HW #2 Software Testing

Com S/SE 417 Fall, 2018

Due at beginning of class Tuesday, Oct. 2 as pdf uploaded to Canvas

Textbook reading assignment: Chapters 4-6, Ammann and Offutt, 2nd ed.

Homework problems are adapted from the 2nd edition of the course textbook, "Introduction to Software Testing," by Ammann & Offutt.

1. (Chapter 3, Ex 8) Develop a set of data-driven JUnit tests for the Min program. These tests should be for normal, not exceptional, returns. Make your @Parameters method produce both String and Integer values. Please refer to https://cs.gmu.edu/~offutt/softwaretest/java/ for an example--DataDrivenCalcTest.java for the given class Calc.java.
   Submit the following:
   (i) DataDrivenMinTest.java
   (ii) Screenshot showing that the tests pass

2. (Chapter 4, based on Ex 1) Chapter 3 contained the program `Calc.java`. It is available on the program listings page on the book website. `Calc` currently implements one function: it adds two integers. Use test-driven design to add additional functionality to subtract two integers, multiply two integers, and divide two integers. First create a failing test for one of the new functionalities, modify the class until the test passes, then perform any refactoring needed. Repeat until all of the required functionality has been added to your new version of `Calc`, and all tests pass. Remember that in TDD, the tests determine the requirements. This means you must encode decisions such as whether the division method returns an integer or a floating point number in automated tests before modifying the software. Submit the following:
   (i) Final version of `Calc.java` and `CalcTest.java`
   (ii) A screen shot showing that all tests pass
   (iii) A narrative describing each TDD test created, the changes needed to make it pass, and any refactoring that was necessary.

3. (Chapter 5, Ex 1) Suppose that coverage criterion C1 subsumes coverage criterion C2. Further suppose that test set T1 satisfies C1 on program P, and test set T2 satisfies C2, also on P.
   (a) Does T1 necessarily satisfy C2? Explain.
   (b) Does T2 necessarily satisfy C1? Explain.
   (c) If P contains a fault, and T2 reveals the fault, T1 does not necessarily also reveal the fault. Explain.

4. (Chapter 6, Ex. 6.1.3) Answer the following questions for the method search() below:

```
public static int search (List list, Object element)
// Effects: if list or element is null throw NullPointerException
//   else if element is in the list, return an index
//   of element in the list; else return -1
//   for example, search ([3,3,1], 3) = either 0 or 1
// search ([1,7,5], 2) = -1
```

Base your answer on the following characteristic partitioning:

Characteristic: Location of element in list
Block 1: element is first entry in list
Block 2: element is last entry in list
Block 3: element is in some position other than first or last

(a) "Location of element in list" fails the disjointness property. Give an example that illustrates this.
(b) "Location of element in list" fails the completeness property. Give an example that illustrates this.
(c) Supply one or more new partitions that capture the intent of "Location of element in list" but do not suffer from completeness or disjointness problems.

5. (Chapter 6, parts of Ex. 6.1.4—we've excluded book's part (c)) Derive input space partitioning test inputs for the GenericStack class assuming the following method signatures:
   - public GenericStack ();
   - public void push (Object X);
   - public Object pop ();
   - public boolean isEmpty ();

   Assume the usual semantics for the GenericStack. Try to keep your partitioning simple and choose a small number of partitions and blocks.
   (a) List all of the input variables, including the state variables.
   (b) Define characteristics of the input variables. Make sure you cover all input variables.
   (c) SKIP
   (d) Partition the characteristics into blocks.
   (e) Define values for each block.