Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 2:
Find Words

Due date: Oct. 10, 11:55 PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures or features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will work on a program that produces all possible words given a set of letters and a dictionary. Your program should produce all anagrams of a given set of letters that are valid words in the provided dictionary. For example, if the letters are

```
recounts
```

your program should print, in alphabetical order and one per line, all different words that can be made out of those letters (assuming that they are present in the dictionary that the program uses):

```
construe
counters
recounts
```

The program that you write has to be command line based (no graphical interface). It should use the dictionary from a file provided as a command line argument and it should prompt the user for a set of letters. The output should be printed to the terminal. Once the anagrams are displayed, the program should terminate.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- using recursion to solve problems
- reading data from input files
- using command line arguments
- writing Java programs

Start early! This program may not require you to write a lot of code, but debugging may be tricky.

The Program Input and Output

You may use any Java classes for reading from input files.

Input File

Your program is given the name of the input text file as its command line argument (the first and only argument used by this program). The text file contains a dictionary that is used by the program. You may assume that the dictionary contains a sorted list of words, one per line. A word is any sequence of lower case letters.

Joanna Klukowska
joannakl@cs.nyu.edu

If the filename is omitted from the command line, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: missing name of the input file").

If the filename is given but the file does not exist or cannot be opened for reading by the program, for any reason, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: file `dictionaryEnglish.txt` does not exist.", but make sure to replace the name with the name of the file with which the program was called).

Your program is NOT ALLOWED to hardcode the input filename in its own code. It is up to the user of the program to specify the name of the input file. Your program should not modify the name of the user-specified file (do not append anything to the name).

Your program may not read the input file more than once.

Your program may not modify the input file.

Output File

Your program does not produce any output files.

User Input

The user should be prompted to enter a string of characters (letters only, no spaces, commas, or any other characters). Your program should accept both upper case and lower case letters. If the user enters any uppercase letters, your program should convert them to lowercase before proceeding.

If the user enters any characters other than letters, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: you entered an invalid character; only letters can be accepted").

Console Output

The program should display in lower case letters all anagrams of the user typed letters that are valid words based on the dictionary that the program uses. The program should display the total number of unique words found (for example: `"Found 3 words"`), followed by the list of words. The words should be displayed one per line and in alphabetical order. If there are any repeated words in the list of generated anagrams, they should not be shown only once in your output.

If there are no valid words that can be created from the user provided letters, the program should display a message: `"No words found"`.

Computational Task

Once the user has entered the letters, the program displays all the words in the dictionary that can be formed as combinations of all the letters entered by the user.

Creating all Possible Words

The task of creating possible words should be achieved recursively using **the backtracking technique**. Make sure to review the examples used in class for creating all possible strings from the given set of characters.

If the user enters n letters there are $n!$ different possible words (note that some of them might repeat, if some of the letters repeat) - but not many of them are going to be found in the dictionary. You should design an algorithm that tries different combinations of letters and checks if they are in the dictionary, but it should do it in a smart way: do not try words that cannot possibly be in the dictionary, i.e., do not follow the paths that do not lead to words. For example, if the user enters `zzasw` and your algorithm starts generating a sequence of characters starting with `zz`, but there are no words in your dictionary that start with `zz`, then there is no point in generating all of the sequences that start with `zz` - your algorithm should skip them.

Joanna Klukowska
joannakl@cs.nyu.edu

Searching in the Dictionary

In order to determine if a given sequence is a valid word in a dictionary, your program needs to perform searches. You need to implement **your own search method** to achieve this. You should use a recursive implementation of a binary search.

The program will also need to be able to determine if there are any words in the dictionary that begin with a particular sequence of characters. The implementation of such method will be similar to the binary search implementation. It should also be recursive.

Program Design

Your program must contain three classes:

- **`Dictionary`** class to represents the collection of words read in from the input file (i.e., the dictionary used by the program). This class is responsible for performing queries in the dictionary. The **`Dictionary`** class should implement the following interface:

```
public interface DictionaryInterface {
  /**
    * This method determines if a given word
    * is in this Dictionary.
    * @param word the word to be checked
    * @return true of the word is in this Dictionary,
    * false otherwise
    */
  boolean findWord ( String word );

  /**
    * This method determines if a given
    * prefix is a prefix of a word that exists in
    * this Dictionary.
    * @param prefix the prefix to be checked
    * @return true if the prefix is in this Dictionary,
    * false otherwise
    */
  boolean findPrefix ( String prefix );
}
```

- **`LetterBag`** class to represent the letters entered by the user/player. This class is responsible for creation of all the different words. It should use the **`Dictionary`** object to accomplish its task. The **`LetterBag`** class should implement the following interface:

```
public interface LetterBagInterface {

    /**
     * This method determines the list of words that can be created
     * from a given LetterBag object that are present in the
     * provided Dictionary object dict.
     * @param dict the Dictionary object to be used
     * @return a list of valid words in alphabetical order
     */
    ArrayList<String> getAllWords ( Dictionary dict);
}
```

- **`FindWords`** class that is the runnable program containing the **`main()`** method. This class is responsible for parsing the command line argument, reading the input file, reading the user input, creating the **`Dictionary`** and **`LetterBag`** objects and then using them to display the results. This class may have methods other than the **`main()`** method.

You will need to have files specifying the above interfaces. You may use additional classes, if you wish.

Joanna Klukowska
joannakl@cs.nyu.edu

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at http://cs.nyu.edu/~joannakl/cs102_f15/notes/CodeConventions.pdf.

You must document all your code using Javadoc. Your class documentation needs to provide a description of what it is used for and the name of its author. Your methods need to have description, specification of parameters, return values, exceptions thrown and any assumptions that they are making.

A class's data fields and methods should not be declared `static` unless they are to be shared by all instances of the class or provide access to such data.

You must use backtracking and recursion to produce the list of words.

You may use `ArrayList` and `String` classes, but you must implement your own search and sort methods.

You may use any classes for reading the input from the file.

You may use any exception related classes (if you wish).

Working on This Assignment

You should start right away! This program does not require you to write much code, but debugging recursive solutions may take some time.

You should modularize your design so that you can test it regularly:

- Start with a program that reads in the input file and the input from the user. Make sure that this works.
- Write a class to represent and store the dictionary. Develop methods that allow you to search in the dictionary (HINT: you will need two search methods: one that looks for words, the other that checks prefixes). Make sure that this works.
- Write a class that represents the letters entered by the user with methods that, given a dictionary object, can produce a list of words that consist of all the letters. Make sure that this works.
- Write the `main` method that combines the above pieces and produces the output file.
- Use different dictionaries, different letters. Try to break your program (and then fix the problems that you found).
- Re-read this project specification and make sure that your program adheres to it.
- Submit your code.

You should backup your code after each time you spend some time working on it. Backup by either saving it to a flash drive, emailing it to yourself, uploading it to your Google drive, or anything else that gives you a second (or maybe a third copy). Computers tend to fail just a few days or even a few hours before the due date - make sure that you have working code in case that happens.

Grading

Make sure that you are following all the rules in the **Programming Rules** section above.

If your program does not compile or crashes (almost) every time it is ran, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

25 points design, implementation and correctness of the `Dictionary` class,

30 points design, implementation and correctness of the `LetterBag` class,

Joanna Klukowska
joannakl@cs.nyu.edu

25 points design, implementation and correctness of the `FindWords` class,

20 points proper documentation and program style.

How and What to Submit

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes. If you are unsure how to create a zip file or how to get to your source code files, you should ask long before the project is due.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at `http://goo.gl/forms/YFDVBlscEB` **before the due date for the assignment**. All freebies are due seven days after the original due date and should be submitted to NYU Classes.