**Homework 4**

1. (20 pts) For the following program, explain the interesting elements related to threads. Focus on explaining the output of the program.
2. public class TaskThreadDemo {
3. public static void main (String args []) {
4. String [] sa = {"a", "X", "+", "."};
5. for (String s: sa) {
6. Runnable ps = new PrintChar (s, 200);
7. Thread ts = new Thread (ps, s);
8. ts.start ();
9. } // end for each character
10. } // end main
11. } // end class TaskThreadDemo

11

12 class PrintChar implements Runnable {

1. String ch;
2. int times;
3. public PrintChar (String c, int n) {
4. ch = c;
5. times = n;
6. } // end constructor
7. public void run () {
8. for (int i = 0; i < times; i++) {
9. System.out.print (ch);
10. } // end for loop
11. } // end method run
12. } // end class PrintChar

**ANSWER:**

**Line 5: Runnable ps = new PrintChar(s, 200);**

**This initiates a Runnable object instance in the PrintChar class with the parameters of**

**(String s, int 200) passing on to the PrintChar constructor. The constructor then takes the values and stores them as variable String ch and int times respectively for this class instance.**

**Line 6: Thread ts = new Thread(ps, s);**

**This initiates a new Thread object instance in the PrintChar class passing the Runnable object of this instance and a string as parameters.**

**Line 7: ts.start();**

**This method begins execution of the thread instance by calling the run() method is the class that has the implemented Runnable interface. In this case, the PrintChar class.**

**Line 12: class PrintChar implements Runnable {**

**Here the PrintChar class is implementing the Runnable interface because the PrintChar class instance is used to execute a thread. The Runnable interface frees the PrintChar class from having to extend the Thread class. This is the preferable way to execute a thread if only the run() method is to be overridden. A class should only extend another class if it intends to modify or enhance the superclass.**

**Line 21: public void run() {**

**This method is called by the start() method and takes no arguments. It contains the executable code for the thread. In this program, it prints the string set by the constructor 200 times before terminating the thread.**

1. (20 pts) What is changed if the method called on line 7, start(), is replaced with run()? Explain (of course). Focus on explaining the output of the program.

**ANSWER:**

**The threads are executed sequentially, one at-a-time because they are no longer simply being started, but rather are being instructed to execute completely. This results in the output being 200 ‘a’s, 200 ‘X’s, 200 ‘+’s, and then 200 ‘.’s. This is because when you call the run() method, the next thread will not be called until the current thread has finished.**

1. (20 pts) What is changed if the method Thread.yield() is added between lines 23 and 24? Explain. Focus on explaining the output of the program.

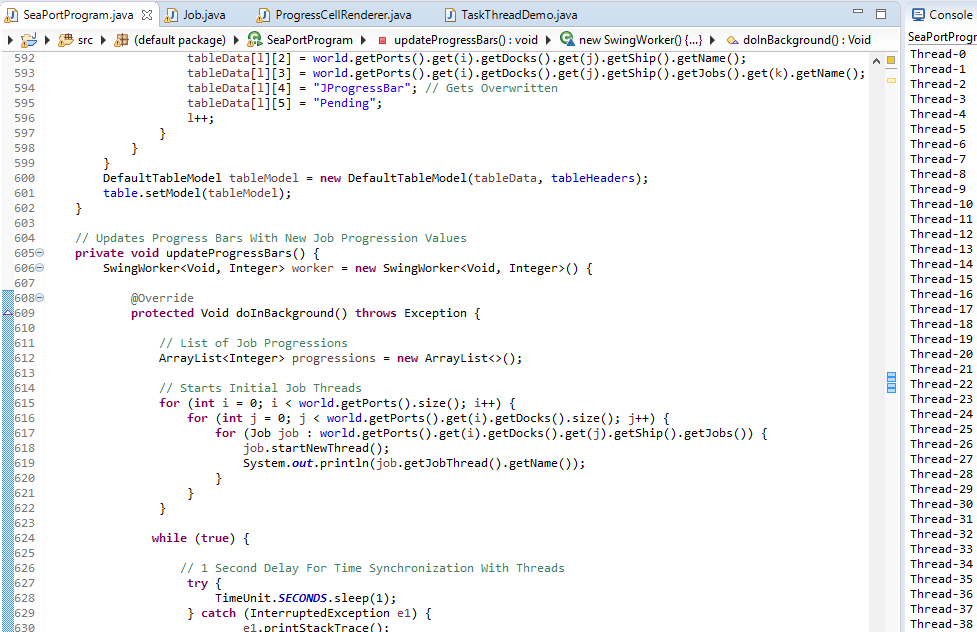
**ANSWER:**

**There doesn’t seem to be any consistent difference in output. This because the Thread.yield() function gives the thread scheduler free reign to determine whether or not to pause execution of the thread. It does this by checking the priorities of all the threads to determine if there are any of equal or higher precedence than the current thread. If none exist, then it continues to execute the current thread. Since none of the threads in this program are assigned a priority, they all share the same priority as the first thread to execute. So effectively, there is no difference in output from the initial program.**

1. (20 pts) Using the jconsole or jvisualvm utilities provided in the JDK, list and explain some of the threads that are created in your code for Project 3. Note that you can name the threads created in the program, as is done on line 6 in Problem 1 above, which can make this discussion a lot easier to follow.

**ANSWER:**

**As of Project 3, each job that belongs to a docked ship has its own thread that is initialized as soon as the user clicks the “Start Jobs” button on the GUI and whenever a new ship takes the place other one that has completed all its jobs. They have generic names such as Thread-0, Thread-1, … etc. Here is a screen capture of initial 39 job threads from the input file aSPad.txt:**



**Once a job is completed, the thread is terminated meaning that the number of threads is always equal to the number of jobs actively being executed.**

1. (20 pts) Explain how the java.util.concurrent.Semaphore class might be used in Project 4 to coordinate the requirements of the various jobs. Then address the question of whether or not this actually makes sense in the context of the requirements of program. In other words, can you suggest approaches to handling shared resource pools that would be simpler than using semaphores?

**ANSWER:**

**The Semaphore class is used to maintain a set of permissions that are either acquired or released with the acquire() and release() methods respectively. Semaphores are commonly used to limit the number of threads that have access to some resource. In the case of Project 4, semaphores can be used to limit the use of resources (persons with certain skills) amongst various threads (jobs). Creating threads for every job and then blocking them is not ideal for multithreading because it increases the amount of memory that has to be allocated to those threads.**

**Instead, it would be best to implement the Executor interface. This interface creates a thread pool where a set number of threads are created to work on tasks. Any new tasks wait until a thread is finished completing a task before being worked on. This can be implemented in Project 4 by assigning a thread for each dock (thread) and then having it work on a docked ship’s jobs (task). Ships that are in the ship queue can wait until a dock is finished with a ship’s tasks before utilizing the dock’s thread to execute its jobs.**

1

**Grading Rubric:**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Meets** | **Does not meet** |
| Problem 1 | **20 points** | **0 points** |
|  | Explains the interesting elements | Does not explain the interesting elements |
|  | related to threads. Focuses on | related to threads. Does not focus on |
|  | explaining the output of the program. | explaining the output of the program. |
| Problem 2 | **20 points** | **0 points** |
|  | Explains what is changed if the | Does not explain what is changed if the |
|  | method called on line 7, start(), is | method called on line 7, start(), is replaced |
|  | replaced with run().Focuses on | with run(). Does not focus on explaining |
|  | explaining the output of the program. | the output of the program. |
| Problem 3 | **20 points** | **0 points** |
|  | Explains what is changed if the | Does not explain what is changed if the |
|  | method Thread.yield() is added | method Thread.yield() is added between |
|  | between lines 23 and 24. Focuses on | lines 23 and 24. Does not focus on |
|  | explaining the output of the program. | explaining the output of the program. |
| Problem 4 | **20 points** | **0 points** |
|  | Lists and explains the threads that | Does not list or explain the threads that |
|  | are created in your code for Project | are created in your code for Project 3. |
|  | 3. |  |
| Problem 5 | **20 points** | **0 points** |
|  | Explains how the | Does not explain how the |
|  | java.util.concurrent.Semaphore | java.util.concurrent.Semaphore might |
|  | might class be used in Project 4 to | class be used in in Project 4 to coordinate |
|  | coordinate the requirements of the | the requirements of the various jobs. |
|  | various jobs. |  |
|  |  | Does not address the question of whether |
|  | Addresses the question of whether or | or not this actually makes sense in the |
|  | not this actually makes sense in the | context of the requirements of program. |
|  | context of the requirements of |  |
|  | program. | Does not suggest approaches to handling |
|  |  | shared resource pools that would be |
|  | Suggests approaches to handling | simpler than using semaphores. |
|  | shared resource pools that would be |  |
|  | simpler than using semaphores. |  |

2