Christy Jacob
CS 4348 Section 001
Professor Greg Ozbirn

The purpose of this project was to create a post office simulation to help display our knowledge of semaphores and threads. We had to create a post office simulation that included customer and employee threads. There were 3 employees or postal workers and 50 customers. Each customer visited the post office and was served by one of the 3 postal workers. Only 10 customers were allowed in the post office at a time and we had to keep track of which postal worker was working with which customer. The tasks for each customer were randomly assigned and one of the tasks required using scales which there was only one of. We had to use mutual exclusion to make sure data wasn't being corrupted. Mutual exclusion and coordination were achieved using semaphores.

The project was implemented using 3 Java classes. One class was called "Project2.java" and it had the main method. Another class was called "Customer.java" which contained the code for the customers' threads. The last class was called "PostalWorker.java" which contained the code for the postal workers' threads. I was able to learn a lot from the java threads example that was given. In the "Sem1.java" example I learned how to create, release, and acquire semaphores. I also learned how to make threads sleep, start, and join. That is also where I learned that you have to surround many functions with a try/catch. If I hadn't looked at that example, I probably would've been stuck on figuring out what an Interrupted Exception was. I looked at the other two examples but they weren't as helpful and had a lot of the same information as the "Sem1.java" example. I started in "Project2.java" by creating 50 customers and their corresponding threads along with 3 postal workers and their corresponding threads. Then I printed the starting message and started all the customer and postal worker threads. I then tried to join all the customer threads since they would finish executing. However, my postal worker threads were in an infinite loop so I didn't try to join them, but instead, I exited the program after all the customer threads joined. I then moved on to working on "Customer.java" since the logic for that was easier than the postal worker file's logic. After finishing "Customer.java", I still didn't completely understand how to integrate it with the postal worker file but after starting to work on "PostalWorker.java", I was able to figure it out. I used mutexes to protect enqueuing and dequeuing from the customer queue and to protect the scales. I used semaphores for the mutexes and to coordinate as was shown in the barbershop example. After finishing the postal worker file, I went back and made the necessary adjustments to the customer file and made sure the code was giving the proper output.

For my personal experience, I didn't have to do as much debugging on this project as I had to do for project 1 so I enjoyed it a whole lot more. This project was easier than project 1 because of its similarities to the barbershop example. The barbershop example was explained very well in class so I was able to figure out the

Christy Jacob
CS 4348 Section 001
Professor Greg Ozbirn

logic for the post office example faster. The hard part was understanding how to implement the semaphores and threads, but the java threads example was very helpful in that regard. I feel like I learned a lot about semaphores and threads, two subjects I was not strong in coding wise. My results seemed pretty accurate although they were not the exact same since they're not supposed to be. I also learned that planning ahead can make life easier when it comes to coding. Usually when I'm stuck, it's because I'm piecing together the logic for the code on the fly. This time, however, I had planned beforehand and had the barbershop example as a guide, so when I was stuck, I was able to piece it together faster. Overall, this project was less of a headache and more enjoyable than project 1 while at the same time, being a good learning experience.