ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA HỆ THỐNG THÔNG TIN

CHƯƠNG 5
BIỂU ĐỒ LỚP
(CLASS DIAGRAM)

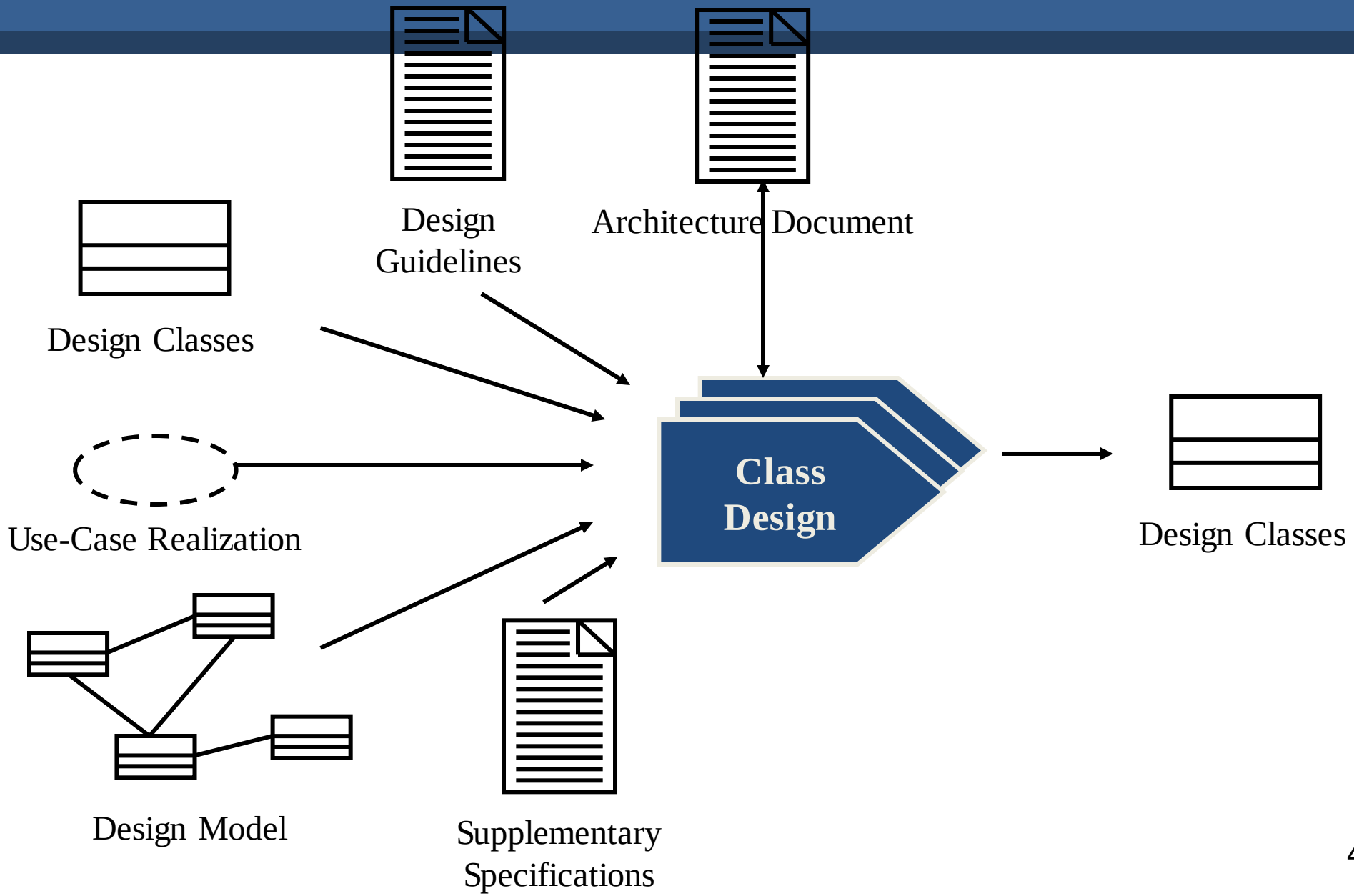
Mục tiêu

- Xác định, mô tả các lớp của hệ thống và mối quan hệ giữa chúng
- Phân tích chi tiết, rõ ràng yêu cầu của hệ thống.
- Căn cứ để thiết kế CSDL.

Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

1. Tổng quan về Class



2. Ký hiệu


Name
Attribute(s)
Operation(s)
Bình thường: Class bình thường
In nghiêng: Class thuần ảo
Gạch dưới: Object (không phải class)
Bình thường: Thuộc tính bình thường
In nghiêng: không sử dụng
Gạch dưới: Thuộc tính static
Bình thường: Phương thức bình thường
In nghiêng: Phương thức virtual
Gạch dưới: Phương thức static

Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mỗi kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

Cần bao nhiêu Class ?

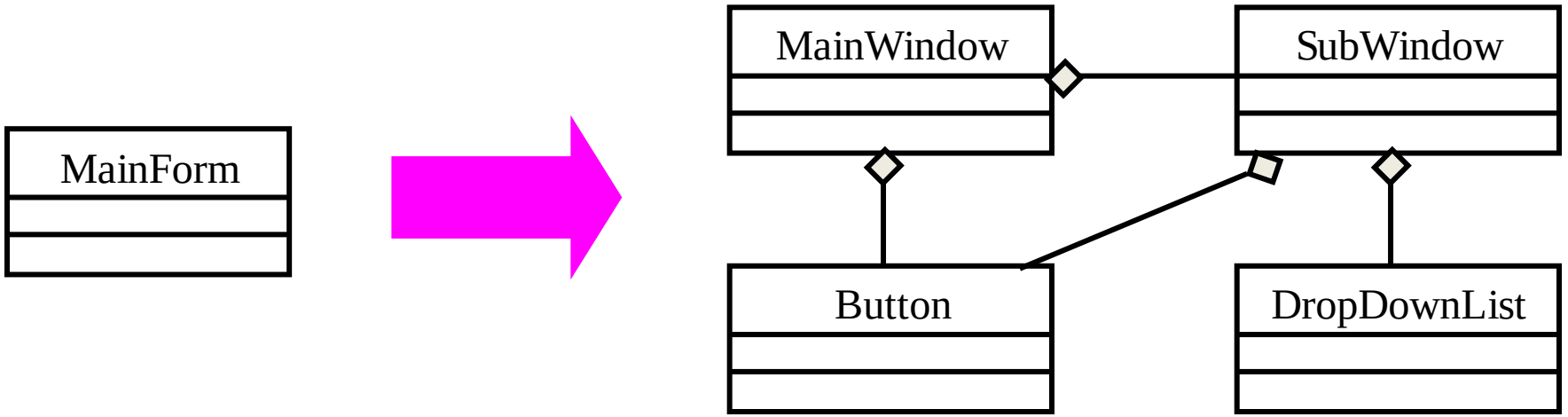
- Nếu nhiều class đơn giản. Nghĩa là mỗi class:
  - Đóng gói một phần ít hơn trên toàn bộ hệ thống
  - Nhiều khả năng dùng lại hơn
  - Dễ cài đặt hơn
- Nếu nhiều class phức tạp. Nghĩa là mỗi class:
  - Đóng gói một phần nhiều hơn trên toàn bộ hệ thống
  - Ít khả năng dùng lại hơn
  - Khó cài đặt hơn

Một class phải có một mục tiêu rõ ràng.

Một class phải làm một việc gì đó và phải làm tốt điều này !

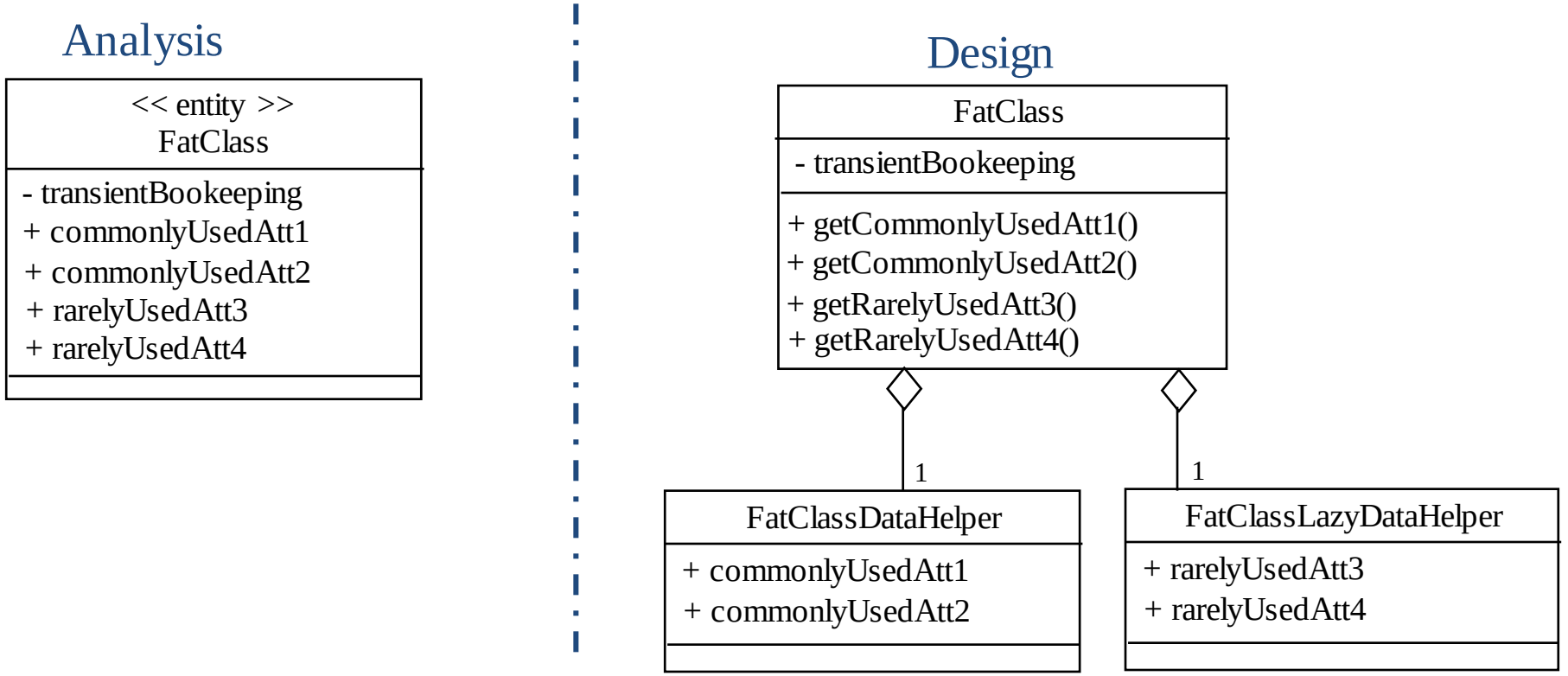
Thiết kế các Boundary Class

- Các User interface (UI) boundary class
  - Công cụ xây dựng giao diện người dùng nào sẽ được sử dụng?
  - Bao nhiêu giao diện có thể được xây dựng bởi công cụ?
- Các External system interface boundary class
  - Thường được mô hình như subsystem



Thiết kế các Entity Class

- Các Entity object thường thụ động và persistent
- Các yêu cầu về hiệu năng có thể buộc ta phải tái xây dựng
- Xem thêm bước xác định Persistent Class



Thiết kế Control Class

- Chuyện gì xảy ra với các Control Class?
  - Chúng thật sự cần thiết?
  - Có phải tách chúng ra không?

- Dựa vào đâu để quyết định?
  - Độ phức tạp
  - Khả năng thay đổi
  - Tính phân tán và hiệu năng
  - Transaction management

Nội dung

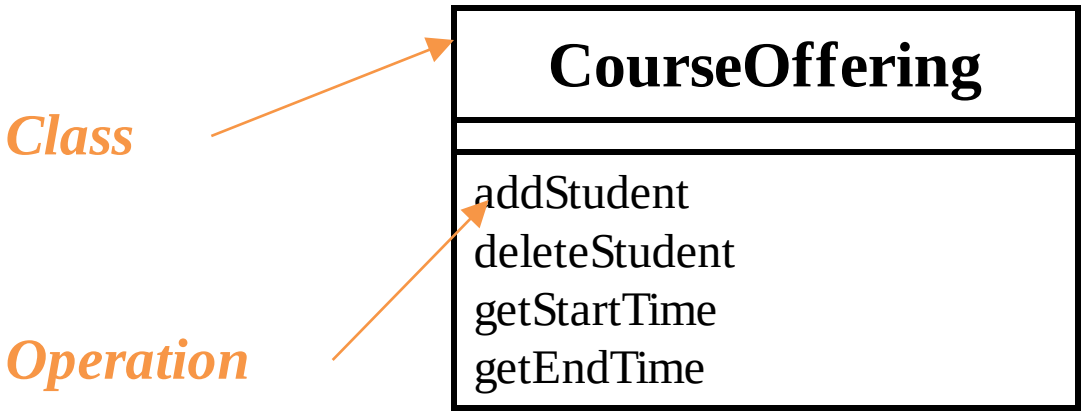
1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

4. Định nghĩa các Operation

- Mục đích
  - Ánh xạ các nhiệm vụ đã xác định ở mức phân tích thành các operation thực hiện chúng

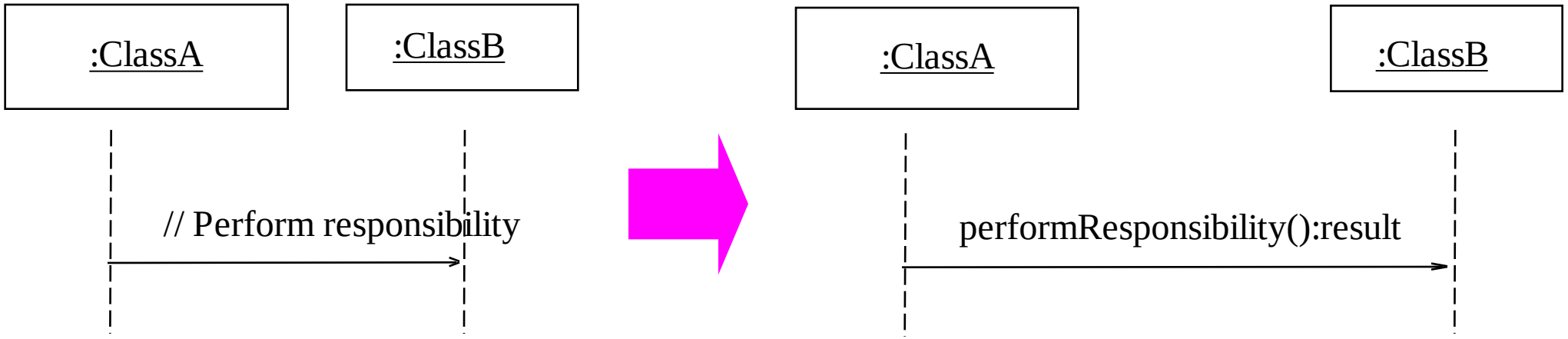
- Những cái cần xem xét:
  - Tên Operation, signature, và mô tả
  - Operation visibility
  - Tầm vực Operation
    - Class operation hay instance operation

Nhắc lại: Operation là gì ?



Operation: Tìm chúng ở đâu?

- Các thông điệp trong các interaction diagram



- Các chức năng phụ thuộc vào cài đặt khác
  - Các chức năng quản trị
  - Các nhu cầu sao chép class
  - Các nhu cầu kiểm tra bằng, khác nhau, …

Đặt tên và mô tả các Operation

- Các tên thích hợp cho operation
  - Chỉ rõ kết quả của operation
  - Đứng dưới góc nhìn của client
  - Nhất quán qua tất cả các class

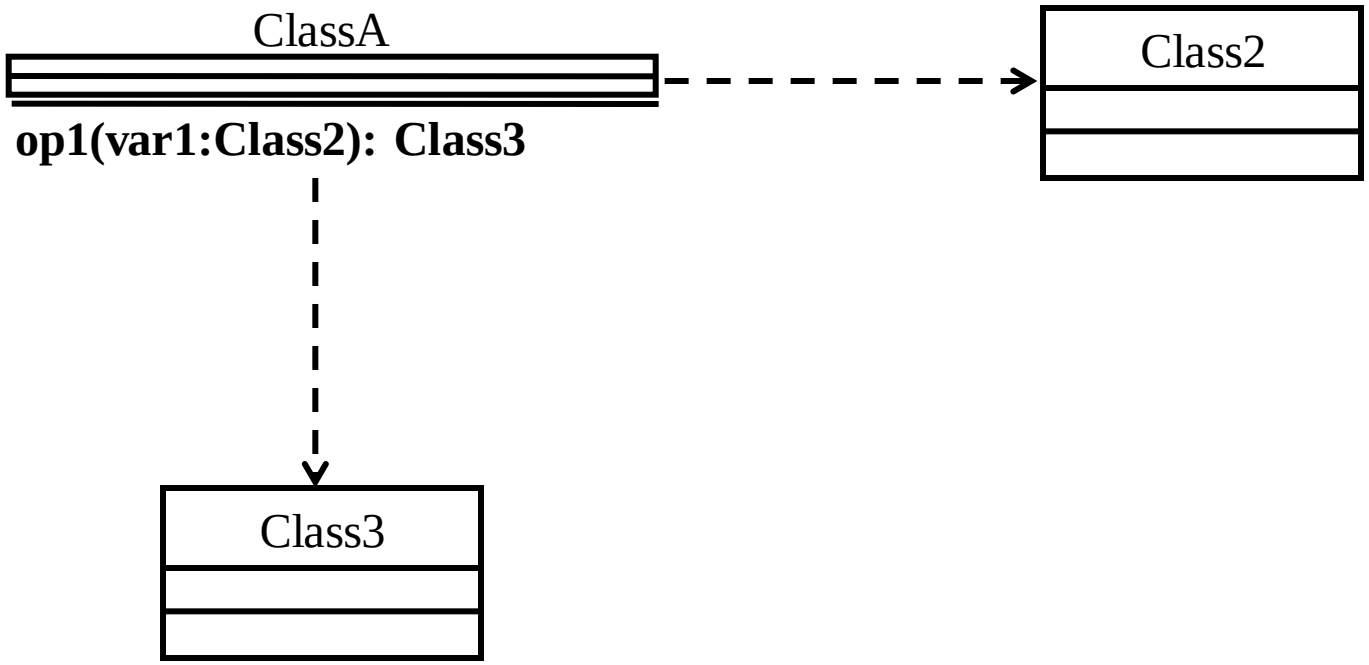
- Định nghĩa operation signature
  - operationName(parameter : class,..) : returnType

- Cung cấp một mô tả ngắn, bao gồm ý nghĩa của tất cả các tham số

Thiết kế Operation Signatures

- Khi thiết kế operation signatures phải bảo đảm hàm chứa:
  - Các tham số truyền theo giá trị hay tham số?
  - Các tham số có bị thay đổi bởi operation?
  - Các tham số là optional?
  - Tham số có giá trị mặc định?
  - Miền tham số hợp lệ?
- Càng ít tham số càng tốt
- Truyền các object thay vì “data bits”

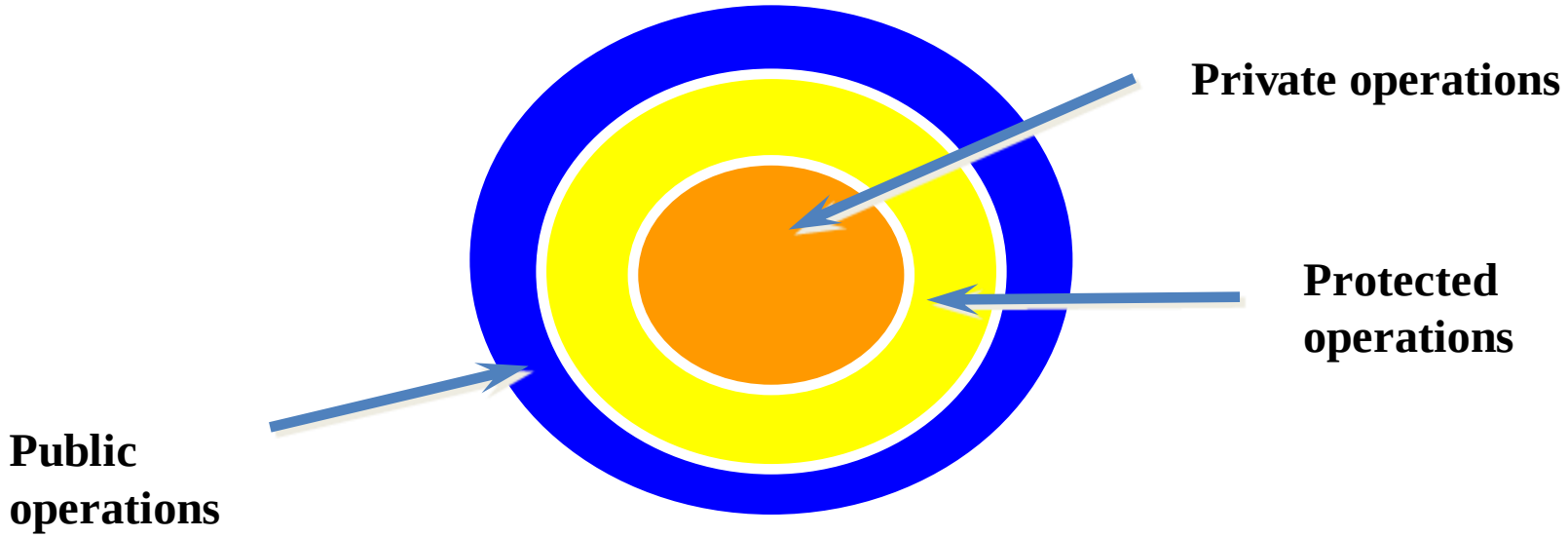
Phát hiện Additional Classes và Relationships



Additional classes và relationships có thể được thêm vào để hỗ trợ signature

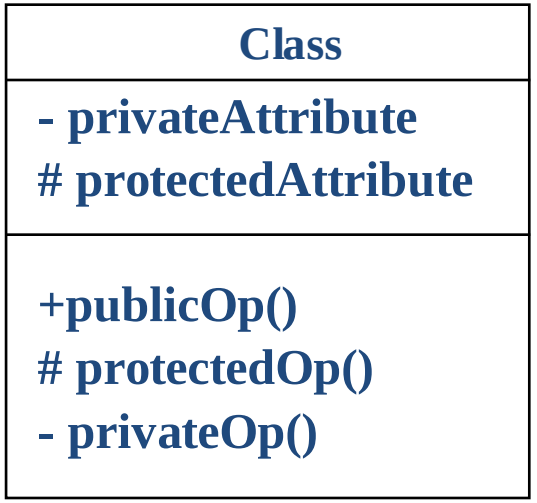
Operation Visibility

- Tính khả kiến được dùng để cung cấp tính đóng gói
- Giá trị có thể là public, protected, hay private



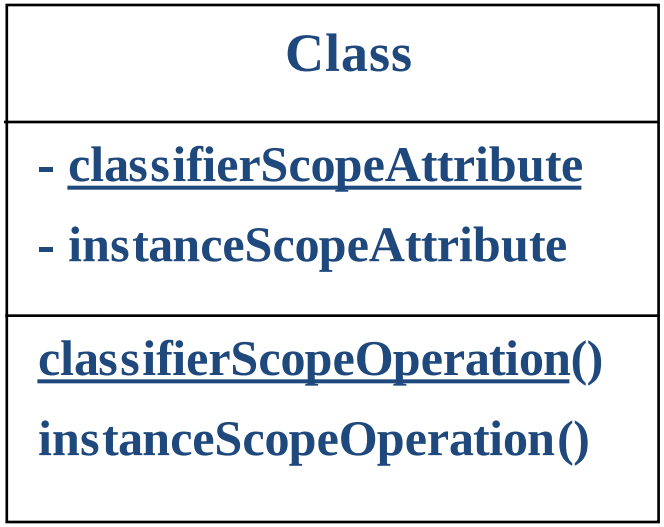
Ký hiệu tính khả kiến?

- Các ký hiệu sau được dùng:
  - \+ Public access
  - # Protected access
  - \- Private access

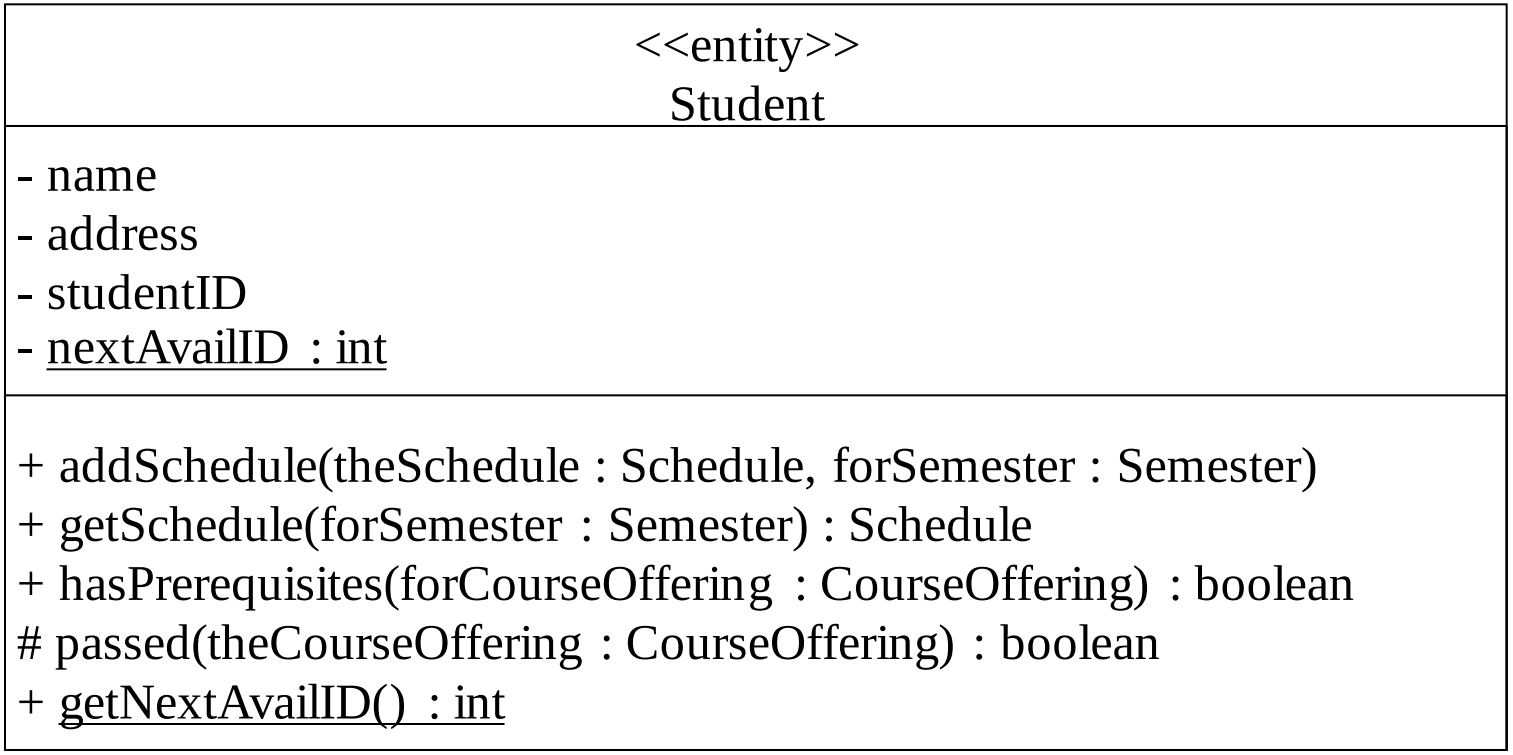


Tầm vực

- Xác định số lượng thể hiện của attribute / operation
  - Instance: 1 instance cho mỗi class instance
  - Classifier: 1 instance cho tất cả class instance
- Tầm vực mức Classifier được ký hiệu bằng cách gạch dưới tên attribute/operation

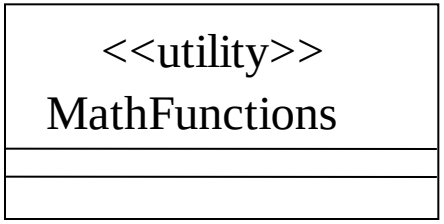


Ví dụ: Scope

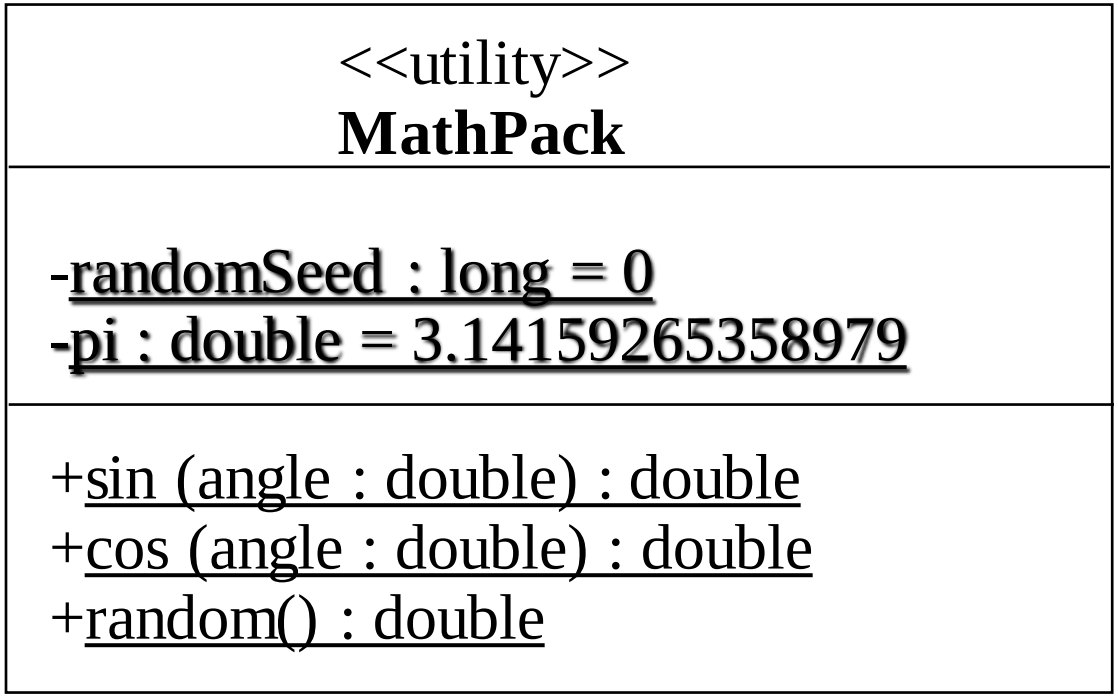

<<entity>>
Student
- name
- address
- studentID
- nextAvailID : int
+ addSchedule(theSchedule : Schedule, forSemester : Semester)
+ getSchedule(forSemester : Semester) : Schedule
+ hasPrerequisites(forCourseOffering : CourseOffering) : boolean
# passed(theCourseOffering : CourseOffering) : boolean
+ getNextAvailID() : int

Utility Classes

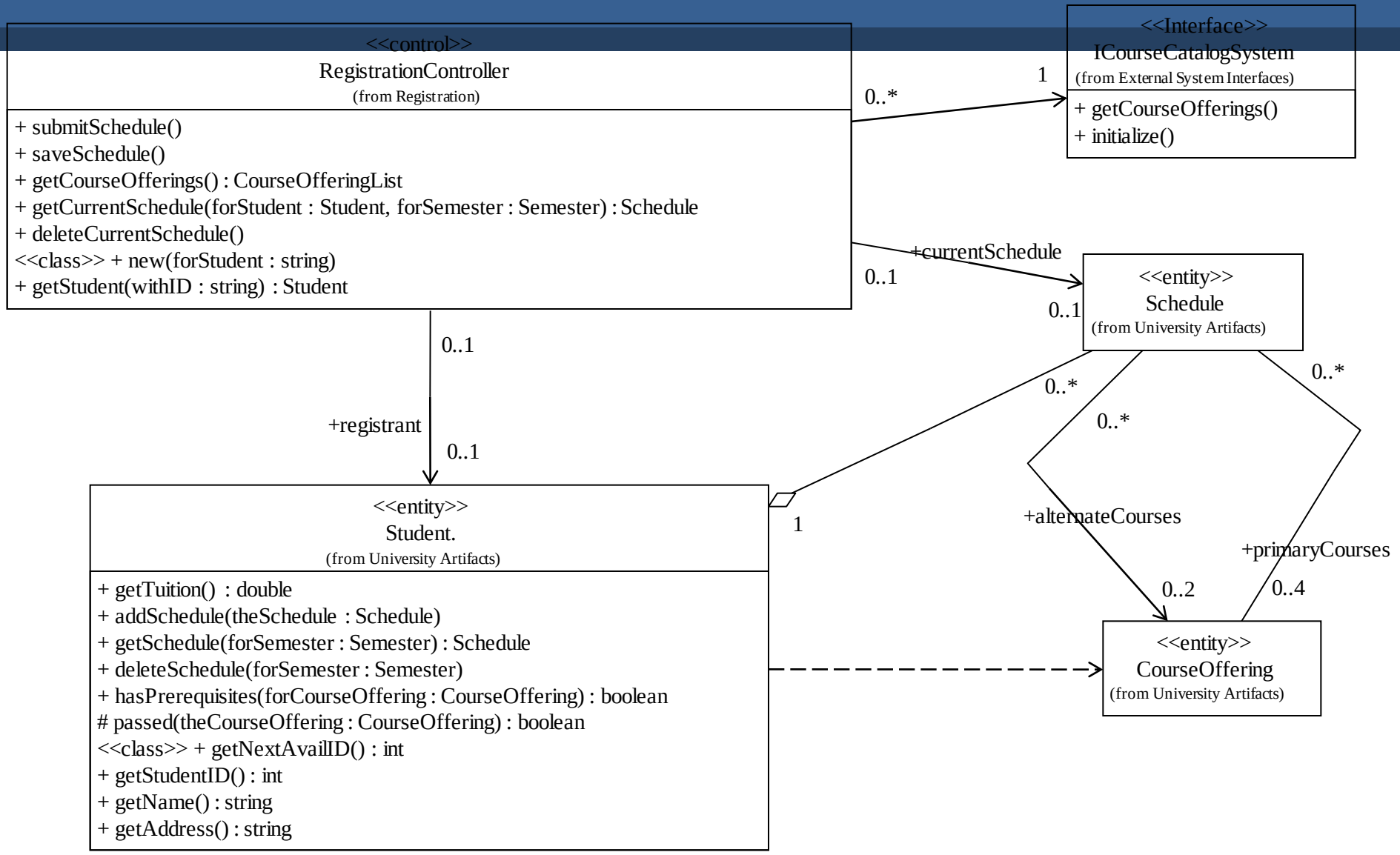
- Thế nào là một Utility Class?
  - Utility là một class stereotype
  - Dùng để chỉ các class chứa một bộ các chương trình con miễn phí
- Tại sao lại dùng chúng?
  - Để cung cấp các dịch vụ có thể hữu dụng trong các ngữ cảnh khác nhau
  - Để gói các hàm thư viện hay các ứng dụng phi đối tượng



Ví dụ: Utility Classes



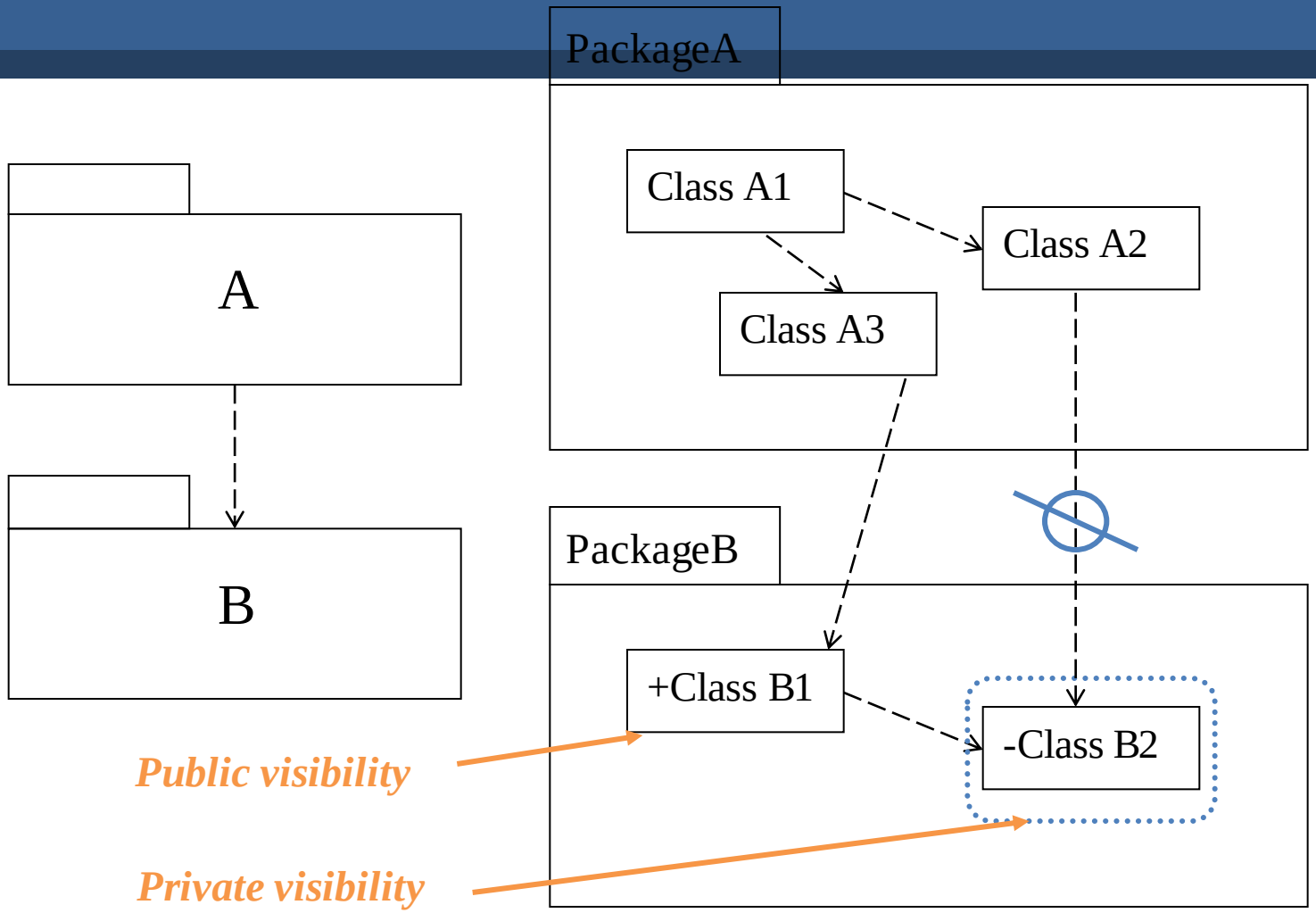
Ví dụ: Định nghĩa các Operation



Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

Nhắc lại: Package Element Visibility



Chỉ có thể tham chiếu tới các public class từ bên ngoài package chứa nó

OO Principle: Encapsulation

Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

6. Định nghĩa các Method

- Method là gì ?
  - Mô tả cài đặt của operation
- Mục đích
  - Định nghĩa các khía cạnh đặc biệt của operation implementation
- Những gì cần xem xét:
  - Các thuật toán đặc biệt
  - Các object và các operation khác cần sử dụng
  - Cách cài đặt và sử dụng các attribute và các tham số
  - Cách cài đặt và sử dụng các mối quan hệ

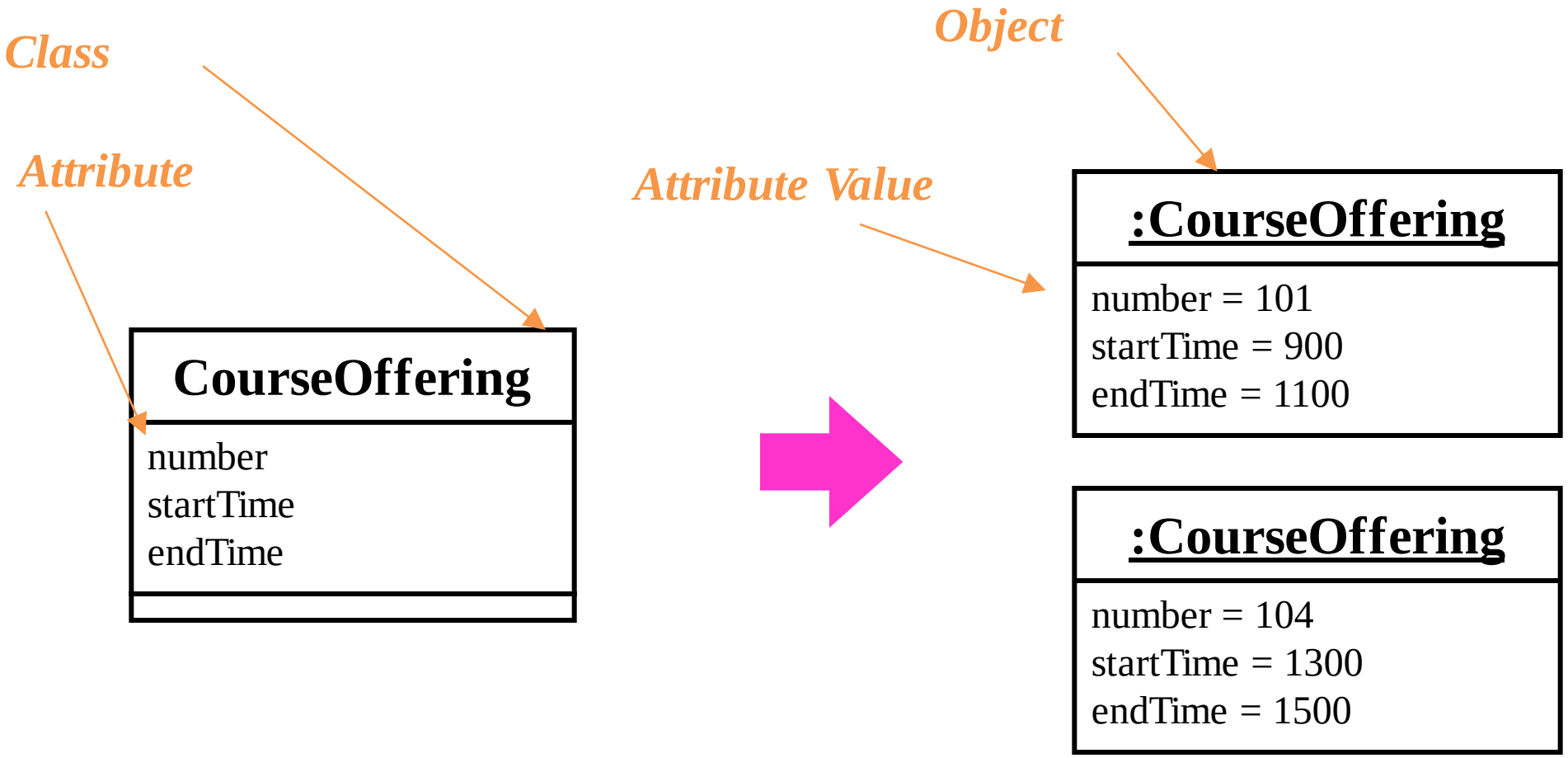
Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

8. Định nghĩa thuộc tính

- Mục đích
  - Chính thức định nghĩa các thuộc tính
- Những gì cần xem xét:
  - Persistency
  - Visibility
  - Tên gọi, kiểu, và giá trị mặc định

Nhắc lại: Thế nào là Attribute?



Cách tìm ra các Attribute?

- Khảo sát mô tả của các method
- Khảo sát các trạng thái
- Bất kỳ thông tin nào mà class cần duy trì

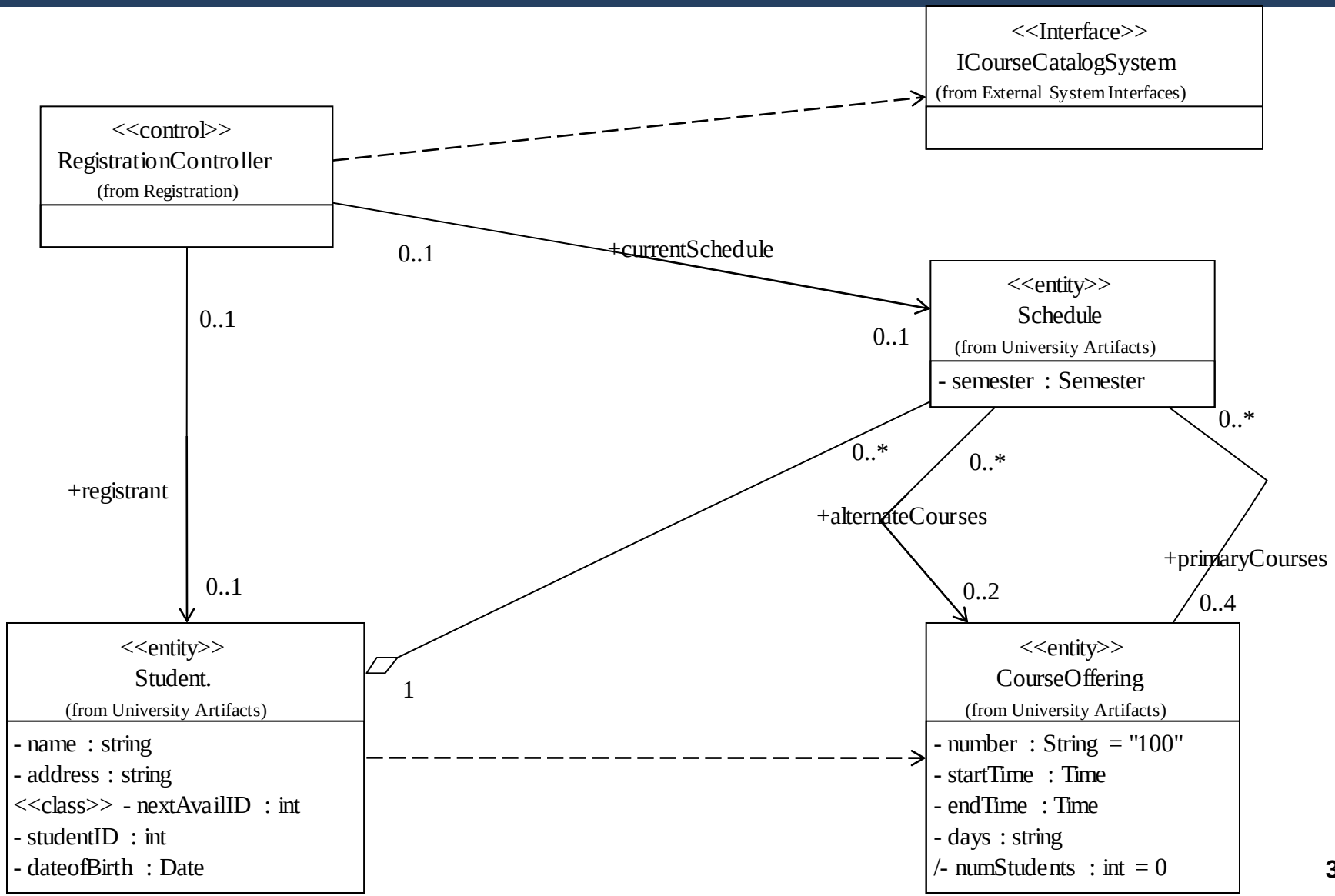
Biểu diễn Attribute

- Mô tả name, type, và giá trị mặc định
  - attributeName : Type = Default
- Tuân thủ qui ước đặt tên của NNLT và dự án
- Type phải là KDL cơ sở trong NNLT cài đặt
  - Các KDL định sẵn, người dùng đ/n
- Mô tả tính khả kiến
  - Public: ‘+’
  - Private: ‘-’
  - Protected: ‘#’

Các thuộc tính dẫn xuất

- Thế nào là thuộc tính dẫn xuất?
  - Một attribute mà giá trị có thể tính từ giá trị của các attribute khác
- Khi nào dùng chúng?
  - Khi không đủ thời gian để tính lại giá trị mỗi khi cần thiết
  - Dung hòa giữa thời gian thực hiện và bộ nhớ sử dụng

Ví dụ: Định nghĩa Attributes

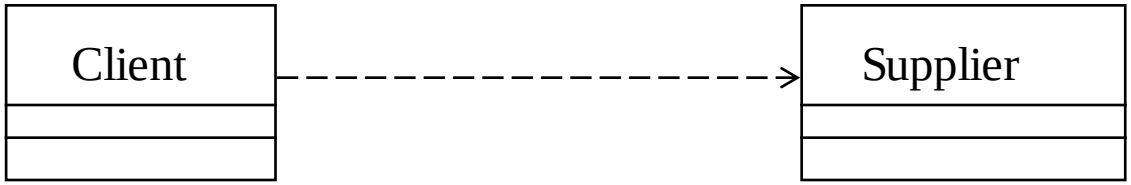


Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

8. Định nghĩa các phụ thuộc

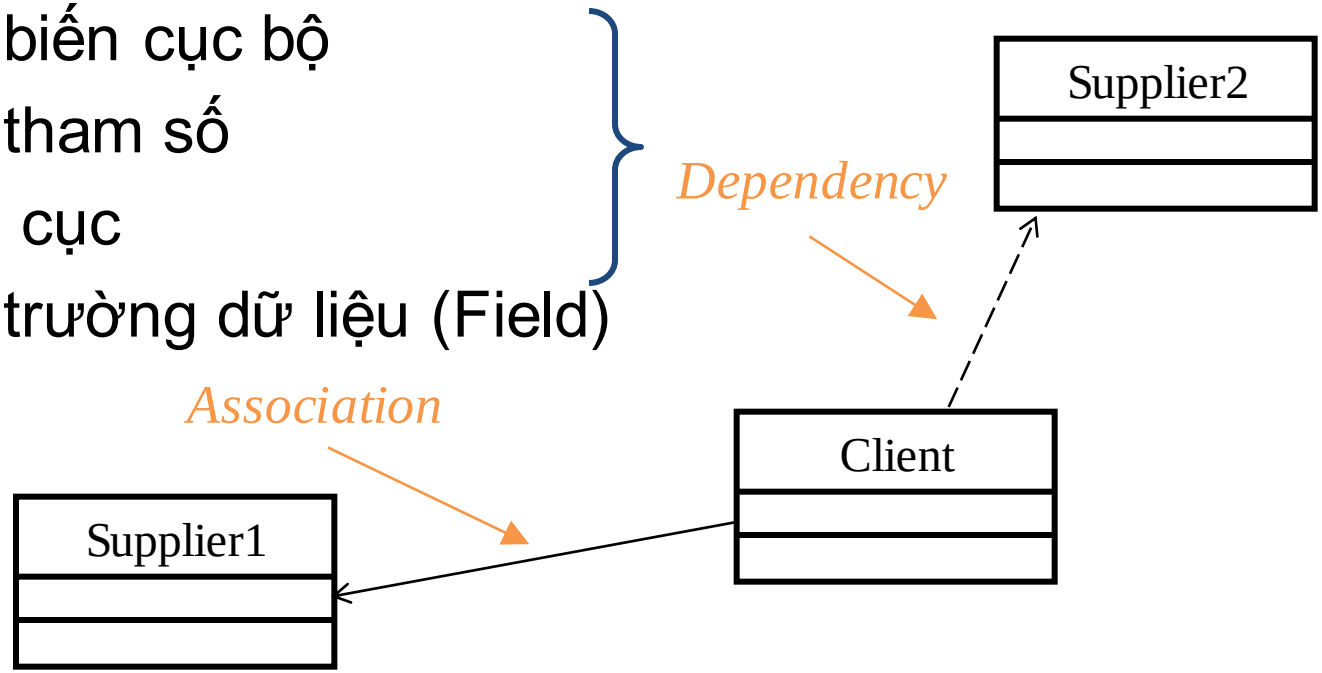
- Dependency là gì?
  - Là một loại quan hệ giữa hai object



- Mục đích
  - Xác định những nơi KHÔNG cần đến các mối quan hệ cấu trúc
- Những gì cần xem xét :
  - Những gì buộc supplier trở nên nhìn thấy được bởi client

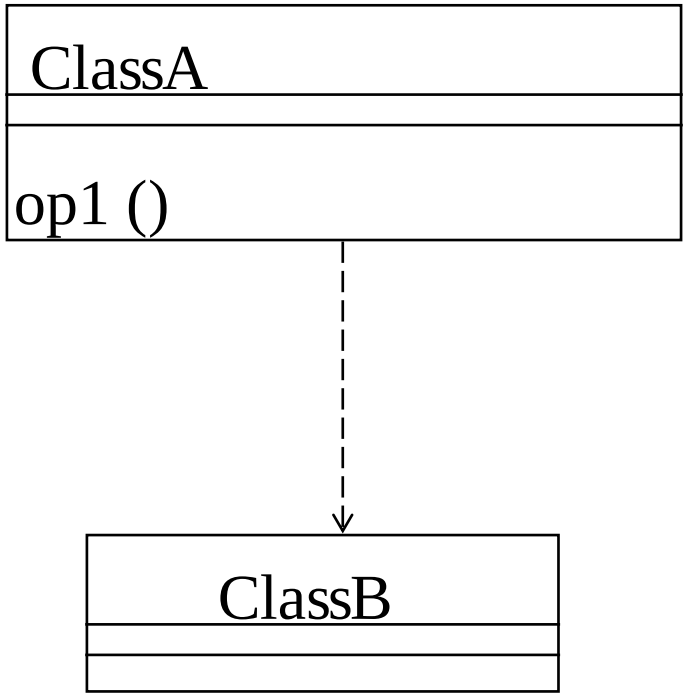
So sánh Dependency và Association

- Association là quan hệ cấu trúc
- Dependency là quan hệ phi cấu trúc
- Để "nói chuyện" được, object phải khả kiến
  - Tham chiếu đến biến cục bộ
  - Tham chiếu đến tham số
  - Tham chiếu toàn cục
  - Tham chiếu đến trường dữ liệu (Field)



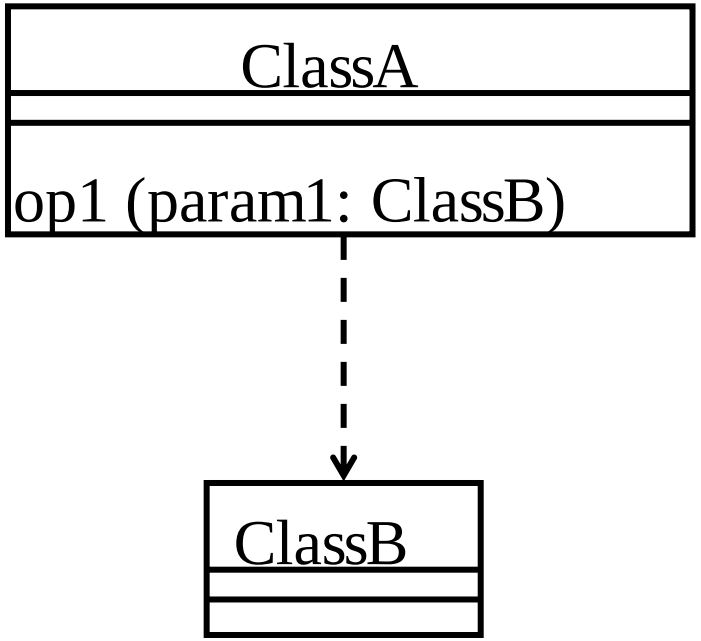
Tham chiếu đến biến cục bộ

- Operation op1() chứa một biến cục bộ có kiểu ClassB



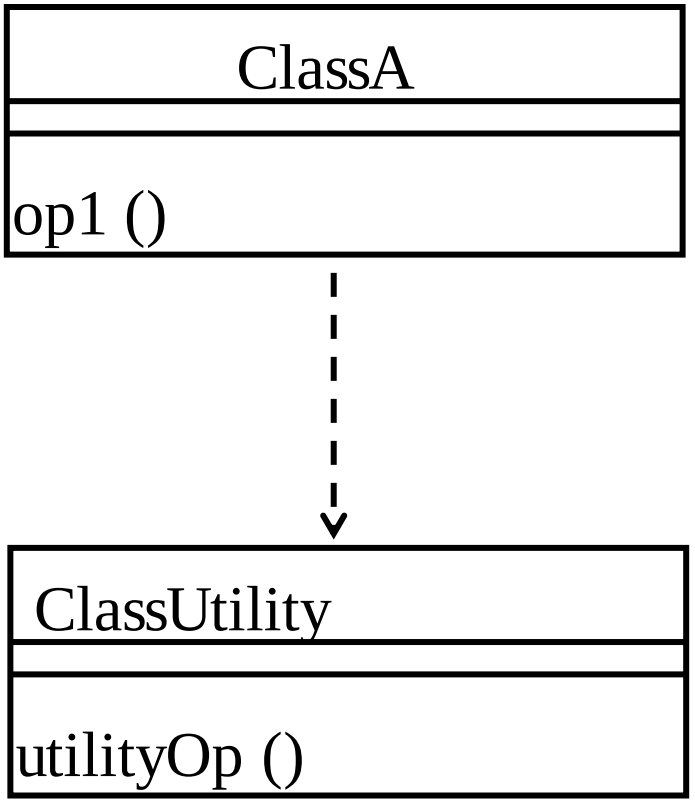
Tham chiếu đến tham số

- Thể hiện của ClassB được truyền đến cho thể hiện của ClassA



Tham chiếu toàn cục

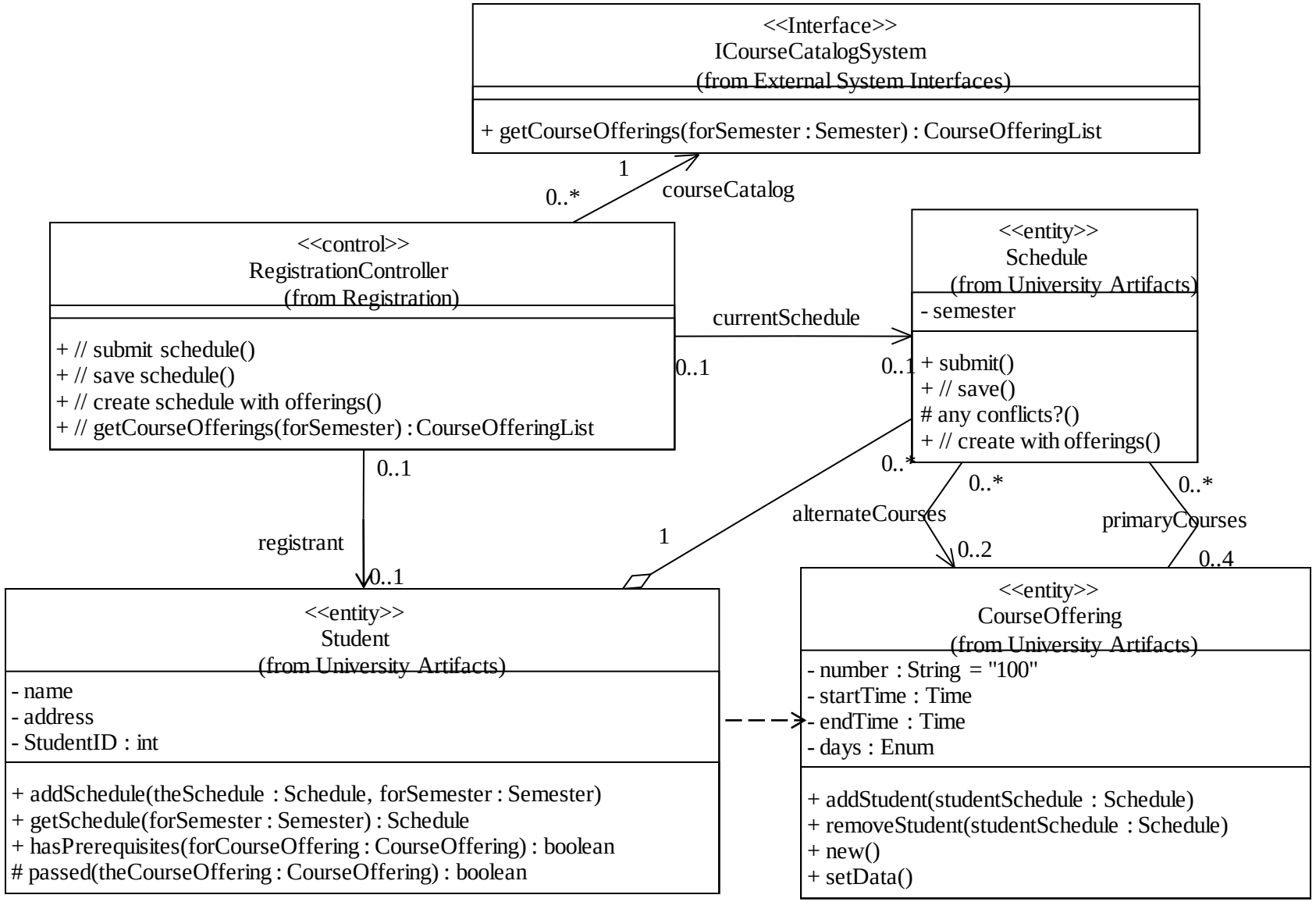
- Thể hiện của ClassUtility khả kiến với mọi đối tượng vì nó là toàn cục (global)



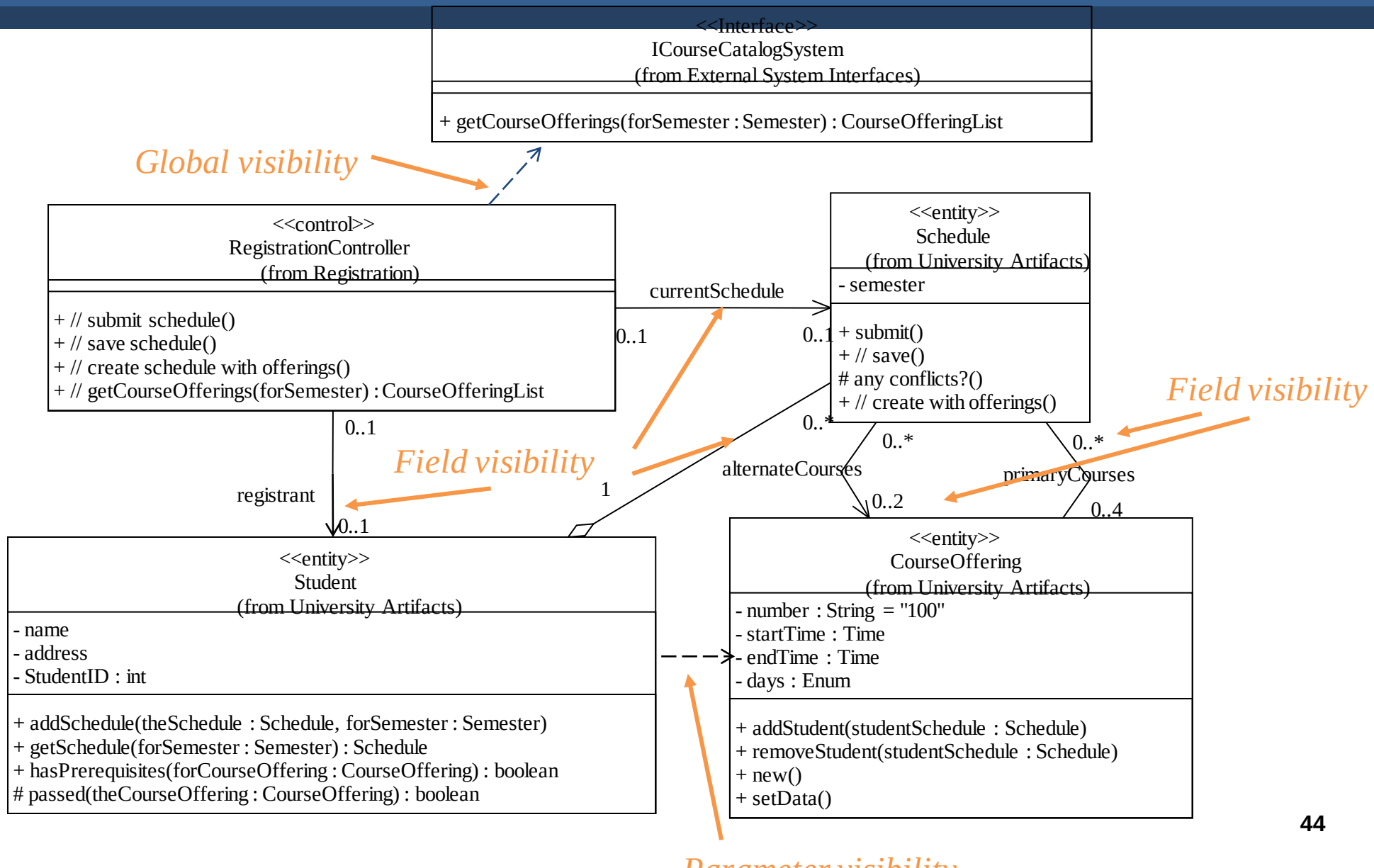
9. Xác định Dependency

- Bám vào các quan hệ trong thế giới thực
- Bám vào các quan hệ yếu nhất (dependency)
- Quan hệ là "thường trực"? Dùng association (field visibility)
- Quan hệ là "nhất thời"? Dùng dependency
  - Nhiều object chia sẻ chung 1 instance
    - Truyền instance như tham số (parameter visibility)
    - Đặt instance là global (global visibility)
  - Các object không chia sẻ cùng 1 instance (local visibility)
- Cần bao nhiêu thời gian để create/destroy?
  - Chi phí ? Dùng field, parameter, hoặc global visibility

Ví dụ: Trước khi định nghĩa Dependency



Ví dụ: Sau khi định nghĩa Dependency



Nội dung

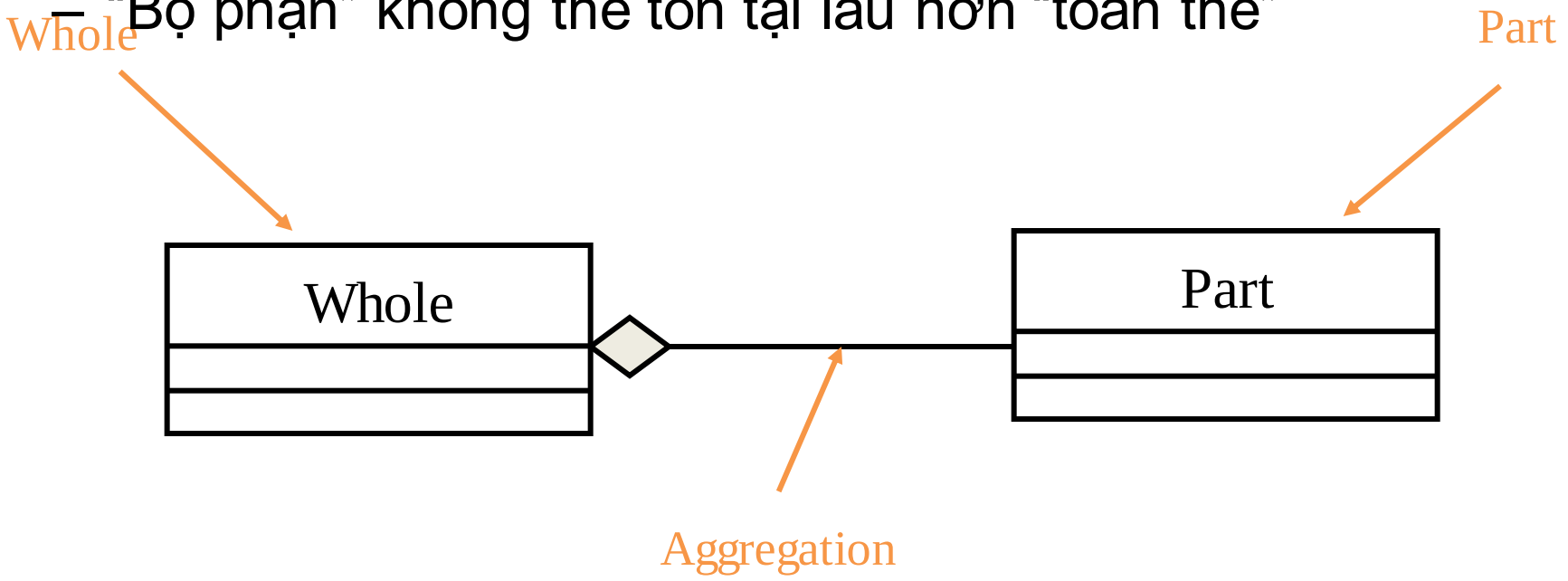
1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

10. Định nghĩa mối kết hợp

- Mục đích
  - Tinh chỉnh các association còn lại
- Những gì cần xem xét :
  - Cân nhắc giữa Association và Aggregation
  - Cân nhắc giữa Aggregation và Composition
  - Cân nhắc giữa Attribute và Association
  - Chiều của quan hệ (Navigability)
  - Thiết kế Association class
  - Thiết kế bản số (Multiplicity design)

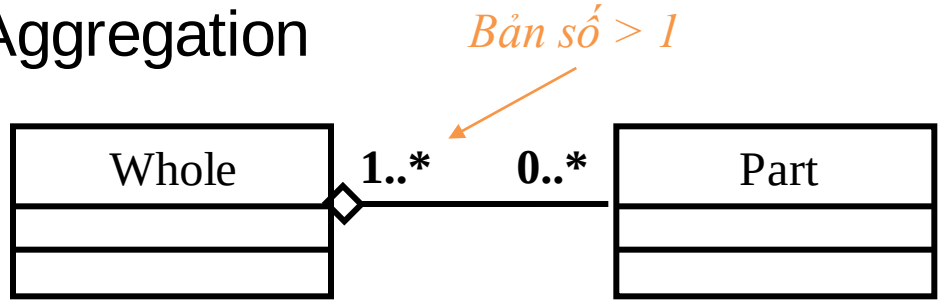
Nhắc lại: Composition là gì ?

- Là một dạng của aggregation với tính sở hữu cao và trùng khớp về chu kỳ sống
  - "Bộ phận" không thể tồn tại lâu hơn "toàn thể"

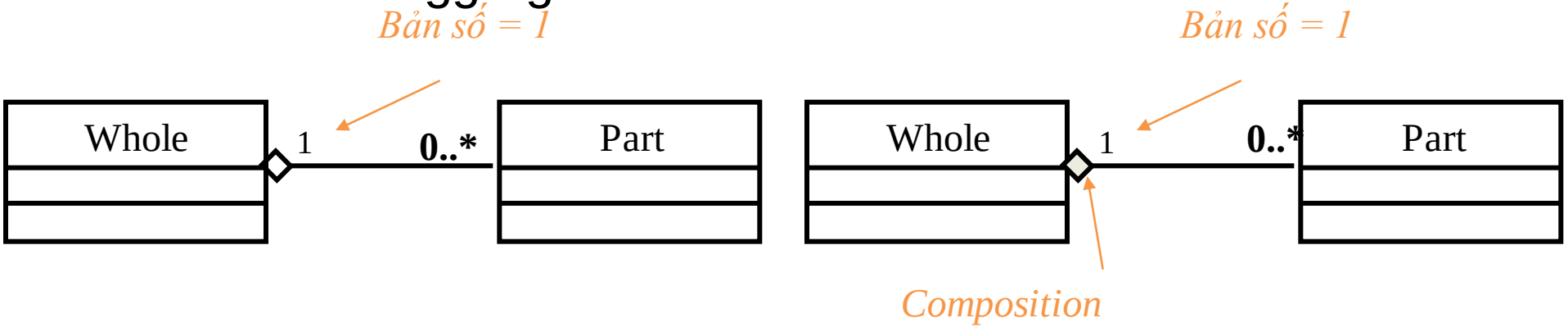


Aggregation: Shared hay không shared

- Shared Aggregation



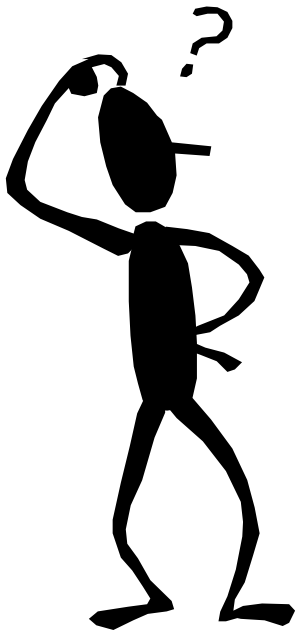
- Non-shared Aggregation

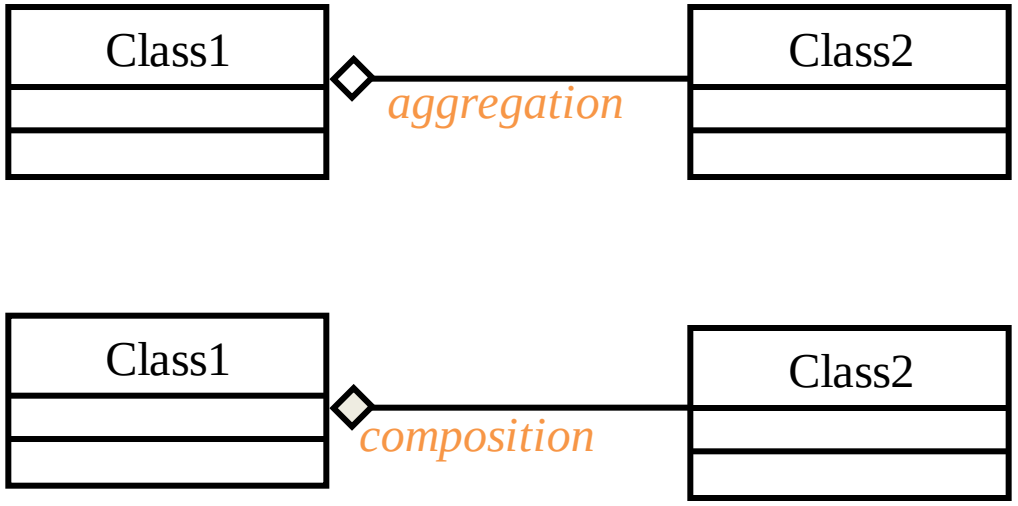


Theo định nghĩa, composition là non-shared aggregation

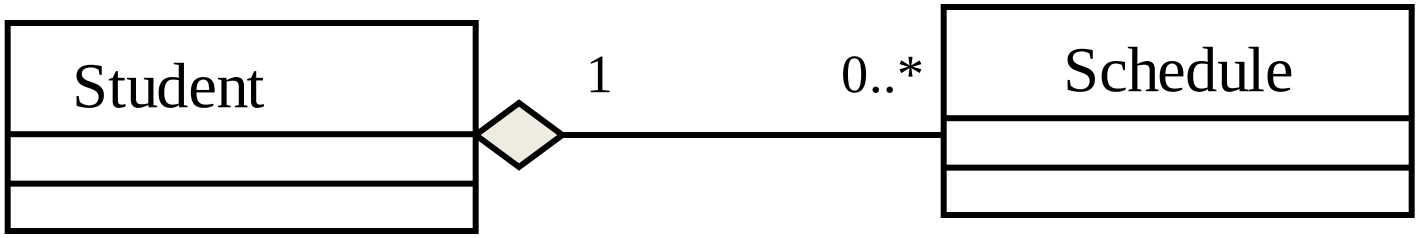
Aggregation hay Composition?

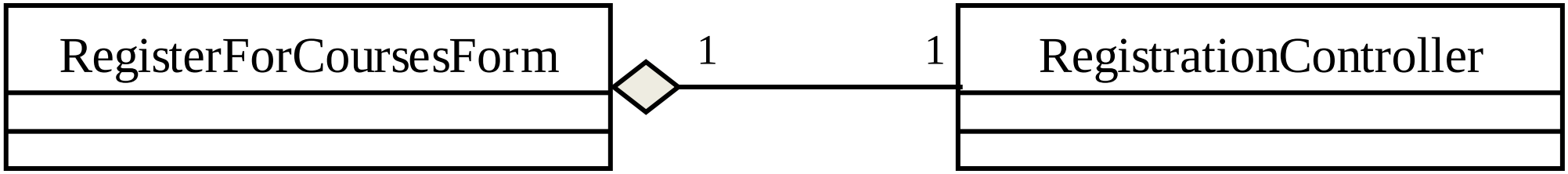
- Xem xét
  - Chu kỳ sống của Class1 và Class2





Ví dụ: Composition

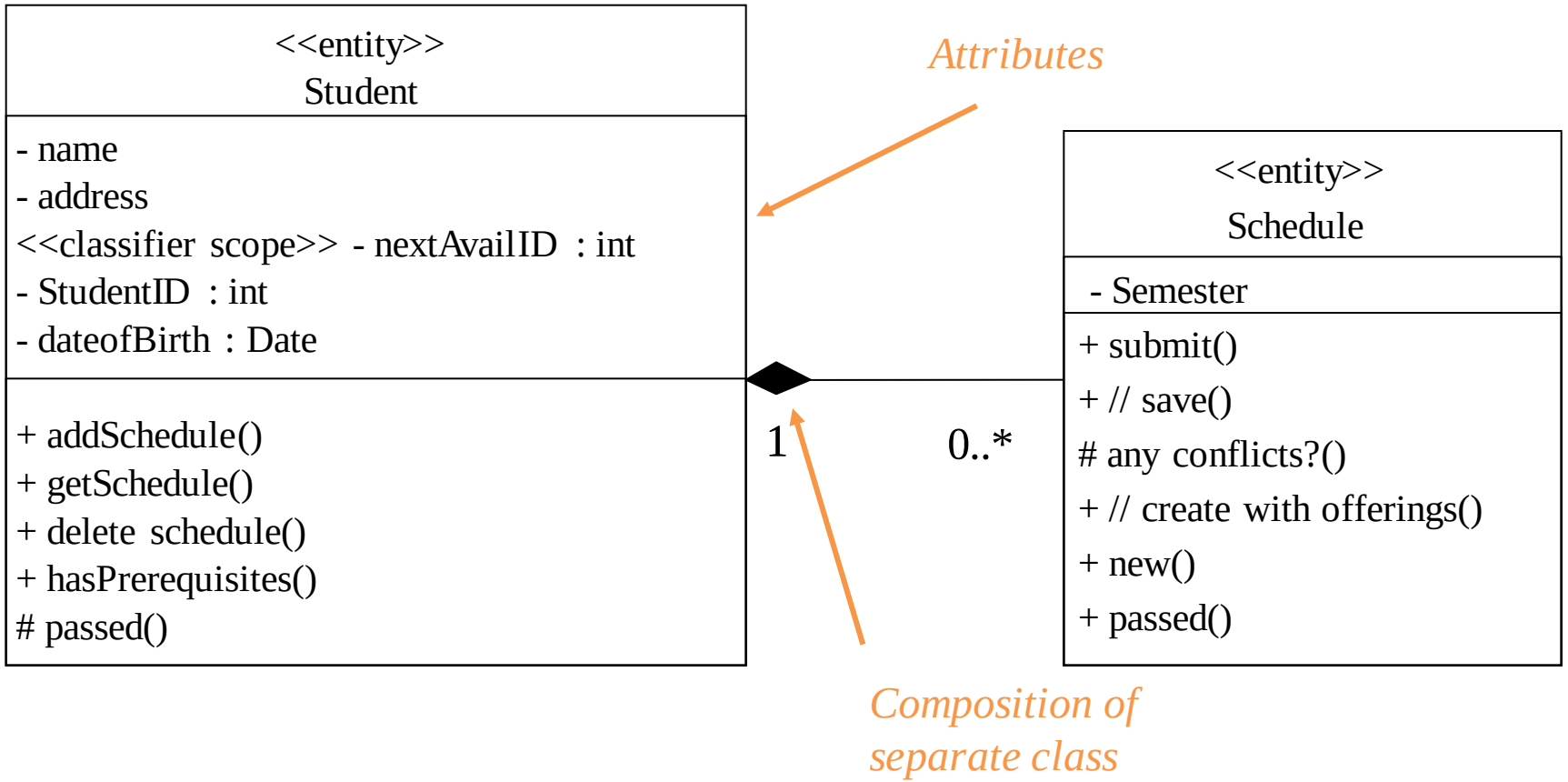




Cân nhắc giữa Attributes và Composition

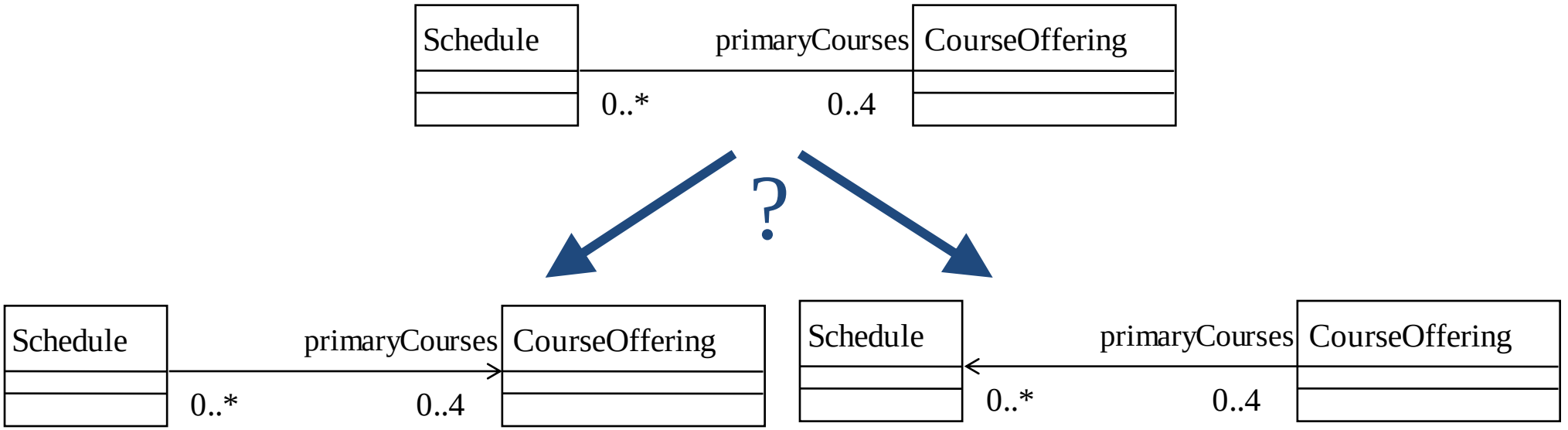
- Dùng composition khi
  - Các thuộc tính cần được nhận dạng độc lập
  - Nhiều class có chung các thuộc tính
  - Các thuộc tính có cấu trúc phức tạp và bản thân chúng cũng có thuộc tính riêng
  - Các thuộc tính có hành vi riêng (phức tạp)
  - Các thuộc tính có quan hệ riêng
- Các trường hợp còn lại dùng attributes

Ví dụ: Attributes/Composition



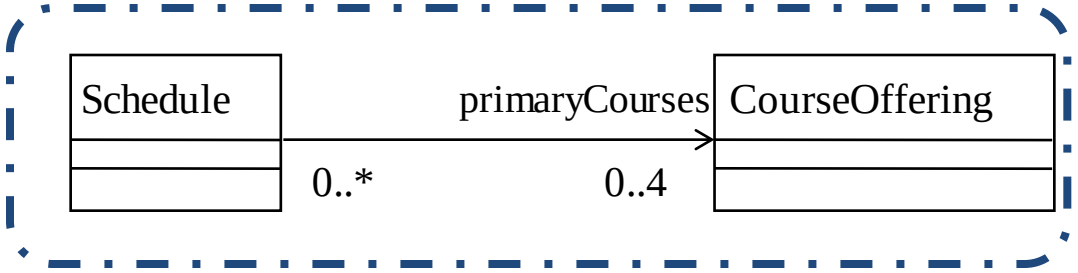
Chiều của quan hệ

- Khảo sát các interaction diagram
- Ngay cả khi cả 2 chiều đều có vẻ cần thiết, vẫn có thể chỉ 1 chiều hoạt động
  - Một chiều quan hệ ít xảy ra
  - Số thể hiện của một class ít

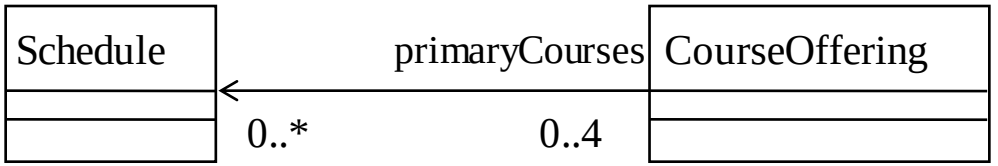


Ví dụ: Tinh chỉnh chiều quan hệ

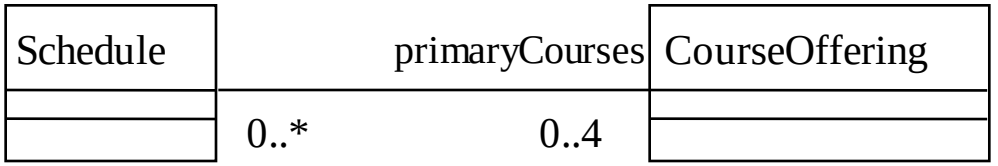
- Tổng số Schedule nhỏ, hay
- Không bao giờ cần một list Schedule có CourseOffering xuất hiện trên đó



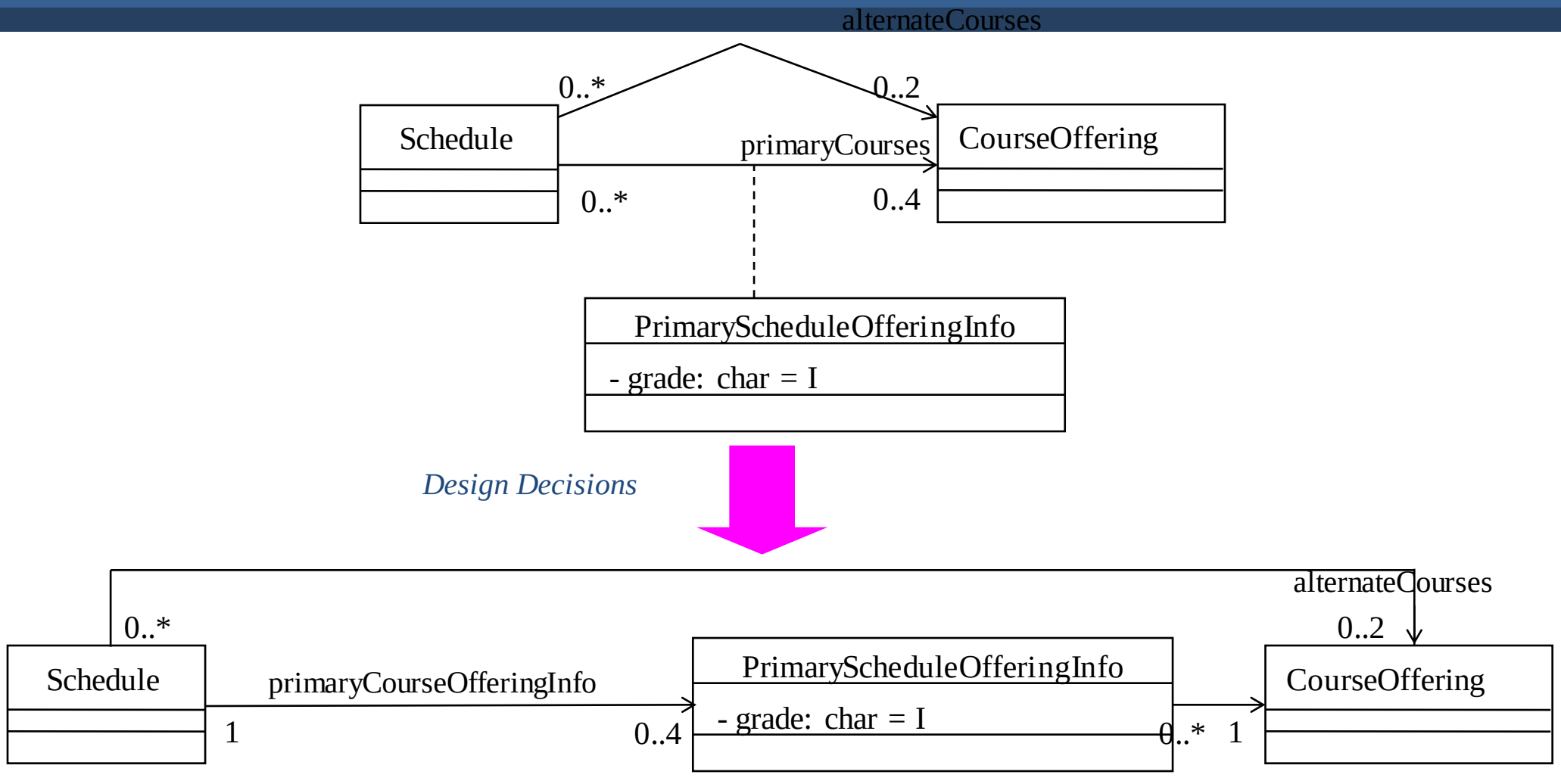
- Tổng số CourseOffering nhỏ, hay
- Không bao giờ cần một list CourseOffering có Schedule xuất hiện trên đó



- Tổng số CourseOffering và Schedule đều không nhỏ
- Phải quan tâm đến cả 2 chiều



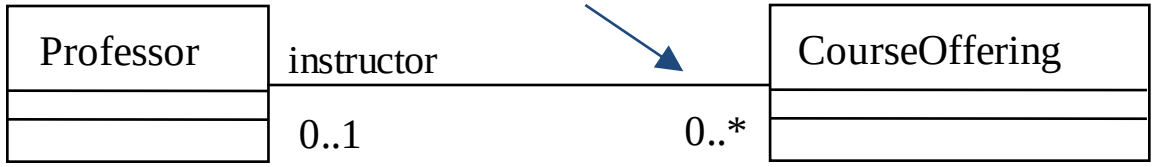
Ví dụ: Thiết kế Association Class



Thiết kế bản số

- Bản số = 1, hay = 0..1
  - Có thể cài đặt đơn giản như một giá trị hay pointer
  - Không cần thiết kế gì thêm

- Bản số > 1
  - Không thể dùng giá trị đơn hay pointer
  - Cần thực hiện thêm một số công việc thiết kế



Cần một container

Nội dung

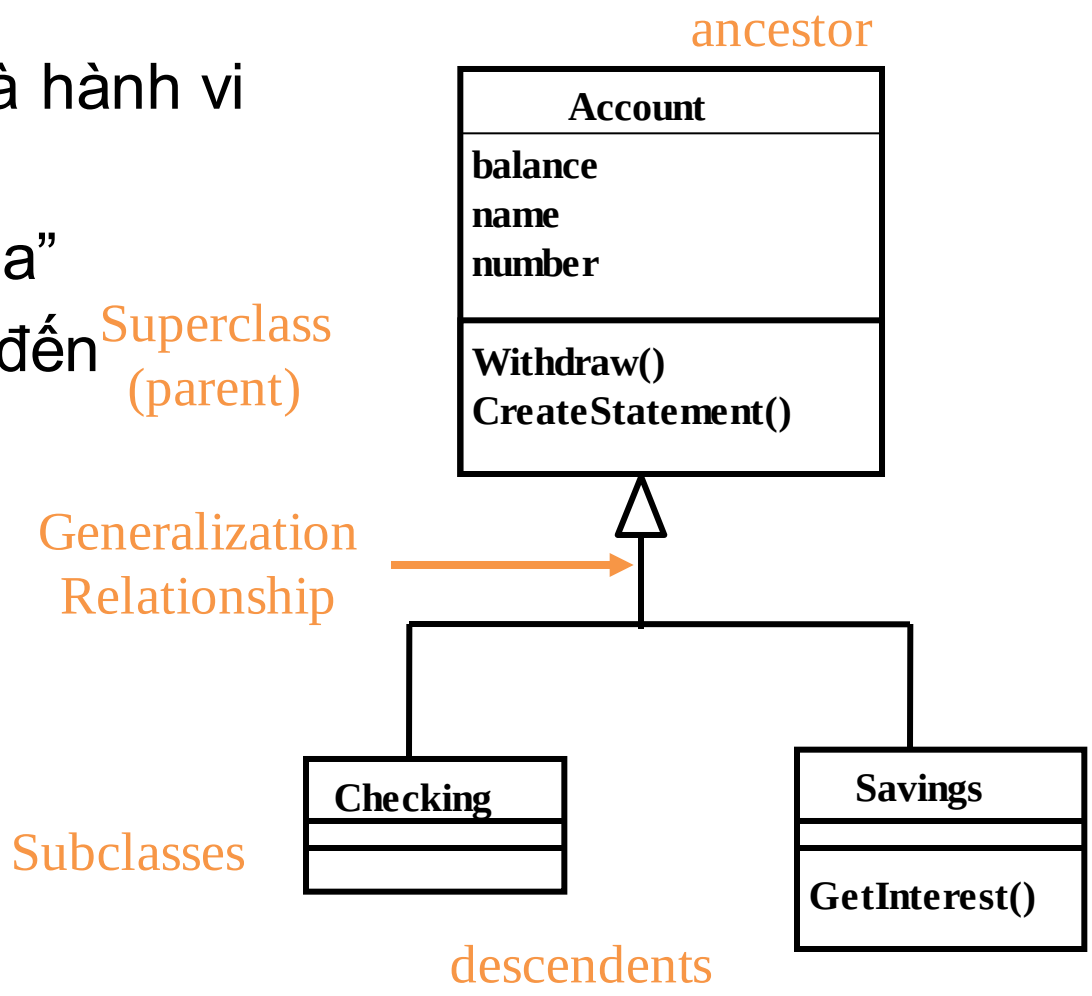
1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

11. Định nghĩa quan hệ tổng quát hóa

- Mục đích
  - Xác định các khả năng dùng lại
  - Tinh chỉnh cây kế thừa để có thể cài đặt hiệu quả
- Những gì cần xem xét:
  - So sánh Abstract classes với concrete classes
  - Bài toán đa kế thừa
  - So sánh Generalization và Aggregation
  - Tổng quát hóa để hỗ trợ tái sử dụng trong cài đặt
  - Tổng quát hóa để hỗ trợ đa xạ (polymorphism)
  - Tổng quát hóa để hỗ trợ đa hình (metamorphosis)
  - Mô phỏng tổng quát hóa

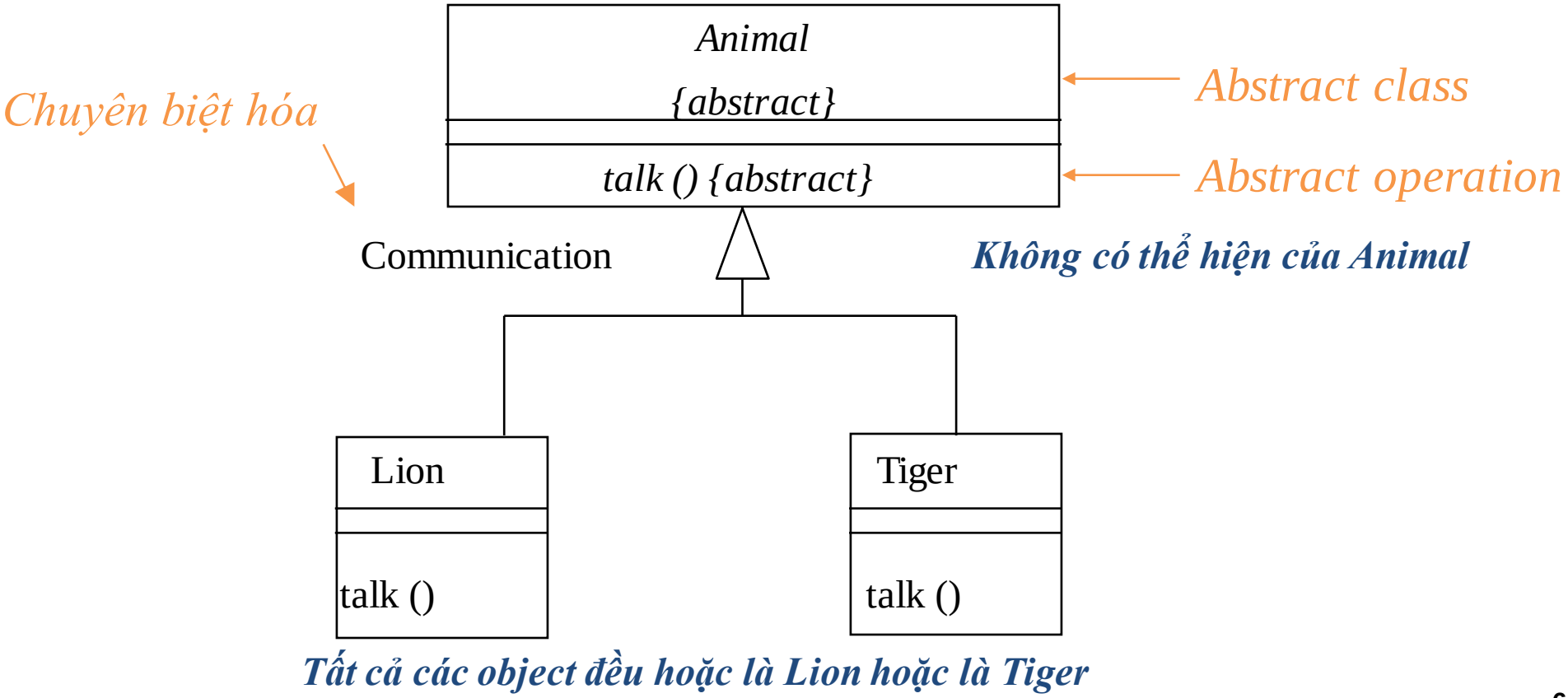
Nhắc lại: Generalization

- Một class chia sẻ cấu trúc và hành vi của một hay nhiều class
- Là quan hệ "Là một dạng của"
- Trong phân tích, ít khi dùng đến



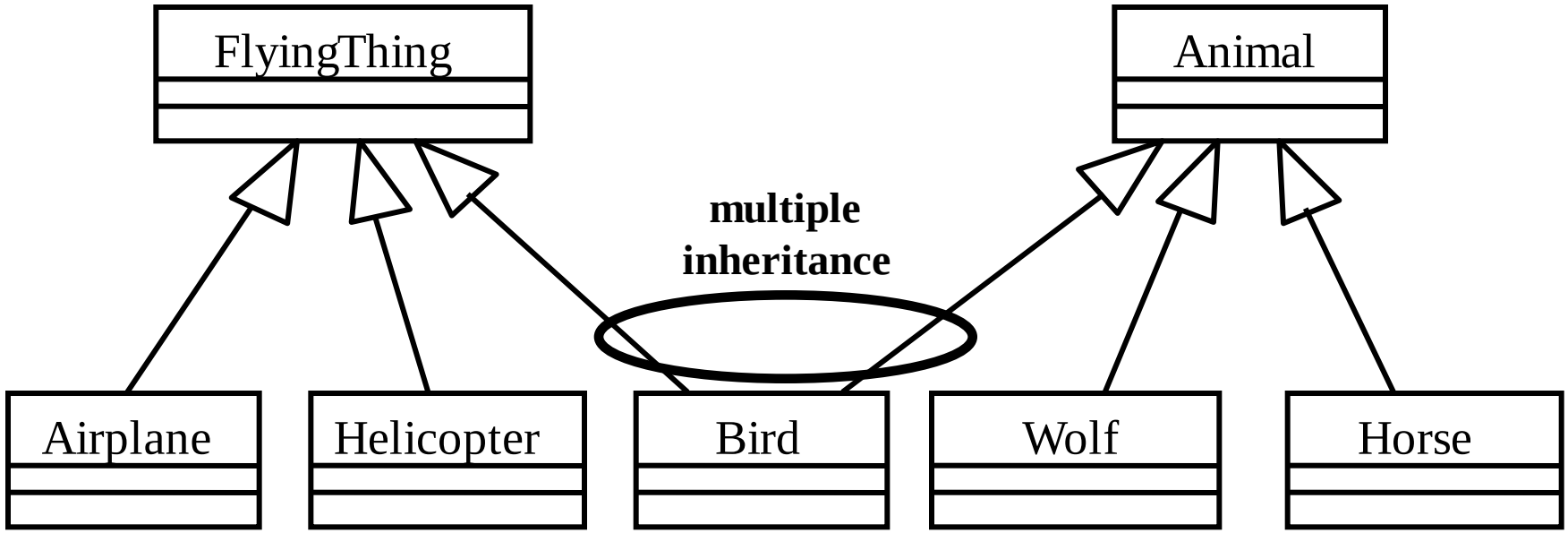
Abstract và Concrete Class

- Abstract class không có bất kỳ thể hiện nào
- Concrete classes có thể có thể hiện (object)



Nhắc lại: Đa kế thừa

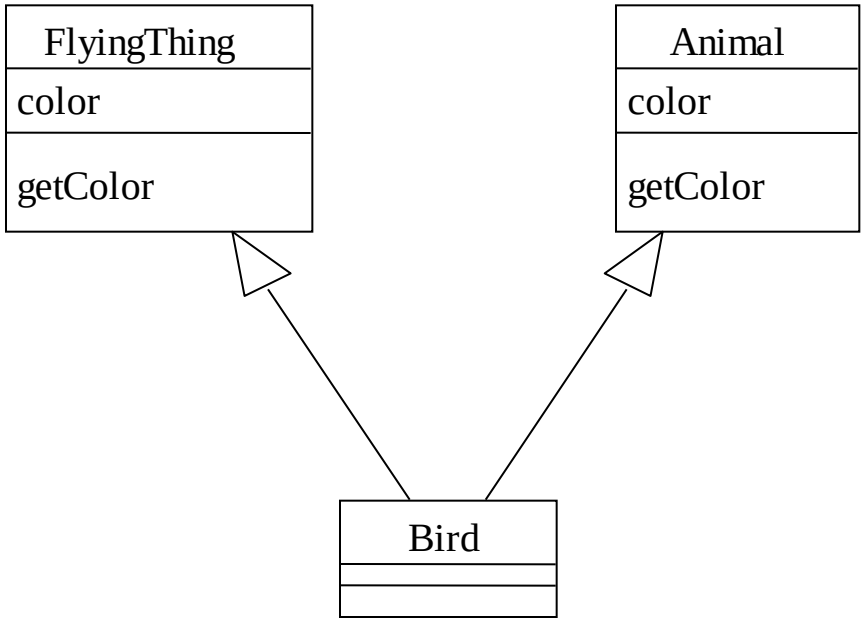
- Một class có thể kế thừa từ nhiều class



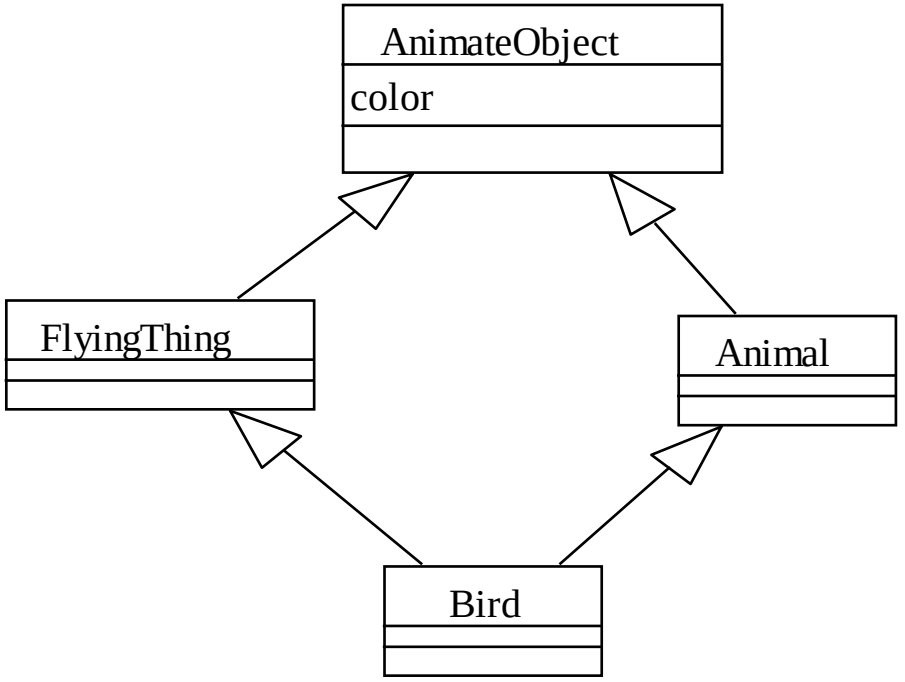
Dùng đa kế thức chỉ khi thật cần thiết, và phải luôn cẩn thận !

Các vấn đề của đa kế thừa

Tên của attribute hay operation bị trùng



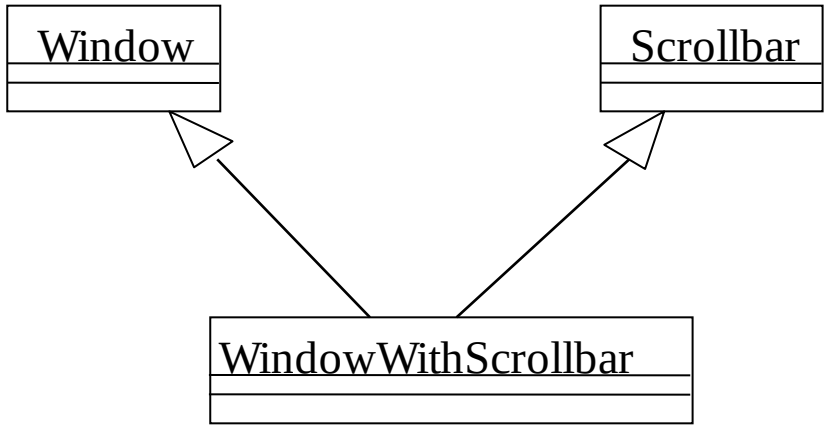
Lặp lại việc kế thừa



Lời giải của các vấn đề trên phụ thuộc cài đặt cụ thể

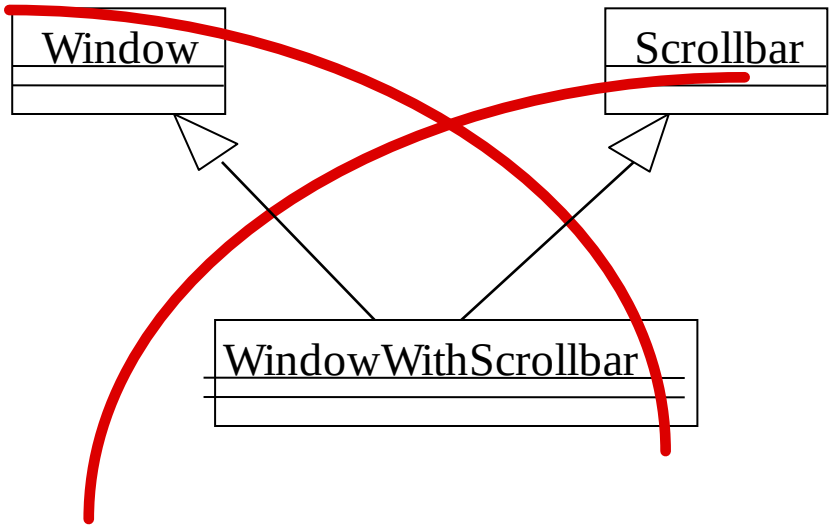
Chọn Generalization hay Aggregation

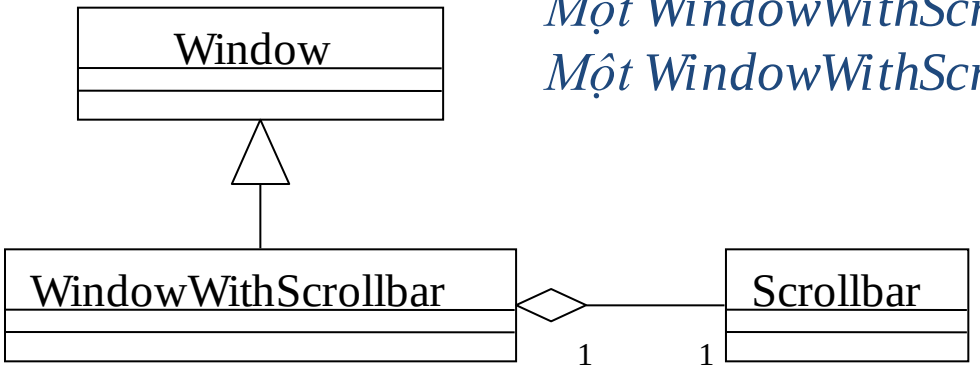
- Rất dễ nhầm lẫm giữa Generalization và aggregation
  - Generalization biểu diễn quan hệ "là một" hay "dạng của"
  - Aggregation biểu diễn quan hệ "một bộ phận của"



Có đúng không?

Chọn Generalization hay Aggregation

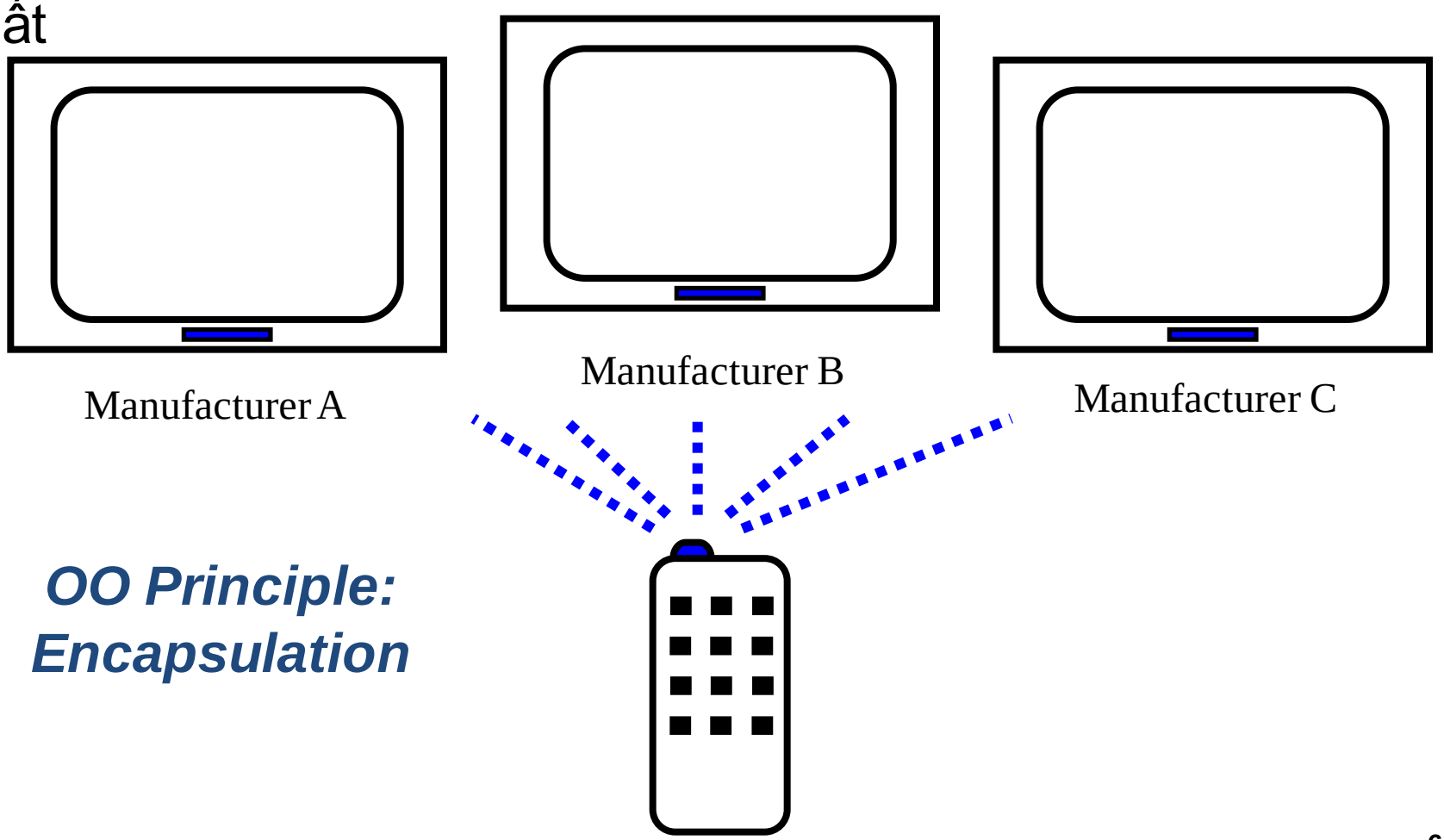




Một WindowWithScrollbar "là một" Window
Một WindowWithScrollbar "chứa một" Scrollbar

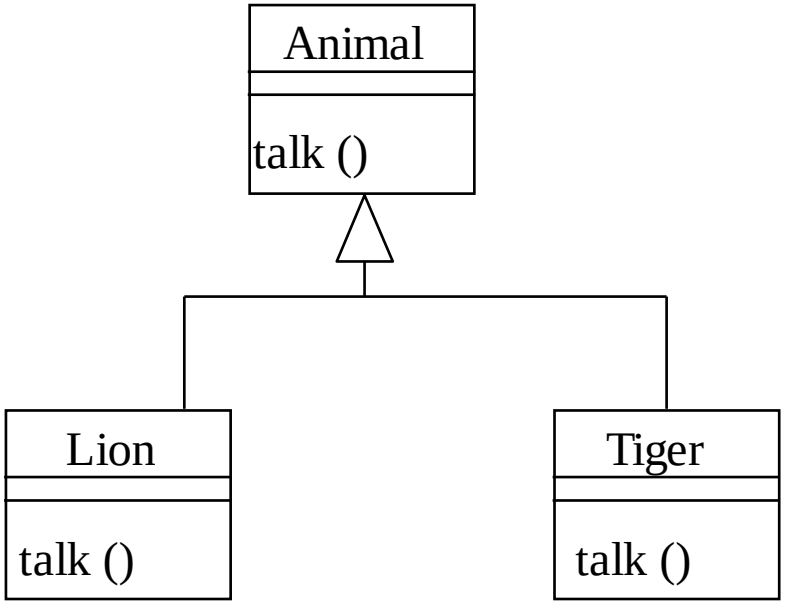
Nhắc lại: Polymorphism là gì ?

- Khả năng che dấu nhiều cài đặt bên dưới một interface duy nhất



OO Principle: Encapsulation

Cài đặt Polymorphism



Không có Polymorphism

```
if animal = "Lion" then
        do the Lion talk
else if animal = "Tiger" then
        do the Tiger talk
end
```

Có Polymorphism

```
do the Animal talk
```

Nội dung

1. Tổng quan về lớp
2. Ký hiệu
3. Xác định lớp
4. Định nghĩa các Operation
5. Định nghĩa Class Visibility
6. Định nghĩa các phương thức
7. Định nghĩa các trạng thái
8. Định nghĩa các thuộc tính
9. Định nghĩa các phụ thuộc
10. Định nghĩa các mối kết hợp
11. Định nghĩa các quan hệ tổng quát hóa
12. Checkpoints

Checkpoints: Các Class

- Tên của mỗi class có phản ánh rõ vai trò của nó không?
- Class có biểu diễn một single well-defined abstraction?
- Tất cả các attribute và trách nhiệm có gắn kết với nhau?
- Có bất kỳ class attribute, operation hay relationship nào cần tổng quát hóa, nghĩa là, chuyển lên tổ tiên không?
- Mọi yêu cầu trên class đã xử lý?
- Mọi đòi hỏi trên class phù hợp với với statecharts mô hình hóa hành vi của class và các thể hiện của nó?
- Đã mô tả trọn vẹn chu kỳ sống của các thể hiện của class ?
- Class thực hiện mọi hành vi cần thiết?

Checkpoints: Operations

- Các operation có dễ hiểu?
- Các mô tả trạng thái của class và hành vi của các object của nó có chính xác?
- Class có thực hiện đúng hành vi yêu cầu nó?
- Bạn đã các định các tham số đúng chưa ?
- Bạn đã gán đầy đủ operations cho các message của mỗi object?
- Các đặc tả cài đặt (nếu có) của operation có chính xác ?
- Các operation signature có phù hợp với ngôn ngữ lập trình cài đặt hệ thống?
- Tất cả các operation đề cần cho use-case realization?

Checkpoints: Attributes

- Mỗi attribute biểu diễn một khái niệm đơn?
- Tên của các attribute có gợi nhớ?
- Tất cả các attribute là cần thiết cho các use-case realization?

Checkpoints: Relationships

- Tên của role gợi nhớ?
- Bản số của các relationship có chính xác?

Nhắc lại: Class Design

- Mục đích của Class Design là gì?
- Các class được tinh chỉnh bằng cách nào?
- Các statechart được tạo cho mỗi class?
- Các component chính của statechart là gì ? Cho mô tả ngắn gọn về mỗi thứ.
- Có những dạng tinh chỉnh relationship nào?
- Sự khác nhau giữa association và dependency?
- Ta phải làm gì với operations và attributes?