EASTERN INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING

AND INFORMATION TECHNOLOGY

Practice Assignment – Quarter 1, 2024-2025

Course Name: OOD

Course Code: CSE 203

Student's Full Name:

Student ID:

Lab 6

1. Shape Drawing Application (Open/Closed Principle)

Description:

- Develop an application that can draw different shapes and calculate their areas.

Requirements:

- Define an abstract class or interface `Shape` with methods `draw()` and `calculateArea()`.
- Implement subclasses like `Circle`, `Rectangle`, and `Triangle`.
- Allow adding new shapes without modifying existing code.

Objectives:

- Apply the **Open/Closed Principle** by designing a system that is open for extension but closed for modification.

2. Employee Management System (Liskov Substitution Principle)

Description**:**

- Create an employee management system that handles full-time and part-time employees.

Requirements:

- Define a base class `Employee` with an abstract method `calculatePay()`.
- Implement subclasses `FullTimeEmployee` (payment is salary) and `PartTimeEmployee` (payment is based on hourly rate and the hours worked).
- Ensure that substituting a subclass instance in place of a base class instance does not break the system.

Objectives**:**

- Practice the **Liskov Substitution Principle** by making subclasses substitutable for their base class.

3. Payment Processing Interfaces (Interface Segregation Principle)

Description:

- Design a payment processing system that supports multiple payment methods.

Requirements:

- Define separate interfaces for different payment capabilities (e.g., `ICreditCardPayment`, `IPayPalPayment`).
- Implement payment classes that only use the interfaces they need.
- Avoid forcing classes to implement methods they do not use.

Objectives:

- Apply the **Interface Segregation Principle** by creating small, client-specific interfaces.

4. Messaging Service (Dependency Inversion Principle)

Description:

- Develop a messaging service that sends notifications via different channels (Email, SMS, Push Notification).

Requirements:

- Create an interface `INotificationService` with a method `send(message)`.
- Implement concrete classes `EmailService`, `SMSService`, and `PushNotificationService`.
- The high-level module (e.g., `NotificationManager`) should depend on the `INotificationService` interface, not concrete classes.

Objectives:

- Practice the **Dependency Inversion Principle** by depending on abstractions rather than concrete implementations.

5. Order Processing System (Single Responsibility and Open/Closed Principles)

Description:

- Build an order processing system for an online store.

Requirements:

- Create an `Order` class responsible for order data.
- Implement separate classes/interfaces for order validation, payment processing, and order fulfillment.
- Allow adding new payment methods or validation rules without modifying existing code.

Objectives:

- Apply the **Single Responsibility Principle** by separating concerns.
- Use the **Open/Closed Principle** to make the system extensible.

6. Vehicle Rental Service (Interface Segregation Principle)

Description:

- Create a vehicle rental service that rents different types of vehicles (Cars, Bikes, Trucks).

Requirements:

- Define interfaces like `IDrivable`, `ICargoCarrier`, `IPassengerCarrier`.
- Implement vehicle classes that only implement the interfaces relevant to them.
- For example, a `Truck` may implement `IDrivable` and `ICargoCarrier` but not `IPassengerCarrier`.

Objectives:

- Practice the **Interface Segregation Principle** by avoiding fat interfaces.

The main method should look like this:

```java
public static void main(String[] args) {
    IDrivable car = new Car();
    IDrivable truck = new Truck();

    car.drive();
    ((Car) car).carryPassengers();

    truck.drive();
    ((Truck) truck).carryCargo();
}
```

7. Plugin System (Open/Closed Principle)

Description:

- Develop a plugin system for an application that allows adding new features through plugins.

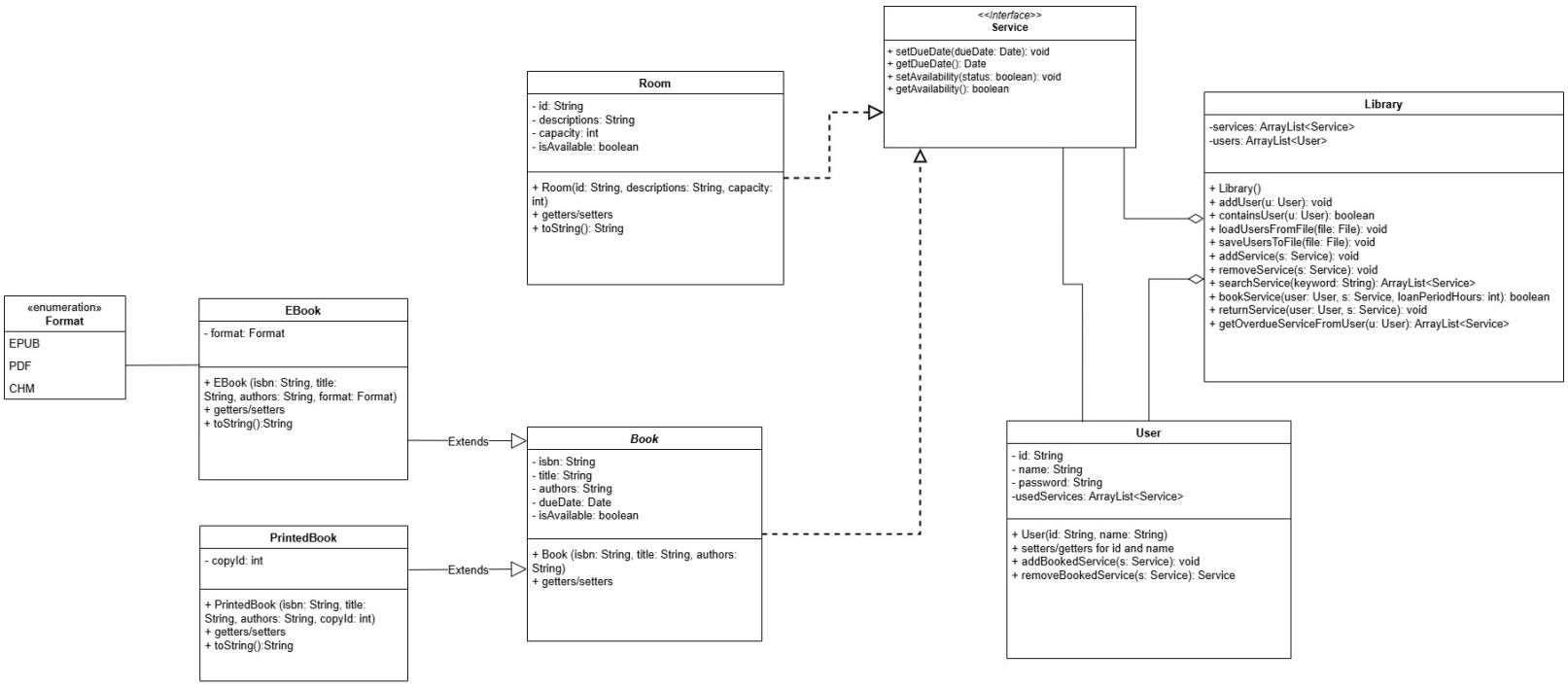
Requirements:

- Define an interface `IPlugin` with a method `execute()`.
- The application should load and execute plugins without modification.
- New plugins (`SavePlugin`, `PrintPlugin`,…) can be added by implementing the `IPlugin` interface.

Objectives:

- Apply the **Open/Closed Principle** to enable extensibility without changing existing code.

8. Library system



A library offers services including Room and Book (eBook and printed book) borrowing. The eBook has 3 file formats including EPUB, PDF and CHM. When a user wants to book a room or borrow a book for a period (days or hours), the librarian first checks if the user is in the system. If he is, then the librarian will check if the service is available, and if the user has any overdue service. If the service is not available or the user has one overdue service, the user will not be allowed to book that service.

The UML of the system has been designed in the above Figure. Implement the code for the system.

Verify the open/close principle for the system by adding a new service Tutoring. The Tutoring class should have fields to contain id, tutor name, subject, availability, and due date as well as other necessary methods.

Demonstrate the whole system in a complete program.

Submit: Must include:

- a pdf file containing your information (student id, name), and images of the class diagrams, codes, answers
- and all source code files (if any)

in a zipped (.zip or *.rar) file to Moodle