**OOP Final Examination**

**Name:** Hamza Irshad

**Roll #:** 232464

**CODE**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

class Customer {

private:

int id;

string name;

string email;

string phone;

string status;

string transactions;

public:

Customer() : id(0), status("inactive") {}

bool loadFromStream(stringstream &ss) {

string item;

try {

getline(ss, item, ',');

id = stoi(item);

getline(ss, name, ',');

getline(ss, email, ',');

getline(ss, phone, ',');

getline(ss, status, ',');

getline(ss, transactions);

} catch (const invalid\_argument &e) {

cerr << "Error: Invalid data format in file." << endl;

return false;

} catch (const out\_of\_range &e) {

cerr << "Error: Integer value out of range in file." << endl;

return false;

}

return true;

}

void saveToStream(ofstream &file) const {

file << id << "," << name << "," << email << "," << phone << "," << status << "," << transactions << endl;

}

void addTransaction(int amount) {

if (!transactions.empty()) {

transactions += ",";

}

transactions += to\_string(amount);

}

void displayTransactions() const {

stringstream ss(transactions);

string transaction;

while (getline(ss, transaction, ',')) {

cout << transaction << " ";

}

cout << endl;

}

void inputNewCustomer(int newId) {

id = newId;

cout << "Enter name: ";

cin.ignore();

getline(cin, name);

cout << "Enter email: ";

getline(cin, email);

cout << "Enter phone: ";

getline(cin, phone);

status = "active";

transactions = "";

}

int getId() const {

return id;

}

string getStatus() const {

return status;

}

void setStatus(const string &newStatus) {

status = newStatus;

}

string getName() const {

return name;

}

};

class Admin {

private:

static const int MAX\_CUSTOMERS = 100;

Customer customers[MAX\_CUSTOMERS];

int customerCount;

Customer\* findCustomerById(int id) {

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

if (customers[i].getId() == id) {

return &customers[i];

}

}

return nullptr;

}

public:

Admin() : customerCount(0) {}

void loadCustomers() {

ifstream file("customers.txt");

string line;

customerCount = 0;

while (getline(file, line) && customerCount < MAX\_CUSTOMERS) {

stringstream ss(line);

if (customers[customerCount].loadFromStream(ss)) {

customerCount++;

} else {

cerr << "Skipping invalid customer data: " << line << endl;

}

}

file.close();

}

void saveCustomers() {

ofstream file("customers.txt");

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

customers[i].saveToStream(file);

}

file.close();

}

void addCustomer() {

if (customerCount >= MAX\_CUSTOMERS) {

cout << "Customer limit reached. Cannot add more customers.\n";

return;

}

int newId;

cout << "Enter ID: ";

cin >> newId;

customers[customerCount].inputNewCustomer(newId);

customerCount++;

saveCustomers();

}

void updateCustomerStatus() {

int id;

cout << "Enter customer ID: ";

cin >> id;

Customer\* customer = findCustomerById(id);

if (customer) {

string status;

cout << "Enter new status (active/inactive): ";

cin >> status;

customer->setStatus(status);

saveCustomers();

} else {

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

}

}

void makeTransaction() {

int id, amount;

cout << "Enter customer ID: ";

cin >> id;

cout << "Enter amount (positive for credit, negative for debit): ";

cin >> amount;

Customer\* customer = findCustomerById(id);

if (customer) {

customer->addTransaction(amount);

saveCustomers();

} else {

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

}

}

void displayTransactionHistory() {

int id;

cout << "Enter customer ID: ";

cin >> id;

Customer\* customer = findCustomerById(id);

if (customer) {

cout << "Transaction history for " << customer->getName() << ":\n";

customer->displayTransactions();

} else {

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

}

}

void transferMoney() {

int fromId, toId, amount;

cout << "Enter source customer ID: ";

cin >> fromId;

cout << "Enter destination customer ID: ";

cin >> toId;

cout << "Enter amount to transfer: ";

cin >> amount;

Customer\* fromCustomer = findCustomerById(fromId);

Customer\* toCustomer = findCustomerById(toId);

if (fromCustomer && toCustomer) {

fromCustomer->addTransaction(-amount);

toCustomer->addTransaction(amount);

saveCustomers();

} else {

cout << "One or both customers not found.\n";

}

}

void adminMenu() {

int choice;

do {

cout << "\nAdmin Menu:\n";

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

cout << "2. Update Customer Status\n";

cout << "3. Make a Transaction\n";

cout << "4. Display Transaction History\n";

cout << "5. Transfer Money\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

addCustomer();

break;

case 2:

updateCustomerStatus();

break;

case 3:

makeTransaction();

break;

case 4:

displayTransactionHistory();

break;

case 5:

transferMoney();

break;

case 6:

break;

default:

cout << "Invalid choice. Please try again.\n";

}

} while (choice != 6);

}

};

int main() {

Admin admin;

admin.loadCustomers();

admin.adminMenu();

return 0;

}

**OUTPUT**



