Bank management system

1.Introduction

The Banking Management System (BMS) is a software application designed to facilitate various banking operations such as account creation, deposits, withdrawals, and balance inquiries. Developed using C++, this project aims to provide an efficient and user-friendly interface for managing bank accounts. The system leverages object-oriented programming principles, enabling modular code organization and reusability. By simulating real-world banking functionalities, this project serves as an educational tool for understanding programming concepts while providing practical experience in software development.

1.1 Problem Statement

In traditional banking systems, managing customer accounts and transactions can be cumbersome and prone to errors when handled manually. Customers often face long wait times for simple transactions, and banks struggle with maintaining accurate records. The primary problems addressed by the Banking Management System include:

Inefficiency in processing banking transactions.

Difficulty in tracking account details and balances.

Lack of a centralized system for managing customer accounts.

Errors arising from manual record-keeping.

The BMS aims to streamline these processes by automating banking operations, ensuring accuracy, and improving customer service.

1.2 Scope

The scope of the Banking Management System includes:

Account Management: Creation, modification, and deletion of bank accounts.

Transaction Handling: Facilitating deposits and withdrawals while ensuring that all operations adhere to banking regulations.

Data Persistence: Implementing file handling techniques to store account information securely.

User Interaction: Providing a console-based interface that allows users to interact with the system seamlessly.

This project is intended for educational purposes and can be expanded into a more comprehensive banking solution by incorporating additional features such as loan management or online banking functionalities.

1.3 Objectives

The objectives of the Banking Management System project are as follows:

To develop a robust application that mimics real-world banking operations.

To implement core functionalities such as account creation, deposits, withdrawals, and balance inquiries.

To utilize object-oriented programming principles for better code organization and maintainability.

To enhance user experience through a simple and intuitive interface.

To ensure data integrity through effective file handling mechanisms.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.Requirements

2.1 Hardware Requirements

To run the Banking Management System effectively, the following hardware specifications are recommended:

**Processor**: **Intel Core i3** or equivalent

**RAM**: Minimum **4 GB**

**Storage**: At least **500 MB** of free disk space

**Display:** **15-inch monitor** or larger with a resolution of **1366 x 768** or higher

**Input Devices**: **Keyboard** and **mouse**

2.2 Software Requirements

The software requirements for developing and running the Banking Management System include:

**Operating System**: **Windows/Linux/Mac OS**

**Compiler**: A C++ compiler like GCC or an Integrated Development Environment (IDE) such as **Code Blocks** or **Visual Studio**

**Text Editor**: Any text editor (e.g., **Notepad++, Sublime Text**) for writing code

**Libraries**: Standard **C++ libraries** (no external libraries are required)

2.3 Functional Requirements

The functional requirements of the Banking Management System are outlined below:

**Account Creation:**

Users can create new bank accounts by providing necessary details (name, account number, type, initial deposit).

**Deposit Functionality:**

Users can deposit money into their accounts by specifying the account number and amount.

**Withdrawal Functionality:**

Users can withdraw money from their accounts with checks to ensure sufficient balance.

**Balance Inquiry:**

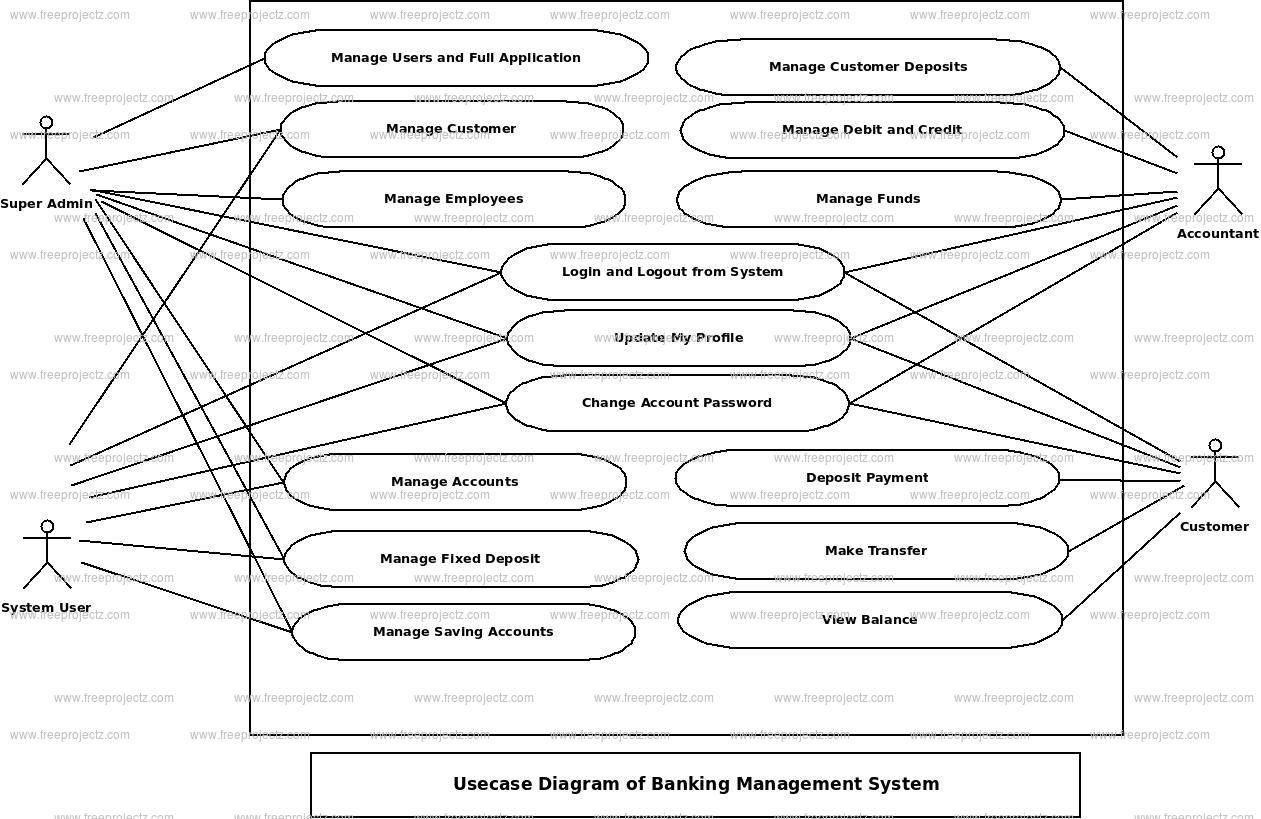
Users can check their account balance along with other details like account holder's name and type.

**Account Management:**

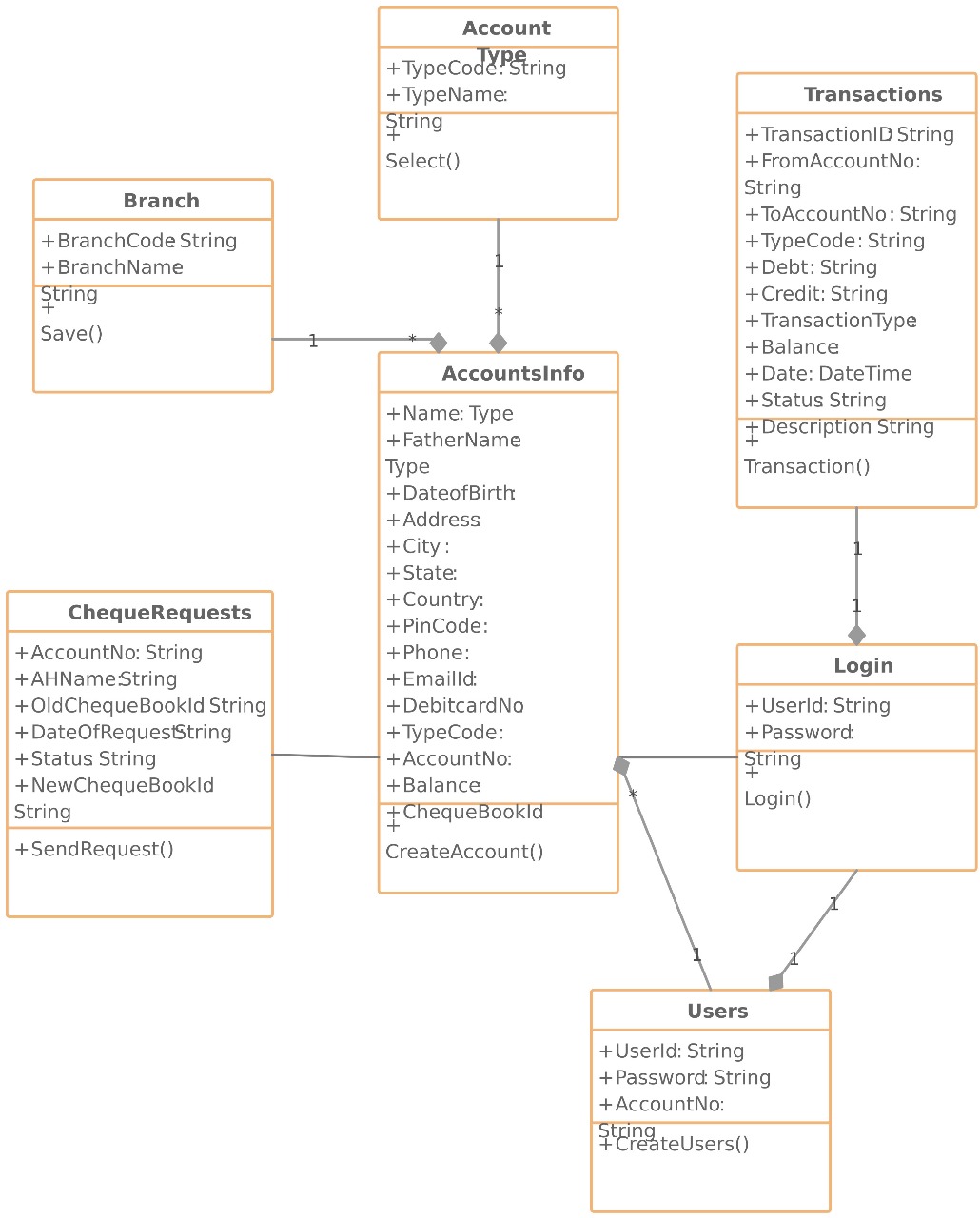
Users can view all accounts, modify account details, or close an account when needed.

3.Diagrams

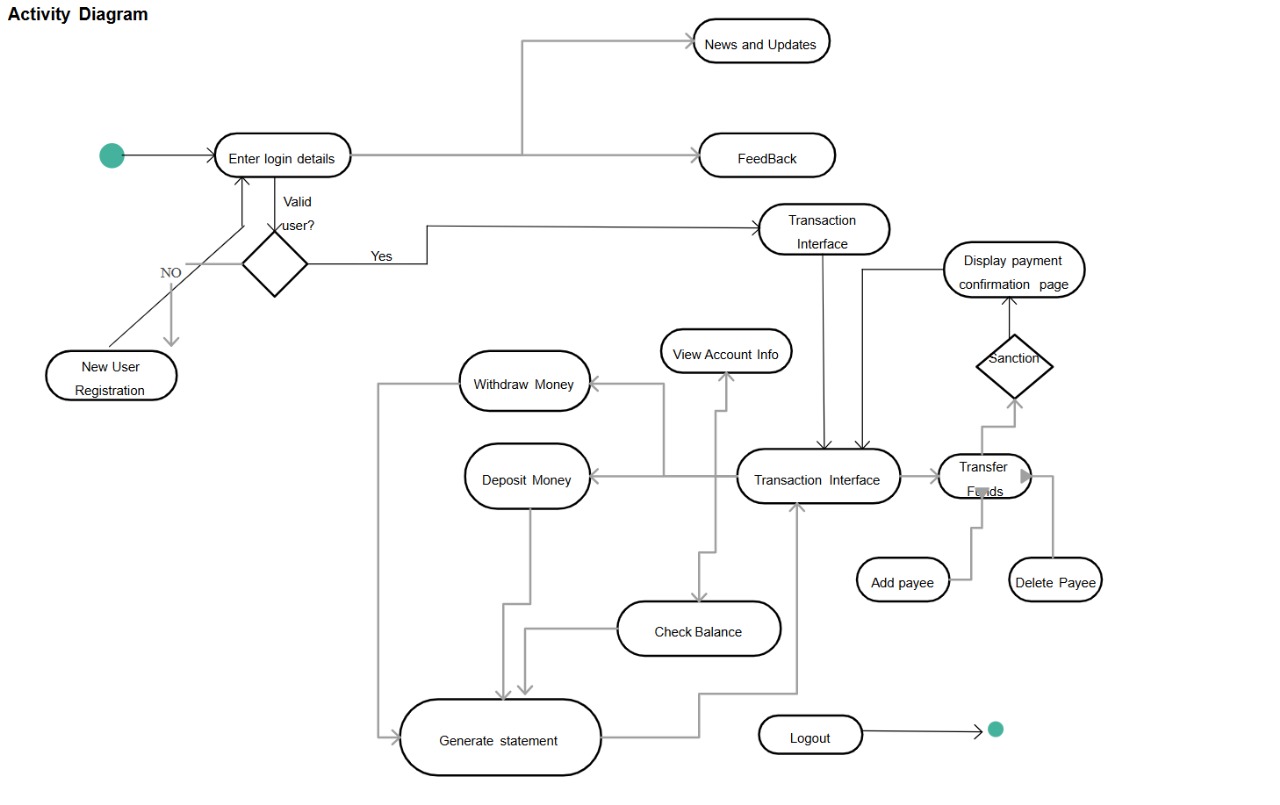
3.1 USE CASE DIAGRAM



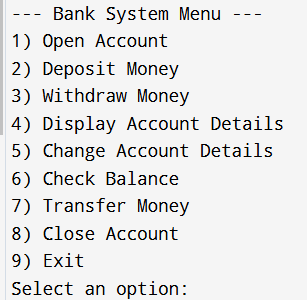
3.2 MODULAR DIAGRAM



3.3 ACTIVITY DIAGRAM



4.Design/Interface



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5.Implementation

#include <iostream>

#include <string>

using namespace std;

class Bank

{

string name, address;

char acc\_type;

float balance;

public:

void open\_account();

void deposit\_money();

void withdraw\_money();

void display\_account();

void change\_account\_details();

void check\_balance();

void transfer\_money(Bank &recipient);

void close\_account();

};

void Bank::open\_account()

{

cout << "Enter your full name: ";

getline(cin, name);

cout << "Enter your address: ";

getline(cin, address);

cout << "What type of account would you like to open? Saving (S) or Current (C): ";

cin >> acc\_type;

cout << "Enter the amount you would like to deposit: ";

cin >> balance;

cout << "Account Created Successfully\n";

}

void Bank::deposit\_money()

{

float amount;

cout << "Enter the amount you want to deposit: ";

cin >> amount;

if (amount > 0)

{

balance += amount;

cout << "\nDeposit Successful. Available Balance: Rs." << balance << endl;

}

else

{

cout << "\nInvalid amount entered.\n";

}

}

void Bank::withdraw\_money()

{

float amount;

cout << "Enter the amount you want to withdraw: ";

cin >> amount;

if (amount > 0 && amount <= balance)

{

balance -= amount;

cout << "\nWithdrawal Successful. Available Balance: Rs." << balance << endl;

}

else

{

cout << "\nInsufficient balance or invalid amount.\n";

}

}

void Bank::display\_account()

{

cout << "\nAccount Details:\n";

cout << "Name: " << name << endl;

cout << "Address: " << address << endl;

cout << "Account Type: " << (acc\_type == 'S' ? "Saving" : "Current") << endl;

cout << "Balance: Rs." << balance << endl;

}

void Bank::change\_account\_details()

{

cout << "Enter new address: ";

cin.ignore();

getline(cin, address);

cout << "Change account type? Saving (S) or Current (C): ";

cin >> acc\_type;

cout << "Account details updated.\n";

}

void Bank::check\_balance()

{

cout << "Available Balance: Rs." << balance << endl;

}

void Bank::transfer\_money(Bank &recipient)

{

float amount;

cout << "Enter the amount to transfer: ";

cin >> amount;

if (amount > 0 && amount <= balance)

{

balance -= amount;

recipient.balance += amount;

cout << "Transfer Successful. Your Available Balance: Rs." << balance << endl;

}

else

{

cout << "Transfer failed: insufficient balance or invalid amount.\n";

}

}

void Bank::close\_account()

{

cout << "Closing your account...\n";

name = address = "";

acc\_type = '\0';

balance = 0;

cout << "Account closed successfully.\n";

}

int main()

{

Bank customer1, customer2;

int choice;

do

{

cout << "\n--- Bank System Menu ---\n";

cout << "1) Open Account\n";

cout << "2) Deposit Money\n";

cout << "3) Withdraw Money\n";

cout << "4) Display Account Details\n";

cout << "5) Change Account Details\n";

cout << "6) Check Balance\n";

cout << "7) Transfer Money\n";

cout << "8) Close Account\n";

cout << "9) Exit\n";

cout << "Select an option: ";

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

customer1.open\_account();

break;

case 2:

customer1.deposit\_money();

break;

case 3:

customer1.withdraw\_money();

break;

case 4:

customer1.display\_account();

break;

case 5:

customer1.change\_account\_details();

break;

case 6:

customer1.check\_balance();

break;

case 7:

cout << "Transferring money to another customer...\n";

customer1.transfer\_money(customer2);

break;

case 8:

customer1.close\_account();

break;

case 9:

cout << "Thank you for using the banking system. Goodbye!\n";

break;

default:

cout << "Invalid option. Please try again.\n";

break;

}

}

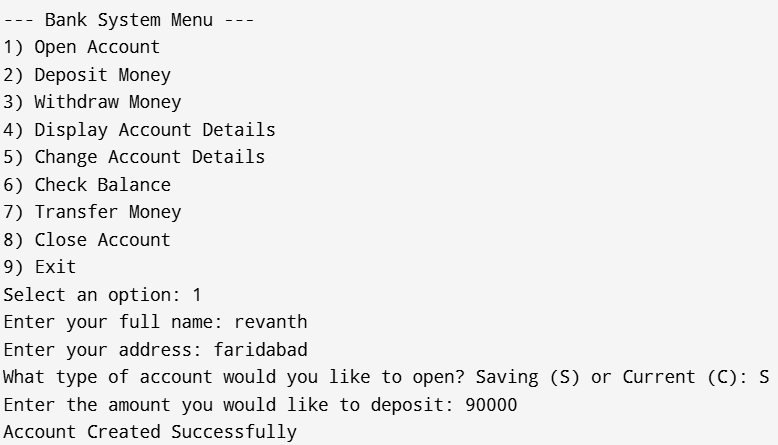
while (choice != 9);

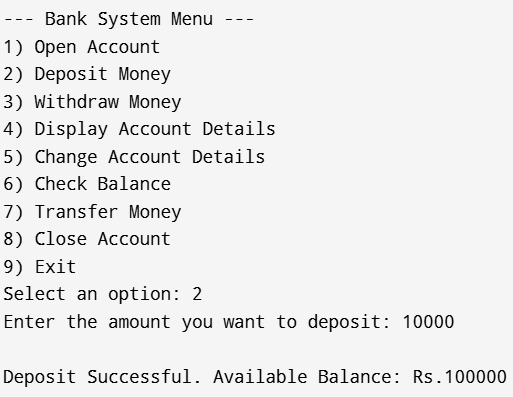
return 0;

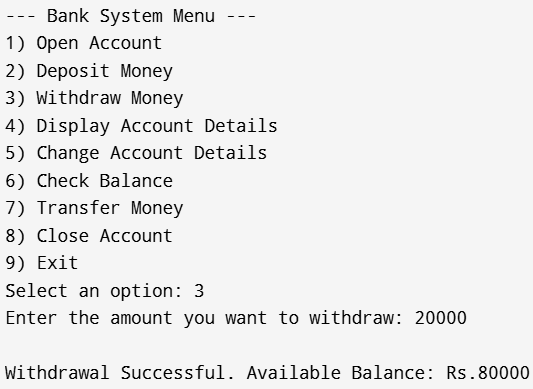
}

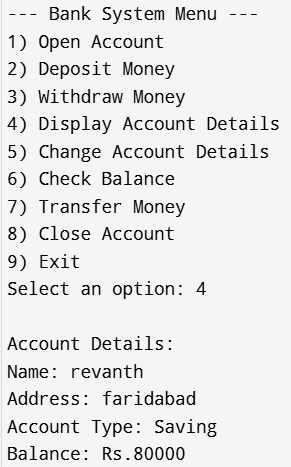
6.Output

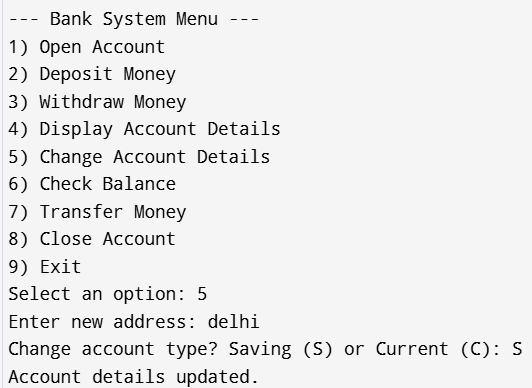
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

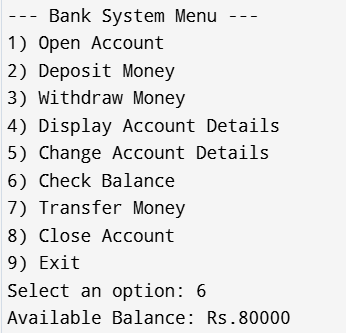


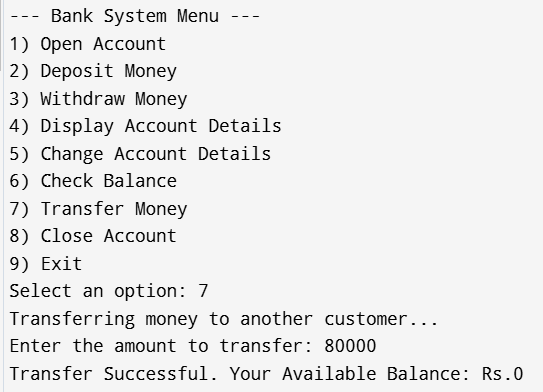


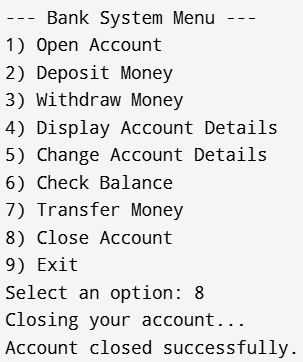


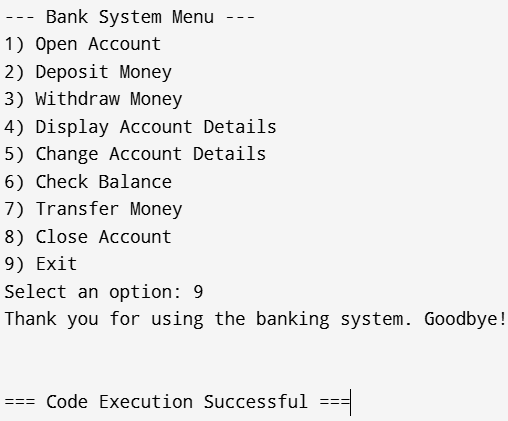












7. Conclusion

The Banking Management System developed in C++ effectively addresses the challenges faced by traditional banking methods through automation and improved efficiency. By implementing essential banking functionalities within a user-friendly interface, this project not only demonstrates fundamental programming skills but also provides insights into real-world applications of software development. The successful implementation of this system lays the groundwork for further enhancements and more complex features that could be integrated into future versions.

8. References

<https://www.includehelp.com/cpp-programs/banking-management-system-using-class.aspx>

<https://t4tutorials.com/bank-management-system-project-in-c/>

<https://codewithcurious.com/projects/banking-management-system-using-c/>

<https://www.youtube.com/watch?v=AH7WCp7JJPk>