

Bank Management System

FINAL PROJECT OF INTERMEDIATE C++
NGUYỄN ĐÌNH

I. Introduction

Object-Oriented Programming (OOP) is a programming paradigm that employs the utilization of objects within the programming process, as implied by its name. The fundamental objective of object-oriented programming (OOP) is to facilitate the representation and manipulation of real-world entities within the context of computer programming. This is achieved through the utilization of various mechanisms such as inheritance, encapsulation, polymorphism, and more. These mechanisms enable the creation of software systems that closely resemble the structure and behavior of real-world objects, thereby enhancing the modularity, reusability, and maintainability of the resulting codebase.

The primary objective of OOP is to establish a cohesive relationship between data and the corresponding functions that manipulate them, thereby restricting access to this data exclusively to the associated function within the code. Therefore, OOP is an extremely essential concept in the field of computer science, especially software engineering due to its allowance for programmers to process and design a system in terms of objects that have both behaviors (methods) and attributes (variables). Such way of implementation will enhance the intuitiveness, comprehensibility, and precision of the software development for a long run.

This report aims to present a bank management system project, which serves as a platform for acquiring a comprehensive comprehension of object-oriented concepts. Given the ubiquity of bank systems in my everyday activities, I perceived an opportunity to undertake the replication of such an application with relative ease compared to other types of applications. This endeavor allowed me to effectively apply object-oriented concepts within my programming framework.

The bank management system under consideration has been specifically developed to cater to the needs of bank tellers. These individuals are responsible for facilitating various financial transactions, such as withdrawals and deposits, as well as assisting customers in opening checking and savings accounts. Furthermore, the system equips tellers with the necessary tools to furnish customers with relevant information pertaining to their accounts. Hence, it can be inferred that the bank system in question demonstrates a commendable capability to effectively handle the tasks.

II. Background

To provide a comprehensive understanding of my project, it is imperative to first delve into the fundamental principles of Object-Oriented Programming (OOP), as they serve as the foundational framework for the development of said project. There are four main concepts of Object-Oriented Programming (OOP) that programmers should be proficient at.

1. **Data Abstraction:**

Data Abstraction is widely recognized as a fundamental and indispensable characteristic within the realm of object-oriented programming. The concept of data abstraction pertains to the act of selectively revealing pertinent information regarding the data to external entities, while concealing the underlying intricacies or implementation specifics.

2. **Encapsulation:**

Encapsulation is enclosing data in a single unit. The mechanism is how code interacts with its data. Encapsulation is a key notion in object-oriented programming (OOP) that hides class variables and data from other classes. These variables can only be accessed and altered by member functions in the same class as their declaration. Encapsulating the data protects data integrity and provides abstraction by restricting direct access to its underlying state. Encapsulation hides a class's internal data from external classes in object-oriented programming.

3. **Inheritance:**

Object-oriented programming relies on inheritance. Inheritance allows a class to inherit attributes and traits from another. We inherit attributes from other classes while writing classes. We don't need to write all the properties and methods over when creating a class because they may be inherited. Inheritance lets users reuse code and eliminate repetition.

4. **Polymorphism:**

The etymology of this concept can be traced back to the Greek language, where it is derived from the combination of the term's "poly" and "morphism." In the realm of programming, polymorphism refers to the inherent capability of an object or a method to exhibit multiple forms or behaviors. One prevalent application of polymorphism in programming involves the utilization of a reference to a parent class for the purpose of referencing an object of a child class.

Throughout the period of design and development, these concepts of OOP are the main foundation that I relied on to create my own Bank Management System.

III. Methods

Specification

In this session, I will detail every library I utilized for my project in this session, along with the rationale behind my selections:

- **iostream:** For C++ applications, this library offers the most basic input and output services.
- **vector:** I included such library to use a dynamic array (vector) to store the list of bank accounts of my Bank Management System
- **type_info:** I included this library to check whether the data type of user's input is right or wrong. (Exceptional Handling)
- **string:** this library provides me tools to work with strings data types in C++.
- **regex:** The C++ programming language offers a comprehensive regex library that includes a dedicated class designed to handle regular expressions. Regular expressions are a specialized form of pattern-matching language employed for searching and manipulating strings. I used to manage user's input and handle exceptional input, which help me prevent unexpected bugs and outcomes.
- **using namespace std:** it makes the compiler assume that I am using the std namespace.

In this project, I separated the classes into headers and source files so that I need to include bellowing headers in source files that need:

```
#include "Bank_Account.h"
#include "Savings_Account.h"
#include "Checking_Account.h"
#include "Bank_System.h"
#include "GUI.h"
```

Also, you need to compile and link all the sources files together to create an executable program (a.exe). Here are my instructions to run my program properly:

1. *Open a terminal in the directory containing your sources files.*
2. *Compile each source file into object file.*

```
Thank you for using our system... (Ctrl-C)  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c Bank_Account.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c Savings_Account.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c Checking_Account.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c Bank_System.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c GUI.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ -c Run.cpp  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> █
```

After that, there will be object files of those classes in your directory.

3. *Link all the object files together to create an executable.*

```
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> g++ Bank_Account.o Savings_Account.o Checking_Account.o Bank_System.o GUI.o Run.o  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> █
```

4. *Run the program.*

```
Thank you for using our system... (Ctrl-C)  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> ./a█
```

Design

My bank management system's source code contains five classes in which I implemented concepts of OOP to design their relationships as reasonably as possible to smooth the functionality of the bank management system. Each class will have a constructor to build instances of that class and will be in charge of a specific task which I will explain and analyze deeply in the following section. Below is the UML diagram for my programs, so that you can have a whole view of how my bank management system look like:

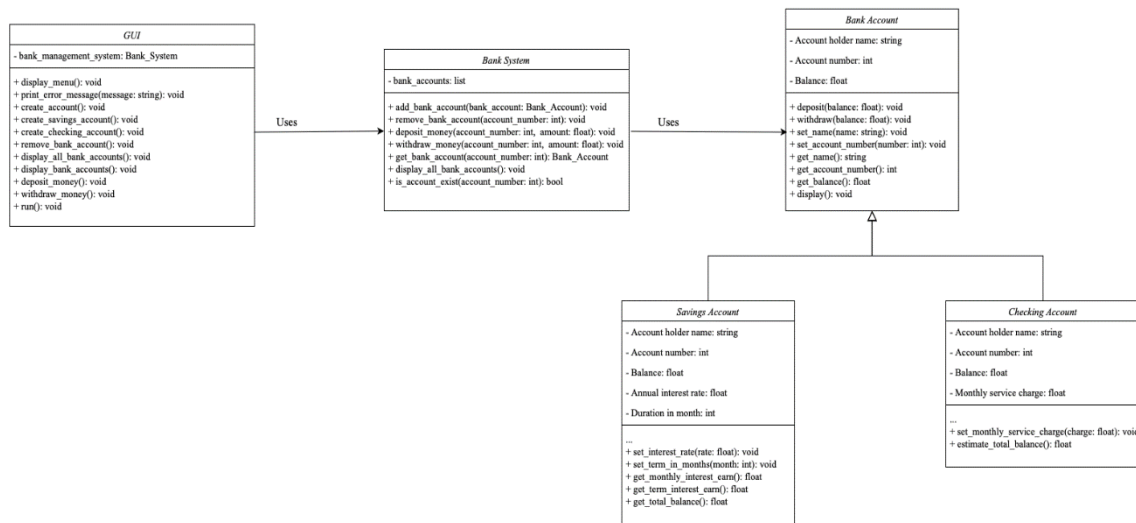


Figure 1: Given is an UML diagram illustrating all five classes and how they relate to each other in my bank management system.

First and foremost, when it comes to a bank management system, we should think about bank accounts first, which is the main object that we need to store and manage well. Since my bank management system has two types of accounts, I need to create an abstract class to represent basic functionality and attributes of a bank account, which in this case is Bank_Account class. This class will be a parent class of two other classes where stored more specific attributes and functions which will depend on what kind of accounts are. In Bank_Account class, I created three attributes (account holder's name, account number and the current of the account) and set it protected so that there will be only two classes that inherit Bank_Account can access and use such attributes. Other classes cannot directly access them, but their values through their setters and getters. Moreover, I also add three basic functionalities of a bank account, which are three virtual methods so that subclasses can apply their own specific functionality due to polymorphism of OOP. Through the management system, bank teller can help owner of the account only to withdraw and deposit their money, but also show detailed information of the accounts.

Secondly, I created a `Savings_Account` class that inherited from `Bank Account` class. This class represents a savings account, which is a type of bank account. Because a savings account must have an annual interest rate and its duration until account holder can earn interest, I add two more attributes in this class as private, so that other classes cannot change or directly access those attributes. Therefore, I also add setters and getters of those attributes so that other classes can work with their values. In this class, I override the display functionality from `Bank_Account` class so that my bank management system can display more specific in terms of what kind of bank account. I also added some functions to this class to calculate the monthly interest earn and balance after period of savings and display it to users through overridden display functionality. Since only saving accounts can have such usage, I set those function as private.

Same as `Savings_Account` class, `Checking_Account` class is derived from `Bank_Account` class. This class represents a checking account. In my bank management system, every checking account will have a monthly service charge to maintain the account, so I created an attribute for it as well as a constructor for it. However, in this specific case, I only create a setter for this attribute because I just use it in `Checking_Account` class in the display () functionality. In addition, I also wrote a method in this class to calculate the net balance after service charge too. In this class, `display()` is also overridden, so that my system can display detailed information about a checking account (for example: net balance after service charge).

The fourth class that I was going to create is `Bank_System` where I provided all the options that users can choose when using this program.

```
private:  
    vector<Bank_Account *> bank_accounts;
```

Figure 2: a vector type `Bank_Account` to store the list of bank accounts in my bank management system.

In fact, we cannot know how many bank accounts will be created in the future. However, in C++ programming language, we need to declare the size of the array when creating it, which is also synonymous with the fact that there will be a chance that there will be a larger number of bank accounts that my bank management system can store. Meanwhile, C++ programming provides us vectors which have the ability to resize itself automatically when an element is inserted or deleted, so that the system can work with bank accounts entity more dynamically. This is the reason why I choose to create a vector type `Bank_Account` and use it in the class `Bank_System`. From then, I started to add functionality of my bank management system using this vector. My bank management system can let users add or remove

a bank account, deposit and withdraw money from an account using the account number which is the unique attribute of each account. My programing can also let user search for a specific account through its account number or display all the bank account as well as number of them when users want to. Lastly, users can check whether an account exists in the system or not.

My final self-created class is GUI class, which is the user interface of the system to interact with the user. I believed that it is a good practice for a software developer to split his system into back-end and font end, which can let he or she to modify the user interface without for exceptional handling without effecting the backend of the system. Therefore, I applied such ideology into my bank management system to freely update and modify the user interface of my bank management system without damaging anything. I must create an object for each remanning classes, which are backend classes for my program, in GUI classes so that I can add user interface for each functionality based on classes. Therefore, users can easily use my bank management system through instructions that the program provides. This class will provide choices for users to interact with the program for example: When bank teller chooses to create an account option in the menu class, there will be three options prompted to the users which are creating a savings account, creating a checking account or back to the main menu.

```
void GUI::create_account()
{
    system("cls");
    cout << "-----" << endl
         << "Creating a new bank account..." << endl
         << "-----" << endl
         << "1. Create a savings account" << endl
         << "2. Create a checking account" << endl
         << "3. Back" << endl
         << "-----" << endl
         << "Enter your choice (1-3): ";

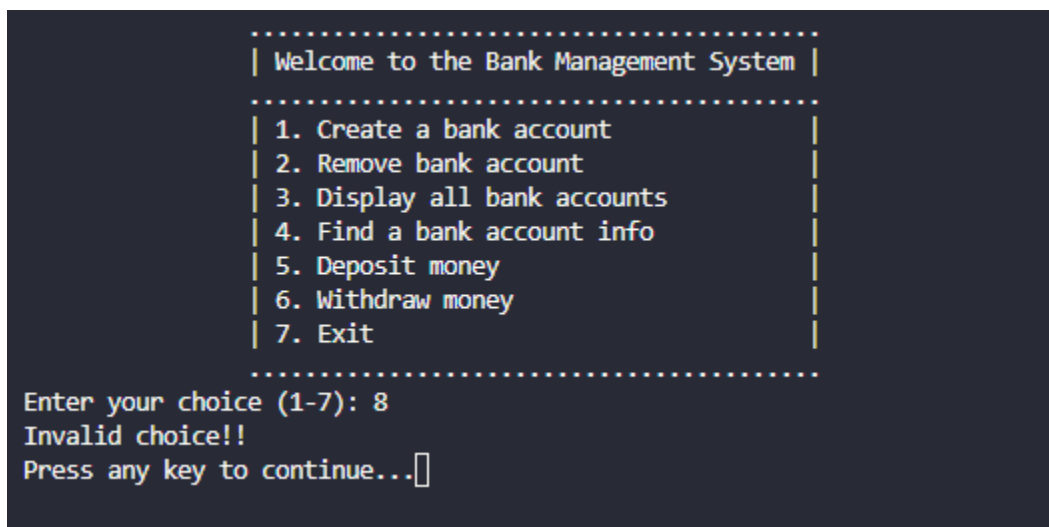
    int choice;
    cin >> choice;
    if (choice < 1 || choice > 3 || cin.fail())
    {
        print_error_message("Invalid choice!!");
        create_account();
        return;
    }
    else if (choice == 1)
    {
        create_savings_account();
    }
    else if (choice == 2)
    {
        create_checking_account();
    }
    else
    {
        cout << "Press any key to continue..." << endl;
        return;
    }
}
```

Figure 3: Function to ask users to tell program know which kind of account they want to create.

This class is also the place I perform exceptional handling as good as possible in order to prevent bugs when users accidentally input “evil values”, and such topic will be mentioned in next section of my report.

Exceptional Handling

In the session, I will mention several situations when users accidentally input wrongly and how I handle it in order to make my program run smoothly. For example, in the main menu of my program, the users can only input from number 1 to number 7, if users input something that are different from 1 – 7, the program will prompt this message to users:



```
.....  
| Welcome to the Bank Management System |  
.....  
| 1. Create a bank account  
| 2. Remove bank account  
| 3. Display all bank accounts  
| 4. Find a bank account info  
| 5. Deposit money  
| 6. Withdraw money  
| 7. Exit  
| .....  
Enter your choice (1-7): 8  
Invalid choice!!  
Press any key to continue...[]
```

Figure 4: Program’s response when users tried to enter invalid input.

After that, users are demanded to press any key to comeback to the main menu and input again until the program has the valid input. This method will be applied to other choices throughout the program. However, in my bank management system, the account holder name only contains alphabetical characters, thus users tried to add invalid characters in the name the program will ask users to press any key twice and try to input again until users input a valid name for the account.

Furthermore, in my bank management system, each bank account will have an unique account number, so when users try to create a new account with a used account number the program will prompt:

```
Creating a new savings account...
-----
Enter account holder name (must be alphabetic only): Hanh Nguoyne
Enter account number: 1
Account number already exists or invalid account number!!
Press any key to continue...[]
```

Figure 5: Program's response when users try to create a new account with a used account number.

The program required users to press any key to get back to this state from the beginning where user has to re-input from the name of the account holder, which will be the same for remaining information of an account (balance, annual interest rate, term in months). When users try to remove an account that does not exist in system, the system will prompt this message to users and then ask users to re-input the account number:

```
-----
Removing a bank account...
-----
1. Remove by account number
2. Back
-----
Enter your choice (1-2): 1
Enter account number: 45
Account number does not exist or invalid account number!!
Press any key once or twice to continue...[]
```

Figure 6: Program's response when users try to remove an account that does not exist.

When users successfully remove an account there will be a message like this and users will be sent back to the main menu after pressing any key:

```
-----  
Removing a bank account...  
-----  
1. Remove by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Bank account removed successfully!!  
Press any key to continue...  
█
```

Figure 7: Program's response after successfully removing the account.

When users try to withdraw more money than the balance of the account, there will be an error message before let user to re-attempt:

```
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Enter amount: 234234  
Not enough money in the account!!  
Press any key once or twice to continue...█
```

Figure 8: Program's response when users try to withdraw more money than the balance.

Until user successfully withdraw their right amount of money there will be a message before sending user back to the main menu so that they can use another functionality of the program if they want:

```
-----  
Withdrawing money from a bank account...  
-----  
1. Withdraw by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 2  
Enter amount: 1  
Money withdrawn successfully!!  
Press any key to continue...  
█
```

Figure 9: Program's response when users successfully withdraw money.

In the end, those are the ways for my programs to deal with exceptional input.

Testing

In the present segment of this report, a comprehensive examination will be conducted to assess the various functionalities of the program. The objective is to provide an illustrative depiction of the bank system's appearance and ascertain the overall robustness of the bank management system.

After running the program:

```
.....
| Welcome to the Bank Management System |
| .....
| 1. Create a bank account               |
| 2. Remove bank account                 |
| 3. Display all bank accounts           |
| 4. Find a bank account info            |
| 5. Deposit money                       |
| 6. Withdraw money                     |
| 7. Exit                               |
| .....
Enter your choice (1-7): 
```

Figure 10: Main Menu

Pressed “1” to create a bank account:

```
-----
Creating a new bank account...
-----
1. Create a savings account
2. Create a checking account
3. Back
-----
Enter your choice (1-3): 
```

Figure 11: Program’s menu when users want to create a new account

Pressed “1” to create a saving account:

```
-----
Creating a new savings account...
-----
Enter account holder name (must be alphabetic only): 
```

Figure 12: Program’s response when users want to create a saving account

Fill information of a saving account:

```
-----  
Creating a new savings account...  
-----  
Enter account holder name (must be alphabetic only): Hanh Nguyen  
Enter account number: 1  
Enter balance: 1000  
Enter annual interest rate: 0.069  
Enter term in months: 13  
Savings account created successfully!!  
Press any key to continue...  
█
```

***Figure 13:** Program's form and prompt for user to enter information.*

After pressing any key, the program gets users back to the main menu

```
.....  
| Welcome to the Bank Management System |  
.....  
| 1. Create a bank account                |  
| 2. Remove bank account                 |  
| 3. Display all bank accounts            |  
| 4. Find a bank account info             |  
| 5. Deposit money                       |  
| 6. Withdraw money                      |  
| 7. Exit                               |  
.....  
Enter your choice (1-7): █
```

***Figure 14:** Program's response when users successfully withdraw money.*

Testing the display all bank accounts feature:

```
-----  
Displaying all bank accounts...  
-----  
Account 1  
Account type: Savings Account  
Account holder name: Hanh Nguyen  
Account number: 1  
Balance: 1000  
Interest rate: 0.069  
Term in months: 13  
Estimated monthly interest earn: 5.75  
Estimated interest earn after term: 74.75  
Estimated balance after term: 1074.75  
  
Press any key to continue...  
█
```

***Figure 15:** Program's response when user asks for all the information of all the account.*

Testing the find a bank account info through account number feature:

```
-----  
Displaying a bank account...  
-----  
1. Display by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Account type: Savings Account  
Account holder name: Hanh Nguyen  
Account number: 1  
Balance: 1000  
Interest rate: 0.069  
Term in months: 13  
Estimated monthly interest earn: 5.75  
Estimated interest earn after term: 74.75  
Estimated balance after term: 1074.75  
Press any key to continue...  
█
```

Figure 16: Program's response when users want to query a specific account

Testing the deposit money feature:

```
-----  
Depositing money to a bank account...  
-----  
1. Deposit by account number  
2. Back  
-----  
Enter your choice (1-2): 
```

Figure 17: Menu Of Deposit Money

```
-----  
Depositing money to a bank account...  
-----  
1. Deposit by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Enter amount: 5000  
Money deposited successfully!!  
Press any key to continue...  

```

Figure 18: Program Prompts Form for User to Input

```
Enter your choice (1-2): 1  
Enter account number: 1  
Account type: Savings Account  
Account holder name: Hanh Nguyen  
Account number: 1  
Balance: 6000  
Interest rate: 0.069  
Term in months: 13  
Estimated monthly interest earn: 34.5  
Estimated interest earn after term: 448.5  
Estimated balance after term: 6448.5  
Press any key to continue...  

```

Figure 19: The Balance of The Account After Depositing money.

Testing the withdraw money feature:

```
-----  
Withdrawing money from a bank account...  
-----  
1. Withdraw by account number  
2. Back  
-----  
Enter your choice (1-2): 
```

Figure 20: Menu Of Withdraw Money

```
-----  
Withdrawing money from a bank account...  
-----  
1. Withdraw by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Enter amount: 1000  
Money withdrawn successfully!!  
Press any key to continue...  

```

Figure 21: Program Prompts Form for User to Input

```
-----  
Displaying a bank account...  
-----  
1. Display by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Account type: Savings Account  
Account holder name: Hanh Nguyen  
Account number: 1  
Balance: 5000  
Interest rate: 0.069  
Term in months: 13  
Estimated monthly interest earn: 28.75  
Estimated interest earn after term: 373.75  
Estimated balance after term: 5373.75  
Press any key to continue...  

```

Figure 22: The Balance of The Account After Withdrawing money.

Testing the remove account feature:

```
-----  
Removing a bank account...  
-----  
1. Remove by account number  
2. Back  
-----  
Enter your choice (1-2): 1  
Enter account number: 1  
Bank account removed successfully!!  
Press any key to continue...  
█
```

Figure 23: The Program Responses After Deleting an Account

```
-----  
Displaying all bank accounts...  
-----  
Press any key to continue...  
█
```

Figure 24: The Account Has Been Deleted Successfully

Program Exited:

```
.....  
| Welcome to the Bank Management System |  
.....  
| 1. Create a bank account |  
| 2. Remove bank account |  
| 3. Display all bank accounts |  
| 4. Find a bank account info |  
| 5. Deposit money |  
| 6. Withdraw money |  
| 7. Exit |  
.....  
Enter your choice (1-7): 7  
Thank you for using our system!!! (= + =)  
PS C:\Users\dphng\OneDrive\Máy tính\Project\Bank Management System> █
```

Figure 25: Program's response when users successfully withdraw money.

Discussion

The present inquiry acknowledges the existence of limitations within the bank management system and expresses a desire to rectify and enhance these shortcomings in the subsequent iteration of the program.

The current implementation of this project lacks the functionality to read and write data to text files. Consequently, there is no provision for maintaining a textual record of the data entered by the user. This limitation may pose challenges for users in terms of storing and managing their data effectively.

Furthermore, it should be noted that the bank management system under consideration possesses a restricted user base, limited exclusively to Bank employees who possess the requisite level of authorization to both access and manipulate the confidential data associated with each individual bank account. According to the user's perspective, the program's potential for improvement lies in the incorporation of login functionality. This addition would enable the program to offer tailored access to bank accounts, contingent upon the identity of the individual utilizing the bank management system.

Moreover, the bank management system developed by the user has the capability to effectively handle monetary transactions between accounts. It is worth noting that the user intends to enhance this functionality in subsequent updates. One additional constraint of the program under consideration pertains to the lack of specificity exhibited by error messages in certain scenarios. Consequently, users may encounter challenges in effectively pinpointing and addressing issues.

The present analysis highlights the identified limitations within the program at hand. It is postulated that through a diligent and time-intensive approach, these limitations can be effectively addressed and subsequently enhanced soon. Finally, it is imperative to acknowledge that there may exist inherent unforeseen bugs or outputs that have not yet been identified. Therefore, I would greatly appreciate receiving feedback and reviews, as they play a vital role in the ongoing development of my bank management system.

Conclusion

In summary, the successful implementation of the bank management system project has served as a valuable tool for comprehending and implementing the fundamental tenets OOP. The project has effectively showcased the ways in which OOP can significantly improve the modularity, reusability, and maintainability of a codebase. This has solidified OOP as a fundamental and indispensable concept within the realms of computer science and software engineering.

The bank management system, which has been specifically developed to meet the requirements of bank tellers, efficiently manages a wide range of financial transactions and customer interactions. The practical implementation of OOP principles serves to validate their efficacy and underscores their significance in the development of user-friendly and accurate software systems.

The successful culmination of this endeavor highlights the significance of OOP in contemporary software engineering and its capacity to streamline intricate systems by breaking them down into manageable, modular elements. The present project is anticipated to function as a foundational element for subsequent investigations and utilization of OOP principles in forthcoming software development undertakings.