1. My project simulation
    a. My project was a Java project using semaphores and threads to simulate the customers, employees, and parts of a DMV, and their concurrent simulations. I realized that the main parts performing the integral actions of the DMV were the customers, agents, information desk, and the announcer, which meant I had to create a thread or an array of threads for each particular action performing component. I started out with creating single threads for the info desk and the announcer, and then I created arrays of agent and customer threads for the number of agents and customers I needed, all before starting threads. Each type of thread was given its own class for its own functions, and each class overridden the main method with a loop, except the customer since that's the only entity that ends once they get their license, unlike other components of DMV. The program started with the customer signaling a semaphore to indicate entry, as that's an indication that DMV needs to be running business in real time, whereas other components wait for a customer, or for other signals directly or indirectly resulting from a new customer entering the DMV.
    b. I've implemented my customer thread class to let the information desk know they came to the DMV, which then queues itself to the queue of newly entered customers. It will then wait for a number, and let the info desk know it got its number and is in the waiting room. From there, the customer waits for the announcer to call them to move to the agent line, when it will then move to the agent line and indicate that they're ready to be served. The customer will then wait to be served, and then let the agent know they're actually being served. Then, the customer will wait until the agent tells it to take an exam, which it will let the agent know when it finishes. After that, the customer will wait to get their license, and leave when it receives the license. Finally, it finishes and frees up another agent for customers waiting in line.
    c. I've implemented my information desk thread class to wait until the customer enters, dequeue the top element from a queue of newly entered customers, assign a number, and assign numbered customers to the waiting room queue; after it is set to let the customer know it has been numbered.
    d. I've implemented my announcer thread class to wait for a spot in line to begin with. Then, I've had the announcer wait for a customer in the waiting room, peek at the waiting room queue, and let that customer know it has been called to wait in the agent line. I've then made it to wait until the customer enters the agent line, dequeue the customer from the waiting room queue and enqueue that customer into the agent line queue, and let the agent(s) know that a customer is waiting in the agent line.
    e. I've implemented my agent thread class to wait until a customer is in the agent line, then dequeue the agent line queue to start serving that customer, then let the customer know it's being served. Then I had the agent wait until the customer was aware of the service, before asking the customer to take the photo and eye exam. After, I had the agent wait until the customer took the exam, and then I had the agent give the customer its license. I've then had the agent wait until the

customer receives the license, before signaling to that customer they're finished with renewing their license. Finally, I had the agent wait until the customer left and freed it up, before signaling to the announcer there's one more spot in the agent line.

f. I've also created a static class in order to use acquire() and release() as wait() and signal() respectively, in order to use terms I was more familiar with, through the pseudocode examples in the slides.

2. Difficulties encountered
    a. I've encountered numerous difficulties throughout this project, both at the initial and latter stages. A major difficulty I've encountered was the negative effects of a moderate amount of procrastination, where I've waited until one week prior to the deadline to start implementing my solution. However, I've started building the threads and the classes for the threads, until I've attempted to make each customer cooperate with the info desk across two different semaphores. The first problem I encountered was I forgot to start all my threads, which prevented my program from running, as all threads needed to be started in a concurrent process programming project. Another difficulty I've encountered was the organization of my parameters, where I haven't organized my semaphores, counting semaphores, semaphore arrays, queues, and other variables; and this has made it more difficult in which I used some semaphores out of the order I've initially stated them in, and I've fixed this by spacing out different types of semaphores. Also, I've also had issues where some runs do not assign a customer to an agent, which I've had to test numerous times across several days. I've fixed this issue by queuing and enqueueing outside the customer thread class, and instead queuing in the other classes, upon instruction from semaphore(s) of the customer thread class. Also, another thing that was helped was to save assigned numbers and agents inside the customer thread class object, and to save the customer servicing in the agent class object, all through pre initialized parameters.
3. What was learned
    a. The main thing I've learned from this project was not to procrastinate, or put off an assignment for more than a few days after introduction, because this has given me less time and put me in a situation where I had to undertake this project amidst many other committments. Another thing I've learned was that queueing and dequeuing has to be under mutual exclusion critical sections (sometimes inside other critical sections) in order for a customer to be served twice, or for a customer to be skipped in the final output. Also, I've learned that code after wait(semaphore1) and before signal(semaphore2) is code in the "critical section" which will be running for the executing object. I've also learned that it's safe to queue and enqueue in objects that keep running, as opposed to one and done like the customer of the DMV. I've also learned that adding a semaphore or two for a remedy between instructions is a safe option. Finally, I've learned that clean code and parameters are better to understand and remember both for short and long term.

4. Results
    a. Despite procrastination and numerous errors that took many different approaches to debug, test, and implement, I'm proud of myself for successfully completing this project. I understand in the output that code is not always ordered, as one customer may finish faster than another, or one agent may take more time with a certain customer, as all threads are running at once until the simulation is over. I knew that each customer was running, and each agent served another customer after the previously served customer was joined. I realized then and there I believed I had implemented it to the best of my ability.