Lê Thanh Đại Dương – 24110085

Small Clinic Management System

Report:

Object-Oriented Analysis (OOA)

1. Identify Objects (Nouns)

- **Patient**
- **ChronicPatient** (special type of Patient)
- **Doctor**
- **Appointment**
- **Clinic** (implicitly represented by interactions between Patient, Doctor, Appointment)

2. Identify Attributes (Descriptive Nouns)

- **Patient**: name, id, age, history, appointments
- **ChronicPatient**: condition, lastCheckup (inherits Patient's attributes)
- **Doctor**: name, id, specialty
- **Appointment**: id, date, time, reason, status, patientName, doctorName

3. Identify Methods (Verbs)

- **Patient**:
    - addHistory()
    - scheduleAppointment()
    - printAppointments()
    - printInfo()
- **ChronicPatient**:
    - scheduleAppointment() (overridden to add condition-specific behavior)
    - printInfo() (extends Patient's method with extra details)

- **Doctor**:
    - printInfo()
- **Appointment**:
    - getId()
    - setStatus()
    - printInfo()

4. Inheritance Relationships

- **ChronicPatient** inherits from **Patient** (class ChronicPatient : public Patient).
- Both **Patient** and **Doctor** use **Appointment** objects but do not inherit from it.

Explanation of Class Design

Appointment – the scheduled meeting between a patient and a doctor.

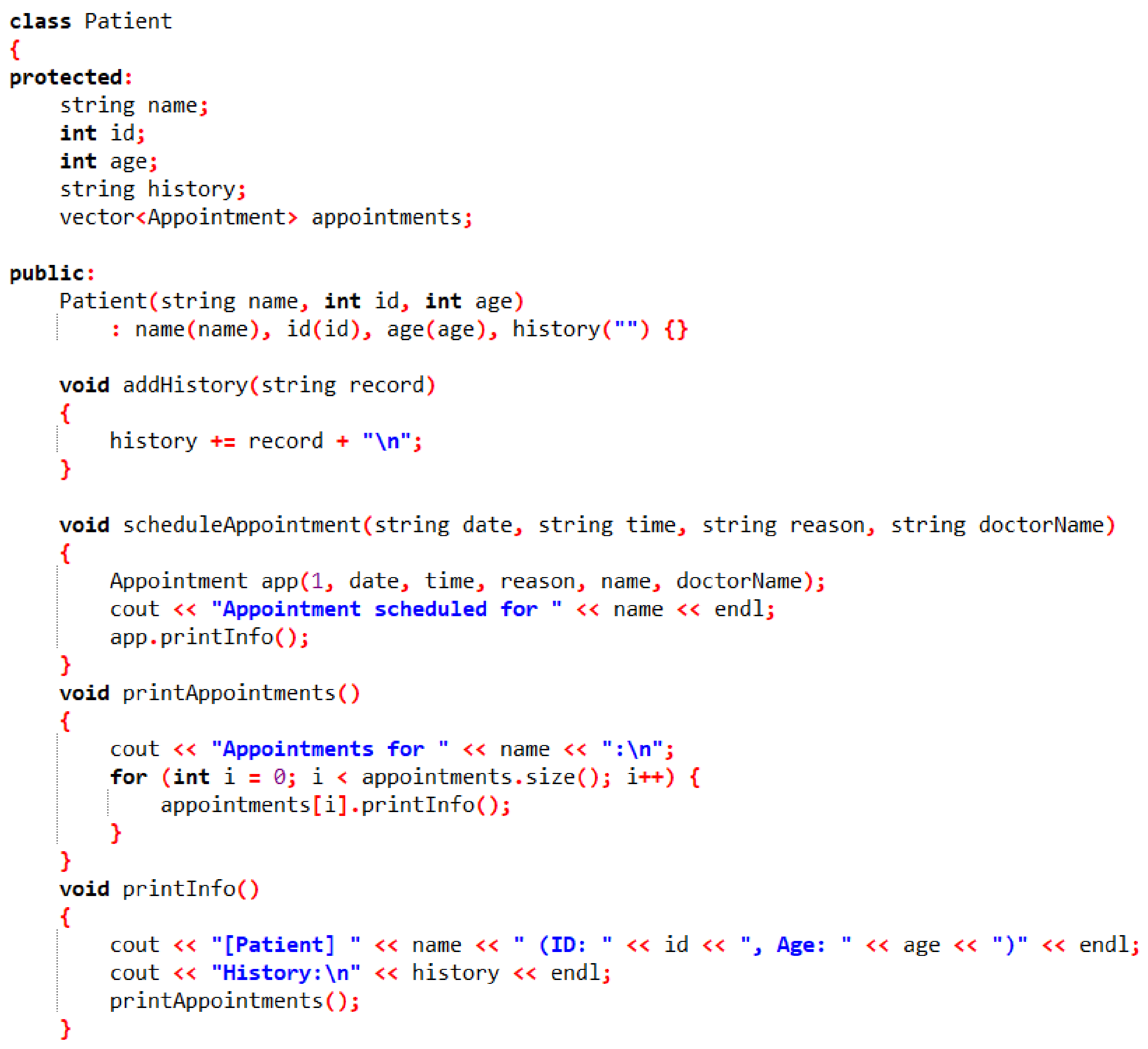
```
class Appointment
{
private:
    int id;
    string date;
    string time;
    string reason;
    string status;
    string patientName;
    string doctorName;

public:
    Appointment(int id, string date, string time, string reason,
                string patientName, string doctorName)
        : id(id), date(date), time(time), reason(reason),
        status("Scheduled"),
        patientName(patientName), doctorName(doctorName) {}

    int getId() { return id; }

    void setStatus(string s) { status = s; }

    void printInfo()
    {
        cout << "[Appointment " << id << "] Date: " << date << " " << time << endl;
        cout << "   Patient: " << patientName << endl;
        cout << "   Doctor: " << doctorName << endl;
        cout << "   Reason: " << reason << endl;
        cout << "   Status: " << status << endl;
    }
};
```

Attributes: id, date, time, reason, status, patientName, doctorName.

Methods: printInfo(), setStatus(), getId().

Allows displaying appointment or updating it.

Patient – the clinic patient.



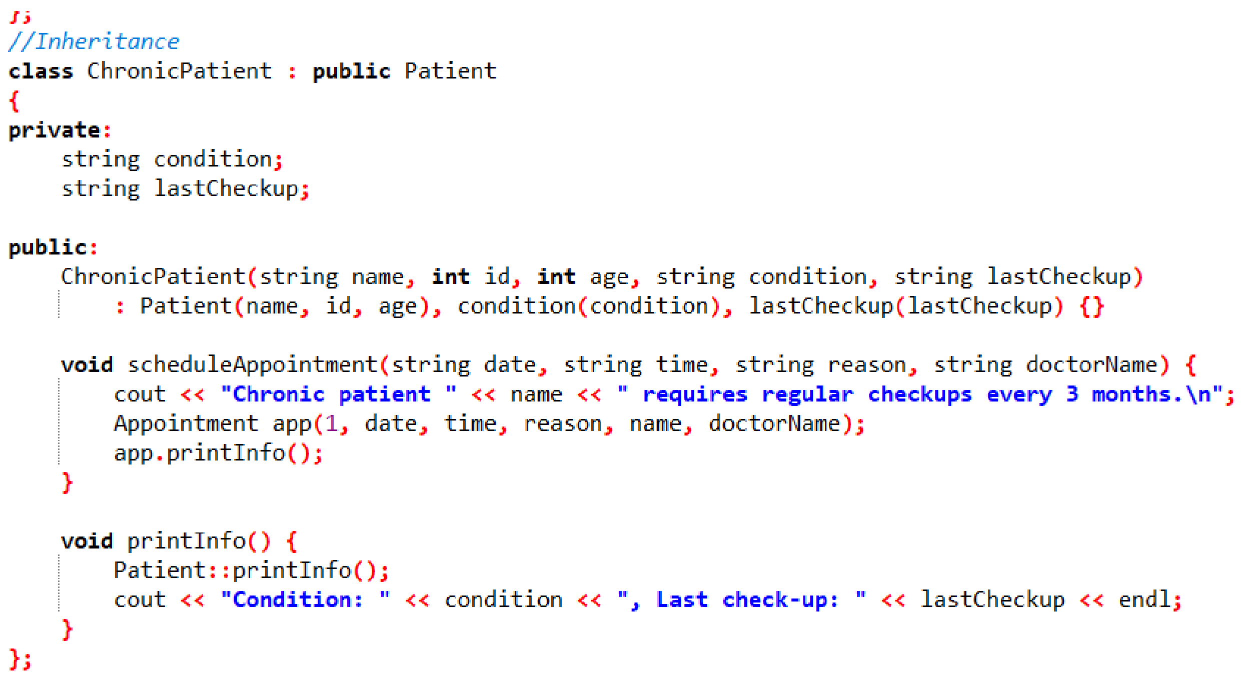
```
class Patient
{
protected:
    string name;
    int id;
    int age;
    string history;
    vector<Appointment> appointments;

public:
    Patient(string name, int id, int age)
        : name(name), id(id), age(age), history("") {}

    void addHistory(string record)
    {
        history += record + "\n";
    }

    void scheduleAppointment(string date, string time, string reason, string doctorName)
    {
        Appointment app(1, date, time, reason, name, doctorName);
        cout << "Appointment scheduled for " << name << endl;
        app.printInfo();
    }
    void printAppointments()
    {
        cout << "Appointments for " << name << ":\n";
        for (int i = 0; i < appointments.size(); i++) {
            appointments[i].printInfo();
        }
    }
    void printInfo()
    {
        cout << "[Patient] " << name << " (ID: " << id << ", Age: " << age << ")" << endl;
        cout << "History:\n" << history << endl;
        printAppointments();
    }
```

Attributes: name, id, age, history, appointments.

Methods: addHistory(), scheduleAppointment(), printInfo(), printAppointments().

Manages basic patient information, medical history, and appointments.

ChronicPatient (inherits from Patient) – the patient with long-term conditions.



```
;
//Inheritance
class ChronicPatient : public Patient
{
private:
    string condition;
    string lastCheckup;

public:
    ChronicPatient(string name, int id, int age, string condition, string lastCheckup)
        : Patient(name, id, age), condition(condition), lastCheckup(lastCheckup) {}

    void scheduleAppointment(string date, string time, string reason, string doctorName) {
        cout << "Chronic patient " << name << " requires regular checkups every 3 months.\n";
        Appointment app(1, date, time, reason, name, doctorName);
        app.printInfo();
    }

    void printInfo() {
        Patient::printInfo();
        cout << "Condition: " << condition << ", Last check-up: " << lastCheckup << endl;
    }
};
```

Extra Attributes: condition, lastCheckup.

Overridden Method: scheduleAppointment() – prints an additional reminder that chronic patients require regular checkups every 3 months.

Using inheritance and method overriding, extending patient functionality without duplicating code.

Doctor – the clinic doctor.

Attributes: name, id, specialty.

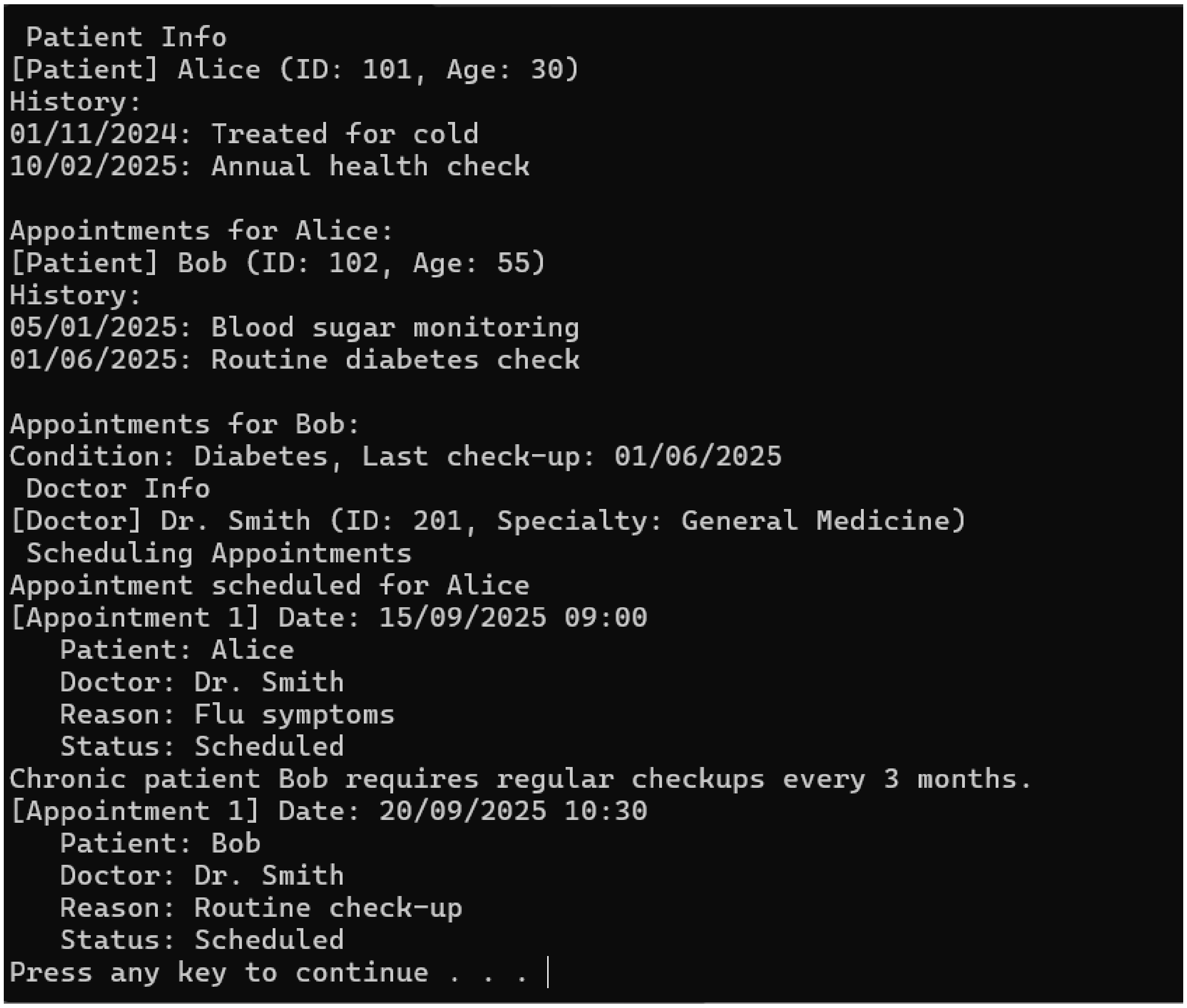
Methods: printInfo().

Manages doctor information and links them with appointments.

Inheritance was applied between Patient and ChronicPatient because both share common information (name, ID, age, history, appointments). Instead of duplicating this logic, the ChronicPatient class reuses Patient’s structure avoid rewriting when add more ChronicPatient.

The overall idea and flow of logic is my own ideas with some help of ChatGPT like appointments linked to patients and doctors, histories stored as text and suggesting minor improvement like help me add history record,…

Test result:



```
 Patient Info
[Patient] Alice (ID: 101, Age: 30)
History:
01/11/2024: Treated for cold
10/02/2025: Annual health check

Appointments for Alice:
[Patient] Bob (ID: 102, Age: 55)
History:
05/01/2025: Blood sugar monitoring
01/06/2025: Routine diabetes check

Appointments for Bob:
Condition: Diabetes, Last check-up: 01/06/2025
 Doctor Info
[Doctor] Dr. Smith (ID: 201, Specialty: General Medicine)
 Scheduling Appointments
Appointment scheduled for Alice
[Appointment 1] Date: 15/09/2025 09:00
   Patient: Alice
   Doctor: Dr. Smith
   Reason: Flu symptoms
   Status: Scheduled
Chronic patient Bob requires regular checkups every 3 months.
[Appointment 1] Date: 20/09/2025 10:30
   Patient: Bob
   Doctor: Dr. Smith
   Reason: Routine check-up
   Status: Scheduled
Press any key to continue . . .
```