EE360T/EE382C-16: Software Testing

Problem Set 1

Out: Sep 7, 2022; Due: Sep 20, 2022 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 sequential

container for boolean values:

```
package pset1;

import java.util.HashSet;

import java.util.Set;

public class SLList {

Node first; // first node in <this> list

Node last; // last node in <this> list

static class Node {

boolean 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

if (first == null || last == null) {

return first == last;

}

Set<Node> visited = new HashSet<Node>();

Node n = first;

while (n != null) {
```

```
if (!visited.add(n)) {
return false;
}
if (n.next == null) {
return n == last;
}
n = n.next;
}
return true;
}
void add(boolean e) {
// precondition: this.repOk()
// postcondition: adds <e> in a new node at the end of <this>
// list; the rest of <this> list is unmodified
// your code goes here
Node newNode = new Node();
    newNode.elem = e;
    newNode.next = null;
    if((this.first == null) && (this.last == null)){
      this.first = newNode;
      this.last = newNode;
    } else if (this.first.next == null){
      this.first.next = newNode;
      this.last = newNode;
    }else {
      this.last.next = newNode;
      this.last = newNode;
    }
}
```

```
}
```

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(true);
// 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.first != null);
// your code goes here
assertTrue(l.last != null);
    SLList.Node node = l.first;
    while(node != null){
      assertTrue(node.elem);
      assertTrue(node.next != null);
      node = node.next;
    }
}
@Test public void test1() {
SLList l = new SLList();
assertTrue(l.repOk());
l.add(true);
assertTrue(l.repOk());
l.add(false);
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.first != null);
// your code goes here
assertTrue(l.last != null);
    SLList.Node node = l.first;
    while(node != null){
      assertTrue(node.elem);
      assertTrue(node.next != null);
      node = node.next;
    }
}
}
```

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.

```
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
l.first = n;
    assertFalse(l.repOk());
}
// your code goes here
@Test public void t2() {
    SLList l = new SLList();
    Node n = new Node();
    // your code goes here
    l.last = n;
    assertFalse(l.repOk());
  }

  @Test public void t3() {
    SLList l = new SLList();
    Node n = new Node();
    // your code goes here
    l.first = n;
    l.last = n;
    assertTrue(l.repOk());
  }

  @Test public void t4() {
    SLList l = new SLList();
    Node n = new Node();
    // your code goes here
    n.next = n;
    l.first = n;
    l.last = n;
    assertFalse(l.repOk());
  }
}
```

2 Testing contracts

Consider the following code snippet that declares a class C:

```
package pset1;
public class C {
int f;
public C(int f) {
this.f = f;
}
@Override
public boolean equals(Object o) {
// assume this method is implemented for you
}
@Override
public int hashCode() {
// assume this method is implemented for you
}
}
```

Consider next the following code snippet that declares a class D as a subclass of C:

```
package pset1;
public class D extends C {
int g;
public D(int f, int g) {
super(f);
this.g = g;
}
@Override
public boolean equals(Object o) {
// assume this method is implemented for you
}
@Override
public int hashCode() {
// assume this method is implemented for you
}
}
```

2.1 Testing equals [15 points]

According to the contract for java.lang.Object any correct Java program must satisfy certain properties with respect to the equals methods; these properties include1:

P1: For any non-null reference value x, x.equals(null) should return false;

P2: It is reflexive: for any non-null reference value x, x.equals(x) should return true;

P3: It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true; and

P4: It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true;

You are to implement a test suite that checks three of the four properties – namely, P1, P2, and P3 – with

respect to the equals methods implemented in the three classes pset1.C, pset1.D, and java.lang.Object.

Specifically, implement test methods in the following class EqualsTester such that: (1) each test method has

exactly one invocation of assertTrue(...) or assertFalse(...); (2) each property is tested with respect

to each of the three equals methods, e.g., the test suite must have three test methods for P1; (3) each

property is tested with respect to each combination of the three object types (C, D, or Object) for the inputs

to equals, e.g., the test suite must have at least nine tests for P3:

```
package pset1;
import static org.junit.Assert.*;
import org.junit.Test;
public class EqualsTester {
/*
* P1: For any non-null reference value x, x.equals(null) should return false.
*/
@Test public void t0() {
assertFalse(new Object().equals(null));
}
// your test methods for P1 go here
@Test public void t1() {
    assertFalse(new C(1).equals(null));
  }

  @Test public void t2() {
    assertFalse(new D(1, 2).equals(null));
    }
/*
```

```
* P2: It is reflexive: for any non-null reference value x, x.equals(x)
* should return true.
*/
// your test methods for P2 go here
@Test public void t3() {
    Object o = new Object();
    assertTrue(o.equals(o));
  }

  @Test public void t4() {
    C c = new C(2);
    assertTrue(c.equals(c));
  }

  @Test public void t5() {
    D d = new D(2, 3);
    assertTrue(d.equals(d));
  }
/*
* P3: It is symmetric: for any non-null reference values x and y, x.equals(y)
* should return true if and only if y.equals(x) returns true.
*/
// your test methods for P3 go here
@Test public void t6() {
    Object o = new Object();
    C c = new C(3);
    if(c.equals(o)) assertTrue(o.equals(c));
  }
```

```
@Test public void t7() {

    Object o = new Object();

    D d = new D(3, 4);

    if(d.equals(o)) assertTrue(o.equals(d));

}


@Test public void t8() {

    Object o1 = new Object();

    Object o2 = new Object();

    if(o1.equals(o2)) assertTrue(o2.equals(o1));

}


@Test public void t9() {

    C c = new C(4);

    Object o = new Object();

    if(o.equals(c)) assertTrue(c.equals(o));

}


@Test public void t10() {

    C c = new C(5);

    D d = new D(5, 6);

    if(d.equals(c)) assertTrue(c.equals(d));

}


@Test public void t11() {

    C c = new C(6);

    C c2 = new C(7);

    if(c2.equals(c)) assertTrue(c.equals(c2));

}
```

```
  @Test public void t12() {
    D d = new D(4, 5);
    Object o = new Object();
    if(o.equals(d)) assertTrue(d.equals(o));
  }

  @Test public void t13() {
    D d = new D(5, 6);
    C c = new C(5);
    if(c.equals(d)) assertTrue(d.equals(c));
  }

  @Test public void t14() {
    D d = new D(6, 7);
    D d2 = new D(7, 8);
    if(d2.equals(d)) assertTrue(d.equals(d2));
  }
/*
* P4: It is transitive: for any non-null reference values x, y, and z,
* if x.equals(y) returns true and y.equals(z) returns true, then
* x.equals(z) should return true.
*/
// you do not need to write tests for P4
}
```

2.2 Testing hashCode [5 points]

The contract for java.lang.Object additionally requires the following property that relates equals and hashCode1:

P5 : If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

Implement test methods in the following class HashCodeTester such that: (1) each test method has exactly one invocation of assertTrue(...) or assertFalse(...); (2) the property is tested with respect to each combination of the three object types (C, D, or Object) for the inputs to equals, so the test suite must have at least nine tests:

```
package pset1;
import static org.junit.Assert.*;
import org.junit.Test;
public class HashCodeTester {
/*
* P5: If two objects are equal according to the equals(Object)
* method, then calling the hashCode method on each of
* the two objects must produce the same integer result.
*/
// your test methods go here
@Test public void t0() {
    Object o = new Object();
    C c = new C(3);
    if(c.equals(o)) assertTrue(c.hashCode() == o.hashCode());
}

@Test public void t1() {
    Object o = new Object();
    D d = new D(3, 4);
    if(d.equals(o)) assertTrue(d.hashCode() == o.hashCode());
}

@Test public void t2() {
    Object o1 = new Object();
    Object o2 = new Object();
    if(o2.equals(o1)) assertTrue(o2.hashCode() == o1.hashCode());
}

@Test public void t3() {
    C c = new C(4);
    Object o = new Object();
    if(o.equals(c)) assertTrue(o.hashCode() == c.hashCode());
}

@Test public void t4() {
    C c = new C(5);
    D d = new D(5, 6);
    if(d.equals(c)) assertTrue(d.hashCode() == c.hashCode());
}

@Test public void t6() {
    C c = new C(6);
    C c2 = new C(7);
    if(c2.equals(c)) assertTrue(c2.hashCode() == c.hashCode());
}

@Test public void t7() {
    D d = new D(4, 5);
    Object o = new Object();
    if(o.equals(d)) assertTrue(o.hashCode() == d.hashCode());
}

@Test public void t8() {
    D d = new D(5, 6);

    C c = new C(5);

    if(c.equals(d)) assertTrue(c.hashCode() == d.hashCode());

}


@Test public void t9() {

    D d = new D(6, 7);

    D d2 = new D(7, 8);

    if(d2.equals(d)) assertTrue(d2.hashCode() == d.hashCode());

}

}
```