**Banking System with OOPs**

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

**Jayanth Ramakrishnan**

**(AP23110010340)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**Kavitha Rani**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[November, 2024]**

# Certificate

Date: 20-Nov-2024

This is to certify that the work present in this Project entitled “**Banking System with OOPs**” has been carried out by **Jayanth Ramakrishnan** 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

# 

# Acknowledgements

Code Draft and Execution : Kaleem Afroz Shaik

Code Debugging and Class Modulation : Sesank

Report Writing and Draft : Jayanth Ramakrishnan

PowerPoint Presentation : Sai Prashanth

# Table of Contents

[Certificate 1](#_Toc182910577)

[Acknowledgements 3](#_Toc182910578)

[Table of Contents 5](#_Toc182910579)

[Introduction 6](#_Toc182910580)

[Methodology 6](#_Toc182910581)

[1)Problem 6](#_Toc182910582)

[Abstract 7](#_Toc182910583)

[Solution Design 7](#_Toc182910584)

[Code Explanation 9](#_Toc182910585)

[Code Explanation 22](#_Toc182910586)

[Discussion 24](#_Toc182910587)

[Future Work 24](#_Toc182910588)

[References 24](#_Toc182910589)

# Introduction

In modern-day banking, managing customer accounts, transactions, and services is a critical operation that requires robust and efficient software systems. This project aims to replicate these functionalities through a banking system implemented in C++. By simulating key banking operations, such as account creation, deposits, withdrawals, transfers, and mobile banking management, this program offers insights into the implementation of real-world applications using fundamental OOP principles like encapsulation, inheritance, and polymorphism.

The system is designed to handle both **Savings** and **Current Accounts**, providing specific functionalities such as interest application for savings accounts and overdraft features for current accounts. The modular and reusable code structure makes it an excellent example for academic and practical learning purposes.

# Methodology

# 1)Problem

The problem involves designing a digital banking system that can:

* Create savings and current accounts.
* Perform transactions such as deposits, withdrawals, and transfers.
* Manage account states (active/inactive) and enable/disable mobile banking.
* Apply unique features such as interest for savings accounts and overdraft for current accounts.

# Abstract

The presented program simulates a comprehensive Banking System using object-oriented programming (OOP) concepts in C++. It encompasses the creation and management of savings and current accounts, deposit and withdrawal transactions, account activation/deactivation, fund transfers, interest application, and mobile banking functionalities. The program adopts an extensible architecture utilizing classes and inheritance to simulate real-world banking operations. It emphasizes ease of interaction through a user-friendly menu-driven interface, showcasing the practicality and flexibility of C++ for complex system simulations.

# Solution Design

The program uses OOP to implement a hierarchical class structure:

1. **Base Class**: Account  
   Handles general properties and behaviors of all accounts.
2. **Derived Classes**: SavingsAccount and CurrentAccount  
   Inherit the base class and add specific functionalities.

A Bank class is implemented to manage multiple accounts and provide an interface for various banking operations.

**3. Implementation**

* **User Interaction**: A menu-driven interface allows users to interact with the system.
* **Dynamic Memory Management**: Accounts are stored using pointers, ensuring runtime flexibility.
* **Polymorphism**: Overridden functions like withdraw and displayAccountInfo customize behavior for different account types.

**4. Testing and Validation**

The program is tested with various cases, such as:

* Valid and invalid account transactions.
* Edge scenarios like zero or negative deposits/withdrawals.
* Overdraft checks for current accounts.

# Code Explanation

#include<iostream>

#include<vector>

#include<string>

using namespace std;

class Account {

protected:

string accountHolderName;

int accountNumber;

double balance;

bool isActive;

bool mobileBanking;

public:

Account(string accName, int accNumber, double accBalance) {

accountHolderName = accName;

accountNumber = accNumber;

balance = accBalance;

isActive=true;

mobileBanking=false;

}

virtual void deposit(double amount) {

if(amount<0){

cout << "Deposit amount should be positive" << amount << endl;

return;

}

balance += amount;

cout << "Deposited: ₹" << amount << endl;

}

virtual bool withdraw(double amount) {

if (!isActive) {

cout << "Cannot withdraw from an inactive account!" << endl;

return false;

}

if (amount > balance) {

cout << "Insufficient balance!" << endl;

return false;

}

if(amount<0){

cout << "Withdraw amount should be positive" << amount << endl;

return false;

}

balance -= amount;

cout << "Withdrawn: ₹" << amount << endl;

return true;

}

virtual void displayAccountInfo() const {

cout << "Account Number: " << accountNumber << endl;

cout << "Account Holder: " << accountHolderName << endl;

cout << "Balance: ₹" << balance << endl;

cout << "Account Status: " << (isActive ? "Active" : "Inactive") << endl;

cout << "Mobile Banking: " << (mobileBanking ? "Enabled" : "Not Enabled") << endl;

}

void deactivateAccount() {

if(!isActive){

cout << "Account is already deactive." << endl;

return;

}

isActive = false;

cout << "Account " << accountNumber << " has been deactivated." << endl;

}

void activateAccount() {

if(isActive){

cout << "Account is already Active." << endl;

return;

}

isActive = true;

cout << "Account " << accountNumber << " has been activated." << endl;

}

void enableMobileBanking() {

if (mobileBanking) {

cout << "Mobile banking is already enabled for this account." << endl;

} else {

mobileBanking = true;

cout << "Mobile banking enabled successfully!" << endl;

}

}

void disableMobileBanking() {

if (!mobileBanking) {

cout << "Mobile banking is already disabled for this account." << endl;

} else {

mobileBanking = false;

cout << "Mobile banking disabled successfully!" << endl;

}

}

bool isAccountActive() const {

return isActive;

}

bool isMobileBankingEnabled() const {

return mobileBanking;

}

int getAccountNumber() const {

return accountNumber;

}

double getBalance() const {

return balance;

}

};

class SavingsAccount : public Account {

double interestRate;

public:

SavingsAccount(string name, int accNumber, double initialBalance, double rate)

: Account(name, accNumber, initialBalance), interestRate(rate) {}

void applyInterest() {

balance += balance \* interestRate;

cout << "Interest applied. New Balance: ₹" << balance << endl;

}

};

class CurrentAccount : public Account {

double overdraftLimit;

public:

CurrentAccount(string name, int accNumber, double initialBalance, double limit)

: Account(name, accNumber, initialBalance), overdraftLimit(limit) {}

bool withdraw(double amount) override {

if (!isActive) {

cout << "Cannot withdraw from an inactive account!" << endl;

return false;

}

if (amount > (balance + overdraftLimit)) {

cout << "Exceeds overdraft limit!" << endl;

return false;

}

balance -= amount;

cout << "Withdrawn: ₹" << amount << endl;

return true;

}

void displayAccountInfo() const override {

Account::displayAccountInfo();

cout << "Overdraft Limit: ₹" << overdraftLimit << endl;

}

};

class Bank {

vector<Account\*> accounts;

int AccountNumber;

public:

Bank() : AccountNumber(1001) {}

void createSavingsAccount(string name, double initialBalance, double interestRate) {

SavingsAccount\* newAccount = new SavingsAccount(name, AccountNumber, initialBalance, interestRate);

accounts.push\_back(newAccount);

cout << "Savings account created successfully!" << endl;

cout << "Your Account Number is: " << AccountNumber << endl;

AccountNumber++;

}

void createCurrentAccount(string name, double initialBalance, double overdraftLimit) {

CurrentAccount\* newAccount = new CurrentAccount(name, AccountNumber, initialBalance, overdraftLimit);

accounts.push\_back(newAccount);

cout << "Current account created successfully!" << endl;

cout << "Your Account Number is: " << AccountNumber << endl;

AccountNumber++;

}

void deposit(int accNumber, double amount) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->deposit(amount);

cout << "Updated Balance: ₹" << accounts[i]->getBalance() << endl; // Show updated balance

return;

}

}

cout << "Account not found!" << endl;

}

void withdraw(int accNumber, double amount) {

for (int i = 0; i < accounts.size(); ++i) {

if (accounts[i]->getAccountNumber() == accNumber) {

if (accounts[i]->withdraw(amount)) {

cout << "Updated Balance: ₹" << accounts[i]->getBalance() << endl; // Show updated balance

}

return;

}

}

cout << "Account not found!" << endl;

}

void displayAccountInfo(int accNumber) const {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->displayAccountInfo();

return;

}

}

cout << "Account not found!" << endl;

}

void transfer(int fromAccNumber, int toAccNumber, double amount) {

Account\* fromAccount = nullptr;

Account\* toAccount = nullptr;

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == fromAccNumber) {

fromAccount = accounts[i];

}

if (accounts[i]->getAccountNumber() == toAccNumber) {

toAccount = accounts[i];

}

}

if (!fromAccount || !toAccount) {

cout << "One or both account numbers are invalid!" << endl;

return;

}

if (fromAccount->withdraw(amount)) {

toAccount->deposit(amount);

cout << "Transferred ₹" << amount << " from Account " << fromAccNumber

<< " to Account " << toAccNumber << endl;

cout << "Sender's Updated Balance: ₹" << fromAccount->getBalance() << endl;

cout << "Receiver's Updated Balance: ₹" << toAccount->getBalance() << endl;

}

}

void applyInterest(int accNumber) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

SavingsAccount\* savingsAccount = dynamic\_cast<SavingsAccount\*>(accounts[i]);

if (savingsAccount) {

savingsAccount->applyInterest();

} else {

cout << "Account is not a Savings Account!" << endl;

}

return;

}

}

cout << "Account not found!" << endl;

}

void activateAccount(int accNumber) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->activateAccount();

return;

}

}

cout << "Account not found!" << endl;

}

void deactivateAccount(int accNumber) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->deactivateAccount();

return;

}

}

cout << "Account not found!" << endl;

}

void enableMobileBanking(int accNumber) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->enableMobileBanking();

return;

}

}

cout << "Account not found!" << endl;

}

void disableMobileBanking(int accNumber) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts[i]->getAccountNumber() == accNumber) {

accounts[i]->disableMobileBanking();

return;

}

}

cout << "Account not found!" << endl;

}

};

int main() {

Bank bank;

int choice, accountNumber, fromAcc, toAcc;

double amount, initialBalance, interestRate, overdraftLimit;

string name;

do {

cout << "\nBanking System Menu:\n";

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

cout << "2. Create Current Account\n";

cout << "3. Deposit\n";

cout << "4. Withdraw\n";

cout << "5. Transfer\n";

cout << "6. Display Account Info\n";

cout << "7. Activate Account\n";

cout << "8. Apply Interest\n";

cout << "9. Deactivate Account\n";

cout << "10. Enable Mobile Banking\n";

cout << "11. Disable Mobile Banking\n";

cout << "12. Exit\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter Account Holder Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Initial Balance: ";

cin >> initialBalance;

cout << "Enter Interest Rate (e.g., 0.03 for 3%): ";

cin >> interestRate;

bank.createSavingsAccount(name, initialBalance, interestRate);

break;

case 2:

cout << "Enter Account Holder Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Initial Balance: ";

cin >> initialBalance;

cout << "Enter Overdraft Limit: ";

cin >> overdraftLimit;

bank.createCurrentAccount(name, initialBalance, overdraftLimit);

break;

case 3:

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Deposit Amount: ";

cin >> amount;

bank.deposit(accountNumber, amount);

break;

case 4:

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Withdrawal Amount: ";

cin >> amount;

bank.withdraw(accountNumber, amount);

break;

case 5:

cout << "Enter Sender's Account Number: ";

cin >> fromAcc;

cout << "Enter Receiver's Account Number: ";

cin >> toAcc;

cout << "Enter Transfer Amount: ";

cin >> amount;

bank.transfer(fromAcc, toAcc, amount);

break;

case 6:

cout << "Enter Account Number: ";

cin >> accountNumber;

bank.displayAccountInfo(accountNumber);

break;

case 7:

cout << "Enter Account Number to Activate: ";

cin >> accountNumber;

bank.activateAccount(accountNumber);

break;

case 8:

cout << "Enter Account Number to Apply Interest: ";

cin >> accountNumber;

bank.applyInterest(accountNumber);

break;

case 9:

cout << "Enter Account Number to Deactivate: ";

cin >> accountNumber;

bank.deactivateAccount(accountNumber);

break;

case 10:

cout << "Enter Account Number to Enable Mobile Banking: ";

cin >> accountNumber;

bank.enableMobileBanking(accountNumber);

break;

case 11:

cout << "Enter Account Number to Disable Mobile Banking: ";

cin >> accountNumber;

bank.disableMobileBanking(accountNumber);

break;

case 12:

cout << "Thank You For Banking With Us" << endl << "Exiting..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

break;

}

} while (choice != 12);

return 0;

}

# Code Explanation

**1. Account Class**

This is the base class representing a generic bank account.  
**Key Features:**

* **Attributes:** accountHolderName, accountNumber, balance, isActive, mobileBanking.
* **Methods:**
  + deposit: Adds funds to the account after validation.
  + withdraw: Deducts funds, ensuring sufficient balance and account activity.
  + displayAccountInfo: Displays account details.
  + activateAccount and deactivateAccount: Change account status.
  + enableMobileBanking and disableMobileBanking: Manage mobile banking access.

**2. SavingsAccount Class**

A derived class representing a savings account with an **interest rate** feature.  
**Key Features:**

* **Constructor:** Initializes account details and interest rate.
* applyInterest: Applies the interest rate to the current balance.

**3. CurrentAccount Class**

A derived class representing a current account with an **overdraft limit** feature.  
**Key Features:**

* **Constructor:** Initializes account details and overdraft limit.
* **Overridden Method:**
  + withdraw: Allows withdrawals even if the balance is insufficient, within the overdraft limit.

**4. Bank Class**

This class manages all accounts and provides a unified interface for various operations.  
**Key Features:**

* **Attributes:**
  + accounts: A vector of pointers to Account objects.
  + AccountNumber: Auto-incremented for unique account identification.
* **Methods:**
  + createSavingsAccount and createCurrentAccount: Instantiate respective accounts and store them.
  + deposit and withdraw: Perform transactions after validating the account.
  + transfer: Transfers funds between two accounts.
  + displayAccountInfo: Fetches and displays account details.
  + applyInterest: Specifically applies interest to savings accounts.
  + Account management methods: Activation, deactivation, and mobile banking toggling.

**5. Main Function**

The entry point of the program.  
**Key Features:**

* Presents a **menu-driven interface** for user interactions.
* Captures user inputs and invokes corresponding Bank class methods.
* Ensures seamless navigation through options using a do-while loop.

# Discussion

This project demonstrates the following features:

- Encapsulation and abstraction through private and public access specifiers.

- Inheritance for code reuse and extending functionalities in derived classes.

- Polymorphism through function overriding to customize behaviors for different account types.

# Future Work

The following enhancements can be made to the system:

- Integrating a database to persist account information across sessions.

- Adding a graphical user interface for better user interaction.

- Implementing multi-threading for concurrent transaction handling.

- Adding more account types like Fixed Deposit or Recurring Deposit accounts.

# References

1. GeeksForGeeks
2. Youtube