Homework 4 Programming

Problem 1 - Implementing a Spell Checker - 35 points

Implement a spell-checker by using a hash table. You will create a class called `SpellChecker` in the file `SpellChecker.java` that implements the interface `SpellCheckerInterface.java`. The object will try to check for spelling errors in an input file, referencing a provided dictionary file. The `SpellChecker` object must accept the filename of the dictionary in its constructor. There is a sample dictionary file in the Codio workspace called `words.txt`.
The constructor of the object must (itself or by calling other methods) parse the dictionary file, storing the words in a `HashSet` instance.

- `public SpellChecker(String filename)` - This is your constructor. It should take in a String of the file name of the dictionary. You may need to throw an exception but this is the parameter it takes in. The constructor isn't explicitly specified in the SpellCheckerInterface, but you must have it exactly like this for your program to be graded.
  The `SpellChecker` object will use this dictionary as a reference when checking for spelling errors in a specified input file. **Convert every word to lowercase and remove all punctuation before inserting it into the `HashSet`**. In addition to the constructor, you must implement two methods to complete the functionality of your `SpellChecker` object.

- `public List<String> getIncorrectWords(String filename)` - This method should return a list of all words in the input file that are incorrectly spelled according to the dictionary file provided to the constructor. The String `filename` contains the name of the file to be spell-checked.
  We define a word as a sequence of characters with whitespace (one or more spaces and/or tabs) on either side. To check for incorrectly spelled words, you must first read the file and process it into words. Note, however, that your program must **convert the file to lowercase and remove all punctuation** in generating its list of words. This means that the line `hey!! it's nice to see you--how are you?` should become `hey, its, nice, to, see, youhow, are, you`. No other input processing is necessary and there are no exceptions to these rules. Punctuation is defined as any character that is not one of the following 1) an upper case letter, or 2) a lower case letter, or 3) a number (0-9). Once you split the line into words by splitting by whitespace, remove all punctuation from every word (and don't forget to lower case each word as well).
  Your output list must contain the incorrect words that remain after processing input data (e.g. `youhow` in the above example).

- `public Set<String> getSuggestions(String word)` - This method should return a set of all potential suggestions for the incorrectly spelled word that is provided as input. If there are no valid suggestions, return an empty set.
  In order to generate a suggestion for a given word, implement the following spell checking techniques, where a character is defined as one of `a, b, c, ..., z`:

- Add one character - add a character at every point in the string (including at the very beginning and end)
- Remove one character - remove one character at a time from each position in the string
- Swap adjacent characters - swap every pair of adjacent characters in the string
  Your method should return all possible suggestions from each one of the techniques above. Note that the use of a `Set` object maintains only unique suggestions (i.e. no duplicates).

You may choose to write a tester file, which might print out each incorrectly spelled word in an input file and its suggestions. Any tester you write will, as usual, not be graded. This is an optional step to encourage you to ensure that your code is indeed functional and correct.
We have provided you a sample test file to spell-check. This file is called `test.txt`.

Problem 2 - K-Best Values - 30 points

Find the k-best (i.e. largest) values in a set of data. Assume you are given a sequence of values, one value at a time. We do not know how many elements there are in this sequence. In fact, there could be infinitely many. Implement the class `KBestCounter<T extends Comparable<? super T>> implements KBest<T>` that keeps track of the k-largest elements seen so far in a sequence of data. The class should have two methods and a constructor:

- `public KBestCounter(int k)` - This is your constructor. It should take in an int k that indicates the amount of largest elements you want to return. The constructor isn't explicitly specified in the KBest interface, but you must have it exactly like this for your program to be graded.
- `public void count(T x)` - process the next element in the set of data. This operation must run in at worst `O(log k)` time.
- `public List<T> kbest()` - return a sorted (smallest to largest) list of the k-largest elements. This must run in at worst `O(k log k)` time. The method must not clobber the state of your class. This means that if you run this method twice in a row, it should return the same values.
  Your `KBestCounter.java` class must implement the provided interface `KBest.java`.
  Use a priority queue to implement this functionality. We suggest using the built-in `java.util.PriorityQueue`.

As always, feel free to implement your own tester file to ensure proper functionality.