1) Remove the comments within the **run** method of the **IncreaseFactorial** class and observe the different threads executing with their various **hold** values.

Do all four threads get to work all of the time?

Answer: No, there are some instances in which all of the work completes in less than 4 threads.

2) Update the code to use only two threads. Does it run faster? Why, or why not?

Answer: The code runs faster with only two threads for 5 factorial. This is likely due to the overhead required to create the new threads. The amount of work done by one thread is less than the amount of overhead required to create additional threads. There is also less work required because with fewer threads, there is no need to execute additional method calls and loops.

4 threads:

Enter n (-1 to stop) > 5

5! is

recursive 120.0

iterative 120.0

result for 100 to 76 = 5.0

result for 75 to 51 = 20.0

result for 50 to 26 = 60.0

result for 25 to 1 = 120.0

iterative parallel idea 120 total time = 2355615

Hello Thread-0

Hello Thread-1

hold = 2

hold = 3

Hello Thread-2

hold = 4

hold = 5

Hello Thread-3

iterative parallel 120 total time = 5866458

result for 100 to 76 = 20.0

result for 75 to 51 = 60.0

result for 50 to 26 = 120.0

result for 25 to 1 = 120.0

iterative parallel recursive 120.0

Enter n (-1 to stop) >

2 threads:

Enter n (-1 to stop) > 5

5! is

recursive 120.0

iterative 120.0

result for 100 to 76 = 5.0

result for 75 to 51 = 20.0

result for 50 to 26 = 60.0

result for 25 to 1 = 120.0

iterative parallel idea 120 total time = 2640522

Hello Thread-1

Hello Thread-0

hold = 3

hold = 2

hold = 4

hold = 5

iterative parallel 120 total time = 4957137

result for 100 to 76 = 20.0

result for 75 to 51 = 60.0

result for 50 to 26 = 120.0

result for 25 to 1 = 120.0

iterative parallel recursive 120.0

Enter n (-1 to stop) >

3) Why does the parallelized method run slower than the serial version?

Answer: There is additional overhead required to create multiple threads. Additionally, the multipleIncrement method is defined with the synchronized keyword, meaning that only one thread can access the method at a time. Because of this, a thread may be waiting for another thread to finish accessing the method. The serial version does not have these obstacles. Depending on the number of executions required to calculate the factorial, the serial version may complete all of the work in a shorter amount of time than with a multithreaded approach.