TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
Viện Toán Ứng dụng và Tin học

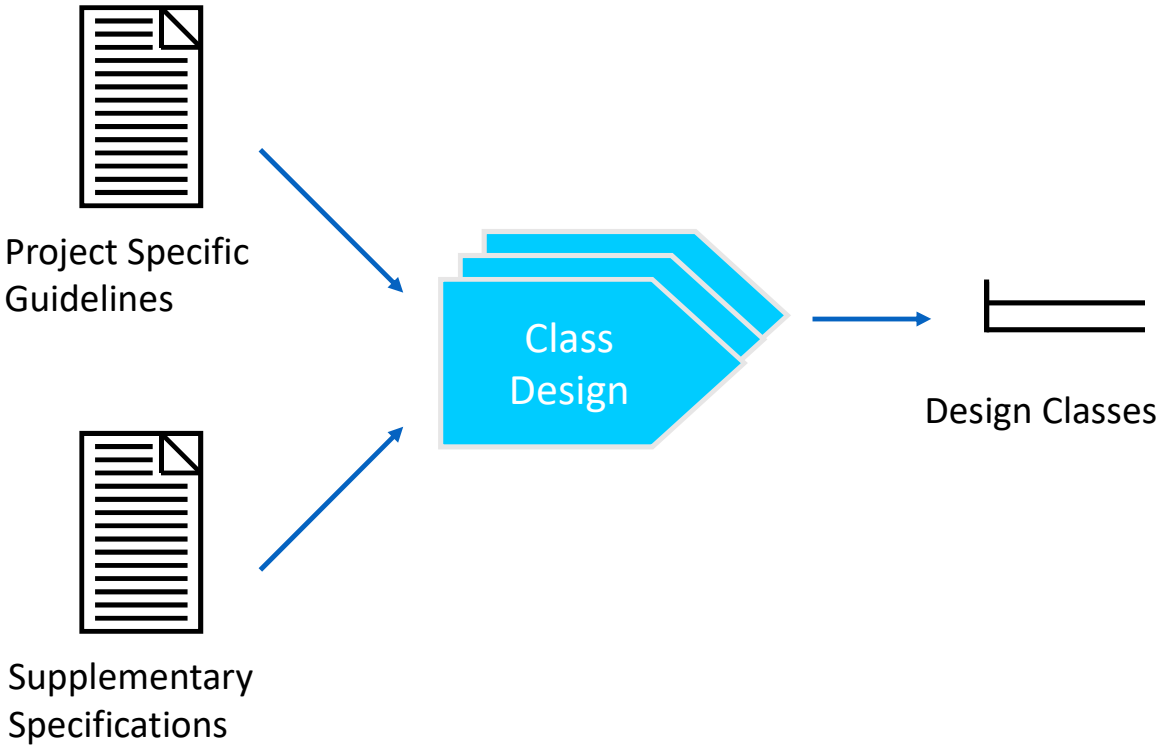
Phân tích thiết kế hệ thống

Thiết kế lớp

Nội dung

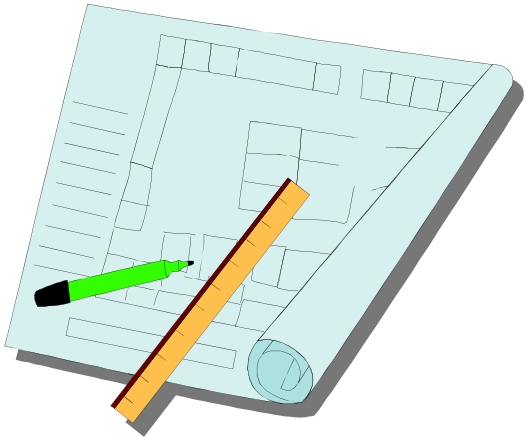
- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết xung đột usecase
- Đáp ứng các đòi hỏi phi chức năng

Thiết kế lớp


Project Specific
Guidelines
Supplementary
Specifications
Class
Design
Design Classes

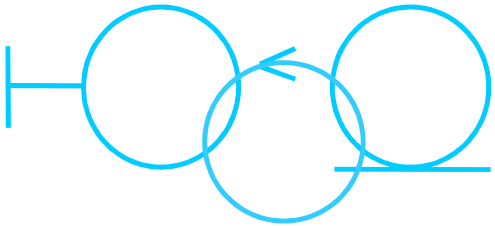
Các bước thiết kế lớp

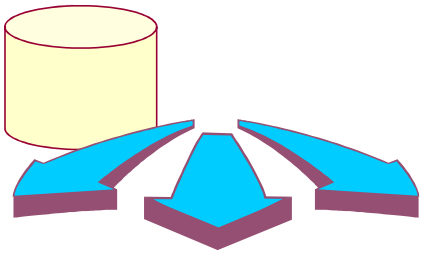
- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các đòi hỏi phi chức năng



Cân nhắc lớp thiết kế

- Các dạng lớp
  - Lớp biên
  - Lớp thực thể
  - Lớp điều khiển
- Các mẫu thiết kế
- Các cơ chế kiến trúc
  - Cơ chế lưu trữ
  - Cơ chế phân tán
  - etc.





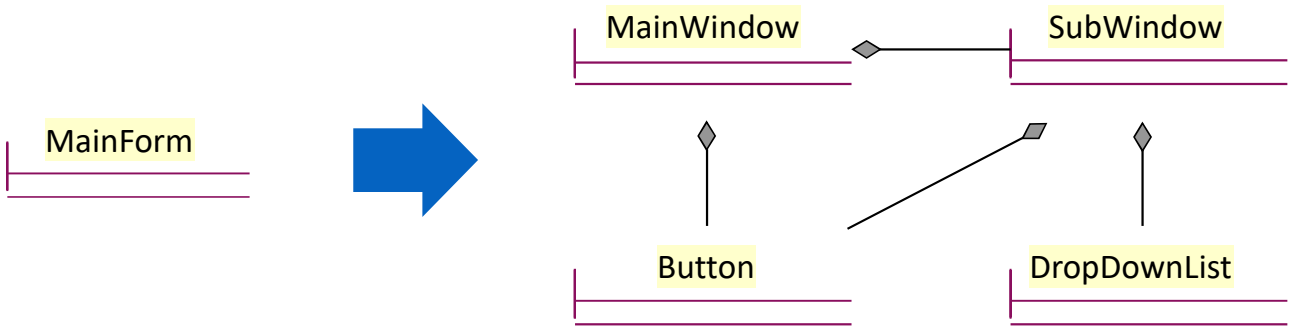
Bao nhiêu lớp thì đủ?

- Nhiều, các lớp sẽ đơn giản và
  - Đóng gói ít nghiệp vụ hệ thống
  - Khả năng sử dụng lại cao
  - Dễ thể hiện
- Ít, các lớp sẽ phức tạp và
  - Đóng gói nhiều nghiệp vụ hệ thống
  - Ít khả năng sử dụng lại
  - Khó thể hiện

Một lớp nên có một mục đích đơn, rõ ràng. Lớp nên làm 1 việc và làm tốt việc đó.

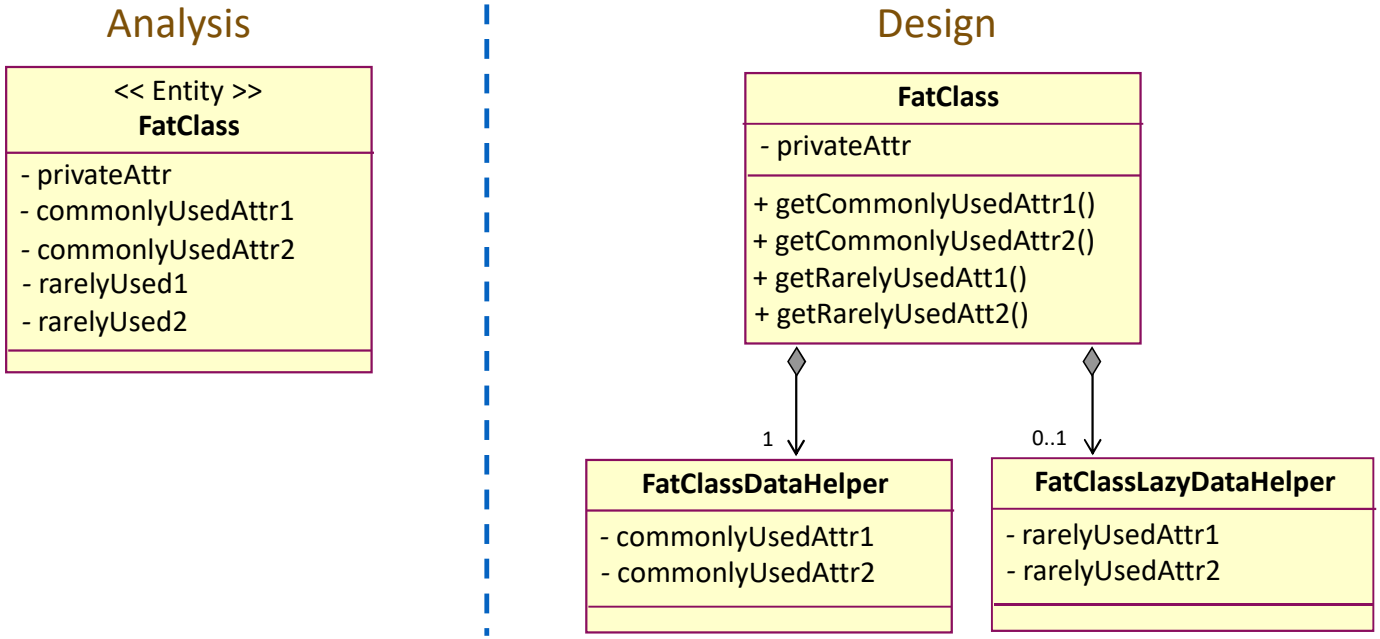
Các chiến lược thiết kế lớp biên

- Các lớp biên giao diện người dùng (UI)
  - Công cụ thiết kế giao diện nào sẽ được dùng?
  - Bao nhiêu giao diện sẽ được tạo bởi công cụ?
- Các lớp biên giao diện hệ thống ngoài
  - Thường được mô hình như 1 hệ thống con



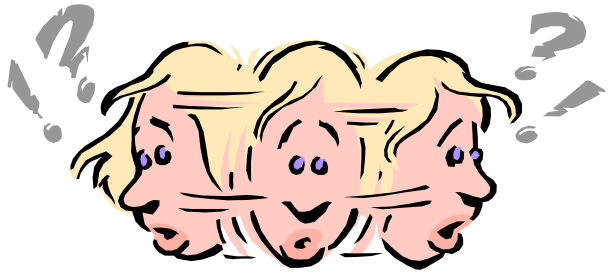
Các chiến lược thiết kế lớp thực thể

- Các đối tượng thực thể thường là đối tượng bị động và được lưu trữ
- Các yêu cầu về hiệu năng có thể đòi hỏi phân rã



Các chiến lược thiết kế lớp điều khiển

- Điều gì xảy ra với các lớp điều khiển?
  - Chúng có thực sự cần thiết?
  - Có nên chia nhỏ chúng ra không?
- Quyết định dựa vào đâu?
  - Độ phức tạp
  - Khả năng thay đổi
  - Sự phân tán và hiệu năng
  - Quản lý giao dịch

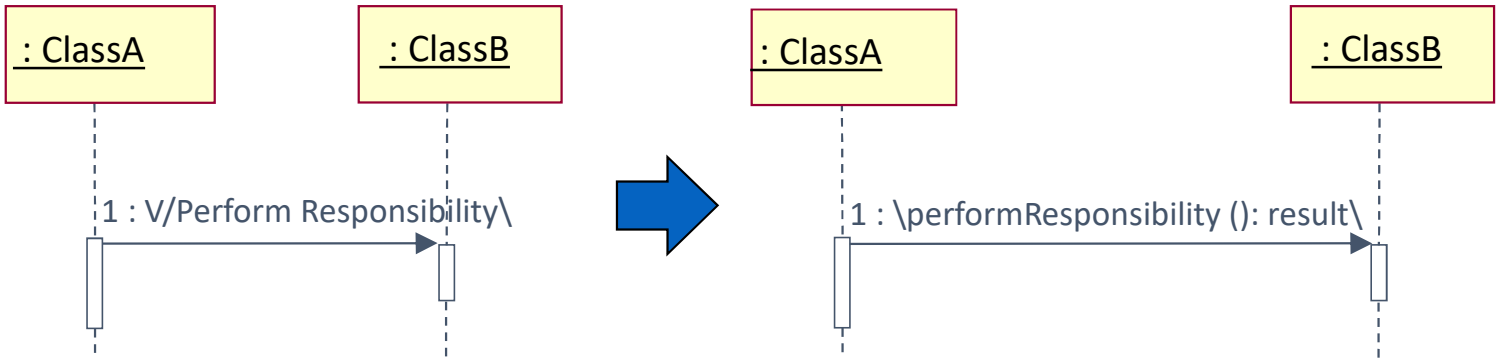


Các bước thiết kế lớp

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

Các hoạt động: Tìm ra chúng từ đâu?

- Các thông báo được trình bày trong biểu đồ tương tác



- Chức năng phụ thuộc thể hiện khác
  - Các chức năng quản lý
  - Yêu cầu copy lớp
  - Yêu cầu kiểm định chất lượng

Tên và mô tả các hoạt động

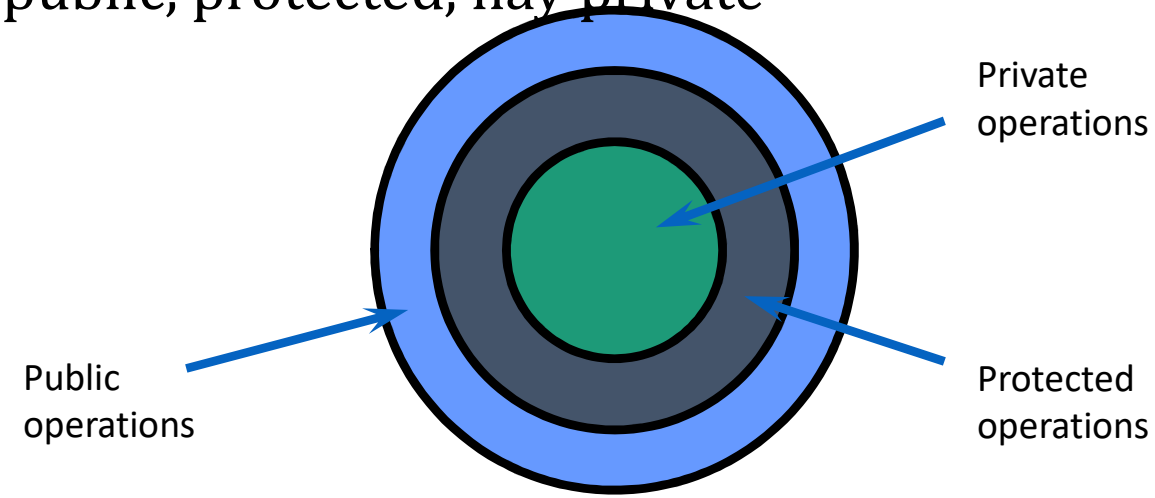
- Đặt tên hoạt động phù hợp
  - Báo hiệu đầu ra
  - Nhìn từ góc độ người dùng
  - Nhất quán giữa các lớp
- Xác định cấu trúc hoạt động
  - operationName([direction]parameter : class,..) : returnType
    - Direction là **in**, **out** hoặc **inout** với mặt định là **in**
- Cung cấp mô tả ngắn, ý nghĩa của tất cả các tham số

Hướng dẫn: thiết kế hoạt động

- Cân nhắc các tham số:
  - Tham trị hay tham biến
  - Có bị biến đổi bởi hoạt động
  - Là lựa chọn hay bắt buộc
  - Các giá trị mặc định
  - Miền giá trị
- Càng ít tham số càng tốt
- Truyền đối tượng thay vì cấu trúc “data bits”

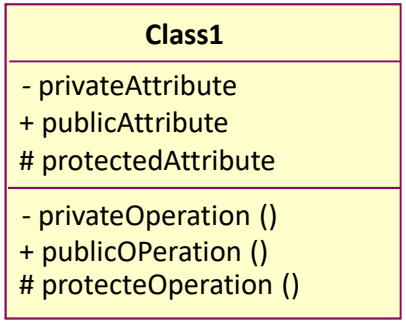
Tính nhìn thấy của hoạt động

- Tính nhìn thấy tạo nên tính đóng gói
- Có thể là public, protected, hay private



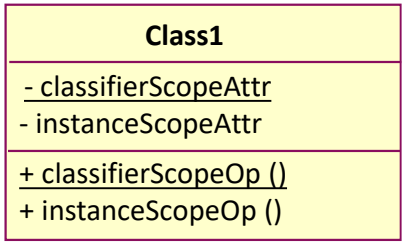
Mô tả tính nhìn thấy?

- Sử dụng các ký hiệu sau:
  - \+ Public access
  - \# Protected access
  - \- Private access

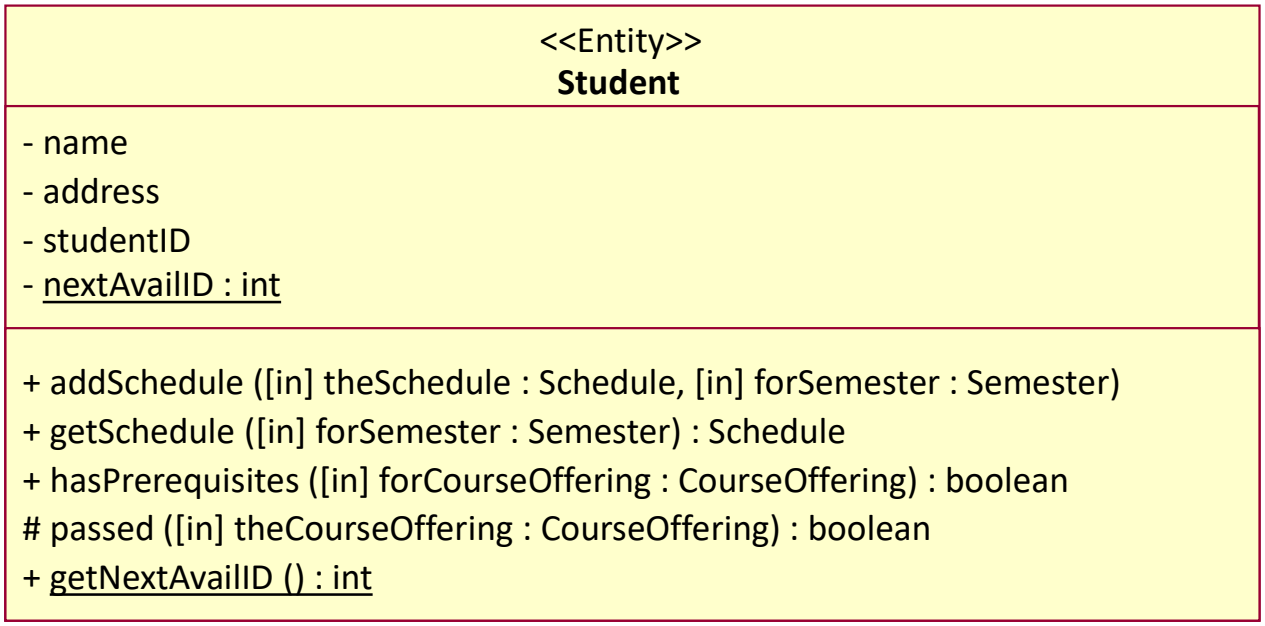


Phạm vi

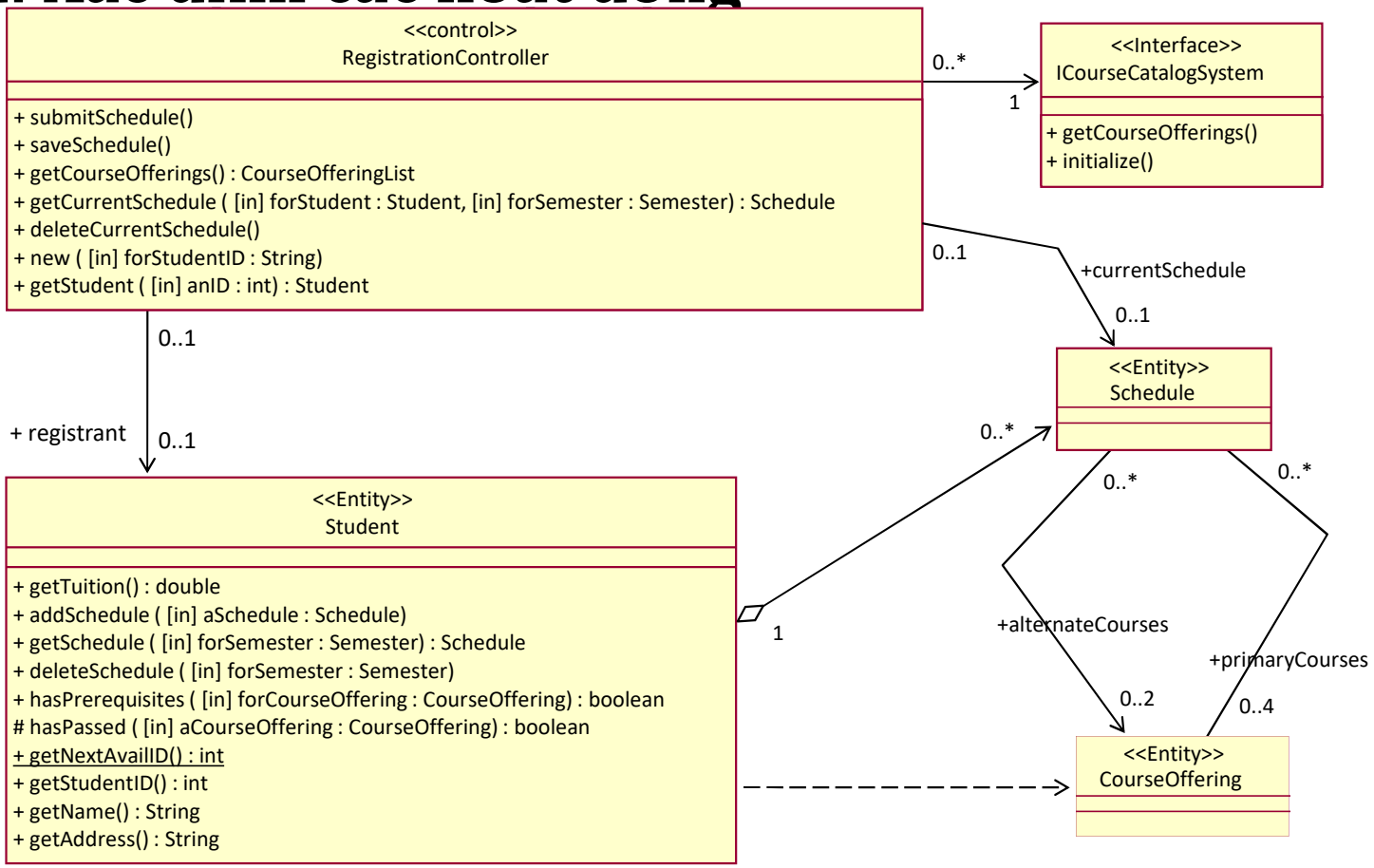
- Xác định số thể hiện của thuộc tính/hoạt động
  - Thể hiện (Instance): một thể hiện cho mỗi thể hiện lớp
  - Lớp (Classifier): một thể hiện cho tất cả các thể hiện của lớp
- Phạm vi lớp được chú giải bằng cách gạch chân.



Thí dụ: Phạm vi



Thí dụ: Xác định các hoạt động



Các bước thiết kế lớp

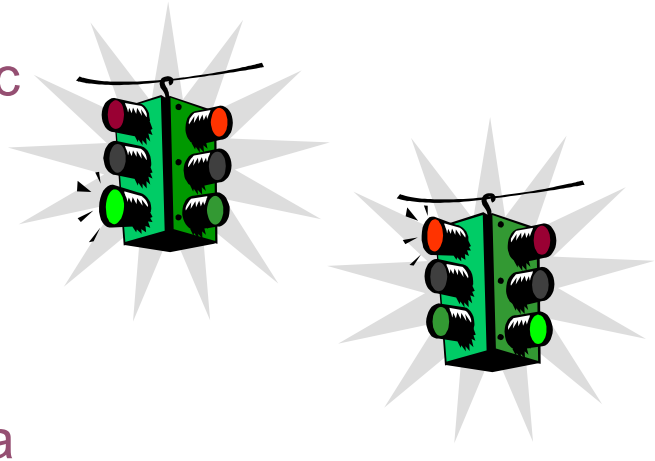
- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

Các định các phương thức

- Phương thức là gì?
  - Mô tả thể hiện của hoạt động
- Mục đích
  - Xác định các khía cạnh thể hiện đặc biệt của hoạt động
- Các nội dung cần cân nhắc:
  - Giải thuật đặc biệt
  - Sử dụng các đối tượng, hoạt động khác
  - Cách các thuộc tính và tham số được thể hiện và sử dụng
  - Cách các quan hệ được thể hiện và sử dụng

Các bước thiết kế lớp

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

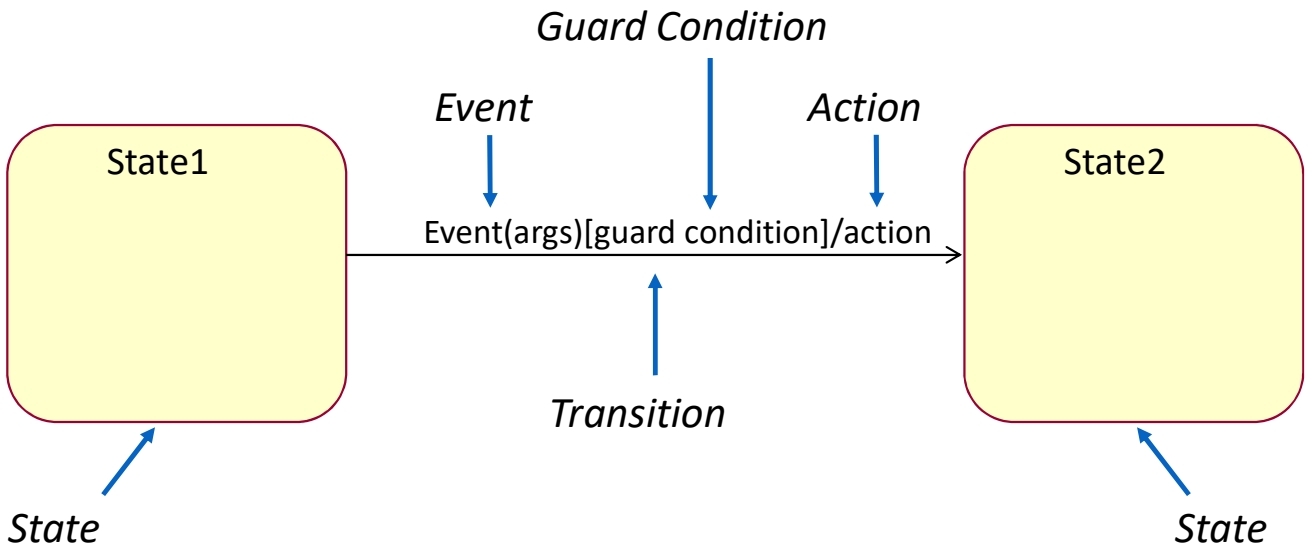


Xác định các trạng thái

- Mục đích
  - Thiết kế cách trạng thái của đối tượng ảnh hưởng hành vi của nó
  - Phát triển biểu đồ trạng thái để mô hình hành vi này
- Các nội dung cần cân nhắc :
  - Các đối tượng nào có trạng thái có nghĩa?
  - Xác định các trạng thái có thể của đối tượng thế nào?
  - Biểu đồ trạng thái trong tổng thể mô hình thế nào?

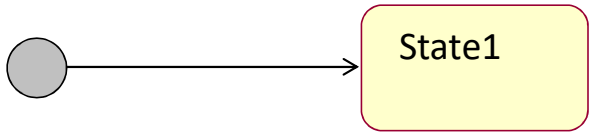
Biểu đồ chuyển đổi trạng thái?

- Đồ thị có hướng của các trạng thái (nodes) được kết nối bởi các đường chuyển đổi (directed arcs)
- Mô tả lịch sử của đối tượng phản ứng

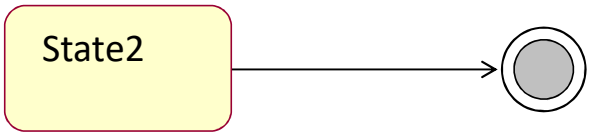


Các trạng thái đặc biệt

- Trạng thái khởi đầu
  - Trạng thái khi đối tượng được tạo ra
  - Bắt buộc
  - Chỉ có 1 trạng thái khởi đầu



- Trạng thái kết thúc
  - Chỉ báo điểm kết thúc của đối tượng
  - Là lựa chọn
  - Có thể có nhiều trạng thái kết thúc



Nhận diện và xác định các trạng thái

- Các thuộc tính động và có ý nghĩa

The maximum number of students per course offering is 10

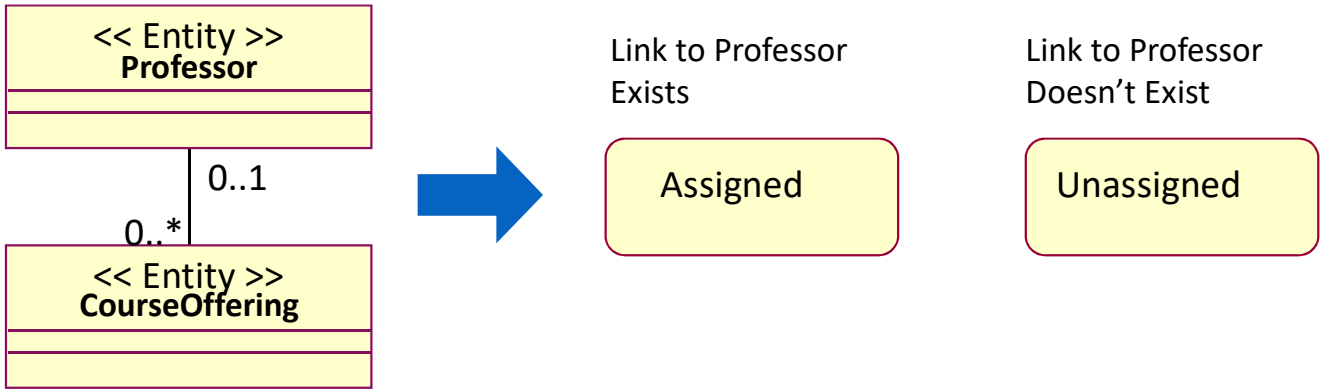
numStudents < 10

Open

numStudents > = 10

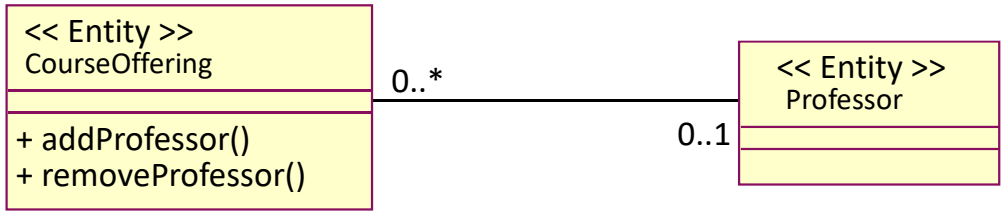
Closed

- Sự tồn tại hay không tồn tại của 1 liên kết nào đó



Nhận diện các sự kiện

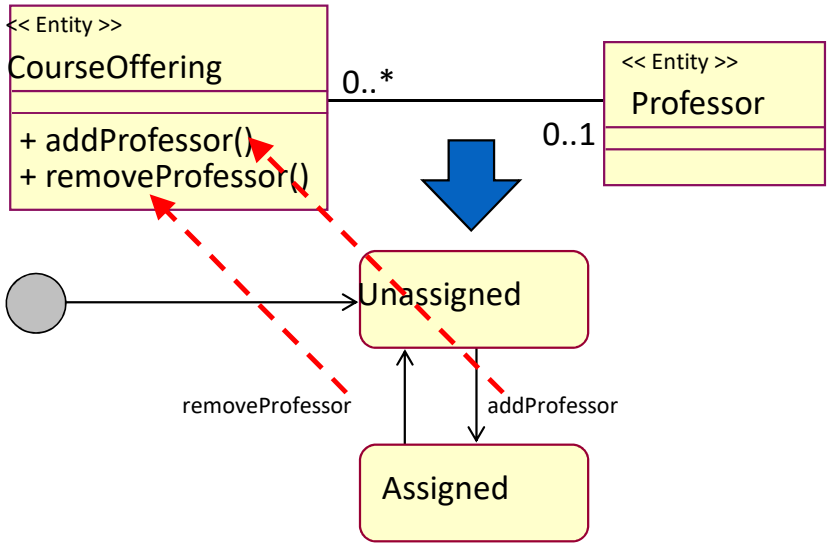
- Xem xét các hoạt động giao diện của lớp



Events: addProfessor, removeProfessor

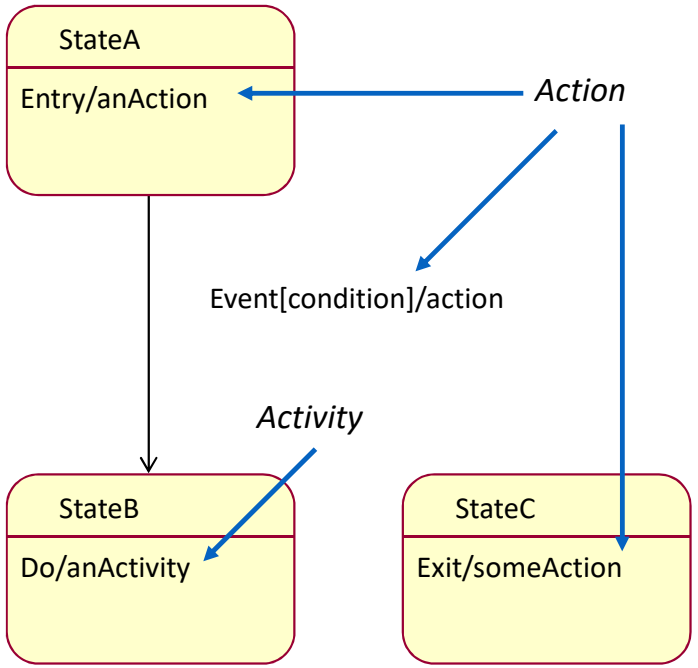
Nhận diện các chuyển đổi

- Với mỗi trạng thái, xác định các sự kiện gây ra sự chuyển đổi, xem xét cả các điều kiện
- Các chuyển đổi mô tả đáp ứng xảy ra khi đối tượng nhận sự kiện

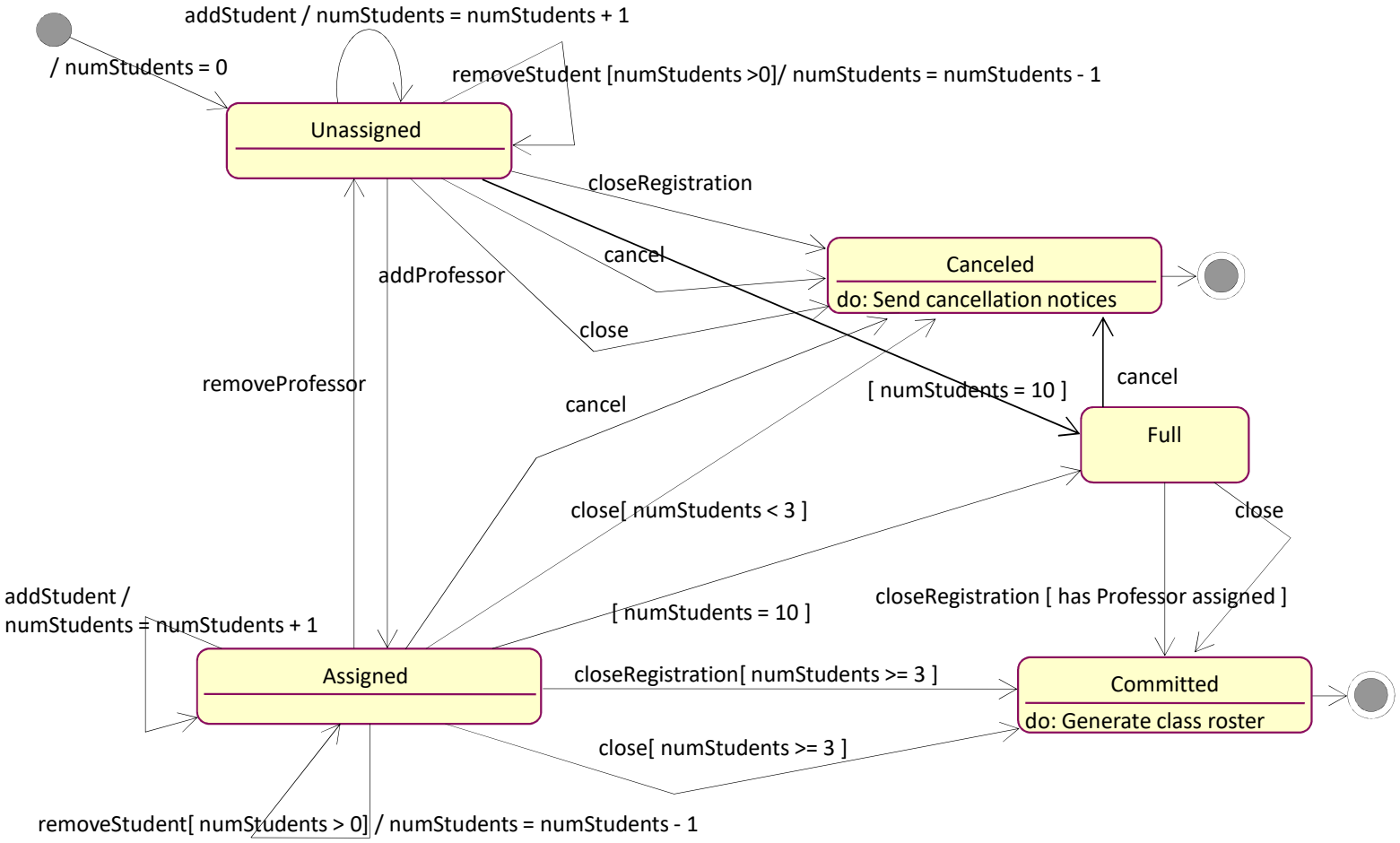


Bổ sung các hoạt động (activity), hành động (action)

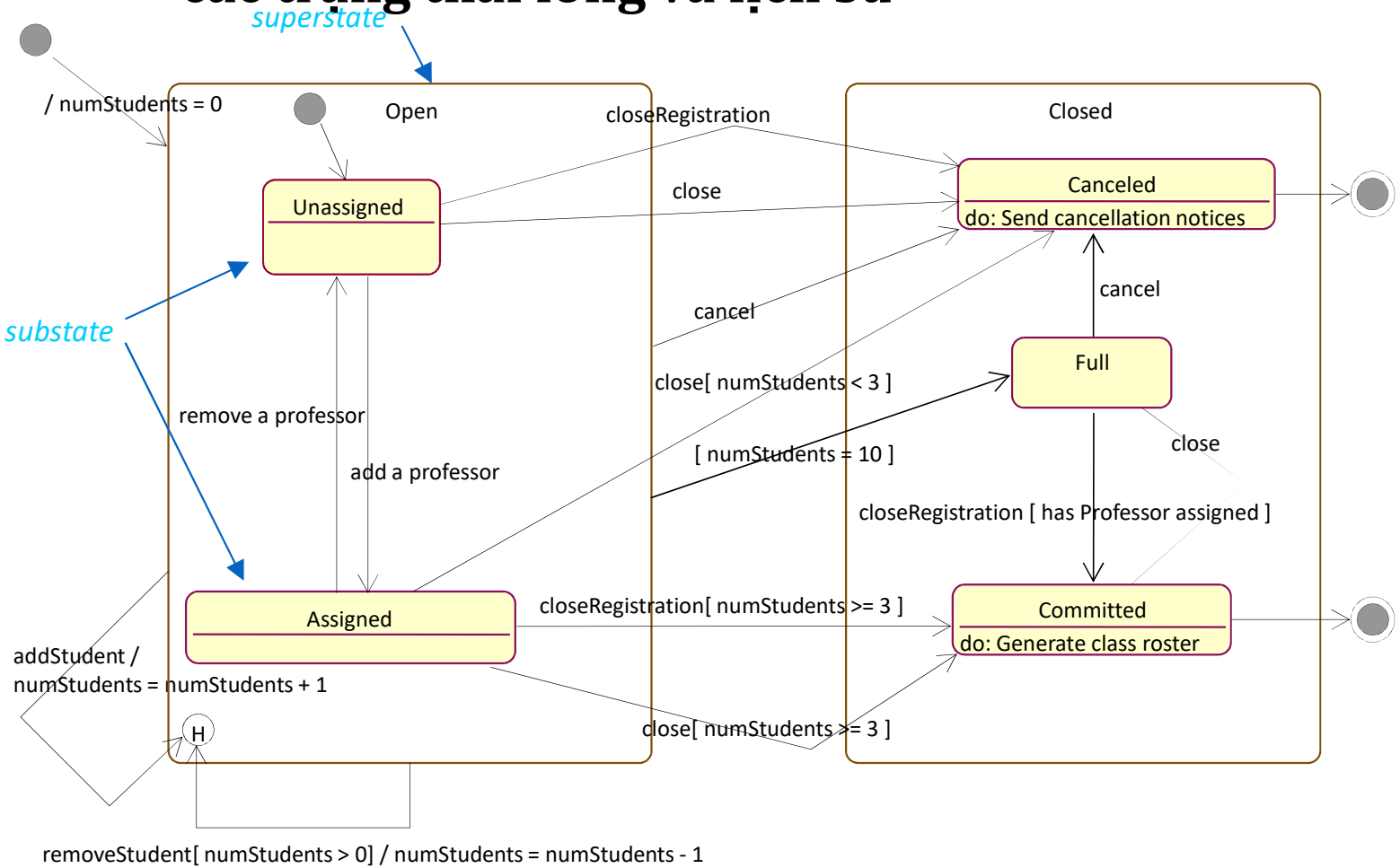
- Hoạt động
  - Liên kết với trạng thái
  - Thực hiện trong trạng thái
  - Có thể tốn thời gian để thực hiện
  - Có thể cắt ngang
- Hành động
  - Liên kết với 1 chuyển đổi
  - Thực hiện nhanh
  - Không thể cắt ngang



Thí dụ: Biểu đồ chuyển đổi trạng thái



Thí dụ: Biểu đồ chuyển đổi trạng thái với các trạng thái lồng và lịch sử

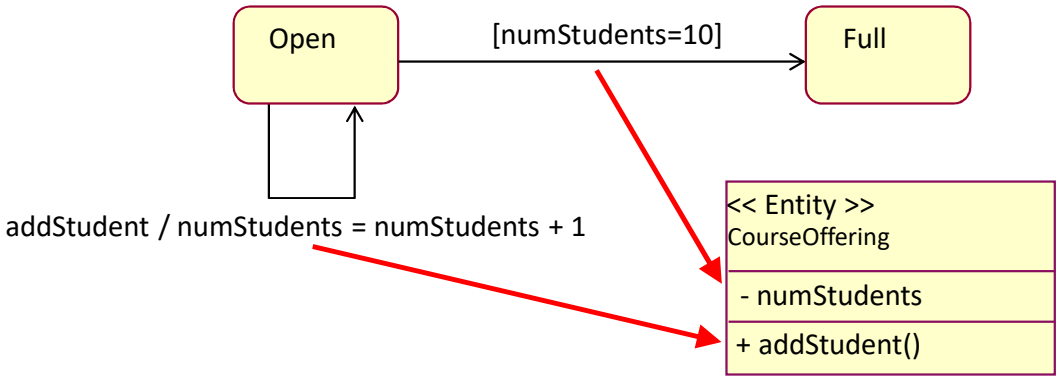


Các đối tượng nào có các trạng thái có nghĩa?

- Các đối tượng mà vai trò của nó được làm rõ bởi các chuyển đổi trạng thái
- Các trường hợp sử dụng phức tạp được điều khiển bởi trạng thái
- Không cần thiết khi mô hình các đối tượng như:
  - Các đối tượng ánh xạ trực tiếp vào thể hiện
  - Các đối tượng không được điểu khiển bởi trạng thái
  - Các đối tượng với chỉ một trạng thái tính toán

Biểu đồ trạng thái và các biểu đồ khác?

- Các sự kiện có thể ánh xạ tới các hoạt động
- Các phương thức nên được cập nhật với các thông tin trạng thái xác định
- Các trạng thái thường được thể hiện bởi các thuộc tính
  - Điều này tạo đầu vào cho bước "Xác định thuộc tính"



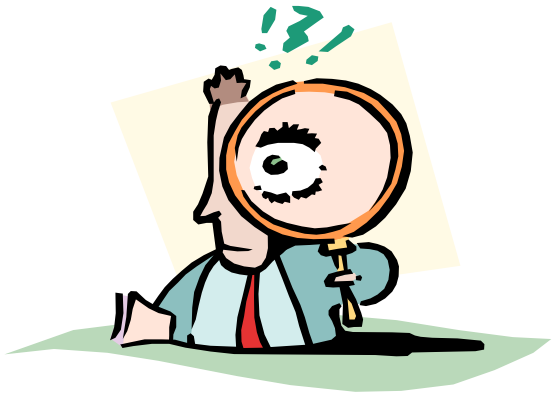
(Stay tuned for derived attributes)

Các bước thiết kế lớp

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

Các thuộc tính: Cách tìm chúng?

- Xem xét các mô tả phương thức
- Xem xét các trạng thái
- Xem xét các thông tin mà lớp cần duy trì



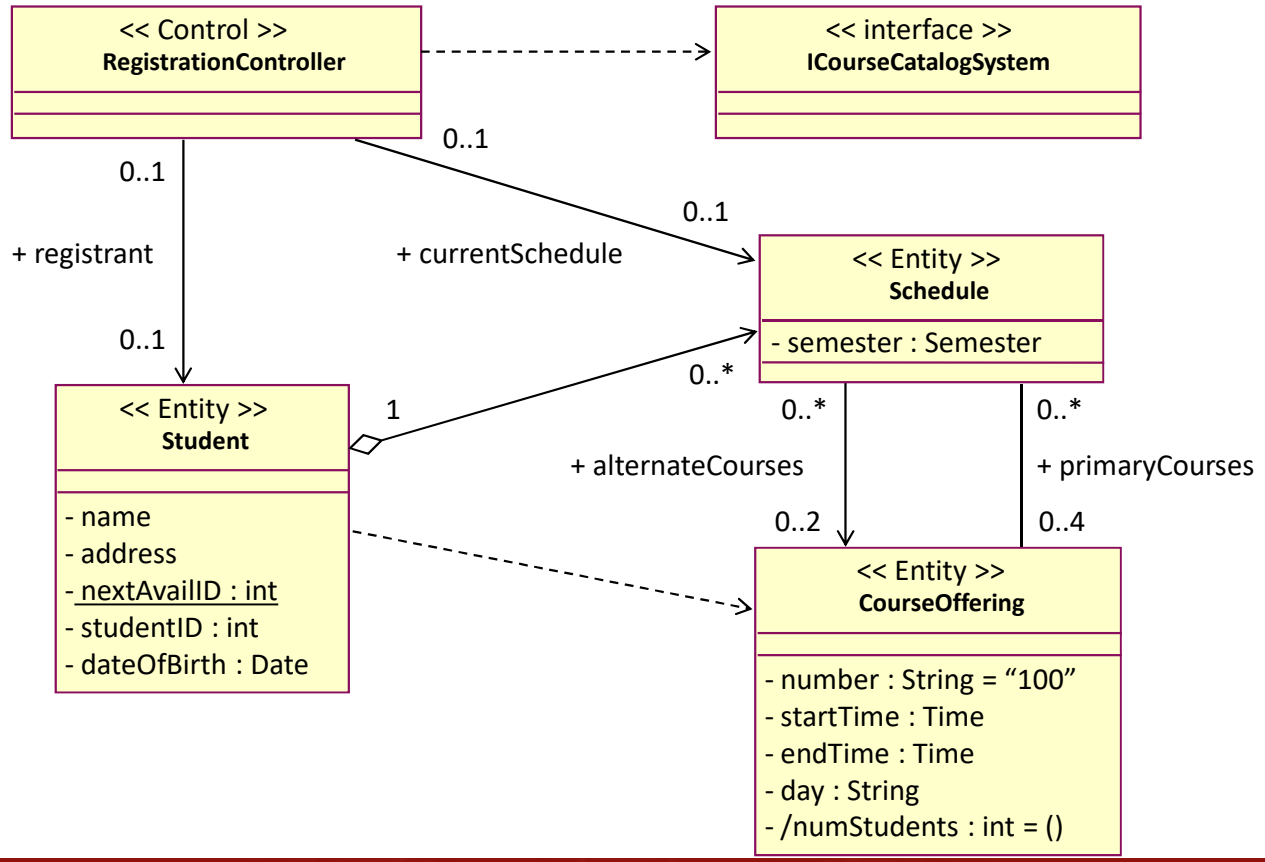
Thể hiện thuộc tính

- Xác định tên, kiểu, giá trị mặc định
  - attributeName : Type = Default
- Tuân thủ quy tắc đặt tên của ngôn ngữ thể hiện
- Kiểu nên là kiểu dữ liệu cơ bản của ngôn ngữ thể hiện
  - Built-in data type, user-defined data type, or user-defined class
- Xác định thuộc tính truy cập
  - Public: +
  - Private: -
  - Protected: #

Các thuộc tính dẫn xuất

- Thuộc tính dẫn xuất là gì?
  - Thuộc tính mà giá trị của nó có thể được tính toán dựa trên giá trị của các thuộc tính khác
- Khi nào sử dụng nó?
  - Khi không có đủ thời gian để tính lại mọi lần cần dùng tới nó
  - Khi thỏa hiệp giữa thời gian chạy và yêu cầu bộ nhớ

Thí dụ: Xác định thuộc tính

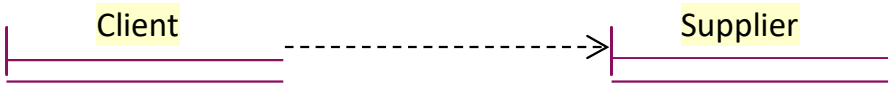


Các bước thiết kế lớp

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

Xác định sự phụ thuộc

- Phụ thuộc là gì?
  - Quan hệ giữa 2 đối tượng

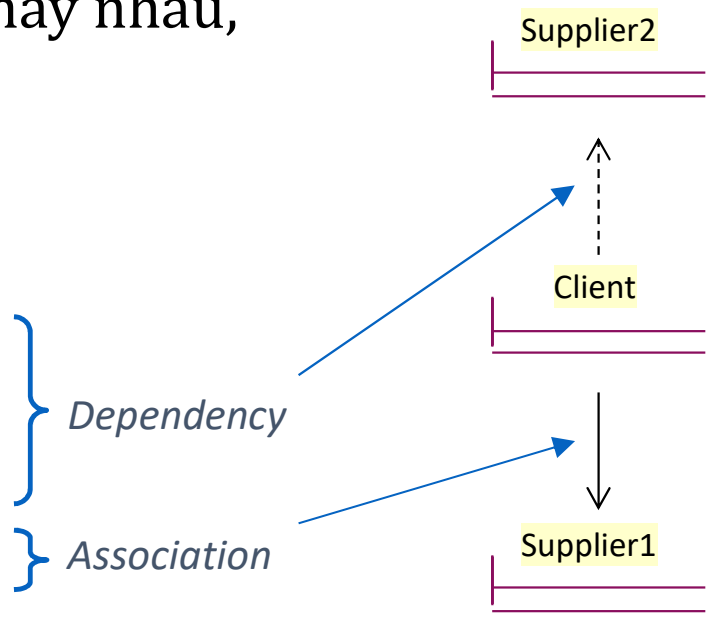


- Mục đích
  - Xác định khi quan hệ cấu trúc không đòi hỏi
- Cân nhắc:
  - Supplier được nhìn thấy bởi client thế nào

Phụ thuộc và Liên kết

- Liên kết là quan hệ cấu trúc
- Phụ thuộc là quan hệ phi cấu trúc
- Để các đối tượng nhìn thấy nhau, chúng phải có

- Local variable reference
- Parameter reference
- Global reference
- Field reference



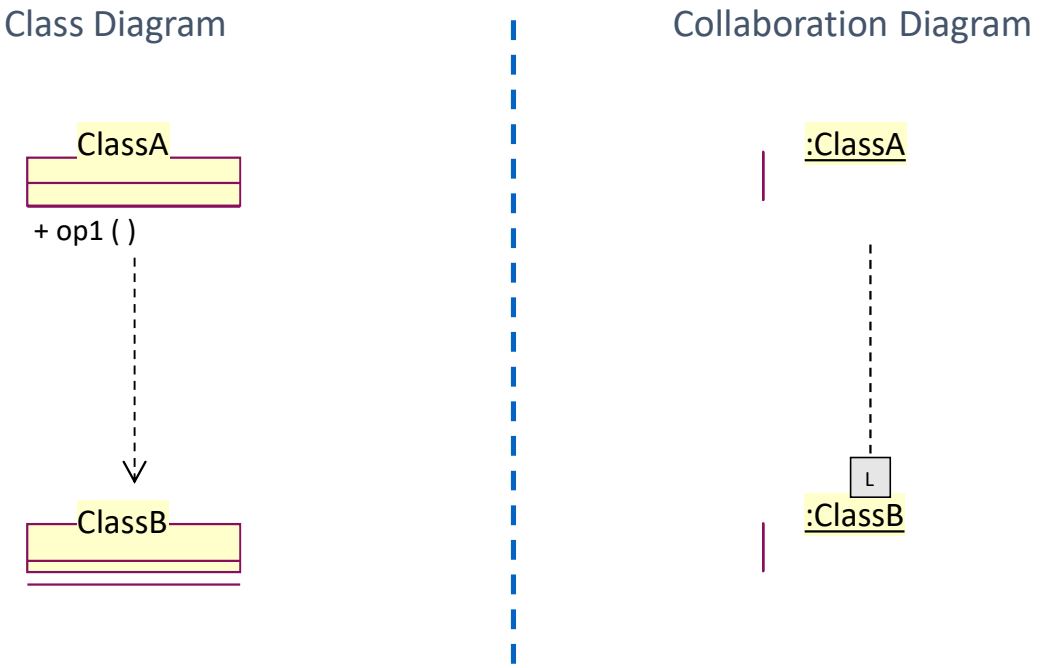
Liên kết và phụ thuộc trong sự cộng tác

- Một thể hiện của liên kết là 1 đường nối
  - Tất cả các đường nối trở thành liên kết nếu nó không thể hiện tham số, biến toán cục hay địa phương
  - Các quan hệ phụ thuộc ngữ cảnh
- Các phụ thuộc là các đường nối tạm thời:
  - Trong khoảng thời gian giới hạn
  - Quan hệ độc lập ngữ cảnh
  - Quan hệ tổng hợp

A dependency is a secondary type of relationship in that it doesn't tell you much about the relationship. For details you need to consult the collaborations.

Khả năng nhìn thấy biến địa phương

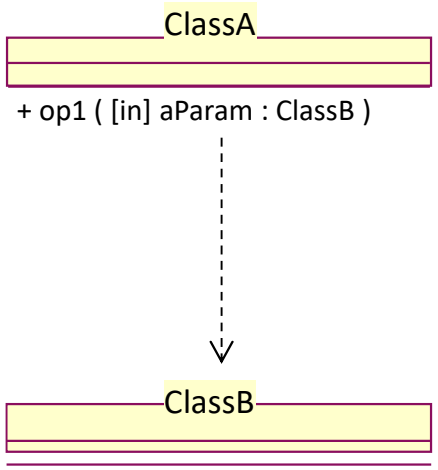
- Toán tử op1() chứa 1 biến địa phương của lớp ClassB



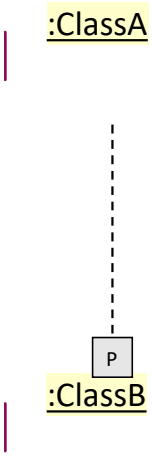
Khả năng nhìn thấy tham số

- Thể hiện của lớp ClassB được truyền tới thể hiện của lớp ClassA

Class Diagram

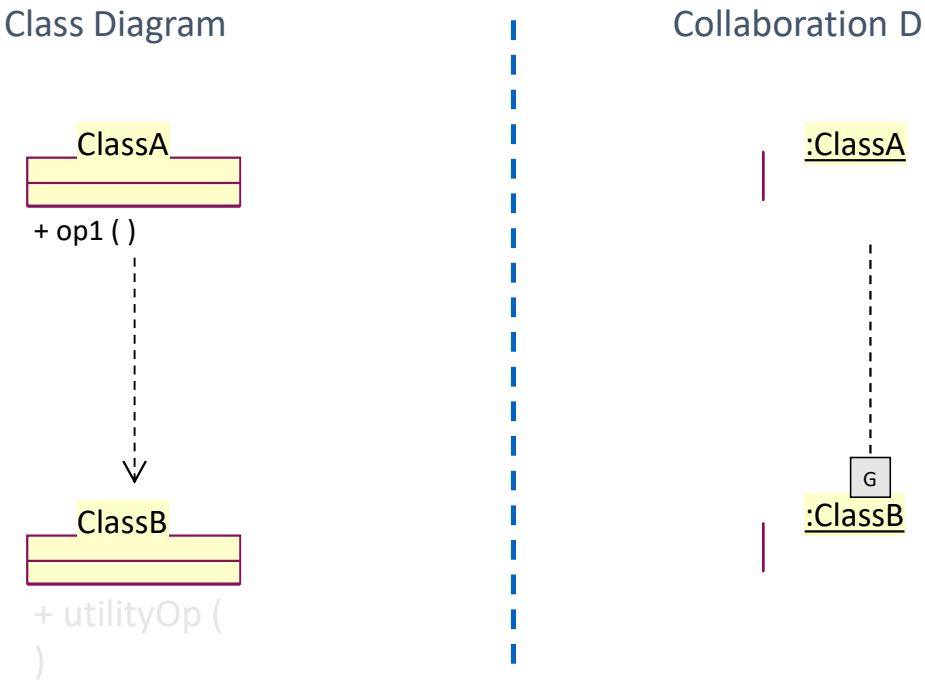


Collaboration Diagram



Khả năng nhìn thấy toàn cục

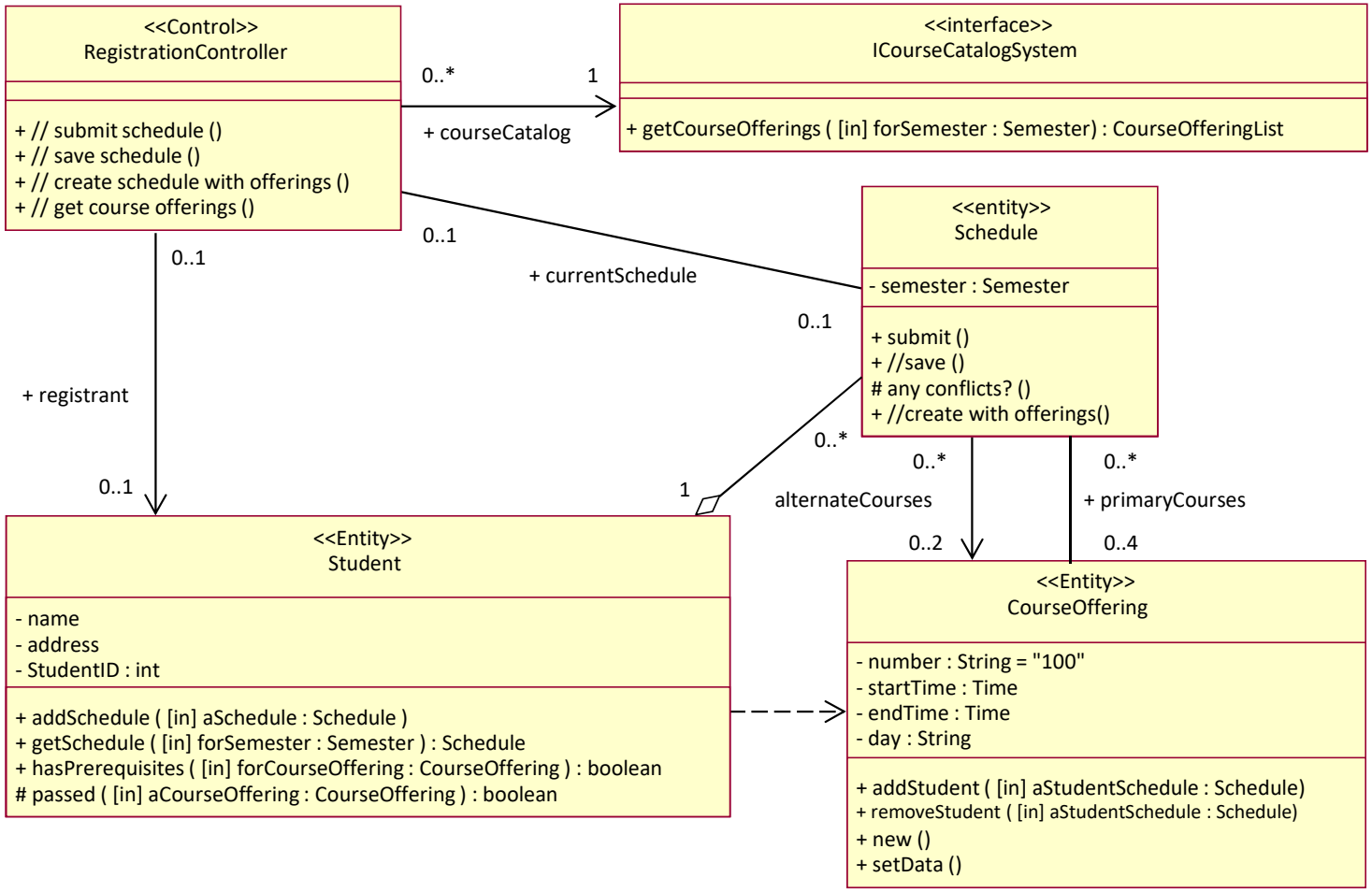
- Thể hiện của lớp ClassB được nhìn thấy bởi nó là toàn cục



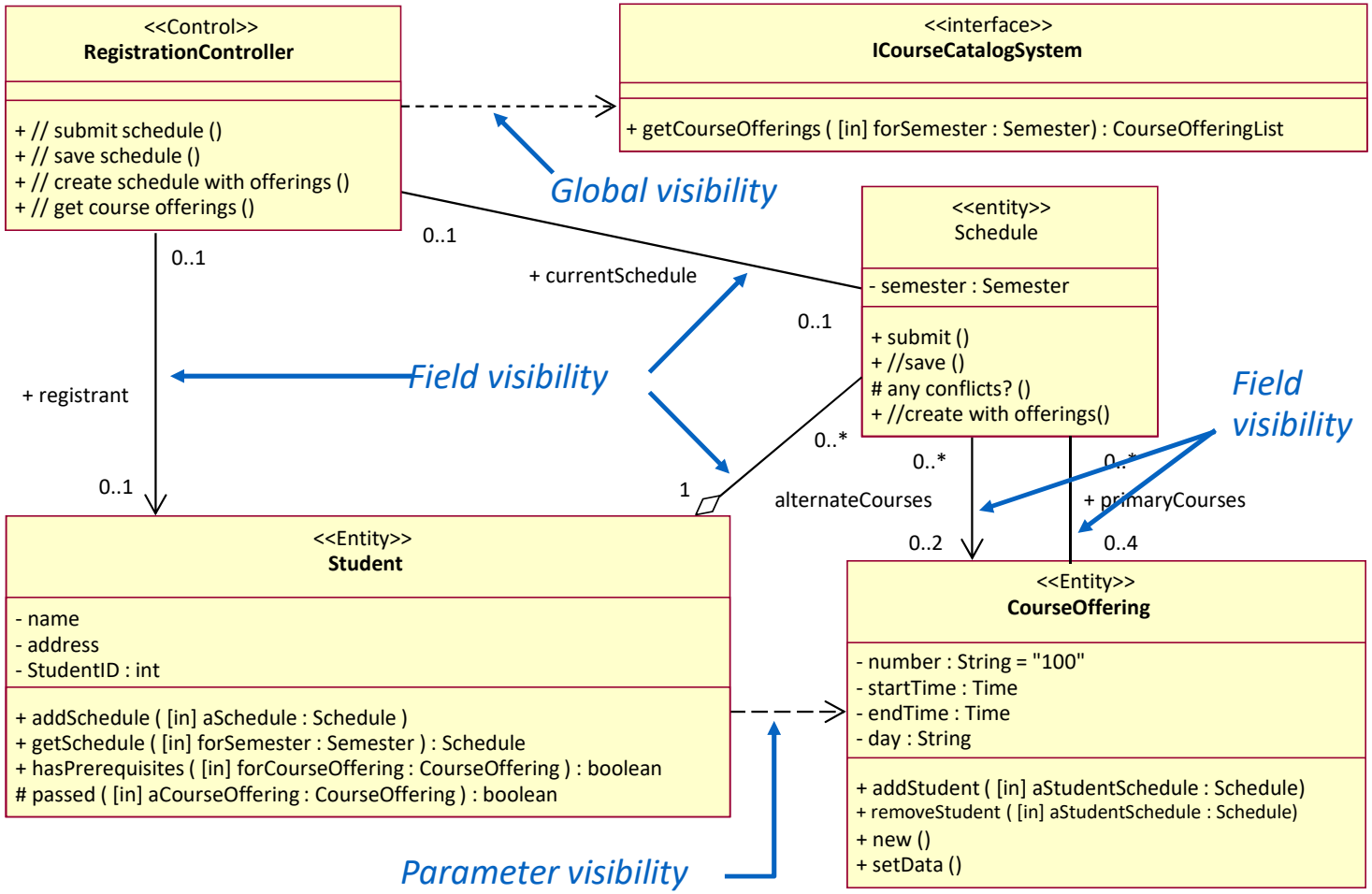
Nhận diện sự phụ thuộc

- Quan hệ lâu dài — Liên kết (Nhìn thấy ở mức trường)
- Quan hệ tạm thời — Phụ thuộc
  - Nhiều đối tượng chia sẻ cùng 1 thể hiện
    - Truyền thể hiện như tham số (parameter visibility)
    - Tạo thể hiện như một đối tượng toàn cục (global visibility)
  - Các đối tượng không chia sẻ cùng 1 thể hiện (local visibility)
- Thời gian để tạo và hủy bỏ?
  - Đắt? Sử dụng trường, tham số hay toàn cục
  - Cố cho quan hệ nhẹ nhất có thể

Thí dụ: Xác định sự phụ thuộc (trước)



Thí dụ: Xác định sự phụ thuộc (sau)



Các bước thiết kế lớp

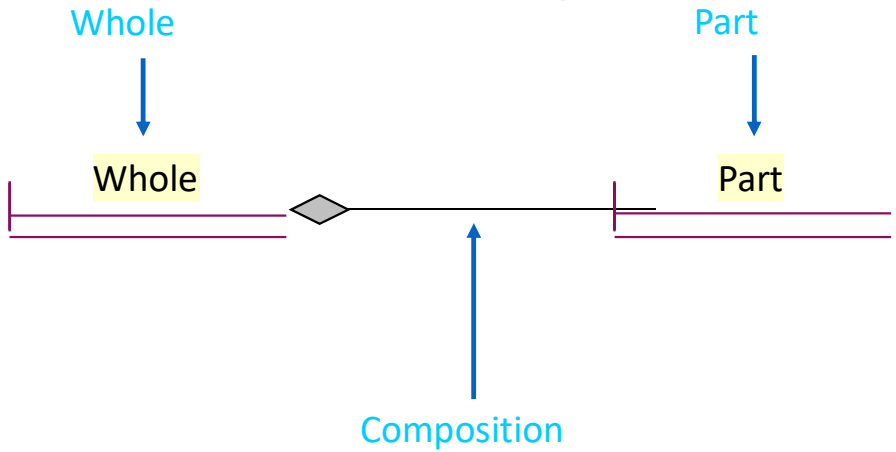
- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định sự tổng quát hóa
- Giải quyết các xung đột usecase
- Đáp ứng các yêu cầu phi chức năng

Xác định các liên kết

- Mục đích
  - Làm tốt các liên kết còn lại
- Cân nhắc các nội dung:
  - Association vs. Aggregation
  - Aggregation vs. Composition
  - Attribute vs. Association
  - Navigability
  - Association class design
  - Multiplicity design

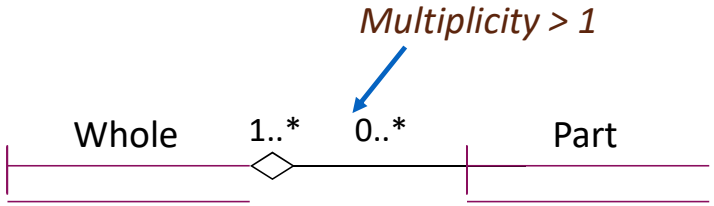
Hợp phần (Composition) là gì?

- Là một dạng của tổ hợp (aggregation), có quan hệ mạnh và cùng vòng đời
  - Bộ phận không thể sống sót khi thiếu tổng thể

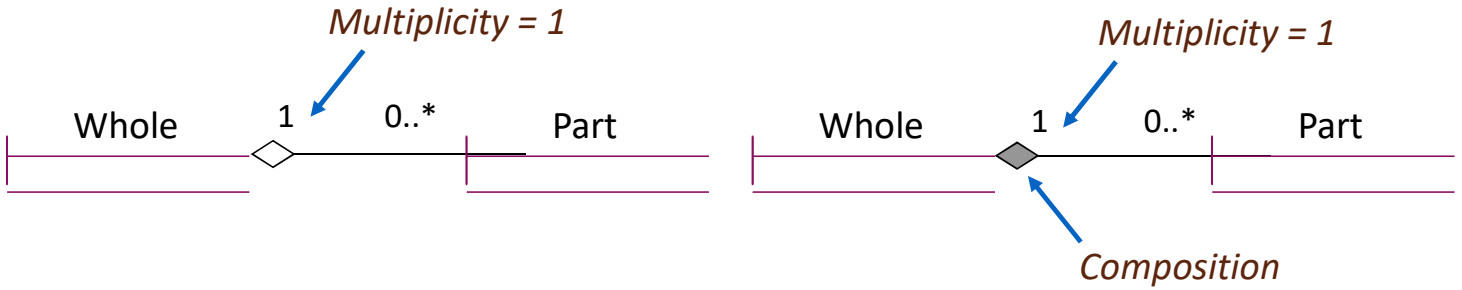


Tổ hợp: Chia sẻ hay không chia sẻ

- Shared Aggregation



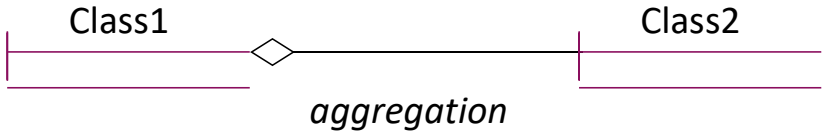
- Non-shared Aggregation



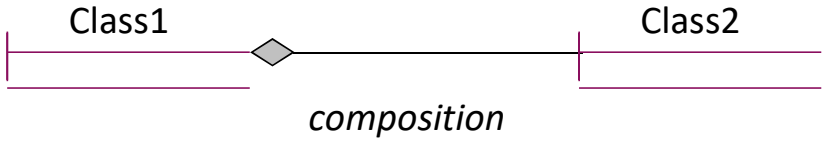
By definition, composition is non-shared aggregation

Tổ hợp hay hợp phần?

- Cân nhắc
  - Lifetimes of Class1 and Class2

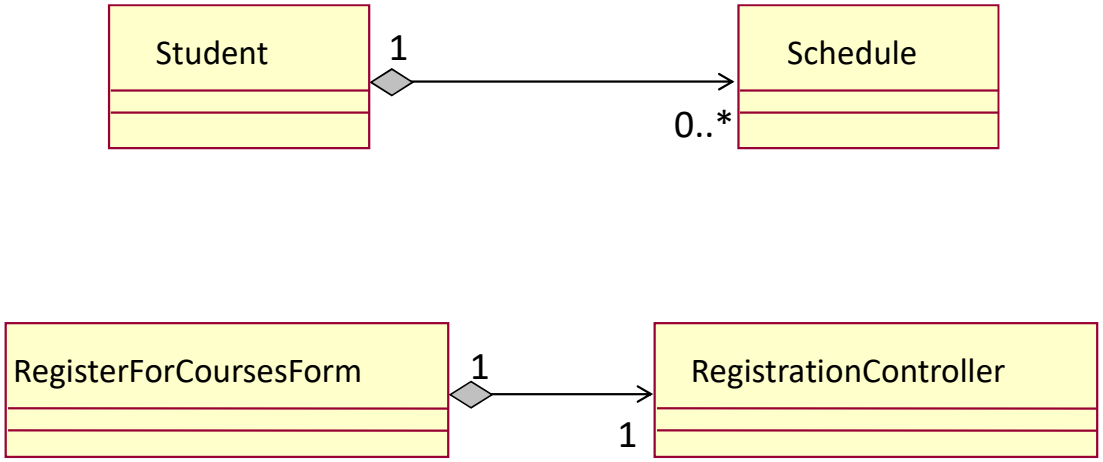


aggregation



composition

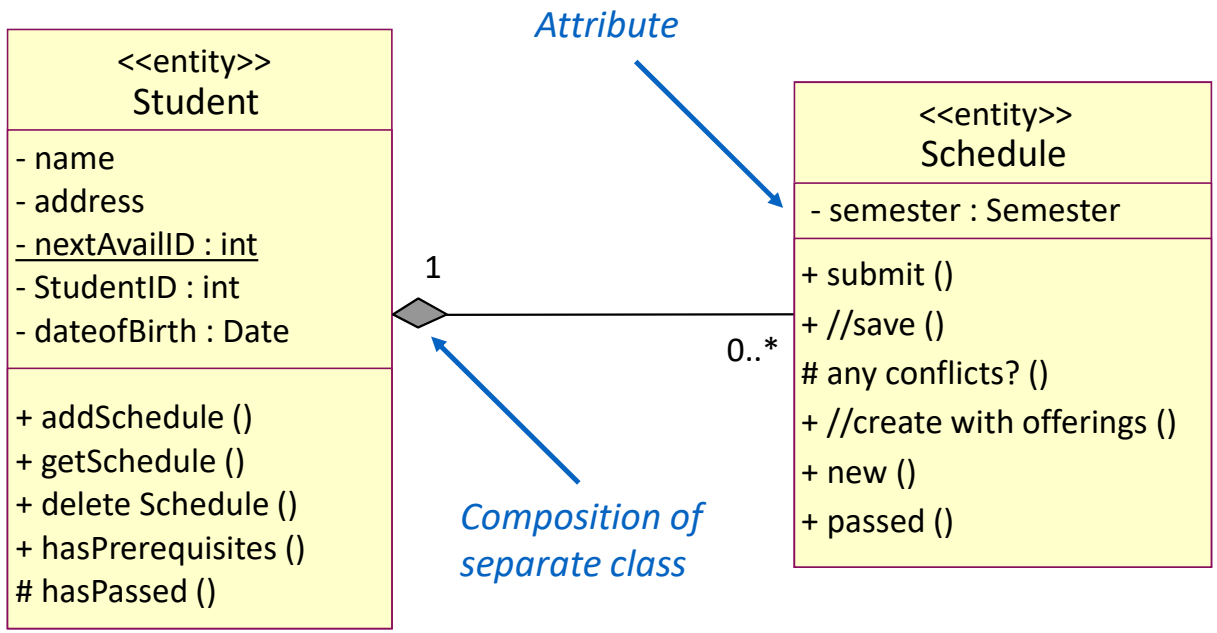
Thí dụ: Composition



Thuộc tính và hợp phần

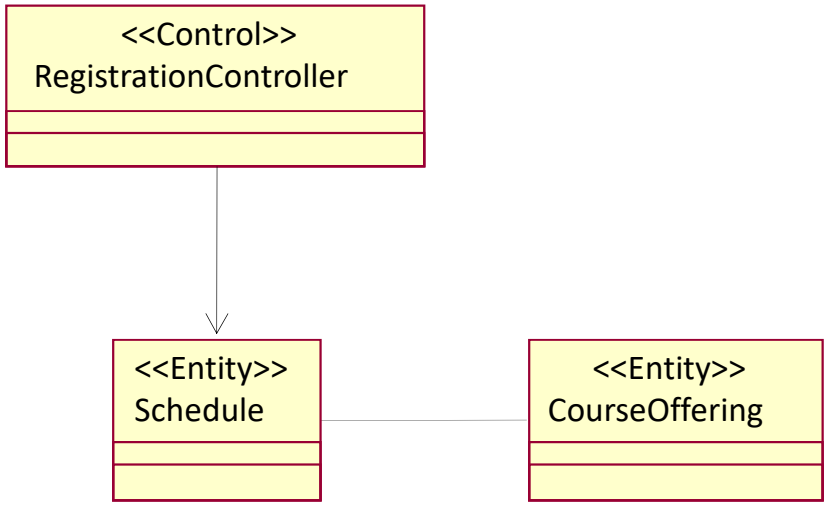
- Sử dụng hợp phần khi:
  - Các thuộc tính cần định danh độc lập
  - Nhiều lớp có cùng các thuộc tính
  - Các thuộc tính có cấu trúc phức tạp và có thuộc tính của chính nó
  - Các thuộc tính có hành vi phức tạp
  - Các thuộc tính có các quan hệ
- Nếu khác đi, nên sử dụng thuộc tính

Thí dụ: Attributes vs. Composition



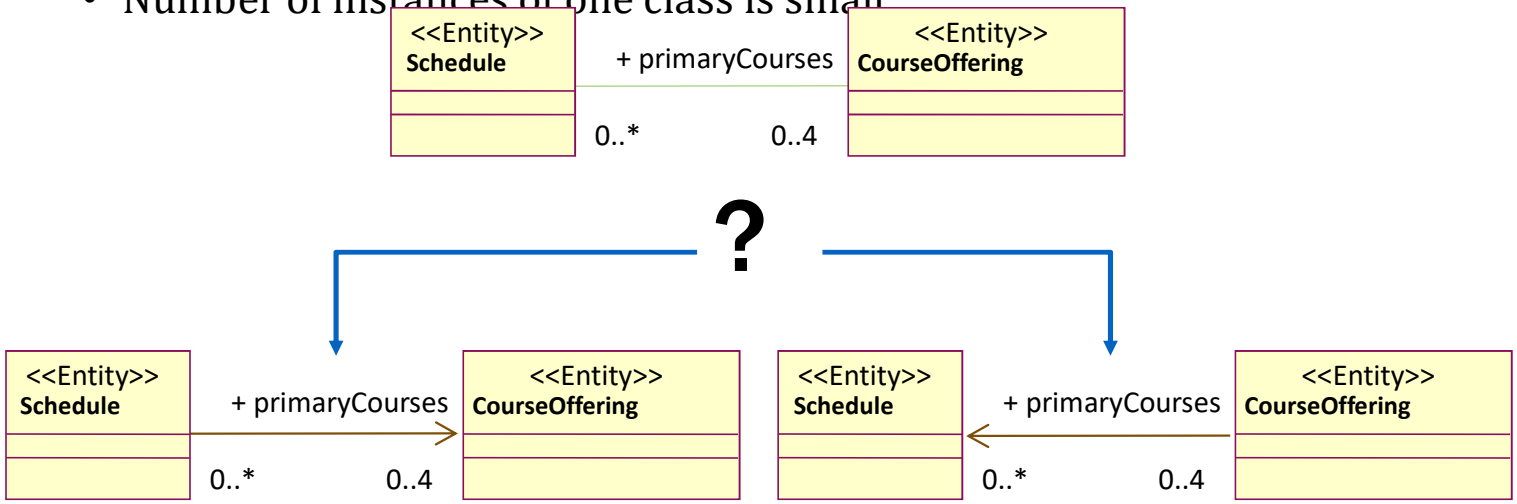
Nhắc lại: Navigability là gì?

- Chỉ báo khả năng di chuyển từ một lớp đang được liên kết tới lớp đích sử dụng liên kết



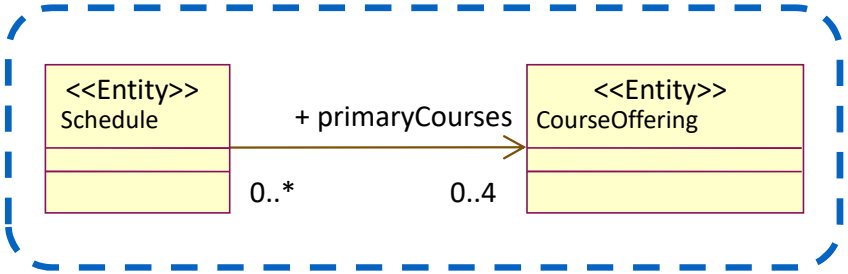
Khả năng dịch chuyển: Hướng nào thực sự cần thiết?

- Xem xét biểu đồ tương tác
- Thậm chí khi đòi hỏi cả 2 hướng, 1 hướng vẫn có thể đáp ứng
  - Navigability in one direction is infrequent
  - Number of instances of one class is small

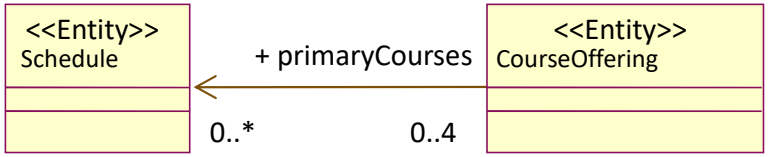


Thí dụ: Navigability Refinement

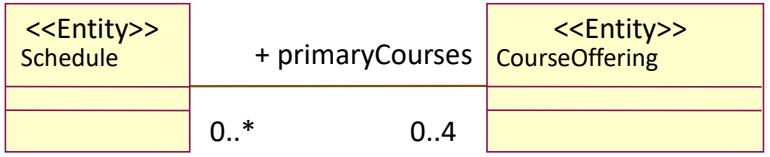
- Total number of Schedules is small, or
- Never need a list of the Schedules on which the CourseOffering appears



- Total number of CourseOfferings is small, or
- Never need a list of CourseOfferings on a Schedule

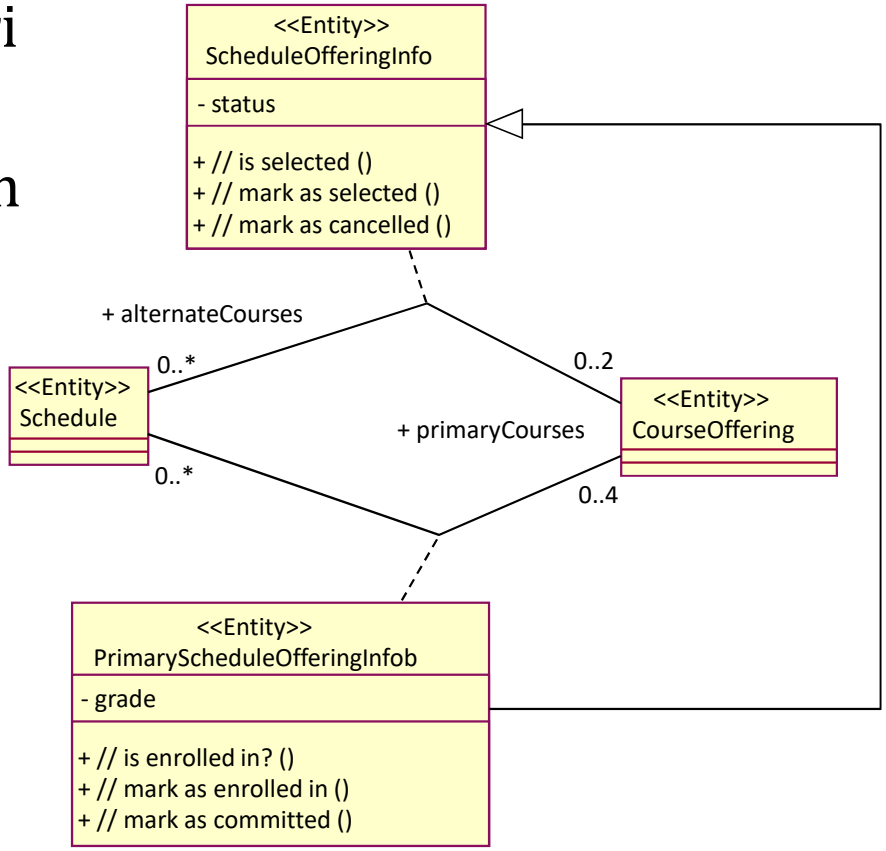


- Total number of CourseOfferings and Schedules are not small
- Must be able to navigate in both directions

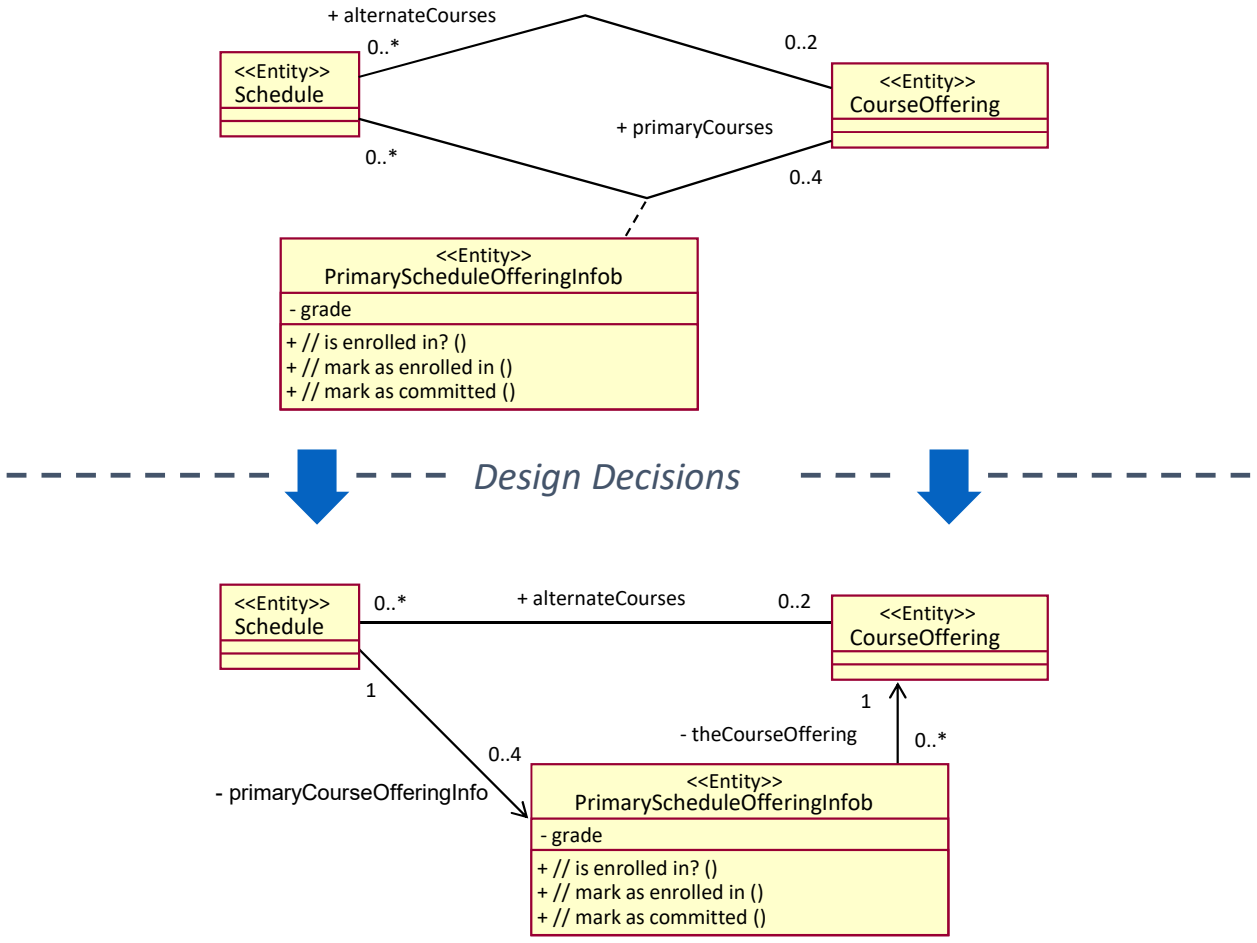


Lớp liên kết

- Lớp được "đính"với 1 liên kết
- Chứa các thuộc tính của liên kết
- Có 1 thể hiện mỗi liên kết

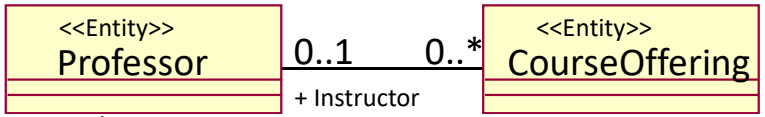


Thí dụ: Thiết kế lớp liên kết



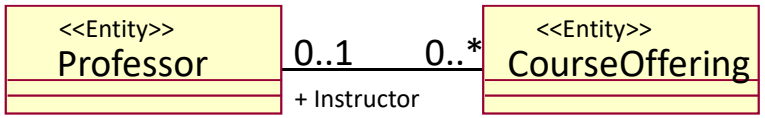
Thiết kế ứng số

- Multiplicity = 1, or Multiplicity = 0..1
  - Có thể được thể hiện trực tiếp như giá trị đơn hay con trỏ
  - Không cần thiết kế thêm

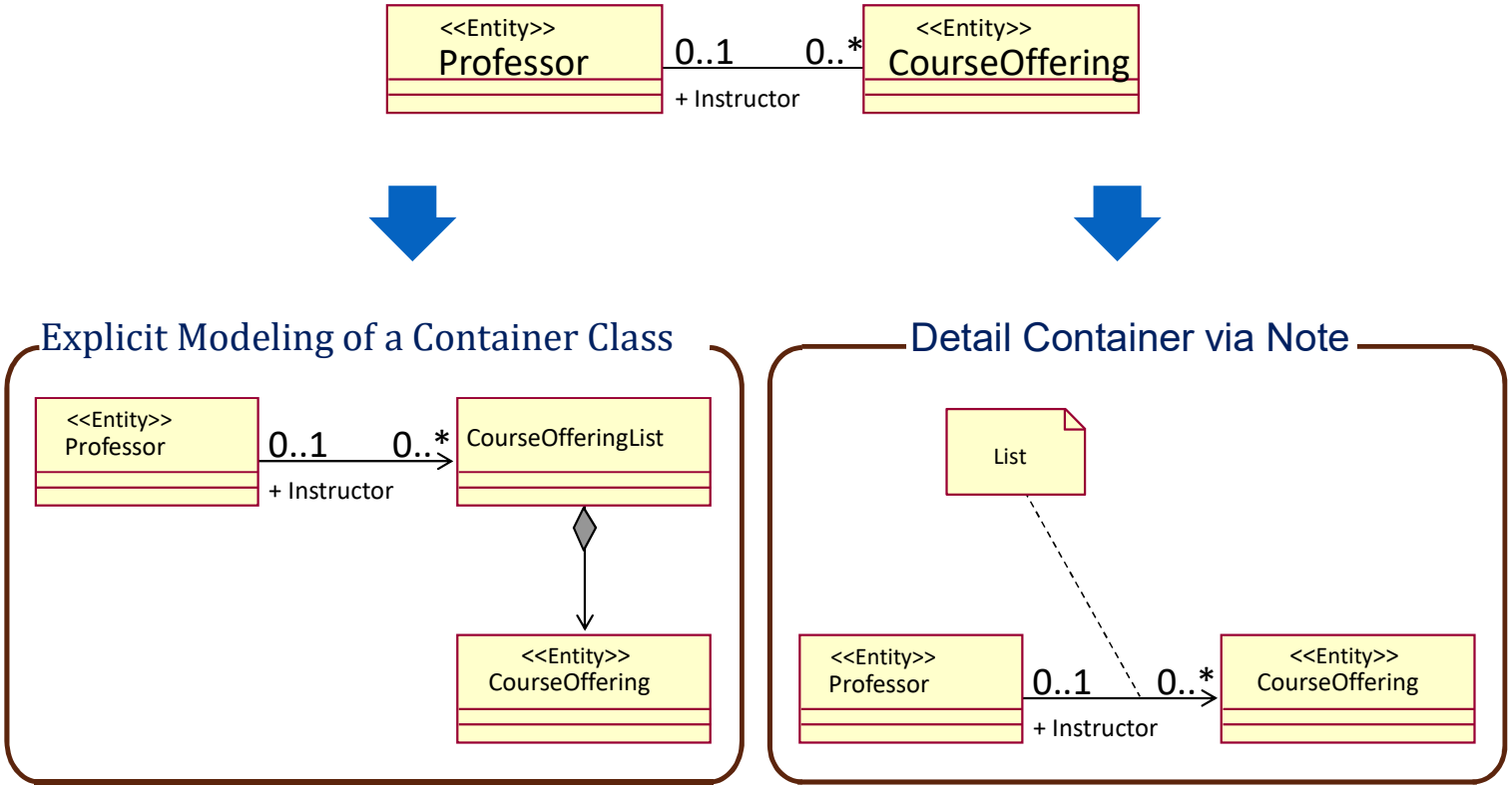


- Multiplicity > 1
  - Không thể sử dụng giá trị đơn hay con trỏ
  - Cần thiết kế thêm

Needs a container for CourseOfferings

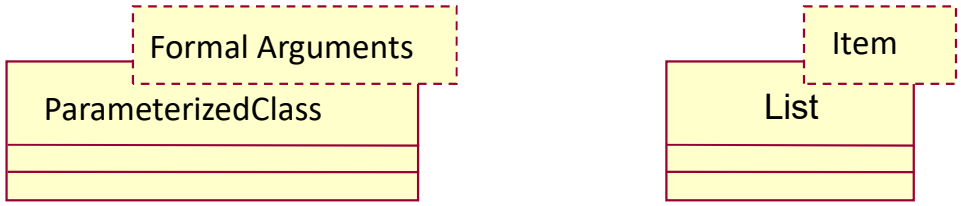


Các lựa chọn thiết kế ứng số

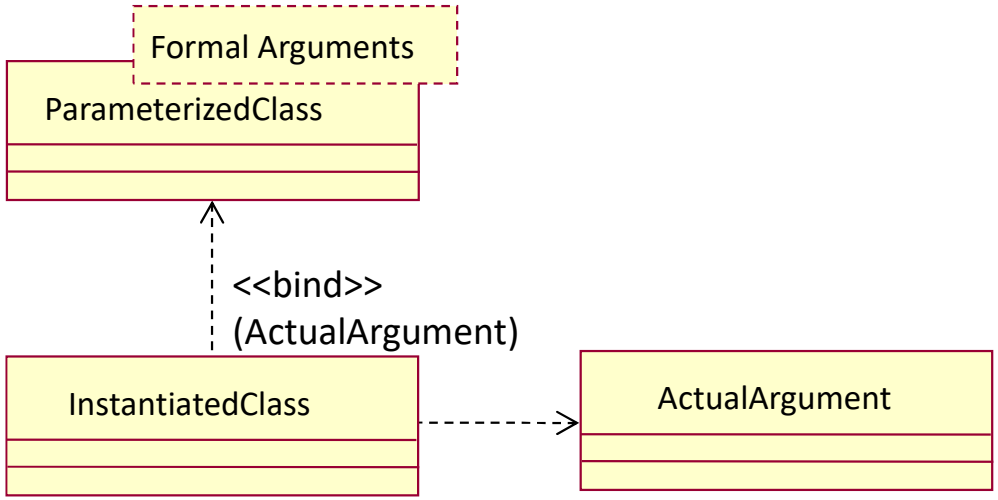


Lớp tham số là gì (template)?

- Định nghĩa lớp xác định họ các lớp khác
- Thường được dùng cho các lớp container
  - Một số lớp container chung:
    - Sets, lists, dictionaries, stacks, queues

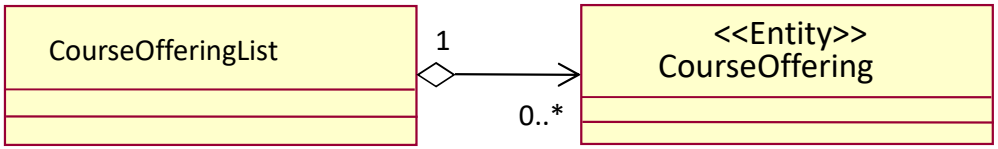


Thể hiện 1 lớp tham số

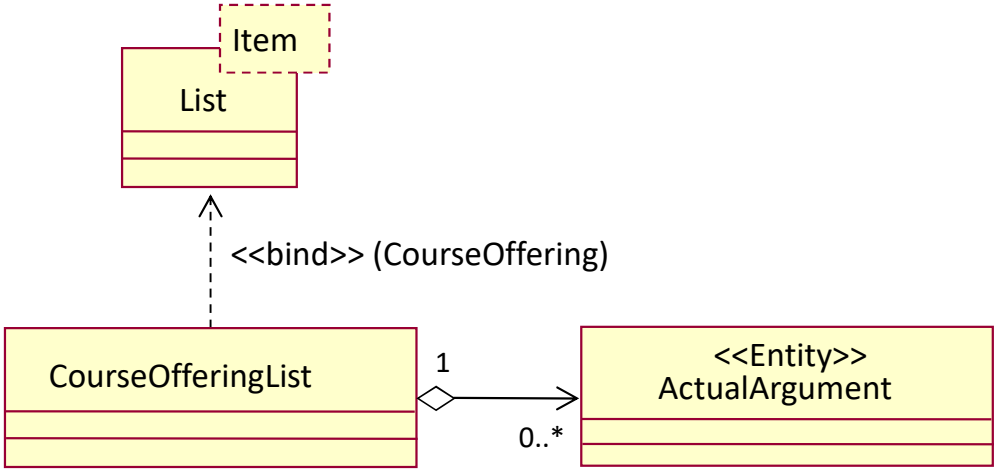


Thí dụ: Thể hiện 1 lớp tham số

Before



After



Thiết kế ứng số: Khả năng lựa chọn

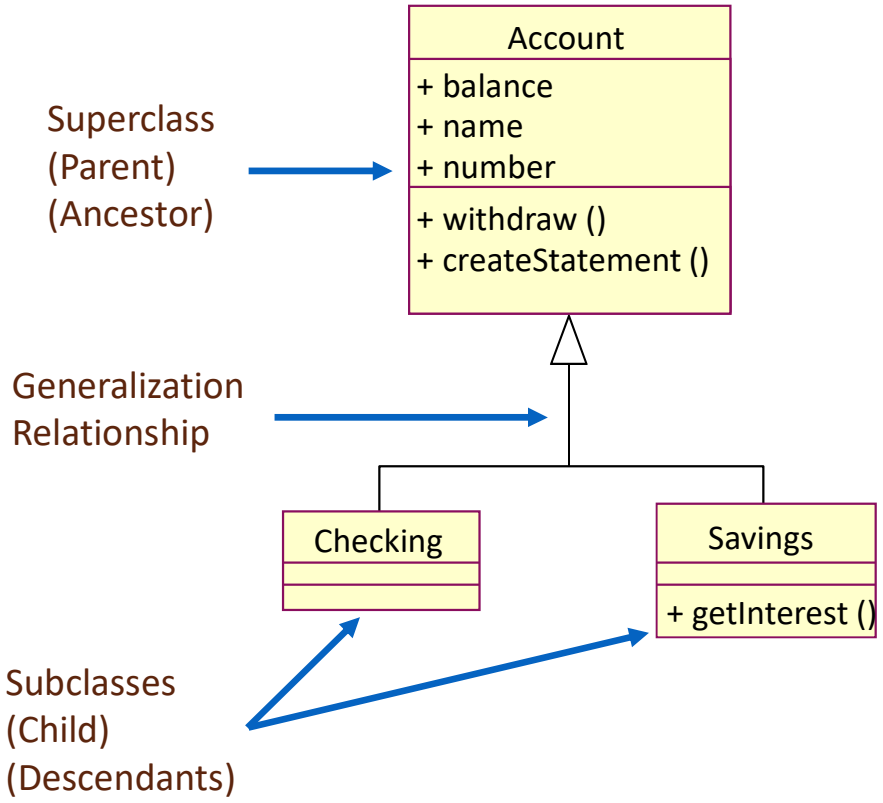
- Nếu liên kết là lựa chọn, nên tạo phương thức kiểm tra sự tồn tại của liên kết

Các bước thiết kế lớp

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định các sự tổng quát hóa
- Giải quyết các xung đột usecase
- Thể hiện các yêu cầu phi chức năng

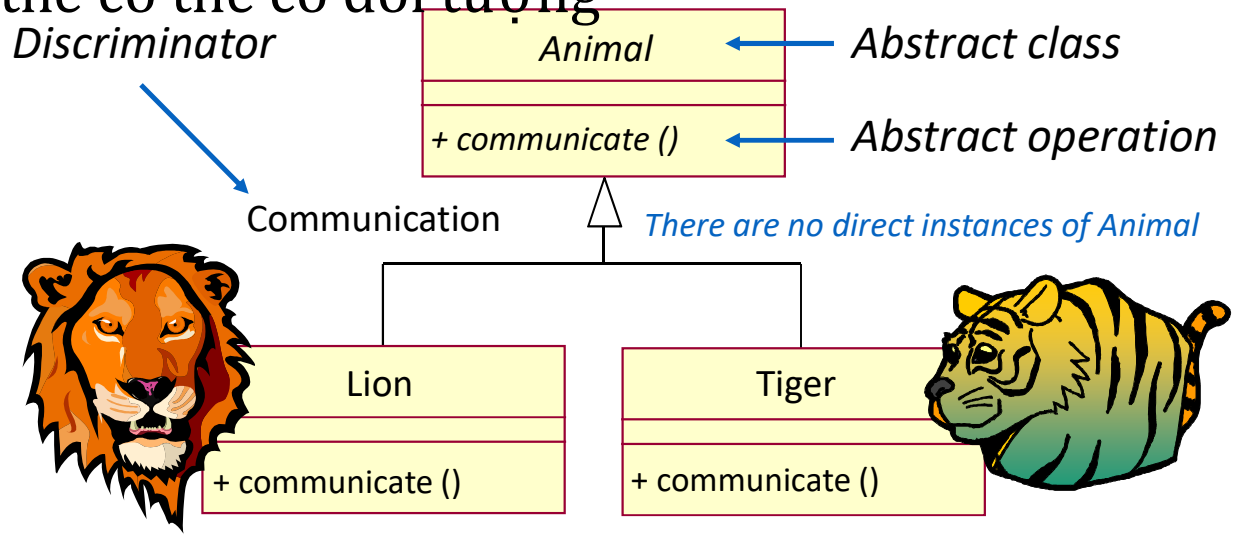
Tổng quát hóa

- Lớp chia sẻ cấu trúc hay hành vi với một hoặc nhiều lớp khác
- Quan hệ "là một dạng của"



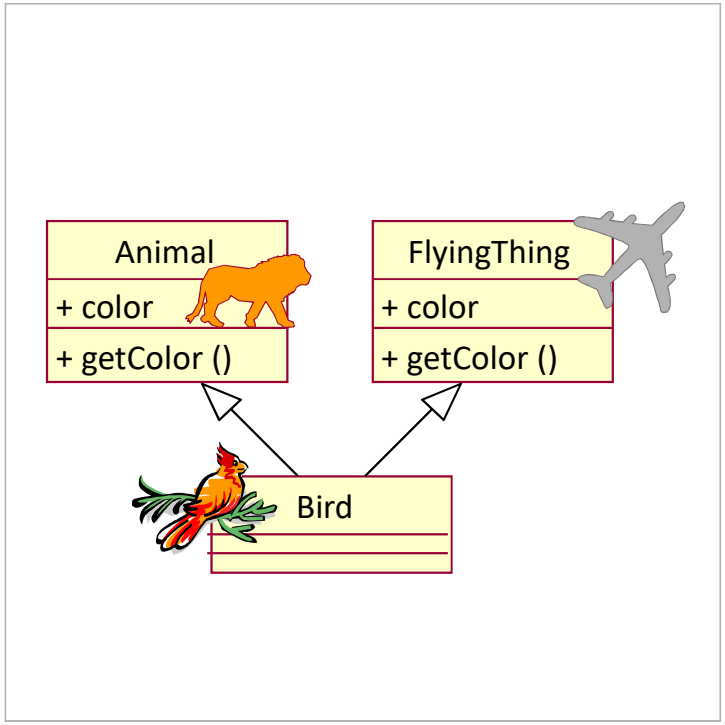
Lớp trừu tượng và lớp cụ thể

- Lớp trừu tượng không có đối tượng
- Lớp cụ thể có thể có đối tượng

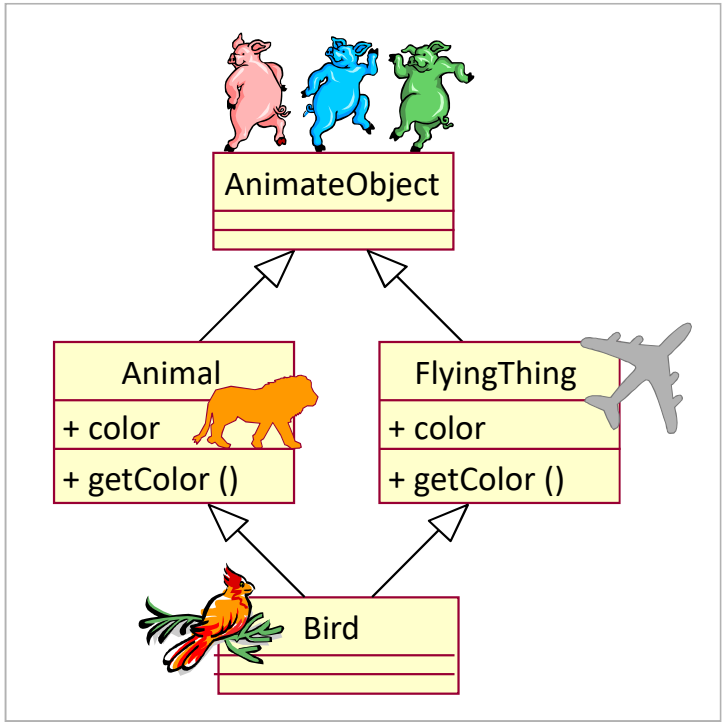


Vấn đề đa thừa kế

Name clashes on
attributes or operations



Repeated inheritance

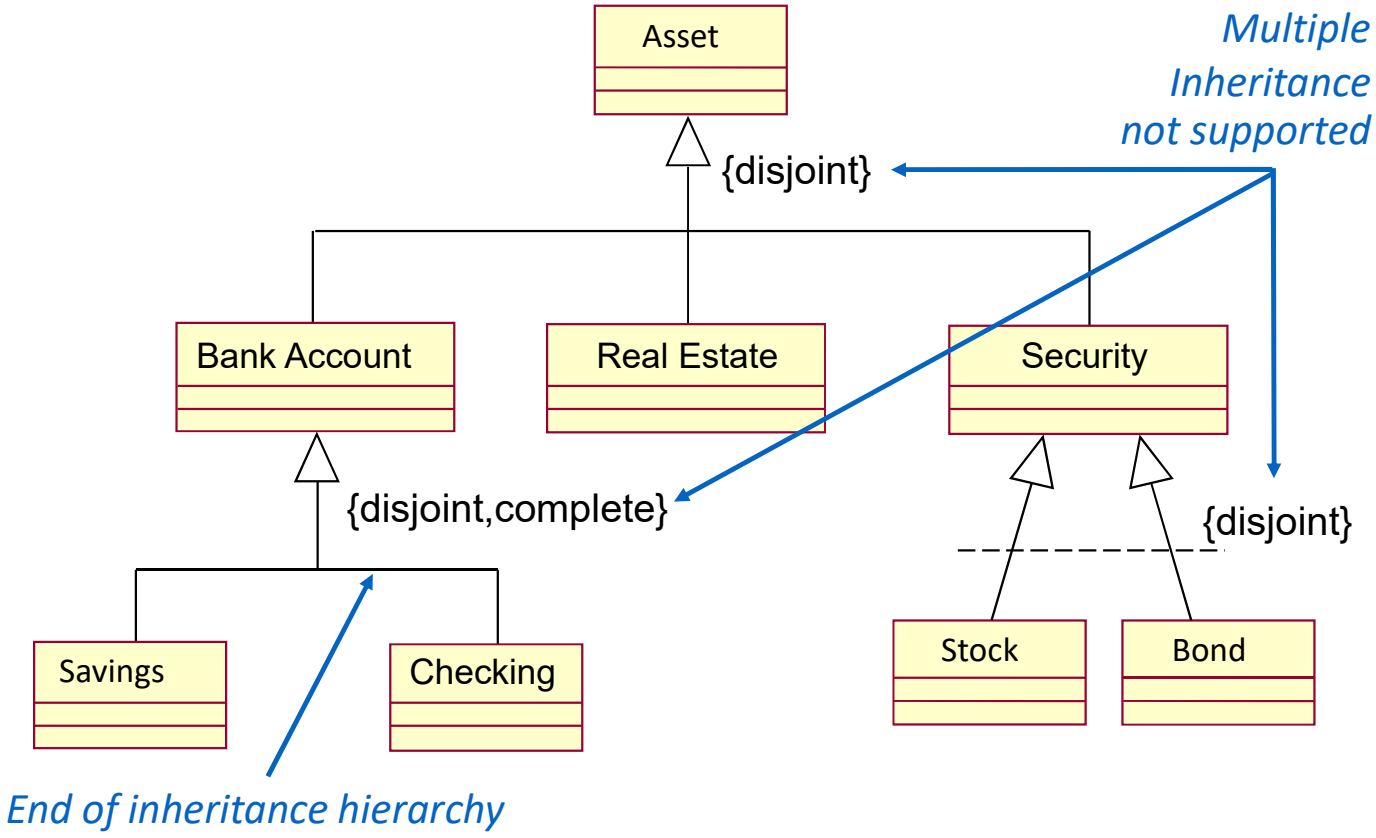


Resolution of these problems is implementation-dependent

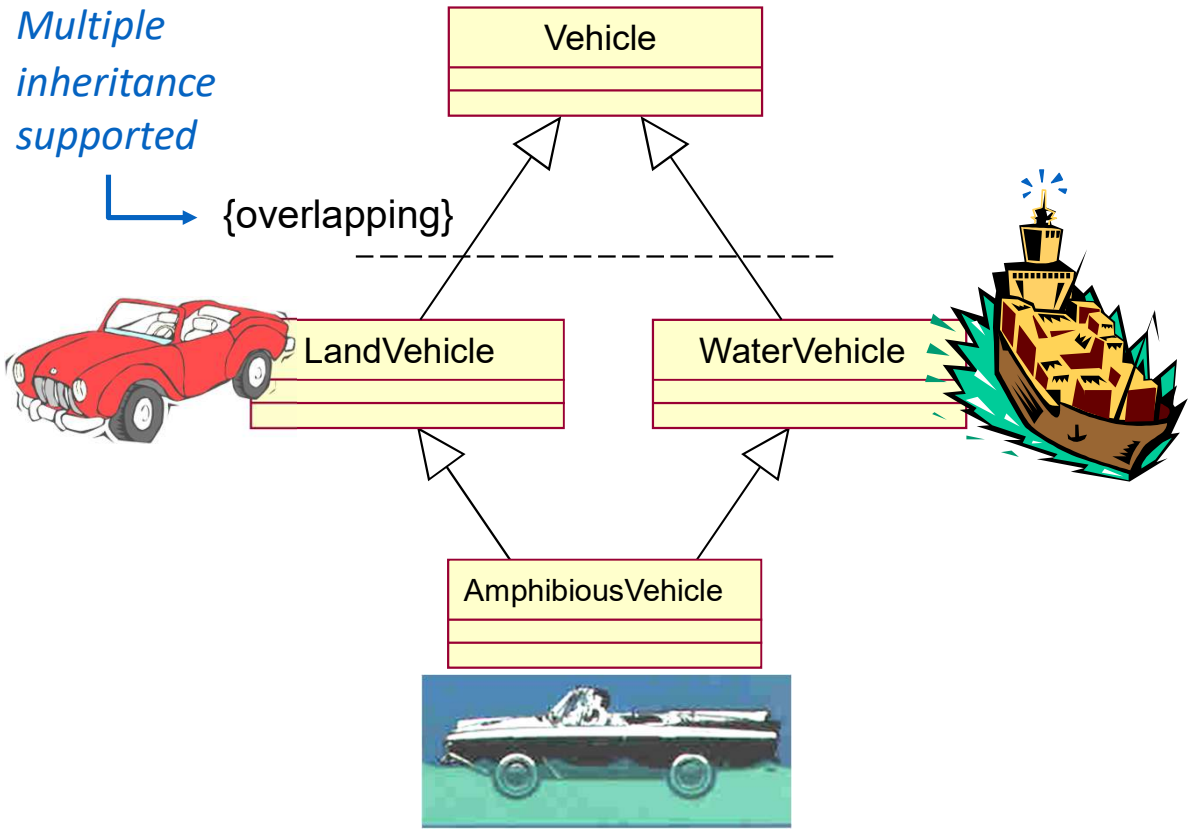
Các ràng buộc tổng quát hóa

- Hoàn chỉnh
  - Cuối cây thừa kế
- Không hoàn chỉnh
  - Cây thừa kế có thể mở rộng
- Không giao nhau
  - Các lớp con loại trừ lẫn nhau
  - Không hỗ trợ đa thừa kế
- Chồng lấn
  - Các lớp con không loại trừ nhau
  - Hỗ trợ đa thừa kế

Thí dụ

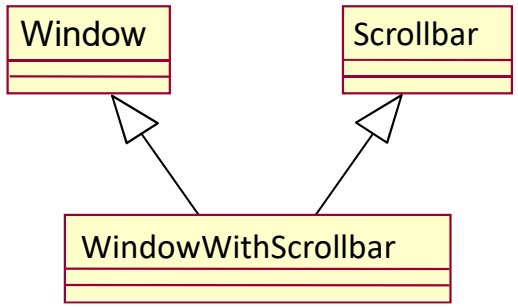

Asset
{disjoint}
Multiple
Inheritance
not supported
Bank Account
Real Estate
Security
{disjoint,complete}
{disjoint}
Savings
Checking
Stock
Bond
End of inheritance hierarchy

Thí dụ (tiếp)


Multiple
inheritance
supported
{overlapping}
Vehicle
LandVehicle
WaterVehicle
AmphibiousVehicle

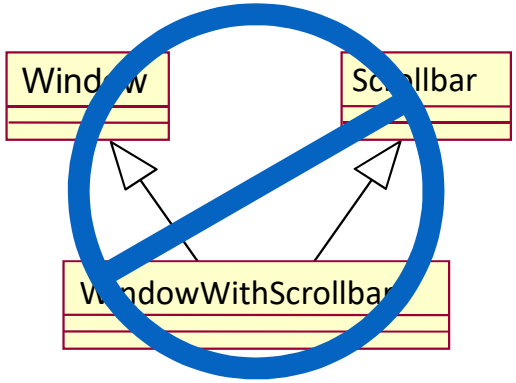
Tổng quát hóa hay tổ hợp

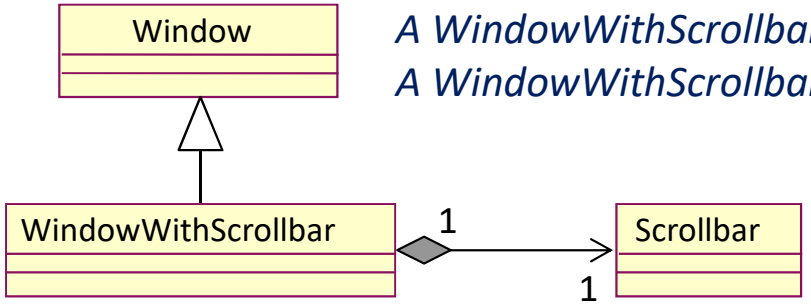
- Thường nhầm lẫn
  - Tổng quát hóa thể hiện quan hệ “is a” hay “kind-of”
  - Tổ hợp thể hiện quan hệ “part-of”



Is this correct?

Tổng quát hóa hay tổ hợp

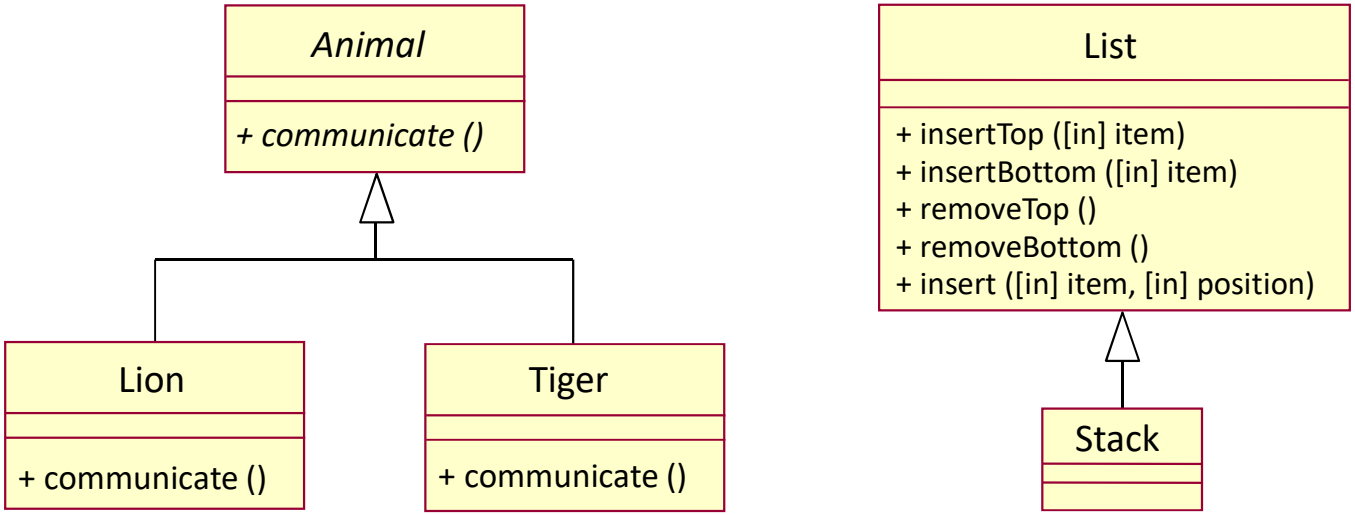




A WindowWithScrollbar "is a" Window
A WindowWithScrollbar "contains a" Scrollbar

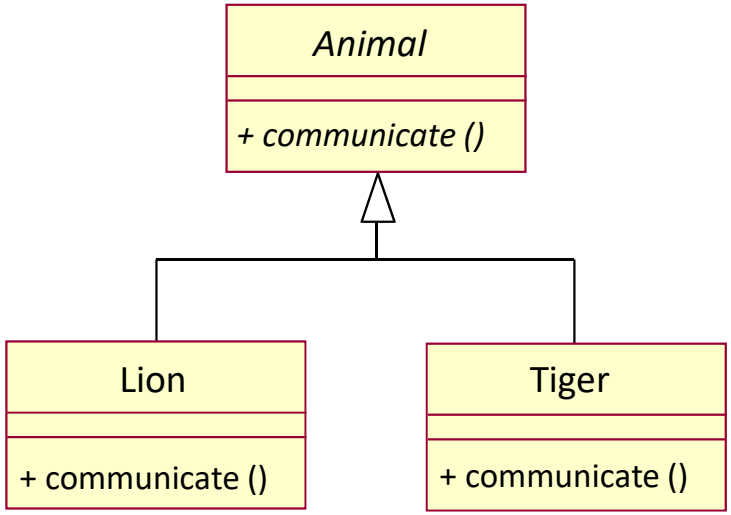
Tổng quát hóa: chia sẻ thuộc tính/hành vi chung

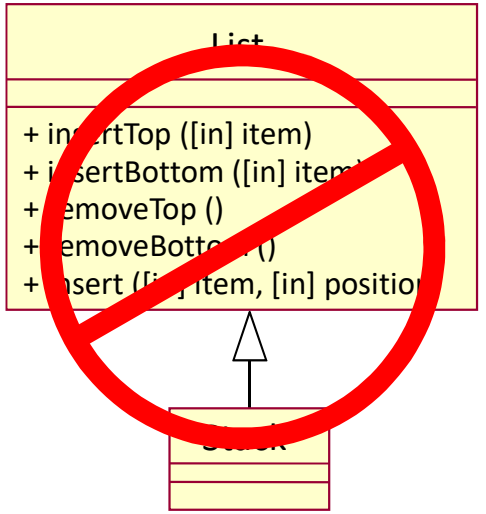
- Tuân thủ dạng chương trình "is a"
- Khả năng thay thế lớp



Do these classes follow the "is a" style of programming?

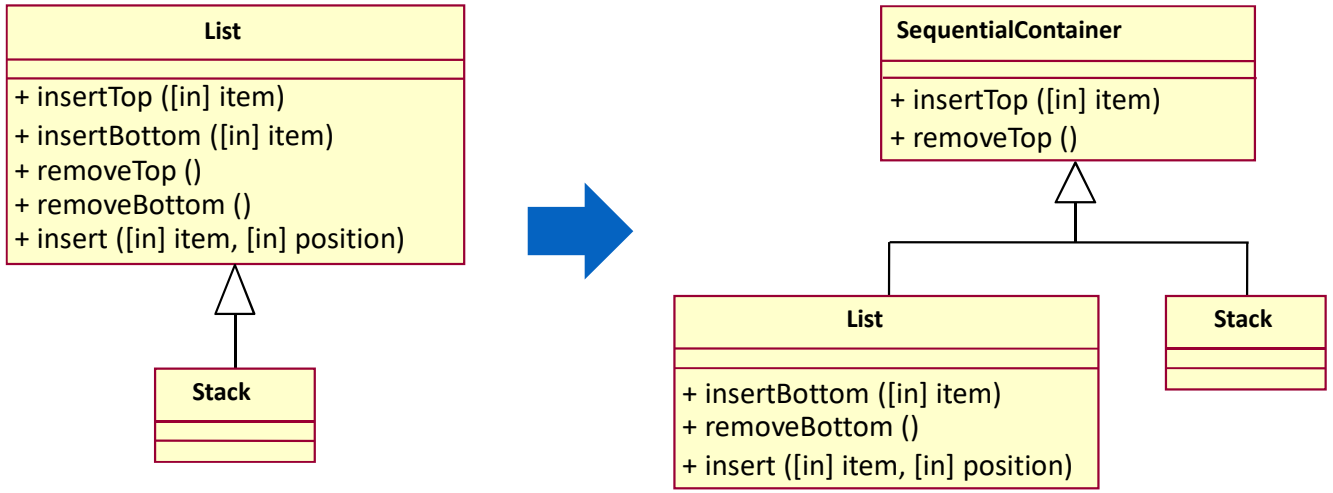
Tổng quát hóa: chia sẻ thuộc tính/hành vi chung (tiếp)





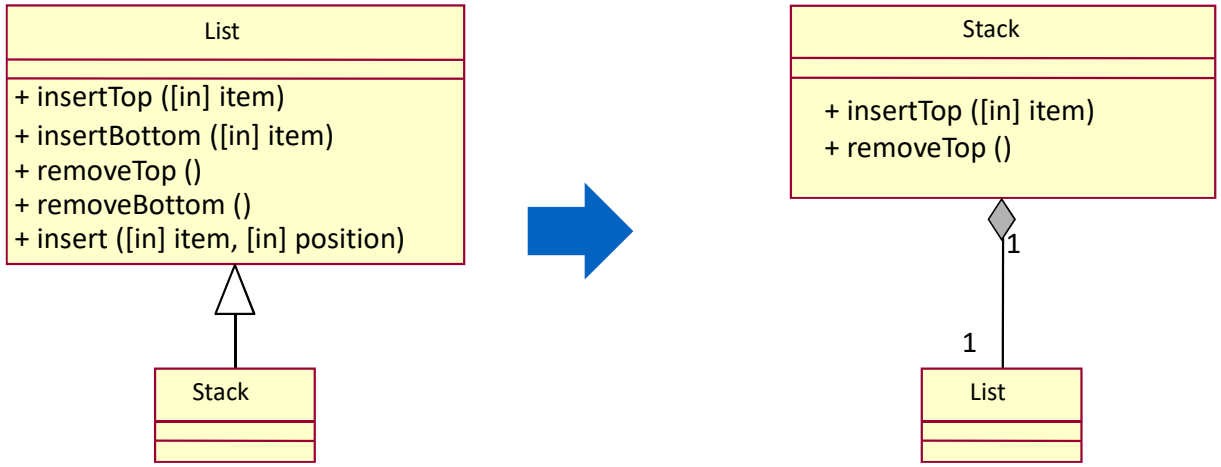
Tổng quát hóa: Chia sẻ thể hiện – phân mảnh

- Hỗ trợ việc tái sử dụng khi thể hiện lớp khác
- Không sử dụng được nếu lớp muốn sử dụng lại không có khả năng thay đổi



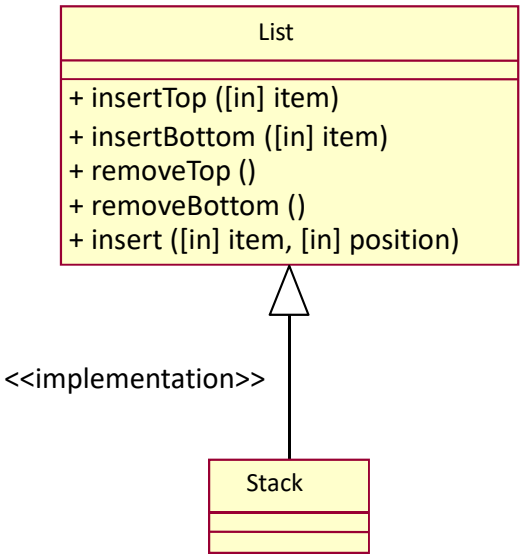
Tổng quát hóa: Chia sẻ thể hiện - ủy quyền

- Hỗ trợ tái sử dụng khi thể hiện lớp khác
- Có thể sử dụng được trong trường hợp lớp muốn sử dụng lại không có khả năng thay đổi



Thể hiện thừa kế

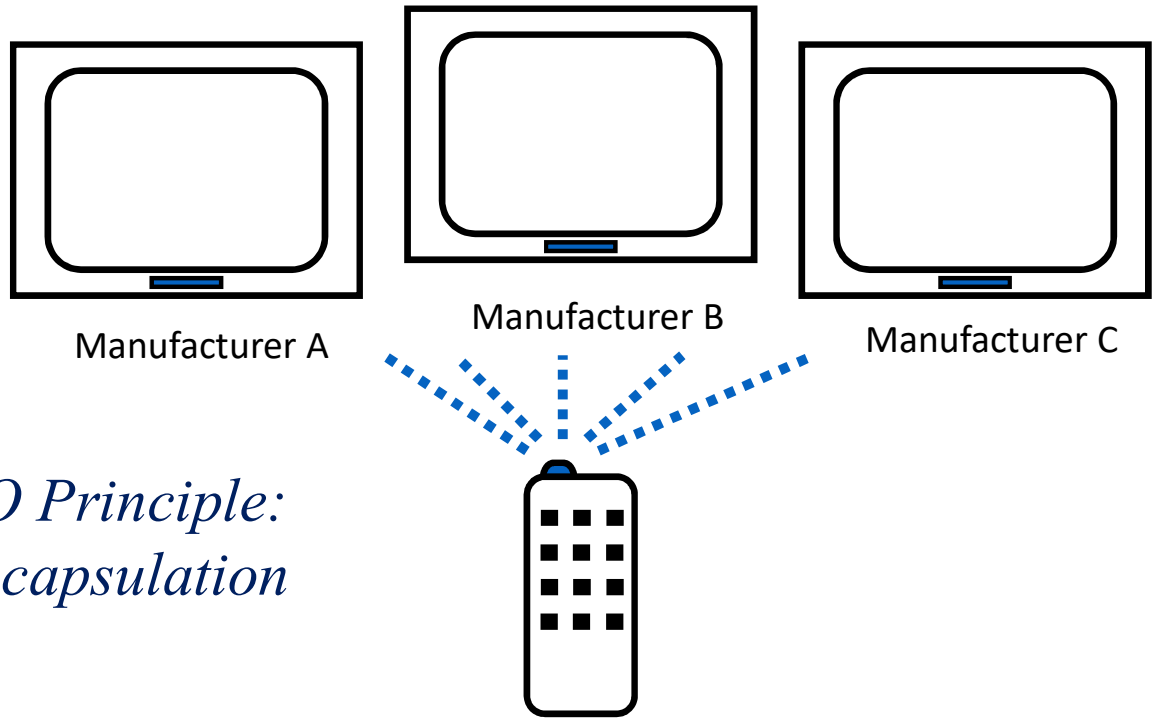
- Các hoạt động, thuộc tính và quan hệ của lớp cha không được nhìn thấy bởi client của lớp dẫn xuất
- Lớp dẫn xuất phải xác định tất cả các truy cập tới lớp cha



push() and pop() can access methods of List but instances of Stack cannot

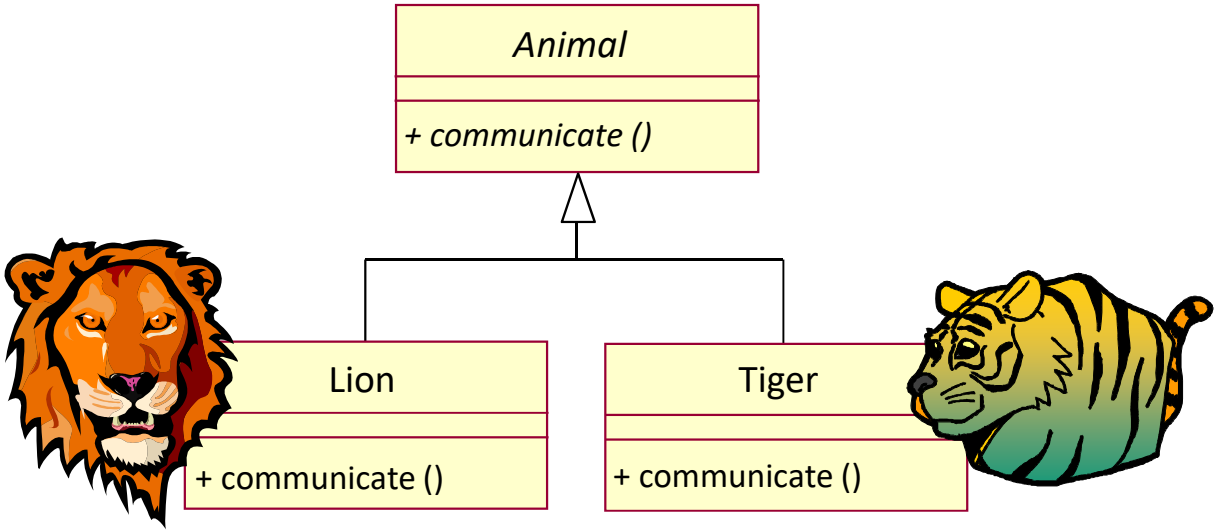
Đa hình

- Khả năng ẩn giấu nhiều thể hiện khác nhau dưới một giao diện



OO Principle: Encapsulation

Tổng quát hóa: Thể hiện đa hình



Without Polymorphism

```
if animal = “Lion” then
        Lion communicate
else if animal = “Tiger” then
        Tiger communicate
end
```

With Polymorphism

```
Animal communicate
```

Đa hình: Sử dụng giao diện hay tổng quát hóa

- Giao diện hỗ trợ thể hiện độc lập tính đa hình
  - Các quan hệ hiện thực hóa có thể liên kết chéo trên cây tổng quát hóa
- Giao diện là các đặc tả thuần túy
  - Lớp cơ sở trừu tượng có thể xác định các hành vi và liên kết
- Giao diện độc lập với thừa kế
  - Tổng quát hóa để sử dụng lại các thể hiện
  - Giao diện để sử dụng lại các đặc tả hành vi
- Tổng quát hóa cung cấp phương thức thể hiện đa hình

Tính đa hình thông qua thiết kế tổng quát hóa

- Cung cấp giao diện chỉ cho lớp con?
  - Thiết kế lớp cha là lớp trừu tượng
  - Tất cả phương thức được thể hiện bởi lớp dẫn xuất
- Cung cấp giao diện và hành vi cho lớp con?
  - Thiết kế lớp cha như lớp cụ thể với các phương thức mặc định
  - Cho phép các hoạt động đa hình
- Cung cấp giao diện và hành vi bắt buộc đối với các lớp dẫn xuất?
  - Thiết kế lớp cha là lớp cụ thể
  - Không cho phép hoạt động đa hình

Biến hình

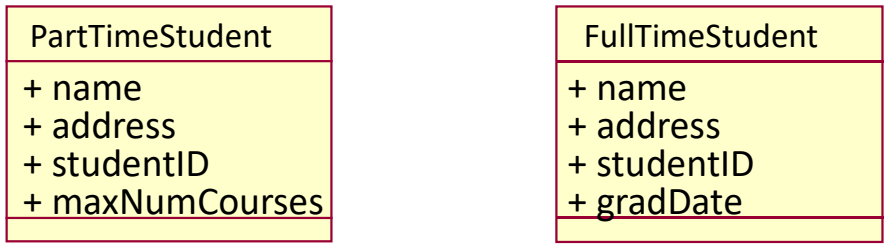
- Biến hình
  - 1. Sự thay đổi hình dạng, cấu trúc hoặc chức năng; thí dụ sự chuyển đổi từ nòng nọc sang ếch.
  - 2. Thay đổi được ghi nhận như tính chất, hình dạng và điều kiện.

~ Webster's New World Dictionary, Simon & Schuster, Inc., 1979

Metamorphosis exists in the real world.
How should it be modeled?

Thí dụ: Biến hình

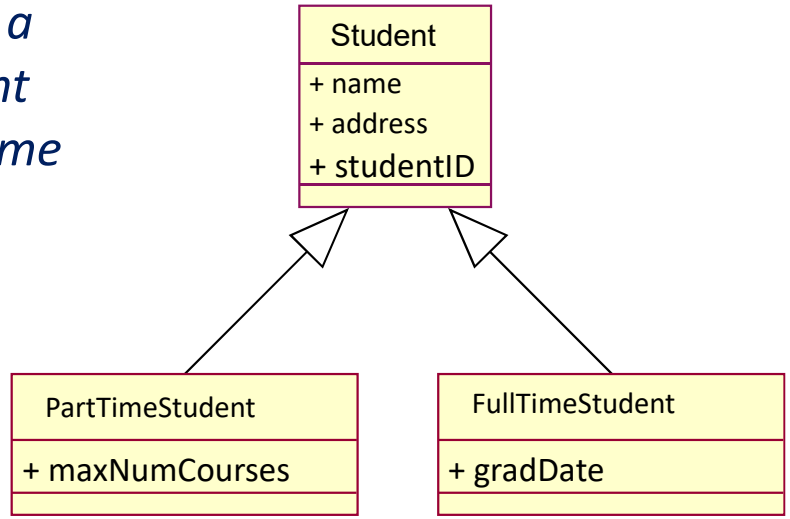
- Trong trường đại học, sinh viên chính quy và sinh viên bán thời gian (vừa học vừa làm)
  - Full-time students have an expected graduation date but part-time students do not
  - Part-time students may take a maximum of three courses but there is no maximum for full-time students



Mô hình hóa biến hình

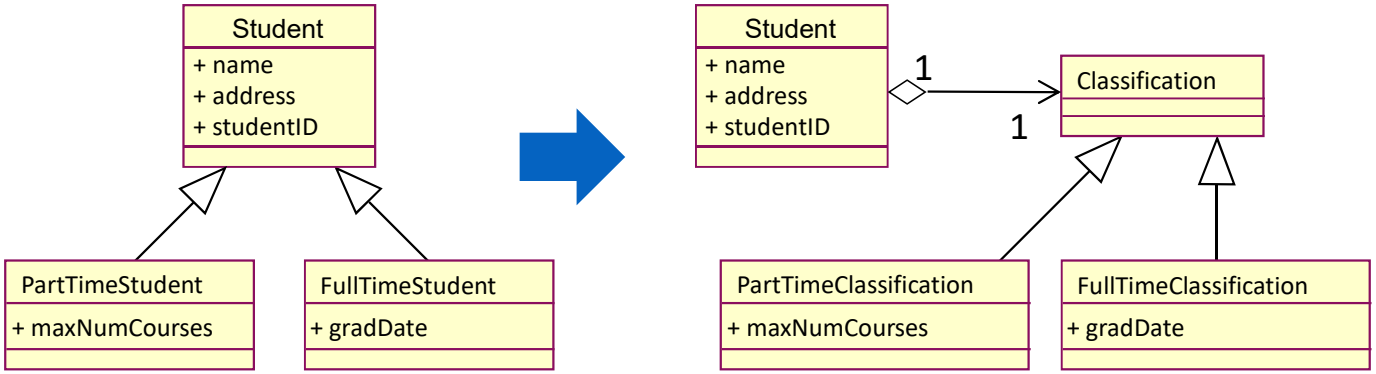
- Có thể tạo một quan hệ tổng quát hóa

What happens if a part-time student becomes a full-time student?



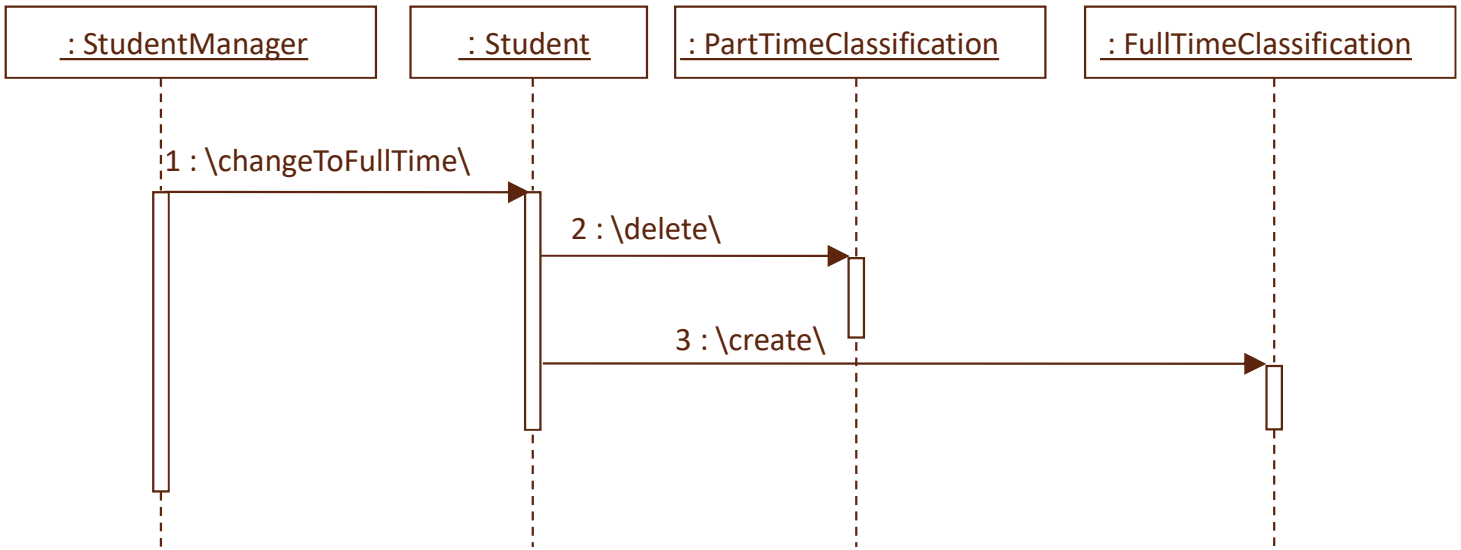
Mô hình hóa biến hình – các tiếp cận khác

- Thừa kế có thể được sử dụng để mô hình cấu trúc, hành vi hay các quan hệ chung của các phần đang thay đổi



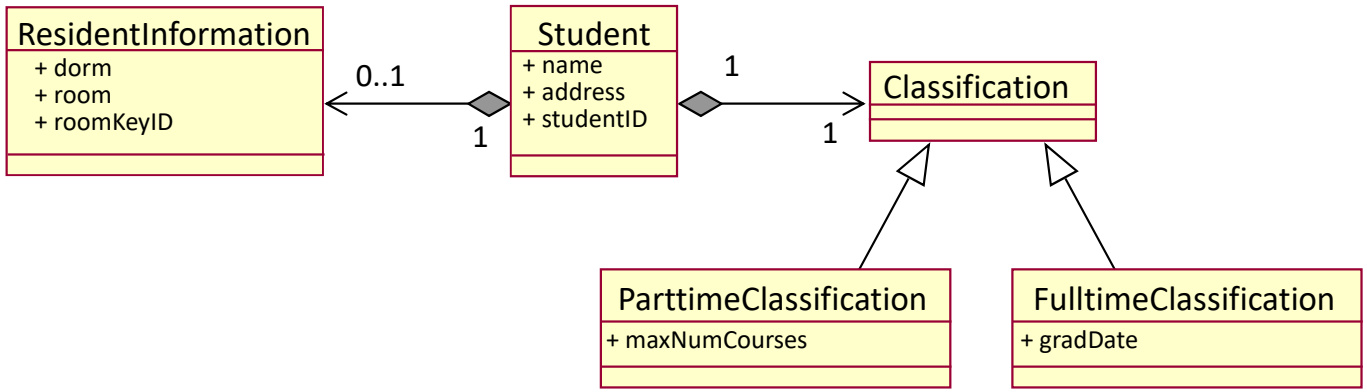
Mô hình hóa biến hình

- Biến hình đạt được bằng cách đối tượng “nói” với các phần đang thay đổi



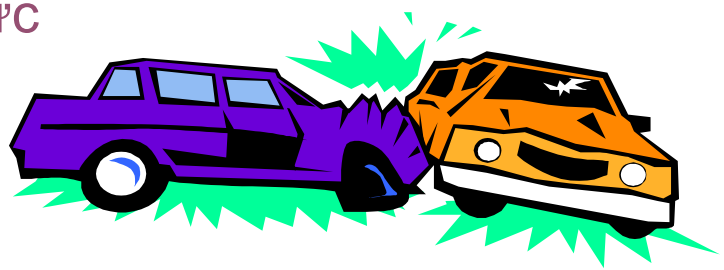
Biến hình và tính mềm dẻo

- Kỹ thuật này bổ sung tính mềm dẻo cho mô hình



Class Design Steps

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định các sự tổng quát hóa
- Giải quyết các xung đột usecase
- Thể hiện các yêu cầu phi chức năng



Xử lý xung đột usecase

- Nhiều usecase có thể đồng thời truy cập tới các đối tượng thiết kế
- Các lựa chọn
  - Sử dụng thông báo đồng bộ => thứ tự xử lý: first-come first-serve
  - Nhận diện các hoạt động (hoặc code) để bảo vệ
  - Áp dụng các cơ chế điều khiển truy cập
    - Hàng đợi thông báo
    - Token
    - Cơ chế khóa khác
- Phụ thuộc vào môi trường thể hiện

Class Design Steps

- Tạo các lớp thiết kế ban đầu
- Xác định các hoạt động
- Xác định các phương thức
- Xác định các trạng thái
- Xác định các thuộc tính
- Xác định các phụ thuộc
- Xác định các liên kết
- Xác định các sự tổng quát hóa
- Giải quyết các xung đột usecase
- Thể hiện các yêu cầu phi chức năng

Giải quyết các yêu cầu phi chức năng

Analysis Class	Analysis Mechanism(s)
Student	Persistency, Security
Schedule	Persistency, Security
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	Distribution