4/18/2023

IT-314 Software Engineering

Lab 7

Zeel Bhanderi
202001412

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?

Inputs and their range:
1.) Day: [1,31]
2.) Month: [1, 12]
3.) Year: [1900, 2015]

Equivalence Class Partitioning:

ID	**Input Date**	**Expected Output**
E1	(15,10, 2022)	(14,10, 2022)
E2	(1, 6, 2018)	(31,5,2018)
E3	(31, 3, 2000)	30, 3, 2000
E4	(29, 2, 2022)	Invalid Date
E5	(31, 4, 2011)	Invalid Date
E6	(30, 2, 2000)	Invalid Date
E7	(0, 5, 2010)	Invalid Date
E8	(15, 13, 2005)	Invalid Date
E9	(31, 12, 1899)	Invalid Date

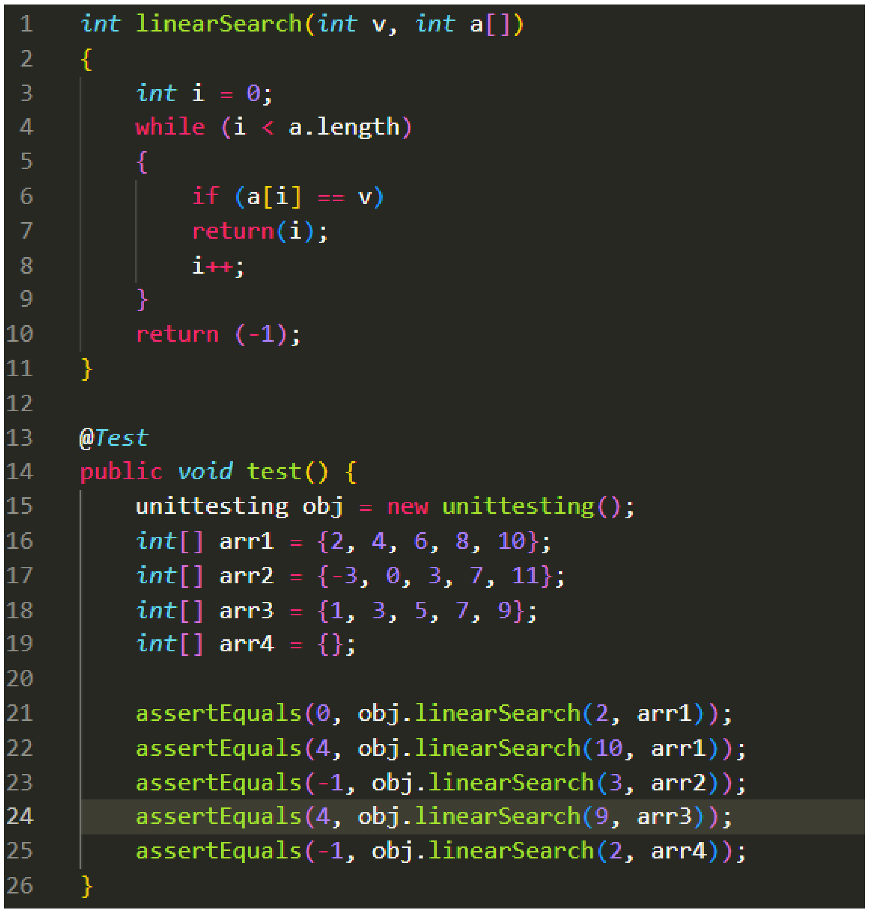
Boundary Value Analysis:

Input Date	Reason	Expected Output
(1, 1, 1900)	The earliest possible date	Invalid date
(31, 12, 2015)	The latest possible date	30, 12, 2015
(1, 2, 2000)	The earliest day of each month	31, 12, 1999
(31, 3, 2000)	The latest day of each month	30, 1, 2000
(29, 2, 2000)	Leap year day	28, 2, 2000
(29, 2, 1900)	Invalid leap year day	Invalid date
(31, 12, 1899)	One day before the earliest date	Invalid date
(1, 1, 2016)	One day after the latest date	Invalid date

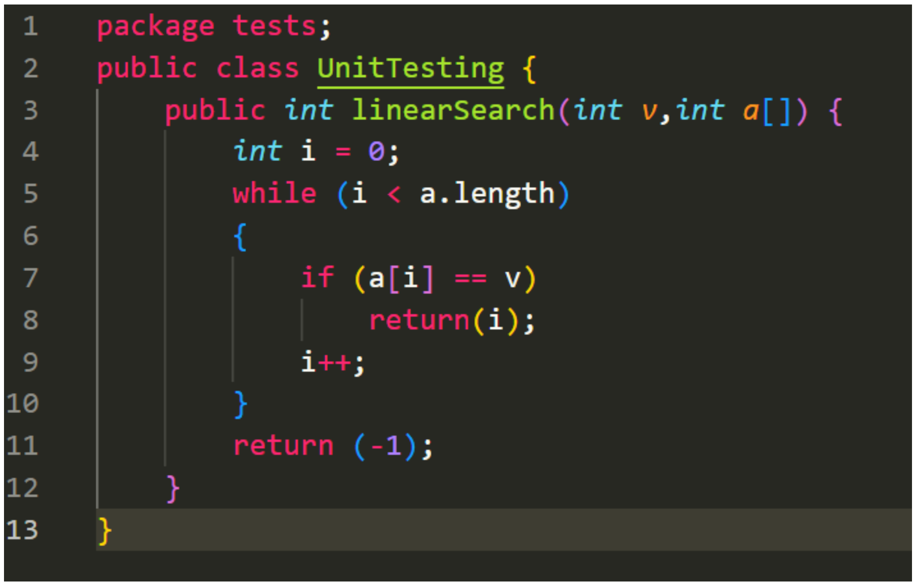
P1.linearSearch

Java Code :

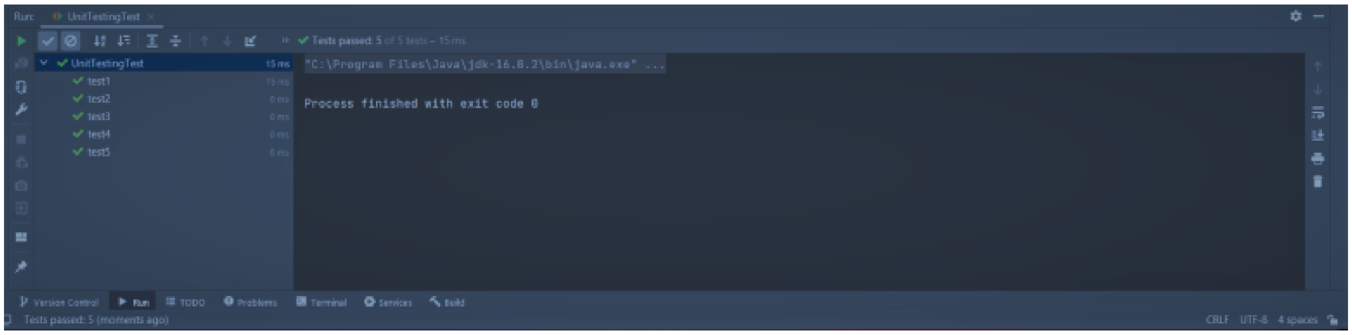
```
int linearSearch(int v, int a[])
{
        int i = 0;
        while (i < a.length)
        {
                if (a[i] == v)
                        return(i);
                i++;
        }
        return (-1);
}
```



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

@Test
public void test() {
    unittesting obj = new unittesting();
    int[] arr1 = {2, 4, 6, 8, 10};
    int[] arr2 = {-3, 0, 3, 7, 11};
    int[] arr3 = {1, 3, 5, 7, 9};
    int[] arr4 = {};

    assertEquals(0, obj.linearSearch(2, arr1));
    assertEquals(4, obj.linearSearch(10, arr1));
    assertEquals(-1, obj.linearSearch(3, arr2));
    assertEquals(4, obj.linearSearch(9, arr3));
    assertEquals(-1, obj.linearSearch(2, arr4));
}
```



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



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 5 (moments ago)

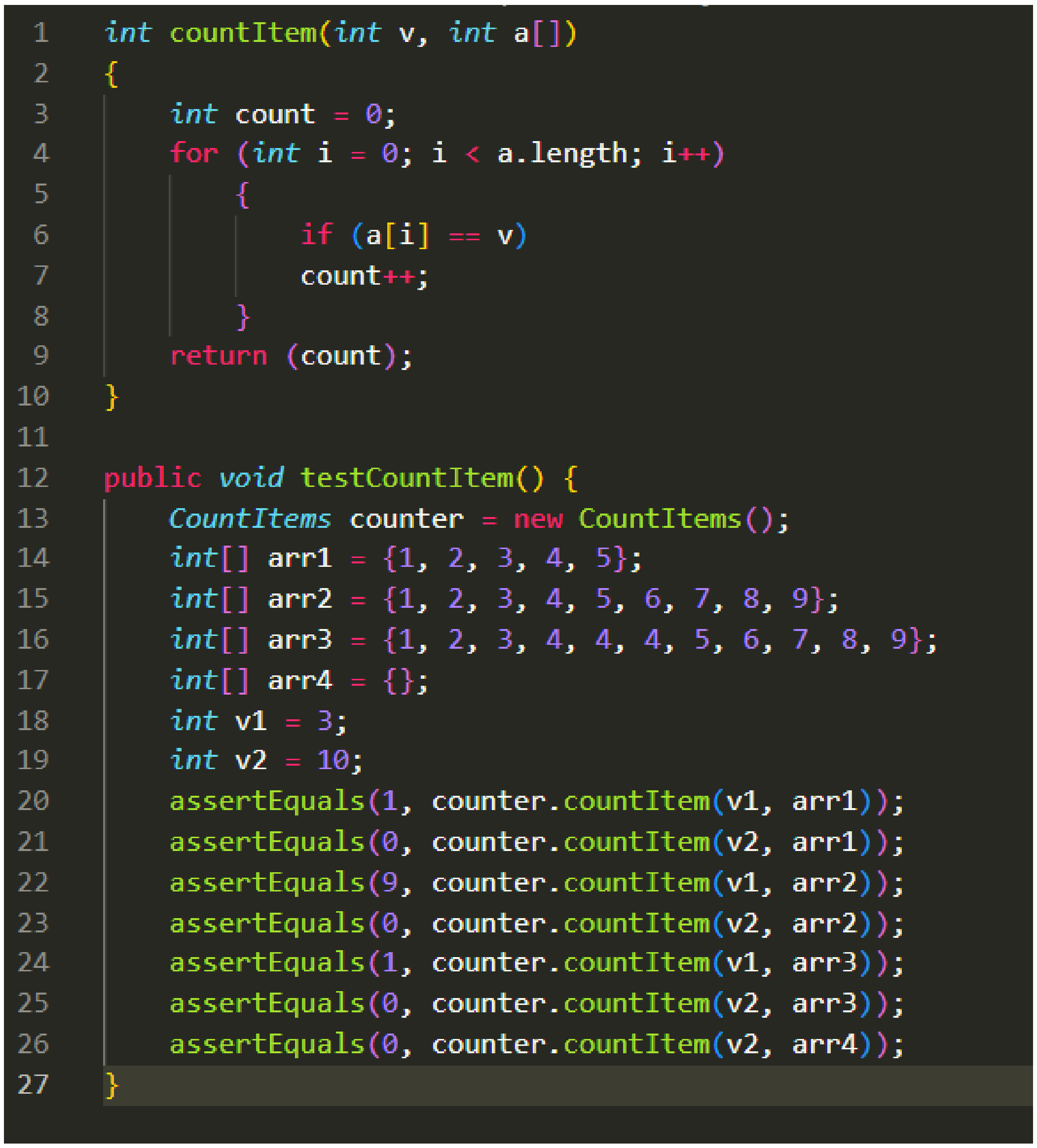
Equivalence Classes:

Input	**Expected Output**	**a**	**v**	**Actual Output**
v is present in a	v is not present in a	[1, 2, 3, 4, 5]	4	3
v is not present in a	-1	[1, 2, 3, 4, 5]	10	-1

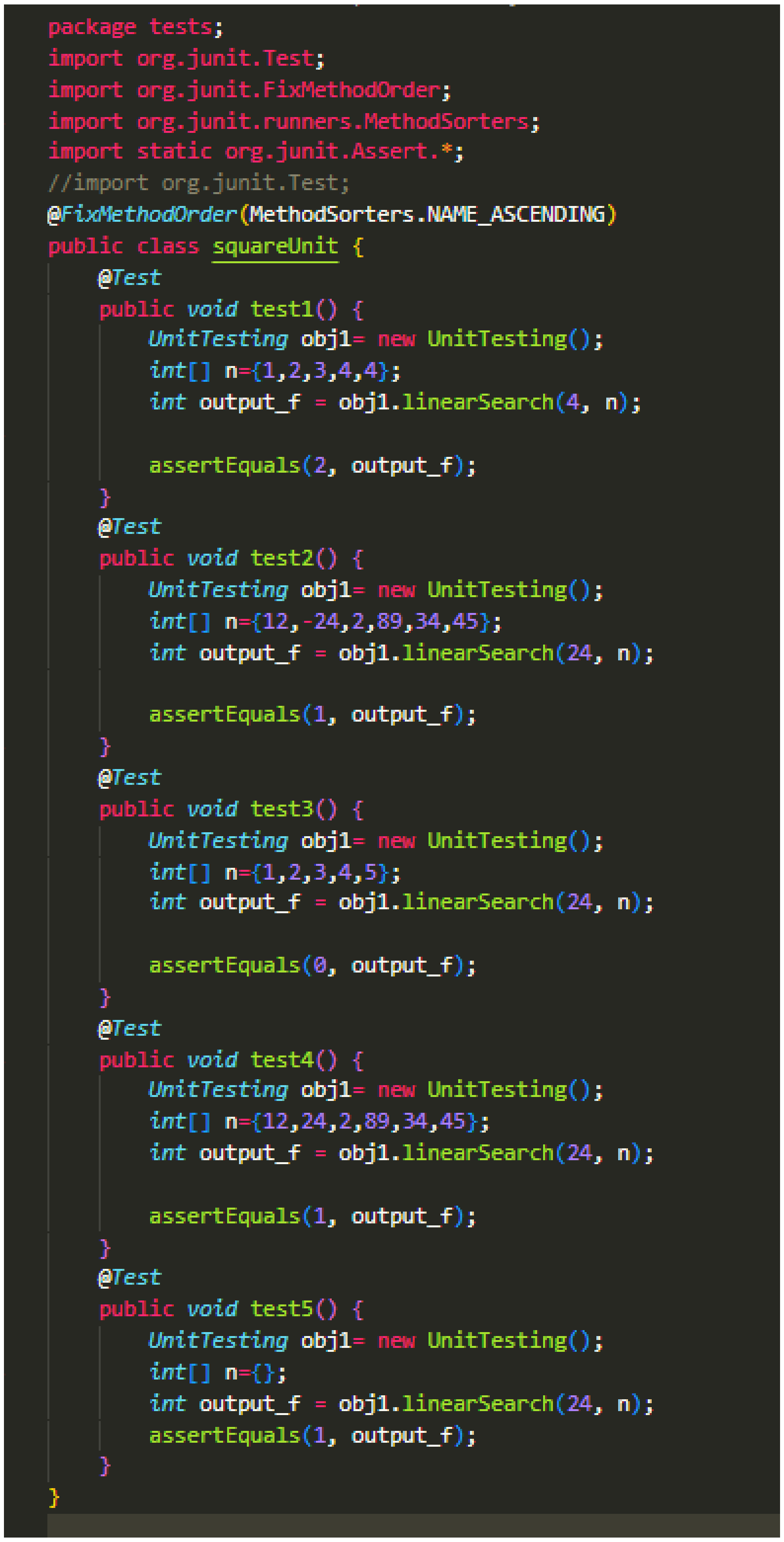
Boundary Value Analysis:

Input	**Expected Output**	**a**	**v**	**Actual Output**
Empty array a	-1	[]	4	-1
v is present at the first index of a	0	[1, 2, 3, 4, 5]	1	-1
v is present at the last index of the length of a	last index	[1, 2, 3, 4, 5]	5	4
v is not present in a	-1	[1, 2, 3, 4, 5]	10	-1

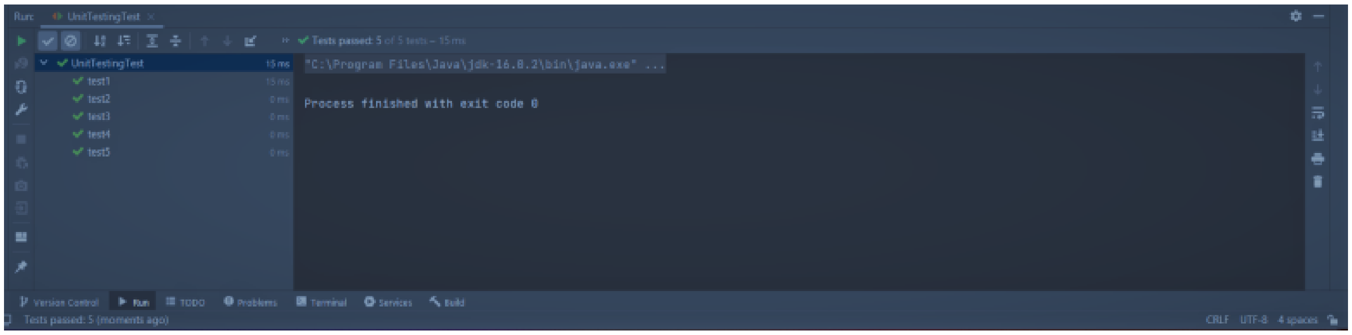
P2.countItem



```
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 void testCountItem() {
    CountItems counter = new CountItems();
    int[] arr1 = {1, 2, 3, 4, 5};
    int[] arr2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int[] arr3 = {1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 9};
    int[] arr4 = {};
    int v1 = 3;
    int v2 = 10;
    assertEquals(1, counter.countItem(v1, arr1));
    assertEquals(0, counter.countItem(v2, arr1));
    assertEquals(9, counter.countItem(v1, arr2));
    assertEquals(0, counter.countItem(v2, arr2));
    assertEquals(1, counter.countItem(v1, arr3));
    assertEquals(0, counter.countItem(v2, arr3));
    assertEquals(0, counter.countItem(v2, arr4));
}
```



```java
package tests;
import org.junit.Test;
import org.junit.FixMethodOrder;
import org.junit.runners.MethodSorters;
import static org.junit.Assert.*;
//import org.junit.Test;
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class squareUnit {
    @Test
    public void test1() {
        UnitTesting obj1= new UnitTesting();
        int[] n={1,2,3,4,4};
        int output_f = obj1.linearSearch(4, n);

        assertEquals(2, output_f);
    }
    @Test
    public void test2() {
        UnitTesting obj1= new UnitTesting();
        int[] n={12,-24,2,89,34,45};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(1, output_f);
    }
    @Test
    public void test3() {
        UnitTesting obj1= new UnitTesting();
        int[] n={1,2,3,4,5};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(0, output_f);
    }
    @Test
    public void test4() {
        UnitTesting obj1= new UnitTesting();
        int[] n={12,24,2,89,34,45};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(1, output_f);
    }
    @Test
    public void test5() {
        UnitTesting obj1= new UnitTesting();
        int[] n={};
        int output_f = obj1.linearSearch(24, n);
        assertEquals(1, output_f);
    }
}
```



Tests passed: 5 of 5 tests – 15 ms

```
UnitTestingTest  15 ms
  test1  15 ms
  test2  0 ms
  test3  0 ms
  test4  0 ms
  test5  0 ms
```

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 5 (moments ago)

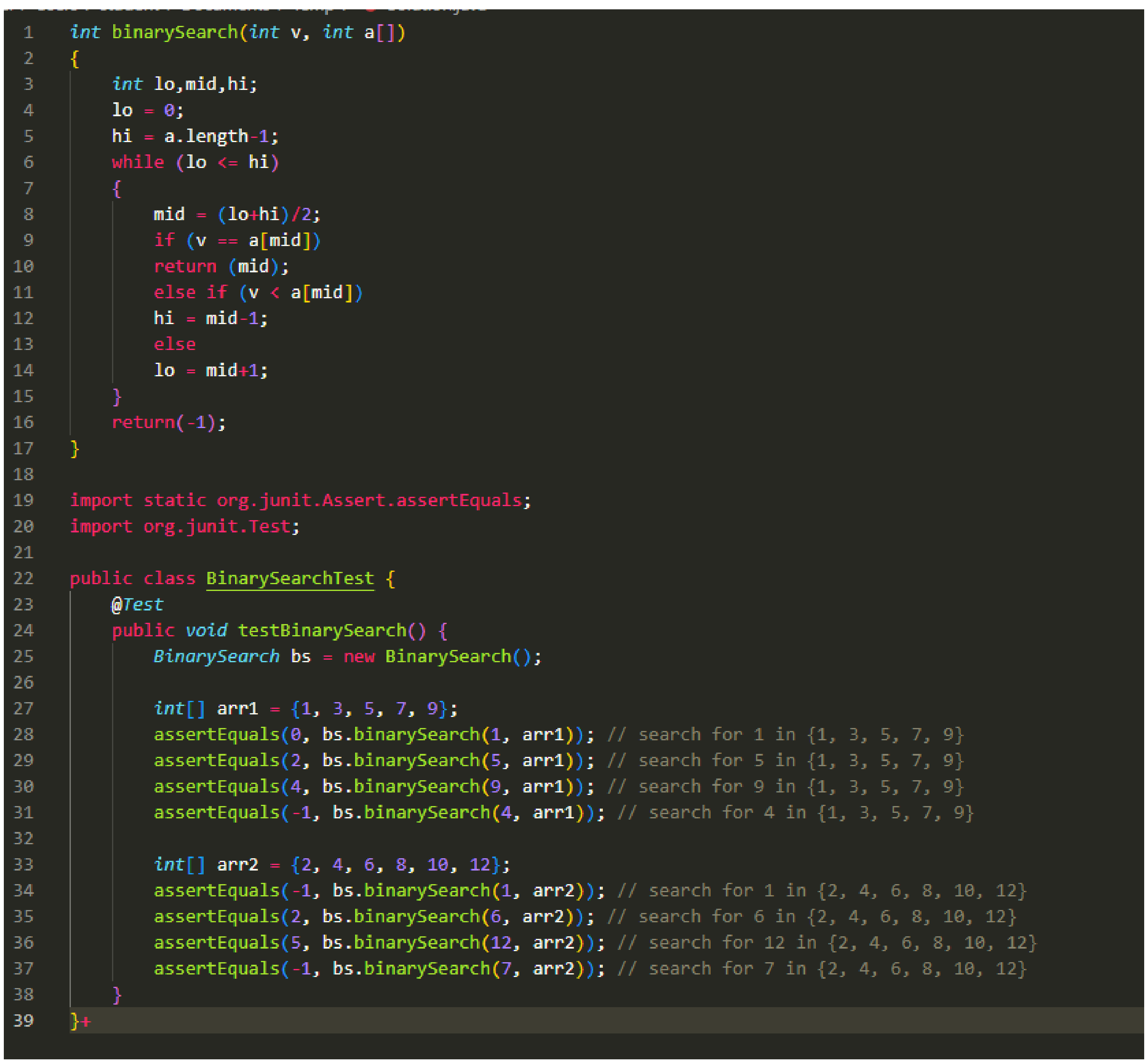
Equivalence Classes :

Input	**Expected Output**	**a**	**v**	**Actual Output**
present in a	Number of times v appears in a	[1, 2, 3, 4, 2]	2	2
v is not present in a	0	[1, 2, 3, 4, 5]	10	0

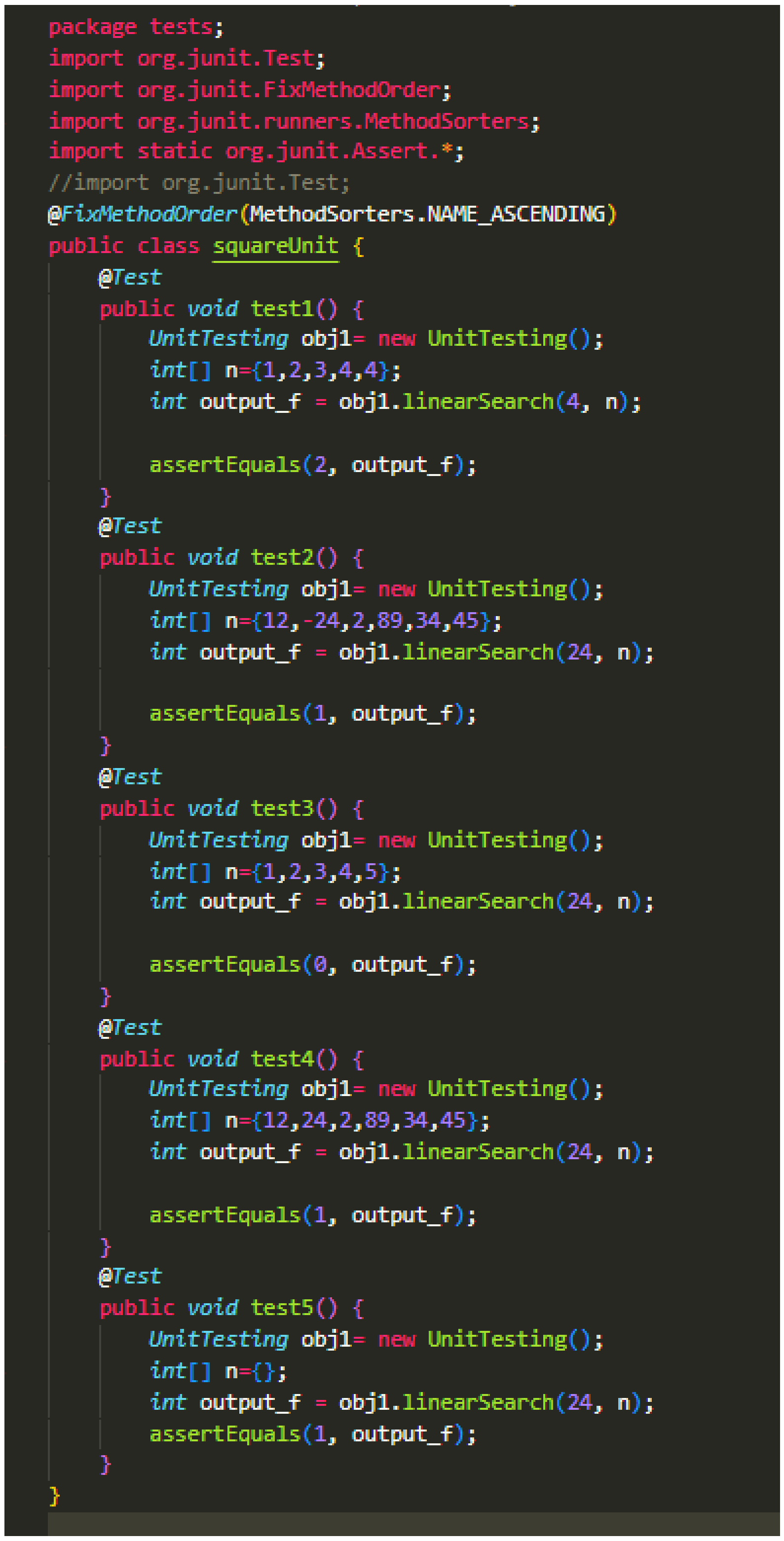
Boundary Value Analysis:

Input	Expected Output	a	v	Actual Output
Empty array a	0	[]	4	0
v is present once in a	1	[1, 2, 3, 4, 5]	1	1
v is present multiple times in a	number of times v appears in a	[1, 2, 1, 4,1]	1	3
v is present at the first index of a	1	[1, 2, 3, 4, 5]	1	1
v is present at the last index of a	1	[1, 2, 3, 4 5]	5	1
v is not present in a	0	[1, 2, 3, 4 5]	10	0

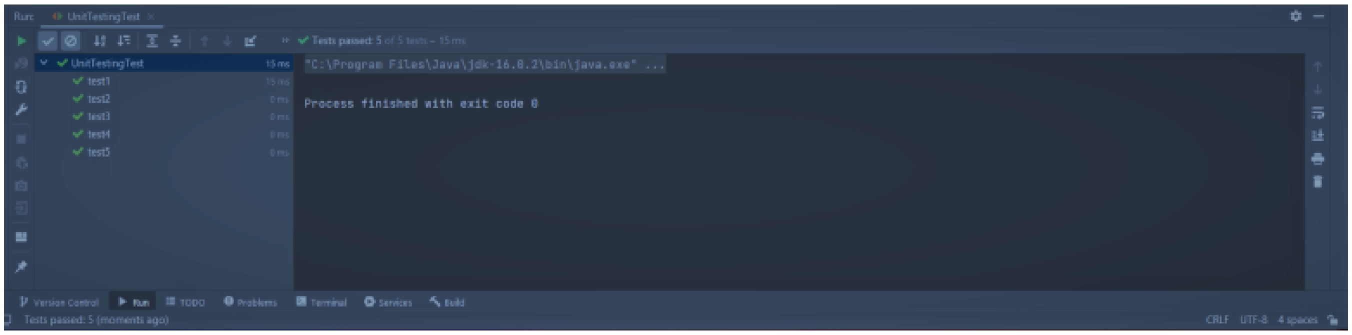
P3. binarySearch



```
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);
}

import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class BinarySearchTest {
    @Test
    public void testBinarySearch() {
        BinarySearch bs = new BinarySearch();

        int[] arr1 = {1, 3, 5, 7, 9};
        assertEquals(0, bs.binarySearch(1, arr1)); // search for 1 in {1, 3, 5, 7, 9}
        assertEquals(2, bs.binarySearch(5, arr1)); // search for 5 in {1, 3, 5, 7, 9}
        assertEquals(4, bs.binarySearch(9, arr1)); // search for 9 in {1, 3, 5, 7, 9}
        assertEquals(-1, bs.binarySearch(4, arr1)); // search for 4 in {1, 3, 5, 7, 9}

        int[] arr2 = {2, 4, 6, 8, 10, 12};
        assertEquals(-1, bs.binarySearch(1, arr2)); // search for 1 in {2, 4, 6, 8, 10, 12}
        assertEquals(2, bs.binarySearch(6, arr2)); // search for 6 in {2, 4, 6, 8, 10, 12}
        assertEquals(5, bs.binarySearch(12, arr2)); // search for 12 in {2, 4, 6, 8, 10, 12}
        assertEquals(-1, bs.binarySearch(7, arr2)); // search for 7 in {2, 4, 6, 8, 10, 12}
    }
}
```



```
package tests;
import org.junit.Test;
import org.junit.FixMethodOrder;
import org.junit.runners.MethodSorters;
import static org.junit.Assert.*;
//import org.junit.Test;
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class squareUnit {
    @Test
    public void test1() {
        UnitTesting obj1= new UnitTesting();
        int[] n={1,2,3,4,4};
        int output_f = obj1.linearSearch(4, n);

        assertEquals(2, output_f);
    }
    @Test
    public void test2() {
        UnitTesting obj1= new UnitTesting();
        int[] n={12,-24,2,89,34,45};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(1, output_f);
    }
    @Test
    public void test3() {
        UnitTesting obj1= new UnitTesting();
        int[] n={1,2,3,4,5};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(0, output_f);
    }
    @Test
    public void test4() {
        UnitTesting obj1= new UnitTesting();
        int[] n={12,24,2,89,34,45};
        int output_f = obj1.linearSearch(24, n);

        assertEquals(1, output_f);
    }
    @Test
    public void test5() {
        UnitTesting obj1= new UnitTesting();
        int[] n={};
        int output_f = obj1.linearSearch(24, n);
        assertEquals(1, output_f);
    }
}
```



```
Tests passed: 5 of 5 tests – 15 ms
UnitTestingTest 15 ms
  test1 15 ms
  test2 0 ms
  test3 0 ms
  test4 0 ms
  test5 0 ms
```

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...

Process finished with exit code 0
```

```
Tests passed: 5 (moments ago)
```

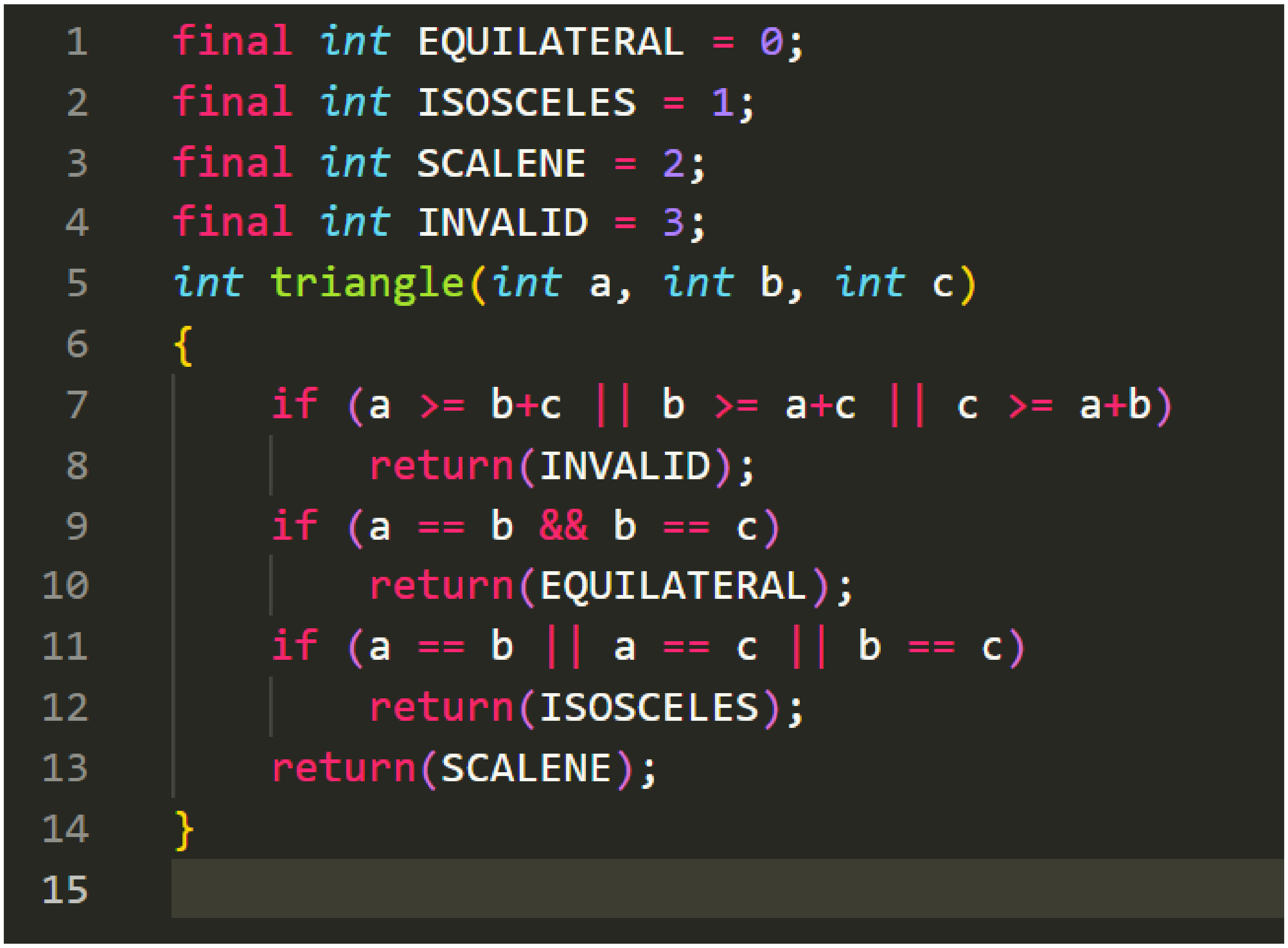
Equivalence Classes:

Input	Expected Output	a	v	Actual Output
v is present in a	Index of v	[1, 2, 3, 4, 2]	2	1
v is not present in a	-1	[1, 2, 3, 4, 5]	10	-1

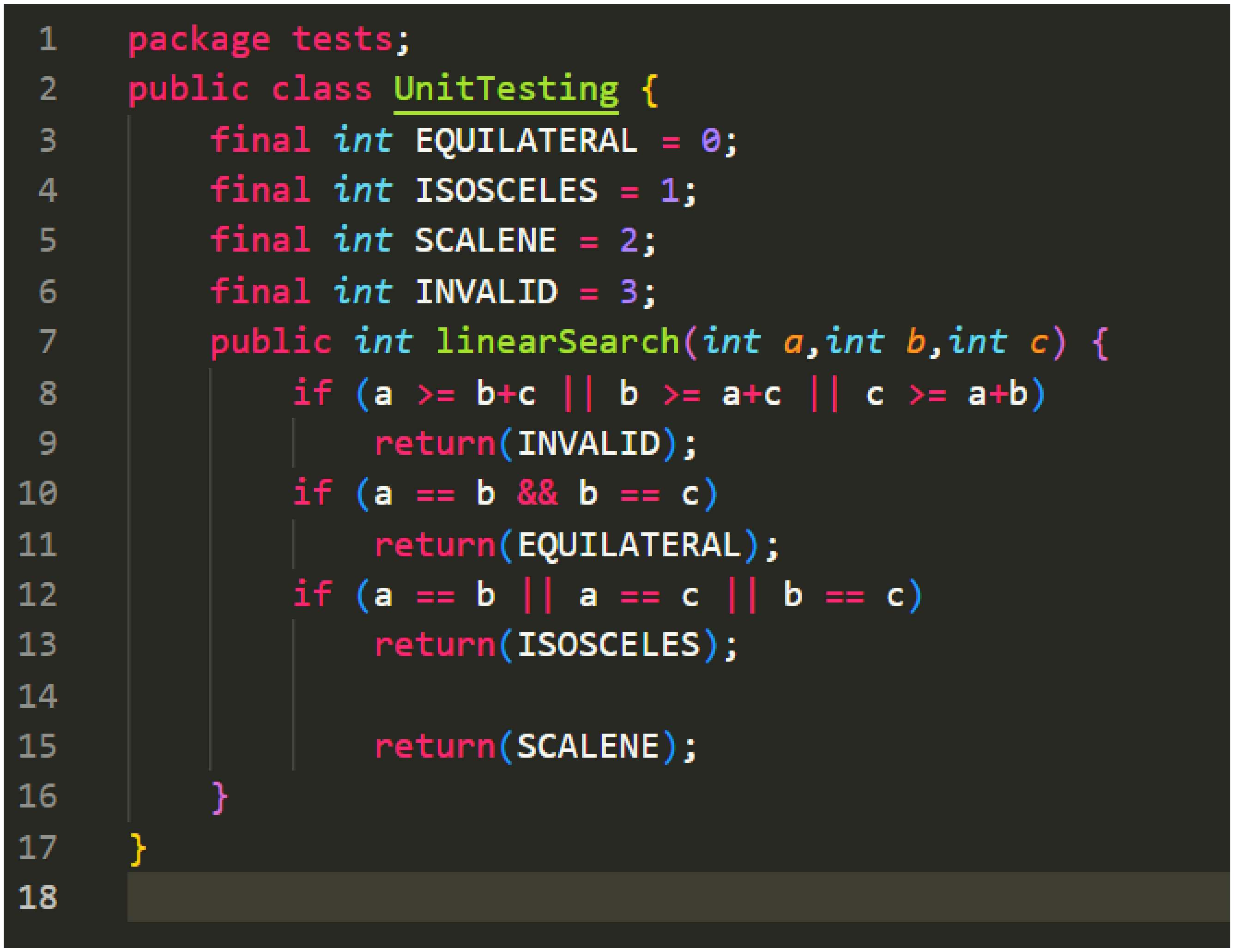
Boundary Value Analysis:

Input	**Expected Output**	**a**	**v**	**Actual Output**
Empty array a	-1	[]	4	-1
v is present at the first index of a	0	[1, 2, 3, 4, 5]	1	0
v is present at the last dex of the length of a	last index	[1, 2, 3, 4, 5]	5	4
v is not present in a	-1	[1, 2, 3, 4, 5]	10	-1

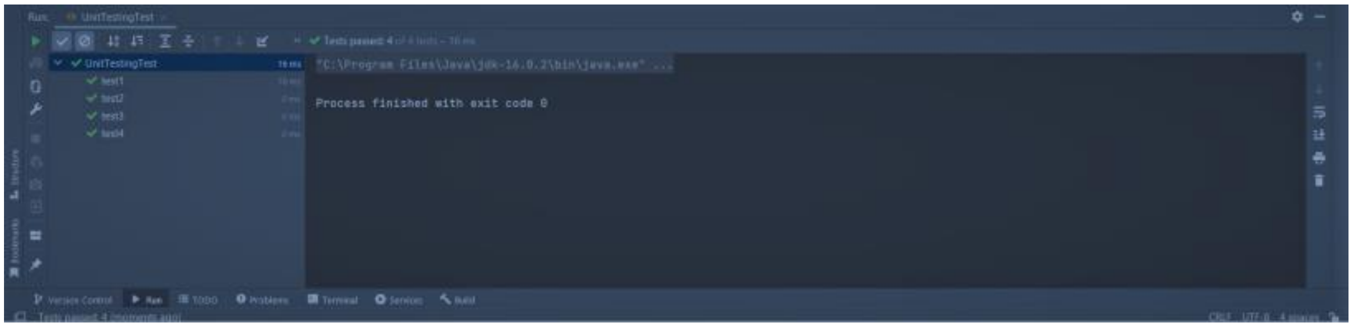
P4. triangle



```
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);
}
```



```java
package tests;
public class UnitTesting {
    final int EQUILATERAL = 0;
    final int ISOSCELES = 1;
    final int SCALENE = 2;
    final int INVALID = 3;
    public int linearSearch(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);
    }
}

```



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...

Process finished with exit code 0
```

Equivalence Classes:

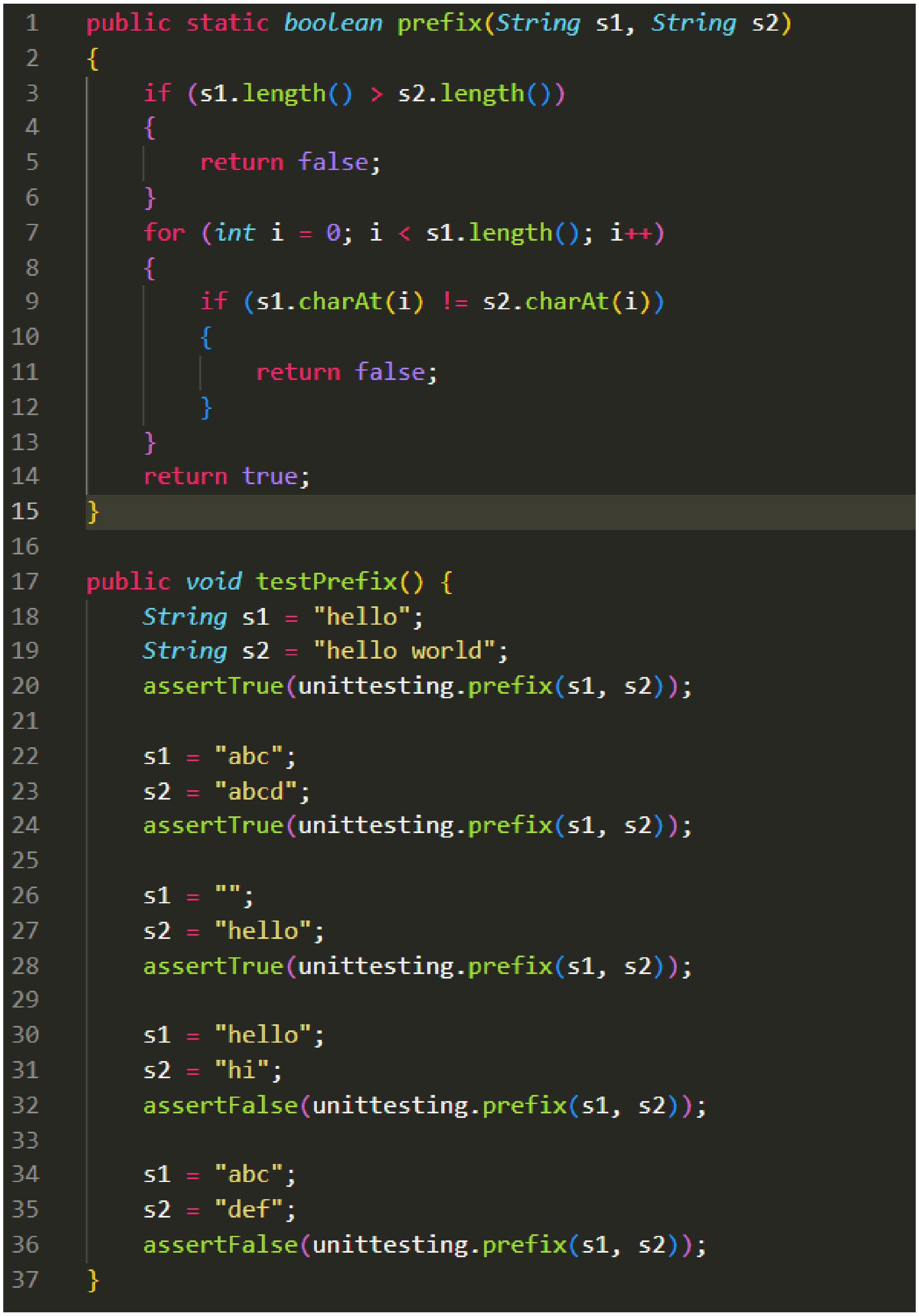
Input	Expected Output	a	b	c	Actual Output
Invalid (a+b<=c)	INVALID	1	1	4	INVALID
Valid Triangle	EQUILATERAL	5	5	5	EQUILATERAL
Valid isosceles triangle (a=b<c)	ISOSCELES	4	4	6	ISOSCELES
Valid scalene triangle (a<b<c)	SCALENE	3	4	5	SCALENE

Boundary value Analysis:

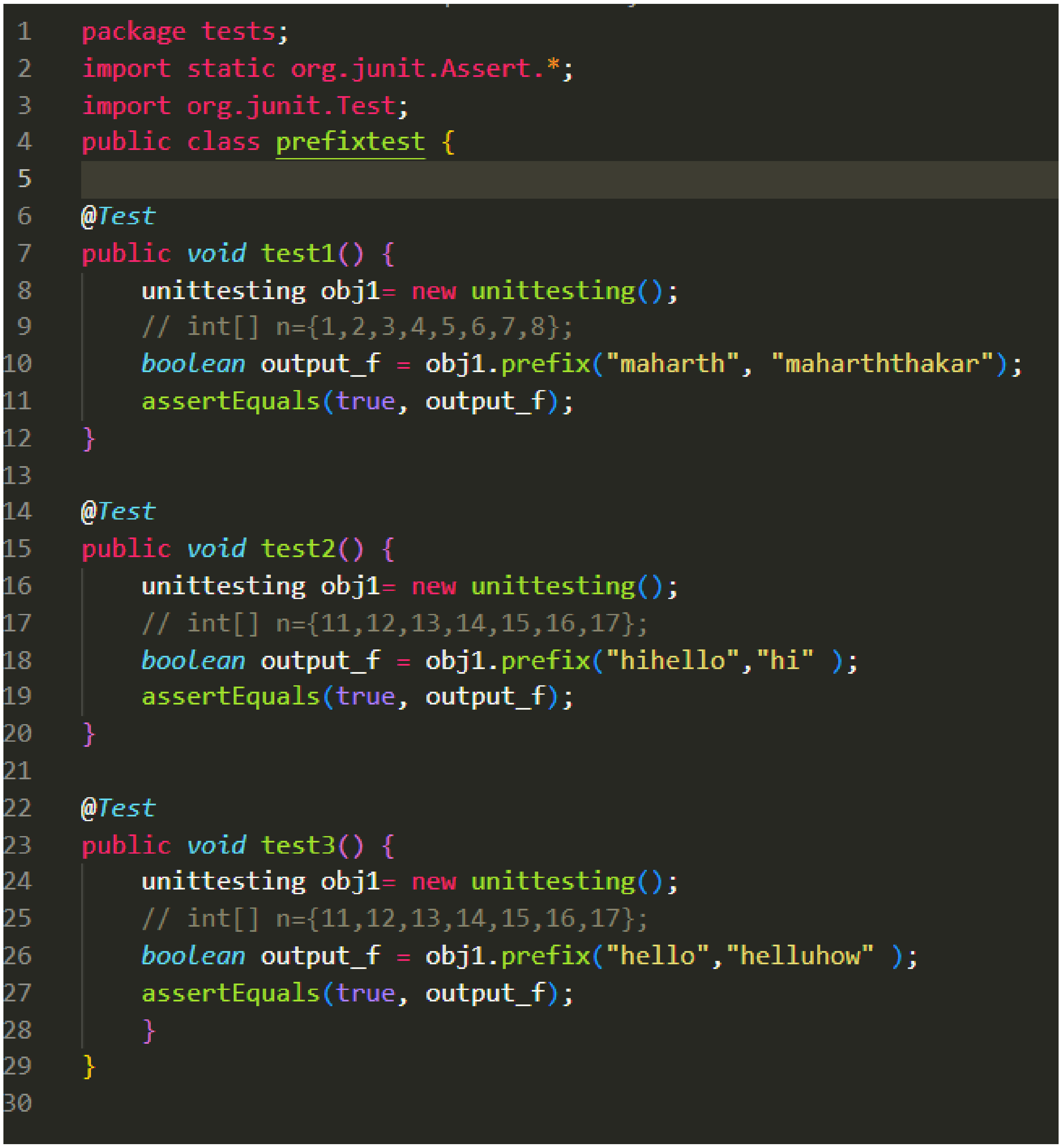
Input	Expected Output	a	b	c	Actual Output
Invalid triangle (a+b<=c)	INVALID	1	3	4	INVALID
Invalid triangle (a+c<b)	INVALID	1	5	1	INVALID
Invalid triangle (b+c<a)	INVALID	4	1	1	INVALID

Valid equilateral triangle (a=b=c)	EQUILATERAL	3	3	3	EQUILATERAL
Valid isosceles triangle (a=b<c)	ISOSCELES	2	2	3	ISOSCELES
Valid isosceles triangle (a=c<b)	ISOSCELES	2	3	2	ISOSCELES
Valid isosceles triangle (b=c<a)	ISOSCELES	3	2	2	ISOSCELES
Valid scalene triangle (a<b<c)	SCALENE	3	4	5	SCALENE

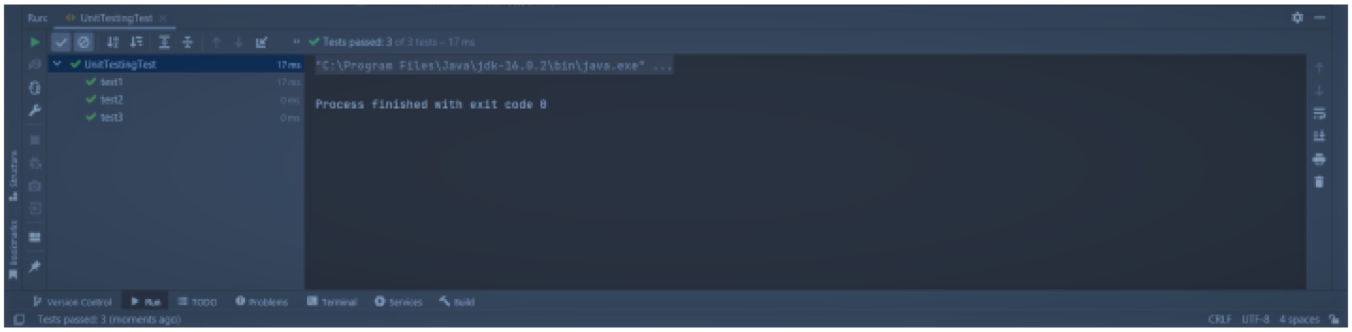
P5. prefix



```java
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 void testPrefix() {
    String s1 = "hello";
    String s2 = "hello world";
    assertTrue(unittesting.prefix(s1, s2));

    s1 = "abc";
    s2 = "abcd";
    assertTrue(unittesting.prefix(s1, s2));

    s1 = "";
    s2 = "hello";
    assertTrue(unittesting.prefix(s1, s2));

    s1 = "hello";
    s2 = "hi";
    assertFalse(unittesting.prefix(s1, s2));

    s1 = "abc";
    s2 = "def";
    assertFalse(unittesting.prefix(s1, s2));
}
```



```java
package tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class prefixtest {

@Test
public void test1() {
    unittesting obj1= new unittesting();
    // int[] n={1,2,3,4,5,6,7,8};
    boolean output_f = obj1.prefix("maharth", "maharththakar");
    assertEquals(true, output_f);
}

@Test
public void test2() {
    unittesting obj1= new unittesting();
    // int[] n={11,12,13,14,15,16,17};
    boolean output_f = obj1.prefix("hihello","hi" );
    assertEquals(true, output_f);
}

@Test
public void test3() {
    unittesting obj1= new unittesting();
    // int[] n={11,12,13,14,15,16,17};
    boolean output_f = obj1.prefix("hello","helluhow" );
    assertEquals(true, output_f);
    }
}

```



Tests passed: 3 of 3 tests – 17 ms

```
UnitTestingTest 17 ms
  test1 17 ms
  test2 0 ms
  test3 0 ms
```

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 3 (moments ago)

Equivalence Classes:

Input	Expected Output	s1	s2	Actual Output
Empty string s1 and s2	True	“”	“”	True
Empty string s1 and non-empty s2	True	“”	“a”	True
Non-empty s1 is a prefix of non-empty s2	True	“ab”	“abc”	True
Non-empty s1 is not a prefix of s2	False	“ab”	“bac”	False
Non-empty s1 is longer than s2	False	“abc”	“ab”	False

Boundary Value Analysis:

Input	Expected Output	s1	s2	Actual Output
Empty string s1 and s2	True	“”	“”	True
Empty string s1 and non-empty s2	True	“”	“a”	True

Non-empty s1 is not a prefix of s2	False	“ab”	“bac”	False
Non-empty s1 is longer than s2	False	“abc”	“ab”	False

P6:

a) Identify the equivalence classes for the system

Equivalence Classes:

EC1: All sides are positive, real numbers.

EC2: One or more sides are negative or zero.

EC3: The sum of the lengths of any two sides is less than or equal to the length of the remaining side (impossible lengths).

EC4: The sum of the lengths of any two sides is greater than the length of the remaining side (possible lengths).

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

Test cases:

TC1 (EC1): A=3, B=4, C=5 (right-angled triangle)

TC2 (EC1): A=5, B=5, C=5 (equilateral triangle)

TC3 (EC1): A=5, B=6, C=7 (scalene triangle)

TC4 (EC1): A=5, B=5, C=7 (isosceles triangle)

TC5 (EC2): A=-2, B=4, C=5 (invalid input)

TC6 (EC2): A=0, B=4, C=5 (invalid input)

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

Test cases for the boundary condition A + B > C: TC7 (EC4): A=2, B=3, C=6 (sum of A and B is equal to C)

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

Test cases for the boundary condition A = C: TC8 (EC4): A=5, B=6, C=5 (A equals to C)

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

Test cases for the boundary condition A = B = C: TC9 (EC4): A=1, B=1, C=1 (all sides are equal)

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

Test cases for the boundary condition A^2 + B^2 = C^2: TC10 (EC4): A=3, B=4, C=5 (right -angled triangle)

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

Test cases for the non-triangle case: TC11 (EC3): A=2, B=2, C=4 (sum of A and B is less than C)

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

Test points for non-positive input:

TP1 (EC2): A=0, B=4, C=5 (invalid input)

TP2 (EC2): A=-2, B=4, C=5 (invalid input) Note: Test cases TC1 to TC10 covers all identified equivalence classes.

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 ith 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.

```
Vector doGraham(Vector p) {
        int i,j,min,M;

        Point t;
        min = 0;

        // search for minimum:
        for(i=1; i < p.size(); ++i) {
            if( ((Point) p.get(i)).y <
                        ((Point) p.get(min)).y )
            {
                min = i;
            }
        }

        // continue along the values with same y component
        for(i=0; i < p.size(); ++i) {
            if(( ((Point) p.get(i)).y ==
                         ((Point) p.get(min)).y ) &&
                 (((Point) p.get(i)).x >
                         ((Point) p.get(min)).x ))
            {
                 min = i;
            }
        }
```

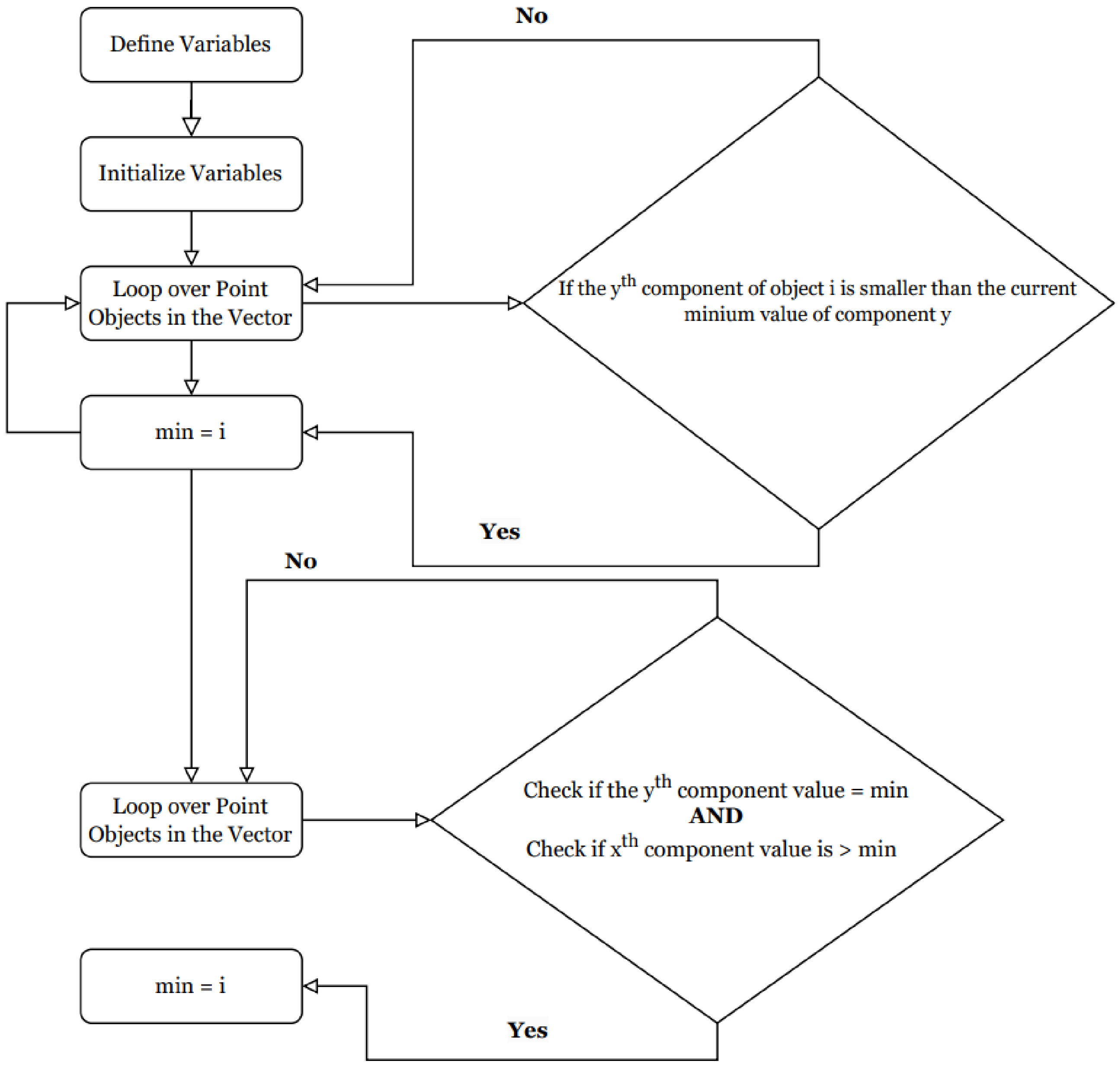
For the given code fragment you should carry out the following activities.

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.

Answer:

1. Control Flow Diagram


Define Variables
Initialize Variables
Loop over Point Objects in the Vector
If the y th component of object i is smaller than the current minium value of component y
No
Yes
min = i
No
Loop over Point Objects in the Vector
Check if the y th component value = min
AND
Check if x th component value is > min
min = i
Yes

Statement coverage test sets: To achieve statement coverage, we need to make sure that every statement in the code is executed at least once.

- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component
- Test 4: p = vector with two points with different y components
- Test 5: p = vector with three or more points with different y components
- Test 6: p = vector with three or more points with the same y component

Branch coverage test sets: To achieve branch coverage, we need to make sure that every possible branch in the code is taken at least once

- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component
- Test 4: p = vector with two points with different y components
- Test 5: p = vector with three or more points with different y components, and none of them have the same x component
- Test 6: p = vector with three or more points with the same y component, and some of them have the same x component
- Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

Basic condition coverage test sets: To achieve basic condition coverage, we need to make sure that every basic condition in the code (i.e., every Boolean subexpression) is evaluated as both true and false at least once

- Test 1: p = empty vector
- Test 2: p = vector with one point
- Test 3: p = vector with two points with the same y component, and the first point has a smaller x component

- Test 4: p = vector with two points with the same y component, and the second point has a smaller x component
- Test 5: p = vector with two points with different y components
- Test 6: p = vector with three or more points with different y components, and none of them have the same x component
- Test 7: p = vector with three or more points with the same y component, and some of them have the same x component
- Test 8: p = vector with three or more points with the same y component, and all of them have the same x component.