**CSC2002S ASSIGNMENT 2 REPORT**

Francis Mutetwa

MTTFRA005

Date: 27/08/2023

Changes and Additions to the Club Program

Added Classes:

No additional classes were added to the existing program.

Modifications to the existing classes and thread safety:

1. **ClubGoer.java**

Within this class I modified the checkPause() and the startSim() methods so that they can be able to check if the buttons within the simulation box are pressed. Both of these methods utilize flags that are initialized as AtomicBoolean within this class. There is the AtomicBoolean pauseFlag that is initially set to false and the AtomicBoolean startFlag, initially set to false as well. Both the methods make use of a spin wait while loop. The simulation will keep waiting until the start button is pressed

1. **ClubSimulation.java**

Within this class there are two methods that were initially empty. The actionPerformed() method for the pause button was initially empty and I edited it such that it can be able to pause and resume the simulation. To do this I utilized a for-each so that I iterate over all the patrons. So, each patron/thread will only be paused or resumed if the pauseFlag is triggered. A similar approach was applied to the actionPerfomed() method for the start button. The only difference is that the start button in this case only works once and I had to change the condition within the while loop. To start the simulation and it does not act as a resume button when the program has been paused.

In this class, I in order to make sure the threads that are paused and resumed are concurrent, I synchronized clubbers

1. **ClubGrid.java**

Within this method I implemented a spinning wait through the use of a while loop and a wait(1) and notifyAll(). The spinning wait I implemented it within the enterClub() method, this was to ensure that the number of patrons in the club did not exceed the specified limit. The leaveClub() method I also had to synchronize it. To ensure threads do not bump into each other once they’ve all be woken up, I used a 1 millisecond delay so that threads do not bump into each other when the spinning wait condition becomes false. To make sure threads are not bumping into each other in the club and are not occupying each other’s grid boxes I synchronized the move() method within this class to ensure liveness. Patrons maintain a realistic distance from each other (one per grid block), when this is done.

The simulation had other behaviour rules that were already in place. The quit button terminates the execution without me editing the code. The Patrons enter through the entrance door, and they exit through the exit doors without any editing as well.

Deadlocks:

The program was free from any deadlocks when I tried it out with different arguments.

Limitations:

I mainly used the spin wait in this program for conditional variables and these could require more computational power compared to other techniques and mechanisms. Again, other drawbacks are when pausing the simulation there is a slight delay when pausing. This could be because of contention caused by a lot of synchronization points because a lot of threads might be contending for the resources therefore it leads to delays in execution for the pause.