Threads Exercises

Re-submit Assignment

Due Feb 18, 2017 by 11:59pm **Points** 100 **Submitting** a file upload
File Types zip

Exercises - Threads

Answer the following questions by implementing the code samples. Upload all project files zipped to the exercise drop box.

Setup

Download the **following zip file** and add its contents to a new Java project in a folder called "lists." This zip folder contains 10 files, each of which has a list of words or phrases. The problems below will focus on querying these files through the use of threads.

Problems

1. Create a secondary thread that prints out the name of all files in your "lists" directory. For example, you would see the following output:

Analyzing lists director... *(printed from the main thread)*
Found file: list_1.txt *(printed from the secondary thread)*
Found file: list_10.txt
Found file: list_2.txt
Found file: list_3.txt
Found file: list_4.txt
Found file: list_5.txt
Found file: list_6.txt
Found file: list_7.txt
Found file: list_8.txt
Found file: list_9.txt
Done analyzing lists directory... *(printed from the main thread)*

Hints and requirements:

- Points will only be awarded if the output above is printed from the designated threads.
- To loop over a directory of files, you can use Java's File class (ie. File directory = new File("lists/");)
  - Hint: Take a look at the File.list() method in Java's API

2a. Create a new thread class that stores the name of a file in your "lists" directory. When ran, the thread should print out the total number of lines in the file (use a Scanner object to traverse the file). For example:

list_1.txt: 60388 *(printed from the secondary thread)*

Your thread class should look something like the following:

```
public class CountListThread extends Thread
{
   private String listName;

   public CountListThread(String listName)
   {
      this.listName = listName;
   }

   @Override
   public void run()
   {
      //count the lines in the file "listName"
   }
}
```

2b. Create a loop in your driver class that takes each file in your "lists" directory and passes the file to your thread class from 2a. This should allow several threads to run simultaneously, calculating the total file count of all files in the "lists" directory. Here is some sample output:

lists/list_2.txt: 4690 *(printed from the secondary thread)*
lists/list_9.txt: 32193
lists/list_10.txt: 22227
lists/list_3.txt: 41242
lists/list_5.txt: 21877
lists/list_6.txt: 82532
lists/list_7.txt: 64662
lists/list_1.txt: 60388
lists/list_4.txt: 81883
lists/list_8.txt: 67725

Hints and requirements:

- The order of lists printed above may not match your own (why?)
- Points will only be awarded if you start up 10 threads, one for each file above. Each of the threads should run simultaneously.

- *Hint: Create an array of Thread objects as displayed in class.*

2c. Practice retrieving the results from your thread objects by using the join() method. Do the following:

- Add a field called "lineCount" to your thread class from 2a.
- Save the result of your computation from run() to your lineCount field.
- Write a getter for lineCount.
- Remove your println() statement in run() that prints out the lineCount.
- Alter your code segment from 2b so that the main thread waits for all ten threads to complete work before retrieving results of each computation using your new getter.
  - *Hint: This will require you to call join() on each of your ten threads as displayed in class.*

3a. Create a word search thread that stores a file name (word list) and a search term (string) as a field. The thread should open the word list specified and loop over each line of the file. If a line matches the search term given than it should print out the following:

"term" found in lists/list_1.txt

Test your word search thread by prompting the user for a search term. Pass the search term and a word list filename to a new search thread. For example, your output might look like the following:

Enter a word to search for: apple
"apple" found in word-phrase lists/list_1.txt

Searching for a missing term should have no output.

3b. Alter your solution to 3a by starting several threads, one for each word list file, that each simultaneously search for the given search term. Make sure that your output prints which file a term was found within, for example:

Enter a word to search for: banana
"banana" found in lists/list_3.txt
"banana" found in lists/list_7.txt
"banana" found in lists/list_4.txt
"banana" found in lists/list_9.txt
"banana" found in lists/list_8.txt
"banana" found in lists/list_1.txt
"banana" found in lists/list_5.txt
"banana" found in lists/list_6.txt

4a. Create a multi-threaded solution for merging all words or phrases from each file into a single file. Use the following technique:

- Create a shared data area (critical region) that manages an array list of string values that will be pulled from each file. Use the following code to get started.

```
public class SharedData
{
   //create a static array list field here...

   public static void add(String word)
   {
      //add word to the array list
   }

   public static int wordPhraseTotal()
   {
      //return the number of elements in the array list
   }

   public static String getWordPhrase(int position)
   {
      //returns the element at index "position" in the array list
   }
}
```

- Create a new thread class that accepts a filename (word list) and opens the given file and adds each line of the file to the shared data class above (using the SharedData.add() method).
- Spin up ten instances of your new thread class, one for each word list file. This will add all file contents for all word lists to the array list stored in your shared data class.
- After starting the threads above, call join() on each thread to pause execution on the main thread until all secondary threads have completed.
- Write a final loop that uses a PrintWriter object to write the contents of your shared data to a file called master_list.txt.
  - You will need to use the wordPhraseTotal() and getWordPhrase() methods to loop over the internal array list in your shared data class.

To test your results, do the following:

- Print out the number of lines in your master_list.txt file using your thread class from 2a.
- Run your program a few times to view your results.
- Do you notice anything inconsistent?

4b. If you haven't already, alter your shared data class to make it thread safe. This will require that you use synchronization blocks with each interaction of your array list.