Object - Oriented Programming

Name: Lê Sĩ Nhật Khuê **Student's ID:** 24110102

Assignment 4 – Week 4: Small Clinic Management System

I/ Object-Oriented Analysis (OOA) Model

Base on the information provided, we have some analysis

- About **objects**, we can define some objects such as Doctor, Patient, ChronicPatient, Appointment, Clinic (for management), Staff.
- About **attributes** for each object:
  + For **Doctor:** name, ID, specialty, appointments.
  + For **Patient:** name, ID, age, medicalHistory.
  + For **ChronicPatient:** extends Patient and add conditionType, lastCheckupData.
  + For **Appointment:** date, time, reason, status.
  + (Extend) For **Clinic:** patients, doctors, staffMembers, appointments.
  + (Extend) For **Staff:** name, ID, role.
- About **methods** for each object:
  + For **Doctor:** viewAppointments, updateAppointmentStatus, displayInfo.
  + For **Patient:** scheduleAppointment, updateMedicalHistory, displayInfo.
  + For **ChronicPatient:** scheduleAppointment, displayInfo.
  + For **Appointment:** updateStatus, displayInfo.
  + For **Clinic:** addPatient, addDoctor, addStaff, scheduleAppointment(), cancelAppointment, displayAllPatient, displayAllDoctor,…
  + For **Staff:** displayInfo.
- About **inheritance relationships**:
  + ChronicPatient inherits from Patient: ChronicPatient is a specialized type of Patient, with additional attributes and overridden behaviors.
  + Patient – Appointment: One patient can have multiple appointments.
  + Doctor – Appointment: One doctor can have multiple appointments.
  + Clinic manages all entities.

II/ Overview of the Clinic Management System

Class Struct Date

- Use ***Struct*** type to format time

Class Appointment

- The **Appointment** class represents a meeting between a patient and a doctor. It keeps information such as the date, time, reason, status (scheduled, completed, or canceled), the patient's ID, and the doctor's ID. This class allows the system to show the details of an appointment or update its status when necessary.

Class Patient

- The **Patient** class represents a normal patient in the clinic. It stores the patient's name, ID, age, medical history, and a list of appointments. Through this class, the system can schedule new appointments for the patient, update their medical history, and display full information about the patient and their scheduled visits.

Class Doctor

- The **Doctor** class represents a doctor who works at the clinic. It includes the doctor's name, ID, specialty, and the list of appointments assigned to them. This class makes it possible to assign appointments to the doctor, check the status of these appointments, and display the doctor's profile together with their schedule.

Class ChronicPatient *(inherits from Patient)*

- The **ChronicPatient** class is a special type of patient that inherits from the Patient class. In addition to the basic patient information, it also stores the type of chronic condition and the date of the last checkup. When a chronic patient schedules a new appointment, this class automatically suggests the next follow-up date, which is usually three months later.

Class Staff

- The **Staff** class represents a staff member working in the clinic, such as a receptionist, nurse, or technician. It stores the staff member's name, ID, and role, and it can be used to display basic information about them.

Class Clinic

- The **Clinic** class acts as the central part of the system. It manages the lists of patients, doctors, staff members, and appointments. With this class, the system can add new people, find patients or doctors by their ID, schedule or cancel appointments, and display all the stored information about the clinic.

Main Function *(Testing the system)*

- We have two test cases to show how the system works:

    + Test case 1: Add patients, a doctor, a staff member, and schedule two appointments (one for a normal patient, one for a chronic patient). Then display everything.

    + Test case 2: Same as test case 1, but with different data.

III/ How to do this assignment

When starting this assignment, I first analyzed the requirements and realized that the clinic management system needed to handle the main entities: **patients, doctors, staff, and appointments**. Therefore, the first step was to identify the necessary classes: Appointment, Patient, Doctor, Staff, and a central manager class Clinic.

During the design process, I focused on applying two OOP concepts: **encapsulation** and **inheritance**.

Encapsulation: I declared class attributes as private (for example, name, ID, age in Patient or date, time, status in Appointment) so that they could not be accessed directly from outside. Then, I created **getter** and **setter** methods to control how data is read or modified. This way, data is protected and can only be changed through the methods I defined.

Inheritance: After implementing the Patient class, I realized that there should be a special type of patient: chronic patients. Instead of rewriting everything, I created the ChronicPatient class that **inherits** from Patient. I added new attributes (conditionType, lastCheckupDate) and **overrode the scheduleAppointment method**. This allowed chronic patients not only to schedule appointments like normal patients but also to automatically receive a follow-up reminder.

Once the classes were ready, I built the **Clinic** class as the central manager that keeps track of all patients, doctors, staff, and appointments. In this class, I implemented functions like addPatient, addDoctor, scheduleAppointment, and cancelAppointment to simulate real operations in a clinic.
Finally, I created **test cases** in the main function to check the system. In each case, I added patients, doctors, and staff, then scheduled appointments and displayed results. This helped me verify that the system worked correctly: data was securely encapsulated, and inheritance made it easier to reuse and extend the functionality.

IV/ Difficulties Encountered

While doing this assignment, I faced several challenges:

- **Designing class relationships:** At first, it was not easy to decide which classes should exist and how they should interact. For example, I had to think carefully about whether Appointment should belong to Patient, Doctor, or both, before deciding to manage them through the Clinic class.

- **Applying encapsulation correctly:** It was a bit confusing to remember to keep attributes private and always use getters and setters. Sometimes I forgot and tried to access attributes directly, which caused errors and reminded me of the importance of encapsulation.

- **Using inheritance effectively:** When creating the ChronicPatient subclass, I had to figure out how to reuse the methods from Patient while still overriding scheduleAppointment to add new behavior. Understanding method overriding in practice was a bit challenging at first.

- **Testing the system:** I needed to design proper test cases in main to make sure all classes worked together. It required several trials to ensure that scheduling, canceling, and displaying appointments behaved as expected.

- **Seeking external support:** During the process, I also used ChatGPT as a supporting tool. It helped me clarify OOP concepts, suggested improvements in class design, and guided me in explaining how encapsulation and inheritance were applied. This assistance was useful in overcoming confusion and speeding up the completion of the assignment.