Tony Diep
03/02/17
CS 2420
Design Document

Design Document: Spell Checker Using the Binary Search Tree (BST)

The purpose of this project is to practice writing and using the BST and apply the BST to a real life program (i.e. the spell checker program used in many famous software programs such as Microsoft Word). The project will continue to use generics, recursion, and effective testing strategies. The BST is an effective data structure because of fast insertions and searching implementations. However, the down side to the BST is adding and removing elements.

There are several classes to be implemented for this project along with a couple of given classes as starting points. We are already given three classes: **SortedSet, SpellChecker,** and **SpellCheckerDemo**. The SpellChecker and SpellCheckerDemo classes depend on the **BinarySearchTree** class (which contains a static helper class called **Node**). The Node class is similar to the other Node class from the last assignment since we previously studied Linked Lists. The SortedSet file is an interface that represents a simpler version of Java's SortedSet interface. In essence, the SortedSet interface provides a list of all methods that are required for any class to be a set. The set can be used in this project to keep track of all of the words that are spelled correctly versus misspelled words. The SpellChecker class encompasses a dictionary and provides functions and operations we can use to retrieve words in the dictionary. The SpellCheckerDemo is a class that demonstrates the spell checker program and thus contains the main method. As always, we have a Timing class that performs timing experiments for our spell checking program. The BinarySearchTree class (with the Node static class inside) is the workhorse of the whole project. If the implementations are incorrect in this class, then the whole program will not work correctly.

We will try to change the way we approach on working on assignments. For instance, It is essential that we focus on the preliminaires first. JUnit Tests are essential to implement first so that we can get a good idea of expected behaviors for the BST functionalities. Thus, we will try and first implement JUnit Tests first and then provide pictures using the Draw.io website. Once we are able to complete our JUnit Tests and drawings, we will then move on to implementing the Node class and go on to implementing the BinarySearchTree class. We hope that this approach will reduce the amount of time it takes to completing the whole assignment. Again, it is imperative that we understand conceptually how the BST works before implementing any code in the Node or BinarySearchTree class.