Catching Plagiarists

This assignment presents a real problem that requires a creative software solution. It was originally formulated by Baker Franke at the University of Chicago.

Problem Statement
A lead Teaching Assistant for an Intro to Physics class at a Big-Ten University had roughly 800 students in the course and about 20 TAs who led lab sections. The lead TA got word that some students were plagiarizing their lab reports. This situation, however, was not an instance where submissions were being copied outright from the web, nor were students so brazen as to turn in another student's work as their own. Instead, the word on the street was the all-too-common digital age occurrence of one friend "helping" another by giving him/her a copy of a completed report, only to have the friend steal sections, words, and phrases from it and turn it in. The problem was that the T/A had no idea who the plagiarists might be, and all they had were 800 lab reports turned in digitally.

Your task:
Given a folder containing a set of text documents, identify the ones that contain a large amount of similar text – in other words, catch the plagiarists.

More formally:

Input: the name of a folder containing the documents to process
Output: some representation that identifies files that have many word sequences in common.

Details
What constitutes plagiarism?
For our purposes on this assignment we will hunt for common *n*-contiguous-word sequences among the documents, where the *n* is up to the user. For example: if a document contains the sentence:

> In 1492, Columbus sailed the ocean blue.

Then for *n* = 4, the 4-contiguous-word sequences from that sentence would be

> In 1492 Columbus sailed
> 1492 Columbus sailed the
> Columbus sailed the ocean
> Sailed the ocean blue

Input: The documents
Three sets of documents are provided for you to use. One set will be small (20 or so documents) for testing purposes. The other two sets will be larger (one is 100 documents, the other over 1300 documents) to test the scalability of your solution. (The documents came from www.freeessays.cc, a repository of *really bad* high school and middle school essays on a variety of topics).

Here is an example of how to read in the various subdirectories of a given directory (folder) into an `ArrayList` of `File` objects. (The `File "."` tells Java to look for the subdirectories in this particular folder. It is important to make this a *relative* reference, rather than *explicit* (i.e. Desktop\CompSci\ . . .) so that your code can find the subdirectory no matter what directory your source code is in.)

```
File dir = new File(".");
ArrayList<File> directories = new ArrayList<File>();
for(File folder : dir.listFiles())
   if(folder.isDirectory())
      directories.add(folder);
```

Here is an example of how to read in a directory of file names into an `ArrayList`, where `directory` is a `String` representing the name of the directory.

```
File dir = new File(directory);
String[] temp = dir.list();
List<String> files = new ArrayList<String>();
for(int i = 0; i < temp.length; i++)
{
   if(temp[i].endsWith(".txt"))
      files.add(temp[i]);
}
```

Here is an example of how to read a certain number of words (`numWords`) from a file, where `filename` is a `String` representing the name of the file.

```
Scanner file = new Scanner(new File(fileName));
while(file.hasNext())
{
   String phrase = "";
   // read 'numWords' words from file
   for(int j = 0; j < numWords; j++)
   {
      if(file.hasNext())
         // strip away all punctuation, and set to lowercase
         phrase += file.next().replaceAll("[^A-z]","").toLowerCase();
      else
         phrase = null;  // not enough words at end of file
   }
```

Output:
You could think of processing everything into an *N* x *N* matrix (where *N* is the number of total documents) with a number in each cell representing the number of "hits" between any pair of documents. For example: below is a small table showing the comparisons between 5 documents:

	A	B	C	D	E
A	-	4	50	700	0
B	-	-	0	0	5
C	-	-	-	50	0
D	-	-	-	-	0
E	-	-	-	-	-

From this table we can conclude that the writers of documents A, C and D share a high number of similar word phrases. We can probably say A and D cheated with a high degree of certainty. For a large set of documents, you may only want to print a matrix for those documents with a high number of commonalities above a certain threshold.

Printing an *N* x *N* matrix may be untenable for large sets, however. In fact, even maintaining a 2D array for large sets may not be tenable. You could instead produce a list of documents ordered by number of hits. For example:

700: A, D
50: A, C
50: C, D
5: B, E
4: A, B

Getting started and Steps to Take
Step 0 : Processing the Documents
See if you can simply output all of the *n*-word phrases from all of the documents in a given directory. This too needs to be done in steps:

0.0 Confirm you can read a directory and scan all the files in the directory.

0.1 Produce all of the *n*-word phrases from some file.

You might consider writing a helper method that, given the name of a file and an *n*, produces an `ArrayList` of all the *n*-word sequences from it. This will help you later on.

Step 1 : Produce a "hit list" for the small set of documents
Produce a pair-wise list of the number of *n*-word sequences each file in a directory has in common. For example, the output might be:

```
[FileA.txt, FileB.txt] -> 247   56.5%
[FileA.txt, FileC.txt] -> 7      0.3%
. . .
[FileY.txt, FileG.txt] -> 876   25.6%
```

1.0 You might want to start by writing a method that accepts two `ArrayLists` of word sequences and returns the number that they have in common.

1.1 Add in the ability to only show hits over a certain threshold.

1.2 Modify it to sort the output so that the greatest number of hits is first (hint: make a tiny object that holds two file names and a hit count. Write a custom comparator so you can use `Collections.sort` to sort them).

Try it out on the medium and large document sets to see what happens. You will get a passing grade for this project if this step works for the medium document set.

For purposes of validating your work, the following are the appropriate results from processing all of the files in the small number of documents directory, searching for 4-word phrases, and ignoring any file parings containing less than 10 duplicates.

```
[jrf1109.shtml.txt,   sra31.shtml.txt]      -> 1,589
[abf0704.shtml.txt,   edo26.shtml.txt]      ->   670
[abf0704.shtml.txt,   abf70402.shtml.txt]   ->   538
[abf70402.shtml.txt,  edo26.shtml.txt]      ->   181
[abf0704.shtml.txt,   edo20.shtml.txt]      ->    10
```

(Note that your specific numeric values may vary slightly from the above, depending on how you handle duplicate phrases that are contained in the same document. The ranking order of the file parings, however, ought to be the same.)

Step 2 : Adding User-Interface
Your program should prompt the user to input the directory of files they want to use, the *n* for the *n*-word sequences and the cutoff threshold for showing hits. A console-based interface is fine. Doing a GUI to collect this information is even better (explained below).

Minimum Requirement:
A console-based program that solves the catching plagiarists problem for the medium document set in a reasonable amount of time (less than 30 seconds). The output of the program MUST BE SORTED by highest hit-count first. Make sure your program can locate the document subdirectories no matter where your source code is located (i. e. don't hard-code the explicit subdirectory paths). It goes without saying, but it is expected that your program will successfully run after you turn it in.

Your project must also contain a README with the following sections.

```
PROJECT: Catching Plagiarists
AUTHOR: (Your name)
DATE: (Today's date)
PROGRAM DESCRIPTION:
(A few sentences only. What does this program do?)
ALGORITHM DESCRIPTION:
(This description should be in plain English for the layman. How
would you explain to a person who doesn't know much about
programming?)
ALGORITHM COMPLEXITY: (Worst-case Big-O notation)
```

Challenge
The real challenge is to catch the plagiarists in the large document set (~1300 documents). This will require some cleverness and ingenuity in order to do it in a reasonable amount of time, where "reasonable" means a minute or two. If the program takes more than a few seconds though it should frequently output some sort of progress indicator so that the user doesn't think the program has stalled.

Above and Beyond (optional)
A more dynamic user interface: allow the user to repeatedly re-process the documents by allowing the user to change the word-sequence length and "noise" level. Basically, a pretty-ification of a console-

based application. Create GUI for user input. The GUI must use something like `JFileChooser` to allow the user to select the document directory, rather than typing it by hand.

Turning it in

- Name your project file directory Catching Plagiarists.
- Zip up the entire project file directory by control-clicking on the directory name and then select Compress "Catching Plagiarists" from the menu.
- Submit the zipped directory using the link on the Schoology assignment page.

Academic Honesty

- This is an individual assignment, and not a group project. In submitting your project, you are implicitly stating that it is your own creation and not someone else's work (apart from what is already provided for you).
- Your code should reflect a unique solution to this challenging problem, which will be significantly different from anyone else's solution. You can count on your code being evaluated line by line against the work of others, so please do not be tempted to submit even portions of code that are not yours.
- Submissions that are deemed to be plagiarized from the work of others will receive a zero on this assignment, and cannot be resubmitted for re-evaluation.