#include <iostream>

#include <string>

#include <stack>

#include <queue>

#include <cmath>

#include <vector>

#include <iomanip>

using namespace std;

// Customer Node (Linked List)

struct CustomerNode {

string name;

int accountNumber;

double balance;

string address;

string phone;

CustomerNode\* next;

};

// Linked List for customer storage

class LinkedList {

private:

CustomerNode\* head;

public:

LinkedList() {

head = nullptr;

}

// Add customer to the linked list

void addCustomer(string name, int accountNumber, double balance, string address, string phone) {

CustomerNode\* newCustomer = new CustomerNode;

newCustomer->name = name;

newCustomer->accountNumber = accountNumber;

newCustomer->balance = balance;

newCustomer->address = address;

newCustomer->phone = phone;

newCustomer->next = nullptr;

if (head == nullptr) {

head = newCustomer;

} else {

CustomerNode\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newCustomer;

}

}

// Display all customers

void displayCustomers() {

CustomerNode\* temp = head;

while (temp != nullptr) {

cout << "Name: " << temp->name << ", Account No: " << temp->accountNumber

<< ", Balance: $" << temp->balance << ", Address: " << temp->address

<< ", Phone: " << temp->phone << endl;

temp = temp->next;

}

}

// Update balance for a customer

void updateBalance(int accountNumber, double amount) {

CustomerNode\* temp = head;

while (temp != nullptr) {

if (temp->accountNumber == accountNumber) {

temp->balance += amount;

cout << "Balance updated successfully! New Balance: $" << temp->balance << endl;

return;

}

temp = temp->next;

}

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

}

};

// Queue for managing customers in line

class Queue {

private:

queue<int> customerQueue;

public:

void addCustomerToQueue(int accountNumber) {

customerQueue.push(accountNumber);

}

int processCustomer() {

if (!customerQueue.empty()) {

int accountNumber = customerQueue.front();

customerQueue.pop();

return accountNumber;

}

return -1;

}

};

// Stack for transaction history

class Stack {

private:

stack<string> transactionHistory;

public:

void pushTransaction(string transaction) {

transactionHistory.push(transaction);

}

void displayHistory() {

stack<string> temp = transactionHistory;

while (!temp.empty()) {

cout << temp.top() << endl;

temp.pop();

}

}

};

// Binary Search Tree for account management

struct TreeNode {

int accountNumber;

double balance;

TreeNode\* left;

TreeNode\* right;

};

class BST {

private:

TreeNode\* root;

TreeNode\* insert(TreeNode\* root, int accountNumber, double balance) {

if (root == nullptr) {

TreeNode\* newNode = new TreeNode;

newNode->accountNumber = accountNumber;

newNode->balance = balance;

newNode->left = newNode->right = nullptr;

return newNode;

}

if (accountNumber < root->accountNumber) {

root->left = insert(root->left, accountNumber, balance);

} else if (accountNumber > root->accountNumber) {

root->right = insert(root->right, accountNumber, balance);

}

return root;

}

public:

BST() {

root = nullptr;

}

void insert(int accountNumber, double balance) {

root = insert(root, accountNumber, balance);

}

void displayAccounts(TreeNode\* root) {

if (root != nullptr) {

displayAccounts(root->left);

cout << "Account No: " << root->accountNumber << ", Balance: $" << root->balance << endl;

displayAccounts(root->right);

}

}

void displayAll() {

displayAccounts(root);

}

};

// Loan Management Class

class LoanManagement {

public:

void applyForLoan(int accountNumber, double amount) {

cout << "Loan application submitted for account " << accountNumber << " for amount: $" << amount << endl;

}

void repayLoan(int accountNumber, double amount) {

cout << "Loan repayment of $" << amount << " made for account " << accountNumber << endl;

}

};

// Card Management Class

class CardManagement {

public:

void issueCard(int accountNumber, const string& cardType) {

cout << cardType << " card issued for account " << accountNumber << endl;

}

void blockCard(int accountNumber, const string& cardType) {

cout << cardType << " card blocked for account " << accountNumber << endl;

}

};

// Notifications Class

class Notifications {

public:

void sendNotification(const string& message) {

cout << "Notification: " << message << endl;

}

};

// Transaction Class

class Transaction {

public:

void deposit(int accountNumber, double amount, LinkedList& customers, Stack& transactionStack) {

customers.updateBalance(accountNumber, amount);

transactionStack.pushTransaction("Deposited $" + to\_string(amount) + " to Account " + to\_string(accountNumber));

}

void withdraw(int accountNumber, double amount, LinkedList& customers, Stack& transactionStack) {

customers.updateBalance(accountNumber, -amount);

transactionStack.pushTransaction("Withdrew $" + to\_string(amount) + " from Account " + to\_string(accountNumber));

}

};

// Main Function

int main() {

LinkedList customers;

Queue customerQueue;

Stack transactionStack;

BST accountTree;

LoanManagement loanManager;

CardManagement cardManager;

Notifications notifier;

Transaction transactionManager;

int choice;

do {

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

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

cout << "2. Display All Customers\n";

cout << "3. Apply for Loan\n";

cout << "4. Repay Loan\n";

cout << "5. Issue Card\n";

cout << "6. Block Card\n";

cout << "7. Send Notification\n";

cout << "8. Display All Accounts\n";

cout << "9. Deposit Amount\n";

cout << "10. Withdraw Amount\n";

cout << "11. Display Transaction History\n";

cout << "12. Exit\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1: {

string name, address, phone;

int accountNumber;

double balance;

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Balance: ";

cin >> balance;

cout << "Enter Address: ";

cin.ignore();

getline(cin, address);

cout << "Enter Phone: ";

getline(cin, phone);

customers.addCustomer(name, accountNumber, balance, address, phone);

break;

}

case 2:

customers.displayCustomers();

break;

case 3: {

int accountNumber;

double amount;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Loan Amount: ";

cin >> amount;

loanManager.applyForLoan(accountNumber, amount);

break;

}

case 4: {

int accountNumber;

double amount;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Repayment Amount: ";

cin >> amount;

loanManager.repayLoan(accountNumber, amount);

break;

}

case 5: {

int accountNumber;

string cardType;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Card Type (Credit/Debit): ";

cin >> cardType;

cardManager.issueCard(accountNumber, cardType);

break;

}

case 6: {

int accountNumber;

string cardType;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Card Type (Credit/Debit): ";

cin >> cardType;

cardManager.blockCard(accountNumber, cardType);

break;

}

case 7: {

string message;

cout << "Enter Notification Message: ";

cin.ignore();

getline(cin, message);

notifier.sendNotification(message);

break;

}

case 8:

accountTree.displayAll();

break;

case 9: {

int accountNumber;

double amount;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Deposit Amount: ";

cin >> amount;

transactionManager.deposit(accountNumber, amount, customers, transactionStack);

break;

}

case 10: {

int accountNumber;

double amount;

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Withdraw Amount: ";

cin >> amount;

transactionManager.withdraw(accountNumber, amount, customers, transactionStack);

break;

}

case 11:

transactionStack.displayHistory();

break;

case 12:

cout << "Exiting the system. Goodbye!" << endl;

break;

default:

cout << "Invalid option! Please try again." << endl;

}

} while (choice != 12);

return 0;

}