IT 314

Software Engineering

Lab 7

Name : Abhay Suvagiya

ID: 202001171

Section A

Q1: Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges
1 <= month <= 12,
1 <= day <= 31,
1900 <= year <= 2015.
The possible output dates would be the previous date or invalid date.
Design the equivalence class test cases?
Ans.
Equivalence classes:
Day

1) day between 1 to 31 (both inclusive) **valid**
2) day more than 31 (31 excluded) **invalid**
3) day less than 1(1 not included) **invalid**

Month

1) month between 1 to 12 (both inclusive) **valid**
2) month more than 12 (12 excluded) **invalid**
3) month less than 1 (1 excluded) **invalid**

Year

1) year between 1900 to 2015 (both inclusive) **valid**
2) year more than 2015 (2015 excluded) **invalid**
3) year less than 1900 (1900 excluded) **invalid**

Some Test Cases:

1. Valid test cases:
    a. (1, 1, 1900) - the minimum valid date
    b. (15, 7, 2006) - a random valid date
    c. (31, 12, 2015) - the maximum valid date
2. Invalid test cases:
    a. (0, 7, 1999) - day is less than 1
    b. (33, 2, 2004) - day is greater than 31
    d. (2, 5, 2017) - the year is greater than 2015
    e. (15, 15, 1998) - month is greater than 12
    f. (-10, -5, 1901) - all parameters are invalid

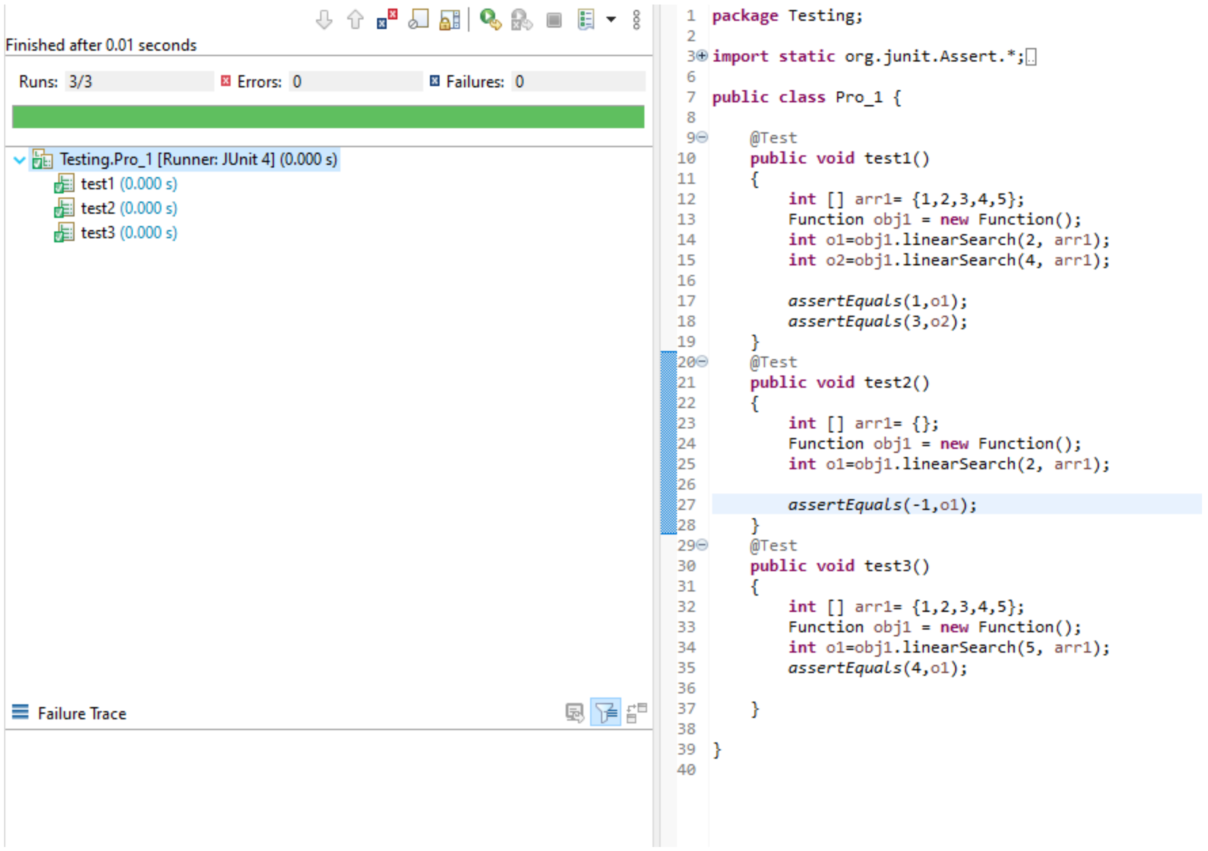
These test cases represent the equivalence classes and should cover all possible scenarios.

Programs:

P1: The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

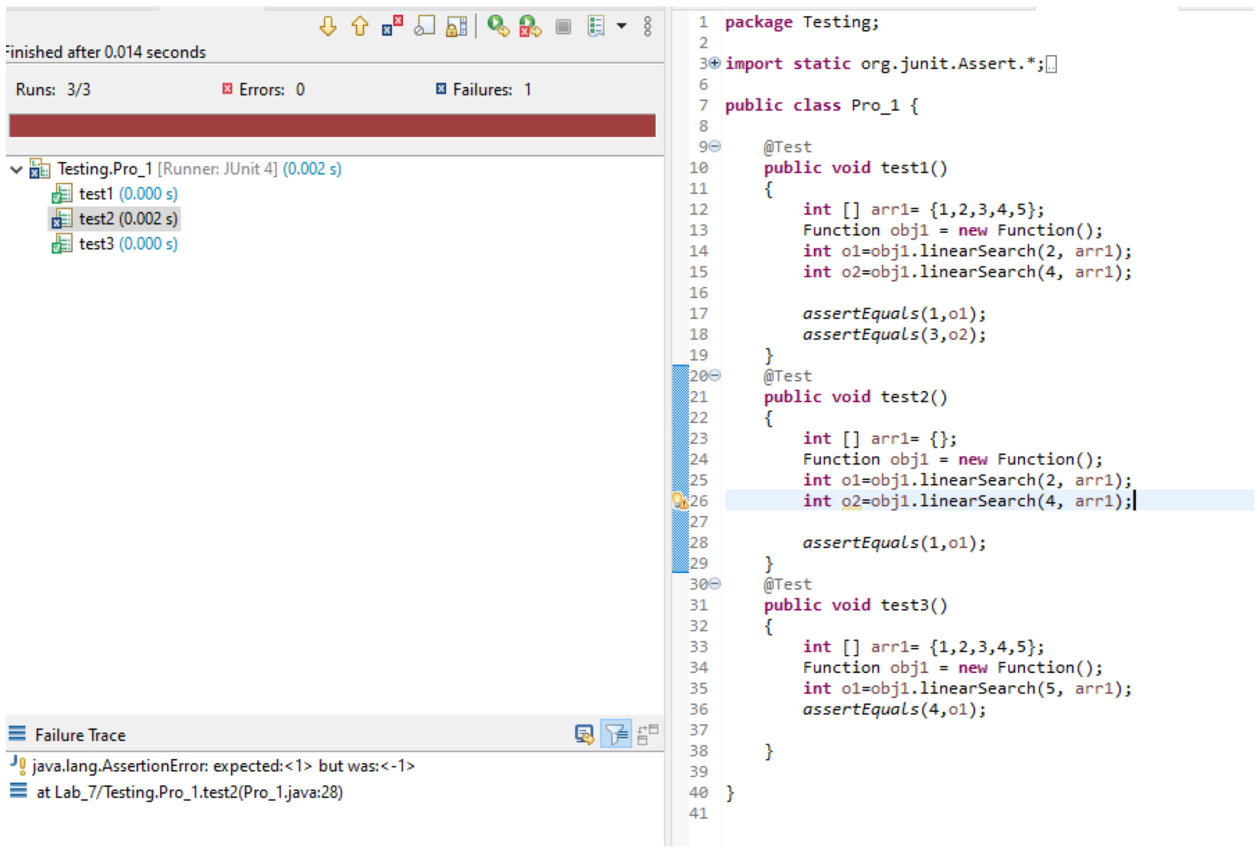
```
int linearSearch(int v, int a[])
{
        int i = 0;
        while (i < a.length)
        {
        if (a[i] == v)
        return(i);
        i++;
        }
        return (-1);
}
```

Test Case in Eclipse:



```
Finished after 0.01 seconds
Runs: 3/3        Errors: 0        Failures: 0

Testing.Pro_1 [Runner: JUnit 4] (0.000 s)
    test1 (0.000 s)
    test2 (0.000 s)
    test3 (0.000 s)

Failure Trace
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_1 {

    @Test
    public void test1()
    {
        int [] arr1= {1,2,3,4,5};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(2, arr1);
        int o2=obj1.linearSearch(4, arr1);

        assertEquals(1,o1);
        assertEquals(3,o2);
    }
    @Test
    public void test2()
    {
        int [] arr1= {};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(2, arr1);

        assertEquals(-1,o1);
    }
    @Test
    public void test3()
    {
        int [] arr1= {1,2,3,4,5};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(5, arr1);
        assertEquals(4,o1);

    }

}
```



```
Finished after 0.014 seconds
Runs: 3/3        Errors: 0        Failures: 1

Testing.Pro_1 [Runner: JUnit 4] (0.002 s)
    test1 (0.000 s)
    test2 (0.002 s)
    test3 (0.000 s)

Failure Trace
java.lang.AssertionError: expected:<1> but was:<-1>
at Lab_7/Testing.Pro_1.test2(Pro_1.java:28)
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_1 {

    @Test
    public void test1()
    {
        int [] arr1= {1,2,3,4,5};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(2, arr1);
        int o2=obj1.linearSearch(4, arr1);

        assertEquals(1,o1);
        assertEquals(3,o2);
    }
    @Test
    public void test2()
    {
        int [] arr1= {};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(2, arr1);
        int o2=obj1.linearSearch(4, arr1);

        assertEquals(1,o1);
    }
    @Test
    public void test3()
    {
        int [] arr1= {1,2,3,4,5};
        Function obj1 = new Function();
        int o1=obj1.linearSearch(5, arr1);
        assertEquals(4,o1);

    }

}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an empty array a[ ]	-1
Test with v as a non-existent value and a non-empty array a[ ]	-1
Test with v as an existent value and an empty array a[ ]	-1
Test with v as an existent value and a non-empty array a[ ] where v exists	the index of v in a[]
Test with v as an existent value and a non-empty array a[ ] where v does not exist	-1

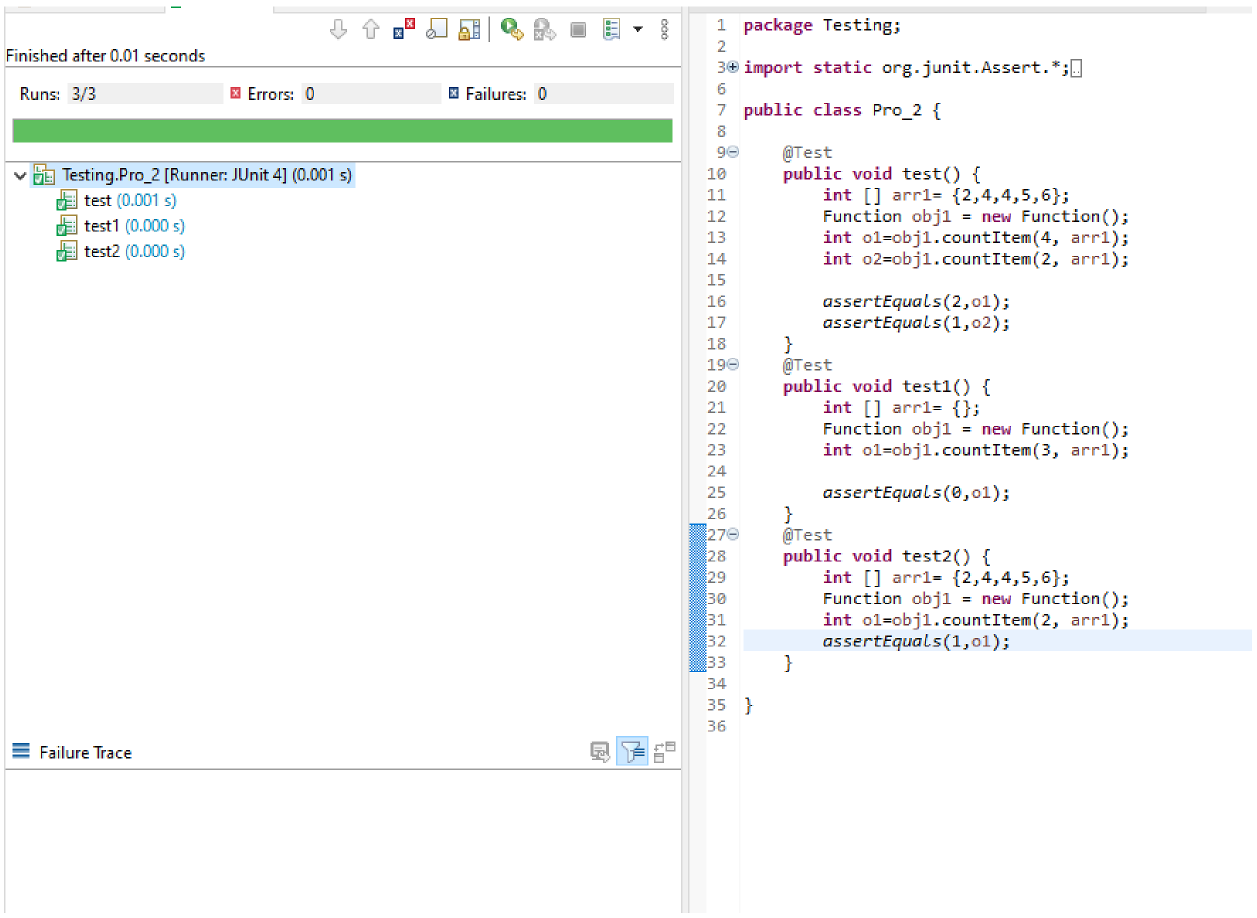
Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an empty array a[ ]	-1
Test with v as a non-existent value and a non-empty array a[ ]	-1
Test with v as an existent value and an empty array a[ ] of length 0	-1
Test with v as an existent value and an array a[] of length 1, where v exists	0
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array	0
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array	the last index

P2: The function countItem returns the number of times a value v appears in an array of integers a.

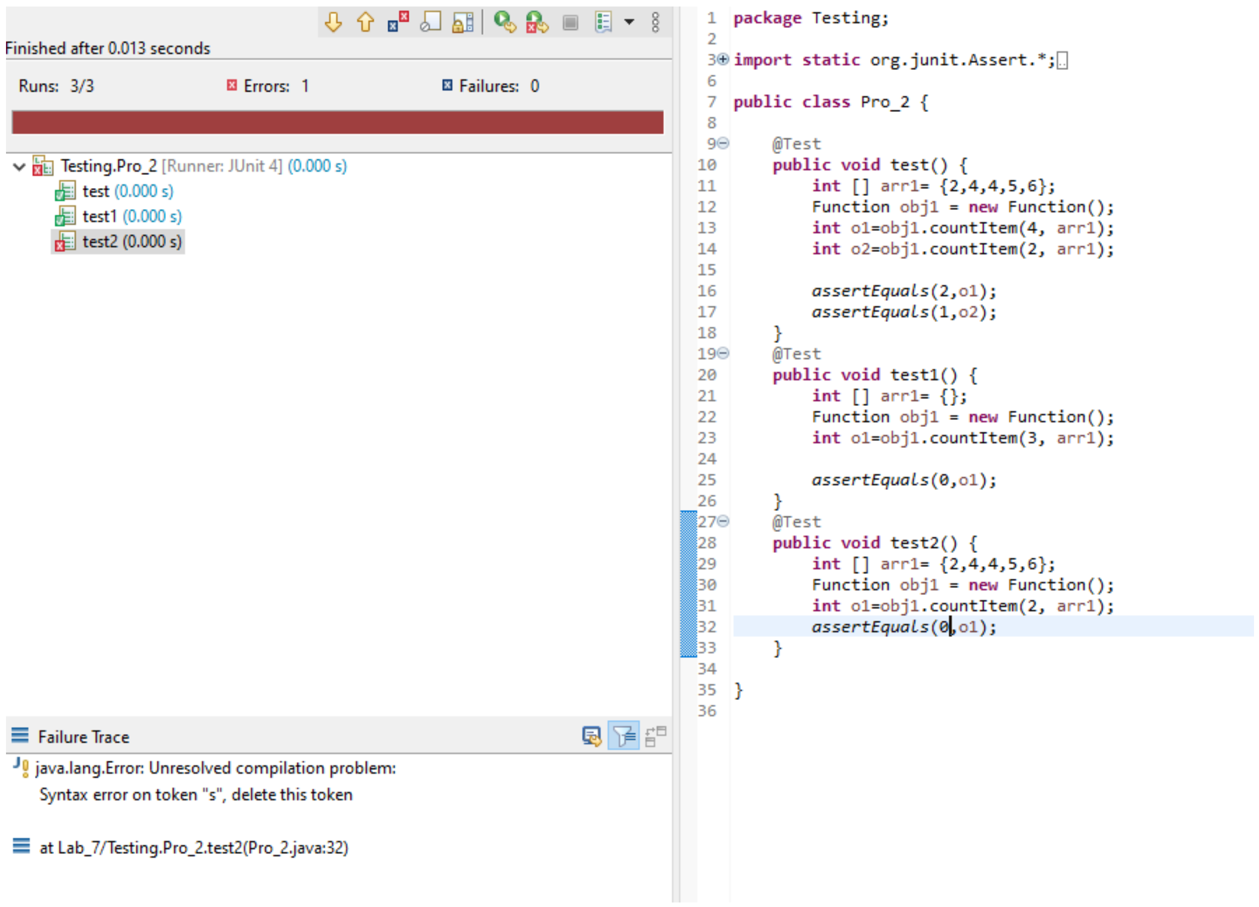
```
    int countItem(int v, int a[])
    {
        int count = 0;
        for (int i = 0; i < a.length; i++)
        {
        if (a[i] == v)
        count++;

        }
        return (count);
    }
```

Test Case in Eclipse:



```
Finished after 0.01 seconds

Runs: 3/3    Errors: 0    Failures: 0

Testing.Pro_2 [Runner: JUnit 4] (0.001 s)
    test (0.001 s)
    test1 (0.000 s)
    test2 (0.000 s)

Failure Trace
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_2 {

    @Test
    public void test() {
        int [] arr1= {2,4,4,5,6};
        Function obj1 = new Function();
        int o1=obj1.countItem(4, arr1);
        int o2=obj1.countItem(2, arr1);

        assertEquals(2,o1);
        assertEquals(1,o2);
    }
    @Test
    public void test1() {
        int [] arr1= {};
        Function obj1 = new Function();
        int o1=obj1.countItem(3, arr1);

        assertEquals(0,o1);
    }
    @Test
    public void test2() {
        int [] arr1= {2,4,4,5,6};
        Function obj1 = new Function();
        int o1=obj1.countItem(2, arr1);
        assertEquals(1,o1);
    }

}
```



```
java.lang.Error: Unresolved compilation problem:
  Syntax error on token "s", delete this token

at Lab_7/Testing.Pro_2.test2(Pro_2.java:32)
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_2 {

    @Test
    public void test() {
        int [] arr1= {2,4,4,5,6};
        Function obj1 = new Function();
        int o1=obj1.countItem(4, arr1);
        int o2=obj1.countItem(2, arr1);

        assertEquals(2,o1);
        assertEquals(1,o2);
    }
    @Test
    public void test1() {
        int [] arr1= {};
        Function obj1 = new Function();
        int o1=obj1.countItem(3, arr1);

        assertEquals(0,o1);
    }
    @Test
    public void test2() {
        int [] arr1= {2,4,4,5,6};
        Function obj1 = new Function();
        int o1=obj1.countItem(2, arr1);
        assertEquals(0,o1);
    }

}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an empty array a[ ]	0
Test with v as a non-existent value and a non-empty array a[ ]	0
Test with v as an existent value and an empty array a[ ]	0
Test with v as an existent value and a non-empty array a[ ] where v exists more than 1	the number of occurrences of v in a[ ]
Test with v as an existent value and a non-empty array a[ ] where v exits only once	1

Boundary Value Analysis:

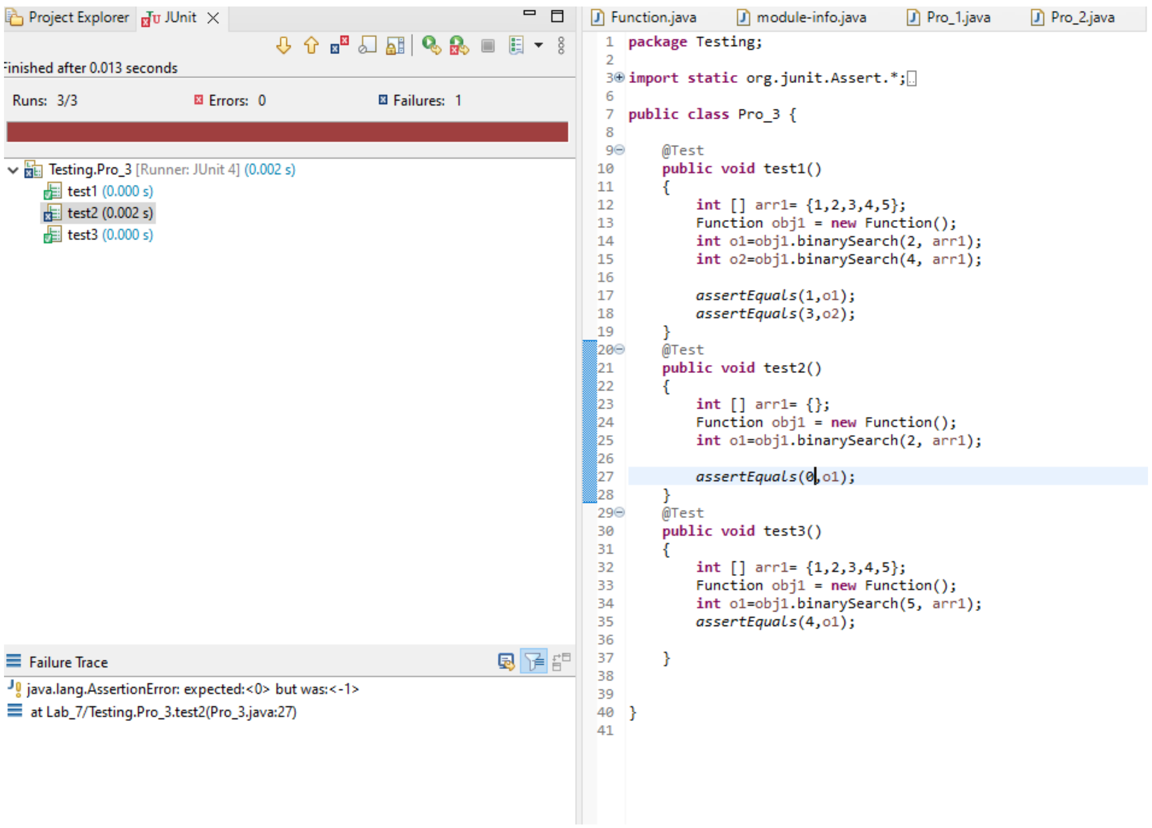
Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an empty array a[ ]	0
Test with v as a non-existent value and a non-empty array a[ ]	0
Test with v as an existent value and an empty array a[ ] of length 0	0
Test with v as an existent value and an array a[] of length 1, where v exists	1
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array	the number of occurrences of v in a[ ]
Test with v as an existent value and an array a[] of length greater than 1, where v exists in the middle of the array	the number of occurrences of v in a[ ]
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array	the number of occurrences of v in a[ ]

P3: The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

Assumption: the elements in the array a are sorted in non-decreasing order.

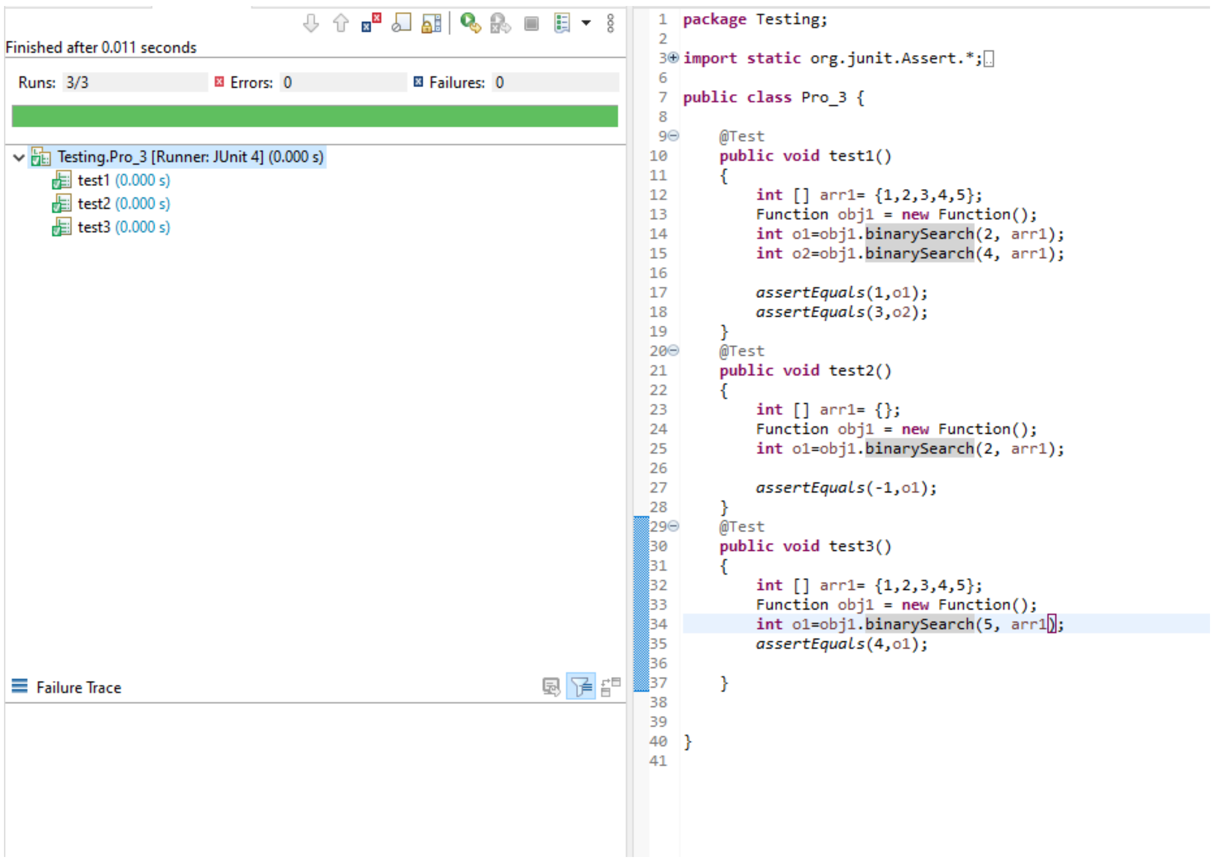
```
int binarySearch(int v, int a[])
{
    int lo,mid,hi;
    lo = 0;
    hi = a.length-1;
    while (lo <= hi)
    {
    mid = (lo+hi)/2;
    if (v == a[mid])
    return (mid);
    else if (v < a[mid])
    hi = mid-1;
    else
    lo = mid+1;

    }
return(-1);
}
```

Test Case in Eclipse:



```
Finished after 0.013 seconds
Runs: 3/3    Errors: 0    Failures: 1

Testing.Pro_3 [Runner: JUnit 4] (0.002 s)
    test1 (0.000 s)
    test2 (0.002 s)
    test3 (0.000 s)

Failure Trace
java.lang.AssertionError: expected:<0> but was:<-1>
at Lab_7/Testing.Pro_3.test2(Pro_3.java:27)
```

```
1  package Testing;
2
3⊕ import static org.junit.Assert.*;
6
7  public class Pro_3 {
8
9⊖      @Test
10      public void test1()
11      {
12          int [] arr1= {1,2,3,4,5};
13          Function obj1 = new Function();
14          int o1=obj1.binarySearch(2, arr1);
15          int o2=obj1.binarySearch(4, arr1);
16
17          assertEquals(1,o1);
18          assertEquals(3,o2);
19      }
20⊖     @Test
21      public void test2()
22      {
23          int [] arr1= {};
24          Function obj1 = new Function();
25          int o1=obj1.binarySearch(2, arr1);
26
27          assertEquals(0,o1);
28      }
29⊖     @Test
30      public void test3()
31      {
32          int [] arr1= {1,2,3,4,5};
33          Function obj1 = new Function();
34          int o1=obj1.binarySearch(5, arr1);
35          assertEquals(4,o1);
36
37      }
38
39
40  }
41
```



```
Finished after 0.011 seconds
Runs: 3/3    Errors: 0    Failures: 0

Testing.Pro_3 [Runner: JUnit 4] (0.000 s)
    test1 (0.000 s)
    test2 (0.000 s)
    test3 (0.000 s)

Failure Trace
```

```
1  package Testing;
2
3⊕ import static org.junit.Assert.*;
6
7  public class Pro_3 {
8
9⊖      @Test
10      public void test1()
11      {
12          int [] arr1= {1,2,3,4,5};
13          Function obj1 = new Function();
14          int o1=obj1.binarySearch(2, arr1);
15          int o2=obj1.binarySearch(4, arr1);
16
17          assertEquals(1,o1);
18          assertEquals(3,o2);
19      }
20⊖     @Test
21      public void test2()
22      {
23          int [] arr1= {};
24          Function obj1 = new Function();
25          int o1=obj1.binarySearch(2, arr1);
26
27          assertEquals(-1,o1);
28      }
29⊖     @Test
30      public void test3()
31      {
32          int [] arr1= {1,2,3,4,5};
33          Function obj1 = new Function();
34          int o1=obj1.binarySearch(5, arr1);
35          assertEquals(4,o1);
36
37      }
38
39
40  }
41
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
v=3, a=[1, 3, 5, 7, 9]	1
v=6, a=[2,4,6,8,9]	2
v=6, a=[6,9,25,37]	0
v=1, a=[2,4,6,8,9]	-1
v=15, a=[3,5,7,9,11]	-1

Boundary Value Analysis:

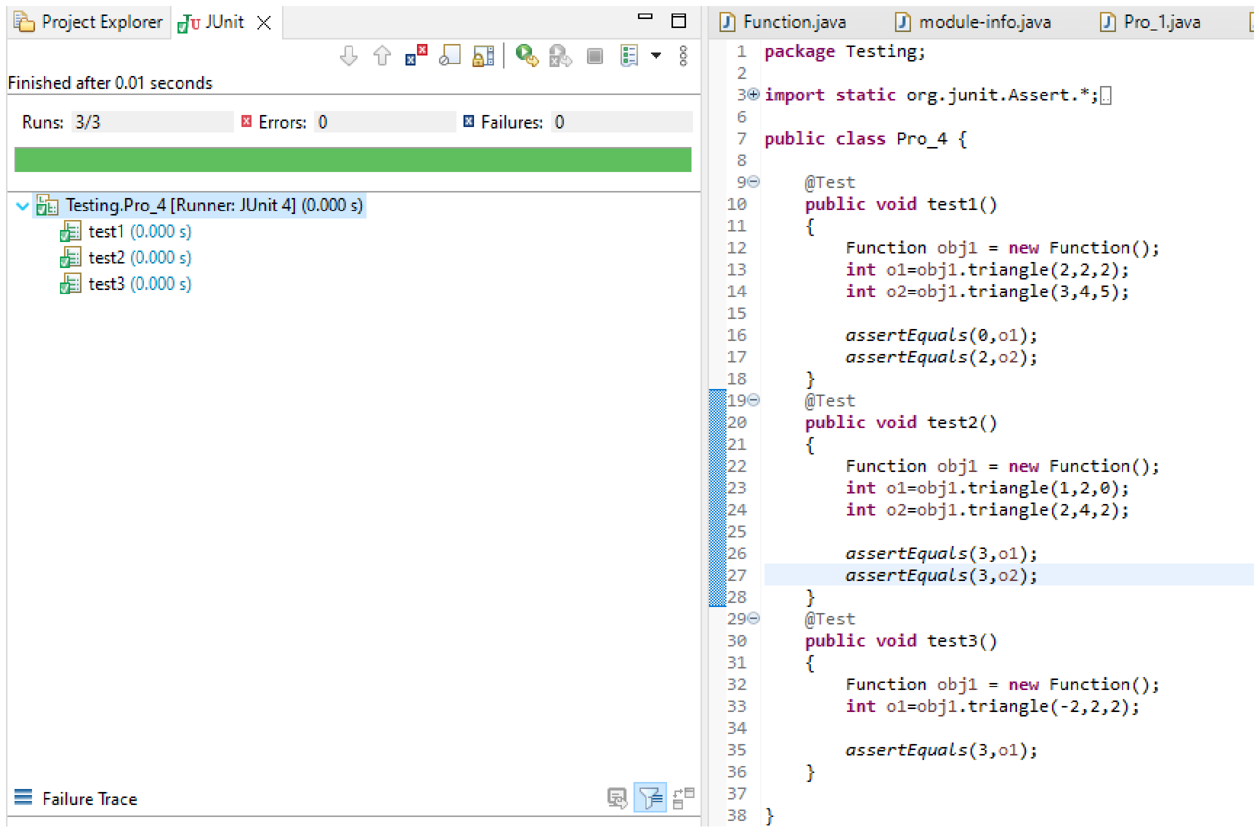
Tester Action and Input Data	Expected Outcome
v=5, a=[5]	0
v=3, a=[ ]	-1
v=1, a=[1,3,9]	0
v=9, a=[1,3,9]	2
v=3, a=[1,3,9]	1

P4: The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

```
    final int EQUILATERAL = 0;
    final int ISOSCELES = 1;
```

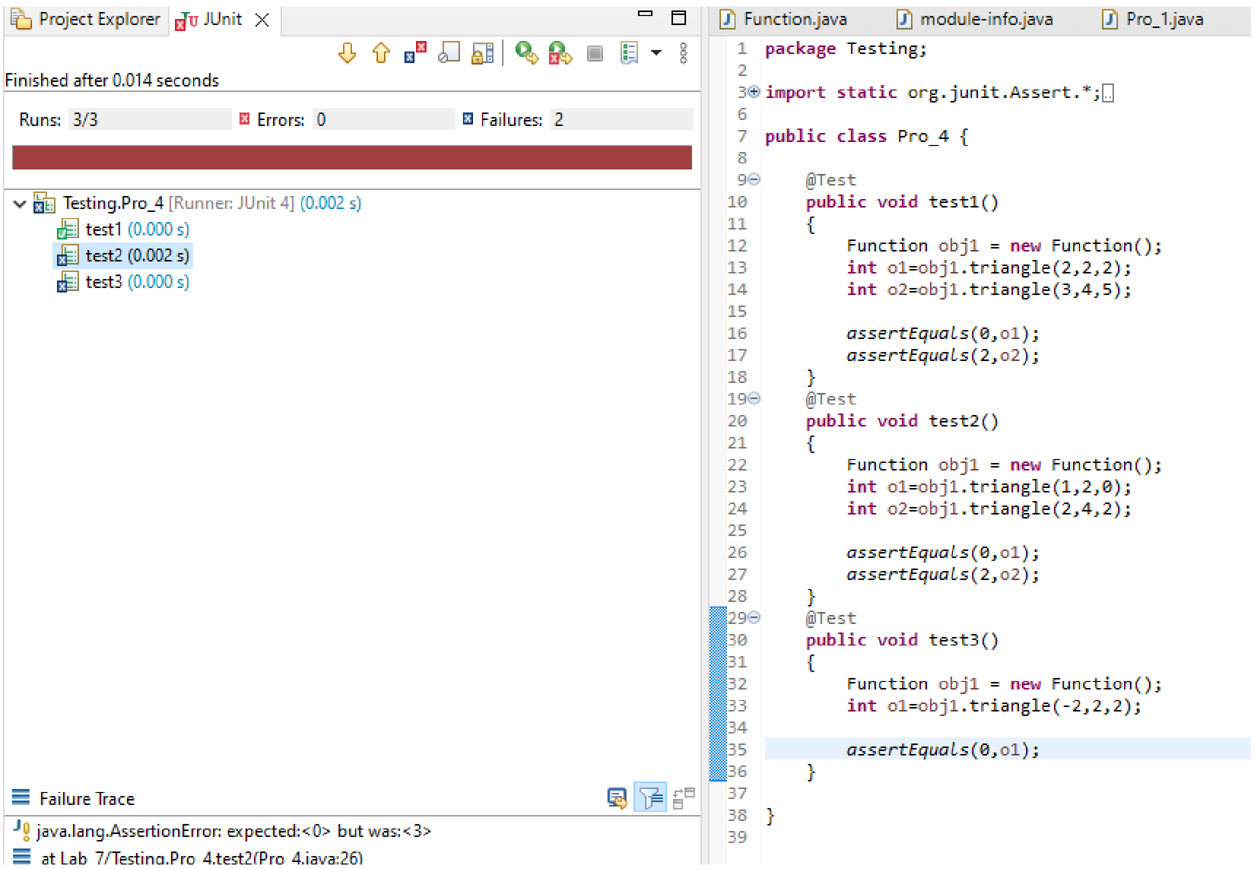
```
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
    if (a >= b+c || b >= a+c || c >= a+b)
    return(INVALID);
    if (a == b && b == c)
    return(EQUILATERAL);
    if (a == b || a == c || b == c)
    return(ISOSCELES);
    return(SCALENE);
}
```

Test Case in Eclipse:



```
Finished after 0.01 seconds
Runs: 3/3    Errors: 0    Failures: 0

Testing.Pro_4 [Runner: JUnit 4] (0.000 s)
    test1 (0.000 s)
    test2 (0.000 s)
    test3 (0.000 s)
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_4 {

    @Test
    public void test1()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(2,2,2);
        int o2=obj1.triangle(3,4,5);

        assertEquals(0,o1);
        assertEquals(2,o2);
    }
    @Test
    public void test2()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(1,2,0);
        int o2=obj1.triangle(2,4,2);

        assertEquals(3,o1);
        assertEquals(3,o2);
    }
    @Test
    public void test3()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(-2,2,2);

        assertEquals(3,o1);
    }
}
```



Project Explorer | JUnit | Function.java | module-info.java | Pro_1.java

```
Finished after 0.014 seconds

Runs: 3/3      Errors: 0      Failures: 2

Testing.Pro_4 [Runner: JUnit 4] (0.002 s)
    test1 (0.000 s)
    test2 (0.002 s)
    test3 (0.000 s)

Failure Trace
java.lang.AssertionError: expected:<0> but was:<3>
at Lab 7/Testing.Pro 4.test2(Pro 4.java:26)
```

```java
package Testing;

import static org.junit.Assert.*;

public class Pro_4 {

    @Test
    public void test1()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(2,2,2);
        int o2=obj1.triangle(3,4,5);

        assertEquals(0,o1);
        assertEquals(2,o2);
    }
    @Test
    public void test2()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(1,2,0);
        int o2=obj1.triangle(2,4,2);

        assertEquals(0,o1);
        assertEquals(2,o2);
    }
    @Test
    public void test3()
    {
        Function obj1 = new Function();
        int o1=obj1.triangle(-2,2,2);

        assertEquals(0,o1);
    }

}

```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
a=4, b=4, c=4	EQUILATERAL (0)
a=3, b=3, c=5	ISOSCELES (1)
a=2, b=2, c=6	ISOSCELES (1)
a=3, b=4, c=5	SCALENE (2)
a =-3, b=2, c=1	INVALID (3)
a=1, b=3, c=2	INVALID (3)
a=2, b=4, c=0	INVALID (3)

a=1, b=0, c=2	INVALID (3)
a=0, b=2, c=5	INVALID (3)
a=0, b=0, c=0	INVALID (3)

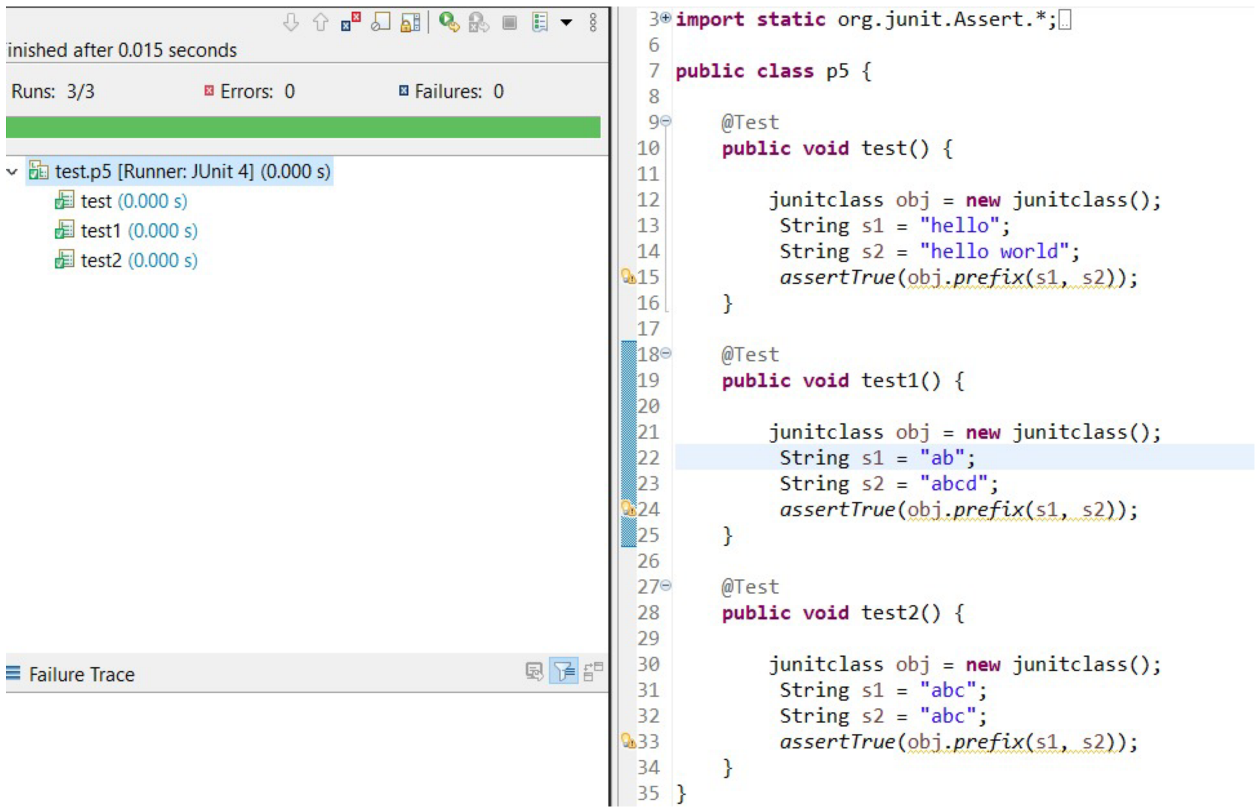
Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
a=0, b=0, c=0	INVALID (3)
a+b=c or, b+c=a or c+a=b	INVALID (3)
a=b=c=3	EQUILATERAL (0)
a=b=10 , c=25	ISOSCELES (1)
b=c=10 , a=25	ISOSCELES (1)
a=c=10 , b=25	ISOSCELES (1)
a>b+c	SCALENE (2)
b>a+c	SCALENE (2)
c>a+b	SCALENE (2)
a,b,c is maximum value of int	INVALID (3)
a,b,c is minimum value of int	INVALID (3)

P5: The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

```
public static boolean prefix(String s1,String
s2)
{
    if (s1.length() > s2.length())
    {
    return false;
    }
    for (int i = 0; i < s1.length(); i++)
    {
    if (s1.charAt(i) != s2.charAt(i))
    {
    return false;
    }
    }
    return true;
}
```

Test Case in Eclipse:



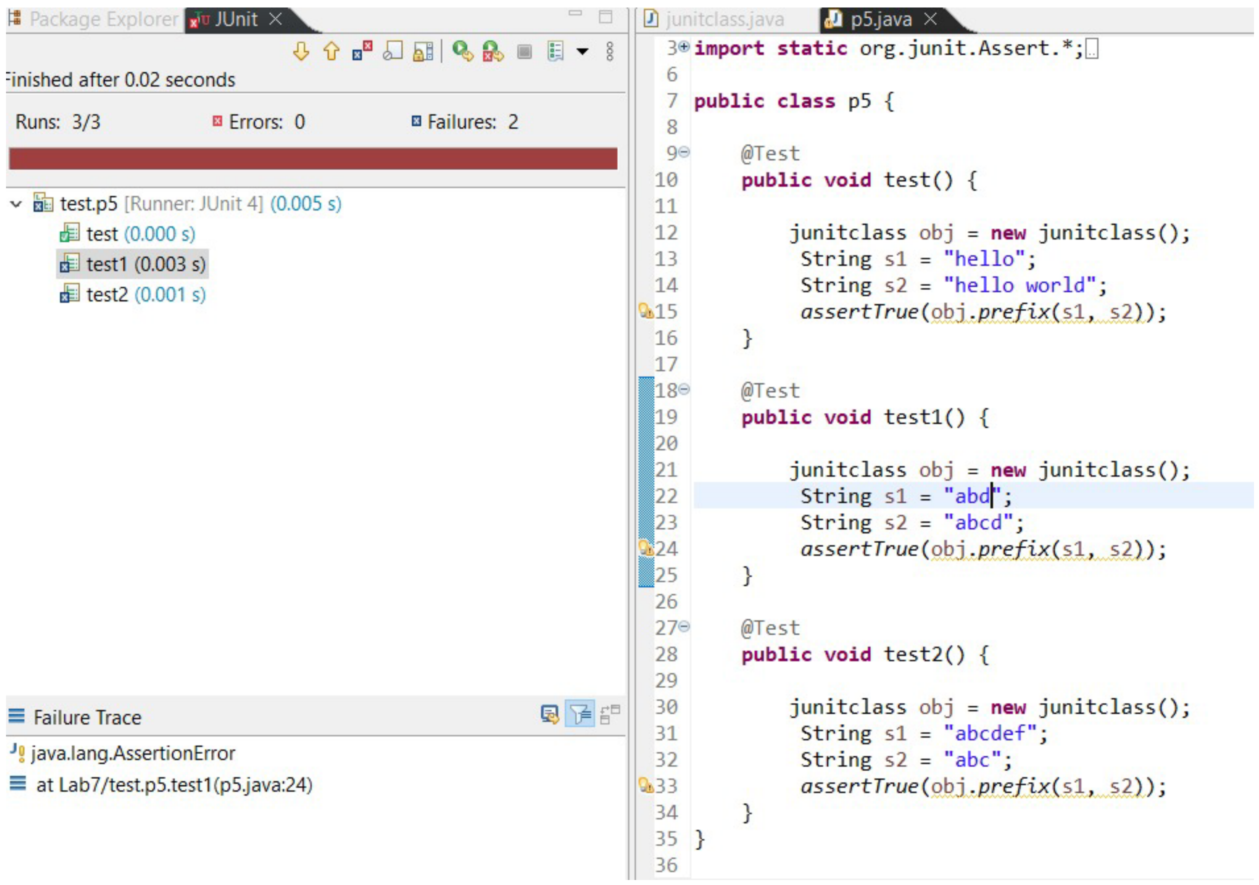
inished after 0.015 seconds

Runs: 3/3 Errors: 0 Failures: 0

```
test.p5 [Runner: JUnit 4] (0.000 s)
    test (0.000 s)
    test1 (0.000 s)
    test2 (0.000 s)
```

Failure Trace

```java
import static org.junit.Assert.*;

public class p5 {

    @Test
    public void test() {

        junitclass obj = new junitclass();
         String s1 = "hello";
         String s2 = "hello world";
         assertTrue(obj.prefix(s1, s2));
    }

    @Test
    public void test1() {

        junitclass obj = new junitclass();
         String s1 = "ab";
         String s2 = "abcd";
         assertTrue(obj.prefix(s1, s2));
    }

    @Test
    public void test2() {

        junitclass obj = new junitclass();
         String s1 = "abc";
         String s2 = "abc";
         assertTrue(obj.prefix(s1, s2));
    }
}
```



Finished after 0.02 seconds

Runs: 3/3 Errors: 0 Failures: 2

```
test.p5 [Runner: JUnit 4] (0.005 s)
    test (0.000 s)
    test1 (0.003 s)
    test2 (0.001 s)
```

Failure Trace

```
java.lang.AssertionError
at Lab7/test.p5.test1(p5.java:24)
```

```java
import static org.junit.Assert.*;

public class p5 {

    @Test
    public void test() {

        junitclass obj = new junitclass();
         String s1 = "hello";
         String s2 = "hello world";
         assertTrue(obj.prefix(s1, s2));
    }

    @Test
    public void test1() {

        junitclass obj = new junitclass();
         String s1 = "abd";
         String s2 = "abcd";
         assertTrue(obj.prefix(s1, s2));
    }

    @Test
    public void test2() {

        junitclass obj = new junitclass();
         String s1 = "abcdef";
         String s2 = "abc";
         assertTrue(obj.prefix(s1, s2));
    }
}

```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
s1 = "hello", s2 = "hello world"	True
s1 = "hello", s2 = " he"	False
s1 = "a", s2 = "abc"	True
s1 = " ", s2 = "abc"	False
s1 = "world", s2 = "hello world"	False
s1 = "abd", s2 = "abcdef"	False

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
s1 = " ", s2 = "xyz"	False
s1 = "ab", s2 = "abcd"	True
s1 = "abcd", s2 = "a"	False
s1 = "hello", s2 = "helloolol"	True
s1 = "a", s2 = "b"	False
s1 = "a", s2 = "a"	True
s1 = "", s2 = ""	True

P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled. Determine the following for the above program:

a) Identify the equivalence classes for the system
b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case
would cover which equivalence class.
(Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence
classes)
c) For the boundary condition $A + B > C$ case (scalene triangle), identify test cases to verify the
boundary.
d) For the boundary condition $A = C$ case (isosceles triangle), identify test cases to verify the
boundary.
e) For the boundary condition $A = B = C$ case (equilateral triangle), identify test cases to verify the
boundary.
f) For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.
g) For the non-triangle case, identify test cases to explore the boundary.
h) For non-positive input, identify test points.

Ans :
Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
a=4, b=4, c=4	EQUILATERAL
a=3, b=3, c=5	ISOSCELES
a=2, b=2, c=6	ISOSCELES
a=3, b=4, c=5	SCALENE Right angle Triangle
a=6 , b=8, c=10	SCALENE Right angle Triangle
a= -3 , b=2, c=1	INVALID
a=2, b=4, c=0	Not a Triangle
a=4,b=5,c=10	SCALENE

Test Case:
Invalid inputs:
- (0,0,0) a+b=c , a+c=b, b+c=a;
- (-1,0,1) a+c=b

Equilateral triangles:
- a = b = c = 1,
- a = b = c = 100

Isosceles triangles:
- a = b = 10, c = 5
- a = c = 10, b = 3
- b = c = 10, a = 6

Scalene triangles:
- a = 4, b = 5, c = 6
- a = 10, b = 11, c = 13

Right angled triangle:

    a = 3, b = 4, c = 5
    a = 6, b = 8, c = 10

Non-triangle:

    a = 1, b = 2, c = 3

Non-positive input:

    a = -1, b = -2, c = -3

c) Boundary condition A + B > C:

    a = Integer.MAX_VALUE,
    b = Integer.MAX_VALUE,
    c = 1
    a = 100
    b = 100
    c = 100

d) Boundary condition A = C:

    a = Integer.MAX_VALUE,
    b = 2,
    c = Integer.MAX_VALUE
    a = 10
    b = 2
    c = 10

e) Boundary condition A = B = C:

    a = Integer.MAX_VALUE,
    b = Integer.MAX_VALUE,
    c = Integer.MAX_VALUE
    a = Double.MAX_VALUE,
    b = Double.MAX_VALUE,
    c = Double.MAX_VALUE

f) Boundary condition $A^2 + B^2 = C^2$:

a = Integer.MAX_VALUE,
b = Integer.MAX_VALUE,
c = Integer.MAX_VALUE
a = Double.MAX_VALUE,
b = Double.MAX_VALUE,
c = Math.sqrt(Math.pow(Double.MAX_VALUE, 2) + Math.pow(Double.MAX_VALUE, 2))

g) Non-triangle:

a = 1, b = 2, c = 4
a = 2, b = 4, c = 8

h) Non-positive input:

a = -1, b = -2, c = -3
a=-1 , b = 3, c=4

Section B

The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code and so the focus is on creating test sets that satisfy some particular coverage criterion.

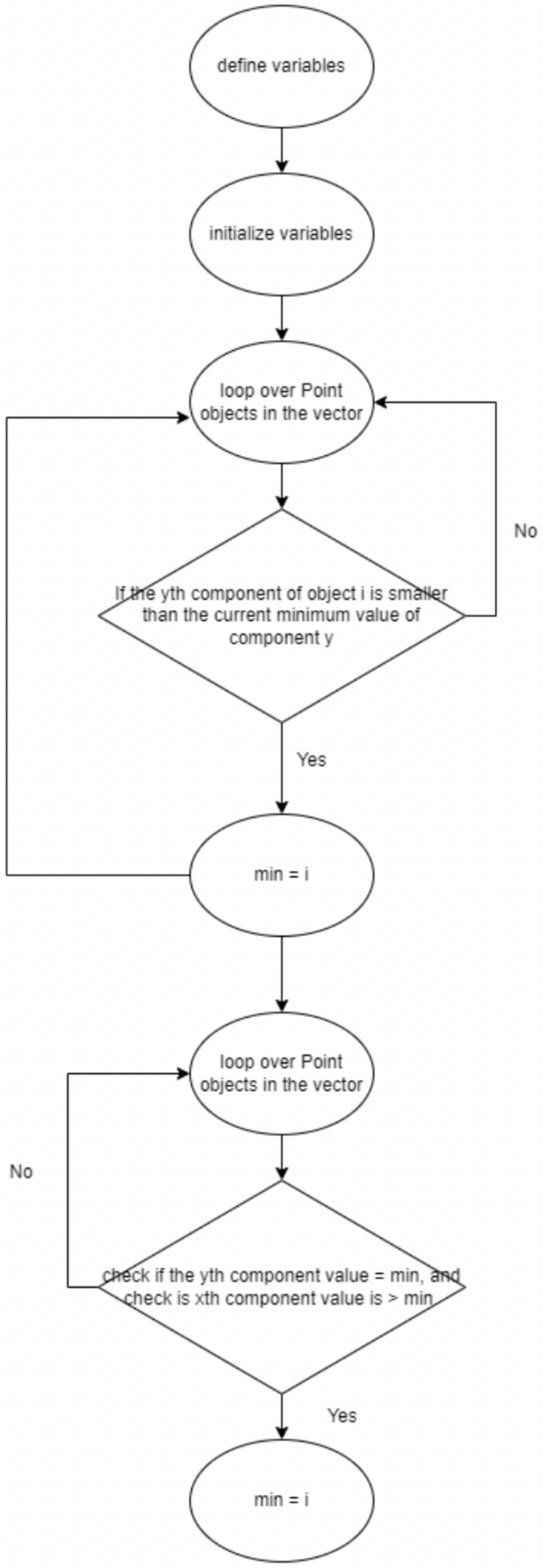
```
Vector doGraham (Vector p)
{
     int i,j,min,M;

     Point t;
     min = 0;

     // search for minimum:
     for (i=1; i < p.size (); ++i)
     {
          if ((Point) p.get (i)).y <
                                        ((Point)p. get(min)).y)
          {
               min = i;
          }
     }
     //continue along the values with same y component
     for(i=0; i < p.size (); ++i)
     {
          if (( ((Point) p.get (i)).y ==
                              ((Point) p.get (min)).y ) &&
                ((((Point) p.get(i)).x >
                                   ((Point) p.get (min)).x ))
          {
               min=i;
```

```
                }
        }
}
```
For the given code fragment you should carry out the following activities.

1. Convert the Java code comprising the beginning of the doGraham method into a control flow graph (CFG).
2. Construct test sets for your flow graph that are adequate for the following criteria:
    a. Statement Coverage.
    b. Branch Coverage.
    c. Basic Condition Coverage.

Ans:

Control Flow Graph (CFG):


define variables
initialize variables
loop over Point objects in the vector
No
If the yth component of object i is smaller than the current minimum value of component y
Yes
min = i
loop over Point objects in the vector
No
check if the yth component value = min, and check is xth component value is > min
Yes
min = i

Test sets for each coverage criterion:

a. Statement Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}

b. Branch Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}
- Test 3: p = {new Point(0, 0), new Point(1, 0), new Point(1, 1)}

c. Basic Condition Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}
- Test 3: p = {new Point(0, 0), new Point(1, 0), new Point(1, 1)}
- Test 4: p = {new Point(0, 0), new Point(1, 0), new Point(0, 1)}
- Test 5: p = {new Point(0, 0), new Point(0, 1), new Point(1, 1)}