COMS 417: Software Testing (Fall 2017)

LAB 2

Graph Criteria, Black Box Tests,
and Parameterized Junit Tests

Due Sept 19, 12:30 p.m.

Objective and Outline

This lab has two parts. The first part focuses on CFG generated tests implemented as simple junit tests. The second part focuses on black-box tests implemented as parameterized junit tests.

For Part I you are to:

1) create a Control Flow Graph (CFG) to model a supplied utility method named getParametersFromFile();
2) use that CFG to create a test set which achieves the most complete edge-pair coverage you can find for the getParametersFromFile() method; and
3) create a non-parameterized junit test class named CfgTests for the test set you created;
4) execute the test class; and
5) create a defect report (see below) for any failures detected.

For Part II you are to:

1) define an input partitioning for the method countOs();
2) use that partitioning to design an all combinations test set for the method;
3) document the design;
4) enter the test cases required by your design into a spread sheet;
5) create a parameterized junit test class PartIITests that will execute all entries in your part II test set against countOs(); and
6) create a separate non-parameterized junit test class DefectsFound with individual tests to replicate any failures you detect.

General Instructions

We have supplied a template package for this lab. The template includes a demo package you can refer to for the basic structure of a Parameterized test. CfgTests will be similar, but *must* dynamically create the object representing the test set.

The template is complete except for the CfgTests, PartIITests, and DefectsFound. All test code *must* be placed in the test source directory. The src directory only contain the demo and util packages. *Do not* attempt to create your own implementation of Counter or Counter.countOs(). In part II, you *must* use the supplied jar as the software under test.

The required test design documents and defect reports are the most important part of this lab. You should expect them to account for about 75% of your grade. Do not omit them. Getting running/passing tests for this lab is *not* what this lab is about. Make certain the documents you turn in clearly demonstrate that you understand the requested test design processes.

Begin by examining the Javadoc in the template and correlating the classes and placeholders there with entities and deliverables mentioned in this document. Be sure you understand the intended role of each part.

While not absolutely necessary, we suggest you start with Part I, as writing the tests for getParametersFromFile() should assure you have a clear understanding of how it works, making it easier for you to incorporate it into PartIITests.

Do Not write your own file reading code for part II. You will lose points if you do *not* use getParametersFromFile() and only getParametersFromFile() to read your test set data.

You should submit your project as a zip. That zip must include the project file/package hierarchy and all artifacts requested in this document – each placed where directed.

Just to avoid counter-productive assumptions: *there are no aspects of Lab1 required to complete this lab!* This lab involves no understanding of RIPR and has no need of instrumentation in the software under test. This lab is about test set design using input-partitioning (part I), graph-based models(Part II), and Parameterized tests (Part II).

All test code *must* be in the test source folder.

There is no application for this lab – there are only classes containing methods that need testing. If you think you need to create a main() method to do this lab, please see the instructor or a TA immediately.

Warning: many programming editors automatically replace tab characters with the right number of spaces. Space delimited columns will not be recognized correctly by getParametersFromFile().

You should complete author tags in the files you write.
You should supply meaningful java doc for individual (non-parameterized) junit tests.
You should submit your complete project (minus bin directory and standard jars) as a zip file, with file hierarchy. We will take of points if we need to reorganize files or supply missing files or deal with huge downloads of stuff that is just a normal part of a java development environment.

Part I Notes:

In the template package (available via blackboard), static method getParametersFromFile() is a member of class FileUtil, located in package edu.iastate.cs417.lab2.util. You should use the source code for and any accompanying javadocs as the basis for your CFG and resulting test set.

Before you write the test class, you should produce a design document that captures your CFG, shows the basic block boundaries you identified in the source, shows the paths you are using to generate your test sets, and identifies the specific test cases associated with each path. Name this document “Part I test design” and include it the submitted project’s “doc” folder.

Your junit test class for Part I (CfgTests) will include one @Test annotation and test function for each test case in your Part I test set. Note, because you are testing a method which reads data from a file, you will need to include appropriate files for your tests to access. Depending on how you partition the input, you will probably only need a small number of these files to support the tests. (Think about the requirements given in the Javadoc and ask yourself "what would a file that triggered the behavior specified in requirement "xxxx" look like?)

Remember: correct behavior is the behavior specified in the requirements … not just behavior that passes or fails some test.

For Part I, FileUtil.getParametersFromFile() *is* the Software Under Test. Part I will not make *any* use of class Counter. Part I will not make *any* use of RunWith or Parameterized. While working on Part I, *pretend Part II and Counter do not exist*.

If any of your test cases uncover a failure, include a file named 'Part I Defects'. In that file, for each detected failure, give the input value which triggers the failure and an explanation of what you think is the underlying fault. If you find no failures, then all tests in CfgTests should complete "green".

Part II Notes:

Important: **If you find defects in Part I, *do not try to fix them for Part II.*** Use only the template's unmodified version of FileUtil.getParametersFromFile() when working on part II.

The countOs() method is supplied as a non-static member of class Counter, in the pre-compiled and obfusucated jar, counter.jar. This jar located in the template project's lib folder. For your test code to be able to execute this jar, the project must be configured to look for it. See the first two answers in this stackoverflow item for guidance:

https://stackoverflow.com/questions/3280353/how-to-import-a-jar-in-eclipse

For Part II, countOs() *is* the software under test and is the *only* software under test. Everything else is just file read and test support.

Because of how the Counter class is packaged, you do not have *any* access to the source code or design documents. Thus, the test set you create in Part II is inherently limited to black box tests. Your only information about this method and its requirements is supplied in the Javadoc description later in this document.

Before you write your test class, document the test design in a file named "Part II Test Design", placed in the project doc folder. The design document should give the characteristics (partitioning rules) used to partition each input, with block (or partition) names for each equivalence set, a table showing how you selected from these to achieve all combinations coverage, and a table showing what representative value from each equivalence set was used as a representative value when composing test cases.

The spread sheet containing your test cases should be three columns wide and have a single line header with the column names: “filename columns expected”. The original spread sheet should be saved in whatever is a native format for your spreadsheet application and placed in the “doc” directory. To use these test case values in test execution, you must export the contents of this file to a tab-delimited text file *named testdata.txt*, located at the root of the project.

You can refer to the parameterized junit test class in the demo package when creating your parameterized PartIITests class. Your class, however *must* use FileUtil.getParametersFromFile() to read the test cases in *testdata.txt* to construct the data object that will represent all the entries in your part II test set.

Important: You must use @RunWith(Parameterized.class) and @Parameterized. No points will be given for any other alternatives.

If your test set identifies failures (defects in countOs()), create a separate test class DefectsFound with separate unit tests specific to each failure.

Notice the package you must use to load Counter.

Class Counter Documentation

```
package edu.iastate.cs417.lab2.demo;

public Counter()
/**
 * Counter is meant (eventually), to collect
 * various string analysis functions into one place.
 *
 * At present Counter contains only one constructor and one method.
 */
/**
 * Default constructor
 */
public Counter()

/**
 * Counts the number of times the letter ‘O’ (oh)
 * appears in the input string (word).
 * Input strings must not contain a tab character.
 * The comparison is case-insensitive.
 *
 * @param word a string
 * @return the number of times ‘O’ (case insensitive) appears in word.
 *
 * throws
 *    IllegalInputException if word contains a tab character.
 *    NullPointerException if word is null.
 *
 */
public Integer countOs(String word);
```

Late Policy

Late labs will only be accepted for 24 hours after the deadline. All late labs will be penalized ten percent.

Academic Honesty

While it is a good practice to talk through your understanding of the lab with others, it is your responsibility to acknowledge work and solutions that are not of your own creation. If you received substantial assistance for some portion of this Lab, you should acknowledge that explicitly. You should *never* copy (or even use with minor changes) code written by someone else.

If there is reason to believe that you have copied all or portions of someone else's work, you will receive a zero for the assignment, and the event will be reported to the Dean of Students Office.

See sec. 4.2.1 of the student conduct code at http://policy.iastate.edu/policy/SDR#4.2.1 for the relevant policies.