SWEN 601 HOMEWORK

Threads and Concurrency

You should have already accepted the GitHub classroom invite for this assignment, but if for some reason you have not, do so now. Create a package for this `homework`, and commit/push your project to GitHub.

For this homework you will solve several small problems using threads to implement programs with concurrent processes.

1. Begin by creating a class named `FileUtils`. Add a method `static void copyFile(String source, String destination)` method that copies the `source` file to the `destination` location. For example, if called as `copyFile("a.txt", "b.txt")` the contents of the file "`a.txt`" would be copied into the file "`b.txt.`"
    a. You must write the code to copy the file using `java.io.*` classes (e.g. `File`, `FileInputStream`, `FileOutputStream`, etc.).
    b. ***Do not*** assume that the files will contain only text.
    c. You should try to copy efficiently (i.e. not a single byte at a time).
    d. It is suggested that you test your method by copying the provided files from the `input` directory to the `output` directory in your project.
    e. The method should rethrow any exceptions that may occur.

2. Add a new method named `clearDirectory(String name)` to your `FileUtils` class.
    a. This method should delete all of the files in the directory with the specified name. Hint: read the `java.io.File` documentation to see how you might list the files in a directory and then delete them.
    b. If a file cannot be deleted for some reason, throw an `IOException` with an appropriate message including the absolute path to the file.
    c. Rethrow any exceptions that might occur. You should test the function using your `output` directory.
    d. ***Be careful*** not to delete the files in the `input` directory or other directories in your project. A backup of the `input` files has been uploaded to MyCourses just in case.

3. Create a class named `FileCopier`. Add a main method.
    a. Two directories should be specified as command line arguments. If there are not exactly *two* command line arguments, print a usage message, e.g. "`Usage: java homework.FileCopier <input dir> <output dir>`", and exit with an error code.

b. Use your `FileUtils` class to make sure that the output directory is cleared.
c. Use your `FileUtils` class to copy the files from the input directory to the output directory. Print a message just before each file is copied.
d. If the file is a directory, skip it. Print a message indicating that it was skipped.
e. Print a message at the end indicating how many files were copied, the total bytes copied, and the time that it took to copy them.
f. If any exceptions occur, print a detailed message and exit with an error code. ***Do not*** rethrow the exception.
g. Below is example output:

```
Clearing directory 'output'.
  Copying file 'books.png'...
  Copying file 'buttercup.jpg'...
  Copying file 'cutie.jpg'...
  Skipping directory 'skip_me'...
  Copying file 'tacos.jpg'...
  Copying file 'words.txt'...
Copied 5 files (14832559 bytes) in 459 milliseconds.
```

4. Create a new class, `CopierThread` that copies a single file in a separate thread of execution.
   a. The class will need to the name of the file to copy as well as the name of the destination directory into which the file should be copied.
   b. You will need to implement the copy in the thread's `run()` method. This method cannot throw checked exceptions (e.g. `IOException`); if an exception occurs, save the exception in a field and abort the copy.
   c. The name of the file should be printed just before it is copied.
   d. It should provide *accessors* for the number of bytes copied, and an exception (if one occurred).
   e. Add a `main` method that creates one `CopierThread` per file in the input directory and produces the same output as the `main` method in `FileCopier` (including if one or more of the threads encounters an exception). This `main` method should keep track of the total time.
   f. Below is example output:

```
Clearing directory 'output'.
  Copying file 'books.png'...
  Skipping directory 'skip_me'...
  Copying file 'buttercup.jpg'...
  Copying file 'cutie.jpg'...
  Copying file 'tacos.jpg'...
  Copying file 'words.txt'...
Copied 5 files (14832559 bytes) in 58 milliseconds.
```

5. Create a new class, `Counter`, that prints a single number in a separate thread.
   a. Add a `main` method that accepts a single command line argument that specifies the number to count up to. If there is not exactly *one* command line argument, print a usage message, e.g. ***"***`Usage: java homework.Counter <number>`***"***, and exit with an error code.
   b. ***Create*** and ***start*** one of your `Counter` threads for each number. For example, if the number is 10, you should create 10 threads with the numbers 1 through 10 (inclusive).
   c. You must insure that the numbers print in order. In order to do this, each thread should wait until the previous thread has finished before printing its own number.
   d. Test your implementation several times with at least 100 threads. Insure that the numbers print in the exact correct order each and every time.
   e. Below is example output:

```
1
2
3
4
5
6
7
8
9
10
```

When you are finished, draft a new GitHub release. Use the "Source code (.zip)" link to download your release. Submit this to the assignment on MyCourses. Late submissions are not accepted.

Grading Rubric

Exceptional Performance 4	Competent Performance 3	Acceptable Performance 2	Developing Performance 1	Beginning Performance 0
All assignment instructions followed. Program runs as described.	A small number of minor problems, e.g. 1-2 instructions not followed, specifications not met, few commits, few comments, etc.	Several minor problems, e.g. several instructions not followed, specifications not met, few commits, few comments, etc.	Many minor or major problems, e.g. code does not compile or run, does not meet functionality requirements, etc.	Very little effort or no submission at all.
100%	88%	75%	50%	0%