ASSIGNMENT 6

COMP-202, Winter 2015, All Sections

Due: Tuesday, April 14, 2015 (23:59)

Please read the entire pdf before starting.

You must do this assignment individually and, unless otherwise specified, you must follow all the general instructions and regulations for assignments. Graders have the discretion to deduct up to 10% of the value of this assignment for deviations from the general instructions and regulations. These regulations are posted on the course website. Be sure to read them before starting.

Question 1:	65 points
Question 2:	35 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment through automated tests. While these tests will not determine your entire grade, it will speed up the process significantly, which will allow the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. Marks can be deducted if comments are missing, if the code is not well structured, or if your solution does not respect the assignment requirements.

Assignment

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Write a program that *opens* a **.txt**, *reads* the contents of the file line by line, and *prints* the content of each line. To do this, you should use look up how to use the `BufferedReader` or `FileReader` class[1]. Remember to use the `try` and `catch` statements to handle errors like trying to open an non-existent file.

Warm-up Question 2 (0 points)

Modify the previous program so that it stores every line in an `ArrayList` of `String` objects. You have to properly declare an `ArrayList` to store the results, and use `add` to store every line that your program reads in the `ArrayList`.

[1]The documentation of the `BufferedReadder` class is available at `http://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html`. You can find an example on how to use it at `http://www.tutorialspoint.com/java/io/bufferedreader_readline.htm`

Warm-up Question 3 (0 points)

Finally, modify your program so that, after reading all the content in the file, it prints how many words are inside the text file. To do this, you should use the `split` method of the `String` class. Assume the only character that separates words is whitespace `" "`.

Part 2

The questions in this part of the assignment will be graded. This assignment asks you to implement methods that perform some specific function. To obtain full marks, you must adhere to the specified requirements for the method names and input parameters. You may in addition implement helper methods to perform some subtask as part of solving the questions below.

Question 1: Language Identification (65 points)

The two most commonly spoken languages in Montreal (and Canada overall) are French and English. Living in Montreal means that we must navigate an environment in which we have to constantly determine what language a road sign, an advertisement, or a menu is written in. In this assignment, you will write a program that will be able to automatically identify whether a given document is written in English or French.

We will *train* the program to do so by feeding it a sample of texts written in French, and another sample written in English. The program will *learn* from these two sets of texts what the typical vocabulary of French and English are. Then, when given a new document that is written in one of the two languages, it will compare the words in that document against the vocabularies of English and French that it has built up to identify the language that the new document is written in.

To begin, download the starter code and the data set we will be using from the course website. The data set consists of three sets of 20 random articles extracted from Wikipedia. The directory structure of the data, once unzipped, is as follows:

docs/

- docs/train
  - docs/train/eng (contains 20 training articles in English)
  - docs/train/fre (contains 20 training articles in French)
- docs/test (contains 20 test articles in both languages)

How the program will work is that you will first give it the two sets of articles in docs/train, which it will know is in English or French respectively. The program will learn a partial vocabulary of English and French from these articles. Then, you will present it with a new set of articles, the ones in docs/test, for which the program does not know the language it is in. The program must determine based on the words in these new articles which language the article is in.

You will separate your code into two classes: `LanguageLearner.java`, and `LanguagePredictor.java`. Your job is to implement the following methods in the specified classes. Feel free to define helper methods to make your code easier to understand.

1.1: `countWords` In `LanguageLearner.java`, implement the method `countWords`. This method takes two arguments, a `String` that points to a directory, and an `int` that refers to how many files in the directory to read. This method should read documents from the indicated directory, starting with the file "1.txt" and proceeding up to the indicated number of files (e.g., "20.txt" if the value of the `int` passed into the method is 20). This method counts the frequencies of the words in the files, and returns the word counts in a `HashMap`. You may use the `split()` method of `String` to get the words in the documents. You must strip any whitespace from the words, and ensure that the empty string "" is not

a vocabulary item in the returned `HashMap`, but you do not need to otherwise process the words, for example by removing punctuation or changing the case of the first letter.

For example, if the method read a single document containing this sentence:

"Cats are the most popular pet in the world, and are now found in almost every place where humans live."

then the `HashMap` should contain the following key:value pairs:

```
Cats:1      are:2       the:2
most:1      popular:1   pet:1
in:2        world,:1    and:1
now:1       found:1     almost:1
every:1     place:1     where:1
humans:1    live.:1
```

To implement this method, think about how to update the `HashMap` when you encounter a word. Either you have never seen this word before, in which case it would not appear as a key in your `HashMap`, or you have seen it before, in which case you already have an entry in the `HashMap` corresponding to this word. What do you need to do to correctly update the `HashMap` in each case?

1.2: `writeVocabulary` In `LanguageLearner.java`, implement the method `writeVocabulary`. This method takes a `HashMap` of word frequencies (as produced by `countWords`), and writes the `HashMap` to the indicated file name as a text file. The format of the output should be such that each word is written on its own line followed by its frequency, separated by a space. The words should be sorted by the default order of `String`s (i.e., by the ASCII value of the characters). You may use existing Java methods to help you sort the words. In particular, you may find the `Collections.sort()` method useful, though you will have to figure out how to turn the keys of the `HashMap` into a `List`.

For example, the first several lines of the `HashMap` from above written to a text file would be:

```
Cats 1
almost 1
and 1
are 2
every 1
...
```

1.3: `readVocabulary` In `LanguagePredictor.java`, implement the method `readVocabulary`, which reads a vocabulary file in the format stored by `writeVocabulary` and returns the `HashMap` of word counts associated with it.

1.4: `classifyDocuments` In `LanguagePredictor.java`, write the method `classifyDocuments`, which takes in four arguments: a `HashMap` of the English vocabulary, a `HashMap` of the French vocabulary, a `String` pointing to a directory, and an `int` that indicates the number of files to read. This method reads the documents that are indicated, which works in the same way as in 1.1, and decides for each document whether it is in English or French, printing out the result.

Specifically, the decision should be made based on the words in the document and the English and French vocabularies that the program has previously learned. For each word in the document, this method checks if that word is a known English or French word, and keeps track of the total number of known English and French words found. After processing the document, the method decides what language the article is in based on whether there are more English or French words in it[2]. You can break ties however you like. Note that a word may be found in one, both, or neither of the vocabularies.

[2]We won't use the word frequencies that you computed in 1.2 in this method.

The method should print out the number of English and French words found in addition to the final decision for each document in the following format:

*docid (tab)*English: *engwords (tab)*French: *frewords (tab)*Decision: *English/French*

For example, this is one possible output (with fictitious word counts):

1 English: 33 French: 216 Decision: French

1.5: `main` methods Write two main methods to use the methods you wrote above. The `main` method of `LanguageLearner.java` should use the code you wrote in 1.1 and 1.2 to create vocabulary files for English and French called "eng_vocab.txt" and "fre_vocab.txt" respectively. **Submit these vocabulary files along with your code.** You may hardcode any directory path names and other file names. The main method of `LanguagePredictor.java` should read the vocabulary files you produced from before, and using the methods you wrote in 1.3 and 1.4, classify the test documents into English and French. You may find it helpful to write these main methods as you are writing the methods 1.1–1.4 to test and debug them, rather than writing the main methods afterward.

Test articles 1–10 are in English, whereas 11–20 are in French. Does your program correctly classify them as such? **Copy the output of `classifyDocuments` and briefly report the result in one sentence in the indicated comment section of `LanguagePredictor.java`.** If you'd like, you can find additional texts, from Wikipedia or elsewhere, in English and French to test your program on. If you do so, you can submit them as additional test cases numbered "21.txt", "22.txt", etc. to show your TA, though this is just for fun and you won't get additional marks for it.

Your code should handle the common failure cases for reading files that we discussed in class using try/catch blocks and exceptions. Marks will be deducted for not properly handling these cases (e.g., if you just catch a generic Exception, rather than handle the FileNotFoundException separately from other Exceptions).

Question 2: Language Family Tree (35 points)

Languages change and evolve over time. If there are multiple groups of speakers of a language, their speech may change and become separate dialects, such as Canadian English vs. British English, or Quebec French vs. Parisian French. Eventually, the dialects may diverge so much that they become separate languages.

Languages can thus be organized into language families to indicate how they are related to each other historically. For example, English belongs to a language family called West Germanic languages, which also includes Dutch, and German. The West Germanic languages are themselves part of a larger group of languages called the Germanic languages, which also include Norwegian, Swedish, and Icelandic, among others. In turn, the Germanic languages are part of the Indo-European languages, the most widely spoken language family in the world. French is also an Indo-European language, but it belongs to another branch of the tree, called the Italic languages. Other large language families of the world include the Sino-Tibetan languages (including various kinds of Chinese), the Niger-Congo languages (including Swahili), the Afroasiatic languages (including Arabic), and the Austronesian languages (including Malay).

This question asks you to implement a data structure that can model language families, and a method to print out a language family. As you implement parts 2.1 and 2.2 below, test the code you have written so far in the main method, by creating a sample language family. See part 2.3 for how to do this. In other words, you should implement 2.3 at the same time as 2.1 and 2.2.

2.1 Data Structure Implement a language family tree as a data structure called `LanguageFamilyNode`. This data structure represents either a group of languages (e.g., Germanic languages), or an individual language (e.g., German). It should contain two private fields: `name`, a String to identify the language family or language, and `children`, an `ArrayList<LanguageFamilyNode>` that represents the languages or language families that descended from this language family. Implement a constructor for this class, in addition to any getters and setters that you think are necessary.

Note that an instance of the LanguageFamilyNode class could have references to other LanguageFamilyNode objects! This is an example of a recursive data structure. You have seen something similar in Assignment 5.

2.2 `printAllDescendants` Write a method called `printAllDescendants` that prints out all the descendants of the tree. It should print out the name of the current language family or language and all of the language families that are descendants of the current language family, all the way down to the individual languages. For each level down the tree, the method must indent the name of the language/language family by two additional spaces.

For example, this might be the output printing out the Germanic languages:

```
Germanic languages
  West Germanic languages
    German
    Low Franconian languages
      Dutch
      Afrikaans
    English
  North Germanic languages
    Norwegian
    Icelandic
    Swedish
  East Germanic languages
    Gothic
```

There are two requirements for the implementation of this method. First, it must be overloaded. One version of this method takes no arguments, and does the printing as described above. The second version takes one argument, a `String` that is a prefix to be printed in front of every line before the name of a language family or language. You need to use this prefix in order to make sure the nodes in the tree are properly indented. The second requirement is that this second version of the method must be recursive; that is, it must somehow call itself.

Think about how you can use the prefix argument in the second version of `printAllDescendants` along with recursion to get the proper indentation that is required. At the top level, the prefix should simply be the empty string "". This corresponds to the line "Germanic languages" in the above example. Then, for every level that you go down in the tree, the prefix will have two additional spaces. So, for the line with "West Germanic languages", the value of prefix should be " ", with two spaces. How do you make the recursive call to `printAllDescendants` so that the prefix argument takes the correct value?

2.3 `main` **method** To test your code, go to Wikipedia and find a language family tree of your choice of a reasonable size (at least 20 nodes in the tree). You can try going to the page for language families in general, or go to the page for a specific language family or to a specific language to find a language family tree. You should make an effort to create a tree as described by linguists, but you won't be marked on the linguistic accuracy or completeness of the tree. Build this tree in the main method of `LanguageFamilyNode.java`, and use the first version of your `printAllDescendants` method to print out the tree.

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 6. If you do not know how to zip files, please enquire that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

`LanguageLearner.java` - Java code for Q1
`LanguagePredictor.java` - Java code for Q1
`eng_vocab.txt` - Text file for Q1
`fre_vocab.txt` - Text file for Q1
`LanguageFamilyNode.java` - Java code for Q2

`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.

Marking Scheme

Up to 30% can be removed for bad indentation of your code as well as omitting comments, coding structure, or missing files. Marks will be removed as well if the class names are not respected.

Question 1

1.1 `countWords`	15 points	
1.2 `writeVocabulary`	15 points	
1.3 `readVocabulary`	10 points	
1.4 `classifyDocuments`	15 points	
1.5 `main` methods	10 points	
	65	**points**

Question 2

2.1 Data Structure	15 points	
2.2 `printAllDescendants`	15 points	
2.3 `main` method	5 points	
	35	**points**