**Assignment1 #1 – 24F Report**

Implementation of Treatment and Prescription Services

Student Number: 041098935

Student Name: KyungA Jang

* Part 3
  1. UML Class Diagram

A diagram of a diagram

Description automatically generated with medium confidence

* 1. Class Responsibilities and Application of SOLID Principles
* Single Responsibility Principal Violation in PatientManagement.java

Current Violation:  
The PatientManagement.java class currently handles multiple responsibilities: managing patient data, creating treatment plans, and generating prescriptions. This violates the Single Responsibility Principle (SRP), which states that a class should only have one reason to change, meaning it should focus on a single responsibility.

Proposed Refactoring:

To adhere to SRP, we can split the responsibilities into separate classes:

* PatientManager: Focuses solely on managing patient information (adding, updating, retrieving patient records).
* TreatmentManager: Handles the creation of treatment plans.
* PrescriptionManager: Deals with generating prescriptions.

With this refactor, each class now has a single, focused responsibility, adhering to the Single Responsibility Principle

2. Liskov Substitution Principal Violation in Patient.java, Inpatient.java, and Outpatient.java

Current Violation:  
The Inpatient.java and Outpatient.java classes inherit from Patient.java, but there are behaviors specific to these subclasses (such as room numbers for inpatients and appointment dates for outpatients) that could result in improper substitution, violating the Liskov Substitution Principle (LSP).

Proposed Refactoring:  
To adhere to LSP, we can refactor Inpatient and Outpatient so that they can be substituted for Patient without causing issues in code that expects the general Patient type. A potential solution is to introduce abstract methods in Patient that both subclasses must implement and make shared functionality work across all patient types. With this approach, both Inpatient and Outpatient can now be safely substituted wherever Patient is expected, thus complying with the Liskov Substitution Principle.

3. Dependency Inversion Principal Violation in PatientManagement.java

Current Violation:  
In the original PatientManagement.java, high-level methods such as createTreatmentPlan() and generatePrescription() directly depend on the concrete implementations of the treatment and prescription logic. This violates the Dependency Inversion Principle (DIP), which states that high-level modules should not depend on low-level modules; both should depend on abstractions.

Proposed Refactoring:  
To align with DIP, we will introduce dependency injection via the TreatmentPlan and PrescriptionService interfaces. This allows PatientManagement to work with any treatment or prescription strategy without being tightly coupled to concrete implementations. With this refactor, the PatientManager class no longer directly depends on specific treatment or prescription services, adhering to the Dependency Inversion Principle.

4. Implementation of MedicationTreatmentPlan and SurgeryTreatmentPlan

I implemented the TreatmentPlan interface in both the MedicationTreatmentPlan and SurgeryTreatmentPlan classes. These classes are responsible for generating specific treatment plans for patients, based on their conditions. These classes now follow the Open/Closed Principle—new treatment plans can be added without modifying existing code, and they align with the Strategy Pattern, allowing flexible switching between treatment strategies.

Conclusion

1. Single Responsibility Principle (SRP): The refactor splits patient management, treatment, and prescription responsibilities into separate classes.
2. Liskov Substitution Principle (LSP): Inpatient and Outpatient are refactored to be substitutable wherever a Patient is expected.
3. Dependency Inversion Principle (DIP): High-level modules now depend on abstractions (interfaces) rather than concrete implementations, providing flexibility and adhering to SOLID design principles.