SUTD 2021 50.003 Problem Set 2

James Raphael Tiovalen

Cohort Exercise 1

The three test cases are written inside the included `FindMaxTest.java` file. The first test is expected to pass, the second test is expected to error and throw an exception (we annotate the test with an expected `ArrayIndexOutOfBoundsException`), and the third test is expected to fail (we annotate the test with an expected `AssertionError`) due to the buggy implementation of `FindMax.java`.

Cohort Exercise 2

We change the condition of the second if statement under the `repOK()` method from `this.empty() != this.elements().hasMoreElements()` to `this.empty() == this.elements().hasMoreElements()`. This is because if the stack is both empty and has non-zero elements, something went wrong, and we return `false` (instead of the double negative).

Cohort Exercise 3

In the `StackTest.java` file, the `testRepOk()` test method has been decomposed into these test methods, each testing a specific behaviour of the stack: `testEmptyStack()`, `testOnePush()`, `testOnePushAndOnePop()` and `testAlternatingPushesAndPops()`.

Cohort Exercise 4

A parameterized test for `QuickSort.java` has been written in the `QuickSortParameterizedTest.java` file. Note that a `getArray()` getter method that returns a clone of the array attribute has been added to the `QuickSort` class in the `QuickSort.java` file to enable ourselves to inspect the content of the sorted array for the purposes of the parameterized test.

Cohort Exercise 5

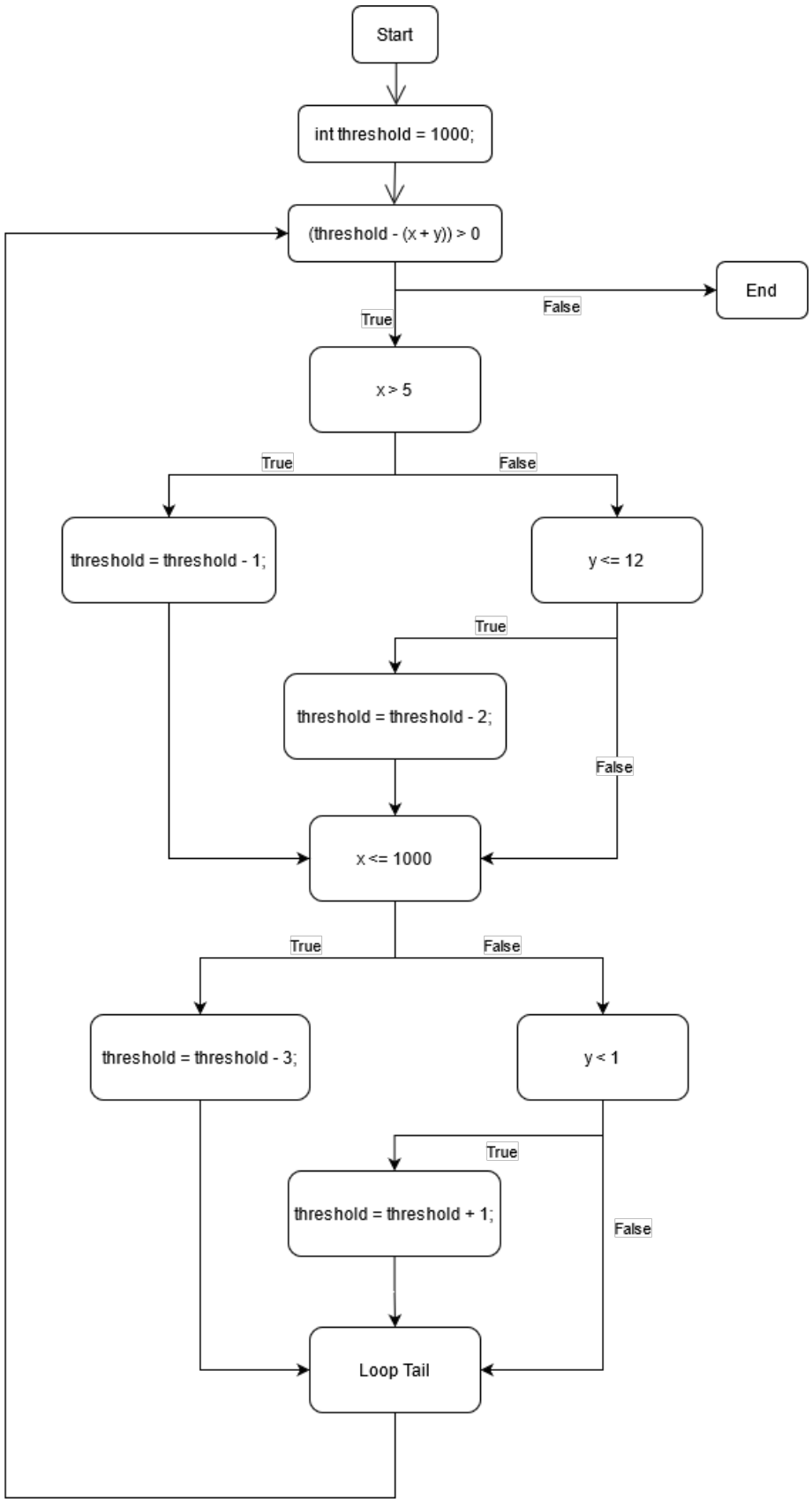
A template code of these test cases is included in the `MailClientTest.java` file.

- Test Case 1:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: mail.sutd.edu.sg

        - ✓ Incoming Mail IMAP Server: imap.sutd.edu.sg
        - ✓ Email Address: james_raphael@mymail.sutd.edu.sg
        - ✓ Subject: "Spectacular Work, Big Boss!"
    - Expected Output: Successful Delivery
- Test Case 2:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: null
        - ✓ Incoming Mail IMAP Server: imap.sutd.edu.sg
        - ✓ Email Address: james_raphael@mymail.sutd.edu.sg
        - ✓ Subject: "Spectacular Work, Big Boss!"
    - Expected Output: Failed Delivery
- Test Case 3:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: mail.sutd.edu.sg
        - ✓ Incoming Mail IMAP Server: null
        - ✓ Email Address: james_raphael@mymail.sutd.edu.sg
        - ✓ Subject: "Spectacular Work, Big Boss!"
    - Expected Output: Failed Delivery
- Test Case 4:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: mail.sutd.edu.sg
        - ✓ Incoming Mail IMAP Server: imap.sutd.edu.sg
        - ✓ Email Address: null
        - ✓ Subject: "Spectacular Work, Big Boss!"
    - Expected Output: Failed Delivery
- Test Case 5:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: mail.sutd.edu.sg
        - ✓ Incoming Mail IMAP Server: imap.sutd.edu.sg
        - ✓ Email Address: james_raphael@mymail.sutd.edu.sg
        - ✓ Subject: [A single line text string of the first 2000 characters taken from Shakespeare's play The Tragedy of Hamlet, Prince of Denmark, violating the RFC 2822 technical specification limit.]
    - Expected Output: Failed Delivery
- Test Case 6:
    - Given Inputs:
        - ✓ Outgoing Mail SMTP Server: mail.sutd.edu.sg
        - ✓ Incoming Mail IMAP Server: imap.sutd.edu.sg
        - ✓ Email Address: james_raphael
        - ✓ Subject: "Spectacular Work, Big Boss!"
    - Expected Output: Failed Delivery

Cohort Exercise 6

This is the control flow graph of the `manipulate()` function of `Disk.java`:



Cohort Exercise 7

Two test cases are the minimum to obtain 100% statement coverage. The test cases are:

- Test Case 1: `{(x, y) = (4, 11)}`
- Test Case 2: `{(x, y) = (1001, -2)}`

The first test case will cover the statements where `x > 5` is false, `y <= 12` is true, and `x <= 1000` is true, i.e., `threshold = threshold - 2` and `threshold = threshold - 3`. The second test case will cover the statements where `x > 5` is true, `x <= 1000` is false and `y < 1` is true, i.e., `threshold = threshold - 1` and `threshold = threshold + 1`. The rest of the statements not mentioned are automatically covered by default, as long as the condition `(threshold - (x + y)) > 0` is true, which is true for both aforementioned test cases. This is the minimum as it is impossible for statements located at different branches of the same conditional statement to be executed at the same time.

Cohort Exercise 8

Three test cases are the minimum to obtain 100% branch coverage. The test cases are:

- Test Case 1: `{(x, y) = (4, 11)}`
- Test Case 2: `{(x, y) = (1001, -2)}`
- Test Case 3: `{(x, y) = (4, 13)}`

The first test case will cover the false branch of the `x > 5` condition, the true branch of the `y <= 12` condition and the true branch of the `x <= 1000` condition. The second test case will cover the true branch of the `x > 5` condition, the false branch of the `x <= 1000` condition and the true branch of the `y < 1` condition. The third test case will cover some overlapping branches as the first test case, but it will also cover the otherwise uncoverable false branch of the `y <= 12` condition. Since the false branch of the `y < 1` is unreachable (as it is impossible for the `(threshold - (x + y)) > 0` condition to be true if both `x > 1000` and `y >= 1` are true at the same time), it will not be considered in the calculation of total branch coverage, and thus these three test cases are the minimum to achieve 100% branch coverage.

Cohort Exercise 9

We first rule out the route whereby the program will run forever, since it will exceed the specified requirement of termination after at most 100 iterations of the while loop. Hence, we exclude the path where `x > 5` is true, `x <= 1000` is false and `y < 1` is true.

We will then consider the route where `x > 5` is true and `x <= 1000` is true. In this path, one loop will decrease the value of the `threshold` variable by a net of 4. As such, `threshold` will only be reduced at most by a net value of 400. Since the condition `(threshold - (x + y)) > 0` need to remain true throughout all of the iterations, the possible values of `x` and `y` are such that `600 <= x + y < 1000`. As such, there are 100 possible unique paths here (with multiple sets of input values covering the same paths).

Next, we will consider the route where `x > 5` is false, `y <= 12` is false and `x <= 1000` is true. In this path, one loop will decrease the value of the `threshold` variable by a net of 3. As such, `threshold` will only be reduced at most by a net value of 300. Since the condition `(threshold - (x + y)) > 0` need to remain true throughout all of the iterations, the possible values of `x` and `y` are such that `700 <= x + y < 1000`. As such, there are 100 possible unique paths here (with multiple sets of input values covering the same paths).

Next, we will consider the route where `x > 5` is false, `y <= 12` is true and `x <= 1000` is true. In this path, one loop will decrease the value of the `threshold` variable by a net of 5. As such, `threshold` will only be reduced at most by a net value of 500. Since the condition `(threshold - (x + y)) > 0` need to remain true throughout all of the iterations, the possible values of `x` and `y` are such that `500 <= x + y < 1000`. **However**, for this path to be executed, the value of `x` needs to be less than or equal to 5 and the value of `y` need to be less than or equal to 12 **at the same time**, both of which will result in the loop being executed way more than 100 times (the condition `500 <= x + y < 1000` is impossible to be satisfied in this route). As such, this route contributes 0 paths to the overall total number of paths.

And finally, we count the one additional path whereby the condition `(threshold - (x + y)) > 0` is not true, thereby terminating the program the moment it was executed. Therefore, there are **201** paths in total.

Cohort Exercise 10

Yes. Since we only consider the feasible outcomes for coverage, we do obtain 100% condition coverage by using our test cases for 100% branch coverage. This is because we do not consider the false condition of the `y <= 1` conditional statement in our calculation since it will never be executed.

Cohort Exercise 11

We can use this example test case that could reveal a bug in the `manipulate()` function: `{(x, y) = (1001, -10)}`. Since `x > 5` is true, the statement `threshold = threshold - 1` will be executed. However, since `x <= 1000` is false and since `y < 1` is true, the statement `threshold = threshold + 1` will also be executed. Thus, the final value of the `threshold` variable after one loop will be unchanged (i.e., remains as `1000`). Hence, this test case will cause the `manipulate()` function to never terminate, thereby violating the required specification. In fact, any set of values of `x` and `y` that satisfies these specific conditions will cause the `manipulate()` function to be non-terminating (another example would be: `{(x, y) = (2000, -2000)}`). A JUnit test has been included in the `DiskTest.java` file, whereby we expect the test to still be running after an arbitrary period of 5 seconds (we utilized an arbitrary timeout since it is impossible to write a test program to deterministically confirm whether a program will halt or not due to the popular and well-known Halting Problem).

Cohort Exercise 12

The three types of test cases (black-box testing, white-box testing with 100% branch coverage and fault-based testing) are written in the included `RussianTest.java` file.

For the black-box tests, we use some random normal inputs of 6 and 9. Then, we test if the program can handle multiplication of two zeroes. Then, we test the program with two large numbers. Then, we test the program with two very negative numbers. Since the program fails, we test the program further by only using a negative number for `m`, followed by a separate test of only using a negative number for `n`. We then discovered that the program cannot handle negative values of `n`.

For the white-box tests, we test all of the possible branches of the two conditions present in the code. We first test the two possible choices when the condition `n > 0` is false (i.e., `n == 0` and `n < 0`). Then, we test with a value of `n > 0` when `n` is odd and when `n` is even, thereby testing the two branches of the conditional statement `n % 2 == 1`.

For the fault-based testing, we observe that the while loop only executes if `n > 0` and as such, using a negative value of `n` would return 0. Another possible bug can be found when we use the maximum value of integers possible in Java (using `Integer.MAX_VALUE`), which would cause an integer overflow. Using the minimum value

of integers possible in Java would only cause the function to return a zero, since the `multiply()` function cannot handle negative values of `n`.