Phân tích thiết kế hướng đối tượng

Bài 15: Các nguyên tắc thiết kế

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

Hà Nội, 2019

Những khái niệm cơ bản trong OOP

- Polymorphism: Tương ứng bội (đa hình)
- Inheritance: Kế thừa
- Encapsulation: Bao gói (đóng gói)
- Abstraction: Trừu tượng hóa

Các nguyên tắc SOLID

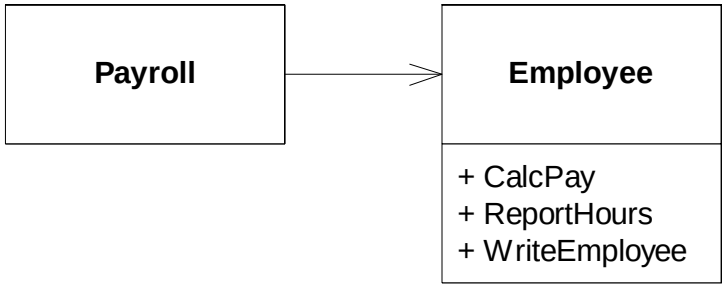
- SRP: The Single Responsibility Principle
- OCP: The Open/Closed Principle
- LSP: The Liskov Substitution Principle
- ISP: The Interface Segregation Principle
- DIP: The Dependency Inversion Principle

Single responsibility

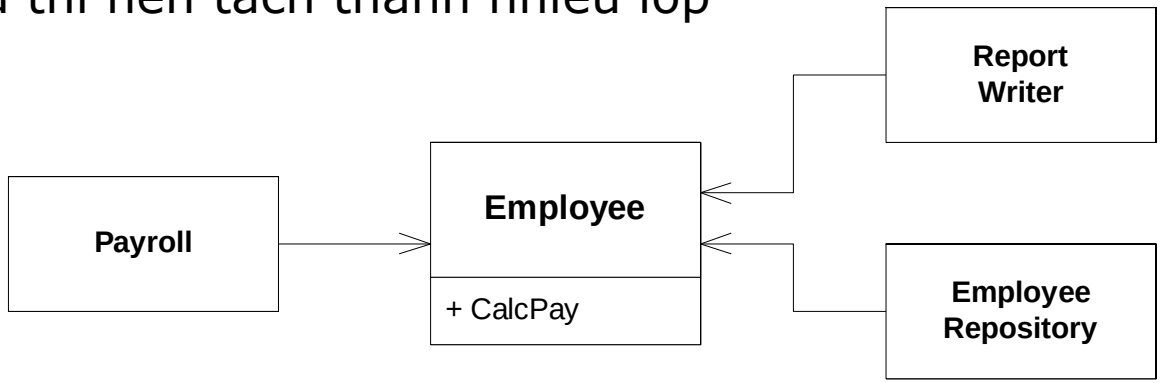
A class should have one, and only one, reason to change

- 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 và sử dụng lại

Ví dụ



- Để đáp ứng nguyên tắc SRP: Nếu lớp Employee giữ 3 trách nhiệm khác nhau thì nên tách thành nhiều lớp



Open-closed principle

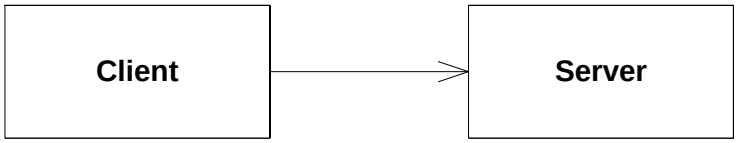
Modules should be open for extension, but closed for modification

- Một lớp nên mở (open) cho việc mở rộng nhưng đóng (close) cho việc sửa đổi

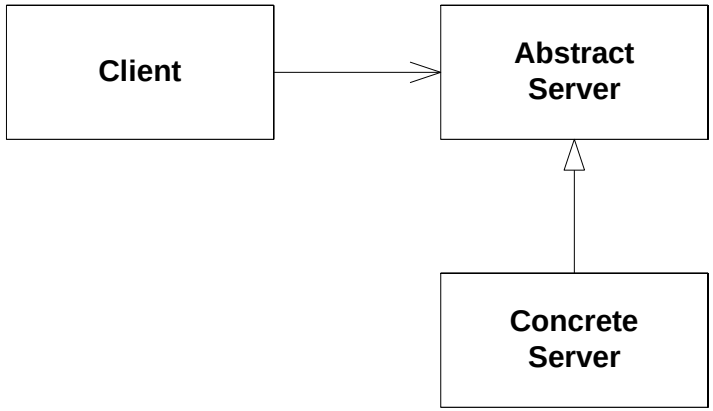
- Lợi ích
  - Lớp bền vững hơn
  - Khả năng sử dụng lại cao hơn

Ví dụ

- Quan hệ client/server là "open" với sự thay đổi
  - Thay đổi ở Server dẫn đến thay đổi ở Client
  - Vi phạm nguyên tắc OCP



- Thêm Abstract Server
  - Phần cài đặt cụ thể (Concrete Server) có thể thay đổi mà không cần thay đổi Client
  - Đáp ứng nguyên tắc OCP



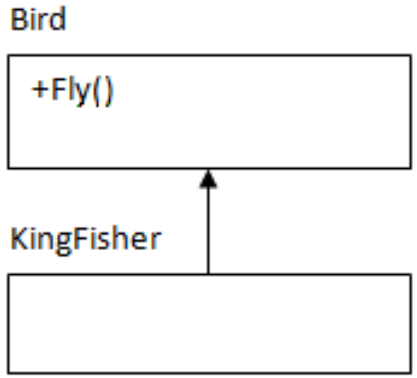
Liskov substitution principle

Derived classes must be usable through the base class interface, without the need for the user to know the difference

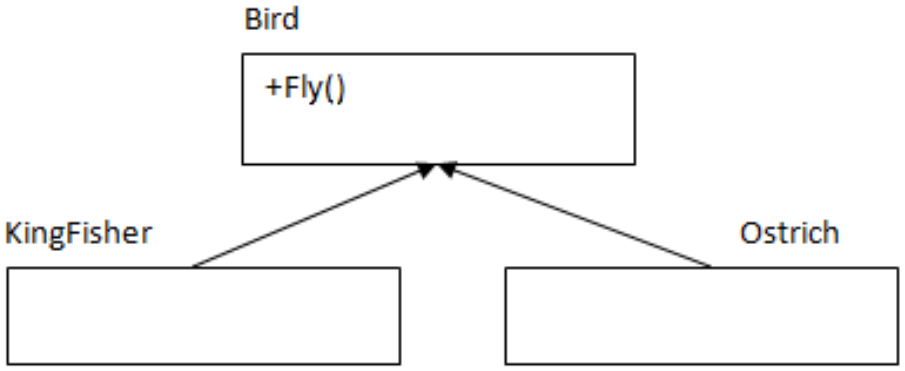
- Đối tượng của lớp dẫn xuất có thể thay thế đối tượng của lớp cơ sở của chúng
- Lợi ích
  - Đảm bảo sự kế thừa được thực hiện đúng

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

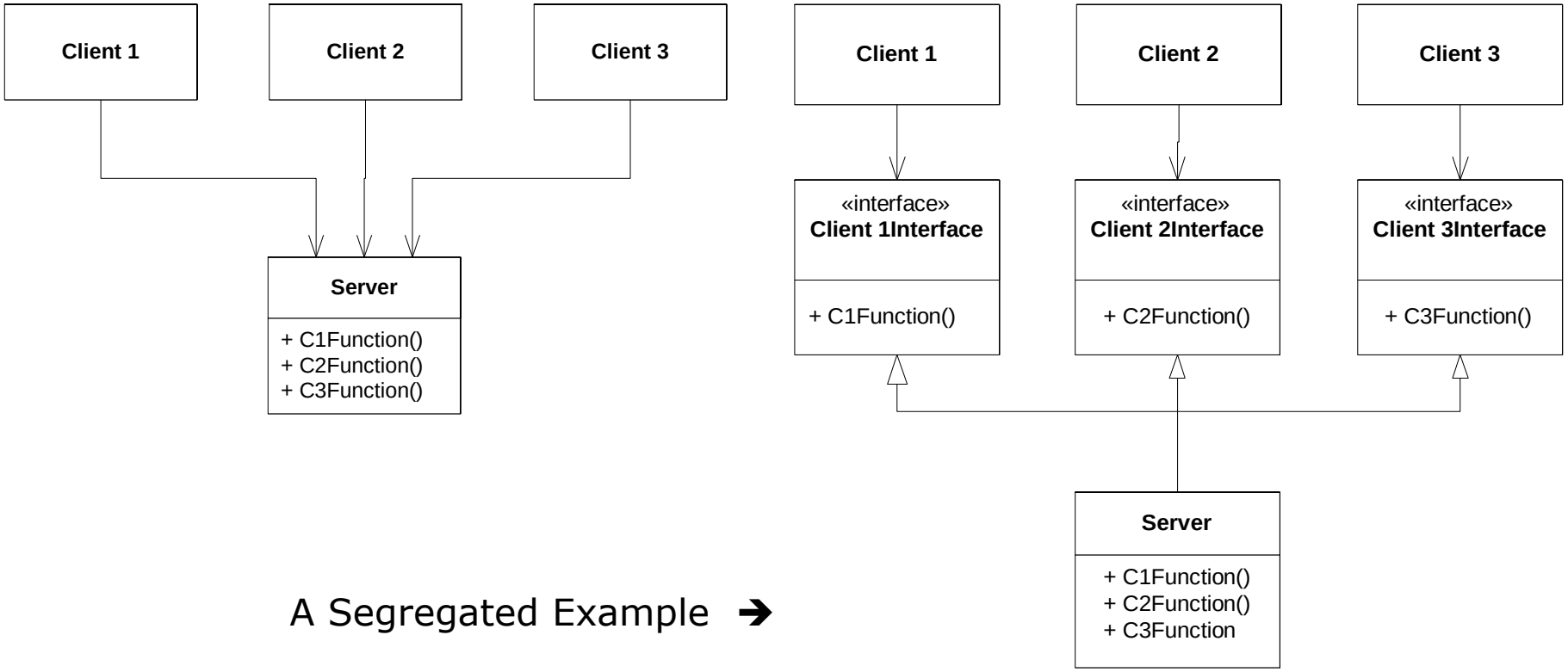
Interface segregation principle

Distinct clients of our class have distinct interface needs

(clients should not be forced to depend upon interface members they do not use. When we have non-cohesive interfaces, the ISP guides us to create multiple, smaller, cohesive interfaces)

- Nếu có một interface lớn (fat interface) thì cần tách thành nhiều interface, với những mục đích cụ thể
- Lợi ích
  - Không bị phân tán bởi những gì không cần thiết

Ví dụ

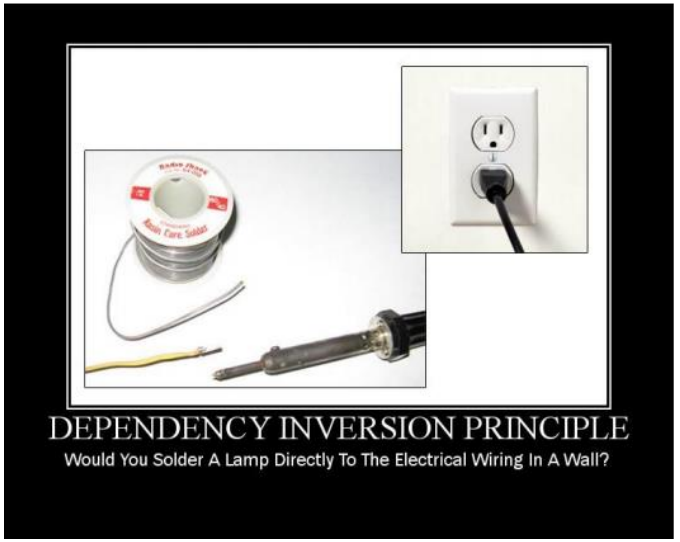


A Segregated Example ➔

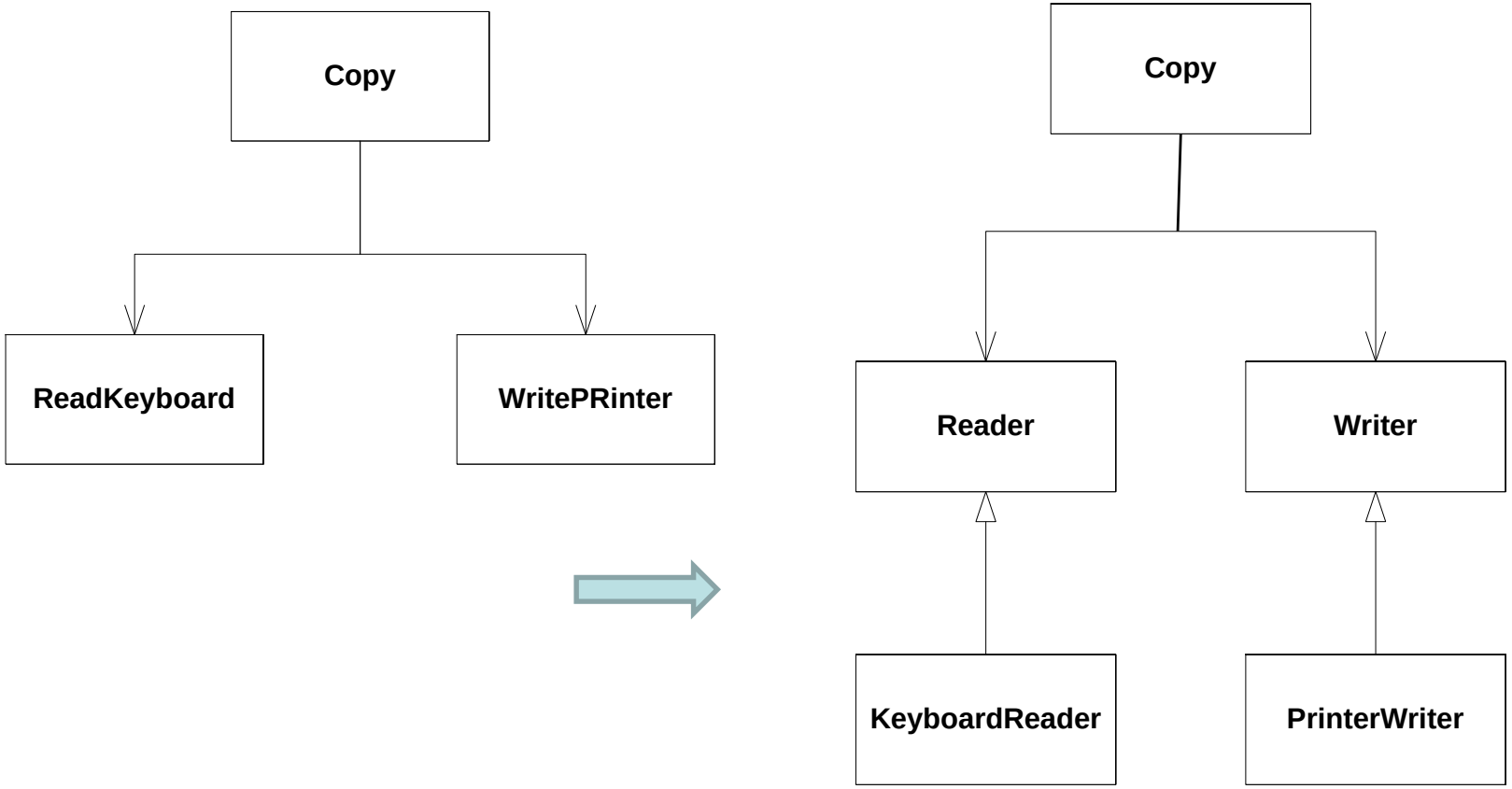
Dependency inversion principle

Details should depend on abstractions.
Abstractions should not depend on details

- Các thành phần mức cao không nên phụ thuộc vào các thành phần ở mức thấp hơn
- Giao diện (interface) không nên phụ thuộc cài đặt (implementation)
- 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 dùng cùng loại ổ cắm (interface)



Ví dụ

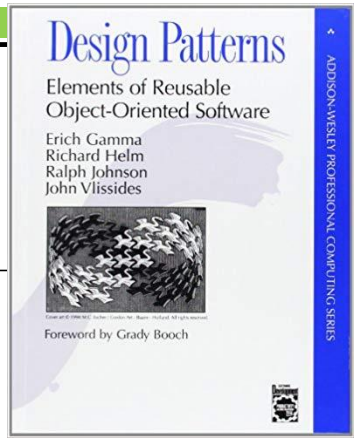

Copy
ReadKeyboard
WritePRinter
Copy
Reader
Writer
KeyboardReader
PrinterWriter

Các mẫu thiết kế

Mẫu thiết kế (Design patterns)

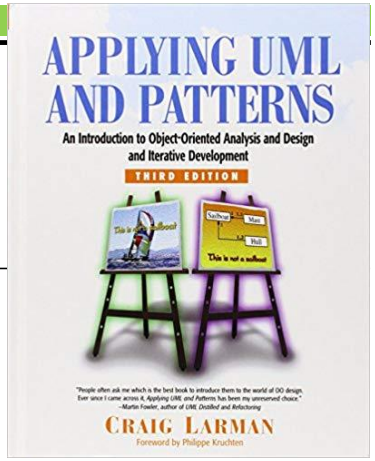
- Một mẫu là một cặp ***(vấn đề, lời giải)*** có thể áp dụng trong các tình huống, ngữ cảnh khác nhau
- Mỗi mẫu thiết kế gồm các phần chính:
  - Tên mẫu
  - Vấn đề (Problem)
  - Giải pháp (Solution)

Lịch sử mẫu thiết kế



- Sau khi cuốn "Design Patterns…" xuất bản (1995)
  - Gồm 23 mẫu thiết kế của 4 tác giả: Erich Gamma, Richard Helm, Ralph Johnson, và John Vlissides;
- Các mẫu này còn được gọi là mẫu GoF (nhóm Gang of Four)
  - Khoảng một nửa số bộ mẫu này có nguồn gốc từ luận án tiến sĩ của Erich Gamma
  - Các tác giả gặp nhau tại các hội nghị OOPSLA'91 và OOPSLA'92 (Object-Oriented Programming Systems, Languages, and Applications Conference)
  - Sau đó cùng làm việc để cùng soạn lại một bộ gồm 23 mẫu và trình bày tại hội nghị ECOOP'93 (European Conference on Object-Oriented Programming).

Mẫu GRASP



- Ngoài GoF có nhiều loại mẫu khác
- Các mẫu GRASP (by Craign Larman)
  - GRASP - **G**eneral **R**esposibility **A**ssignment **S**oftware **P**atterns
- Một số mẫu GRASP
  - Low Coupling
    Support low dependency and increased reuse
  - High Cohesion
    How to keep complexity manageable?
  - Information Expert
    Who, in the general case, is responsible?
  - Creator
    Who creates?
  - Polymorphism
    Who, when behavior varies by type?

Coupling

- Sự phụ thuộc của một lớp vào các đối tượng của các lớp khác
- Kết nối là cần thiết nếu muốn các đối tượng có thể truyền thông điệp cho nhau
- Nếu các lớp có kết nối mạnh
  - Dễ bị tác động bởi các lớp liên quan
  - Khó hiểu, khó bảo trì và khó sử dụng lại
- *Làm sao để duy trì kết nối yếu?*

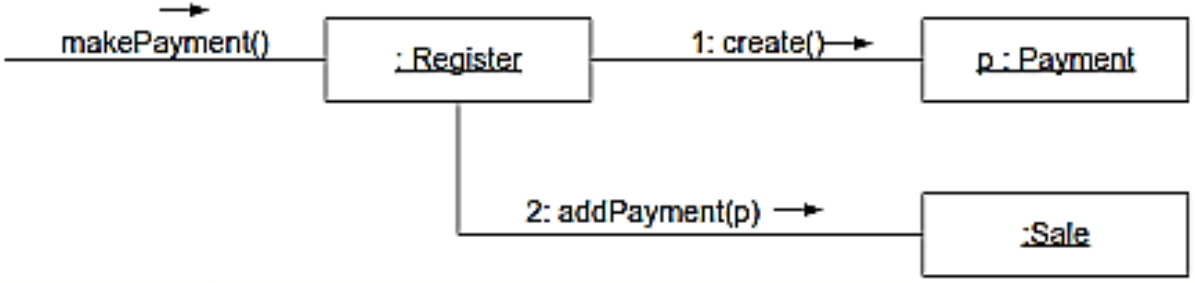
Low coupling

- Problem
  - Làm sao để hỗ trợ kết nối giữa các thành phần là yếu?
- Solution
  - Gán các trách nhiệm sao cho một lớp cần phải biết ít về lớp khác

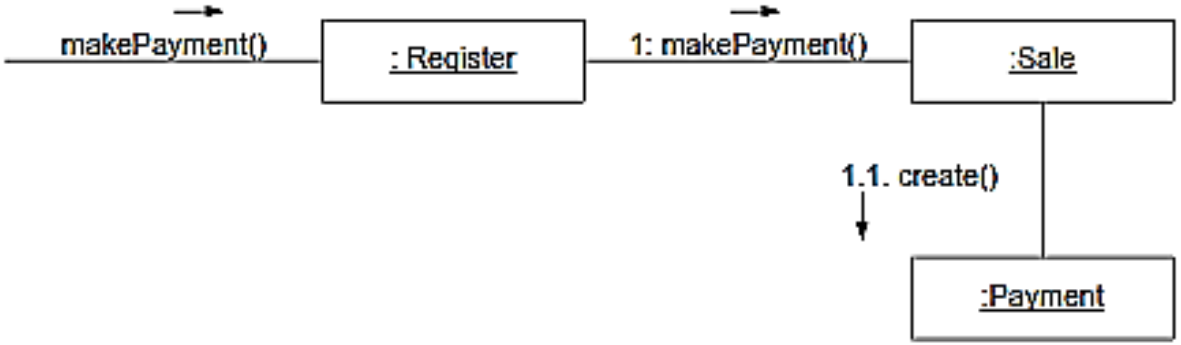
- Ví dụ:
  - Có ba lớp dưới đây, cần tạo các liên kết thế nào để có thể thanh toán

Ví dụ

- Phương án 1
  - Thêm kết nối giữa Register và Payment → làm tăng thêm kết nối



- Phương án 2
  - Kết nối yếu!

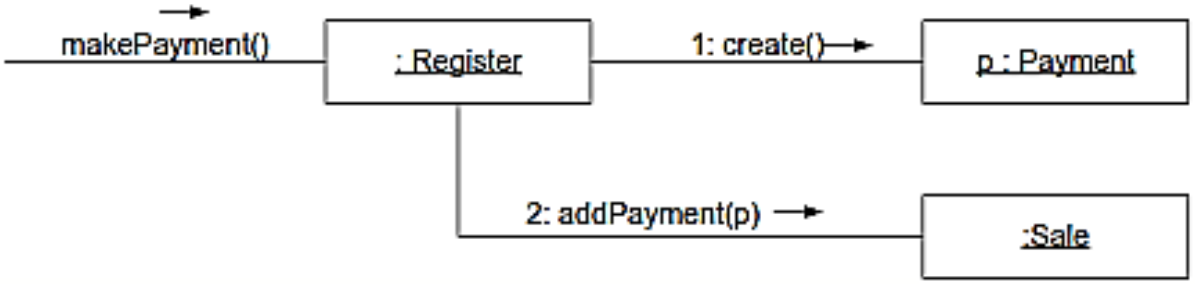


Cohesion

- Xác định mức độ liên hệ giữa các trách nhiệm trong một lớp.
- Một lớp có sự cố kết yếu
  - Có nhiều trách nhiệm không liên quan đến nhau hoặc có quá nhiều trách nhiệm
  - Lớp khó hiểu, khó bảo trì và khó sử dụng lại
- *Làm sao để duy trì cố kết mạnh cho lớp?*

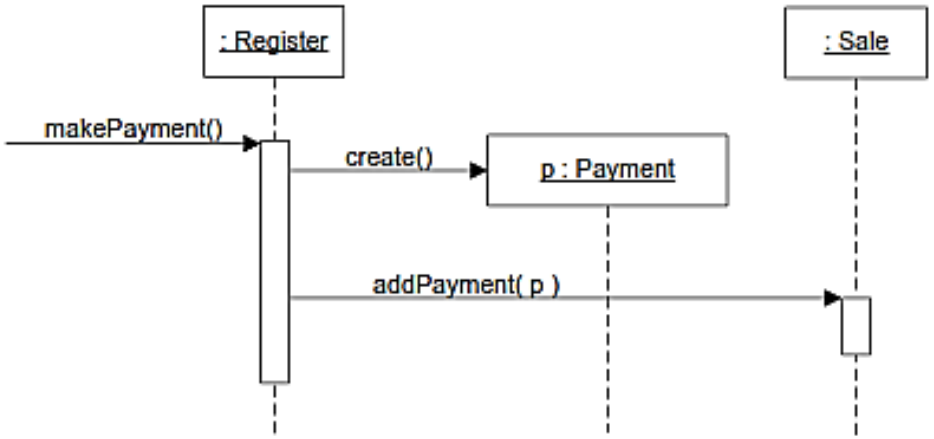
High cohesion

- Problem
  - Làm sao giữ cho lớp được cố kết mạnh?
- Solution
  - Gán trách nhiệm sao cho các lớp không phải làm quá nhiều việc hoặc nhiều việc khác nhau
- Ví dụ:
  - Register có tính cố kết không cao, cần ủy thác thanh toán cho Sale?

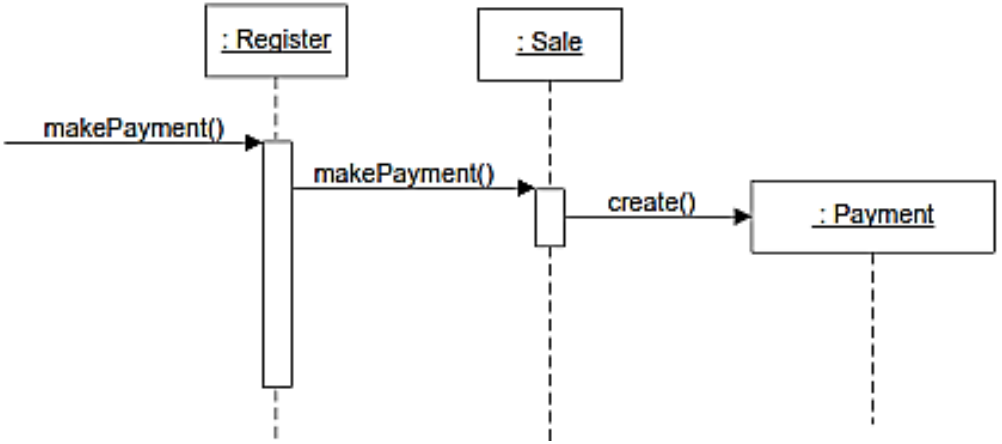


Ví dụ

- Phương án 1
  - Lớp Register làm nhiều việc



- Phương án 2
  - Ủy thác cho lớp Sale
  - Mỗi lớp có tính cố kết cao hơn



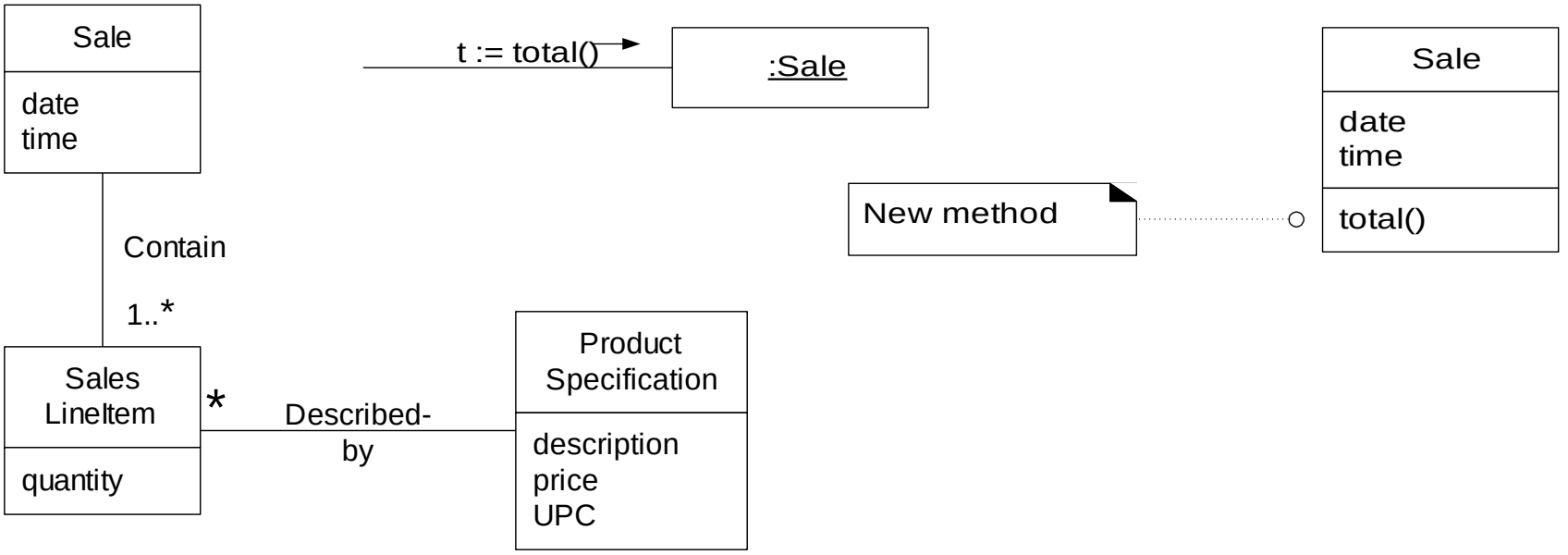
Expert

- Problem
  - Nguyên tắc chung để gán các trách nhiệm cho các đối tượng là gì?
- Solution
  - Gán trách nhiệm cho lớp có đủ *thông tin cần thiết* để hoàn thành trách nhiệm đó

Ví dụ

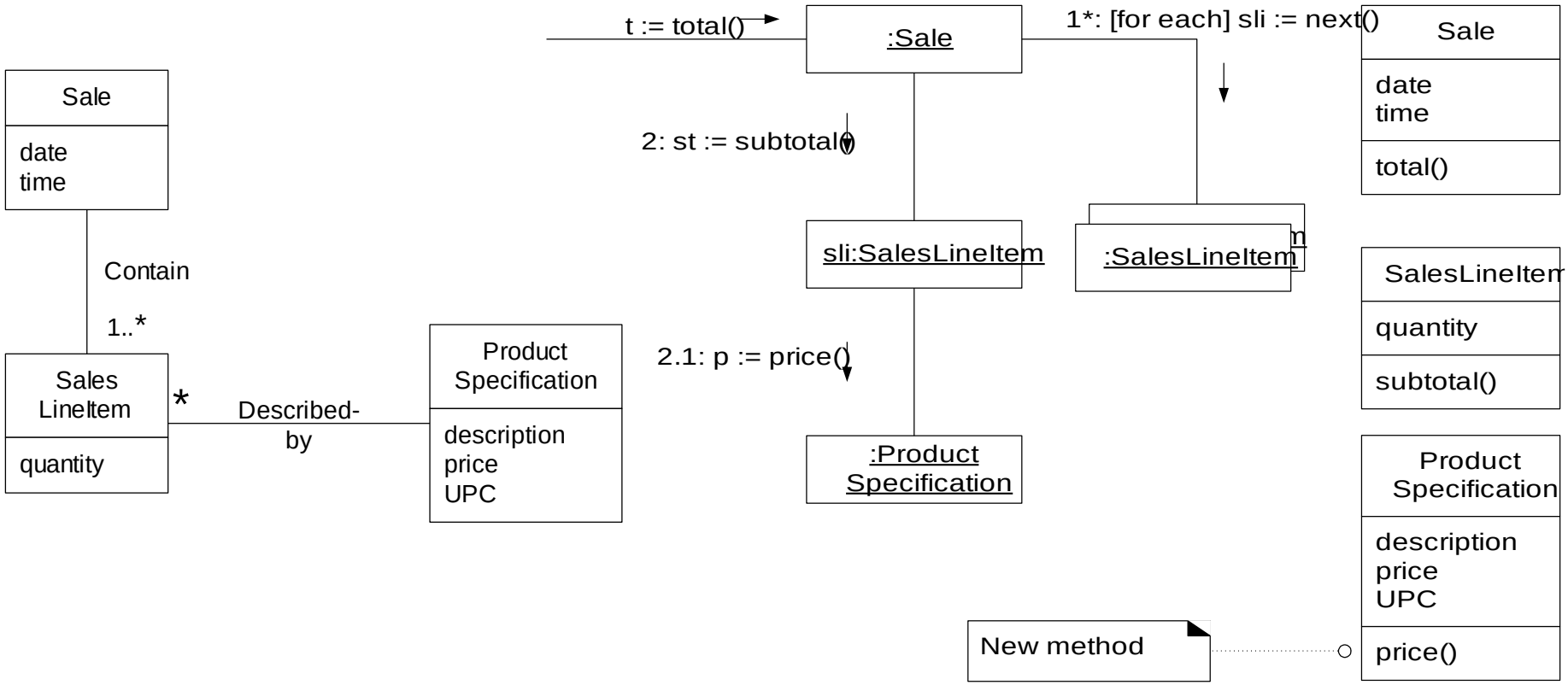
Lớp đối tượng nào có trách nhiệm (đủ thông tin) để biết **tổng số tiền** bán hàng trong ứng dụng?

Sale có đủ thông tin để tính tổng số tiền (Sale là "information expert")



Ví dụ

Lớp nào biết **giá của mỗi sản phẩm**? Trong mỗi phiên cần tính số tiền (subtotal). Lớp nào có trách nhiệm biết **số tiền bán của mỗi phiên**?



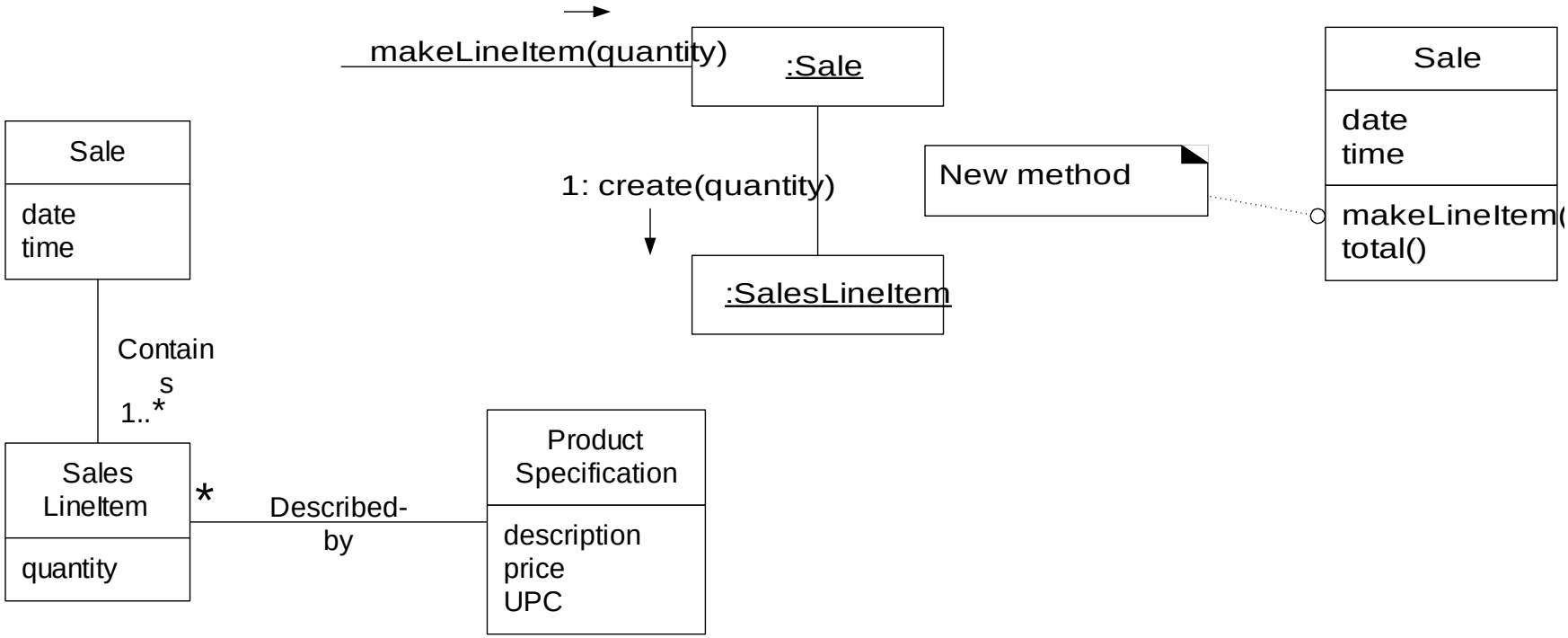
Creator

- Problem
  - "Ai" có trách nhiệm tạo ra một thực thể mới của một lớp đối tượng?
- Solution
  - Lớp A có trách nhiệm tạo một thực thể của một lớp B dựa trên quan hệ giữa lớp tạo thực thể A (creator) và lớp được tạo thực thể B
    - A bao gồm các đối tượng B
    - A sử dụng các đối tượng B

Ví dụ

Lớp nào có trách nhiệm tạo các đối tượng của SalesLineItem?

Lớp **Sale** vì: Sale *contains* SalesLineItems



Tóm tắt

- Các khái niệm cơ bản trong OOP
  - Polymorphism; Inheritance; Encapsulation
- Các nguyên tắc thiết kế SOLID
  - Single responsibility; Open/close; Liskov substitution; Interface segregation; Dependecy inversion
- Mẫu thiết kế (Design Patterns)
- Một số mẫu thiết kế GRASP