Clinic Management Program in Java Report

Abstract:

The Clinic Management Program developed in Java aims to efficiently manage a clinic's operations by handling doctors, receptionists, cashiers, and patients. The program allows the admin to add medical staff and patients, schedule patient visits, and manage the payment process. This report presents a detailed overview of the program's design, structure, and functionalities, including the implementation of key classes and their respective methods.

1. Introduction:

The Clinic Management Program is designed to streamline the operations of a medical clinic by automating administrative tasks and patient flow. It ensures efficient handling of patient visits, doctor assignments, and billing processes, thereby improving the overall clinic management.

2. Class Design:

2.1 Admin Class:

The Admin class acts as the central hub of the program, holding and managing various lists and queues. First, in the Admin class, the variables of doctors, the list of receptionists, cashiers, patients, and the queue of patients are created and set in the constructor.
It has methods to add doctors, cashiers, and receptionists to their respective lists. addDoctor, addCashier, and addReceptionist methods are added to their lists based on the names and IDs attributed to them as random variables.
Additionally, it features methods to select a doctor based on the patient's disease type, manage the patient treatment flow, and display information about patients, doctors, receptionists, and cashiers. In the selectDoctor method, a doctor was randomly selected from specified doctors according to the type of disease of the patient.
In the starPatientTreatmentFlow method, the path from the beginning of admission to entering the queue of the required doctor. In the next method, the patient is entered into the queue of the determined doctor.
In the removePatientFromPatientQueue method, which is performed after the patient is visited, the patient is removed from the patient queue. Then, in the following methods, the patients in the waiting queue, all the patients, doctors, receptionists, and cashiers can be seen in order.
In the next method, we receive the information of doctors and patients.

2.2 Doctor Class:

The Doctor class represents medical practitioners. Each doctor has an ID, name, type, and a queue of patients awaiting their consultation.
First, the ID, the doctor's name, the type of doctor, the queue of patient variables, and an object from the admin class were taken and initialized in the constructor.
Then the type of doctors is categorized in a countable data type.

In the next step, in the isSpecialist method, whether the doctor is a specialist or a generalist was specified, and in the visit method, first, the specified patient from the doctor's waiting queue was removed and placed in the visited state, then in the addPatinet method, the patient is placed in the patient queue in the waiting state.
We implement the getter method for doctor ID and doctor type.

2.3 Patient Class:

The Patient class encapsulates patient information, including name, disease type, status, and assigned doctor (which is determined by the DoctorType method of the doctor class). The status of the patient changes as they progress through the treatment flow. The different states of patients are placed in an enumeration variable. In the constructor of this class, the variables are initialized, which are set in DEFAULT status if the treatment flow does not start. The class provides getter and setter methods for disease type, doctor assignment, and payment status(which is set to true in the payBill method).

2.4 Receptionist Class:

The Receptionist class uses an EnumMap to manage the process of assigning a suitable doctor to patients based on their disease type. In the receipt method, we take the patient's type of disease using an object from the patient class. If a suitable doctor is found, the patient's status is set to ACCEPTED. Otherwise, an exception is thrown.

2.5 Cashier Class:

The Cashier class handles the billing process and doctor salary payment. It calculates the cost of treatment for each patient based on the doctor's type and the ratio of the doctors' income to the total income. These variables are initialized in the constructor. In the createBill method, the amount of the cost for the patient according to the type of doctor will be specified. It also ensures that patients pay their bills. In the startPatientFlow method, and if a patient pays the bill, then the doctor will be assigned; otherwise, an error will be thrown. It also manages the distribution of doctors' income using a HashMap, which determines the number of patients visited by each doctor with the doctor's ID key. Then, in the payDoctorSalary method, according to the doctors’ counter, we give them half of the income, which is executed when the administrator calls it.

Cancer Specialist and Neurologist Classes:

Both of these classes inherit from the doctor class, and their ID, name, and type are initialized in the constructor.

Class clinicManagement Class:

Regarding an output from the program, the clinicManagement class is the test class, in which the inputs (doctor, staff, and patient) are defined and execute their methods and output different queues for the doctor and admin, and the output of this class is actually It is the output of the program.

3. Implementation Details:

The implementation of the Clinic Management Program follows the Object-Oriented Programming (OOP) principles, utilizing classes and their methods to model the clinic's entities and their interactions.

User Interaction:

The program allows user interaction through the clinicManagement class, acting as the main entry point. It initializes doctors, staff, and patients, executes relevant methods, and displays outputs, including queues for doctors and administrators.

Conclusion:

The Clinic Management Program presents an effective solution for managing clinic operations, improving patient flow, and ensuring accurate billing and salary distribution. The program's modular design and OOP principles make it scalable and maintainable for future enhancements. By automating administrative tasks, the program enhances clinic efficiency and delivers a smoother patient experience.