Name : Viradiya Abhay
ID : 202001174
Lab 7

Section A

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 previous date or invalid date. Design the equivalence class test cases?

- Equivalence Class Partitions

We can divide the input space into these equivalence classes:

Equivalence class for day
Valid class : 1 <= day <= 31(c1)
Invalid class : day < 1(c2) , day > 31(c3)

Equivalence class for Month
Valid class : 1 <= month <= 12 (c4)
Invalid class : month < 1 (c5) , month> 12 (c6)

Equivalence class for Year
Valid class : 1900 <= year <= 2015 (c7)
Invalid class : year < 1900 (c8) , year> 2016 (c9)

Equivalence class for leap year
Invalid class : day=29 month=2 and not leap year (c10)

Based on these equivalence class, we can design the following test cases:

Test Case	Valid/Invalid	Equivalence class
day=1, month=1, year=1900	Valid	c1 , c4 , c7
day=15, month=6, year=2005	Valid	c1 , c4 , c7
day=31, month=12, year=2015	Valid	c1 , c4 , c7
day=0, month=6, year=2000	Invalid	c2
day=32, month=2, year=2001	Invalid	c3
day=15, month=0, year=2001	Invalid	c5
day=15, month=13, year=2001	Invalid	c6
day=15, month=6, year=1889	Invalid	c8
day=15, month=6, year=2016	Invalid	c9
day=29, month=2, year=1900	Invalid	c10

- Boundary Value Analysis:

Here are boundary values for each equivalence class:

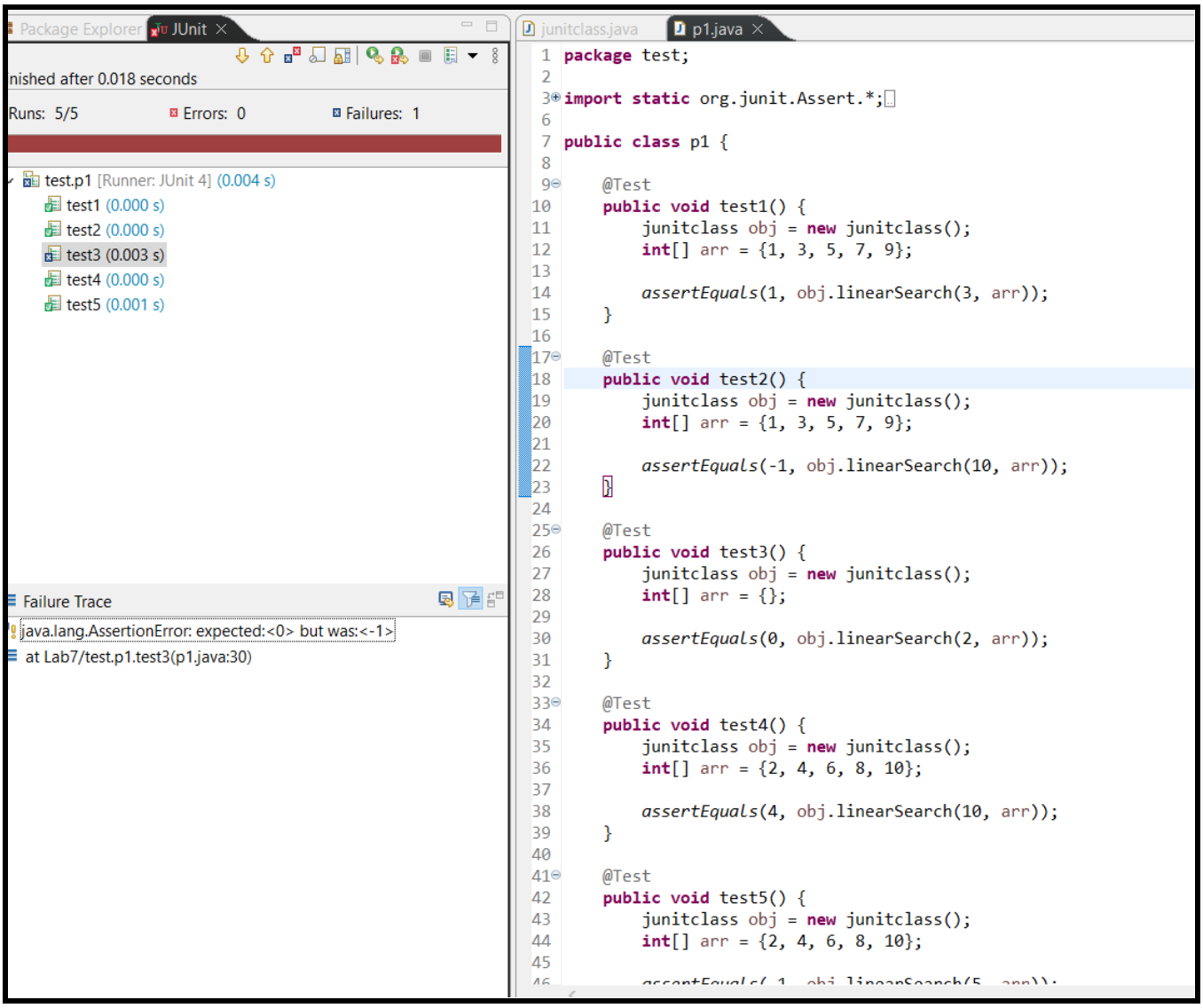
The earliest possible date: (1, 1, 1900)
The latest possible date: (31, 12, 2015)
The earliest day of each month: (1, 1, 2000), (1, 2, 2000), (1, 3, 2000),..., (1, 12, 2000)
The latest day of each month: (31, 1, 2000), (28, 2, 2000), (31, 3, 2000),..., (31, 12, 2000)
Leap year day: (29, 2, 2000)
Invalid leap year day: (29, 2, 1900)
One day before earliest date: (31, 12, 1899)
One day after latest date: (1, 1, 2016)

Based on these, we can design the following test cases:

Test Case	Valid/Invalid
day=1, month=1, year=1900	Valid
day=31, month=12, year=2015	Valid
day=0, month=6, year=2000	Invalid
day=32, month=6, year=2000	Invalid
day=29, month=2, year=2000	Invalid
day=1, month=6, year=2000	Valid
day=31, month=5, year=2000	Valid
day=15, month=6, year=2000	Valid
day=31, month=4, year=2000	Invalid

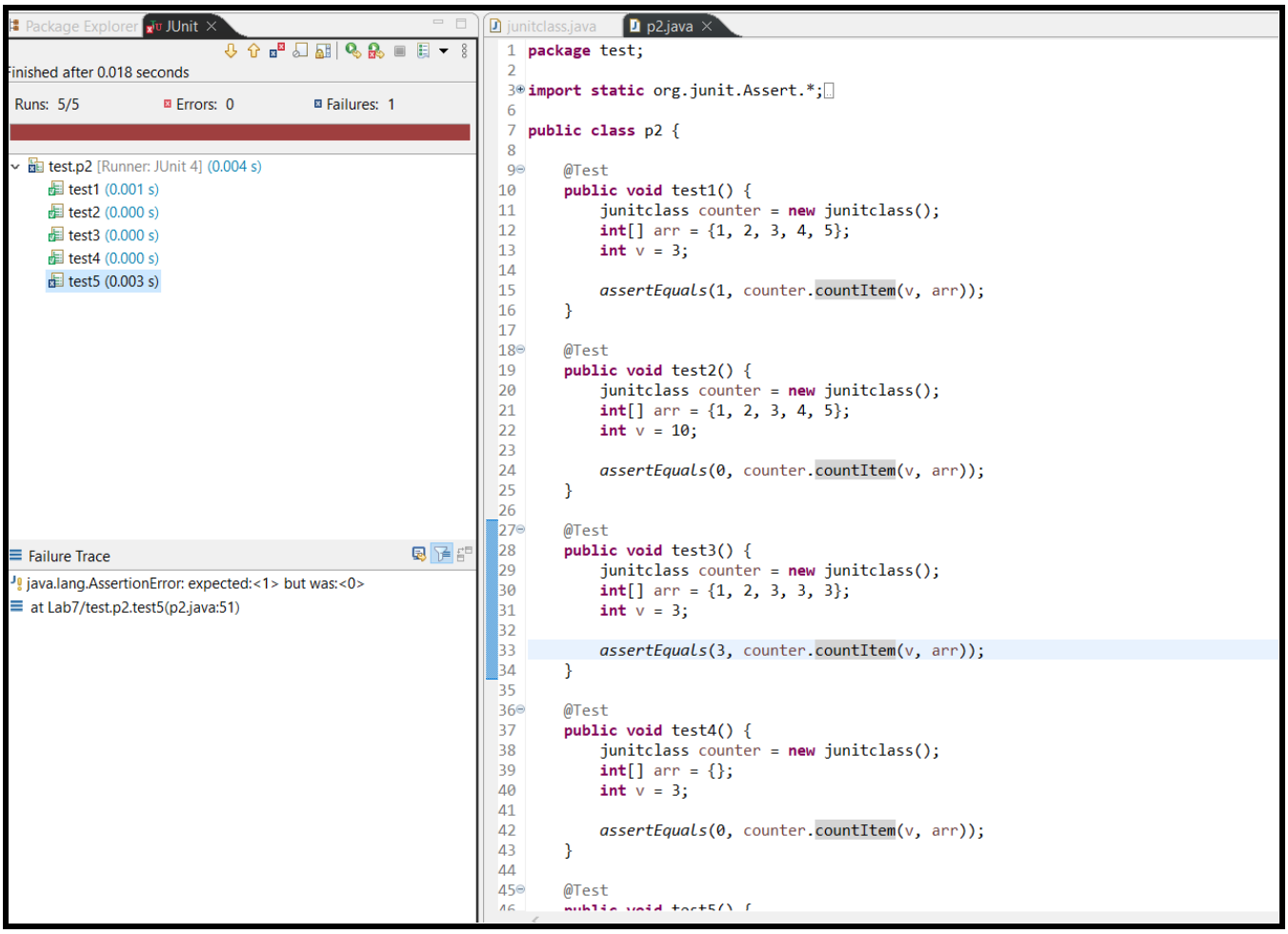
P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
Value present in array. a = [1, 3, 5, 7, 9], v = 5	2
Value not present in array. Input: a = [1, 3, 5, 7, 9], v = 4	-1
Invalid Input. Input: a = [], v =	An error message
Invalid Input. Input: a = [], v = 10	An error message
Invalid Input. Input: a = [1, 3, 5, 7], v = 2.2	An error message
Boundary Value Analysis	
Array with the minimum length possible. Input: a = [0], v = 0	0
Array with the maximum length possible. Input: a = [1, 2, ..., 9998, 9999], v = 3	2
Search value at the beginning of the array. Input: a = [10, 20, 30, 40, 50], v = 10	0
Search value at the end of the array. Input: a = [10, 20, 30, 40, 50], v = 50	4


Package Explorer
JUnit
nished after 0.018 seconds
Runs: 5/5
Errors: 0
Failures: 1
test.p1 [Runner: JUnit 4] (0.004 s)
test1 (0.000 s)
test2 (0.000 s)
test3 (0.003 s)
test4 (0.000 s)
test5 (0.001 s)
Failure Trace
java.lang.AssertionError: expected:<0> but was:<-1>
at Lab7/test.p1.test3(p1.java:30)
junitclass.java
p1.java
1 package test;
2
3 import static org.junit.Assert.*;
6
7 public class p1 {
8
9 @Test
10 public void test1() {
11 junitclass obj = new junitclass();
12 int[] arr = {1, 3, 5, 7, 9};
13
14 assertEquals(1, obj.linearSearch(3, arr));
15 }
16
17 @Test
18 public void test2() {
19 junitclass obj = new junitclass();
20 int[] arr = {1, 3, 5, 7, 9};
21
22 assertEquals(-1, obj.linearSearch(10, arr));
23 }
24
25 @Test
26 public void test3() {
27 junitclass obj = new junitclass();
28 int[] arr = {};
29
30 assertEquals(0, obj.linearSearch(2, arr));
31 }
32
33 @Test
34 public void test4() {
35 junitclass obj = new junitclass();
36 int[] arr = {2, 4, 6, 8, 10};
37
38 assertEquals(4, obj.linearSearch(10, arr));
39 }
40
41 @Test
42 public void test5() {
43 junitclass obj = new junitclass();
44 int[] arr = {2, 4, 6, 8, 10};
45

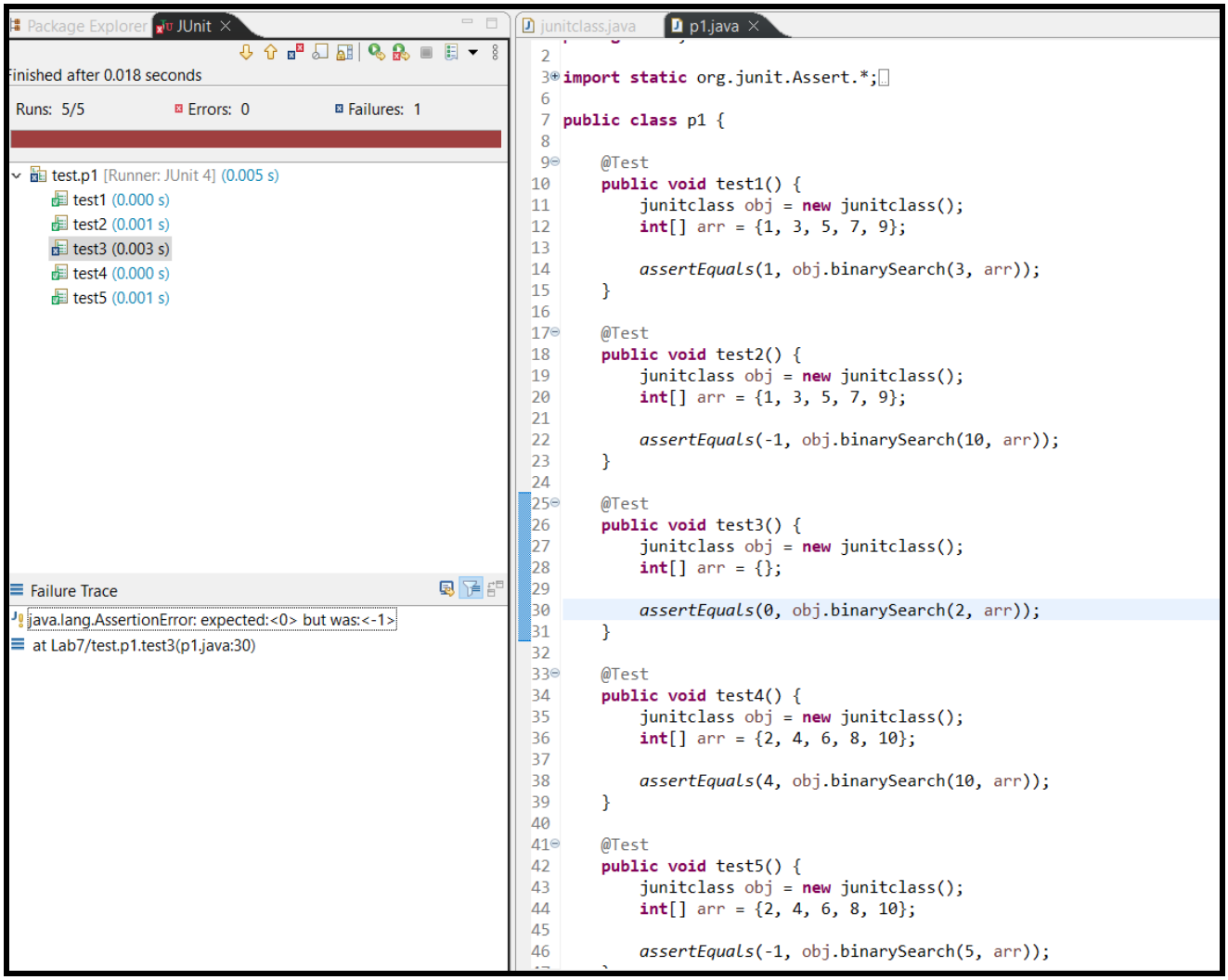
P2. The function countItem returns the number of times a value v appears in an array of integers a.

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
v = 10, a = {1, 5, 6, 5, 2}	0
v = 0, a = (0, 0, 0, 0, 0}	5
Invalid Input: v = 'a', a = {1, 2, 3, 4, 5}	An error message
Invalid Input: v = 3, a = null	An error message
Boundary Value Analysis	
Array is empty. Input: v = 3 , a = {}	0
Value present only once. Input: v = 3 , a = { 1, 2, 3}	1
Value present multiple times. Input: v = 3 , a = {3, 3, 3, 3, 3}	5


Package Explorer
JUnit
Finished after 0.018 seconds
Runs: 5/5
Errors: 0
Failures: 1
test.p2 [Runner: JUnit 4] (0.004 s)
test1 (0.001 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.003 s)
Failure Trace
java.lang.AssertionError: expected:<1> but was:<0>
at Lab7/test.p2.test5(p2.java:51)
junitclass.java
p2.java
package test;
import static org.junit.Assert.*;
public class p2 {
@Test
public void test1() {
junitclass counter = new junitclass();
int[] arr = {1, 2, 3, 4, 5};
int v = 3;
assertEquals(1, counter.countItem(v, arr));
}
@Test
public void test2() {
junitclass counter = new junitclass();
int[] arr = {1, 2, 3, 4, 5};
int v = 10;
assertEquals(0, counter.countItem(v, arr));
}
@Test
public void test3() {
junitclass counter = new junitclass();
int[] arr = {1, 2, 3, 3, 3};
int v = 3;
assertEquals(3, counter.countItem(v, arr));
}
@Test
public void test4() {
junitclass counter = new junitclass();
int[] arr = {};
int v = 3;
assertEquals(0, counter.countItem(v, arr));
}
@Test

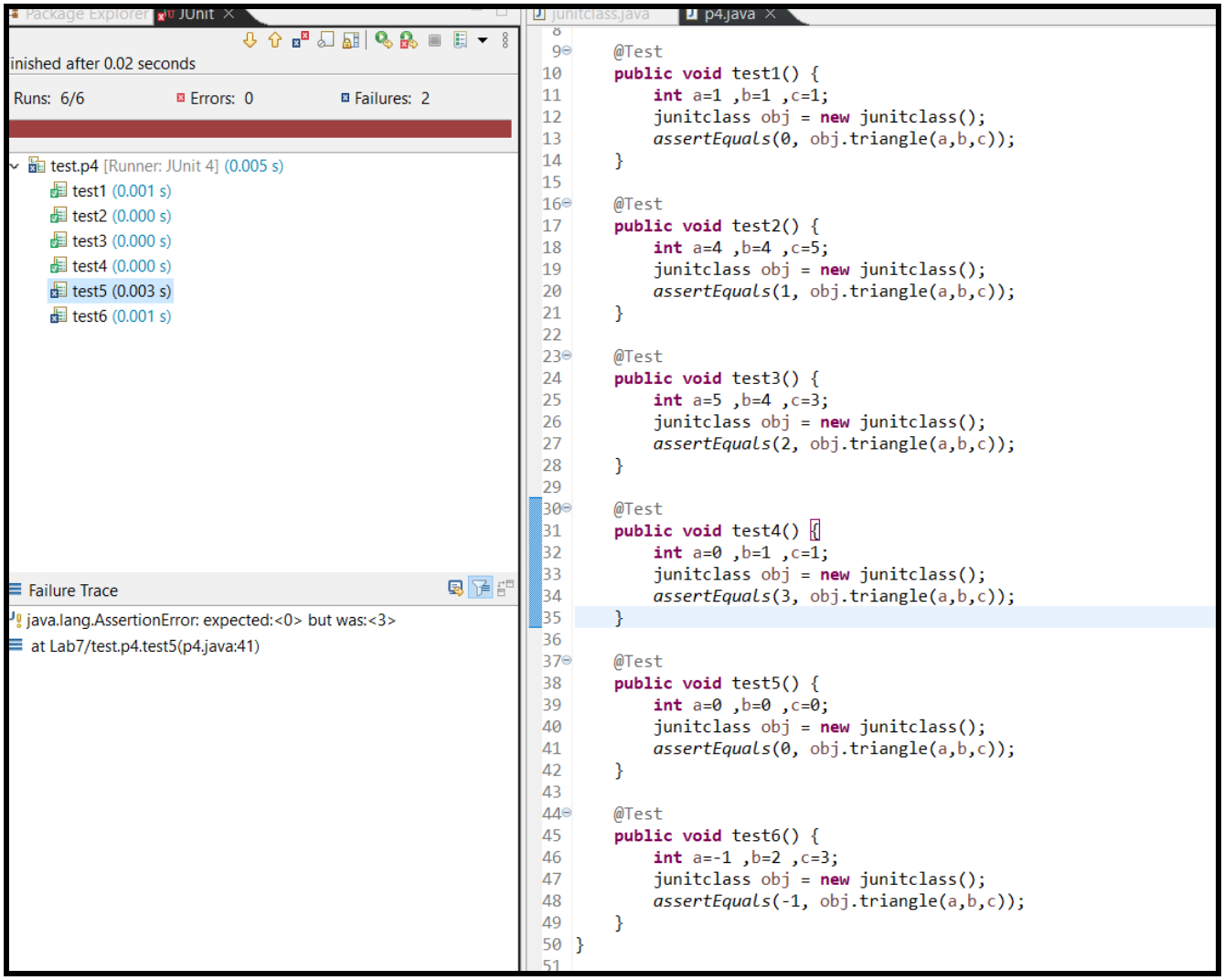
P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
Value present in array. a = [1, 3, 5, 7, 9], v = 5	2
Value not present in array. Input: a = [1, 3, 5, 7, 9], v = 4	-1
Invalid Input. Input: a = [], v =	An error message
Invalid Input. Input: a = [], v = 10	An error message
Invalid Input. Input: a = [1, 3, 5, 7], v = 2.2	An error message
Boundary Value Analysis	
Array with the minimum length possible. Input: a = [0], v = 0	0
Array with the maximum length possible. Input: a = [1, 2, ..., 9998, 9999], v = 3	2
Search value at the beginning of the array. Input: a = [10, 20, 30, 40, 50], v = 10	0
Search value at the end of the array. Input: a = [10, 20, 30, 40, 50], v = 50	4


Package Explorer
JUnit
Finished after 0.018 seconds
Runs: 5/5
Errors: 0
Failures: 1
test.p1 [Runner: JUnit 4] (0.005 s)
test1 (0.000 s)
test2 (0.001 s)
test3 (0.003 s)
test4 (0.000 s)
test5 (0.001 s)
Failure Trace
java.lang.AssertionError: expected:<0> but was:<-1>
at Lab7/test.p1.test3(p1.java:30)
junitclass.java
p1.java
import static org.junit.Assert.*;
public class p1 {
@Test
public void test1() {
junitclass obj = new junitclass();
int[] arr = {1, 3, 5, 7, 9};
assertEquals(1, obj.binarySearch(3, arr));
}
@Test
public void test2() {
junitclass obj = new junitclass();
int[] arr = {1, 3, 5, 7, 9};
assertEquals(-1, obj.binarySearch(10, arr));
}
@Test
public void test3() {
junitclass obj = new junitclass();
int[] arr = {};
assertEquals(0, obj.binarySearch(2, arr));
}
@Test
public void test4() {
junitclass obj = new junitclass();
int[] arr = {2, 4, 6, 8, 10};
assertEquals(4, obj.binarySearch(10, arr));
}
@Test
public void test5() {
junitclass obj = new junitclass();
int[] arr = {2, 4, 6, 8, 10};
assertEquals(-1, obj.binarySearch(5, arr));

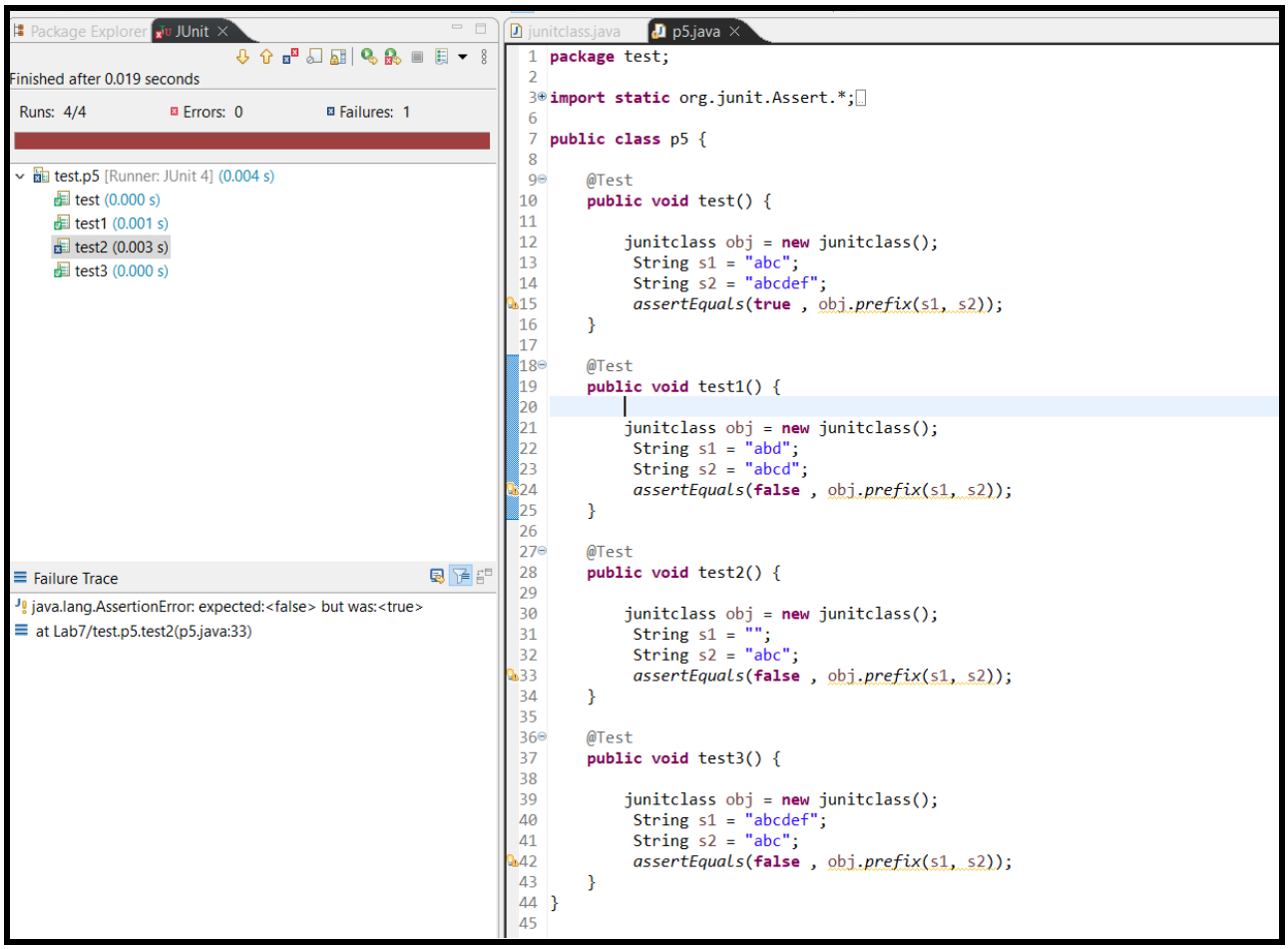
P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths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

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
Valid input: a=3, b=3, c=3	EQUILATERAL
Valid input: a=4, b=4, c=5	ISOSCELES
Valid input: a=5, b=4, c=3	SCALENE
Valid input: a=0, b=1, c=1	INVALID
Valid input: a=0, b=0, c=0	INVALID
Invalid input: a=0, b=0, c=0	An error message
Invalid input: a=-1, b=2, c=3	An error message
Invalid input: a=1.2, b=2, c=3	An error message
Boundary Value Analysis	
Maximum values: a, b, c = Integer.MAX_VALUE	INVALID
Minimum values: a, b, c = Integer.MIN_VALUE	INVALID
a = 0, b = 0, c = 0	INVALID
Equilateral triangles: a = b = c = 100	EQUILATERAL
Isosceles triangles: a = b ≠ c = 10	ISOSCELES
Isosceles triangles: a ≠ b = c = 10	ISOSCELES
Isosceles triangles: a = c ≠ b = 10	ISOSCELES


Package Explorer
JUnit
inished after 0.02 seconds
Runs: 6/6
Errors: 0
Failures: 2
test.p4 [Runner: JUnit 4] (0.005 s)
test1 (0.001 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
test5 (0.003 s)
test6 (0.001 s)
Failure Trace
java.lang.AssertionError: expected:<0> but was:<3>
at Lab7/test.p4.test5(p4.java:41)
junitclass.java
p4.java
@Test
public void test1() {
int a=1 ,b=1 ,c=1;
junitclass obj = new junitclass();
assertEquals(0, obj.triangle(a,b,c));
}
@Test
public void test2() {
int a=4 ,b=4 ,c=5;
junitclass obj = new junitclass();
assertEquals(1, obj.triangle(a,b,c));
}
@Test
public void test3() {
int a=5 ,b=4 ,c=3;
junitclass obj = new junitclass();
assertEquals(2, obj.triangle(a,b,c));
}
@Test
public void test4() {
int a=0 ,b=1 ,c=1;
junitclass obj = new junitclass();
assertEquals(3, obj.triangle(a,b,c));
}
@Test
public void test5() {
int a=0 ,b=0 ,c=0;
junitclass obj = new junitclass();
assertEquals(0, obj.triangle(a,b,c));
}
@Test
public void test6() {
int a=-1 ,b=2 ,c=3;
junitclass obj = new junitclass();
assertEquals(-1, obj.triangle(a,b,c));
}
}

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).

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
s1 = “abc” , s2 = “abcdef”	TRUE
s1 = “ad” , s2 = “abcdef”	FALSE
s1 = null , s2 = “abcdef”	An error message
s1 = “abc” , s2 = null	An error message
Boundary Value Analysis	
s1 = “” , s2 = “”	TRUE
s1 = “” , s2 = “abcd”	TRUE
s1 = “abcdef” , s2 = “abc”	FALSE
s1 = “abc” , s2 = “abcf”	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 Equivalence Classes:

C1: Invalid inputs (negative or zero values)
C2: Non-triangle (sum of the two shorter sides is not greater than the longest side)
C3: Scalene triangle (no sides are equal)
C4: Isosceles triangle (two sides are equal)
C5: Equilateral triangle (all sides are equal)
C6: Right-angled triangle (satisfies the Pythagorean theorem)

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)

Test cases:
TC1: -1, 0 , 2
TC2: 1, 2, 5
TC3: 3, 4, 5
TC4: 5, 5, 7
TC5: 6, 6, 6
TC6: 3, 4, 5

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

Valid - 3, 4, 5
Invalid - 3, 4, 8

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

Valid - 1, 2, 1
Valid - 5, 6, 5
Invalid - 0, 2, 0

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

Valid - 5.1, 5.1, 5.1
Invalid - 0, 0, 0
Invalid - -1 , -1 , -1

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

Valid - 3, 4, 5
Invalid - 0, 0, 0
Invalid - -3, -4, -5

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

A=2, B=2, C=4 (sum of A and B is less than C)

h) For non-positive input, identify test points.

A=0, B=4, C=5 (invalid input)
A=-2, B=4, C=5 (invalid input)

Section B

---

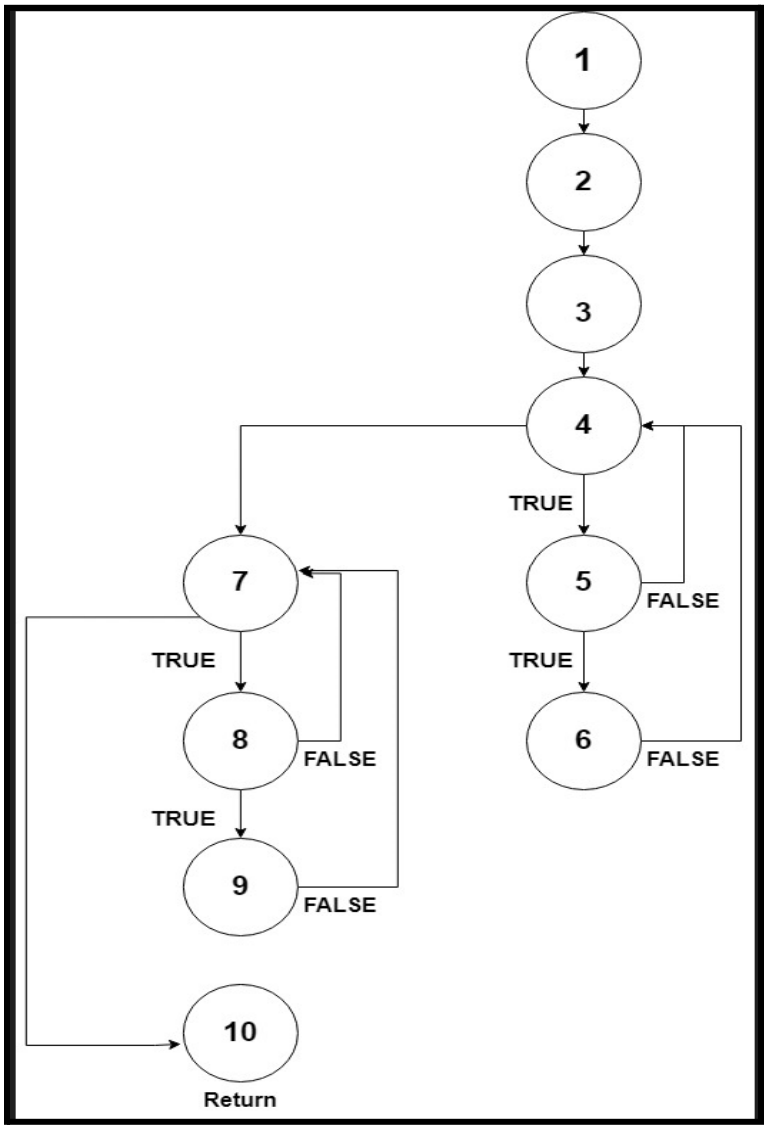
1. Convert the Java code comprising the beginning of the doGraham method into a control flow graph (CFG).

```
Vector doGraham (Vector p) {

int i,j,min, M;  // 1
Point t; // 2
min= 0; // 3

// search for minimum:
for(i=1; i < p.size(); ++i) // 4
{
        if( ((Point) p.get(i)).y < ((Point) p.get(min)).y) // 5
                min = i; // 6
}
```

```
// continue along the values with same y component
for(i=0; i<p.size(); ++i) // 7
{
        if(( ((Point) p.get(i)).y == ((Point) p.get(min)).y)  &&  (((Point) p.get(i)).x >
        ((Point) p.get(min)).x))  // 8
                min = i; // 9
}
//10
```



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.

- 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 Set 1:
p = { (0, 0) }

Test Set 2:
p = { (0, 0), (1, 1) }

Test Set 3:
p = { (0, 0), (1, 1), (2, 0) }

- 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 Set 1:
p = { (0, 0) , (1, 0) , (2, 0) }

Test Set 2:
p = { (0, 0), (1, 1) }

Test Set 3:
p = { (0, 0), (1, 1), (2, 0) }

- 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 Set 1:
p = { (0, 0) , (0, 1) , (1, 0)}

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

Test Set 3:
p = { (0, 0), (1, 1), (2, 0) }

Test Set 4:
p = { (-2, -2), (0, 0), (1, 1), (2, 2) }