Problem 1: Start button initiates the simulation

I created a j**ava.util.concurrent.CountDownLatch** object with only 1 countdown,
I called await() on **ClubSimulation** in line 159, prior starting the Clubgoer threads(patrons).
I called countDown() when the start button is pressed and then the simulations begins;

Problem 2; Game pauses when user presses pause button;

I created a flag boolean using AtomicBoolean class, **pause**. Prior the start of simulation, pause == false, and when the the game has started and the user presses pause, Pause state changes to its opposite state, **pause.set(!pause.ge())**; If pause.get() changes to false, then the main threads wakes up all the 'patrons' (and Andre)

checkPause() method in **Clubgoer (and Barman)** class, first, I synchronized the block using **this** keyword as a key and called **wait().**

Problem 3: Inside the club, patrons maintain a realistic distance from each other (one per grid block).

There's an AtomicInteger object in Class **GridBlock**, that returns -1 if it is not occupied.
So, if a patron wants to occupy a certain block, they first check whether the grid they want to occupy is occupied or not. AtomicInteger was used instead of a normal integer, because, if a normal integer were used, Clubgoer threads can read stale value of the integer, thus making them occupy the same square.

Problem 4: Patrons enter through the entrance door and exit through the exit door

3.1 Entrance door

After the the simulation has started, the patrons has started, and the threads all want to enter to the club at the entrance door simultaneously.
If a patron wants to get into the club, they must enter through the entrance grid. So, they must hold the entrance key, and then check whether the entrance block is occupied or not.
Condition variable:If the entrance is occupied, then a method called wait() is made using the entrance **GridBlock** as a caller, so that the holder of the key gives away the key, until it is woken up.
Since I'm using a while loop, if threads are woken, they will again check whether the entrance is occupied or not.

3.2 Exit door

I used the same philosophy as 3.1. When a patron want to exit, they must first be the holder of the key to the exit door. The key is the exit **GridBlock**, and the holder must (**Condition variable)**check whether the exit door is occupied or not, else they cannot occupy the grid, thus they cannot exit.
So the patron must wait till the exit door is unoccupied.

Problem 5: Patrons must wait if the club limit is reached or the entrance door is occupied

Within the same breath of Problem 3, before the patron enters club, not only do they check whether the entrance door is occupied, but also check whether the number of patrons inside is less than the maximum capacity stated.

Extra Problem: Andrew the Barman

For this problem, I needed to create a separate class, Barman.class extends Clubgoer.
In this class, I have a PriorityQueue<Clubgoer> object, which will store the patrons that are at the bar.
When a thirsty patron arrives at the bar, the patron is added on the queue.
Class Barman have a method called customerArrived(Clubgoer obj) and this method is called everytime the patron arrives at the club. Then I called wait() upon the patron
There is also a void method called serveCustomer().
Inside, the serveCustomer method, I poll a Clubgoer object in the queue.A local boolean variable initially at a true state called serve. The serve state is changed when Andre is opposite;y facing the patron. And the served patron is awakened.
Clubgoer.getX() == Barman.getX();

I instantiated Andre, Barman object, inside the ClubGrid class, so, each and every ClubGrid object, there's 1 Barman object.
When the simulation starts, Barman is positioned behind the Bar counter, and directly opposite the entrance door.