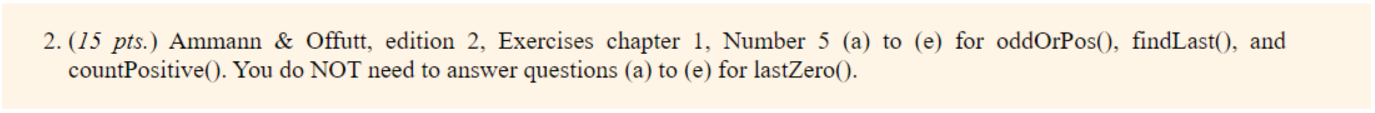
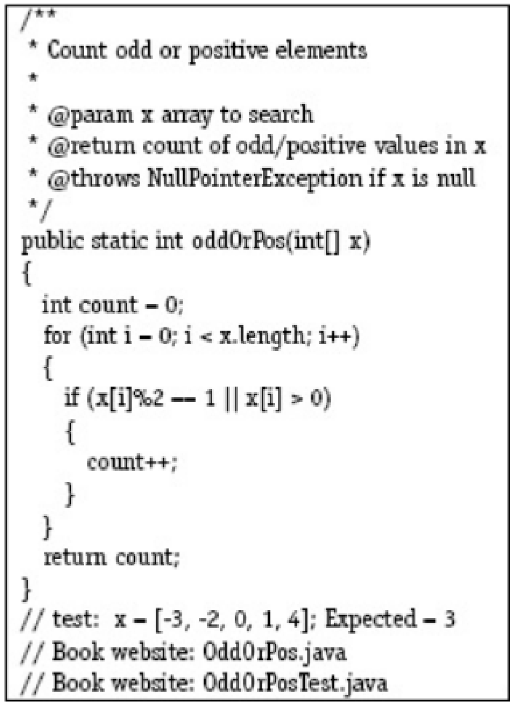
# Assignment 1

**1.**



2. (*15 pts.*) Ammann & Offutt, edition 2, Exercises chapter 1, Number 5 (a) to (e) for oddOrPos(), findLast(), and countPositive(). You do NOT need to answer questions (a) to (e) for lastZero().



```
/**
 * Count odd or positive elements
 *
 * @param x array to search
 * @return count of odd/positive values in x
 * @throws NullPointerException if x is null
 */
public static int oddOrPos(int[] x)
{
  int count = 0;
  for (int i = 0; i < x.length; i++)
  {
    if (x[i]%2 == 1 || x[i] > 0)
    {
      count++;
    }
  }
  return count;
}
// test:  x = [-3, -2, 0, 1, 4]; Expected = 3
// Book website: OddOrPos.java
// Book website: OddOrPosTest.java
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

In the given code, ‘if’ condition is not checking for the negative odd numbers. It is only checking for the positive reminder but not for the negative remainder -1. The fault is in the if condition. We can modify the code by adding the below condition, then it gives the actual output 3 instead of 2 for the input x = [-3,-2,0,1,4]

Code snippet:

```
If(x[i]%2 == -1 || x[i] >0){
    Count++;
}
```

(b) If possible, give a test case that does not execute the fault. If not, briefly explain why not.

For any input, the fault code will be executed, and the respective output will be shown except the case where the empty array with no values. When the array is empty or null, the fault code will not be executed as there are no elements in the array to process.

Test case: x = []

(c) If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.

Let us consider the test case x = [2,3,4,5,6], excepted o/p: 5 and actual o/p: 5. In this test case, the fault has been executed but does not result in an error state as there are only positive numbers in the array. In this case even if a fault is executed, the result is not affected.

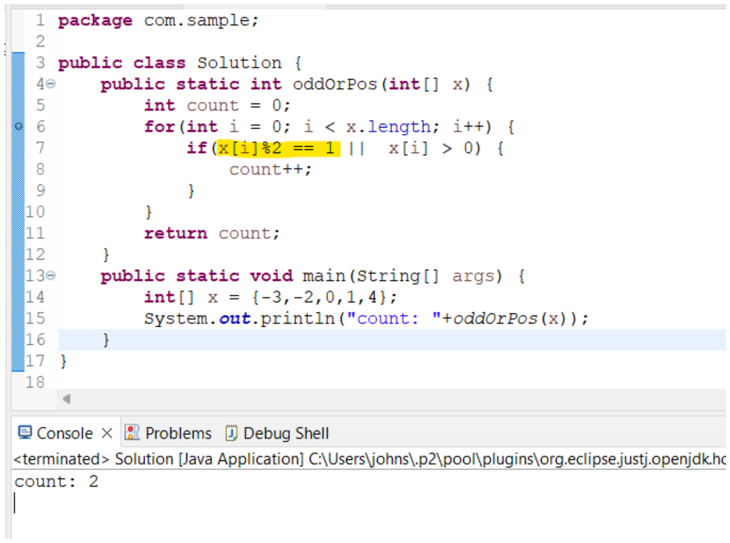
(d) If possible, give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Do not forget about the program counter.
Let us consider the test case x = [2,3,4,5,6] correctly evaluates to 5, while test case x = [-3, -2,0,1,4] incorrectly evaluates to 2. In both tests the faulty statement is executed. Although both tests result in an error, only the second results in failure.

For any software code there are 3 stages fault-> error (Internal state) -> failure (External state). If a code has a fault, it leads to an error, which results in a failure. For the given code for the inputs fault will be executed and if it contains a negative number it results in error, which leads to failure.

(e) For the given test case, describe the first error state. Be sure to describe the complete state.
For the given test case, we can see the first error in the if condition where x[] values are being checked. The error state is not being able to consider the negative value reminder.

I/p: x = [-3,-2,0,1,4]
Expected O/p: 3
Actual O/p: 2
Count = 0, i = 1, PC = “i<x.length”

The yellow marked statement is the error state.



```
package com.sample;

public class Solution {
    public static int oddOrPos(int[] x) {
        int count = 0;
        for(int i = 0; i < x.length; i++) {
            if(x[i]%2 == 1 ||  x[i] > 0) {
                count++;
            }
        }
        return count;
    }
    public static void main(String[] args) {
        int[] x = {-3,-2,0,1,4};
        System.out.println("count: "+oddOrPos(x));
    }
}
```

```
<terminated> Solution [Java Application] C:\Users\johns\.p2\pool\plugins\org.eclipse.justj.openjdk.hc
count: 2
```

findLast()

(a) The for loop terminates before reaching the beginning of the array. It will stop at index 1 instead of index 0. The logical condition could be rewritten like so:
Before: for (int i=x.length-1; i > 0; i--)
After: for (int i=x.length-1; i >= 0; i--)

(b) // test: x = null; y = 3; Expected = NullPointerException

A NullPointerException will be thrown when the initialization step of the for loop tries to access x.length.

The logical condition will never be evaluated.

(c) // test: x = [2, 3, 5]; y = 5; Expected = 2

(d) // test: x = [7]; y = 2; Expected = -1

The initialization part of the for loop will set i = 0. The logical condition (the fault) will be evaluated, and the for loop will not execute. An error is encountered (the entire array has not been checked), but it happens that 2 was never in the array to begin with, yielding the expected result.

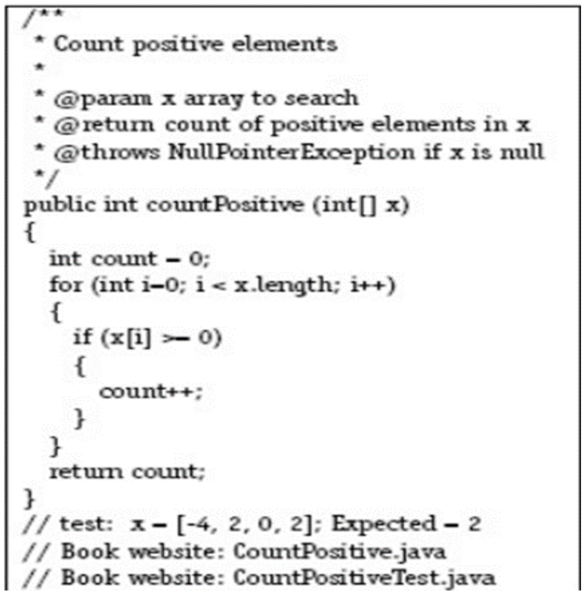
(e) The first error state is below:

|  |  |
|---|---|
| x: | [2, 3, 5] |
| y: | 2 |
| i: | 0 |
| PC: | return -1; |

i has just been decremented to 0, the logical condition i > 0 has been checked, and the program is exiting the for loop. What it *should* be doing is running the for loop one last time for i == 0.

**CountPositive:**

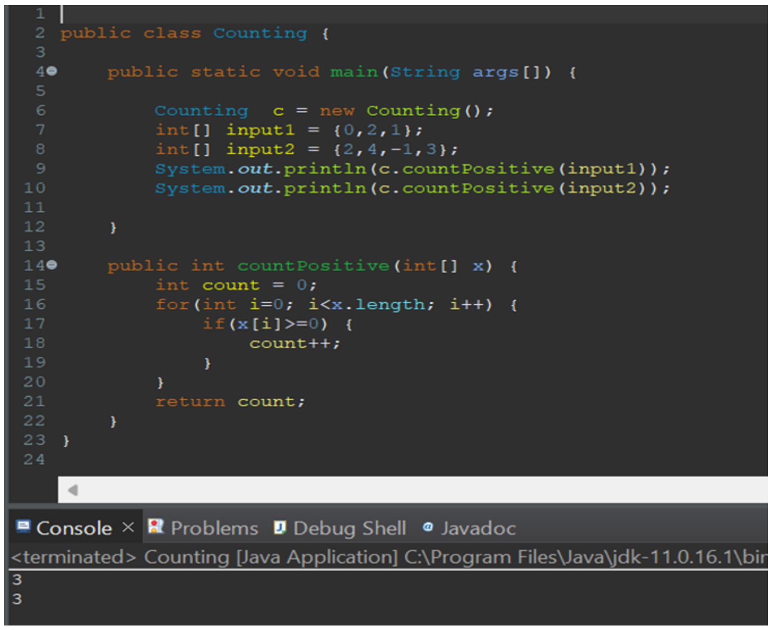
Given Code:



```
/**
 * Count positive elements
 *
 * @param x array to search
 * @return count of positive elements in x
 * @throws NullPointerException if x is null
 */
public int countPositive (int[] x)
{
   int count = 0;
   for (int i=0; i < x.length; i++)
   {
     if (x[i] >= 0)
     {
       count++;
     }
   }
   return count;
}
// test:  x = [-4, 2, 0, 2]; Expected = 2
// Book website: CountPositive.java
// Book website: CountPositiveTest.java
```

a. **Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.**

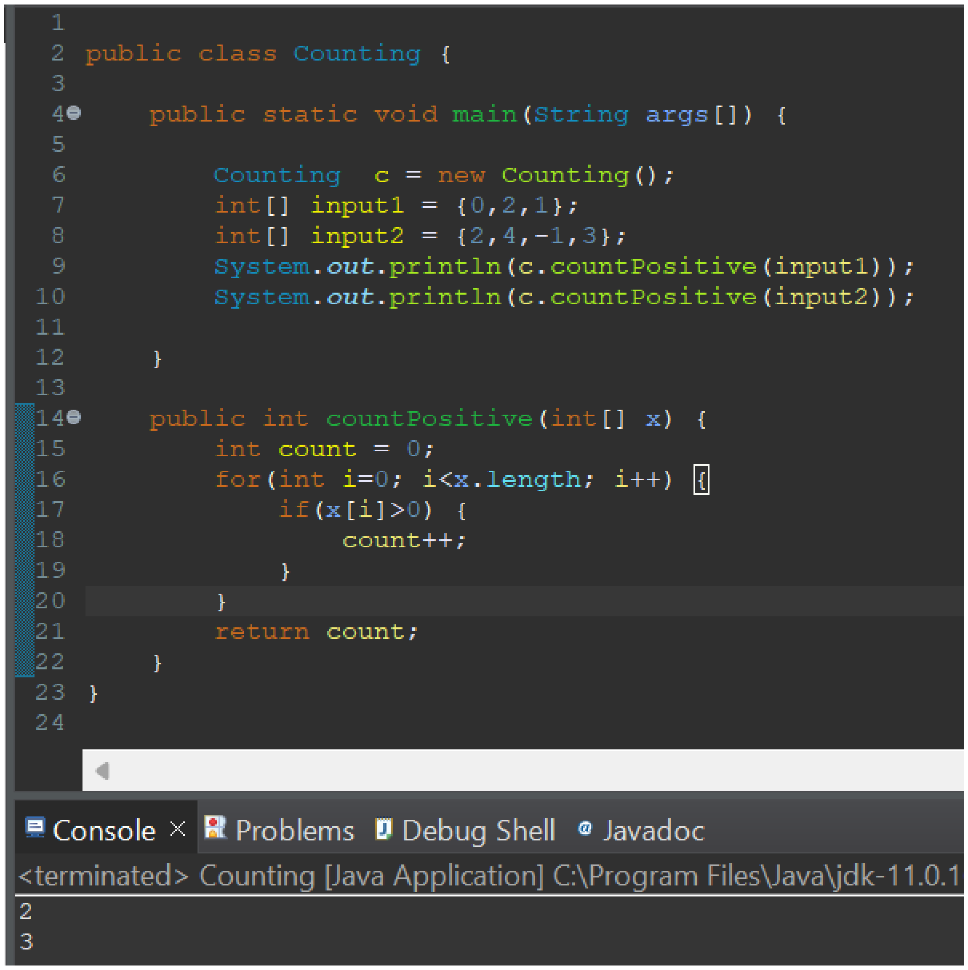
For the above code, below is the driver code I used to check its behavior:



```java

public class Counting {

    public static void main(String args[]) {

        Counting  c = new Counting();
        int[] input1 = {0,2,1};
        int[] input2 = {2,4,-1,3};
        System.out.println(c.countPositive(input1));
        System.out.println(c.countPositive(input2));

    }

    public int countPositive(int[] x) {
        int count = 0;
        for(int i=0; i<x.length; i++) {
            if(x[i]>=0) {
                count++;
            }
        }
        return count;
    }
}

```

```
<terminated> Counting [Java Application] C:\Program Files\Java\jdk-11.0.16.1\bir
3
3
```

So, the fault here is that for input1, the expected answer should have been 2 because 0 is not a positive integer. The following is the modified code that satisfies the code specifications:



```java

public class Counting {

    public static void main(String args[]) {

        Counting  c = new Counting();
        int[] input1 = {0,2,1};
        int[] input2 = {2,4,-1,3};
        System.out.println(c.countPositive(input1));
        System.out.println(c.countPositive(input2));

    }

    public int countPositive(int[] x) {
        int count = 0;
        for(int i=0; i<x.length; i++) {
            if(x[i]>0) {
                count++;
            }
        }
        return count;
    }
}

```

```
<terminated> Counting [Java Application] C:\Program Files\Java\jdk-11.0.1
2
3
```

So here, I have changed the if condition to not count 0 as a positive integer.

b. **If possible, give a test case that does not execute the fault. If not, briefly explain why not.**

All test cases execute faults, but the only case where it is not is when the array is either empty or null.

c. **If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.**

Again, as mentioned in my above driver code, any test case that does not contain a 0 executes the fault but does not result in an error state because it is not going to cause a failure. Example: Input1{2,4, -1,3}

d. **If possible, give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don't forget about the program counter.**

Let us consider the test case x = [2, 4, -1, 3] correctly evaluates to 3, while test case x = [0,2,1] incorrectly evaluates to 2. In both tests the faulty statement is executed. Only the second results in failure.

e. **For the given test case, describe the first error state. Be sure to describe the complete state.**

For input1(0,2,1), the expected result should be 2, but the obtained result is 3. So, this is the first error state. This happens because the count variable is incremented even for a zero, which is incorrect. The first error state is shown as below:

Input = [0,2,1]

Expected Answer = 2

Obtained Answer = 3

Increment variable, i = 0

Counter variable, count = 1

PC = i<x.length