Lab – Java – Threads Using Executor

Overview

Write an application that runs threads using an Executor.

Create Console Application

Create a new Java console application.

Single Thread Pool

- Create a single thread executor thread pool in main.
- Create a for loop in main that runs 10 times.
    - Each time through the loop submit a task to the executor. The task's code should print its thread name and the loop number (do not print the loop number in the main thread). The other thread that is created should NOT have a loop inside of it (only need one print statement in the task, no loop in the task itself). Make the output statement look like the following:
      Message from pool-1-thread-1, loop num = 0
      You can write a printf similar to the following (tName is a variable holding the thread name and num is a variable holding the loop number):
      System.out.printf("Message from %s, loop num = %d\n", tName, num);
- The executor should be shutdown. Make sure that the main thread waits a little while for the executor to shutdown.

Note on Thread Names: pool1-thread-1 is the thread name that gets returned by the system. It is indicating that it is the first thread in thread pool 1. Use the following to get the thread name of the current thread:
Thread.currentThread().getName()

Here is some sample output:
Message from pool-1-thread-1, loop num 0
Message from pool-1-thread-1, loop num 1
Message from pool-1-thread-1, loop num 2
Message from pool-1-thread-1, loop num 3
Message from pool-1-thread-1, loop num 4
Message from pool-1-thread-1, loop num 5
Message from pool-1-thread-1, loop num 6
Message from pool-1-thread-1, loop num 7
Message from pool-1-thread-1, loop num 8
Message from pool-1-thread-1, loop num 9

Fixed Thread Pool

Add code to create another executor for a fixed thread pool that contains 3 threads. Submit 10 tasks and print the thread name and loop num as before. Make sure the main thread cleans up the executor and waits a little while for its threads to complete (hint: shutdown and awaitTermination).

Here is some sample output (yours may differ slightly):
Message from pool-1-thread-2, loop num 1
Message from pool-1-thread-3, loop num 2
Message from pool-1-thread-1, loop num 0
Message from pool-1-thread-1, loop num 5
Message from pool-1-thread-3, loop num 4
Message from pool-1-thread-3, loop num 7
Message from pool-1-thread-2, loop num 3
Message from pool-1-thread-3, loop num 8
Message from pool-1-thread-1, loop num 6
Message from pool-1-thread-2, loop num 9

Sum Data in Files Using Threads

You will write code to calculate the total sum of data that is spread across five different files using threads. Use a fixed thread pool of two threads when coding the solution.

- First, create five different files with data in them. Put the numbers 1-5 in the first file, 6-10 in the second file and so on. You can name the files whatever you want.
- In the Main class declare a static int member variable to hold the grand total.
- In the Main class declare a static Queue of String member variable to hold the filenames.
- Add the names of the five different files to the queue (you can do this in main).
- Create a static method to add all the numbers in a file:
  public static void sumFile()
  This method should do the following:
    - Remove one filename from the queue.
    - Open the file.
    - Sum all the data in that file.
    - Add the file's total to the grand total.

  IMPORTANT: You will need to use synchronization when removing a name from the queue and when updating the grand total. Use synchronized blocks. You should create two static "lock" members in the Main class.
- In main (or another method if you want), create an instance of a fixed thread pool of 2 threads.
- In main (or another method if you want), create a loop to run sumFile on 5 different threads. Each time through the loop it should create and run a thread. The thread should call the sumFile method.
- Print the grand total of all numbers in all files at the end of main. You must add code to the application that ensures that the main thread only prints this total after all threads have finished.
    - Declare a static CountDownLatch member in the Main class.
    - Decrement the CountDownLatch in sumFile after the file has been processed.
    - Add code to main to wait on the CountDownLatch (after starting the threads).

After you code the solution remove the code that waits on the threads and see what happens. You will most likely see a grand total of 0. Why?