Name: Khulekani Jali

Student No.: JLXKHU003

Report

This is the report for Assignment 4: Enforcing Pandemic Lockdown Level 3 Rules. I did not make any changes to the CustomerLocation class. I may use customers and threads interchangeably within the report.

1. a. At the beginning of the program, 20 customer threads start running, but as the customers leave the shop, the number gets decremented.
   b. Counter display threads, keeps track of customers waiting, inside, and left inside the shop
   c. Shop View threads, runs the simulation animations.
   d. Inspector threads. That check if social distancing violations have been broken.
2. Shared classes: ShopGrid, GridBlock, CustomerLocation, PeopleCounter.

3. For the ShopGrid class, I added 4 Semaphores.
   The first one is called *move* and is initialised to 1(to allow the first customer to enter). The purpose of this semaphore is to block an incoming customer from entering the shop when the previous customer hasn't yet moved from the entrance block. This is achieved by acquiring the semaphore for the first customer to enter the shop, and releasing the semaphore if *that* customer moves to a separate block. I did this by making a check in the *move()* method, checking whether the customer's current location is the origin block. Once released the value becomes 1 again, allowing the next customer to enter, without them occupying the same block.
   Note: *To prevent threads from moving back to the entry block and bypassing the above mechanism. I prevented all threads movement to the entry point.*

   The second is called *space,* and is initialised to 0. This semaphore works with another semaphore called *empty* which is initialised to minus the maximum number of people allowed in the store i.e. –(maxPeople). It's purpose is to keep track of the number of people that have entered the shop. The *empty* semaphore releases(increments) every time a customer enters the shop, when the semaphore value is 0(meaning we have reached the maximum), the next thread i.e. (maxPeople+1) calls *space.acquire()* and waits for the *leaveShop()* method to be called(a customer leaves the shop) so the *space* semaphore can be released then allow the next customer to enter the shop, decrementing the semaphore value back to 0 and blocking the next customer from entering. Repeats till the last thread leaves the shop.

   I Also inserted a *synchronized* in the *move()* method, this is to make sure only one thread can update the occupied status of a given block at a time.

   For the GridBlock class I made sure that the *get()* method returns true if block is empty and false otherwise. Also added *synchronized* in the *release()* method, this is to prevent data races. Only one method may release a block at time.

4. By making sure each semaphore is released at the appropriate time.

5. This was not necessary as there weren't any circular-waits. Since threads waiting on a semaphore release are never holding any resources, this means deadlocks can never occur.