

**Phạm Trần Gia Hung**

**2331200153**

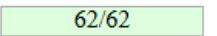
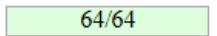
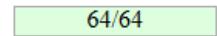
**Lab4**

## **Practice Assignment 4**

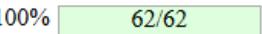
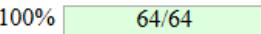
**Pit test Report:**

### **Pit Test Coverage Report**

#### **Project Summary**

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
3	100% 	100% 	100% 

#### **Breakdown by Package**

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
<a href="#">Class</a>	3	100% 	100% 	100% 

---

Report generated by [PIT](#) dev-SNAPSHOT

Enhanced functionality available at [arcmutate.com](#)

**Save the test code, coverage report and mutation report as per question numbers. Make a zip folder of all the files and upload in Moodle.**

1. Set.class represents sets of integers. The elements of a set are stored in an ArrayList. They are sorted and without duplicates to speed up some operations. Two methods might need an explanation:

public void section(Set s) { ... }: removes from this set any element that is equal to an element in s.

public boolean containsArithTriple() { ... }: returns true if there are three elements, x, y and z, in this set such that  $y - x = z - y$ .

a. Execute test cases that give you 100% branch coverage for this source code. Upload the test code and coverage report in Moodle. 10 Points

- a. There are bugs in the implementation. Find the bug and write the description of bugs in Set.java file as a comment. Upload the file alone with the bug-description in Moodle. 5 Points
- c. Do mutation test for these test cases to achieve 100% mutation coverage. Submit your mutation test report and modified test cases in Moodle. 20 Points

### Bug:

```

37
38
39
40
41
42
43
44
45
for (int i = 0; i < a.size(); i++) {
    //wrong operator for sort operation, should be < and not >
    // set should main descending order
    //if (a.get(i) > x) {
    if (a.get(i) < x) {
        a.add(i, x);
        flag=1;
        break;
}

```

### Test Coverage:

```

✓  ✓ SetTest (Test)      4 ms
    ✓ testContainsArithT 4 ms
    ✓ testContainsArithTi 0 ms
    ✓ testContainsArithTi 0 ms
✓ 16 tests passed 16 tests total, 4 ms
C:\Users\User\.jdks\ms-17.0.18\bin\
Process finished with exit code 0

Set.java      100% [34/34]      100% [30/30]      100% [30/30]

```

### 2. Encryption( ) method in EncryptMessage.java file has encrypted an input message in two steps as follows:

- Swap the 1st and 2nd character of the message, then swap the 3rd and 4th character, then the 5th and 6th character and so on. If the length of message is odd, the last character should not be swapped with any other.
- Replace each occurrence of the letter 'a' in the message obtained after the first step by the letter 'z', each occurrence of 'b' by 'y', each occurrence of 'c' by 'x', etc., and each occurrence of 'z' in the message obtained by 'a'.

Encryption( ) method only takes lower case alphabets as input.

- a. Create and execute test cases to achieve 100% branch coverage. Upload the test code and coverage report in Moodle. 10 Points
- b. The implementation of EncryptMessage.java has bugs. Identify bugs and write the description of the bugs in the EncryptMessage.java file as a comment. Upload the file alone with the bugdescription in Moodle. 5 Points.
- c. Do mutation test for these test cases to achieve 100% mutation coverage. Submit your mutation test report and modified test cases in Moodle. 20 Points

### Bug:

```
15      }
16  ↘      //Incorrect odd/even logic, correct should be odd not even
17
18  ↘      //if (length % 2 == 0) {
19      if (length % 2 == 1) {
20          sb1.append(message.charAt(length - 1));
21
22      } else {
23          // wrong way to do character substitute, should be - and not +
24          //currentChar = (char) ('z' + 'a' + currentChar);
25          currentChar = (char) ('z' + 'a' - currentChar);
26          sb2.append(currentChar);
27
28      }
29
30 }
```

### Test Coverage:

```
EncryptMessageTest 8 ms
  ✓ 12 tests passed 12 tests total, 8 ms
    ✓ testSwapOnlyEffect 4 ms
    ✓ testSingleCharacter 2 ms
    ✓ testLongString      1 ms
C:\Users\User\.jdks\ms-17.0.18\bin\java.exe ...
Process finished with exit code 0
```

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">EncryptMessage.java</a>	100% <div style="width: 100%;">18/18</div>	100% <div style="width: 100%;">17/17</div>	100% <div style="width: 100%;">17/17</div>

**3.** fizz\_buzz( ) method in FizzBuzz.java file accepts input from 0 to 35. The method returns “Fizz” if the input number is divided by 3. The method returns “Buzz” if the input number is divided by 5. The method returns “Fizz!Buzz!”, if the number is divided by both 3 and 5. Otherwise, the method returns the number itself.

- a. Create and execute test cases to achieve 100% branch coverage. Upload the test code and coverage report in Moodle. 10 Points
- b. The implementation of EncryptMessage.java has bugs. Identify bugs and write the description of the bugs in the EncryptMessage.java file as a comment. Upload the file alone with the bugdescription in Moodle. 5 Points.
- c. Do mutation test for these test cases to achieve 100% mutation coverage. Submit your mutation test report and modified test cases in Moodle. 15 Points

### Bug:

```
5     public String fizz_buzz(int i) { 9 usages
6         // wrong upper bound check, off by 1, should be <=
7         //if (i < 35 && i >= 0) {
8         if (i <= 35 && i >= 0) {
```

### Test Coverage:

The screenshot shows a Java IDE interface with a test runner window. The test class is FizzBuzzTest. It contains three test methods: testWrongInput\_To, testBuzz, and testFizz. All tests passed in 7ms. The terminal output shows the command used to run the tests and the exit code 0. Below the terminal, there are four progress bars for the files FizzBuzz.java, TestFizz.java, TestBuzz.java, and TestWrongInput.java, all showing 100% coverage with 17/17 tests passed.

File	Coverage	Passed	Total
FizzBuzz.java	100%	17/17	17/17
TestFizz.java	100%	17/17	17/17
TestBuzz.java	100%	17/17	17/17
TestWrongInput.java	100%	17/17	17/17