Bank Management

Project submitted to the

SRM University – AP, Andhra Pradesh

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

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Candidate Name

AP22110010197 Somanadha Sai

AP22110010168 Bhuvanesh Reddy

AP22110010141 Roshini

AP22110010189 Chehak Saluja

Under the Guidance of

K.Kavitha Rani

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Certificate

Date: 10-Dec-23

This is to certify that the work present in this Project entitled "**Bank Management**" has been carried out by AP22110010197 Somanadha Sai, AP22110010168 Bhuvanesh Reddy, AP22110010141 Roshini, AP22110010189 Chehak Saluja . 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

Prof. / Dr. K.Kavitha Rani

Assistant Professor, CSE Department,

SRM UNIVERSITY, AP

Acknowledgements: -

I would like to thank my teacher, K.kavitha Rani, forgiving me the opportunity to work on this project. This project taught me a lot about different core concepts of Object-Oriented Programming, such as dataen capsulation and static member variable, exception handling, objects, and classes etc. I am extremely grateful land express my profound gratitude and indebtedness to my project guide teacher for the kind help and forgiving me the necessary guidance and valuable suggestions in completing this project work.

This C++ code is a basic implementation of a banking system and seems to be written by an individual developer. The code structure, use of classes, and overall logic are clear and well-organized. It provides a foundation for a simple interactive banking program.

Keep in mind that coding styles and practices may vary, and there could be different ways to implement similar functionality. Additionally, it's important to consider security aspects, error handling, and potential improvements based on specific requirements.

Table of Contents

Abstract:-

This report presents the design and implementation of a simple banking system using C++. The system includes classes for managing accounts and a basic banking application with functionalities such as account creation, removal, balance management, and transactions.

The program defines two classes, Account and Bank, to model individual bank accounts and a collection of accounts, respectively. Users can perform operations such as adding accounts, removing accounts, displaying account information, and conducting transactions (deposits and withdrawals).

The Account class encapsulates account-related attributes and functionalities, including constructors for account initialization, accessors for retrieving account details, and methods for depositing and withdrawing funds. The Bank class manages a vector of accounts and provides methods for adding, removing, displaying, and performing transactions on accounts.

The main function orchestrates user interactions through a simple console menu, where users can choose from options to display all accounts, add an account, remove an account, perform transactions, or exit the program. The program uses standard input/output operations for user input and feedback.

Overall, this code serves as a fundamental framework for a basic banking application, allowing users to manage accounts and perform essential banking operations through a straightforward console interface.

1.Introduction

The provided C++ code presents a fundamental implementation of a console-based banking system. This system is designed to manage individual bank accounts through the use of two classes: `Account` and `Bank`. The `Account` class encapsulates the attributes and functionalities of an individual bank account, including the account number, account holder's name, and the account balance. The `Bank` class, on the other hand, serves as a container for managing a collection of accounts, offering operations such as adding accounts, removing accounts, displaying account information, and performing transactions.

Users interact with the system through a straightforward console menu in the `main` function. The menu prompts users to choose from various options, such as displaying all accounts, adding a new account, removing an existing account, performing financial transactions (deposits and withdrawals), or exiting the program. The program employs standard input/output operations and leverages C++ features such as classes, vectors, and algorithms for efficient account management.

This banking system provides a solid foundation for understanding basic object-oriented programming principles and serves as a starting point for building more sophisticated banking applications with additional features and enhanced functionalities.

2. Methodology

1.Account Class:-The Account class represents a bank account and has private data members for account number, account holder's name, and balance.

It includes a constructor for initializing the account, accessor methods to retrieve account information, and methods for depositing and withdrawing money.

2.Bank Class:-The Bank class manages a collection of accounts using a vector. It includes methods for adding an account, removing an account, displaying all accounts, and performing transactions.

3.AddAccount: Takes user input to create a new account and adds it to the vector of accounts.

3.1.RemoveAccount: Takes an account number as input, searches for the account in the vector, and removes it if found.

4.Menu and Main Function: Prints a menu with options for displaying all accounts, adding an account, removing an account, performing transactions, and exiting the program.

5.User Input and Output: The code uses cin for user input and cout for output.

It includes input validation to handle invalid choices and numeric inputs.

Overall, this code demonstrates a basic implementation of a banking system with a user-friendly menu interface. It showcases object-oriented principles such as encapsulation and separation of concerns by using classes for representing accounts and the bank.

3.Functionalities:-

1. Add an Account (`void addAccount()`)

- Prompt the user to input an account number, account holder's name, and initial balance.

- Create a new `Account` object with the provided information.

- Add the new account to the vector of accounts in the `Bank` class.

2. Remove an Account (`void remove Account()`)

- Prompt the user to enter the account number of the account to be removed.

- Search for the account in the vector of accounts using `find_if`.

- If the account is found, remove it from the vector; otherwise, display an error message.

3. Display All Accounts (`void display All Accounts() const`)

- Print a table header with columns: Account, Holder, Balance.

- Iterate through the vector of accounts and display each account's information.

4. Perform Transactions (`void perform Transactions()`)

- Prompt the user to input the account number for the transaction.

- Search for the account in the vector of accounts.

- If the account is found, prompt the user to enter the transaction amount and choose between deposit and withdrawal.

- Update the account balance accordingly, displaying success or error messages.

5. Display Menu (`void display Menu()`)
   - Print a menu with options to:
     - Display all accounts
     - Add an account
     - Remove an account
     - Perform transaction
   - Exit the program
6. Main Function (`int main()`)
   - Create an instance of the `Bank` class.
   - Use a do-while loop to repeatedly display the menu and execute the selected functionality until the user chooses to exit.

These functionalities collectively provide a basic console-based interface for managing bank accounts, allowing users to perform essential banking operations such as adding accounts, removing accounts, viewing account information, and conducting transactions.

4.Objectivies:-

The objectives of the provided C++ code, implementing a simple banking system, are as follows:

1. Account Management:

- Allow users to create new bank accounts by providing an account number, account holder's name, and initial balance.

- Enable users to view a list of all existing accounts, displaying account numbers, holders, and balances.

2. Account Removal:

- Provide functionality to remove a bank account by entering the account number.

- Display appropriate messages if the account to be removed is not found.

3. Financial Transactions:

- Allow users to perform financial transactions on existing accounts, including deposits and withdrawals.

- Prompt users to input the account number, transaction amount, and choose between deposit ('D') or withdrawal ('W').

- Display success messages for successful transactions and inform users if the account is not found or if there are insufficient funds.

4. User Interaction:

- Implement a menu-driven interface to facilitate user interaction.

- Display a menu with options to:

  - Display all accounts

  - Add a new account

  - Remove an existing account

  - Perform financial transactions

  - Exit the program

5. Code Organization and Readability:

- Implement a well-organized and readable code structure using classes for `Account` and `Bank`.

- Utilize standard C++ features such as vectors and algorithms for efficient account management.

6. Input Handling:

- Ensure proper input handling by clearing the input buffer when necessary.

- Display informative messages for invalid input or choices.

7. Program Termination:

- Allow users to exit the program when they choose option '5' from the menu.

These objectives collectively aim to create a user-friendly, functional, and organized banking system program that can manage accounts, perform transactions, and provide essential feedback to users.

Code implementation:-

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

using namespace std;

class Account {
private:
   int accountNumber;
   string accountHolder;
   double balance;

public:
   // Constructor
   Account(int number, string holder, double initialBalance)
      : accountNumber(number), accountHolder(holder), balance(initialBalance) {}

   // Accessors
   int getAccountNumber() const { return accountNumber; }
   string getAccountHolder() const { return accountHolder; }
   double getBalance() const { return balance; }

   // Functions
```

```
    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            cout << "Insufficient funds!" << endl;
        }
    }
};

class Bank {
private:
    vector<Account> accounts;

public:
    void addAccount() {
        int number;
        string holder;
        double initialBalance;

        cout << "Enter account number: ";
        cin >> number;

        cout << "Enter account holder's name: ";
```

```
    cin.ignore();  // Clear the newline left in the input buffer
    getline(cin, holder);

    cout << "Enter initial balance: ";
    cin >> initialBalance;

    Account newAccount(number, holder, initialBalance);
    accounts.push_back(newAccount);

    cout << "Account added successfully." << endl;
}

void removeAccount() {
    int accountNumber;
    cout << "Enter account number to remove: ";
    cin >> accountNumber;

    auto it = find_if(accounts.begin(), accounts.end(),
                      [accountNumber](const Account& acc) {
                          return acc.getAccountNumber() == accountNumber;
                      });

    if (it != accounts.end()) {
        accounts.erase(it);
        cout << "Account removed successfully." << endl;
    } else {
        cout << "Account not found." << endl;
```

```
    }
}

void displayAllAccounts() const {
    cout << "Account\tHolder\tBalance" << endl;
    for (const auto& account : accounts) {
        cout << account.getAccountNumber() << "\t" << account.getAccountHolder() << "\t$"
             << fixed << setprecision(2) << account.getBalance() << endl;
    }
}

void performTransactions() {
    int accountNumber;
    cout << "Enter account number: ";
    cin >> accountNumber;

    auto it = find_if(accounts.begin(), accounts.end(),
                      [accountNumber](const Account& acc) {
                          return acc.getAccountNumber() == accountNumber;
                      });

    if (it != accounts.end()) {
        double amount;
        cout << "Enter transaction amount: ";
        cin >> amount;

        char choice;
```

```
            cout << "Enter 'D' for deposit or 'W' for withdrawal: ";
            cin >> choice;

            if (choice == 'D' || choice == 'd') {
                it->deposit(amount);
                cout << "Deposit successful." << endl;
            } else if (choice == 'W' || choice == 'w') {
                it->withdraw(amount);
                cout << "Withdrawal successful." << endl;
            } else {
                cout << "Invalid choice." << endl;
            }
        } else {
            cout << "Account not found." << endl;
        }
    }
};

void displayMenu() {
    cout << "\nMenu:\n";
    cout << "1. Display All Accounts\n";
    cout << "2. Add an Account\n";
    cout << "3. Remove an Account\n";
    cout << "4. Perform Transactions\n";
    cout << "5. Exit\n";
}
```

```
int main() {
    Bank bank;

    int choice;
    do {
        displayMenu();
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                bank.displayAllAccounts();
                break;
            case 2:
                bank.addAccount();
                break;
            case 3:
                bank.removeAccount();
                break;
            case 4:
                bank.performTransactions();
                break;
            case 5:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice. Try again.\n";
```

```
        }

    } while (choice != 5);

    return 0;
}
```

Output:-

Menu:

1. Display All Accounts

2. Add an Account

3. Remove an Account

4. Perform Transactions

5. Exit

Enter your choice: 2

Enter account number: 345

Enter account holder's name: bhuvan

Enter initial balance: 40000

Account added successfully.

Menu:

1. Display All Accounts

2. Add an Account

3. Remove an Account

4. Perform Transactions

5. Exit

Enter your choice: 4

Enter account number: 345

Enter transaction amount: 300

Enter 'D' for deposit or 'W' for withdrawal: D

Deposit successful.

Menu:

1. Display All Accounts

2. Add an Account

3. Remove an Account

4. Perform Transactions

5. Exit

Enter your choice: 4

Enter account number: 345

Enter transaction amount: 300

Enter 'D' for deposit or 'W' for withdrawal: w

Withdrawal successful.

Menu:

1. Display All Accounts

2. Add an Account

3. Remove an Account

4. Perform Transactions

5. Exit

Enter your choice: 3

Enter account number to remove: 345

Account removed successfully.

Conclusion:-

In conclusion, the provided C++ code presents a straightforward implementation of a basic banking system. It incorporates fundamental object-oriented principles by defining classes for `Account` and `Bank`, offering users functionalities such as adding accounts, removing accounts, displaying account information, and performing transactions. The code employs standard input/output for user interactions and leverages vector and algorithm functionalities from the C++ Standard Library.

The structure of the code is clear and organized, making it easy to comprehend and maintain. It demonstrates the use of constructors, member functions, vectors, and lambda expressions for efficient account management. The inclusion of a menu-driven interface enhances user experience and provides a user-friendly way to interact with the banking system.

However, it's essential to note that this implementation is intended for educational and illustrative purposes. In a real-world scenario, additional considerations such as error handling, security measures, and more complex transaction logic would need to be incorporated.

In summary, the provided code serves as a foundational example of a console-based banking system, offering room for further enhancements and modifications based on specific requirements or advanced features that might be desired in a comprehensive banking application.