

IT314 - Software Engineering

Lab - 7

Name: Hiten Mistry

Student Id: 202001182

Section A

Q1: Consider a program for determining the previous date. Its input is 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?

Ans:

1. Equivalence Classes for the Day parameter:

- a. Valid classes: $1 \leq \text{day} \leq 31$
- b. Invalid classes: $\text{day} < 1, \text{day} > 31$

2. Equivalence Classes for the Month parameter:

- a. Valid classes: $1 \leq \text{month} \leq 12$
- b. Invalid classes: $\text{month} < 1, \text{month} > 12$

3. Equivalence Classes for the Year parameter:

- a. Valid classes: $1900 \leq \text{year} \leq 2015$
- b. Invalid classes: $\text{year} < 1900, \text{year} > 2015$

Some Test Cases:

1. Valid test cases:

- a. (1, 1, 1900) - the minimum valid date
- b. (15, 7, 2005) - a random valid date
- c. (31, 12, 2015) - the maximum valid date

2. Invalid test cases:

- a. (0, 7, 1999) - day is less than 1
- b. (32, 1, 2008) - day is greater than 31
- c. (29, 2, 1900) - the year is not a leap year, but the day is 29
- d. (31, 4, 2016) - the year is greater than 2015
- e. (15, 13, 1998) - month is greater than 12
- f. (-10, -5, 2022) - all parameters are invalid

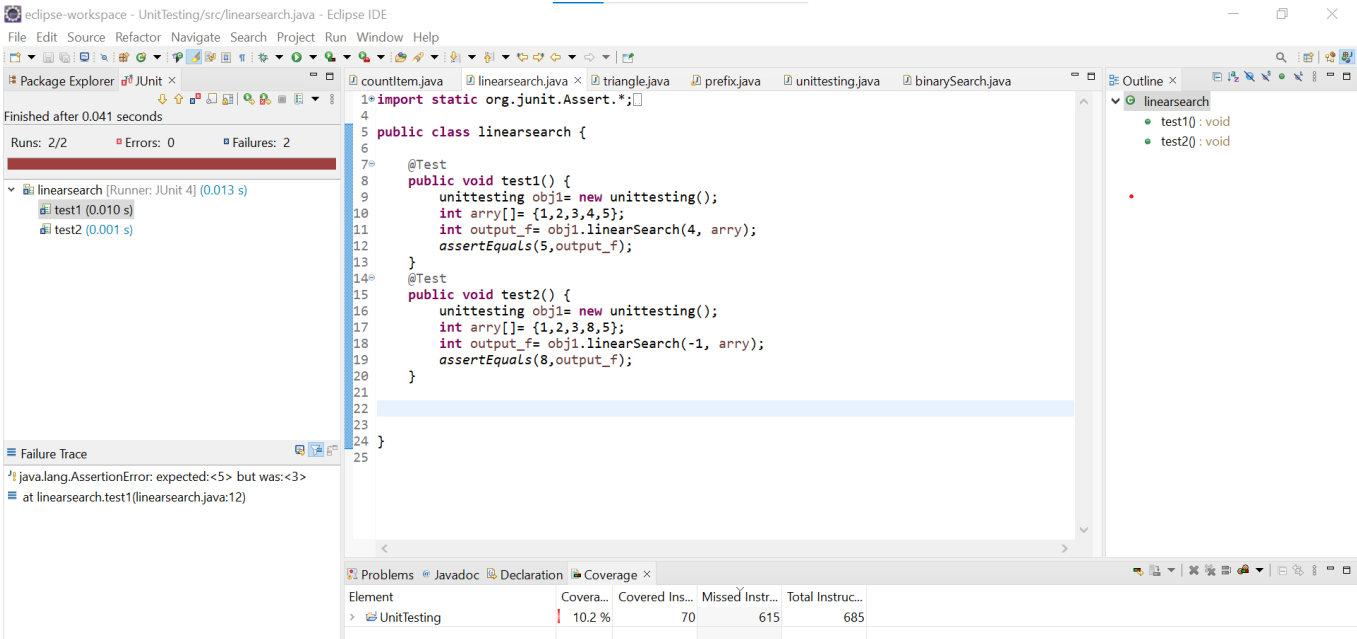
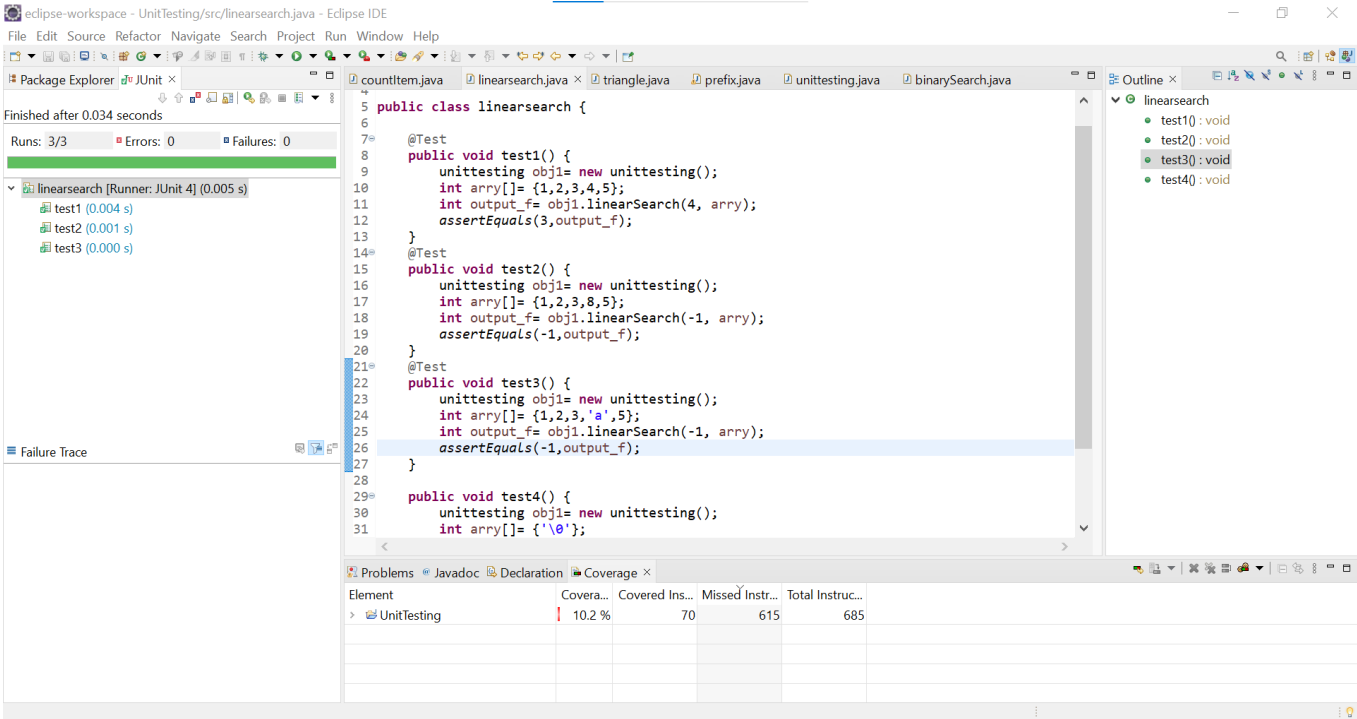
These test cases represent the equivalence classes and should cover all possible scenarios.

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);
}
```

Test Case in Eclipse



Equivalence Partitioning:

| Tester Action and Input Data | Expected Outcome |
|--|-----------------------|
| Test with v as a non-existent value and an empty array a[] | -1 |
| Test with v as a non-existent value and a non-empty array a[] | -1 |
| Test with v as an existent value and an empty array a[] | -1 |
| Test with v as an existent value and a non-empty array a[] where v exists | the index of v in a[] |
| Test with v as an existent value and a non-empty array a[] where v does not exist | -1 |

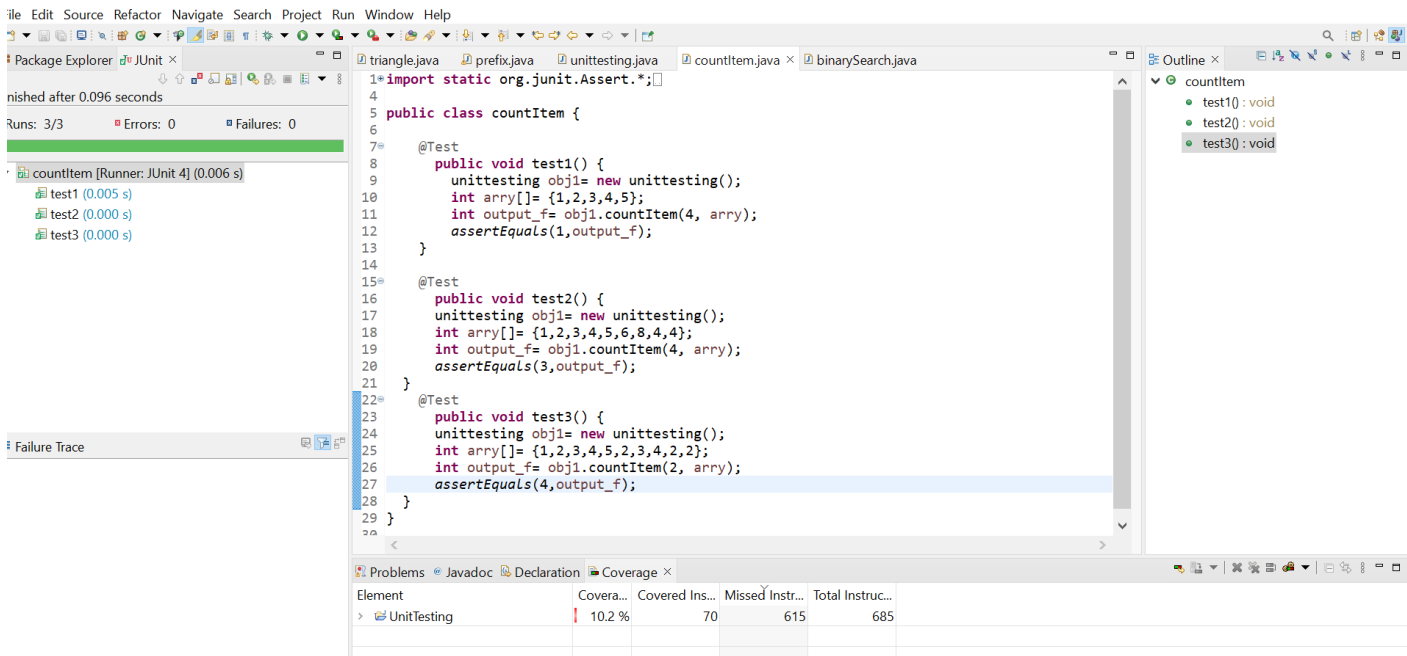
Boundary Value Analysis:

| Tester Action and Input Data | Expected Outcome |
|--|---------------------------------|
| Test with v as a non-existent value and an empty array a[] | -1 |
| Test with v as a non-existent value and a non-empty array a[] | -1 |
| Test with v as an existent value and an array a[] of length 0 | -1 |
| Test with v as an existent value and an array a[] of length 1, where v exists | 0 |
| Test with v as an existent value and an array a[] of length 1, where v does not exist | -1 |
| Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array | 0 |
| Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array | the last index where v is found |

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);  
}
```

Test Case in Eclipse



Package ExplorerJUnit ×

Finished after 0.109 seconds

Runs: 3/3Errors: 0Failures: 3

countItem [Runner: JUnit 4] (0.035 s)

- test1 (0.027 s)
- test2 (0.003 s)
- test3 (0.003 s)

Failure Trace

- java.lang.AssertionError: expected:<0> but was:<1>
- at countItem.test1(countItem.java:12)

triangle.java prefix.java unittesting.java countItem.java × binarySearch.java

```
1*import static org.junit.Assert.*;
4
5 public class countItem {
6
7     @Test
8     public void test1() {
9         unittesting obj1= new unittesting();
10        int array[]= {1,2,3,4,5};
11        int output_f= obj1.countItem(4, array);
12        assertEquals(0,output_f);
13    }
14
15    @Test
16    public void test2() {
17        unittesting obj1= new unittesting();
18        int array[]= {1,2,3,4,5,6,8,4,4};
19        int output_f= obj1.countItem(4, array);
20        assertEquals(2,output_f);
21    }
22
23    @Test
24    public void test3() {
25        unittesting obj1= new unittesting();
26        int array[]= {1,2,3,4,5,2,3,4,2,2};
27        int output_f= obj1.countItem(2, array);
28        assertEquals(6,output_f);
29    }
30 }
```

Problems Javadoc Declaration Coverage ×

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|---------------|-----------|----------------|-----------------|------------------|
| > UnitTesting | 10.2 % | 70 | 615 | 685 |

Equivalence Partitioning:

| Tester Action and Input Data | Expected Outcome |
|--|---------------------------------------|
| Test with v as a non-existent value and an empty array a[] | 0 |
| Test with v as a non-existent value and a non-empty array a[] | 0 |
| Test with v as an existent value and an empty array a[] | 0 |
| Test with v as an existent value and a non-empty array a[] where v exists multiple times | the number of occurrences of v in a[] |
| Test with v as an existent value and a non-empty array a[] where v exists only once | 1 |

Boundary Value Analysis:

| Tester Action and Input Data | Expected Outcome |
|--|---------------------------------------|
| Test with v as a non-existent value and an empty array a[] | 0 |
| Test with v as a non-existent value and a non-empty array a[] | 0 |
| Test with v as an existent value and an array a[] of length 0 | 0 |
| Test with v as an existent value and an array a[] of length 1, where v exists | 1 |
| Test with v as an existent value and an array a[] of length 1, where v does not exist | 0 |
| Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array | the number of occurrences of v in a[] |
| Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array | the number of occurrences of v in a[] |
| Test with v as an existent value and an array a[] of length greater than 1, where v exists in the middle of the array | the number of occurrences of v in a[] |

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 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);
```

```
}
```

Test Case in Eclipse:

The screenshot displays the Eclipse IDE interface with a Java project named 'binarySearch'. The main editor shows the source code for 'binarySearch.java', which includes four test methods: `test1()`, `test2()`, `test3()`, and `test4()`. Each test method uses `@Test` and `assertEquals` to verify the output of the `binarySearch` method. The left sidebar shows the 'JUnit' runner results, indicating that all four tests passed successfully. The bottom panel shows the 'Coverage' report, which indicates that the code is 10.2% covered by the tests.

Package Explorer: triangle.java, prefix.java, unittesting.java, binarySearch.java

JUnit: Finished after 0.089 seconds. Runs: 4/4, Errors: 0, Failures: 0.

binarySearch [Runner: JUnit 4] (0.008 s)

- test1 (0.006 s)
- test2 (0.001 s)
- test3 (0.001 s)
- test4 (0.000 s)

Failure Trace

binarySearch

- test10 : void
- test20 : void
- test30 : void
- test40 : void

Problems | Javadoc | Declaration | Coverage

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|-------------|-----------|----------------|-----------------|------------------|
| UnitTesting | 10.2 % | 70 | 615 | 685 |

Finished after 0.134 seconds

Runs: 4/4Errors: 0Failures: 4

binarySearch [Runner: JUnit 4] (0.049 s)

test1 (0.032 s)

test2 (0.004 s)

test3 (0.003 s)

test4 (0.004 s)

Failure Trace

java.lang.AssertionError: expected:<-1> but was:<3>
at binarySearch.test1(binarySearch.java:12)

```
4
5 public class binarySearch {
6
7     @Test
8     public void test1() {
9         unittesting obj1= new unittesting();
10        int array[]= {1,2,3,4,5};
11        int output_f= obj1.binarySearch(4, array);
12        assertEquals(-1,output_f);
13    }
14    @Test
15    public void test2() {
16        unittesting obj1= new unittesting();
17        int array[]= {1,2,3,8,5};
18        int output_f= obj1.binarySearch(-1, array);
19        assertEquals(3,output_f);
20    }
21    @Test
22    public void test3() {
23        unittesting obj1= new unittesting();
24        int array[]= {1,2,3,4,5};
25        int output_f= obj1.binarySearch(-1, array);
26        assertEquals(4,output_f);
27    }
28    @Test
29    public void test4() {
30        unittesting obj1= new unittesting();
31        int array[]= {'\0'};
32        int output_f= obj1.linearSearch(-1, array);
33        assertEquals(1,output_f);
34    }
35 }
```

Problems Javadoc Declaration Coverage x

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|---------------|-----------|----------------|-----------------|------------------|
| > UnitTesting | 10.2 % | 70 | 615 | 685 |

Equivalence Partitioning:

| Tester Action and Input Data | Expected Outcome |
|------------------------------|------------------|
| v=5, a=[1, 3, 5, 7, 9] | 2 |
| v=1, a=[1, 3, 5, 7, 9] | 0 |
| v=9, a=[1, 3, 5, 7, 9] | 4 |
| v=4, a=[1, 3, 5, 7, 9] | -1 |
| v=11, a=[1, 3, 5, 7, 9] | -1 |

boundary Value Analysis:

| Tester Action and Input Data | Expected Outcome |
|------------------------------|-----------------------------------|
| v=1, a=[1] | 0 |
| v=9, a=[9] | 0 |
| v=5, a=[] | -1 |
| v=5, a=[5, 7, 9] | 0 (smallest element in the array) |
| v=5, a=[1, 3, 5] | 2 (largest element in the array) |

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(SCALED);
```

Test Case in Eclipse:

The screenshot shows the Eclipse IDE with the JUnit runner. The Package Explorer on the left shows the 'triangle' class with four test cases: test1, test2, test3, and test4. The JUnit runner indicates that all tests passed successfully. The main editor shows the source code of the 'triangle' class, which includes four test methods: test1, test2, test3, and test4. The test methods are annotated with @Test and use assertEquals to verify the output of the triangle method.

```
1 *import static org.junit.Assert.*;
4
5 public class triangle {
6
7     @Test
8     public void test1() {
9         unittesting obj1= new unittesting();
10        int output_f= obj1.triangle(4,4,4);
11        assertEquals(0,output_f);
12    }
13    @Test
14    public void test2() {
15        unittesting obj1= new unittesting();
16        int output_f= obj1.triangle(2,1,4);
17        assertEquals(3,output_f);
18    }
19    @Test
20    public void test3() {
21        unittesting obj1= new unittesting();
22        int output_f= obj1.triangle(2,4,4);
23        assertEquals(1,output_f);
24    }
25    @Test
26    public void test4() {
27        unittesting obj1= new unittesting();
28        int output_f= obj1.triangle(3,4,5);
29        assertEquals(2,output_f);
30    }
31 }
32
33
```

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|-------------|-----------|----------------|-----------------|------------------|
| UnitTesting | 10.2 % | 70 | 615 | 685 |

The screenshot shows the Eclipse IDE with the JUnit runner. The Package Explorer on the left shows the 'triangle' class with two test cases: test1 and test2. The JUnit runner indicates that test1 failed and test2 passed. The main editor shows the source code of the 'triangle' class, which includes two test methods: test1 and test2. The test methods are annotated with @Test and use assertEquals to verify the output of the triangle method. The failure trace at the bottom shows the error message: 'java.lang.AssertionError: expected:<0> but was:<3>' at triangle.test1(triangle.java:11).

```
1 *import static org.junit.Assert.*;
4
5 public class triangle {
6
7     @Test
8     public void test1() {
9         unittesting obj1= new unittesting();
10        int output_f= obj1.triangle(4,4,9);
11        assertEquals(0,output_f);
12    }
13    @Test
14    public void test2() {
15        unittesting obj1= new unittesting();
16        int output_f= obj1.triangle(2,1,-1);
17        assertEquals(3,output_f);
18    }
19
20
21 }
22
```

Failure Trace

```
java.lang.AssertionError: expected:<0> but was:<3>
at triangle.test1(triangle.java:11)
```

Equivalence Partitioning:

| Tester Action and Input Data | Expected Outcome |
|-------------------------------|------------------|
| Valid input: a=3, b=3, c=3 | EQUILATERAL |
| Valid input: a=4, b=4, c=5 | ISOSCELES |
| Valid input: a=5, b=4, c=3 | SCALENE |
| Invalid input: a=0, b=0, c=0 | INVALID |
| Invalid input: a=-1, b=2, c=3 | INVALID |
| Valid input: a=1, b=1, c=1 | EQUILATERAL |
| Valid input: a=2, b=2, c=1 | ISOSCELES |
| Valid input: a=3, b=4, c=5 | SCALENE |
| Invalid input: a=0, b=1, c=1 | INVALID |
| Invalid input: a=1, b=0, c=1 | INVALID |
| Invalid input: a=1, b=1, c=0 | INVALID |

Boundary Value Analysis:

| Tester Action and Input Data | Expected Outcome |
|---|------------------|
| Invalid inputs: a = 0, b = 0, c = 0 | INVALID |
| Invalid inputs: a + b = c or b + c = a or c + a = b (a=3, b=4, c=8) | INVALID |
| Equilateral triangles: a = b = c = 1 | EQUILATERAL |
| Equilateral triangles: a = b = c = 100 | EQUILATERAL |
| Isosceles triangles: a = b \neq c = 10 | ISOSCELES |
| Isosceles triangles: a \neq b = c = 10 | ISOSCELES |
| Isosceles triangles: a = c \neq b = 10 | ISOSCELES |

| | |
|---|---------|
| Scalene triangles: $a = b + c - 1$ | SCALENE |
| Scalene triangles: $b = a + c - 1$ | SCALENE |
| Scalene triangles: $c = a + b - 1$ | SCALENE |
| Maximum values: $a, b, c = \text{Integer.MAX_VALUE}$ | INVALID |
| Minimum values: $a, b, c = \text{Integer.MIN_VALUE}$ | INVALID |

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;
}
```


Test Case in Eclipse:

Package ExplorerJUnit ×

Finished after 0.042 seconds

Runs: 3/3Errors: 0Failures: 0

▼ prefix [Runner: JUnit 4] (0.001 s)

test1 (0.000 s)

test2 (0.000 s)

test3 (0.000 s)

Failure Trace

unittesting.java × prefix.java × linearsearch.java

```
8 public void test1() {
9     new unittesting();
10    String a="he";
11    String b="het";
12    boolean output_f= unittesting.prefix(a, b);
13    assertEquals(true,output_f);
14 }
15 @Test
16 public void test2() {
17     new unittesting();
18     String a="e";
19     String b="he";
20
21     boolean output_f= unittesting.prefix(a, b);
22     assertEquals(false,output_f);
23 }
24 @Test
25 public void test3() {
26     new unittesting();
27
28     String a="ht";
29     String b="htee";
30     boolean output_f= unittesting.prefix(a, b);
31     assertEquals(true,output_f);
32 }
```

Outline

▼ pref

• t

• t

• t

Problems Javadoc Declaration Coverage ×

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|---------------|-----------|----------------|-----------------|------------------|
| > UnitTesting | 10.2 % | 70 | 615 | 685 |

Package ExplorerJUnit ×

Finished after 0.035 seconds

Runs: 3/3Errors: 0Failures: 3

▼ prefix [Runner: JUnit 4] (0.009 s)

test1 (0.006 s)

test2 (0.001 s)

test3 (0.001 s)

Failure Trace

java.lang.AssertionError: expected:<true> but was:<false>
at prefix.test1(prefix.java:13)

unittesting.java × prefix.java × linearsearch.java

```
1 *import static org.junit.Assert.*;
4
5 public class prefix {
6
7     @Test
8     public void test1() {
9         new unittesting();
10        String a="ht";
11        String b="het";
12        boolean output_f= unittesting.prefix(a, b);
13        assertEquals(true,output_f);
14    }
15    @Test
16    public void test2() {
17        new unittesting();
18        String a="eh";
19        String b="he";
20
21        boolean output_f= unittesting.prefix(a, b);
22        assertEquals(true,output_f);
23    }
24    @Test
25    public void test3() {
26        new unittesting();
27    }
```

Outline

▼ pr

•

•

•

Problems Javadoc Declaration Coverage ×

| Element | Covera... | Covered Ins... | Missed Instr... | Total Instruc... |
|---------------|-----------|----------------|-----------------|------------------|
| > UnitTesting | 10.2 % | 70 | 615 | 685 |

Equivalence Partitioning:

| Tester Action and Input Data | Expected Outcome |
|--|------------------|
| Valid Inputs: s1 = "hello", s2 = "hello world" | true |
| Valid Inputs: s1 = "a", s2 = "abc" | true |
| Invalid Inputs: s1 = "", s2 = "hello world" | false |
| Invalid Inputs: s1 = "world", s2 = "hello world" | false |

Boundary Value Analysis:

| Tester Action and Input Data | Expected Outcome |
|------------------------------|------------------|
| s1 = "", s2 = "abc" | False |
| s1 = "ab", s2 = "abc" | True |
| s1 = "abc", s2 = "ab" | False |
| s1 = "a", s2 = "ab" | True |
| s1 = "hello", s2 = "hellooo" | True |
| s1 = "abc", s2 = "abc" | True |
| s1 = "a", s2 = "b" | False |
| s1 = "a", s2 = "a" | 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:

- 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
- e. boundary.
- f. For the boundary condition $A = B = C$ case (equilateral triangle), identify test cases to verify the boundary.
- g. For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.
- h. For the non-triangle case, identify test cases to explore the boundary.
- i. For non-positive input, identify test points.

Ans:

Equivalence Class:

| Tester Action and Input Data | Expected Outcome |
|------------------------------|-------------------------------|
| $a = -1, b = 2, c = 3$ | Invalid input |
| $a = 1, b = 1, c = 1$ | Equilateral triangle |
| $a = 2, b = 2, c = 3$ | Isosceles triangle |
| $a = 3, b = 4, c = 5$ | Scalene right-angled triangle |
| $a = 3, b = 5, c = 4$ | Scalene right-angled triangle |
| $a = 5, b = 3, c = 4$ | Scalene right-angled triangle |
| $a = 3, b = 4, c = 6$ | Not a triangle |

Test Case:

Invalid inputs:

$a = 0, b = 0, c = 0, a + b = c, b + c = a, c + a = b$

Invalid inputs:

$$a = -1, b = 1, c = 1, a + b = c$$

Equilateral triangles:

$$a = b = c = 1, a = b = c = 100$$

Isosceles triangles:

$$a = b = 10, c = 5;$$

$$a = c = 10, b = 3;$$

$$b = c = 10,$$

$$a = 6$$

Scalene triangles:

$$a = 4, b = 5, c = 6;$$

$$a = 10, b = 11, c = 13$$

Right angled triangle:

$$a = 3, b = 4, c = 5;$$

$$a = 5, b = 12, c = 13$$

Non-triangle:

$$a = 1, b = 2, c = 3$$

Non-positive input:

$$a = -1, b = -2, c = -3$$

c) Boundary condition $A + B > C$:

$a = \text{Integer.MAX_VALUE}$, $b = \text{Integer.MAX_VALUE}$, $c = 1$

$a = \text{Double.MAX_VALUE}$, $b = \text{Double.MAX_VALUE}$, $c = \text{Double.MAX_VALUE}$

d) Boundary condition $A = C$:

$a = \text{Integer.MAX_VALUE}$,

$b = 2$,

$c = \text{Integer.MAX_VALUE}$

$a = \text{Double.MAX_VALUE}$,

$b = 2.5$,

$c = \text{Double.MAX_VALUE}$

e) Boundary condition $A = B = C$:

$a = \text{Integer.MAX_VALUE}$, $b = \text{Integer.MAX_VALUE}$, $c = \text{Integer.MAX_VALUE}$ a

$= \text{Double.MAX_VALUE}$, $b = \text{Double.MAX_VALUE}$, $c = \text{Double.MAX_VALUE}$ f)

Boundary condition $A^2 + B^2 = C^2$:

$a = \text{Integer.MAX_VALUE}$,

$b = \text{Integer.MAX_VALUE}$,

$c = \text{Integer.MAX_VALUE}$

$a = \text{Double.MAX_VALUE}$,

$b = \text{Double.MAX_VALUE}$,

$c = \text{Math.sqrt}(\text{Math.pow}(\text{Double.MAX_VALUE}, 2) + \text{Math.pow}(\text{Double.MAX_VALUE}, 2))$

g) Non-triangle:

$a = 1, b = 2, c = 4$ $a = 2, b = 4, c = 8$

h) Non-positive input:

$$a = -1, b = -2, c = -3 \quad a = 0, b = 1, c = 2$$

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:

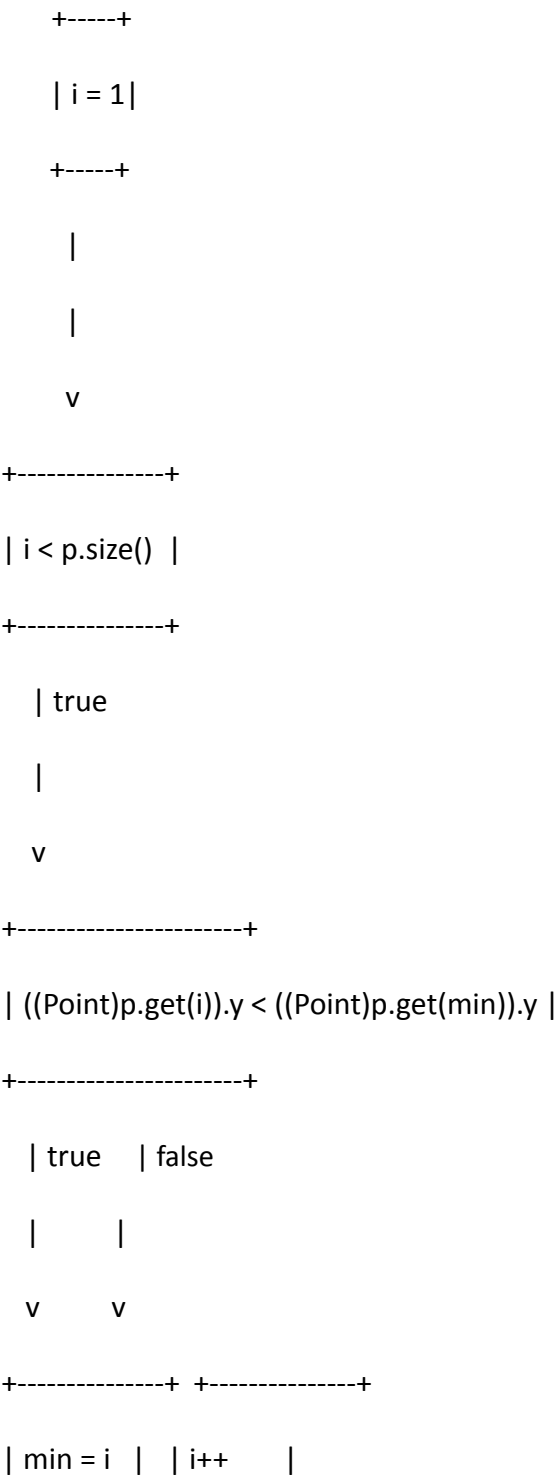
a. Statement Coverage.

b. Branch Coverage.

c. Basic Condition Coverage.

Ans:

Control Flow Graph (CFG):



```
+-----+ +-----+
|      |
|      |
v      v
```

```
+-----+ +-----+
| i < p.size() | | return doStack(p) |
```

```
+-----+ +-----+
| false      |
|            |
v            v
```

```
+-----+
| ((Point)p.get(i)).y == ((Point)p.get(min)).y |
```

```
+-----+
| true   | false
|        |
v        v
```

```
+-----+ +-----+
| ((Point)p.get(i)).x > ((Point)p.get(min)).x |
```

```
+-----+ +-----+
| true   | false
```

```

      |      |
      v      v
+-----+ +-----+
| min = i | | i++   |
+-----+ +-----+
      |      |
      v      v
+-----+ +-----+
| i <     | | return
p.size()  doStack(p) |
+-----+ +-----+
      | false  |
      |      |
      v      v

+-----+

| return doStack(p) |

+-----+

```

Test sets for each coverage criterion:

a. Statement Coverage:

Test 1: p = {new Point(0, 0), new Point(1, 1)}

Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}

b. Branch Coverage:

Test 1: p = {new Point(0, 0), new Point(1, 1)}

Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}

Test 3: $p = \{\text{new Point}(0, 0), \text{new Point}(1, 0), \text{new Point}(1, 1)\}$

c. Basic Condition Coverage:

Test 1: $p = \{\text{new Point}(0, 0), \text{new Point}(1, 1)\}$

Test 2: $p = \{\text{new Point}(0, 0), \text{new Point}(1, 0), \text{new Point}(2, 0)\}$

Test 3: $p = \{\text{new Point}(0, 0), \text{new Point}(1, 0), \text{new Point}(1, 1)\}$

Test 4: $p = \{\text{new Point}(0, 0), \text{new Point}(1, 0), \text{new Point}(0, 1)\}$

Test 5: $p = \{\text{new Point}(0, 0), \text{new Point}(0, 1), \text{new Point}(1, 1)\}$