**ATM Assignment Report**

Team 4:  
\* Ahmed Youssef (2507690)  
\* Ramin Hashemi (2508573)  
\* Anas Saad (2510059)

Word Count: 700

**DESIGN**

We started the assignment off by designing the interface for the main bank form, and we decided that we would include an option for number of ATMs, and an option to switch the race condition setting. For the actual ATM form, which changed a lot during the writing of the code as we discovered necessary changes that had to be made to fit the requirements of the assignment. We wanted the ATM form to feel/look like an actual ATM, so we added a small screen with 6 side buttons, along with a number pad, a cancel button which takes the user’s card out, a clear button which erases current text being entered, and an enter button which submits current information being entered.

The first thing we wrote in the code was an Account class, derived from the sample code provided, to hold account information and account related methods. Quickly after that we started implementing the code for the main form, letting the user choose the number of ATMs to run, and whether the race condition setting is turned on or off. When pressing the Start ATMs button, every new ATM would launch on it’s own thread, with the state race condition setting being passed onto the constructor of every instance of the ATM form.

Maintaining or knowing which stage/phase the user is at whilst using the ATM is crucial for the side buttons and numpad functionality, as the buttons behave differently depending on the phase the user is currently in. Therefore, we added the setPhase function that changes the current phase to a desired phase, and does whatever else is necessary in terms of hiding / showing elements to fit the desired phase’s UI.

We also added the ability to view logs from the main form, by clicking on the “View Logs”. Every action done by the user is recorded and saved into a logs.txt file that is used to store all the logs of a single session.

One last thing to add is that we not only implemented the simulation of a race condition in withdrawals, but also deposits.

**PROBLEMS / SOLUTIONS**

The first problem we encountered was whether to implement the race condition fix in the deposit and withdraw implementations that are in the Account class, or the ones in the main form. We decided to implement them in the main form’s deposit and withdraw methods as it made it much easier to change later on without having to alter and go back to the Account class every so often.

The only other big problem that we’ve had is surprisingly not the race condition fix, which was just an implementation of a semaphore, that we chose to do in a separate function than the main ones, resulting in one function for race condition implementation, and one function for no race condition implementation, for each depositing and withdrawing. The main problem with this was how to actually simulate the race condition itself. Even when using an artificial delay (thread sleep) for 3-5 seconds for example, the balance would still be correct.

After a long time of trying different fixes and different approaches, we successfully simulated a race condition by making the withdraw/deposit functions take in a specified amount passed in as a parameter and store the balance BEFORE the withdrawal/deposit. This way, the function sets the active accounts balanced to (stored balance + specified parameter amount), thus negating any changes to the balance that might have happened during the thread's delay, and successfully simulating two SIMULTANEOUS withdrawals since it's impossible to do two simultaneous withdrawals at the same time using a user interface.

As an example, if £10 is withdrawn on two ATMs, both threads sleep for 10 seconds. If thread 1 changes the balance first, then balance changes to £290, so when thread 2 comes to change it, it should change it to £280, but the withdrawal is based on the temporary balance that was stored BEFORE the thread sleep so it would withdraw £10 from £300 and not £290, thus demonstrating the inconsistency in the balance as a result of a data race