BÁO CÁO BÀI TẬP THỰC HÀNH

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

*Họ và tên SV:*

*MSV:*

*Lớp:*

# Phần 1

1. Giải thích lý do vi phạm nguyên lý SRP của đoạn code.  
   Lớp Invoice hiện có 3 trách nhiệm khác nhau:

* Quản lý dữ liệu hóa đơn (amount, customerName),
* In hóa đơn (printInvoice()),
* Lưu hóa đơn vào cơ sở dữ liệu (saveToDatabase()).

🡪 Khi thay đổi cách in hoặc cách lưu dữ liệu, ta phải sửa trong cùng một lớp → vi phạm SRP, vì một lớp chỉ nên có một lý do để thay đổi.

1. Refactor để code tuân thủ SRP và giải thích lý do của những việc đã làm.

Tách các chức năng in và lưu ra khỏi lớp Invoice.

* Invoice chỉ còn 1 trách nhiệm duy nhất: quản lý dữ liệu hóa đơn
* Code rõ ràng, dễ bảo trì, dễ mở rộng khi cần thay đổi cách in hoặc lưu.

1. Nộp file code Lab1SRP.java lên Classroom

# Phần 2

1. Cho biết điều gì xảy ra khi thêm lớp Triangle

Khi thêm Triangle, ta phải sửa đổi lớp AreaCalculator để thêm điều kiện instanceof Triangle.  
🡪 Code bị phụ thuộc vào loại hình cụ thể.

🡪 vi phạm OCP, vì lớp không “đóng với sửa đổi”.

1. Refactor để code tuân thủ OCP và giải thích lý do của những việc đã làm.

Tạo interface Shape có phương thức area().  
🡪 Mỗi hình tự định nghĩa cách tính diện tích riêng, không cần sửa AreaCalculator.  
🡪 Mở rộng được mà không sửa code cũ.

1. Nộp file code Lab1OCP.java (có code minh họa việc thêm lớp Triangle) lên Classroom

# Phần 3

1. Giải thích tại sao code vi phạm LSP

Ostrich kế thừa Bird nhưng không thể bay, nên override fly() để ném lỗi.  
🡪 Khi thay Bird bằng Ostrich, chương trình lỗi.  
🡪 Không thể thay thế an toàn.

1. Refactor để code tuân thủ LSP và giải thích lý do của những việc đã làm.

Phân tách hành vi bay Flyable khỏi lớp Bird.  
🡪 Chỉ những loài có thể bay mới implement Flyable.  
🡪 Đảm bảo rằng mọi lớp con có thể thay thế lớp cha mà không gây lỗi.

1. Nộp file code Lab1LSP.java lên Classroom

# Phần 4

1. Giải thích tại sao code vi phạm ISP

* Interface Worker chứa cả work() và eat().
* Robot không cần ăn, nhưng vẫn bị ép cài đặt eat().

1. Refactor để code tuân thủ ISP và giải thích lý do của những việc đã làm.

Tách interface lớn thành nhiều interface nhỏ, đúng với từng loại đối tượng.  
🡪 Human thực hiện cả 2 hành vi, Robot chỉ cần làm việc.  
🡪 Giúp code linh hoạt, dễ mở rộng, không ép buộc vô lý.

1. Nộp file code Lab1ISP.java lên Classroom

# Phần 5

1. Giải thích tại sao code vi phạm DIP

* Lớp Switch phụ thuộc trực tiếp vào lớp cụ thể LightBulb, không qua abstraction.
* Nếu thay Fan, Heater 🡪 phải sửa lại code trong Switch.

1. Refactor để code tuân thủ DIP và giải thích lý do của những việc đã làm.

* Tạo interface Switchable làm lớp trừu tượng trung gian.
* Switch chỉ phụ thuộc vào Switchable, không cần biết đối tượng cụ thể là gì.

🡪 **Dễ mở rộng**, thay đèn bằng quạt hoặc thiết bị khác mà không cần sửa Switch.

1. Nộp file code Lab1DIP.java lên Classroom