Start Assignment

Due Apr 28 by 11:59pm **Points** 48 **Submitting** a file upload **File Types** java
Available Mar 10 at 12am - Apr 28 at 11:59pm about 2 months

Unit 7: Linked Lists, Binary Search Trees, Stacks, and Queues **7 of 8**

9 Assignment 9 - Spell Checker

Computer SCIENCE

IMPORTANT: Please review the statement on Academic Integrity. Submitting code, without permission, that is not your own may result in an automatic F for the class: Statement on Academic Integrity

For this assignment you are developing a spell checker. The binary search tree is the underlying data structure you'll use to make the performance of testing each word in a document to be very fast. You are supplied a dictionary of about 230,000 words that you'll load into your search tree and then use those words to test a document for misspelled words.

Assignment

Create a Generics based binary search tree (BST), then use this tree as the storage for a dictionary of words used to check the spelling of documents; *see the Notes section below for an important item about this code*. The binary tree must have the following characteristics:

- It must be a binary search tree, relying on the order of the `Comparable` interface for data type used for the values of the tree.
- Public methods:
  - `boolean insert(E value)` : inserts a value into the tree, does not allow duplicates. If the value is already in the tree do not add the duplicate and return `false`, `true` otherwise.
  - `boolean remove(E value)` : deletes a value from the tree. If the value was in the tree and deleted, return `true`, otherwise, return `false`.
  - `boolean search(E value)` : tests to see if a value exists in the tree.
  - `void display(String message)` : Performs an in-order traversal of the tree, displaying the value at each node. Use the `message` parameter as a header/title for the traversal.
  - `int numberNodes()` : returns a count of all nodes in the tree.
  - `int numberLeafNodes()` : returns a count of all the leaf nodes in the tree.
  - `int height()` : returns the height of the tree.

You are provided with a driver file, SpellChecker.java. Update this driver to read the dictionary "dictionary.txt", where each word is on a separate line, and stores these words into your binary search tree. After reading the dictionary into your tree, have your program report the number of nodes, the number of leaf nodes, and height (depth) of the tree.

The following is an example report:

```
-- Tree Stats --
Total Nodes: 234371
Leaf Nodes:  78025
Tree Height: 43
```

*It **is** expected the tree height and number of leaf nodes will be slightly different each time you run the program if you use the randomization technique (described below) when reading in the words.*

Finally, read the file "letter.txt" and output to the console all the misspelled words (***by definition, for this class, any word not found in the dictionary is considered misspelled***), keep in mind the graders may use a different file when doing the grading. You'll need to remove casing (capitalization) from the words and remove any punctuation characters from the words in the letter; the following characters should be sufficient: " , : . ! ? ( )

One problem you need to resolve: The dictionary is in alphabetic order. If you insert the words in alphabetic order, your tree will essentially be a linked list and of not much use (with respect to performance). The words must be inserted in random order to make a tree that can be efficiently searched. I suggest using the `java.util.Collections.shuffle` method to randomize the words: `java.util.Collections.shuffle(words, new java.util.Random(System.currentTimeMillis()));` Then, you can iterate through the array and add items to your tree, resulting in a reasonably balanced and efficient binary search tree. *If you use the randomization technique, the height and number of leaf nodes in your tree will vary between runs of your program, that **is** expected.* Alternatively, you can do a "binary traversal" of the ordered words and insert the words into the tree in that order. This approach will give you the best possible tree. Either technique you use is acceptable for this assignment.

Tree Testing Example

In addition to the provided unit tests, there is a method in the driver code called `testTree`. This method performs the basic tree operations and reports the results to the console. The purpose of this method is to help you know if your code is coming along correctly (or not). The following is an example run from this method...

```
--- Initial Tree State ---
Arden
Benjamin
Chia
Judie
Olga
Santiago
Tanesha
```

```
Tisa
Tomeka
Ulysses
-- Tree Stats --
Total Nodes : 10
Leaf Nodes  : 5
Tree Height : 3

--- After Adding Duplicate ---
Arden
Benjamin
Chia
Judie
Olga
Santiago
Tanesha
Tisa
Tomeka
Ulysses
-- Tree Stats --
Total Nodes : 10
Leaf Nodes  : 5
Tree Height : 3

--- Removing Existing Values ---
Benjamin
Chia
Judie
Santiago
Tanesha
Tisa
Tomeka
Ulysses
-- Tree Stats --
Total Nodes : 8
Leaf Nodes  : 4
Tree Height : 3

--- Removing A Non-Existent Value ---
Benjamin
Chia
Judie
Santiago
Tanesha
Tisa
Tomeka
Ulysses
-- Tree Stats --
Total Nodes : 8
Leaf Nodes  : 4
Tree Height : 3
```

Include the call to the `testTree` method in your submission so the graders can quickly know if your code correctly executes the provided code.

Reading From a File

For this assignment you need to read the `Dictionary.txt` file, along with the `Letter.txt` file. Reading a file using Java is very simple, you already know most of it! The following code segments demonstrates opening a file and reading strings/lines from it...

```
File file = new File("MyFile.txt");
try (Scanner input = new Scanner(file)) {
    while (input.hasNextLine()) {
        String word = input.nextLine();
        System.out.println(word);
    }
}
catch (java.io.IOException ex) {
    System.out.println("An error occurred trying to read the file: " + ex);
}
```

All you have to do is create a `File` object, then associate it with a `Scanner`, and then everything you already know about `Scanner` applies.

File I/O in Java has checked exceptions, therefore, the exception handling you see is necessary.

Notes & Submission

- **This assignment may not be turned in late!!**
- You may ***not*** use the binary search tree code in the book at the BST code for this project. You can select to use the BST code provided by me in Files/Sample Code within Canvas or write your own from scratch, but you can not use the BST code from your book (or anywhere else).
- Turn in the following Java source files
    - **SpellChecker.java**
    - **BinarySearchTree.java**
- You are provided with a skeleton project to work from that has JUnit setup and includes the unit tests for this assignment. The project can be found at this **link** **(https://usu.instructure.com/courses/639065/files/79619923/download?download_frd=1)** .
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 1410 coding standard: **link**
- Java SDK Docs: **link** **(https://docs.oracle.com/en/java/javase/15/)**

Assignment 9 - Grading Criteria

Criteria	Ratings		Pts
Compiles without any warnings.	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Follows required course code style	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Passes all unit tests	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Randomization of words going into the tree Alternatively, binary traversal	**2 pts Full Marks**	**0 pts No Marks**	2 pts
insert method	**5 pts Full Marks**	**0 pts No Marks**	5 pts
remove method	**5 pts Full Marks**	**0 pts No Marks**	5 pts
search method	**5 pts Full Marks**	**0 pts No Marks**	5 pts
numberNodes method	**3 pts Full Marks**	**0 pts No Marks**	3 pts
numberLeafNodes method	**3 pts Full Marks**	**0 pts No Marks**	3 pts

Criteria	Ratings		Pts
height method	**3 pts Full Marks**	**0 pts No Marks**	3 pts
Removal of punctuation from words in the letter	**3 pts Full Marks**	**0 pts No Marks**	3 pts
Spell checking	**8 pts Full Marks**	**0 pts No Marks**	8 pts
Generics-based implementation There are no points for doing this, but negative points if it isn't a generic-based BST implementation. -10 points if it is hard-coded to a type, like a String, rather than being a generic implementation.	**5 pts Full Marks**	**0 pts No Marks**	5 pts
			Total Points: 48