Summary

In this summary of bank simulation project, we will discuss three main parts. The implementation of the classes, its functions and role, what was learned and results. The project code is split within five different classes. Firstly the Project2.java that is in charge of the running the project, creating different threads for tellers, customers and loner and finally displaying the overall results (balance and requested loan of customer). Bank.java serves the purpose of shared resources and hold all the objects/variables/semaphores that could be used in the project. This includes the semaphores, integer counters, and object variables of loaners, customers and tellers. In addition, it also lists other constants and lists, like the task time constants as well as queue that will serve the loan customers and teller customers. Loaner.java handles all the loaner related tasks, including the actual list of process it goes through to handle the client, as well as other helper functions. The logical idea behind this class is that, it waits for a client to trigger a ProcessLoan signal at which Loaner will start processing the loan and signal the Reciept back to the client, then it waits for client to leave and signals itself as available for the next client. Teller.java is similar to Loaner.java, except since there could be two total tellers instead of one, there has to be further validation to make sure that no customer overlaps each other, or no customers go to the teller when both are busy. Its logical workflow is as follows: the teller waits for the customer to trigger the Ready\_for\_teller signal (which can be maximum of two), when it receives the request then it starts preparing for the customer. During the preparation, the teller identifies the customer needs such as what it is they are wanting to do (deposit or withdrawal) and for how much. Once all the information is gathered from the customer, it then lets customer know that is ready to start serving the client, when client sends back process signal, teller starts processing and signals the receipt back to the client. When the client leaves the teller, teller makes himself available again for the next client. Bank.Teller is a counting semaphore with the maximum value of two, therefore it gives us a guarantee that there will always be at most two customers served at the same time. Customer.java is the class that is in charge of the trigger all the aspects of the project. The workflow of this class is as follows: it first randomly chooses a transaction from deposit, withdrawal and loan. If the choice was a deposit, it first enrolls itself in the queue of the teller, and signals the teller to let them know that they are ready to be serviced. When the signal comes back as that teller is ready for a customer it then sends the request of deposit and signals the teller to start processing. When teller sends back a signal that the transaction was serviced and receives the receipt, the client gets the receipt and leaves. Process is the same with the withdrawal except it requests for the withdrawal instead of the deposit. When customer is chosen to make a loan request, first it signals the loaner to see if there is any available. Once the signal comes back it then sends the request for the loan and signals the loaner to start processing. Then waits for the receipt. When the receipt comes back, customer gets the receipts and leaves the loaner.

I have learned a lot from this project as when I started I didn’t not fully understand how semaphores worked and how threads were able to wait and signal on a task. I also learned that mutual exclusion can be achieved in many different ways. For instance, in my program I was able to achieve mutual exclusion in a way that didn’t require a variable for mutex teller one and two. Rather I kept the variables stored locally of customer class, which automatically implied the mutual exclusiveness as no two threads have access to each other to update the local variables. Some of the variables stored locally were: loan amount, total balance, customer number and others. This way when the program got to the point to display the results all it had to do was go to the object of each customer and retrieve the data.