



Software Engineering

IT314: Lab7

Name : Jatin Ranpariya

StudentID : 202001226

Section : B

Date : 13-04-2023

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 <= year <= 2015}

E8 = {year < 1900}

E9 = {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 Date
E3	34	5	2000	Invalid Date
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'

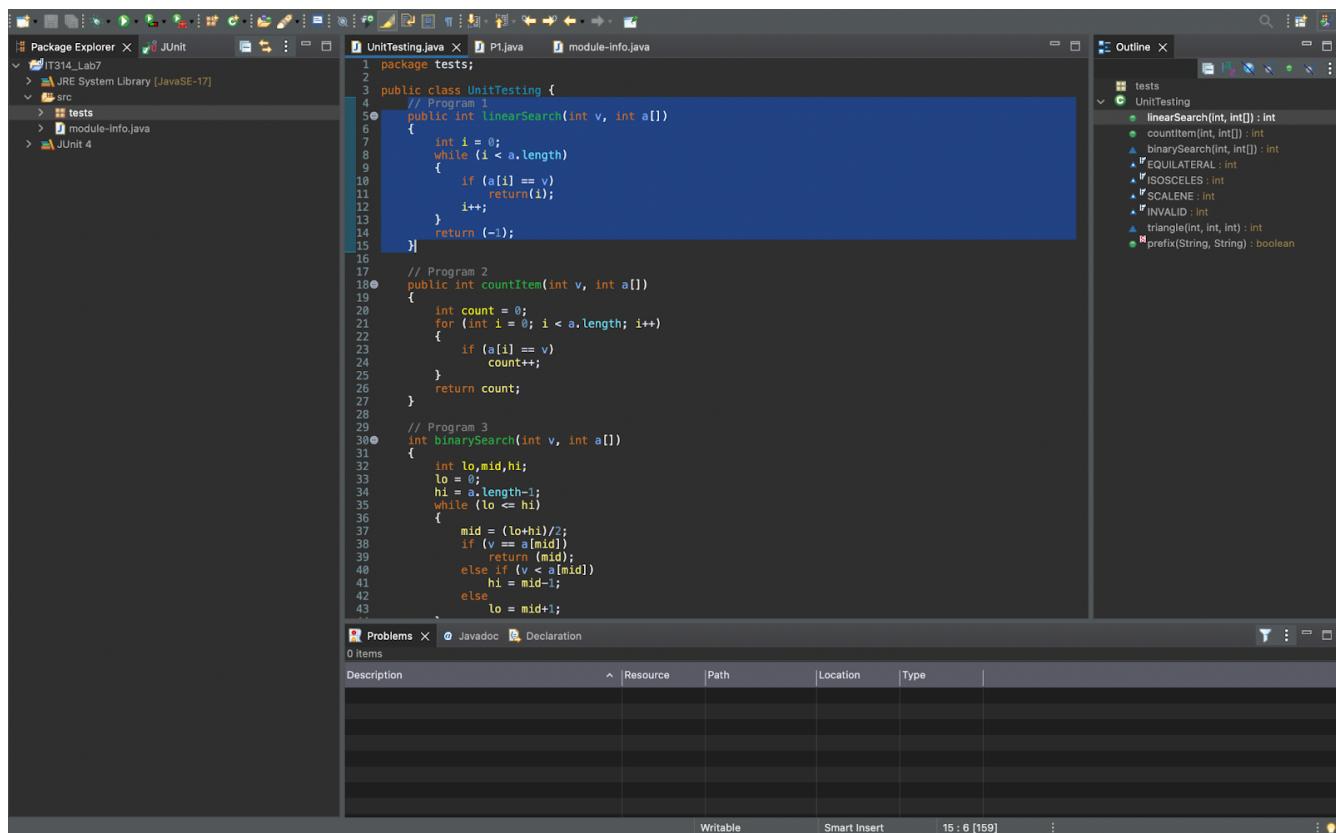
Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array 'a'	-1
'v' is present at 2 nd 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:

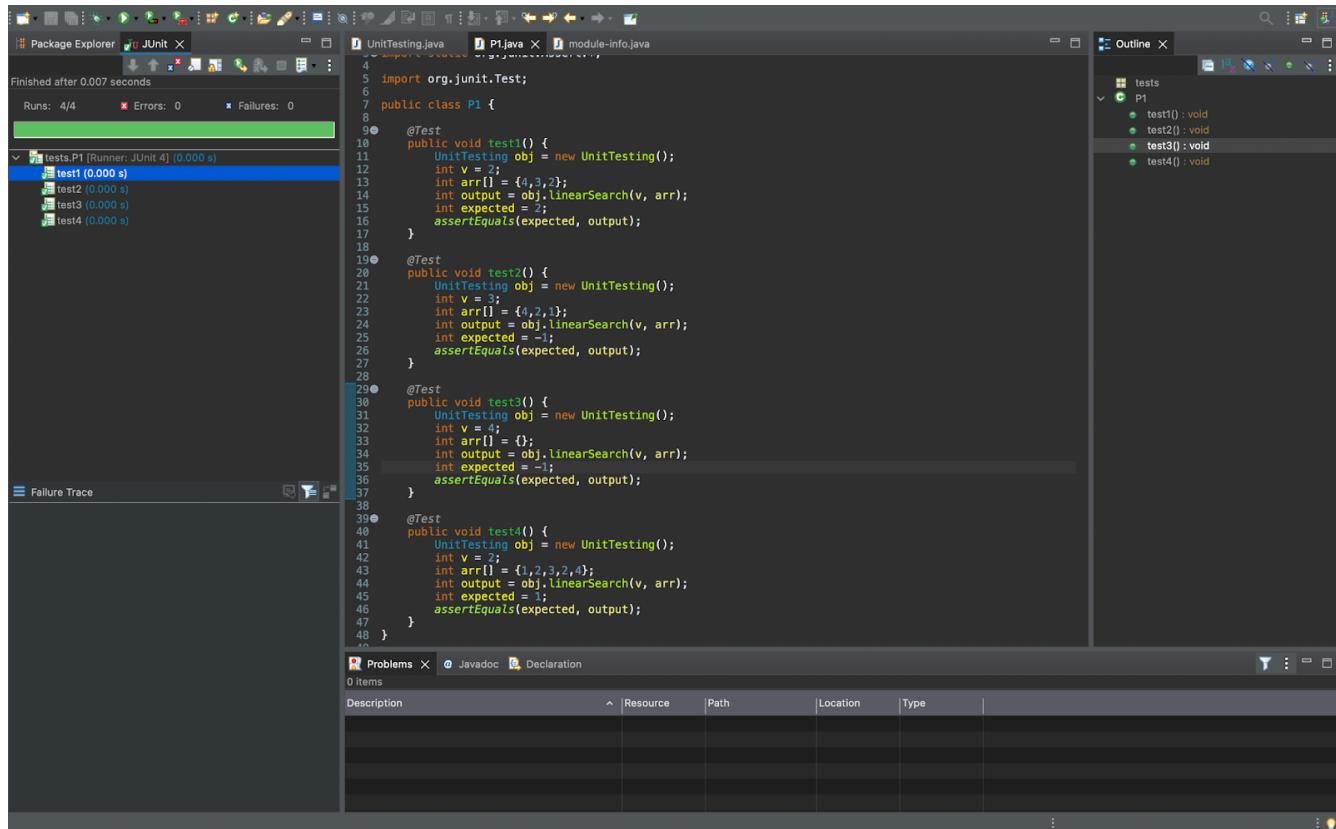


The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with a file named `UnitTesting.java` selected.
- UnitTesting.java Content:**

```

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 }
```
- Outline View:** Shows the test classes and methods defined in the code:
 - `tests`
 - `UnitTesting`
 - `linearSearch(int, int) : int`
 - `countItem(int, int) : int`
 - `binarySearch(int, int) : int`
 - `EQUILATERAL : int`
 - `ISOSCELES : int`
 - `SCALENE : int`
 - `INVALID : int`
 - `triangle(int, int, int) : int`
 - `prefix(String, String) : boolean`
- Problems View:** Shows 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'

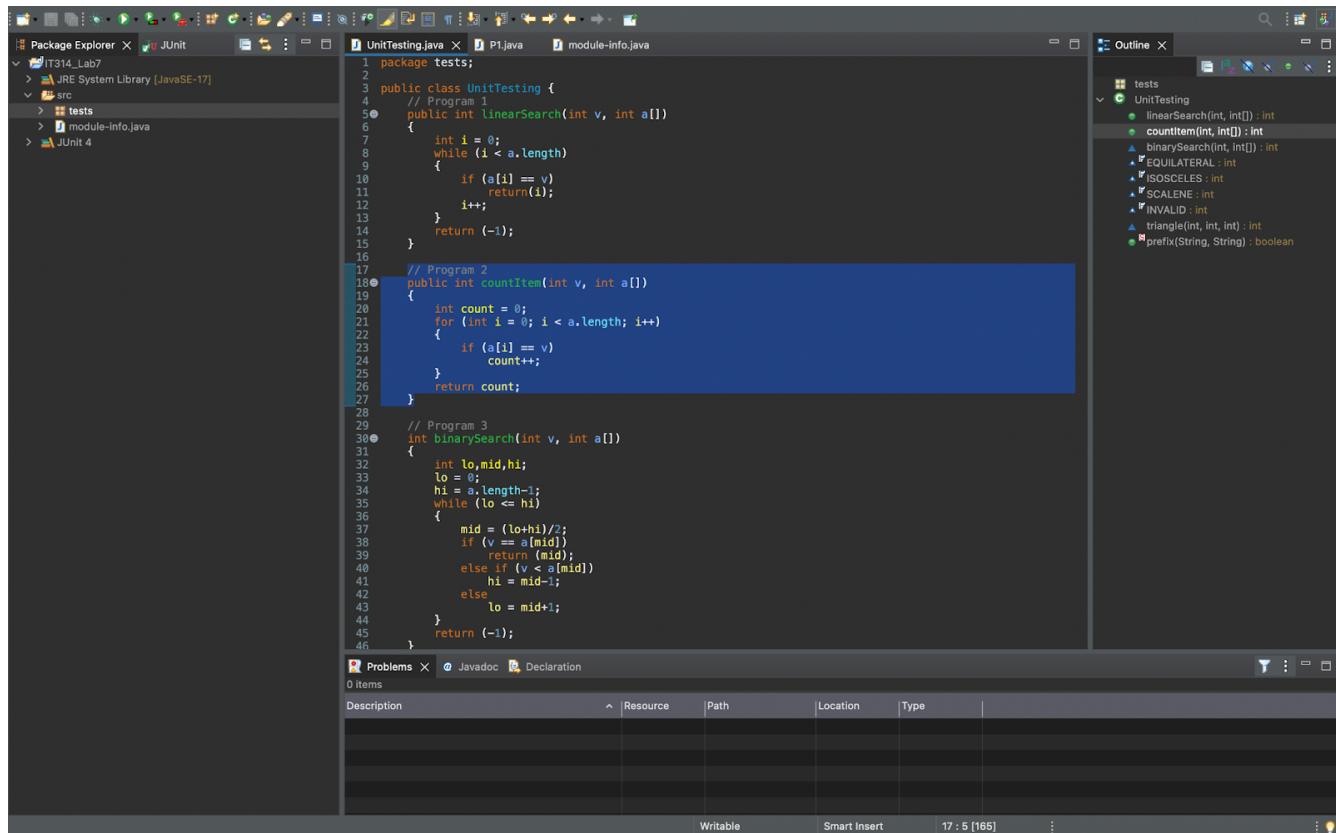
Boundary Value Analysis:

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 details:

- Package Explorer:** Shows the project structure with a package named "tests" containing a class "UnitTesting".
- Editor:** Displays the Java code for the "UnitTesting" class. The code includes three methods: linearSearch, countItem, and binarySearch.
- Outline View:** Shows the class hierarchy and method definitions.
- Problems View:** Shows 0 items.
- Bottom Status Bar:** Shows "17 : 5 [165]".

```

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     }

```

```

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 = 3;
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 = 0;
41        int arr[] = {};
42        int output = obj.countItem(v, arr);
43        int expected = 0;
44        assertEquals(expected, output);
45    }
46}
47
48

```

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);
}

```

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:

- v = 2, a = {0,1,2,3,4}, expected output: 2
- v = -4, a = {1,2,3,4,5}, expected output: -1
- v = 5, a = {2,3,4,5,5,6}, expected output: 3 or 4

JUnit Testing:

```

1 package IT314_Lab7;
2
3 import org.junit.Test;
4 import static org.junit.Assert.*;
5
6 public class UnitTesting {
7     @Test
8     public void testLinearSearch() {
9         int[] a = {1, 2, 3, 4, 5};
10        int v = 3;
11        int count = linearSearch(v, a);
12        assertEquals(2, count);
13    }
14
15    @Test
16    public void testCountItem() {
17        int[] a = {1, 2, 3, 4, 5};
18        int item = 1;
19        int count = countItem(item, a);
20        assertEquals(2, count);
21    }
22
23    @Test
24    public void testBinarySearch() {
25        int[] a = {1, 2, 3, 4, 5};
26        int v = 3;
27        int result = binarySearch(v, a);
28        assertEquals(2, result);
29    }
30
31    @Test
32    public void testTriangle() {
33        int a = 3;
34        int b = 4;
35        int c = 5;
36        int result = triangle(a, b, c);
37        assertEquals(1, result);
38    }
39
40    @Test
41    public void testPrefix() {
42        String s1 = "abc";
43        String s2 = "a";
44        boolean result = prefix(s1, s2);
45        assertTrue(result);
46    }
47
48    // Program 4
49    final int EQUILATERAL = 0;
50    final int ISOSCELES = 1;
51    final int SCALENE = 2;
52    final int INVALID = 3;
53
54    int triangle(int a, int b, int c) {
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    }
63
64    // Program 5
65    public static boolean prefix(String s1, String s2) {
66    }
67
68    @Test
69    public void testTriangle() {
70        int a = 3;
71        int b = 4;
72        int c = 5;
73        int result = triangle(a, b, c);
74        assertEquals(1, result);
75    }
76
77    @Test
78    public void testPrefix() {
79        String s1 = "abc";
80        String s2 = "a";
81        boolean result = prefix(s1, s2);
82        assertTrue(result);
83    }
84
85    @Test
86    public void testLinearSearch() {
87        int[] a = {1, 2, 3, 4, 5};
88        int v = 3;
89        int result = linearSearch(v, a);
90        assertEquals(2, result);
91    }
92
93    @Test
94    public void testCountItem() {
95        int[] a = {1, 2, 3, 4, 5};
96        int item = 1;
97        int result = countItem(item, a);
98        assertEquals(2, result);
99    }
100
101    @Test
102    public void testBinarySearch() {
103        int[] a = {1, 2, 3, 4, 5};
104        int v = 3;
105        int result = binarySearch(v, a);
106        assertEquals(2, result);
107    }
108
109    @Test
110    public void testEQUILATERAL() {
111        int a = 3;
112        int b = 3;
113        int c = 3;
114        int result = triangle(a, b, c);
115        assertEquals(0, result);
116    }
117
118    @Test
119    public void testISOSCELES() {
120        int a = 3;
121        int b = 4;
122        int c = 4;
123        int result = triangle(a, b, c);
124        assertEquals(1, result);
125    }
126
127    @Test
128    public void testSCALENE() {
129        int a = 3;
130        int b = 4;
131        int c = 5;
132        int result = triangle(a, b, c);
133        assertEquals(2, result);
134    }
135
136    @Test
137    public void testINVALID() {
138        int a = 3;
139        int b = 2;
140        int c = 1;
141        int result = triangle(a, b, c);
142        assertEquals(3, result);
143    }
144
145    @Test
146    public void testPrefix() {
147        String s1 = "abc";
148        String s2 = "a";
149        boolean result = prefix(s1, s2);
150        assertTrue(result);
151    }
152
153    @Test
154    public void testLinearSearch() {
155        int[] a = {1, 2, 3, 4, 5};
156        int v = 3;
157        int result = linearSearch(v, a);
158        assertEquals(2, result);
159    }
160
161    @Test
162    public void testCountItem() {
163        int[] a = {1, 2, 3, 4, 5};
164        int item = 1;
165        int result = countItem(item, a);
166        assertEquals(2, result);
167    }
168
169    @Test
170    public void testBinarySearch() {
171        int[] a = {1, 2, 3, 4, 5};
172        int v = 3;
173        int result = binarySearch(v, a);
174        assertEquals(2, result);
175    }
176
177    @Test
178    public void testEQUILATERAL() {
179        int a = 3;
180        int b = 3;
181        int c = 3;
182        int result = triangle(a, b, c);
183        assertEquals(0, result);
184    }
185
186    @Test
187    public void testISOSCELES() {
188        int a = 3;
189        int b = 4;
190        int c = 4;
191        int result = triangle(a, b, c);
192        assertEquals(1, result);
193    }
194
195    @Test
196    public void testSCALENE() {
197        int a = 3;
198        int b = 4;
199        int c = 5;
200        int result = triangle(a, b, c);
201        assertEquals(2, result);
202    }
203
204    @Test
205    public void testINVALID() {
206        int a = 3;
207        int b = 2;
208        int c = 1;
209        int result = triangle(a, b, c);
210        assertEquals(3, result);
211    }
212
213    @Test
214    public void testPrefix() {
215        String s1 = "abc";
216        String s2 = "a";
217        boolean result = prefix(s1, s2);
218        assertTrue(result);
219    }
220
221    @Test
222    public void testLinearSearch() {
223        int[] a = {1, 2, 3, 4, 5};
224        int v = 3;
225        int result = linearSearch(v, a);
226        assertEquals(2, result);
227    }
228
229    @Test
230    public void testCountItem() {
231        int[] a = {1, 2, 3, 4, 5};
232        int item = 1;
233        int result = countItem(item, a);
234        assertEquals(2, result);
235    }
236
237    @Test
238    public void testBinarySearch() {
239        int[] a = {1, 2, 3, 4, 5};
240        int v = 3;
241        int result = binarySearch(v, a);
242        assertEquals(2, result);
243    }
244
245    @Test
246    public void testEQUILATERAL() {
247        int a = 3;
248        int b = 3;
249        int c = 3;
250        int result = triangle(a, b, c);
251        assertEquals(0, result);
252    }
253
254    @Test
255    public void testISOSCELES() {
256        int a = 3;
257        int b = 4;
258        int c = 4;
259        int result = triangle(a, b, c);
260        assertEquals(1, result);
261    }
262
263    @Test
264    public void testSCALENE() {
265        int a = 3;
266        int b = 4;
267        int c = 5;
268        int result = triangle(a, b, c);
269        assertEquals(2, result);
270    }
271
272    @Test
273    public void testINVALID() {
274        int a = 3;
275        int b = 2;
276        int c = 1;
277        int result = triangle(a, b, c);
278        assertEquals(3, result);
279    }
280
281    @Test
282    public void testPrefix() {
283        String s1 = "abc";
284        String s2 = "a";
285        boolean result = prefix(s1, s2);
286        assertTrue(result);
287    }
288
289    @Test
290    public void testLinearSearch() {
291        int[] a = {1, 2, 3, 4, 5};
292        int v = 3;
293        int result = linearSearch(v, a);
294        assertEquals(2, result);
295    }
296
297    @Test
298    public void testCountItem() {
299        int[] a = {1, 2, 3, 4, 5};
300        int item = 1;
301        int result = countItem(item, a);
302        assertEquals(2, result);
303    }
304
305    @Test
306    public void testBinarySearch() {
307        int[] a = {1, 2, 3, 4, 5};
308        int v = 3;
309        int result = binarySearch(v, a);
310        assertEquals(2, result);
311    }
312
313    @Test
314    public void testEQUILATERAL() {
315        int a = 3;
316        int b = 3;
317        int c = 3;
318        int result = triangle(a, b, c);
319        assertEquals(0, result);
320    }
321
322    @Test
323    public void testISOSCELES() {
324        int a = 3;
325        int b = 4;
326        int c = 4;
327        int result = triangle(a, b, c);
328        assertEquals(1, result);
329    }
330
331    @Test
332    public void testSCALENE() {
333        int a = 3;
334        int b = 4;
335        int c = 5;
336        int result = triangle(a, b, c);
337        assertEquals(2, result);
338    }
339
340    @Test
341    public void testINVALID() {
342        int a = 3;
343        int b = 2;
344        int c = 1;
345        int result = triangle(a, b, c);
346        assertEquals(3, result);
347    }
348
349    @Test
350    public void testPrefix() {
351        String s1 = "abc";
352        String s2 = "a";
353        boolean result = prefix(s1, s2);
354        assertTrue(result);
355    }
356
357    @Test
358    public void testLinearSearch() {
359        int[] a = {1, 2, 3, 4, 5};
360        int v = 3;
361        int result = linearSearch(v, a);
362        assertEquals(2, result);
363    }
364
365    @Test
366    public void testCountItem() {
367        int[] a = {1, 2, 3, 4, 5};
368        int item = 1;
369        int result = countItem(item, a);
370        assertEquals(2, result);
371    }
372
373    @Test
374    public void testBinarySearch() {
375        int[] a = {1, 2, 3, 4, 5};
376        int v = 3;
377        int result = binarySearch(v, a);
378        assertEquals(2, result);
379    }
380
381    @Test
382    public void testEQUILATERAL() {
383        int a = 3;
384        int b = 3;
385        int c = 3;
386        int result = triangle(a, b, c);
387        assertEquals(0, result);
388    }
389
390    @Test
391    public void testISOSCELES() {
392        int a = 3;
393        int b = 4;
394        int c = 4;
395        int result = triangle(a, b, c);
396        assertEquals(1, result);
397    }
398
399    @Test
400    public void testSCALENE() {
401        int a = 3;
402        int b = 4;
403        int c = 5;
404        int result = triangle(a, b, c);
405        assertEquals(2, result);
406    }
407
408    @Test
409    public void testINVALID() {
410        int a = 3;
411        int b = 2;
412        int c = 1;
413        int result = triangle(a, b, c);
414        assertEquals(3, result);
415    }
416
417    @Test
418    public void testPrefix() {
419        String s1 = "abc";
420        String s2 = "a";
421        boolean result = prefix(s1, s2);
422        assertTrue(result);
423    }
424
425    @Test
426    public void testLinearSearch() {
427        int[] a = {1, 2, 3, 4, 5};
428        int v = 3;
429        int result = linearSearch(v, a);
430        assertEquals(2, result);
431    }
432
433    @Test
434    public void testCountItem() {
435        int[] a = {1, 2, 3, 4, 5};
436        int item = 1;
437        int result = countItem(item, a);
438        assertEquals(2, result);
439    }
440
441    @Test
442    public void testBinarySearch() {
443        int[] a = {1, 2, 3, 4, 5};
444        int v = 3;
445        int result = binarySearch(v, a);
446        assertEquals(2, result);
447    }
448
449    @Test
450    public void testEQUILATERAL() {
451        int a = 3;
452        int b = 3;
453        int c = 3;
454        int result = triangle(a, b, c);
455        assertEquals(0, result);
456    }
457
458    @Test
459    public void testISOSCELES() {
460        int a = 3;
461        int b = 4;
462        int c = 4;
463        int result = triangle(a, b, c);
464        assertEquals(1, result);
465    }
466
467    @Test
468    public void testSCALENE() {
469        int a = 3;
470        int b = 4;
471        int c = 5;
472        int result = triangle(a, b, c);
473        assertEquals(2, result);
474    }
475
476    @Test
477    public void testINVALID() {
478        int a = 3;
479        int b = 2;
480        int c = 1;
481        int result = triangle(a, b, c);
482        assertEquals(3, result);
483    }
484
485    @Test
486    public void testPrefix() {
487        String s1 = "abc";
488        String s2 = "a";
489        boolean result = prefix(s1, s2);
490        assertTrue(result);
491    }
492
493    @Test
494    public void testLinearSearch() {
495        int[] a = {1, 2, 3, 4, 5};
496        int v = 3;
497        int result = linearSearch(v, a);
498        assertEquals(2, result);
499    }
500
501    @Test
502    public void testCountItem() {
503        int[] a = {1, 2, 3, 4, 5};
504        int item = 1;
505        int result = countItem(item, a);
506        assertEquals(2, result);
507    }
508
509    @Test
510    public void testBinarySearch() {
511        int[] a = {1, 2, 3, 4, 5};
512        int v = 3;
513        int result = binarySearch(v, a);
514        assertEquals(2, result);
515    }
516
517    @Test
518    public void testEQUILATERAL() {
519        int a = 3;
520        int b = 3;
521        int c = 3;
522        int result = triangle(a, b, c);
523        assertEquals(0, result);
524    }
525
526    @Test
527    public void testISOSCELES() {
528        int a = 3;
529        int b = 4;
530        int c = 4;
531        int result = triangle(a, b, c);
532        assertEquals(1, result);
533    }
534
535    @Test
536    public void testSCALENE() {
537        int a = 3;
538        int b = 4;
539        int c = 5;
540        int result = triangle(a, b, c);
541        assertEquals(2, result);
542    }
543
544    @Test
545    public void testINVALID() {
546        int a = 3;
547        int b = 2;
548        int c = 1;
549        int result = triangle(a, b, c);
550        assertEquals(3, result);
551    }
552
553    @Test
554    public void testPrefix() {
555        String s1 = "abc";
556        String s2 = "a";
557        boolean result = prefix(s1, s2);
558        assertTrue(result);
559    }
560
561    @Test
562    public void testLinearSearch() {
563        int[] a = {1, 2, 3, 4, 5};
564        int v = 3;
565        int result = linearSearch(v, a);
566        assertEquals(2, result);
567    }
568
569    @Test
570    public void testCountItem() {
571        int[] a = {1, 2, 3, 4, 5};
572        int item = 1;
573        int result = countItem(item, a);
574        assertEquals(2, result);
575    }
576
577    @Test
578    public void testBinarySearch() {
579        int[] a = {1, 2, 3, 4, 5};
580        int v = 3;
581        int result = binarySearch(v, a);
582        assertEquals(2, result);
583    }
584
585    @Test
586    public void testEQUILATERAL() {
587        int a = 3;
588        int b = 3;
589        int c = 3;
590        int result = triangle(a, b, c);
591        assertEquals(0, result);
592    }
593
594    @Test
595    public void testISOSCELES() {
596        int a = 3;
597        int b = 4;
598        int c = 4;
599        int result = triangle(a, b, c);
600        assertEquals(1, result);
601    }
602
603    @Test
604    public void testSCALENE() {
605        int a = 3;
606        int b = 4;
607        int c = 5;
608        int result = triangle(a, b, c);
609        assertEquals(2, result);
610    }
611
612    @Test
613    public void testINVALID() {
614        int a = 3;
615        int b = 2;
616        int c = 1;
617        int result = triangle(a, b, c);
618        assertEquals(3, result);
619    }
620
621    @Test
622    public void testPrefix() {
623        String s1 = "abc";
624        String s2 = "a";
625        boolean result = prefix(s1, s2);
626        assertTrue(result);
627    }
628
629    @Test
630    public void testLinearSearch() {
631        int[] a = {1, 2, 3, 4, 5};
632        int v = 3;
633        int result = linearSearch(v, a);
634        assertEquals(2, result);
635    }
636
637    @Test
638    public void testCountItem() {
639        int[] a = {1, 2, 3, 4, 5};
640        int item = 1;
641        int result = countItem(item, a);
642        assertEquals(2, result);
643    }
644
645    @Test
646    public void testBinarySearch() {
647        int[] a = {1, 2, 3, 4, 5};
648        int v = 3;
649        int result = binarySearch(v, a);
650        assertEquals(2, result);
651    }
652
653    @Test
654    public void testEQUILATERAL() {
655        int a = 3;
656        int b = 3;
657        int c = 3;
658        int result = triangle(a, b, c);
659        assertEquals(0, result);
660    }
661
662    @Test
663    public void testISOSCELES() {
664        int a = 3;
665        int b = 4;
666        int c = 4;
667        int result = triangle(a, b, c);
668        assertEquals(1, result);
669    }
670
671    @Test
672    public void testSCALENE() {
673        int a = 3;
674        int b = 4;
675        int c = 5;
676        int result = triangle(a, b, c);
677        assertEquals(2, result);
678    }
679
680    @Test
681    public void testINVALID() {
682        int a = 3;
683        int b = 2;
684        int c = 1;
685        int result = triangle(a, b, c);
686        assertEquals(3, result);
687    }
688
689    @Test
690    public void testPrefix() {
691        String s1 = "abc";
692        String s2 = "a";
693        boolean result = prefix(s1, s2);
694        assertTrue(result);
695    }
696
697    @Test
698    public void testLinearSearch() {
699        int[] a = {1, 2, 3, 4, 5};
700        int v = 3;
701        int result = linearSearch(v, a);
702        assertEquals(2, result);
703    }
704
705    @Test
706    public void testCountItem() {
707        int[] a = {1, 2, 3, 4, 5};
708        int item = 1;
709        int result = countItem(item, a);
710        assertEquals(2, result);
711    }
712
713    @Test
714    public void testBinarySearch() {
715        int[] a = {1, 2, 3, 4, 5};
716        int v = 3;
717        int result = binarySearch(v, a);
718        assertEquals(2, result);
719    }
720
721    @Test
722    public void testEQUILATERAL() {
723        int a = 3;
724        int b = 3;
725        int c = 3;
726        int result = triangle(a, b, c);
727        assertEquals(0, result);
728    }
729
730    @Test
731    public void testISOSCELES() {
732        int a = 3;
733        int b = 4;
734        int c = 4;
735        int result = triangle(a, b, c);
736        assertEquals(1, result);
737    }
738
739    @Test
740    public void testSCALENE() {
741        int a = 3;
742        int b = 4;
743        int c = 5;
744        int result = triangle(a, b, c);
745        assertEquals(2, result);
746    }
747
748    @Test
749    public void testINVALID() {
750        int a = 3;
751        int b = 2;
752        int c = 1;
753        int result = triangle(a, b, c);
754        assertEquals(3, result);
755    }
756
757    @Test
758    public void testPrefix() {
759        String s1 = "abc";
760        String s2 = "a";
761        boolean result = prefix(s1, s2);
762        assertTrue(result);
763    }
764
765    @Test
766    public void testLinearSearch() {
767        int[] a = {1, 2, 3, 4, 5};
768        int v = 3;
769        int result = linearSearch(v, a);
770        assertEquals(2, result);
771    }
772
773    @Test
774    public void testCountItem() {
775        int[] a = {1, 2, 3, 4, 5};
776        int item = 1;
777        int result = countItem(item, a);
778        assertEquals(2, result);
779    }
780
781    @Test
782    public void testBinarySearch() {
783        int[] a = {1, 2, 3, 4, 5};
784        int v = 3;
785        int result = binarySearch(v, a);
786        assertEquals(2, result);
787    }
788
789    @Test
790    public void testEQUILATERAL() {
791        int a = 3;
792        int b = 3;
793        int c = 3;
794        int result = triangle(a, b, c);
795        assertEquals(0, result);
796    }
797
798    @Test
799    public void testISOSCELES() {
800        int a = 3;
801        int b = 4;
802        int c = 4;
803        int result = triangle(a, b, c);
804        assertEquals(1, result);
805    }
806
807    @Test
808    public void testSCALENE() {
809        int a = 3;
810        int b = 4;
811        int c = 5;
812        int result = triangle(a, b, c);
813        assertEquals(2, result);
814    }
815
816    @Test
817    public void testINVALID() {
818        int a = 3;
819        int b = 2;
820        int c = 1;
821        int result = triangle(a, b, c);
822        assertEquals(3, result);
823    }
824
825    @Test
826    public void testPrefix() {
827        String s1 = "abc";
828        String s2 = "a";
829        boolean result = prefix(s1, s2);
830        assertTrue(result);
831    }
832
833    @Test
834    public void testLinearSearch() {
835        int[] a = {1, 2, 3, 4, 5};
836        int v = 3;
837        int result = linearSearch(v, a);
838        assertEquals(2, result);
839    }
840
841    @Test
842    public void testCountItem() {
843        int[] a = {1, 2, 3, 4, 5};
844        int item = 1;
845        int result = countItem(item, a);
846        assertEquals(2, result);
847    }
848
849    @Test
850    public void testBinarySearch() {
851        int[] a = {1, 2, 3, 4, 5};
852        int v = 3;
853        int result = binarySearch(v, a);
854        assertEquals(2, result);
855    }
856
857    @Test
858    public void testEQUILATERAL() {
859        int a = 3;
860        int b = 3;
861        int c = 3;
862        int result = triangle(a, b, c);
863        assertEquals(0, result);
864    }
865
866    @Test
867    public void testISOSCELES() {
868        int a = 3;
869        int b = 4;
870        int c = 4;
871        int result = triangle(a, b, c);
872        assertEquals(1, result);
873    }
874
875    @Test
876    public void testSCALENE() {
877        int a = 3;
878        int b = 4;
879        int c = 5;
880        int result = triangle(a, b, c);
881        assertEquals(2, result);
882    }
883
884    @Test
885    public void testINVALID() {
886        int a = 3;
887        int b = 2;
888        int c = 1;
889        int result = triangle(a, b, c);
890        assertEquals(3, result);
891    }
892
893    @Test
894    public void testPrefix() {
895        String s1 = "abc";
896        String s2 = "a";
897        boolean result = prefix(s1, s2);
898        assertTrue(result);
899    }
900
901    @Test
902    public void testLinearSearch() {
903        int[] a = {1, 2, 3, 4, 5};
904        int v = 3;
905        int result = linearSearch(v, a);
906        assertEquals(2, result);
907    }
908
909    @Test
910    public void testCountItem() {
911        int[] a = {1, 2, 3, 4, 5};
912        int item = 1;
913        int result = countItem(item, a);
914        assertEquals(2, result);
915    }
916
917    @Test
918    public void testBinarySearch() {
919        int[] a = {1, 2, 3, 4, 5};
920        int v = 3;
921        int result = binarySearch(v, a);
922        assertEquals(2, result);
923    }
924
925    @Test
926    public void testEQUILATERAL() {
927        int a = 3;
928        int b = 3;
929        int c = 3;
930        int result = triangle(a, b, c);
931        assertEquals(0, result);
932    }
933
934    @Test
935    public void testISOSCELES() {
936        int a = 3;
937        int b = 4;
938        int c = 4;
939        int result = triangle(a, b, c);
940        assertEquals(1, result);
941    }
942
943    @Test
944    public void testSCALENE() {
945        int a = 3;
946        int b = 4;
947        int c = 5;
948        int result = triangle(a, b, c);
949        assertEquals(2, result);
950    }
951
952    @Test
953    public void testINVALID() {
954        int a = 3;
955        int b = 2;
956        int c = 1;
957        int result = triangle(a, b, c);
958        assertEquals(3, result);
959    }
960
961    @Test
962    public void testPrefix() {
963        String s1 = "abc";
964        String s2 = "a";
965        boolean result = prefix(s1, s2);
966        assertTrue(result);
967    }
968
969    @Test
970    public void testLinearSearch() {
971        int[] a = {1, 2, 3, 4, 5};
972        int v = 3;
973        int result = linearSearch(v, a);
974        assertEquals(2, result);
975    }
976
977    @Test
978    public void testCountItem() {
979        int[] a = {1, 2, 3, 4, 5};
980        int item = 1;
981        int result = countItem(item, a);
982        assertEquals(2, result);
983    }
984
985    @Test
986    public void testBinarySearch() {
987        int[] a = {1, 2, 3, 4, 5};
988        int v = 3;
989        int result = binarySearch(v, a);
990        assertEquals(2, result);
991    }
992
993    @Test
994    public void testEQUILATERAL() {
995        int a = 3;
996        int b = 3;
997        int c = 3;
998        int result = triangle(a, b, c);
999        assertEquals(0, result);
1000
1001    @Test
1002    public void testISOSCELES() {
1003        int a = 3;
1004        int b = 4;
1005        int c = 4;
1006        int result = triangle(a, b, c);
1007        assertEquals(1, result);
1008
1009    @Test
1010    public void testSCALENE() {
1011        int a = 3;
1012        int b = 4;
1013        int c = 5;
1014        int result = triangle(a, b, c);
1015        assertEquals(2, result);
1016
1017    @Test
1018    public void testINVALID() {
1019        int a = 3;
1020        int b = 2;
1021        int c = 1;
1022        int result = triangle(a, b, c);
1023        assertEquals(3, result);
1024
1025    @Test
1026    public void testPrefix() {
1027        String s1 = "abc";
1028        String s2 = "a";
1029        boolean result = prefix(s1, s2);
1030        assertTrue(result);
1031    }
1032
1033    @Test
1034    public void testLinearSearch() {
1035        int[] a = {1, 2, 3, 4, 5};
1036        int v = 3;
1037        int result = linearSearch(v, a);
1038        assertEquals(2, result);
1039    }
1040
1041    @Test
1042    public void testCountItem() {
1043        int[] a = {1, 2, 3, 4, 5};
1044        int item = 1;
1045        int result = countItem(item, a);
1046        assertEquals(2, result);
1047    }
1048
1049    @Test
1050    public void testBinarySearch() {
1051        int[] a = {1, 2, 3, 4, 5};
1052        int v = 3;
1053        int result = binarySearch(v, a);
1054        assertEquals(2, result);
1055    }
1056
1057    @Test
1058    public void testEQUILATERAL() {
1059        int a = 3;
1060        int b = 3;
1061        int c = 3;
1062        int result = triangle(a, b, c);
1063        assertEquals(0, result);
1064
1065    @Test
1066    public void testISOSCELES() {
1067        int a =
```

The screenshot shows the Eclipse IDE interface. The top menu bar includes 'File', 'Run As', 'Project', 'Build', 'Import', 'Export', 'Help', and 'Exit'. The left sidebar has 'Package Explorer' and 'JUnit X' sections. The main workspace contains four tabs: 'UnitTesting.java', 'P1.java', 'P2.java', and 'P3.java'. The 'UnitTesting.java' tab shows a class with methods for binary search. The 'P3.java' tab shows a class with three test methods: 'test1()', 'test2()', and 'test3()'. The 'tests.P3 [Runner: JUnit 4] (0.000 s)' section in the Package Explorer shows three green icons for 'test1', 'test2', and 'test3', indicating they all passed. The bottom right corner shows the 'Outline' view with 'tests' expanded to show 'P3', 'test1()', 'test2()', and 'test3()'.

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 }

48 // Program 4
49 final int EQUALILATERAL = 0;
50 final int ISOSCELES = 1;
51 final int SCALENE = 2;
52 final int INVALID = 3;
53
54 int triangle(int a, int b, int c)
55 {
56     if (a >= b+c || b >= a+c || c >= a+b)
57         return (INVALID);
58     if (a == b && b == c)
59         return (EQUALILATERAL);
60     if (a == b || a == c || b == c)
61         return (ISOSCELES);
62     return (SCALENE);
63 }

64 // Program 5
65 public static boolean prefix(String s1, String s2)
66 {
67     if (s1.length() > s2.length())
68     {
69         return false;
70     }
71     for (int i = 0; i < s1.length(); i++)
72     {
73         if (s1.charAt(i) != s2.charAt(i))
74         {
75             return false;
76         }
77     }
78     return true;
79 }
    
```
- Outline View:** Shows the class structure with methods linearSearch, countitem, binarySearch, EQUALILATERAL, ISOSCELES, SCALENE, and INVALID.
- 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:**

```

24
25 @Test
26 public void test2() {
27     UnitTesting obj = new UnitTesting();
28     int a = 1, b = 2, c = 3;
29     int output = obj.triangle(a, b, c);
30     int expected = INVALID;
31     assertEquals(expected, output);
32 }
33 @Test
34 public void test3() {
35     UnitTesting obj = new UnitTesting();
36     int a = 1, b = 1, c = 3;
37     int output = obj.triangle(a, b, c);
38     int expected = INVALID;
39     assertEquals(expected, output);
40 }
41 @Test
42 public void test4() {
43     UnitTesting obj = new UnitTesting();
44     int a = 3, b = 4, c = 5;
45     int output = obj.triangle(a, b, c);
46     int expected = SCALENE;
47     assertEquals(expected, output);
48 }
49
50 @Test
51 public void test5() {
52     UnitTesting obj = new UnitTesting();
53     int a = 5, b = 5, c = 9;
54     int output = obj.triangle(a, b, c);
55     int expected = ISOSCELES;
56     assertEquals(expected, output);
57 }
58
59 @Test
60 public void test6() {
61     UnitTesting obj = new UnitTesting();
62     int a = 1, b = 5, c = 10;
63     int output = obj.triangle(a, b, c);
64     int expected = INVALID;
65     assertEquals(expected, output);
66 }
    
```
- Outline View:** Shows the class structure with methods test1, test2, test3, test4, test5, test6, EQUALILATERAL, ISOSCELES, SCALENE, and INVALID.
- Problems View:** Shows 0 items.

Execution results summary:

- Runs: 6/6
- Errors: 0
- Failures: 0

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 <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 not a prefix of <code>s2</code>	False
Non-empty <code>s1</code> is longer than <code>s2</code>	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. $s1 = \text{"one"}, s2 = \text{"two"},$ expected output: False
5. $s1 = \text{""}, s2 = \text{"pdf"},$ expected output: True

JUnit Testing:

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

- Package Explorer:** Shows the project structure with files P1.java, P2.java, P3.java, P4.java, P5.java, and UnitTesting.java.
- UnitTesting.java:** The main source file containing code for various programs (Program 1-5) and a prefix method. The prefix method is highlighted in blue.
- Outline:** Shows the class structure with methods like linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, INVALID, and prefix.
- Problems:** Shows 0 items.
- Test Results:** Shows 5/5 runs completed successfully with 0 errors and 0 failures.
- Outline:** Shows the class structure with methods like linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, INVALID, and prefix.

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

- Package Explorer:** Shows the project structure with files P1.java, P2.java, P3.java, P4.java, P5.java, and UnitTesting.java.
- UnitTesting.java:** The main source file containing code for various programs (Program 1-5) and a prefix method. The prefix method is highlighted in blue.
- Outline:** Shows the class structure with methods like linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, INVALID, and prefix.
- Problems:** Shows 0 items.
- Test Results:** Shows 5/5 runs completed successfully with 0 errors and 0 failures.
- Outline:** Shows the class structure with methods like linearSearch, countItem, binarySearch, EQUILATERAL, ISOSCELES, SCALENE, INVALID, and prefix.

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:

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

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

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

2. EQUILATERAL case:

$$\rightarrow E1: a = b, b = c, c = a$$

3. ISOSCELES case:

$$\rightarrow E1: a = b, a \neq c$$

$$\rightarrow E1: a = c, a \neq b$$

$$\rightarrow E1: b = c, b \neq a$$

4. SCALENE case:

$$\rightarrow E1: a \neq b, b \neq c, c \neq a$$

5. RIGHT-ANGLED TRIANGLE case:

$$\rightarrow E1: a^2 + b^2 = c^2$$

$$\rightarrow E1: b^2 + c^2 = a^2$$

$$\rightarrow E1: a^2 + c^2 = b^2$$

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 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;

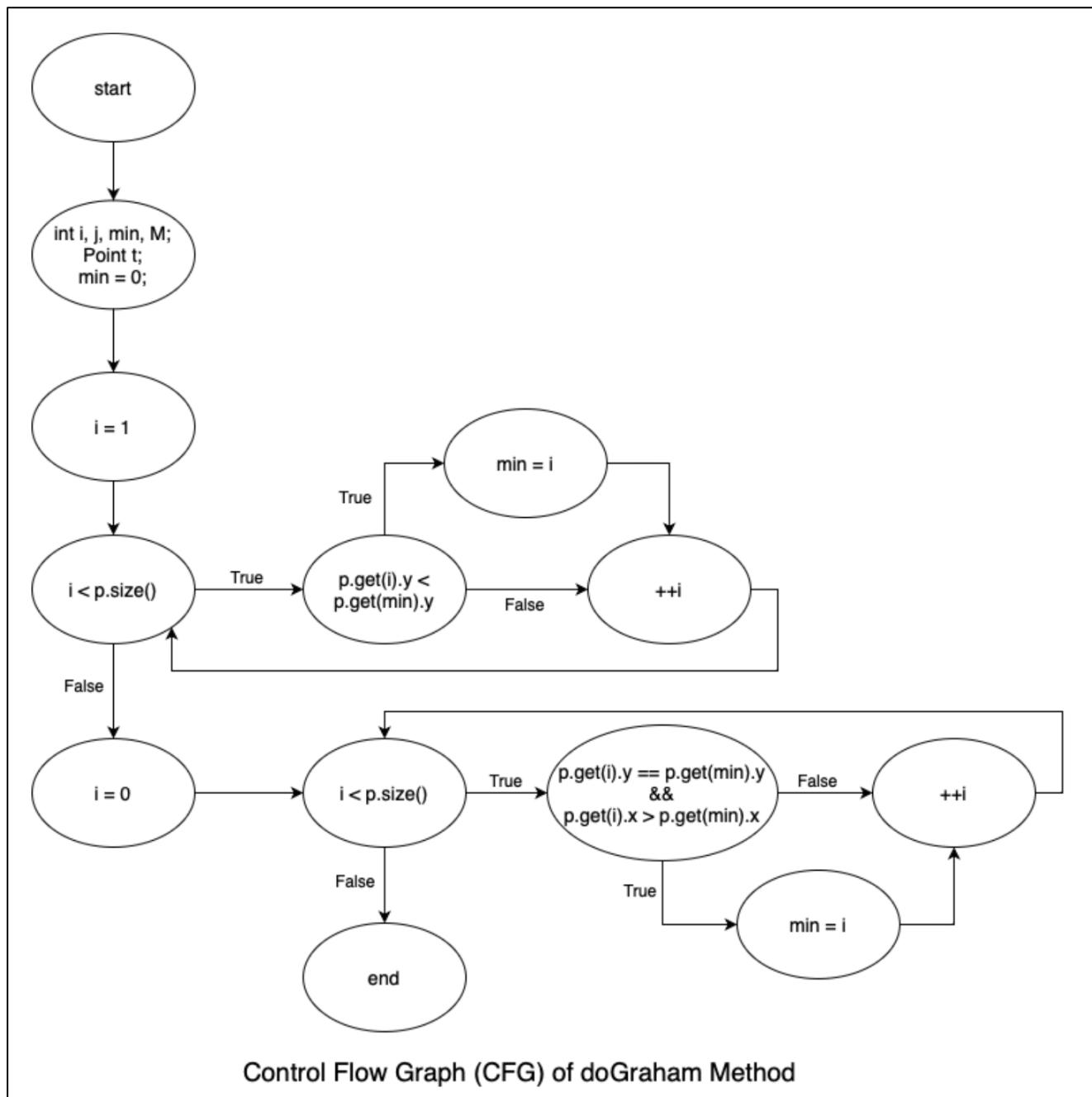
    Point t;
    min = 0;

    // 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.
- 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.