**SOFTWARE ENGINEERING DEPARTMENT**

# Total Marks: 100

**Obtained Marks:**

**Project Assignment**

**Last date of Submission: 28 May 2025**

# Submitted To: Shakeel Ahmad

**Student Name: Haider Ali,Ehsan Basit ,Zain Ali,Abdurrehman**

# Reg. Number: 4384,4358,4365,4351

**Main Features:  
✅ Add, delete, update, and view accounts  
✅ Search and filter accounts  
✅ Admin and User login  
✅ Input validation  
✅ Inheritance and polymorphism  
✅ Operator overloading for deposits/withdrawals  
✅ File handling for account and transaction data persistence  
✅ Transaction history logging  
✅ Interest calculation  
✅ Encapsulation for secure data access**

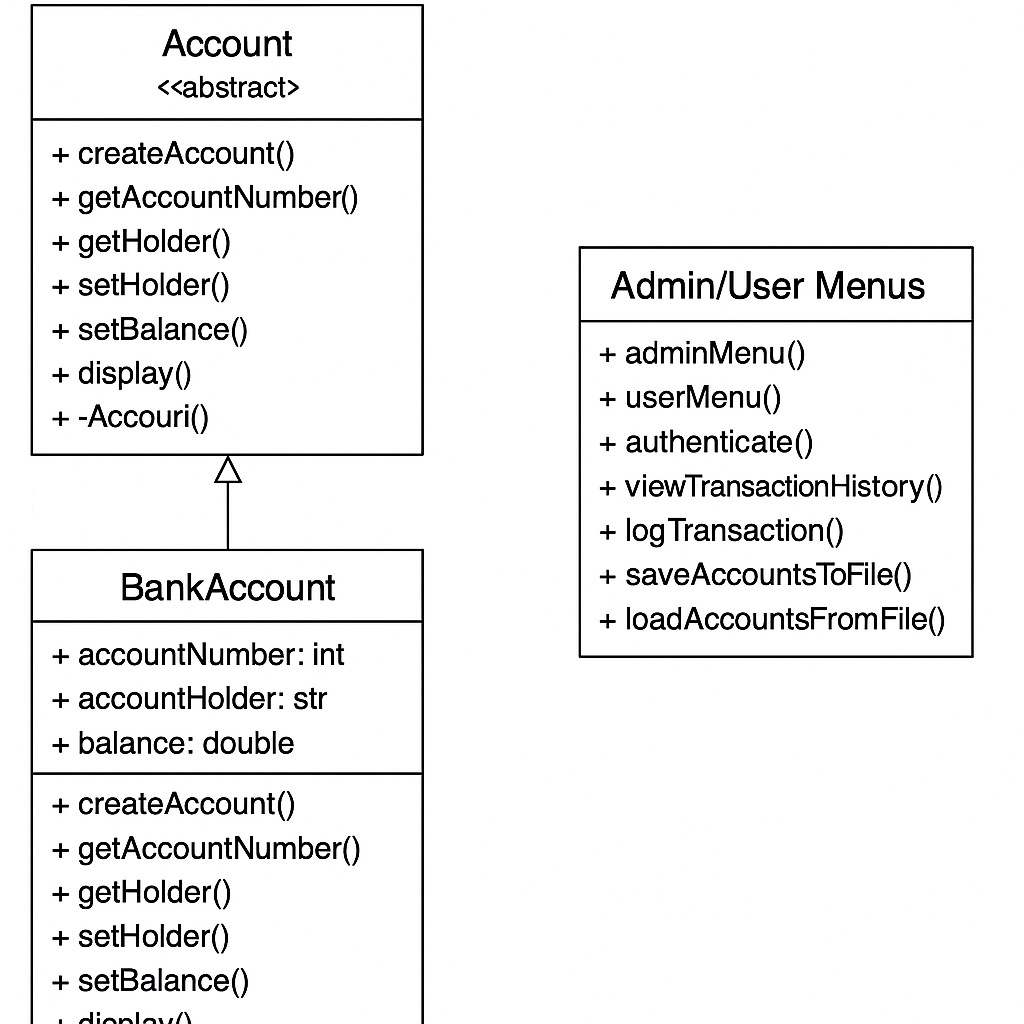
**Key OOP Concepts:**

* **Classes & Objects: Account (abstract base class) and BankAccount (derived class).**
* **Encapsulation: Account data (like balance, accountNumber) are private members with public getters and setters.**
* **Inheritance: BankAccount inherits from Account.**
* **Polymorphism: Virtual functions for dynamic binding (createAccount, display, etc.).**
* **Operator Overloading: Deposit (+) and withdraw (-) operators overloaded for BankAccount.**
* **File Handling: Save/load data from text files (accounts.txt, transactions.txt).**

**Use Cases:**

* **Admin:**
  + **Create, view, search, update, and delete accounts**
  + **View all accounts**
* **User:**
  + **View account, deposit, withdraw, apply interest**
  + **View transaction history**

**UML CASE DIRAGRAM:**





**ONLINE BANKING SYSTEM USING OOP CONCEPT**

**FEATURES OF THIS PROJECT:**

1. **Add Accounts**
2. **Update Accounts.**
3. **View Account.**
4. **Delete Account.**
5. **Search Account.**
6. **Admin Login.**
7. **User Login.**
8. **Input Validation.**
9. **Use File Handling.**
10. **Use inheritance.**
11. **Use Encapsulation**

**PROJECT: ONLINE BANKING SYSTEM**

**STEP 1:**

**ADDING ADD,VIEW,DELETE,UPDATE ACCOUNTS Feature .**

**CODE EXPLANATION:**

**The program displays a menu for the user with options to manage bank accounts and repeatedly prompts the user to enter their choice within a loop. Based on the user's input, it performs different actions: if the user selects option 1, the program calls addAccount() to create a new account; option 2 calls deleteAccount() to remove an account by its account number; option 3 calls updateAccount() to modify the account holder's name or balance; option 4 calls viewAccounts() to display all existing accounts; and option 5 exits the program. If the user enters an invalid choice, the program shows an error message. This menu continues to be displayed and user choices processed until the user selects the "Exit" option.**

**CODE:**

#include <iostream>

#include <string>

using namespace std;

const int MAX\_ACCOUNTS = 100;

class BankAccount {

private:

int accountNumber;

string accountHolder;

double balance;

public:

BankAccount() {

accountNumber = 0;

accountHolder = "";

balance = 0.0;

}

void createAccount(int accNum, string holder, double bal) {

accountNumber = accNum;

accountHolder = holder;

balance = bal;

}

int getAccountNumber() const {

return accountNumber;

}

void setHolder(string holder) {

accountHolder = holder;

}

void setBalance(double bal) {

balance = bal;

}

void display() const {

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

cout << "Holder Name: " << accountHolder << endl;

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

}

};

// Global array and count

BankAccount accounts[MAX\_ACCOUNTS];

int accountCount = 0;

// Add a new account

void addAccount() {

if (accountCount >= MAX\_ACCOUNTS) {

cout << "Cannot add more accounts.\n";

return;

}

int accNum;

string name;

double bal;

cout << "Enter Account Number: ";

cin >> accNum;

cin.ignore();

cout << "Enter Account Holder Name: ";

getline(cin, name);

cout << "Enter Initial Balance: ";

cin >> bal;

accounts[accountCount].createAccount(accNum, name, bal);

accountCount++;

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

}

// Delete an account by number

void deleteAccount() {

int accNum;

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

cin >> accNum;

bool found = false;

for (int i = 0; i < accountCount; ++i) {

if (accounts[i].getAccountNumber() == accNum) {

for (int j = i; j < accountCount - 1; ++j) {

accounts[j] = accounts[j + 1]; // shift left

}

accountCount--;

found = true;

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

break;

}

}

if (!found) {

cout << "Account not found.\n";

}

}

// Update an account

void updateAccount() {

int accNum;

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

cin >> accNum;

for (int i = 0; i < accountCount; ++i) {

if (accounts[i].getAccountNumber() == accNum) {

string newName;

double newBalance;

cin.ignore();

cout << "Enter new Holder Name: ";

getline(cin, newName);

cout << "Enter new Balance: ";

cin >> newBalance;

accounts[i].setHolder(newName);

accounts[i].setBalance(newBalance);

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

return;

}

}

cout << "Account not found.\n";

}// View all accounts

void viewAccounts() {

if (accountCount == 0) {

cout << "No accounts to display.\n";

return; }

for (int i = 0; i < accountCount; ++i) {

cout << "---- Account " << (i + 1) << " ----\n";

accounts[i].display(); }}

int main() {

int choice;

do {

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

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

cout << "2. Delete Account\n";

cout << "3. Update Account\n";

cout << "4. View All Accounts\n";

cout << "5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

addAccount();

break;

case 2:

deleteAccount();

break;

case 3:

updateAccount();

break;

case 4:

viewAccounts();

break;

case 5:

cout << "Exiting...\n";

break;

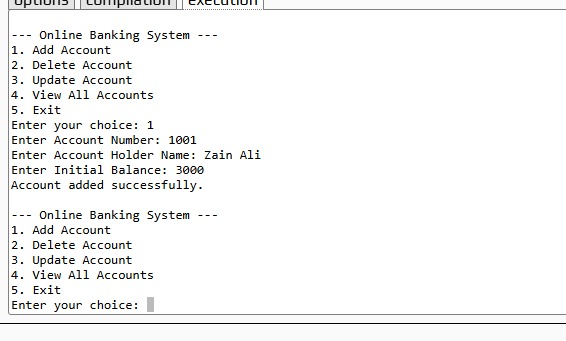
default:

cout << "Invalid choice.\n";

} } while (choice != 5);

return 0;

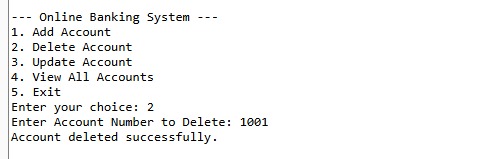
}

* **OUTPUT FOR ADD FUNCTION**

A computer screen shot of a message

AI-generated content may be incorrect.

* **OUTPUT FOR DELETE FUNCTION**



* **OUTPUT FOR UPDATE FUNCTIOn**

A screenshot of a computer

AI-generated content may be incorrect.

* **OUTPUT FOR VIEW FUNCTION**

A white background with black dots

AI-generated content may be incorrect.

**STEP 02:**

**ADDING INHERITANCE AND VITUAL FUNCTION FOR SEEARCH &FILTER ACCOUNT**

**CODE EXPLANATION:**

**The program features a menu-driven interface with six options: the first four allow basic operations—adding, deleting, updating, and viewing accounts—while the fifth introduces a new search and filter feature that enables searching by account number or partial name match. The sixth option allows the user to exit the program. Key improvements over the previous version include the use of polymorphism through an array of base class pointers (Account), which enhances future extensibility. It also implements dynamic memory management by allocating and deallocating BankAccount objects using new and delete. The added search functionality supports partial name matching, making it easier to find accounts. Proper cleanup of allocated memory is ensured before the program exits. The program runs in a loop, continuously displaying the menu and calling the appropriate functions based on user input until the exit option is chosen. Upon exiting, all dynamically allocated objects are deleted. Additionally, the base class includes a virtual destructor to guarantee that derived objects are properly cleaned up, enhancing program safety and preventing memory leaks.**

**CODE:**

#include <iostream>

#include <string>

using namespace std;

const int MAX\_ACCOUNTS = 100;

// Base class

class Account {

public:

virtual void createAccount(int accNum, string holder, double bal) = 0;

virtual void display() const = 0;

virtual int getAccountNumber() const = 0;

virtual string getHolder() const = 0;

virtual void setHolder(string name) = 0;

virtual void setBalance(double bal) = 0;

virtual ~Account() {} // Virtual destructor

};

// Derived class

class BankAccount : public Account {

private:

int accountNumber;

string accountHolder;

double balance;

public:

BankAccount() {

accountNumber = 0;

accountHolder = "";

balance = 0.0;

}

void createAccount(int accNum, string holder, double bal) override {

accountNumber = accNum;

accountHolder = holder;

balance = bal;

}

int getAccountNumber() const override {

return accountNumber;

}

string getHolder() const override {

return accountHolder;

}

void setHolder(string holder) override {

accountHolder = holder;

}

void setBalance(double bal) override {

balance = bal;

}

void display() const override {

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

cout << "Holder Name: " << accountHolder << endl;

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

}

};

// Use array of base class pointers

Account\* accounts[MAX\_ACCOUNTS];

int accountCount = 0;

void addAccount() {

if (accountCount >= MAX\_ACCOUNTS) {

cout << "Account limit reached.\n";

return;

}

int accNum;

string name;

double bal;

cout << "Enter Account Number: ";

cin >> accNum;

cin.ignore();

cout << "Enter Account Holder Name: ";

getline(cin, name);

cout << "Enter Initial Balance: ";

cin >> bal;

accounts[accountCount] = new BankAccount();

accounts[accountCount]->createAccount(accNum, name, bal);

accountCount++;

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

}

void deleteAccount() {

int accNum;

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

cin >> accNum;

for (int i = 0; i < accountCount; ++i) {

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

delete accounts[i];

for (int j = i; j < accountCount - 1; ++j) {

accounts[j] = accounts[j + 1];

}

accountCount--;

cout << "Account deleted.\n";

return;

}

}

cout << "Account not found.\n";

}

void updateAccount() {

int accNum;

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

cin >> accNum;

for (int i = 0; i < accountCount; ++i) {

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

string name;

double bal;

cin.ignore();

cout << "Enter new Holder Name: ";

getline(cin, name);

cout << "Enter new Balance: ";

cin >> bal;

accounts[i]->setHolder(name);

accounts[i]->setBalance(bal);

cout << "Account updated.\n";

return;

}

}

cout << "Account not found.\n";

}

void viewAccounts() {

if (accountCount == 0) {

cout << "No accounts available.\n";

return;

}

for (int i = 0; i < accountCount; ++i) {

cout << "--- Account " << i + 1 << " ---\n";

accounts[i]->display();

}

}

void searchAccount() {

int choice;

cout << "\nSearch By:\n";

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

cout << "2. Account Holder Name\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

int accNum;

cout << "Enter Account Number: ";

cin >> accNum;

for (int i = 0; i < accountCount; ++i) {

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

accounts[i]->display();

return;

}

}

cout << "Account not found.\n";

}

else if (choice == 2) {

string name;

cin.ignore();

cout << "Enter part of Account Holder Name: ";

getline(cin, name);

bool found = false;

for (int i = 0; i < accountCount; ++i) {

if (accounts[i]->getHolder().find(name) != string::npos) {

accounts[i]->display();

cout << "----------------------\n";

found = true;

}

}

if (!found) cout << "No matching records found.\n";

}

else {

cout << "Invalid choice.\n";

}

}

int main() {

int choice;

do {

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

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

cout << "2. Delete Account\n";

cout << "3. Update Account\n";

cout << "4. View All Accounts\n";

cout << "5. Search / Filter Account\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: addAccount(); break;

case 2: deleteAccount(); break;

case 3: updateAccount(); break;

case 4: viewAccounts(); break;

case 5: searchAccount(); break;

case 6: cout << "Exiting...\n"; break;

default: cout << "Invalid choice.\n";

}

} while (choice != 6);

for (int i = 0; i < accountCount; ++i) {

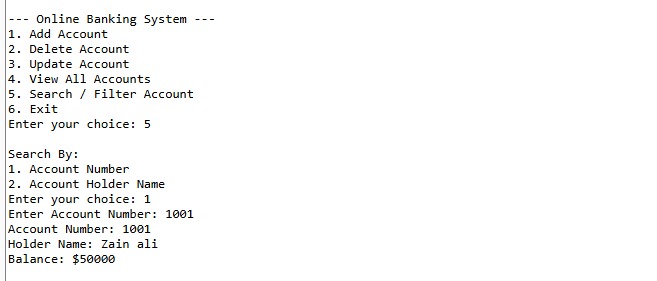
delete accounts[i];

}

return 0;

}

**OUTPUT SCREEN:**



**STEP 03:**

**ADDING All the features:**

**CODE EXPLANATION:**

**The main function begins by loading account data from a file at the program start using the loadAccountsFromFile() function. It then displays the main menu with three options: Admin Login, User Login, and Exit. The program takes user input for the menu choice, ensuring input validation. Depending on the user's selection, it handles authentication by verifying credentials. If the user chooses Admin Login and the credentials are correct, the program calls the adminMenu() function. Similarly, if the user selects User Login with valid credentials, it calls the userMenu() function. The menu continues to be displayed and processed in a loop until the user selects the Exit option. Upon exiting, the program deletes all dynamically allocated account objects to free memory and finally displays an exit message before terminating.**

**CODE:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

const int MAX\_ACCOUNTS = 100;

const double INTEREST\_RATE = 0.05;

// Input validation part

int getValidatedInt(string prompt)

{

int val;

while (true)

{

cout << prompt;

cin >> val;

if (!cin.fail()) break;

cout << "Invalid input. Try again.\n";

cin.clear();

cin.ignore(1000, '\n');

}

return val;

}

double getValidatedDouble(string prompt)

{

double val;

while (true)

{

cout << prompt;

cin >> val;

if (!cin.fail() && val >= 0) break;

cout << "Invalid input. Try again.\n";

cin.clear();

cin.ignore(1000, '\n');

}

return val;

}

// Abstract base class

class Account

{

public:

virtual void createAccount(int accNum, string holder, double bal) = 0;

virtual int getAccountNumber() const = 0;

virtual string getHolder() const = 0;

virtual void setHolder(string holder) = 0;

virtual void setBalance(double bal) = 0;

virtual void display() const = 0;

virtual ~Account() {}

};

// Derived class

class BankAccount : public Account

{

private:

int accountNumber;

string accountHolder;

double balance;

public:

BankAccount()

{

accountNumber = 0;

accountHolder = "";

balance = 0.0;

}

void createAccount(int accNum, string holder, double bal) override

{

accountNumber = accNum;

accountHolder = holder;

balance = bal;

}

int getAccountNumber() const override

{

return accountNumber;

}

string getHolder() const override

{

return accountHolder;

}

void setHolder(string holder) override

{

accountHolder = holder;

}

void setBalance(double bal) override

{

balance = bal;

}

double getBalance() const

{

return balance;

}

void display() const override

{

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

cout << "Holder Name: " << accountHolder << endl;

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

}

BankAccount operator+(double amount)

{

BankAccount temp = \*this;

temp.balance += amount;

return temp;

}

BankAccount operator-(double amount)

{

BankAccount temp = \*this;

if (amount > temp.balance)

{

cout << "Insufficient balance.\n";

} else

{

temp.balance -= amount;

}

return temp;

}

friend void printSummary(const BankAccount& acc);

void saveToFile(ofstream& out)

{

out << accountNumber << "," << accountHolder << "," << balance << "\n";

}

void loadFromFile(int accNum, string holder, double bal)

{

accountNumber = accNum;

accountHolder = holder;

balance = bal;

}

};

void printSummary(const BankAccount& acc)

{

cout << "\n--- Account Summary ---\n";

cout << "Account: " << acc.accountNumber << "\n";

cout << "Holder: " << acc.accountHolder << "\n";

cout << "Balance: $" << acc.balance << "\n";

cout << "------------------------\n";

}

BankAccount\* accounts[MAX\_ACCOUNTS];

int accountCount = 0;

void saveAccountsToFile()

{

ofstream out("accounts.txt");

for (int i = 0; i < accountCount; ++i)

{

accounts[i]->saveToFile(out);

}

out.close();

}

void loadAccountsFromFile()

{

ifstream in("accounts.txt");

accountCount = 0;

int accNum;

string name;

double bal;

string line;

while (getline(in, line))

{

size\_t pos1 = line.find(',');

size\_t pos2 = line.rfind(',');

if (pos1 == string::npos || pos2 == string::npos || pos1 == pos2) continue;

accNum = stoi(line.substr(0, pos1));

name = line.substr(pos1 + 1, pos2 - pos1 - 1);

bal = stod(line.substr(pos2 + 1));

accounts[accountCount] = new BankAccount();

accounts[accountCount]->createAccount(accNum, name, bal);

accountCount++;

}

in.close();

}

void logTransaction(int accNum, const string& type, double amount)

{

ofstream out("transactions.txt", ios::app);

if (!out) return;

out << accNum << "," << type << "," << amount << "\n";

out.close();

}

void viewTransactionHistory(int accNum)

{

ifstream in("transactions.txt");

if (!in) {

cout << "No transactions found.\n";

return;

}

cout << "\n--- Transaction History for Account #" << accNum << " ---\n";

string line;

bool found = false;

while (getline(in, line)) {

size\_t pos1 = line.find(',');

if (pos1 == string::npos) continue;

int acc = stoi(line.substr(0, pos1));

if (acc == accNum) {

string rest = line.substr(pos1 + 1);

cout << rest << "\n";

found = true;

}

}

if (!found) cout << "No transactions for this account.\n";

cout << "-----------------------------------------\n";

in.close();

}

void adminMenu()

{

int choice;

do

{

cout << "\n--- Admin Menu ---\n";

cout << "1. Create Account" << endl;

cout << "2. View All Accounts" << endl;

cout << "3. Search by Account Number" << endl;

cout << "4. Update Account" << endl;

cout << "5. Delete Account" << endl;

cout << "6. Back to Main Menu" << endl;

choice = getValidatedInt("Enter choice: ");

if (choice == 1)

{

int acc = getValidatedInt("Enter Account Number: ");

string name;

cout << "Enter Holder Name: ";

cin >> ws;

getline(cin, name);

double bal = getValidatedDouble("Enter Initial Balance: ");

accounts[accountCount] = new BankAccount();

accounts[accountCount]->createAccount(acc, name, bal);

accountCount++;

saveAccountsToFile();

logTransaction(acc, "Account Created", bal);

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

}

else if (choice == 2)

{

for (int i = 0; i < accountCount; ++i) accounts[i]->display();

}

else if (choice == 3)

{

int num = getValidatedInt("Enter Account Number to Search: ");

bool found = false;

for (int i = 0; i < accountCount; ++i)

{

if (accounts[i]->getAccountNumber() == num)

{

accounts[i]->display();

found = true;

}

}

if (!found) cout << "Account not found.\n";

}

else if (choice == 4)

{

int num = getValidatedInt("Enter Account Number to Update: ");

bool updated = false;

for (int i = 0; i < accountCount; ++i)

{

if (accounts[i]->getAccountNumber() == num)

{

string name;

cout << "Enter new name: ";

cin >> ws;

getline(cin, name);

accounts[i]->setHolder(name);

saveAccountsToFile();

cout << "Account updated.\n";

updated = true;

break;

}

}

if (!updated) cout << "Account not found.\n";

}

else if (choice == 5)

{

int num = getValidatedInt("Enter Account Number to Delete: ");

bool deleted = false;

for (int i = 0; i < accountCount; ++i)

{

if (accounts[i]->getAccountNumber() == num)

{

delete accounts[i];

for (int j = i; j < accountCount - 1; ++j)

{

accounts[j] = accounts[j + 1];

}

accountCount--;

saveAccountsToFile();

cout << "Account deleted.\n";

deleted = true;

break;

}

}

if (!deleted) cout << "Account not found.\n";

}

}

while (choice != 6);

}

void userMenu()

{

int num = getValidatedInt("Enter your Account Number: ");

for (int i = 0; i < accountCount; ++i)

{

if (accounts[i]->getAccountNumber() == num)

{

BankAccount\* acc = dynamic\_cast<BankAccount\*>(accounts[i]);

int choice;

do

{

cout << "\n--- User Menu ---\n";

cout << "1. View Account" << endl;

cout << "2. Deposit" << endl;

cout << "3. Withdraw" << endl;

cout << "4. View Summary" << endl;

cout << "5. Apply Interest" << endl;

cout << "6. View Transaction History" << endl;

cout << "7. Exit" << endl;

choice = getValidatedInt("Enter choice: ");

if (choice == 1)

{

acc->display();

}

else if (choice == 2)

{

double amt = getValidatedDouble("Enter amount to deposit: ");

\*acc = \*acc + amt;

logTransaction(num, "Deposit", amt);

saveAccountsToFile();

}

else if (choice == 3)

{

double amt = getValidatedDouble("Enter amount to withdraw: ");

if (amt <= acc->getBalance()) {

\*acc = \*acc - amt;

logTransaction(num, "Withdraw", amt);

saveAccountsToFile();

}

}

else if (choice == 4)

{

printSummary(\*acc);

}

else if (choice == 5)

{ // Apply interest

double interest = acc->getBalance() \* INTEREST\_RATE;

\*acc = \*acc + interest;

cout << "Interest applied at " << (INTEREST\_RATE \* 100) << "%.\n";

logTransaction(num, "Interest Applied", interest);

saveAccountsToFile();

}

else if (choice == 6)

{

viewTransactionHistory(num);

}

}

while (choice != 7);

return;

}

}

cout << "Account not found.\n";

}

bool authenticate(string role)

{

string user, pass;

cout << "Enter username: "; cin >> user;

cout << "Enter password: "; cin >> pass;

if (role == "admin")

{

if (user == "Zain" && pass == "4365")

{

return true;

}

}

else if (role == "user")

{

if ((user == "Haider" && pass == "4384") ||

(user == "Rehman" && pass == "4351") ||

(user == "Ehsan" && pass == "4358"))

{

return true;

}

}

cout << "Ivalid credentials.\n";

return false;

}

int main()

{

loadAccountsFromFile();

int choice;

do

{

cout << "\n=== Online Banking System ===\n";

cout << "1. Admin Login" << endl;

cout << "2. User Login" << endl;

cout << "3. Exit" << endl;

choice = getValidatedInt("Enter your choice: ");

if (choice == 1 && authenticate("admin")) adminMenu();

else if (choice == 2 && authenticate("user")) userMenu();

}

while (choice != 3);

for (int i = 0; i < accountCount; ++i) delete accounts[i];

cout << "\nThank you for using the system.\n";

return 0;

}

* **OUTPUT SCREEN:**
* **Output for Withdraw Features:**



* **Output for Deposit Feature**A black and white screen

  AI-generated content may be incorrect.:
* **Output for Transaction History Feature:**

A close-up of a computer screen

AI-generated content may be incorrect.

* **Data Save In Text File:**

**A screenshot of a computer program

AI-generated content may be incorrect.**