Homework 4: Answers to Questions

Jade Pearl

CMSC 335-7381 Object-Oriented and Concurrent Programming

Professor Andrew Seely

**Question 1**:

It seems that the given program uses the PrintChar class to create and run multiple threads. The program prints one of 4 characters in a string array: “a”, “X”, “+”, or “.”. And each character can print a total of 200 times (making a total of 800 characters in the output). Here are the interesting elements related to threads that I found:

* The program uses the **String []** **sa** to create a new thread for each character in the array.
* The **PrintChar** class uses the **Runnable** interface which provides a **run()** method that executes at the start of each thread to print the strings
* The constructor in the **PrintChar** class kind of plays with the threads a bit because it is used in each thread’s instance and initializes the String used in that thread, **ch**, and sets the number of times each String is printed in the thread (variable **n**)
* The for loop in the main method calls **PrintChar** 4 times and initializes 4 different threads when it initializes a new **Thread** object, **ts** and calls the **ts.start()** method.
* The run() method in the PrintChar class executes with each thread’s start call. It prints the specified character from the String array, **ch**, **times** number of times (as specified in the for loop).

Output 1: A screen shot of a computer code

Description automatically generated

Output 2:

A screenshot of a computer screen

Description automatically generated

These two outputs are clearly different. The order of characters printed is unpredictable due to the concurrent nature of the program. Since the threads are all running at the same time (one thread for each string in the array), the strings are trying to print at the same time, so they output to the console in randomized orders. However, each character prints no more than 200 times as intended.

**Question 2**:

The next question is to understand what changes when the call to start() on line 7 of the program is changed to a call to the run() method. To explore this change, I commented out the **ts.start()** on line 7 and then replaced it with **ts.run()**. The output of the program changed drastically. All strings from the String array in main printed out in order 200 times per string following the order of the array.

My output:

A screen shot of a computer screen

Description automatically generated

From this result, I can conclude that using **run()** directly instead of the **start()** method caused the concurrency of the program to halt multithreading. Henceforth, each string is printed on a thread one after the other instead of all at once. Instead of the output being unpredictable, the user will know the output by default due to the sequential nature of multithreading being disabled.

**Question 3**:

Here are the first two runs without adding Thread.yield():

A screen shot of a computer code

Description automatically generated

And here are the two runs after adding the Thread.yield() call between lines 23 and 24:

A computer screen shot of a code

Description automatically generated

The output after adding the Thread.yield() statement becomes more sporadic in the order of which each string prints. There are visibly more interruptions of sequences being printed than the output has without the use of yield(). This method gives the threads the ability to allow other threads to have the same priority of execution. That is why the output of strings seem to be a mix of the characters printed in smaller groups because they are all trying to print more concurrently. The threads are allowing each other to go with the same priority so there is likely smaller chunks of a string being printed at once than before yield() is added.

**Question 4**:

Here is my edit of the code so that **Thread.sleep()** is called after every character is printed to sleep for 500 milliseconds:

A screen shot of a computer program

Description automatically generated

The output of this code shows each character printed is not repeated twice in a row and almost in a pattern that does change once or twice:

A screen shot of a computer program

Description automatically generated

This change provided a much more structured output with delays and the program took a lot longer to execute. Before, the execution was instant, but now it took almost a whole 2 minutes to print everything it needed to print. The sleep of each thread makes it this way because when the first thread prints, it sleeps and the next thread prints and sleeps, and the next, and the last. After a 500ms sleep, the thread prints again and sleeps, repeating the whole process. Because of this pause, the concurrency of the program shuts down a little and the program takes significantly longer due to the pauses from the **Thread.sleep(500)** statement.

**Question 5**:

Here is my edit of the code so that **Thread.sleep()** is called after each Thread is created in main and sleeps for 500 milliseconds:

A screen shot of a computer program

Description automatically generated

The output strangely shows a sequential printing of the characters like in question 2 when ts.run() is called instead of ts.start() which got rid of multithreading (the box around the a was my cursor, it is not part of the output):

A computer screen with white and blue text

Description automatically generated

Adding the Thread.sleep() statement visibly gets rid of the multithreading in the program because each thread’s activity already finishes and when the next thread is created, the preceding thread has completed its sleep and printed out the strings. It prints sequentially in the order of the String array, but it takes a little longer to process and print because of the 500 millisecond sleep in each thread. The ***same*** ***output*** happens if the try/catch code block with the Thread.sleep() statement is written before the ts.start() statement.