**Task 1: Analysis**

**SOLID Principles Violations**

**Single Responsibility Principle (SRP)**

**Violation**: The student class not only holds student data but also handles updating its own data (update student) and displaying its information (display student). The StudentManagementSystem class handles student creation, deletion, updates, and interaction with the database, leading to multiple responsibilities.

**Solution**: Separate the responsibilities by creating dedicated classes or methods for each function, such as a Student Service class for managing student data.

**Open/Closed Principle (OCP)**

**Violation**: The system is not easily extendable. For example, if a new type of student (e.g., an international student) were introduced, you would have to modify the student class and other related methods.

**Solution:** Use inheritance or interfaces to allow for extension without modifying existing code.

**Leskov Substitution Principle (LSP)**

**Violation**: The current code doesn’t violate this principle directly but would likely do so if new types of students were introduced without proper design. For instance, if a subclass of Student doesn't fully implement the expected behaviors, it could lead to issues.

**Solution**: Ensure that any subclass of Student can be used interchangeably with the student class..

**DRY (Don't Repeat Yourself)**

**Violation**: The code repeats logic in various places, like the search for a student by ID in both update\_student\_info and delete\_student. This repetition leads to redundant code.

Solution: Extract the repeated logic into a helper method, reducing redundancy.

**KISS (Keep It Simple Stupid)**

**Violation**: The code unnecessarily combines student management with the display logic within the student class, making it more complex than needed.

**Solution**: Keep classes focused on a single responsibility and avoid mixing concerns.

**YAGNI (You Ain't Gonna Need It)**

**Violation**: The current system doesn’t appear to have unnecessary features, but adding complexity, like extra methods or classes without clear justification, would violate YAGNI.

**Solution**: Ensure that all added features are necessary for the current requirements.

**Task 2: Refactoring**

# student.py

class Student:

def \_\_init\_\_(self, id, name, age, major):

self.id = id

self.name = name

self.age = age

self.major = major

# student\_service.py

class StudentService:

def \_\_init\_\_(self, database):

self.database = database

def add\_student(self, student):

self.database.add\_student(student)

def update\_student\_info(self, student\_id, name=None, age=None, major=None):

student = self.database.get\_student\_by\_id(student\_id)

if student:

if name:

student.name = name

if age:

student.age = age

if major:

student.major = major

def delete\_student(self, student\_id):

self.database.remove\_student(student\_id)

# student\_database.py

class StudentDatabase:

def \_\_init\_\_(self):

self.students = []

def add\_student(self, student):

self.students.append(student)

def get\_student\_by\_id(self, student\_id):

return next((student for student in self.students if student.id == student\_id), None)

def remove\_student(self, student\_id):

student = self.get\_student\_by\_id(student\_id)

if student:

self.students.remove(student)

def display\_all\_students(self):

for student in self.students:

print(f"ID: {student.id}, Name: {student.name}, Age: {student.age}, Major: {student.major}")

# main.py

class MenuSystem:

def \_\_init\_\_(self, student\_service):

self.student\_service = student\_service

def display\_menu(self):

while True:

print("\n1. Add Student")

print("2. Update Student")

print("3. Delete Student")

print("4. View All Students")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == '1':

self.add\_student()

elif choice == '2':

self.update\_student()

elif choice == '3':

self.delete\_student()

elif choice == '4':

self.view\_all\_students()

elif choice == '5':

break

else:

print("Invalid choice! Please try again.")

def add\_student(self):

id = int(input("Enter ID: "))

name = input("Enter Name: ")

age = int(input("Enter Age: "))

major = input("Enter Major: ")

student = Student(id, name, age, major)

self.student\_service.add\_student(student)

def update\_student(self):

id = int(input("Enter Student ID to Update: "))

name = input("Enter New Name (leave blank to keep current): ")

age = input("Enter New Age (leave blank to keep current): ")

major = input("Enter New Major (leave blank to keep current): ")

self.student\_service.update\_student\_info(id, name if name else None, int(age) if age else None, major if major else None)

def delete\_student(self):

id = int(input("Enter Student ID to Delete: "))

self.student\_service.delete\_student(id)

def view\_all\_students(self):

self.student\_service.database.display\_all\_students()

# example usage

database = StudentDatabase()

student\_service = StudentService(database)

menu = MenuSystem(student\_service)

menu.display\_menu()

**Task 3: Documentation**

**Comments:**

**Student Class**: Holds student data and represents the basic entity in the system.

**StudentService Class**: Contains business logic related to student operations like adding, updating, and deleting students.

**StudentDatabase Class**: Manages the collection of students and performs database-like operations such as adding, removing, and retrieving students.

**MenuSystem Class**: Provides a simple text-based interface for user interaction, enabling users to perform various student management tasks.

**README.md**

# Student Management System

## Overview

This Student Management System allows educational institutions to manage student data, including adding, updating, deleting, and viewing student information.

## Features

- Add new students

- Update existing student information

- Delete students

- View all students

## Installation

1. Clone the repository:

```

git clone https://github.com/yourusername/student-management-system.git

```

2. Navigate to the project directory:

```

cd student-management-system

```

3. Run the system:

```

python main.py

```

## Code Structure

- `student.py`: Contains the `Student` class, which represents the student entity.

- `student\_service.py`: Contains the `StudentService` class, which handles the business logic for managing students.

- `student\_database.py`: Contains the `StudentDatabase` class, which mimics database operations for storing students.

- `main.py`: Contains the `MenuSystem` class, which provides a simple menu for interacting with the system.

## Changes Made

- Refactored code to follow SOLID principles.

- Removed redundant code (DRY).

- Simplified the design (KISS).

- Implemented a user-friendly menu system.