ReadMe

For this assignment, we started by taking the code uploaded to sakai for the start of the client and server. We put each of the functions provided in infinite loops to create a connection that was perpetually renewing until the "exit" command is entered at which point the client thread disconnects and leaves the server thread continuously 'spinning'. The client attempts to connect to the server every two seconds and prints the command "Attempting to connect to the server...". Once the server and client are connected, the user is prompted to press enter to proceed. The user is then prompted to either "open", "start" or "exit". The open command first checks to see if another count is in the process of being opened and if the accounts are currently printing. If either of those conditions are true, the client is forced to wait for a couple of seconds until either of the other functions are complete. Otherwise, the program enters the openAccount function where it creates a new account object with the fields accountName [100], float balance and int inUse. openAccount then attributes the correct name to the account, initializes the balance to zero and sets the inUse flag to zero (indicating that the account is not in use). openAccount then returns to the connection_handler (where openAccount was called from). Once back in connection_handler, the user is prompted again to enter "open" or "start". If the user enters the command "start" followed by the account name, the function startSession is called. Start session searches the array of accounts and if it does not find the account, it returns a -5 which is interpreted as "account not found" by the connection_handler function. If the account is currently in use (indicated by the inUse flag that each account has), startSession returns a -1. This is interpreted as "account in use" and tells the client that this is the case. If startSession finds the account, it returns the index at which the account was found. At this point, connection_handler calls "getNextCommand". getNextCommand prompts the user to enter either "credit", "debit" (followed by amount) or "balance" or "finish". If credit is chosen, the account is credited by adding money to the account, if debited, the account is debited. If the amount debited is greater than the current balance, the account is not debited and the user is notified of such. If the user enters "balance" the balance is printed out. If the user enters finish, getNextCommand returns to the connection_handler thread to prompt the user for the next account they would like to create or access. If the user enters "exit" when back in the connection_handler, the client thread is closed and the server continues to run waiting for access from future clients.

As soon as the program starts, a thread is started and detatched into the method "printAccounts" (I am not sure whether this is proper terminology but the thread is spun and put into printAccounts where it pleasantly carries on the rest of its thready life). Print accounts uses the sleep function to make sure the thread is active only once every twenty seconds. While the accounts are printing, a global Boolean flag (we used an int for this) is switched to 1 when the accounts are printing so other accounts cannot be opened during the printing. As soon as printing is finished, the Boolean is set to 0 and the thread goes to sleep for another 20 seconds. This continues as long as the server is turned on.