PIC 20A, Spring 2016 – Homework 7

Assigned 5/13/2017, due 11 p.m. on 5/31/2017. Although your solution is not due until after the second midterm exam, I suggest you complete parts 1-3 and practice writing to files for part 7 *before* the exam, as this is a good example of file input/output.
Corresponds with Lectures 6F-8F. **To receive credit for this assignment, you must adhere to submission instructions at the end of this document.**

In this homework, you will analyze the vocabulary used in the complete text of Agatha Christie's novel "The Mysterious Affair at Styles".

Some background about the utility of this assignment is available at:
http://www.theguardian.com/books/2009/apr/03/agatha-christie-alzheimers-research

You may find it helpful to search for code inspiration online. If you do adapt code found online, please cite your sources in comments.

1. Download the plain text version of the novel from: http://www.gutenberg.org/ebooks/863 and save as a plain text file. Remove the header (everything before the heading "CHAPTER I") and the footer (everything after "THE END") as they are not part of the text of the book.
2. In the main function of `HW7Main.java`, read the complete text of the file into a string. Ensure that the read operation is buffered for efficiency. Once you have this working, change it: for even greater efficiency, assemble the text as a `StringBuilder` (not a `String`), *then* convert to a string at the end. ***Answer the question in comments: Why is this more efficient?*** You'll have to research the difference between `String` and `StringBuilder` to answer.
3. Prepare the text for analysis:
    a. Convert the text to lower case and try to remove all non-text characters and line breaks. Ideally, all processed text should consist of individual words separated by single spaces, *but do not spend much of time perfecting this.* You are welcome to borrow example code for this part (and cite your reference), or research regular expressions and write your own.
    b. Split the string into an array of strings in which each element consists of a single word. This is called "tokenization"; each element of the array is a "token".
    c. Save the Porter Stemmer available at http://tartarus.org/martin/PorterStemmer/java.txt into a `.java` file and use it to stem each of the words in the array. (The purpose of a "stemmer" is to reduce each token to its "stem" by eliminating common prefixes, suffixes, etc… For instance, "jumps", "jumping", and "jumped" should all be reduced to the stem "jump" for the purpose of counting distinct words. For more information, the Wikipedia article on "Stemming" may be of interest. *Note that stem does not necessarily mean "root" word;* ***not all stems are words at all and this is OK***. For more information, see http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html) It would be nice if there were a method that accepts a string and returns its stem, but that's not how this code works. I want you to figure it out based on the examples in the code and testing. Once you figure it out, I suggest you write a class or a method in you `HW7Main` class that works more intuitively.
4. Convert the array of word stems to a `HashMap`, then use it to count the number of times each word stem is used. In the comments of your code, answer the question: Why is a `HashMap` ideal for this purpose?
5. Order the word stems according to the number of times they are used (descending). Note that you may want to convert to some other kind of collection for this purpose. (I have considered several

ways to go about it, none of which are trivial or entirely automatic, hence the high point value of this part. Use any method that makes sense to you.)

6. Approximate the total number of words in the text (as measured by the number of not-necessarily-unique stems, about 58000) and the number of unique words (as measured by the number of unique stems, about 3900) and print these numbers to the console.
7. Save the results to files in two different formats:
    a. A text file `WordCount.txt` with each line containing "<stem>: <count>"
    b. A binary file `WordCount.dat` containing the (serialized) results. In order to receive credit for this, you must provide a script `HW7BinaryRead.java` that includes in its main method commands that load the data, deserialize it, and print the results to the console.

Important submission instructions:
All source files must begin with the line:
```
package hw7;
```
The main method of a single file, `HW7Main.java`, should perform all the processing except for loading the binary file created in 7b, which should be performed in `HW7BinaryRead.java`. Compress your entire project folder `HW7`, including the text of the book, and submit the resulting `HW7.zip` on CCLE.

Example contents of `WordCount.txt`. (Results may vary somewhat significantly depending on 3a.)

```
the: 2648
i: 1788
to: 1370
of: 1269
a: 1220
and: 1160
it: 1109
that: 1013
…
```

Example contents of `WordCount.dat`, interpreted as text. (Yes, it's mostly gibberish because it's a serialized object. It should be interpreted by `HW7BinaryRead.java`.)

```
¬í sr java.util.ArrayListxÒ™Ça———————————————————— I
sizexp
Iw
Isr
hw7.StemCount
ðš‰9<'! L countt Ljava/lang/Integer;L
stemt Ljava/lang/String;xpsr java.lang.Integerâ ¤÷‡8 I valuexr java.lang.Number†¬•
"à‹  xp
Xt ————————————————————thesq ~ sq ~   üt isq ~ sq ~   Zt tosq ~ sq ~
õt ofsq ~ sq ~
Ät asq ~ sq ~
^t ————————————————————andsq ~ sq ~
Ut itsq ~ sq ~   ————————————————————õt
thatsq ~ sq ~   ————————————————————°t ————————————————————yousq ~ …
```