Joanna Klukowska
joannakl@cs.nyu.edu

Project 2: Scrabble Helper

Due date: March 2, 11:55PM 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 (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). 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 and 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 collection 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 printed words should not contain any repeats.

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 and backtracking to solve a problem
- using the `ArrayList` class
- reading data from input files
- using and understanding 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

The program should be designed to expect two command-line arguments: the first one is the name/path of the file containing the dictionary, the second one is the sequence of letters to be used in anagram creation.

Input File

The program is given the name of the input text file as its first command line argument. 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. 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 error message should indicate what went wrong (for example:

```
Error: missing name of the input file.
```

or

```
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).

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

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

The program may not modify the input file.

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

Letter Sequence

The program is given a string of characters (letters only, no spaces, commas, or any other characters - this should be validated) as a second command-line argument. The program should accept both upper case and lower case letters. If the user enters any uppercase letters, the program should convert them to lowercase before proceeding.

If the user enters any characters other than letters, it is an error. If the user does not provide any 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.
```

or

```
Error: no letters provided, cannot compute any words.
```

Output File

The program does not produce any output files.

User Input

This program is not interactive and the user should not be prompted for any input.

Console Output

The program should display in lower case letters all anagrams of the user provided 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. Each word should appear only once.

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

Here are a few sample runs of the program:

```
>> java -cp bin project2.ScrabbleHelper dictionary.txt aeicdlm
Found 5 words:
   camelid
   claimed
   decimal
   declaim
   medical
```

```
>>  java -cp bin project2.ScrabbleHelper dictionary.txt drroptea
Found 5 words:
  parroted
  predator
  prorated
  protrade
  teardrop
```

```
>> java -cp bin project2.ScrabbleHelper dictionary.txt zziat
No words found
```

Computational Task

The program should displays all the words in the dictionary that can be formed as combinations of all the letters provided by the user.

There are efficient and inefficient ways of going about it. If you solution is not an efficient one, you will lose points for efficiency (see the rubric below), but you will not lose points for correctness.

Creating all Possible Permutations of Characters

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.

The program needs to provide a method that calculates all of these permutations - see the class spec below. (Hopefully this method will not be used in the final program, but it is handy to have it as a starting point or as a backup.)

Creating all Possible Words

The ultimate goal is to determine the list of the permutations from the previous section that are the actual words.

Inefficient approach: Generate all permutations and then iterate through them to check which of them are in the dictionary.

Efficient approach: Write a method similar to the one that generates all permutations, but stops generating a given string as soon as it determines that it is not a prefix for any word in the dictionary, i.e., the efficient approach should not follow the paths that do not lead to words. For example, if the sequence of letters is `zzasw` and the algorithm starts generating a sequence of characters starting with `zz`, but there are no words in the dictionary that start with `zz`, then there is no point in generating all of the sequences that start with `zz` - the algorithm should skip them.

Searching in the Dictionary

Searching for words: In order to determine if a given sequence is a valid word in a dictionary, the program needs to perform searches in the data structure that stores the dictionary words. You need to implement **your own search method** to achieve this (do not use any Java API methods that provide this functionality). You should use **a recursive implementation of a binary search**.

Searching for prefixes: (This is part of the efficient implementation discussed above.) The program also needs 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 should be similar to the binary search implementation. It should also be recursive.
HINT: you may want to read about `startswith()` method in the `String` class.

Joanna Klukowska
joannakl@cs.nyu.edu

Program Design

Your program must contain three classes described below. You may use additional classes, if you wish.

`Dictionary` class

The `Dictionary` class 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 store all words in an `ArrayList<String>` object (this is the has-a relationship, not the is-a relationship).

It should provide the following methods:

- **`public Dictionary( File f )throws IllegalArgumentException`**
  A one parameter constructor that takes a `File` object as a parameter. The dictionary should be populated with the words stored in the file. If anything goes wrong (the file does not exist, it is not readable) the constructor should throw an instance of `IllegalArgumentException` with an appropriate message.
- **`public boolean isWord( String str )`**
  A method that determines if the argument string `str` is one of the words stored in this dictionary. This method should use binary search.
- **`public boolean isPrefix( String str )`**
  A method that determines if the argument string `str` is a prefix for at least one of the words stored in this dictionary. This method should use binary search like approach.
  If the program does not implement the efficient version, this method is not needed. In such a case, it should be provided, but it should throw (and declare) an instance of `UnsupportedOperationException`.

You may need additional private methods that provide the actual recursive implementation of algorithms that search for words and prefixes.

`Permutations` class

`Permutations` class represents the sequence of letters from which the words should be constructed. This class constructs all the words and permutations. It should use the `Dictionary` object to accomplish its tasks.

The `Permutations` class should provide the following methods:

- **`Permutations(String letters)throws IllegalArgumentException`**
  A one parameter constructor that takes a String object containing the sequence of characters to be used. The constructor should throw an instance of `IllegalArgumentException` with an appropriate message if the `letters` argument contains illegal characters. The only legal characters are letters `a-z` and `A-Z`.
- **`public ArrayList<String> getAllPermutations()`**
  A method that computes and returns a list of ALL PERMUTATIONS of letters in this permutation object. (Warning: this method may take a lot of time for large number of letters in the object. This method may throw an instance of `OutOfMemoryError` for objects that contain more that 10 letters.)
- **`public ArrayList<String> getAllWords(Dictionary dictionary )`**
  A method that computes and returns a list of ALL WORDS contained in the `dictionary` object that can be created from the letters in this permutation object. (Warning: this method may take a lot of time for large number of letters in the object if it is not implemented efficiently.)

`ScrabbleHelper` class

`ScrabbleHelper` class is the runnable program containing the `main()` method. This class is responsible for parsing the command line arguments, creating the `Dictionary` and `Permutations` objects and then using them to display the results. This class may have methods other than the `main()` method.

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_s17/notes/CodeConventions.pdf`.

The data file should be read only once! Your program needs to store the data in memory resident data structures.

You may not use any of the collection classes that were not covered in cs101 (for this assignment, do not use `LinkedList`, `Stack`, `Queue`. `TreeSet`, `PriorityQueue`. or any classes implementing `Map` interface).

You may use any exception-related classes.

You may use any classes to handle the file I/O, but probably the simplest ones are `File` and `Scanner` classes.

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. Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should start with the inefficient implementation - this should be simpler to write and gives you a working program. Once that works correctly, you can concentrate on re-writing the relevant parts of the `Permutations` class to work more efficiently.

You should make sure that you are testing the program on much smaller data set for which you can determine the correct output manually. You should create your own small test files for that purpose. (Feel free to share those with other students on Piazza.)

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens. (A second copy of the files on the same computer is a good idea to keep multiple versions, but it is NOT a good backup since you do not have access to it if there are problems with your computer.)

Grading

If your program does not compile or if it crashes (almost) every time it is run, 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.

20 points efficient design of the algorithms

40 points program and class correctness: the correct values and format of output, correct behavior of methods, handling of invalid arguments

20 points design and implementation of the three required classes and any additional classes

20 points proper documentation, program style and format of submission

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.

You can produce a zip file directly from Eclipse:

Joanna Klukowska
joannakl@cs.nyu.edu

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select "Save in zip format", "Compress the contents of the file" and "Create only selected directories"
- click Finish

Do not submit any data files in your zip file!