Antonio Scalfaro

CMSC 335

Homework #4

1. The interesting part of the TaskThreadDemo is the use of threads in its execution. I would first start by thinking about what the program may look like without the use of threads. You would see that each character is printed out 200 times in order that they appear in the array. However, the use of threads changes the output of the program because it will print the characters concurrently. Every loop in the main method creates a new Thread instance and is started at the end of the loop. The thread will start to execute the PrintChar run() method but will continue on to the next iteration of the loop. As the next Thread is created and started, the output will be paused from the first thread to start executing the new thread. The process continues like this as more threads are created and executed. The characters will continue to print in short bursts before pausing and handing off resources to the next thread. This will continue until the threads have completed their task and each character has been printed 200 times.
2. If the ts.start() call on line 7 is changed to ts.run(), the program will be significantly changed. While the start() method will create a new thread for execution on every call, concurrently printing the characters, the run() method would invoke the run() method of the Thread object in the same thread the run() method is being called in. This creates a situation where it is sequential instead of concurrent and therefore each character would be printed 200 times one after the other, instead of the bursts of characters a concurrent program would print.
3. If Thread.yield() is added between lines 23 and 24, the output may look fairly similar to the original code, characters being printed by concurrent threads in bursts. The difference between adding yield and the original code is that the scheduler is given more flexibility to pause a thread and execute another thread of similar priority. This will lead to more varied output than not having the yield call.
4. If Thread.sleep(500) is used in place of Thread.yield(), the effect on the performance of the program would be substantial. The sleep() method call will create a situation where the interleaving of the threads is more evenly spaced and the output becomes more consistent and predictable. Each thread will print the character, wait the allotted time, and then print the character again. In the waiting period, resources will be allocated to the next thread, and it will perform the same task. This creates a more predictable output pattern.
5. If the Thread.sleep(500) call was moved from the run() method to the main() method right after the creation of the thread, it would have a significant change to the way the program performs. The sleep call would pause the main thread for the signified time, but the created thread would continue its execution in that time. The result would be output that looks like the sequential output; each character being printed 200 times before the next character is printed. This is specific to this case, however. Take for instance, if the sleep call is only 1 millisecond or the task being performed is more complex than printing a character 200 times, the threads would start to interleave as the next thread was created and started. Our program will appear as a sequential call only because 500 milliseconds will be enough time for the program to finish printing the character 200 times before the next thread is created and started.