IT314 Software Engineering

Lab Session 7

From:

Team No. 31 | Group 6

Name:

Jaimin Baurasi 202001403

Section - A

Equivalence classes:

- Valid dates: The input triple (day, month, year) representing a valid date in the Gregorian calendar, such as (3, 4, 1995).
- Invalid dates: The input triple (day, month, year) that represent an invalid date, such as (31, 2, 2022) or (29, 2, 1900).
- Out-of-range dates: The input triple (day, month, year) that are outside the allowed ranges, such as (0, 5, 2010) or (15, 13, 2005). Based on these equivalence classes, we can design the following test cases:

Tester Action and Input Data Expected Outcome:

Out-of-range dates:

- Calculate the previous date for (0, 5, 2010) Invalid date
- Calculate the last date for (15, 13, 2005) Invalid date
- Calculate the last date for (31, 12, 1899) Invalid date

Invalid dates:

- Calculate the previous date for (29, 2, 2022) Invalid date
- Calculate the last date for (31, 4, 2010) Invalid date
- Calculate the last date for (30, 2, 2000) Invalid date

Valid dates:

- Calculate the previous date for (15, 10, 2022) 14, 10, 2022
- Calculate the last date for (1, 1, 2015) 31, 12, 2014
- Calculate the last date for (31, 3, 2000) 30, 3, 2000

Boundary Value Analysis:
Using boundary value analysis, we can identify the following boundary test cases:

- The earliest possible date: (1, 1, 1900)
- The latest possible date: (31, 12, 2015)
- The earliest day of each month: (1, 1, 2000), (1, 2, 2000), (1, 3, 2000),..., (1, 12, 2000)
- The latest day of each month: (31, 1, 2000), (28, 2, 2000), (31, 3, 2000),..., (31, 12, 2000)
- Leap year day: (29, 2, 2000)
- Invalid leap year day: (29, 2, 1900)
- One day before the earliest date: (31, 12, 1899)

- One day after the latest date: (1, 1, 2016)
- Based on these boundary test cases, we can design the following test cases:
- Tester Action and Input Data Expected Outcome

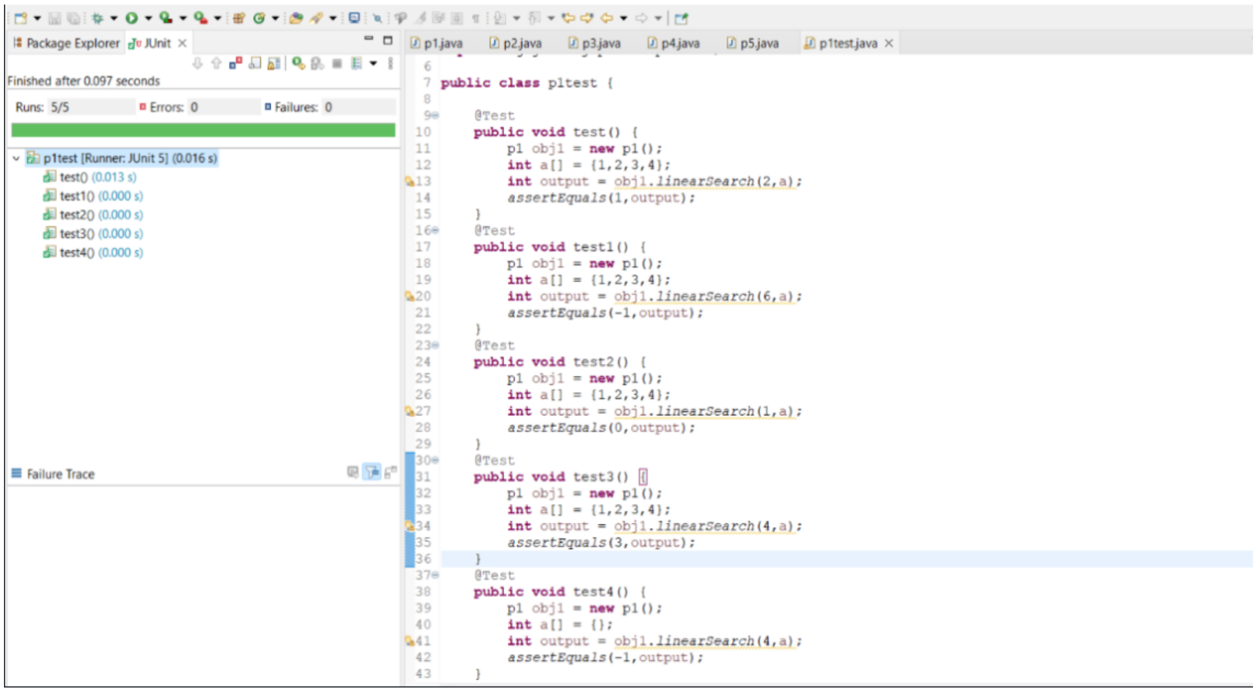
Boundary Test Cases:

- Calculate the previous date for (1, 1, 1900) Invalid date
- Calculate the last date for (31, 12, 2015) 30, 12, 2015
- Calculate the last date for (1, 1, 2000) 31, 12, 1999
- Calculate the last date for (31, 1, 2000) 30, 1, 2000
- Calculate the previous date for (29, 2, 2000) 28, 2, 2000
- Calculate the last date for (29, 2, 1900) Invalid date
- Calculate previous

Program 1:

The function linear search 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);
}
```



```
Finished after 0.097 seconds
Runs: 5/5    Errors: 0    Failures: 0

p1test [Runner: JUnit 5] (0.016 s)
    test() (0.013 s)
    test1() (0.000 s)
    test2() (0.000 s)
    test3() (0.000 s)
    test4() (0.000 s)
```

```java
public class p1test {

    @Test
    public void test() {
        p1 obj1 = new p1();
        int a[] = {1,2,3,4};
        int output = obj1.linearSearch(2,a);
        assertEquals(1,output);
    }
    @Test
    public void test1() {
        p1 obj1 = new p1();
        int a[] = {1,2,3,4};
        int output = obj1.linearSearch(6,a);
        assertEquals(-1,output);
    }
    @Test
    public void test2() {
        p1 obj1 = new p1();
        int a[] = {1,2,3,4};
        int output = obj1.linearSearch(1,a);
        assertEquals(0,output);
    }
    @Test
    public void test3() {
        p1 obj1 = new p1();
        int a[] = {1,2,3,4};
        int output = obj1.linearSearch(4,a);
        assertEquals(3,output);
    }
    @Test
    public void test4() {
        p1 obj1 = new p1();
        int a[] = {};
        int output = obj1.linearSearch(4,a);
        assertEquals(-1,output);
    }
```

Equivalence Partioning

Tester Action and Input Data	Expected Outcome
v is an invalid number, and a is empty v = NULL, a = [ ]	-1
v is a valid number, and a is empty v = 8, a = [ ]	-1
v is an invalid number, and a is non-empty v = NULL, a = [2, 4, 7]	-1
v is a valid number, a is non-empty, and v is not present v = 2, a = [1, 3, 6]	-1

v is a valid number, a is non-empty, and v is present v = 6, a = [2, 4, 6, 7	Index of v 2

Boundary value analysis

Tester Action and Input Data	Expected Outcome
v is a valid number, a is non-empty, and v is present at 1st index v = 7, a = [7, 6, 4]	0
v is a valid number, a is non-empty, and v is present at the last index v = 4, a = [7, 6, 4]	a.size() -1
v is a valid number, a is non-empty, and v is not present v = 9, a = [7, 6, 4]	-1

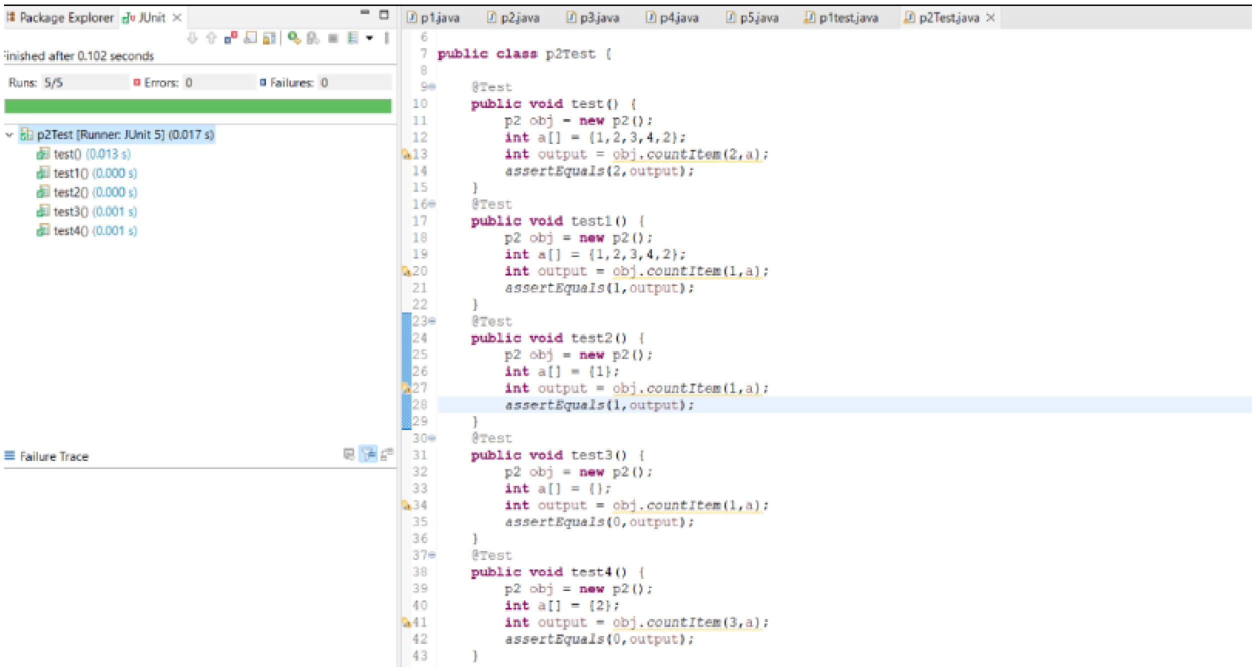
Program 2:

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);
}
```



```
Package Explorer   JUnit
Finished after 0.102 seconds
Runs: 5/5    Errors: 0    Failures: 0

p2Test [Runner: JUnit 5] (0.017 s)
    test() (0.013 s)
    test1() (0.000 s)
    test2() (0.000 s)
    test3() (0.001 s)
    test4() (0.001 s)

Failure Trace
```

```
p1.java  p2.java  p3.java  p4.java  p5.java  p1test.java  p2Test.java

public class p2Test {

    @Test
    public void test() {
        p2 obj = new p2();
        int a[] = {1,2,3,4,2};
        int output = obj.countItem(2,a);
        assertEquals(2,output);
    }
    @Test
    public void test1() {
        p2 obj = new p2();
        int a[] = {1,2,3,4,2};
        int output = obj.countItem(1,a);
        assertEquals(1,output);
    }
    @Test
    public void test2() {
        p2 obj = new p2();
        int a[] = {1};
        int output = obj.countItem(1,a);
        assertEquals(1,output);
    }
    @Test
    public void test3() {
        p2 obj = new p2();
        int a[] = {};
        int output = obj.countItem(1,a);
        assertEquals(0,output);
    }
    @Test
    public void test4() {
        p2 obj = new p2();
        int a[] = {2};
        int output = obj.countItem(3,a);
        assertEquals(0,output);
    }
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
v is an invalid number, and a is empty v = NULL, a = [ ]	0
v is a valid number, and a is empty v = 7, a = [ ]	0

v is an invalid number, and a is non-empty v = NULL, a = [7, 6, 4]	0
v is a valid number, a is non-empty, and v is present one time v = 7, a = [7, 6, 4]	1
v is a valid number, a is non-empty, and v is present multiple times v = 7, a = [7, 7, 4]	Number of occurrences of v 2

Boundary Value analysis

Tester Action and Input Data	Tester Action and Input Data
v is a valid number, a is non-empty, and v is present at 1st index v = 7, a = [7, 6, 4]	1
v is a valid number, a is non-empty, and v is present at the last index v = 4, a = [7, 6, 4]	1
v is a valid number, a is non-empty, and v is not present v = 5, a = [7, 6, 4]	0
v is a valid number, a is non-empty, and v is present once v = 6, a = [7, 6, 4]	1

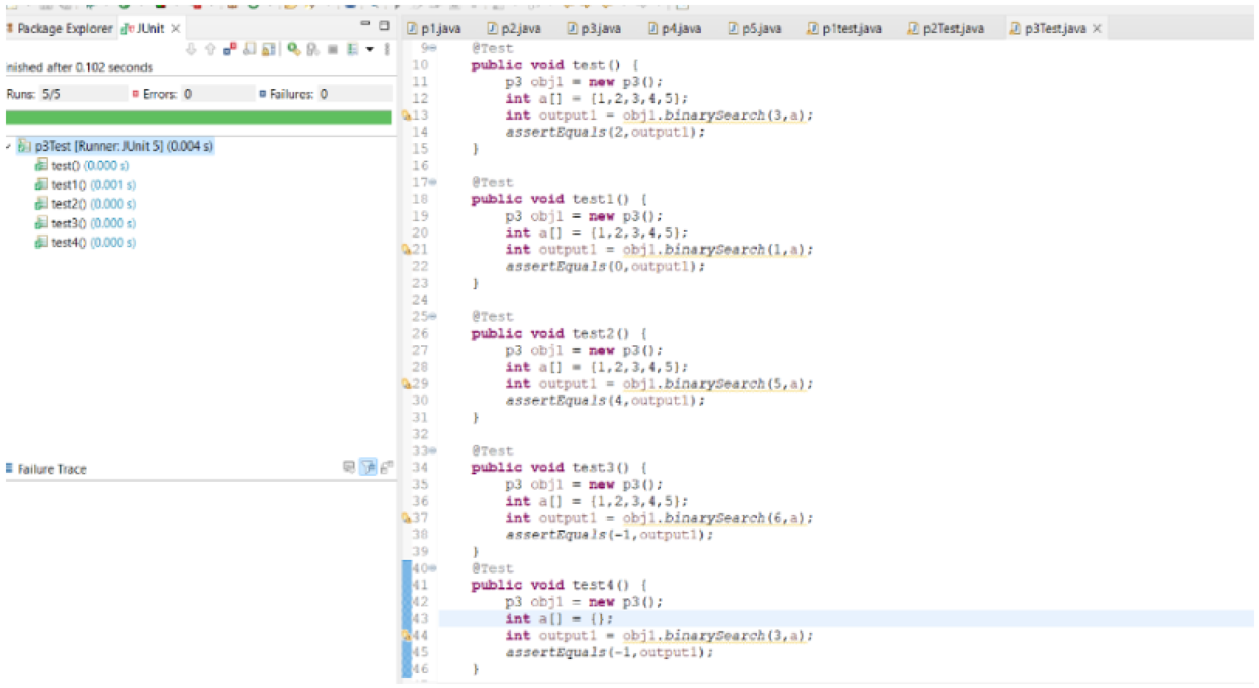
v is a valid number, a is non-empty, and v is present multiple times v = 6, a = [6, 6, 6, 4]	Number of occurrences of v 3

Program 3:

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 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);
```

}



Package Explorer | JUnit

nished after 0.102 seconds

Runs: 5/5 Errors: 0 Failures: 0

```
p3Test [Runner: JUnit 5] (0.004 s)
    test() (0.000 s)
    test1() (0.001 s)
    test2() (0.000 s)
    test3() (0.000 s)
    test4() (0.000 s)
```

Failure Trace

p1.java | p2.java | p3.java | p4.java | p5.java | p1test.java | p2Test.java | p3Test.java

```java
    @Test
    public void test() {
        p3 obj1 = new p3();
        int a[] = {1,2,3,4,5};
        int output1 = obj1.binarySearch(3,a);
        assertEquals(2,output1);
    }

    @Test
    public void test1() {
        p3 obj1 = new p3();
        int a[] = {1,2,3,4,5};
        int output1 = obj1.binarySearch(1,a);
        assertEquals(0,output1);
    }

    @Test
    public void test2() {
        p3 obj1 = new p3();
        int a[] = {1,2,3,4,5};
        int output1 = obj1.binarySearch(5,a);
        assertEquals(4,output1);
    }

    @Test
    public void test3() {
        p3 obj1 = new p3();
        int a[] = {1,2,3,4,5};
        int output1 = obj1.binarySearch(6,a);
        assertEquals(-1,output1);
    }
    @Test
    public void test4() {
        p3 obj1 = new p3();
        int a[] = {};
        int output1 = obj1.binarySearch(3,a);
        assertEquals(-1,output1);
    }
```

Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
String s1 empty and String s2 empty s1=””, s2=””	true
String s1 empty and String s2 non-empty s1=””, s2=”abc”	true
String s1 non-empty and String s2 empty s1=”abc”, s2=””	false
String s1 non-empty, String s2 non-empty, and s1.size() > s2.size() s1=”abc”, s2=”a”	false
String s1 non-empty, String s2 non-empty, s1.size() <=	true

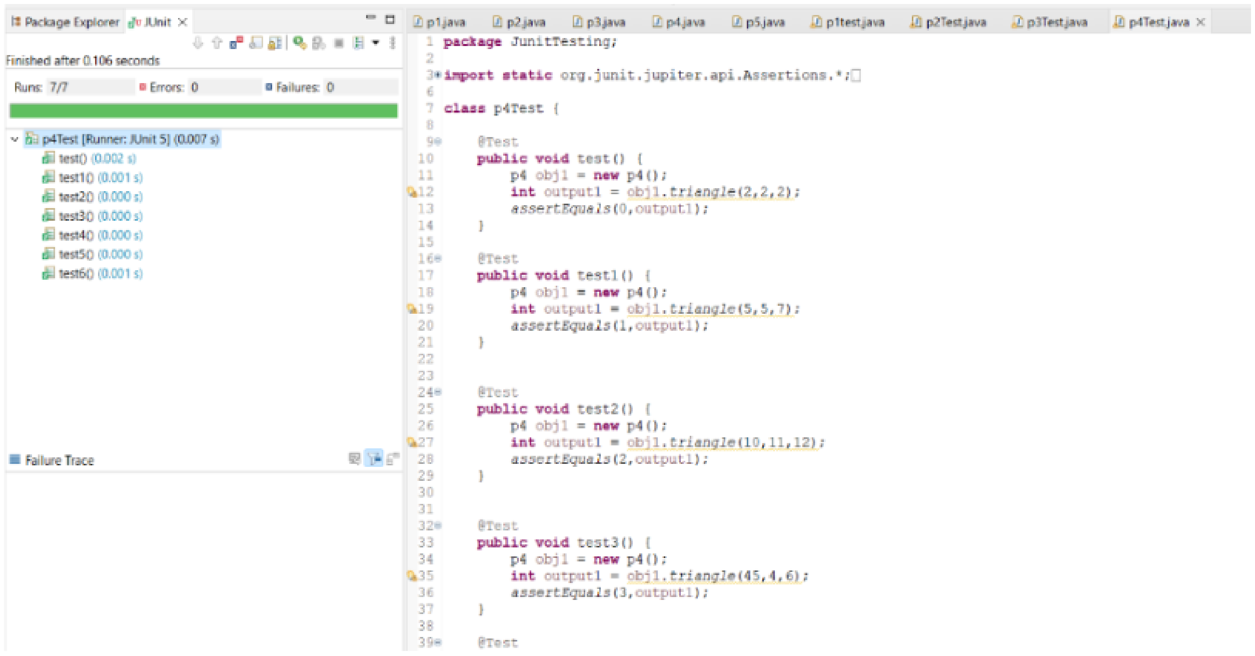
s2.size() and s1 is prefix of s2 s1=”abc”, s2=”abcd”	
String s1 non-empty, String s2 non-empty, s1.size() <= s2.size() and s1 is not a prefix of s2 s1=”abc”, s2=”defg”	false

Program 4:

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);
}
```



```
Finished after 0.106 seconds
Runs: 7/7    Errors: 0    Failures: 0

p4Test [Runner: JUnit 5] (0.007 s)
    test() (0.002 s)
    test1() (0.001 s)
    test2() (0.000 s)
    test3() (0.000 s)
    test4() (0.000 s)
    test5() (0.000 s)
    test6() (0.001 s)
```

```java
package JunitTesting;

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

class p4Test {

    @Test
    public void test() {
        p4 obj1 = new p4();
        int output1 = obj1.triangle(2,2,2);
        assertEquals(0,output1);
    }

    @Test
    public void test1() {
        p4 obj1 = new p4();
        int output1 = obj1.triangle(5,5,7);
        assertEquals(1,output1);
    }


    @Test
    public void test2() {
        p4 obj1 = new p4();
        int output1 = obj1.triangle(10,11,12);
        assertEquals(2,output1);
    }


    @Test
    public void test3() {
        p4 obj1 = new p4();
        int output1 = obj1.triangle(45,4,6);
        assertEquals(3,output1);
    }

    @Test
```

Equilvalence Partioning

Tester action and input data	Expected outcome
Invalid triangle (a+b<=c) a=3, b=4, c=11	Invalide
Valid equilateral triangle (a=b=c) a=4, b=4, c=4	equilateral
Valid isosceles triangle (a=b<c) a=4, b=4, c=5	isoceles
Valid scalene triangle (a<b<c) a=3, b=4, c=6	scalene

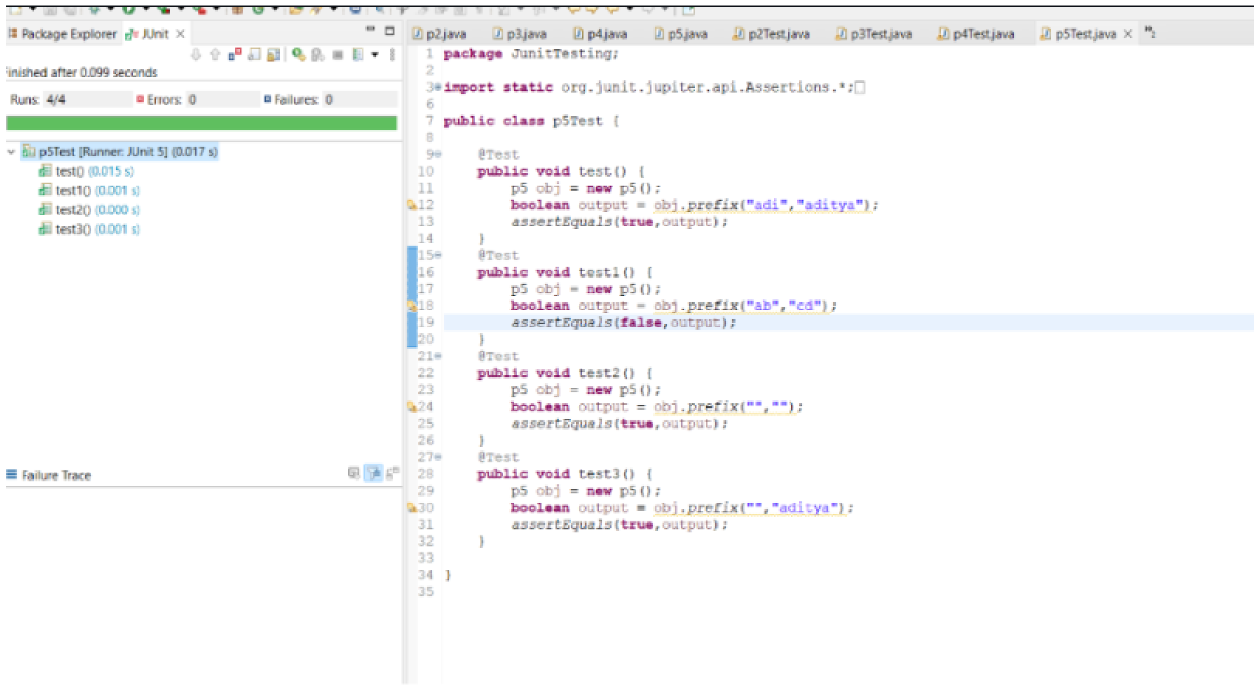
Boundary value analysis

Tester Action and Input Data	Expected Outcome
Invalid triangle (a+b<=c) a=3, b=4, c=10	invalid
Invalid triangle (a+c<b) a=3, b=9, c=5	invalid
Invalid triangle (b+c<a) a=12, b=4, c=6	invalid
Valid equilateral triangle (a=b=c) a=3, b=3, c=3	equilateral
Valid isosceles triangle (a=b<c) a=3, b=3, c=4	isoceles
Valid isosceles triangle (a=c<b) a=3, b=4, c=3	isoceles
Valid isosceles triangle (b=c<a) a=4, b=3, c=3	isoceles
Valid scalene triangle (a<b<c) a=3, b=4, c=6	scalene

Program 5:

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 0;
    }
    for (int i = 0; i < s1.length(); i++)
    {
    if (s1.charAt(i) != s2.charAt(i))
    {
    return 0;
    }
    }
    return 1;
}
```



```
Finished after 0.099 seconds
Runs: 4/4    Errors: 0    Failures: 0

p5Test [Runner: JUnit 5] (0.017 s)
    test() (0.015 s)
    test1() (0.001 s)
    test2() (0.000 s)
    test3() (0.001 s)
```

```java
package JunitTesting;

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

public class p5Test {

    @Test
    public void test() {
        p5 obj = new p5();
        boolean output = obj.prefix("adi","aditya");
        assertEquals(true,output);
    }
    @Test
    public void test1() {
        p5 obj = new p5();
        boolean output = obj.prefix("ab","cd");
        assertEquals(false,output);
    }
    @Test
    public void test2() {
        p5 obj = new p5();
        boolean output = obj.prefix("","");
        assertEquals(true,output);
    }
    @Test
    public void test3() {
        p5 obj = new p5();
        boolean output = obj.prefix("","aditya");
        assertEquals(true,output);
    }

}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
String s1 empty and String s2 empty s1=””, s2=””	true
String s1 empty and String s2 non-empty s1=””, s2=”abc”	true
String s1 non-empty and String s2 empty s1=”abc”, s2=””	false
String s1 non-empty, String s2 non-empty, and s1.size() > s2.size() s1=”abc”, s2=”a”	false
String s1 non-empty, String s2 non-empty, s1.size() <= s2.size() and s1 is prefix of s2 s1=”abc”, s2=”abcd”	true
String s1 non-empty, String s2 non-empty, s1.size() <= s2.size() and s1 is not a prefix of s2 s1=”abc”, s2=”defg”	false

Boundary value analysis

Tester Action and Input Data	Expected Outcome
Empty string s1 and s2 s1=””, s2=””	true
Empty string s1 and	true

non-empty s2 s1="", s2="software"	
Non-empty s1 is not a prefix of s2 s1="cloud", s2="software"	false
Non-empty s1 is longer than s2 s1="software", s2="soft"	false

Program 6:

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

Equivalence Classes:
EC1: All sides are positive, real numbers.
EC2: One or more sides are negative or zero.
EC3: The sum of the lengths of any two sides is less than or equal to the length of the remaining side (impossible lengths).

EC4: The sum of the lengths of any two sides is greater than the length of the remaining side (possible lengths).

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 ensure that the identified set of test cases cover all identified equivalence classes)

Test cases:
TC1 (EC1): A=3, B=4, C=5 (right-angled triangle)
TC2 (EC1): A=5, B=5, C=5 (equilateral triangle)
TC3 (EC1): A=5, B=6, C=7 (scalene triangle)
TC4 (EC1): A=5, B=5, C=7 (isosceles triangle)
TC5 (EC2): A=-2, B=4, C=5 (invalid input)
TC6 (EC2): A=0, B=4, C=5 (invalid input)

c) For the boundary condition $A + B > C$ case (scalene triangle), identify test cases to verify the boundary.

Test cases for the boundary condition $A + B > C$:
TC7 (EC4): A=2, B=3, C=6 (sum of A and B is equal to C)

d) For the boundary condition $A = C$ case (isosceles triangle), identify test cases to verify the boundary.

Test cases for the boundary condition A = C:
TC8 (EC4): A=5, B=6, C=5 (A equals to C)

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

Test cases for the boundary condition A = B = C:
TC9 (EC4): A=1, B=1, C=1 (all sides are equal)

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

Test cases for the boundary condition A^2 + B^2 = C^2:
TC10 (EC4): A=3, B=4, C=5 (right-angled triangle)

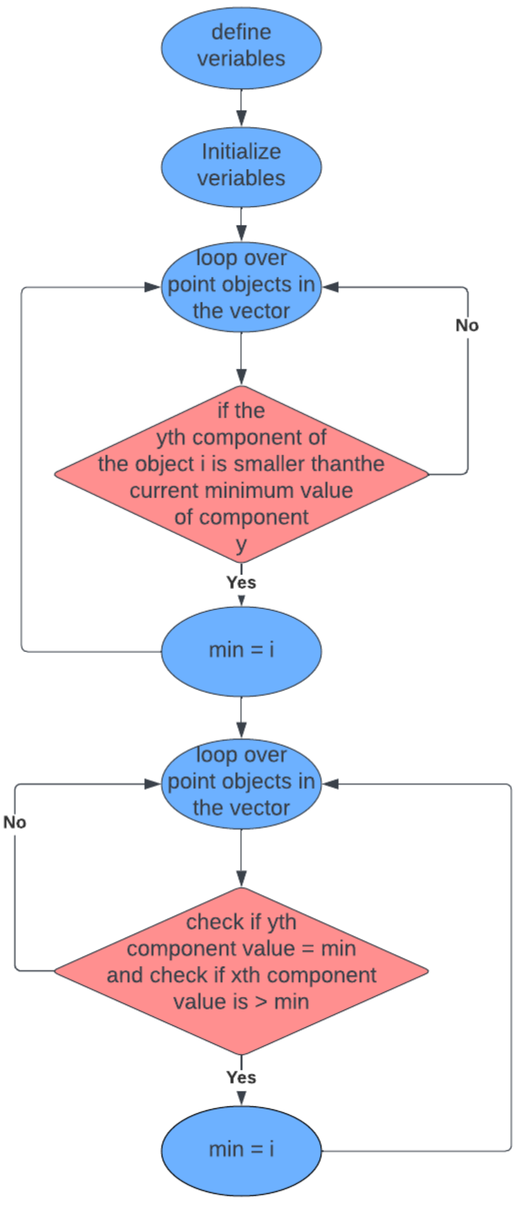
g) For the non-triangle case, identify test cases to explore the boundary.

Test cases for the non-triangle case:
TC11 (EC3): A=2, B=2, C=4 (sum of A and B is less than C)

h) For non-positive input, identify test points.
Test points for non-positive input:
TP1 (EC2): A=0, B=4, C=5 (invalid input)
TP2 (EC2): A=-2, B=4, C=5 (invalid input)

Note: Test cases TC1 to TC10 covers all identified equivalence classes.

Control flow diagram:


define veriables
Initialize veriables
loop over point objects in the vector
if the yth component of the object i is smaller thanthe current minimum value of component y
No
Yes
min = i
loop over point objects in the vector
check if yth component value = min and check if xth component value is > min
No
Yes
min = i

Test sets:
a) Statement coverage test sets: To achieve statement coverage, we need to make sure that every statement in the code is executed at least once.

- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component
- Test 4: p = vector with two points with different y components
- Test 5: p = vector with three or more points with different y components
- Test 6: p = vector with three or more points with the same y component

b) Branch coverage test sets: To achieve branch coverage, we need to make sure that every possible branch in the code is taken at least once
- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component
- Test 4: p = vector with two points with different y components
- Test 5: p = vector with three or more points with different y components, and none of them have the same x component
- Test 6: p = vector with three or more points with the same y component, and some of them have the same x component
- Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

c) Basic condition coverage test sets: To achieve basic condition coverage, we need to make sure that every basic condition in the code (i.e., every Boolean subexpression) is evaluated as both true and false at least once

- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component, and the first point has a smaller x component
- Test 4: p = vector with two points with the same y component, and the second point has a smaller x component
- Test 5: p = vector with two points with different y components
- Test 6: p = vector with three or more points with different y components, and none of them have the same x component
- Test 7: p = vector with three or more points with the same y component, and some of them have the same x component .
- Test 8: p = vector with three or more points with the same y component, and all of them have the same x component.