**Design**:

1. **List of Semaphores:**

* ***Semaphore max\_capacity***: This semaphore represents the max customers come inside Post office at any time. The max\_capacity id 10.
* ***Semaphore service\_counter:*** This represents number of postal workers providing service to customers. We have 3 postal workers to service customers.
* ***Semaphore cust\_ready:*** This represents that customer has entered post office and is ready to be served by Postal worker. Initial value is 0.
* ***Semaphore Scales:*** This represents a resource postal worker will use while mailing a package. Only one Postal worker can use this resource at a time. Initial value is 1.
* ***Semaphore leave\_counter:*** This represents that customer has been served by postal worker and it is ready to leave counter. Initial value is 0.
* ***Semaphore mutex1:*** This represents that only 3 customers can be served at any given time as there are 3 postal workers. Initial value is 3.
* ***Semaphore mutex2:*** This represents that only one Customer can go into queue. Initial value is 1.
* ***Arraylist<Semaphore> service\_finished:*** This represents total number of customers in post office and their initial values which is 0.As soon as they are served they will Join.

**SUMMARY:**

Normal use of threads may not give mutual exclusion all the times as OS will decide about which threads to process at any point of time. Having these in mind, Sometimes Mutual exclusion is required in order to access critical section. Using only threads will not give us that privacy in critical section as it is dynamic. To avoid this, we use semaphores with threads which give us Mutual Exclusion with increased accuracy and efficiency. Semaphores make sure that only process can access critical Section while other waits for the process to complete. The main motive of our project was to serve customers in postal office in a parallel and synchronized way. To attain this parallelism, we used Semaphores for Customers and Postal workers to provide mutual exclusion while serving different customers. Once the customer goes to any of the postal worker assigned, it leaves only when it is served and no customer can interrupt in between while the customer is served. To make sure that this property holds, we use semaphores. The main difficulty in this project was to attain Synchronization. We need to make sure where the semaphores should signal or wait for other process and whole program should work accordingly. Using threads with semaphores gave me a clearer idea of the use of threads, and how use of semaphores gave more sustainability to program. As far as learning is concerned, implementing this gave me some insights about how threads and semaphores actually works, the logic behind

Actual implementation. The result of the program was synchronized, like first the customers are created, max capacity customers entered post office, then postal workers are created, and postal worker will start serving customers one by one. Once the customer is served, it will signal another customer to come and get served. After customer is served, it will terminated and Join another customers. Once all customers are served, the system will exit.