Northeastern University
CS5500 – Managing Software Development
Spring 2018, Professors Annunziato and Weintraub

Homework 4: Testing

The purpose of this assignment is to provide some experience applying the concepts we've discussed on functional testing.

1. Please identify the boundaries for the following functions. For each one, provide a set of test cases that would qualify as a good test suite. (10 points each)

   (a) Takes an integer as input and determines if it lies in the range: $-10 \leq x \leq 10$.

   (b) Takes a float as input and determines if it lies in the range: $(x < -10)$ or $(x > 10)$.

   (c) Takes in a 10 digit telephone number as a string and returns true if the string is a legitimate phone number and false otherwise. Legitimate means:
   - properly formed (10 digits)
   - does not begin with a 1 or 0
   - the first three digits cannot be 555

2. Using the provided template for functional units, produce a test specification for the following component interface. Please highlight cases that are boundary conditions and should cover at least the minimal set of tests needed. (10 points)

```
1
2       // this does interesting calculations
3
4       public interface homework {
5
6       //  produces the square root of a non-negative input number n
7
8          double sqrt (int n);
9
10      // produces the square of an input number n
11
12         int sqr (int n);
13
14        // produces n!
15
16         int factorial (int n);
17      }
```

3. A team-mate gives you a test specification and asks you to review it before coding begins. Do accept or reject the spec? Justify your answer if you accept it. If you reject it, explain why and either correct or fill in any gaps. (10 points)

Name	Adder
Summary	Takes two integers as input and returns their sum (as an integer)
Input	**Expected Output**
1, 1	2
5, 35	40
100,936, 1,327,927	1,428,863
-3, -5	-8
-100,936, -1,327,927	-1,428,863
1, MAX_INT	Overflow Exception
1, -MAX_INT	Underflow Exception

4. Create a suite of at least 20 tests. Please commit these tests in your repository in a class Tests in package "tests". In these tests, you may assume for now that you can create a sorter by calling a method createSorter(). (20 points)

   Consider the interface below, which specifies a Sorter that sorts the elements of an array in place.

```
1  package util;
2
3  /**
4   * Generic interface for sorting an array of elements in-place.
5   */
6
7  public interface Sorter<T extends Comparable<T>> {
8      void sort(T[] list);
9  }
```

5. Create two different implementations of the Sorter interface that incorporate sorting algorithms of your choice. These algorithms must be efficient and at least be of $\mathcal{O}(n \log n)$ on average. For example, you could implement a `QuickSorter` class that uses Hoare's famous QuickSort algorithm. Commit these sorters along with the Sorter interface in a package `sorters`. (20 points)

   Given your two sorters, `Sorter1` and `Sorter2`, create two versions of the test-suite from the previous question, called `Sorter1Tests` and `Sorter2Tests` with appropriate definitions of `createSorter()` in each case, and place these in package `tests2`. Run these test suites to ensure that your sorters behave as expected. (20 points)

6. This question asks for structural testing coverage over the code you created in question 5. You are encouraged to install and use an automated tool for measuring test coverage. In Eclipse, there is a plug-in available called `EclEmma` (see `http://www.eclemma.org/`). Intellij does have some coverage capabilities as well, but it cannot automatically produce all the reports you will need. If you use intellij, you'll have to craft the report by hand. See the end note for more information. For other environments such as atom, you have to look to see what plug-ins work for your preferred tool. (30 points)

In all cases, you need to provide the actual counts per code line for the test suites as well as the coverage numbers.

(a) For each of your sorters, create a report listing statement coverage and branch coverage achieved by the test suites from Question 4.

(b) Attached to this assignment is another test suite for sorters. For each of your sorters, create a report listing statement coverage and branch coverage achieved by this test suite.

(c) Extend your test suites from Question 4 by adding tests needed to achieve 100% statement coverage. Place the resulting test suites in a package `tests3c`.

(d) Extend the test suite that you downloaded by adding tests necessary to achieve 100% branch coverage. Place the resulting test suite in a package `tests3d`.

End Note on IntelliJ

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:

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

2. 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.