[**Department of Computer Science**](http://www.cs.umd.edu/)

**CMSC132** Fall 2016

**Project:** Binary Search Tree

**Due Date:** Friday 11/11, 11:00 pm

**Assignment Type:** Closed

## Overview

You will implement the methods of the [EmptyTree](http://docs.google.com/doc/searchTree/EmptyTree.html) and [NonEmptyTree](http://docs.google.com/doc/searchTree/NonEmptyTree.html) classes. We have provided a [Tree](http://docs.google.com/doc/searchTree/Tree.html) interface and a partial implementation of the [SearchTreeMap](http://docs.google.com/doc/searchTree/SearchTreeMap.html) class (you must implement the keyList and subMap methods). Complete documentation for these classes can be found at [javadoc documentation](http://docs.google.com/doc/index.html).

## Objectives

The objective of this project is to implement a polymorphic binary search tree. This project is designed to help you develop your skills at recursion, polymorphism and testing.

## Grading

* (90%) Automated Tests
  + (10%) Public JUnit tests
  + (48%) Release JUnit tests
  + (32%) Secret JUnit tests
* (10%) Style

## Binary Search Tree Specifications

### Testing

As always, you are expected to develop test cases that test your code. You only get a few test cases in the class PublicTests, and the release tests all have unhelpful names such as testOne and testTwo.

In addition to the release tests, there are additional secret test cases. Don't assume that just because your implementation passes the release tests, it is correct.

The WordCountApplet is not really part of this assignment, but it has been provided for fun. This applet uses a SearchTreeMap to count the number of times each word appears in the text file specified by a URL. You may use the applet to gauge how well your SearchTreeMap implemention works. The default URL is a text file containing the complete works of Shakespeare (give it a few seconds to load after pressing the button!) Of course you may enter any URL you wish into the box.

### Design

The SearchTreeMap class implements some of the functionality of the Map interface (although not all). A SearchTreeMap object is just a wrapper around a Tree, which is actually used to implement binary search trees.

Note that the insert and delete methods on Tree objects return references to Tree objects. In many cases, these functions may return a reference to the **this** object. However, in some cases they can't. For example. EmptyTree.getInstance.insert("a", "1") has to return an instance of an NonEmptyTree object.

### Design and Implementation Restrictions

* **You may not use any form of looping construct anywhere in this project. If you do, your grade on the entire project will be 0. No for-loops, no while-loops, no do-while-loops, no for-each-loops!**
* You may not use any arrays.
* You may not use any sorting algorithm anywhere at any time.
* You may not explicitly check a Tree to see whether it is an EmptyTree or NonEmptyTree.
* Your EmptyTree and NonEmptyTree implementations may not have any comparisons against the null value.
* You should not need to do any casting for this project. If you are casting then you are probably not using generic programming correctly.
* If you insert multiple values with the same key, only the value associated with the most recent put( ) will be saved in the map.
* You may not use any classes from java.util except ArrayList and only to implement the keyList( ) method.
* You may not check whether a tree is empty by using comparisons similar to the following:
  + left==EmptyTree.getInstance()
  + tree.size()==0
  + Other comparisons similar to the above

You are expected to use polymorphism (and exception handling, where appropriate) to handle the differences between empty and nonempty trees. Failure to do so will result in a large negative adjustment to your project grade.

* You may not use instanceof.
* You may not use getClass().
* The delete method must use the approach described in the lecture slides, or something equally efficient. You may not implement delete by creating a new tree and inserting all the keys from the source tree except the one you want to delete.
* For the Subtree method you may not create an empty tree and traverse the whole tree inserting only those entries within the specified range. If a simple check will tell you that an entire subtree can be excluded, your implementation should not traverse that subtree.

**The above restrictions do not apply to your test cases; you may write them however you wish.**

## Suggestions on How to Start/Implement This Programming Assignment

* Understand how a traditional (non-polymorphic) tree works.
* Check the polymorphic list implementation we discussed in lecture.

[Web Accessibility](https://www.umd.edu/web-accessibility)