OOP Week 4 Assignment - Clinic Management System

Student: Doan Trong Trung (24110140)

Object-Oriented Analysis (OOA)

Objects Identified

From the clinic management scenario, I identified these key objects: Patient, ChronicPatient, Doctor, Appointment, Medicine, Prescription, Bill, ClinicManagementSystem

Attributes and Methods

- **Patient: name, patientID, age, medicalHistory | addMedicalRecord(), displayInfo(), getAppointmentFrequency()**
- **ChronicPatient: conditionType, lastCheckupDate + Patient attributes | overridden displayInfo(), getAppointmentFrequency()**
- **Doctor: name, doctorID, specialty, appointmentIDs | assignAppointment(), displayInfo()**
- **Appointment: appointmentID, date, time, status | reschedule(), setStatus()**

Inheritance Relationship

ChronicPatient inherits from Patient - ChronicPatients ARE patients but need specialized appointment scheduling (regular checkups vs. as-needed visits).

Class Design and Inheritance Explanation

Why I Used Inheritance

I implemented Patient as base class with ChronicPatient as derived class because:

1. **Real-world relationship: ChronicPatients ARE patients with additional needs**
2. **Shared behavior: Both types need basic patient functionality (name, ID, medical records)**
3. **Specialized behavior: ChronicPatients require different appointment frequency ("Every 3 months" vs. "As needed")**
4. **Polymorphism opportunity: Same method calls can produce different behaviors**

Inheritance Implementation

```
class Patient {
protected:
   string name; int patientID; int age;
public:
   virtual string getAppointmentFrequency() const {
      return "As needed";
   }
   virtual string getPatientType() const {
      return "Regular Patient";
   }
};

class ChronicPatient : public Patient {
private:
   string conditionType, lastCheckupDate;
public:
   string getAppointmentFrequency() const override {
      return "Every 3 months (chronic condition)";
   }
   string getPatientType() const override {
      return "Chronic Patient";
   }
};
```

Design Benefits

- **Code Reusability: Common patient functionality written once in base class**

- **Polymorphism: Same interface produces different behaviors for different patient types**
- **Extensibility: Easy to add new patient types (PediatricPatient, SeniorPatient)**
- **Maintainability: Changes to basic patient functionality automatically apply to all types**

Code Walkthrough

Key Implementation Features

1. Virtual Functions for Polymorphism

```
virtual string getAppointmentFrequency() const
```

- **Purpose: Allows different patient types to have different appointment scheduling logic**
- **Benefit: Same method call produces appropriate behavior for each patient type**

2. Constructor Chaining

```
ChronicPatient(const string& name, int id, int age, const string& condition)
   : Patient(name, id, age), conditionType(condition) {}
```

- **Purpose: Properly initializes base class data before adding chronic-specific data**
- **Benefit: Ensures all patient data is correctly set up regardless of patient type**

3. Method Override with Base Class Reuse

```
void displayInfo() const override {
   Patient::displayInfo();  // Call base class method
   cout << "Chronic Condition: " << conditionType << endl;
}
```

- **Purpose: Extends base functionality rather than replacing it completely**
- **Benefit: Avoids code duplication while adding specialized information**

4. Polymorphic Storage

```
vector<Patient*> patients;  // Can store both Patient and ChronicPatient objects
```

- **Purpose: Enables treating different patient types uniformly while preserving individual behaviors**

- **Benefit: Simplifies system management while maintaining type-specific functionality**

Memory Management

- **Virtual Destructor: virtual ~Patient() {} ensures proper cleanup of derived objects**
- **RAII Principle: ClinicManagementSystem destructor handles all dynamic memory cleanup**

Test Results and Analysis

Test 1: Polymorphic Behavior

Code Executed:

```
cms.addPatient("Doan Trong Trung", 22);
cms.addChronicPatient("Pham Van Hoa", 60, "Diabetes", "2025-08-10");
cms.displayAllPatients();
```

Output:

```
--- Patient 1 ---
Patient Type: Regular Patient
Name: Doan Trong Trung
Appointment Frequency: As needed

--- Patient 2 ---
Patient Type: Chronic Patient
Name: Pham Van Hoa
Appointment Frequency: Every 3 months (chronic condition)
Chronic Condition: Diabetes
```

What This Demonstrates:

- **Virtual function override working: Same displayInfo() call produces different outputs**
- **Polymorphism in action: Base class pointer correctly calls derived class methods**

- **Inheritance benefit: ChronicPatient adds specialized info while reusing base functionality**

Test 2: System Integration

Code Executed:

cms.scheduleAppointment("2025-09-15", "09:00", "Checkup", 1, 1);

cms.createPrescription(1, 1, "2025-09-15", "Take after meals", {1});

cms.generateBill(1, 20.0, "2025-09-15");

Output:

Appointment scheduled successfully. Appointment ID: 1

Prescription created. ID: 1 for Patient ID: 1

Bill generated successfully. Bill ID: 1

Total Amount: $25 (Consultation: $20, Medication: $5)

What This Demonstrates:

- **Complete workflow: Patient registration → appointment → prescription → billing**
- **Object relationships: Different classes working together seamlessly**
- **Data integrity: Information flows correctly between related objects**

LLM Usage Documentation

How I Used AI Assistance

1. Initial Class Structure Planning

- **Prompt: "What classes should I include in a clinic management system with inheritance? What should inherit from what?"**
- **AI Response: Suggested Patient as base class with specialized patient types, plus supporting classes**
- **How I Applied: Used Patient→ChronicPatient inheritance suggestion, designed my own supporting classes (Medicine, Prescription, Bill)**

2. Virtual Method Identification

- **Prompt: "What methods should be virtual in a Patient class to demonstrate polymorphism in a clinic system?"**
- **AI Response: Recommended appointment frequency and patient type identification as good virtual method candidates**
- **How I Applied: Implemented getAppointmentFrequency() and getPatientType() as virtual methods with meaningful overrides**

3. Testing Strategy Development

- **Prompt: "How should I test inheritance and polymorphism in C++ to show they work correctly?"**
- **AI Response: Suggested creating objects through base class pointers and demonstrating different behaviors**
- **How I Applied: Developed test cases showing same method calls producing different results for different patient types**

4. Memory Management Guidance

- **Prompt: "How do I properly manage memory when using inheritance with polymorphic objects in containers?"**
- **AI Response: Recommended virtual destructors and storing objects as base class pointers**
- **How I Applied: Implemented virtual destructor in Patient class and used vector<Patient*> for polymorphic storage**

Ethical Usage Statement

- **AI Role: Provided conceptual guidance and learning support only**
- **My Contribution: All actual code implementation, logic design, and creative decisions were original work**
- **Academic Integrity: Used AI as a brainstorming and learning tool, not as a code generator**
- **Verification: Personally understood and validated all AI suggestions before implementing them**