The purpose of this project is to model the interaction between customers and postal workers in a post office environment. In this simulation, customers perform tasks such as buying stamps, mailing letters, and mailing packages. During this, postal workers are responsible for serving customers and using the scales for specific tasks. I coded this using Java and made use of threads, semaphores, and concurrency control. I intended to program project1 in java as well, but I found the moving of objects between classes using multi-processing problematic. This project being multithreading with semaphores has made this much easier.

During the development, the main difficulty was managing concurrent access to shared resources, such as the scales and the queues for customers and postal workers. Semaphores were employed to ensure that only a limited number of customers could enter the post office at once, and that only one postal worker could access the scales at a time. Additionally, concurrent data structures such as ConcurrentLinkedQueue and ConcurrentHashMap were used to manage individual queues and tasks between customers and postal workers initially. I further iterated on this later simplifying it.

Initially I also ran into issues with thread synchronization and communication. After getting a better understanding of semaphores I fixed this. Synchronization was necessary to ensure that the threads correctly interacted with shared resources and each other. Not being able to use the ‘synchronized’ keyword caused me a fair bit of trouble to find a way around.

Managing thread initializations with semaphores taught me a lot about concurrent access to shared resources and synchronization of multiple threads. Additionally, the use of concurrent data structures highlighted the importance of designing systems to handle concurrent access and avoid potential pitfalls such as race conditions and deadlocks.

After getting the program to run decently efficiently, it is clear the chosen structure in modeling the post office environment with semaphores is one of the more practical approaches. Semaphores allowed the program to enforce constraints on the number of customers in the post office and the number of postal workers using the scales. This ensured that these resources were not overutilized. Furthermore, the use of concurrent data structures and synchronization techniques guaranteed the program could accurately represent the interactions between customers and postal workers.

In conclusion, the post office simulation provided valuable learning experience in developing concurrent programs and managing shared resources. The use of threads, semaphores, and concurrent data structures enabled the program to accurately model the interactions between customers and postal workers. Despite the challenges encountered during initial development, I am reasonably happy with how my program executes the simulation.