Northeastern University
CS4500 – Software Development
Spring 2018, Professors Annunziato, Derbinsky, and Weintraub

Homework 4: Structural Testing

The purpose of this assignment is to provide some experience applying the concepts we've discussed on functional testing. Please place your answers in your personal class github.ccs.neu.edu repo in <your-CCIS-ID>/homework-4.

1. Given the following method, identify the number of test cases necessary to achieve 100% code coverage, 100% branch coverage, and 100% condition coverage. Provide one instance of each test contributing to a coverage criterion and express each test using a test spec style. Indicate what the test covers in your answer. You may reuse tests across each criterion. You do not need to write JUnit for this problem.

   An example of *test spec style* is given below - it's adapted from the functional testing work you've done.

```
1       /** Determines if three sides can form a triangle, and if so
2      *  is it a right triangle.
3      *
4      *  @param a   side a
5      *  @param b   side b
6      *  @param c   side c
7      */
8
9     public void classifier (double a, double b, double c) {
10
11        // you can assume a, b, and c > 0
12
13        if ((a + b > c) && (a + c > b) && (b + c > a)){
14            System.out.printf("it's a triangle ");
15            if (a * a + b * b == c * c) {
16                System.out.println("and it's a right triangle");
17            }
18        }
19    }
```

Unit Name	classifier()		
Summary (e.g. Interface spec)	Determines if three		
Input(s)	Expected Result	Coverage Criterion (statement, branch, simple condition,...)	Covers (what is being covered in the code)
(3, 4, 5)	it's a triangle and it's a right triangle	statement	gives 100% statement coverage
(3, 4, 5)	""	branch	covers T branch at line 13 and T branch at line 15
(1, 1, 1)	it's a triangle	branch	covers T branch at line 13 and F branch at line 15
(9, 1, 1)	—	branch	covers F branch at line 13
(9, 1, 1)	—	simple condition	covers F at line 13
(3, 4, 5)	it's a triangle and it's a right triangle	simple condition	covers T at line 13 and covers T at line 15
(1, 1, 1)	it's a triangle	simple condition	covers F branch at line 15

solution

The point of this exercise is for students to see first hand the difference in coverage criteria. While this is a simple code, it highlights the differences.

For statement coverage, at least one test is necessary. A right triangle will cover all the lines.

For branch coverage, two tests for each branch of each conditional is necessary. You need:

- one case where it isn't a triangle. In case you've forgotten your geometry, the sum of any two sides must be greater than the third side. So, one case where $a + b < c$ should exist.
- one case where the input is a triangle and it is a right triangle
- one case where the input is a triangle and it is not a right triangle.

The first case will be combined with the third or the fourth case, so only three tests are necessary.

For condition coverage, the first `if` has three conditions. Each one needs to be set to true or to false. The second if has only one condition, so two test cases are needed - one where it evaluates to T and one, F.

2. Given the following method, `Example.consolidateSpace(String)`,

   (a) construct its CFG. It must be rendered in a tool. Graphviz (`http://graphviz.org/`) is a nice graph drawing tool. It lacks nice WYSIWYG front-end tools. You may find a plug-in for Eclipse, Intellij, or Atom. Lucidchart (`https://www.lucidchart.com/`) is a free on-line tool for drawing flows. Powerpoint is another option.

   (b) Using JUnit,

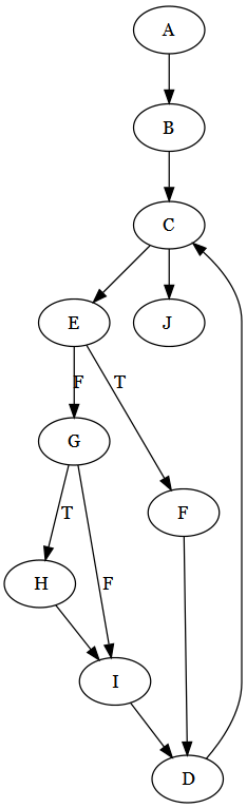
      i. Develop a set of test cases and submit both the test cases and a report created by a tool demonstrating that this test suite meets 100% statement coverage.

      ii. Develop and submit a set of test cases and submit a report created by a tool demonstrating that this test suite meets 100% branch coverage.

      iii. Describe the bug(s) in the code and identify the specific line(s) where the bug(s) is? Identify where in the CFG it is. List the test(s) that illuminated the issue?

solution

```
1  public class Example {
2
3    public  String consolidateSpace (String inputText) {
4
5    /** replace all multiple contiguous spaces
6     *      with a single space
7     *
8     *  @param inputText  a string that may contain
9              *      more than one contiguous space
10    *  @return a string with at most one (1)
11    *           contiguous space character
12    *
13    */
14
15        StringBuilder answer = new StringBuilder();
16   A
17        boolean foundSpace = false;
18
19                    B             C
20        for (int i = 0; i < inputText.length();
21             i++) D
22        {           E
23          if (inputText.charAt(i) == ' ') {
24              foundSpace = true;   F
25          }
26          else {   G
27              if (foundSpace) {
28                  answer.append(' ');    H
29                  foundSpace = false;
30              }
31              answer.append(inputText.charAt(i)); I
32          }
33        }
34
35        return (answer.toString());   J
36
37     }
38
39      public static void main(String[] args){
40
41        Example example = new Example();
42
43        String givenString = "Hello   there      buddy";
44        System.out.println("Input: " + givenString);
45
46        String result  = example.consolidateSpace(givenString);
47
48        System.out.println("Output: " + result);
49      }
50   }
```



Drawn with graphviz using `https://dreampuf.github.io/GraphvizOnline/`

For questions 2(b)i and 2(b)ii, you should create `Junit` tests that would cover the following situations (where *TEXT-STRING* is a non-null string composed of some number of non-blank characters):

(a) A case with 0, 1, and 2 leading spaces in *TEXT-STRING*. (Two spaces are a boundary. More spaces are possible, but bonus)

(b) A case with 0, 1, or 2 spaces embedded in *TEXT-STRING*. (Again, two spaces are a boundary. More spaces are possible, but bonus)

(c) A case with 0, 1, and 2 trailing spaces in *TEXT-STRING*. (Two spaces are a boundary. More spaces are possible, but bonus)

The example given in the problem ("Hello there buddy") will give 100% statement coverage and 100% branch coverage.

For question 2(b)iii, the bug is trailing spaces do not get consolidated into one space—they get dropped. The bug occurs either at block F, needing a check if one is at end of the string, or at block J, (checking if `foundSpace == true`). Note that we hit 100% coverage, but not 100% correct behavior (modulo the caveat that follows in the next paragraph).

However, many students pointed out the description says there needs to be non-space characters surrounding the spaces that get reduced. Given Java does not use a character to delimit the end of a string, you could claim there isn't a bug in the code. In this case, the bug is with the professor. You can make a similar argument about leading spaces.

If we ran this `main`, the output would look like:
`Input: Hello␣␣␣␣there␣␣␣␣␣buddy`
`Output: Hello␣there␣buddy`

IMPORTANT NOTE about generating coverage reports:

Eclipse users You can generate statement and branch coverage reports in Eclipse using Emma. See `https://wiki.eclipse.org/Code_Coverage_with_Emma` for more information.

Intellij users Intellij can generate a statement coverage report. However, it captures branch coverage data, but does not generate a nice report for branch coverage. However, you can see branch coverage if you follow these steps:

(a) Edit the run configurations. In the Code Coverage tab, select tracing and add your class/-package under Packages and classes to record data.

(b) Run the code with coverage. Now, in the editor you can select the coverage bars on the left (with the line numbers). See `https://confluence.jetbrains.com/display/IDEADEV/IDEA+Coverage+Runner` for more information.

For this assignment, if you use intellij, you can record the "hits" the tool found for your tests and create a report manually.

Other environments You have to look to see what plug-ins work for your preferred tool.