Small Clinic Management System – Documentation

Name: Ly Tran Gia Khang

ID: 24110098

---

1. Object-Oriented Analysis (OOA)

Following the 4-step OOA model, the system has the following objects:

1.1 Objects (Nouns)

- Patient
- ChronicPatient (inherits from Patient)
- Doctor
- Appointment
- Clinic

1.2 Attributes (Descriptive Nouns)

- Patient: name, ID, age, medicalHistory
- ChronicPatient: (inherits Patient) + conditionType, lastCheckup
- Doctor: name, ID, specialty
- Appointment: appID, date, time, reason, status, doctor, patient
- Clinic: list of patients, list of doctors, list of appointments

1.3 Methods (Verbs)

- Patient: displayInfo(), addHistory(), removeHistory(), showHistory(), virtual scheduleAppointment()
- ChronicPatient: override scheduleAppointment()
- Doctor: displayInfo(), getters/setters

- Appointment: cancel(), complete(), displayInfo()
- Clinic: addPatient(), addDoctor(), addAppointment(), display lists, choosePatient(), chooseDoctor(), chooseAppointment()

1.4 Inheritance Relationships

- ChronicPatient : public Patient
- Inherits all attributes and methods from Patient.
- Overrides scheduleAppointment() to enforce frequent check-ups and add notes in the history.

2. Class Design Explanation

The system is designed around five main classes:

- Doctor: stores doctor information, includes getters/setters and displayInfo().
- Patient: manages patient data and medical history. Includes a virtual method scheduleAppointment().
- ChronicPatient: extends Patient with chronic condition attributes. Overrides scheduleAppointment() to highlight condition-specific appointments.
- Appointment: represents a link between Doctor and Patient, containing date, time, reason, and status (Scheduled, Cancelled, Completed).
- Clinic: central manager of patients, doctors, and appointments. Provides methods for adding, displaying, and selecting objects.

Why inheritance?

- Patient serves as the base class for all patients.
- ChronicPatient adds specific attributes (conditionType, lastCheckup).
- scheduleAppointment() is virtual in Patient and overridden in ChronicPatient → demonstrates polymorphism.

3. Code Walkthrough

Example 1 – Override in ChronicPatient:

```
void scheduleAppointment(string appID, string date, string time,
string reason) override{
    cout << "Chronic patient requires regular check-up...!" <<
endl;
    cout << "Appointment set on " << date << " at " << time
         << " for " << reason << " (Condition: " << conditionType
<< ")." << endl;
    addHistory(date + " - " + time + ": " + reason + "
[Chronic]");
}
```

-> Chronic patients log their appointments differently, marking them with [Chronic].

Example 2 – Clinic manages all objects:

```
void addPatient(Patient *_patient){patients.push_back(_patient);}
void addDoctor(Doctor *_doctor){doctors.push_back(_doctor);}
void addAppointment(Appointment
*_appointment){appointments.push_back(_appointment);}
```

-> The Clinic class uses vectors to store patients, doctors, and appointments, providing centralized management.

4. Testing & Output

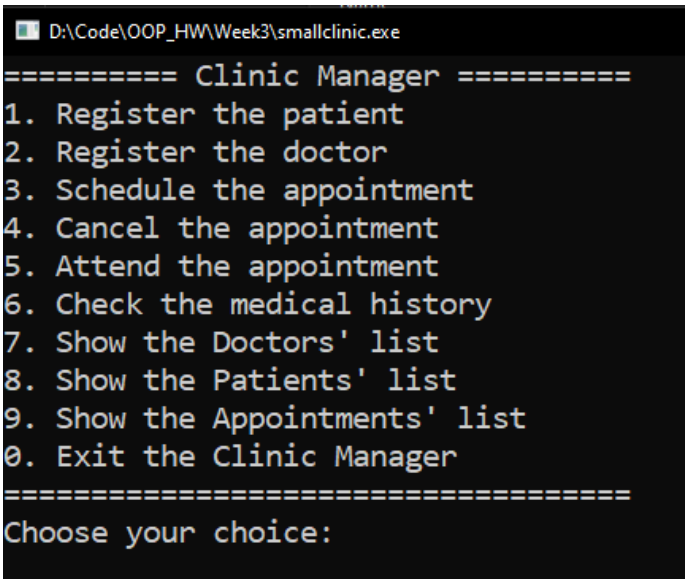
4.1 Test Cases

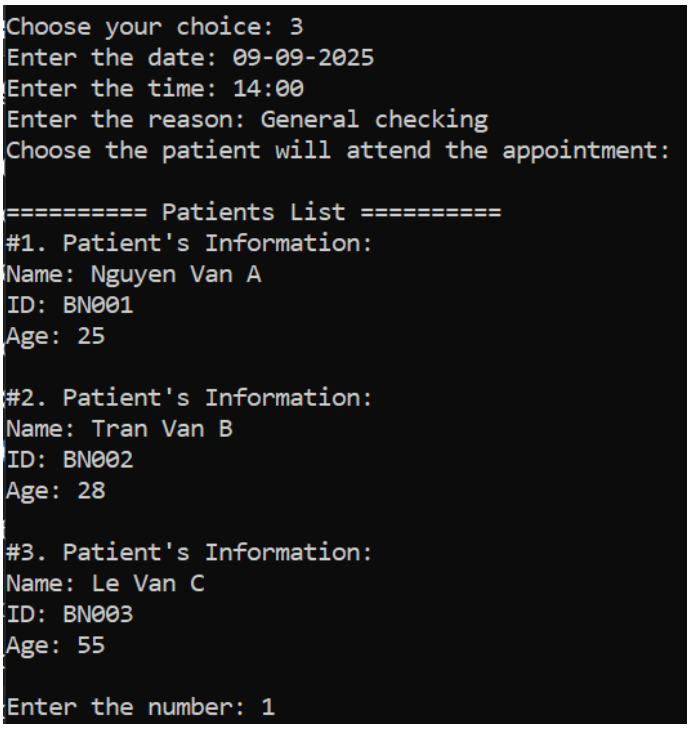
The main() function demonstrates the following:

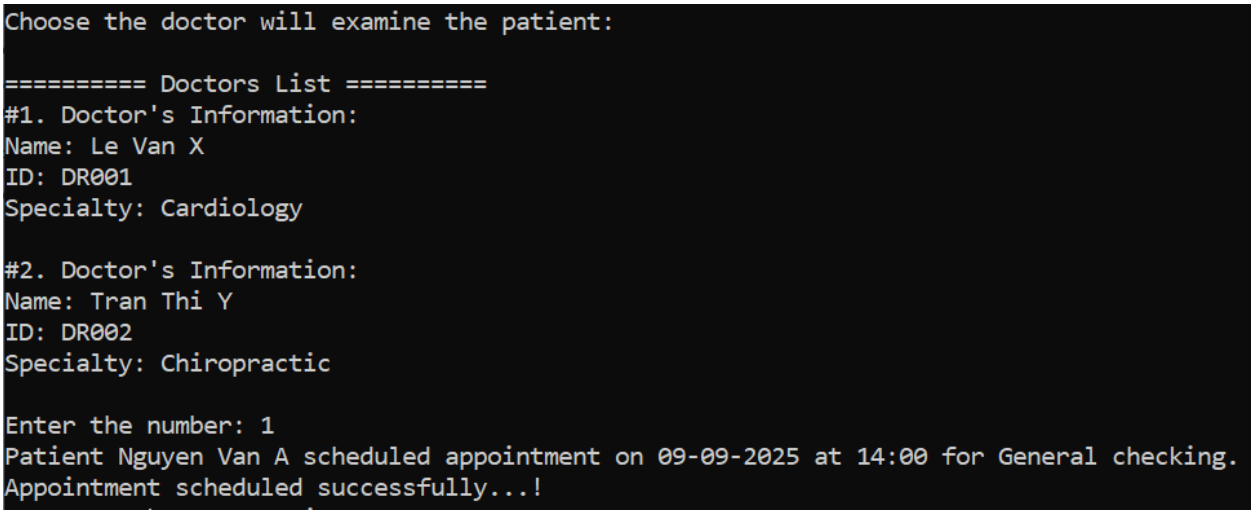
- Registering regular and chronic patients.
- Registering doctors.

- Scheduling appointments.
- Canceling appointments.
- Completing appointments.
- Viewing patient medical history.
- Displaying lists of doctors, patients, and appointments.

4.2 Sample Output (excerpt)







-> Output proves the system works for both regular and chronic patients.

5. Use of LLM (ChatGPT)

During development, I used ChatGPT as a support tool, not as a replacement.

How I used it:

- Brainstorming possible methods for the Appointment class.
- Asking for clarification on how to override a virtual method in C++.
- Getting sample ideas for menu-driven programs.

Example Prompt:

"Suggest methods for an Appointment class in a small clinic management system."

Response:

ChatGPT suggested methods like cancel(), complete(), displayInfo(). I adapted these to my final design.

Note: The final code and report are my own work. The LLM was used only for brainstorming and debugging support.

6. Conclusion

The Small Clinic Management System successfully demonstrates key OOP concepts:

- Encapsulation: private attributes with getters/setters.

- Inheritance & Polymorphism: ChronicPatient inherits from Patient and overrides scheduleAppointment().
- Abstraction: clear separation into five classes with dedicated responsibilities.

The program compiles and runs without errors, and testing confirms it supports all required operations. This is a practical application of OOP for real-world clinic management.