Lab-7_202001101
Section A:

1)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?

SOLUTION:

To design the equivalence class test cases for the program, we need to consider the input ranges and identify the valid and invalid equivalence classes. Here are the equivalence classes for the input parameters:

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)

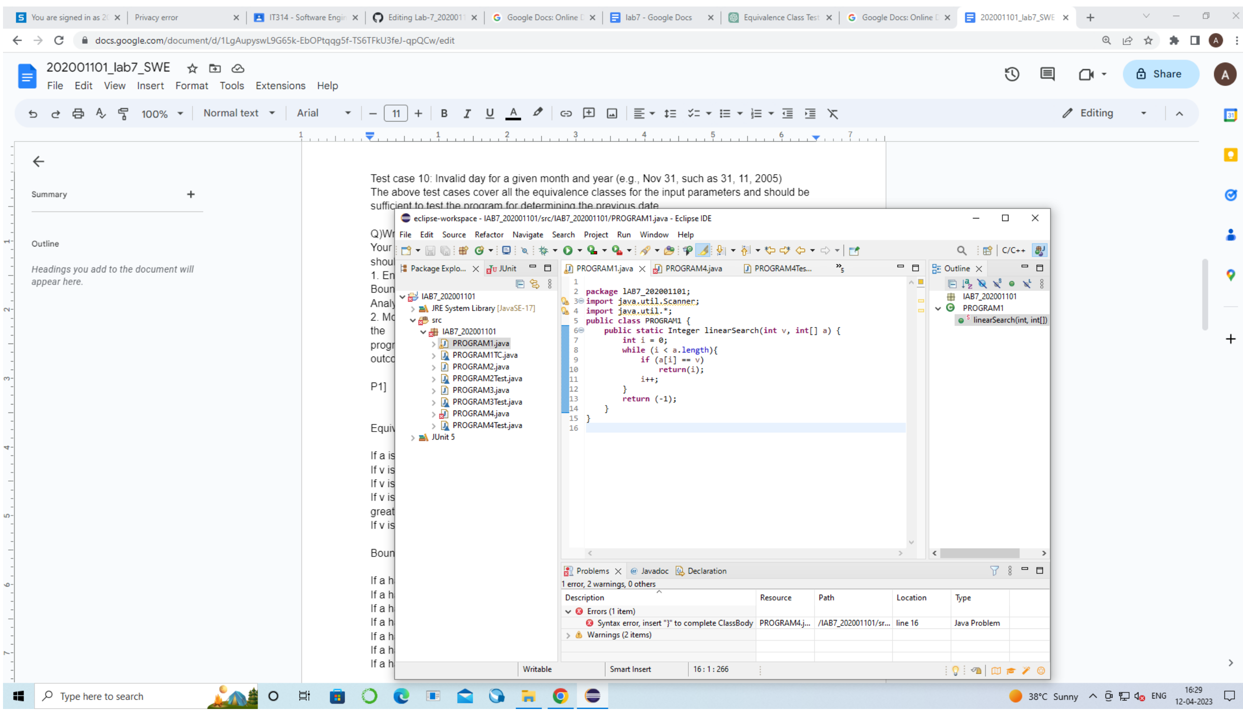
Invalid equivalence class test cases:

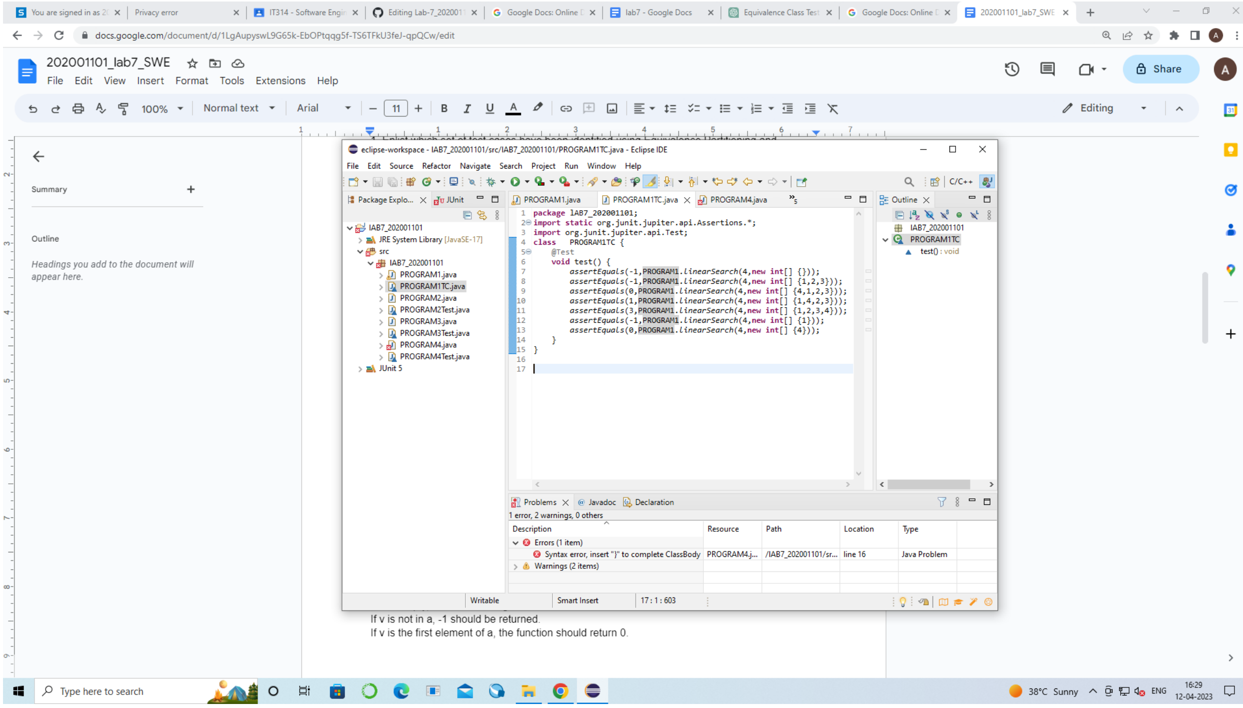
Test case 4: Invalid day, month, and year (e.g., 0, 0, 1899)
Test case 5: Invalid day, month, and year (e.g., 32, 13, 2016)
Test case 6: Invalid day for a given month and year (e.g., Feb 29 in a non-leap year, such as 29, 2, 2001)
Test case 7: Invalid day for a given month and year (e.g., Apr 31, such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

Q)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.

P1]





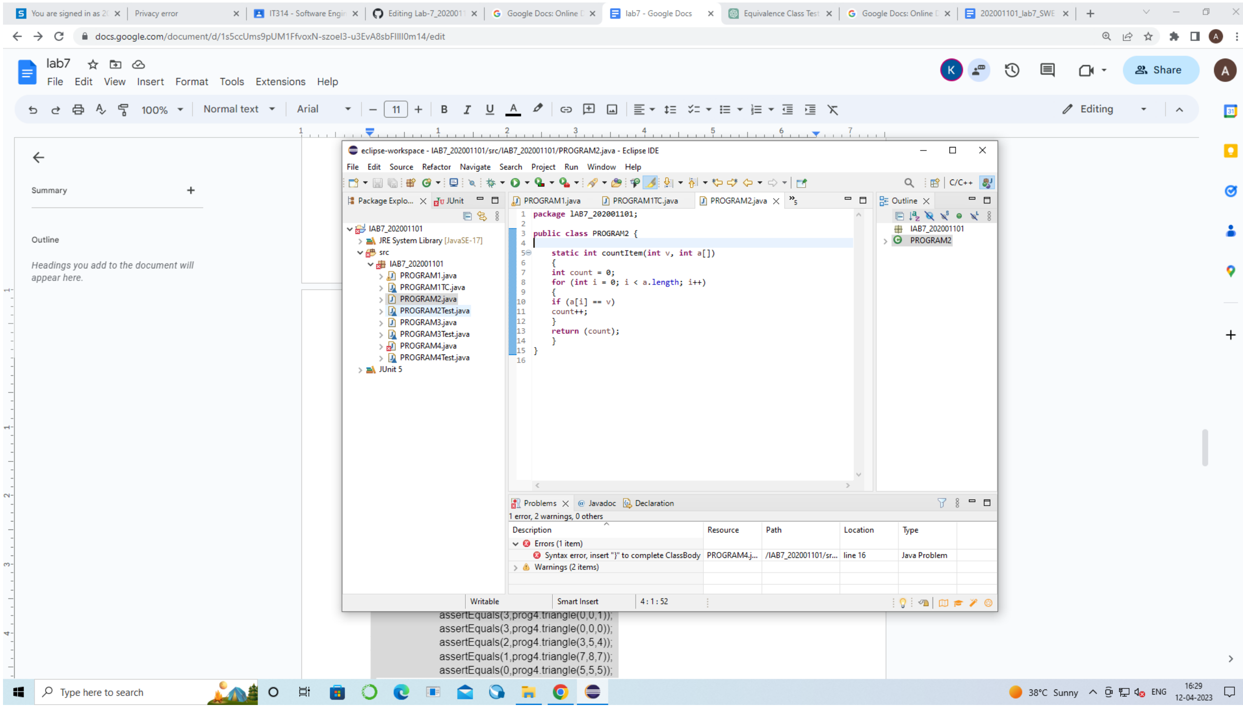
Equivalence Partitioning:

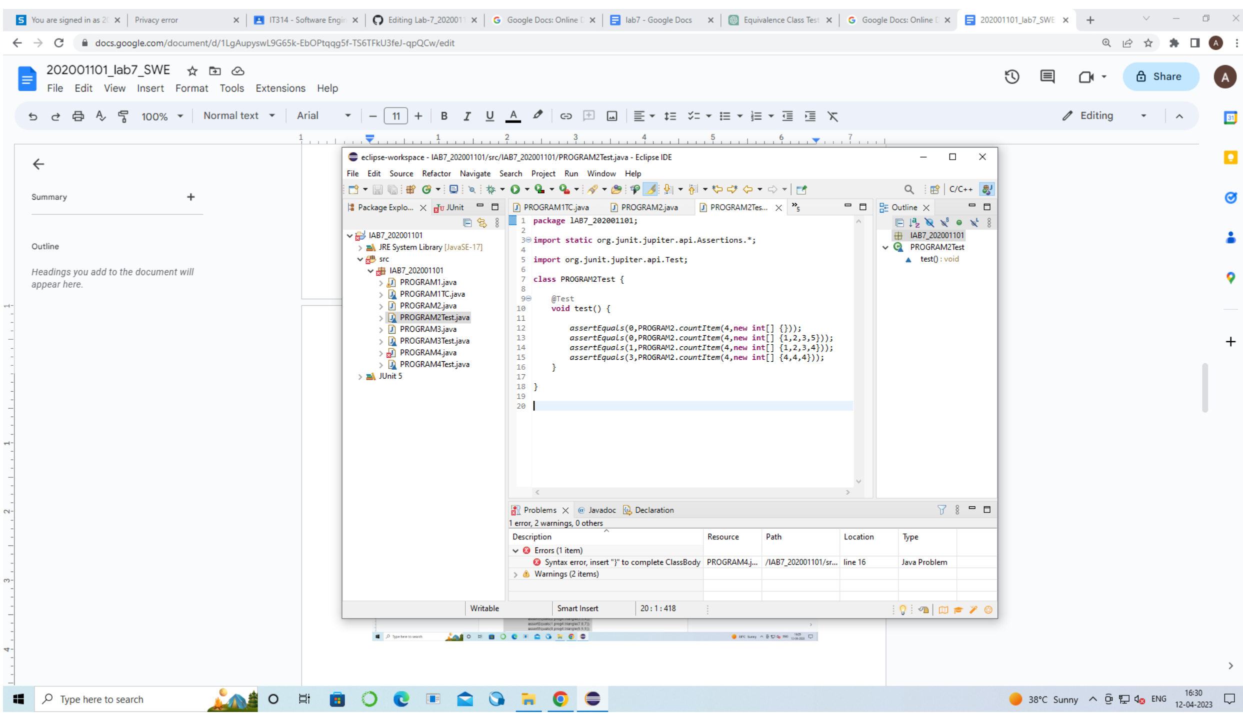
If a is empty, an error message should be returned.
If v is not in a, -1 should be returned.
If v is the first element of a, the function should return 0.
If v is in the middle of a, the function should return the first index i such that a[i] == v, where i is greater than 0 and less than a.length - 1.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

Boundary Value Analysis:

If a has one element and it's 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.

P2]





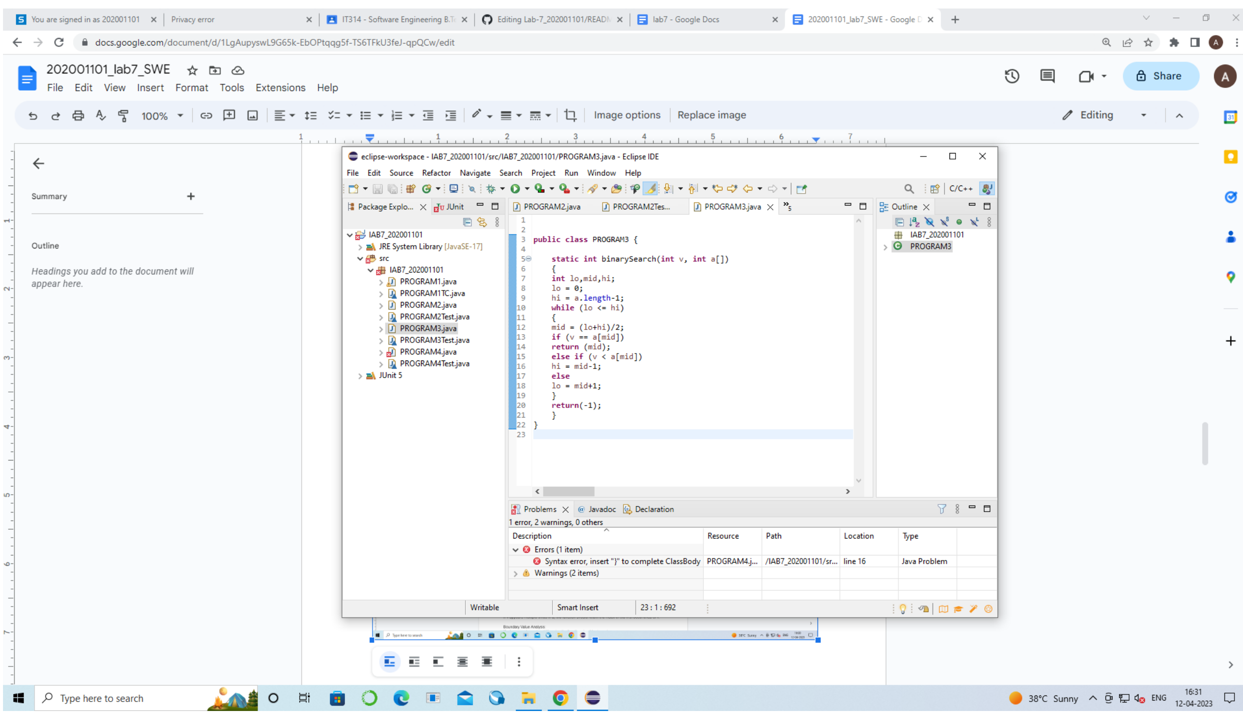
Equivalence Partitioning:

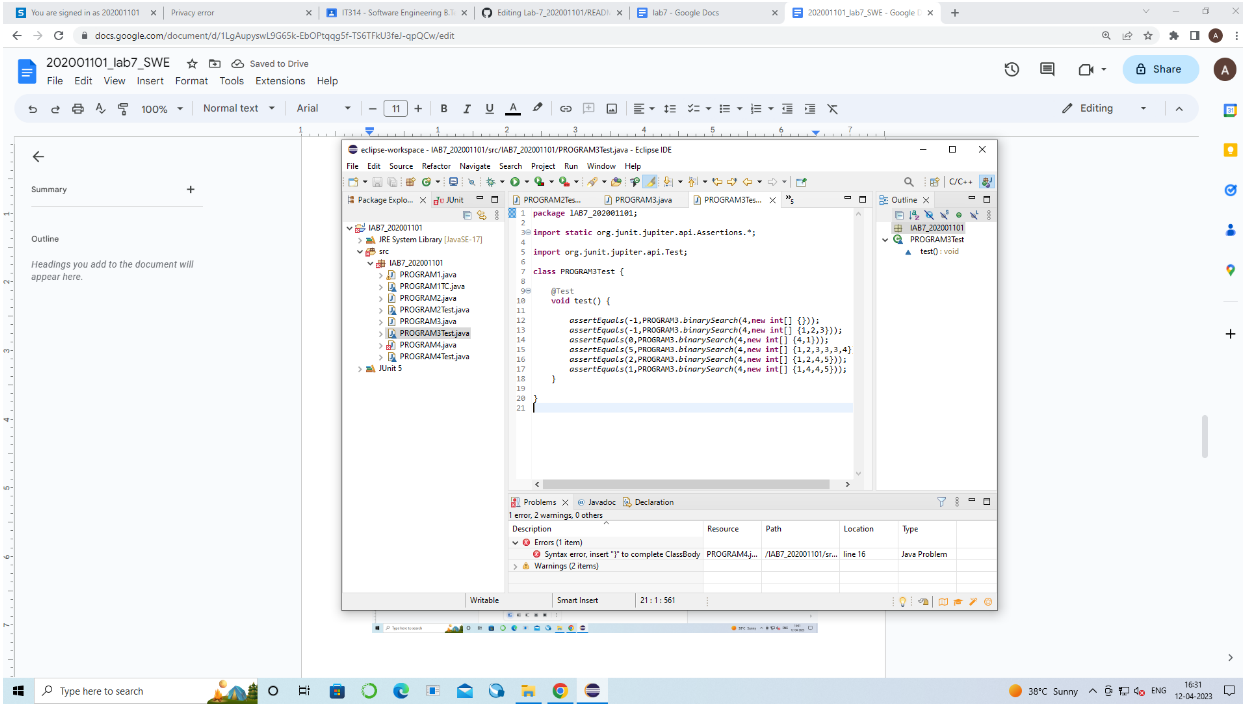
If a is empty, the function should return 0.
If v is not in a, the function should return 0.
If v appears once in a, the function should return 1.
If v appears multiple times in a, the function should return the number of occurrences of v.

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.

P3]

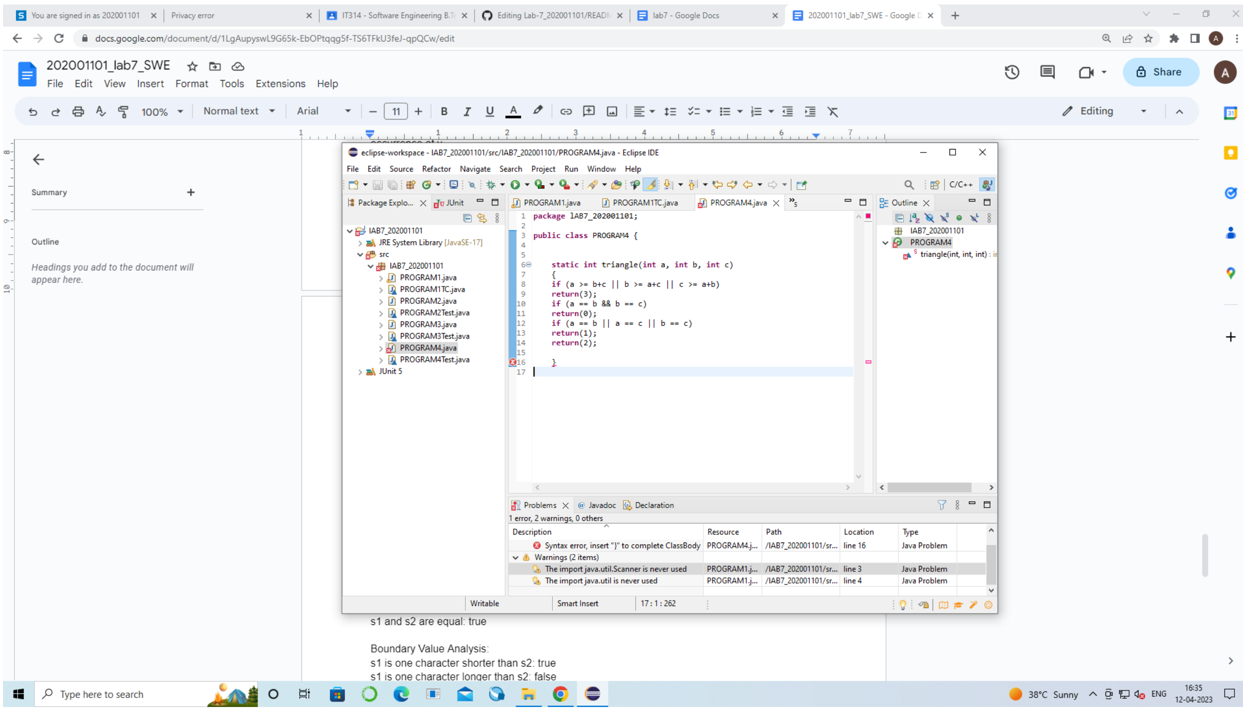


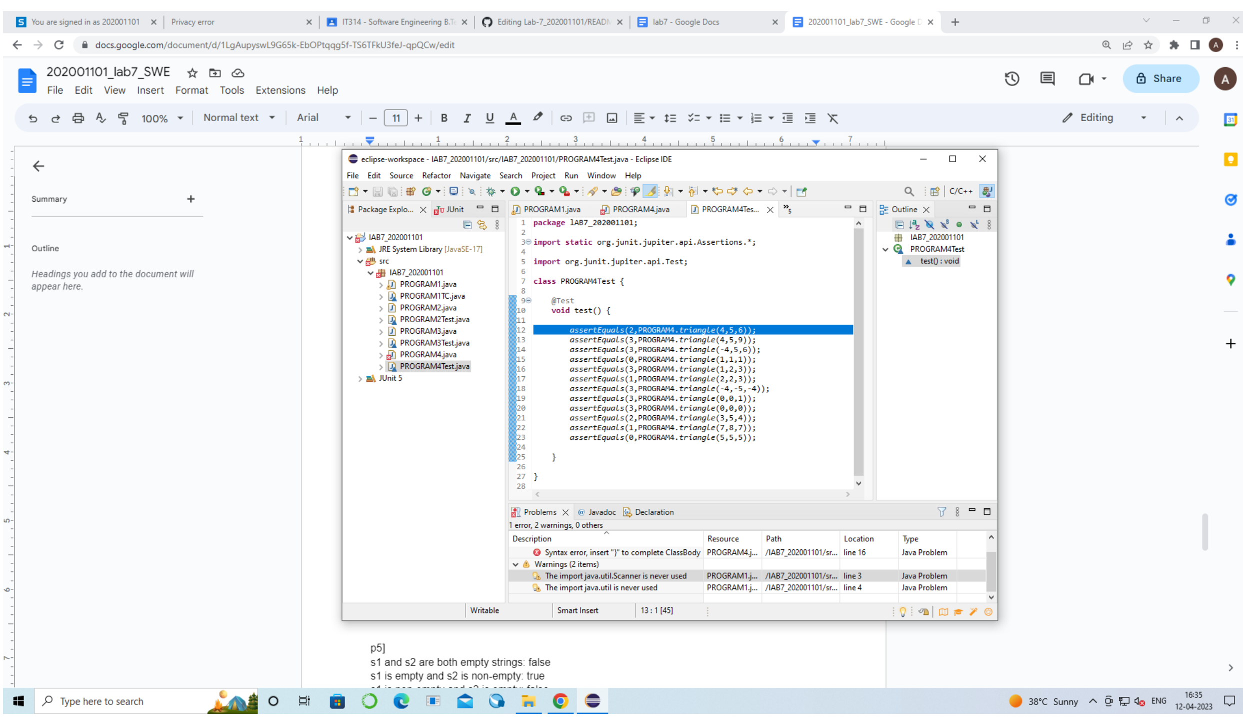


Equivalence Partitioning:
If a is empty, the function should return -1.
If v is not in a, the function should return -1.
If v is the first element in a, the function should return 0.
If v is the last element in a, the function should return a.length-1.
If v appears once in a, the function should return the index of v.
If v appears multiple times in a, the function should return the index of the first occurrence of v.

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.

P4]





Equilateral triangle (a=a, b=a, c=a): expected outcome is EQUILATERAL (0)
Isosceles triangle (a=a, b=b, c=c): expected outcome is ISOSCELES (1)
Scalene triangle (a=b, b=c, c=a): expected outcome is SCALENE (2)
Invalid triangle (a=b+c): expected outcome is INVALID (3)

Invalid triangle (a=b-c): expected outcome is INVALID (3)
Invalid triangle (a=b+c-1): expected outcome is INVALID (3)
Invalid triangle (a=-1, b=-1, c=-1): expected outcome is INVALID (3)
Invalid triangle (a=0, b=0, c=0): expected outcome is INVALID (3)
Invalid triangle (a=1, b=2, c=4): expected outcome is INVALID (3)

p5]
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

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

Test cases code:

```
package lab7_202001101;

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

import org.junit.jupiter.api.Test;

class prog5Test {

	@Test
	void test() {
		assertEquals(false,prog5.prefix("",""));
		assertEquals(false,prog5.prefix("","a"));
		assertEquals(false,prog5.prefix("a",""));
		assertEquals(false,prog5.prefix("abc","acbd"));
		assertEquals(true,prog5.prefix("abc","abcde"));
		assertEquals(false,prog5.prefix("abc","ab"));
		assertEquals(false,prog5.prefix("abc","a"));
		assertEquals(true,prog5.prefix("ab","ab"));

	}

}
```

p6]a) Equivalence classes for the system:

Invalid triangle (a >= b+c or b >= a+c or c >= a+b)
Scalene triangle (a < b+c and b < a+c and c < a+b, and a != b != c)
Isosceles triangle (a < b+c and b < a+c and c < a+b, and (a == b and a != c) or (a == c and a != b) or (b == c and b != a))
Equilateral triangle (a < b+c and b < a+c and c < a+b, and a == b == c)
Right-angled triangle (a < b+c and b < a+c and c < a+b, and $a^2 + b^2 = c^2$ or $a^2 + c^2 = b^2$ or $b^2 + c^2 = a^2$)
b) Test cases to cover the identified equivalence classes:

Invalid triangle:
a. (-1, 2, 3)
b. (2, 2, 4)
Scalene triangle:
a. (3, 4, 5)
b. (6, 7, 8)
Isosceles triangle:
a. (3, 3, 4)
b. (5, 7, 5)
Equilateral triangle:
a. (1, 1, 1)
b. (2.5, 2.5, 2.5)
Right-angled triangle:
a. (3, 4, 5)
b. (5, 12, 13)
c) Test cases to verify the boundary A + B > C for scalene triangle:
a. (2, 2, 4)
b. (1.5, 2.5, 3)

d) Test cases to verify the boundary A = C for isosceles triangle:
a. (2, 3, 2)
b. (5, 2, 5)

e) Test cases to verify the boundary A = B = C for equilateral triangle:
a. (1, 1, 1)
b. (2.5, 2.5, 2.5)

f) Test cases to verify the boundary $A^2 + B^2 = C^2$ for right-angled triangle:
a. (3, 4, 5)
b. (5, 12, 13)

g) Test cases to explore the boundary for non-triangle:
a. (0, 0, 0)

b. (1, 2, 3)
c. (3, 3, 6)

h) Test points for non-positive input:
a. (-1, 2, 3)
b. (0, 0, 0)
c. (-2, -3, -4)

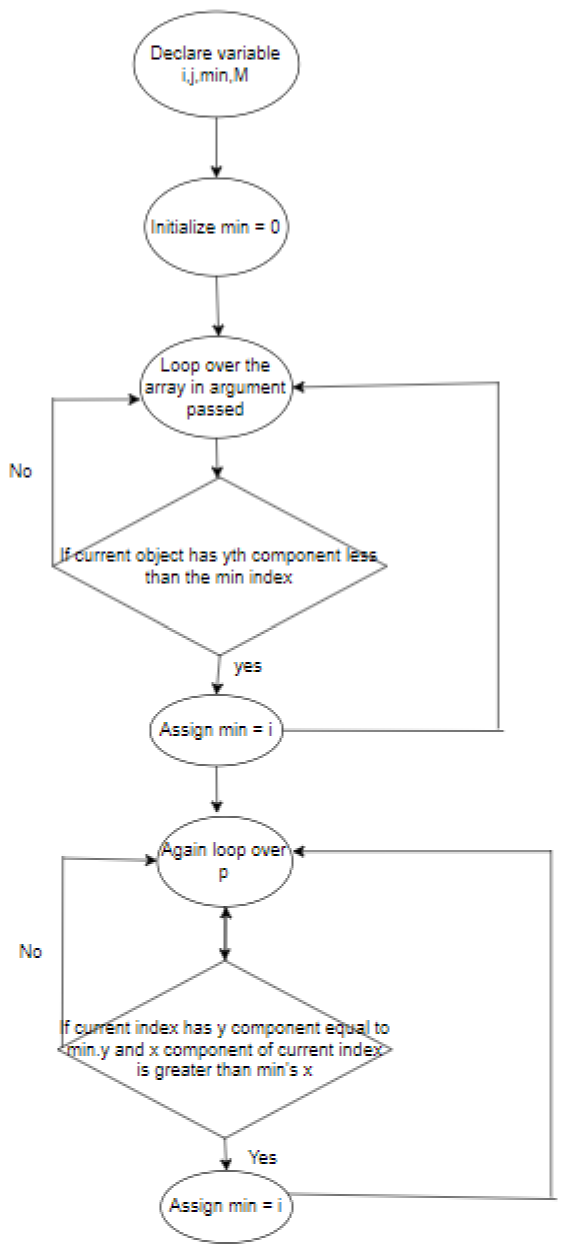
Test Case Code:

```
package lab7_202001101;

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

import org.junit.jupiter.api.Test;

class prog6Test {

	@Test
	void test() {
		assertEquals(2,prog6.triangle(4,5,6));
		assertEquals(4,prog6.triangle(4,5,3));
		assertEquals(0,prog6.triangle(1,1,1));
		assertEquals(3,prog6.triangle(4,5,1));
		assertEquals(3,prog6.triangle(4,5,-3));
		assertEquals(1,prog6.triangle(4,3,3));
		assertEquals(2,prog6.triangle(0.1,0.2,0.3));
		assertEquals(3,prog6.triangle(0.1,0.2,0.1));
		assertEquals(2,prog6.triangle(0.1,0.2,0.22361));
		assertEquals(0,prog6.triangle(0.1,0.1,0.1));
		assertEquals(3,prog6.triangle(-0.1,0.2,0.3));

	}

}
```

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.



(2) Test Cases ** **(a) Statement coverage test set: ** In this all the statements in code should be covered

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

b)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

c)

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