From: **Alan Liang** alan.5100@yahoo.com
Subject: lab09
Date: October 27, 2016 at 4:33 PM
To: Winnie Liang winnieliang27@yahoo.com

AL

lab09 a53

lab09

Due today at 11:59 pm

This lab will have very little instructions. Instead we give you JUnit test cases and you must write the code that satisfies these tests.

This lab is all about recursion. What is recursion? Recursion is the solving of a problem by using the solutions of smaller subproblems. What is the fifth Fibonacci number? It is the sum of the fourth and third Fibonacci number. What is the fourth Fibonacci number? and so on...

Important: You are not allowed to use (Array)Lists or arrays (except in the `getElems` method of `BinarySearchTree`). You are not allowed to use for/while/do-while loops; only recursion. Points will be deducted for failure to follow this.

You'll only need one class: `BinarySearchTree`, with the following methods:

- `public void insert(int i)`
- `public Integer find(int i)`
- `public List<Integer> getElems()`
    - HINT: this method will be helpful

A BinarySearchTree is a often-used structure in computer science. First, let's establish what a binary tree is, then we'll talk about a binary search tree. A binary tree is either:

- a node with a value, called the key, and at most two child trees, call them the left and right children or the left and right subtrees.
- empty

This is a recursive defintion: a binary tree is defined in terms of further binary trees. A valid binary tree could have 0 children, 1 child (either left or right), or 2 children (both left and right). These children could themselves have between 0 and 2 children and so on.

A binary search tree is a binary tree which satisfies an additional search property. The property is that the key of each node has a value which is

- strictly greater than all the keys in the left subtree
- strictly less than all the keys in the right subtree

Thus, binary search trees have no duplicates. This property needs to be maintained at all times, but greatly simplifies searching for elements (why?) and allows other nice things.

We'll make it easier and only have our `BinarySearchTree` store integers. That way, you can use the familiar <, >, and == operators for comparison. `BinarySearchTree` will have a `private Integer value` field, in addition to others. It is crucial the type of `value` is `Integer` and not `int`.

Here is the tester `BinaryTreeTester`. Read the tester carefully to understand how it expects the

Here is the tester, `BinaryTreeTester`. Read the tester carefully to understand how it expects the methods of `BinarySearchTree` to work. The lab is complete when your code, with the tester, compiles and all tests pass.

```
package lab09;

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

import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

public class BinaryTreeTester {
    List<Integer> list;
    BinarySearchTree bst;

    @Before
    public void setup() {
        list = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5, 0, -1,
2, 3));
        bst = new BinarySearchTree();
        for (int i : list) {
            bst.insert(i);
        }
    }

    @Test
    public void insertFound() {
        for (int i : list) {
            assertNotNull(bst.find(i));
        }
    }

    @Test
    public void othersNotFound() {
        assertNull(bst.find(6));
        assertNull(bst.find(-3));
        assertNull(bst.find(-4));
    }

    @Test
    public void elemsFound() {
        List<Integer> elems = bst.getElems();
        for (int i : elems) {
            assertNotNull(bst.find(i));
        }
    }

    @Test
    public void elemsWereAdded() {
        List<Integer> elems = bst.getElems();
        assertEquals(7, elems.size());
```

```
        assertEquals(7, elems.size());
        for (int i : elems) {
            assertNotNull(list.contains(i));
        }
    }

    @Test
    public void elemsCorrect() {
        List<Integer> elems = bst.getElems();
        for (int i : list) {
            assertTrue(elems.contains(i));
        }
    }

    @Test
    public void elemsInSortedOrder() {
        List<Integer> elems = bst.getElems();
        for (int i = 1; i < elems.size(); i++) {
            assertTrue(elems.get(i-1) < elems.get(i));
        }
    }
}
```