**Word Cloud**

You've no doubt seen "word clouds" before, like the one shown above / right:

Before building these images, the program needs to know how much "weight" to give each word in the cloud. This is generally done by counting how many times each word occurs in the input text.

*There are actually quite a few interesting applications of word clouds. For example, to find the mood of the country throughout US history, check out the word clouds of presidential inaugural addresses* [*here*](http://www.nytimes.com/interactive/2009/01/17/washington/20090117_ADDRESSES.html)*.*

Your job is to create a program that will scan input text (a file) and find the most frequently occurring words.

To begin, make a class called Word that describes an individual word in the input text. The Word class is a convenient bundle of a couple variables and methods, and is relatively simple. Watching [this](http://www.showme.com/sh/?h=D3ReVkG) video before starting may help.

Fields (instance variables)

* + private String word-the text (word) itself
  + private int count - how many times this word has occurred in the text

Methods

* + public Word(String w) – constructor, initializes instance variables
  + getWord() and getCount() getter methods
  + public void increment() – adds 1 to a Word object's count field
  + public String toString() – this method allows a Word object to be printed in a useful way. Copy / paste the code below directly into your Word class:

public String toString()

{

return word + "\t\t" + count;

}

Make a class called WordCloud that contains all the application logic and supporting methods to generate the Word Cloud's "statistics", given a particular text file. Help with the algorithm can be found at the end.

**Fields (instance variables)**

* + private ArrayList<Word> words – list of all the (unique) words in the input file
  + private ArrayList<Word> topHits – list of the top 30 most frequently occurring words in the input file, generated *after* the words list has been populated
  + private int totalWords – total number of words in this WordCloud
  + private int uniqueWords – total number of unique words in this WordCloud

**Methods**

* + public WordCloud(String fileName) – constructor, should initialize the instance variables, then call the load() method (read on for more info)

/\* The line **throws IOException** must be added after **EVERY** method header that reads from a file – the constructor AND the load() method \*/

* + private int getIndex(String str) – helper method, returns the index of the first Word object whose word matches the parameter (for incrementing its count), or -1 if it can't be found in the words list. Used to locate a Word in the list.

**Note:** you can't use the ArrayList class' indexOf() method, for reasons you will learn later this year (in the inheritance unit), to locate Word objects in the list. You'll have to make your own.

* + private void load(String fileName) – should load all the words in the text file (given the fileName parameter) into the words list.
    - Words that already exist in the list should have their count incremented, while words that don't exist in the list should be added as new Word objects (use getIndex())
    - Use Scanner's next() method to tokenize the input.
    - **There are some special cases you'll need to consider:**
* ***The*** and ***the*** both count as occurrences of the. In other words, you may want to convert each word to lower case before adding / counting.
* ***Hyphenated*** words (e.g. sixty-three) count as one word

//in other words, "interior" punctuation will not be removed

* ***Hyphenated words separated by a space*** (e.g. joyous – sparkling) count as two words; the hyphen (-) will have a blank space on each side

//words that contain *only* punctuation should not be added to the list

* ***Shortened words*** (e.g. 'tis) count as one word, and the leading / trailing punctuation should be removed before adding (e.g. 'tis should be added as tis)
* ***Trailing punctuation*** should be removed (e.g. hello? is added as hello)
* The Character.isLetterOrDigit(char c) method will help with determining the indexes of the first and last non-punctuation characters in the String
  + public ArrayList<Word> getTopHits() – populates and returns an ArrayList of the 30 most frequently occurring Words. Should be invoked (called) by the load() method, after the words list has been populated.
  + public void printInfo() – prints the "stats" and the 30 most frequently occurring words in this WordCloud.

**Help with WordCloud's algorithm and an example of the output can be found on the next page.**

WordCloud algorithm:

Number of unique words >>> 526

Total # of words >>> 1578

Word Frequency

1) the 103

2) of 97

3) to 59

4) and 43

5) a 36

6) we 32

7) be 32

8) will 26

9) that 24

10) is 21

11) have 19

12) freedom 19

13) in 18

14) this 18

15) from 18

16) as 15

17) negro 14

18) not 14

19) our 14

20) i 14

21) let 14

22) one 12

23) ring 12

24) with 11

25) dream 11

26) every 10

27) day 10

28) must 9

29) come 9

30) nation 9

*Add a loop that reads the file*

*Read a word from the file*

*Make sure to filter out punctuation, if necessary!*

*If the word has been seen before, increment its* count

*If it's a new word, create a new Word object and add it to the list*

*Count the number of unique words*

*Count the total number of words*

*For the topHits ArrayList…*

*Add a loop that runs 30 times (NO sorting API methods)*

*Find the most frequently occurring word*

*Add it to the topHits list, remove from original list*

*Print a report similar to the one shown on the right*

The text for Dr. King's "I Have a Dream" speech is included in a text file in the lab folder**.** In a Runner class with a main() method, make a WordCloud object and test your program against the graphic shown previously to verify that your program is working correctly. Do not worry if your program outputs slightly different results than shown above (e.g. your total words is off by a couple, or one of the top 30 words has a slightly different frequency) - different techniques for handling punctuation can cause minor variations in the numbers, and is totally fine.

You could also test your program with other documents. For example, run your program with one of your English papers (saved as a .txt) to gauge the "tone" of your paper. Filtering out common words (e.g. "the") might be helpful.

**(Advanced) Graphical word cloud**

Create graphical word cloud (using a WordCloud object). Creating graphics in Java is beyond the scope of this class, but it is not hard to Google some graphics starter code (traditionally done with a JFrame and JPanel object, though these classes are now deprecated in lieu of the JavaFX library).

If you get stuck, you can see a nice GUI class contributed by Liberty student David Poe [here](https://www.dropbox.com/s/pt288full4kz12p/WordCloudGUI.java?dl=0).

