CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 2 – **Due date: December 10 (for pseudo code)/December 14 (for Java code)**

The project must be done individually. No exceptions.
You are asked to synchronize the threads of the following story using semaphores and operations on semaphores. Do NOT use busy waiting.

The initial regular weight for project 2 (as a programming project) was 6%. You have the following choices:
1. You can submit a pseudo-code implementation; in that case the project will weight 5% instead of 6 % (you lose 1% on the weight). Due Date: December 10th.
Or
2. You can submit a java-code implementation; in that case the project will weight 7% instead of 6% (you get 1% EC on project 2's weight). I wished I could give you more but too many of you are plagiarizing.
Due Date: December 14th.

DO NOT SUBMIT BOTH TYPES OF IMPLEMENTATION: EITHER YOU SUBMIT THE PSEUDO-CODE, OR YOU SUBMIT THE JAVA-CODE.

1. Pseudo-code implementation
You are asked to synchronize the threads of the following story using semaphores and operations on semaphores. Do NOT use busy waiting. Use pseudo-code similar to the one used in class (NOT java pseudo-code). Mention the operations that can be simulated by a fixed or random sleep_time.
Your documentation should be clear and extensive.
Explain the reason for each semaphore that you used in your implementation. Explain each semaphore type and initialization. Discuss the possible flows of your implementation. Deadlock is not allowed.

Default values for some variables are:
Customers: 15
Floor-clerks: 3
Cashiers: 2
Mini-size: 4

CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 2 – **Due date: December 10 (for pseudo code)/December 14 (for Java code)**

2. java-code implementation

The number of customers should be read as command line arguments.

Default values for some of the variables are:
Customers: 15
Floor-clerks: 3
Cashiers: 2
Mini-size: 4

Java – Code Implementation

Using Java programming, synchronize the threads, in the context of the problem. Closely follow the implementation requirements. The synchronization should be implemented through Java semaphores and operations on semaphores (acquire and release)

For Mutual Exclusion implementation use **Mutex semaphores**, no volatile variables.

For semaphore constructors, use ONLY:

Semaphore(int permits, boolean fair)
Creates a Semaphore with the given number of permits and the given fairness setting.

As methods use ONLY: acquire(), release(); You can also use:

getQueueLength()
Returns an estimate of the number of threads waiting to acquire.

hasQueuedThreads()
Queries whether any threads are waiting to acquire.

DO NOT USE ANY OF THE OTHER METHODS of the semaphore's class, besides the ones mentioned above.

Any wait must be implemented using P(semaphores) (acquire).

Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Document your project and explain the purpose and the initialization of each semaphore.

DO NOT use synchronized methods (beside the operations on semaphores).

Do NOT use wait( ), notify( ) or notifyAll( ) as monitor methods. Whenever a synchronization issue can be resolved use semaphores and not a different type of implementation.

CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 2 – **Due date: December 10 (for pseudo code)/December 14 (for Java code)**

You should keep the concurrency of the threads as high as possible, however the access to shared structures has to be done in a Mutual Exclusive fashion, using a mutex semaphore.

Many of the activities can be simulated using the sleep(of a random time) method.

Use appropriate System.out.println( ) statements to reflect the time of each particular action done by a specific thread. This is necessary for us to observe how the synchronization is working.

Submission similar to project1. Name your project: **YourLastname_Firstname_CS340_p2 Upload it on Blackboard.**

Shopping at the Mega store

A customer walks into a Mega showroom and browses around the store for a while before (s)he finds an item (s)he likes. When a customer does so, (s)he must go to the floor-clerks and receive a slip with which (s)he can pay for the item and then take it home.

Floor clerks **wait** (use semaphore(s)) for customers to arrive; then, they help them with whatever information they need. However, floor clerks can only help one customer at a time; therefore, a customer must **wait** for an available clerk to help him/her.

After all of the customers are assisted (implement **ME with semaphores** if shared variables) the floor clerks **wait** for closing time.

Once a customer gets the information needed from the floor clerk, (s)he will go to the cashier to pay for the item.
There are two cashiers. One takes only cash, the other only credit cards. The customer will decide in which way (s)he will pay, cash or credit (determined randomly) and will get on the corresponding line.

After the item is paid for, the customer will take a break in the cafeteria and let the rest of the customers do their shopping.

When the shopping is done, customers will use minibuses to get to their new destination. The minibus has a capacity of *mini_size* seats. Before they board the minibus customers form groups of size *mini_size*. Next they will wait for the departure. Once a bus can be filled up, it is time for departure; it is the closing time for all the threads. Note: the very last bus might be incomplete but if no customers are left then it will depart.

CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 2 – **Due date: December 10 (for pseudo code)/December 14 (for Java code)**

The clerks wait for customers and serve the customers. Once they are done serving they will wait for closing time. Once all the buses are departed, it is closing time.

Good luck.