Software Testing, Quality Assurance & Maintenance (ECE453/CS447/SE465): Assignment 2 (v1)

Patrick Lam

Due: February 3, 2010

You may discuss the assignment with others, but I expect each of you to do the assignment independently. I will follow UW's Policy 71 if I discover any cases of plagiarism.

Question 1 (25 points)

(Original version by Sarfraz Khurshid, with modifications by Patrick Lam.)

Implement a class `TestGenerator` that generates tests for the `static` methods in a given class:

```
public class TestGenerator {
     public String createTests(String className) throws ClassNotFoundException {
        // base this implementation on CFGTest from A1.
     }
}
```

The output of the method `createTests` is a `String` representation of JUnit tests that, for each static method in the given class, checks the method against all combinations of previously defined default values for the argument types of the method. Also provide a method `main()` so that I can run your test generator.

Formally, let C be the given class, and m be a static method in C. Let the argument types of m be $T_1, \ldots, T_k$. Let the set of default values for type T_i be $S_i (1 \leq i \leq k)$. Then for method m, your implementation should generate $|S_1| \times \cdots \times |S_k|$ tests, one for each combination of the default values.

The following class `Domain` represents default values for the types that may appear as method argument types:

```
public class Domain {
     public static final int[] INT = new int[]{ -1, 0, 1 };
     public static final boolean[] BOOLEAN = new boolean[]{ false, true };
     public static final String[] OBJECT = new String[]{ "null", "new Object()" };
     public static final String[] STRING =
          new String[]{ null, "", "0", "Hello" };
}
```

Behaviour illustration

Here is an example of what your code should do. Consider the following class to be tested:

```
public class Demo {
     static void m(int x) {
          ...
     }
     static void n(boolean b, String s) {
          ...
     }
```

For class `Demo`, your implementation should generate 3 tests for method *m*:

```
     @Test public void test_m_1() { m(-1); }
     @Test public void test_m_2() { m(0); }
     @Test public void test_m_3() { m(1); }
```

and 8 tests for the method n, e.g.:

```
     @Test public void test_n_1() { n(false, null); }
     @Test public void test_n_2() { n(false, ""); }
     @Test public void test_n_3() { n(false, "0"); }
     @Test public void test_n_4() { n(false, "Hello"); }
```

Question 2 (50 points)

We will identify test requirements and augment a test suite to achieve prime path coverage of a specific method in the free Java application SweetHome3D, available at

`http://www.sweethome3d.eu.`

Download the source code. We will work with `applyFactorToTextSize()` in the `PlanController` class. Note that this class already has a JUnit test `PlanControllerTest` in the `tests` directory; you will extend this test class.

Creating a CFG (10 points). Draw the control-flow graph for the method `applyFactorToTextSize()`. It will be unmanageable unless you group together statements into basic blocks.

Identifying requirements for PPC (5 points). Identify the test requirements for prime path coverage in `applyFactorToTextSize()`.

Identifying requirements for ADC, AUC, ADUPC (10 points). Identify the test requirements for All-Defs Coverage, All-Uses Coverage, and All-du-paths Coverage in `applyFactorToTextSize()`.

Comparing coverage criteria (5 points). Compare the sets of test requirements you've collected. Point out strengths and weaknesses of these criteria, for instance by giving scenarios where one criterion is going to help you find something that another criterion wouldn't; discuss whether the additional number of test requirements is worth it in this case.

Creating tests (20 points). Create tests in `PlanControllerTest` which achieve prime path coverage in `applyFactorToTextSize()`. Note that your test method `applyFactorToTextSize()` is private. Don't change this (you don't need to call the method directly).