

IT314 – Software Engineering

Name: Bhatt Meet Dhavalbhai

Student ID: 202001267

Lab 7

Section A

Consider a program for determining the previous date. Its input is a triple of day, month and year with the following ranges $1 \leq \text{month} \leq 12$, $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$. The possible output dates would be the previous date or invalid date. Design the equivalence class test cases?

Solution:

From the given constraints $1 \leq \text{month} \leq 12$, $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$, The following are the equivalence classes obtained.

Equivalent Classes:

There are a total of 9 equivalence classes.

$$E1 = \{1 \leq \text{date} \leq 31\}$$

$$E2 = \{\text{date} < 1\}$$

$$E3 = \{\text{date} > 31\}$$

$$E4 = \{1 \leq \text{month} \leq 12\}$$

$$E5 = \{\text{month} < 1\}$$

$$E6 = \{\text{month} > 12\}$$

$$E7 = \{1900 \leq \text{year} \leq 2015\}$$

$$E8 = \{\text{year} < 1900\}$$

$$E9 = \{\text{year} > 2015\}$$

The following are the weak normal equivalence class test cases:

Equivalent Class	Day	Month	Year	Output
E1	2	3	2011	1/3/2011
E2	0	4	2022	Invalid
E3	34	5	2000	Invalid
E4	1	1	1980	31/12/1989
E5	21	-4	1970	Invalid
E6	20	15	1943	Invalid
E7	4	5	1980	3/5/1980
E8	5	6	1899	Invalid
E9	4	3	2016	Invalid

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.

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

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
'v' is not present in array 'a'	-1
'v' is present in array 'a'	Index of 'v'

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array 'a'	-1
'v' is present at 2nd index in array 'a'	2
'v' is not present in array 'a'	-1

Test Cases:

1. v = 2, a = {4,3,2}, Expected Output: 2
2. v = 3, a = {4,2,1}, Expected Output: -1
3. v = 4, a = {}, Expected Output: -1
4. v = 2, a = {1,2,3,2,4}, Expected Output: 1

JUnit Testing:

```

1 package tests;
2
3 public class UnitTesting {
4     // Program 1
5     public int linearSearch(int v, int a[])
6     {
7         int i = 0;
8         while (i < a.length)
9         {
10             if (a[i] == v)
11                 return(i);
12             i++;
13         }
14         return (-1);
15     }
16
17     // Program 2
18     public int countItem(int v, int a[])
19     {
20         int count = 0;
21         for (int i = 0; i < a.length; i++)
22         {
23             if (a[i] == v)
24                 count++;
25         }
26         return count;
27     }
28
29     // Program 3
30     int binarySearch(int v, int a[])
31     {
32         int lo,mid,hi;
33         lo = 0;
34         hi = a.length-1;
35         while (lo <= hi)
36         {
37             mid = (lo+hi)/2;
38             if (v == a[mid])
39                 return (mid);
40             else if (v < a[mid])
41                 hi = mid-1;
42             else
43                 lo = mid+1;
44         }
45     }
46 }

```

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with files like UnitTesting.java, P1.java, and module-info.java.
- JUnit View:** Displays the test results:
 - Finished after 0.007 seconds
 - Runs: 4/4
 - Errors: 0
 - Failures: 0
- Code Editor:** Shows the source code for UnitTesting.java, which contains four JUnit tests (test1, test2, test3, test4) for the linearSearch method.
- Outline View:** Shows the class structure with methods test1(), test2(), test3(), and test4().
- Problems View:** Shows a table with 0 items.

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);
}

```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
'v' is not present in array 'a'	0
'v' is present in array 'a'	Number of times 'v' appears in array 'a'

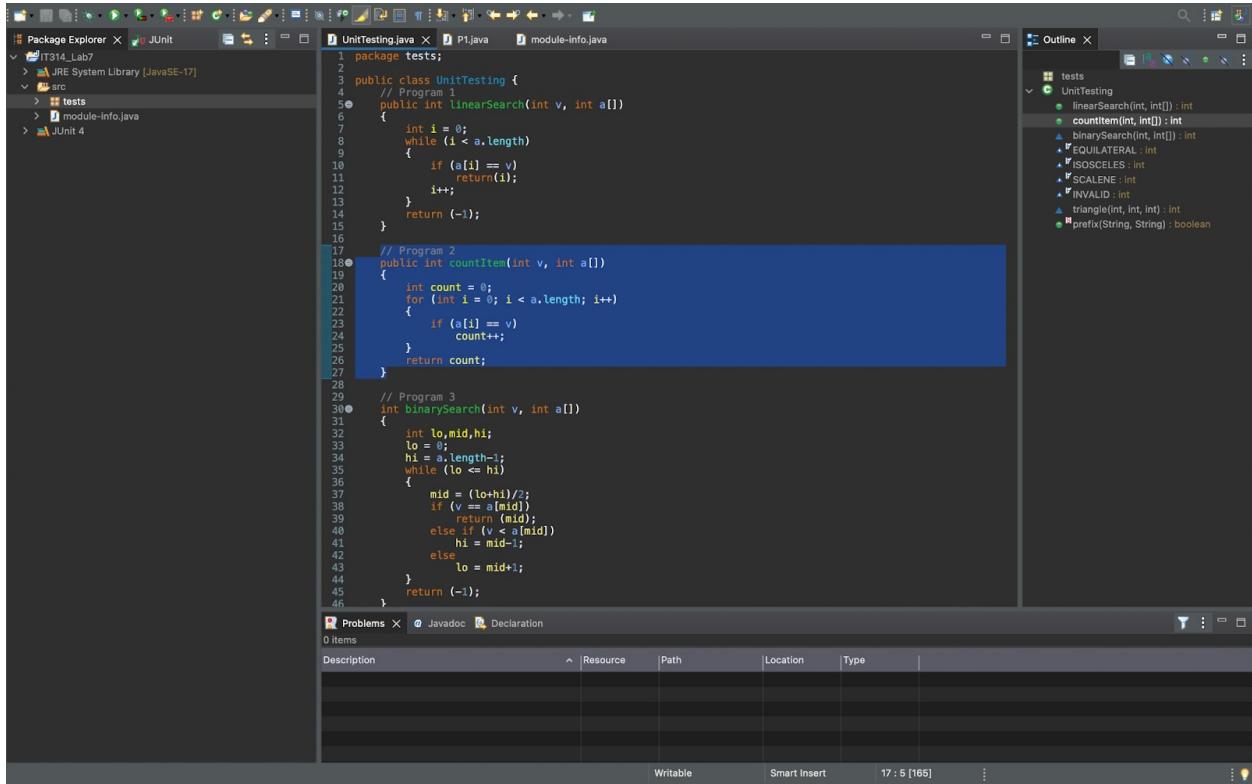
Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array 'a'	-1
'v' is present once in array 'a'	1
'v' is present multiple times in array 'a'	Number of times 'v' is present
'v' is not present in array 'a'	0

Test Cases:

1. $v = 2, a = \{4,2,3,2,1\}$, Expected Output: 2
2. $v = 3, a = \{4,2,3\}$, Expected Output: 1
3. $v = 20, a = \{1,2,3\}$, Expected Output: 0
4. $v = 1, a = \{\}$, Expected Output: 0

JUnit Testing:



The screenshot shows the Eclipse IDE interface with the following components visible:

- Package Explorer:** Shows the project structure with packages like `I314_Lab7`, `JRE System Library [JavaSE-17]`, and `src`.
- JUnit View:** Shows the `JUnit` tab.
- Outline View:** Shows the class `UnitTesting` with methods: `linearSearch(int v, int a[])`, `countItem(int v, int a[])`, `binarySearch(int v, int a[])`, and others.
- Problems View:** Shows 0 items.
- Code Editor:** Displays the `UnitTesting.java` file containing Java code for three programs: linear search, counting occurrences, and binary search.

```

1 package tests;
2
3 public class UnitTesting {
4     // Program 1
5     public int linearSearch(int v, int a[])
6     {
7         int i = 0;
8         while (i < a.length)
9         {
10             if (a[i] == v)
11                 return(i);
12             i++;
13         }
14         return (-1);
15     }
16
17     // Program 2
18     public int countItem(int v, int a[])
19     {
20         int count = 0;
21         for (int i = 0; i < a.length; i++)
22         {
23             if (a[i] == v)
24                 count++;
25         }
26         return count;
27     }
28
29     // Program 3
30     int binarySearch(int v, int a[])
31     {
32         int lo,mid,hi;
33         lo = 0;
34         hi = a.length-1;
35         while (lo <= hi)
36         {
37             mid = (lo+hi)/2;
38             if (v == a[mid])
39                 return (mid);
40             else if (v < a[mid])
41                 hi = mid-1;
42             else
43                 lo = mid+1;
44         }
45         return (-1);
46     }
}

```

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows a project named "tests" containing files "P1.java" and "P2.java".
- JUnit View:** Displays "JUnit" results with "Runs: 4/4", "Errors: 0", and "Failures: 0".
- Code Editor:** The "P2.java" file is open, showing the following code:


```

1 package tests;
2
3 import static org.junit.Assert.*;
4
5 public class P2 {
6
7     @Test
8     public void test1() {
9         UnitTesting obj = new UnitTesting();
10        int v = 2;
11        int arr[] = {4,2,3,2,1};
12        int output = obj.countItem(v, arr);
13        int expected = 2;
14        assertEquals(expected, output);
15    }
16
17    @Test
18    public void test2() {
19        UnitTesting obj = new UnitTesting();
20        int v = 1;
21        int arr[] = {4,2,3};
22        int output = obj.countItem(v, arr);
23        int expected = 1;
24        assertEquals(expected, output);
25    }
26
27    @Test
28    public void test3() {
29        UnitTesting obj = new UnitTesting();
30        int v = 20;
31        int arr[] = {1,2,3};
32        int output = obj.countItem(v, arr);
33        int expected = 0;
34        assertEquals(expected, output);
35    }
36
37    @Test
38    public void test4() {
39        UnitTesting obj = new UnitTesting();
40        int v = 1;
41        int arr[] = {};
42        int output = obj.countItem(v, arr);
43        int expected = 0;
44        assertEquals(expected, output);
45    }
46}
      
```
- Outline View:** Shows the class structure with methods: test1(), test2(), test3(), and test4().
- Failure Trace:** An empty list.
- Problems View:** An empty list.

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 increasing 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);
}

```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
'v' is not present in array 'a'	-1
'v' is present in array 'a'	Index of 'v' in array 'a'

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array 'a'	-1
'v' is present at first index in array 'a'	0

'v' is not present in array 'a'	-1
---------------------------------	----

Test Cases:

1. v = 2, a = {0,1,2,3,4}, Expected Output: 2
2. v = -4, a = {1,2,3,4,5}, Expected Output: -1
3. v = 5, a = {2,3,4,5,5,6}, Expected Output: 3 or 4

JUnit Testing:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows a project named "IT314_Lab7" containing "src" and "tests" packages. "src" contains P1.java, P2.java, and P3.java. "tests" contains UnitTesting.java.
- UnitTesting.java (Active Editor):**

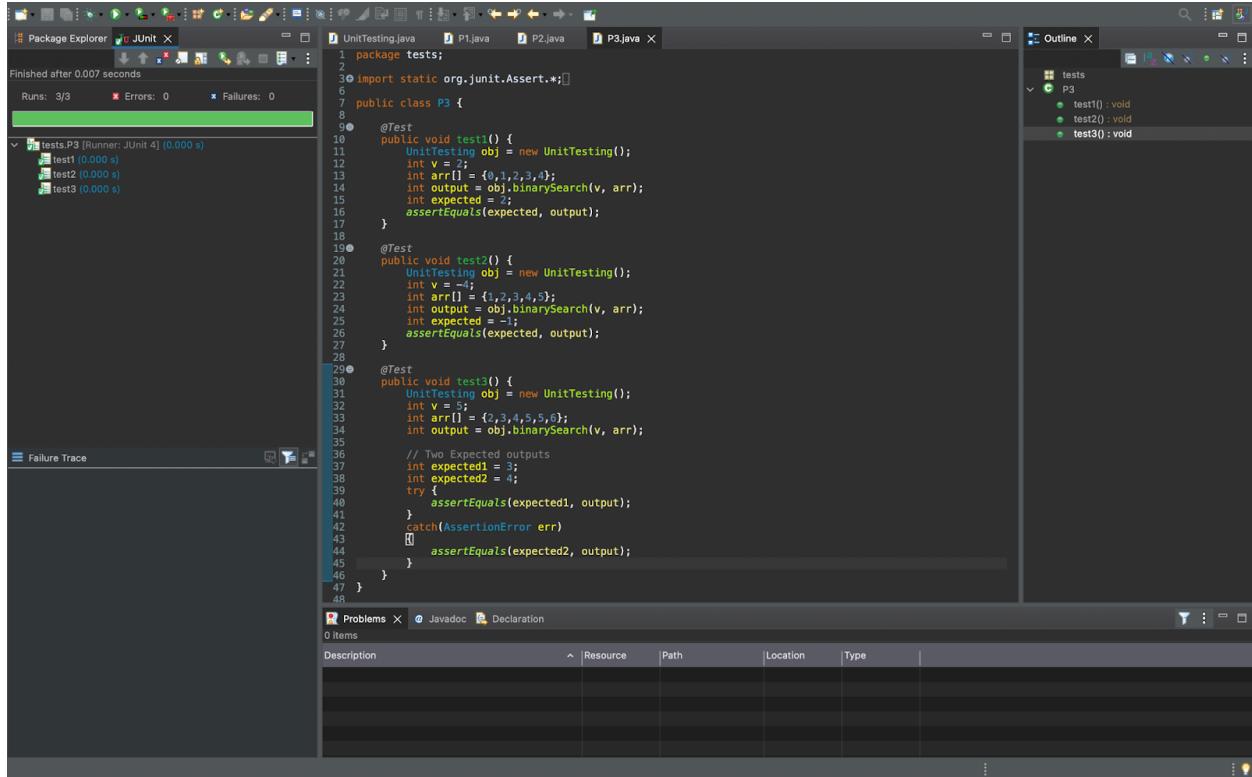
```

22     {
23         if (a[i] == v)
24             count++;
25     }
26     return count;
27 }

29 // Program 3
30 int binarySearch(int v, int a[])
31 {
32     int lo,mid,hi;
33     lo = 0;
34     hi = a.length-1;
35     while (lo <= hi)
36     {
37         mid = (lo+hi)/2;
38         if (v == a[mid])
39             return (mid);
40         else if (v < a[mid])
41             hi = mid-1;
42         else
43             lo = mid+1;
44     }
45     return (-1);
46 }

47 // Program 4
48 final int EQUILATERAL = 0;
49 final int ISOSCELES = 1;
50 final int SCALENE = 2;
51 final int INVALID = 3;
52 int triangle(int a, int b, int c)
53 {
54     if (a >= b+c || b >= a+c || c >= a+b)
55         return (INVALID);
56     if (a == b && b == c)
57         return (EQUILATERAL);
58     if (a == b || a == c || b == c)
59         return (ISOSCELES);
60     return (SCALENE);
61 }

62 // Program 5
63 public static boolean prefix(String s1, String s2)
64 {
65 }
```
- Outline View:** Shows a tree structure of test methods under the "tests" package, including linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, and INVALID.
- Problems View:** Shows 0 items.
- Bottom Status Bar:** Displays "46 : 6 [248]".



The screenshot shows an IDE interface with several windows:

- Package Explorer:** Shows files UnitTesting.java, P1.java, P2.java, and P3.java.
- JUnit Runner:** Shows "Finished after 0.007 seconds" with "Runs: 3/3", "Errors: 0", and "Failures: 0".
- Code Editor:** Displays the content of P3.java, which contains JUnit test cases for a binary search method.
- Outline:** Shows the class structure with methods test1(), test2(), and test3().
- Failure Trace:** Shows the execution details of the tests.
- Problems:** Shows 0 items.

```

1 package tests;
2
3 import static org.junit.Assert.*;
4
5 public class P3 {
6
7     @Test
8     public void test1() {
9         UnitTesting obj = new UnitTesting();
10        int v = 2;
11        int arr[] = {0,1,2,3,4};
12        int output = obj.binarySearch(v, arr);
13        int expected = 2;
14        assertEquals(expected, output);
15    }
16
17    @Test
18    public void test2() {
19        UnitTesting obj = new UnitTesting();
20        int v = -1;
21        int arr[] = {1,2,3,4,5};
22        int output = obj.binarySearch(v, arr);
23        int expected = -1;
24        assertEquals(expected, output);
25    }
26
27    @Test
28    public void test3() {
29        UnitTesting obj = new UnitTesting();
30        int v = 5;
31        int arr[] = {2,3,4,5,6};
32        int output = obj.binarySearch(v, arr);
33
34        // Two Expected outputs
35        int expected1 = 3;
36        int expected2 = 4;
37        try {
38            assertEquals(expected1, output);
39        } catch(AssertionError err) {
40            assertEquals(expected2, output);
41        }
42    }
43
44}
45
46}
47
48

```

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);
}

```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Invalid triangle ($a+b \leq c$)	INVALID
Valid equilateral triangle ($a=b=c$)	EQUILATERAL
Valid isosceles triangle ($a=b < c$)	ISOSCELES
Valid scalene triangle ($a < b < c$)	SCALENE

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
-------------------------------------	-------------------------

Invalid triangle ($a+b \leq c$)	INVALID
Invalid triangle ($a+c \leq b$)	INVALID
Invalid triangle ($b+c \leq a$)	INVALID
Valid equilateral triangle ($a=b=c$)	EQUILATERAL
Valid isosceles triangle ($a=b < c$)	ISOSCELES
Valid isosceles triangle ($a=c < b$)	ISOSCELES
Valid isosceles triangle ($b=c < a$)	ISOSCELES
Valid scalene triangle ($a < b < c$)	SCALENE

Test Cases:

1. $a = 4, b = 4, c = 4$, Expected Output: EQUILATERAL
2. $a = 1, b = 2, c = 3$, Expected Output: INVALID
3. $a = -1, b = 2, c = 3$, Expected Output: INVALID
4. $a = 3, b = 4, c = 5$, Expected Output: SCALENE
5. $a = 5, b = 5, c = 9$, Expected Output: ISOSCELES
6. $a = 5, b = 5, c = 10$, Expected Output: INVALID

JUnit Testing:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with files P1.java, P2.java, P3.java, P4.java, and UnitTesting.java.
- UnitTesting.java Content:**

```

35     while (lo <= hi)
36     {
37         mid = (lo+hi)/2;
38         if (v == a[mid])
39             return (mid);
40         else_if (v < a[mid])
41             hi = mid-1;
42         else
43             lo = mid+1;
44     }
45     return (-1);
46 }

// Program 4
49 final int EQUILATERAL = 0;
50 final int ISOSCELES = 1;
51 final int SCALENE = 2;
52 final int INVALID = 3;
53
54 {
55     if (a >= b+c || b >= a+c || c >= a+b)
56         return (INVALID);
57     if (a == b && b == c)
58         return (EQUILATERAL);
59     if (a == b || a == c || b == c)
60         return (ISOSCELES);
61     return (SCALENE);
62 }

// Program 5
64 public static boolean prefix(String s1, String s2)
65 {
66     if (s1.length() > s2.length())
67     {
68         return false;
69     }
70     for (int i = 0; i < s1.length(); i++)
71     {
72         if (s1.charAt(i) != s2.charAt(i))
73         {
74             return false;
75         }
76     }
77     return true;
78 }

```
- Outline View:** Shows the class structure and methods: linearSearch(int, int[]), countItem(int, int[]), binarySearch(int, int[]), EQUILATERAL, ISOSCELES, SCALENE, INVALID, triangle(int, int, int), and prefix(String, String).
- Problems View:** Shows 0 items.

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with files P1.java, P2.java, P3.java, P4.java, and UnitTesting.java.
- UnitTesting.java Content:**

```

22
23
24 @Test
25 public void test2() {
26     UnitTesting obj = new UnitTesting();
27     int a = 1, b = 2, c = 3;
28     int output = obj.triangle(a, b, c);
29     int expected = INVALID;
30     assertEquals(expected, output);
31 }
32
33 @Test
34 public void test3() {
35     UnitTesting obj = new UnitTesting();
36     int a = -1, b = 2, c = 3;
37     int output = obj.triangle(a, b, c);
38     int expected = INVALID;
39     assertEquals(expected, output);
40 }
41
42 @Test
43 public void test4() {
44     UnitTesting obj = new UnitTesting();
45     int a = 1, b = 3, c = 5;
46     int output = obj.triangle(a, b, c);
47     int expected = SCALENE;
48     assertEquals(expected, output);
49 }
50
51 @Test
52 public void test5() {
53     UnitTesting obj = new UnitTesting();
54     int a = 5, b = 5, c = 9;
55     int output = obj.triangle(a, b, c);
56     int expected = ISOSCELES;
57     assertEquals(expected, output);
58 }
59
60 @Test
61 public void test6() {
62     UnitTesting obj = new UnitTesting();
63     int a = 5, b = 5, c = 10;
64     int output = obj.triangle(a, b, c);
65     int expected = INVALID;
66     assertEquals(expected, output);
67 }

```
- Outline View:** Shows the class structure and methods: test1(), test2(), test3(), test4(), test5(), test6(), EQUILATERAL, ISOSCELES, SCALENE, INVALID, triangle(int, int, int), and prefix(String, String).
- Problems View:** Shows 0 items.

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;
}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Empty string <code>s1</code> and <code>s2</code>	True
Empty string <code>s1</code> and non-empty <code>s2</code>	True
Non-empty <code>s1</code> is a prefix of non-empty <code>s2</code>	True
Non-empty <code>s1</code> is not a prefix of <code>s2</code>	False
Non-empty <code>s1</code> is longer than <code>s2</code>	False

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
-------------------------------------	-------------------------

Empty string s1 and s2	True
Empty string s1 and non-empty s2	True
Non-empty s1 is not a prefix of s2	False
Non-empty s1 is longer than s2	False

Test Cases:

1. s1 = "soft", s2 = "software", Expected Output: true
2. s1 = "abd", s2 = "abc", Expected Output: False
3. s1 = "health", s2 = "health", Expected Output: True
4. s1 = "one", s2 = "two", Expected Output: False
5. s1 = "", s2 = "pdf", Expected Output: True

JUnit Testing:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with packages like IT314_Lab7, JRE System Library [JavaSE-17], and src. Under src, there are tests, P1.java, P2.java, P3.java, P4.java, P5.java, and UnitTesting.java.
- UnitTesting.java Content:**

```

36     {
37         mid = (lo+hi)/2;
38         if (v == a[mid])
39             return (mid);
40         else if (v < a[mid])
41             hi = mid-1;
42         else
43             lo = mid+1;
44     }
45     return (-1);
46 }
47 // Program 4
48 final int EQUILATERAL = 0;
49 final int ISOSCELES = 1;
50 final int SCALENE = 2;
51 final int INVALID = 3;
52 int triangle(int a, int b, int c)
53 {
54     if (a >= b+c || b >= a+c || c >= a+b)
55         return (INVALID);
56     if (a == b && b == c)
57         return (EQUILATERAL);
58     if (a == b || a == c || b == c)
59         return (ISOSCELES);
60     return (SCALENE);
61 }
62
63 // Program 5
64 public boolean prefix(String s1, String s2)
65 {
66     if (s1.length() > s2.length())
67     {
68         return false;
69     }
70     for (int i = 0; i < s1.length(); i++)
71     {
72         if (s1.charAt(i) != s2.charAt(i))
73         {
74             return false;
75         }
76     }
77     return true;
78 }
79
80 }
```
- Outline View:** Shows the class UnitTesting with methods linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, and INVALID, along with a static method triangle and a boolean method prefix.
- Problems View:** Shows 0 items.

```

1  package tests;
2
3  import org.junit.*;
4
5  public class P6 {
6
7      @Test
8      public void test1() {
9          UnitTesting obj = new UnitTesting();
10         String s1 = "soft", s2 = "software";
11         boolean output = obj.prefix(s1, s2);
12         boolean expected = true;
13         assertEquals(expected, output);
14     }
15
16     @Test
17     public void test2() {
18         UnitTesting obj = new UnitTesting();
19         String s1 = "abd", s2 = "abc";
20         boolean output = obj.prefix(s1, s2);
21         boolean expected = false;
22         assertEquals(expected, output);
23     }
24
25     @Test
26     public void test3() {
27         UnitTesting obj = new UnitTesting();
28         String s1 = "health", s2 = "health";
29         boolean output = obj.prefix(s1, s2);
30         boolean expected = true;
31         assertEquals(expected, output);
32     }
33
34     @Test
35     public void test4() {
36         UnitTesting obj = new UnitTesting();
37         String s1 = "one", s2 = "two";
38         boolean output = obj.prefix(s1, s2);
39         boolean expected = false;
40         assertEquals(expected, output);
41     }
42
43     @Test
44     public void test5() {
45         UnitTesting obj = new UnitTesting();
46         String s1 = "", s2 = "pdf";
47         boolean output = obj.prefix(s1, s2);
48         boolean expected = true;
49         assertEquals(expected, output);
50     }
51
52 }

```

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

The following are the equivalence classes for different types of triangles.

1. INVALID case:

- E1: $a + b \leq c$
- E1: $a + c \leq b$
- E1: $b + c \leq a$

2. EQUILATERAL case:

→ E1: $a = b, b = c, c = a$

3. ISOSCELES case:

→ E1: $a = b, a \neq c$

→ E1: $a = c, a \neq b$

→ E1: $b = c, b \neq a$

4. SCALENE case:

→ E1: $a \neq b, b \neq c, c \neq a$

5. RIGHT-ANGLED TRIANGLE case:

→ E1: $a^2 + b^2 = c^2$

→ E1: $b^2 + c^2 = a^2$

→ E1: $a^2 + c^2 = b^2$

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)

Test Case	Output	Equivalence Class
$a = 1.5, b = 2.6, c = 4.1$	INVALID	E1
$a = -1.6, b = 5, c = 6$	INVALID	E2
$a = 7.1, b = 6.1, c = 1$	INVALID	E3
$a = 5.5, b = 5.5, c = 5.5$	EQUILATERAL	E4
$a = 4.5, b = 4.5, c = 5$	ISOSCELES	E5
$a = 6, b = 4, c = 6$	ISOSCELES	E6

$a = 8, b = 5, c = 5$	ISOSCELES	E7
$a = 6, b = 7, c = 8$	SCALENE	E8
$a = 3, b = 4, c = 5$	RIGHT-ANGLED TRIANGLE	E9
$a = 0.13, b = 0.12, c = 0.05$	RIGHT-ANGLED TRIANGLE	E10
$a = 7, b = 25, c = 23$	RIGHT-ANGLED TRIANGLE	E11

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

The test cases to verify boundary condition:

1. $a = 5, b = 4, c = 5$
2. $a = 5, b = 5, c = 9$
3. $a = 5, b = 6, c = 12$

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

The test cases to verify boundary condition:

1. $a = 5, b = 4, c = 5$
2. $a = 5, b = 4, c = 5.1$
3. $a = 5, b = 4, c = 4.9$

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

The test cases to verify boundary condition:

1. $a = 5, b = 5, c = 5$ ($a = b = c$)
2. $a = 10, b = 10, c = 9$ ($a = b$ but $a \neq c$)
3. $a = 10, b = 11, c = 10$ ($a = c$ but $a \neq b$)

f) For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.

The test cases to verify boundary condition:

1. $a = 3, b = 4, c = 5$
2. $a = 5, b = 12, c = 13$

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

The test cases to verify boundary condition:

1. $a = 1, b = 2, c = 3$
2. $a = 4.5, b = 5.5, c = 10$

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

The test points for non-positive inputs:

1. $a = -4.0, b = 3.2, c = 4.5$
2. $a = 5, b = -4.2, c = -3.2$

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;

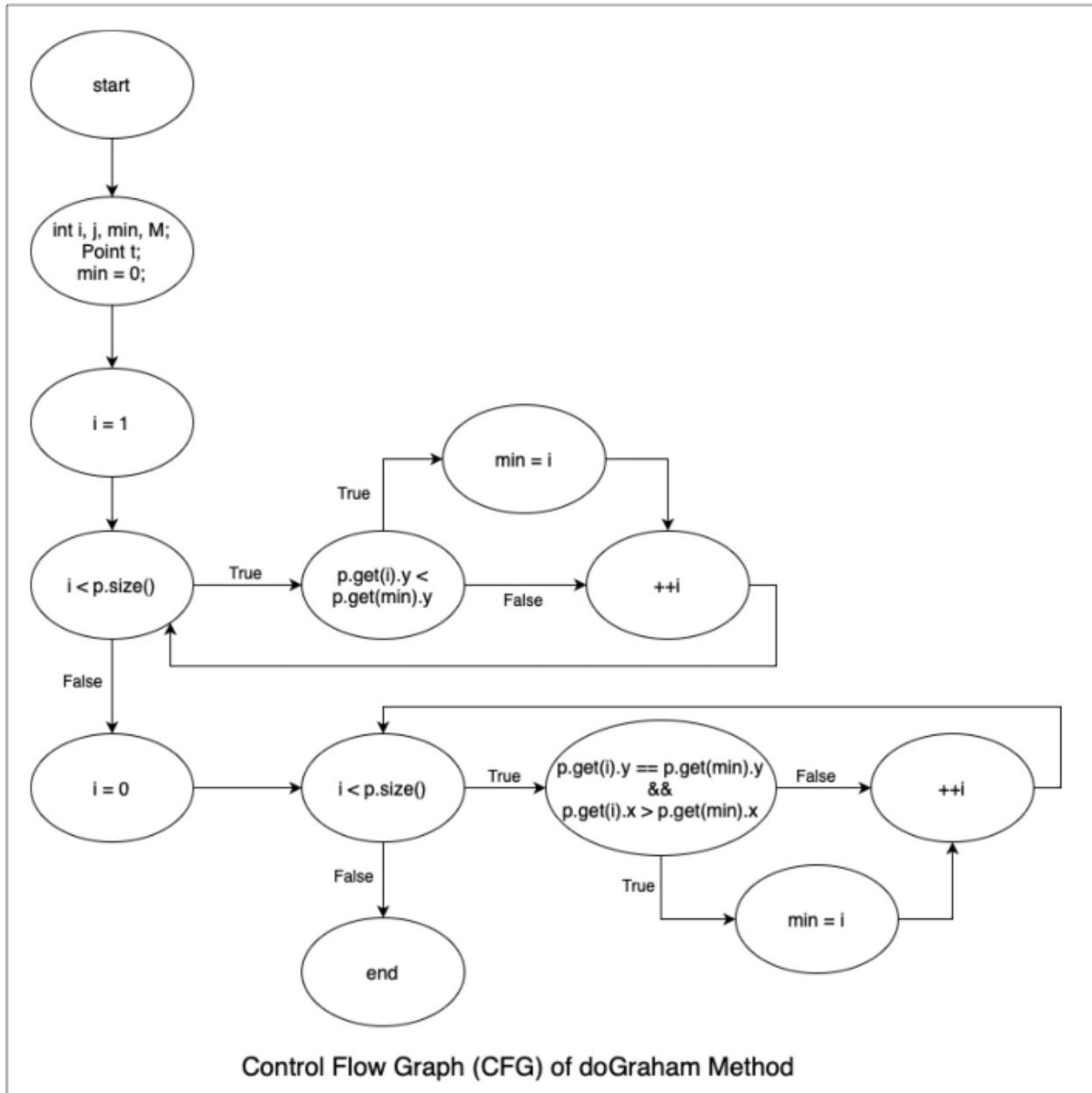
    Point t;
    min = 0;

    // 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:
- Statement Coverage.
 - Branch Coverage.

c. Basic Condition Coverage.

The following are the test cases and their corresponding coverage of statements:

Test cases:

1. $p=[(x = 2, y = 2), (x = 2, y = 3), (x = 1, y = 3), (x = 1, y = 4)]$
 Statements covered = {1, 2, 3, 4, 5, 7, 8}
 Branches covered = {5, 8}
 Basic conditions covered = {5 - false, 8 - false}

2. $p=[(x = 2, y = 3), (x = 3, y = 4), (x = 1, y = 2), (x = 5, y = 6)]$
 Statements covered = {1, 2, 3, 4, 5, 6, 7}
 Branches covered = {5, 8}
 Basic conditions covered = {5 - false, true, 8 - false}

3. $p=[(x = 1, y = 5), (x = 2, y = 7), (x = 3, y = 5), (x = 4, y = 5), (x = 5, y = 6)]$
 Statements covered = {1, 2, 3, 4, 5, 6, 7, 8, 9}
 Branches covered = {5, 8}
 Basic conditions covered = {5 - false, true, 8 - false, true}

4. $p=[(x = 1, y = 2)]$
 Statements covered = {1, 2, 3, 7, 8}
 Branches covered = {8}
 Basic conditions covered = {}

5. $p=[]$
 Statements covered = {1, 2, 3}
 Branches covered = {}
 Basic conditions covered = {}

Thus, the above 5 test cases are covering all statements, branches and conditions. These 5 test cases are adequate for statement coverage, branch coverage and basic condition coverage.