IT-314 Software Engineering

Lab-7 Testing

Name: Shrey Shah
ID: 202001431

Section - A

Consider a program for determining the previous date. Its input is a 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?

Write a set of test cases (i.e.,test suite) – specific set of data – to properly test the programs. Your test suite should include both correct and incorrect inputs.

> **1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.**
>
> **2. Modify your programs such that it runs on eclipse IDE, and then execute your test suites on the program. While executing your input data in a program, check whether the identified expected outcome (mentioned by you) is correct or not.**

Answer:

Equivalence Classes:

1. Date
Valid: 1 ≤ day ≤ 31
Invalid: day < 1, day > 31

2. Month
Valid: 1 ≤ month ≤ 12
Invalid: month < 1, month > 12

3. Year
Valid: 1900 ≤ year ≤ 2015
Invalid: year < 1900, year > 2015

Test Cases:

Partition 1: Valid dates with a day between 1 and 31, a month between 1 and 12, and a year between 1900 and 2015.
Partition 2: Invalid dates with a day less than 1 or greater than 31.
Partition 3: Invalid dates with a month less than 1 or greater than 12.
Partition 4: Invalid dates with a year less than 1900 or greater than 2015.
Partition 5: Invalid dates with a day that is out of range for a given month (e.g., February 30).

Partition 6: Invalid dates with a day that is out of range for a given year (e.g., February 29 in a non-leap year).

Some sample test cases for different partitions:

Partition 1: 01/01/2009, 15/03/1990, 31/12/2004
Partition 2: 00/01/2004, -10/03/2001, 32/12/2000
Partition 3: 01/00/2001, 15/13/2011, 31/15/2010
Partition 4: 01/01/0000, 15/03/10000, 31/12/99999
Partition 5: 30/02/2022, 31/04/2023, 28/02/2100
Partition 6: 29/02/2021, 29/02/1900, 29/02/2100

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.

Answer :

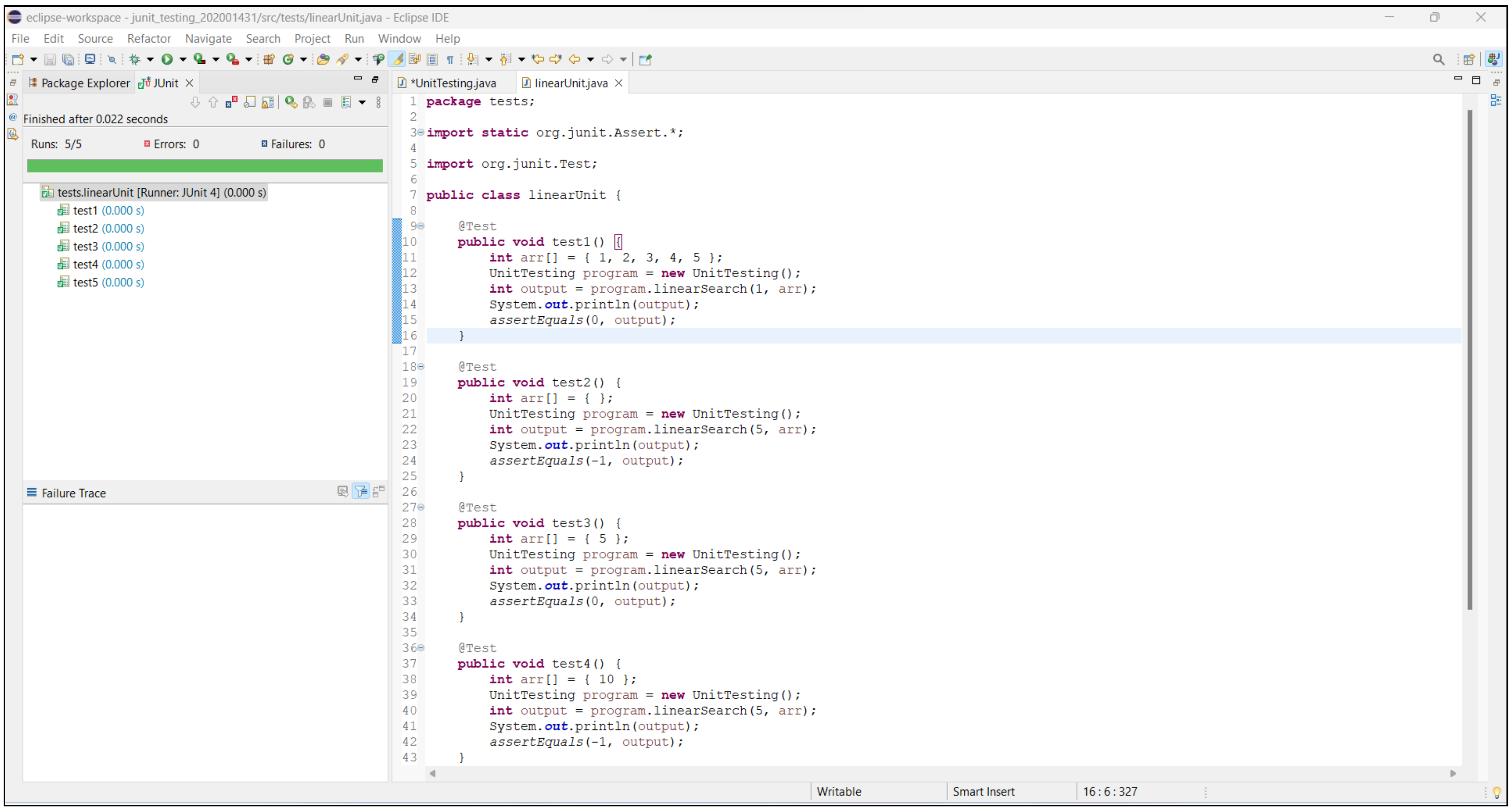
```
int linearSearch(int v, int a[])
{
    int i = 0;
    while (i < a.length)
    {
        if (a[i] == v)
            return(i);
        i++;
    }
    return (-1);
}
```

Equivalence Partitioning

Tester Action and Input Data	Expected Output
v = 5, a = { }	-1
v = 10, a = { 1, 2, 3, 4, 5 }	-1
v = 5, a = { 1, 2, 3, 4, 5 }	4

Boundary Value Analysis

Tester Action and Input Data	Expected Output
v = 1, a = { }	-1
v = 1, a = { 1 }	0
v = 2, a = { 1 }	-1
v = 1, a = { 1, 2, 3, 4, 5 }	0
v = 5, a = { 1, 2, 3, 4, 5 }	4


eclipse-workspace - junit_testing_202001431/src/tests/linearUnit.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.022 seconds
Runs: 5/5
Errors: 0
Failures: 0
tests.linearUnit [Runner: JUnit 4] (0.000 s)
test1 (0.000 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.000 s)
Failure Trace
*UnitTesting.java
linearUnit.java
package tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class linearUnit {
@Test
public void test1() {
int arr[] = { 1, 2, 3, 4, 5 };
UnitTesting program = new UnitTesting();
int output = program.linearSearch(1, arr);
System.out.println(output);
assertEquals(0, output);
}
@Test
public void test2() {
int arr[] = { };
UnitTesting program = new UnitTesting();
int output = program.linearSearch(5, arr);
System.out.println(output);
assertEquals(-1, output);
}
@Test
public void test3() {
int arr[] = { 5 };
UnitTesting program = new UnitTesting();
int output = program.linearSearch(5, arr);
System.out.println(output);
assertEquals(0, output);
}
@Test
public void test4() {
int arr[] = { 10 };
UnitTesting program = new UnitTesting();
int output = program.linearSearch(5, arr);
System.out.println(output);
assertEquals(-1, output);
}
Writable
Smart Insert
16 : 6 : 327

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

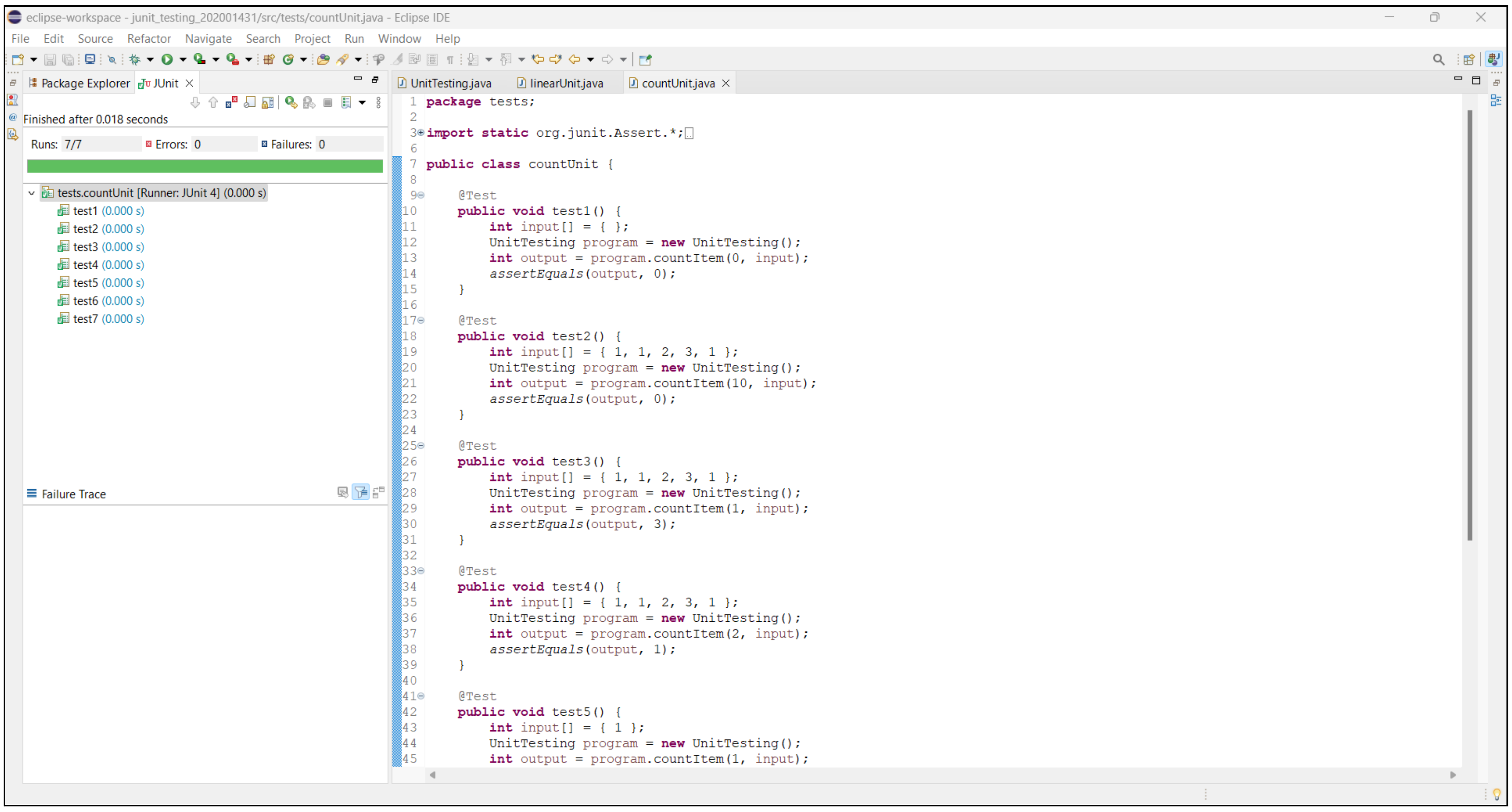
```
int countItem(int v, int a[])
{
    int count = 0;
    for (int i = 0; i < a.length; i++)
    {
        if (a[i] == v)
            count++;

    }
    return (count);
}
```

Equivalence Partitioning

Tester Action and Input Data	Expected Output
v = 1, a = { }	0
v = 10, a = { 1, 1, 2, 3, 1 }	0
v = 1, a = { 1, 1, 2, 3, 1 }	3

Boundary Value Analysis

Tester Action and Input Data	Expected Output
v = 1, a = { }	0
v = 1, a = {1}	1
v = 1, a = {2}	0
v = 1, a = { 1, 1, 2, 3, 1 }	3
v = 4, a = { 1, 1, 2, 3, 1 }	0


eclipse-workspace - junit_testing_202001431/src/tests/countUnit.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.018 seconds
Runs: 7/7
Errors: 0
Failures: 0
tests.countUnit [Runner: JUnit 4] (0.000 s)
test1 (0.000 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.000 s)
test6 (0.000 s)
test7 (0.000 s)
Failure Trace
UnitTesting.java
linearUnit.java
countUnit.java
package tests;
import static org.junit.Assert.*;
public class countUnit {
@Test
public void test1() {
int input[] = { };
UnitTesting program = new UnitTesting();
int output = program.countItem(0, input);
assertEquals(output, 0);
}
@Test
public void test2() {
int input[] = { 1, 1, 2, 3, 1 };
UnitTesting program = new UnitTesting();
int output = program.countItem(10, input);
assertEquals(output, 0);
}
@Test
public void test3() {
int input[] = { 1, 1, 2, 3, 1 };
UnitTesting program = new UnitTesting();
int output = program.countItem(1, input);
assertEquals(output, 3);
}
@Test
public void test4() {
int input[] = { 1, 1, 2, 3, 1 };
UnitTesting program = new UnitTesting();
int output = program.countItem(2, input);
assertEquals(output, 1);
}
@Test
public void test5() {
int input[] = { 1 };
UnitTesting program = new UnitTesting();
int output = program.countItem(1, input);

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.

Answer :

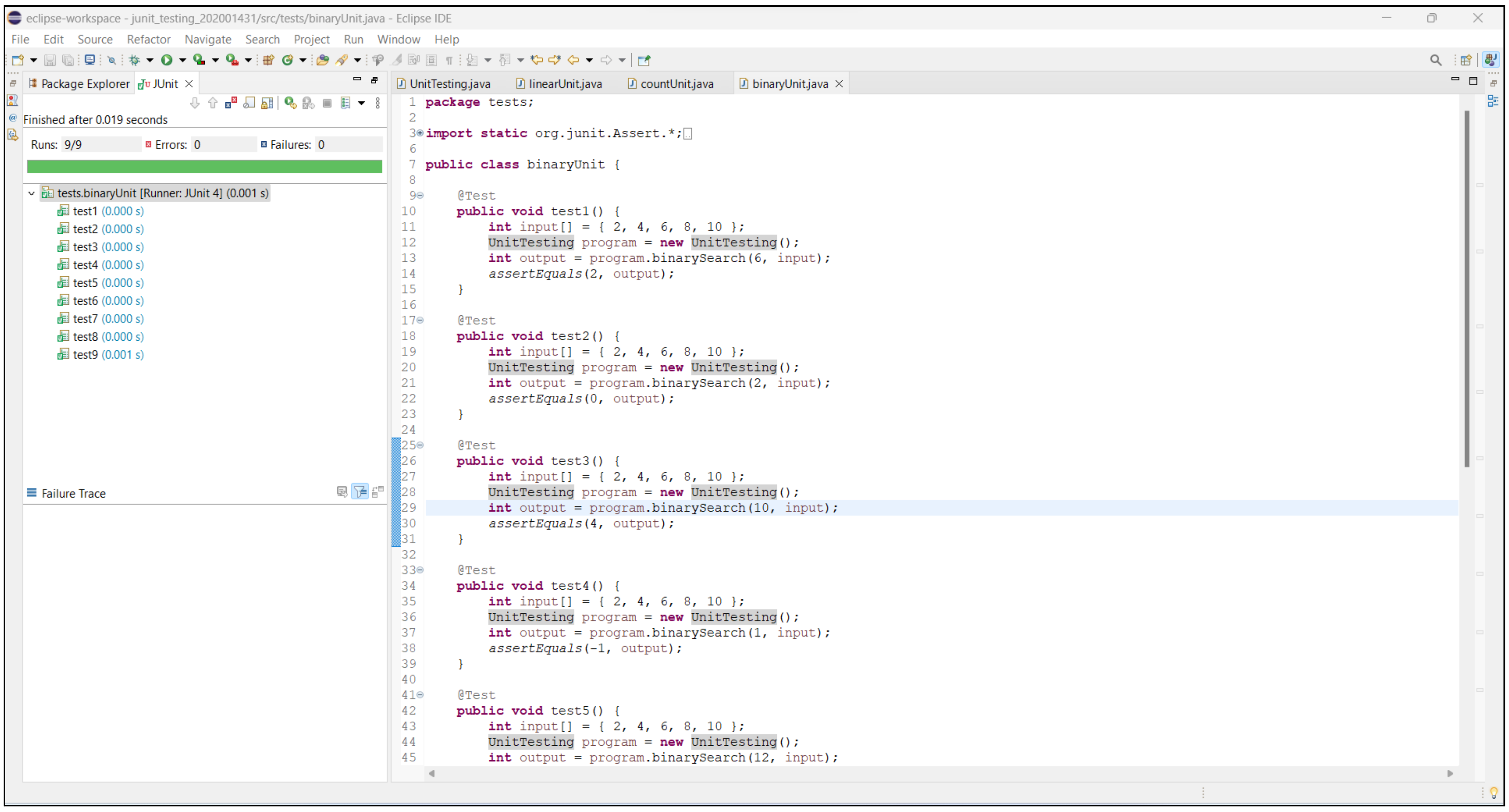
```
int binarySearch(int v, int a[])
{
    int lo,mid,hi;
    lo = 0;
    hi = a.length-1;
    while (lo <= hi)
    {
        mid = (lo+hi)/2;
        if (v == a[mid])
            return (mid);
        else if (v < a[mid])
            hi = mid-1;
        else
            lo = mid+1;

    }
    return(-1);
}
```

Equivalence Partitioning

Tester Action and Input Data	Expected Output
v=6, a={2, 4, 6, 8, 10}	2
v=2, a={2, 4, 6, 8, 10}	0
v=10, a={2, 4, 6, 8, 10}	4
v=1, a={2, 4, 6, 8, 10}	-1

Boundary Value Analysis

Tester Action and Input Data	Expected Output
v=10, a={10}	0
v=5, a{}	-1
v=5, a={5, 7, 9}	0
v=5, a={1, 3, 5}	2


eclipse-workspace - junit_testing_202001431/src/tests/binaryUnit.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
linearUnit.java
countUnit.java
binaryUnit.java
Finished after 0.019 seconds
Runs: 9/9
Errors: 0
Failures: 0
tests.binaryUnit [Runner: JUnit 4] (0.001 s)
test1 (0.000 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.000 s)
test6 (0.000 s)
test7 (0.000 s)
test8 (0.000 s)
test9 (0.001 s)
Failure Trace
package tests;
import static org.junit.Assert.*;
public class binaryUnit {
@Test
public void test1() {
int input[] = { 2, 4, 6, 8, 10 };
UnitTesting program = new UnitTesting();
int output = program.binarySearch(6, input);
assertEquals(2, output);
}
@Test
public void test2() {
int input[] = { 2, 4, 6, 8, 10 };
UnitTesting program = new UnitTesting();
int output = program.binarySearch(2, input);
assertEquals(0, output);
}
@Test
public void test3() {
int input[] = { 2, 4, 6, 8, 10 };
UnitTesting program = new UnitTesting();
int output = program.binarySearch(10, input);
assertEquals(4, output);
}
@Test
public void test4() {
int input[] = { 2, 4, 6, 8, 10 };
UnitTesting program = new UnitTesting();
int output = program.binarySearch(1, input);
assertEquals(-1, output);
}
@Test
public void test5() {
int input[] = { 2, 4, 6, 8, 10 };
UnitTesting program = new UnitTesting();
int output = program.binarySearch(12, input);

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).

Answer :

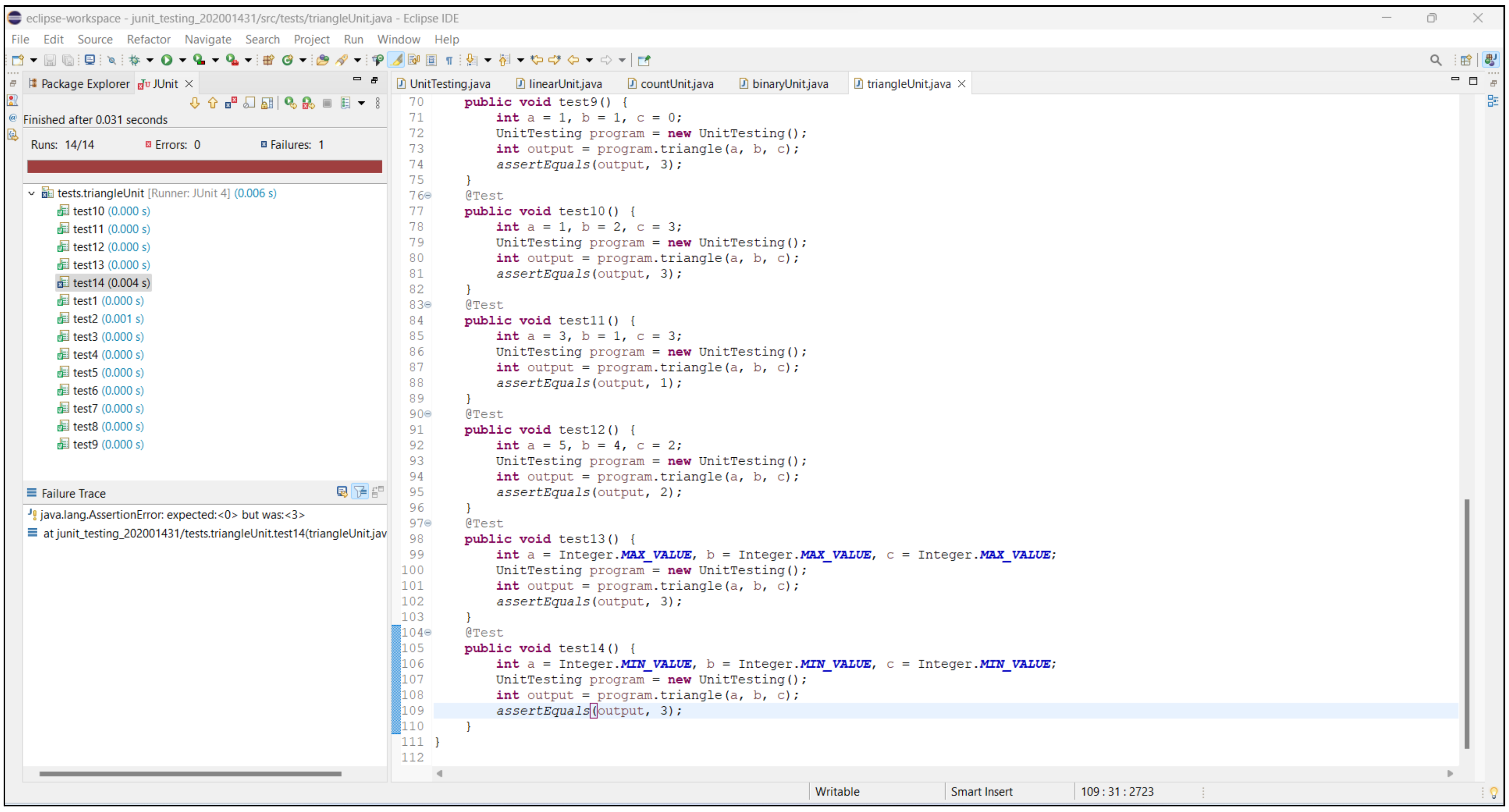
```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;

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);

}
```

Equivalence Partitioning

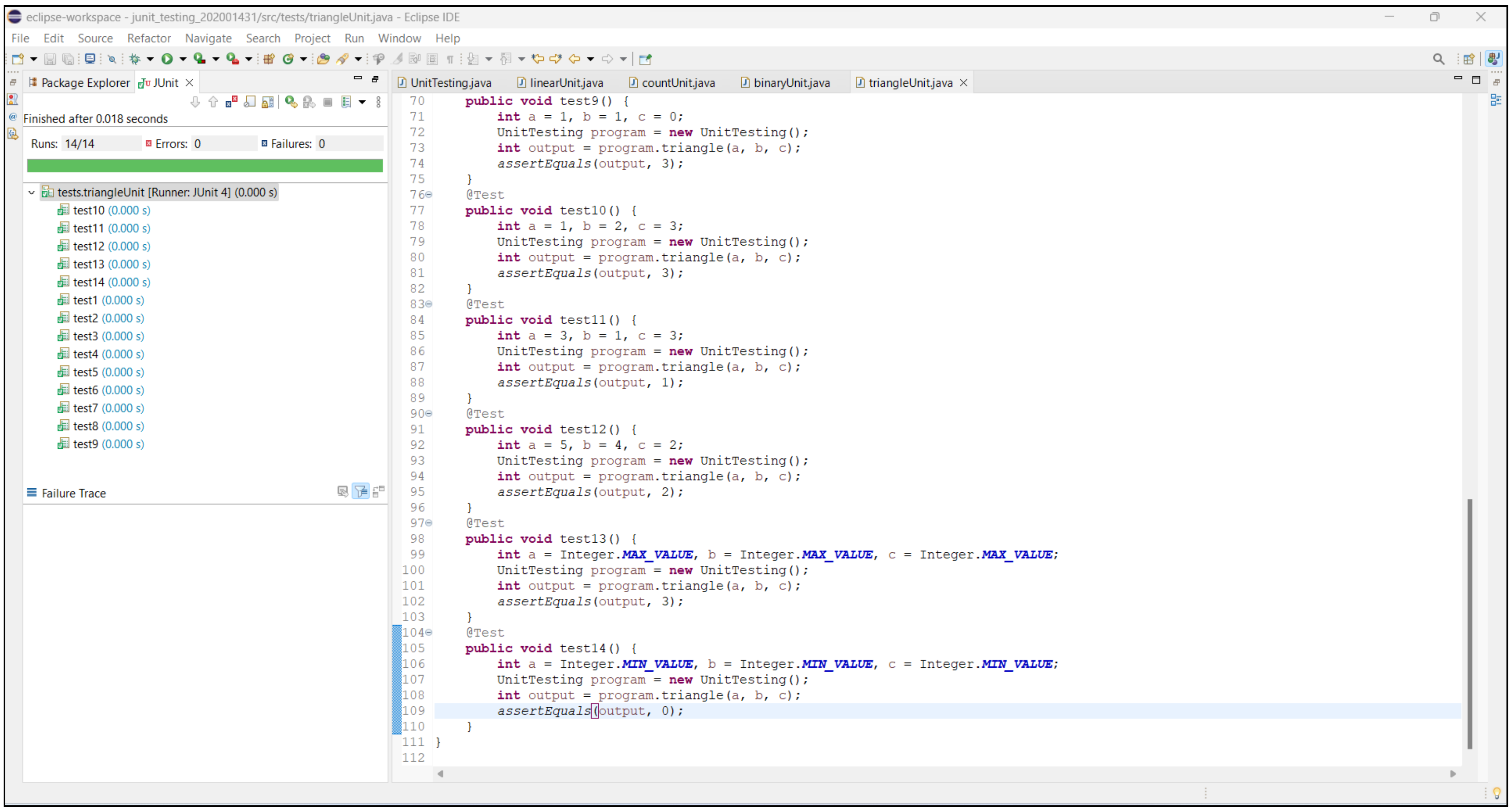
Tester Action and Input Data	Expected Output
(2, 2, 2)	0
(3, 3, 4)	1
(6, 5, 4)	2
(0, 0, 0)	3
(-1, -1, 5)	3
(2, 2, 1)	1
(0, 1, 1)	3
(1, 0, 1)	3
(1, 1, 0)	3

Boundary Value Analysis

Tester Action and Input Data	Expected Output
(0, 0, 0)	3
(5, 5, 5)	0
a = b = c = Integer.MAX_VALUE	3
a = b = c = Integer.MIN_VALUE	3



Here, we can see that the test case (Integer.MIN_VALUE, Integer.MIN_VALUE, Integer.MIN_VALUE) fails. This is because Integer.MIN_VALUE = -2147483648, which when added to itself overflows and becomes 0.



The test case works when set equal to 0.

P5. The function prefix (String s1, String s2) returns whether or notthe string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).
Answer :

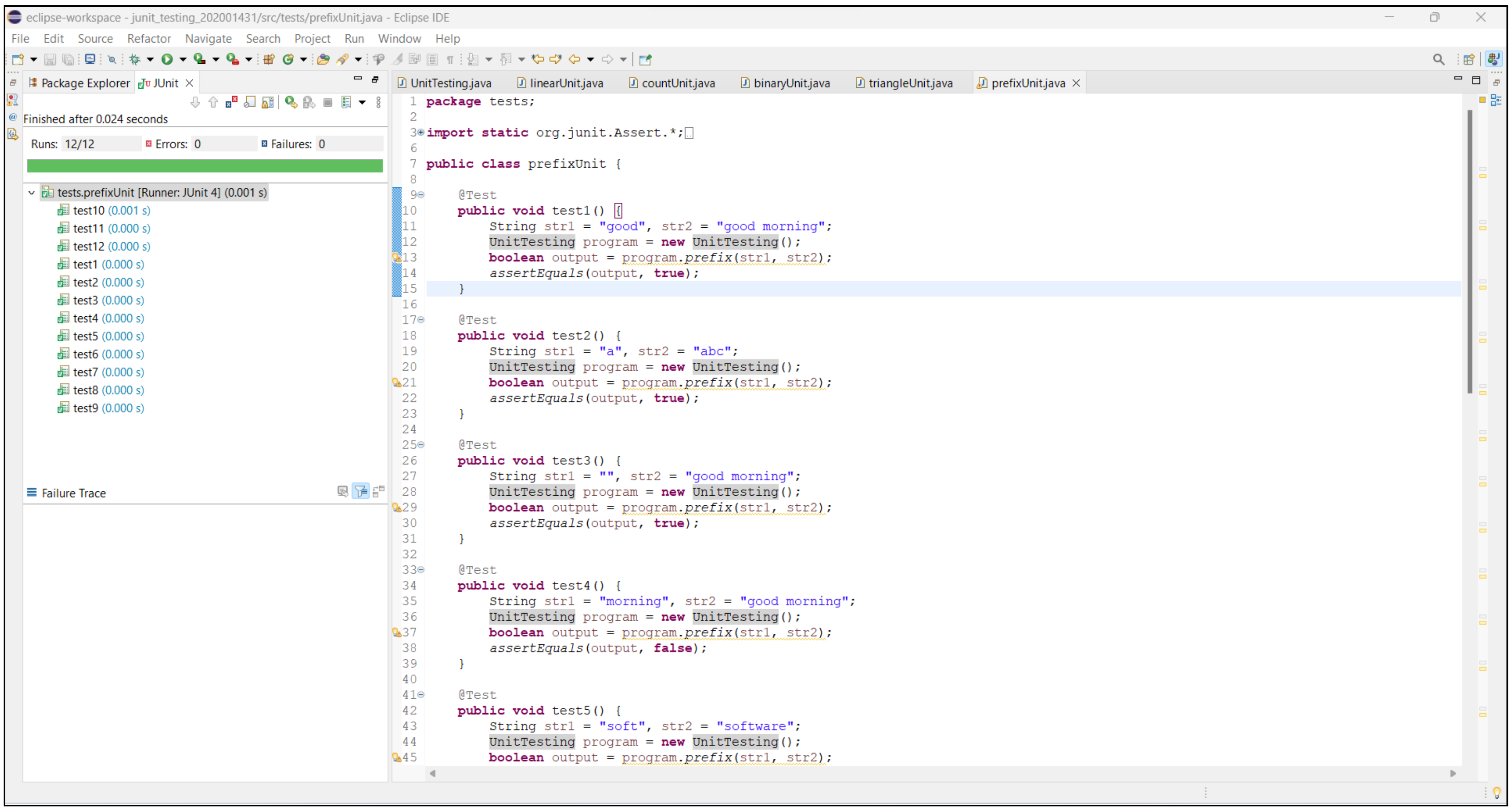
```
public static boolean prefix(String s1, String s2)
{
    if (s1.length() > s2.length())
    {
        return false;
    }
    for (int i = 0; i < s1.length(); i++)
    {
        if (s1.charAt(i) != s2.charAt(i))
        {
            return false;
        }
    }
    return true;
}
```

Equivalence Partitioning

Tester Action and Input Data	Expected Output
(“”, “software”)	true
(“software”, “”)	false
(“soft”, “software”)	true
(“softy”, “software”)	false
(“ware”, “software”)	false

Boundary Value Analysis

Tester Action and Input Data	Expected Output
(“abc”, “abc”)	true
(“a”, “b”)	false
(“a”, “a”)	true
(“”, “”)	true


eclipse-workspace - junit_testing_202001431/src/tests/prefixUnit.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.024 seconds
Runs: 12/12
Errors: 0
Failures: 0
tests.prefixUnit [Runner: JUnit 4] (0.001 s)
test10 (0.001 s)
test11 (0.000 s)
test12 (0.000 s)
test1 (0.000 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.000 s)
test6 (0.000 s)
test7 (0.000 s)
test8 (0.000 s)
test9 (0.000 s)
Failure Trace
UnitTesting.java
linearUnit.java
countUnit.java
binaryUnit.java
triangleUnit.java
prefixUnit.java
package tests;
import static org.junit.Assert.*;
public class prefixUnit {
@Test
public void test1() {
String str1 = "good", str2 = "good morning";
UnitTesting program = new UnitTesting();
boolean output = program.prefix(str1, str2);
assertEquals(output, true);
}
@Test
public void test2() {
String str1 = "a", str2 = "abc";
UnitTesting program = new UnitTesting();
boolean output = program.prefix(str1, str2);
assertEquals(output, true);
}
@Test
public void test3() {
String str1 = "", str2 = "good morning";
UnitTesting program = new UnitTesting();
boolean output = program.prefix(str1, str2);
assertEquals(output, true);
}
@Test
public void test4() {
String str1 = "morning", str2 = "good morning";
UnitTesting program = new UnitTesting();
boolean output = program.prefix(str1, str2);
assertEquals(output, false);
}
@Test
public void test5() {
String str1 = "soft", str2 = "software";
UnitTesting program = new UnitTesting();
boolean output = program.prefix(str1, str2);

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
Answer :

Class Name	Equivalent Classes	Expected Output
E1	negative or zero values	Invalid inputs
E2	$a + b \leq c$	Non-triangle
E3	a != b, b != c, c != a	Scalene triangle
E4	a = b, a != c	Isosceles triangle
E5	a = b, b = c, c = a	Equilateral triangle
E6	$a^2 + b^2 = c^2$	Right-angled triangle

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)
Answer :

Test cases(a, b, c)	Output	Equivalence Class Covered
-1, 2, 0	Invalid inputs	E1
1, 2, 5	Non-triangle	E2
3, 4, 6	Scalene triangle	E3
4, 4, 6	Isosceles triangle	E4
4, 4, 4	Equilateral triangle	E5
3, 4, 5	Right-angled triangle	E6

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

1. (4, 5, 9) $(a + b = c)$
2. (4, 5, 8.9) $(a + b > c)$
3. (4, 5, 9.1) $(a + b < c)$

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

1. (4, 5, 4) (a = c)
2. (4, 5, 3.9) (a > c)
3. (4, 5, 4.1) (a < c)

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

1. (4, 4, 4) (a = b = c)
2. (4, 4.1, 4) (a != b && a = c)
3. (4, 4, 4.1) (a = b && a != c)
4. (4.1, 4, 4) (a != b && a != c)

f) For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.
Answer :

1. (3, 4, 5) ($a^2 + b^2 = c^2$)
2. (0.12, 0.5, 0.14) ($a^2 + b^2 < c^2$)
3. (7, 23, 24) ($a^2 + b^2 > c^2$)

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

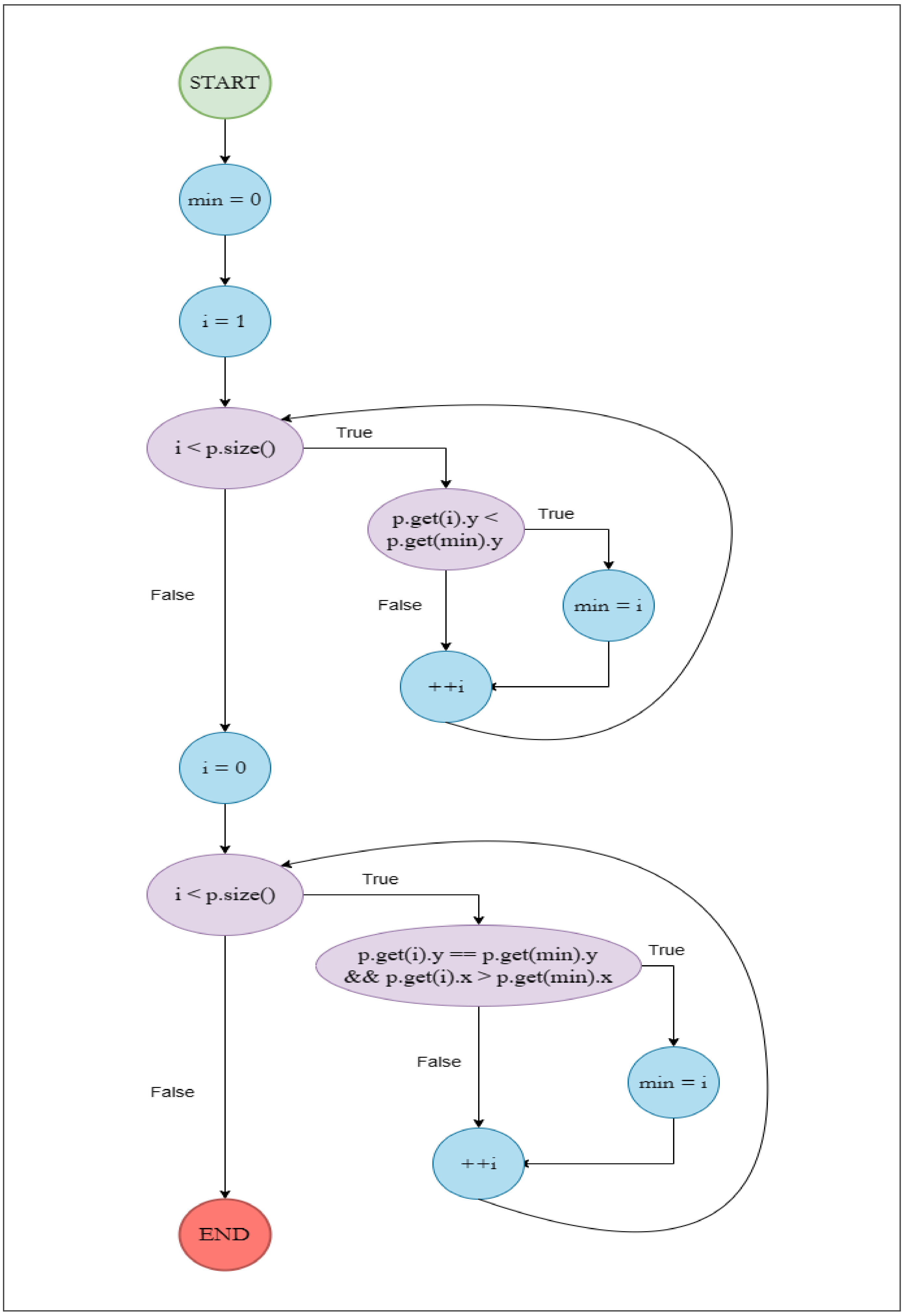
1. (1, 2, 3)
2. (5, 5, 10)
3. (0, 0, 0)

h) For non-positive input, identify test points.
Answer :
1. (-4.0, 4.2, 4.5)
2. (5, -4.2, -3.2)
3. (4, 5, -10)

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 i th 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.

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

```
1. int i, j, min, M;
2. Point t;
3. min = 0;
4. for(i = 1; i < p.size(); ++i) {
5.     if(((Point) p.get(i)).y < ((Point) p.get(min)).y) {
6.         min = i;
     }
}
7. for(i = 0; i < p.size(); ++i) {
8.     if(((Point) p.get(i)).y == ((Point) p.get(min)).y
             && ((Point) p.get(i)).x > ((Point) p.get(min)).x) {
9.         min = i;
     }
}
```

Answer.

To satisfy statement coverage, we need to ensure that each statement in the control flow graph is executed at least once.

To satisfy branch coverage, we need to ensure that each branch in the control flow graph is executed at least once.

To satisfy basic condition coverage, we need to ensure that each condition in the control flow graph is evaluated to both true and false at least once.

The following test cases will cover all the criterias:

Test 1:
p = [(10, 20)]
Statements covered = { 1, 2, 3, 7, 8 }
Branches covered = { 8 }
Basic conditions covered = { }

Test 2:
p=[ ]
Statements covered = { 1, 2, 3 }
Branches covered = {}
Basic conditions covered = {}

Test 3:
p = [(10, 50), (20, 70), (30, 50), (40, 50), (50, 60)]
Statements covered = { 1, 2, 3, 4, 5, 6, 7, 8, 9 }
Branches covered = { 5, 8 }
Basic conditions covered = { 5-false,true, 8-false,true }

Test 4:
p = [(20, 30), (30, 40), (10, 20), (50, 60)]
Statements covered = { 1, 2, 3, 4, 5, 6, 7 }
Branches covered = { 5, 8 }
Basic conditions covered = { 5-false,true, 8-false }