Small Clinic Management System − Documentation

1. OOA Analysis (Step 1)

Dựa theo mô hình OOA 4 bước, hệ thống được phân tích như sau:

1.1 Objects

Patient — bệnh nhân chung.

ChronicPatient — bệnh nhân mãn tính.

Regular — bệnh nhân thường.

Doctor — bác sĩ.

Appointment — lịch hẹn.

(Tuỳ chọn) **Clinic/Scheduler** — quản lý tổng thể.

1.2 Attributes

Patient: name, age, ID, medicalHistory, appointments.

ChronicPatient: condition, lastCheckupDate.

Regular: lastVisitDate.

Doctor: name, specialization, appointments.

Appointment: date, time, reason, status, patient, doctor.

1.3 Methods

Patient: displayInfo(), addAppointment(), removeAppointment(), updateMedicalHistory().

ChronicPatient: override displayInfo(), lastCheckup().

Regular: override displayInfo(), scheduleCheckup().

Doctor: displayInfo(), addAppointment(), showAppointments().

Appointment: displayAppointment(), cancelAppointment(), completeAppointment(), rescheduleAppointment().

1.4 Inheritance & Relationships

Patient là lớp cha, ChronicPatient và Regular kế thừa từ Patient.

Patient 1–* Appointment.

Doctor 1–* Appointment.

Appointment liên kết **1 Patient** và **1 Doctor**.

---

2. Class Design & Inheritance

Trong thiết kế C++:

Patient là lớp cơ sở, cung cấp thuộc tính và hành vi chung cho tất cả bệnh nhân.

ChronicPatient và **Regular** kế thừa Patient, override phương thức displayInfo() để bổ sung thông tin riêng.

Doctor quản lý danh sách lịch hẹn riêng.

Appointment đóng vai trò "cầu nối" (association) giữa Patient và Doctor.

Lý do sử dụng inheritance:

Tránh trùng lặp code (reusability).

Giúp hệ thống dễ mở rộng: chỉ cần tạo subclass mới cho từng loại bệnh nhân khác nhau.

Tận dụng polymorphism (đa hình) để xử lý danh sách bệnh nhân hỗn hợp.

---

3. Code Walkthrough

3.1 Patient & Subclasses

```
class Patient {protected:
    string name;
    int age;
    string ID;
    string medicalHistory;
    vector<string> Appointments; // (đơn giản hóa)public:
    virtual void displayInfo();
    void addAppointment(string appt);
    void updateMedicalHistory(string entry);
};
```

Thuộc tính chung: name, age, ID, medicalHistory.

Dùng virtual để cho phép override.

Subclass **ChronicPatient** thêm condition và override displayInfo().

Subclass **Regular** thêm lastVisitDate.

3.2 Doctor

```
class Doctor {
    string name;
    string specialization;
    vector<string> Appointments;public:
    void displayInfo();
    void showAppointments();
};
```

Quản lý lịch hẹn liên kết đến bác sĩ.

3.3 Appointment

```
class Appointment {
    string date, time, reason, status;public:
    void displayAppointment();
    void cancelAppointment();
    void completeAppointment();
};
```

Xử lý logic của một lịch hẹn (trạng thái, reschedule).

---

4. Test Results

Ví dụ test trong main():

Patient* p1 = new ChronicPatient("Alice", 45, "P001", "Diabetes");Doctor d1("Dr. Smith", "Cardiology");Appointment a1("2025-09-15", "10:00", "Regular checkup");

p1->displayInfo();
d1.displayInfo();
a1.displayAppointment();

Sample Output

Chronic Patient: Alice (ID: P001, Age: 45)Condition: DiabetesDoctor: Dr. Smith, Specialization: CardiologyAppointment on 2025-09-15 at 10:00 for: Regular checkupStatus: Scheduled

✅ Output cho thấy:

Polymorphism: displayInfo() của ChronicPatient override thành công.

Appointment hiển thị đầy đủ thông tin.

Doctor lưu specialization.

---

5. LLM Usage

Trong quá trình làm bài, em có sử dụng **ChatGPT** để hỗ trợ brainstorming và tham khảo cách thiết kế.

Ví dụ:

Em đã hỏi: *"Suggest methods for an Appointment class in a clinic system."*

ChatGPT gợi ý các phương thức: cancelAppointment(), completeAppointment(), rescheduleAppointment().

Sau đó, em **tự viết lại code** bằng C++, chỉnh sửa theo nhu cầu bài tập và kịch bản riêng.

Em có nhờ ChatGPT để hỗ trợ chú thích cho rõ ràng nhất cho cụ thể nhất.

Em không sao chép nguyên code, mà chỉ dùng LLM để tham khảo ý tưởng về thiết kế OOP.

---

6. Kết luận

Hệ thống thể hiện đầy đủ OOP concepts: **Encapsulation**, **Inheritance**, **Polymorphism**, **Association**.

Code chạy ổn, có test case minh họa.

Có thể cải tiến bằng việc dùng vector<Appointment*> thay vì vector<string> để tăng tính liên kết thực tế.