In-class 8: Graph Coverage for Design Elements
Due 25-October-2018 by 11:59pm

Names:

Purpose: Understand and apply graph coverage to test various parts of the software design
Instruction: Work with your neighbors in groups.

Consider the trashgraph class

```java
// Modified from https://cs.gmu.edu/~offutt/softwaretest/java/TrashAndTakeOut.java

public class trashgraph {
    public static void trash(int x) {
        int m, n;
        m = 0;
        if (x > 0) m = 4;
        if (x > 5) n = 3 * m;
        else n = 4 * m;
        int o = takeOut(m, n);
        System.out.println("o is: " + o);
    }

    public static int takeOut(int a, int b) {
        int d, e;
        d = 42 * a;
        if (a > 0) e = 2 * b + d;
        else e = b + d;
        return(e);
    }
}
```

1. Locate a call site between trash() and takeOut() methods. You may circle directly in the source code.

2. Write down all variables that are shared or passed between trash() and takeOut() methods.

3. Find all the last-defs and first-uses. You may circle directly in the source code. Write down whether that circle is for last-def or first-use.
4. Draw a graph representation of the `trash()` and `takeOut()` method. Be sure to include coupling between the caller and the callee.

5. Find all the **coupling DU-pairs**.
   Write them down as pairs of triplets. Each triplet is a method name, variable name, and node. The pair has the **last-def** on the left and a **first-use** on the right. For example:
   
   `(method1(), var1, node1) – (method2(), var2, node2)`
   
   where last-def of a variable `var1` is in method `method1` at node `node1` and first-use of a variable `var1` appears as a variable `var2` in method `method2` at node `node2`.
   
   Note: in general, if a graph is unavailable or has not been created, each triplet can be written as
   
   `(method1(), var1, line1) – (method2(), var2, line2)`
   
   where last-def of a variable `var1` is in method `method1` at line number `line1` and first-use of a variable `var1` appears as a variable `var2` in method `method2` at line number `line2`. 
6. List test requirements (i.e., Coupling DU-paths for each variable) that satisfy **All-Coupling-Defs** coverage

7. List test requirements (i.e., Coupling DU-paths for each variable) that satisfy **All-Coupling-Uses** coverage

8. List test requirements (i.e., Coupling DU-paths for each variable) that satisfy **All-Coupling-DU-Paths** coverage

[optional] For more hands-on experience, automate your test set (in JUnit) and run it against the `trashgraph` class.

**Grading rubric**

[Total: 10 points]: Done (or provide evidence of your attempt)

**Submission**

Submit the paper in class before you leave or take screen shots of your in-class exercise and submit them to Collab/inclass8. **Everyone submits this in-class exercise**, even if you work with partners.