Phân tích thiết kế hướng đối tượng

Bài 16: Các nguyên tắc thiết kế

TS. Nguyễn Hiếu Cường
Bộ môn CNPM, Khoa CNTT
Trường ĐH GTVT
cuonggt@gmail.com

Hà Nội, 2018

Các nguyên tắc SOLID

- **S**ingle resposibility
- **O**pen/close
- **L**iskov substitution
- **I**nterface segregation
- **D**ependecy inversion

Single responsibility

- Mỗi lớp chỉ nên có một trách nhiệm
- Nếu lớp có nhiều hơn một trách nhiệm nên tách thành nhiều lớp
- Lợi ích
  - Dễ hiểu
  - Dễ bảo trì
  - Dễ sử dụng lại

Open-closed principle

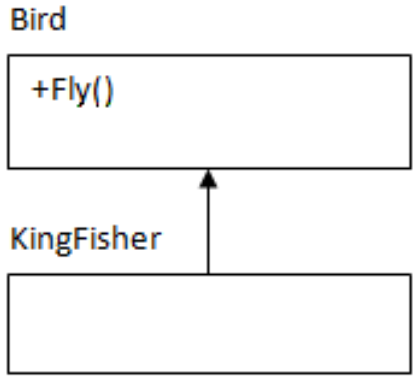
- Một lớp nên mở (open) cho việc mở rộng nhưng đóng (close) cho việc sửa đổi
- Có thể mở rộng một lớp (bằng kế thừa) nhưng không nên thay đổi lớp đã có
- Lợi ích
  - Lớp bền vững hơn
  - Khả năng sử dụng lại cao hơn

Liskov substitution principle

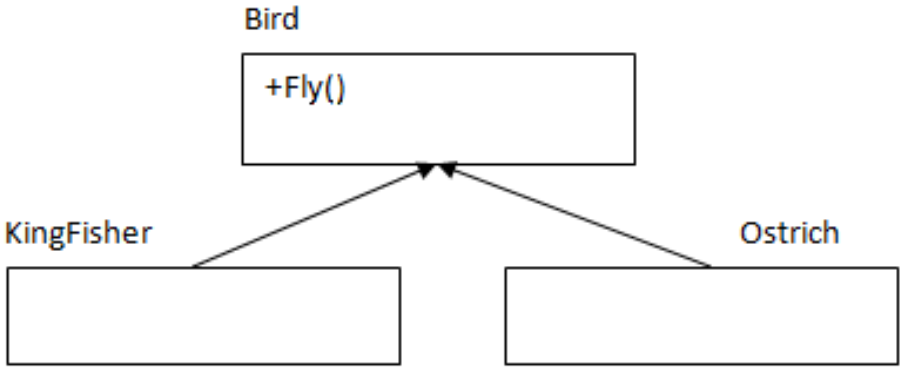
- Các kiểu dẫn xuất phải có thể thay thế cho các kiểu cơ sở của nó
- Đối tượng của lớp dẫn xuất (lớp con) có thể thay thế đối tượng của lớp cơ sở (lớp cha) của chúng
- Lợi ích
  - Đảm bảo sự kế thừa được thực hiện đúng
- Ví dụ (hình bên)
  - Các class Vịt_Bầu, Vịt_Xiêm kế thừa lớp Vịt, và chương trình chạy bình thường.
  - Nếu có lớp Vịt_Pin kế thừa lớp Vịt, nhưng **cần pin mới chạy** thì sẽ vi phạm nguyên tắc Liskov?

Ví dụ

- Thỏa mãn nguyên tắc thay thế Liskov



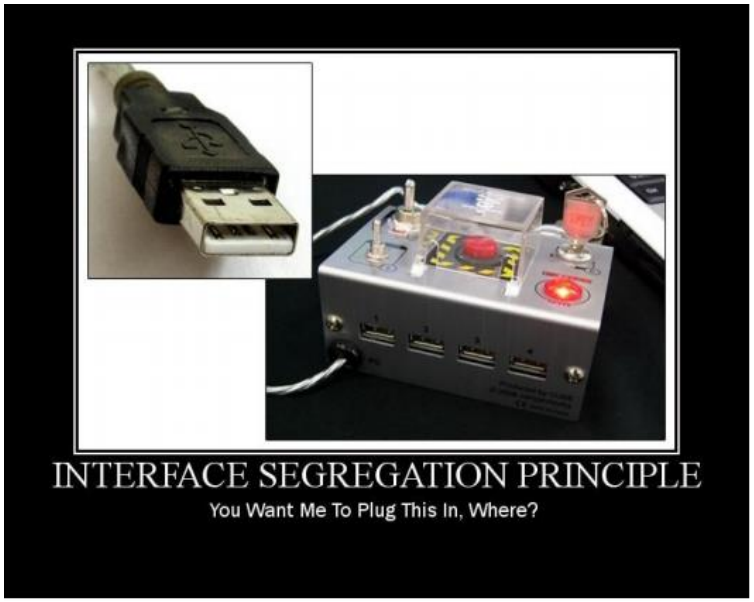
- Vi phạm nguyên tắc trên
  - Do Ostrich là chim nhưng nó không thể bay



Kingfisher: chim bói cá
Ostrich: chim đà điểu

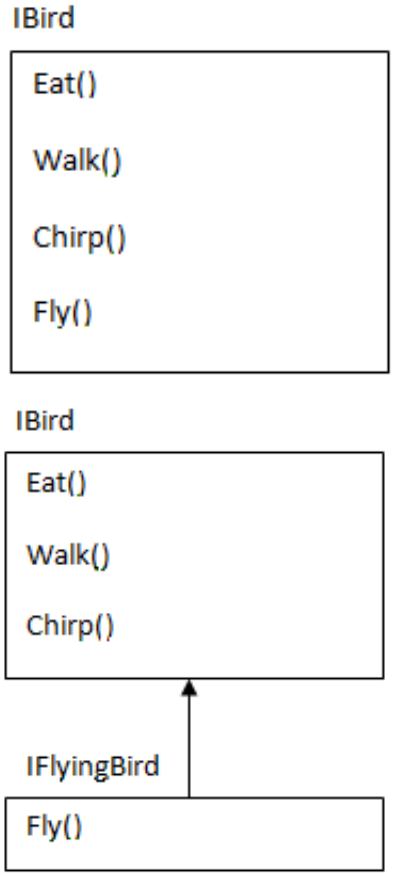
Interface segregation principle

- Khách hàng không nên bị bắt phụ thuộc vào những interface mà họ không dùng
- Nếu có một interface lớn (fat interface) thì cần tách thành nhiều interface nhỏ hơn, với những mục đích cụ thể
- Lợi ích
  - Không bị phân tán bởi những gì cần thiết



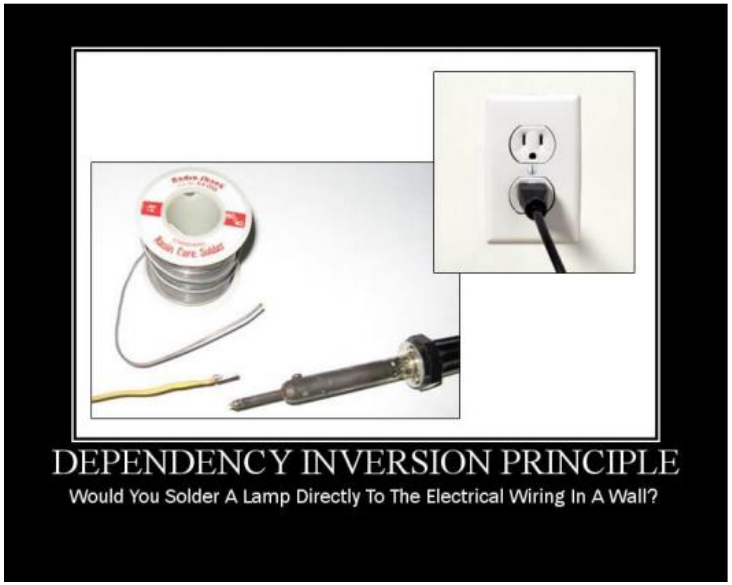
Ví dụ

- Như ví dụ bên là một "fat interface"
  - Một số loại chim không bay được cũng cần phải cài đặt Fly()
  - Vi phạm nguyên tắc Interface segregation
- Giải pháp?
- Một thiết kế tốt hơn
  - Phù hợp nguyên tắc ISP



Dependency inversion principle

- Các thành phần mức cao không nên phụ thuộc vào các thành phần ở mức thấp
- Giao diện không nên phụ thuộc cài đặt (implementation), mà ngược lại
- Ví dụ
  - Các thiết bị không nối trực tiếp vào hệ thống điện theo các cách khác nhau
  - Các thiết bị khác nhau (implementation) nhưng đều có cùng cách giao tiếp (interface) với hệ thống điện



Ví dụ

- Máy tính của chúng ta hoạt động được thì phải có đủ Mainboard, CPU, RAM, ổ cứng. Về phần ổ cứng, có thể dùng loại ổ SSD đời mới để chạy cho nhanh, tuy nhiên cũng có thể dùng ổ đĩa quay HDD thông thường. Nhà sản xuất Mainboard không thể nào biết bạn sẽ dùng ổ SSD hay loại HDD, tuy nhiên họ sẽ luôn đảm bảo rằng bạn có thể dùng bất cứ thứ gì bạn muốn, miễn là ổ đĩa cứng đó phải có chuẩn giao tiếp SATA để có thể gắn được vào bo mạch chủ. Ở đây chuẩn giao tiếp SATA chính là interface, còn SSD hay HDD đĩa quay là implement cụ thể.

- Giả sử một lớp trừu tượng Employee sẽ có phương thức working(), đây là một dạng abstraction, với từng loại nhân viên cụ thể thì sẽ implement hàm này khác nhau ứng với các công việc như developSoftware, testSoftware… Bằng cách này, lúc gọi xử lí, không cần biết là đối tượng của ta thuộc loại nào, ta chỉ cần gọi Employee->working() là sẽ có kết quả mong muốn. Ở đây working() là chuẩn giao tiếp…

Tóm tắt

- Các khái niệm cơ bản trong OOP **P.I.E.**
  - Polymorphism
  - Inheritance
  - Encapsulation
- Các nguyên tắc thiết kế **S.O.L.I.D.**
  - Single resposibility
  - Open/close
  - Liskov substitution
  - Interface segregation
  - Dependecy inversion