Dhirubhai Ambani Institute of Information and Communication Technology

DA-IICT

IT314 - Software Engineering

Lab-07

Name : Harsh Acharya

ID : 202001131

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 <= month <= 12,

1 <= day <= 31,

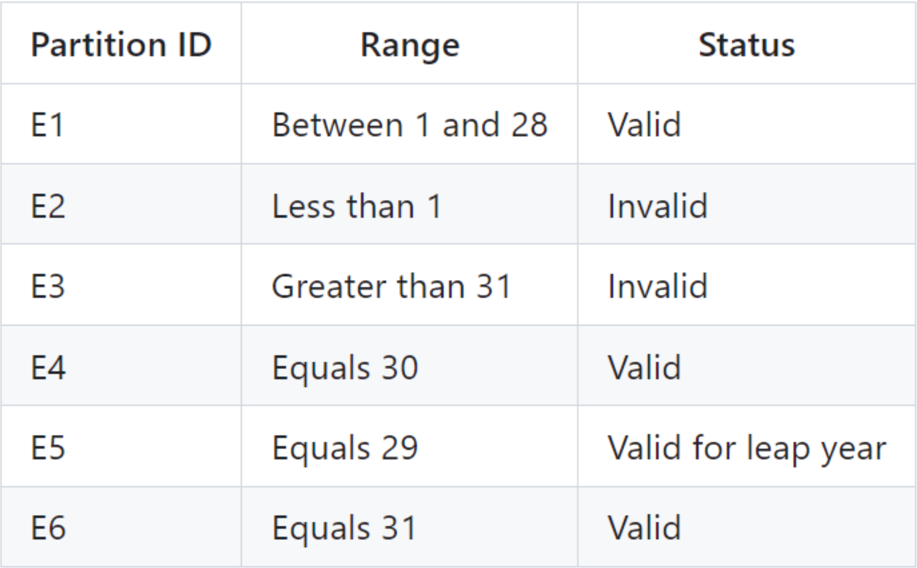
1900 <= year <= 2015.

The possible output dates would be the previous date or invalid date. Design theequivalence class test cases?

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

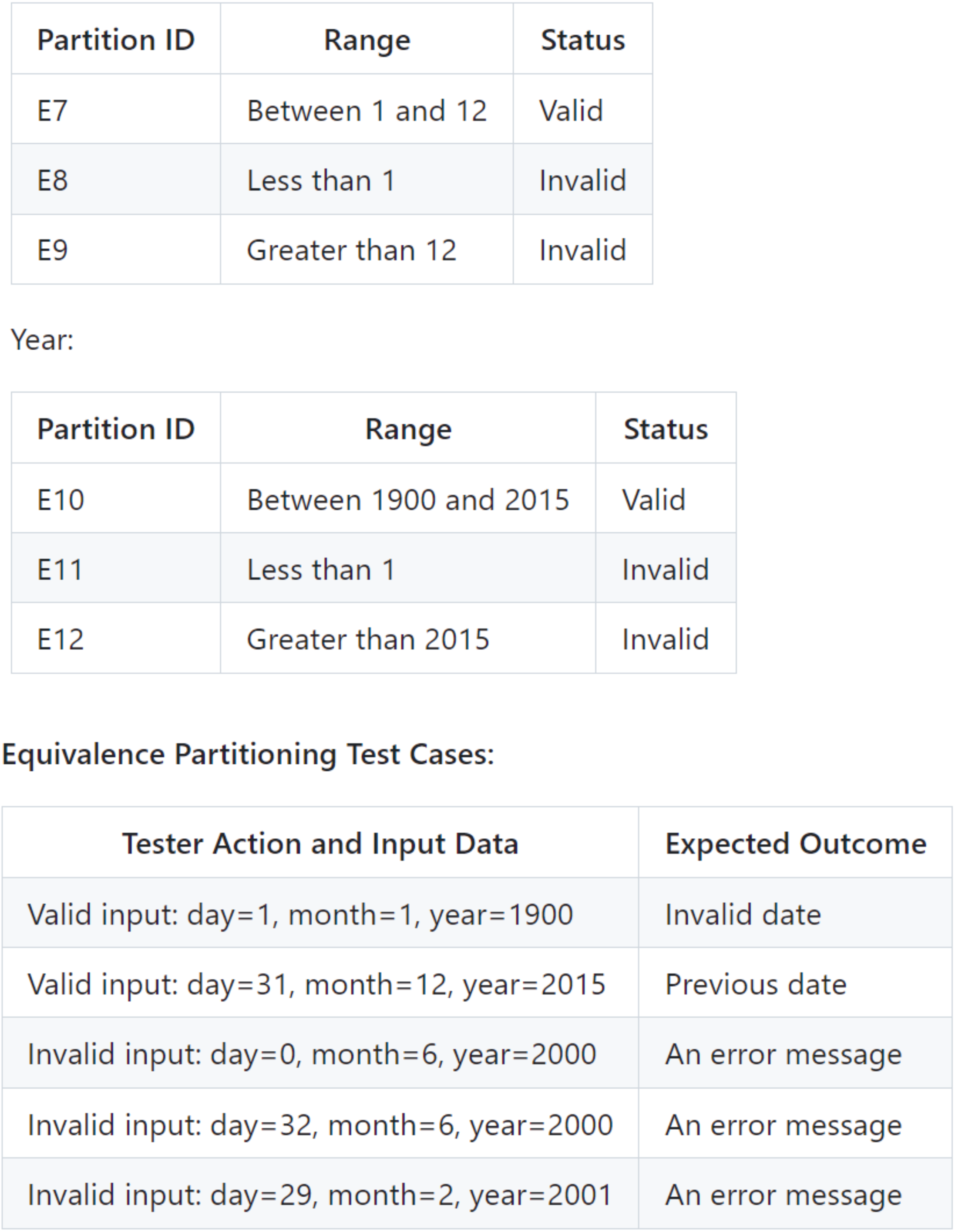
Equivalence Class Partitions :

Day:



Partition ID	Range	Status
E1	Between 1 and 28	Valid
E2	Less than 1	Invalid
E3	Greater than 31	Invalid
E4	Equals 30	Valid
E5	Equals 29	Valid for leap year
E6	Equals 31	Valid

Month:



Partition ID	Range	Status
E7	Between 1 and 12	Valid
E8	Less than 1	Invalid
E9	Greater than 12	Invalid

Year:

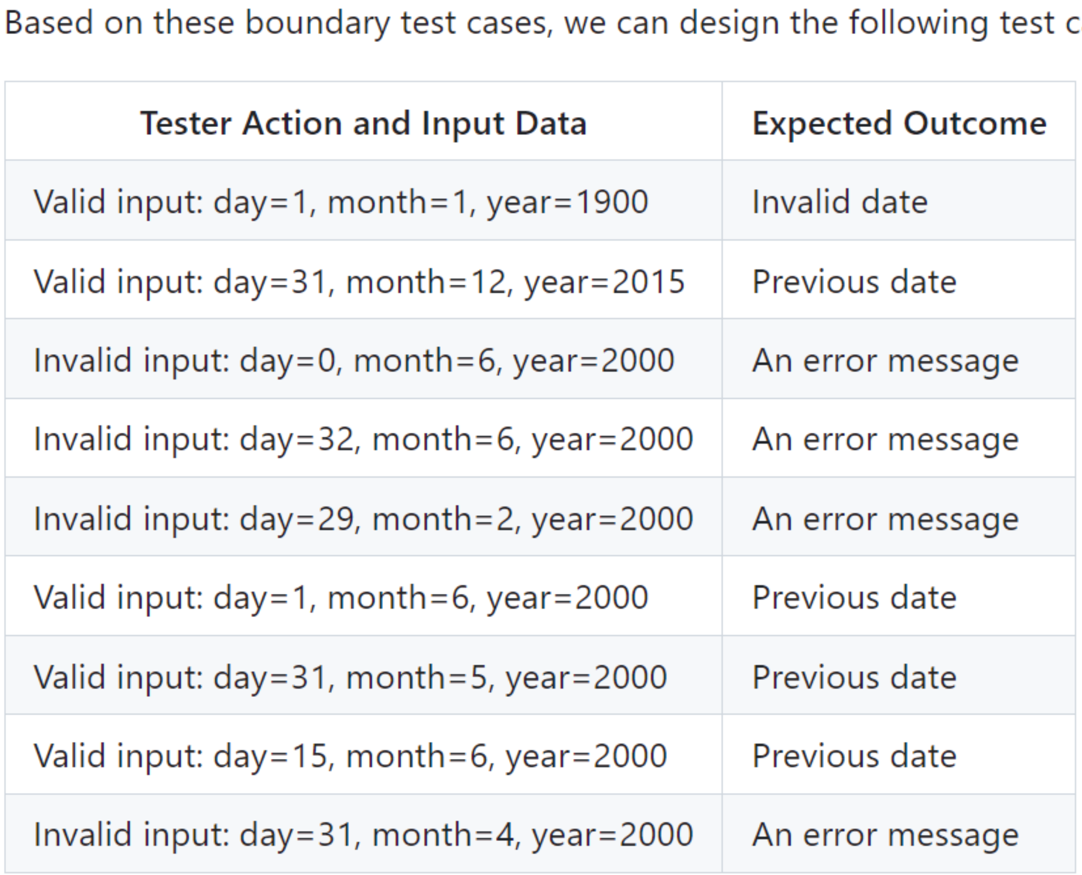
Partition ID	Range	Status
E10	Between 1900 and 2015	Valid
E11	Less than 1	Invalid
E12	Greater than 2015	Invalid

Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome
Valid input: day=1, month=1, year=1900	Invalid date
Valid input: day=31, month=12, year=2015	Previous date
Invalid input: day=0, month=6, year=2000	An error message
Invalid input: day=32, month=6, year=2000	An error message
Invalid input: day=29, month=2, year=2001	An error message

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

1. The earliest possible date: (1, 1, 1900)
2. The latest possible date: (31, 12, 2015)
3. The earliest day of each month: (1, 1, 2000), (1, 2, 2000), (1, 3, 2000),..., (1, 12, 2000)
4. The latest day of each month: (31, 1, 2000), (28, 2, 2000), (31, 3, 2000),..., (31, 12, 2000)
5. Leap year day: (29, 2, 2000)
6. Invalid leap year day: (29, 2, 1900)
7. One day before earliest date: (31, 12, 1899)
8. One day after latest date: (1, 1, 2016)



Based on these boundary test cases, we can design the following test cases:

Tester Action and Input Data	Expected Outcome
Valid input: day=1, month=1, year=1900	Invalid date
Valid input: day=31, month=12, year=2015	Previous date
Invalid input: day=0, month=6, year=2000	An error message
Invalid input: day=32, month=6, year=2000	An error message
Invalid input: day=29, month=2, year=2000	An error message
Valid input: day=1, month=6, year=2000	Previous date
Valid input: day=31, month=5, year=2000	Previous date
Valid input: day=15, month=6, year=2000	Previous date
Invalid input: day=31, month=4, year=2000	An error message

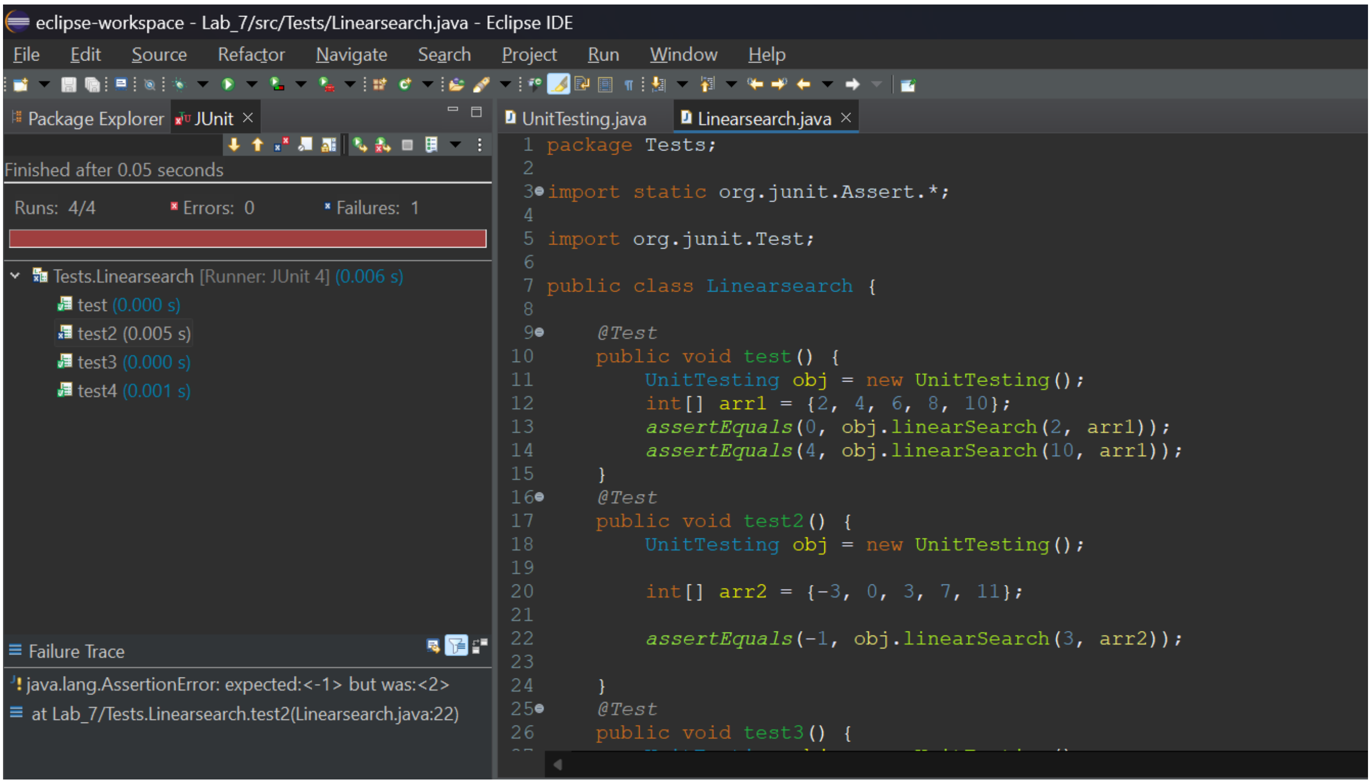
P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

Code :

```
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 :

```
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class Linearsearch {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
        int[] arr1 = {2, 4, 6, 8, 10};
        assertEquals(0, obj.linearSearch(2, arr1));
        assertEquals(4, obj.linearSearch(10, arr1));
    }
    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();

        int[] arr2 = {-3, 0, 3, 7, 11};
        assertEquals(-1, obj.linearSearch(3, arr2));

    }
    @Test
    public void test3() {
        UnitTesting obj = new UnitTesting();

        int[] arr3 = {1, 3, 5, 7, 9};


        assertEquals(4, obj.linearSearch(9, arr3));

    }
    @Test
    public void test4() {
        UnitTesting obj = new UnitTesting();

        int[] arr4 = {};

        assertEquals(-1, obj.linearSearch(2, arr4));
    }



}
```


eclipse-workspace - Lab_7/src/Tests/Linearsearch.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
Linearsearch.java
Failure Trace

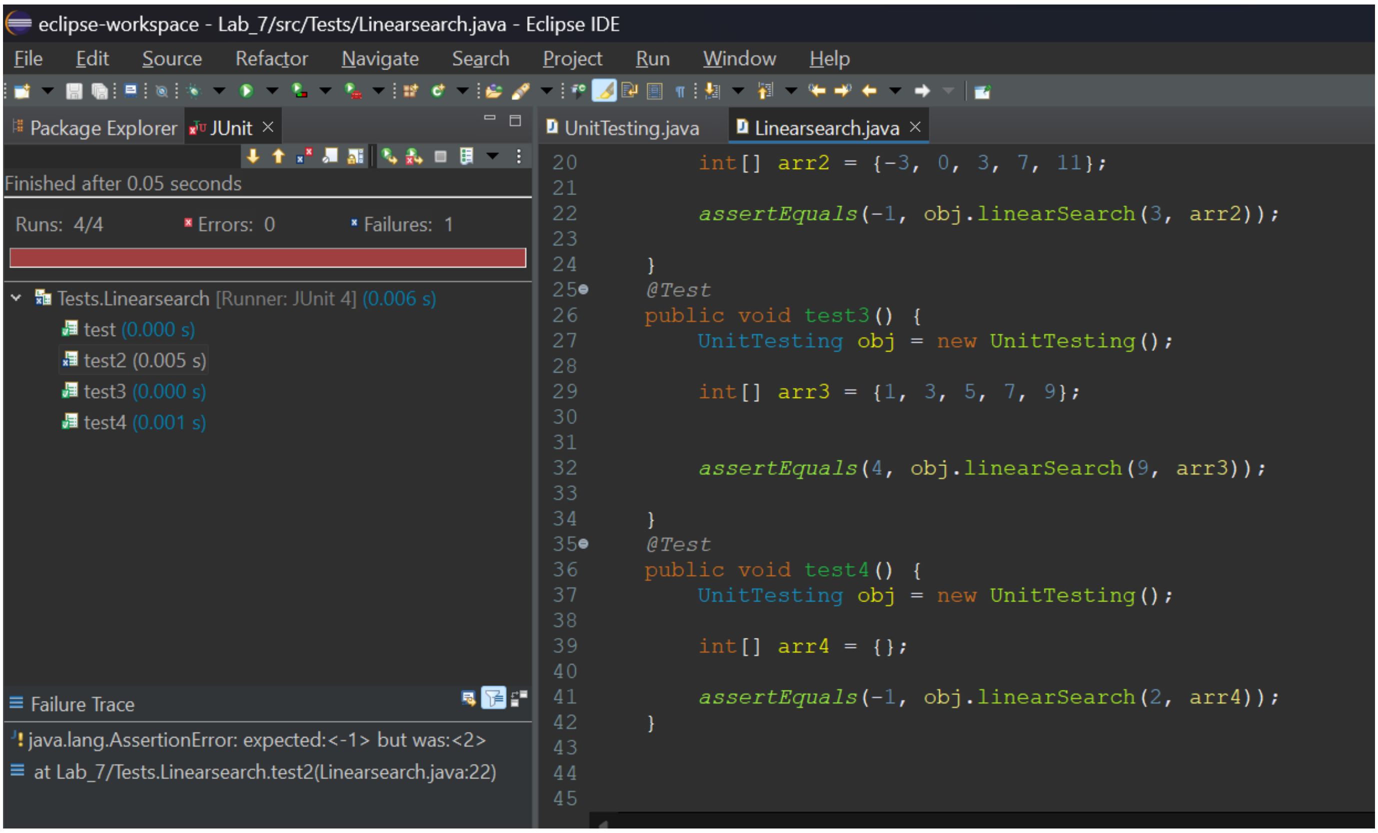
Finished after 0.05 seconds

Runs: 4/4 Errors: 0 Failures: 1

```
Tests.Linearsearch [Runner: JUnit 4] (0.006 s)
    test (0.000 s)
    test2 (0.005 s)
    test3 (0.000 s)
    test4 (0.001 s)
```

```
java.lang.AssertionError: expected:<-1> but was:<2>
at Lab_7/Tests.Linearsearch.test2(Linearsearch.java:22)
```

```java
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class Linearsearch {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
        int[] arr1 = {2, 4, 6, 8, 10};
        assertEquals(0, obj.linearSearch(2, arr1));
        assertEquals(4, obj.linearSearch(10, arr1));
    }
    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();

        int[] arr2 = {-3, 0, 3, 7, 11};

        assertEquals(-1, obj.linearSearch(3, arr2));

    }
    @Test
    public void test3() {
```


eclipse-workspace - Lab_7/src/Tests/Linearsearch.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
Linearsearch.java
Failure Trace

Finished after 0.05 seconds

Runs: 4/4 Errors: 0 Failures: 1

```
Tests.Linearsearch [Runner: JUnit 4] (0.006 s)
    test (0.000 s)
    test2 (0.005 s)
    test3 (0.000 s)
    test4 (0.001 s)
```

```
java.lang.AssertionError: expected:<-1> but was:<2>
at Lab_7/Tests.Linearsearch.test2(Linearsearch.java:22)
```

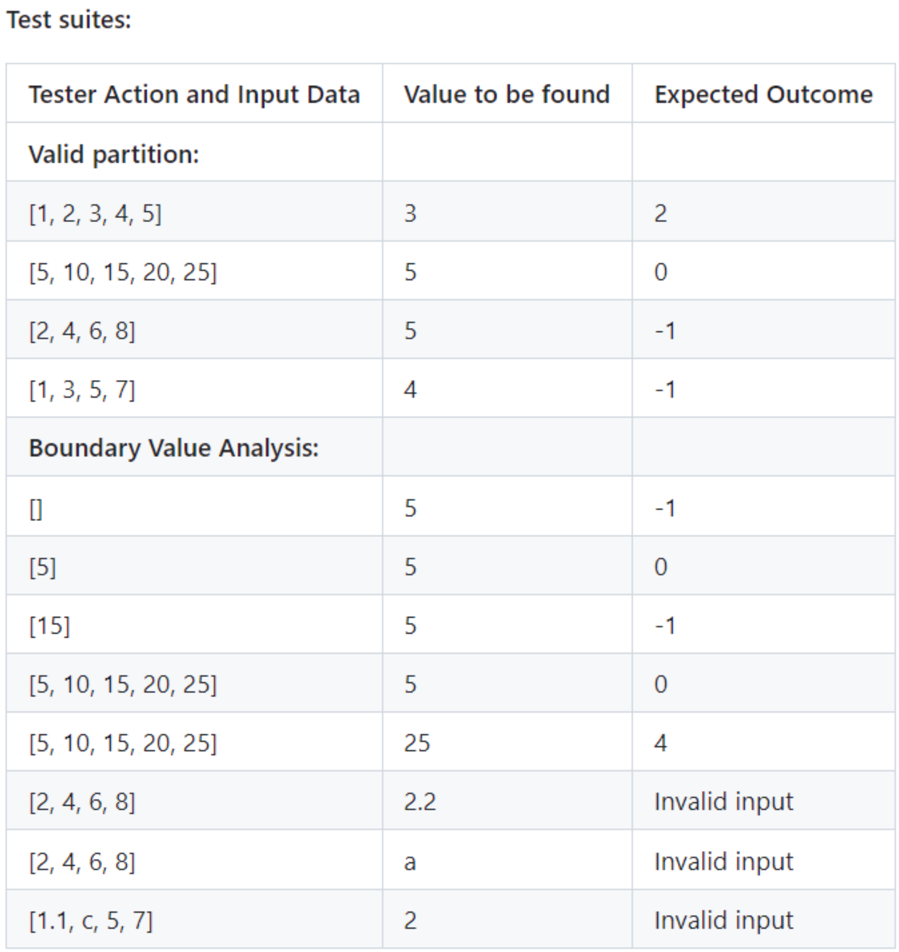
```java
        int[] arr2 = {-3, 0, 3, 7, 11};

        assertEquals(-1, obj.linearSearch(3, arr2));

    }
    @Test
    public void test3() {
        UnitTesting obj = new UnitTesting();

        int[] arr3 = {1, 3, 5, 7, 9};


        assertEquals(4, obj.linearSearch(9, arr3));

    }
    @Test
    public void test4() {
        UnitTesting obj = new UnitTesting();

        int[] arr4 = {};

        assertEquals(-1, obj.linearSearch(2, arr4));
    }



```

Equivalence Partitioning:

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

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array a	-1
v is present at the first index of a	0
v is present at the last index of a length of a	a-1
v is not present in a	-1



Test suites:

Tester Action and Input Data	Value to be found	Expected Outcome
Valid partition:		
[1, 2, 3, 4, 5]	3	2
[5, 10, 15, 20, 25]	5	0
[2, 4, 6, 8]	5	-1
[1, 3, 5, 7]	4	-1
Boundary Value Analysis:		
[]	5	-1
[5]	5	0
[15]	5	-1
[5, 10, 15, 20, 25]	5	0
[5, 10, 15, 20, 25]	25	4
[2, 4, 6, 8]	2.2	Invalid input
[2, 4, 6, 8]	a	Invalid input
[1.1, c, 5, 7]	2	Invalid input

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

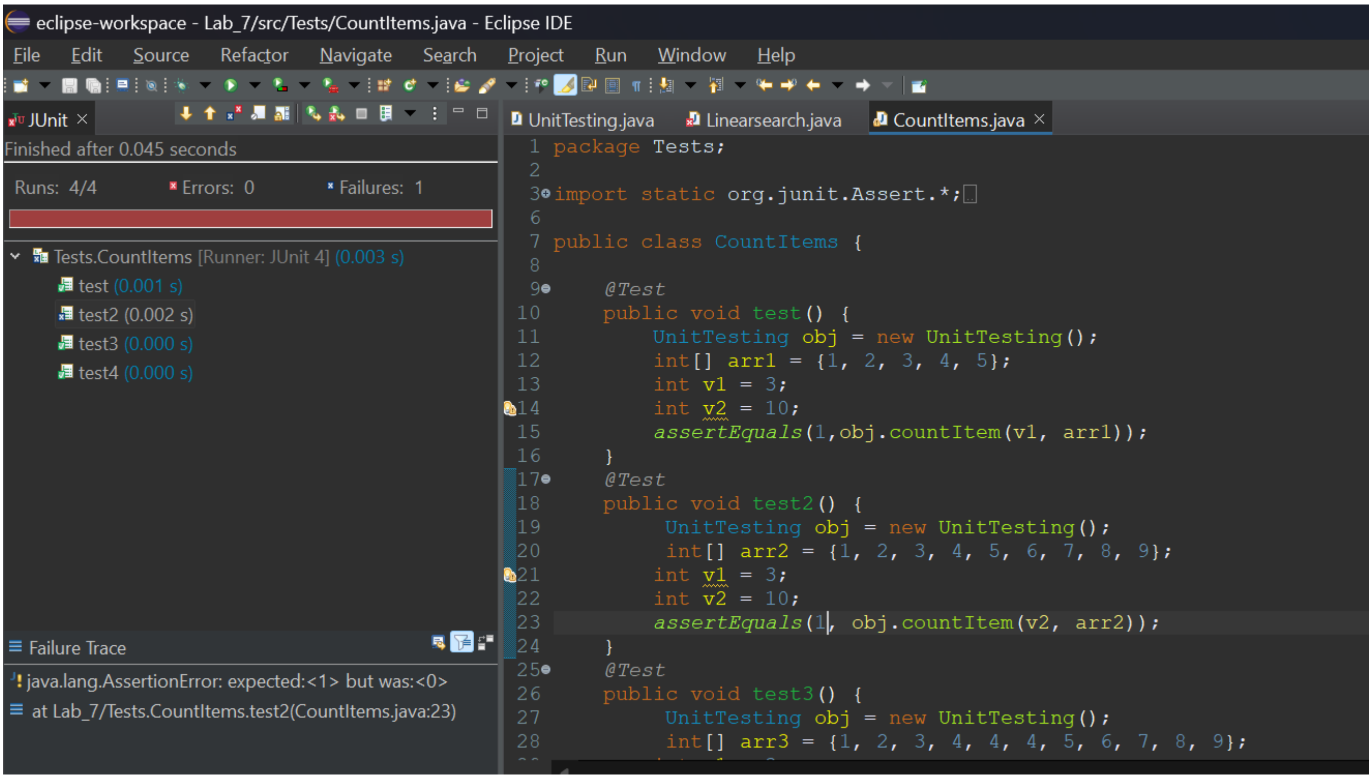
Code :

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

Testing code :

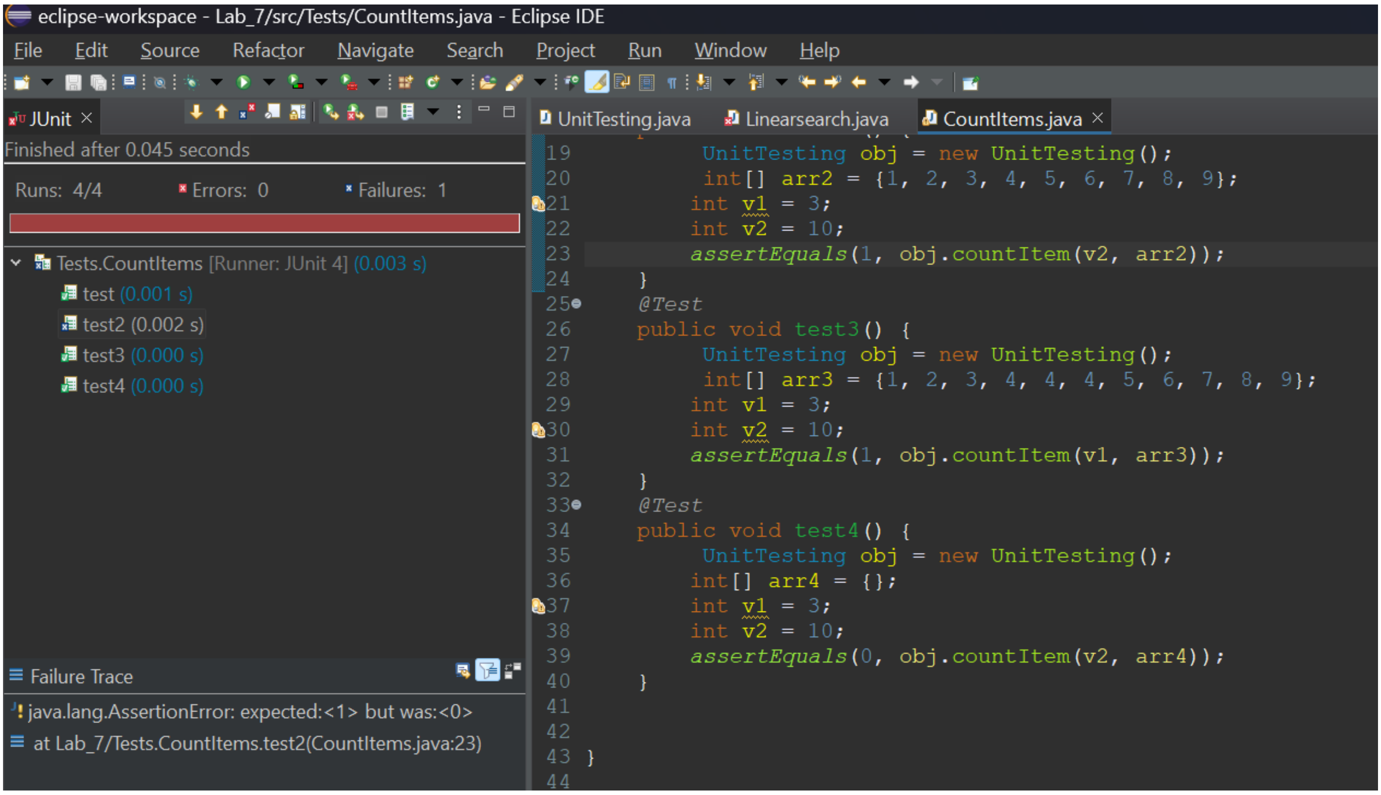
```
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class CountItems {

     @Test
     public void test() {
         UnitTesting obj = new UnitTesting();
         int[] arr1 = {1, 2, 3, 4, 5};
         int v1 = 3;
         int v2 = 10;
         assertEquals(1,obj.countItem(v1, arr1));
     }
     @Test
     public void test2() {
           UnitTesting obj = new UnitTesting();
           int[] arr2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
         int v1 = 3;
         int v2 = 10;
         assertEquals(1, obj.countItem(v2, arr2));
     }
     @Test
     public void test3() {
           UnitTesting obj = new UnitTesting();
           int[] arr3 = {1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 9};
         int v1 = 3;
         int v2 = 10;
```

```
            assertEquals(1, obj.countItem(v1, arr3));
        }
        @Test
        public void test4() {
                UnitTesting obj = new UnitTesting();
             int[] arr4 = {};
            int v1 = 3;
            int v2 = 10;
            assertEquals(0, obj.countItem(v2, arr4));
        }



}
```


eclipse-workspace - Lab_7/src/Tests/CountItems.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit
UnitTesting.java
Linearsearch.java
CountItems.java
Failure Trace

```
Finished after 0.045 seconds
Runs: 4/4      Errors: 0      Failures: 1

Tests.CountItems [Runner: JUnit 4] (0.003 s)
    test (0.001 s)
    test2 (0.002 s)
    test3 (0.000 s)
    test4 (0.000 s)

java.lang.AssertionError: expected:<1> but was:<0>
at Lab_7/Tests.CountItems.test2(CountItems.java:23)
```

```
package Tests;

import static org.junit.Assert.*;

public class CountItems {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
        int[] arr1 = {1, 2, 3, 4, 5};
        int v1 = 3;
        int v2 = 10;
        assertEquals(1,obj.countItem(v1, arr1));
    }
    @Test
    public void test2() {
         UnitTesting obj = new UnitTesting();
         int[] arr2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int v1 = 3;
        int v2 = 10;
        assertEquals(1, obj.countItem(v2, arr2));
    }
    @Test
    public void test3() {
         UnitTesting obj = new UnitTesting();
         int[] arr3 = {1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 9};
```


eclipse-workspace - Lab_7/src/Tests/CountItems.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit
UnitTesting.java
Linearsearch.java
CountItems.java

Finished after 0.045 seconds

Runs: 4/4 Errors: 0 Failures: 1

```
Tests.CountItems [Runner: JUnit 4] (0.003 s)
    test (0.001 s)
    test2 (0.002 s)
    test3 (0.000 s)
    test4 (0.000 s)
```

Failure Trace

```
java.lang.AssertionError: expected:<1> but was:<0>
at Lab_7/Tests.CountItems.test2(CountItems.java:23)
```

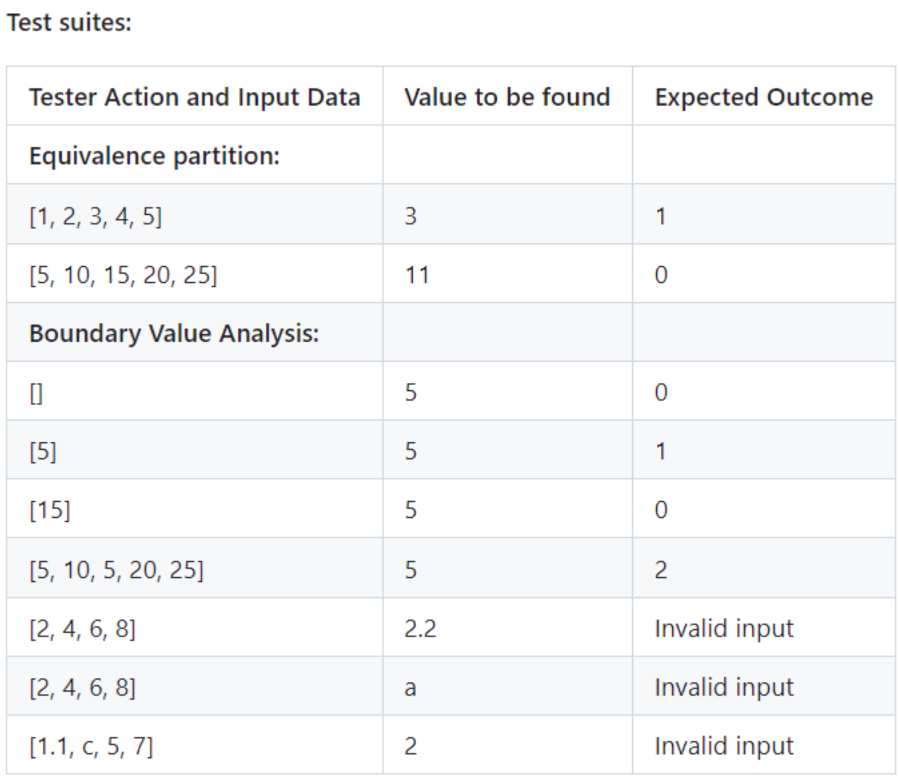
```
         UnitTesting obj = new UnitTesting();
         int[] arr2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int v1 = 3;
        int v2 = 10;
        assertEquals(1, obj.countItem(v2, arr2));
    }
    @Test
    public void test3() {
         UnitTesting obj = new UnitTesting();
         int[] arr3 = {1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 9};
        int v1 = 3;
        int v2 = 10;
        assertEquals(1, obj.countItem(v1, arr3));
    }
    @Test
    public void test4() {
         UnitTesting obj = new UnitTesting();
        int[] arr4 = {};
        int v1 = 3;
        int v2 = 10;
        assertEquals(0, obj.countItem(v2, arr4));
    }


}
```

Equivalence Partitioning:

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

Boundary Value Analysis:

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



Test suites:

Tester Action and Input Data	Value to be found	Expected Outcome
Equivalence partition:		
[1, 2, 3, 4, 5]	3	1
[5, 10, 15, 20, 25]	11	0
Boundary Value Analysis:		
[]	5	0
[5]	5	1
[15]	5	0
[5, 10, 5, 20, 25]	5	2
[2, 4, 6, 8]	2.2	Invalid input
[2, 4, 6, 8]	a	Invalid input
[1.1, c, 5, 7]	2	Invalid input

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.

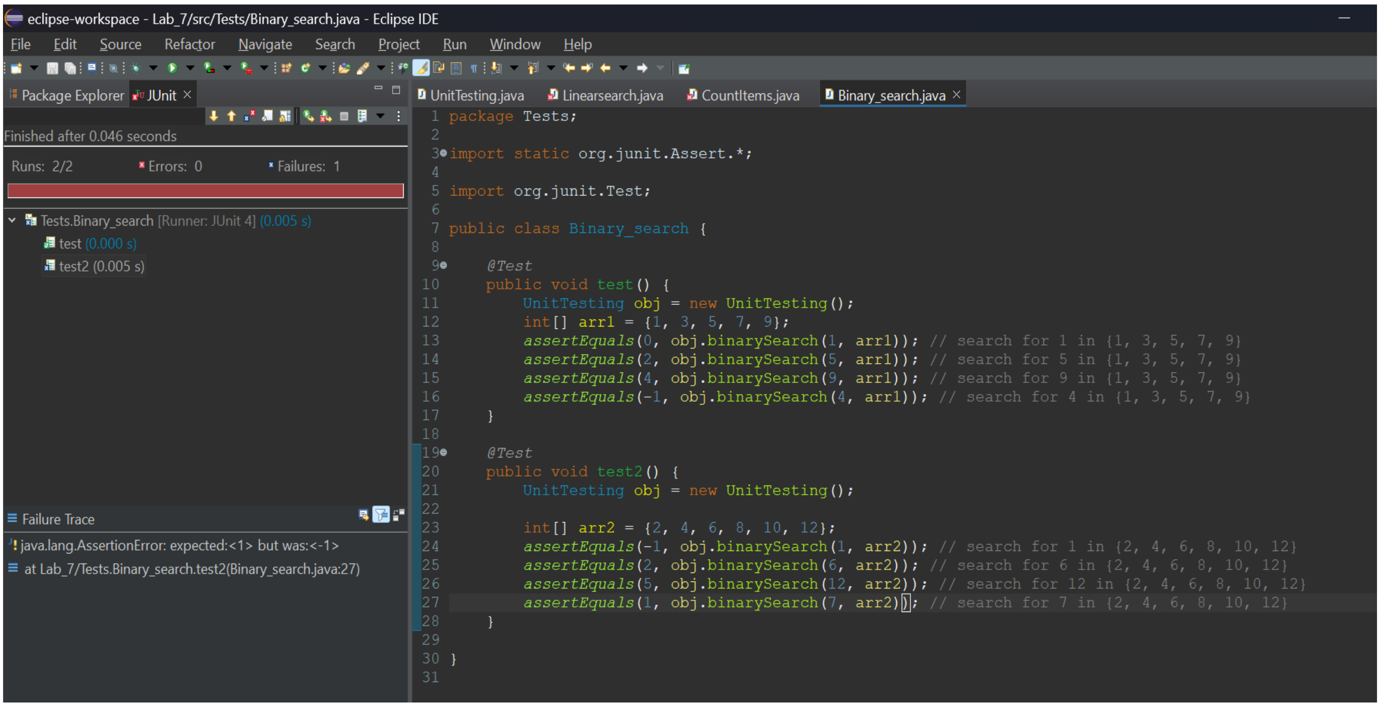
Code :

```
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);
}
```

Testing code :

```
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class Binary_search {

     @Test
     public void test() {
         UnitTesting obj = new UnitTesting();
         int[] arr1 = {1, 3, 5, 7, 9};
        assertEquals(0, obj.binarySearch(1, arr1)); // search
for 1 in {1, 3, 5, 7, 9}
```

```
        assertEquals(2, obj.binarySearch(5, arr1)); // search
for 5 in {1, 3, 5, 7, 9}

        assertEquals(4, obj.binarySearch(9, arr1)); // search
for 9 in {1, 3, 5, 7, 9}

        assertEquals(-1, obj.binarySearch(4, arr1)); // search
for 4 in {1, 3, 5, 7, 9}

     }



     @Test

     public void test2() {

         UnitTesting obj = new UnitTesting();



        int[] arr2 = {2, 4, 6, 8, 10, 12};

        assertEquals(-1, obj.binarySearch(1, arr2)); // search
for 1 in {2, 4, 6, 8, 10, 12}

        assertEquals(2, obj.binarySearch(6, arr2)); // search
for 6 in {2, 4, 6, 8, 10, 12}

        assertEquals(5, obj.binarySearch(12, arr2)); // search
for 12 in {2, 4, 6, 8, 10, 12}

        assertEquals(1, obj.binarySearch(7, arr2)); // search
for 7 in {2, 4, 6, 8, 10, 12}

     }



}
```


eclipse-workspace - Lab_7/src/Tests/Binary_search.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
Linearsearch.java
CountItems.java
Binary_search.java
Failure Trace

```
Finished after 0.046 seconds
Runs: 2/2    Errors: 0    Failures: 1

Tests.Binary_search [Runner: JUnit 4] (0.005 s)
    test (0.000 s)
    test2 (0.005 s)
```

```
java.lang.AssertionError: expected:<1> but was:<-1>
at Lab_7/Tests.Binary_search.test2(Binary_search.java:27)
```

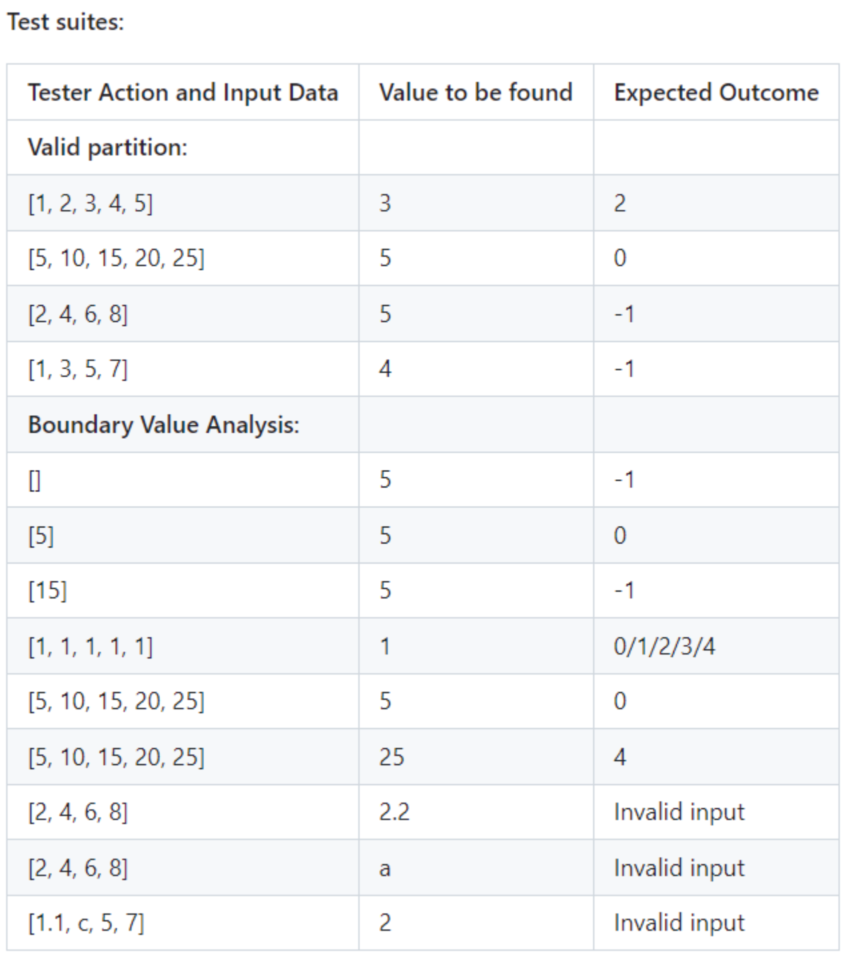
```java
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class Binary_search {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
        int[] arr1 = {1, 3, 5, 7, 9};
        assertEquals(0, obj.binarySearch(1, arr1)); // search for 1 in {1, 3, 5, 7, 9}
        assertEquals(2, obj.binarySearch(5, arr1)); // search for 5 in {1, 3, 5, 7, 9}
        assertEquals(4, obj.binarySearch(9, arr1)); // search for 9 in {1, 3, 5, 7, 9}
        assertEquals(-1, obj.binarySearch(4, arr1)); // search for 4 in {1, 3, 5, 7, 9}
    }

    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();

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

}
```

Equivalence Partitioning:

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

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array a	-1
v is present at the first index of a	0
v is present at the last index of a length of a	a-1
v is not present in a	-1



Test suites:

Tester Action and Input Data	Value to be found	Expected Outcome
Valid partition:		
[1, 2, 3, 4, 5]	3	2
[5, 10, 15, 20, 25]	5	0
[2, 4, 6, 8]	5	-1
[1, 3, 5, 7]	4	-1
Boundary Value Analysis:		
[]	5	-1
[5]	5	0
[15]	5	-1
[1, 1, 1, 1, 1]	1	0/1/2/3/4
[5, 10, 15, 20, 25]	5	0
[5, 10, 15, 20, 25]	25	4
[2, 4, 6, 8]	2.2	Invalid input
[2, 4, 6, 8]	a	Invalid input
[1.1, c, 5, 7]	2	Invalid input

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).

Code :

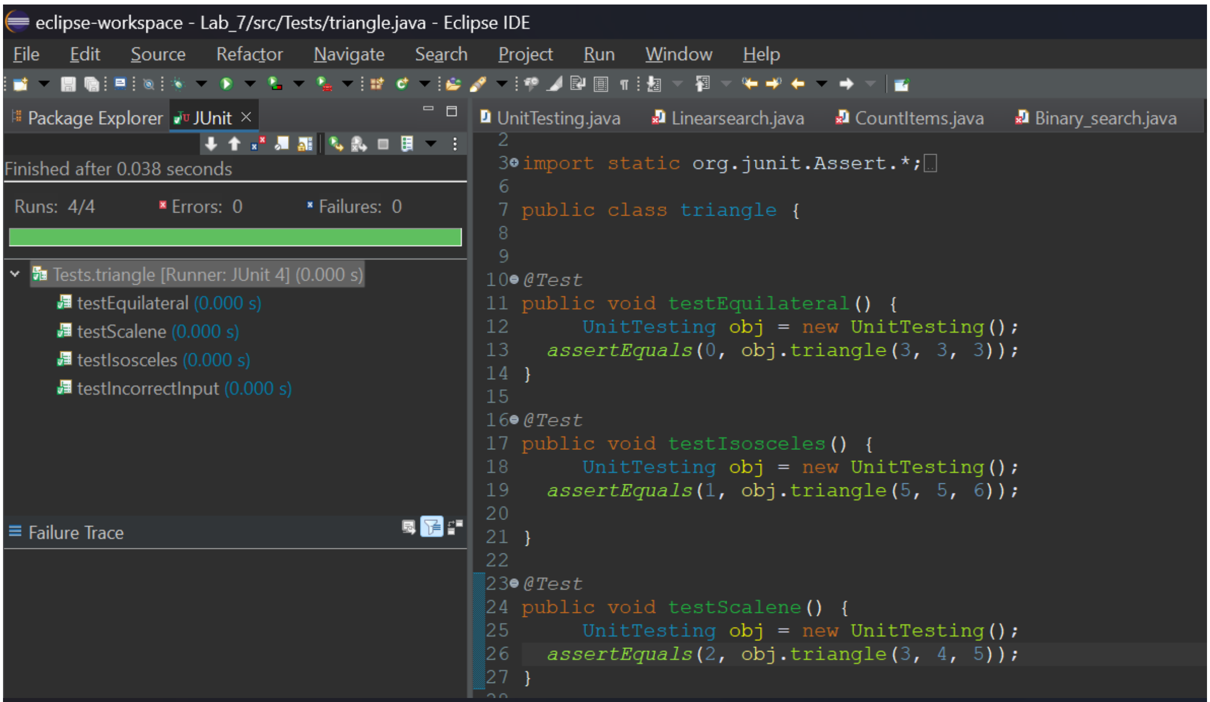
```
final int EQUILATERAL = 0; final int ISOSCELES = 1; final int SCALENE = 2;
final int INVALID = 3;



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);

}
```

Testing code :

```
package Tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class triangle {
```

```
@Test
public void testEquilateral() {
      UnitTesting obj = new UnitTesting();
  assertEquals(0, obj.triangle(3, 3, 3));
}

@Test
public void testIsosceles() {
      UnitTesting obj = new UnitTesting();
  assertEquals(1, obj.triangle(5, 5, 6));

}

@Test
public void testScalene() {
      UnitTesting obj = new UnitTesting();
  assertEquals(2, obj.triangle(3, 4, 5));
}

@Test
public void testIncorrectInput() {
      UnitTesting obj = new UnitTesting();

  assertEquals(3, obj.triangle(1, 2, 3));

}

}
```

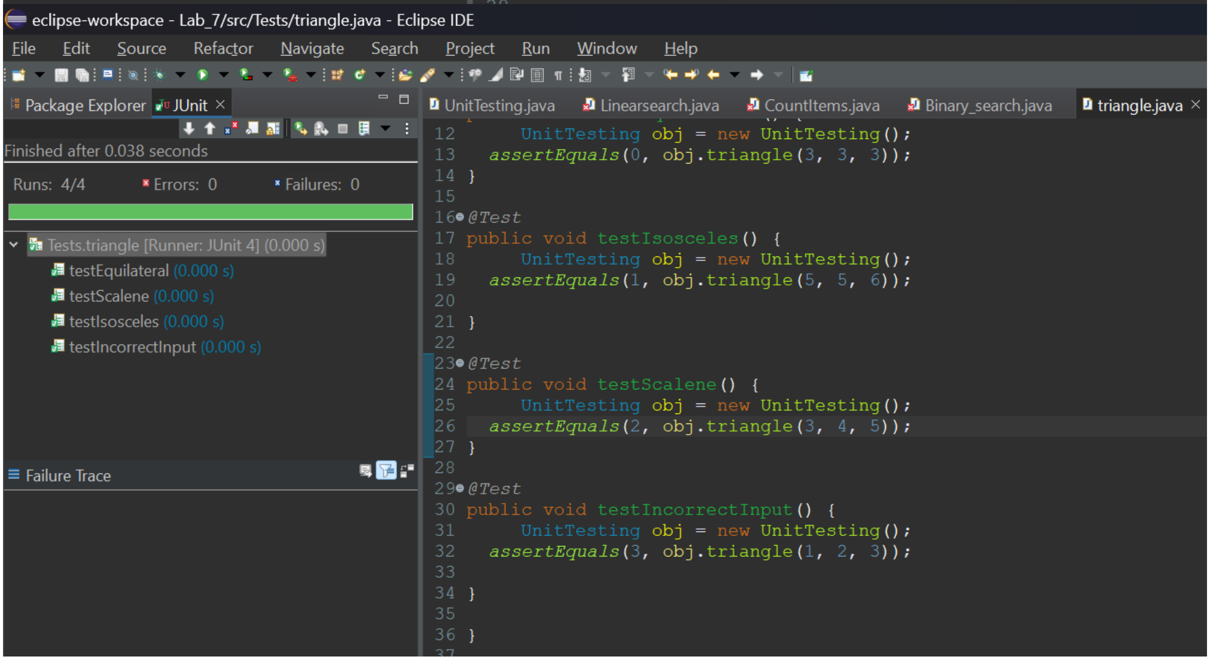

eclipse-workspace - Lab_7/src/Tests/triangle.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
Linearsearch.java
CountItems.java
Binary_search.java
Failure Trace

Finished after 0.038 seconds

Runs: 4/4 Errors: 0 Failures: 0

- Tests.triangle [Runner: JUnit 4] (0.000 s)
  - testEquilateral (0.000 s)
  - testScalene (0.000 s)
  - testIsosceles (0.000 s)
  - testIncorrectInput (0.000 s)

```
import static org.junit.Assert.*;

public class triangle {


@Test
public void testEquilateral() {
        UnitTesting obj = new UnitTesting();
    assertEquals(0, obj.triangle(3, 3, 3));
}

@Test
public void testIsosceles() {
        UnitTesting obj = new UnitTesting();
    assertEquals(1, obj.triangle(5, 5, 6));

}

@Test
public void testScalene() {
        UnitTesting obj = new UnitTesting();
    assertEquals(2, obj.triangle(3, 4, 5));
}
```

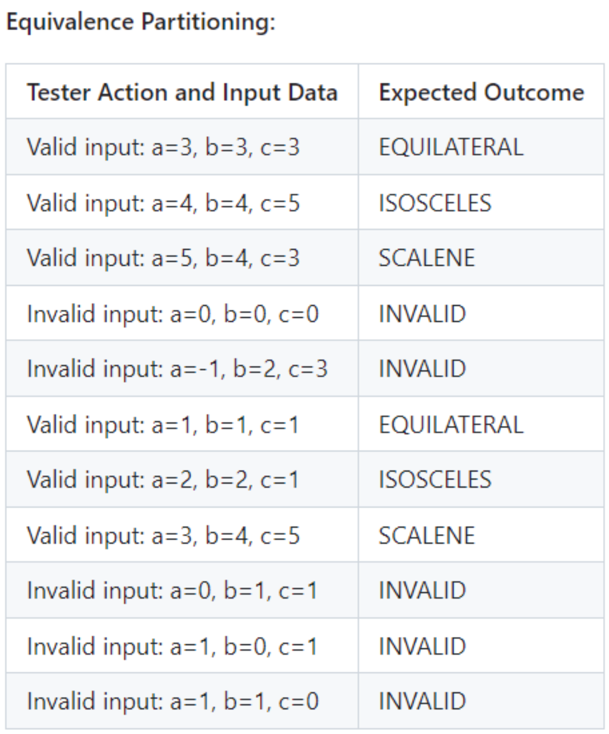

eclipse-workspace - Lab_7/src/Tests/triangle.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
UnitTesting.java
Linearsearch.java
CountItems.java
Binary_search.java
triangle.java
Failure Trace

Finished after 0.038 seconds

Runs: 4/4 Errors: 0 Failures: 0

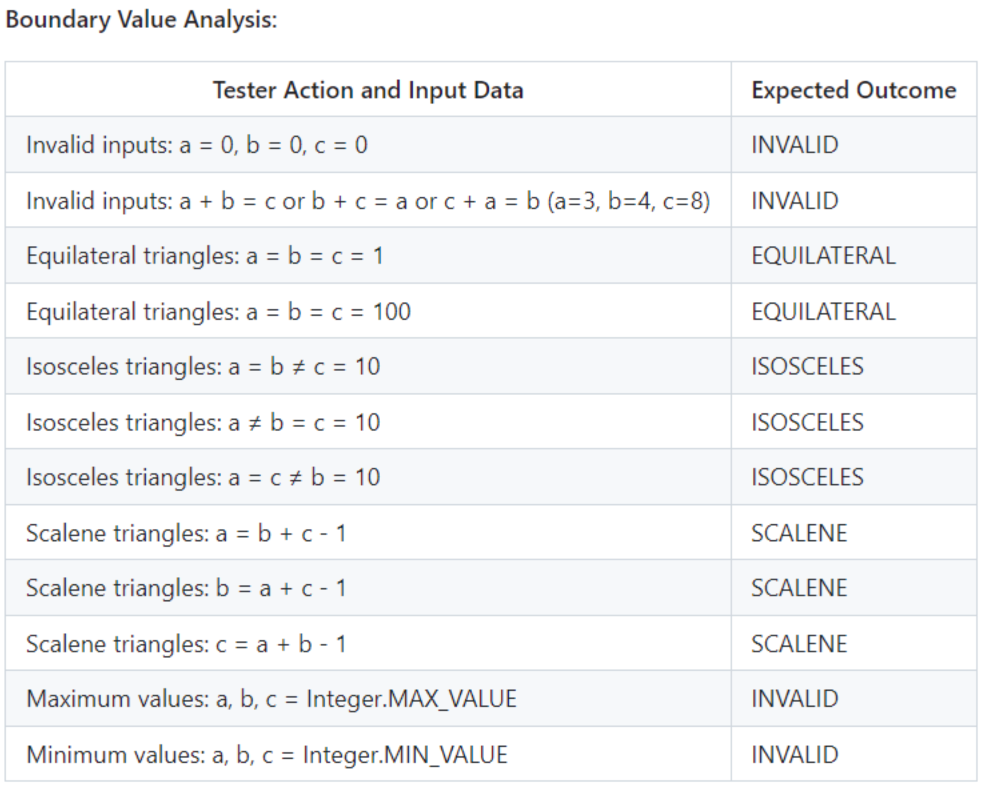
- Tests.triangle [Runner: JUnit 4] (0.000 s)
  - testEquilateral (0.000 s)
  - testScalene (0.000 s)
  - testIsosceles (0.000 s)
  - testIncorrectInput (0.000 s)

```
        UnitTesting obj = new UnitTesting();
    assertEquals(0, obj.triangle(3, 3, 3));
}

@Test
public void testIsosceles() {
        UnitTesting obj = new UnitTesting();
    assertEquals(1, obj.triangle(5, 5, 6));

}

@Test
public void testScalene() {
        UnitTesting obj = new UnitTesting();
    assertEquals(2, obj.triangle(3, 4, 5));
}

@Test
public void testIncorrectInput() {
        UnitTesting obj = new UnitTesting();
    assertEquals(3, obj.triangle(1, 2, 3));

}

}
```



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 ≠ c = 10	ISOSCELES
Isosceles triangles: a ≠ b = c = 10	ISOSCELES
Isosceles triangles: a = c ≠ 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 = Integer.MAX_VALUE	INVALID
Minimum values: a, b, c = Integer.MIN_VALUE	INVALID

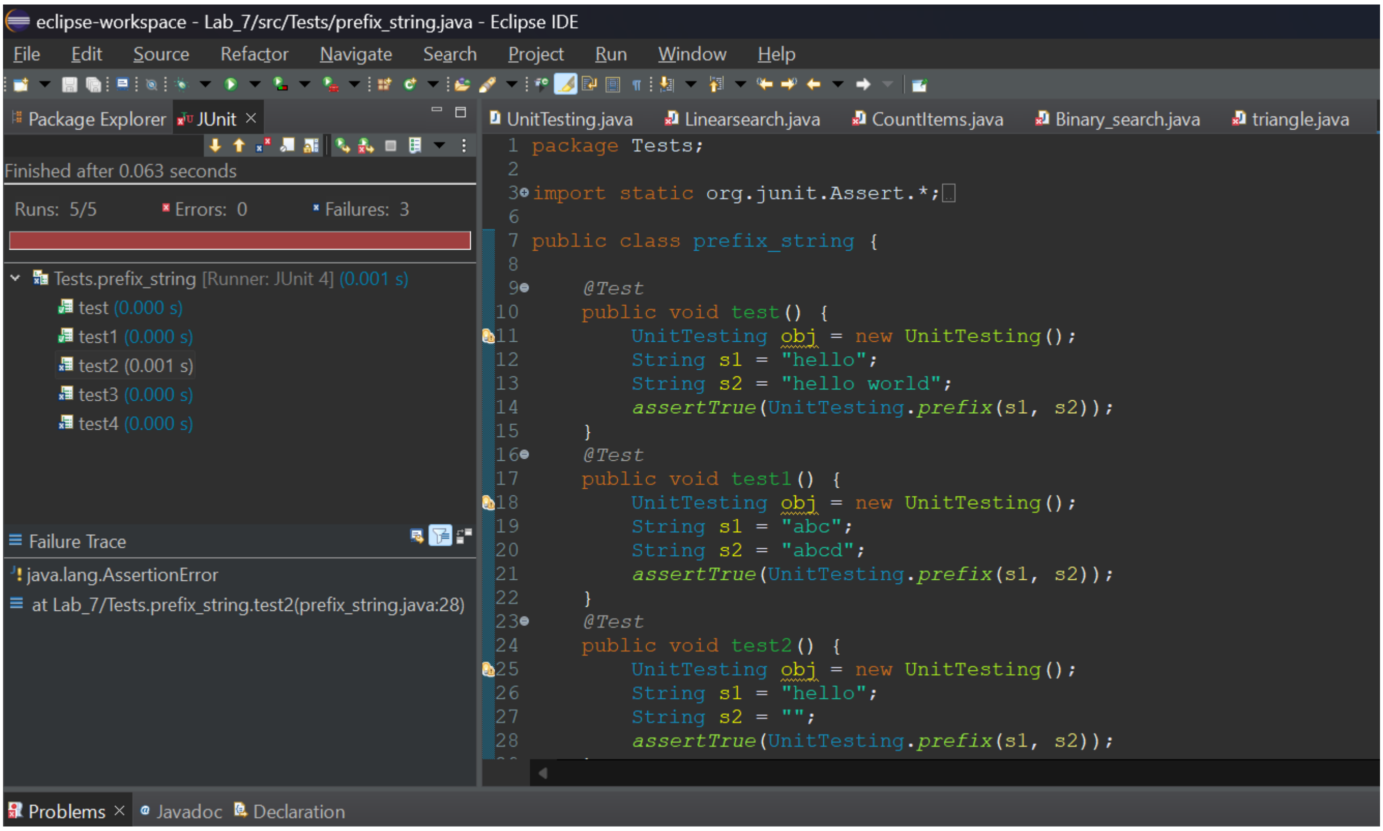
P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

Code :

```
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;
}
```

Testing code :

```
package Tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class prefix_string {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
    String s1 = "hello";
    String s2 = "hello world";
    assertTrue(UnitTesting.prefix(s1, s2));
}
@Test
public void test1() {
    UnitTesting obj = new UnitTesting();
    String s1 = "abc";
    String s2 = "abcd";
    assertTrue(UnitTesting.prefix(s1, s2));
}
@Test
public void test2() {
    UnitTesting obj = new UnitTesting();
    String s1 = "hello";
    String s2 = "";
    assertTrue(UnitTesting.prefix(s1, s2));
}
@Test
public void test3() {
    UnitTesting obj = new UnitTesting();
    String s1 = "hello";
    String s2 = "hi";
    assertTrue(UnitTesting.prefix(s1, s2));
}
@Test
public void test4() {
        UnitTesting obj = new UnitTesting();

        String s1 = "abc";

        String s2 = "def";

        assertTrue(UnitTesting.prefix(s1, s2));

    }

}
```



eclipse-workspace - Lab_7/src/Tests/prefix_string.java - Eclipse IDE

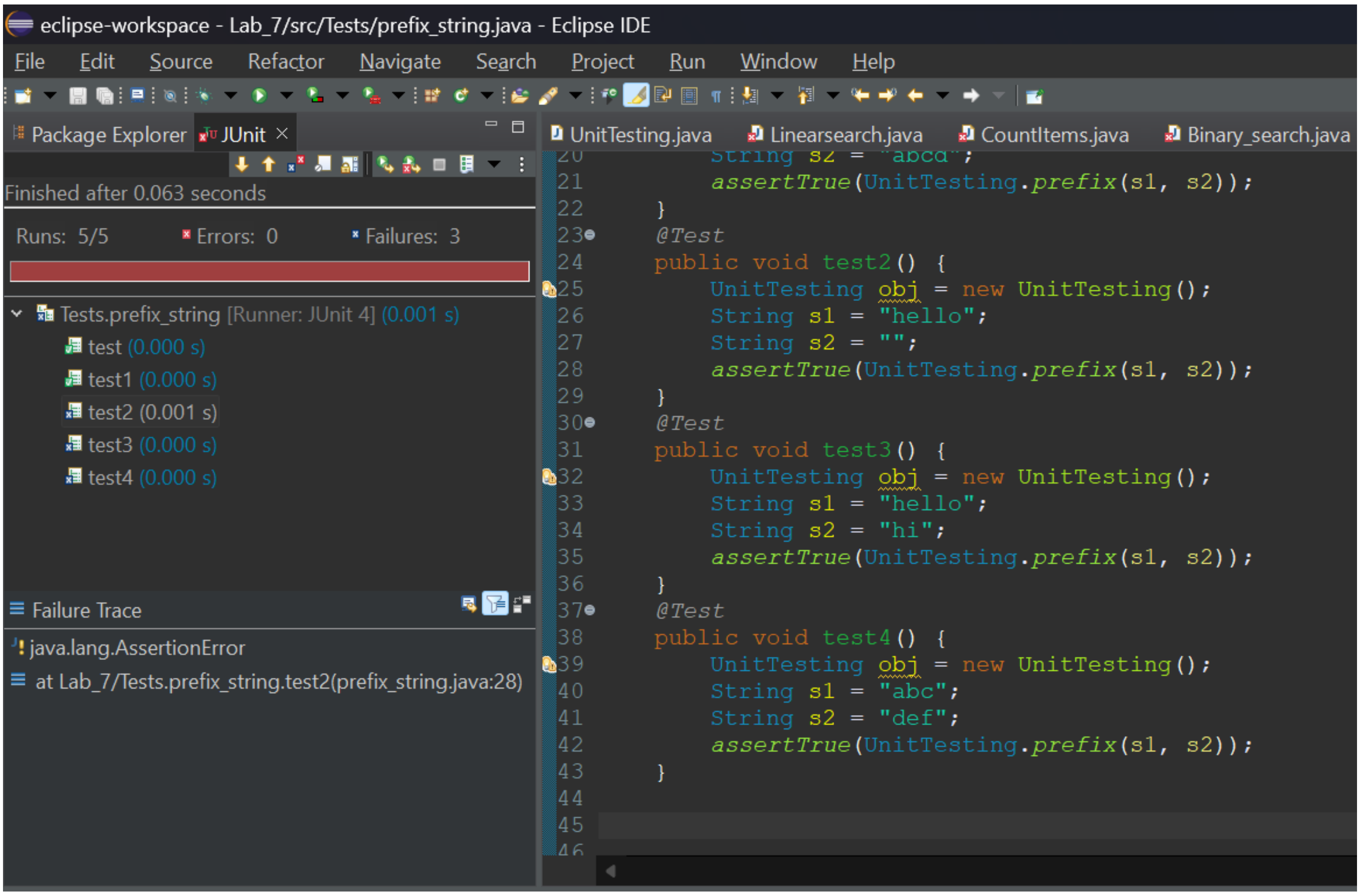
Finished after 0.063 seconds

Runs: 5/5 Errors: 0 Failures: 3

```
Tests.prefix_string [Runner: JUnit 4] (0.001 s)
    test (0.000 s)
    test1 (0.000 s)
    test2 (0.001 s)
    test3 (0.000 s)
    test4 (0.000 s)
```

```
java.lang.AssertionError
at Lab_7/Tests.prefix_string.test2(prefix_string.java:28)
```

```
package Tests;

import static org.junit.Assert.*;

public class prefix_string {

    @Test
    public void test() {
        UnitTesting obj = new UnitTesting();
        String s1 = "hello";
        String s2 = "hello world";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
    @Test
    public void test1() {
        UnitTesting obj = new UnitTesting();
        String s1 = "abc";
        String s2 = "abcd";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();
        String s1 = "hello";
        String s2 = "";
        assertTrue(UnitTesting.prefix(s1, s2));
```



```
Finished after 0.063 seconds
Runs:  5/5        Errors:  0        Failures:  3

Tests.prefix_string [Runner: JUnit 4] (0.001 s)
    test (0.000 s)
    test1 (0.000 s)
    test2 (0.001 s)
    test3 (0.000 s)
    test4 (0.000 s)
```

```
java.lang.AssertionError
at Lab_7/Tests.prefix_string.test2(prefix_string.java:28)
```

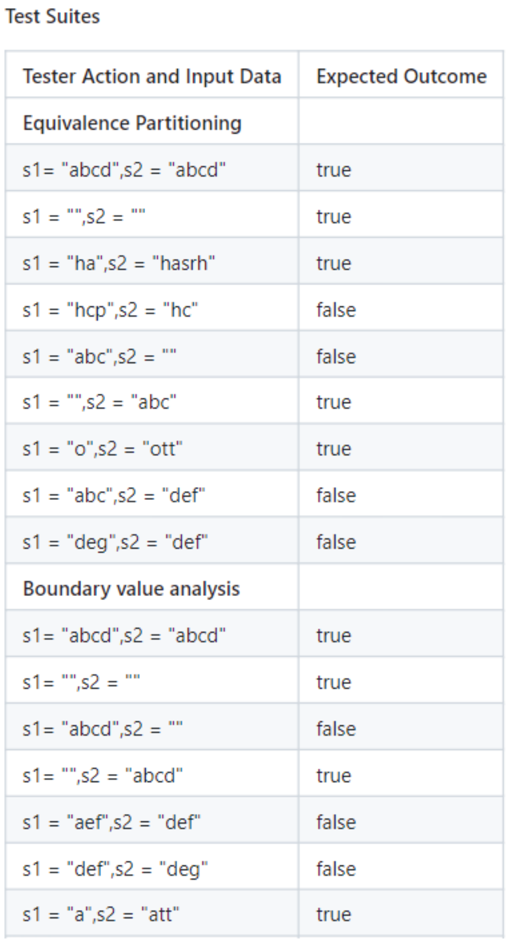
```java
        String s2 = "abcd";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();
        String s1 = "hello";
        String s2 = "";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
    @Test
    public void test3() {
        UnitTesting obj = new UnitTesting();
        String s1 = "hello";
        String s2 = "hi";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
    @Test
    public void test4() {
        UnitTesting obj = new UnitTesting();
        String s1 = "abc";
        String s2 = "def";
        assertTrue(UnitTesting.prefix(s1, s2));
    }
```

Equivalence Partitioning:

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 a prefix of non-empty s2	True
Non-empty s1 is not a prefix of s2	False
Non-empty s1 is longer than s2	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 Suites

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
s1= "abcd",s2 = "abcd"	true
s1 = "",s2 = ""	true
s1 = "ha",s2 = "hasrh"	true
s1 = "hcp",s2 = "hc"	false
s1 = "abc",s2 = ""	false
s1 = "",s2 = "abc"	true
s1 = "o",s2 = "ott"	true
s1 = "abc",s2 = "def"	false
s1 = "deg",s2 = "def"	false
Boundary value analysis	
s1= "abcd",s2 = "abcd"	true
s1= "",s2 = ""	true
s1= "abcd",s2 = ""	false
s1= "",s2 = "abcd"	true
s1 = "aef",s2 = "def"	false
s1 = "def",s2 = "deg"	false
s1 = "a",s2 = "att"	true

P6: Consider again the triangle classification program (P4) with a slightly different specification: The programreads floating values from the standard input. The three values A, B, and C are interpreted asrepresenting 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

Equivalence Classes:
EC1: Invalid inputs (negative or zero values)
EC2: Non-triangle (sum of the two shorter sides is not greater than the longest side)
EC3: Scalene triangle (no sides are equal)
EC4: Isosceles triangle (two sides are equal)
EC5: Equilateral triangle (all sides are equal)
EC6: Right-angled triangle (satisfies the Pythagorean theorem)

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 cases:
TC1: -1, 0
TC2: 1, 2, 5
TC3: 3, 4, 5
TC4: 5, 5, 7
TC5: 6, 6, 6
TC6: 3, 4, 5

Test case 1 covers class 1, test case 2 covers class 2, test case 3 covers class 3, test case 4 covers class 4, test case 5 covers class 5, and test case 6 covers class 6

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

2, 3, 6
3, 4, 8
Both test cases have two sides that are shorter than the third side, and should not form a triangle

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

1, 2, 1
0, 2, 0
5, 6, 5
Both test cases have two sides that are equal, but only test case 2 should form an isosceles triangle, other input are invalid.

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

5, 5, 5
0, 0, 0
Both test cases have all sides equal, but only test case 1 should form an equilateral triangle, other input are invalid.

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

3, 4, 5 0, 0, 0 -3, -4, -5 Both test cases satisfy the Pythagorean theorem, but only test case 1 should form right-angled triangle, other input are invalid. triangle

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

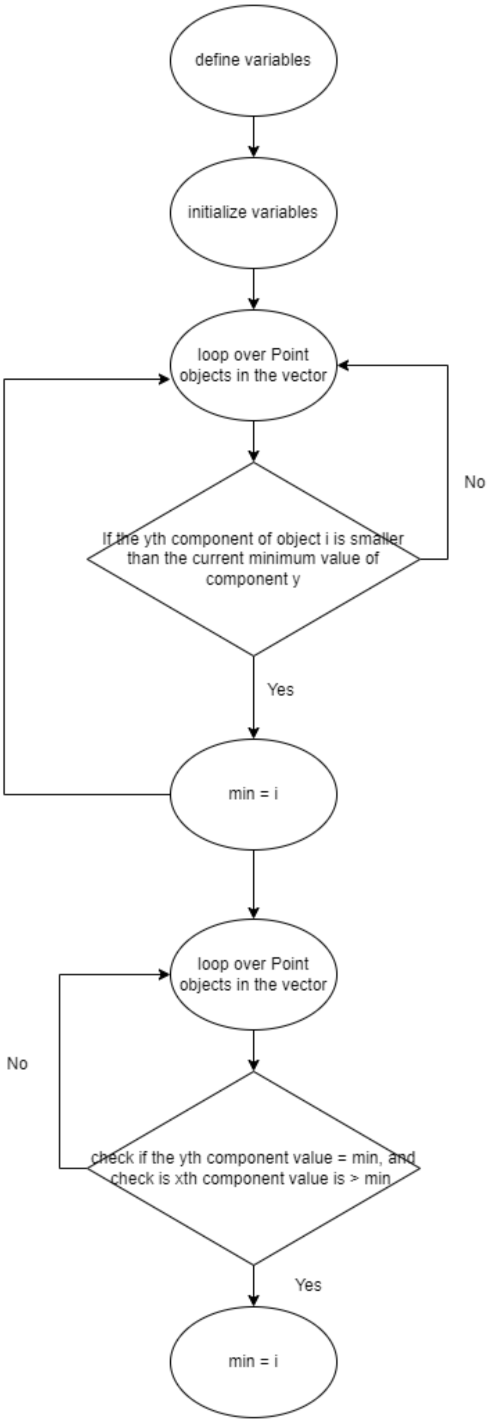
Test cases for the non-triangle case:
TC11 (EC3): A=2, B=2, C=4 (sum of A and B is less than C)

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

Test points for non-positive input:
TP1 (EC2): A=0, B=4, C=5 (invalid input)
TP2 (EC2): A=-2, B=4, C=5 (invalid input)
Note: Test cases TC1 to TC10 covers all identified equivalence classes.

Section B:

1. Control flow diagram


define variables
initialize variables
loop over Point objects in the vector
No
If the yth component of object i is smaller than the current minimum value of component y
Yes
min = i
loop over Point objects in the vector
No
check if the yth component value = min, and check is xth component value is > min
Yes
min = i

2. Test sets

Statement coverage test sets: To achieve statement coverage, we need to make sure that every statement in the code is executed at least once.
Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components
Test 6: p = vector with three or more points with the same y component

Branch coverage test sets: To achieve branch coverage, we need to make sure that every possible branch in the code is taken at least once
Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components, and none of them have the same x component
Test 6: p = vector with three or more points with the same y component, and some of them have the same x component
Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

Basic condition coverage test sets: To achieve basic condition coverage, we need to make sure that every basic condition in the code (i.e., every Boolean subexpression) is evaluated as both true and false at least once
Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component, and the first point has a smaller x component
Test 4: p = vector with two points with the same y component, and the second point has a smaller x component
Test 5: p = vector with two points with different y components
Test 6: p = vector with three or more points with different y components, and none of them have the same x component
Test 7: p = vector with three or more points with the same y component, and some of them have the same x component
Test 8: p = vector with three or more points with the same y component, and all of them have the same x component.