**IndexMaker**

Original lab from: Litvin, Gary and Litvin, Maria. *Java Methods: Object Oriented Programming and Data Structures.* Skylight Publishing, 2011. pp 370-374.

In this lab you will write a program that reads a text file and generates an index of line numbers for each word. All the words that occur in the text should be listed in the index in upper case in alphabetical order. Each word should be followed by a list of all the line numbers of lines that contain that word. Study the input file fish.txt on the left and how it relates to the output file fishIndex.txt on the right. (Note the commas!)

A 12, 14, 15

ARE 16

BLACK 6

BLUE 4, 7

CAR 14

FISH 1, 2, 3, 4, 6, 7, 8, 9, 16

HAS 11, 14

LITTLE 12, 14

LOT 15

NEW 9

OF 16

OLD 8

ONE 1, 11, 14

RED 3

SAY 15

STAR 12

THERE 16

THIS 11, 14

TWO 2

WHAT 15

One fish

two fish

red fish

blue fish.

Black fish

blue fish

old fish

new fish.

This one has

a little star.

This one has a little car.

Say! What a lot

of fish there are.

This application consists of three user-defined classes: the driver, DocumentIndex, and IndexEntry.

The program uses ArrayLists in two ways: DocumentIndex *isa* ArrayList of IndexEntries and each IndexEntry *hasa* ArrayList field that holds the line numbers. Study this UML:

*Comparable<IndexEntry>*

ArrayList<IndexEntry>

String

*isa*

IndexMaker

IndexEntry

DocumentIndex

ArrayList<Integer>

*hasa*

The IndexMaker class is the driver class and is provided for you. Its main method prompts the user for the names of the input and output files, opens the input file, creates an output file, reads and processes all the lines from the input file, then saves the resulting document index in the output file. You don't have to code for reading or writing files in this lab.

Writing the IndexEntry and DocumentIndex classes is your job.

**IndexEntry**. An IndexEntry object represents one index entry. It has two fields:

private String word;

private ArrayList<Integer> numsList;

Write a constructor that takes a given word (a String), converts it to upper case, and saves it in word. The constructor should also initialize numsList to an empty ArrayList<Integer>.

The class should implement the following three public instance methods:

/\*\* appends num to numsList, but only if it is not already in that list. \*/

**void** add(**int** num)

/\*\* this is a standard accessor method \*/

String getWord()

/\*\* returns a string representation of this Index Entry in the format as   
 shown on each line of the output file--see the previous page. \*/

String toString()

**DocumentIndex**. A DocumentIndex object represents the entire index for a document. The index entries should always be arranged in alphabetical order. DocumentIndex extends ArrayList<IndexEntry>. Provide two constructors, one that instantiates a list with the default capacity, the other that instantiates a list with a given capacity. These constructors simply call the constructors of the superclass.

DocumentIndex should have the following instance methods:

/\*\* replaces all punctuation with "". Then calls addWord() on each string  
 in str. \*/  
**public void** addAllWords(String str, **int** lineNum)

/\*\* calls foundOrInserted, which returns a position. At that position,   
 updates that IndexEntry's list of line numbers with lineNum. \*/

**public void** addWord(String word, **int** lineNum)

/\*\* linear-search this DocumentIndex comparing word to the words in the   
 IndexEntry objects in this list, looking for the correct position of   
 word. If the IndexEntry for that word is already there, return its   
 position. If an IndexEntry with word is not in the list, then instantiate

and insert a new IndexEntry at the correct position. Then return that   
 position. \*/

**private int** foundOrInserted(String word)

**Text Files.** Test your program thoroughly on different text data files (fish.txt), including an empty file (fish1.txt), a file with several blank lines (fish2.txt), a file with lines that have leading spaces or punctuation (fish3.txt), a file with multiple occurrences of a word on the same line (fish4.txt), and a file with the same word on different lines (fish4.txt).

**Extension 1**: Make IndexEntry extend ArrayList<Integer>. What are the consequences?

**Extension 2:** Search using a binary search, not a linear search. The see the difference in efficiency, you will keep count of the total numbers of comparisons using two extra fields in the DocumentIndex class:

/\* EXTENSION ONLY \*/  
public static int linearCount = 0;//every time in the linearSearch a comparison   
 //is done, increase this variable.   
public static int binaryCount = 0;//every time in the binarySearch a comparison   
 //is done, increase this variable.