CSCI 136 Programming Exam #1
Fundamentals of Computer Science II
Spring 2013

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. Carnivore was a system implemented by the FBI to monitor Internet traffic. Your task is to develop a similar program that can check a bunch of email messages to determine which messages contain certain keywords (e.g. "bomb", "cocaine"). To speed up scanning, your program should scan each email message in parallel. Each email is stored in an individual file. Start by downloading the file located at:

http://katie.mtech.edu/classes/csci136/carnivore.zip

Details. The program you are developing takes two command-line arguments. If given no arguments, it prints a helpful message and terminates:

```
% java Carnivore
Carnivore <file> <keyword 1> [keyword 2] ...
```

The first argument is the filename of a text file that contains a list of other text files which contain the actual email messages. For example, here is `file6.txt`:

```
mail/arnold-j/54
mail/bass-e/382
mail/baughman-d/367
mail/cuilla-m/76
mail/dasovich-j/602
mail/saibi-e/124
```

Each of the above is the path to a file that is in the `mail` subdirectory included in the `carnivore.zip` file. The user specifies one or more keywords on the command-line. You can assume keywords given on the command-line are in lowercase and only consist of the letters a-z and apostrophe.

Below are some example runs of the final finished program. Note that the order of keywords and the order of the list after the keyword may vary depending upon your implementation.

```
% java Carnivore file6.txt plutonium
plutonium: mail/saibi-e/124

% java Carnivore file6.txt bomb
bomb: mail/dasovich-j/602 mail/arnold-j/54

% java Carnivore file6.txt plutonium bomb silly
plutonium: mail/saibi-e/124
bomb: mail/dasovich-j/602 mail/arnold-j/54

% java Carnivore file6.txt afghanistan
afghanistan: mail/baughman-d/367 mail/dasovich-j/602 mail/arnold-j/54

% java Carnivore file60.txt plutonium bomb cocaine smuggling afghanistan
cocaine: mail/bass-e/382 mail/cuilla-m/76
plutonium: mail/saibi-e/124
afghanistan: mail/baughman-d/365 mail/dasovich-j/602 mail/baughman-d/367 mail/ar
nold-j/54
smuggling: mail/baughman-d/367 mail/dasovich-j/602
bomb: mail/dasovich-j/602 mail/arnold-j/54
```

Part 1. Develop a class `CarnivoreHits` that keeps track of which keywords have been found and where. Note this class should ***not*** be doing any file I/O. This class simply tracks the list of files that a given keyword matched as reported by a client program via the `add()` method. It can also print out the list of keywords that have been matched and in what file(s). The class has a simple API with two methods:

```
public class CarnivoreHits
-----------------------------------------------------------------------------------------
void add(String keyword, String matched) // track a keyword that has been found
void print()                             // print out hits to standard output
```

We have provided a test `main()` program. Here is our output:

```
cocaine: msg10.txt
bomb: msg1 msg3 msg10.txt
foo: bar
```

Output format. Each line of output should start with the keyword followed by a colon and then a space. After this should be listed all filenames that matched that keyword separated by spaces. You can output the keywords in *any* order you like. The matching filenames may be in *any* order, but a given filename should *only appear once* in a list for a particular keyword (e.g. `msg1` only appears once in the `bomb:` list even though it was added twice by the test `main()` program).

Part 2. The search for keywords needs to occur in parallel via a multi-threaded program. As such, you first need to develop the `CarnivoreWorker` class. An instance of this class is responsible for scanning the contents of a single filename for keywords and updating the `CarnivoreHits` object if a keyword is discovered. Here is a suggested API for the class:

```
public class CarnivoreWorker implements Runnable
-----------------------------------------------------------------------------------------
     CarnivoreWorker(String filename, ArrayList<String> keywords, CarnivoreHits hits)
void run()
```

Keyword matching. A worker should check each word in the file it is responsible for. Words in the files are assumed to be separated by whitespace. The emails contain mixed case as well as punctuation such as commas. You should match a keyword even if the email text is in a different case or contains character besides a-z and apostrophe. For example, if the email contains the sentence "In the Times, I've not found much information", the keyword "times" would match as would "i've".

Part 3. Create the main program `Carnivore`. It should operate as described in the introduction making use of your `CarnivoreHits` and `CarnivoreWorker` classes. Each file in the list of files given to Carnivore should be handled by a separate thread, doing its search in parallel. Make sure to handle concurrency properly so an accurate report of matching keywords is obtained regardless of how many files are searched simultaneously.