Summary

In this task we had to use threads to simulate a bank activity.

Bank has 2 tellers, 1 loan officer and 5 customers. To represent all of them, I use one thread for each actor.

First of all I wrote a pseudocode to represent all activities for each class.

I started with the Customer class. When customer enters the bank, he would do next steps: choose what he want to do (for example, proceed withdrawal or a deposit), then get into the appropriate line, wait and tell the employee what he wants to do. After the employee completed the task, the customer would get receipt and leave the bank until he need to come to the bank again.

The teller’s and officer’s class are very similar. Employee must wait until the next customer in the queue approaches him, then go to the next client's service, get the information he needs to complete the task. In doing so, the employee must inform the client that he is fulfilling his task.

Program contains classes Customer, Teller and LoanOfficer implement the Runnable interface. The most difficult part of this task was to determine when and what resources to block using semaphores and when to release them.

After I finished the pseudocode for each of this classes, the task was much easier. All I needed to do in the Main class was create all customers and employees threads, join it and get the summary in the end.

In doing this task, I realized that it would be much easier to do it using the keyword "synchronized" and blocking queues. But in the process of working with semaphores, I better understand how protecting shared data works when multiple threads work with them.