**PROJECT NAME: BR\_N BANK**

**NANA BOATENG AMOAH**

**PROFESSOR : KENT JONES**

**COURSE : CS-172-1**

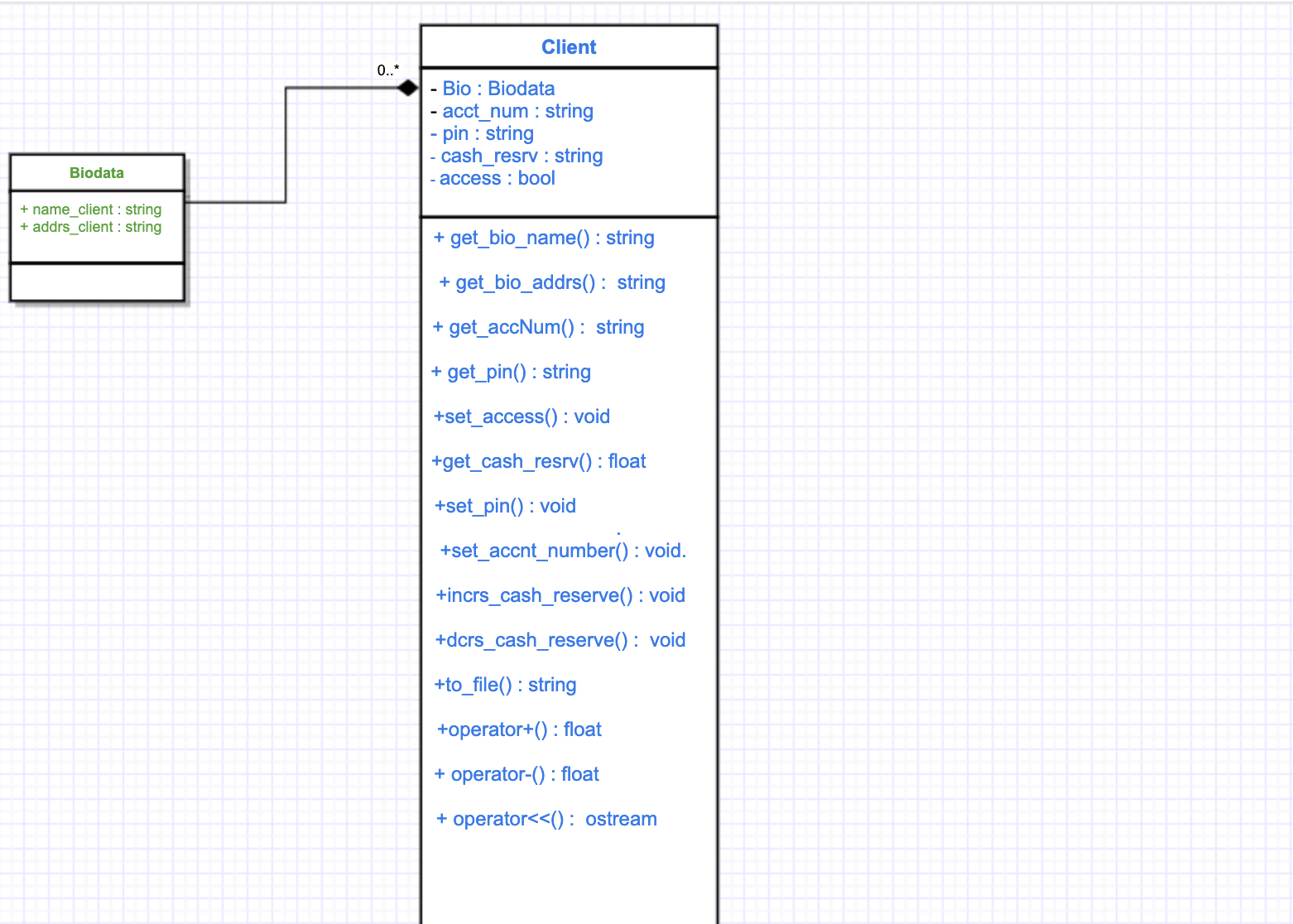
**Purpose of the Project**

The project attempts to simulate a micro banking system. The name of the Bank is called BR\_N bank. The project’s features are basically two. The first feature is, it allows a user to create an account in the bank. The program then takes in the user’s data and subsequently generates further data (an account number) for the user in accessing her/his account. The second feature of the program is that it allows a user to login with her/his account details to perform nominal bank transactions like withdraw, deposit and

transfer.

**Specific detail of the Project**

The program utilizes OOP to abstract the User’s (Customer) interaction with the bank. The class that defines the customer of the bank is called **“Client”.** Client contains members and behaviors that simulates an average bank. Client is also composed of a class called “**Biodata**” that stores a customer Biographical information. The image of the UML diagram demonstrates the members and behaviors of the bank.



The program has three subprograms mainly, Createacct, Login, and Transactions. The three subprograms are directly accessed by a file called ‘**Home.h’**. The subprograms are all also defined in their respective header file to aid in abstraction and testing. Also, all the subprograms have their respective display files to aid in output and note there are some display functions that are shared amongst the subprograms like “**clear\_disp ()”.**

The first subprogram, **Createacct**, is a program that assists the user into creating a new account. It gets initialized when called from **Home.h**. The Createacct program guides the user to collect the specific data required for setting up an account. It contains essential error checking functions for bad user input. The function that takes in user input (**input\_gettor ())** is templatized to take in input of any data type. After the program has taken the necessary user input, it pushes all the data to a vector of type string. Then a temporary client object is allocated with the respective data from the vector as parameters. Then the client data is written to the file “**Vault.dat**” in a standard with help from the client class method “**to\_file ()”.** Then after the data is sent to file, the Createacct program exits back to home signaling that an account has been created.

The second subprogram is **Login**. It deals with the processes that allows the user to login into an account. The program initializes from **Home.h** also. When called, it guides the user to collect these specific inputs: **Account number and PIN.** The account number and PIN are stored in the file “**Vault.dat**”, when an account was created. Login then takes in the program and utilizes functions and efficient displays to take in user’s PIN and Account Number. After the two data are taken and stored in memory, Login opens **Vault.dat** then gets a line of data. Then after the line of data is taken, it is sent to a function called **line\_septr ()**. Line\_septr () separates the line of characters into its specific data using the program’s standard separator (**,**) as reference. Then after all the meaningful data is separated and taken, it is pushed to a vector. The Account Number and PIN from Vault.dat are then compared with the users input values taken earlier in Login. If they are equal, Login proceeds to Create a Client object from the data else it loops till the data is found (or exits when data is not found at all in file). When the client object is created, it is passed to **mother\_func ()** in Transactions.h to perform the necessary transactions requested by the user.

The third sub program is **Transaction.** Transaction deals with basic transactions the customer might need. Transaction takes in a client object from Login using the function **mother\_func ().** The client object is stored in a stack to encapsulate and prevent issues. When the client object is transferred by the stack to mother\_fucn () the client can interact with the bank’s services like withdrawing cash, depositing and transferring cash. The interactions of the Client object lead to update of the cash of the client of the object in memory. After the necessary transactions are made, the specific client’s account data in Vault.dat needs to be updated. So, to do that the program utilizes a function called **over\_ridefile ()** which takes in a stack of type client as parameter. Then over\_ridefile () loads all the data in **Vault.dat** into a vector of type string. The file is then deleted of its contents. The data is then rewritten to Vault.dat, and if data in an index of the vector is equal to client object’s data, then that Client object in memory sends its data to Vault to replace that particular index data of the vector. After, all the nonequal data gets written to vault in their respective manner until none is left in the vector. Then the program exits to home.

**Assumptions made**

I made a lot of assumptions about this project.

My first assumption was that how I displayed the information to console affected how a user interacted with the program. And hence, in order to prevent that, I included a clear screen function that clears previous non-essential output from the screen so the user’s eyes could gaze on what was required or really important at runtime and thereby increasing her/his engagement with the program. Also, I ensured that my output was graphically intuitive (in command line context), hence I created shapes, and nice formatting with strings to make the program feel like an ATM machine (at least) so it does not overwhelm the user during execution.

My second assumption was that, since the program required a lot of user input, and a wrong input could cause the program to fail, I developed specialized functions that takes in user input then the specialized functions verify that the input is correct, before it was released to other parts of the program during runtime.

Another assumption was with regards to file writing. Since, each line of characters in “Vault.dat” (the file that stores the banks data) was utilized to create a client object, I developed an overridden function that overwrites a specific file line with new changes that happened during runtime for that specific account.

Also, in relation to my file handling, I sent each user’s member data into the program using a class method. The class method has a standard separator, that separates specific content like name and address from each other so that the program could detect the type of data easily when creating the Client object from it during runtime.

Sources

https://www.tutorialspoint.com/cplusplus/cpp\_date\_time.htm