CMP2801M Advanced Programming

Assessment 1

Contents

1 Application Design - Implemented Functionality

Upon running this application, the user is greeted with a list of commands they can perform. They are able to create only one current account, and as many savings and ISA accounts as they please, and can put an initial deposit into these accounts too; ISA accounts however require an initial deposit of at least $1000. I had to change currency from Pound Sterling to Dollars as the terminal would output an ascii variable, although no functionality has been altered. The user, after creating an account, is able to deposit and withdraw, as well as transfer between accounts as long as two accounts have been created and there is a sufficient balance. When ‘view’ is inputted as a command, all created accounts can be viewed, which includes their balance, account ID, overdraft (for current accounts), interest rate (for savings and ISA accounts), and all transction history. As savings accounts and ISA accounts have an interest rate, using the ‘project’ command allows the user to see their chosen account’s compound interest for a specified number of years (the interest calculated on the initial amount and the accumulated interest from the previous months). Finally, the transaction history of every account can be searched for either by its value (for example, $100) or by the transaction’s type (whether it was a deposit or a withdrawal). All available commands can be seen again by entering ‘options’, and to end the program, the user can input ‘exit’ at any time during the application’s run.

I created an abstract class ‘Account’ which would act as a parent class for the child classes - ‘Current’ and ‘Savings - to override its functions. the ‘Savings’ class also implements the function from the ‘InterestEarning’ interface to allow the compound interest to be calculated. These classes were defined in order and then separated into header files to allow for the hierarchy to be maintained in inheritance. Additional values were created for the ‘Current’ and ‘Savings’ classes, such as ‘accountID’ which stores the ID of the account, and constructors, so important variables were defined from the creation of an instance of the class. Accessors and mutators were implemented too so the private variables are able to be returned in the main function, as these values were made private, therefore being not accessible if called directly. Files such as the input and output stream were also implemented, as well the vector file. This is taken from the ‘Standard Template Library’ which allowed data to be stored in a dynamic array, so elements could be added or removed during run-time. All pure virtual functions from the parent classes were overridden in the child classes too as they share the same function name, return type and parameters, but perform in different ways.

I have implemented a linear search to tackle the search task, so the program stores every transaction of every open account and stores them in a separate vector. Based on what criteria was specified by the user, the program looks through every transaction and matches either the amount of money or the transaction type, and prints the transactions that fit the criteria.

2 Application Testing and Evaluation

Briefly describe how you tested your program. For each command (where applicable) demonstrate some test cases in a tabular form and indicate whether the tests were passed or not. The table below represents a small subset of possible test cases – yours should be more comprehensive.

Test Case	Input Values	Expected output	Passed?
User opens a current account	1	**$0 have been successfully deposited into your account with ID: 1**	YES
User opens a current account	1 10	**$10 have been successfully deposited into your account with ID: 1**	YES
user opens current account	1 -20	Error message: **Your input was incorrect.** **If you would like to see all available options, type 'options'.**	YES
Open savings	2	**$0 have been successfully deposited into your account with ID: 2**	YES
Open Savings	2 60	**$60 have been successfully deposited into your account with ID: 3**	YES
Open ISA	3 2000	**$2000 have been successfully deposited into your account with ID: 4**	YES
Deposit into Current	200	**$200 have been successfully deposited into your account with ID: 1**	YES
Withdraw form ISA	200	**$200 have been successfully withdrawn from your account with ID: 3**	YES
Transfer between Current and Savings	1 2 20	**$20 have been successfully withdrawn from your account with ID: 1** **$20 have been successfully deposited into your account with ID: 2**	YES
Transfer between Savings and Current	2 1 20	**$20 have been successfully withdrawn from your account with ID: 2** **$20 have been successfully deposited into your account with ID: 1**	YES
Project Savings	5	**The projected earnings for this account at 0.850000% interest rate.** **Projected Balance: £61.782764**	NO
search by $200	search, value, 200	**-- Deposit of $200 at Thu Jan 6 11:59:38 2022** **-- Withdraw of $200 at Thu Jan 6 11:59:38 2022** **-- Withdraw of $200 at Thu Jan 6 11:59:38 2022** **>>> Your input was incorrect.** **If you would like to see all available options, type 'options'.**	NO

My linear search is short, concise and successfully records all transactions that match with the desired criteria. However, with a time complexity of O(N), this is not the most efficient algorithm if it were to be scaled up to a more realistic scale, but is suitable for this small scale application. Memory management could have been developed more as no destructors were used to delete pthe large objects being stored and worked with, and functions could have passed by reference rather than by value to prevent the use of duplication of variables when used as function arguments.