**Name: Baravaliya Divyesh Tarunbhai**
**ID: 202001229**
**IT314 - Software Engineering**
**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 a previous date or an invalid date. Design the equivalence class test cases.**

**Equivalence Partitioning:**

Equivalence partitioning is a technique used in software testing to divide the input data of a program into different partitions or subsets in order to ensure that each partition is tested at least once.

Based on the given input ranges, we can identify the following equivalence classes:

Valid input dates: These are the dates that fall within the given range of 1 <= month <= 12, 1 <= day <= 31, and 1900 <= year <= 2015.
Invalid input dates: These are the dates that fall outside the given range.

Using Equivalence Partitioning, we can identify the following test cases:

| | | |
|---|---|---|
| Valid input date | 2 - 1 - 1900 | 1 - 1 - 1900 |
| Valid input date | 28 - 2 - 1901 | 27 - 2 - 1901 |
| Valid input date | 5 - 4 - 2011 | 4 - 3 - 2011 |
| Valid input date | 7 - 8 - 2000 | 6 - 8 - 2000 |

| | | |
|---|---|---|
| Valid input date | 1 - 1 - 1901 | 31 - 12 - 1900 |
| Valid input date | 3 - 9 - 1988 | 2 - 9 - 1988 |
| Valid input date | 11 - 12 - 2014 | 10 - 12 - 2014 |
| Valid input date | 7 - 12 - 2015 | 6 - 12 - 2015 |
| Valid input date | 3 - 6 - 2010 | 2 - 6 - 2010 |
| Invalid input date | 0 - 1 - 1901 | Invalid |
| Invalid input date | 32 - 1 - 1901 | Invalid |
| Invalid input date | 1 - 0 - 1901 | Invalid |
| Invalid input date | 1 - 13 - 1901 | Invalid |
| Invalid input date | 1 - 1 - 1899 | Invalid |
| Invalid input date | 32 - 16 - 2016 | Invalid |

**Boundary Value Analysis:**

Boundary value analysis is a technique used in software testing to identify test cases at the boundaries of input domains. The idea is to test the program behavior at the extremes of the input ranges.

Based on the given input ranges, we can identify the following boundary values:

Minimum valid day: 1
Maximum valid day: 31
Minimum valid months: 1
Maximum valid month: 12
Minimum valid year: 1900
Maximum valid year: 2015

Using Boundary Value Analysis, we can identify the following test cases:

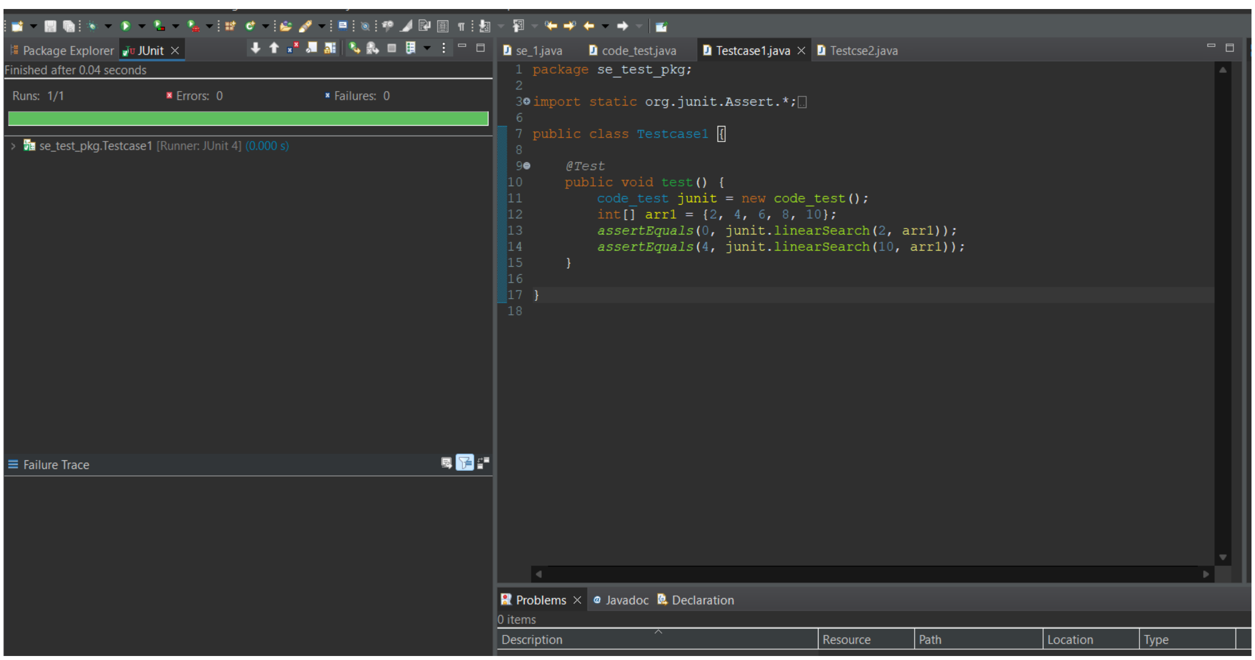
| | | |
|---|---|---|
| Minimum valid day | 1 - 1 - 1901 | YES |
| Minimum valid day | 31 - 12 - 2015 | YES |
| Minimum valid month | 1 - 1 - 1901 | YES |
| Minimum valid month | 12 - 12 - 2015 | YES |
| Minimum valid year | 1 - 1 - 1900 | Invalid |
| Minimum valid year | 1 - 1 - 2016 | Invalid |

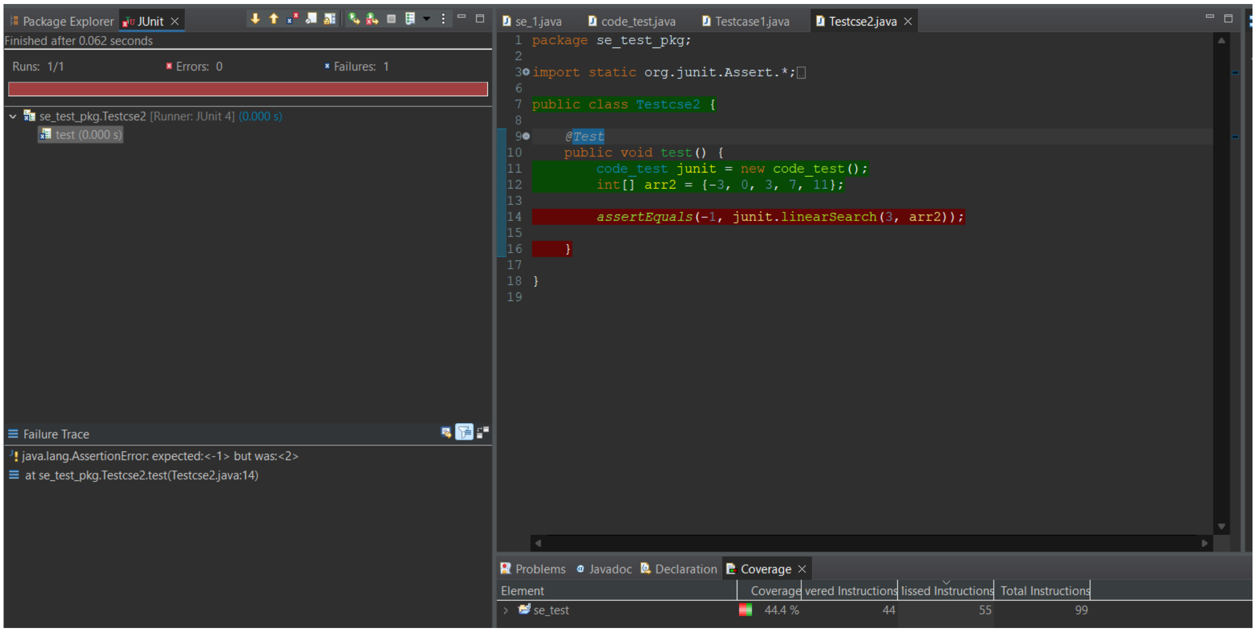
**Programs:**
**P1: The function linear search searches for a value v in an array of integers a. If v appears in the array**
**a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.**

```
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)
}
```

| 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 |

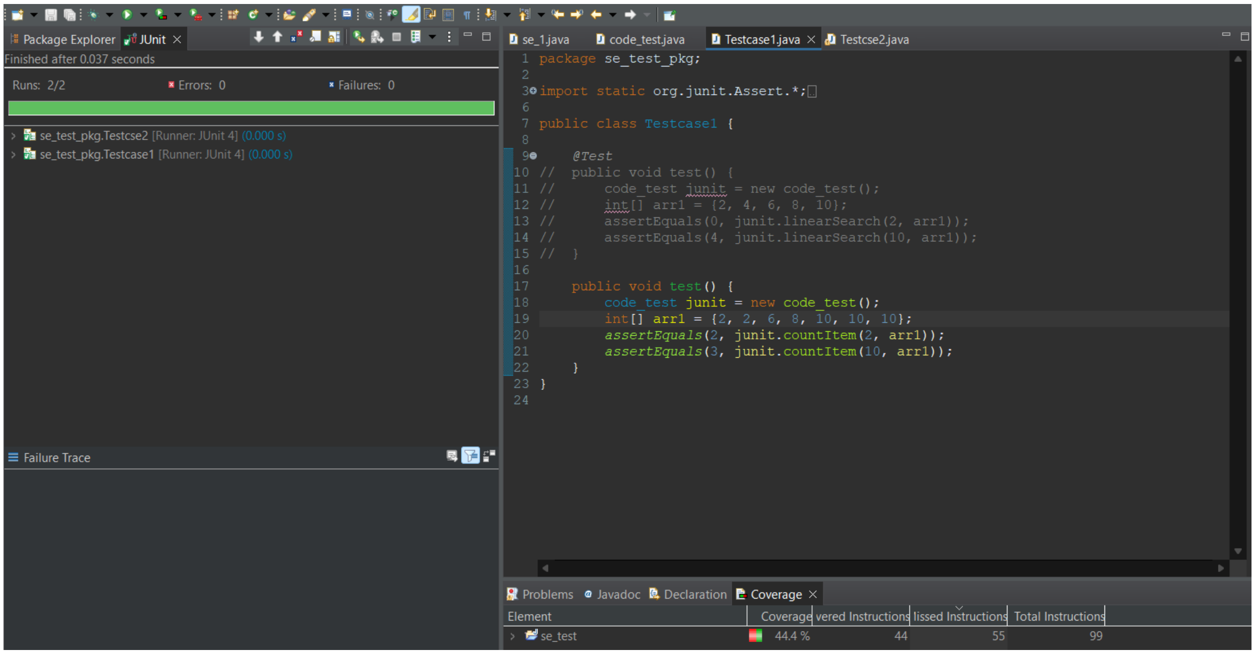
| Boundary Value Analysis | |
|---|---|
| Array with the minimum length possible. Input: a = [0], v = 0 | 0 |
| Array with the maximum length possible. Input: a = [1, 2, ..., 9998, 9999], v = 9999 | 9999 |
| Search value at the beginning of the array. Input: a = [10, 20, 30, 40, 50], v = 10 | 0 |
| Search 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 |

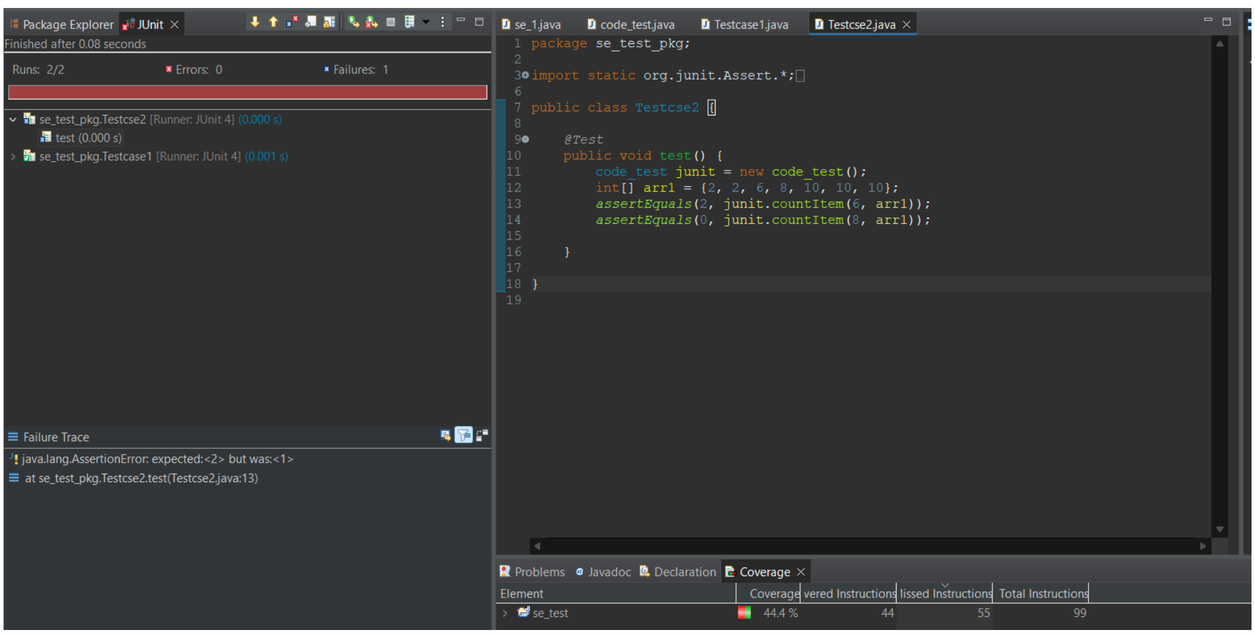





**P2:The function countItem returns the number of times a value v appears in an array of integers a.**

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





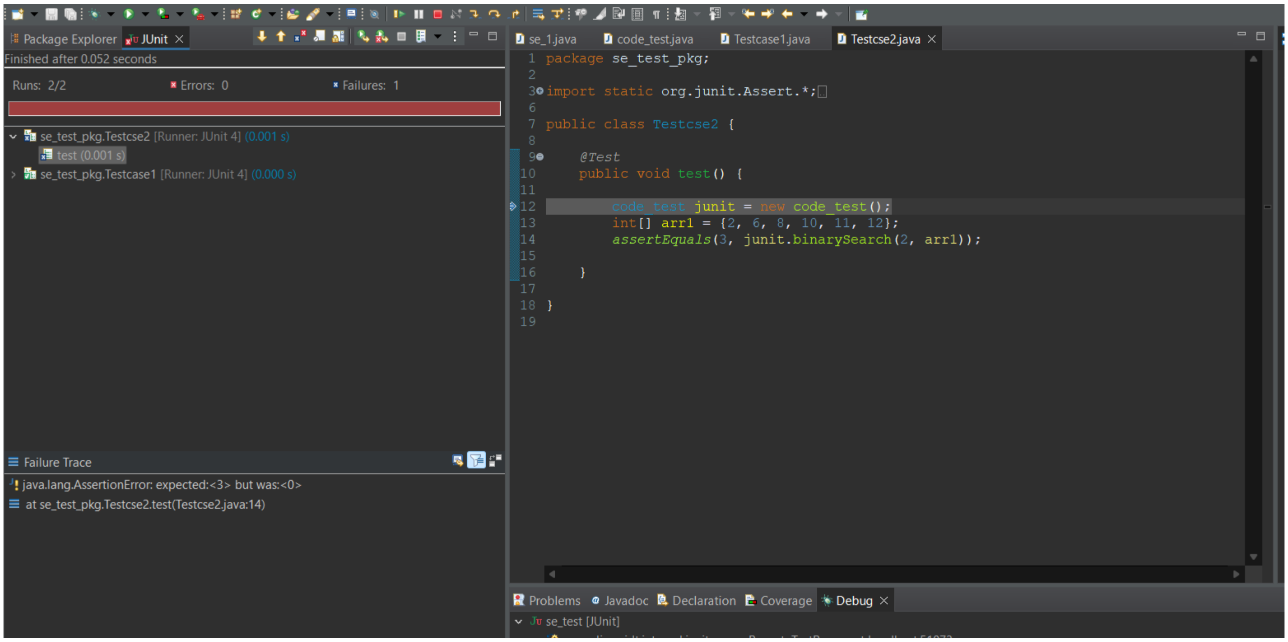
| Tester Action and Input Data | Expected Outcome |
| --- | --- |
| Equivalence Partitioning | |
| v = 5, a = {1, 5, 6, 5, 2} | 2 |
| v = 0, a = {0, 0, 0, 0, 0} | 5 |
| Invalid Input: v = 'a', a = {1, 2, 3, 4, 5} | An error message |
| Invalid Input: v = 3, a = null | An error message |
| Boundary Value 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 |

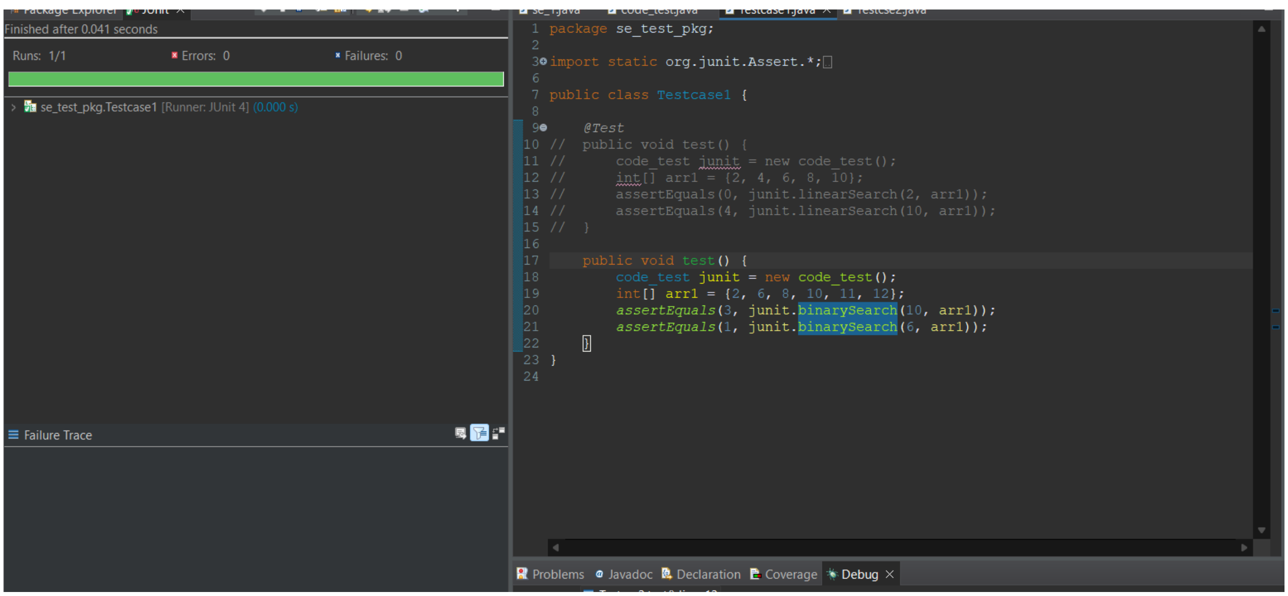
**P3: The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.**
**Assumption: the elements in the array a are sorted in non-decreasing order.**

```
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);
}
```


Package Explorer
JUnit
se_1.java
code_test.java
Testcase1.java
Testcse2.java
Finished after 0.052 seconds
Runs: 2/2
Errors: 0
Failures: 1
se_test_pkg.Testcse2 [Runner: JUnit 4] (0.001 s)
test (0.001 s)
se_test_pkg.Testcase1 [Runner: JUnit 4] (0.000 s)
Failure Trace
java.lang.AssertionError: expected:<3> but was:<0>
at se_test_pkg.Testcse2.test(Testcse2.java:14)
package se_test_pkg;
import static org.junit.Assert.*;
public class Testcse2 {
@Test
public void test() {
code_test junit = new code_test();
int[] arr1 = {2, 6, 8, 10, 11, 12};
assertEquals(3, junit.binarySearch(2, arr1));
}
}
Problems
Javadoc
Declaration
Coverage
Debug
se_test [JUnit]


Finished after 0.041 seconds
Runs: 1/1
Errors: 0
Failures: 0
se_test_pkg.Testcase1 [Runner: JUnit 4] (0.000 s)
Failure Trace
package se_test_pkg;
import static org.junit.Assert.*;
public class Testcase1 {
@Test
// public void test() {
// code_test junit = new code_test();
// int[] arr1 = {2, 4, 6, 8, 10};
// assertEquals(0, junit.linearSearch(2, arr1));
// assertEquals(4, junit.linearSearch(10, arr1));
// }
public void test() {
code_test junit = new code_test();
int[] arr1 = {2, 6, 8, 10, 11, 12};
assertEquals(3, junit.binarySearch(10, arr1));
assertEquals(1, junit.binarySearch(6, arr1));
}
}
Problems
Javadoc
Declaration
Coverage
Debug

| 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 |

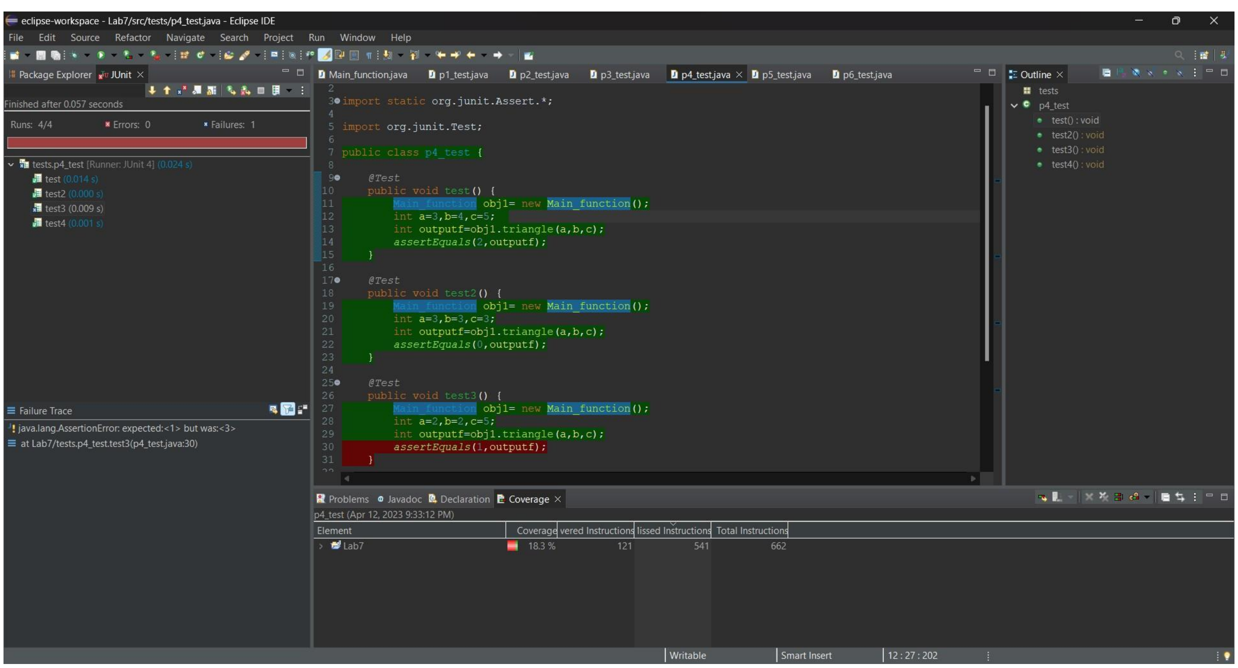
**P4: The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two 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);

}
```



| 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;**

**}**

**}**

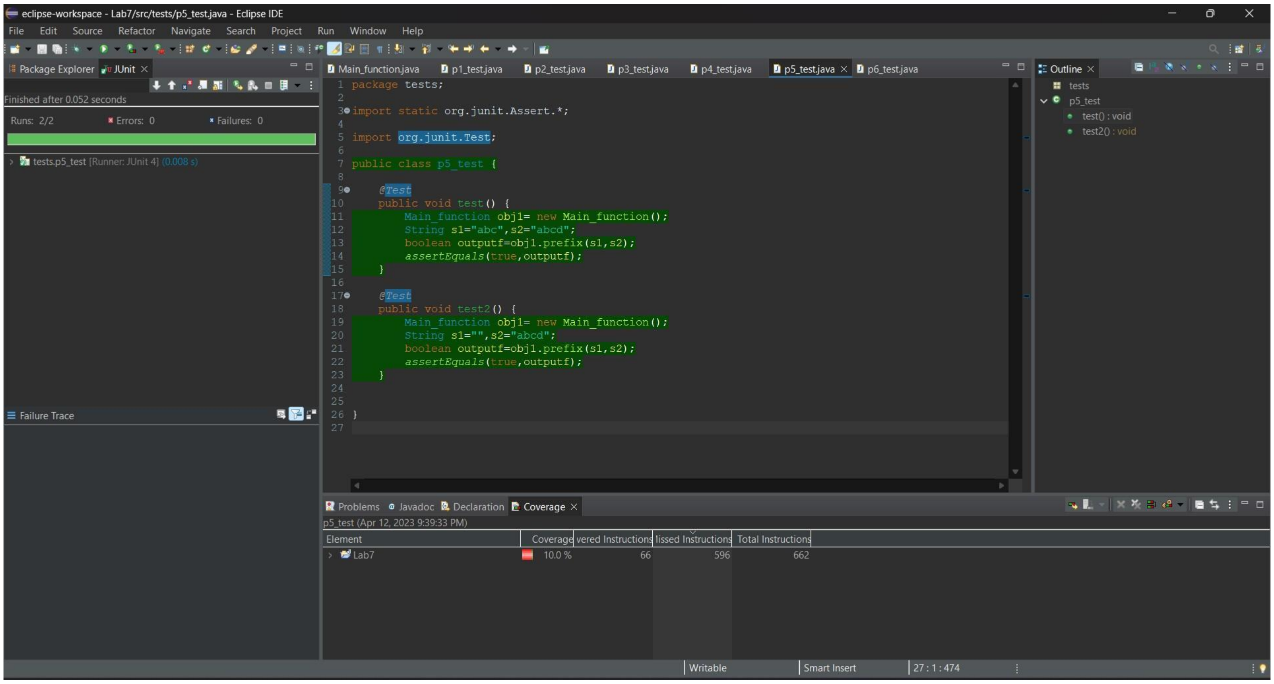
**return true;**

**}**

```
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 |


eclipse-workspace - Lab7/src/tests/p5_test.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit
Finished after 0.052 seconds
Runs: 2/2 Errors: 0 Failures: 0
tests.p5_test [Runner: JUnit 4] (0.008 s)
Failure Trace
Main_function.java p1_test.java p2_test.java p3_test.java p4_test.java p5_test.java p6_test.java
package tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class p5_test {
@Test
public void test() {
Main_function obj1= new Main_function();
String s1="abc",s2="abcd";
boolean outputf=obj1.prefix(s1,s2);
assertEquals(true,outputf);
}
@Test
public void test2() {
Main_function obj1= new Main_function();
String s1="",s2="abcd";
boolean outputf=obj1.prefix(s1,s2);
assertEquals(true,outputf);
}
}
Outline
tests
p5_test
test() : void
test2() : void
Problems Javadoc Declaration Coverage
p5_test (Apr 12, 2023 9:39:33 PM)
Element Coverage vered Instructions lissed Instructions Total Instructions
Lab7 10.0 % 66 596 662
Writable Smart Insert 27 : 1 : 474

**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 A**

**2 + 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:

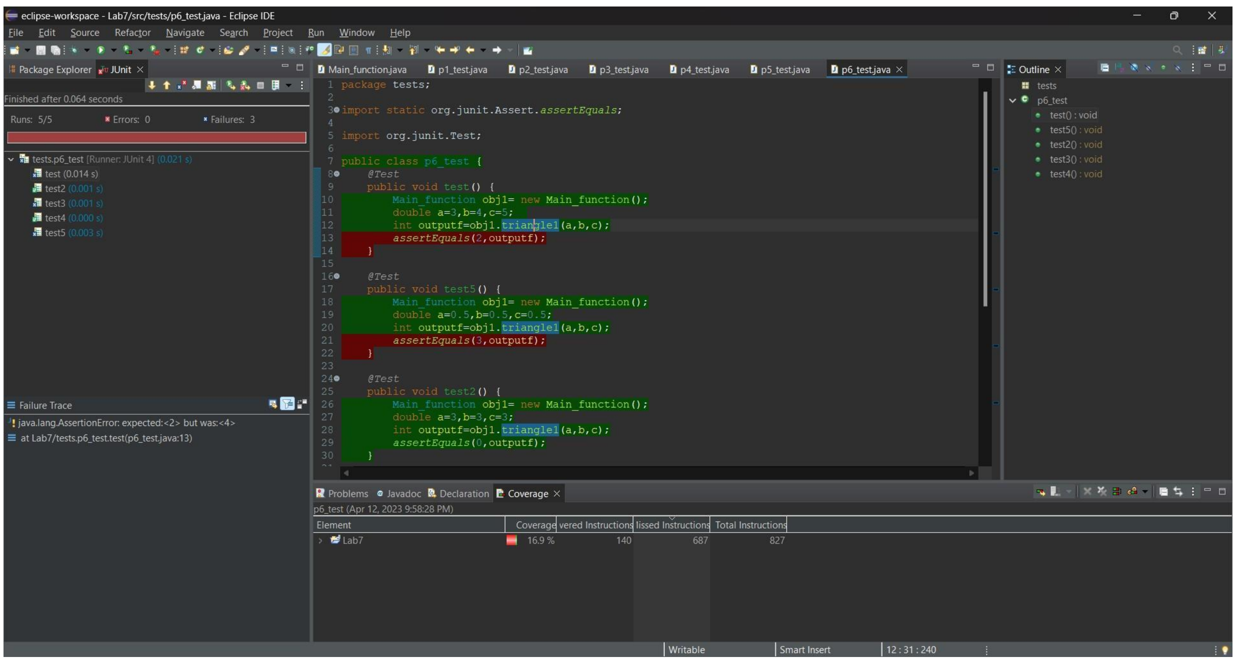
1. 1, 2, 3
2. 2, 3, 5
3. 4, 5, 9

### 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;
    }
```

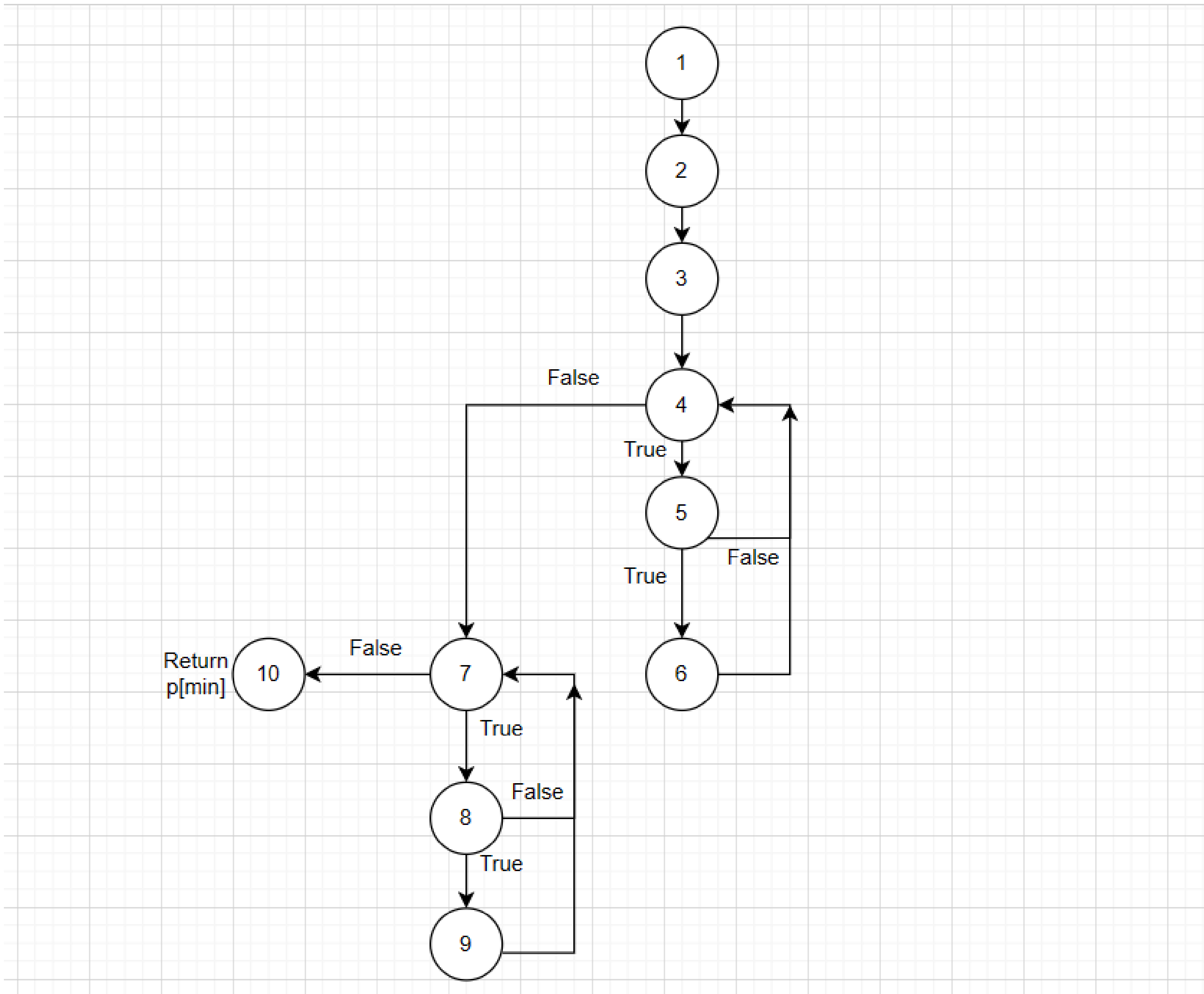


**Section B:**

**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.