Program #4

Due: **11/16/18 at 10:00 PM**
Late Submissions: **11/17/18 at 10:00PM**
11/18/18 at 10:00 PM
Program Files (2): **Jumble.java, Dictionary.java**

In this assignment, you will write a program that can unscramble English words. Your program will start with a command line argument that is the name of an ASCII text dictionary file that contains a list of words, one on each line, sorted first by the number of letters in the word, and then each group of words with the same length is sorted alphabetically. It will ask the user to enter a scrambled word, consider all of the permutations of the word, and return any that are found in the dictionary. It will continue to ask the user to enter new words until the user presses return without typing anything. You are expected to use *recursion* both to discover the permutations of a word and to search the dictionary for a word.

Your first step should be to solve the problem of finding all of the permutations of a string *using recursion*. Write a **Jumble** class, and include a method with the following UML signature:

+findPermutations(s:String): String[]

You'll probably want to create the following recursive helper method too:

-findPermutations(s1:String,s2:String, permutations:ArrayList<String>):void

In this method, s1 is a string that represents an input word, while s2 is the (possibly empty) string of characters representing current progress on building the permutation one character at a time. Whenever the method builds a complete permutation of the original string, the permutation is added to the ArrayList referenced by the method's parameter. By using an ArrayList here, you can avoid having to know the number of permutations in advance (which actually is not hard), but also you do not need to keep track of how many permutations have been generated so far. To receive full credit, you should avoid creating multiple copies of the same permutation.

Once you are confident that you can correctly find all of the permutations of a word, you need to work on reading in the dictionary file and storing it in a way that allows you to easily look up a word. Recall that the dictionary file is sorted by word length, and then alphabetically for each length. As a result, the most convenient storage mechanism is probably to store a number of sorted word lists, where each list only contains words of the same length. You cannot make assumptions about how many words will have any given length. Create a class called **Dictionary** to store the words. See the next page for more information on this class.

Your **main** method should be in your **Jumble** class. This method should examine the command line arguments. If there are two arguments, then the first should be an integer indicating the length of the longest words in the dictionary file, and the second argument should be the path to the dictionary file. If there is a single argument, then this is simply the path to the dictionary file, and the default value of 10 should be used for the maximum word length. Your program will open the file and read the words, where there is one word on each line. As it reads each word, it should add it to a **Dictionary** object. After the file has been completely read, you should print a count of how many words are in the dictionary. At this point your program should enter a loop, where the user can enter scrambled words, and your program will print out all of the permutations that match words in the dictionary. Note, some scrambled words may have many permutations that are dictionary words, and you should print all of these. If the user presses **Enter** without entering a string, then the program should terminate. A sample interaction of the program using the 704KB **us10.dic** file provided with the assignment is given on the next page.

At a minimum, the **Dictionary** class will need the following methods (with signatures described using UML notation):

+Dictionary(maxWordLength: int)	A one-argument constructor that takes an argument representing the maximum length for words that will be stored in the dictionary.
+Dictionary()	A no-argument constructor that assumes that the dictionary will not store words longer than 10 letters.
+addEntry(word:String):void	Add a word to the dictionary. This method adds the word to the word list with the correct length. You can assume that the method will only be invoked such that words of the same length are added in alphabetical order.
+lookup(s:String):boolean	Returns true if the string is found in the dictionary. This method **must** use a *recursive binary search* on the appropriate word list to find out whether the string is in the dictionary or not. Hint: this will actually need a recursive helper method.

You should always use generic instantiation with your ArrayLists (i.e., no raw types). When your program is compiled, the only warnings that should be produced are those that cannot otherwise be eliminated. **Hint:** In at least one situation, you will need to handle a consequence of type erasure.

Note, your program should be robust against error conditions. It should terminate gracefully when the user provides an incorrect number of command-line arguments, provides arguments of an incorrect type or specifies a file that cannot be opened. If the dictionary file contains words that are longer than the specified maximum, then it should simply skip those words. If the user provides a scrambled word that is longer than the maximum, it should inform the user and ask for a new input.

Comments:
At a minimum provide a comment that summarizes each class, and comments that explain what each method does. Include additional comments for sections of code that are especially complicated. At the top of each file, include a comment with the following form:

```
/*
CSE 17
Your name
Your user id
Tutor: name (e-mail)   [only include if a tutor aided you]
Program #4      DEADLINE: November 16, 2018
Program Description: Word Jumble Solver
*/
```

Submission:
Make sure that you have named your classes and files as specified, and that your main method is in the **Jumble** class. Once the program compiles, runs, and has been tested to your satisfaction, upload both of the **.java** files to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the “Add submission” button at the bottom of the next page. Drag and drop each file into the area under “File submissions”. If necessary, you can update your submission at any time before the deadline passes.

Sample Output:
Users inputs are shown in bold.

```
> java Jumble us10.dic
Read in 82556 words

Enter a scrambled word: ginor
The words formed from 'ginor' are:
groin

Enter a scrambled word: xyz
No words are formed from 'xyz'

Enter a scrambled word: droino
The words formed from 'droino' are:
indoor

Enter a scrambled word: tra
The words formed from 'tra' are:
tar
rat
art

Enter a scrambled word:

Goodbye!
>
```