CS162 - Assignment 5: Anagram Solver

This assignment is intended to give you experience with recursion and search algorithms. It also will solidify your skills with Java Interfaces, File Reading, and Polymorphism.
Turn this assignment in to TEACH by: 09 September, 2012 23:59

Assignment Overview:

You will be implementing an anagram solver. An anagram is a word in the English language whose letters have been *scrambled*. An example is the anagram "elohl", whose solution is the word "hello". The solver will recursively generate the permutations of a given anagram. A permutation is a reordering of the letters in the word. As it generates permutations, it will run a search algorithm to see if the permutation is a word in the English language.

To see if the word is in the English language, you will need a list of real words. A file containing a sorted list of relevant words will be provided to you. These words will be stored in a class that implements the **Dictionary** abstract class. You will need to write this abstract class. It will need an abstract method called *contains* that accepts a String parameter and returns a boolean. It will also need instance fields to keep track of the average lookup time, which is the average time it takes the *contains* method to look up a particular word in the dictionary. You will also need a getter for the average lookup time instance field. The getter should be called *getAverageLookupTime*. The average time should be stored as a double.

You will create three classes that extend **Dictionary:**

- **LinearDictionary** - Searches linearly for words.
- **BinaryDictionary** - Uses binary search to find words.
- **HashDictionary** - Stores words in a Hashtable for quick retrieval.

Each of the above Dictionary implementations needs to take a String filename into the constructor. Don't catch the FileNotFound exception here. Remember to compute the average lookup time in the contains method for each implementation!

The **AnagramSolver** class is responsible for recursively trying permutations of a word and checking to see if it's in a **Dictionary**. It should take a **Dictionary** object in the constructor. It will use this list when attempting to solve an anagram. Your **AnagramSolver** class will have a *solve* method that accepts a String and returns the solved String (or null if there is no solution). You may find it useful to use a *helper method* to make your recursive solution. In your recursive solution, you must use the **Dictionary**'s *contains* method to see if a generated string is a word or not.

Requirements

1. **Requirement 1:** You must implement the three kinds of **Dictionaries** (Linear, Binary, Hash):
    - Each will have different implementations of the *contains* method. This method will look through your list of words. If it finds a matching word, it will return true, otherwise it will return false.
    - **LinearDictionary** will use an array or ArrayList to store the words. It performs a linear search to find a word.

    - **BinaryDictionary** will use an array or ArrayList to store the words. You can assume the word list we give you is in sorted alphabetical order. You will need to perform Binary search to find the words. **Hint: look at the notes on searching for code for binary search. Your binary search does NOT need to be recursive**.
    - **HashDictionary** will store the words in a Hashtable instead of an array or ArrayList. Use the Hashtable class from Java. A Hashtable stores elements in key-value pairs.
2. **Requirement 2:** You must use recursion to solve the anagram.
    - When the *solve* method is called, you should use recursion to find a solution to the anagram. To do this, you will need to reorder the characters given to you into every possible combination and try each of them. You only need to return the first solution to the anagram (if there are multiple solutions possible) **Hint: Look at the permutations() function in the class notes on Recursion.**
3. **Requirement 3:** You must read the list of words from a file.
    - The name of the file containing words will be provided via command line arguments. You must read the words from this file in each **Dictionary**. **Hint: Look at the lab on File I/O.**
4. **Requirement 4:** Each time you run the anagram solver with an anagram, it must print out the unscrambled word and print out the average lookup time in seconds for the *contains* method.
5. **Requirement 5:** The command line arguments for AnagramSolver are as follows:
   AnagramSolver < *anagram* > < *dictionary file* > < *dictionary type* >
   Where *dictionary type* can be l, b, or h for linear, binary, and hash respectively.

Before you begin, make sure you understand the order you need to make your objects in. Remember, your **AnagramSolver** will need a **Dictionary** and your **Dictionary** will need a file name.

Code

You will not be given any starter code for this assignment.

Files

Files you will need:

- **words.txt** (A list of words in the English language)

Hints

- To measure the time it takes for your lookup, you will need to use the *System.currentTimeMillis* or *System.nanoTime* methods to get the start time. Run the method again when you finish and subtract the start time from the end time to get the elapsed time. **Make sure you print out the elapsed time in seconds**.

What to Turn In: (Remember to turn in the .java files NOT the .class files)

Please hand in all the .java files for your assignment. Turn them in via Blackboard

- All .java files for your assignment. Do NOT hand in the .class files.

Grading Scheme:

You will be graded on the functionality of the anagram solver, the three search algorithms, and the use of correct coding conventions.