**Student:**

**Muhammad Hamza Mukry**

**Assignment 3: Object-Oriented Design & Programming**

**Date: Nov 22nd 2024**

Table of Contents

[Program Code 3](#_Toc180938863)

[GIT Output 8](#_Toc180938864)

[Program Logic / Flow 162](#_Toc180938865)

# Program Code

The program have the following modules:

o **application.py**: Implements the interactivity with the user (inputs/outputs). Call the various methods of the other modules based on user input. Also published the sample stock report in the requested format.

Created following methods:

* ShowMainMenu(): This method loops to display options like Open Account, Select Account and Exit.
* ShowAccountMenu(): This method loops to display options like check balance, deposit, withdraw and exit account until the user chooses to exit the Account Menu.
* Run(): This is used to execute the program
* get\_user\_input\_float: This function makes sure that if input integer is greater than minimum value or less than maximum value then you return the input integer or if this is not the thing then print incorrect value

o **Account.py**: Represents a bank account. It is the base class for classes SavingsAccount and ChequingAccount.

created following methods:

* deposit: method that takes the amount to deposit and adds it to the current balance.
* print\_Account: method to print all the attributes of the account class.
* get\_account\_type: allows the user to search an account type
* set\_account\_type: allows the user to set account type
* get\_account\_holder\_name: allows the user to search an account holder name
* set\_account\_holder\_name: allows the user to set account holder name
* get\_account\_number: The application can search for accounts based on account numbers
* set\_account\_number: allows the user to set an account number
* get\_rate\_of\_interest: allows the user to search an account number
* set\_rate\_of\_interest: allows the user to set an account number
* get\_current\_balance: allows the user to search a current balance
* set\_current\_balance: allows the user to set a current balance

o **Bank.py**: Implements the business logic required for the banking. Keeps track of all the accounts. Allows the user to open a new account or to search for an existing account.

created following methods:

* Open\_Account(): allows the user to open a new account
* Search\_Account():  this allows the user to enter the account number of the account they want to work with. Upon searching the account successfully, the application will call the method showAccountMenu to display the Account Menu as described next.
* **\_\_init\_\_** constructor to initializes the instance of the class Bank.
  + bank\_Name: Attribute of the bank class set at the time of initialization

o **SavingsAccount.py**: Child Class of Account Class to represent SavingsAccount.

created following methods:

* withdraw: method that takes the amount to withdraw and removes it from the current balance. It also checks that after the withdrawal amount the current balance remains more than the minimum balance.
* \_\_init\_\_: constructor to initialize the instance of the class SavingsAccount

o **ChequingAccount.py**: Child Class of Account Class to represent ChequingAccount.

created following methods:

* withdraw: method that takes the amount to withdraw and removes it from the current balance. It also checks that current balance added with overdraft limit is greater than the withdrawal amount.
* \_\_init\_\_: constructor to initialize the instance of the class ChequingAccount

|  |  |  |  |
| --- | --- | --- | --- |
| No: | Object | Datatype: User-defined Class | Datatype: Predefined class/type (python) |
| 1 | Account | Account | * deposit() * print\_Account() * get\_account\_type() * set\_account\_type() * get\_account\_holder\_name() * set\_account\_holder\_name() * get\_account\_number () * set\_account\_number () * get\_rate\_of\_interest() * set\_rate\_of\_interest() * get\_current\_balance() * set\_current\_balance() * \_**\_init\_\_** constructor to initialize the instance of the class Account |
| 2 | application | Not applicable | Not applicable  Implements the interactivity with the user (inputs/outputs). Call the various methods of the other modules based on user input. |
| 3 | Bank | Bank | * Open\_Account() * Search\_Account() * \_\_init\_\_ constructor to initializes the instance of the class Bank. |
| 4 | SavingsAccount | SavingsAccount | * withdraw() * \_\_init\_\_: constructor to initialize the instance of the class SavingsAccount |
| 5 | ChequingAccount | ChequingAccount | * withdraw() * \_\_init\_\_: constructor to initialize the instance of the class ChequingAccount |

**UML Class Diagram**

**A diagram of a computer

Description automatically generated**

**Assignment Report:**

1. First, I wrote the menu options for showMainMenu and showAccountMenu in Application class.
2. Then I implemented Bank and Account classes to implement logic for OpenAccount and SearchAccount methods.
3. Then I implemented child classes (such as SavingsAccount and ChequingAccount) of Account class.
4. Then I moved some of the program logic to application module and used the four classes and their relevant methods to implement the program logic.
5. Finally, my code was efficiently organized by using classes.
6. The implementation was tidy and easy to handle since distinct functionalities were kept in different classes. Code structure and variable naming best practices were emphasized. I have implemented basic logic to check user input based on attribute types. However, this could be enhanced further by adding better error messages and retry logic.
7. To ensure the simulation works well in a variety of circumstances, load testing should be done using much larger data combinations.

# GIT Output

# Program Logic / Flow

**Program input message**

**Program output**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated