**Part 1: Refactor with SOLID Principles**

**1. Refactor PatientManagement.java for Single Responsibility Principle (SRP)**

**Problem:**

* The PatientManagement class is handling validation, calculating patient age, creating treatment plans, and generating prescriptions, which violates SRP. Each of these responsibilities should be handled by separate classes.

**Solution:**

* Split responsibilities into individual classes:
  + PatientValidator for validating patient data.
  + PatientAgeCalculator for calculating patient age.
  + TreatmentPlanManager for creating treatment plans.
  + PrescriptionServiceManager for managing prescriptions.

Here’s an example of refactoring:

java

Copy code

// PatientValidator.java

public class PatientValidator {

public boolean validatePatient(Patient patient) {

return isValidName(patient.getName()) &&

isValidAlphaNumeric(patient.getName()) &&

isValidEmail(patient.getEmail()) &&

isValidPhone(patient.getPhoneNumber());

}

private boolean isValidName(String value) {

return value != null && value.trim().length() > 0;

}

private boolean isValidAlphaNumeric(String value) {

Pattern pattern = Pattern.compile("[^A-Za-z0-9]");

Matcher matcher = pattern.matcher(value);

return !matcher.find();

}

private boolean isValidEmail(String value) {

Pattern pattern = Pattern.compile("^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,6}$");

Matcher matcher = pattern.matcher(value);

return matcher.find();

}

private boolean isValidPhone(String value) {

Pattern pattern = Pattern.compile("^\\+?[0-9. ()-]{7,25}$");

Matcher matcher = pattern.matcher(value);

return matcher.find();

}

}

Now PatientManagement will delegate validation tasks to PatientValidator.

**2. Ensure Liskov Substitution Principle (LSP) is followed for Patient.java, Inpatient.java, and Outpatient.java**

**Problem:**

* In the PatientManagement class, both Inpatient and Outpatient inherit from Patient, but the createTreatmentPlan() method assumes that all patients will have surgery, which violates LSP. These subclasses should be substitutable for Patient without any unexpected behavior.

**Solution:**

* Create separate treatment logic for Inpatient and Outpatient patients to adhere to LSP.

For example:

java

Copy code

public class Inpatient extends Patient {

// Inpatients may have surgery or other detailed treatments

}

public class Outpatient extends Patient {

// Outpatients may only require medication

}

Ensure each subclass defines its unique behavior without making the base class (Patient) dependent on specific implementations.

**3. Refactor for Dependency Inversion Principle (DIP)**

**Problem:**

* createTreatmentPlan() and generatePrescription() in PatientManagement directly depend on specific implementations like SurgeryTreatmentPlan and OnlinePrescriptionService, violating DIP.

**Solution:**

* Refactor these methods to depend on abstractions (interfaces) instead of concrete implementations.

java

Copy code

// TreatmentPlanManager.java

public class TreatmentPlanManager {

private TreatmentPlan treatmentPlan;

public void createTreatmentPlan(Patient patient, TreatmentPlan treatmentPlan) {

this.treatmentPlan = treatmentPlan;

treatmentPlan.createTreatmentPlan(patient);

}

}

// PrescriptionServiceManager.java

public class PrescriptionServiceManager {

private PrescriptionService prescriptionService;

public void generatePrescription(Patient patient, PrescriptionService prescriptionService, String filename) throws IOException {

BufferedWriter bwriter = new BufferedWriter(new FileWriter(filename));

bwriter.write(prescriptionService.generatePrescription(patient));

bwriter.close();

}

}

Now the PatientManagement class doesn’t rely on specific implementations, making it more flexible.

**4. Implement MedicationTreatmentPlan.java and SurgeryTreatmentPlan.java**

java

Copy code

// MedicationTreatmentPlan.java

public class MedicationTreatmentPlan implements TreatmentPlan {

@Override

public void createTreatmentPlan(Patient patient) {

// Implement treatment logic based on patient life stage

String lifeStage = patient.getLifeStage();

int doseFrequency = (lifeStage.equals(Constants.CHILD)) ? 1 :

(lifeStage.equals(Constants.YOUTH)) ? 2 : 3;

// Set up duration based on the condition

// Example logic for setting medication duration

// patient.setTreatmentDetails(...);

}

}

// SurgeryTreatmentPlan.java

public class SurgeryTreatmentPlan implements TreatmentPlan {

@Override

public void createTreatmentPlan(Patient patient) {

// Implement surgery treatment logic with risk factors, fasting times, etc.

long age = patient.getAge();

// Apply logic based on age risk factors, fasting times, and follow-up

}

}

**5. Implement OnlinePrescriptionService.java and PrintablePrescriptionService.java**

java

Copy code

// OnlinePrescriptionService.java

public class OnlinePrescriptionService implements PrescriptionService {

@Override

public String generatePrescription(Patient patient) {

return "<html><body><h1>Prescription for " + patient.getName() + "</h1></body></html>";

}

}

// PrintablePrescriptionService.java

public class PrintablePrescriptionService implements PrescriptionService {

@Override

public String generatePrescription(Patient patient) {

return "Prescription for " + patient.getName();

}

}

**Part 2: JUnit Tests**

For testing your refactored classes, you'll need to create unit tests that validate the business logic for each class. Here's an example of how to test the medication treatment plan:

java

Copy code

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class MedicationTreatmentPlanTest {

@Test

public void testChildDoseFrequency() {

Patient childPatient = new Patient();

childPatient.setLifeStage(Constants.CHILD);

MedicationTreatmentPlan medicationPlan = new MedicationTreatmentPlan();

medicationPlan.createTreatmentPlan(childPatient);

// assertEquals logic based on expected values

}

// Similar tests for youth and adult patients

}

**Part 3: UML and SOLID Report**

For your UML diagram:

* Show relationships between classes (e.g., PatientManagement uses TreatmentPlanManager).
* Include interfaces (TreatmentPlan and PrescriptionService) and their implementations (SurgeryTreatmentPlan, MedicationTreatmentPlan, OnlinePrescriptionService, PrintablePrescriptionService).
* Explain how each refactor aligns with the SOLID principles in your report.