Lab-7 Junit Testing Framework.

Name: Manan Modi
Student ID: 202001254
Group: 24

Section A:

Q. 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?

For designing the equivalence test case class we first need to check for the input to identify the valid and invalid equivalence classes.

Valid equivalence class:

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)

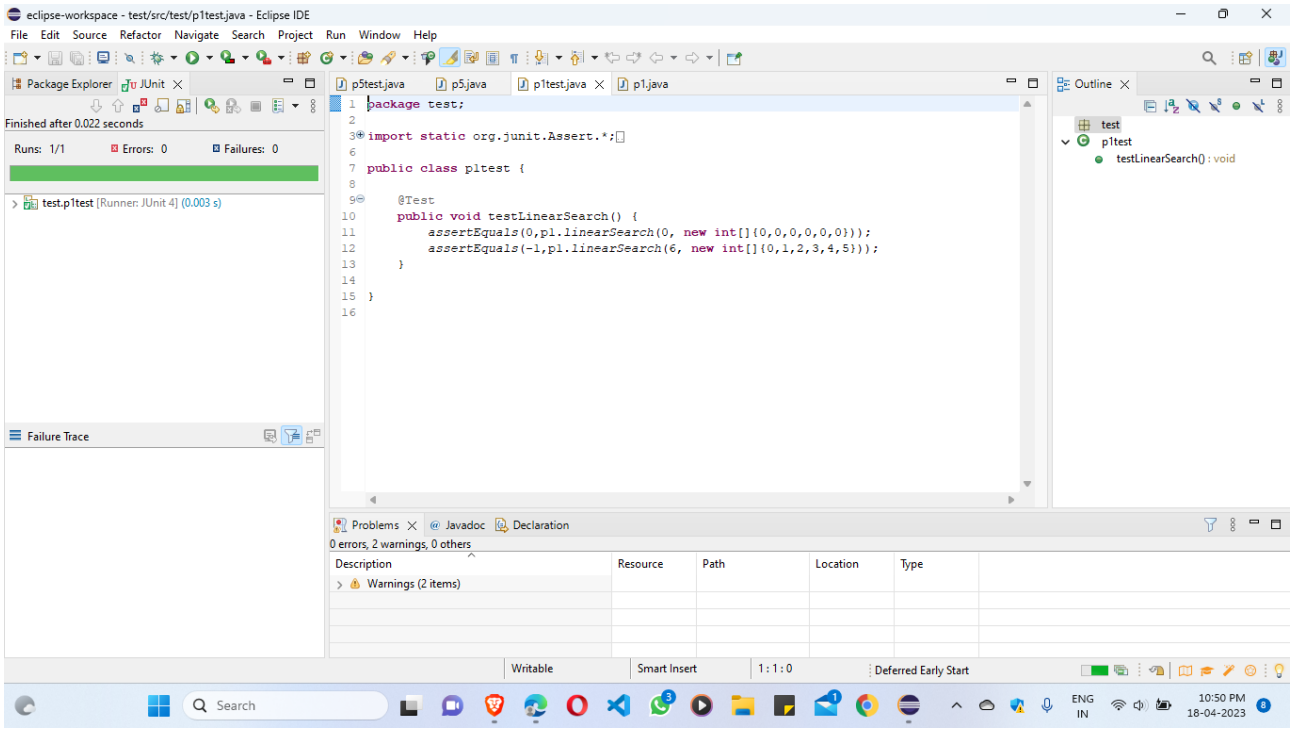
Invalid equivalence class:

Test 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)

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.

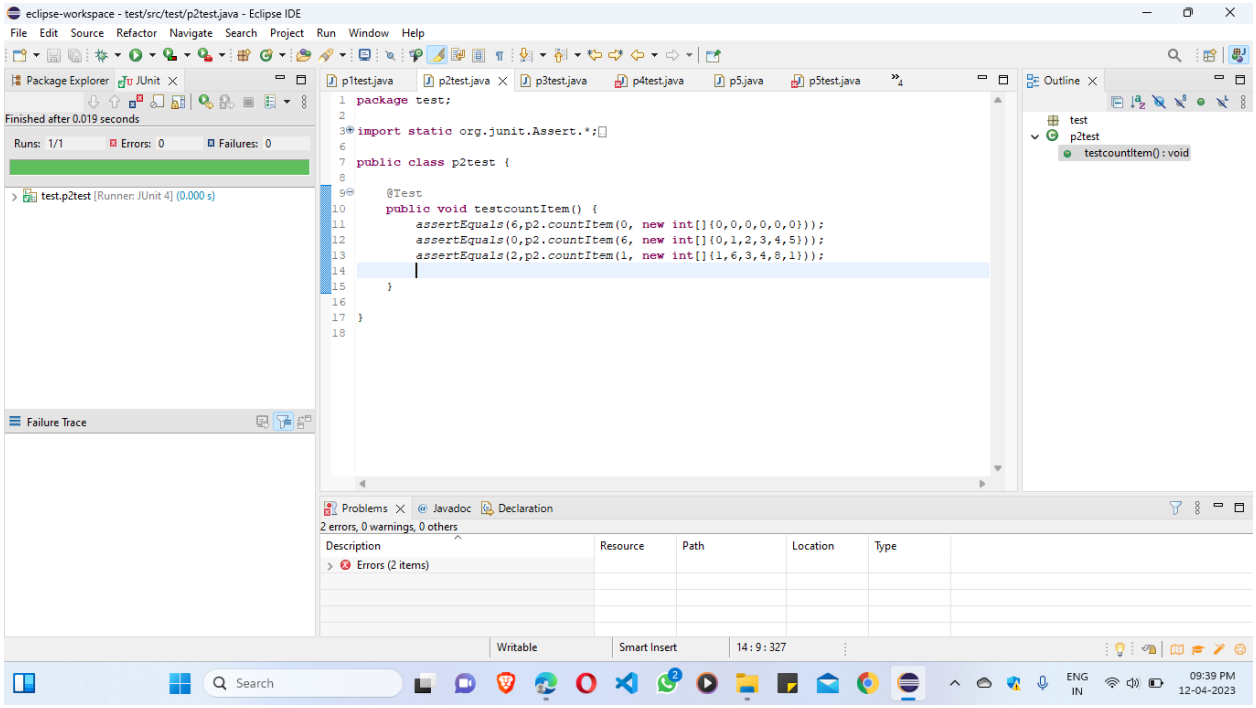
```
int linearSearch(int v, int a[])
{
 int i = 0;
 while (i < a.length)
 {
```

```
 if (a[i] == v)
{
return(i);
}
 I++;
 }
 return (-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);
}
```



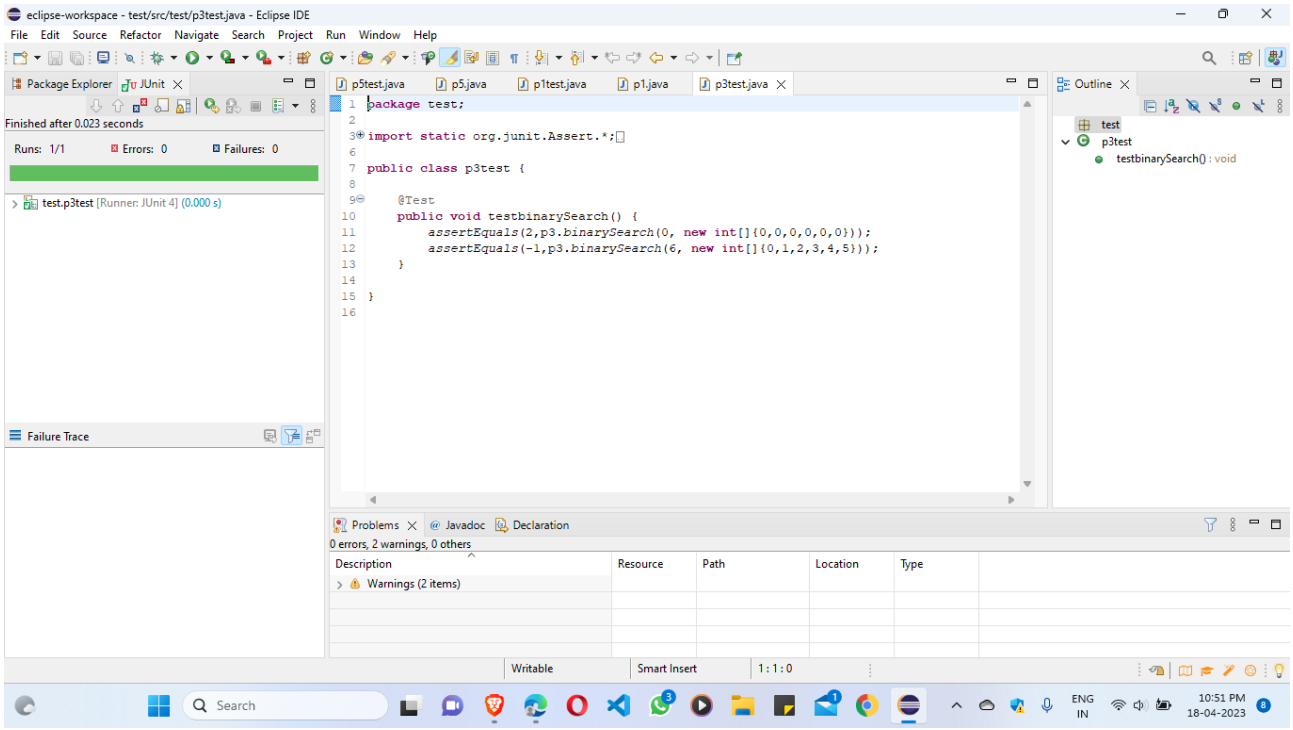
P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

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);

}



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);

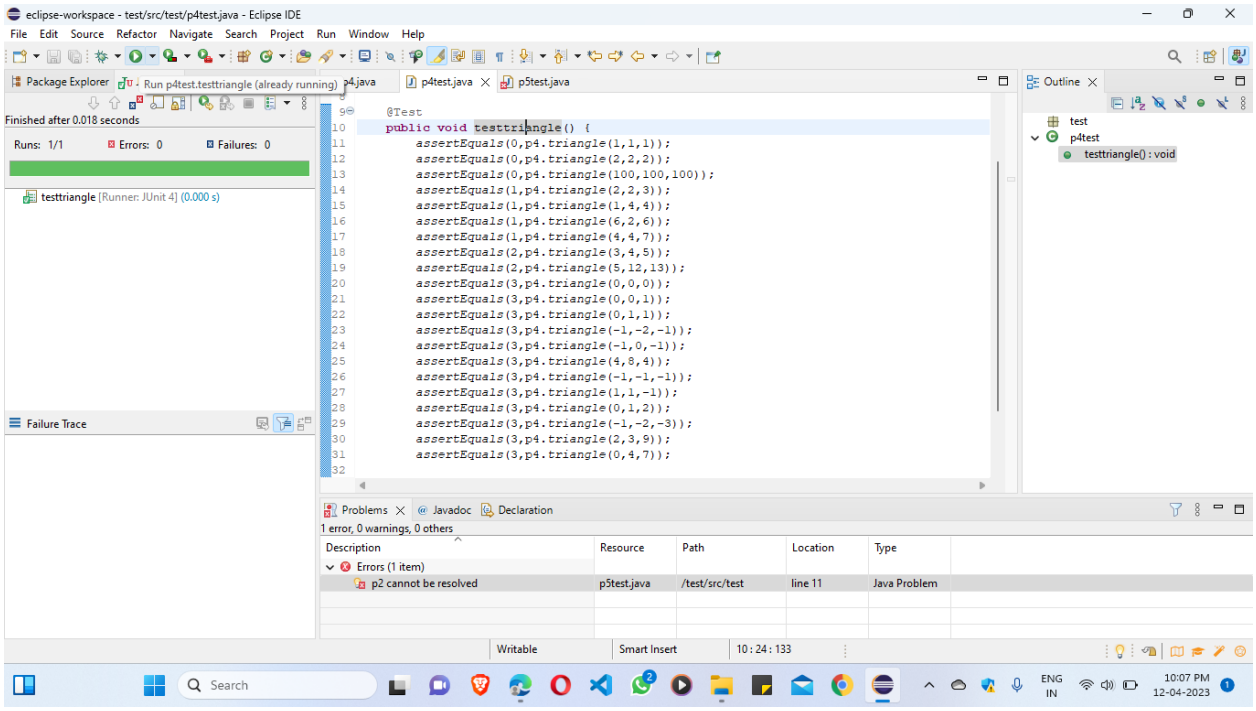
if (a == b && b == c)

return(EQUILATERAL);

if (a == b || a == c || b == c)

return(ISOSCELES);

return(SCALENE);



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))

{

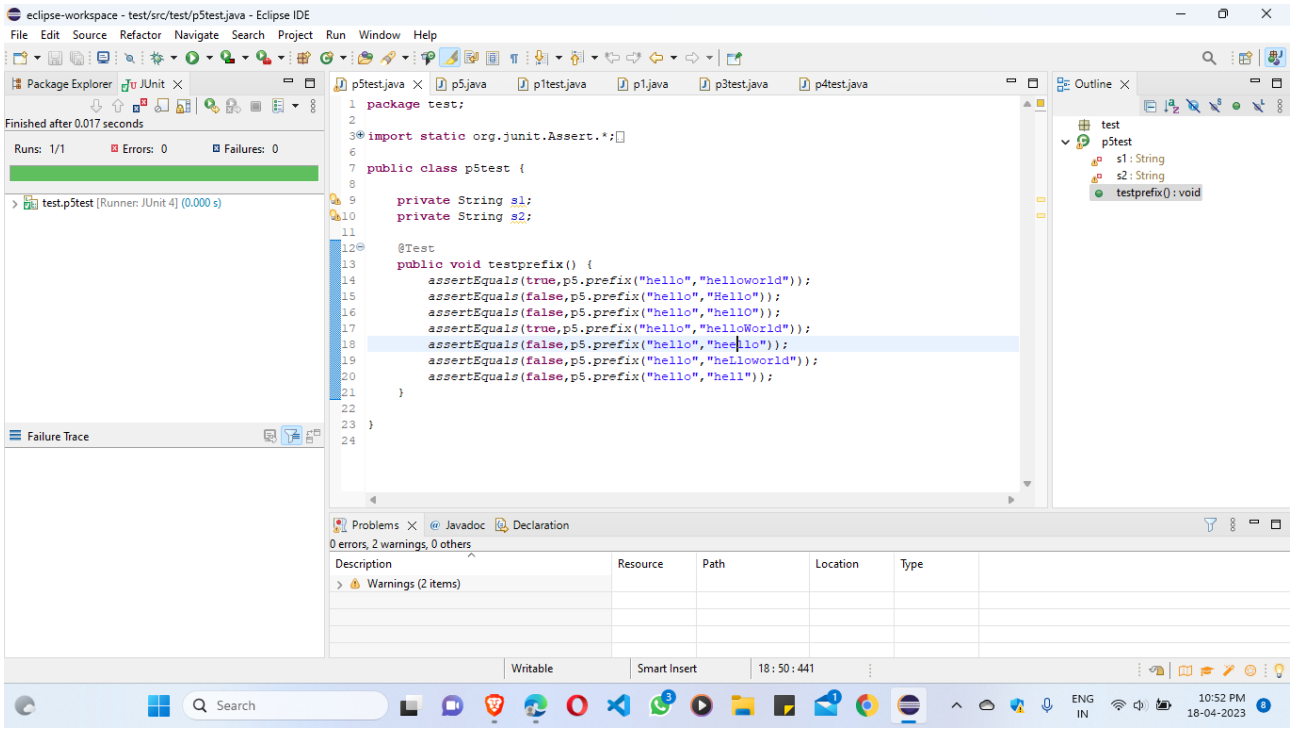
return false;

}

}

return true;

}



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

- Equilateral
- Isosceles
- Scalene
- One or more zero
- Sum of any 2 side more than third

- One ore more negative value

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

- 1-3
- 4-7
- 8-9
- 10-22

(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.

- 13-16, 18, 20-23

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

- 4-8

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

- 4-7

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

- 8-11

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

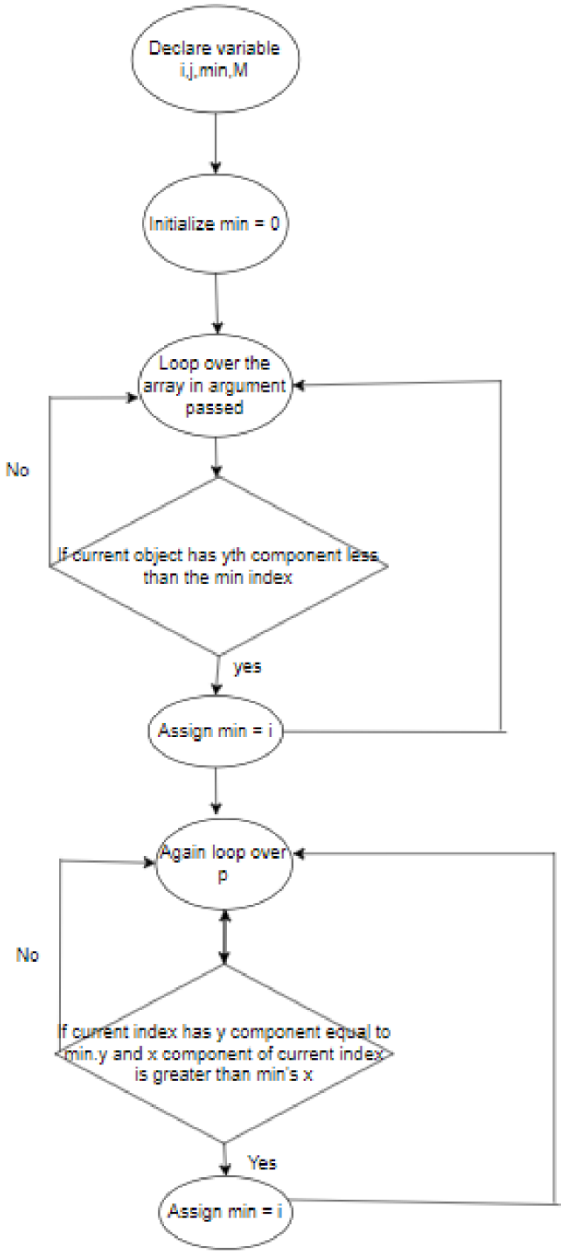
- 10-22

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

- 13,14,16,17,19,20

Section B:

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



2) Construct test sets for your flow graph that are adequate for the following criteria:

A. Statement Coverage.

Test Number	Test Case
1.	P is empty array
2.	P has one point object
3.	P has two point 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

B. Branch Coverage.

Test number	Test case
1.	P is empty array
2.	P has one point object
3.	P has two point 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

C. Basic Condition Coverage.

Test numbers	Test case
1.	P is empty array

2.	P has one point object
3.	P has two point 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
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