# APCS-A Java Classwork – Word List Examiner

You are to write code for a class named "WordListExaminer" that reads all of the words from a disk text file, converts them to lowercase, uses those words to create an ArrayList of 'WordCount' objects (in ascending alphabetical order by word), and then displays, in a neat table, every unique word in the list, along with each word's frequency (the number of times the word appears in the disk text file).

The name of the disk text file read by your program should be "listofwords.txt", and it may contain any number of words, one per line, in any order. When I grade your assignment, I will run your program, but will use my own list of words. Your program should assume that the disk text file is in the same location as the "WordListExaminer.java" and "WordCount.java" files.

Your class should store all of the unique words and their frequencies in a single ArrayList of type 'WordCount', which is a custom class designed to store words and their frequencies. (The code for the 'WordCount' class is provided below, along with the template for the 'WordListExaminer' class.) Each instance of 'WordCount' contains a single word, along with the number of times that word appears in the "listofwords.txt" disk text file. Objects of type 'WordCount' must always be inserted into the ArrayList in ascending alphabetical order based on the 'myWord' field of the 'WordCount' class.

In the accompanying template, the 'main' and 'readWordsFromDiskFile' methods have been completed for you. You will need to write the code for the 'addWord' and 'displayWordCounts' methods. The template also provides some additional lines of code at the top of the program.

The 'addWord' method, which is called from the 'readWordsFromDiskFile' method, should add 'WordCount' objects to, and update 'WordCount' objects already in, the ArrayList of 'WordCount' objects. The method takes one argument, a String, which is a single word to be added to the ArrayList of 'WordCount' objects.

The "displayWordCounts" method takes no arguments and should display the data stored in the 'WordCount' objects in the ArrayList. Output from the "displayWordCounts" method must include the information shown in the sample output below, and it must be formatted in a similar manner (columns, with a blank line after every five rows of data).

You must use the provided template when writing your program. With the exception of writing code where indicated for the 'addWord' and 'displayWordCounts" methods, you may not rearrange any of the code, add any code to the template, or modify the existing code in any way. Also, you must use the ArrayList 'wordList' of 'WordCount' objects, and you may not use any other Lists or Collections.

Your program does not need to perform any error trapping, other than what is already part of the provided template. You may assume that the word list will contain words with only uppercase and/or lowercase letters, and that each line will contain only one word.

## Sample "listofwords.txt" Word List

```
cat
go
elephant
CAT
bear
computer
CHAIR
cat
light
comPuteR
bEaR
aardvark
restaurant
area
COMPUTER
go
suspicious
telephone
computer
```

## Output Using Above Sample Word List

```
Alphabetical Listing of Words and Frequencies from "listofwords.txt":

 #    Frequency    Word
==============================
 1        1        aardvark
 2        1        area
 3        2        bear
 4        3        cat
 5        1        chair

 6        4        computer
 7        1        elephant
 8        2        go
 9        1        light
10        1        restaurant

11        1        suspicious
12        1        telephone

Total number of words:   19
Number of unique words:  12
```

## WordCount Class

```
// This class encapsulates a single word and its
//  corresponding frequency in a disk text file
public class WordCount
{
    private String myWord; // The word
    private int myCount; // Frequency of the word

    // Constructor
    public WordCount(String word, int count)
    {
        myWord = word;
        myCount = count;
    }

    public int getCount()
    {
        return myCount;
    }

    public void setCount(int count)
    {
        myCount = count;
    }

    public String getWord()
    {
        return myWord;
    }
}
```

## Template for WordListExaminer Class

```
import java.util.*;
import java.io.*;

public class WordListExaminer
{
    // Define an ArrayList
    private static ArrayList<WordCount> wordList;

    // Define a Scanner
    private static Scanner inFile;

    // Set the name of the disk text file containing the words to be read
    private static String fileName = "listofwords.txt";


    public static void main(String[] args)
    {
        // Create an ArrayList ('wordList') of 'WordCount' objects
        wordList = new ArrayList<WordCount>();

        // Read in all of the words from the disk text file
        readWordsFromDiskFile();

        // Use the ArrayList of 'WordCount' objects to display a table with each unique
        // word, along with the number of times the word appears in the disk text file
        displayWordCounts();
    }
```

```
    // This method reads all of the words in the disk text file 'fileName' and calls the
    // method 'addWord' to store the words and their frequencies; note that the ArrayList
    // 'wordList', which consists of 'WordCount' objects (which each hold a single word
    // and corresponding frequency), contains no 'WordCount' objects with duplicate words;
    // instead, the ArrayList contains only 'WordCount' objects with unique words and the
    // number of occurrences of each of those words in the disk text file
    private static void readWordsFromDiskFile()
    {
        try
        {
            // Create a Scanner object to read words from the disk text file
            inFile = new Scanner(new File(fileName));
        }
        catch (FileNotFoundException error)
        {
            System.out.println("ERROR:  File '" + fileName + "' not found!");
            return;
        }

        // Loop through the entire disk text file, until there are no words left to read
        while (inFile.hasNext())
        {
            String word = inFile.next();

            // Convert the current word to all lowercase letters and then call the
            // 'addWord' method to add the word to the ArrayList 'wordList'
            if (word.trim().length() > 0)
            {
                addWord(word.toLowerCase());
            }
        }
    }


    // This method adds a single word to the ArrayList ('wordList') of 'WordCount'
    // objects; if the word is already in the list (in the 'myWord' field of a
    // 'WordCount' object), the corresponding 'myCount' field will be incremented by 1;
    // if the word is not already in the list, it will be inserted alphabetically into
    // the list (into the 'myWord' field of a new 'WordCount' object) and its
    // corresponding 'myCount' field will be set to 1
    private static void addWord(String nextWord)
    {
        // [Put Your Code Here]
    }


    // This method displays the frequency of every unique word in the ArrayList
    // 'wordList', along with the number of unique words and the total number of
    // words in the disk text file 'fileName'; a blank line is displayed after
    // every five rows of data
    private static void displayWordCounts()
    {
        if (wordList.size() == 0)
            return;

        // [Put Your Code Here]
    }
}
```

## Advanced Options

1) In addition to displaying the word frequency list sorted alphabetically by word, display the list a second time, but have it sorted by frequency, with the most-used word shown first and the least-used word shown last. Since there may be multiple words with the same frequency, when displaying the list sorted by frequency, within each group of words that have the same frequency, make sure the sublist is sorted alphabetically in ascending order by word. For this option you may use multiple additional local (not global) lists within the 'displayWordCounts' method.

2) Thoroughly handle all types of entries when reading from the word list disk text file:
   a) Ignore blank lines within the text file
   b) Ignore (discard) words that contain hyphens or numbers (digits)
   c) Read lines containing multiple words (separated by one or more spaces or tabs)
   d) Accept words that contain apostrophes (e.g., don't, it's, I've, they'd, you'll)
   e) Accept and strip sentence-ending punctuation from words (period, question mark, exclamation point)
   f) Strip punctuation marks between or around words (brackets, braces, parentheses, quotation marks, ellipsis, comma, colon, semicolon, dash)

   For this option you may modify and add/delete code within the 'readWordsFromDiskFile' method.

3) Write the modified word list (see #2 above) to a new text file after each word has been processed. The words in the new text file must be in the same order as they appear in the original text file. Based on whatever processing is required in #2 above, the new text file may contain fewer words and fewer punctuation marks. In addition, the new text file should contain only one word per line, even if the original file contained lines with multiple words. You may either use a preset file name for the new text file, or ask the user to enter a filename. As a way to "test" this option, read the modified word list back into your program to see if the output is exactly the same as it is with the original word list. For this option you should create a new method named 'writeWordsToDiskFile' which you should call from the 'readWordsFromDiskFile' method.