SE 317, Lab 2

Name: Aina Qistina Azman

Net ID: 457 464 051

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

Method	Fault	Proposed Modification
findLast	The for loop's condition statement has an error. The condition is set to $i > 0$, which causes the first element of the array to be skipped in the comparison.	The condition statement should be adjusted to $i >= 0$ or $i > -1$.
lastZero	The purpose of the method is to find the last index of zero. But given the way the for loop were set up, it will be returning the first occurrence index of zero, not the last occurrence index of zero.	The for loop initialization statement condition statement and increment/decrement statement should be modified to: $for(i = x.length - 1; i >= 0; i--)$
countPositive	Positive numbers are numbers that is greater than zero. But the comparison in if statement includes 0.	The if statement should compare: $if(x[i] > 0)$
oddOrPos	The $x[i]\%2 == 1$ only evaluates for positive numbers. For negative odd numbers, the equation above will results in -1, and thus making negative odd numbers to not be included/counted as expected.	The if statement should compare: $if(Math.abs(x[i]\%2 == 1) \| \| x[i] > 0)$

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

1. **findLast**

```
@Test
public void findLast_doesntExecuteFault() {

    findLast finder = new findLast();

    int[] x = null;
    int y = 1;
    assertThrows(NullPointerException.class, () -> finder.findLast(x, y));
}
```

2. **lastZero**

```
@Test
public void lastZero_doesntExecuteFault() {

    lastZero findLastZero = new lastZero();

    int[] x = null;
    assertThrows(NullPointerException.class, ()-> findLastZero.lastZero(x));
}
```

3. **countPositive**

```
    @Test
    public void countPositive_doesntExecuteFault() {

        countPositive countP = new countPositive();
        int[] x = null;
        assertThrows(NullPointerException.class, ()-> count
P.countPositive(x));
    }
```

4. **oddOrPos**

```
@Test
    public void oddOrPos_doesntExecuteFault() {

        oddOrPos count = new oddOrPos();

        int[] x = null;
        assertThrows(NullPointerException.class, ()-> coun
t.oddOrPos(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.**

1. **findLast**

```
    @Test
    public void findLast_executeFaultNoError() {

        findLast finder = new findLast();

        int[] x = {1, 2, 3, 2, 4};
        int y = 2;
        assertEquals(3, finder.findLast(x, y));
    }
```

2. **lastZero**

```
    @Test
    public void lastZero_executeFaultNoError() {

        lastZero findLastZero = new lastZero();

        int[] x = {1, 2, 4, 5, 0};
        assertEquals(4, findLastZero.lastZero(x));

    }
```

3. **countPositive**

```
@Test
    public void countPositive_executeFaultNoError() {

        countPositive countP = new countPositive();

        int[] x = {1, 2, 3, 9, -2, -3};
        assertEquals(4, countP.countPositive(x));
    }
```

4. **oddOrPos**

```
    @Test
    public void oddOrPos_executeFaultNoError() {

        oddOrPos count = new oddOrPos();

        int[] x = {1, 2, 3, -2, 5};
        assertEquals(4, count.oddOrPos(x));
    }
```

(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.**

1. **findLast**

```
    @Test
    public void findLast_errorNoFailure() {

        findLast finder = new findLast();

        int[] x = {4, 2, 3, 4, 4};
        int y = 4;
        assertEquals(4, finder.findLast(x, y));

    }
```

2. **lastZero**

```
@Test
public void lastZero_errorNoFailure() {

        lastZero findLastZero = new lastZero();

        int[] x = {0, 2, 4, 5, 0};
        assertEquals(0, findLastZero.lastZero(x));

    }
```

3. **countPositive**

```
    @Test
    public void countPositive_errorNoFailure() {

    countPositive countP = new countPositive();

        int[] x = {0, 2, 3, 9, -2, -3};
        assertEquals(4, countP.countPositive(x));

    }
```

4. **oddOrPos**

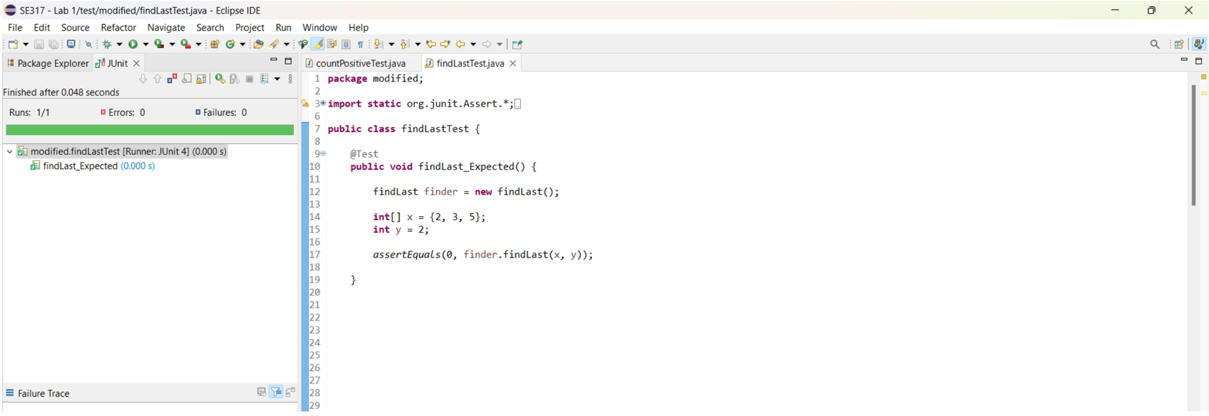
```
@Test
    public void oddOrPos_errorNoFailure() {

        oddOrPos count = new oddOrPos();

        int[] x =  {1, 2, 3, -3, 5, -7};
        assertEquals(4, count.oddOrPos(x));
    }
```

(e) **Implement your repair and verify that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.**

1. **findLast**

```
@Test
    public void findLast_Expected() {

        findLast finder = new findLast();

        int[] x = {2, 3, 5};
        int y = 2;

        assertEquals(0, finder.findLast(x, y));

    }
```

Evidence for modified findLast:


SE317 - Lab 1/test/modified/findLastTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
countPositiveTest.java
findLastTest.java
Failure Trace

Finished after 0.048 seconds

Runs: 1/1 Errors: 0 Failures: 0

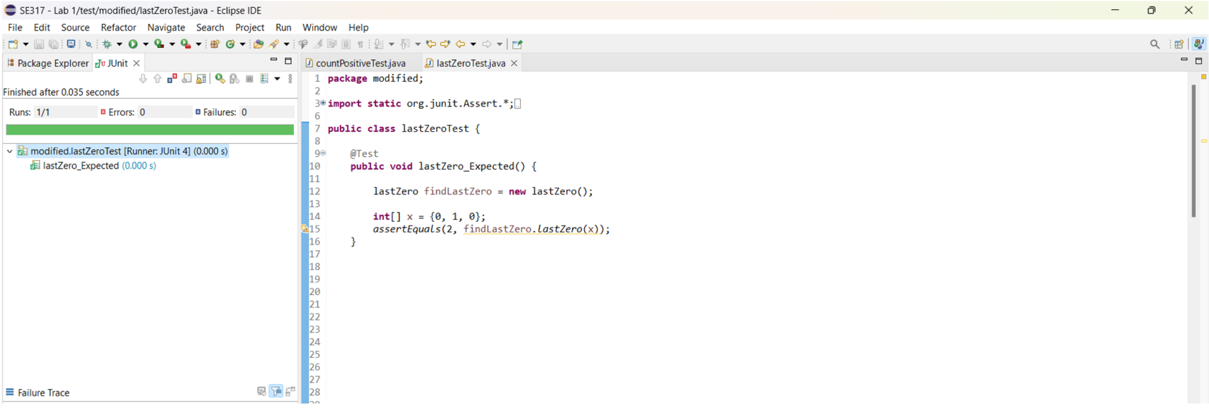
modified.findLastTest [Runner: JUnit 4] (0.000 s)
findLast_Expected (0.000 s)

```
package modified;

import static org.junit.Assert.*;

public class findLastTest {

    @Test
    public void findLast_Expected() {

        findLast finder = new findLast();

        int[] x = {2, 3, 5};
        int y = 2;

        assertEquals(0, finder.findLast(x, y));

    }
```

2. **lastZero**

```
    @Test
    public void lastZero_Expected() {

        lastZero findLastZero = new lastZero();

        int[] x = {0, 1, 0};
        assertEquals(2, findLastZero.lastZero(x));
    }
```

Evidence for modified lastZero:


SE317 - Lab 1/test/modified/lastZeroTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
countPositiveTest.java
lastZeroTest.java
Failure Trace

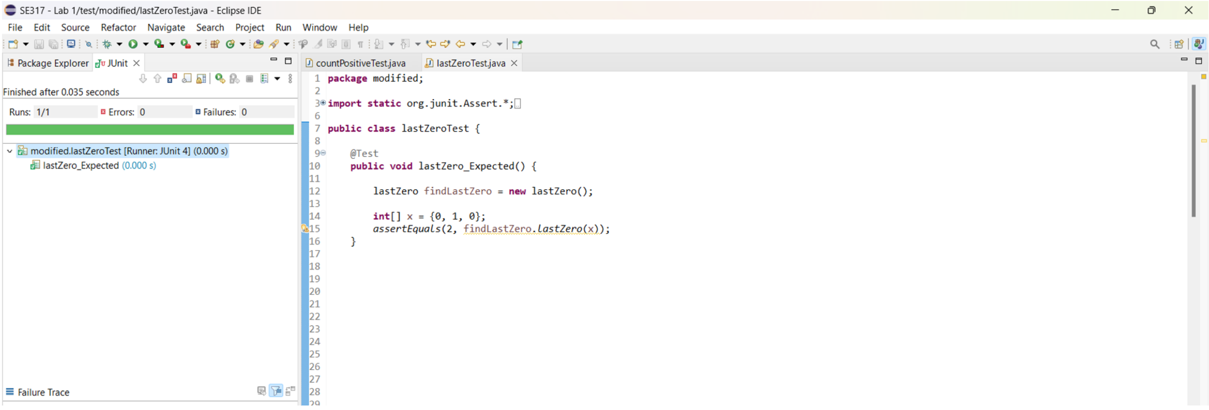
```
Finished after 0.035 seconds
Runs: 1/1    Errors: 0    Failures: 0
modified.lastZeroTest [Runner: JUnit 4] (0.000 s)
    lastZero_Expected (0.000 s)
```

```java
package modified;

import static org.junit.Assert.*;

public class lastZeroTest {

    @Test
    public void lastZero_Expected() {

        lastZero findLastZero = new lastZero();

        int[] x = {0, 1, 0};
        assertEquals(2, findLastZero.lastZero(x));
    }
```

3. countPositive

```java
@Test
public void countPositive_Expected() {

    countPositive countP = new countPositive();

    int[] x = {-4, 2, 0, 2};
    assertEquals(2, countP.countPositive(x));
```

Evidence for modified countPositive:


SE317 - Lab 1/test/modified/lastZeroTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
countPositiveTest.java
lastZeroTest.java
Failure Trace

Finished after 0.035 seconds
Runs: 1/1   Errors: 0   Failures: 0
modified.lastZeroTest [Runner: JUnit 4] (0.000 s)
lastZero_Expected (0.000 s)

package modified;

import static org.junit.Assert.*;

public class lastZeroTest {

@Test
public void lastZero_Expected() {

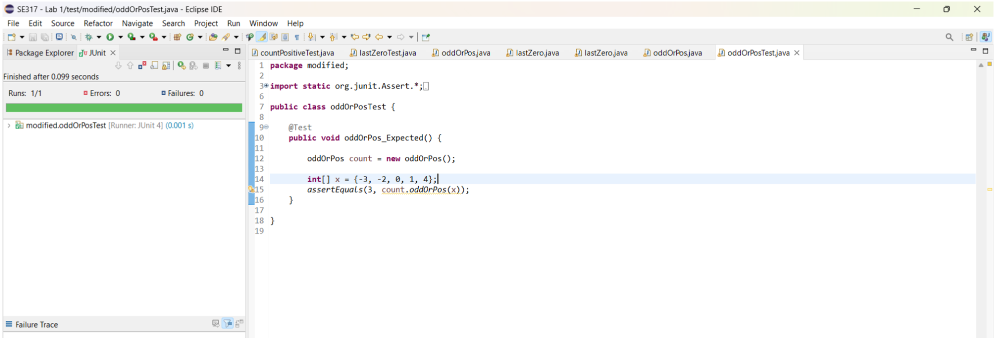
lastZero findLastZero = new lastZero();

int[] x = {0, 1, 0};
assertEquals(2, findLastZero.lastZero(x));
}

4. oddOrPos

```java
@Test
public void oddOrPos_Expected() {
    oddOrPos count = new oddOrPos();

    int[] x = {-3, -2, 0, 1, 4};
    assertEquals(3, count.oddOrPos(x));
}
```

Evidence for modified oddOrPos:



```
Finished after 0.099 seconds
Runs: 1/1    Errors: 0    Failures: 0
modified.oddOrPosTest [Runner: JUnit 4] (0.001 s)
```

```
package modified;

import static org.junit.Assert.*;

public class oddOrPosTest {

    @Test
    public void oddOrPos_Expected() {

        oddOrPos count = new oddOrPos();

        int[] x = {-3, -2, 0, 1, 4};
        assertEquals(3, count.oddOrPos(x));
    }

}
```