Homework 6 (due by 11 am on May 5, 2015)

Objectives:

- Implement a binary search tree from scratch and use it in an application
- Try to implement Comparable and Comparator interfaces and use them in your application.

In homework 4, you implemented MyHashTable class that *could* help you search for a word very quickly. The MyHashTable class also has a few other capabilities and one of them is keeping track of the frequency of each word.

That was pretty useful. But, you were not totally satisfied with it. *Now, you want to have a lot more useful information and features such as printing values in any order that you can specify.*

While reading a book, one of the things you would do is using the index at the end of the book where you can find pages where a word or a phrase is used in the book.

Your major goal, this time, is to make your own program that builds an index of words from a file that you are to parse.

Wouldn't you want to see not only how many words are in a file but also how often each word appears and where they are in terms of line numbers of the file?

Also, it would be great to sort them by alphabetically as well as by their frequencies.

You are to write a program that parses words from a text file and produces a very useful index ***stored in a binary search tree that you ought to implement*** for this assignment.

There are a few classes that you need to write for the program.

1. Word class

public class Word implements Comparable<Word>

This class has three private variables:

- String word
- Set<Integer> index
- int frequency

The index variable stores line numbers of each word.

A word is a sequence of letters [a..zA..Z] that does not include digits [0..9] and the underscore character.

Here are examples of non-words: abc123, a12bc, 1234, ab_c, abc_

2. BST class

public class BST<T extends Comparable<T>> implements Iterable<T>, BSTInterface<T>

This class has two variables:

- Comparator<T> comparator
- Node<T> root

There is also the private static nested class Node:

private static class Node<T>

Note that you may store Word objects in your program. However, ***you need to make your BST class very general so that it can handle any type of objects***. **You should NOT have any reference to Word class in your BST class.**

You need to implement a few methods. But, the most important two methods are insert and search.

public void insert(T toInsert)

This method accepts any comparable object toInsert and ***recursively*** inserts the object into the BST based on either natural ordering (provided by the Comparable interface) or the specified comparator, passed into the tree through a constructor.

public T search(T toSearch)

This method ***recursively*** searches the BST. Just like insertion, this search method should be able to compare objects in the BST either using compareTo() or compare().

Hint: For this reason, the BST class has to have comparator as its variable.

Iterator()

There should be ***Iterator implementation***. You need to implement in-order iterator that traverses the elements of the BST in an ordered sequence. But, ***it has to be done iteratively, NOT recursively.*** Keep in mind that the first element that your iterator should output is the leftmost child of the root. How can we do this? ***Do not implement remove() method in your iterator class.***

In this class, you are not allowed to use the Java Collections Framework except for your Iterator implementation where you may need.

3. Index class

public class Index

This class is in charge of building an index tree in three different ways using the following three different methods

public BST<Word> buildIndex(String fileName)

that pares an input text file and build an index tree using a natural alphabetical order.

public BST<Word> buildIndex(String fileName, Comparator<Word> comparator)

that parses in input text file and build an index tree using a specific ordering among words provided by a specific comparator.

public BST<Word> buildIndex(ArrayList<Word> list, Comparator<Word> comparator)

that allows to rebuild an index tree using a different ordering specified by a comparator.

For example, you call the second method using one comparator and realized that you want to organize words with the same frequency to be sorted by alphabetically.

Example usages of these methods are provided in the MainDriver.

In this class, you may use the Java Collections Framework.

4. Comparator classes

public class IgnoreCase implements Comparator<Word>

that sorts words by case insensitive alphabetical order. In other words, if this comparator is passed into buildIndex method, then all of the words need to be converted into lowercase and then added into the BST. **The conversion step should be done in the Index class.**

public class Frequency implements Comparator<Word>

that sorts words according to their frequencies. (A word with highest frequency comes first.) ***Do not get confused! This comparator will NEVER be used to build the BST.***

public class AlphaFreq implements Comparator<Word>

that sorts words according to alphabets first and if there is a tie, then words are sorted by their frequencies.

The example used in the lecture note 9 will be helpful for your implementation of the classes above.

To save your time, starter code files (all of the classes along with the MainDriver class) have been provided. Use them for your implementation and don't forget to write your andrew id and name in all the classes you are to implement!

Deliverables:

- Sheets of paper that have your initial code as well as your comments (Submit this in class that is on the due date.)
  - Remember to focus on your mistakes and be more detailed on what you learned as you write comments on papers!
  - ***To ease your workload, for this homework, you can only focus on BST class for your coding practice on paper.***
  - However, if you are willing to include all of the other classes for your paperwork, ***there will be bonus points. (up to 10 points)***
- Your source code files. (Zip Word.java, BST.java, Index.java, IgnoreCase.java, Frequency.java, and AlphaFreq.java files first with the name as "homework6.zip" and submit the zip file using Autolab by the due.)
- ***Once again, the MainDriver is just a starting point for your testing. Test extensively!***

Grading:

Some scores will be determined by the Autolab grading script. In case you do not pass test case(s), please spend some time to think about edge cases you may have missed and test thoroughly before submitting again.

In addition, the teaching assistants will read your code and award additional points or deduct some points based on code quality. The most important criterion is always correctness. Buggy code is useless (even if you may think the found bug is very minor), and is likely to get a low score. It is important that your code be readable and well-organized. This includes proper use of the module system and clear comments. Points will be deducted for poor design decisions, uncommented and unreadable code. Your comments on your paper should be able to demonstrate your proper understanding of the code. ***We will be stricter in grading since this is your last homework.***

- Working code: 70 points
  - File exists and File compiles : 5 points
  - Word.java : 5 points
  - BST.java : 30 points
  - Index.java : 25 points
  - Comparators : 5 points
- Coding style (refer to the guideline): 15 points
- Your paper code's comments: 15 points

Late submissions will NOT be accepted and, if you have multiple versions, make sure to submit the correct version. Also, make sure to submit source code files (.java extension). Only the version that is submitted before the due will be graded.