

4a

1. FindLast.java

a. Whats Wrong

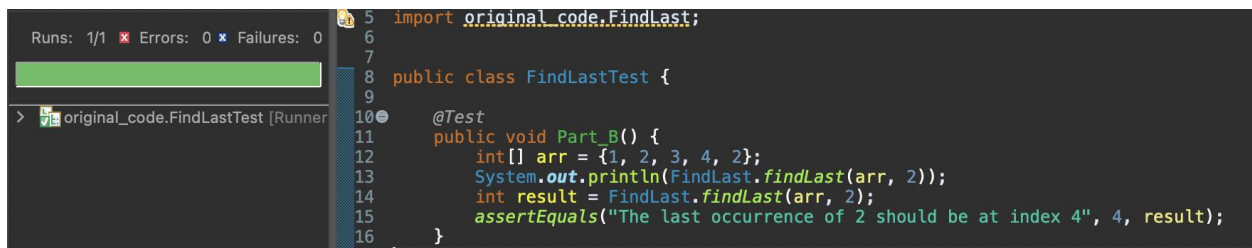
Fault: The loop uses the condition $i > 0$, which misses checking the element at index 0.

Proposed Modification: Change the loop condition to $i \geq 0$ so that index 0 is included.

```
18         for (int i = x.length - 1; i >= 0; i--) //now include index 0
```

b. Test Case, Does Not Execute the Fault

Uses an array where the searched value is not only at index 0.

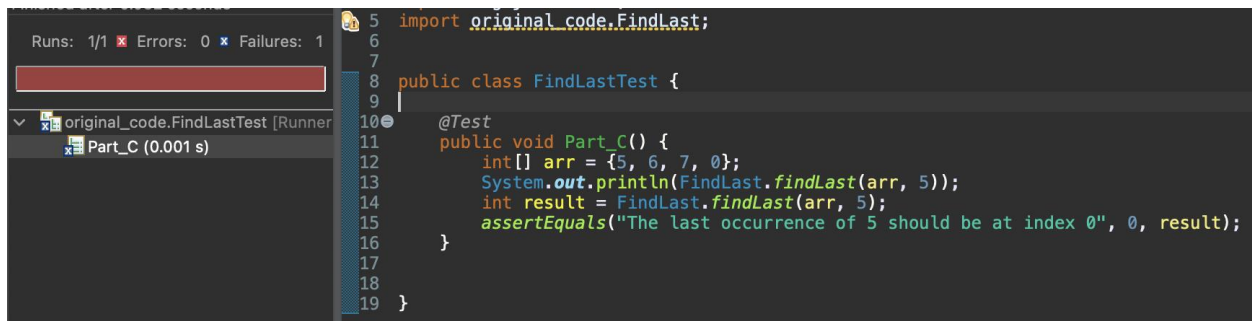


The screenshot shows an IDE with a test runner on the left and code on the right. The test runner shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The code on the right is for a test case named 'Part_B' which uses an array {1, 2, 3, 4, 2} and searches for the value 2. The assertion passes, indicating the fault was not executed.

```
5 import original_code.FindLast;
6
7
8 public class FindLastTest {
9
10     @Test
11     public void Part_B() {
12         int[] arr = {1, 2, 3, 4, 2};
13         System.out.println(FindLast.findLast(arr, 2));
14         int result = FindLast.findLast(arr, 2);
15         assertEquals("The last occurrence of 2 should be at index 4", 4, result);
16     }
17 }
```

c. Test Case, Executes the Fault Without an Error

Uses an array where the only occurrence is at index 0.



The screenshot shows an IDE with a test runner on the left and code on the right. The test runner shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 1'. The code on the right is for a test case named 'Part_C' which uses an array {5, 6, 7, 0} and searches for the value 5. The assertion fails because the last occurrence of 5 is at index 0, not index 0 as expected. This indicates the fault was executed.

```
5 import original_code.FindLast;
6
7
8 public class FindLastTest {
9
10     @Test
11     public void Part_C() {
12         int[] arr = {5, 6, 7, 0};
13         System.out.println(FindLast.findLast(arr, 5));
14         int result = FindLast.findLast(arr, 5);
15         assertEquals("The last occurrence of 5 should be at index 0", 0, result);
16     }
17
18
19 }
```

d. Test Case, Results an Error

The only way I was able to produce an error was by providing the code with something it wasn't designed to handle. In this case, and in every other Part D, I passed "null" to the array. Since the code wasn't expecting it, it resulted in an error.

```

3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import original_code.FindLast;
6
7
8 public class FindLastTest {
9
10     @Test
11     public void Part_D() {
12         int[] arr = null;
13         System.out.println(FindLast.findLast(arr, 1));
14         int result = FindLast.findLast(arr, 1);
15         assertEquals("when pass null, the method should return -1", -1, result);
16     }
17
18 }

```

e. Implement the Repair and Verify

As seen in the SS below we implemented the fix for the “for loop” and added a “if statement” so handle the “null” so it can pass all test cases

```

16 public static int findLast (int[] x, int y)
17 {
18     if (x==null)
19         return -1;
20
21     for (int i = x.length - 1; i >= 0; i--) //now include index 0
22     {
23         if (x[i] == y)
24         {
25             return i;
26         }
27     }
28     return -1;
29 }

```

```

1 package modified_code;
2
3 import static org.junit.Assert.*;
4
5
6 public class FindLastTest {
7
8     @Test
9     public void Part_B() {
10         int[] arr = {1, 2, 3, 4, 2};
11         System.out.println(FindLast.findLast(arr, 2));
12         int result = FindLast.findLast(arr, 2);
13         assertEquals("The last occurrence of 2 should be at index 4", 4, result);
14     }
15
16     @Test
17     public void Part_C() {
18         int[] arr = {5, 6, 7, 0};
19         System.out.println(FindLast.findLast(arr, 5));
20         int result = FindLast.findLast(arr, 5);
21         assertEquals("The last occurrence of 5 should be at index 0", 0, result);
22     }
23
24     @Test
25     public void Part_D() {
26         int[] arr = null;
27         System.out.println(FindLast.findLast(arr, 1));
28         int result = FindLast.findLast(arr, 1);
29         assertEquals("when pass null, the method should return -1", -1, result);
30     }
31
32 }

```

2. LastZero.java

a. Whats Wrong

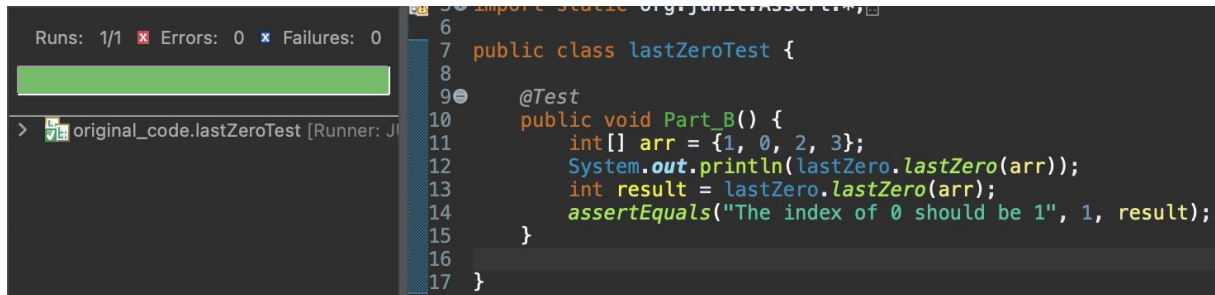
Fault: The method iterates from the beginning, returning the first zero rather than the last.

Proposed Modification: Reverse the loop to start from the end of the array

```
19     for (int i = x.length - 1; i >= 0; i--) //now iterate from the end of the array
```

b. Test Case, Does Not Execute the Fault

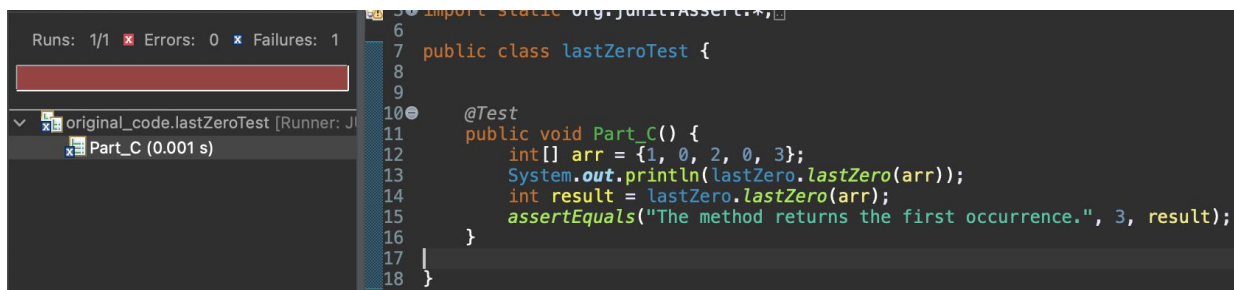
When there is only one 0 in the array, the first zero is also the last zero.



```
6 import static org.junit.Assert.*;
7 public class lastZeroTest {
8
9     @Test
10    public void Part_B() {
11        int[] arr = {1, 0, 2, 3};
12        System.out.println(lastZero.lastZero(arr));
13        int result = lastZero.lastZero(arr);
14        assertEquals("The index of 0 should be 1", 1, result);
15    }
16
17 }
```

c. Test Case, Executes the Fault Without an Error

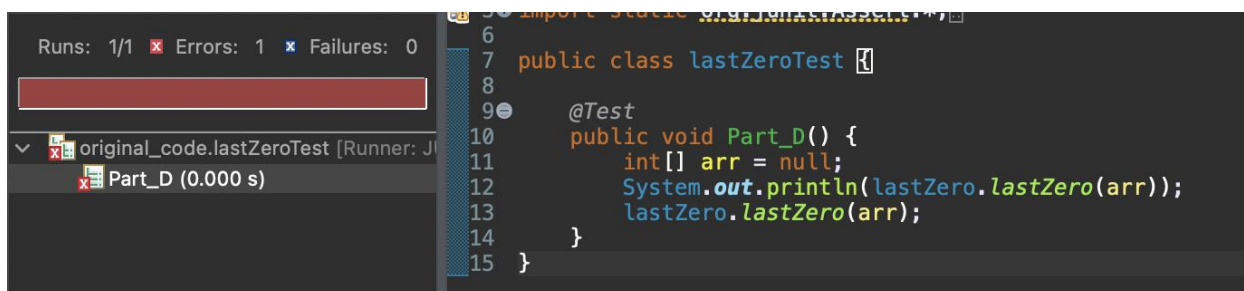
When the array contains multiple zeros, the faulty code returns the first zero's index instead of the last.



```
6 import static org.junit.Assert.*;
7 public class lastZeroTest {
8
9
10    @Test
11    public void Part_C() {
12        int[] arr = {1, 0, 2, 0, 3};
13        System.out.println(lastZero.lastZero(arr));
14        int result = lastZero.lastZero(arr);
15        assertEquals("The method returns the first occurrence.", 3, result);
16    }
17
18 }
```

d. Test Case, Results an Error

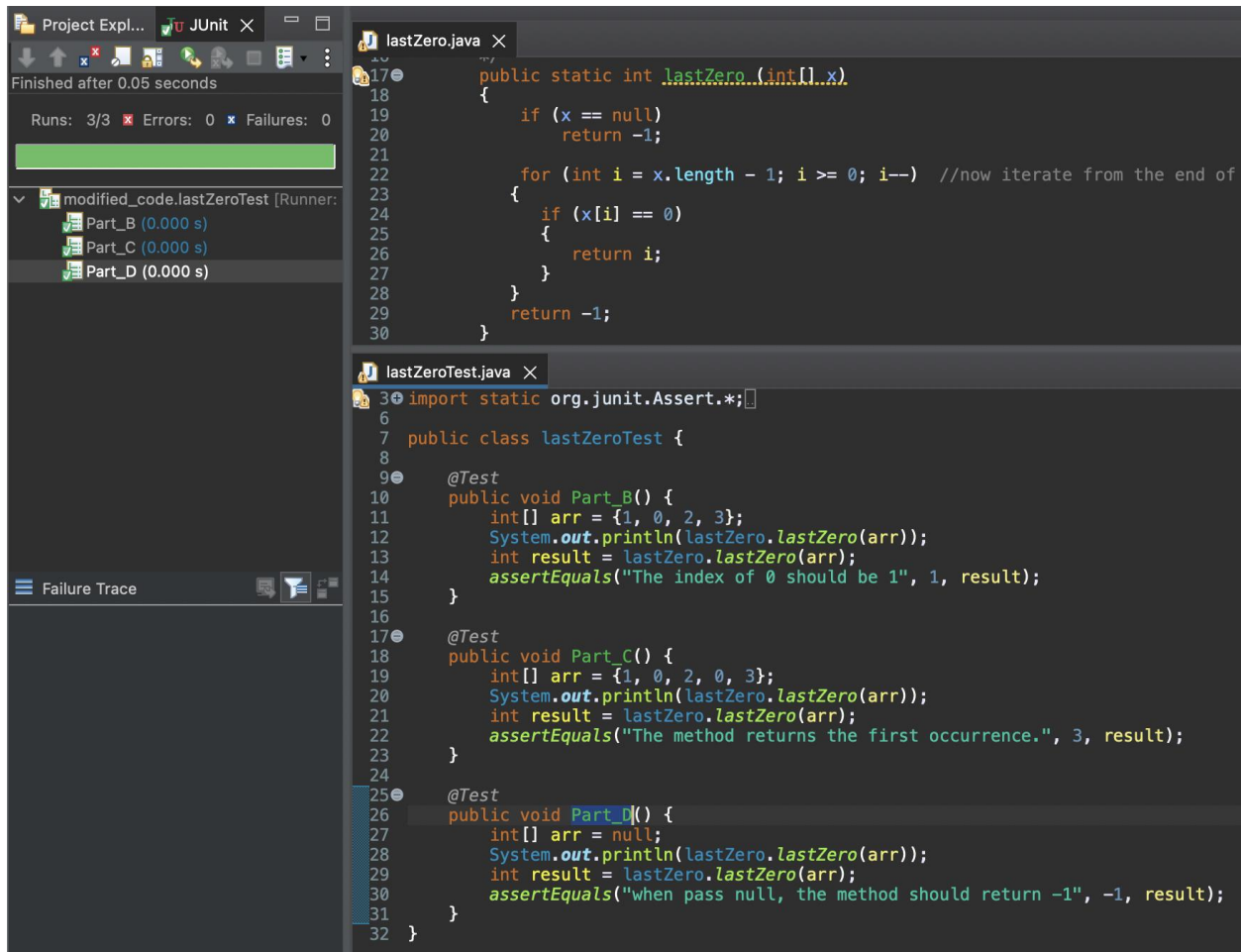
Passing a null array causes the error like I explained in FindLast Part D



```
6 import static org.junit.Assert.*;
7 public class lastZeroTest {
8
9
10    @Test
11    public void Part_D() {
12        int[] arr = null;
13        System.out.println(lastZero.lastZero(arr));
14        lastZero.lastZero(arr);
15    }
16 }
```

e. Implement the Repair and Verify

As seen in the SS below we implemented the fix for the “for loop” and added a “if statement” so handle the “null” so it can pass all test cases



3. CountPositive

a. Whats Wrong

Fault: The condition `if (x[i] >= 0)` erroneously counts zero as positive.

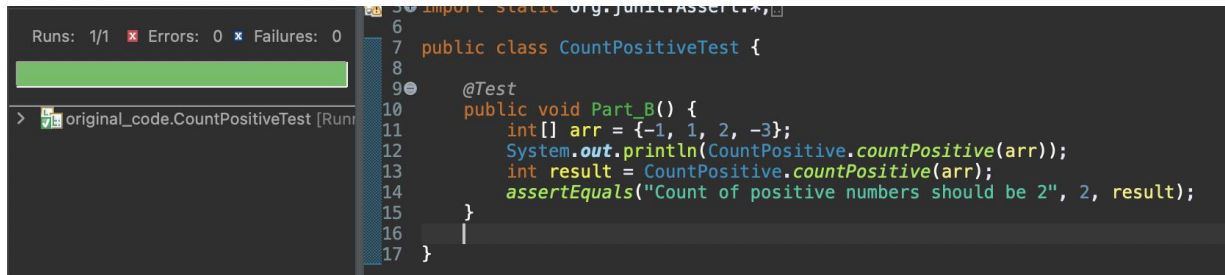
Proposed Modification: Change the condition to `if (x[i] > 0)` so that only numbers greater than

zero are counted.

```
18         if (x[i] > 0) //now count only strictly positive numbers
```

b. Test Case, Does Not Execute the Fault

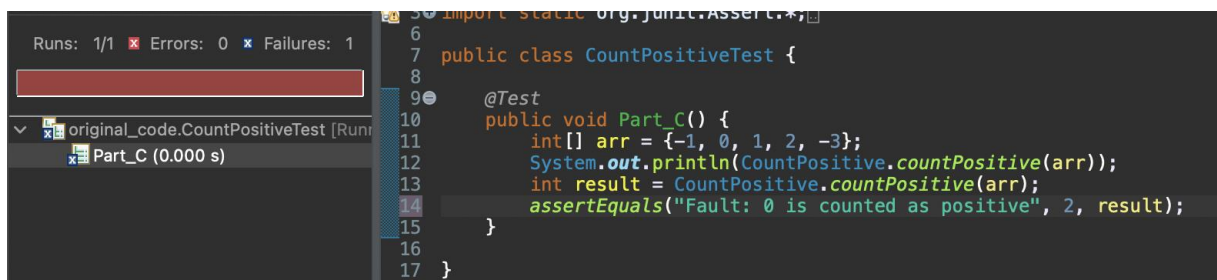
Choose an array without any 0. Then the faulty code and the intended behavior coincide.



```
50 import static org.junit.Assert.*;
6
7 public class CountPositiveTest {
8
9     @Test
10    public void Part_B() {
11        int[] arr = {-1, 1, 2, -3};
12        System.out.println(CountPositive.countPositive(arr));
13        int result = CountPositive.countPositive(arr);
14        assertEquals("Count of positive numbers should be 2", 2, result);
15    }
16
17 }
```

c. Test Case, Executes the Fault Without an Error

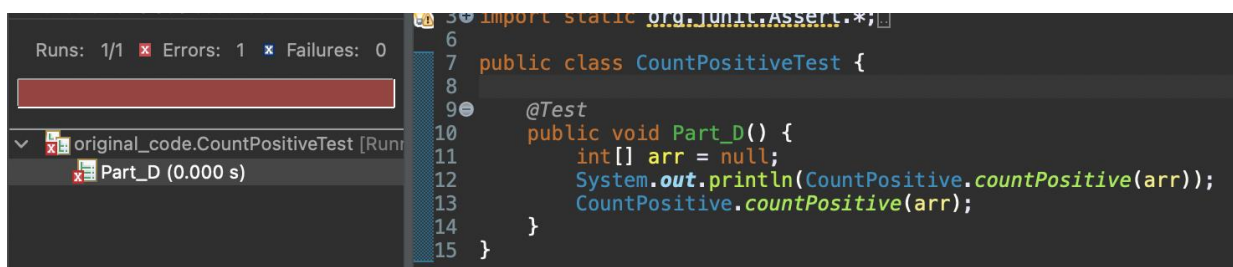
Include 0 in the array. The faulty code counts 0 as positive.



```
50 import static org.junit.Assert.*;
6
7 public class CountPositiveTest {
8
9     @Test
10    public void Part_C() {
11        int[] arr = {-1, 0, 1, 2, -3};
12        System.out.println(CountPositive.countPositive(arr));
13        int result = CountPositive.countPositive(arr);
14        assertEquals("Fault: 0 is counted as positive", 2, result);
15    }
16
17 }
```

d. Test Case, Results an Error

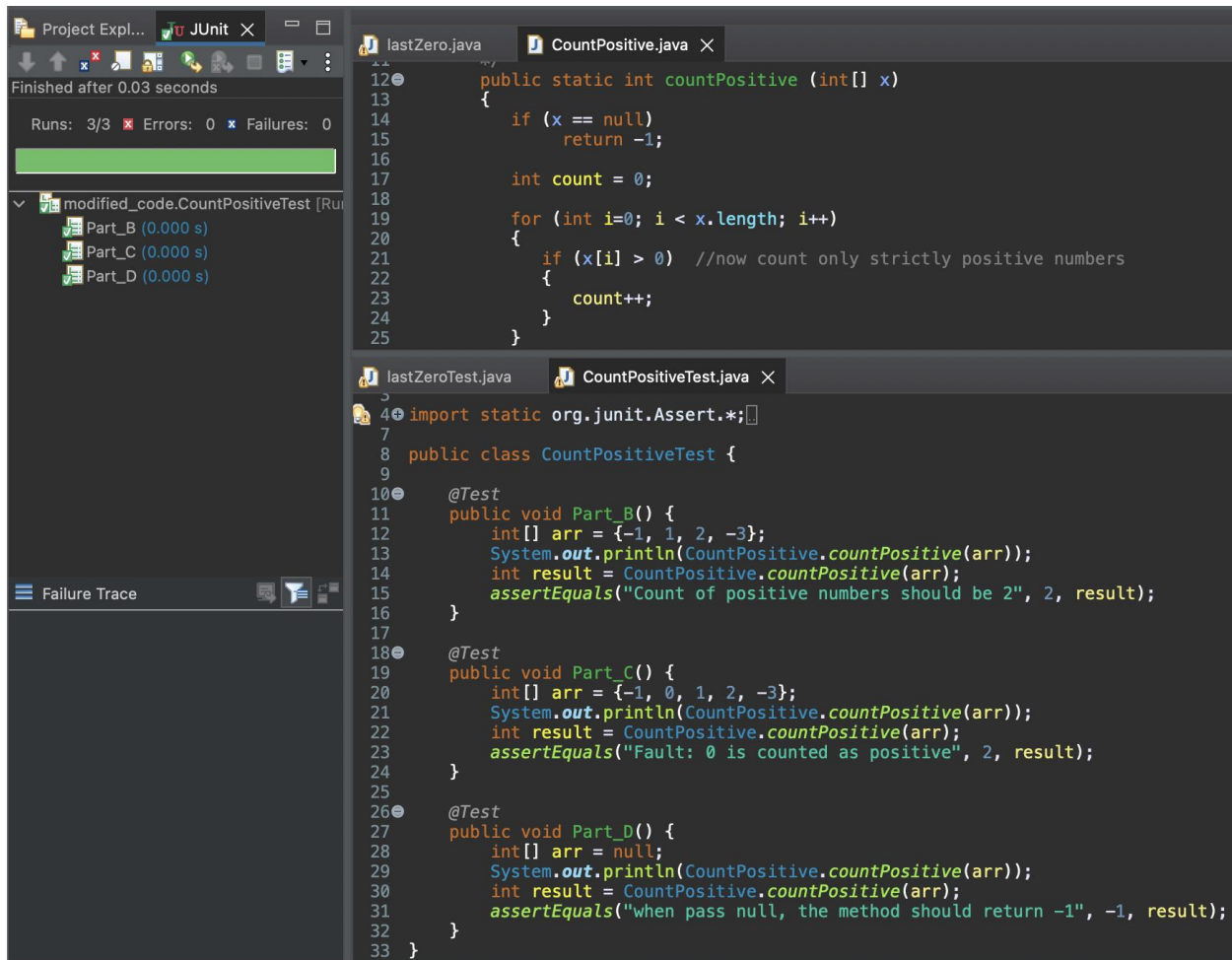
Passing a null array causes the error like I explained in FindLast Part D



```
50 import static org.junit.Assert.*;
6
7 public class CountPositiveTest {
8
9     @Test
10    public void Part_D() {
11        int[] arr = null;
12        System.out.println(CountPositive.countPositive(arr));
13        CountPositive.countPositive(arr);
14    }
15 }
```

e. Implement the Repair and Verify

As seen in the SS below we implemented the fix for the “if statement regarding positive numbers” and added a “if statement” so handle the “null” so it can pass all test cases



4. OddOrPos

a. Whats Wrong

Fault: The original condition uses `if (x[i] % 2 != 1 || x[i] > 0)`, which fails to correctly identify odd numbers (especially negative odd numbers) because for a negative odd (like -3), `-3 % 2` yields -1.

Proposed Modification: Change the condition to use `if (x[i] % 2 != 0 || x[i] > 0)` so that any

number with a nonzero remainder when divided by 2 (i.e., any odd number) is counted.

```
21         if (x[i]%2 != 0 || x[i] > 0) // now detect odd numbers regardless of their sign
```

b. Test Case, Does Not Execute the Fault

Choose an array that avoids negative odd numbers. For instance, if all odd numbers are positive, the faulty condition may not be triggered.

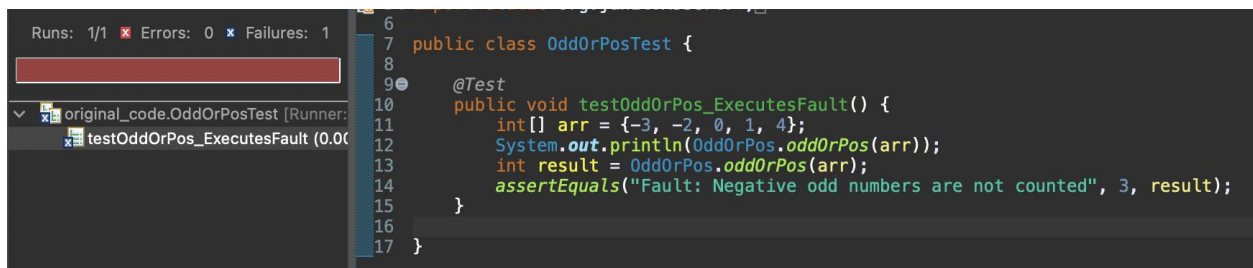


The screenshot shows an IDE with a test runner on the left and code on the right. The test runner indicates 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A green progress bar is shown. The code on the right is for a test class 'OddOrPosTest' with a method 'testOddOrPos_NoFault()' that uses an array of positive numbers {2, 4, 1, 6} and asserts that the count is 4.

```
6 import static org.junit.Assert.*;
7 public class OddOrPosTest {
8
9     @Test
10    public void testOddOrPos_NoFault() {
11        int[] arr = {2, 4, 1, 6};
12        System.out.println(OddOrPos.oddOrPos(arr));
13        int result = OddOrPos.oddOrPos(arr);
14        assertEquals("The count should be 4", 4, result);
15    }
16
17 }
```

c. Test Case, Executes the Fault Without an Error

Use an array that includes a negative odd number. In the faulty code, negative odd numbers might not be detected correctly.



The screenshot shows the same IDE with a different test case. The test runner indicates 'Runs: 1/1', 'Errors: 0', and 'Failures: 1'. A red progress bar is shown. The code on the right is for a test class 'OddOrPosTest' with a method 'testOddOrPos_ExecutesFault()' that uses an array containing a negative odd number {-3, -2, 0, 1, 4} and asserts that the count is 3.

```
6
7 public class OddOrPosTest {
8
9     @Test
10    public void testOddOrPos_ExecutesFault() {
11        int[] arr = {-3, -2, 0, 1, 4};
12        System.out.println(OddOrPos.oddOrPos(arr));
13        int result = OddOrPos.oddOrPos(arr);
14        assertEquals("Fault: Negative odd numbers are not counted", 3, result);
15    }
16
17 }
```

d. Test Case, Results an Error

Passing a null array causes the error like I explained in FindLast Part D

```

3  import static org.junit.Assert.*;
6
7  public class OddOrPosTest {
8
9      @Test
10     public void testOddOrPos_Error() {
11         int[] arr = null;
12         System.out.println(OddOrPos.oddOrPos(arr));
13         OddOrPos.oddOrPos(arr);
14     }
15 }

```

Runs: 1/1 Errors: 1 Failures: 0

original_code.OddOrPosTest [Runner: ...]

testOddOrPos_Error (0.000 s)

e. Implement the Repair and Verify

As seen in the SS below we implemented the fix for the “if statement regarding numbers sign” and added a “if statement” so handle the “null” so it can pass all test cases

```

16 // are either odd or positive (or both)
17 if (x == null)
18     return -1;
19
20 int count = 0;
21
22 for (int i = 0; i < x.length; i++)
23 {
24     if (x[i]%2 != 0 || x[i] > 0) // now detect odd numbers regardless of sign
25     {
26         count++;
27     }
28 }
29 return count;
30 }

```

```

3  import static org.junit.Assert.*;
6
7  public class OddOrPosTest {
8
9      @Test
10     public void testOddOrPos_NoFault() {
11         int[] arr = {2, 4, 1, 6};
12         System.out.println(OddOrPos.oddOrPos(arr));
13         int result = OddOrPos.oddOrPos(arr);
14         assertEquals("The count should be 4", 4, result);
15     }
16
17     @Test
18     public void testOddOrPos_ExecutesFault() {
19         int[] arr = {-3, -2, 0, 1, 4};
20         System.out.println(OddOrPos.oddOrPos(arr));
21         int result = OddOrPos.oddOrPos(arr);
22         assertEquals("Fault: Negative odd numbers are not counted", 3, result);
23     }
24
25     @Test
26     public void testOddOrPos_Error() {
27         int[] arr = null;
28         System.out.println(OddOrPos.oddOrPos(arr));
29         int result = OddOrPos.oddOrPos(arr);
30         assertEquals("when pass null, the method should return -1", -1, result);
31     }
32 }
33

```

Project Expl... JUnit X

Finished after 0.025 seconds

Runs: 3/3 Errors: 0 Failures: 0

modified_code.OddOrPosTest [Runner: ...]

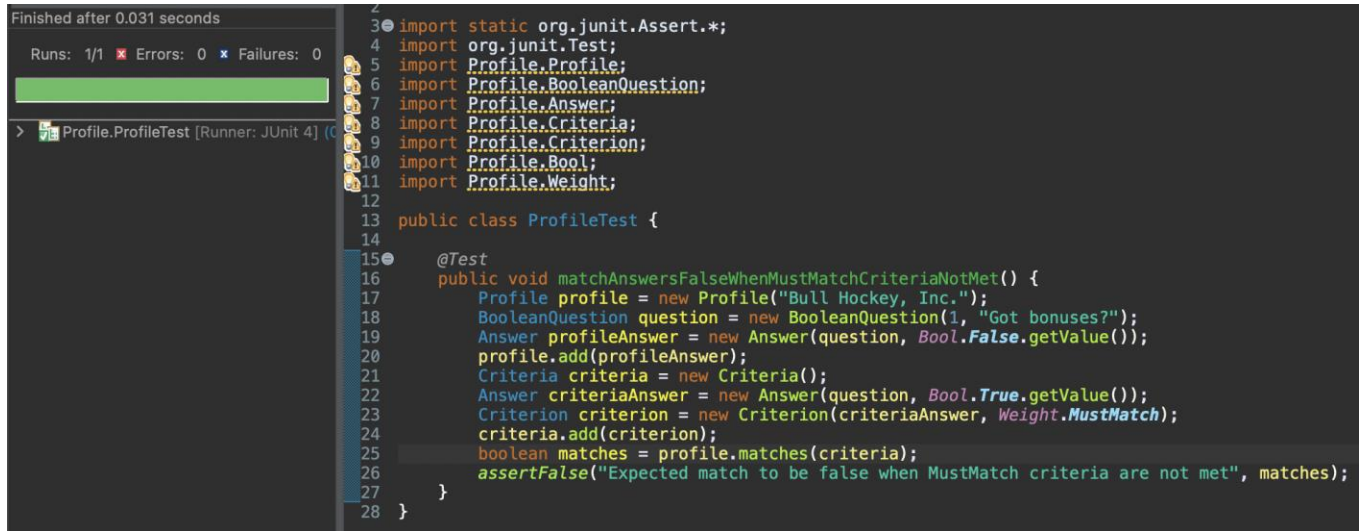
testOddOrPos_ExecutesFault (0.000 s)

testOddOrPos_Error (0.000 s)

testOddOrPos_NoFault (0.000 s)

Failure Trace

4b



The screenshot shows an IDE with a test run on the left and source code on the right. The test run panel shows 'Finished after 0.031 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A green progress bar is visible. The test is 'Profile.ProfileTest [Runner: JUnit 4]'. The source code on the right is a Java file with the following content:

```
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import Profile.Profile;
6 import Profile.BooleanQuestion;
7 import Profile.Answer;
8 import Profile.Criteria;
9 import Profile.Criterion;
10 import Profile.Bool;
11 import Profile.Weight;
12
13 public class ProfileTest {
14
15     @Test
16     public void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
17         Profile profile = new Profile("Bull Hockey, Inc.");
18         BooleanQuestion question = new BooleanQuestion(1, "Got bonuses?");
19         Answer profileAnswer = new Answer(question, Bool.False.getValue());
20         profile.add(profileAnswer);
21         Criteria criteria = new Criteria();
22         Answer criteriaAnswer = new Answer(question, Bool.True.getValue());
23         Criterion criterion = new Criterion(criteriaAnswer, Weight.MustMatch);
24         criteria.add(criterion);
25         boolean matches = profile.matches(criteria);
26         assertFalse("Expected match to be false when MustMatch criteria are not met", matches);
27     }
28 }
```

Runs: 1/1 Errors: 0 Failures: 0

Profile.ProfileTest [Runner: JUnit 4] (C)

```
30 import static org.junit.Assert.*;
4  import org.junit.Test;
5  import Profile.Profile;
6  import Profile.BooleanQuestion;
7  import Profile.Answer;
8  import Profile.Criteria;
9  import Profile.Criterion;
10 import Profile.Bool;
11 import Profile.Weight;
12
13 public class ProfileTest {
14
15     @Test
16     public void matchAnswersTrueForAnyDontCareCriteria() {
17         Profile profile = new Profile("Bull Hockey, Inc.");
18         BooleanQuestion question = new BooleanQuestion(1, "Got milk?");
19         Answer profileAnswer = new Answer(question, Bool.False.getValue());
20         profile.add(profileAnswer);
21         Criteria criteria = new Criteria();
22         Answer criteriaAnswer = new Answer(question, Bool.True.getValue());
23         Criterion criterion = new Criterion(criteriaAnswer, Weight.DontCare);
24         criteria.add(criterion);
25         boolean matches = profile.matches(criteria);
26         assertTrue("Expected match to be true when using DontCare criteria", matches);
27     }
28
29 }
```

Runs: 2/2 Errors: 0 Failures: 0

> Profile.ProfileTest [Runner: JUnit 4] (C)

```
30 import static org.junit.Assert.*;
4  import org.junit.Before;
5  import org.junit.Test;
6  import Profile.Profile;
7  import Profile.BooleanQuestion;
8  import Profile.Answer;
9  import Profile.Criteria;
10 import Profile.Criterion;
11 import Profile.Bool;
12 import Profile.Weight;
13
14 public class ProfileTest {
15
16     private Profile profile;
17     private BooleanQuestion question;
18     private Criteria criteria;
19
20     @Before
21     public void create() {
22         profile = new Profile("Bull Hockey, Inc.");
23         question = new BooleanQuestion(1, "Got bonuses?");
24         criteria = new Criteria();
25     }
26
27     @Test
28     public void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
29         profile.add(new Answer(question, Bool.False.getValue()));
30         criteria.add(new Criterion(new Answer(question, Bool.True.getValue()), Weight.MustMatch));
31
32         boolean matches = profile.matches(criteria);
33         assertFalse("Expected match to be false when MustMatch criteria not met", matches);
34     }
35
36     @Test
37     public void matchAnswersTrueForAnyDontCareCriteria() {
38         profile.add(new Answer(question, Bool.False.getValue()));
39         criteria.add(new Criterion(new Answer(question, Bool.True.getValue()), Weight.DontCare));
40
41         boolean matches = profile.matches(criteria);
42         assertTrue("Expected match to be true for DontCare criteria", matches);
43     }
44 }
```

Failure Trace

Question 1:

If JUnit runs the test method `matchAnswersTrueForAnyDontCareCriteria()` first, the following may happen to my understanding:

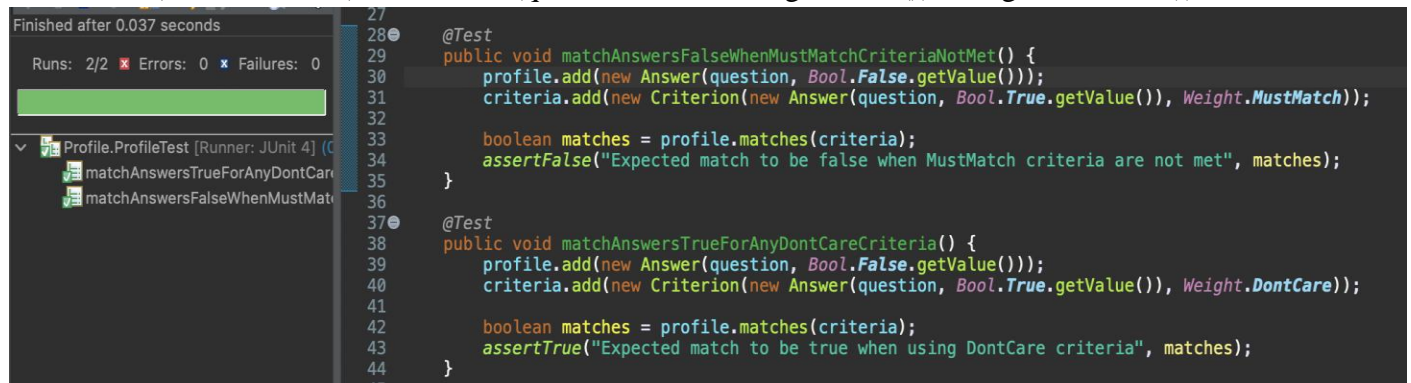
1. Test Instance Creation:
JUnit creates a new instance of the `ProfileTest` class.
2. @Before Method Execution:
Before any test method runs, the method annotated with `@Before` is executed. This method:
 - a. Instantiates a new `Profile` object.
 - b. Creates a new `BooleanQuestion`.
 - c. Initializes a new `Criteria` object.These objects are fresh and local to this test instance.
3. Test Method Execution:
The `matchAnswersTrueForAnyDontCareCriteria()` method is executed. Within this method:
 - a. An Answer is added to the `Profile`.
 - b. A Criterion is created and added to the `Criteria`.
 - c. The `Profile`'s `matches(criteria)` method is called, and the test asserts that the result is true.
4. Test Teardown:
Once the test method completes, the `ProfileTest` instance used for this test is discarded. No static or shared state persists between tests.
5. Next Test Instance Creation:
When the next test is run, a new instance of `ProfileTest` is created, and the `@Before` method is executed again to provide a clean slate.

Question 2:

To reduce the impact of one test on another and avoid unnecessary local variables, we can inline the objects directly into method calls. This makes the arrange portion concise and ensures that each test is self-contained.

```
profile.add(new Answer(question, Bool.False.getValue()));
```

```
criteria.add(new Criterion(new Answer(question, Bool.True.getValue()), Weight.DontCare));
```

The screenshot shows an IDE with a test runner on the left and Java code on the right. The test runner shows 'Finished after 0.037 seconds', 'Runs: 2/2', 'Errors: 0', and 'Failures: 0'. It lists two tests: 'Profile.ProfileTest [Runner: JUnit 4] (C)' and 'matchAnswersFalseWhenMustMatchCriteriaNotMet'. The code on the right is a Java class with two test methods. The first method, 'matchAnswersFalseWhenMustMatchCriteriaNotMet', adds a criterion with 'Weight.MustMatch' and asserts that the match is false. The second method, 'matchAnswersTrueForAnyDontCareCriteria', adds a criterion with 'Weight.DontCare' and asserts that the match is true.

```
27
28
29 @Test
30 public void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
31     profile.add(new Answer(question, Bool.False.getValue()));
32     criteria.add(new Criterion(new Answer(question, Bool.True.getValue()), Weight.MustMatch));
33
34     boolean matches = profile.matches(criteria);
35     assertFalse("Expected match to be false when MustMatch criteria are not met", matches);
36 }
37
38 @Test
39 public void matchAnswersTrueForAnyDontCareCriteria() {
40     profile.add(new Answer(question, Bool.False.getValue()));
41     criteria.add(new Criterion(new Answer(question, Bool.True.getValue()), Weight.DontCare));
42
43     boolean matches = profile.matches(criteria);
44     assertTrue("Expected match to be true when using DontCare criteria", matches);
45 }
```

Question 3:

The length limitation of the algorithm is determined by two main factors:

1. Exponential Growth:

The total number of combinations for a given length n is 26^n .

This quickly increases the number of combinations, which in turn affects both memory usage and processing time.

2. System Resources and Practical Limits:

- Memory:** The system must have enough memory to store 26^n strings. For larger values of n , this may become infeasible.
- Time Complexity:** The time required to generate all combinations increases exponentially with n . Even if memory is sufficient, the time to compute all combinations might be prohibitive.

In practice, algorithms generating all possible combinations are used for relatively small values of n (less than 5) because of these exponential limitations.

ChatGPT problem

```
part_A.java X part_C.java part_E.java
1 package ChatGPT_part2;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class part_A {
7
8     // Generate all possible two-letter combinations
9     public static List<String> generateTwoLetterCombinations() {
10         List<String> combinations = new ArrayList<>();
11         for (char first = 'a'; first <= 'z'; first++)
12             for (char second = 'a'; second <= 'z'; second++)
13                 combinations.add("" + first + second);
14     }
15
16     return combinations;
17 }
18 }

part_B.java X part_D.java part_F.java
1 package ChatGPT_part2Test;
2
3 import static org.junit.Assert.*;
4
5 public class part_B {
6
7     @Test
8     public void testTwoLetterCombinations() {
9         List<String> twoLetter = part_A.generateTwoLetterCombinations();
10         assertEquals("Expected 676 two-letter combinations", 676, twoLetter.size());
11         assertEquals("First combination should be 'aa'", "aa", twoLetter.get(0));
12         assertEquals("Last combination should be 'zz'", "zz", twoLetter.get(twoLetter.size() - 1));
13     }
14 }
15 }
```

```
part_A.java X part_C.java part_E.java
1 package ChatGPT_part2;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class part_A {
7
8     // Generate all possible two-letter combinations
9     public static List<String> generateTwoLetterCombinations() {
10         List<String> combinations = new ArrayList<>();
11         for (char first = 'a'; first <= 'z'; first++)
12             for (char second = 'a'; second <= 'z'; second++)
13                 combinations.add("" + first + second);
14     }
15
16     return combinations;
17 }
18 }

part_B.java part_D.java X part_F.java
1 package ChatGPT_part2Test;
2
3 import static org.junit.Assert.*;
4 import java.util.List;
5 import org.junit.Test;
6 import ChatGPT_part2.part_C;
7
8 public class part_D {
9
10     @Test
11     public void testThreeLetterCombinations() {
12         List<String> threeLetter = part_C.generateThreeLetterCombinations();
13         assertEquals("Expected 17576 three-letter combinations", 17576, threeLetter.size());
14         assertEquals("First combination should be 'aaa'", "aaa", threeLetter.get(0));
15         assertEquals("Last combination should be 'zzz'", "zzz", threeLetter.get(threeLetter.size() - 1));
16     }
17 }
18 }
```

```
part_A.java X part_C.java part_E.java
1 package ChatGPT_part2;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class part_A {
7
8     // Generate all possible two-letter combinations
9     public static List<String> generateTwoLetterCombinations() {
10         List<String> combinations = new ArrayList<>();
11         for (char first = 'a'; first <= 'z'; first++)
12             for (char second = 'a'; second <= 'z'; second++)
13                 combinations.add("" + first + second);
14     }
15
16     return combinations;
17 }
18 }

part_B.java part_D.java part_F.java X
1 package ChatGPT_part2Test;
2
3 import static org.junit.Assert.*;
4 import java.util.List;
5 import org.junit.Test;
6 import ChatGPT_part2.part_E;
7
8 public class part_F {
9
10     @Test
11     public void testGeneralizedLetterCombinations() {
12         List<String> oneLetter = part_E.generateLetterCombinations(1);
13         assertEquals("Expected 26 one-letter combinations", 26, oneLetter.size());
14         assertEquals("First should be 'a'", "a", oneLetter.get(0));
15         assertEquals("Last should be 'z'", "z", oneLetter.get(oneLetter.size() - 1));
16
17         List<String> twoLetter = part_E.generateLetterCombinations(2);
18         assertEquals("Expected 676 two-letter combinations", 676, twoLetter.size());
19         assertEquals("First should be 'aa'", "aa", twoLetter.get(0));
20         assertEquals("Last should be 'zz'", "zz", twoLetter.get(twoLetter.size() - 1));
21
22         List<String> threeLetter = part_E.generateLetterCombinations(3);
23         assertEquals("Expected 17576 three-letter combinations", 17576, threeLetter.size());
24         assertEquals("First should be 'aaa'", "aaa", threeLetter.get(0));
25         assertEquals("Last should be 'zzz'", "zzz", threeLetter.get(threeLetter.size() - 1));
26     }
27 }
28 }
```