CSC3002F Assignment OS2

Sabir Buxsoo - BXSMUH001

1. **Which threads run in the programs?**
    a. **CounterDisplay** - This thread updates display of the counters
    b. **Customer** - This thread contains all the necessary methods for each Customer
    c. **Inspector** -This thread checks for social distancing violation
    d. **ShopView** - Updates canvas and buttons for visualizing the simulation

2. **Which classes are shared among threads?**
    a. **CustomerLocation** - Used by Customer, Inspector, ShopView
    b. GridBlock - Used by ShopView
    c. ShopGrid - Used by ShopView
    d. PeopleCounter - Used by CounterDisplay, Inspector

3. **Synchronization Mechanisms added to each class and why**
    a. CustomerLocation: No Modifications were made to this class. However, The Atomic Variables mechanism was used here. When it comes to multithreading, if two or more threads try to get and update the value at the same time, it may result in lost updates. When two or more threads are trying to access the variables in this class a lock is placed on the variable and no updates can be made until the lock is released. Atomic variables are thus used for synchronization purposes, providing the lock mechanism, preventing race condition and enforcing liveness.
    b. GridBlock: In this class, Semaphores are used to enforce Mutual Exclusion. Mutual Exclusion guarantees that only one thread can access a shared variable at a time. In this case, Mutual Exclusion is used to ensure that only one customer is in a single grid block as any time. This is designed so that no two customers are sharing a grid block which will violate the social distancing rule.
    c. ShopGrid: In this class, the Multiplex Pattern is used (Reference: Little Book of Semaphores). The Multiplex Pattern is an extension of the Mutual Exclusion pattern. It allows multiple threads to run in the critical section at the same time. However, an upper limit is enforced on the number of concurrent threads. It is made sure that no more than `n` threads can run in the critical section at once. For this situation, we are basically limiting the number of customers in the shop at once and making sure that only one customer is at the door at once. By making use of Semaphores, the synchronization constraint is enforced by keeping track of the number of people inside the shop at once (maxPeople, and upper limit of threads allowed) and barring arrivals when the shop is at maximum capacity. Another Semaphore is used to keep only 1 customer at the door.
    d. PeopleCounter: All methods in this class have had the `synchronize` keyword added to them. The use of `synchronized` methods ensures that those methods are locked for shared recourses. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes the task. This ensures no race condition or deadlock happen.

4. **How liveness was ensured in code?**
   Liveness was ensured in the code by providing mechanisms to allow methods and variables to be blocked while being used by threads to prevent other threads from accessing them. This guarantees that threads can take turns in running the critical sections. For example, in the code atomic variables are used in CustomerLocation to enforce liveness. Another example is by making use of Semaphores or the `synchronized` keyword across the classes.
5. **How did you protect against deadlock? Was it necessary?**
   Deadlock occurs when multiple threads require the same lock but obtain them in different order. In this assignment it was necessary to protect against deadlock to ensure a smooth running of the simulation. The way deadlock was prevented was by making sure that threads acquiring was in the correct order. Another situation was in the case of ShopGrid where two Semaphores were defined (multiplex and door). In this case it was important to have the acquire() and release() methods in correct order to avoid processes from blocking indefinitely.

Reference:

http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf
https://www.baeldung.com/java-atomic-variables
https://www.geeksforgeeks.org/semaphore-in-java/