Programming Project 5 (Due 12-8-2016 at 2:00 PM)

Write a JAVA program to solve the following problem. Your program should properly compile and run. Your code MUST follow the documentation style used in your textbook. You need to upload into Moodle the following:

- From this point on, each of your projects is expected to be composed of multiple source code files. Thus to organize these files, all of them have to be part of a single package. Your package name MUST match the directory containing your code file. Finally, you need to export your package as a .JAR file then upload this single file into Moodle. If you are not familiar with how to export your code into a .JAR file, please check the following link. http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftasks-33.htm

Very Important Note: Do NOT upload any other files into Moodle; just ONE file with the extension ".jar" that is it.

Problem Statement:

In this project, you are asked to write a Java application that utilizes your knowledge about a number of data structures we have been discussing throughout the course of this semester. **The main task of this application is to automatically generate a book index for a given arbitrary text file**.

As you know, a traditional book index lists on which page each important/key word occurs. In the application that you will develop, you are required to generate an index for **ALL** words in the given file. To standardized testing of all students' submissions, all of you are required to use the given text file posted online next to this project statement. The file name is "**alice30.txt**" and it contains the famous ***Alice in Wonderland*** book that is freely available via Project Gutenberg. The simple given test file that you MUST use does NOT have page numbers and thus you will use chapters instead in your indexing.

While considering which data structure that can best fit this application, please remember that your index will look similar to the following:

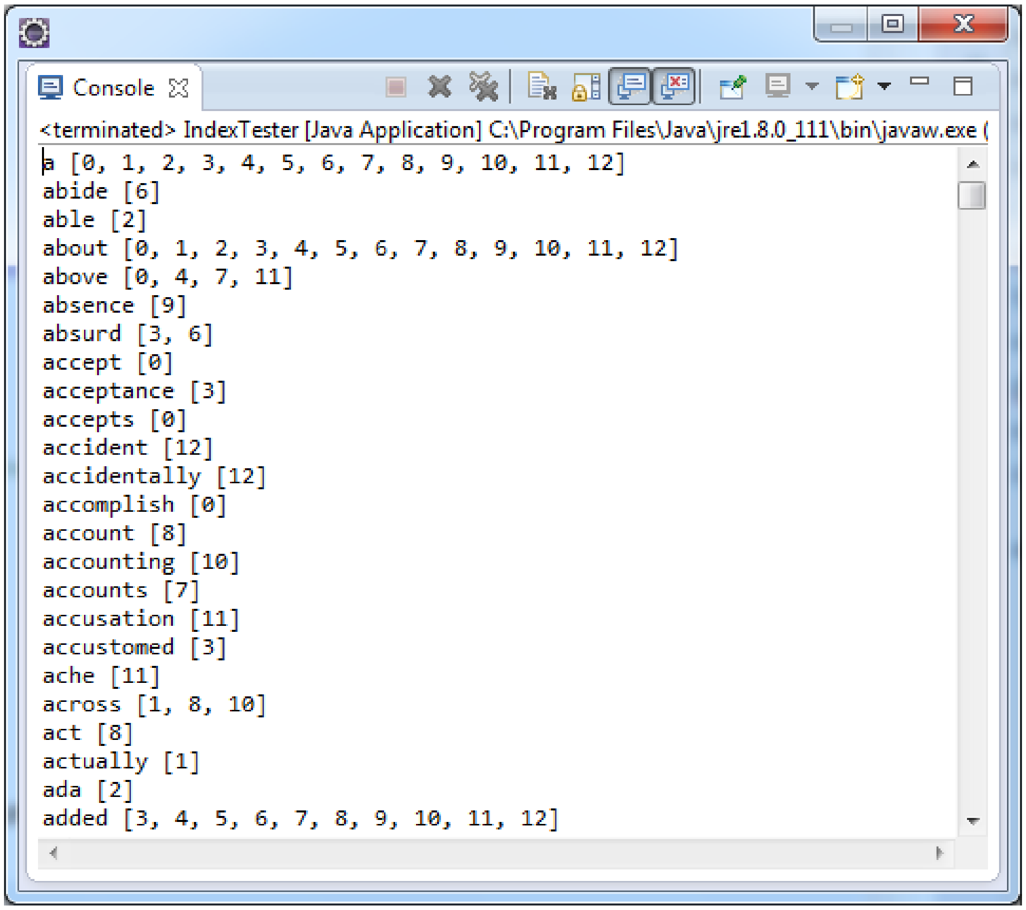
"Keyword" → {2, 3, 7, 9}

Where "keyword" is the word that you are trying to index and {2, 3, 7, 9} is the set of chapters that "keyword" occurs in. In other words, "keyword" occurs in chapters 2, 3, 7, and 9. If a keyword occurs multiple times in the same chapter, your index will ONLY list the chapter one time and thus maintain the set property. The structure of such index can be implemented using a Map whose keys are the Strings representing the words that you are indexing and the value associated with each key is a set of integer values denoting which chapters a particular key word occurs. Hint: your main data structure can take the following form. The choice of TreeMap and TreeSet will ensure that the data stored in these structures are sorted.

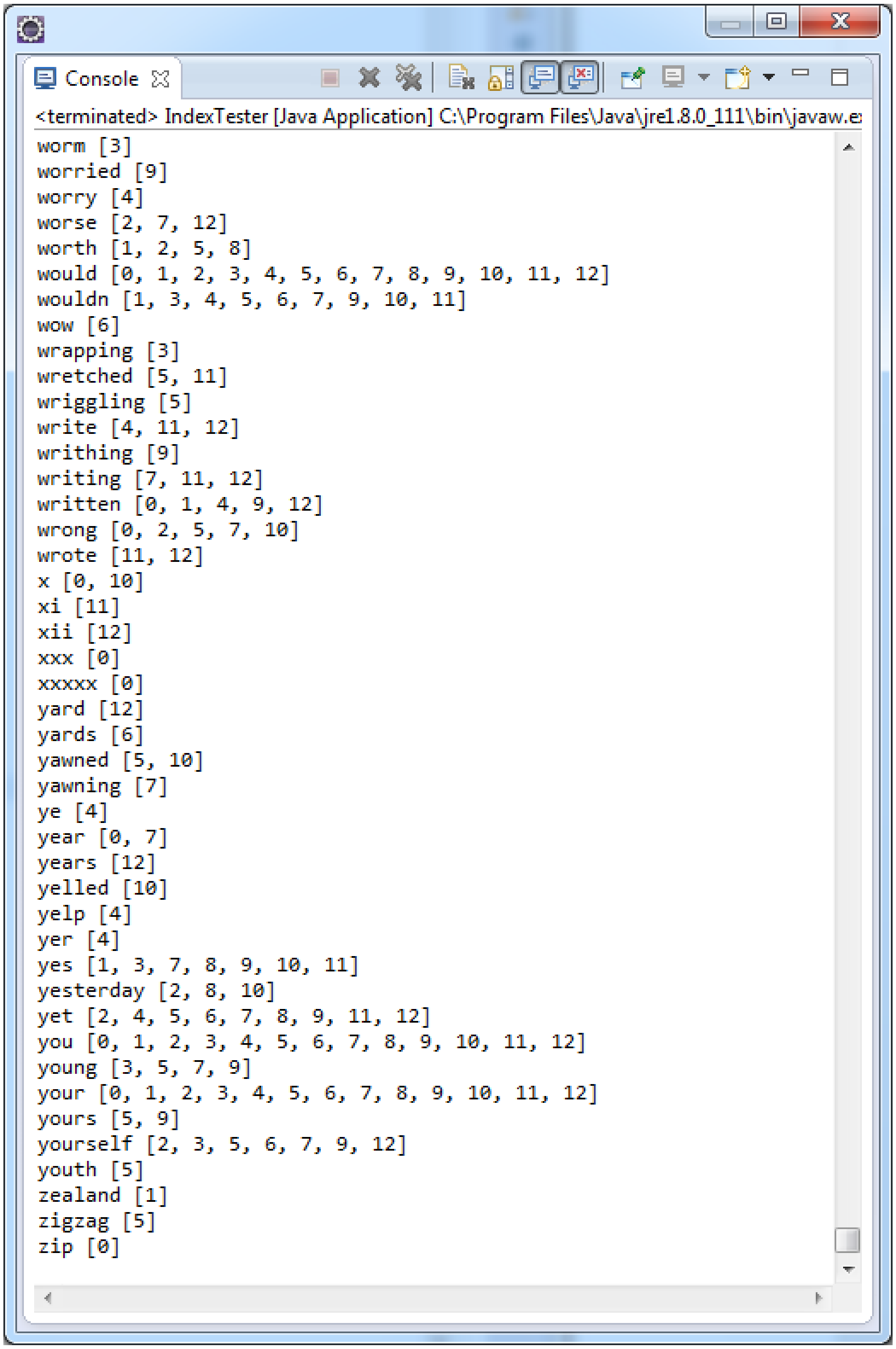
TreeMap<String, TreeSet<Integer>>

Your code is expected to have two files with the following functionalities:

- A Driver program that will create a Scanner object to open the given input file and make sure that ALL non-alphabetical characters are skipped. To do that you need to use the appropriate regular expressions with the .useDelimiter() method of the scanner class. Then, all data from the input file will be read and converted to lower case. The driver program will then invoke the appropriate methods from the MainIndexingClass to generate the desired index then display the generated index on the monitor. **See the two snapshots below for the first and last portions of the generated index**.
- A MainIndexingClass file that defines the selected data structure then provides appropriate constructor to initialize that TreeMap. This class also will provide all needed functionalities to generate and maintain the required index structure.



```
<terminated> IndexTester [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (
a [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
abide [6]
able [2]
about [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
above [0, 4, 7, 11]
absence [9]
absurd [3, 6]
accept [0]
acceptance [3]
accepts [0]
accident [12]
accidentally [12]
accomplish [0]
account [8]
accounting [10]
accounts [7]
accusation [11]
accustomed [3]
ache [11]
across [1, 8, 10]
act [8]
actually [1]
ada [2]
added [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```



```
<terminated> IndexTester [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.e
worm [3]
worried [9]
worry [4]
worse [2, 7, 12]
worth [1, 2, 5, 8]
would [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
wouldn [1, 3, 4, 5, 6, 7, 9, 10, 11]
wow [6]
wrapping [3]
wretched [5, 11]
wriggling [5]
write [4, 11, 12]
writhing [9]
writing [7, 11, 12]
written [0, 1, 4, 9, 12]
wrong [0, 2, 5, 7, 10]
wrote [11, 12]
x [0, 10]
xi [11]
xii [12]
xxx [0]
xxxxx [0]
yard [12]
yards [6]
yawned [5, 10]
yawning [7]
ye [4]
year [0, 7]
years [12]
yelled [10]
yelp [4]
yer [4]
yes [1, 3, 7, 8, 9, 10, 11]
yesterday [2, 8, 10]
yet [2, 4, 5, 6, 7, 8, 9, 11, 12]
you [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
young [3, 5, 7, 9]
your [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
yours [5, 9]
yourself [2, 3, 5, 6, 7, 9, 12]
youth [5]
zealand [1]
zigzag [5]
zip [0]
```

(End of Project)