SYSC 2100 Algorithms and Data Structures
Winter 2020
Assignment 5: Tables and Trees
Due: April 7, 2020

You are required to implement a way to manage a Table/Dictionary, with operations on that ADT specified in the interface *Dictionary.java*. The Table stores *Strings*, referenced by *SortableStrings* (Strings that implement the *Sortable* interface). Your implementation should internally use a Binary Search Tree (BST). The trees in turn are to be implemented using references.

The course cuLearn page provides you a source file *DictionaryTest.java* with the main routine that instantiates a dictionary and invokes the methods defined in the interface *Dictionary* to add and deletes entries, print the tree, search for entries, and to print the depth of the underlying binary tree. The course cuLearn page also provides the interface definition *Dictionary.java*. You have to provide an implementation of *BSTDictionary*, which implement this interface. To get you started, the course page also provides source files for the *Sortable* interface and the *SortableString* class, as well as the source for a *BSTNode*.

Binary Search Trees are described in the textbook in Section 11.3, starting on page 551 (2nd edition) or page 594 (3rd edition). Note that unlike the discussion in the textbook, the *Dictionary* interface manages a table where the search key of type *K* is a separate data item from the entry of type *E* (with both being stored in a tree node). Also, the key type *K* is not of type *KeyedItem* but has to implement/extend the interface *Sortable*.

Bonus part (For the champions! No marks allocated):
Implement Table/Dictionary with AVL trees. AVL trees are discussed in the textbook in Section 13.1, starting at page 689/page 700. We will only discuss them briefly in class. A more in-depth description of AVL trees is available at http://en.wikipedia.org/wiki/AVL_tree. The class *AVLDictionary* should build and maintain a balanced binary search tree. To get you started on this part, the course website provides the source for a class *AVLNode*. To test and compare the two different dictionary implementations, use the *DictionaryAdvancedTest.java* source file. When running the main routine, you should be able to verify that the depth of the AVL tree is significantly less than the depth of the BST tree.

Note: the TA(s) will test your submission by using multiple dictionaries in parallel. For example, if you only implemented the *BSTDictionary*, they will use the *DictionaryAdvancedTest* program with your implementation of the *BSTDictionary* and the sample solution for the *AVLDictionary*. As discussed in the comments in *DictionaryAdvancedTest*, while the trees may have differing depths, the *printTree()* method should properly generate a sorted output of a tree with all elements. And since both trees will have the same add and delete operations applied to them, the search for specific entries should yield the same results (i.e., a specific entry either exists in both dictionaries or in none).

Submission Requirements: Submit your assignment (the source files) using ***cuLearn***. Your program should compile and run as is in the default lab environment, and the code should be well documented. Submit all files without using any archive or compression as separate files. Submit your solution by providing the source files for *BSTDictionary* and other auxiliary classes. When combined with the source files provided on ***cuLearn***, your submission should compile and run as is, and the code should be well documented.

Marks will be based on:
- Completeness of your submission
- Correct solution to the problem
- Following good coding style

- Sufficient and high-quality in-line comments
- Adhering to the submission requirements (in particular the naming convention)

The due date is based on the time of the ***cuLearn*** server and will be strictly enforced. If you are concerned about missing the deadline, here is a tip: multiple submissions are allowed. So you can always submit a (partial) solution early, and resubmit an improved solution later. This way, you will reduce the risk of running late, for whatever reason (slow computers/networks, unsynchronized clocks, failure of the Internet connection at home, etc.).
In ***cuLearn***, you can manage the submission until the deadline, taking it back, deleting/adding files, etc, and resubmitting it. The system also provides online feedback whether you submitted something for an assignment. It may take a while to learn the submission process, so I would encourage you to experiment with it early and contact the TA(s) in case you have problems, as only assignments properly and timely submitted using ***cuLearn*** will be marked and will earn you assignment credits.