[Banking System]

Project submitted to the SRM University – AP, Andhra Pradesh

For the partial fulfillment of the requirements to award the degree of

Bachelor of Technology In Computer Science and Engineering School of Engineering and Sciences

Submitted by

Gunavanth Reddy Pappula

(AP23110010505)

Balaji Bhavani Shankar Gogula

(AP23110010546)

Under the Guidance of

(Prof. / Dr. **Kavitha Rani Karnena)**

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November,2024]

Certificate

Date: 16-Nov-2024

This is to certify that the work present in this Project entitled "**Banking System**" has been carried out by **[Gunavanth Reddy(AP23110010505),Balaji Bhavani Shankar(AP23110010546)]** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. / Dr. kavitha Rani Karnena

Designation,

Affiliation.

Co-supervisor

(Signature)

Prof. / Dr. [Name]

Designation,

Affiliation.

Acknowledgements

We would like to express our gratitude to Dr. Aurobindo Behera for assisting us with the research at every step. Without the advice and assistance of our adviser, this study effort would not be feasible. The suggestions, feedbacks, and encouragement from our advisors were extremely helpful in assisting us in finishing this research.

Abstract

This report details the development of a comprehensive banking system using C++ object-oriented programming (OOP) concepts. The program simulates essential banking functionalities, including account creation, deposits, withdrawals, balance inquiries, and fund transfers. The system is structured using OOP principles for modular and secure data handling, prioritizing encapsulation, data abstraction, and user accessibility. The program utilizes clear and well-defined classes and functions to ensure secure operations, efficient management of customer information, and accurate account balances.

1. Introduction

This banking system program is a practical demonstration of the application of C++ OOP concepts in real-world scenarios. The program comprises two core classes: Account and BankingSystem. The Account class encapsulates the functionalities of an individual bank account, managing account details, deposits, withdrawals, and transfers. The BankingSystem class acts as the main interface, handling user interactions, managing accounts, and coordinating various banking operations.

1.1 Core Classes

1. Account Class:

- **Data Members:**
    - accountNumber: Unique identifier for each account (integer).
    - accountHolderName: Name of the account holder (string).
    - balance: Current account balance (double).
- **Methods:**
    - **getAccountNumber()**: Returns the account number.
    - **getAccountHolderName()**: Returns the account holder name.
    - **getBalance()**: Returns the current account balance.
    - **deposit(double amount)**: Adds the specified amount to the account balance.
    - **withdraw(double amount)**: Deducts the specified amount from the balance if sufficient funds are available.
    - **transfer(Account &receiver, double amount)**: Transfers the specified amount from the current account to another account.

2. BankingSystem Class:

- **Data Members:**
  - **accounts**: A vector to store all Account objects.
- **Methods:**
  - **createAccount()**: Creates a new Account object with a unique account number and the account holder's name.
  - **deposit()**: Handles deposit operations, prompting the user for the account number and deposit amount.
  - **withdraw()**: Handles withdrawal operations, prompting the user for the account number and withdrawal amount.
  - **balanceEnquiry()**: Displays the balance of a specified account.
  - **transfer()**: Handles fund transfers between accounts, prompting for sender, receiver account numbers, and the amount to be transferred.
  - **menu()**: Presents a menu of available options to the user for interacting with the banking system.

1.2 Key Features

- **Encapsulation:** The program utilizes private data members within each class, ensuring data integrity and security. These private members are accessible only through public methods, preventing unauthorized manipulation.
- **Data Abstraction:** The program provides user-friendly interfaces to interact with the system. The underlying implementation details are hidden from the user, making the system easy to use.
- **Modularity:** The program is divided into distinct classes, each responsible for specific functionalities. This modular structure promotes code readability, maintainability, and extensibility.

1.3 Security Measures

- **Private Access Specifiers:** The account balance and account holder information are private members, ensuring that sensitive data is protected from unauthorized access.

- **Input Validation:** The program checks for valid amounts during deposits, withdrawals, and transfers, preventing negative or invalid transactions.
- **Error Handling:** The system provides feedback for invalid operations, such as insufficient funds or incorrect account numbers, ensuring users are informed of any issues.

2. Methodology

The development of the banking system followed a structured methodology:

1. **Class Design:** The Account and BankingSystem classes were designed to encapsulate related properties and methods, adhering to OOP principles for clear responsibility division.
2. **Function Implementation:** Core functionalities such as account creation, deposits, withdrawals, balance inquiries, and transfers were implemented to handle transactions securely and ensure accurate balance updates.
3. **User Interaction:** A menu-driven interface allows users to perform various banking operations interactively, enhancing user experience and engagement.
4. **Testing and Validation:** Each method was rigorously tested with various inputs to ensure accurate results, proper error handling, and adherence to the program's secure banking requirements.

2.1 Modular Design

The program's modular design separates functionalities across the Account and BankingSystem classes. Each class encapsulates distinct features, enabling organized code, easier debugging, and the potential for future modifications. This design aligns with key principles of Object-Oriented Programming (OOP), such as encapsulation and abstraction, which protect data and simplify program structure.

3. Discussion

3.1 Input of the Code

```
#include <iostream>

#include <vector>

#include <limits>


using namespace std;


class Account {

private:

    int accountNumber;

    string accountHolderName;

    double balance;


public:

    Account(int accNum, string name) : accountNumber(accNum),
accountHolderName(name), balance(0.0) {}


    int getAccountNumber() const {

        return accountNumber;

    }


    string getAccountHolderName() const {

        return accountHolderName;

    }
```

```
double getBalance() const {
    return balance;
}

void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        cout << "Deposited: $" << amount << endl;
    } else {
        cout << "Invalid deposit amount!" << endl;
    }
}

void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        cout << "Withdrawn: $" << amount << endl;
    } else {
        cout << "Invalid withdrawal amount!" << endl;
    }
}

void transfer(Account &receiver, double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        receiver.deposit(amount);
```

```
            cout << "Transferred: $" << amount << " to account " << receiver.getAccountNumber()
<< endl;
        } else {
            cout << "Invalid transfer amount!" << endl;
        }
    }
};

class BankingSystem {
private:
    vector<Account> accounts;

public:
    void createAccount() {
        int accNum = accounts.size() + 1; // Auto-increment account number
        string name;
        cout << "Enter account holder name: ";
        cin.ignore(); // Clear the newline character in the input buffer
        getline(cin, name);
        accounts.emplace_back(accNum, name);
        cout << "Account created successfully! Account Number: " << accNum << endl;
    }

    void deposit() {
        int accNum;
        double amount;
        cout << "Enter account number: ";
        cin >> accNum;
    cout << "Enter amount to deposit: ";
    cin >> amount;

    if (accNum > 0 && accNum <= accounts.size()) {
        accounts[accNum - 1].deposit(amount);
    } else {
        cout << "Invalid account number!" << endl;
    }
}

void withdraw() {
    int accNum;
    double amount;
    cout << "Enter account number: ";
    cin >> accNum;
    cout << "Enter amount to withdraw: ";
    cin >> amount;

    if (accNum > 0 && accNum <= accounts.size()) {
        accounts[accNum - 1].withdraw(amount);
    } else {
        cout << "Invalid account number!" << endl;
    }
}

void balanceEnquiry() {
    int accNum;
    cout << "Enter account number: ";
```

```
        cin >> accNum;

        if (accNum > 0 && accNum <= accounts.size()) {
            cout << "Balance for account " << accNum << ": $" << accounts[accNum -
1].getBalance() << endl;
        } else {
            cout << "Invalid account number!" << endl;
        }
    }

    void transfer() {
        int senderAccNum, receiverAccNum;
        double amount ;
        cout << "Enter sender account number: ";
        cin >> senderAccNum;
        cout << "Enter receiver account number: ";
        cin >> receiverAccNum;
        cout << "Enter amount to transfer: ";
        cin >> amount;

        if (senderAccNum > 0 && senderAccNum <= accounts.size() && receiverAccNum > 0 &&
receiverAccNum <= accounts.size()) {
            accounts[senderAccNum - 1].transfer(accounts[receiverAccNum - 1], amount);
        } else {
            cout << "Invalid account numbers!" << endl;
        }
    }

    void menu() {
```

```
int choice;

do {

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

   cout << "1. Create Account\n";

   cout << "2. Deposit\n";

   cout << "3. Withdraw\n";

   cout << "4. Balance Enquiry\n";

   cout << "5. Transfer\n";

   cout << "6. Exit\n";

   cout << "Enter your choice: ";

   cin >> choice;


   switch (choice) {

      case 1:

         createAccount();

         break;

      case 2:

         deposit();

         break;

      case 3:

         withdraw();

         break;

      case 4:

         balanceEnquiry();

         break;

      case 5:

         transfer();

         break;
```

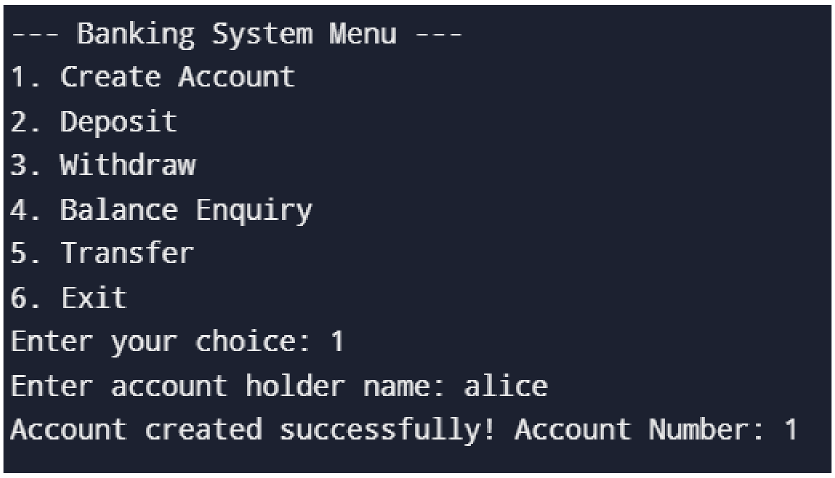
```
                case 6:
                    cout << "Exiting the Banking System. Goodbye!" << endl;
                    break;
                default:
                    cout << "Invalid choice! Please try again." << endl;
                    break;
            }
        } while (choice != 6);
    }
};

int main() {
    BankingSystem bankSystem;
    bankSystem.menu();
    return 0;
}
```

3.2 Output of Code

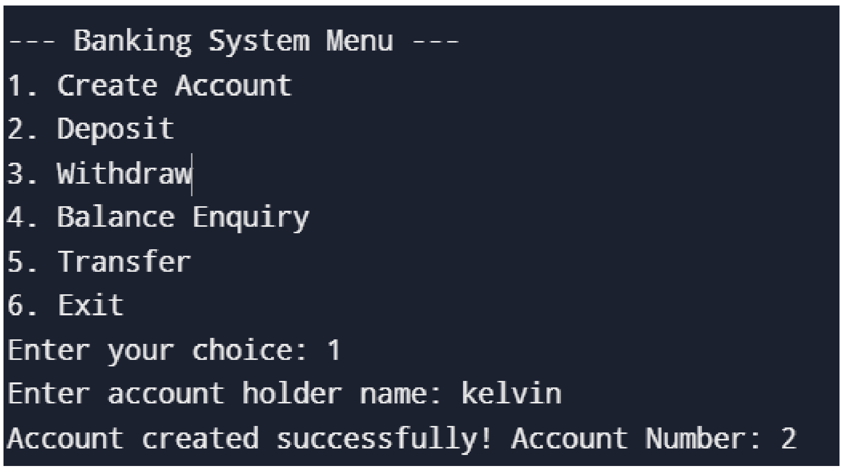
The output of the banking system program will vary based on user interactions. Below is a sample interaction:

#account 1 created



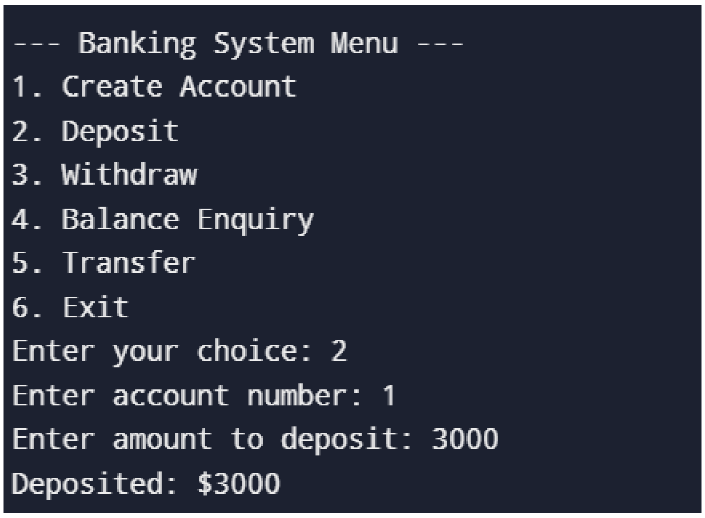
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 1
Enter account holder name: alice
Account created successfully! Account Number: 1
```

#account 2 created



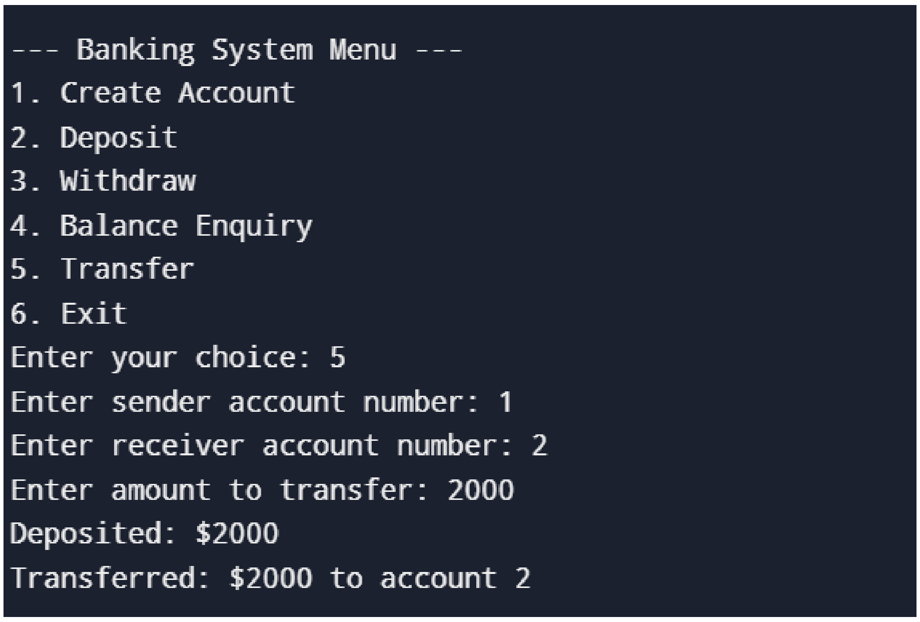
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 1
Enter account holder name: kelvin
Account created successfully! Account Number: 2
```

#amount deposited into account 1



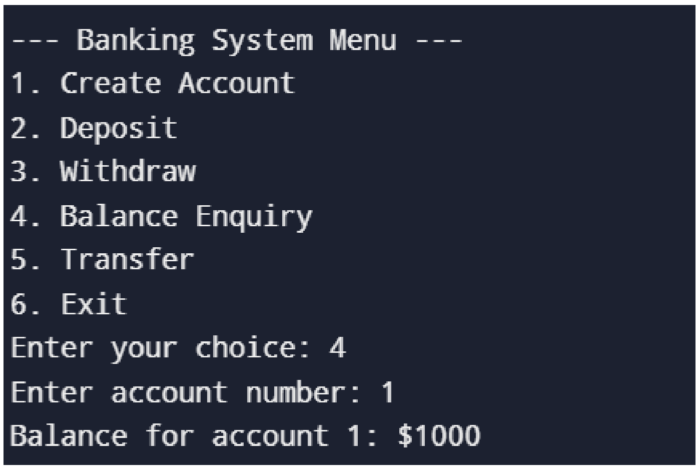
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 2
Enter account number: 1
Enter amount to deposit: 3000
Deposited: $3000
```

#amount transferred to account 2 from account 1



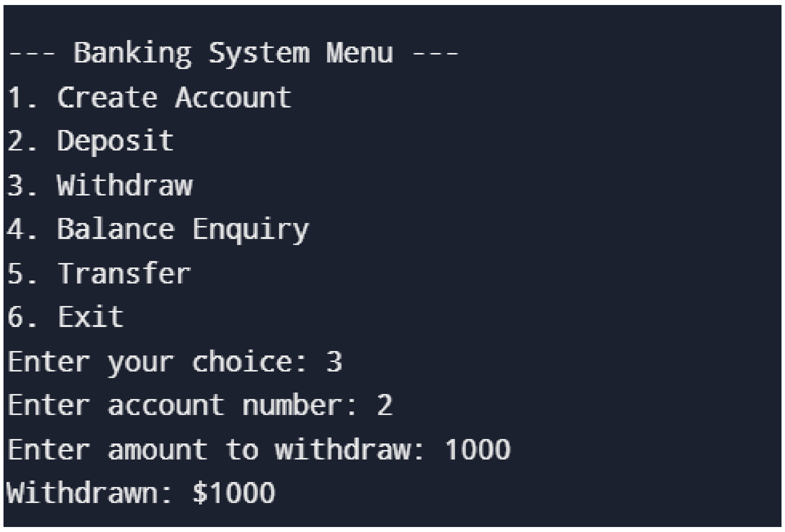
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 5
Enter sender account number: 1
Enter receiver account number: 2
Enter amount to transfer: 2000
Deposited: $2000
Transferred: $2000 to account 2
```

#balance amount of account 1 displayed



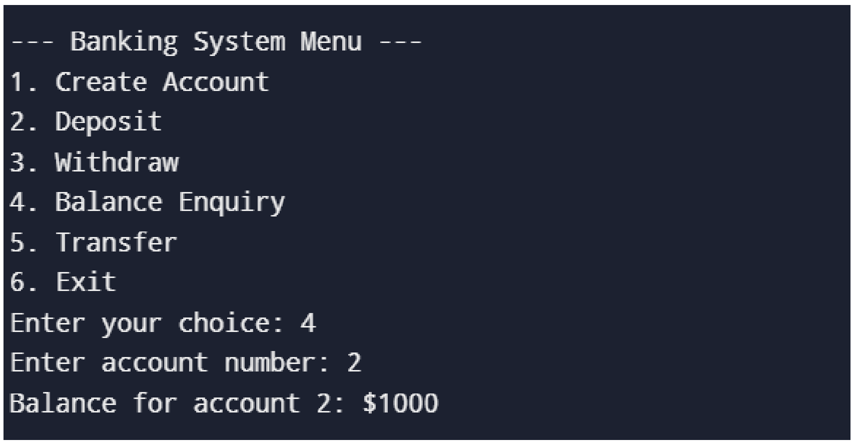
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 4
Enter account number: 1
Balance for account 1: $1000
```

#withdrawn amount from account 2



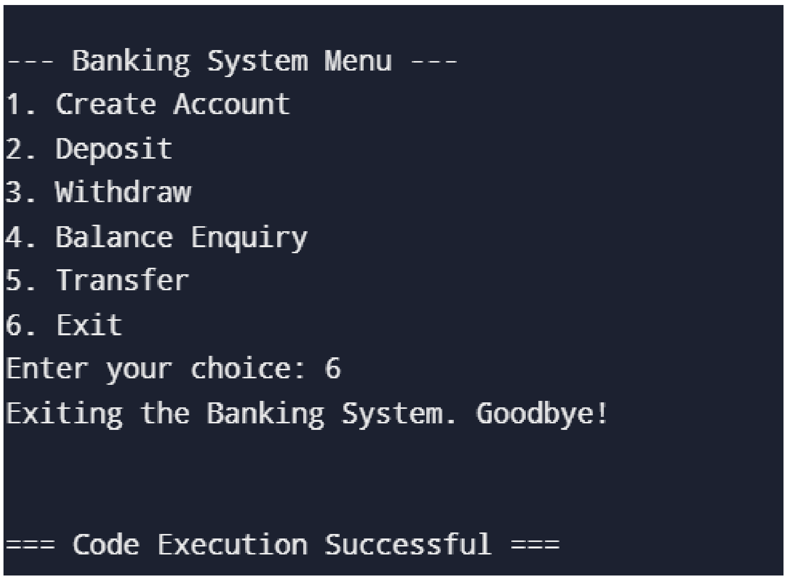
```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 3
Enter account number: 2
Enter amount to withdraw: 1000
Withdrawn: $1000
```

#balance amount of account 2 showed



```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 4
Enter account number: 2
Balance for account 2: $1000
```

# program exited



```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Transfer
6. Exit
Enter your choice: 6
Exiting the Banking System. Goodbye!


=== Code Execution Successful ===
```

4.Concluding Remarks

This C++ banking system program showcases how object-oriented programming (OOP) principles can help create a secure and user-friendly banking experience. Using classes like \`Account\` and \`BankingSystem\`, the program organizes account operations and manages user interactions effectively.

The program supports core banking functions: opening new accounts, making deposits and withdrawals, checking balances, and transferring funds between accounts. With private access specifiers and input validation, it maintains data security and reliability. Its modular design makes it easy to update and expand, leaving room for potential upgrades like transaction history tracking or user authentication.

Overall, this project is a straightforward yet powerful example of OOP in action, highlighting how these concepts can be applied in real-world financial software.

References

1. The C++ Programming Language by Bjarne Stroustrup
2. C++ Primer by Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo
3. Object-Oriented Programming in C++ by Robert Lafore