

Phân tích thiết kế Hệ thống

Giảng viên: Nguyễn Bá Ngọc

Hà Nội-2021

Thiết kế lớp

Nội dung

- Vấn đề thiết kế lớp
- Các chỉ số chất lượng
- Các nguyên lý SOLID

Nội dung

- Vấn đề thiết kế lớp
- Các chỉ số chất lượng
- Các nguyên lý SOLID

Các hoạt động thiết kế đối tượng

- Mở rộng và tiếp tục phát triển các mô hình phân tích.
- Tạo các mô tả cho thiết kế chi tiết lớp và phương thức
 - Bổ xung các đặc tả cho mô hình hiện có
 - Xác định các tiềm năng tái sử dụng
 - Tái cấu trúc thiết kế
 - Tối ưu hóa thiết kế
 - Ánh xạ các lớp lĩnh vực ứng dụng và ngôn ngữ triển khai
- Thay đổi 1 lớp trên 1 tầng có thể khiến các lớp khác (có thể trên các tầng khác) có gắn kết với lớp đó thay đổi theo.

Các đặc tả bổ xung

- Kiểm tra các mô hình cấu trúc và hành vi để đảm bảo các thành phần đầy đủ và cần thiết.
- Thiết lập giới hạn nhìn của các thuộc tính và phương thức của mỗi lớp (gắn với ngôn ngữ triển khai).
- Xác định nguyên mẫu của các phương thức: Tên, các tham số, kiểu trả về.
- Xác định các ràng buộc mà đối tượng phải đáp ứng
 - Tiền điều kiện;
 - Hậu điều kiện;
 - Tính chất bất biến.
 - *(Có thể được mô tả bằng OCL).*
- Các thông tin bổ xung được mô tả bằng hợp đồng, hoặc được thêm vào thẻ CRC và sơ đồ lớp.

Các đặc tả bổ xung₍₂₎

- Chúng ta cũng cần xác định cách xử lý các ngoại lệ:
 - *(phát sinh nếu các ràng buộc không được đảm bảo)*
 - Hệ thống có thể xử lý đơn giản bằng cách bỏ qua?
 - Hoàn tác các thay đổi đã thực hiện dẫn đến ngoại lệ?
 - Để người dùng lựa chọn phương án xử lý ngoại lệ nếu có nhiều phương án?
- Người thiết kế xác định các lỗi mà hệ thống phải xử lý
 - Gắn kết với ngôn ngữ lập trình
 - Sử dụng mã lỗi
 - Sử dụng cơ chế exception (như trong C++ hoặc Java)

Xác định các tiềm năng tái sử dụng

- Trong pha thiết kế chúng ta có thể sử dụng:
 - Mẫu thiết kế;
 - Nền tảng;
 - Thư viện;
 - Thành phần.
- Tiềm năng tái sử dụng có thể thay đổi theo tầng
 - Ví dụ: 1 thư viện có thể ít hữu ích trong tầng lĩnh vực ứng dụng, nhưng lại rất hữu ích trong tầng nền tảng.
- Các mẫu thiết kế
 - Rất hữu ích để gom nhóm các lớp tương tác có thể cung cấp giải pháp cho 1 vấn đề phổ biến. Có thể giải quyết “1 vấn đề thiết kế diễn hình trong 1 ngữ cảnh cụ thể”.
 - *(Các chi tiết sẽ được cung cấp sau).*

Nền tảng và thư viện

- Nền tảng cung cấp 1 tập lớp có thể được sử dụng làm khung phát triển chương trình ứng dụng
 - Có thể triển khai toàn bộ hoặc 1 phần của hệ thống.
 - CORBA, DCOM, Spring Boot, Struts, Qt, v.v..
- Thư viện bao gồm 1 tập lớp, tương tự nền tảng, được thiết kế để tái sử dụng
 - Có nhiều thư viện hỗ trợ các mảng ứng dụng
 - Ví dụ: Xử lý số học và thống kê; Phát triển giao diện đồ họa (tầng HCI); Kết nối với CSDL (tầng quản lý dữ liệu - DAM).
- Nền tảng và thư viện thường cho phép kế thừa các lớp của nó, có thể kế thừa để tái sử dụng lớp hoặc tạo đối tượng trực tiếp từ các lớp đã có.
 - *(Làm tăng tính ghép nối).*

Thành phần

- Thành phần/component:
 - Mảnh phần mềm, được đóng gói để có thể nhúng vào 1 hệ thống để cung cấp 1 tập chức năng cụ thể.
 - Ví dụ các thành phần được triển khai dựa trên các công nghệ ActiveX hoặc JavaBean.
 - Cung cấp 1 tập phương thức giao diện để tương tác với các đối tượng có trong nó.
 - Lô-gic hoạt động bên trong thành phần được ẩn sau các API.
 - Có thể được triển khai bằng các lớp thư viện và nền tảng, nhưng cũng có thể được sử dụng để triển khai nền tảng.
 - Nếu giao diện không thay đổi, thì cập nhật phiên bản mới của thành phần thường không yêu cầu thay đổi mã nguồn
 - Có thể không yêu cầu biên dịch lại.
 - Thường được sử dụng để giảm lược các chi tiết triển khai.

Tái cấu trúc

- Đóng gói phần chung / factoring
 - Lớp mới có thể được gắn kết với các lớp cũ thông qua kế thừa, tổng hợp hoặc liên kết
 - Lưu ý các vấn đề ghép nối, thống nhất, và đồng sinh.
- Triển khai các mối quan hệ trên sơ đồ lớp như những thuộc tính.
 - Các ngôn ngữ hướng đối tượng hầu như không phân biệt các quan hệ như bộ phận-tổng thể và liên quan.
 - Các mối quan hệ phải được chuyển thành các thuộc tính trong lớp.

Đóng gói phần chung

- Tách và đóng gói phần giống nhau và khác nhau của các thứ được quan tâm
- Các lớp mới được hình thành thông qua:
 - Quan hệ khái quát hóa (thuộc loại), hoặc
 - Tạo lớp bậc cao hơn (ví dụ, tạo lớp Employee từ tập vị trí công việc)
 - Chi tiết hóa, hoặc
 - Tạo lớp cụ thể (ví dụ, tạo lớp Secretary hoặc Bookkeeper từ lớp Employee)
 - Quan hệ tổng hợp (có các thành phần)

Tối ưu hóa thiết kế

Có thể được thực hiện để tạo thiết kế hiệu quả hơn

- Kiểm tra các đường dẫn tới đối tượng:
 - Có những trường hợp thông điệp từ 1 đối tượng tới 1 đối tượng khác phải qua nhiều bước trung gian.
 - Nếu đường dẫn dài và thông điệp được gửi thường xuyên, thì cân nhắc rút ngắn đường truyền thông điệp.
 - Có thể bổ xung thuộc tính vào đối tượng gửi thông điệp để có thể truy cập trực tiếp.

Tối ưu hóa thiết kế: Thuộc tính

- Xác định những phương thức sử dụng thuộc tính và những đối tượng sử dụng phương thức.
- Nếu chỉ có phương thức đọc và cập nhật sử dụng thuộc tính, và chỉ có đối tượng thuộc 1 lớp gửi thông điệp đọc và cập nhật đối tượng
 - Thuộc tính đó có thể thuộc về lớp gọi thay vì lớp được gọi
 - Chuyển thuộc tính sang lớp gọi có thể giúp hệ thống hoạt động nhanh hơn.

Tối ưu hóa thiết kế: Luồng ra

- Luồng ra bao gồm các thông điệp được gửi trực tiếp và gián tiếp trong thời gian thực hiện phương thức
 - Thông điệp trực tiếp được gửi bởi chính phương thức đó
 - Thông điệp gián tiếp được gửi bởi phương thức được gọi trong khi thực hiện phương thức đó
- Nếu kích thước luồng ra quá lớn so với các phương thức khác trong hệ thống, thì phương thức đó có thể cần được điều chỉnh.

Tối ưu hóa thiết kế: Thứ tự thực hiện

- Kiểm tra thứ tự thực hiện các lệnh trong phương thức được sử dụng thường xuyên
 - Có những trường hợp có thể sắp xếp lại các câu lệnh để đạt được hiệu quả cao hơn.
 - Ví dụ: Giả định kịch bản tìm kiếm, trong đó phạm vi tìm kiếm được thu hẹp sau khi tìm kiếm theo 1 thuộc tính.
 - Có thể tìm cách tối ưu hóa xử lý theo 1 thứ tự thuộc tính tối ưu.
 - $A \cap B \cap C$ vs. $A \cap C \cap B$

Tối ưu hóa thiết kế: Sử dụng bộ nhớ đệm

- Tránh tính toán lại thuộc tính suy diễn:
 - Ví dụ tạo thuộc tính tổng/total để lưu tổng giá trị đơn hàng.
 - Thiết lập cơ chế kích hoạt tiến trình tính toán.
 - Chỉ tính lại giá trị của thuộc tính suy diễn khi thay đổi các thuộc tính thành phần được sử dụng trong tính toán.

Ảnh xạ các lớp lĩnh vực ứng dụng

- Tái cấu trúc các thành phần đa kế thừa nếu ngôn ngữ chỉ hỗ trợ đơn kế thừa.
- Tái cấu trúc các thành phần kế thừa nếu ngôn ngữ không hỗ trợ kế thừa.
- Tránh triển khai 1 thiết kế hướng đối tượng với 1 ngôn ngữ lập trình không hỗ trợ lập trình hướng đối tượng.

Các ràng buộc và các hợp đồng

- Hợp đồng bao gồm 1 tập ràng buộc/constraints và các đảm bảo / guarantees
 - Nếu đối tượng gửi (client) đáp ứng các ràng buộc, thì đối tượng cung cấp dịch vụ (server) đảm bảo hành vi mong đợi
- Hợp đồng mô tả thông điệp được gửi giữa các đối tượng
 - Được tạo cho các phương thức công khai của lớp.
 - Cần chứa đủ thông tin để người lập trình có thể hiểu hành vi mong đợi của phương thức.
- Các loại ràng buộc:
 - Tiền điều kiện - Phải đúng trước khi phương thức được thực hiện
 - Hậu điều kiện - Phải đúng sau khi phương thức kết thúc
 - Bất biến - Phải luôn đúng đối với tất cả các đối tượng.

Biểu diễn bất biến

Mặt trước:

| | | |
|--------------------------------------|--------------|---|
| Tên lớp: Order | ID: 1 | Kiểu: Cụ thể, Lĩnh vực |
| Mô tả: Biểu diễn đơn hàng | | Ca sử dụng liên quan: 1, 2, 3, ... |
| <u>Các trách nhiệm</u> ... | | <u>Các đối tác</u> ... |

Mặt sau:

Các thuộc tính:

order_id: long

customer: Customer

cust_id: long {cust_id = customer.getCustID()}

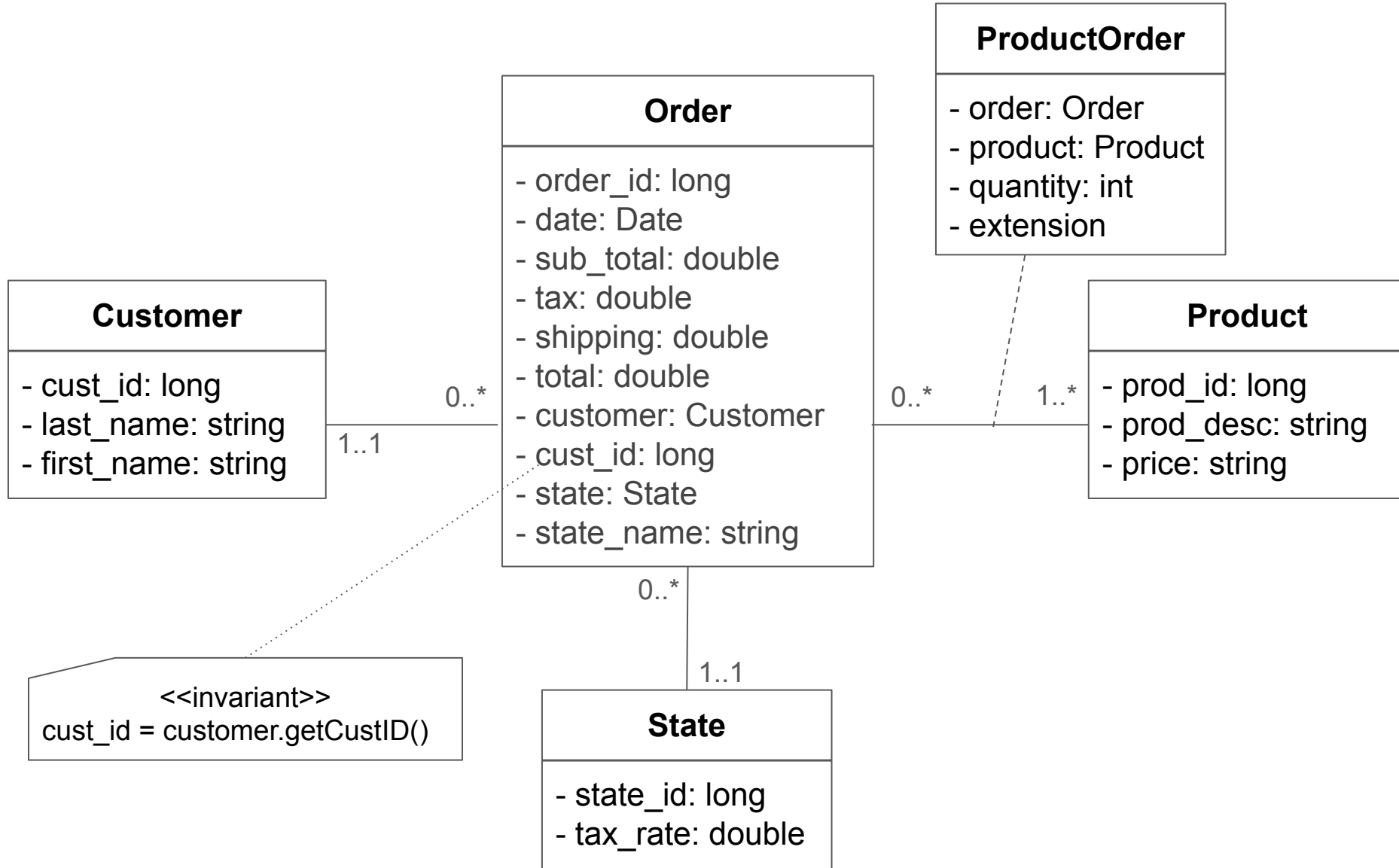
Các mối quan hệ:

Khái quát hóa (thuộc loại):

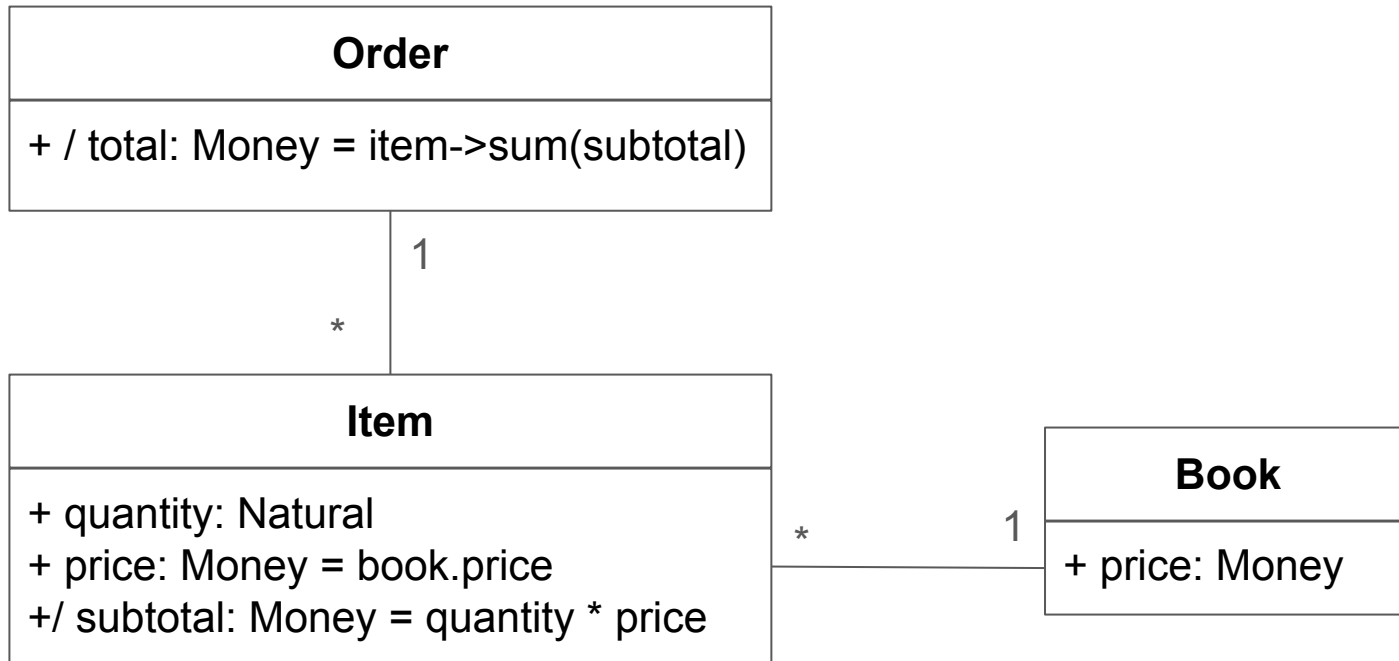
Tổng hợp (Có các phần):

Các mối liên quan khác:

Ví dụ 5.1. Biểu diễn bất biến trên sơ đồ



Ví dụ 5.2. Biểu diễn bất biến ngoài sơ đồ



Context Order::total:Money
derive:
self.item->sum(subtotal)

Context Item::subtotal:Money
derive:
self.quantity * self.price

Các mô tả
OCL

Biểu mẫu hợp đồng thông điệp

(Phi chuẩn, đặc tả thông điệp)

| | | |
|------------------------|----------|--------|
| Tên phương thức: | Tên lớp: | Mã số: |
| Mã khách: | | |
| Ca sử dụng liên quan: | | |
| Mô tả các trách nhiệm: | | |
| Các tham số nhận được: | | |
| Kiểu dữ liệu trả về: | | |
| Tiền điều kiện: | | |
| Hậu điều kiện: | | |

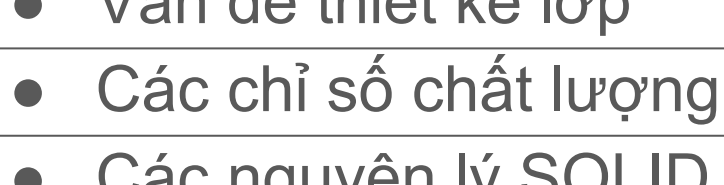
Đặc tả phương thức

- Mô tả các chi tiết của mỗi phương thức
 - Để người lập trình viết mã nguồn
 - *(đây là tài liệu thiết kế rất gần triển khai)*
- Không có quy chuẩn, tuy nhiên có thể bao gồm các mục:
 - Thông tin tổng quan (Tên phương thức, tên lớp, v.v...)
 - Sự kiện - Kích hoạt phương thức (ví dụ khi nhấn chuột)
 - Cấu trúc thông điệp được truyền tới bao gồm cả các tham số và giá trị trả về
 - Đặc tả giải thuật
 - Các thông tin liên quan khác

Biểu mẫu đặc tả phương thức

| | | |
|--|---------------------------|-----------|
| Tên phương thức: | Tên lớp: | ID: |
| Mã thỏa thuận: | Lập trình viên: | Thời hạn: |
| Ngôn ngữ lập trình: <input type="checkbox"/> Visual Basic <input type="checkbox"/> Smalltalk <input type="checkbox"/> C++ <input type="checkbox"/> Java | | |
| Kích hoạt/Sự kiện: | | |
| Các tham số nhận được: Kiểu dữ liệu: | | Ghi chú: |
| | | |
| Thông điệp đã gửi & tham số: TênLớp.TênPhươngThức | Kiểu dữ liệu của tham số: | Ghi chú: |
| | | |
| | | |
| Tham số trả về: Kiểu dữ liệu: | Ghi chú | |
| | | |
| Đặc tả giải thuật: | | |
| Các ghi chú khác: | | |

Nội dung

- Vấn đề thiết kế lớp
 - Các chỉ số chất lượng
 - Các nguyên lý SOLID
- 

Chỉ số chất lượng thiết kế

- Ghép nối/Coupling - Phản ánh mức độ gắn kết giữa các thành phần (lớp, phương thức, v.v.).
- Thống nhất/Cohesion - Phản ánh mức độ chuyên dụng của 1 thành phần.
- Đồng sinh/Connascence - Khái quát hóa đồng thời kết hợp chỉ số ghép nối và chỉ số thống nhất.
 - *(Xuất hiện cùng nhau/Connascence means to be born together).*

Tính ghép nối

- Ghép nối chặt khiến thay đổi trong 1 phần của thiết kế có thể yêu cầu thay đổi trong các phần khác
- Phân loại
 - Ghép nối tương tác - hình thành từ trao đổi thông điệp
 - Ghép nối kế thừa - hình thành từ quan hệ kế thừa
 - [*Coad and Edward Yourdon, Object-Oriented Design, 1991*]
- Các giới hạn được áp dụng để giảm mức ghép nối
 - Ghép nối tương tác: Quy tắc Demeter
 - Ghép nối kế thừa: Duy trì ý nghĩa khái quát hóa/chi tiết hóa, áp dụng quy tắc khả thay/substitutability

Quy tắc Demeter

Thông điệp chỉ nên được gửi từ 1 đối tượng tới:

- Chính nó;
- Đối tượng là thuộc tính của nó hoặc lớp trên;
- Đối tượng là tham số của phương thức
- Đối tượng được tạo bởi phương thức
- Đối tượng được lưu trong biến toàn cục

Các trường hợp làm tăng mức độ ghép nối


- Phương thức gọi truyền các thuộc tính cho phương thức được gọi
- Phương thức gọi phụ thuộc vào giá trị được trả về bởi phương thức được gọi

Các mức ghép nối tương tác

Theo thứ tự từ lỏng (được mong đợi) đến chặt (không mong muốn):

- Không có ghép nối trực tiếp / No Direct Coupling
- Dữ liệu / Data
- Dấu ấn / Stamp
- Điều khiển / Control
- Chung hoặc toàn cục / Common or Global
- Nội dung hoặc biểu diễn / Content or Pathological

Các mức ghép nối tương tác₍₂₎

| Mức độ | | Mô tả |
|--|-----------------------------|---|
| <div>Lỏng</div>  <div>Chặt</div> | Không có ghép nối trực tiếp | Các phương thức không liên kết với nhau/không gọi lẫn nhau |
| | Dữ liệu | Phía gọi truyền tham số cho phương thức được gọi. Nếu đó là 1 đối tượng bao gồm nhiều thuộc tính thì tất cả các thuộc tính đều được sử dụng bởi phương thức được gọi để thực hiện trách nhiệm của nó. |
| | Dấu ấn | Phía gọi truyền tham số, nhưng phương thức được gọi chỉ sử dụng 1 phần dữ liệu để thực hiện trách nhiệm của nó. |
| | Điều khiển | Phía gọi truyền 1 biến điều khiển được sử dụng để điều khiển phương thức được gọi. |
| | Chung hoặc toàn cục | Các phương thức dùng chung "1 vùng dữ liệu toàn cục" nằm ngoài phạm vi của đối tượng. |
| | Nội dung hoặc Biểu diễn | Phương thức của 1 đối tượng truy cập biểu diễn bên trong của đối tượng khác. Hình thành ngoại lệ đối với quy tắc đóng gói và ẩn dữ liệu. (ví dụ Lớp bạn/friend trong C++). |
| Nguồn: Meilir Page-Jones, <i>The Practical Guide to Structured Systems Design</i> , 2nd, 1978. | | |

Ghép nối kế thừa

- Có nhiều luồng quan điểm cho rằng ghép nối kế thừa là cần thiết.
- Tuy nhiên vẫn cần cân nhắc các trường hợp:
 - Phương thức được định nghĩa trong lớp dưới có nên gọi phương thức được định nghĩa trong lớp trên?
 - Phương thức được định nghĩa trong lớp dưới có nên sử dụng thuộc tính được định nghĩa trong lớp trên?
 - Phương thức được định nghĩa trong lớp trên có nên dựa vào các lớp con của nó để định nghĩa 1 phần xử lý?
 - *(Có thể triển khai dựa trên phương thức ảo và đa hình)*
- Người thiết kế cần cân đối việc phá vỡ quy tắc đóng gói và ẩn dữ liệu và tăng những ghép nối mong đợi giữa các lớp dưới và các lớp trên.

Tính ghép nối trong thiết kế

- Hướng tới ghép nối lỏng, đưa chỉ số ghép nối về mức nhỏ nhất có thể.
- Trong thực tế có những ghép nối khó tránh:
 - Ví dụ 1 lớp HCI phụ thuộc vào 1 lớp lĩnh vực ứng dụng - lớp biểu diễn dữ liệu được hiển thị trên giao diện -
 - *Biểu mẫu thông tin sinh viên hiển thị các thông tin của 1 đối tượng thuộc lớp Student trong tầng lĩnh vực ứng dụng.*


Tính thống nhất

- Mức độ thống nhất cao thể hiện tính chuyên dụng của mỗi thành phần: 1 lớp chỉ biểu diễn 1 loại đối tượng, 1 phương thức chỉ có 1 trách nhiệm.
- Phân loại:
 - Thống nhất phương thức
 - 1 phương thức có đảm nhận nhiều hơn 1 trách nhiệm?
 - Càng nhiều trách nhiệm thì càng phức tạp và khó triển khai.
 - Thống nhất lớp
 - Các thuộc tính và phương thức có biểu diễn 1 loại đối tượng?
 - Không nên trộn lẫn nhiều vai trò, loại đối tượng vào 1 lớp.
 - Thống nhất cây kế thừa
 - Quan hệ giữa các lớp trong 1 cây kế thừa nên biểu diễn nhất quán quan hệ "1 loại của"/"a-kind-of", không phải tổng hợp hoặc liên quan.
 - Các lớp tương thích với nguyên lý khả thay / Substitutability
- Hướng tới thống nhất ở mức độ cao nhất.

Thống nhất phương thức

- Thể hiện tính thống nhất trong 1 phương thức
 - Các lệnh trong phương thức cùng xử lý 1 công việc.
 - Mỗi phương thức chỉ nên làm 1 việc - lãnh 1 trách nhiệm
- Các mức thống nhất phương thức theo thứ tự giảm dần từ cao (được mong đợi) đến thấp (không mong muốn)
 - Chức năng / Functional
 - Tuần tự / Sequential
 - Giao tiếp / Communicational
 - Tiến trình / Procedural
 - Tạm thời hoặc cổ điển / Temporal or Classical
 - Lô-gic / Logical
 - Ngẫu nhiên / Coincidental

Các mức thống nhất phương thức

| Mức độ | | Mô tả |
|---|-----------------------|--|
| <div>Cao</div>  <div>Thấp</div> | Chức năng | Mỗi phương thức thực hiện 1 công việc (ví dụ, tính GPA hiện tại) |
| | Tuần tự | Phương thức kết hợp 2 chức năng, trong đó đầu ra của chức năng đầu tiên được sử dụng như đầu vào của chức năng tiếp theo (ví dụ, chuẩn hóa và kiểm tra GPA hiện tại) |
| | Giao tiếp | Phương thức kết hợp 2 chức năng sử dụng chung tập thuộc tính (ví dụ, tính điểm GPA hiện tại và điểm GPA tích lũy) |
| | Tiến trình | Phương thức hỗ trợ nhiều chức năng liên kết yếu, ví dụ, tính GPA của sinh viên, in thông tin sinh viên, tính GPA tích lũy. |
| | Tạm thời hoặc cổ điển | Phương thức hỗ trợ nhiều chức năng liên quan theo thời gian (ví dụ, khởi tạo tất cả các thuộc tính) |
| | Lô-gic | Phương thức hỗ trợ nhiều chức năng liên quan, nhưng lựa chọn chức năng cụ thể được thực hiện dựa trên biến điều khiển được cung cấp như tham số. Ví dụ, phương thức có thể mở tài khoản tiết kiệm, tài khoản ghi nợ, hoặc tính lãi dựa theo thông điệp được gửi tới bởi phương thức gọi. |
| | Ngẫu nhiên | Không thể xác định mục đích của phương thức hoặc nó thực hiện nhiều chức năng không liên quan. Ví dụ, phương thức có thể cập nhật hồ sơ khách hàng, tính lãi khoản vay, và phân tích cấu trúc giá của đối thủ. |
| Nguồn: Page-Jones, <i>The Practical Guide to Structured System</i> , và Myers, <i>Composite/Structured Design</i> . | | |

Tổng nhất lớp

Các yếu tố đảm bảo tổng nhất lớp:


- Tất cả các thuộc tính và các phương thức của 1 lớp, cùng hỗ trợ biểu diễn 1 thứ
- Tất cả các thuộc tính và các phương thức trong 1 lớp đều cần thiết
 - Mỗi lớp chỉ nên có các thuộc tính và phương thức cần thiết để biểu diễn các đối tượng của nó.
- Ví dụ,
 - Lớp Student cần có các thuộc tính như: MSSV, Họ, Tên, Địa Chỉ
 - Nhưng không có các thuộc tính như: CPU, RAM, OS

Các mức thống nhất lớp

Các mức thống nhất lớp theo thứ tự giảm dần:

- Lý tưởng / Ideal
- Hỗn hợp vai trò / Mixed-Role
- Hỗn hợp lĩnh vực / Mixed-Domain
- Hỗn hợp đối tượng / Mixed-Instance

Các mức thống nhất lớp₍₂₎

| Mức độ | | Mô tả |
|--|-------------------|---|
| <div>Cao</div>  <div>Thấp</div> | Lý tưởng | Lớp không có yếu tố hỗn hợp |
| | Hỗn hợp vai trò | Lớp có thành phần trực tiếp kết nối đối tượng của lớp với đối tượng thuộc lớp khác trên cùng tầng (ví dụ, tầng lĩnh vực ứng dụng), nhưng các thuộc tính đó không liên quan với ý nghĩa cơ bản của lớp. |
| | Hỗn hợp lĩnh vực | Lớp có thành phần trực tiếp kết nối đối tượng thuộc lớp với đối tượng thuộc lớp khác tầng, nhưng thành phần đó không liên quan đến ý nghĩa cơ bản của lớp. Trong những trường hợp này, các thành phần liên kết thuộc về lớp trên tầng kia. Ví dụ, thuộc tính cổng nằm trong 1 lớp lĩnh vực đúng ra phải nằm trong lớp thuộc tầng kiến trúc hệ thống liên quan với lớp lĩnh vực. |
| | Hỗn hợp đối tượng | Lớp biểu diễn nhiều loại đối tượng. Lớp như vậy cần được phân tách thành các lớp riêng biệt. Thông thường, mỗi đối tượng chỉ sử dụng 1 phần định nghĩa của lớp. |

Nguồn: Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

Lớp tổng quát lý tưởng

- Chứa nhiều phương thức có thể được nhìn thấy từ bên ngoài lớp,
- Mỗi phương thức công khai chỉ thực hiện 1 chức năng,
- Các phương thức chỉ sử dụng các thuộc tính và các phương thức khác được định nghĩa trong cùng lớp hoặc (các) lớp trên của nó,
- Không có ghép nối mức điều khiển giữa các phương thức công khai của nó.

[Glenford J. Myers, Composite/Structured Design, 1998]

Mức thống nhất: Hỗn hợp vai trò

- Tính hỗn hợp vai trò làm giảm khả năng tái sử dụng
- Ví dụ 1: lớp Student và Room cùng thuộc tầng lĩnh vực ứng dụng, nhưng phòng học không phải thuộc tính mô tả sinh viên.
 - Thiết kế lớp Student trực tiếp gắn kết với lớp Room làm phát sinh vấn đề hỗn hợp vai trò.
- Ví dụ 2: Lớp Student và lớp Registration cũng cùng thuộc tầng lĩnh vực ứng dụng, và sinh viên là 1 phần trong đăng ký học tập
 - Thiết kế lớp Registration trực tiếp gắn kết với lớp Student không phát sinh vấn đề hỗn hợp vai trò

Mức thống nhất: Hỗn hợp lĩnh vực

- Phát sinh khi lớp chứa thành phần trực tiếp gắn kết với lớp trên 1 tầng khác / không thuộc tầng của nó.
- Kiểm tra tính năng có thiết yếu đối với lớp không?
 - Ví dụ 1: Lớp Order và lớp Printer thuộc các tầng khác nhau
 - Triển khai phương thức in hóa đơn trong lớp đơn hàng? Có thể triển khai lớp đơn hàng mà không có phương thức in hay không?
 - Ví dụ 2: Lớp OrderDetailView và lớp Order cũng thuộc các tầng khác nhau
 - Có thể triển khai giao diện đơn hàng mà không đọc dữ liệu đơn hàng hay không?

Mức thống nhất: Hỗn hợp đối tượng

- Lớp biểu diễn nhiều hơn 1 nhóm đối tượng / Có thành phần không được sử dụng cho 1 số đối tượng
- Ví dụ: Giả sử chúng ta đang triển khai 1 lớp Person với các phương thức:
 - GetSalary() trả về lương nếu là nhân viên.
 - GetCashBack() trả về số dư tài khoản hoàn tiền nếu là khách hàng.
 - Với mỗi đối tượng Person là của 1 nhân viên hoặc 1 khách hàng chúng ta đều có tính năng không sử dụng đến
 - Cho thấy lớp Person đang quá rộng.
 - Giải pháp:
 - Tách các tính năng gắn với nhân viên và đưa vào 1 lớp Employee
 - Tách các tính năng gắn với khách hàng và đưa vào 1 lớp Customer
 - Các lớp Employee và Customer kế thừa lớp Person

Đồng sinh

- Các thành phần phụ thuộc lẫn nhau khiến thay đổi trong 1 thành phần kéo theo thay đổi trong các thành phần khác.
- Kết hợp chỉ số ghép nối và chỉ số thống nhất trong các giới hạn đóng gói
- Có 3 mức đóng gói được sử dụng:
 - Đóng gói mức 0: Đóng gói trong phạm vi 1 câu lệnh
 - Đóng gói mức 1: Phương thức - tổng hợp nhiều câu lệnh
 - Kết hợp thống nhất phương thức và ghép nối tương tác.
 - Đóng gói mức 2: Lớp - Chứa cả thuộc tính và phương thức
 - Kết hợp thống nhất lớp, thống nhất cây kế thừa, và ghép nối kế thừa

Đánh giá mức độ đồng sinh

- Mức độ đồng sinh thường được đánh giá trong phạm vi đóng gói mức 1 (mức phương thức), và đóng gói mức 2 (mức lớp).
- Mã nguồn cần hướng tới:
 - Cực tiểu hóa đồng sinh giữa các giới hạn đóng gói (ít liên kết phụ thuộc giữa các thành phần).
 - Cực đại hóa đồng sinh trong phạm vi đóng gói (nhiều quan hệ phụ thuộc trong 1 thành phần).
 - => Lớp con không nên trực tiếp truy cập thuộc tính hoặc phương thức của lớp cha

Các trường hợp đồng sinh

- Tên / Name
- Kiểu hoặc Lớp / Type or Class
- Quy ước / Convention
- Giải thuật / Algorithm
- Vị trí / Position

Các trường hợp đồng sinh₍₂₎

| Phân loại | Mô tả |
|---|--|
| Tên | Nếu 1 phương thức sử dụng 1 thuộc tính, thì nó được gắn với tên của thuộc tính. Nếu tên thuộc tính thay đổi, nội dung của phương thức cũng sẽ phải thay đổi. |
| Kiểu hoặc Lớp | Nếu 1 lớp có 1 thuộc tính thuộc kiểu A, nó được gắn với kiểu của thuộc tính. Nếu kiểu của thuộc tính thay đổi, khai báo thuộc tính cũng sẽ phải thay đổi. |
| Quy ước | Một lớp có 1 thuộc tính mà miền giá trị có ý nghĩa (ví dụ, số tài khoản với các giá trị trong phạm vi từ 1000 tới 1999 là các tài sản). Nếu khoảng thay đổi, thì tất cả phương thức sử dụng thuộc tính cũng cần phải thay đổi. |
| Giải thuật | Hai phương thức khác nhau của 1 lớp cùng phụ thuộc vào 1 giải thuật để chạy đúng (ví dụ, kiểm tra tính hợp lệ của địa chỉ email khi tạo tài khoản và khi gửi yêu cầu khôi phục mật khẩu). Nếu giải thuật cơ sở cần thay đổi, thì phương thức thêm và tìm kiếm cũng sẽ phải thay đổi. |
| Vị trí | Thứ tự mã nguồn trong 1 phương thức hoặc thứ tự các tham số trong 1 phương thức là đặc biệt quan trọng để phương thức chạy đúng. Nếu bất kỳ thứ tự nào sai, thì phương thức có thể hoạt động sai. |
| Nguồn: Meilir Page-Jones, <i>Comparing Techniques by Means of Encapsulation and Connascence</i> and Meilir Page-Jones, <i>Fundamentals of Object-Oriented Design in UML</i> . | |

Ví dụ 5.3. Đồng sinh tên

```
struct student {  
    long id;  
    char fullname[256];  
};
```

Mã nguồn sử dụng cấu trúc student phụ thuộc vào các định danh của cấu trúc student.

Ví dụ 5.4. Đồng sinh kiểu

```
char *s = NULL;
```

```
s = "Hi!"; // OK
```

```
s = 100; // NOK
```

Mã nguồn sử dụng biến s phụ thuộc vào kiểu của biến s.

Ví dụ 5.5. Đồng sinh ý nghĩa

```
struct bst_node *ret = bst_put(bst, key, value);  
  
if (ret == NULL) {  
    printf("Khóa mới.");  
} else {  
    printf("Khóa đã tồn tại.");  
}
```

Mã nguồn sử dụng hàm bst_put phụ thuộc vào quy ước đối với giá trị trả về.

Ví dụ 5.6. Đồng sinh vị trí

```
int less(int a, int b) {  
    return a < b;  
}
```

```
if (less(3, 5)) {  
    printf("OK!");  
}
```

Hàm less kiểm tra mệnh đề $a < b$, mã nguồn sử dụng hàm less cần truyền tham số theo đúng thứ tự.

Ví dụ 5.7. Đồng sinh thuật toán

```
char *vb_encode(int *inp);
```

```
int *vb_decode(char *inp);
```

```
int arr[100]; // Sử dụng lính canh để đánh dấu điểm cuối
```

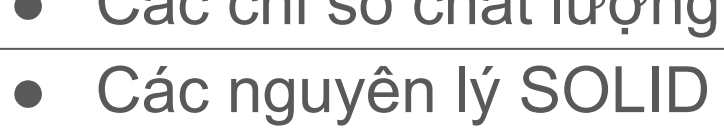
```
...
```

```
char *data = vb_encode(arr);
```

```
int *arr = vb_decode(data);
```

Thuật toán giải nén phụ thuộc vào thuật toán nén.

Nội dung

- Vấn đề thiết kế lớp
 - Các chỉ số chất lượng
 - Các nguyên lý SOLID
- 

Nguyên lý trách nhiệm duy nhất

- Nguyên lý trách nhiệm duy nhất / Single Responsibility Principle (SRP)

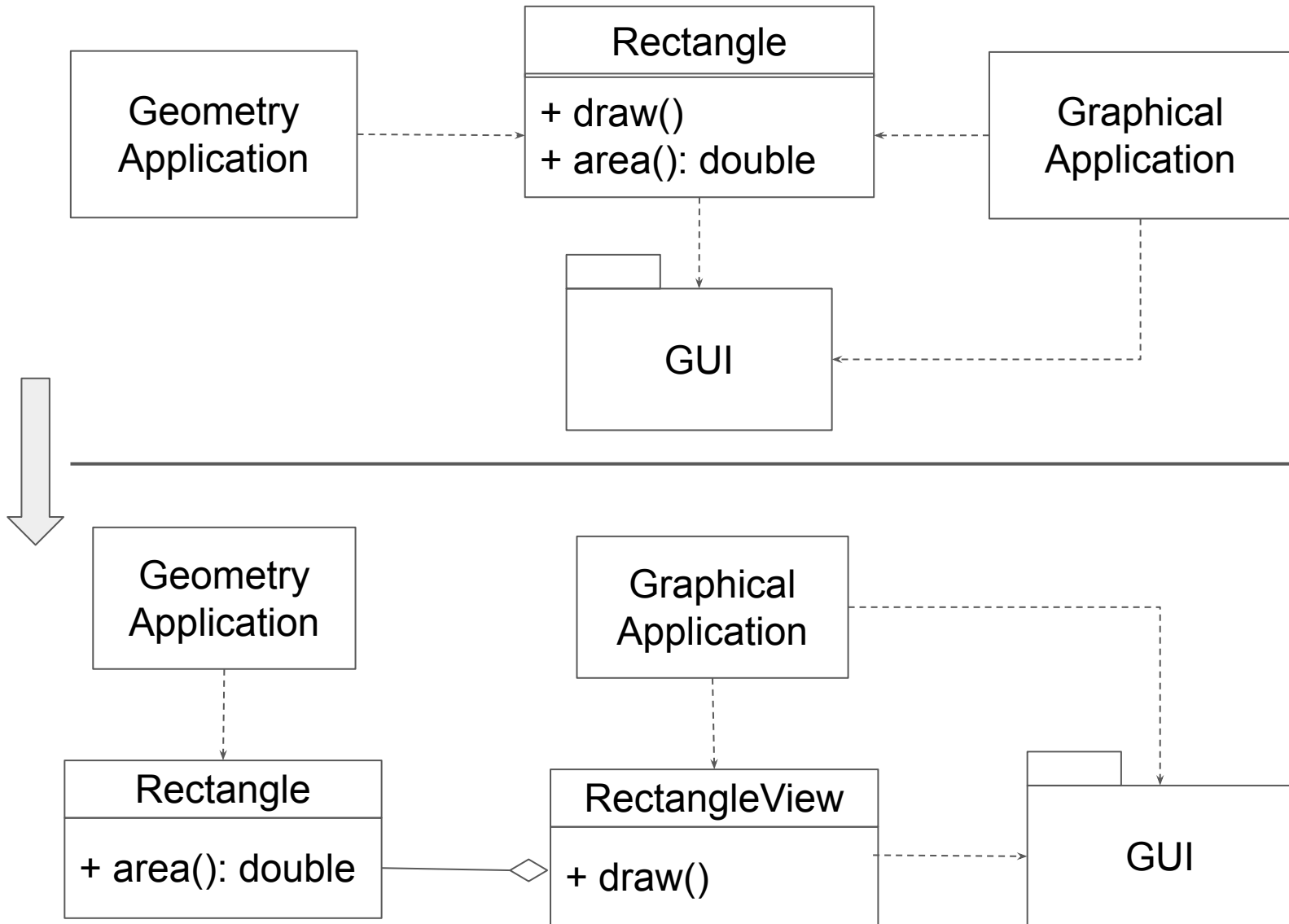
*Mỗi lớp chỉ nên có 1 lý do duy nhất để thay đổi /
A class should have only one reason to change*
[Robert .C Martin]

- Trách nhiệm được gán cho lớp
- Nếu phát sinh nhu cầu triển khai chức năng mới, hoặc chức năng đang có thay đổi, thì các trách nhiệm của các lớp phải thay đổi.
- Lớp có 1 trách nhiệm sẽ chỉ có 1 lý do để thay đổi.
Mỗi thành phần chỉ nên có đảm nhận 1 trách nhiệm

Trách nhiệm và tính thống nhất

- Lớp có tính thống nhất ở mức cao chỉ triển khai 1 trách nhiệm (hoặc ít trách nhiệm) / Lớp chỉ triển khai 1 trách nhiệm có tính thống nhất ở mức cao.
- Lớp có tính thống nhất ở mức thấp thường vi phạm nguyên lý SRP / Lớp vi phạm SRP thường có tính thống nhất ở mức thấp.
- Ưu điểm: Các lớp 1 trách nhiệm thường nhỏ, dễ tái sử dụng, và dễ hiểu hơn, và ít thay đổi.
- Lưu ý: Phân tách trách nhiệm có thể phức tạp và có hiệu ứng phụ, áp dụng SRP khi thực sự cần thiết, thực sự có thay đổi thường xuyên.

Ví dụ 5.8. SRP



Nguyên lý Mở-Đóng

- Nguyên lý Mở-Đóng / Open-Close Principle (OCP)

Các thực thể phần mềm cần có khả năng mở rộng mà không thay đổi /

Software entities should be open for extension, but closed for modifications.

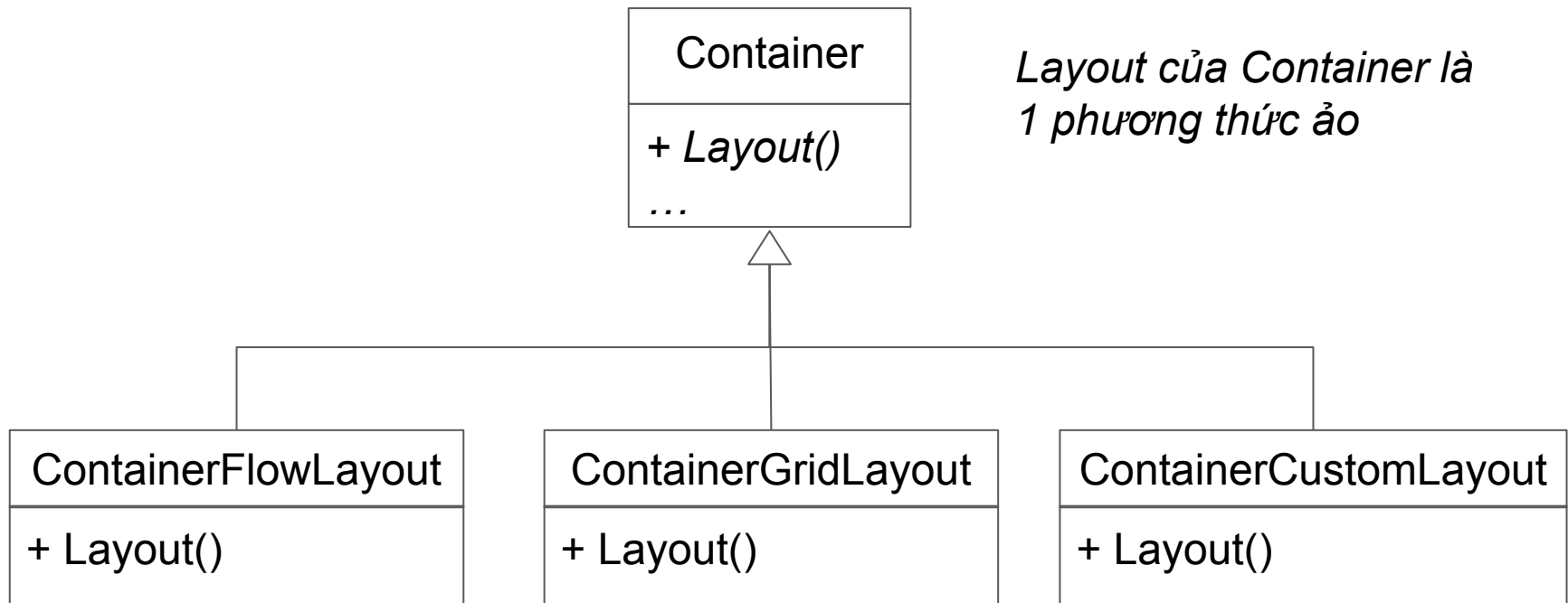
[Bertrand Meyer]

- Mở rộng hành vi của 1 thành phần:
 - Khi yêu cầu ứng dụng thay đổi, chúng ta có thể mở rộng thành phần đã có, thêm vào các hành vi mới.
- Không thay đổi mã nguồn đã có:
 - Mã nguồn đã có được giữ nguyên trong quá trình bổ xung hành vi mới.
 - Đem lại nhiều lợi ích trong quá trình phát triển.

Trừu tượng hóa là chìa khóa

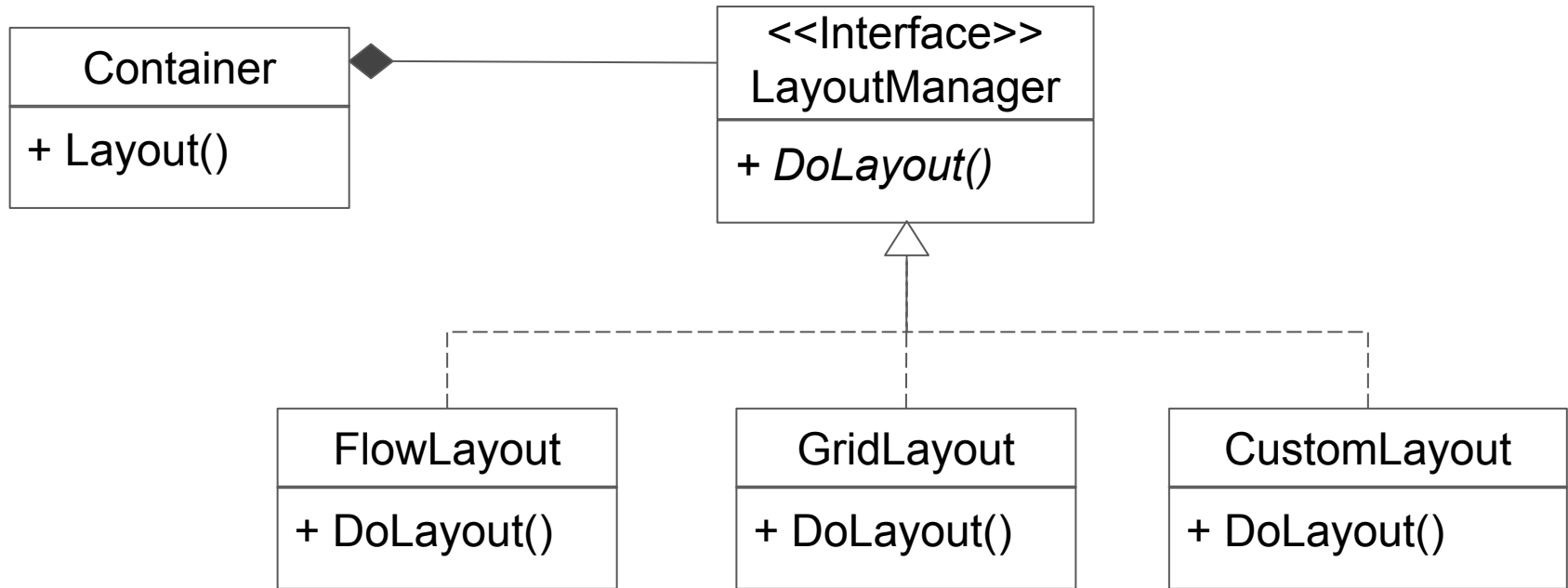
- Để mở rộng một thành phần mà không thay đổi nó, thì phần thay đổi trong triển khai của thành phần đó phải được trừu tượng hóa.
- Lập trình hướng đối tượng: Lớp ảo, giao diện, phương thức ảo, kế thừa, v.v..
- Lập trình hàm: Kiểu khái quát, hàm khái quát, v.v..
- *(Khả năng tránh thay đổi có tính tương đối: Rất khó tuyệt đối tránh thay đổi mã nguồn / Thường vẫn có những yêu cầu thay đổi dẫn đến thay đổi mã nguồn).*

Khái quát hóa dựa trên kế thừa



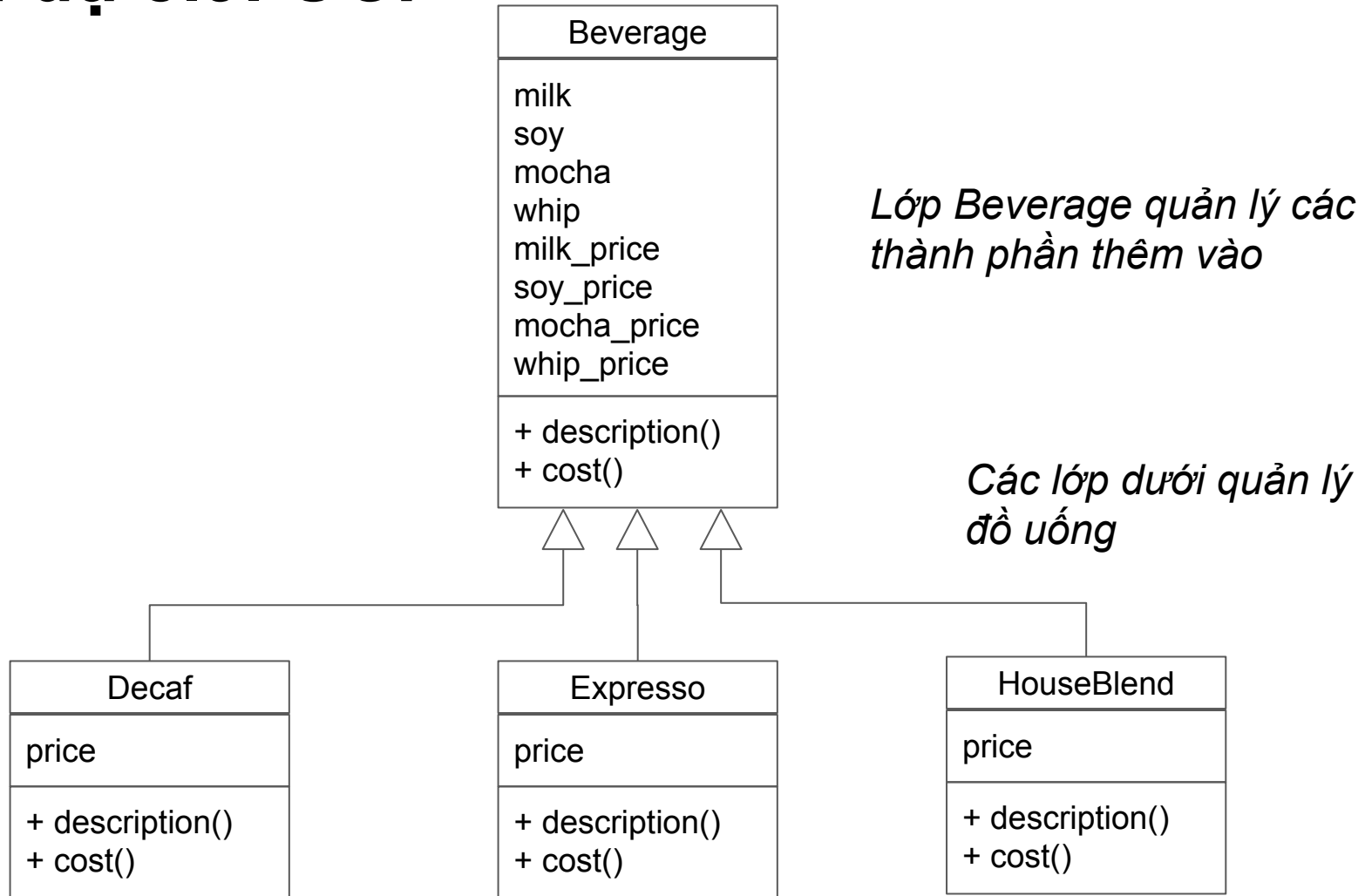
Lớp con sẽ cung cấp triển khai cụ thể cho Layout

Khái quát hóa dựa trên tổng hợp



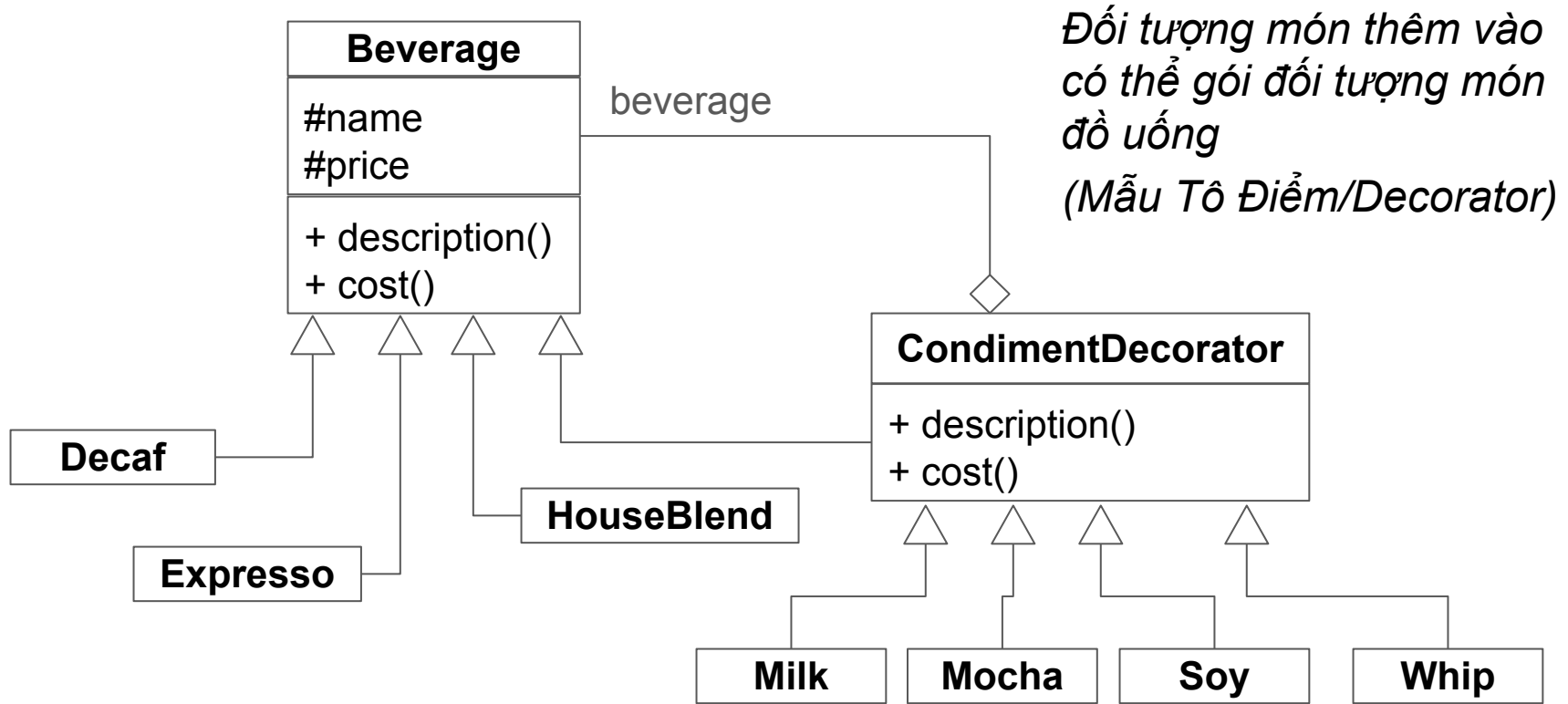
Chúng ta có thể thay đổi hành vi của Container bằng cách thiết lập 1 đối tượng quản lý bố cục phù hợp (kế thừa LayoutManager)

Ví dụ 5.9. OCP



Thiết kế này không tương thích OCP, để thay đổi giá của thành phần thêm vào hoặc bổ sung món mới chúng ta phải sửa lớp Beverage.

Ví dụ 5.9. OCP₍₂₎



- Gói đối tượng Beverage trong đối tượng CondimentDecorator.
- name và price được tính theo cấu trúc đóng gói.

Thiết kế này đáp ứng được OCP.

Nguyên lý khả thay Liskov

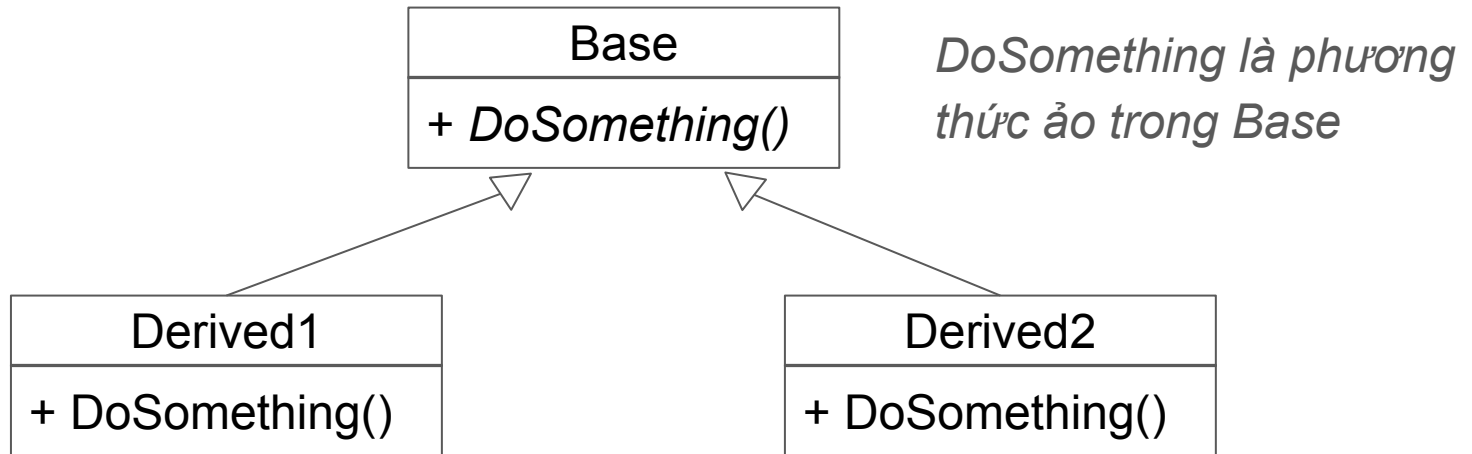
- Nguyên lý khả thay Liskov / Liskov Substitution Principle (LSP)

Các lớp con phải có khả năng thay thế các lớp cơ sở về mặt hành vi/

Subtypes must be behaviorally substitutable for their base types. [Barbara Liskov, 1988]

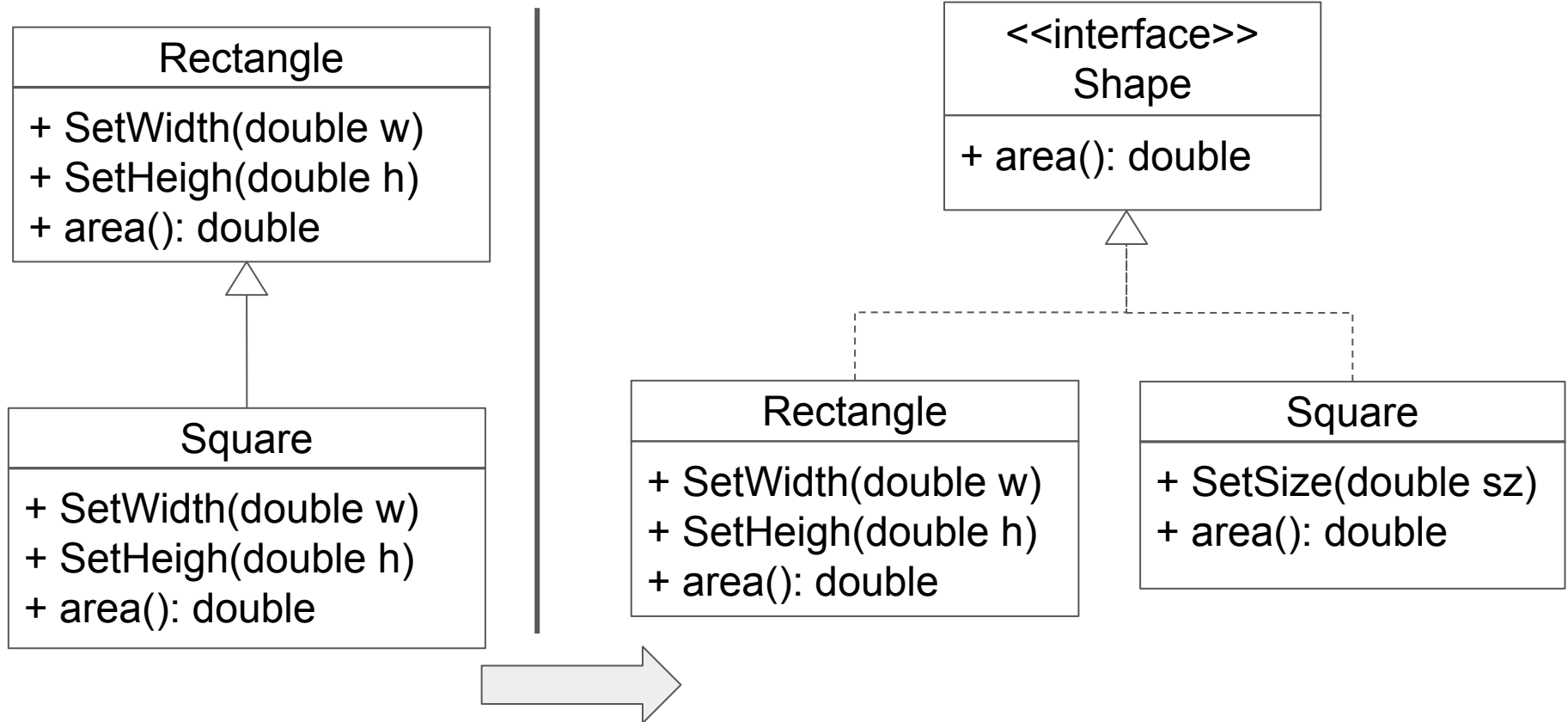
- Như dấu hiệu chất lượng thiết kế kế thừa.
- Hạn chế các vấn đề có thể phát sinh khi lập trình với các giao diện khái quát.

Khả thay hành vi



- Ngoài định nghĩa lại phương thức `DoSomething()` trong **Derived1** và **Derived2**, LSP còn ràng buộc về hành vi đối với các đối tượng thuộc các lớp **Derived1** và **Derived2**:
 - Có thể diễn đạt một cách gần đúng là bảo toàn ý nghĩa của lớp cơ sở.

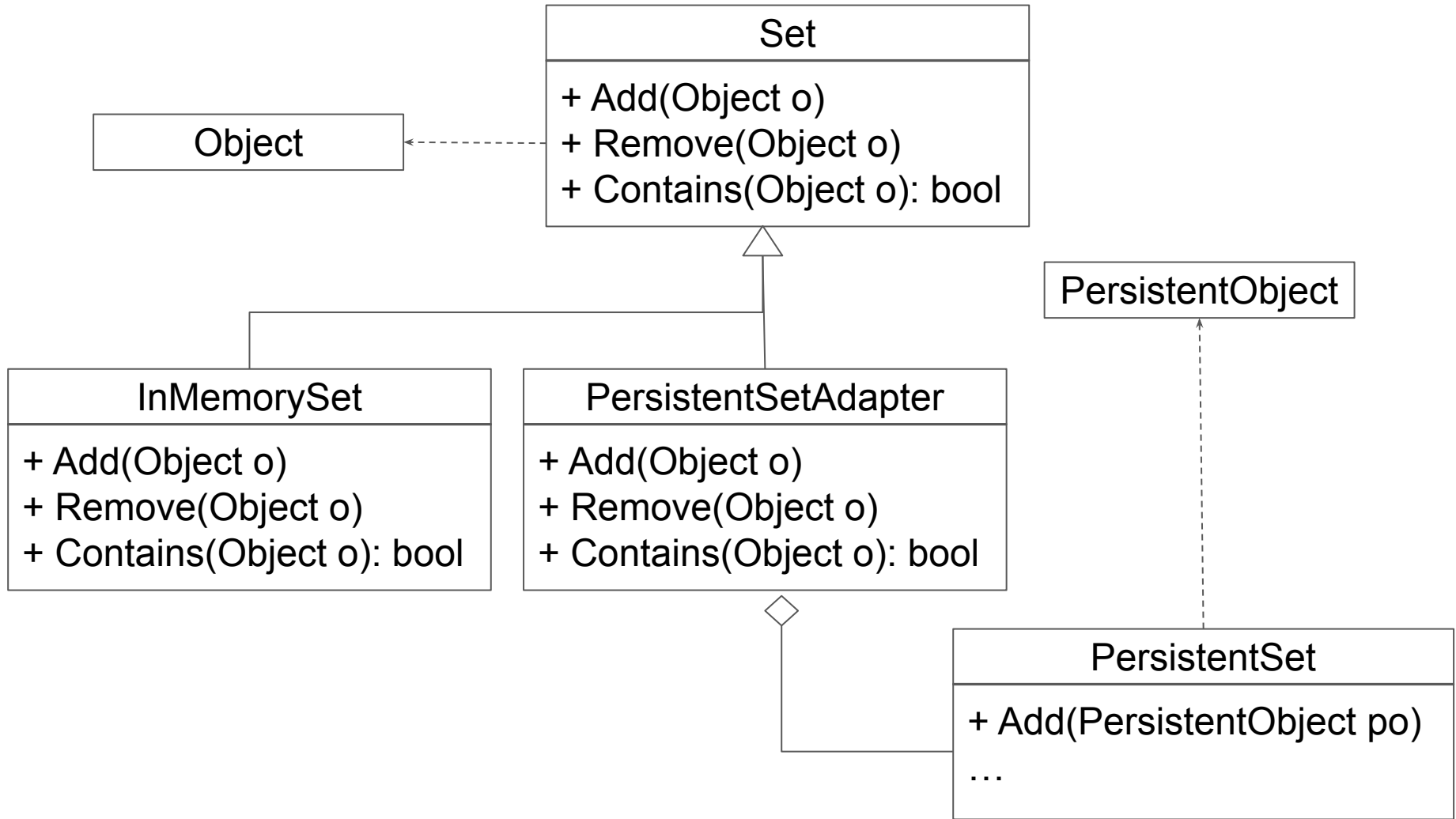
Ví dụ 5.10. LSP



Không tương thích LSP, SetHeigh và SetWidth có ý nghĩa khác nhau trong Rectangle và Square.

Đáp ứng được LSP

Ví dụ 5.10. LSP₍₂₎



Đánh giá thiết kế và đề xuất phương án nếu cần?

Nguyên lý tách bạch giao diện

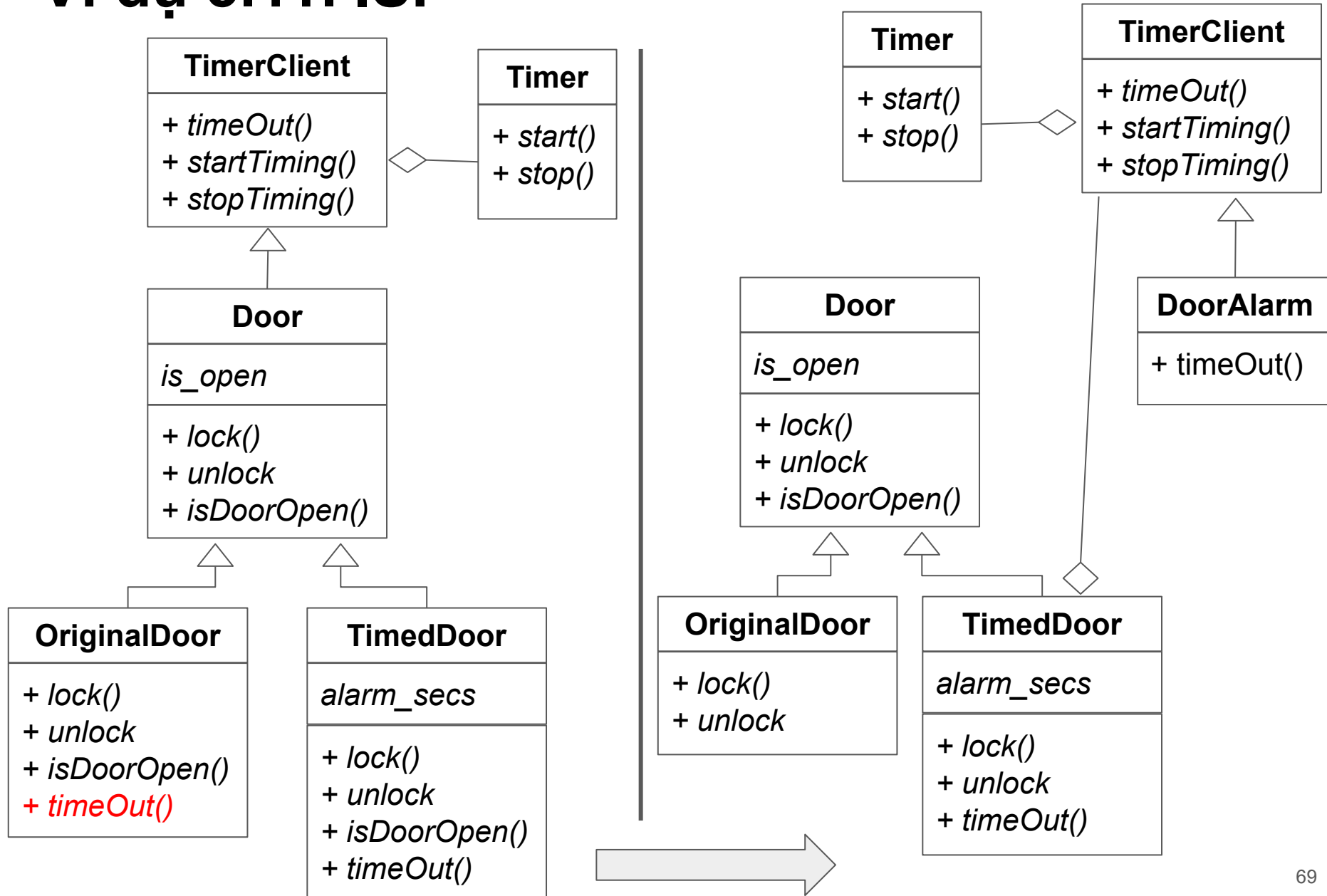
- Nguyên lý tách bạch giao diện / Interface Segregation Principle (ISP)

*Không ép phía khách phụ thuộc vào các phương thức không được sử dụng trong 1 giao diện /
Clients should not be forced to depend on methods that they do not use*

[Robert C. Martin]

- Các giao diện chứa phương thức không liên quan ép các thành phần sử dụng giao diện đó chịu các thay đổi đúng ra không ảnh hưởng tới chúng.
- Các giao diện pha trộn cần được phân tách
- Nhưng cũng cần tránh làm nát vụn các giao diện

Ví dụ 5.11. ISP



Nguyên lý đảo ngược phụ thuộc

- Nguyên lý đảo ngược phụ thuộc / Dependency Inversion Principle (DIP)

Các mô-đun bậc cao không nên phụ thuộc vào các mô-đun bậc thấp. Cả 2 nên phụ thuộc vào các thành phần trừu tượng.

Các thành phần trừu tượng không nên phụ thuộc vào các thành phần cụ thể. Các thành phần cụ thể nên phụ thuộc vào các thành phần trừu tượng. /

High-level modules should not depend on low-level modules. Both should depend on abstractions.

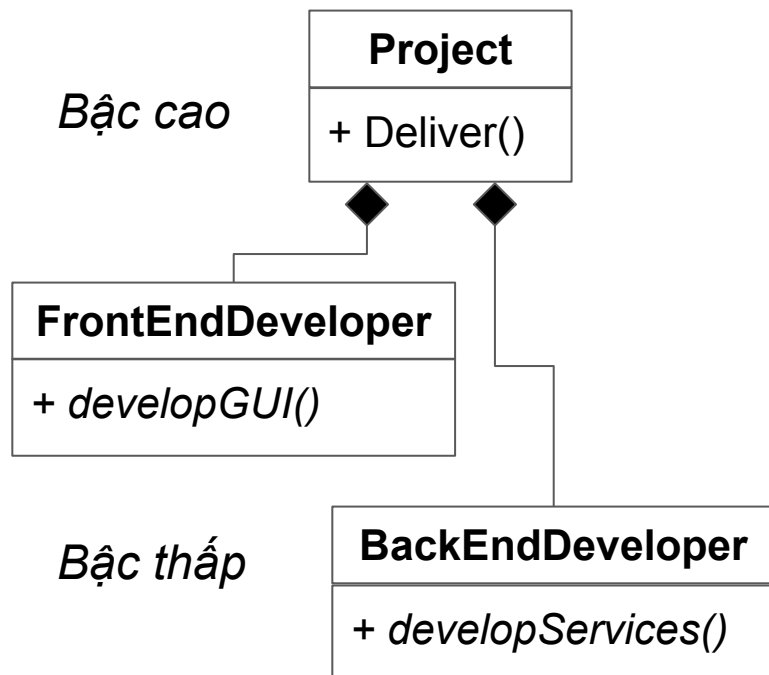
Abstractions should not depend upon details. Details should depend upon abstractions.

[Robert C. Martin]

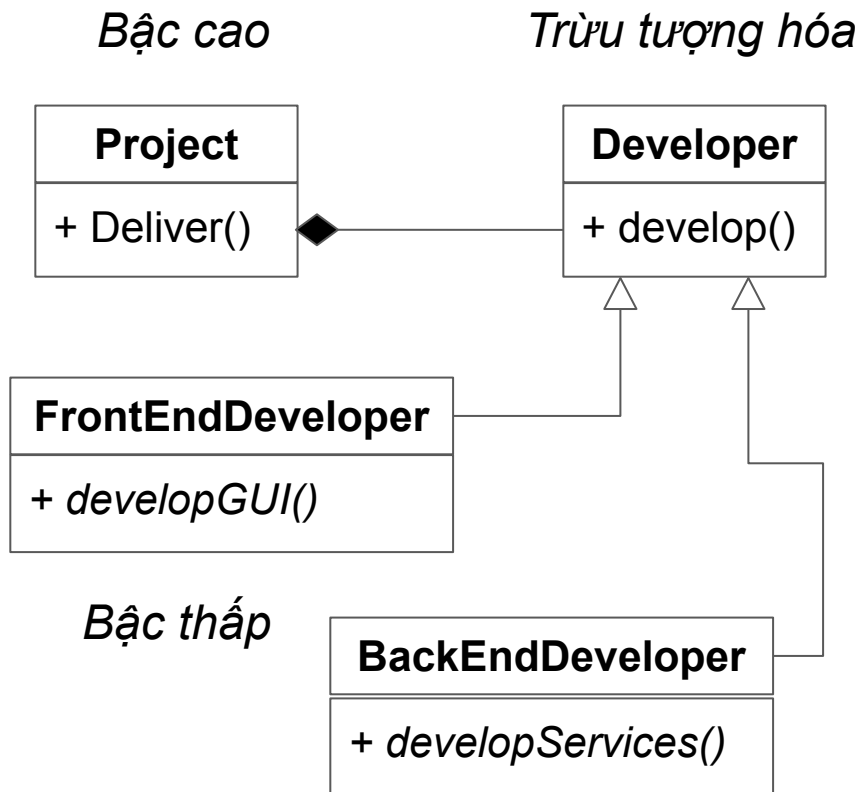
Nguyên lý đảo ngược phụ thuộc₍₂₎

- Phần mềm với thiết kế tốt thường được chia nhỏ thành các mô-đun
 - Mô-đun bậc thấp cung cấp các dịch vụ để triển khai mô-đun bậc cao.
- Phụ thuộc vào các cơ chế trừu tượng giúp nâng cao khả năng tái sử dụng các thành phần.
- Áp dụng DIP giúp nới lỏng các phụ thuộc.

Ví dụ 5.12. DIP



Thiết kế này không tương thích với DIP do Project phụ thuộc trực tiếp vào FrontEndDeveloper và BackEndDeveloper.



Triển khai cụ thể phụ thuộc vào cơ chế trừu tượng.

Đáp ứng được DIP, chiều phụ thuộc vào mô-đun bậc thấp đã được đảo ngược: Các mô-đun bậc thấp phụ thuộc vào giao diện khái quát được dùng trong mô-đun bậc cao.

Chèn phụ thuộc

- Chèn phụ thuộc / Dependency Injection (DI)
- Phương pháp nghịch đảo phụ thuộc cụ thể, có thể được sử dụng để nghịch đảo phụ thuộc.
- Đối tượng thích hợp được tạo và đưa vào từ bên ngoài.
- Đối tượng phụ thuộc có thể được đưa vào thông qua:
 - Tham số cho hàm tạo (được sử dụng cho những phụ thuộc bắt buộc)
 - Như trong mẫu thiết kế Tô điểm / Decorator
 - Tham số cho phương thức thiết lập phụ thuộc (được sử dụng cho các phụ thuộc không bắt buộc)
 - Như trong mẫu thiết kế Trạng thái / State

