EE360T/EE382V: Software Testing
Problem Set 1

Out: Feb 3, 2016; **Due: Feb 13, 2016 11:59pm**
Submission: *.zip via Canvas
Maximum points: 40

1 Testing data structures

Consider the following implementation of a singly-linked list data structure, which represents a sorted container for integer values:

```
package pset1;

import java.util.HashSet;
import java.util.Set;

public class SLList {
    Node header;

    static class Node {
        int elem;
        Node next;
    }

    boolean repOk() {
        // postcondition: returns true iff <this> is an acyclic list, i.e.,
        //                          there is no path from a node to itself,
        //                          and the elements are sorted (<=)

        Set<Node> visited = new HashSet<Node>();
        Node n = header;
        while (n != null) {
            if (!visited.add(n)) {
                return false;
            }
            if (n.next != null) {
                if (n.next.elem > n.elem) return false
            }
            n = n.next;
        }
        return true;
    }

    void add(int e) {
        // precondition: this.repOk()
        // postcondition: adds <e> in a new node in the list in sorted order;
        //                the rest of the list is unmodified

        // your code goes here

    }
}
```

1.1 Implementing `add` [4 points]

Implement the method `add` as specified.

1.2 Testing `add` [6 points]

Implement the two test methods in the following class `SLListAddTester` as specified:

```
package pset1;

import static org.junit.Assert.*;
import org.junit.Test;

public class SLListAddTester {
    @Test public void test0() {
        SLList l = new SLList();
        assertTrue(l.repOk());
        l.add(0);

        // write a sequence of assertTrue method invocations that
        // perform checks on the values for all the declared fields
        // of list and node objects reachable from l

        assertTrue(l.header != null);
        // your code goes here

    }

    @Test public void test1() {
        SLList l = new SLList();
        assertTrue(l.repOk());
        l.add(0);
        assertTrue(l.repOk());
        l.add(1);
        assertTrue(l.repOk());

        // write a sequence of assertTrue method invocations that
        // perform checks on the values for all the declared fields
        // of list and node objects reachable from l

        assertTrue(l.header != null);
        // your code goes here
    }
}
```

1.3 Testing `repOk` [10 points]

Consider testing the method `repOk` by writing a test suites that consists of valid or invalid lists. Specifically, implement test methods in the following class `SLListRepOkTester` such that: (1) each test allocates exactly one list `l`; (2) each test method makes exactly one invocation `l.repOk()`; (3) each test method invokes `assertTrue(l.repOk())` or `assertFalse(l.repOk())` as its last statement; (4) no invocation of `add` is made in any test method; (5) the test suite as a whole consists of all singly-linked list data structures – whether acyclic or not – that can possibly be constructed using up to 2 nodes with integer values 0 and 1.

```
package pset1;

import static org.junit.Assert.*;
import org.junit.Test;
import pset1.SLList.Node;
```

```
public class SLListRepOkTester {
    @Test public void t0() {
        SLList l = new SLList();
        assertTrue(l.repOk());
    }

    @Test public void t1() {
        SLList l = new SLList();
        Node n = new Node();
        // your code goes here

    }

    // your code goes here
}
```

2 Textbook execrises

Solve the following problems from the Software Testing textbook:

1. [6 points] Exercises – Section 2.2.1 Question 5 (Page 43)
2. [7 points] Exercises – Section 2.2.1 Question 6 (Page 43)
3. [7 points] Exercises – Section 2.3 Question 1 (Pages 60–61)