(ECE453/CS447/ECE653/CS647/SE465)
Software Testing, Quality Assurance, and Maintenance
Assignment/Lab 1 (70 Points), Version 2

Instructor: Lin Tan
Release Date: January 21, 2013
Revision Date: January 25, 2013

Due: 5:00 PM, Thursday, Feb. 7, 2013
Submit: An electronic copy on LEARN

Please download a1-skeleton.tar.gz from the course website to get the necessary source code and test cases needed for finishing this assignment/lab.

I expect each of you to do the assignment independently. I will follow UW's Policy 71 if I discover any cases of plagiarism.

Submission Instructions:

Please read the following instructions carefully. **If you do not follow the instructions, you may be penalized up to 5 points.**

Electronic submission: Go to "Dropbox" $-\rangle$ "Submit A1" on LEARN. Submit only one file in .tar.gz format. Please name your file

<FirstName>-<LastName>-<StudentNo>.tar.gz

For example, use John-Smith-12345678.tar.gz if you are John Smith with student No 12345678. *You must include your file name as part of your submission comments.* The .tar.gz file should contain the following items:

- a single pdf file "a1_sub.pdf". You must include a cover page that contains your full name, 8-digit student No, the class number (one of ECE453, CS447, ECE653, CS647, & SE465), and your uwaterloo email address.
- a directory "q1" that contains your code (source code only; no binaries or object files) for Question 1
- "TestM.java" for Question 3, and
- "CFGTest.java" and "CFG.java" for Question 4.

You can submit multiple times. After submission, you can view your submissions to make sure you have uploaded the right files/versions.

Question 1 (10 points)

Write one simple program that prints out the value of -5 modulo 2 (e.g., -5%2 in C) in at least 4 programming languages (including scripting languages), e.g., C/C++, Java, Perl, and Python (You will write 4 programs in total). Report what you have found, and discuss at least 2 strategies to cope with such a problem. Hint: Think about how to change the developers to write different code, how to change the compilers, and how to change the standard, etc.

Question 2 (10 points)

Below is a faulty *Java* program, which includes a test case that results in failure. The if-statement needs to take account of negative values. A possible fix is:

```
if (x[i]%2 != 0)
```

Answer the following questions for this program.

(a) If possible, identify a test case that does not execute the fault.

(b) If possible, identify a test case that executes the fault, but does not result in an error state.

(c) If possible identify a test case that results in an error, but not a failure.

(d) For the given test case, identify the first error state. Be sure to describe the complete state.

```
public static int odd(int[] x)  {
// Effects: if x==null throw NullPointerException,
// else return the number of elements in x that are odd
        int count = 0;
        for (int i =0; i < x.length; i++) {
                if (x[i]%2==1) {
                        count++;
                }
        }
        return count;
}
// test: x = [-10, -9, 0, 99, 100]
// Expected: 2
```

Question 3 (20 points)

(Adapted from the original version by Patrick Lam.)

Consider the following (contrived) program:

```
class M {
        public static void main(String [] argv){
                M obj = new M();
                if (argv.length > 0)
                        obj.m(argv[0], argv.length);
        }

        public void m(String arg, int i) {
                int q = 1;
                A o = null;
                Impossible nothing = new Impossible();
                if (i == 0)
                        q = 4;
                q++;
                switch (arg.length()) {
                        case 0: q /= 2; break;
                        case 1: o = new A(); new B(); q = 25; break;
                        case 2: o = new A(); q = q * 100;
                        default: o = new B(); break;
                }
                if (arg.length() > 0) {
                        o.m();
                } else {
                        System.out.println("zero");
                }
                nothing.happened();
        }
}

class A {
        public void m() {
                System.out.println("a");
        }
}

class B extends A {
```

```
        public void m() {
                System.out.println("b");
        }
}

class Impossible{
        public void happened() {
                // "2b||!2b?", whatever the answer nothing happens here
        }
}
```

(a) Draw a Control Flow Graph (CFG) for method `M.m()` and be sure to include it in your a1.pdf. Use basic block CFG. See lecture notes on Structural Coverage and CFG for examples.

(b) List the sets of Test Requirements (TRs) with respect to the CFG you drew in part (a) for each of the following coverages: node coverage; edge coverage; edge-pair coverage; and prime path coverage. In other words, write four sets—TR_{NC}, TR_{EC}, TR_{EPC}, and TR_{PPC}. If there are infeasible test requirements, please list them separately and explain why they are infeasible.

(c) Using `TestM-skeleton.java` as a starting point, write one JUnit Test Class that achieves, for method `M.m()`, each of the following coverages: (1) node coverage but not edge coverage; (2) edge coverage but not edge-pair coverage; (3) edge-pair coverage but not prime path coverage; and (4) prime path coverage. In other words, you will write four test sets (groups of JUnit test functions) in total: one test set satisfies (1), one satisfies (2), one satisfies (3), and the last satisfies (4). For each test written, provide a simple documentation in the form of a few comment lines above the test function, listing which TR(s) is/are satisfied by the particular test. Consider feasible test requirements only for this part.

Our ECE Linux systems (ecelinux.uwaterloo.ca) have JUnit installed (in directory /opt/). If you have any questions regarding the access to these machines, please contact our lab instructor Bernie. If you use your own machine, you can get the JUnit jar file here: `https://github.com/downloads/KentBeck/junit/junit-4.8.2.jar`. JUnit is included in many standard Java code development environments such as Eclipse.

The discussion about JUnit 3 and JUnit 4 may be useful for Q3 and Q4: `http://www.ibm.com/developerworks/java/library/j-junit4.html`

Question 4 (30 points)

(Adapted from the original version by Sarfraz Khurshid)

You are to construct a partial[1] control-flow graph from the bytecode[2] of a given Java class using the ASM framework, version 4.0 (`http://asm.ow2.org`). You can download `http://download.forge.objectweb.org/asm/asm-4.0-bin.zip` and use all/asm-all-4.0.jar. If you use ecelinux machines, the jar file is in directory /opt/.

To illustrate, consider the following class C:

```
public class C {
    int max(int x, int y) {
        if (x < y) {
            return y;
        } else return x;
    }
}
```

which can be represented in bytecode as (e.g. the output of `javap -c`):

```
Compiled from "C.java"
public class ee379k.pset1.C extends java.lang.Object{
public C();
  Code:
   0:   aload_0
   1:   invokespecial   #8; //Method java/lang/Object."<init>":()V
   4:   return

int max(int, int);
  Code:
```

[1] This assignment ignores the labels that are traditionally annotated on nodes and edges, as well as the edges that correspond to method invocations or `jsr[ w]` bytecodes.

[2] `http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html`

```
    0:   iload_1
    1:   iload_2
    2:   if_icmpge        7
    5:   iload_2
    6:   ireturn
    7:   iload_1
    8:   ireturn
}
```

You can easily draw the corresponding control-flow graph.

Graph representation of control-flow. Implement the class `CFG` to model control-flow in a Java bytecode program. A `CFG` object has a set of nodes that represent bytecode statements and a set of edges that represent the flow of control (branches) among statements. Each node contains:

- an integer that represents the position (bytecode line number) of the statement in the method.
- a reference to the method (an object of `org.objectweb.asm.tree.MethodNode`) containing the bytecode statement; and
- a reference to the class (an object of class `org.objectweb.asm.tree.ClassNode`) that defines the method.

Represent the set of nodes using a `java.util.HashSet` object, and the set of edges using a `java.util.HashMap` object, which maps a node to the set of its neighbors. Ensure the sets of nodes and edges have values that are consistent, i.e., for any edge, say from node a to node b, both a and b are in the set of nodes. Moreover, ensure that for any node, say, n, the map maps n to a non-null set, which is empty if the node has no neighbours.

You can find a partial implementation of the `CFG` class in `CFG-skeleton.java`. You'll need to rename it to `CFG.java` to continue. Your first task is to fill in the implementation of this class. Then you will write JUnit test cases to achieve some coverage criterion.

(a) Adding a node (2 points). Implement the method `CFG.addNode` such that it creates a new node with the given values and adds it to `nodes` as well as initializes `edges` to map the node to an empty set. If the graph already contains a node that is `.equals` to the new node, `addNode` does not modify the graph.

(b) Adding an edge (2 points). Implement the method `CFG.addEdge` such that it adds an edge from the node `(p1, m1, c1)` to the node `(p2, m2, c2)`. Your implementation should update `nodes` to maintain its consistency with `edges` as needed. If the node does not exist in the graph, add the node.

(c) Deleting a node (4 points). Implement the method `CFG.deleteNode` such that it deletes a node with the given values, and all edges connected to the node, if any. If the graph does not contain a node that is `.equals` to the new node, `deleteNode` does not modify the graph. Your implementation should update `nodes` to maintain its consistency with `edges` as needed.

(d) Deleting an edge (4 points). Implement the method `CFG.deleteEdge` such that it deletes an edge from the node `(p1, m1, c1)` to the node `(p2, m2, c2)`.

(e) Reachability (6 points). Implement the method `CFG.isReachable` such that it traverses the control-flow graph starting at the node represented by `(p1, m1, c1)` and ending at the node represented by `(p2, m2, c2)` to determine if there exists any path from the given start node to the given end node. If the start node or the end node are not in the graph, the method returns false.

(f) Testing (12 points). I have also provided a class `CFGTest` in `CFGTest.java`, which exercises your `CFG` class. Examine this test class. Does it satisfy NC and EC for each of the method you write? Justify your answer. If it does not satisfy either coverage criterion, write additional JUnit test cases to satisfy one of them.