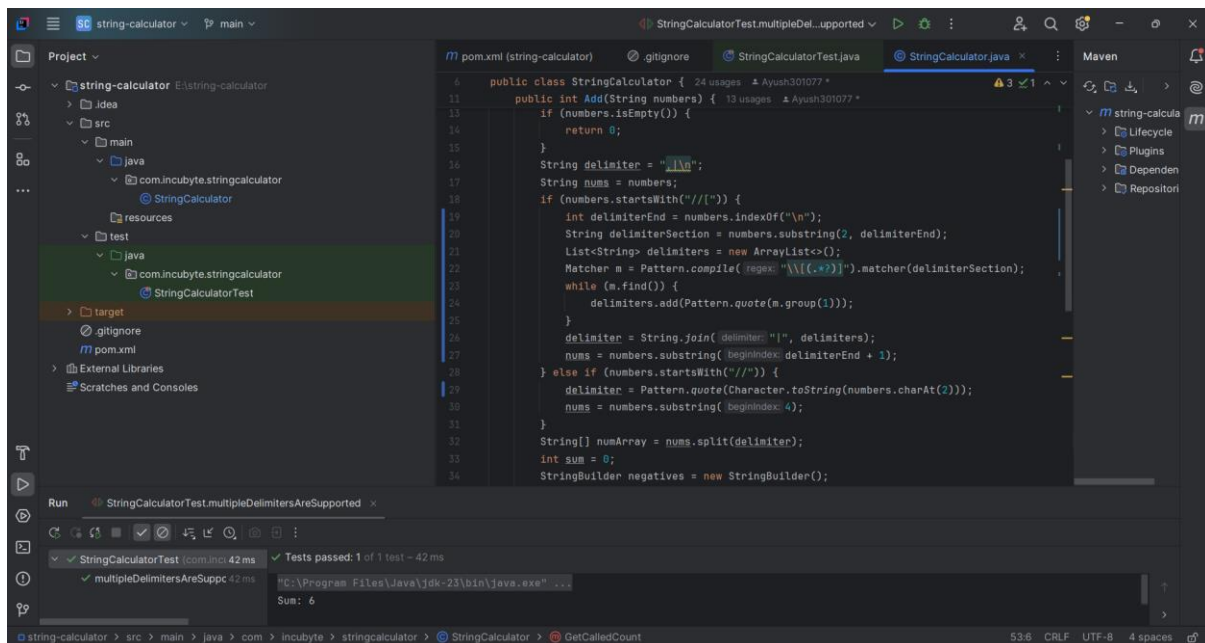
    @Test
    void multipleDelimitersAreSupported() {
        StringCalculator calculator = new StringCalculator();
        int result = calculator.Add(numbers: "[*][%]\n1*2%3");
        System.out.println("Sum: " + result);
        assertEquals(expected: 6, result);
    }
}

```

```

Run StringCalculatorTest.multipleDelimitersAreSupported
StringCalculatorTest (com.incl 42 ms)
multipleDelimitersAreSuppc 42 ms
Tests failed: 1 of 1 test - 42 ms
C:\Program Files\Java\jdk-23\bin\java.exe ...
java.lang.NumberFormatException: For input string: "1*2%3"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:588)
    at java.base/java.lang.Integer.parseInt(Integer.java:685)
    at com.incubyte.stringcalculator.StringCalculator.Add(StringCalculator.java:27)
    at com.incubyte.stringcalculator.StringCalculatorTest.multipleDelimitersAreSupported(StringCalculatorTest.java:104)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)

```



```

public class StringCalculator {
    public int Add(String numbers) {
        if (numbers.isEmpty()) {
            return 0;
        }
        String delimiter = ",\n";
        String nums = numbers;
        if (numbers.startsWith("//[") {
            int delimiterEnd = numbers.indexOf("\n");
            String delimiterSection = numbers.substring(2, delimiterEnd);
            List<String> delimiters = new ArrayList<>();
            Matcher m = Pattern.compile("\\[(.*?)\\]").matcher(delimiterSection);
            while (m.find()) {
                delimiters.add(Pattern.quote(m.group(1)));
            }
            delimiter = String.join("[", delimiters);
            nums = numbers.substring(beginIndex: delimiterEnd + 1);
        } else if (numbers.startsWith("//[") {
            delimiter = Pattern.quote(Character.toString(numbers.charAt(2)));
            nums = numbers.substring(beginIndex: 4);
        }
        String[] numArray = nums.split(delimiter);
        int sum = 0;
        StringBuilder negatives = new StringBuilder();
    }
}

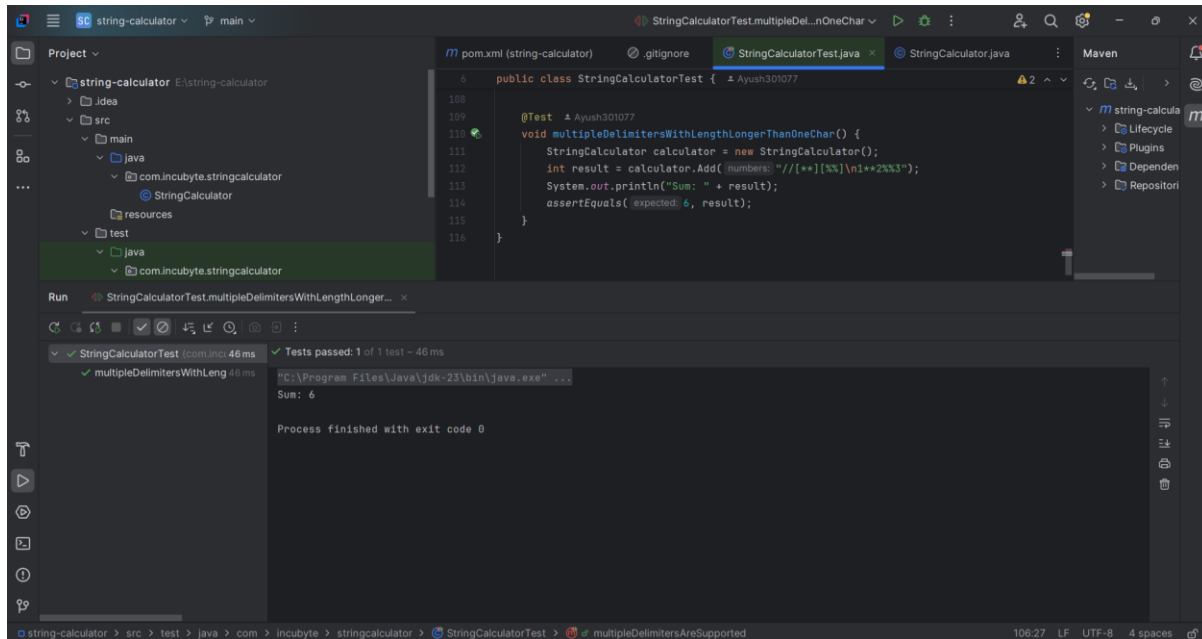
```

```

Run StringCalculatorTest.multipleDelimitersAreSupported
StringCalculatorTest (com.incl 42 ms)
multipleDelimitersAreSuppc 42 ms
Tests passed: 1 of 1 test - 42 ms
C:\Program Files\Java\jdk-23\bin\java.exe ...
Sum: 6

```

Step 13: Test for Multiple Delimiters with Length Longer Than One Character This step verifies that the StringCalculator can handle multiple delimiters, each of any length, specified in the format "[delim1][delim2]\n". The test checks that calling `Add("//[**][%%]\n1**2%%3")` returns 6, as required by the kata. The green checkmark and "Sum: 6" message will confirm that the implementation is correct for this case.



Below screenshot shows the successful completion of all TDD steps for the String Calculator kata. All 13 test cases are passing, confirming that the implementation correctly handles all the required tests and the code is developed using TDD approach

