Name: Rishit Shah
Student ID: 202001411
Course: IT314(Software Engineering)

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?

Valid equivalence classes:

Valid day, month, and year
Valid day, month, and minimum year (1900)
Valid day, month, and maximum year (2015)

Invalid equivalence classes:

Invalid day, month, and year (e.g., day 0, day 32, month 0, month 13, year < 1900 or year > 2015)
Invalid day for a given month and year (e.g., Feb 29 in a non-leap year, Apr 31, Jun 31, Sep 31, Nov 31)
Based on these equivalence classes, here are the test cases that should be considered:

Valid equivalence class test cases:

Test case 1: Valid day, month, and year (e.g., 15, 7, 2005)
Test case 2: Valid day, month, and minimum year (e.g., 1, 1, 1900)
Test case 3: Valid day, month, and maximum year (e.g., 31, 12, 2015)

Invalid equivalence class test cases:

Test case 4: Invalid day, month, and year (e.g., 0, 0, 1899)
Test case 5: Invalid day, month, and year (e.g., 32, 13, 2016)
Test case 6: Invalid day for a given month and year (e.g., Feb 29 in a non-leap year, such as 29, 2, 2001)
Test case 7: Invalid day for a given month and year (e.g., Apr 31, such as 31, 4, 2005)
Test case 8: Invalid day for a given month and year (e.g., Jun 31, such as 31, 6, 2005)
Test case 9: Invalid day for a given month and year (e.g., Sep 31, such as 31, 9, 2005)
Test case 10: Invalid day for a given month and year (e.g., Nov 31, such as 31, 11, 2005)

The above test cases cover all the equivalence classes for the input parameters and should be sufficient to test the program for determining the previous date.

Test Case ID	Day	Month	Year	Expected Output
1	1	6	2000	31-5-2000
2	2	6	2015	1-6-2015
3	2	6	2016	Invalid
4	1	1	1900	31-12-1899
5	31	12	1899	Invalid
6	31	12	1900	30-12-1900
7	29	2	2012	28-2-2012
8	1	3	2012	29-2-2012
9	29	2	2011	Invalid
10	30	2	2020	Invalid

Equivalence Class Partitions:

Day:

Partition ID	Range	Status
E1	Between 1 and 28	Valid
E2	Less than 1	Invalid
E3	Greater than 31	Invalid
E4	Equals 30	Valid
E5	Equals 29	Valid for leap year
E6	Equals 31	Valid

Month:

Partition ID	Range	Status
E7	Between 1 and 12	Valid
E8	Less than 1	Invalid
E9	Greater than 12	Invalid

Year:

Partition ID	Range	Status
E10	Between 1900 and 2015	Valid
E11	Less than 1	Invalid
E12	Greater than 2015	Invalid

Q. Write a set of test cases (i.e., test suite) – a 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.

Program 1

Equivalence Partitioning:

If a is empty, an error message should be returned.
If v is not in a, -1 should be returned.
If v is a's first element, the function should return 0.
If v is in the middle of a, the function should return the first index i such that a[i] == v, where i is greater than 0 and less than a.length - 1.
If v is the last element of a, the function should return a.length - 1.

Tester Action and Input Data	Expected Outcome
[2, 4, 6, 8, 10], v = 2	0
[1, 3, 5, 7, 9], v = 2	-1
[2, 4, 6, 8, 10], v = 11	-1
[-100, 100]	0
[1,2,3,4,5,6], v = 6	5
[], v = 10	-1
NULL, v = 111	-1

Boundary Value Analysis:
If a has one element, not v, -1 should be returned.
If a has one element, and it's v, the function should return 0.
If a has two elements and v is the first element, the function should return 0.
If a has two elements and v is the second element, the function should return 1.
If a has n elements, and v is the first element, the function should return 0.
If a has n elements, and v is the last element, the function should return n-1. If a has n elements, and v is not in a, -1 should be returned.

Tester Action and input data	Expected Outcome

NULL	-1
[], v = 5	-1
[5], v = 5	0
[5], v = 60	-1
[3,5], v = 3	0
[3,5], v = 5	1
[3,5], v = 14	-1
[1,3,5], v = 1	0
[1,3,5], v = 5	2
[1,3,5], v = 10	-1
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 1	0
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 111	12
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 1111	-1

Program 2

Equivalence Partitioning:

If a is empty, the function should return 0.

If v is not in a, the function should return 0.

If v appears once in a, the function should return 1.

If v appears multiple times in a, the function should return the number of occurrences of v.

Tester Action and Input Data	Expected Outcome
[265, 41, 60, 80, 100],v = 100	1
[2655, 451, 6560, 1050, 1050],v = 1050	2
[[265545, 451, 65460, 1050, 105024]],v = 1	0
[[10,10,10,10]],v = 11	0
[],v = 100	0
NULL,v = 51	0
[0],v = 0	1
[-89,-89],v = -89	2

Boundary Value Analysis:
If a has one element, and it's not v, the function should return 0.
If a has one element, and it's v, the function should return 1.
If a has two elements and v is the first element, the function should return 1.
If a has two elements and v is the second element, the function should return 1.
If a has n elements, and v appears once, the function should return 1.
If a has n elements, and v appears multiple times, the function should return the number of occurrences of v.
If a has n elements, and v is not in a, the function should return 0.

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4],v = 2	2
15, 10, 15, 15],v = 15	3
[],v = 100	0
NULL,v = 51	0
[-100,100,100,100],v = 10000	0
[-89,89],v = -89	1
[-890,890],v = 890	1

Program 3

Equivalence Partitioning:

If a is empty, the function should return -1.

If v is not in a, the function should return -1.

If v is the first element in a, the function should return 0.

If v is the last element in a, the function should return a.length-1.

If v appears once in a, the function should return the index of v.

If v appears multiple times in a, the function should return the index of the first occurrence of v.

Tester Action and input data	Expected outcome
[1, 21, 30, 40, 50],v = 21	1

[10, 20, 30, 40, 50, 60],v = 30	2
[10,100,1000,10000],v = 100000	-1
[,11,22,33,44],v = 444	-1
[11,20,200,300],v=11	0
[-100,-90,-80,100,1000],v = 10000	4
[],v = 12	-1
NULL,v = 168	-1
[1,2],v = 3	-1
[1,3],v=3	1

Boundary Value Analysis:
If a has one element, and it's not v, the function should return -1.
If a has one element, and it's v, the function should return 0.
If a has two elements and v is the first element, the function should return 0.
If a has two elements and v is the second element, the function should return 1.
If a has n elements, and v is the first element, the function should return 0.
If a has n elements, and v is the last element, the function should return n-1. If a has n elements, and v appears once, the function should return the index of v.
If a has n elements and v appears multiple times, the function should return the index of the first occurrence of v.
If a has n elements, and v is not in a, the function should return -1.

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4, 5],v = 2	1
[1, 2, 2, 351, 551],v = 2	2
[1,22,33,44,55],v = 66	-1
[2, 4, 6, 8, 10],v = 51	-1
[-100, 0, 1000],v = -100	0
[-100, 0, 1000],v = 1000	2
[],v=0	-1
NULL,v = 4	-1

Program 4

Equivalence Partitioning:

Equilateral triangle (a=a, b=a, c=a): the expected outcome is EQUILATERAL (0)

Isosceles triangle (a=a, b=b, c=c): the expected outcome is ISOSCELES (1) Scalene triangle (a=b, b=c, c=a): the expected outcome is SCALENE (2) Invalid triangle (a=b+c): the expected outcome is INVALID (3)

Invalid triangle (a=b-c): the expected outcome is INVALID (3)

Invalid triangle (a=b+c-1): the expected outcome is INVALID (3)

Invalid triangle (a=-1, b=-1, c=-1): the expected outcome is INVALID (3)

Invalid triangle (a=0, b=0, c=0): the expected outcome is INVALID (3)

Invalid triangle (a=1, b=2, c=4): the expected outcome is INVALID (3)

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=1,b=1,c=1	EQUILATERAL
a=0,b=0,c=0	INVALID
a=-1,b=-1,c=-1	INVALID
a=10,b=10,c=0	INVALID
a=17,b=17,c=5	ISOSCELES
a=15,b=2,c=15	ISOSCELES
a=6,b=11,c=5	SCALENE
a=16,b=21,c=25	SCALENE
a=-1,b=21,c=25	INVALID
a=2,b=3,c=4	SCALENE

Program 5

Equivalence Partitioning:

s1 and s2 are both empty strings: false

s1 is empty and s2 is non-empty: true

s1 is non-empty and s2 is empty: false

s1 is a proper prefix of s2: true

s1 is not a prefix of s2: false

s1 and s2 are equal: true

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1 = "",s2 = ""	true
s1 = "po",s2 = "poojan"	true
s1 = "poo",s2 = "po"	false
s1 = "abc",s2 = ""	false
s1 = "",s2 = "abc"	true
s1 = "o",s2 = "ott"	true
s1 = "abc",s2 = "def"	false
s1 = "deg",s2 = "def"	false

Boundary Value Analysis:
s1 is one character shorter than s2: true
s1 is one character longer than s2: false
s1 and s2 have the same length: true

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1= "",s2 = ""	true

s1= "abcd",s2 = ""	false
s1= "",s2 = "abcd"	true
s1 = "aef",s2 = "def"	false
s1 = "def",s2 = "deg"	false
s1 = "a",s2 = "att"	true
s1 = "poojan",s2 = "patel"	false

Program 6

a) Equivalence classes: A, B, and C form a valid triangle
A, B, and C do not form a valid triangle

Class ID	Class
E1	All sides are positive
E2	two of its sides are zero
E3	One of its sides are negative
E4	Sum of two sides is less than third side
E5	Any of the side/sides is negative

b) Test cases:
A=4, B=4, C=4 (Equilateral triangle)
A=4, B=4, C=5 (Isosceles triangle)
A=4, B=5, C=6 (Scalene triangle)
A=3, B=4, C=5 (Right-angle triangle)
A=1, B=2, C=3 (Does not form a valid triangle)

Test Case ID	Class ID	Test Case
T1	E1	A = 1,B = 1,C = 1
T2	E1	A = 3, B = 4, C= 5
T3	E2	A = 0,B = 0,C = 1
T4	E3	A = 0,B = 1,C = 2
T5	E4	A = 1, B = 3, C = 8
T6	E5	A = -1,C = 1,D = 5

c) Test cases for boundary condition A+B>C:
A=0.1, B=0.2, C=0.3 (Smallest valid scalene triangle)
A=0.1, B=0.1, C=0.2 (Smallest invalid triangle)

d) Test cases for boundary condition
A=C: A=3, B=4, C=3 (Isosceles triangle with equal sides A and C)
A=0.1, B=0.2, C=0.1 (Smallest isosceles triangle with equal sides A and C)
A=1, B=2, C=1 (Smallest invalid triangle with equal sides A and C)

e) Test cases for boundary condition
A=B=C: A=5, B=5, C=5 (Equilateral triangle)
A=0.1, B=0.1, C=0.1 (Smallest equilateral triangle)
A=1, B=2, C=3 (Smallest invalid triangle with equal sides A, B, and C)

f) Test cases for boundary condition
A^2 + B^2 = C^2: A=3, B=4, C=5 (Right-angle triangle)
A=0.1, B=0.2, C=0.22361 (Smallest right-angle triangle)
A=1, B=1, C=1.41421 (Smallest invalid right-angle triangle)

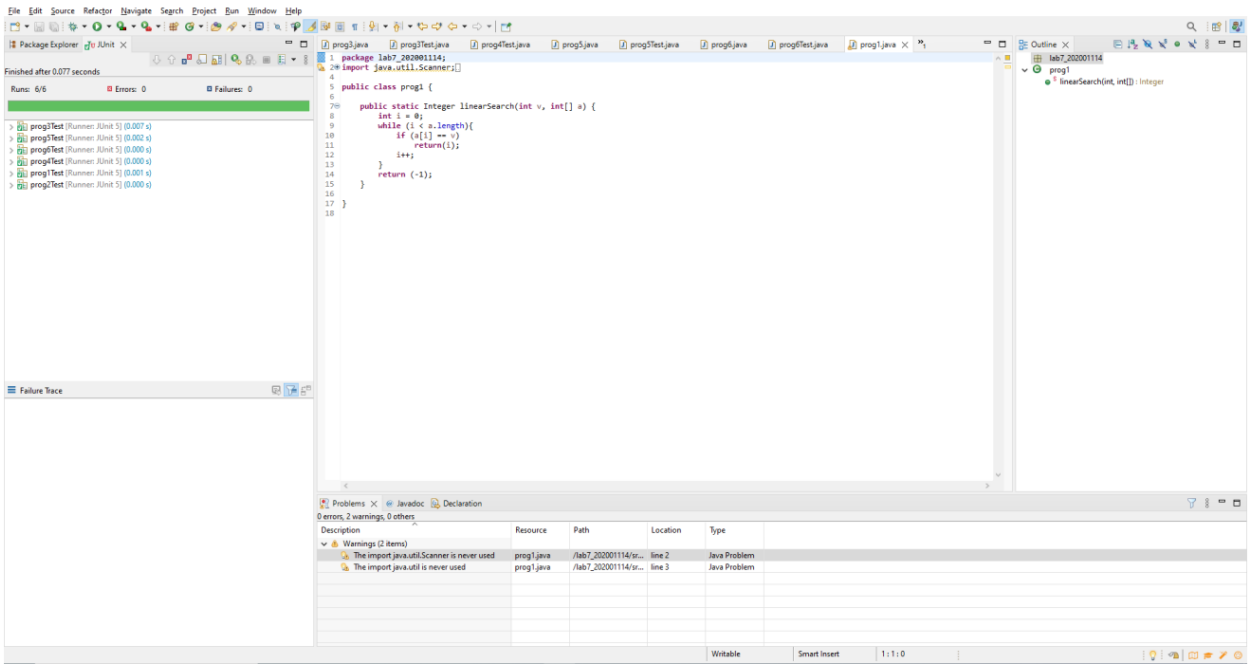
g) Test cases for non-triangle case:
A=1, B=2, C=10 (A + B < C)
A=1, B=10, C=2 (A + C < B)
A=10, B=1, C=2 (B + C < A)

h) Test cases for non-positive input:
A=-1, B=2, C=3
A=1, B=-2, C=3
A=1, B=2, C=-3
A=-1, B=-2, C=-3

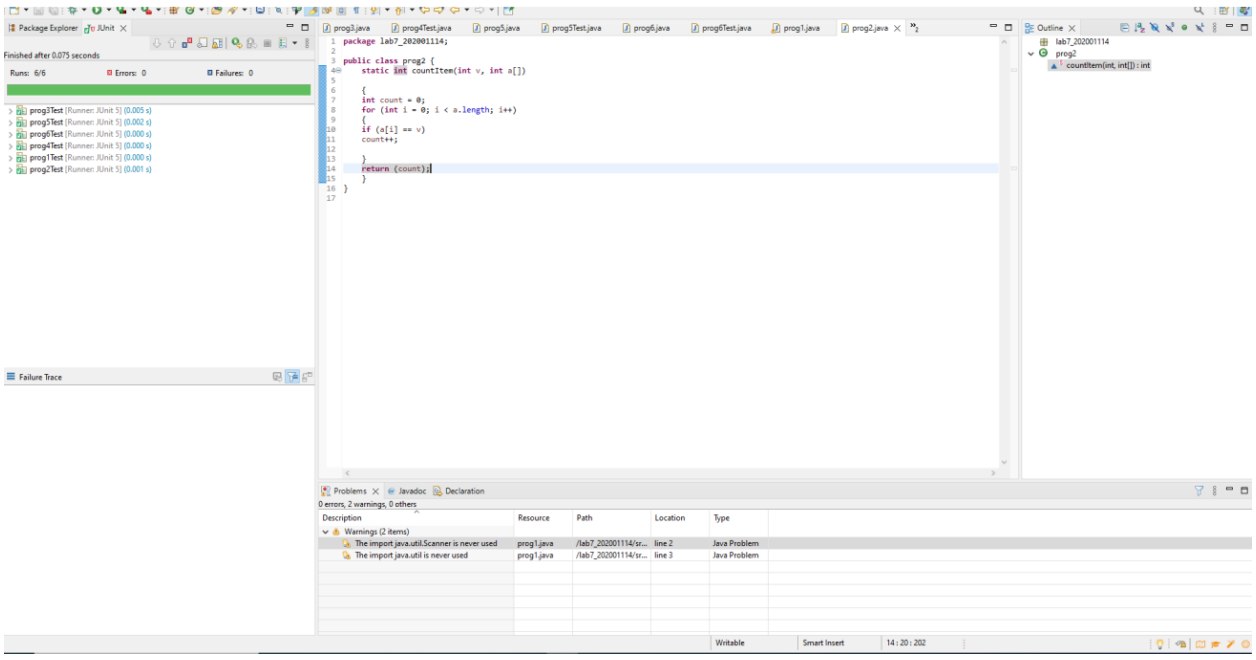
Screenshots:

Correct test cases for all programs:

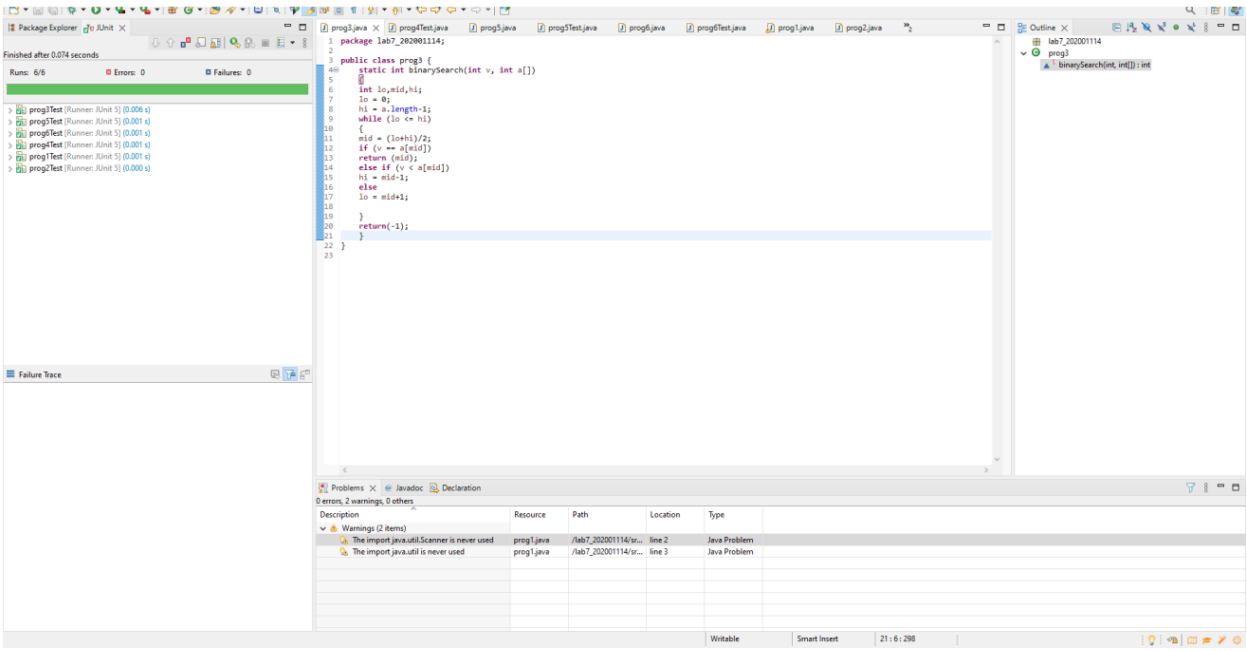
1.



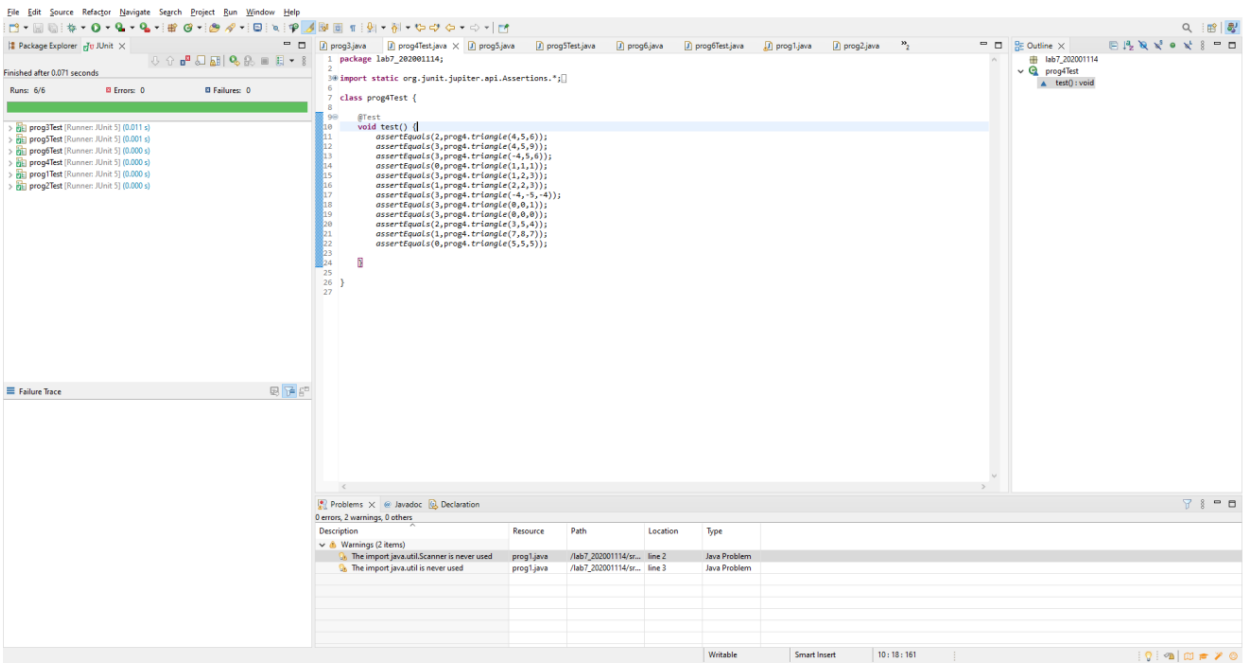
2.



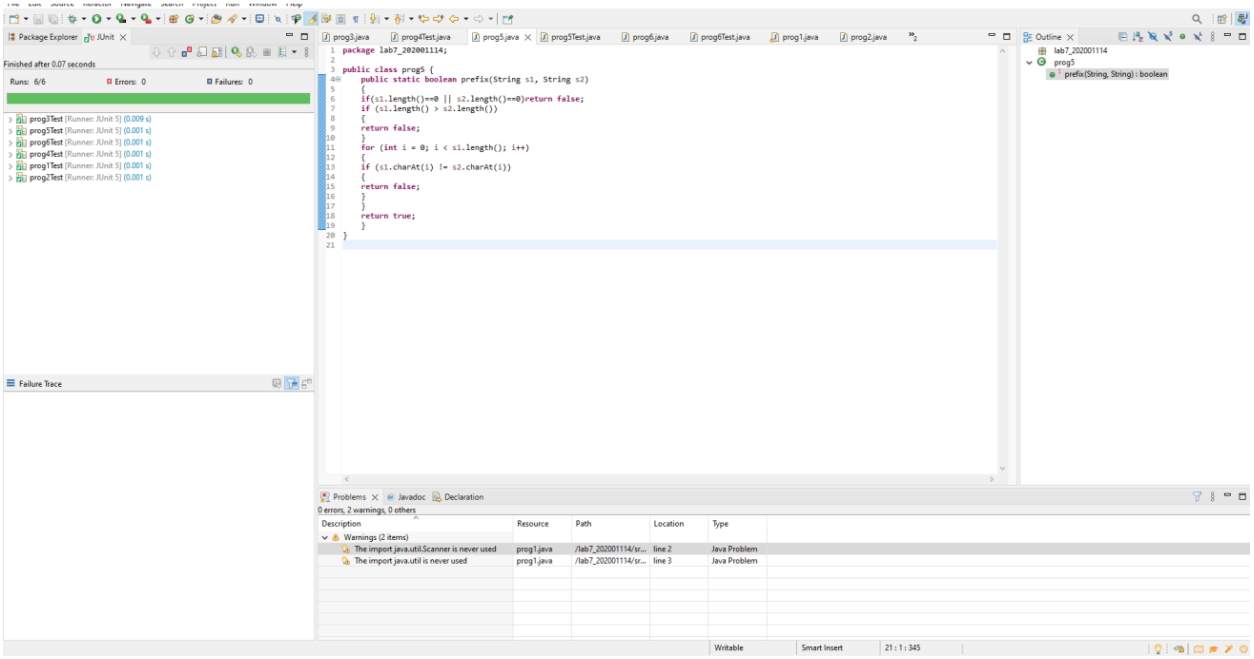
3.


Package Explorer
JUnit
Finished after 0.074 seconds
Runs: 6/6
Errors: 0
Failures: 0
prog3Test [Runner: JUnit 5] (0.006 s)
prog5Test [Runner: JUnit 5] (0.001 s)
prog6Test [Runner: JUnit 5] (0.001 s)
prog4Test [Runner: JUnit 5] (0.001 s)
prog1Test [Runner: JUnit 5] (0.001 s)
prog2Test [Runner: JUnit 5] (0.000 s)
Failure Trace
prog3.java
prog4Test.java
prog5.java
prog5Test.java
prog6.java
prog6Test.java
prog1.java
prog2.java
package lab7_202001114;
public class prog3 {
static 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);
}
}
Outline
lab7_202001114
prog3
binarySearch(int, int[]) : int
Problems
Javadoc
Declaration
0 errors, 2 warnings, 0 others
Description
Resource
Path
Location
Type
Warnings (2 items)
The import java.util.Scanner is never used
prog1.java
/lab7_202001114/sr...
line 2
Java Problem
The import java.util is never used
prog1.java
/lab7_202001114/sr...
line 3
Java Problem
Writable
Smart Insert
21 : 6 : 298

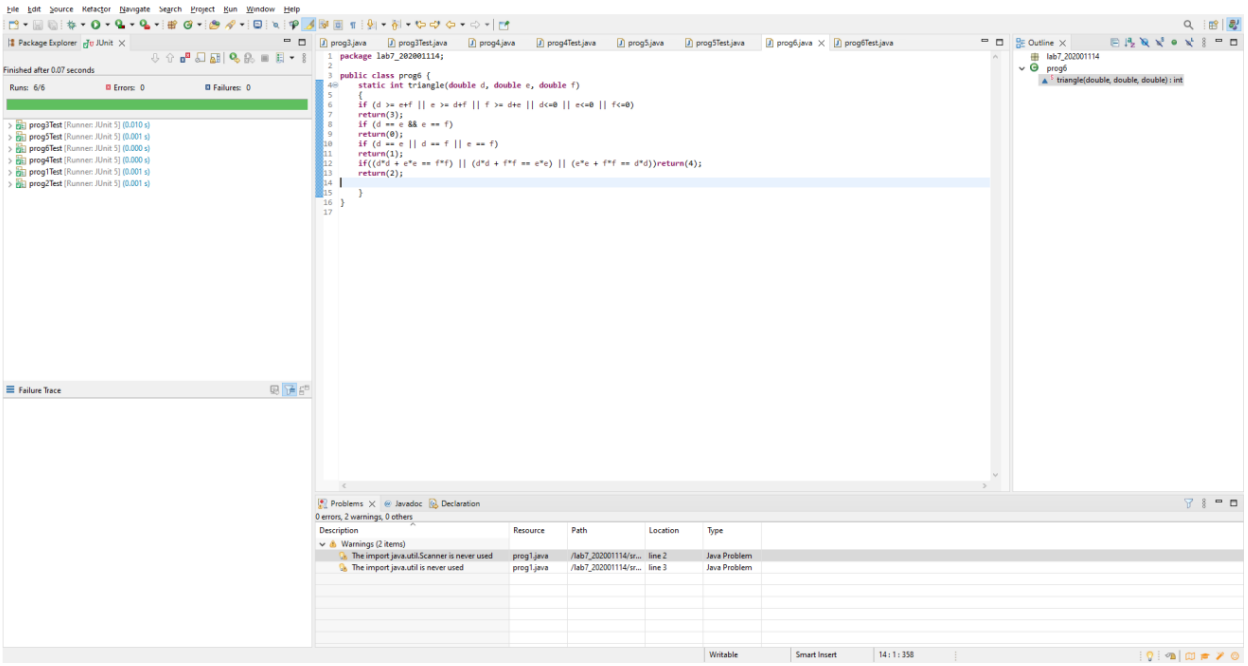
4.


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.071 seconds
Runs: 6/6
Errors: 0
Failures: 0
prog3Test [Runner: JUnit 5] (0.011 s)
prog5Test [Runner: JUnit 5] (0.001 s)
prog6Test [Runner: JUnit 5] (0.000 s)
prog4Test [Runner: JUnit 5] (0.000 s)
prog1Test [Runner: JUnit 5] (0.000 s)
prog2Test [Runner: JUnit 5] (0.000 s)
Failure Trace
prog3.java
prog4Test.java
prog5.java
prog5Test.java
prog6.java
prog6Test.java
prog1.java
prog2.java
package lab7_202001114;
import static org.junit.jupiter.api.Assertions.*;
class prog4Test {
@Test
void test() {
assertEquals(2,prog4.triangle(4,5,6));
assertEquals(3,prog4.triangle(4,5,9));
assertEquals(3,prog4.triangle(-4,5,6));
assertEquals(0,prog4.triangle(1,1,1));
assertEquals(3,prog4.triangle(1,2,3));
assertEquals(1,prog4.triangle(2,2,3));
assertEquals(3,prog4.triangle(-4,-5,-4));
assertEquals(3,prog4.triangle(0,0,1));
assertEquals(3,prog4.triangle(0,0,0));
assertEquals(2,prog4.triangle(3,5,4));
assertEquals(1,prog4.triangle(7,8,7));
assertEquals(0,prog4.triangle(5,5,5));
}
}
Outline
lab7_202001114
prog4Test
test() : void
Problems
Javadoc
Declaration
0 errors, 2 warnings, 0 others
Description
Resource
Path
Location
Type
Warnings (2 items)
The import java.util.Scanner is never used
prog1.java
/lab7_202001114/sr...
line 2
Java Problem
The import java.util is never used
prog1.java
/lab7_202001114/sr...
line 3
Java Problem
Writable
Smart Insert
10 : 18 : 161

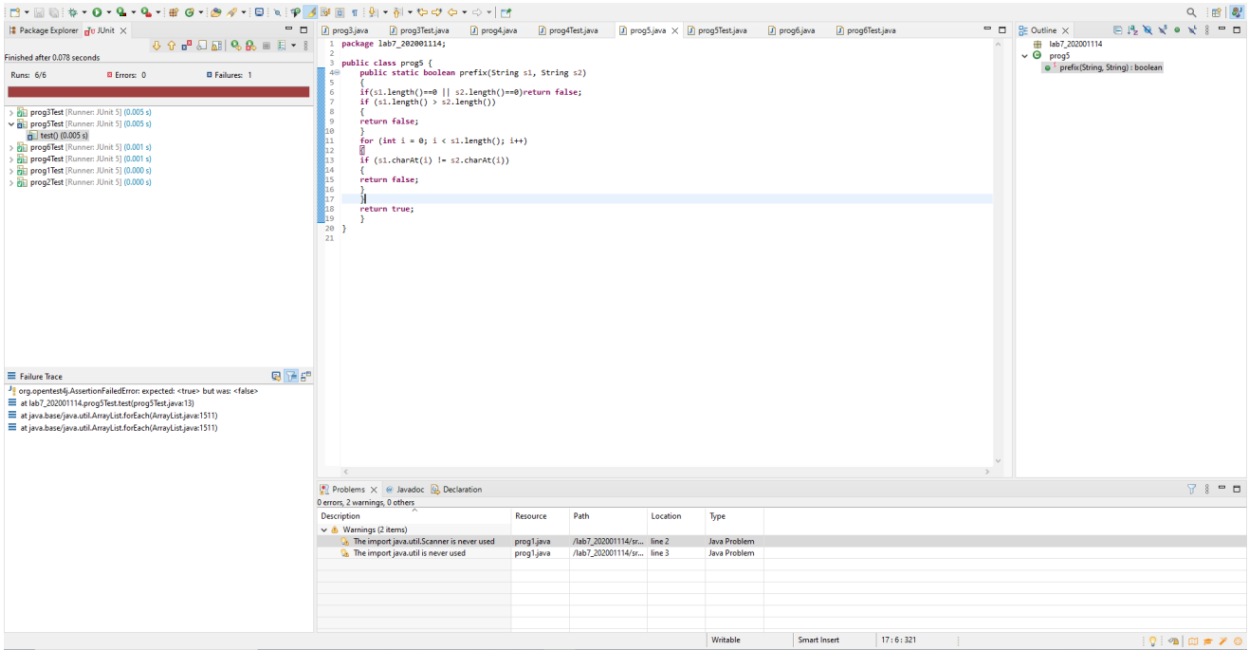
5.



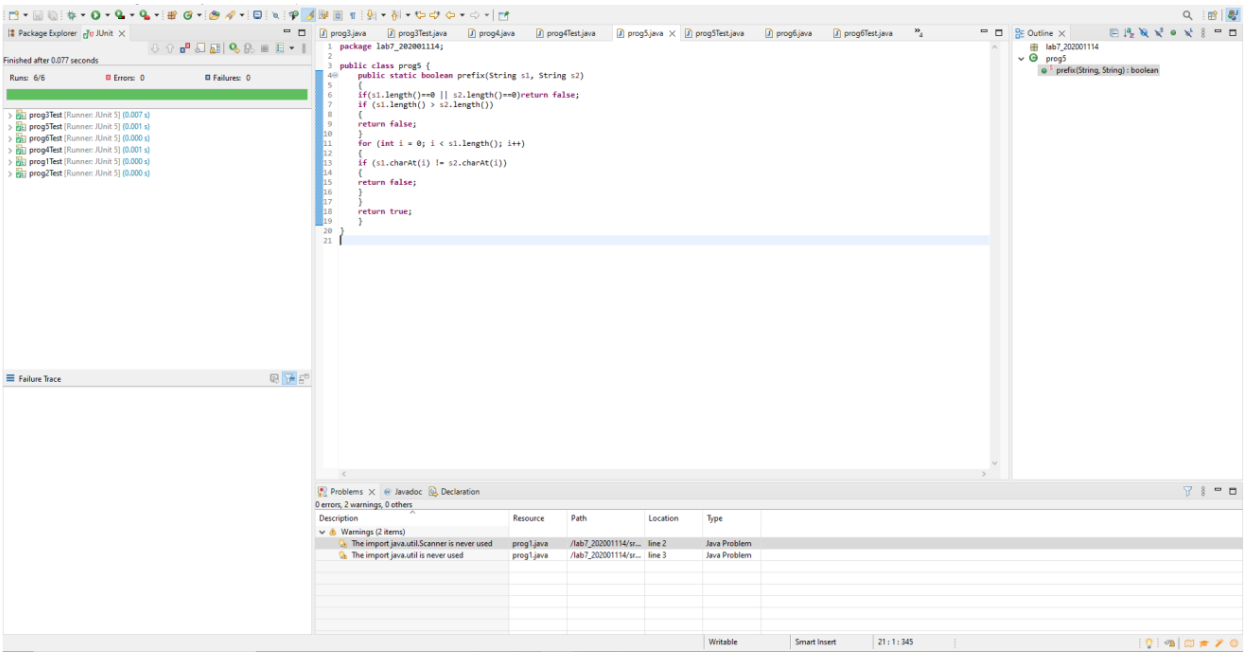
6.



On deliberately reversing a test case output in program 5:

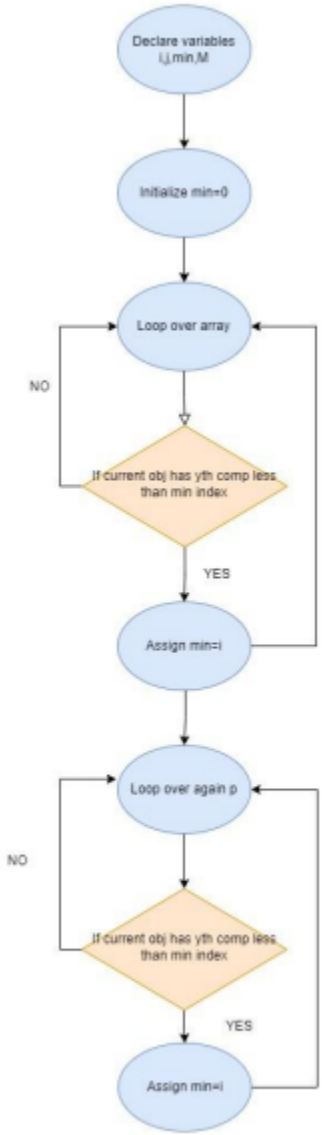


Again reversing the incorrect test case output in program 5 and running:



Section- B

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.
-> Statement Coverage

Test Number	Test Case
1	p is an empty array
2	p has one point object
3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component

->Branch Coverage

Test Number	Test Case
1	p is an empty array
2	p has one point object

3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component
6	p has three or more point object with same y component
7	p has three or more point object with all same x component
8	p has three or more point object with all different x component
9	p has three or more point object with some same and some different x component

-> Basic Condition Coverage

Test Number	Test Case
1	p is empty array
2	p has one point object
3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component
6	p has three or more point object with same y component

7	p has three or more point object with all same x component
8	p has three or more point object with all different x component
9	p has three or more point object with some same and some different x component
10	p has three or more point object with some same and some different y component
11	p has three or more point object with all different y component
12	p has three or more point object with all same y component