The **Liskov Substitution Principle (LSP)** states that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. Let's examine the provided classes and identify LSP violations:

**Violation:**

In the Outpatient class, the admit () method throws an UnsupportedOperationException. This violates the LSP because, if the Patient class is used polymorphically, the behavior of calling admit() on an Outpatient instance differs drastically from calling it on an Inpatient instance.

This violation happens because the Patient class assumes that every patient can be admitted to a hospital, which isn't true for outpatients. To comply with LSP, we should ensure that both Inpatient and Outpatient classes can be used interchangeably without breaking functionality.

**Solution:**

To fix this, we'll introduce a method in the Patient class that can be implemented differently based on whether the patient is an inpatient or an outpatient. For example, instead of calling admit() in a way that assumes hospital admission, we can create a more general method such as processPatient(). This will allow both Inpatient and Outpatient classes to have meaningful implementations without violating LSP.

Here’s the refactored code:

**Refactoring the Patient Class:**

java

Copy code

package com.algonquin.cst8288.fall24.assignment1.patient;

/\*\*

\* Patient class that holds all patient data.

\*/

public abstract class Patient {

private String id;

private String name;

private String email;

private String phoneNumber;

private String dateOfBirth;

private long age;

private String lifeStage;

private String plannedTreatment;

// Constructor and Getters/Setters

public Patient(String id, String name, String email, String phoneNumber, String dateOfBirth) {

this.id = id;

this.name = name;

this.email = email;

this.phoneNumber = phoneNumber;

this.dateOfBirth = dateOfBirth;

}

public String getId() { return id; }

public String getName() { return name; }

public String getEmail() { return email; }

public String getPhoneNumber() { return phoneNumber; }

public String getDateOfBirth() { return dateOfBirth; }

public long getAge() { return age; }

public String getLifeStage() { return lifeStage; }

public String getPlannedTreatment() { return plannedTreatment; }

public void setId(String id) { this.id = id; }

public void setName(String name) { this.name = name; }

public void setEmail(String email) { this.email = email; }

public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }

public void setDateOfBirth(String dateOfBirth) { this.dateOfBirth = dateOfBirth; }

public void setAge(long age) { this.age = age; }

public void setLifeStage(String lifeStage) { this.lifeStage = lifeStage; }

public void setPlannedTreatment(String plannedTreatment) { this.plannedTreatment = plannedTreatment; }

// Generalized method to process the patient (instead of forcing admission)

public abstract void processPatient();

}

**Refactoring the Inpatient Class:**

java

Copy code

package com.algonquin.cst8288.fall24.assignment1.patient;

public class Inpatient extends Patient {

private String roomNumber;

public Inpatient(String id, String name, String email, String phoneNumber, String dateOfBirth, String roomNumber) {

super(id, name, email, phoneNumber, dateOfBirth);

this.roomNumber = roomNumber;

}

public String getRoomNumber() {

return roomNumber;

}

@Override

public void processPatient() {

System.out.println(getName() + " has been admitted to " + getRoomNumber());

}

}

**Refactoring the Outpatient Class:**

java

Copy code

package com.algonquin.cst8288.fall24.assignment1.patient;

public class Outpatient extends Patient {

private String appointmentDate;

public Outpatient(String id, String name, String email, String phoneNumber, String dateOfBirth, String appointmentDate) {

super(id, name, email, phoneNumber, dateOfBirth);

this.appointmentDate = appointmentDate;

}

public String getAppointmentDate() {

return appointmentDate;

}

@Override

public void processPatient() {

System.out.println(getName() + " has an appointment on " + getAppointmentDate());

}

}

**Explanation of Refactoring:**

* **Inpatient Class:** The processPatient() method for inpatients performs admission-related tasks, such as admitting the patient to a room.
* **Outpatient Class:** The processPatient() method for outpatients handles outpatient-specific tasks, such as scheduling or displaying the appointment date.

By introducing the processPatient() method, we maintain polymorphism without violating LSP. Both Inpatient and Outpatient classes can now be used interchangeably through the Patient interface.

This ensures that the behavior of each class is respected without throwing exceptions, and both classes can be substituted without changing the overall functionality of the system.

Would you like to proceed with the implementation of the remaining tasks for this assignment? ​