IT314
Software
Engineering LAB - 7

Name : Harshit Vadodia
ID : 202001426

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?

Answer:

Equivalence Classes:-

1. Months are less than 1.
2. Months are between 1 and 12.
3. Months are greater than 12.
4. Day is less than 1.
5. Day is between 1 and 31.
6. Day is greater than 31.
7. Year is less than 1900.
8. Year is between 1900 and 2015.
9. Year is greater than 2015.

No.	Test Case	Valid / Invalid	Equivalence class
1	22-10-2010	Valid	C2,C5,C8
2	2-0-2001	Invalid	C1
3	2-13-1999	Invalid	C3
4	0-10-2002	Invalid	C4
5	1-6-1899	Invalid	C7
6	2-10-2022	Invalid	C9
7	5-6-2007	Valid	C2,C5,C8
8	32-10-2009	Invalid	C6
9	2-10-2017	Valid	C2,C5
10	31-2-2000	Valid	C2,C5,C8

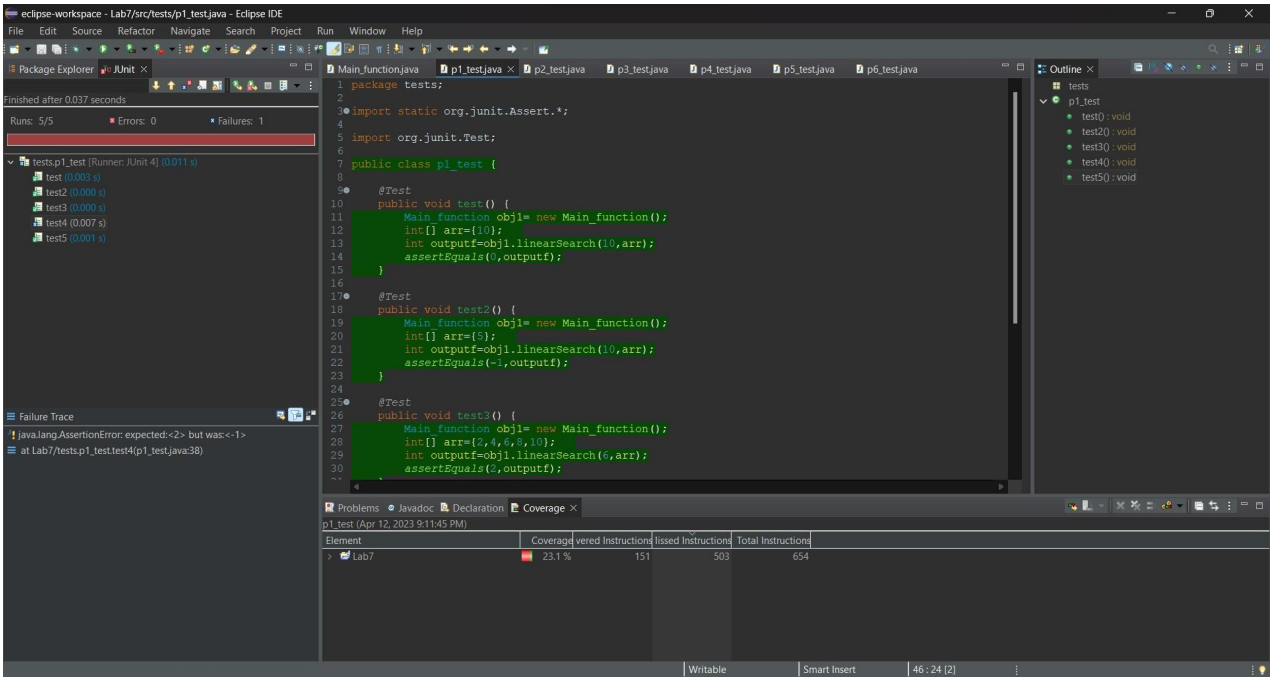
Programs:

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.

```
int linearSearch(int v, int a[])
{
	int i = 0; (1)
	while (i < a.length) (2)
	{
		if (a[i] == v) (3)
			return(i); (4)
		i++; (5)
	}
	return (-1); (6)
}
```

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

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a = [10], v = 10	0
a = [5], v = 10	-1
a = [2, 4, 6, 8, 10], v = 6	2
a = [], v = 10	-1
a = [1, 3, 5, 7, 9], v = 4	-1
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 = 9999	9999
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
Search value not in the array, but adjacent to an element in the array. Input: a = [10, 20, 30, 40, 50], v = 35	-1

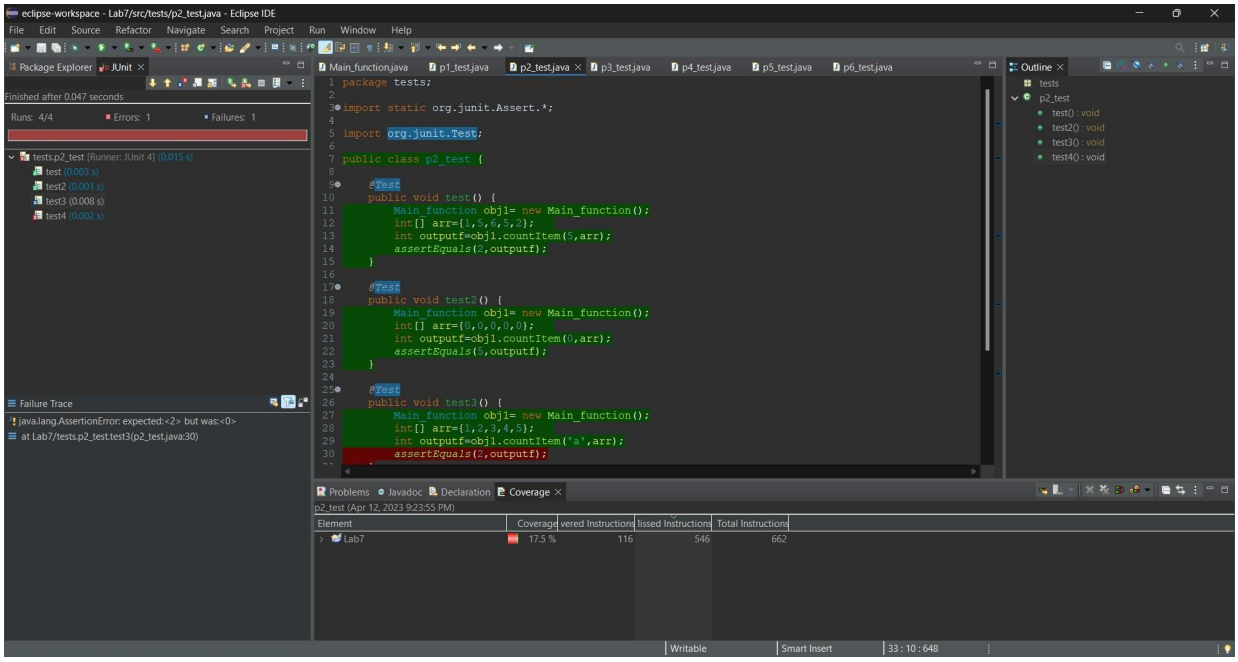


P2:The function countItem returns the number of times a value v appears in an array of integers a.

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

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

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
v = 5, a = {1, 5, 6, 5, 2}	2
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	
v = 0, a = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	10
v = -2147483648, a = {-2147483648, 1, 2, 3, 4}	1
Invalid Input: v = 6, a = {}	0
v = 3, a = {1, 2, 3, 4, 5}	1

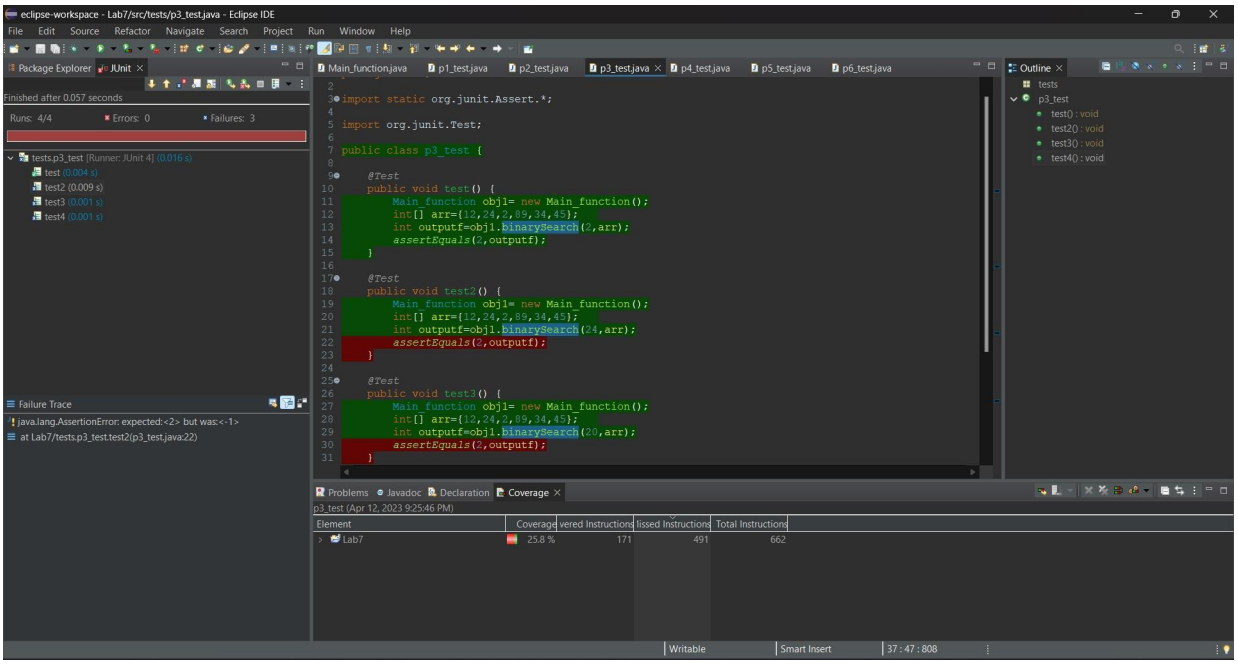


P3: The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.
Assumption: the elements in the array a are sorted in non-decreasing order.

```
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);
}
```

```java
public 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;
}
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a = [10], v = 10	0
a = [5], v = 10	-1
a = [2, 4, 6, 8, 10], v = 6	2
a = [], v = 10	-1
a = [1, 3, 5, 7, 9], v = 4	-1
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 = 9999	9999
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
Search value not in the array, but adjacent to an element in the array. Input: a = [10, 20, 30, 40, 50], v = 35	-1



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).

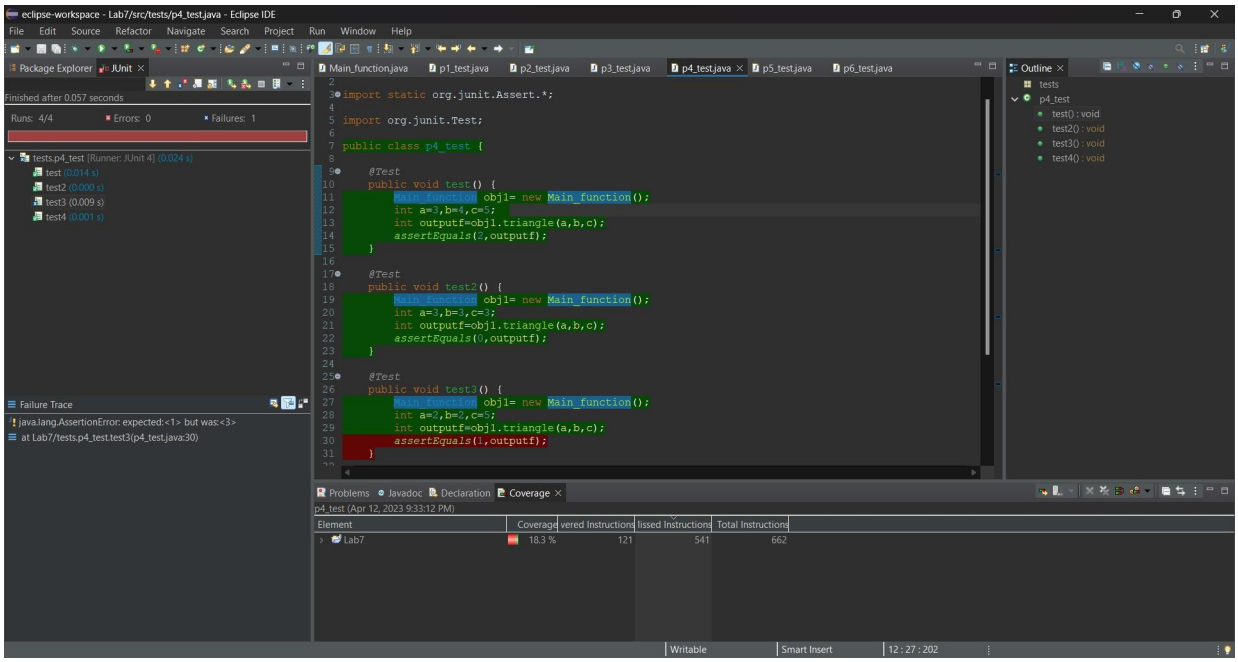
```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
        if (a >= b+c || b >= a+c || c >= a+b)
                return(INVALID);
        if (a == b && b == c)
```

```
        return(EQUILATERAL)
    ; if (a == b || a == c || b == c)
        return(ISOSCELES);
    return(SCALENE);

}
```

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

    public 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;
    }
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=b=c=0	Invalid
a=b=c=2	Equilateral
a=b>c	Isoscale
a+b=c	Invalid
a+b>c	Scalene
Boundary Value Analysis	
a=b=c=200	Equilateral

a = 1, b = 2, c = 4	Invalid
a=2,b=2,c=3	Isoscale



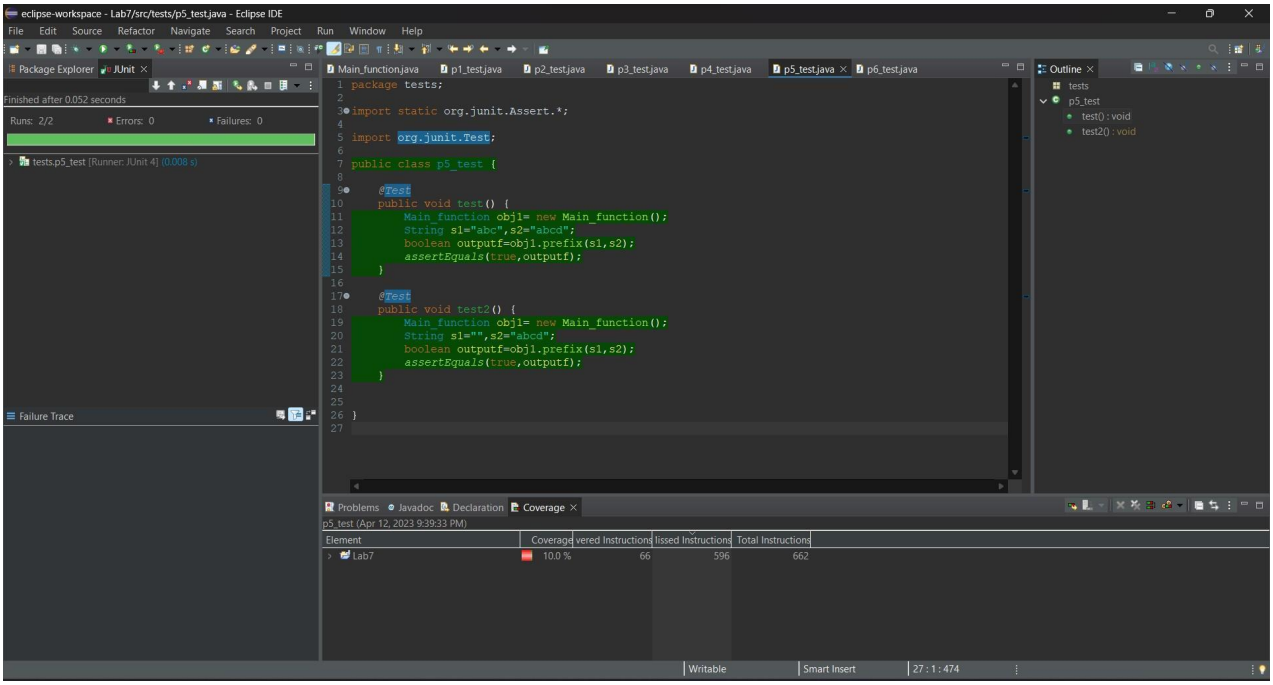
P5: The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix
of string s2 (you may assume that neither s1 nor s2 is null).

```
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;
}
```

```java
public  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;
}
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
s1= “”, s2= “abc”	true
s1= “abc”, s2= “abcd”	true
s1= “bcd”, s2= “abcd”	false
Boundary Value Analysis	
s1= “a”, s2= “abc”	true
s1= “abc”, s2= “abc”	true
s1= “abcd”, s2= “abc”	false



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

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)

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

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

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

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

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

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

Ans:

a) Equivalence Classes:

1) Invalid inputs: When any of the input values are non-numeric or negative.
2) Non-triangle: When the sum of the lengths of any two sides is less than or equal to the length of the third side.
3) Equilateral triangle: When all sides are equal in length.
4) Isosceles triangle: When two sides are equal in length and the third side is different.
5) Scalene triangle: When all sides are different in length.
6) Right-angled triangle: When the sum of the squares of the lengths of the two shorter sides is equal to the square of the length of the longest side.

b) Test cases:

1. Invalid inputs: "a", -5, "c"

2. Non-triangle: 1, 2, 5
3. Equilateral triangle: 3, 3, 3
4. Isosceles triangle: 5, 7, 5
5. Scalene triangle: 3, 4, 5
6. Right-angled triangle: 3, 4, 5

c) Test cases for boundary condition A + B > C:

1. 1, 2, 3
2. 3, 4, 7
3. 5, 6, 11

d) Test cases for boundary condition A = C:

1. 1, 2, 2
2. 4, 5, 4
3. 6, 7, 6

e) Test cases for boundary condition A = B = C:

1. 0.5, 0.5, 0.5
2. 1, 1, 1
3. 10, 10, 10

f) Test cases for boundary condition A^2 + B^2 = C^2:

1. 3, 4, 5
2. 5, 12, 13
3. 7, 24, 25

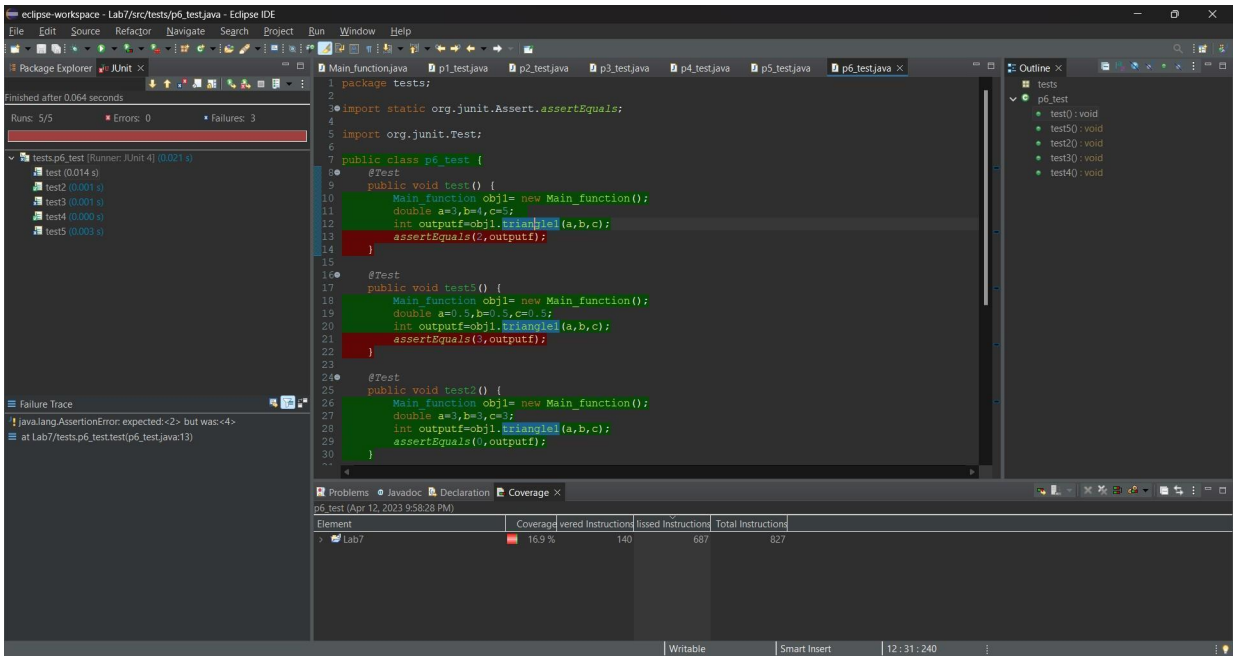
g) Test cases for non-triangle:

1. 1, 2, 3
2. 2, 3, 5
3. 4, 5, 9

h) Test cases for non-positive input:

1. 0, 1, 2
2. -1, -2, -3

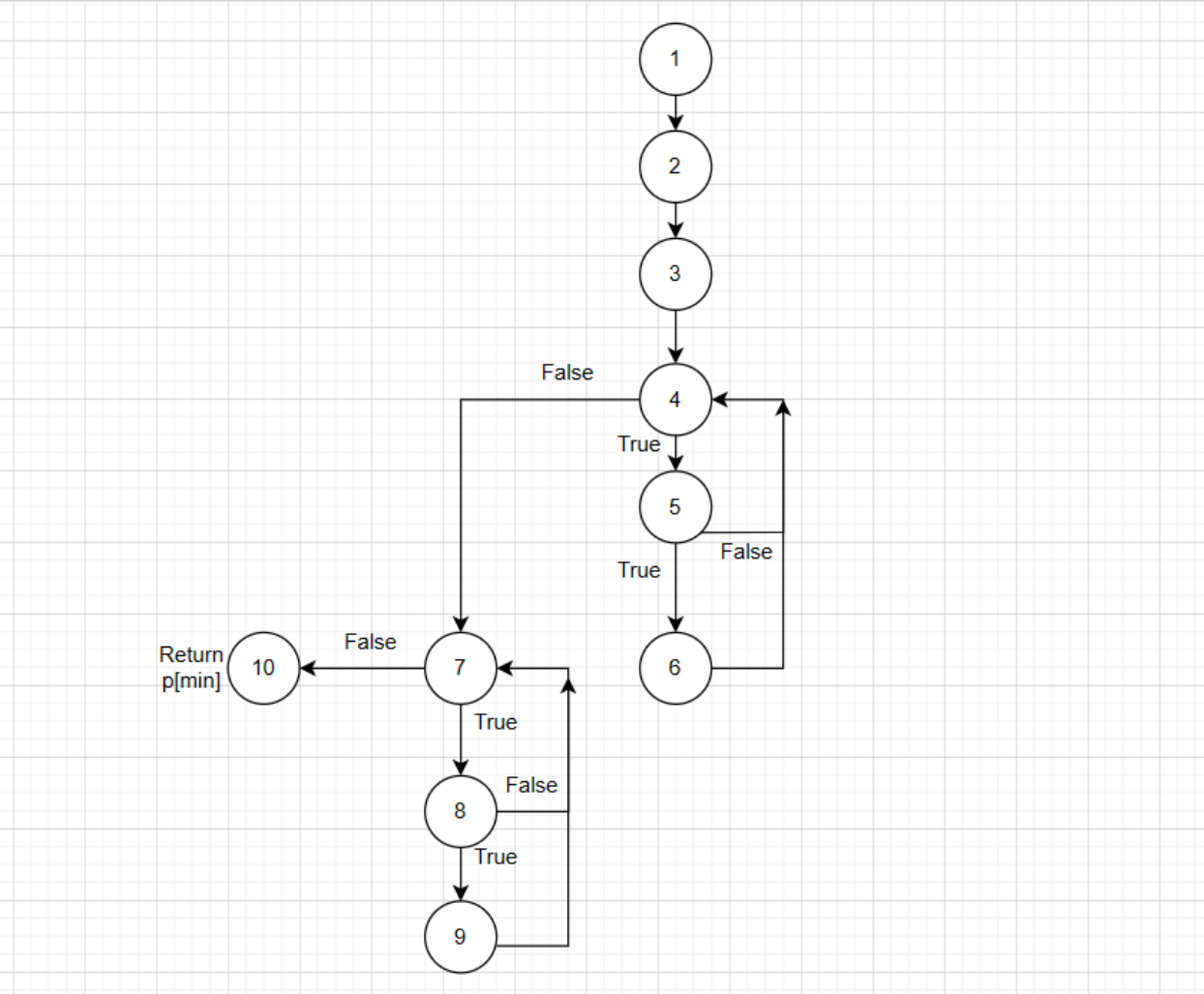
```java
    public static final int EQUILATERAL1 = 0;
    public static final int ISOSCELES1 = 1;
    public static final int SCALENE1 = 2;
    public static final int INVALID1 = 3;
    public static final int RIGHT_ANGLE1 = 4;

    public int triangle1(double a, double b, double c) {
        if(a*a + b*b == c*c) return RIGHT_ANGLE1;
        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;
    }
```



Section B:

Below is the java code of pseudo code given in the question:

```
public class Point {
    public double x;
    public double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}

public Point doGraham(Point[] p) {
    int i, j, min, M; //1
    Point t; //2
    min = 0; //3
    for (i = 1; i < p.length; i++) { //4
        if (p[i].y < p[min].y) { //5
            min = i; //6
        }
    }
    for (i = 0; i < p.length; i++) { //7
        if ((p[i].y == p[min].y) && (p[i].x > p[min].x)) { //8
            min = i; //9
        }
    }
    return p[min]; //10
}
```



a. Test set for Statement Coverage:

The test set should cover every statement in the code at least once.
Test Set:

- p = new Point[]{new Point(0,0), new Point(1,1)}
- doGraham(p)

Explanation:

This test set contains two points, one with coordinates (0,0) and the other with coordinates (1,1). The doGraham() method is called with these two points as input. This test set will cover every statement in the code at least once.

b. Test set for Branch Coverage:

The test set should cover every possible branch in the code.
Test Set:

- p = new Point[]{new Point(0,0), new Point(1,1), new Point(-1,-1)}
- doGraham(p)

Explanation:

This test set contains three points, one with coordinates (0,0), one with coordinates (1,1) and one with coordinates (-1,-1). The doGraham() method is called with these three points as input. This test set will cover every possible branch in the code.

c. Test set for Basic Condition Coverage:

The test set should cover every possible condition in the code, including both true and false evaluations.
Test Set:

- p = new Point[]{new Point(0,0), new Point(1,1), new Point(-1,-1)}
- doGraham(p)

Explanation:

This test set contains three points, one with coordinates (0,0), one with coordinates (1,1) and one with coordinates (-1,-1). The doGraham() method is called with these three points as input. This test set will cover every possible condition in the code, including both true and false evaluations.