# TASK 2:

### **Aim:** To design test cases for following problem using path testing.

**Problem**:

### A mobile service provider calculate the bill payment of a customer as Follows

* If the number of calls are 120 then minimum payment is 300
* Plus Rs.1 for each call for the next 70 calls
* Plus Rs.0.82 for each call for the next 50 calls
* Plus Rs.0.45 for each call for more than 240 calls

**Program:**

#include <stdio.h> int main()

{

int calls; float pay=0;

printf("Enter number of calls"); scanf("%d",&calls);

if(calls>0)

{

if(calls<120) pay=300;

else if(calls>120 && calls<=190) pay=300+(calls-120)\*1;

else if(calls>190 && calls<=240) pay=300+ (70\*1) +(calls-190)\*0.82;

else

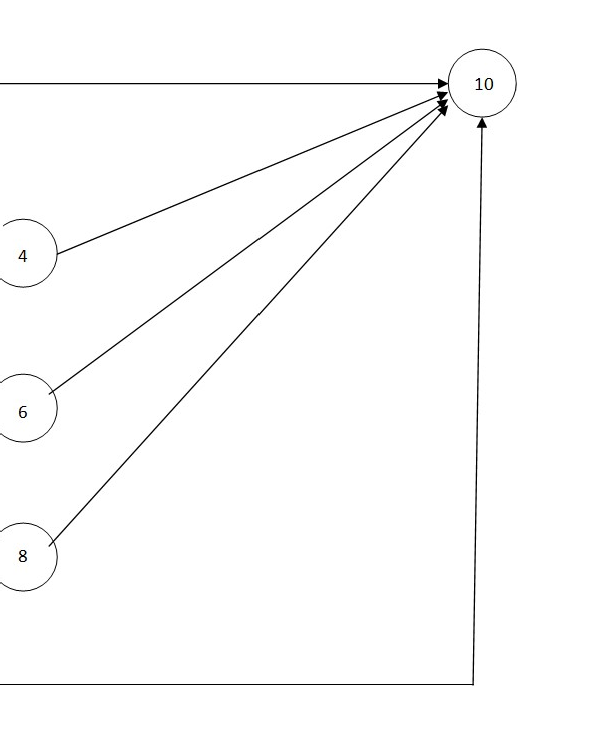
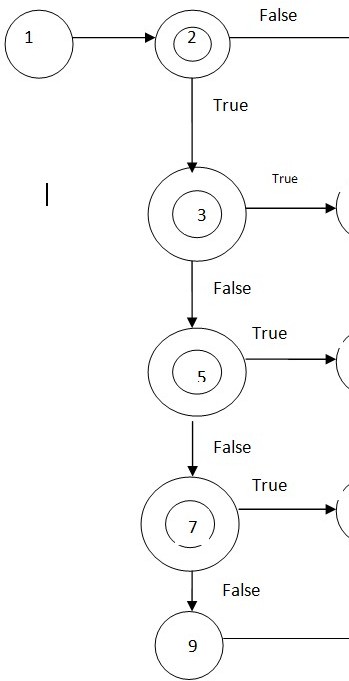
pay=300+ (70\*1) + (50\*0.82) +(calls-240)\*0.45;

}

printf("%0.2f",pay); return 0;

}

**Flow Graph:**



#### Cyclomatic Complexity:

The cyclomatic complexity of the above flow graph is,

Number of edges=13 Number of nodes=10

Therefore, Cyclomatic Complexity = Number of edges-Number of nodes+2

= 13-10+2

= 5

#### Independents Paths:

The possible independent paths are, Path-1: 1-2-10

Path-2: 1-2-3-4-10

Path-3: 1-2-3-5-6-10

Path-4: 1-2-3-5-7-8-10

Path-5:1-2-3-5-7-9-10

## Test Matrix:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Path | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |

**Test suite design:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project Name**: Bill Payment Calculation | | | | | | |
| **Test case id**:ID\_2 **Test Designed by**: Kundana  **Test Priority**: low **Test Designed Date**:  **Module Name**: Bill payment **Test Executed by**: Kundana  **Test Title:** White box  **Test Executed Date**:  **Description**: Test case for problem using Path Testing | | | | | | |
| **Pre-Condition**: Customer should make at least one call | | | | | | |
| **Stage** | **Test steps** | **Test Data** | **Expected Result** | **Actual Result** | **status (Pass/Fail)** | **Remarks** |
| 1 | path 1 | 20 | 300.00 | 300.00 | Pass | Nil |
| 2 | path 2 | 150 | 330.00 | 330.00 | Pass | Nil |
| 3 | path 3 | 210 | 386.39 | 386.39 | Pass | Nil |
| 4 | path 4 | 250 | 415.50 | 415.50 | Pass | Nil |
| **Post Condition**: Bill payment of a customer is calculated based on the number of calls | | | | | | |

### **Result:** Thus, design of test cases for the problem using path testing is done successfully.

**TASK 3:**



**Aim:** To perform parameterized testing using Junit

**Procedure:**

**SSteps:**

1. Creating Java Project
   * Click on File and select New project
   * Enter project name as com.vogella.Junit.Addition
   * Click on Next and then on Finish
2. Creating java Test project
   * Right click on com.vogella.Junit.Addition
   * Click on properties and select tab build path
   * Click on source and click on Create New Folder
   * Give the folder name as Test and click on next
   * Click on Finish and then on OK
3. Creating java class
   * Right click on com.vogella.Junit.Addition and click on New
   * Click on class and give class name as Add
   * Click on Finish
   * Type the following code

**Add.java**

**package** com.vogella.JUnit.Addition;

**public class** Add {

**public int** addNumbers(**int** a,**int** b)

{

**int** sum=a+b;

**return sum;**

}

}

1. Create java test class
   * Right click on com.vogella.Junit.Addition and then click on new
   * Click on Junit test case
   * Change the name of folder src to test in source folder tab
   * Click on browse and select Add class and click on Next
   * Click on the Finish and then on OK



* + Add the following code

**Add1.java**

**package** com.vogella.JUnit.Addition;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

**import static** org.junit.Assert.\*;

**import** org.junit.Test; @RunWith(Parameterized.**class**) **public class** Add1 {

**private int** expected,first,second;

**public** Add1(**int** expectedResult,**int** firstNumber,**int** secondNumber)

{

**this**.expected=expectedResult; **this**.first=firstNumber; **this**.second=secondNumber;

}

@Parameters

**public static** Collection<Integer[]> addedNumbers(){

**return** Arrays.*asList*(**new** Integer[][]{{10,6,4},{2,1,1},{6,3,3},{20,15,5},{13,9,4}});

}

@Test

**public void** test() {

Add a=**new** Add();

System.***out***.println("Addition with parameters :"+first+" and "+second);

*assertEquals*(expected,a.addNumbers(first,second));

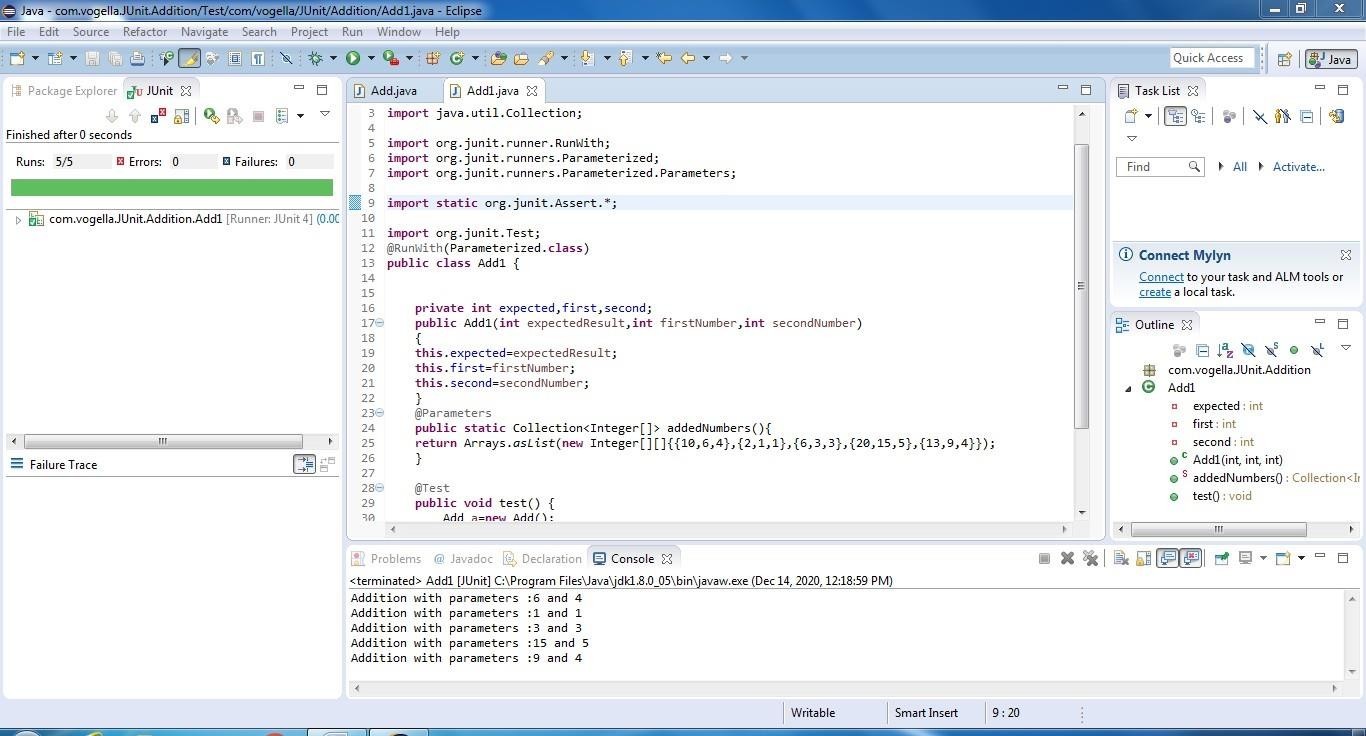
}

}

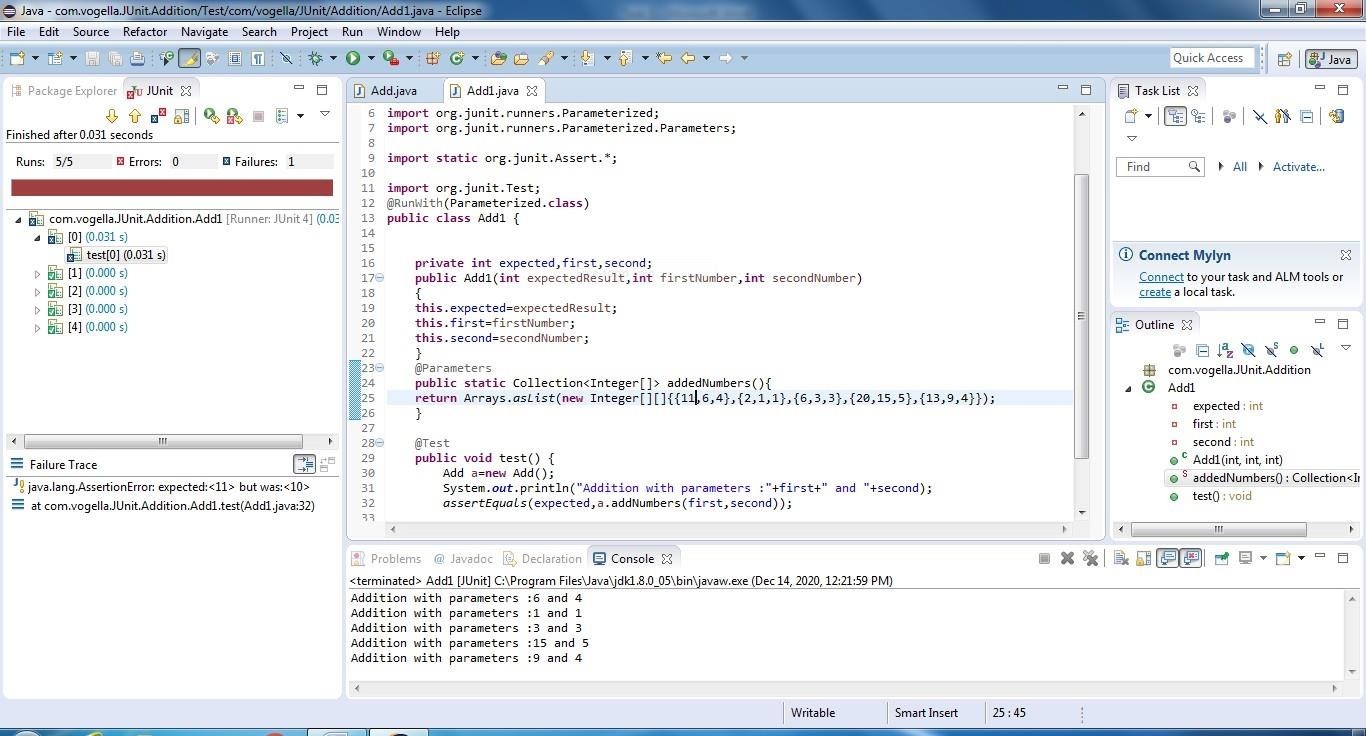
**Outputs:**



Pass case for Addition



Fail case for Addition



**T est suite design:**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project Name**: Parameterized Testing | | | | | | |
| **Test case id**: ID\_3 **Test Designed by:** Kundana  **Test Priority**: low **Test Designed Date**:  **Module Name**: Junit Testing **Test Executed by**: Kundana  **Test Title**: Blackbox testing  **Test Executed Date**:  **Description:** Test case for problem using Parameterized Testing | | | | | | |
| **Pre-Condition**: User should give two input numbers and one expected output | | | | | | |
| **Stage** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **status (Pass/Fail)** | **Remarks** |
| 1 | Two valid integer  values | 6,4 | 10 | 10 | Pass | Nil |
| 2 | 1,1 | 2 | 2 | Pass | Nil |
| 3 | 3,3 | 6 | 6 | Pass | Nil |
| 4 | 15,5 | 20 | 20 | Pass | Nil |
| 5 | 9,4 | 13 | 11 | Fail | Nil |
| **Post condition**: Addition of two numbers should match the defined expected | | | | | | |

**R esult:** Performing parameterized testing using Junit has been done successfully.

# TASK 4:

**Aim :** To perform parameterized testing using Junit

#### Procedure:

**Steps:**

1. Creating Java Project
   * Click on File and select New project
   * Enter project name as com.vogella.Junit.Triangle
   * Click on Next and then on Finish
2. Creating java Test project
   * Right click on com.vogella.Junit. Triangle
   * Click on properties and select tab build path
   * Click on source and click on Create New Folder
   * Give the folder name as Test and click on next
   * Click on Finish and then on OK
3. Creating java class
   * Right click on com.vogella.Junit. Triangle and click on New
   * Click on class and give class name as Triangle
   * Click on Finish
   * Type the following code

#### Triangle.java

**package** com.vogella.JUnit.Triangle;

**public class** Triangle {

**public int** check(**int** a,**int** b,**int** c)

{

**if**(a==b && b==c && a==c)

**return** 1;

**else if**(a==b || b==c || c==a)

**return** 2;

**else**

}

}

**return** 3;

1. Create java test class
   * Right click on com.vogella.Junit.Triangle and then click on new
   * Click on Junit test case
   * Change the name of folder src to test in source folder tab
   * Click on browse and select Triangle1 class and click on Next
   * Click on the Finish and then on OK
   * Add the following code

#### Triangle1.java

**package** com.vogella.JUnit.Triangle;

**import static** org.junit.Assert.\*;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.**class**)

**public class** Triangle1 {

**private int** a; **private int** b; **private int** c; **private int** result;

**public** Triangle1(**int** res,**int** a1,**int** b1,**int** c1)

{

**this**.result=res; **this**.a=a1; **this**.b=b1; **this**.c=c1;

}

@Parameters

**public static** Collection<Integer[]> addNum()

{

**return** Arrays.*asList*(**new** Integer[][]{{1,2,2,2},{2,3,2,2},{3,1,2,3}});

}

@Test

**public void** test() {

Triangle t=**new** Triangle(); **int** n=t.check(a, b, c); **switch**(n)

{

**case** 1: System.***out***.println("It is Equilateral triangle");

**break**;

**case** 2: System.***out***.println("It is Isosceles triangle");

**break**;

**case** 3: System.***out***.println("It is Scalene triangle");

**break**;

**default**: System.***out***.println("It is not a valid triangle");

**break**;

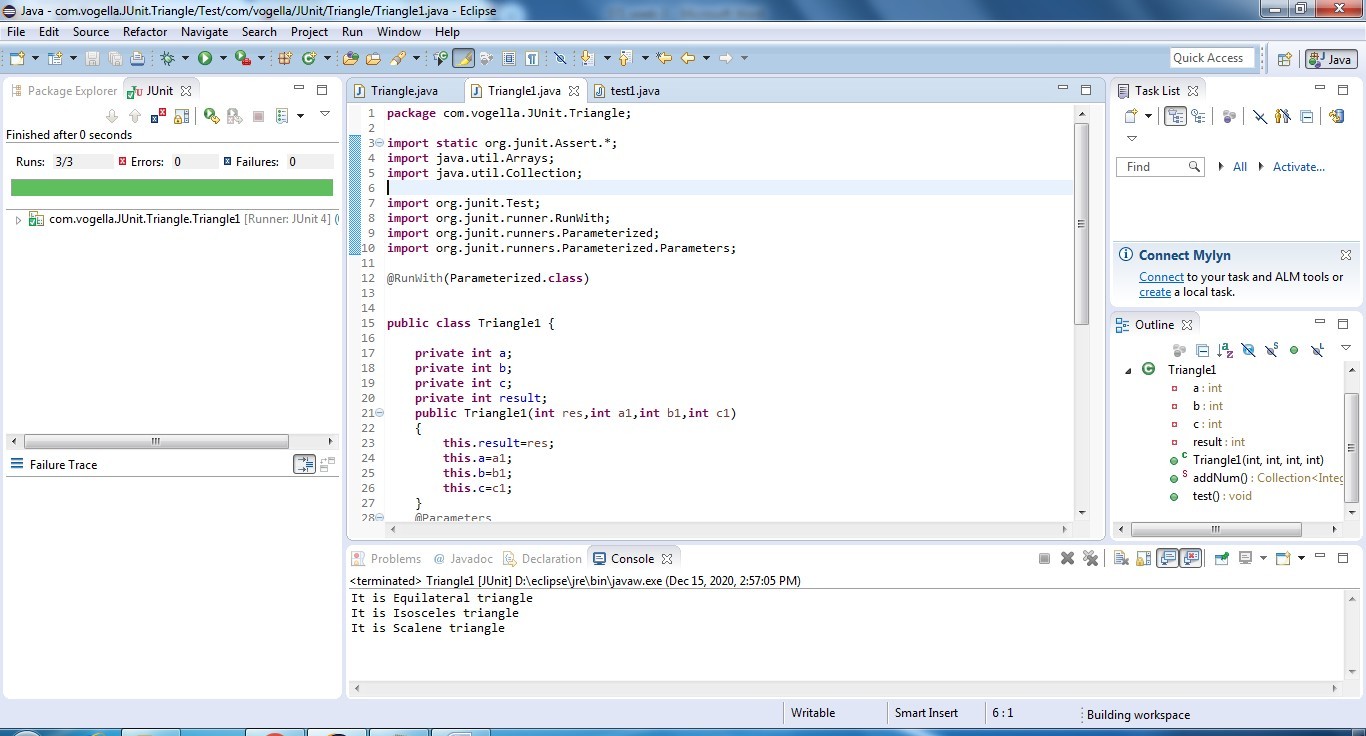
}

*assertEquals*(result,t.check(a, b,c));

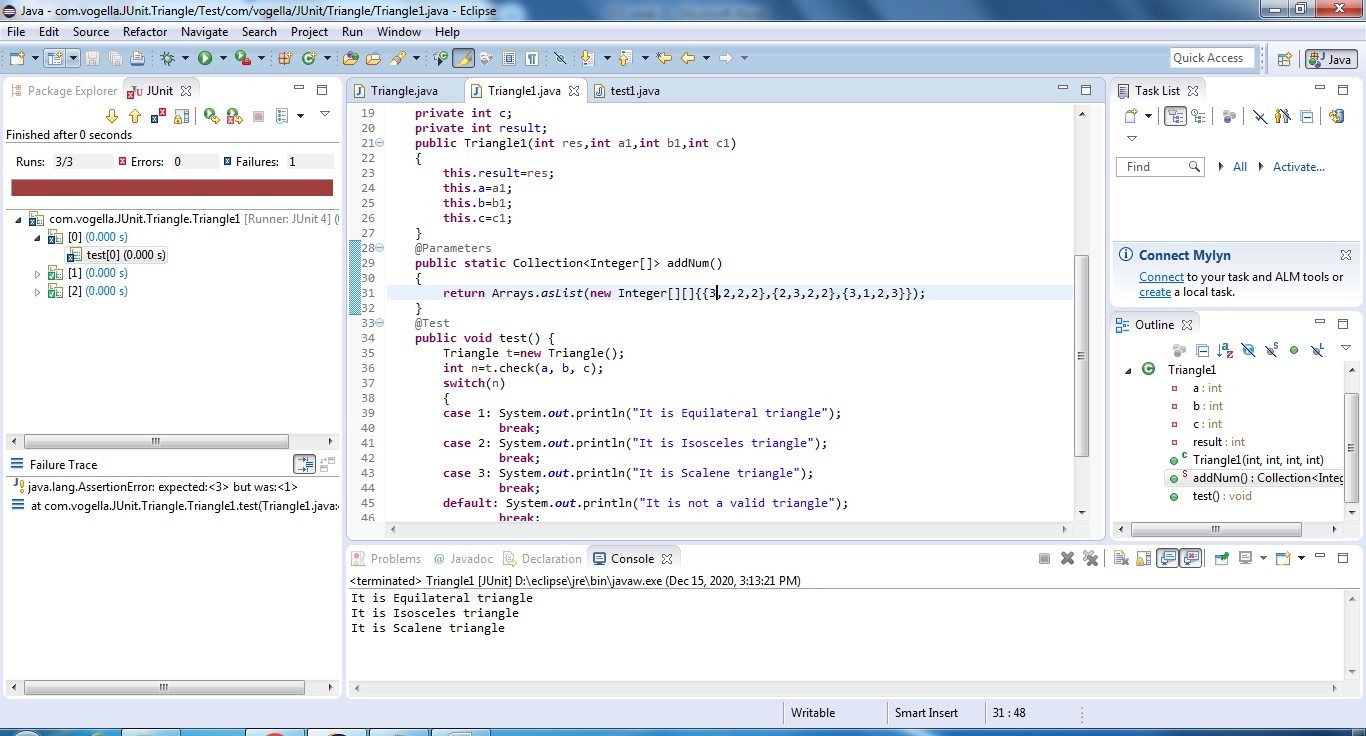
}

}

#### Outputs:

Pass case for Triangle Type

Fail case for Triangle Type



#### Test suite design:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project Name**: Parameterized Testing | | | | | | |
| **Test case id**: ID\_4 **Test Designed by:** Leela  **Test Priority**: low **Test Designed Date**:  **Module Name**: Junit Testing **Test Executed by**: Leela  **Test Title**: Blackbox testing  **Test Executed Date**:  **Description:** Test case for problem using Parameterized Testing | | | | | | |
| **Pre-Condition**: User should give three input numbers and one expected output | | | | | | |
| **Stage** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **status (Pass/Fail)** | **Remarks** |
| 1 | Three valid integer values | 1,1,1 | 1 | 1 | Pass | Nil |
| 2 | 4,3,3 | 2 | 2 | Pass | Nil |
| 3 | 8,2,5 | 3 | 3 | Pass | Nil |
| 4 | 1,2,1 | 2 | 2 | Pass | Nil |
| 5 | 3,2,2 | 1 | 3 | Fail | Nil |
| **Post condition**: Expected result should match with value returned by function | | | | | | |

**Result:** Performing parameterized testing using Junit has been done successfully.