EE360T/EE382V: Software Testing
Problem Set 5

Out: Apr 4, 2018; **Due: Apr 16, 2016 11:59pm**
Submission: *.zip via Canvas
Maximum points: 40

1 Implementing a graph data structure

Consider the following partial implementation of a graph data structure:

```
package pset5;

import java.util.Arrays;
import java.util.Set;

public class Graph {
    private int numNodes; // number of nodes in the graph

    private boolean[][] edges;
    // edges[i][j] is true if and only if there is an edge from node i to node j

    // class invariant: fields "edges" is non-null;
    //                  "edges" is a square matrix;
    //                  numNodes is number of rows in "edges"

    public Graph(int size) {
        numNodes = size;

        // your code goes here
        // ...

    }

    public String toString() {
        return "numNodes: " + numNodes + "\n" + "edges: " + Arrays.deepToString(edges);
    }

    public boolean equals(Object o) {
        if (o.getClass() != Graph.class) return false;
        return toString().equals(o.toString());
    }

    public void addEdge(int from, int to) {
        // postcondition: adds a directed edge "from" -> "to" to this graph

        // your code goes here
        //...

    }

    public boolean reachable(Set<Integer> sources, Set<Integer> targets) {
        if (sources == null || targets == null) throw new IllegalArgumentException();

        // postcondition: returns true if (1) "sources" does not contain an illegal node,
```

```
        //                          (2) "targets" does not contain an illegal node, and
        //                          (3) for each node "m" in set "targets", there is some
        //                          node "n" in set "sources" such that there is a directed
        //                          path that starts at "n" and ends at "m" in "this"; and
        //                          false otherwise

        // your code goes here
        //...

    }
}
```

1.1 Implementing `Graph` [2 points]

Implement the constructor `Graph` as specified. Make sure your implementation satisfies the class invariant for `Graph` (as given in comments).

1.2 Implementing `addEdge` [4 points]

Implement the method `addEdge` as specified. Make sure your implementation satisfies the class invariant for `Graph` (as given in comments).

1.3 Implementing `reachable` [9 points]

Implement the method `reachable` (and any helper methods you need) as specified.

2 Testing your graph implementation [25 points]

Implement a test suite to test the `addEdge` and `reachable` methods in the following class `GraphTester` as specified:

```
package pset5;

import static org.junit.Assert.*;
import java.util.TreeSet;
import java.util.Set;
import org.junit.Test;

public class GraphTester {
    // tests for method "addEdge" in class "Graph"
    @Test public void tae0() {
        Graph g = new Graph(2);
        g.addEdge(0, 1);
        System.out.println(g);
        assertEquals(g.toString(), "numNodes: 2\nedges: [[false, true], [false, false]]");
    }

    // your tests for method "addEdge" in class "Graph" go here

    // you must provide at least 4 test methods;
    // each test method has at least 1 invocation of addEdge;
    // each test method creates exactly 1 graph
    // each test method creates a unique graph w.r.t. "equals" method
    // each test method has at least 1 test assertion;
    // your test methods provide full statement coverage of your
    //   implementation of addEdge and any helper methods
    // no test method directly invokes any method that is not
    //   declared in the Graph class as given in this homework
```

```
    // ...


    // tests for method "reachable" in class "Graph"

    @Test public void tr0() {
        Graph g = new Graph(1);
        Set<Integer> nodes = new TreeSet<Integer>();
        nodes.add(0);
        assertTrue(g.reachable(nodes, nodes));
    }

    // your tests for method "reachable" in class "Graph" go here

    // you must provide at least 6 test methods;
    // each test method must have at least 1 invocation of reachable;
    // each test method must have at least 1 test assertion;
    // at least 2 test methods must have at least 1 invocation of addEdge;
    // your test methods must provide full statement coverage of your
    //  implementation of reachable and any helper methods
    // no test method directly invokes any method that is not
    //  declared in the Graph class as given in this homework

    // ...

}
```