Lab 1: Áp dụng nguyên lý thiết kế SOLID trong Java

1. Mục tiêu

Sau khi hoàn thành bài lab này, sinh viên sẽ có thể:

- Hiểu và giải thích được 5 nguyên lý thiết kế SOLID.
- Nhận diện được các vi phạm SOLID trong mã nguồn.
- Tái cấu trúc (refactor) mã nguồn để tuân thủ các nguyên lý SOLID.
- Thấy được lợi ích của thiết kế phần mềm mô-đun, linh hoạt và dễ bảo trì.

2. Kiến thức nền tảng

Nguyên lý	Mô tả
S - Single Responsibility Principle (SRP)	Mỗi lớp chỉ nên có một lý do để thay đổi, tức là đảm nhận một trách nhiệm duy nhất.
O - Open/Closed Principle (OCP)	Lớp nên mở rộng được (open for extension) nhưng không cần chỉnh sửa mã gốc (closed for modification).
L - Liskov Substitution Principle (LSP)	Các lớp con có thể thay thế an toàn cho lớp cha mà không phá vỡ tính đúng đắn của chương trình.
I - Interface Segregation Principle (ISP)	Một interface không nên ép các lớp triển khai những phương thức mà chúng không cần.
D - Dependency Inversion Principle (DIP)	Các mô-đun cấp cao nên phụ thuộc vào trừu tượng, không phải phụ thuộc vào hiện thực cụ thể.

3. Các bước thực hành

Phần 1: Vi phạm nguyên lý SRP (Single Responsibility)

Đọc đoạn mã sau và cho biết vì sao nó vi phạm SRP? Sau đó refactor để tuân theo SRP.

```
class Invoice {
    private double amount;
    private String customerName;

    public Invoice(double amount, String customerName) {
        this.amount = amount;
        this.customerName = customerName;
    }

    public double getAmount() {
```

```
        return amount;
    }

    public String getCustomerName() {
        return customerName;
    }

    public void printInvoice() {
        System.out.println("Hóa đơn cho " + customerName + ": $" +
amount);
    }

    public void saveToDatabase() {
        System.out.println("Đang lưu hóa đơn vào cơ sở dữ liệu...");
    }
}
```

Gợi ý: xem xét số lượng trách nhiệm của lớp.

Phần 2: Nguyên lý OCP (Open/Closed)

Cho mã vi phạm OCP sau. Hãy cho biết khi thêm lớp **Triangle**, chuyện gì xảy ra? Sau đó refactor để tuân theo OCP.

```
class AreaCalculator {
    public double calculateArea(Object shape) {
        if (shape instanceof Rectangle) {
            Rectangle r = (Rectangle) shape;
            return r.getWidth() * r.getHeight();
        } else if (shape instanceof Circle) {
            Circle c = (Circle) shape;
            return Math.PI * c.getRadius() * c.getRadius();
        }
        return 0;
    }
}

class Rectangle {
    private double width, height;
    // getters, setters...
}

class Circle {
    private double radius;
    // getters, setters...
}
```

Gợi ý: Có cần chỉnh sửa AreaCalculator không?

Phần 3: Nguyên lý LSP (Liskov Substitution)

Tại sao đoạn mã này vi phạm LSP? Hãy refactor để tuân theo LSP.

```
class Bird {
    public void fly() {
        System.out.println("Đang bay...");
    }
}

class Ostrich extends Bird {
    @Override
    public void fly() {
        throw new UnsupportedOperationException("Đà điểu không thể
bay!");
    }
}
```

Gợi ý: Có thể thay thế Ostrich (Đà điểu) bằng Bird (Chim) một cách an toàn không?

Phần 4: Nguyên lý ISP (Interface Segregation)

Cho mã vi phạm ISP sau. Hãy refactor để nó tuân theo ISP.

```
interface Worker {
    void work();
    void eat();
}

class Robot implements Worker {
    public void work() {
        System.out.println("Robot đang làm việc...");
    }

    public void eat() {
        throw new UnsupportedOperationException();
    }
}
```

Phần 5: Nguyên lý DIP (Dependency Inversion)

Cho mã vi phạm DIP sau. Hãy refactor để tuân theo DIP.

```
class LightBulb {
    public void turnOn() { System.out.println("Bóng đèn bật"); }
```

```
    public void turnOff() { System.out.println("Bóng đèn tắt"); }
}

class Switch {
    private LightBulb bulb;

    public Switch(LightBulb bulb) {
        this.bulb = bulb;
    }

    public void operate(boolean on) {
        if (on) bulb.turnOn();
        else bulb.turnOff();
    }
}
```

4. Bài tập về nhà / Mở rộng

- Tái cấu trúc một dự án nhỏ (ví dụ: Quản lý Sinh viên) để áp dụng SOLID.
- Xác định hai vị trí trong dự án có thể áp dụng OCP hoặc DIP để cải thiện.
- Viết và kiểm thử thêm lớp mới để minh họa SOLID trong thực tế.