**Course Work Cover Sheet - The School of Computing**

Author’s Name: Kyle Harrison

Matriculation

Numbers: 110009870

Module Code: Ac22005

Assignment Title: Computer Systems 2B – Coursework 3

Date Submitted: 29/03/12

Date Due to be 29/02/12

Submitted:

Word Count 499

|  |  |
| --- | --- |
| Your course work may be used by The School of Computing for demonstration purposes with other classes and/or members of the public, unless you indicate otherwise. Tick the box at the right if you do not wish your work to be used in this way. |  |

I certify that this assignment represents all my own work and that no parts of it have been copied and that no collusion has taken place with any other person. I have also read and understood The School of Computing regulations as regards course work and I certify that this course work was performed according to those regulations.

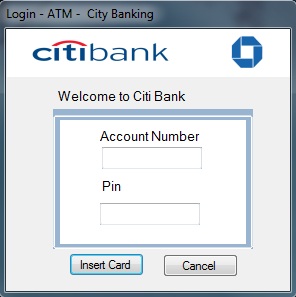
I certify that any disks submitted with this assignment have been virus checked and have no viruses on them.

Signed: ………….…..<Name>

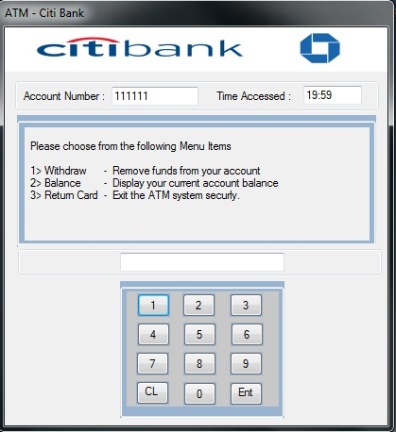
**ATM Simulator using threads in C#**



My initial approach was to familiarise myself with the sample C# code and modify this to complete each of the tasks specified in the brief to create a threaded ATM application. However this proved to be troublesome as I was unable to implement multiple consoles in C# so instead I produced a multiple-window GUI display based on what features seemed appropriate for an ATM and from the assignment brief.   
  
The solution created uses three forms (central bank, login and menu). The central bank form allows unlimited new ATM login forms to be created in separate threads (and by extension ATM menu forms through the login form) allowing access to an individual account when the user enters valid account credentials.



Upon creation of a new ATM the login form which will accept an array of accounts as a parameter and check the user’s login credentials against this account array to determine if they are valid. If the details are correct it will pass only that specific account to the menu form where the user can access the account to withdraw funds and display their balance. Upon exit they are return to the login form where another user can use the ATM to access their bank account.

Creating threads was somewhat problematic as it wasn’t quite clear from the thread based tutorial how the code of threads was structured however after viewing an online tutorial this aided me in understanding how to best use threads in my ATM effectively and helped resolve issues with cross threading that was causing issues in the ATM system.

As there is no artificial limit to the amount of ATM’s that can be generated in the program (provided the number of ATMS do not exceed the hardware limitations of the system) multiple ATMs can be used at once with each being able to share access to the same account and make synchronised withdraws to prevent a data race.

However the solution implemented for stepping through the code and preventing a data race will cause a 3 second delay in-between each transaction so the user can follow the operation manually causing an artificial limitation on the system by forcing the thread to sleep and slowing down the system. Although preventing a data race would normally slow the system down by a small amount for each thread running when synchronised this should not be noticeable by the user unless there is vast amounts of threads competing for access to the same resource. Therefore in this simulator the delay between each ATM completing it code is subject to this sleep call to the thread to demonstrate the thread process to the user.

Inevitably mimicking an ATM procedural system in an event driven environment was a complication that derived from following the diagram flow in the brief. To produce this I created a label which is updated with text when Boolean conditions representing events are true which then displays the user menu when the user interacts with the virtual keypad.