Erroll Barker 564842330

4a

1. FindLast.java

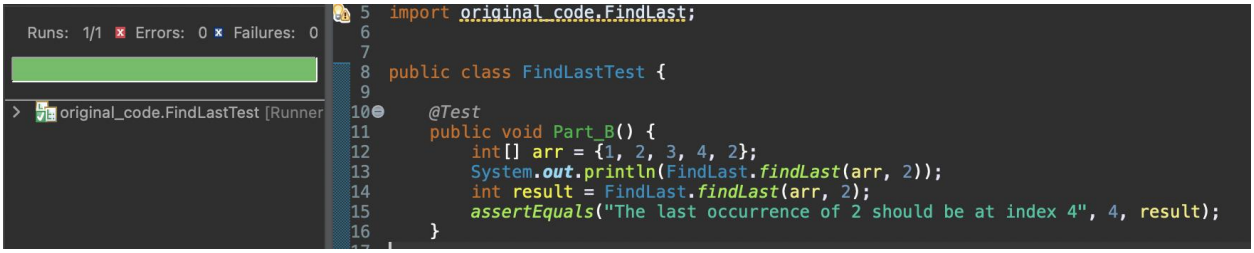
a. Whats Wrong

Fault: The loop uses the condition i > 0, which misses checking the element at index 0.
Proposed Modification: Change the loop condition to i >= 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.

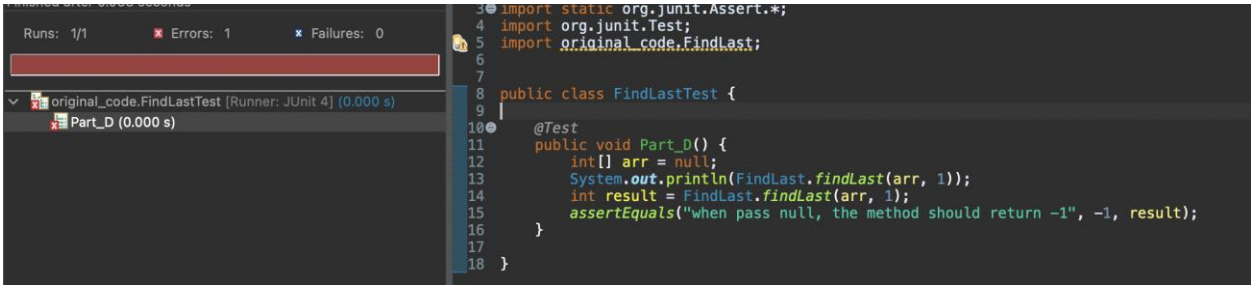


c. Test Case, Executes the Fault Without an Error

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

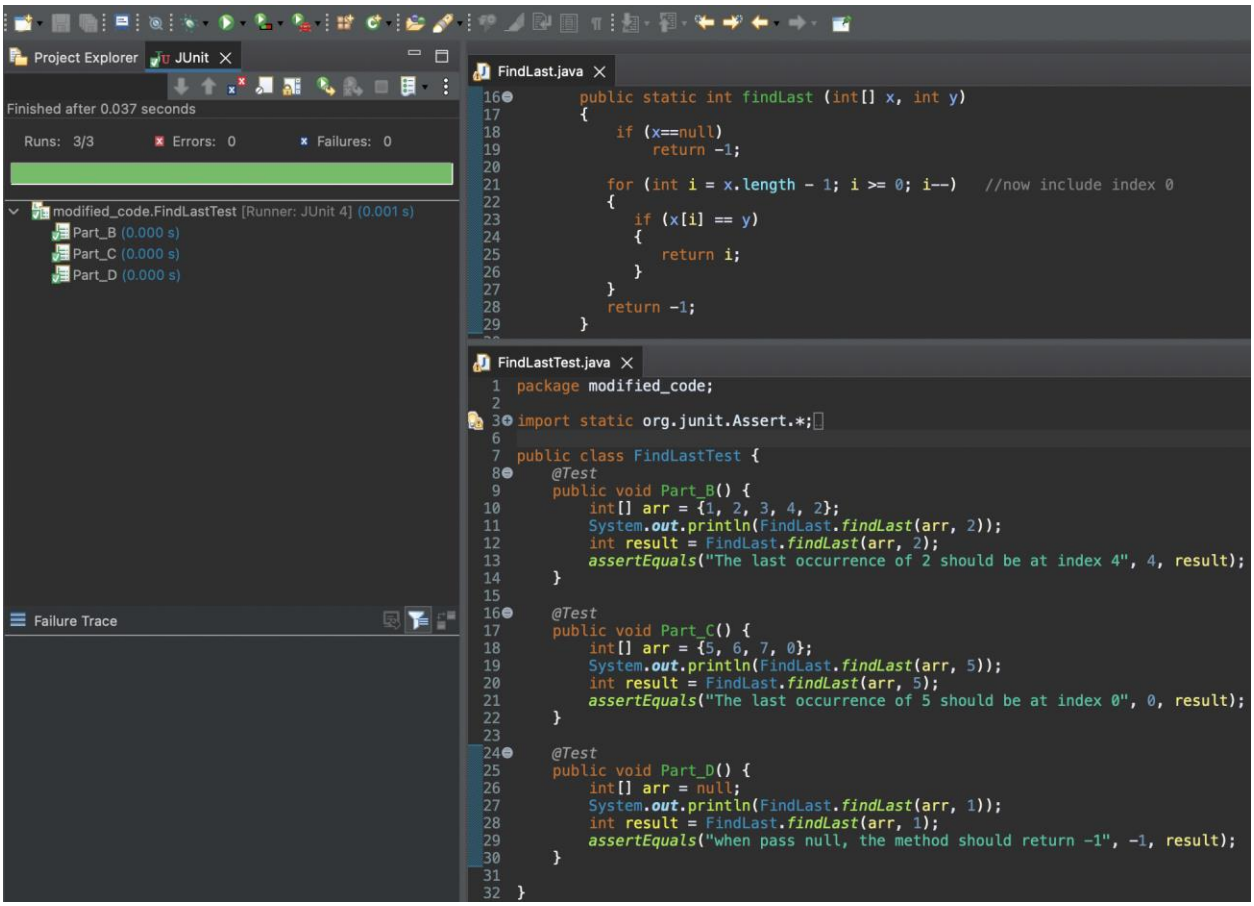
d. Test Case, Results an Error

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.



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



2. LastZero.java

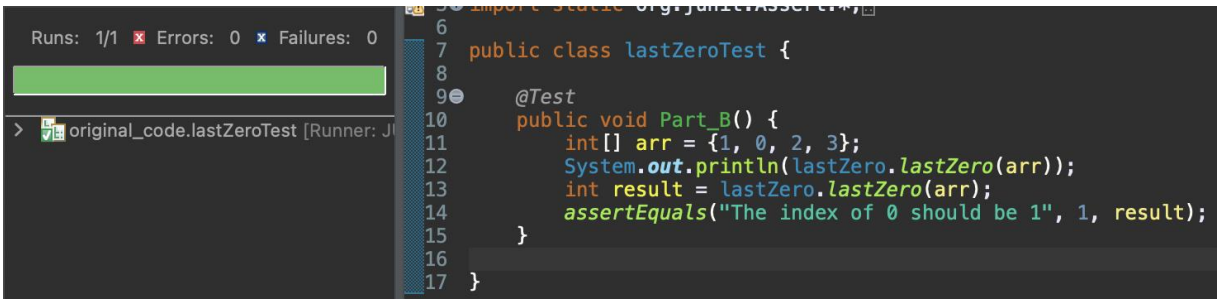
a. Whats Wrong

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
```

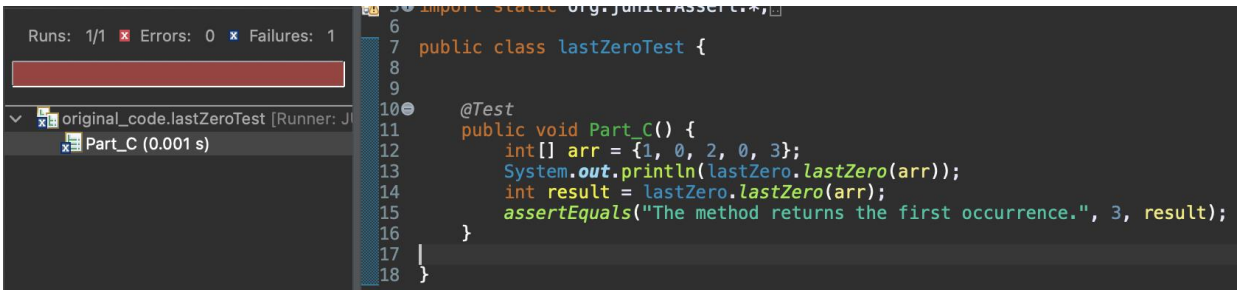
b. Test Case, Does Not Execute the Fault

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



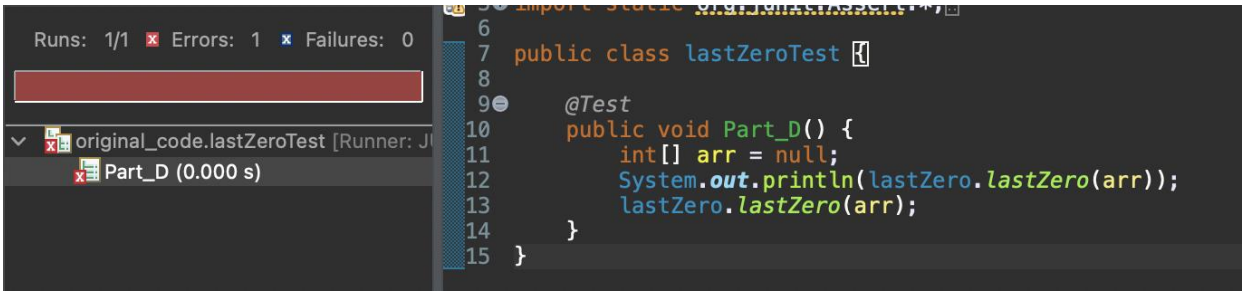
c. Test Case, Executes the Fault Without an Error

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



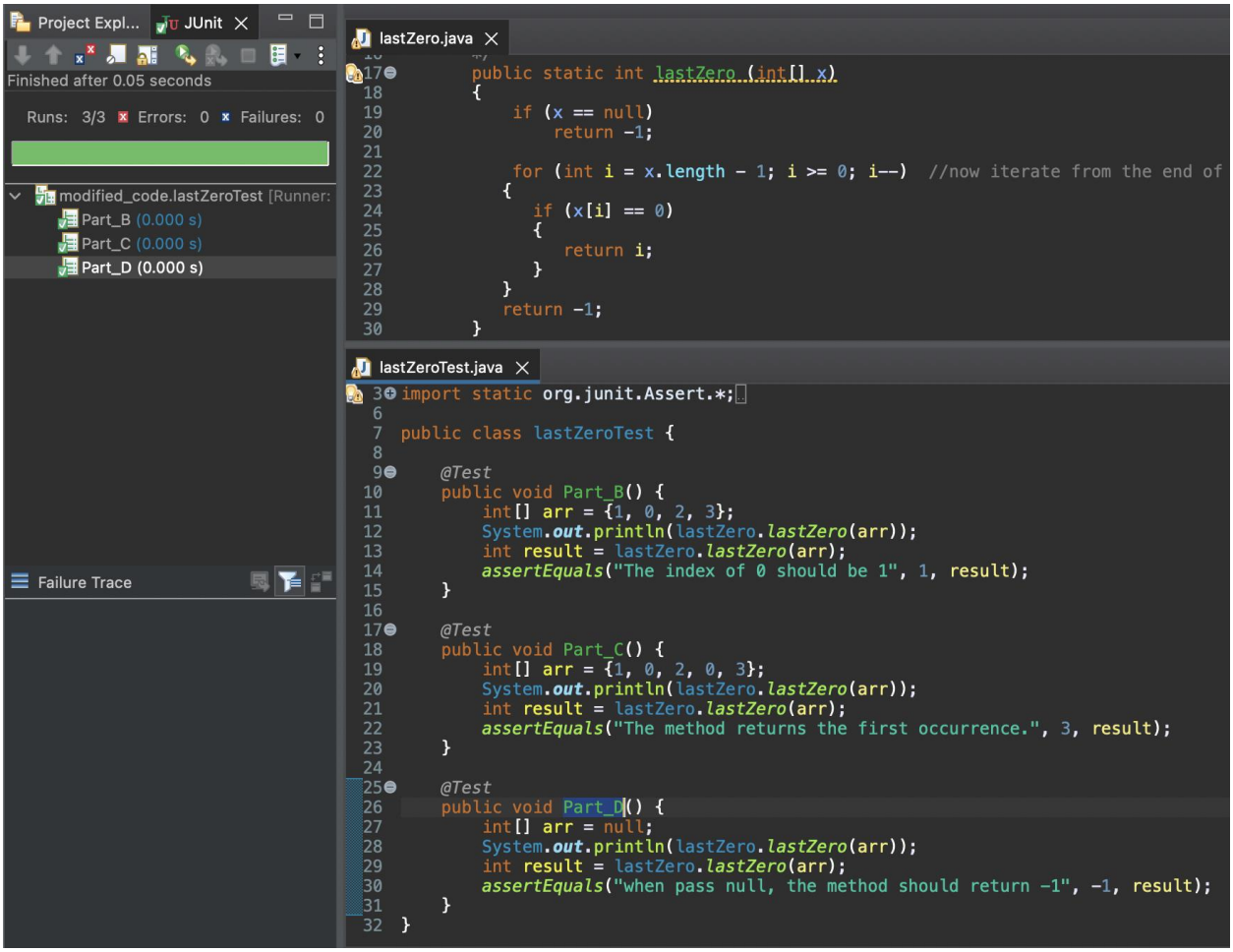
d. Test Case, Results an Error

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



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

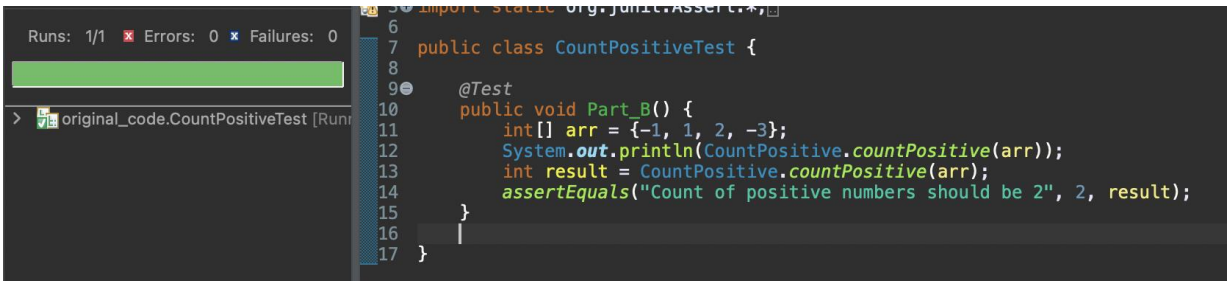
Fault: The condition if (x[i] >= 0) erroneously counts zero as positive.
Proposed Modification: Change the condition to if (x[i] > 0) so that only numbers greater than

zero are counted.

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

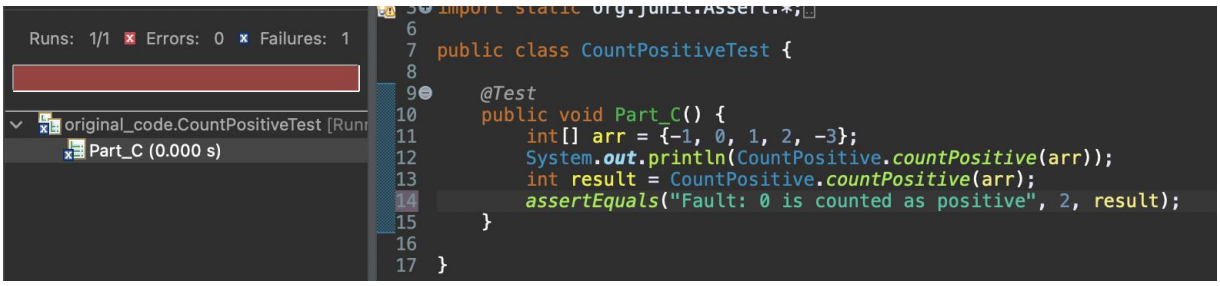
b. Test Case, Does Not Execute the Fault

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



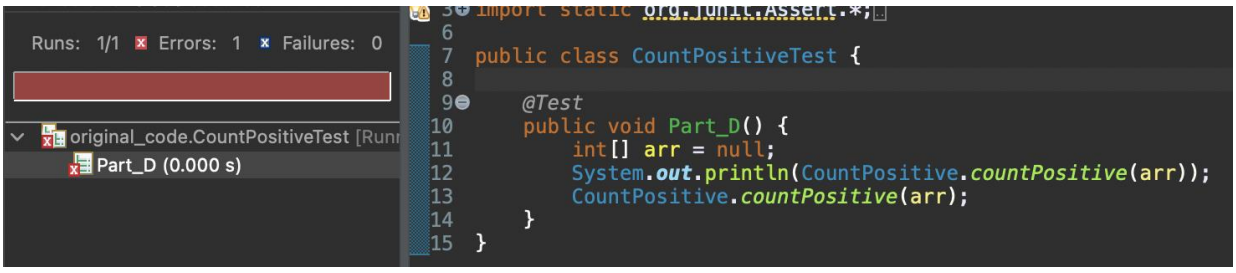
c. Test Case, Executes the Fault Without an Error

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



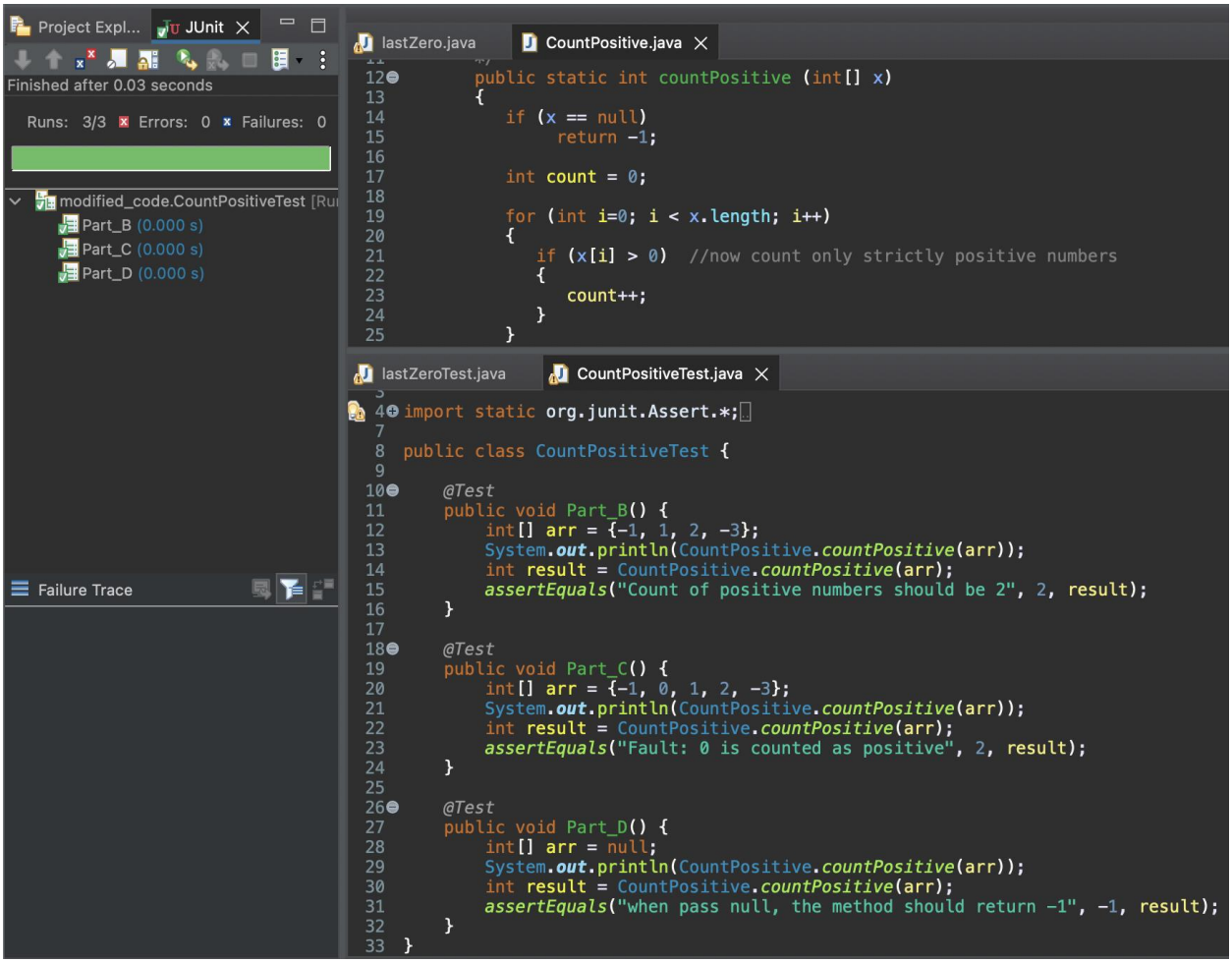
d. Test Case, Results an Error

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



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.

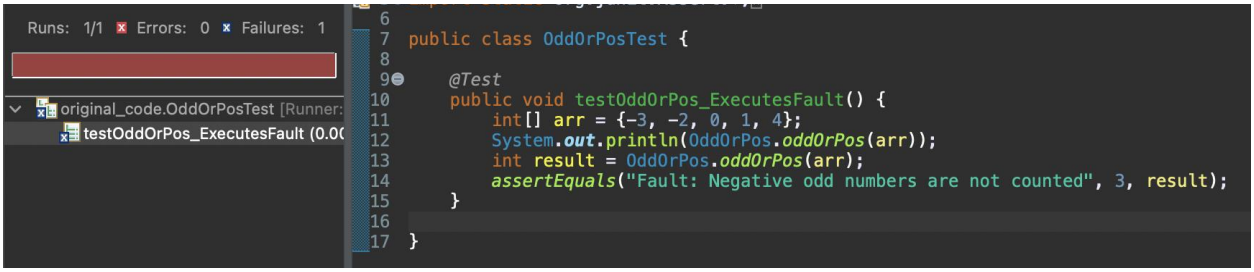
```
21                  if (x[i]%2 != 0 || x[i] > 0)   //  now detect odd numbers regardless of their sign
```

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.

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.

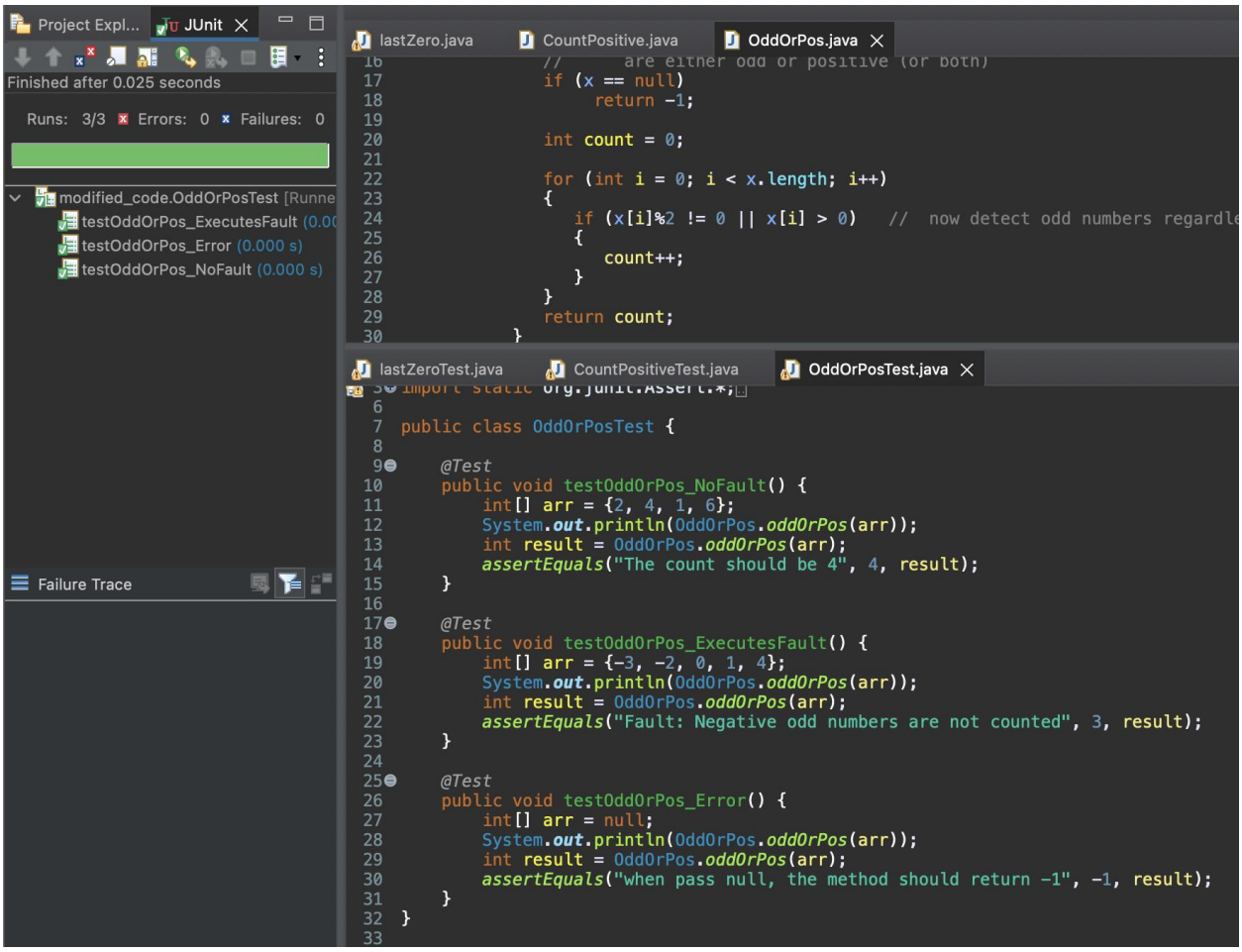


d. Test Case, Results an Error

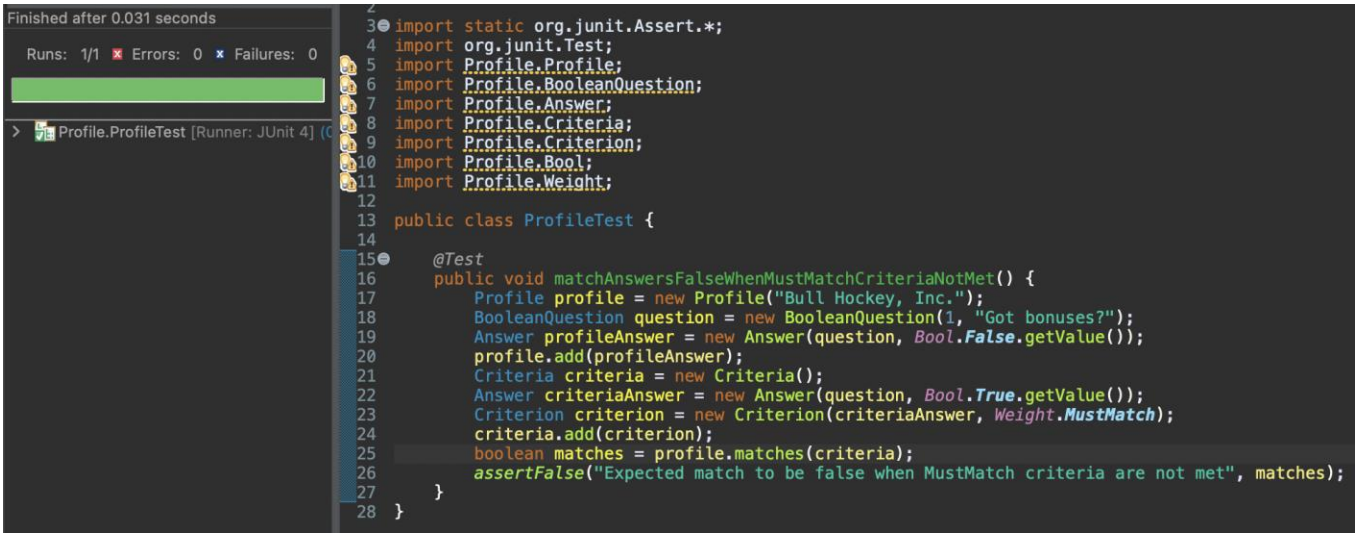
Passing a null array causes the error like I explained in FindLast Part D

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



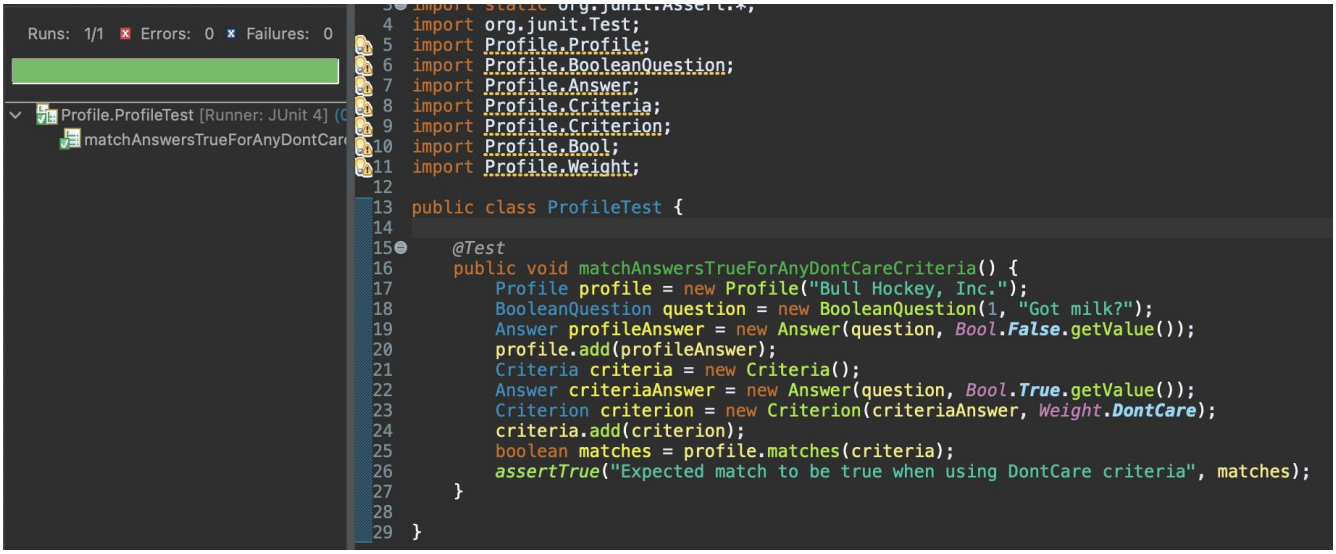
4b




```java
import static org.junit.Assert.*;
import org.junit.Test;
import Profile.Profile;
import Profile.BooleanQuestion;
import Profile.Answer;
import Profile.Criteria;
import Profile.Criterion;
import Profile.Bool;
import Profile.Weight;

public class ProfileTest {

    @Test
    public void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
        Profile profile = new Profile("Bull Hockey, Inc.");
        BooleanQuestion question = new BooleanQuestion(1, "Got bonuses?");
        Answer profileAnswer = new Answer(question, Bool.False.getValue());
        profile.add(profileAnswer);
        Criteria criteria = new Criteria();
        Answer criteriaAnswer = new Answer(question, Bool.True.getValue());
        Criterion criterion = new Criterion(criteriaAnswer, Weight.MustMatch);
        criteria.add(criterion);
        boolean matches = profile.matches(criteria);
        assertFalse("Expected match to be false when MustMatch criteria are not met", matches);
    }
}
```

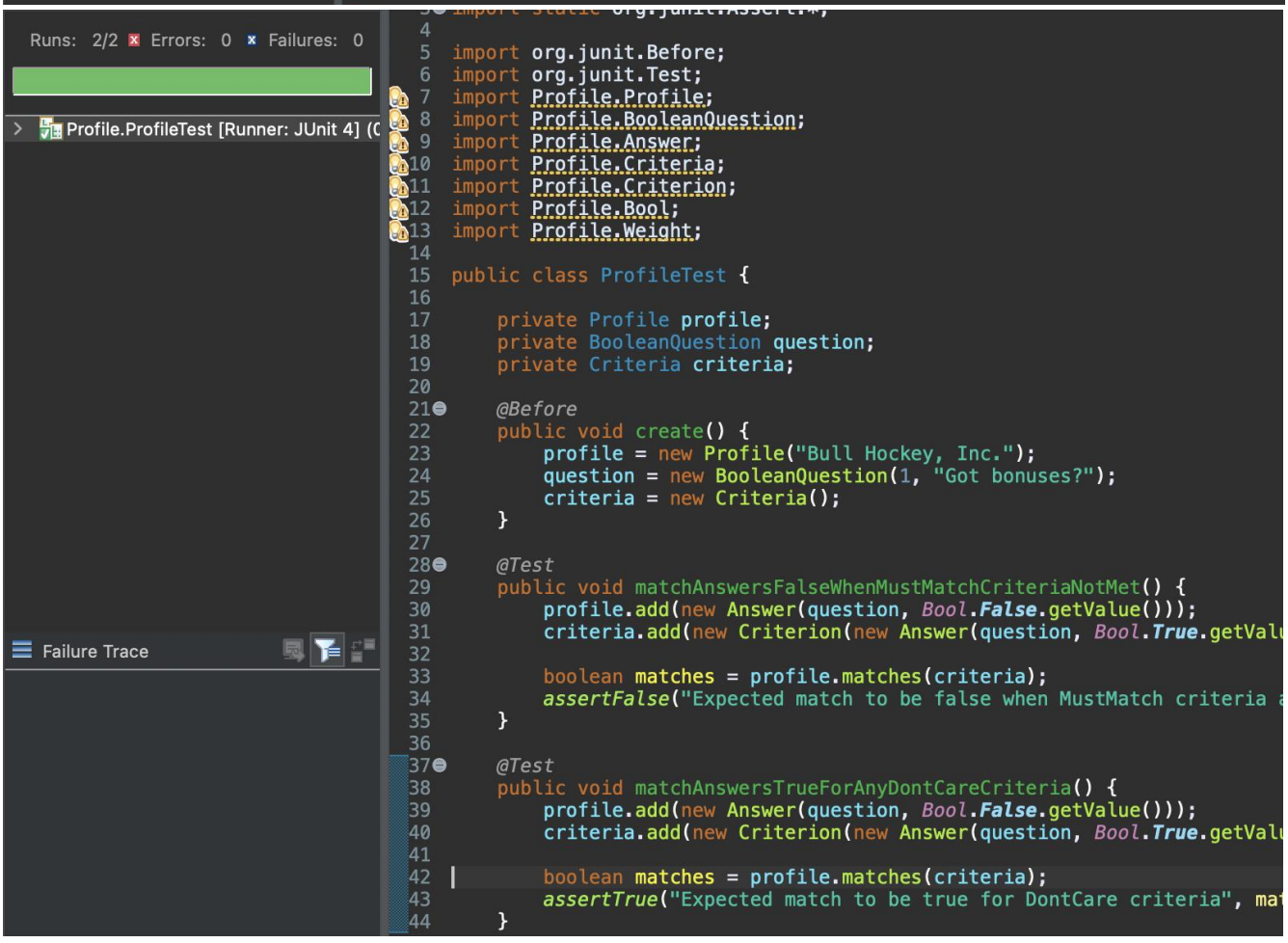



```java
import org.junit.Test;
import Profile.Profile;
import Profile.BooleanQuestion;
import Profile.Answer;
import Profile.Criteria;
import Profile.Criterion;
import Profile.Bool;
import Profile.Weight;

public class ProfileTest {

    @Test
    public void matchAnswersTrueForAnyDontCareCriteria() {
        Profile profile = new Profile("Bull Hockey, Inc.");
        BooleanQuestion question = new BooleanQuestion(1, "Got milk?");
        Answer profileAnswer = new Answer(question, Bool.False.getValue());
        profile.add(profileAnswer);
        Criteria criteria = new Criteria();
        Answer criteriaAnswer = new Answer(question, Bool.True.getValue());
        Criterion criterion = new Criterion(criteriaAnswer, Weight.DontCare);
        criteria.add(criterion);
        boolean matches = profile.matches(criteria);
        assertTrue("Expected match to be true when using DontCare criteria", matches);
    }

}
```




```java
import org.junit.Before;
import org.junit.Test;
import Profile.Profile;
import Profile.BooleanQuestion;
import Profile.Answer;
import Profile.Criteria;
import Profile.Criterion;
import Profile.Bool;
import Profile.Weight;

public class ProfileTest {

    private Profile profile;
    private BooleanQuestion question;
    private Criteria criteria;

    @Before
    public void create() {
        profile = new Profile("Bull Hockey, Inc.");
        question = new BooleanQuestion(1, "Got bonuses?");
        criteria = new Criteria();
    }

    @Test
    public void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
        profile.add(new Answer(question, Bool.False.getValue()));
        criteria.add(new Criterion(new Answer(question, Bool.True.getValu

        boolean matches = profile.matches(criteria);
        assertFalse("Expected match to be false when MustMatch criteria a
    }

    @Test
    public void matchAnswersTrueForAnyDontCareCriteria() {
        profile.add(new Answer(question, Bool.False.getValue()));
        criteria.add(new Criterion(new Answer(question, Bool.True.getValu

        boolean matches = profile.matches(criteria);
        assertTrue("Expected match to be true for DontCare criteria", mat
    }
```

Question 1:

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

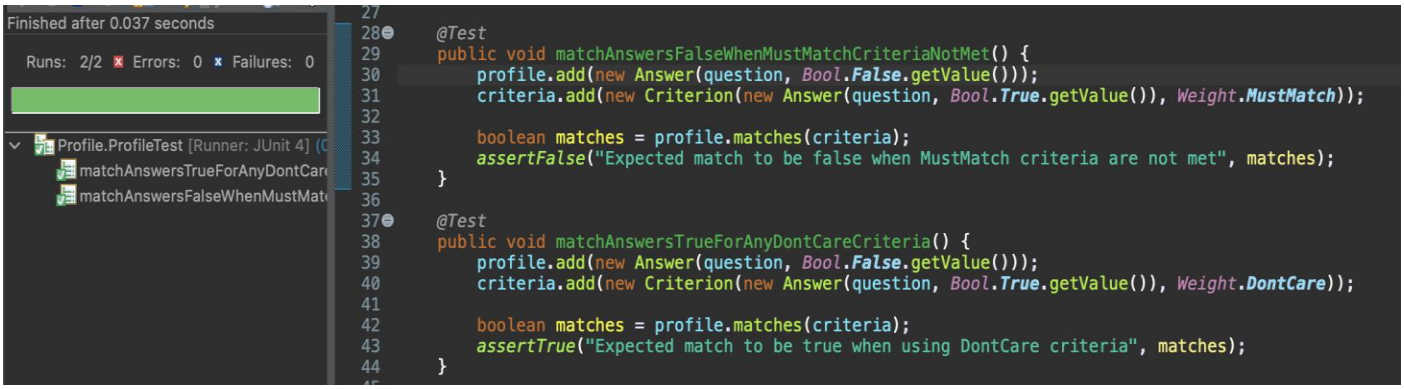
1. Test Instance Creation:
   JUnit creates a new instance of the ProfileTest class.
2. @Before Method Execution:
   Before any test method runs, the method annotated with @Before is executed. This method:
   a. Instantiates a new Profile object.
   b. Creates a new BooleanQuestion.
   c. Initializes 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));



Question 3:

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

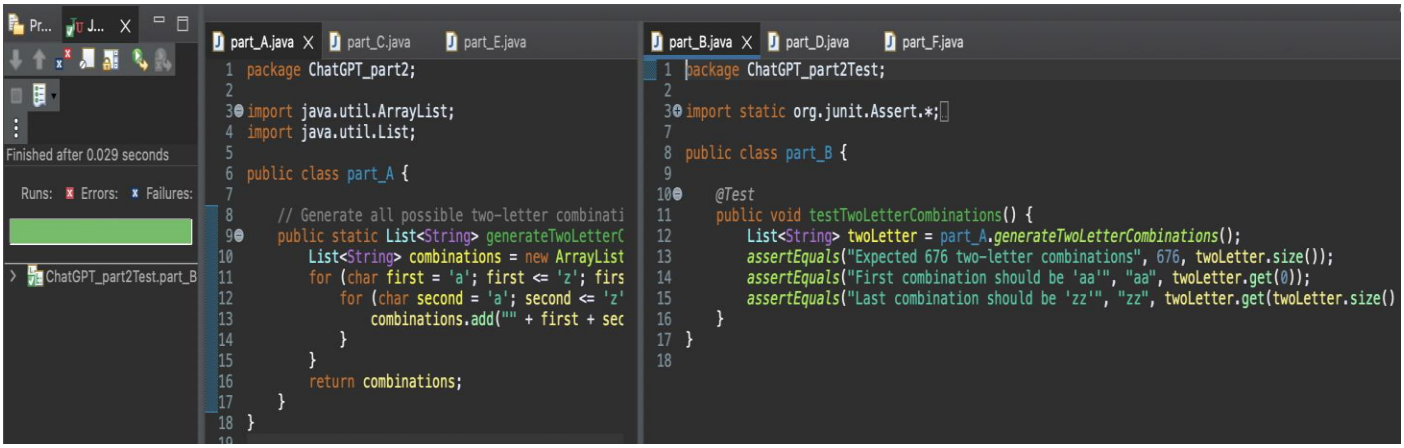
1. **Exponential Growth:**
   The total number of combinations for a given length n is 26^n.

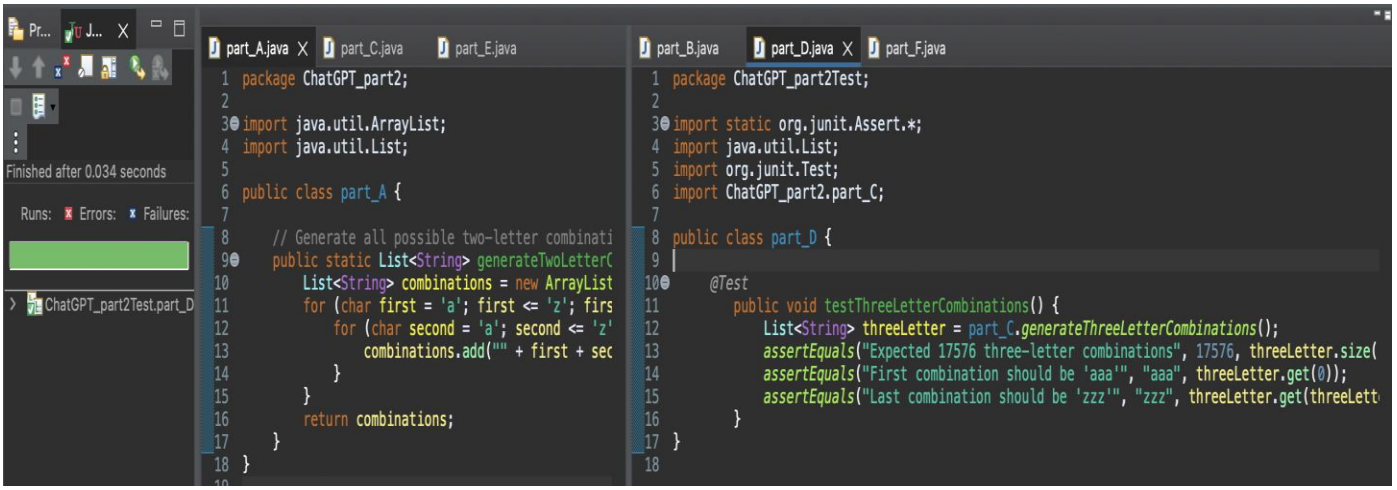
This quickly increases the number of combinations, which in turn affects both memory usage and processing time.

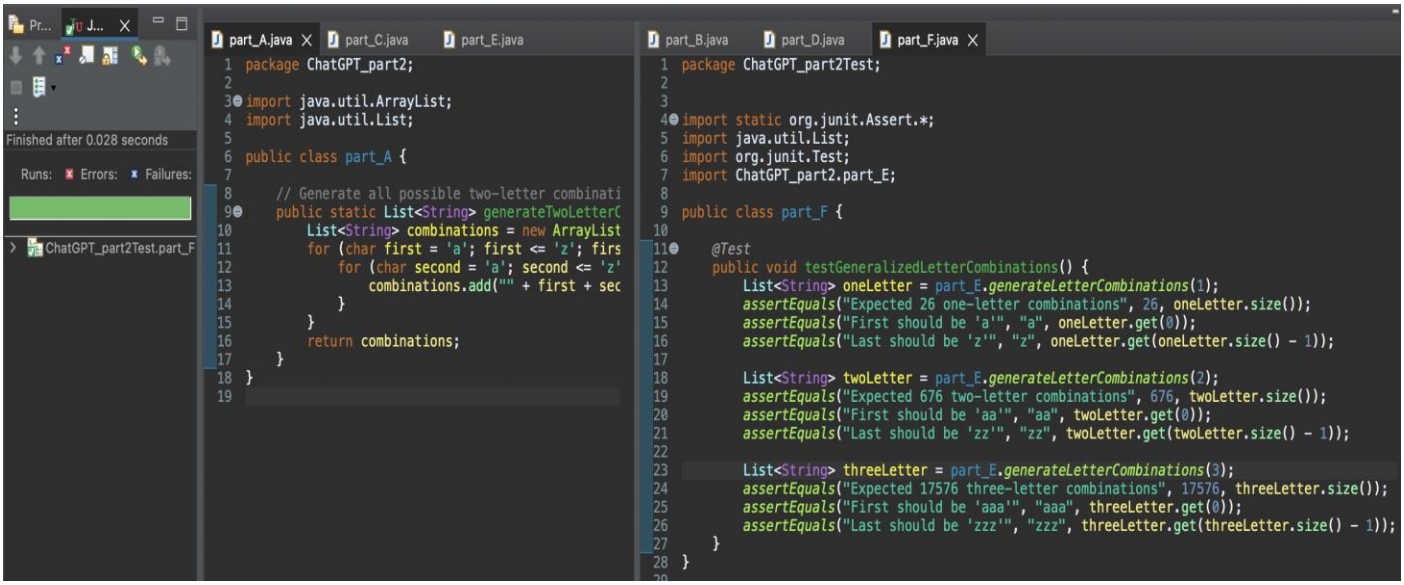
2. **System Resources and Practical Limits:**
   a. **Memory:** The system must have enough memory to store 26^n strings. For larger values of n, this may become infeasible.
   b. **Time Complexity:** The time required to generate all combinations increases exponentially with n. Even if memory is sufficient, 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


Finished after 0.029 seconds
Runs: Errors: Failures:
ChatGPT_part2Test.part_B
part_A.java
part_C.java
part_E.java
part_B.java
part_D.java
part_F.java
package ChatGPT_part2;
import java.util.ArrayList;
import java.util.List;
public class part_A {
// Generate all possible two-letter combinati
public static List<String> generateTwoLetterC
List<String> combinations = new ArrayList
for (char first = 'a'; first <= 'z'; firs
for (char second = 'a'; second <= 'z'
combinations.add("" + first + sec
}
}
return combinations;
}
}
package ChatGPT_part2Test;
import static org.junit.Assert.*;
public class part_B {
@Test
public void testTwoLetterCombinations() {
List<String> twoLetter = part_A.generateTwoLetterCombinations();
assertEquals("Expected 676 two-letter combinations", 676, twoLetter.size());
assertEquals("First combination should be 'aa'", "aa", twoLetter.get(0));
assertEquals("Last combination should be 'zz'", "zz", twoLetter.get(twoLetter.size()
}
}


Finished after 0.034 seconds
Runs: Errors: Failures:
ChatGPT_part2Test.part_D
part_A.java
part_C.java
part_E.java
part_B.java
part_D.java
part_F.java
package ChatGPT_part2;
import java.util.ArrayList;
import java.util.List;
public class part_A {
// Generate all possible two-letter combinati
public static List<String> generateTwoLetterC
List<String> combinations = new ArrayList
for (char first = 'a'; first <= 'z'; firs
for (char second = 'a'; second <= 'z'
combinations.add("" + first + sec
}
}
return combinations;
}
}
package ChatGPT_part2Test;
import static org.junit.Assert.*;
import java.util.List;
import org.junit.Test;
import ChatGPT_part2.part_C;
public class part_D {
@Test
public void testThreeLetterCombinations() {
List<String> threeLetter = part_C.generateThreeLetterCombinations();
assertEquals("Expected 17576 three-letter combinations", 17576, threeLetter.size(
assertEquals("First combination should be 'aaa'", "aaa", threeLetter.get(0));
assertEquals("Last combination should be 'zzz'", "zzz", threeLett
}
}


Finished after 0.028 seconds
Runs: Errors: Failures:
ChatGPT_part2Test.part_F
part_A.java
part_C.java
part_E.java
part_B.java
part_D.java
part_F.java
package ChatGPT_part2;
import java.util.ArrayList;
import java.util.List;
public class part_A {
// Generate all possible two-letter combinati
public static List<String> generateTwoLetterC
List<String> combinations = new ArrayList
for (char first = 'a'; first <= 'z'; firs
for (char second = 'a'; second <= 'z'
combinations.add("" + first + sec
}
}
return combinations;
}
}
package ChatGPT_part2Test;
import static org.junit.Assert.*;
import java.util.List;
import org.junit.Test;
import ChatGPT_part2.part_E;
public class part_F {
@Test
public void testGeneralizedLetterCombinations() {
List<String> oneLetter = part_E.generateLetterCombinations(1);
assertEquals("Expected 26 one-letter combinations", 26, oneLetter.size());
assertEquals("First should be 'a'", "a", oneLetter.get(0));
assertEquals("Last should be 'z'", "z", oneLetter.get(oneLetter.size() - 1));
List<String> twoLetter = part_E.generateLetterCombinations(2);
assertEquals("Expected 676 two-letter combinations", 676, twoLetter.size());
assertEquals("First should be 'aa'", "aa", twoLetter.get(0));
assertEquals("Last should be 'zz'", "zz", twoLetter.get(twoLetter.size() - 1));
List<String> threeLetter = part_E.generateLetterCombinations(3);
assertEquals("Expected 17576 three-letter combinations", 17576, threeLetter.size());
assertEquals("First should be 'aaa'", "aaa", threeLetter.get(0));
assertEquals("Last should be 'zzz'", "zzz", threeLetter.get(threeLetter.size() - 1));
}
}