ADMSHA064 – Shaahid Adams  
CSC2002S  
Assignment 2 – Concurrency Club Simulation  
24 September 2023

**Start**  
A latch object was utilized to create the start functionality. The Club Simulation object holds the latch while all Clubgoers threads have a reference to the latch object; they are put into a waiting state until the latch is opened. This is more efficient than a ‘spin’ because the waiting threads are not utilising CPU time. When the user clicks the start button, the latch is opened in the “OnClick” event of the start button.

**Pause – Resume**

An atomic Boolean, stored in the Club Simulation as a static variable was used to keep track of the running state of the program. Having this variable be static meant that all Clubgoer threads could read from it as they checked the pause state throughout their runtime. Each Clubgoer thread would regularly check if the program was paused and change to a state of waiting if necessary. Importantly, they could be notified by the Club Simulation button click event to wake up and resume execution.   
The pause/resume “OnClick” event inside the Club Simulation served a dual purpose, switching the run state to the opposite of whatever it previously was and notifying all threads when needed.

**Counters**

The People Counter object was shared between the Club Simulation, Counter Display and Club Grid objects. To prevent interleaving, all compound methods on the People Counter object were synchronized. Atomic variables were also utilized to prevent race conditions.  
  
It was important to maintain an accurate account of all people waiting outside the club, those inside, and people that had left. These three numbers needed to always add up to the max people. If not, that would mean somebody got deleted or wrongfully created i.e. a concurrency error.

**Entrance and exit locking**

In the Club Grid object, validation to enter club method was done in the “enterClub” function. The code to check if the club had space and the entrance was not blocked had to be synchronized to prevent interleaving, as well as to put the thread into a waiting state if the conditions on the entrance atomic Boolean were not met.

The “leaveClub” method was used to ‘talk’ to the ‘enterClub’ function. Here the counters were correctly incremented and decremented. Notify all was called to tell the waiting threads to try again to enter.

**Maintaining 1 block per clubgoer**The get Boolean inside the Club grid was synchronised to prevent two threads from standing on the same block. The release method did not need to be synchronised as it was modifying an atomic variable.

**Issues and challenges**This wait-notify system, mentioned above, was not entirely successful as first. This happened because the notify that was triggered on leave club was often lost as someone was standing in the doorway and thus could not enter. To Fix this, I also called the notify inside the move function, checking if the current thread had moved off the entrance.

**Liveness and deadlock**

To prevent liveness issues, I only synchronised methods that were absolutely necessary. Patrons are able to move around, all at the same time. Deadlock was prevented by avoiding the use of two locks inside a synchronised block.