**ATM Simulator Report**

**Features**

My ATM Simulator has a class that simulates the central bank computer. This stores an array of the customer accounts and an array of several ATMs through which the customers can access their accounts.

On the ATM, a customer is prompted to enter their account number and pin number. These are verified and they are taken to the main ATM menu. This allows them to withdraw cash, view their balance or quit the ATM. Each time they press a button on the ATM screen the options change and after each transaction completes (successfully or unsuccessfully), they are given the option to either return to the main menu to carry out another transaction or quit the ATM.

**Approach to the problem**

As we had been provided with the main functionality, my first step was to create the ATM GUI in a Windows Form. This was fairly easy to do in Visual Studio as it is very easy to arrange and customise controls. I then changed the sample code to be run from the event handlers of the controls rather than in an infinite loop in the console window. Once I had got one ATM functioning, I used the code from the Horse Race lab to implement multi-threading and run two instances of the ATM concurrently. As the code did not address the data racing problems, I then set about restricting access to the critical parts of code using flags that each thread had to check before changing shared resources (the account balance).

**Difficulties encountered**

When I had created the basic ATM GUI, I realised that as the user navigates through the ATM’s menus, the buttons would need to complete different actions depending on the current context. Rather than create a new button to handle each unique event (and show/hide them on every menu change), I used a state machine approach whereby a string variable in the ATM tracks which screen the user is currently in. The event handler for each of the 6 screen buttons then checks which screen is displayed and completes the correct action. Although this resulted in slightly convoluted code and large IF statements, it saved a lot of repetition showing/hiding controls and having lots of buttons on the form.

Initially, I had some difficulty understanding how to multi-thread. I was unsure how to code a method to create new, separate instances of the ATM form that could access the Account objects by reference so that any changes to account balances were propagated across all instances of the ATM forms. By modifying the Horse Race code, I was able to achieve this.

The last problem I faced was how to prevent race conditions. I searched the Microsoft documentation about threads and thread locking and found several functions (such as the ‘Lock’ keyword) that could restrict certain sections of code to being accessed by only one thread at a time. However, I was unsure how to implement this in relation to this assignment and instead opted for a cruder solution where the Account class has a ‘locked’ Boolean field. Before the critical code (withdrawing cash) was executed, it would change this Boolean to ‘lock’ the account and the account would only be ‘unlocked’ after a transaction is completed. This meant that only one thread could execute the critical code at once.

**Word Count: 553**