Vaasan yliopisto
UNIVERSITY OF VAASA

NGUYEN Trung R5 – VAMK – e1500953

Exercise 5

Software testing: Test-Driven Development (TDD)

Part 1 –

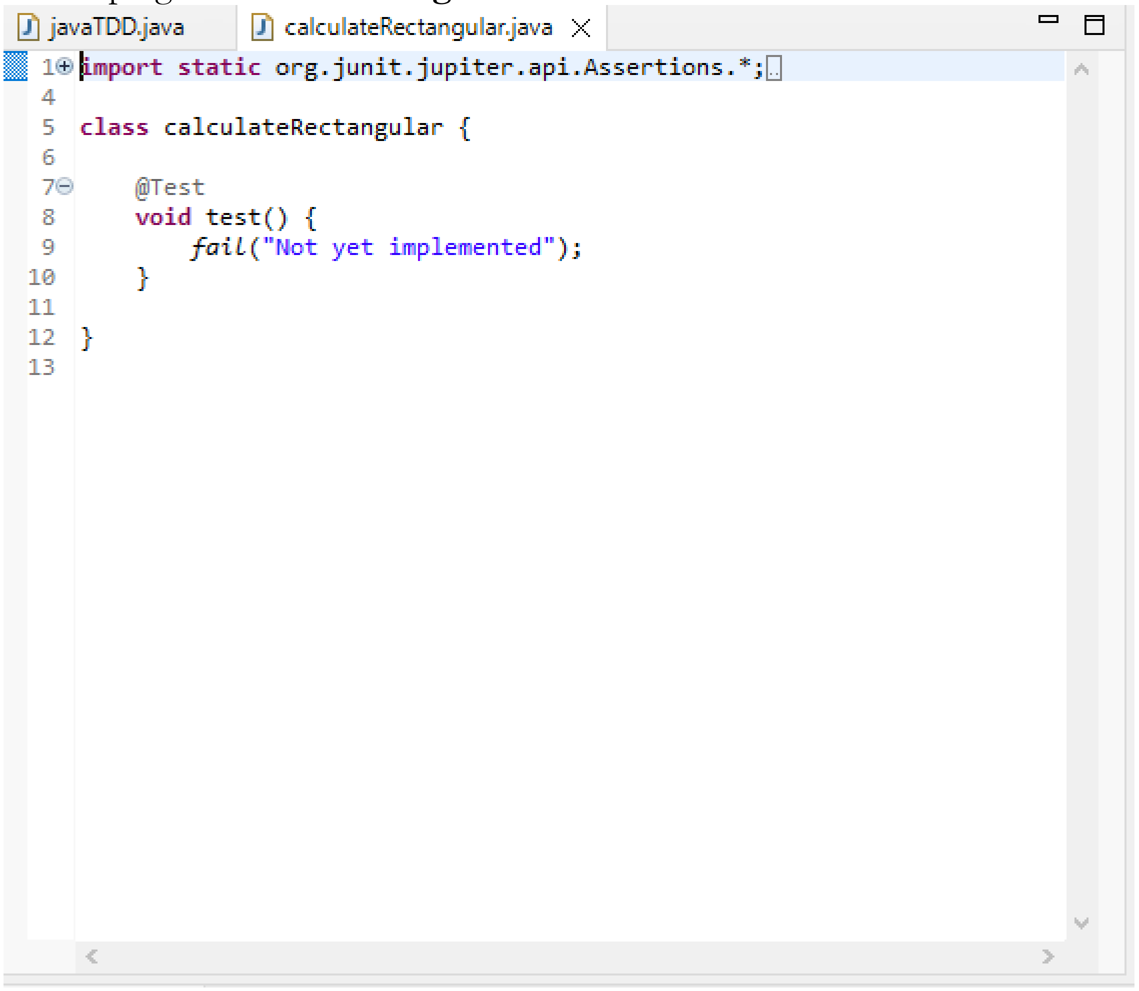
- Provide a short summary of your comments on the TDD approach. List what you considered good and what felt difficult. Use bullet points if needed.
- ANSWER:

  Test-Driven Development (TDD) is a software development process that emphasizes writing automated tests before writing the actual code. The primary advantage of using TDD is that it promotes better code quality, as developers are forced to think more about the design of their code before implementing it. This approach can lead to fewer bugs, easier maintenance, and more reliable software.

  However, there are also some difficulties associated with using TDD. One of the main challenges is that it can be time-consuming to write tests before implementing the code. This can slow down the development process and make it harder to meet tight deadlines. Additionally, writing good tests can require a significant amount of expertise and effort, which may be a barrier for less experienced developers. Finally, some developers may find it difficult to write tests for certain types of code, such as user interfaces or complex algorithms, which can make TDD less effective in those situations.

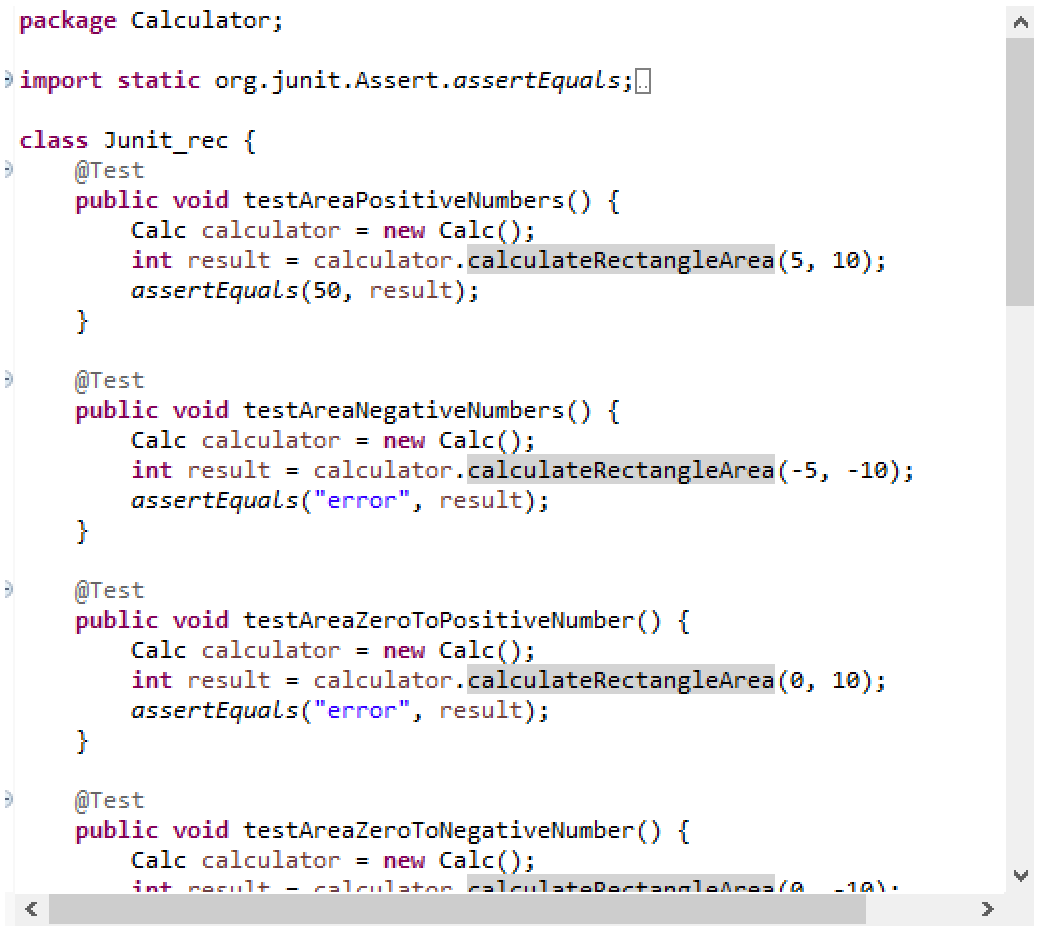
Part 2 –

- Include a screenshot showing all steps of developing the new functions using Test-Driven Development (TDD)
  - Developing **calculateRectangleArea** TDD



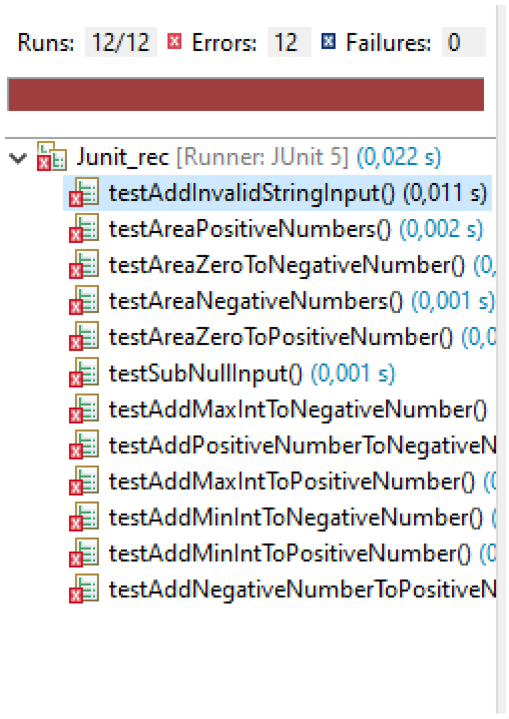
```
import static org.junit.jupiter.api.Assertions.*;

class calculateRectangular {

    @Test
    void test() {
        fail("Not yet implemented");
    }

}
```

Add unit test.



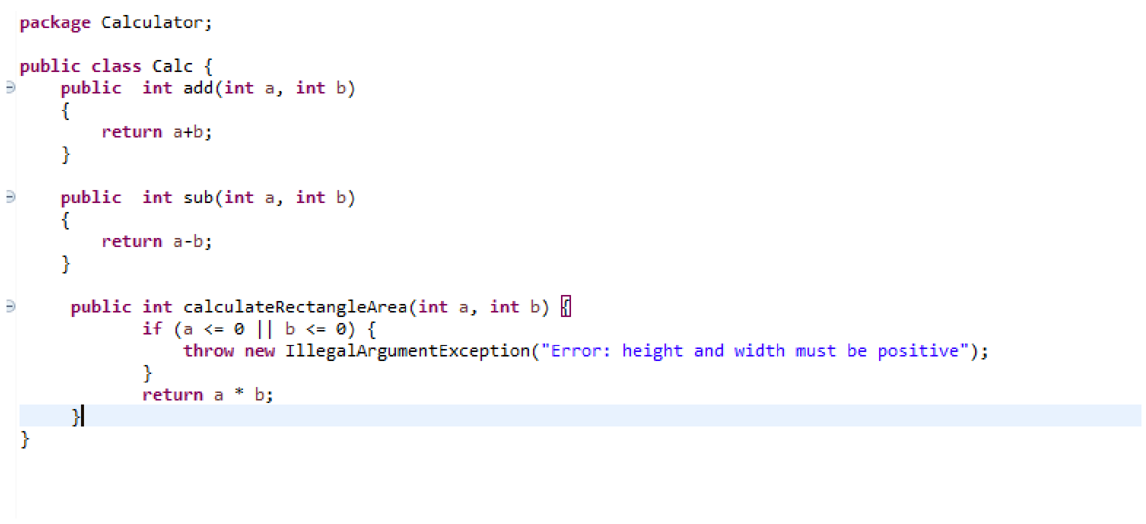
```
package Calculator;

import static org.junit.Assert.assertEquals;

class Junit_rec {
    @Test
    public void testAreaPositiveNumbers() {
        Calc calculator = new Calc();
        int result = calculator.calculateRectangleArea(5, 10);
        assertEquals(50, result);
    }

    @Test
    public void testAreaNegativeNumbers() {
        Calc calculator = new Calc();
        int result = calculator.calculateRectangleArea(-5, -10);
        assertEquals("error", result);
    }

    @Test
    public void testAreaZeroToPositiveNumber() {
        Calc calculator = new Calc();
        int result = calculator.calculateRectangleArea(0, 10);
        assertEquals("error", result);
    }

    @Test
    public void testAreaZeroToNegativeNumber() {
        Calc calculator = new Calc();
        int result = calculator.calculateRectangleArea(0, -10);
```

Add @Test cases for the Junit test of calculating area



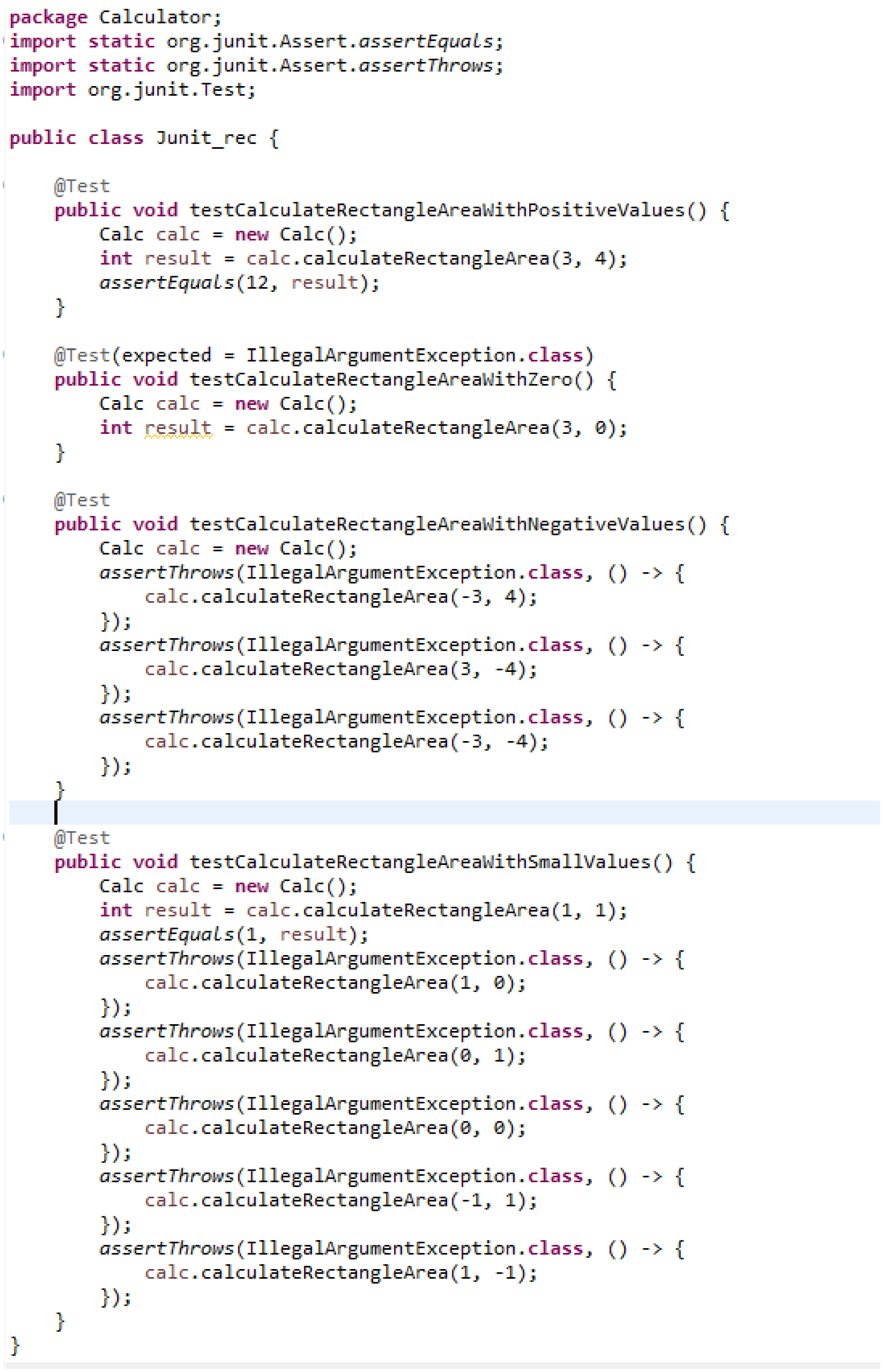
```
Runs: 12/12   Errors: 12   Failures: 0

Junit_rec [Runner: JUnit 5] (0,022 s)
    testAddInvalidStringInput() (0,011 s)
    testAreaPositiveNumbers() (0,002 s)
    testAreaZeroToNegativeNumber() (0,
    testAreaNegativeNumbers() (0,001 s)
    testAreaZeroToPositiveNumber() (0,0
    testSubNullInput() (0,001 s)
    testAddMaxIntToNegativeNumber()
    testAddPositiveNumberToNegativeN
    testAddMaxIntToPositiveNumber() ((
    testAddMinIntToNegativeNumber() (
    testAddMinIntToPositiveNumber() ((
    testAddNegativeNumberToPositiveN
```

All the test failed.



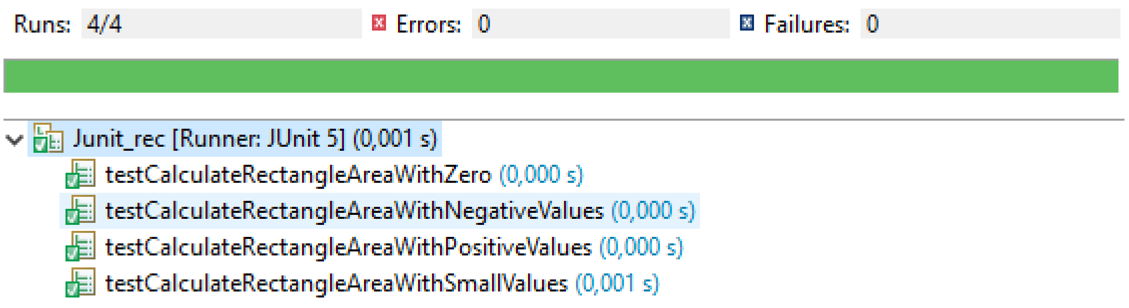
```
package Calculator;

public class Calc {
    public  int add(int a, int b)
    {
        return a+b;
    }

    public  int sub(int a, int b)
    {
        return a-b;
    }

     public int calculateRectangleArea(int a, int b) {
            if (a <= 0 || b <= 0) {
                throw new IllegalArgumentException("Error: height and width must be positive");
            }
            return a * b;
     }
}
```

write the function for this test. when a and b >0 return a*B, all other situation, throw the exceoption.



```java
package Calculator;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertThrows;
import org.junit.Test;

public class Junit_rec {

    @Test
    public void testCalculateRectangleAreaWithPositiveValues() {
        Calc calc = new Calc();
        int result = calc.calculateRectangleArea(3, 4);
        assertEquals(12, result);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testCalculateRectangleAreaWithZero() {
        Calc calc = new Calc();
        int result = calc.calculateRectangleArea(3, 0);
    }

    @Test
    public void testCalculateRectangleAreaWithNegativeValues() {
        Calc calc = new Calc();
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(-3, 4);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(3, -4);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(-3, -4);
        });
    }

    @Test
    public void testCalculateRectangleAreaWithSmallValues() {
        Calc calc = new Calc();
        int result = calc.calculateRectangleArea(1, 1);
        assertEquals(1, result);
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(1, 0);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(0, 1);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(0, 0);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(-1, 1);
        });
        assertThrows(IllegalArgumentException.class, () -> {
            calc.calculateRectangleArea(1, -1);
        });
    }
}
```

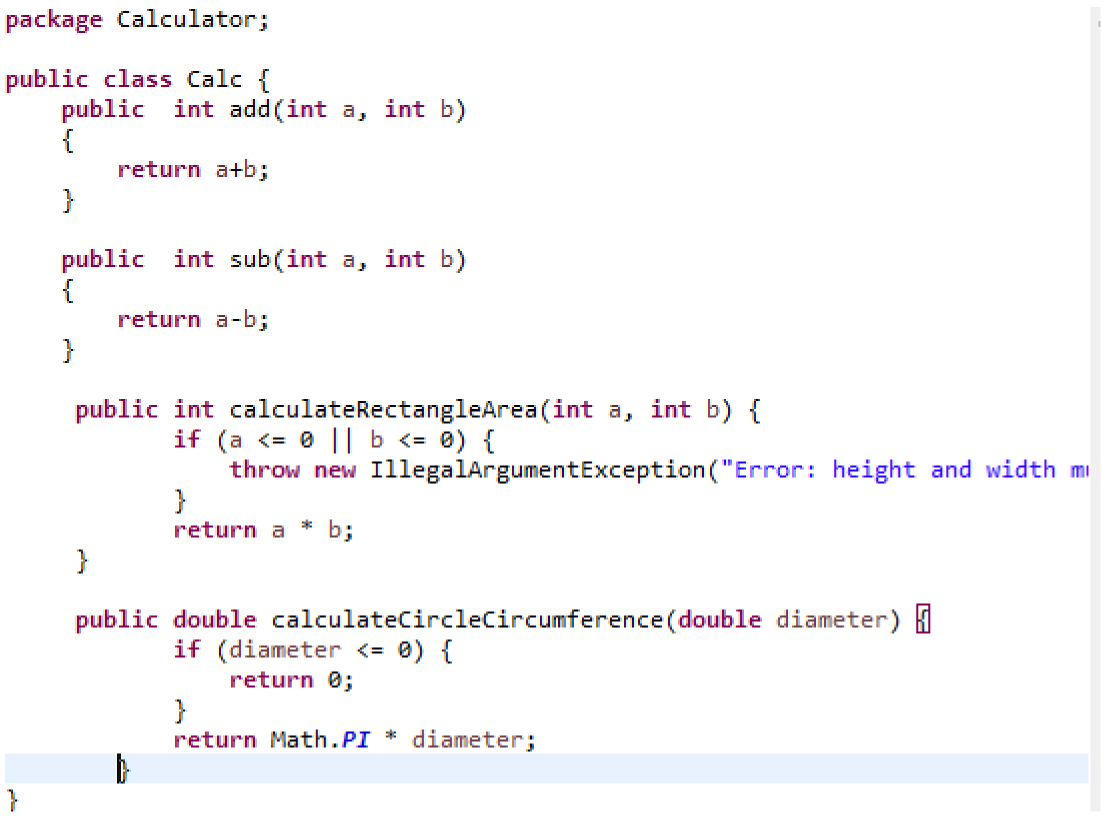
Fix the test, and refactor the code



```
Runs: 4/4        Errors: 0        Failures: 0

Junit_rec [Runner: JUnit 5] (0,001 s)
    testCalculateRectangleAreaWithZero (0,000 s)
    testCalculateRectangleAreaWithNegativeValues (0,000 s)
    testCalculateRectangleAreaWithPositiveValues (0,000 s)
    testCalculateRectangleAreaWithSmallValues (0,001 s)
```

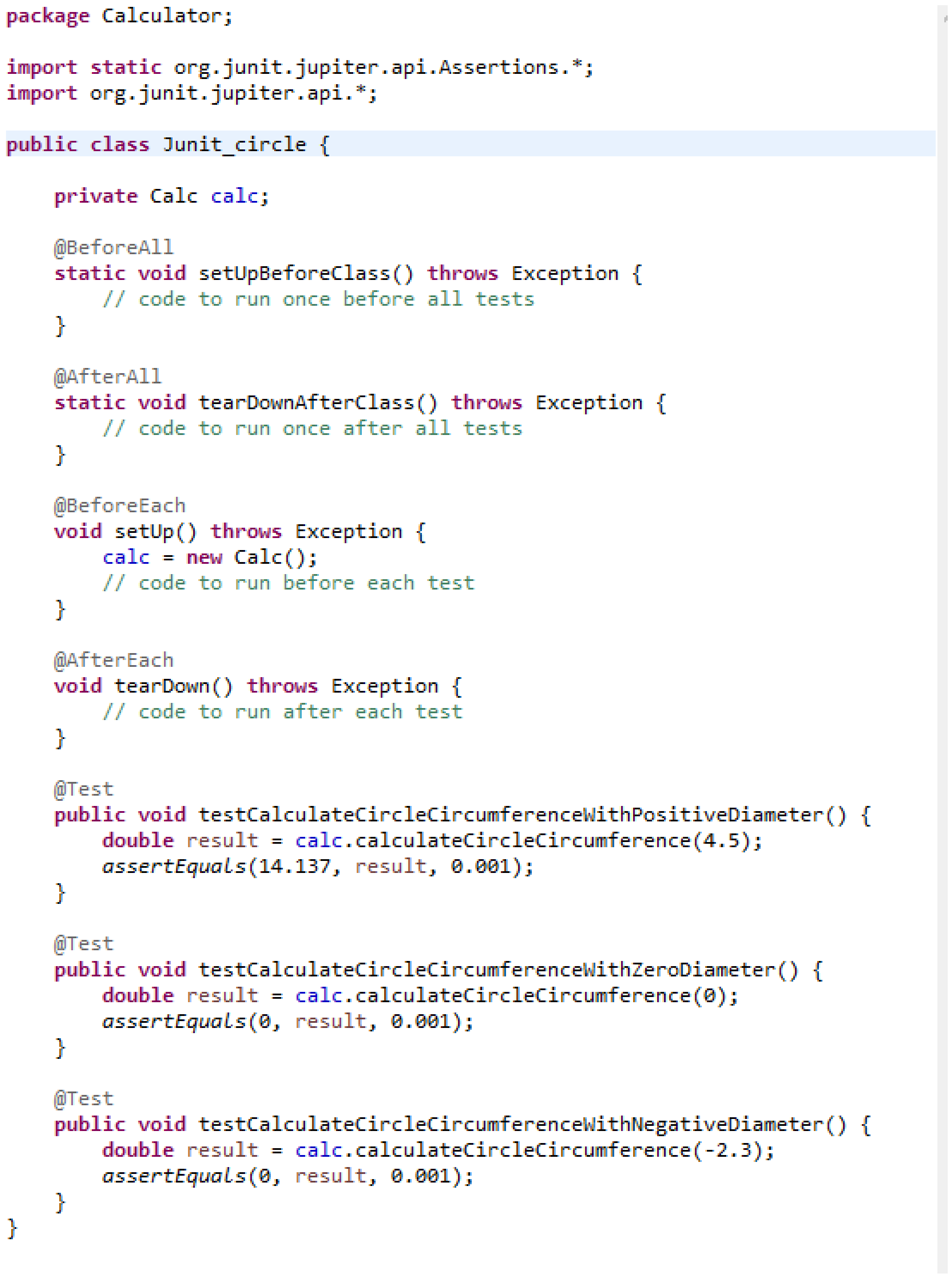
Result of the test for newly added fuction.

Based on the instruction in the exercise. I do the same for circle function follow TDD steps. After testing and refactoring. here is the version of the code



```java
package Calculator;

public class Calc {
    public  int add(int a, int b)
    {
        return a+b;
    }

    public  int sub(int a, int b)
    {
        return a-b;
    }

     public int calculateRectangleArea(int a, int b) {
            if (a <= 0 || b <= 0) {
                throw new IllegalArgumentException("Error: height and width m
            }
            return a * b;
     }

     public double calculateCircleCircumference(double diameter) {
            if (diameter <= 0) {
                return 0;
            }
            return Math.PI * diameter;
        }
}
```

And the Junit test for new function :



```java
package Calculator;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.*;

public class Junit_circle {

    private Calc calc;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        // code to run once before all tests
    }

    @AfterAll
    static void tearDownAfterClass() throws Exception {
        // code to run once after all tests
    }

    @BeforeEach
    void setUp() throws Exception {
        calc = new Calc();
        // code to run before each test
    }

    @AfterEach
    void tearDown() throws Exception {
        // code to run after each test
    }

    @Test
    public void testCalculateCircleCircumferenceWithPositiveDiameter() {
        double result = calc.calculateCircleCircumference(4.5);
        assertEquals(14.137, result, 0.001);
    }

    @Test
    public void testCalculateCircleCircumferenceWithZeroDiameter() {
        double result = calc.calculateCircleCircumference(0);
        assertEquals(0, result, 0.001);
    }

    @Test
    public void testCalculateCircleCircumferenceWithNegativeDiameter() {
        double result = calc.calculateCircleCircumference(-2.3);
        assertEquals(0, result, 0.001);
    }
}
```

I use @BeforeAll and @AfterAll to set up and tear down the test environment once before and after all tests, respectively. I use @BeforeEach and @AfterEach to set up and tear down the test environment before and after each test, respectively.

I also define three test cases for the calculateCircleCircumference method: one with a positive diameter, one with a zero diameter, and one with a negative diameter. Each test case uses the assertEquals method to check if the expected result matches the actual result returned by the method.