Fhatuwani Mokwenda – PCP2 Report

CLUB SIMULATION REPORT

The aim of this project was to correct and extend an existing simulation code of patrons in a club. I had to use synchronization methods to ensure that the existing code adhered to synchronization limits and maintain thread safety and liveness. (See README file)

The following were the behaviour/ concurrency issues I observed in the existing simulation:

1. The START button did not start the simulation, simulation started automatically when the program is ran.

2. Pause button did not pause/resume the simulation (basically had no impact on the simulation).

3. Multiple patrons entering through the entrance at the same time.

4. Maximum limit is not upheld (more that the max patrons are entering the club) .

5. No liveliness.

6. Potential for deadlock.

<u>**My Solutions**</u>:

ClubSimulation:

1. I added the flags, “paused” and “simStarted” to represent the state of the simulation, I made “paused” a volatile variable because I wanted its state to be immediately visible to all threads when it is modified in order to prevent potential synchronization issues and race conditions.
2. I added two synchronized method, “isSimStarted()” and “isPaused()” that are used to query the stat of the simulation, either if its paused or started. This ensures that there are less shared variables between the classes while ensuring that multiple threads and access and modify the state of the simulation without conflicts or contradictions.

GridBlock:

1. I implemented a thread-safe task tracking by changing the type of the variable "isOccupied" from an int to an AtomicInteger. This was to further ensure that only one thread can occupy a grid block at a time.
2. I added a ReentrantLock to ensure the exclusive access to methods like get() and release(). This was to maintain a thread safe environment when multiple threads attempt to interact with the same grid block.

ClubGrid:

1. I added a try-finally block in the move() method to ensure that the acquired resources are properly released, enhancing robustness and further reducing the potential for deadlock.

Clubgoer:

1. I added a flag that is of the type AtomicBoolean to ensure safe visibility and modification across threads.
2. I implemented the checkPause() and checkStart() methods with safe variables that ensure or rather enhance the safety of the threads' interaction.

Lessons Learnt:

I think the biggest lesson for me was the difference in volatile and Atomic variables. I tried to make the variable, "paused" and AtomicBoolean since, it is the obvious better option but this lead to unnecessary complexity and did not provide any significant benefits. My code ran way more slowly when "paused" was an AtomicBoolean which lead me to believe that this change added additional synchronization overhead. In other words, sometimes volatile is the better option, just like in this case.