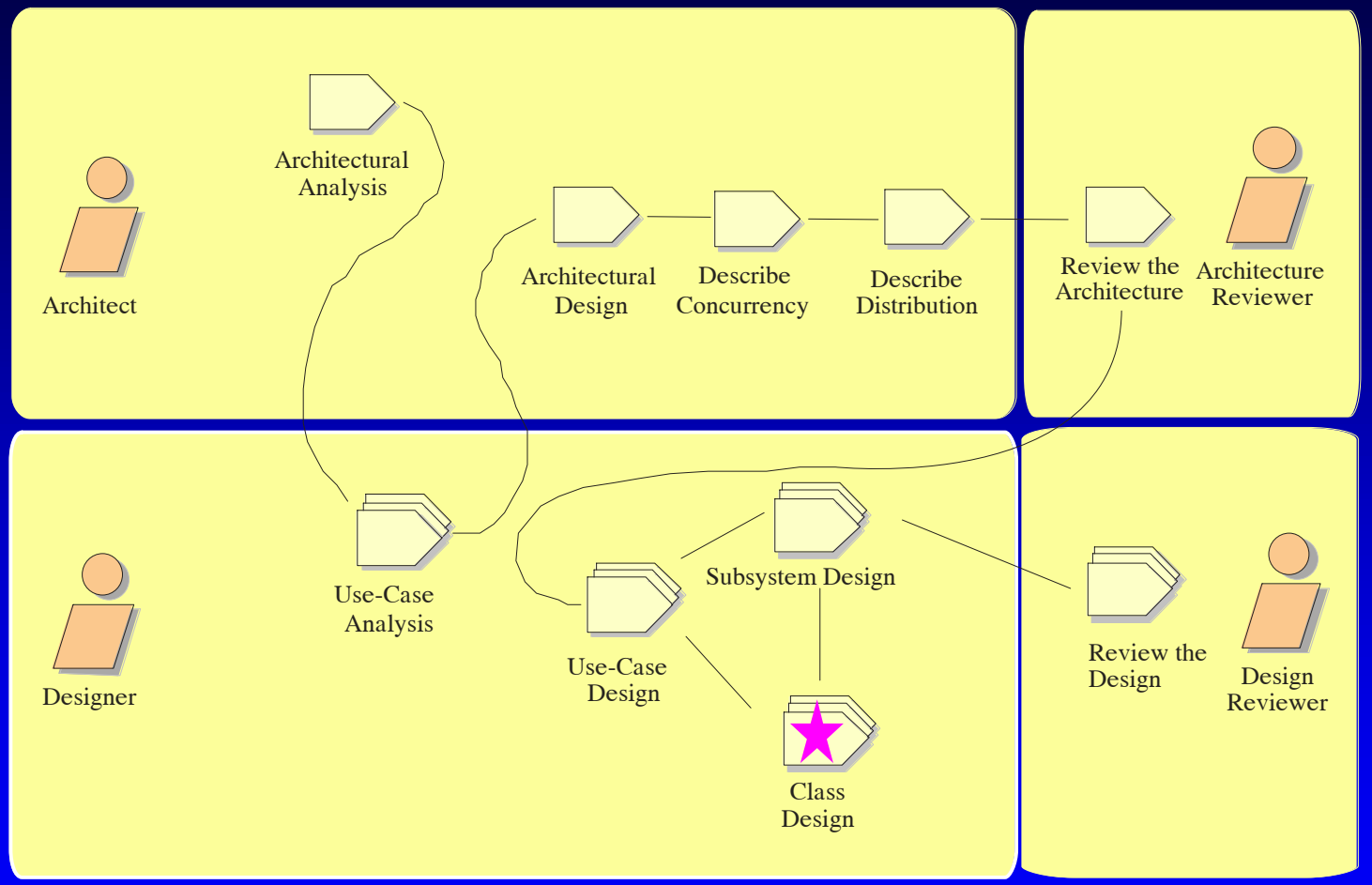
Phân tích và Thiết kế Hướng đối tượng dùng UML

Module 13: Thiết kế Class

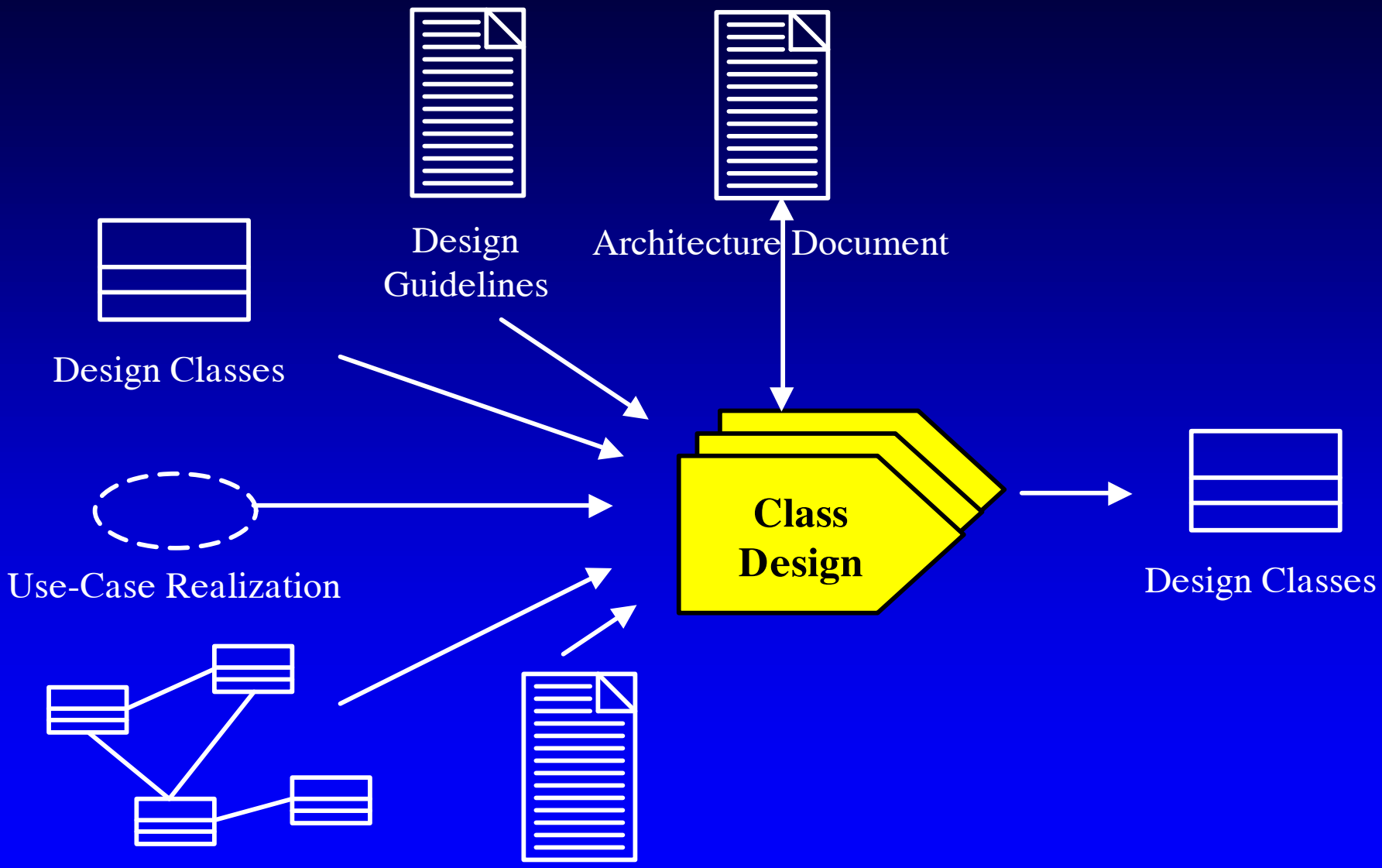
Mục tiêu

- Tìm hiểu mục đích của bước thiết kế Class và vị trí của công đoạn này trong qui trình
- Xác định bổ sung các class và quan hệ của chúng cần để hỗ trợ cho việc cài đặt các cơ chế kiến trúc đã chọn
- Xác định và phân tích việc chuyển đổi trạng thái các đối tường trong các class kiểm soát được trạng thái
- Tinh chỉnh các quan hệ, operation, và thuộc tính

Vị trí của Thiết kế Class



Tổng quan về Class



Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Các khảo sát khi thiết kế Class

- Class stereotype
  - Boundary
  - Entity
  - Control
- Các design pattern khả dụng
- Các cơ chế kiến trúc
  - Persistence
  - Distribution
  - …

Cần bao nhiêu Class ?

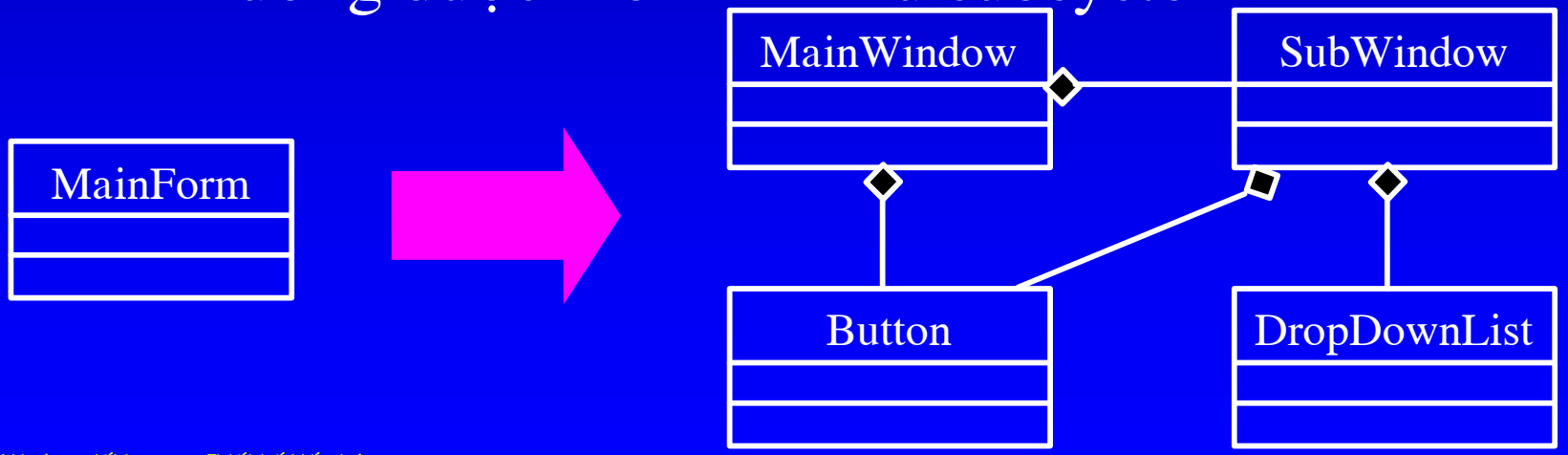
- Nếu nhiều class đơn giản. Nghĩa là mỗi class:
  - Đóng gói một phần ít hơn trên toàn bộ hệ thống
  - Nhiều khả năng dùng lại hơn
  - Dễ cài đặt hơn
- Nếu nhiều class phức tạp. Nghĩa là mỗi class:
  - Đóng gói một phần nhiều hơn trên toàn bộ hệ thống
  - Ít khả năng dùng lại hơn
  - Khó cài đặt hơn

Một class phải có một mục tiêu rõ ràng.

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

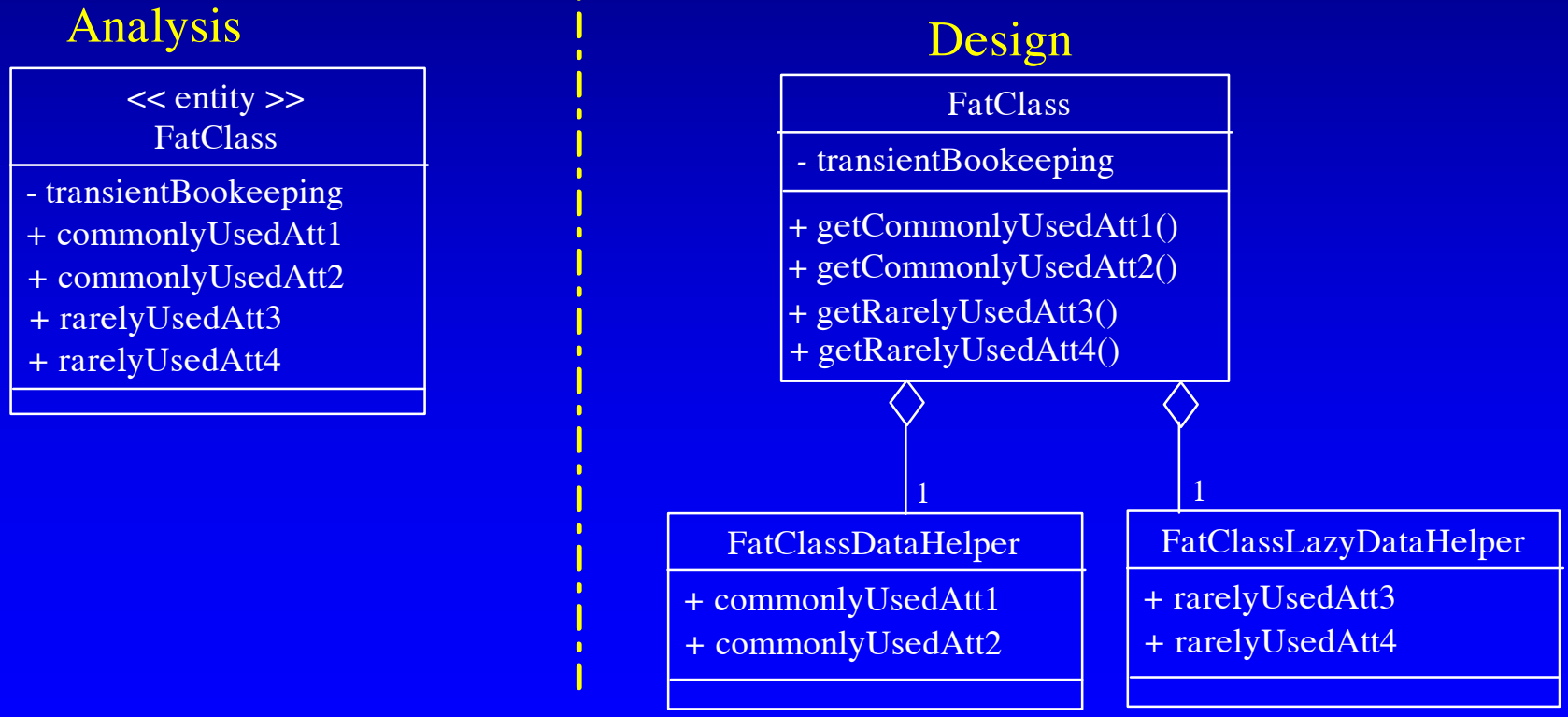
Thiết kế các Boundary Class

- 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

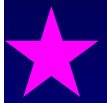
- Các Entity object thường thụ động và persistent
- Các yêu cầu về hiệu năng có thể buộc ta phải tái xây dựng
- Xem thêm bước xác định Persistent Class



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

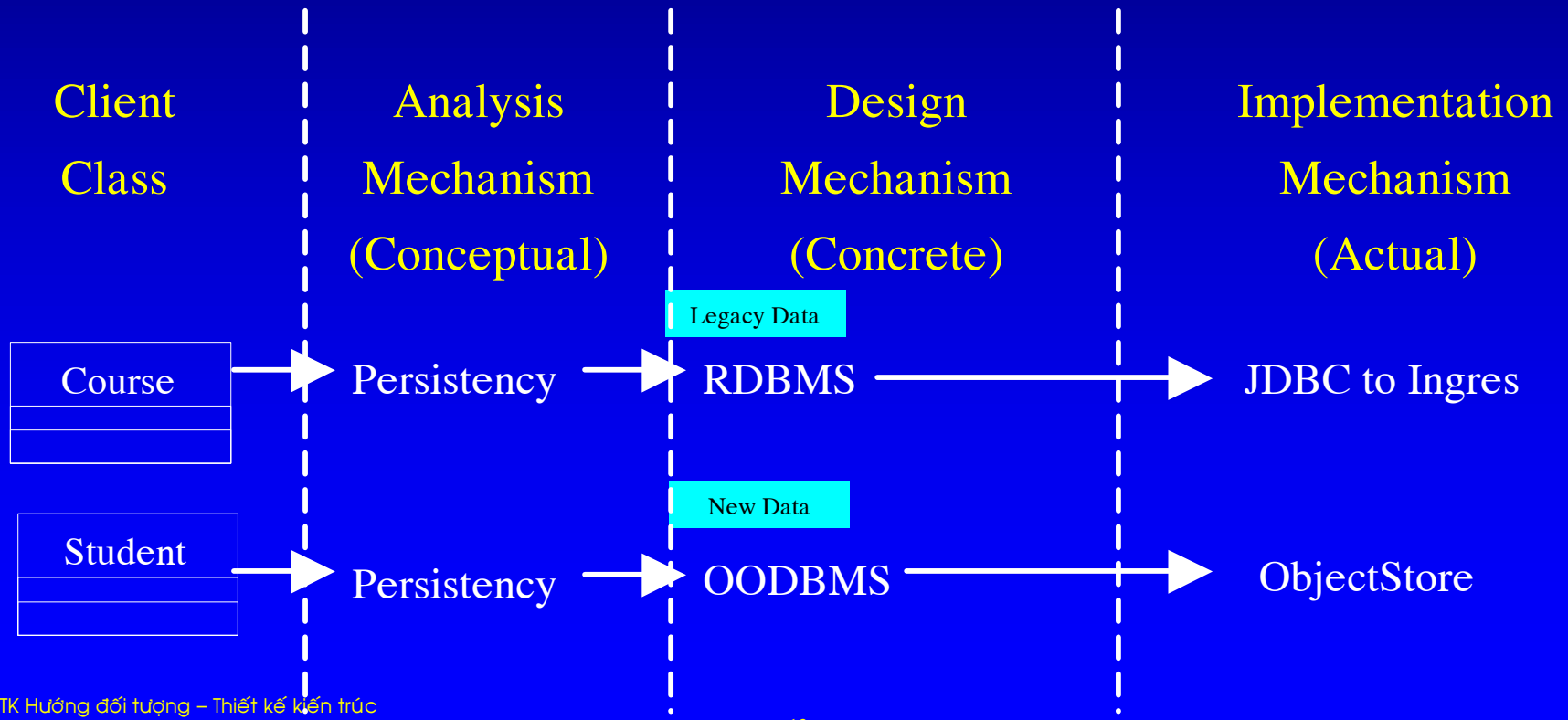
Các bước thiết kế Class



- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

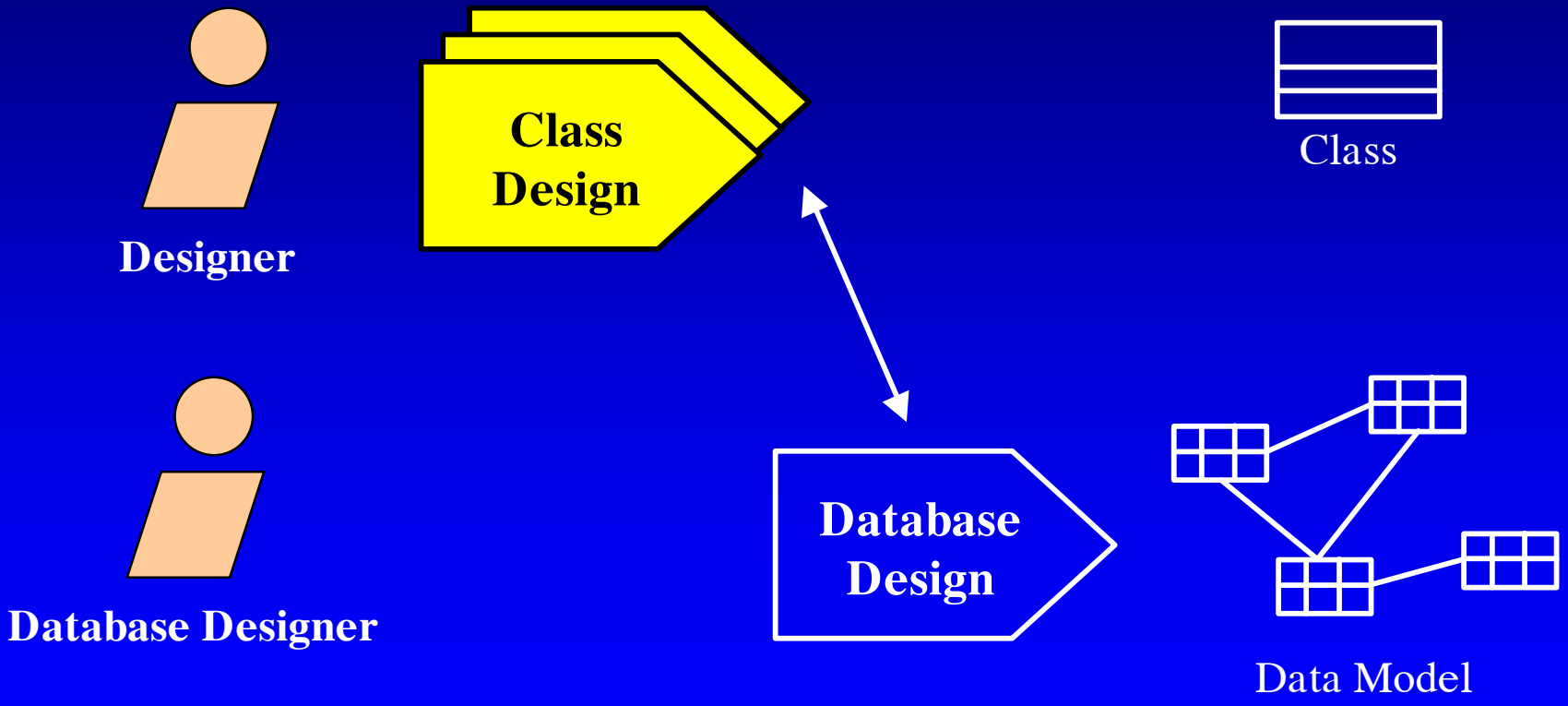
Xác định Persistent Class

- Mọi thể hiện của class đều đòi hỏi phải lưu giữ trạng thái của nó
- Các Persistent class được gán với cơ chế persistence



Database Design Preview

- Persistence strategy must be coordinated
- Ở đây, nhớ rằng các class đều persistent



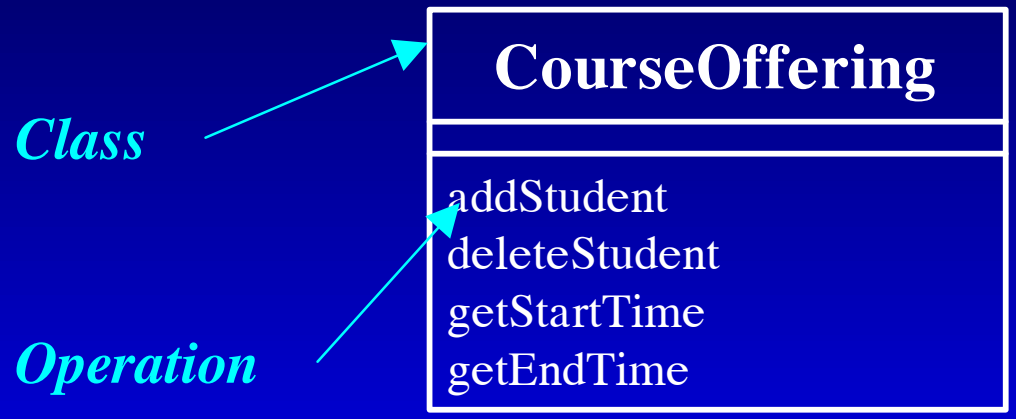
Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Định nghĩa các Operation

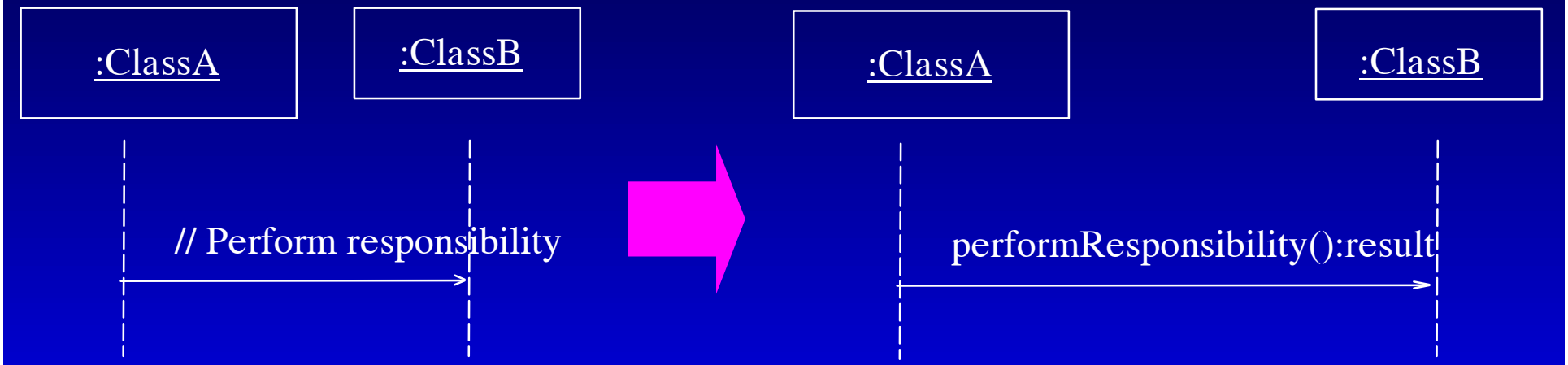
- Mục đích
  - Ánh xạ các nhiệm vụ đã xác định ơ mức phân tích thành các operation thực hiện chúng
- Những cái cần xem xét:
  - Tên Operation, signature, và mô tả
  - Operation visibility
  - Tầm vực Operation
    - Class operation hay instance operation

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, …

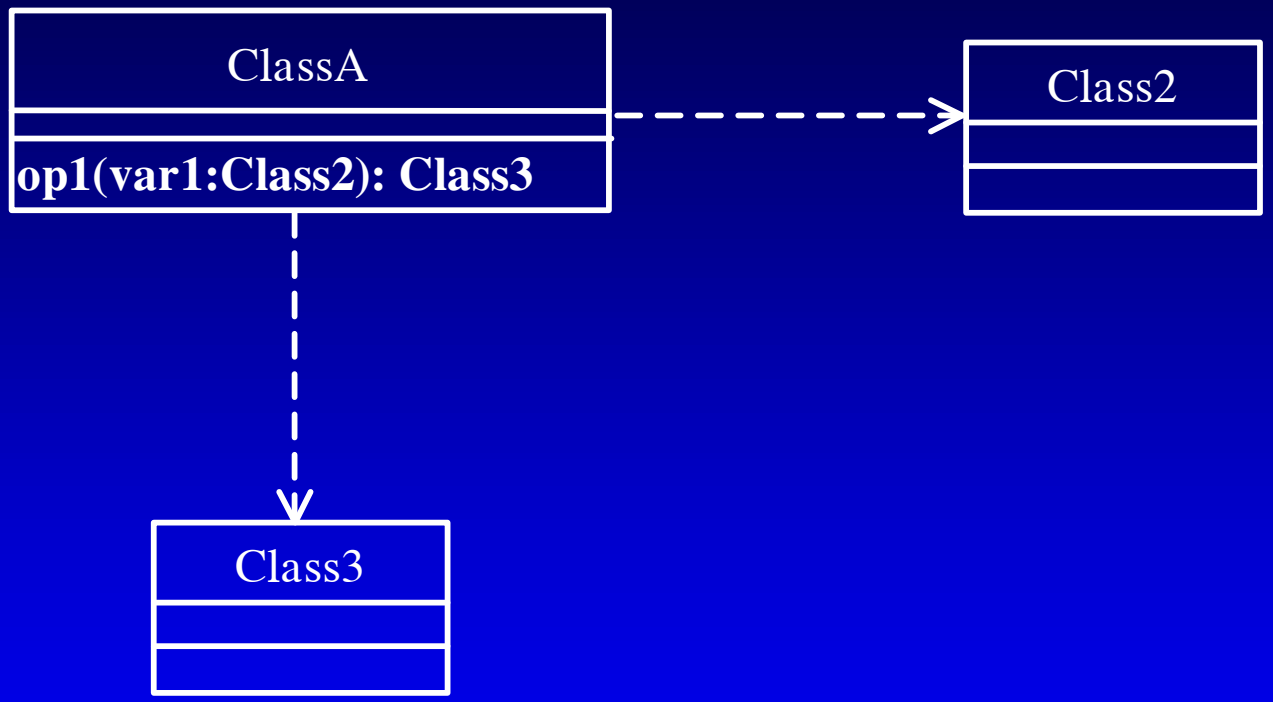
Đặt tên và mô tả các Operation

- 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

Guidelines: 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"

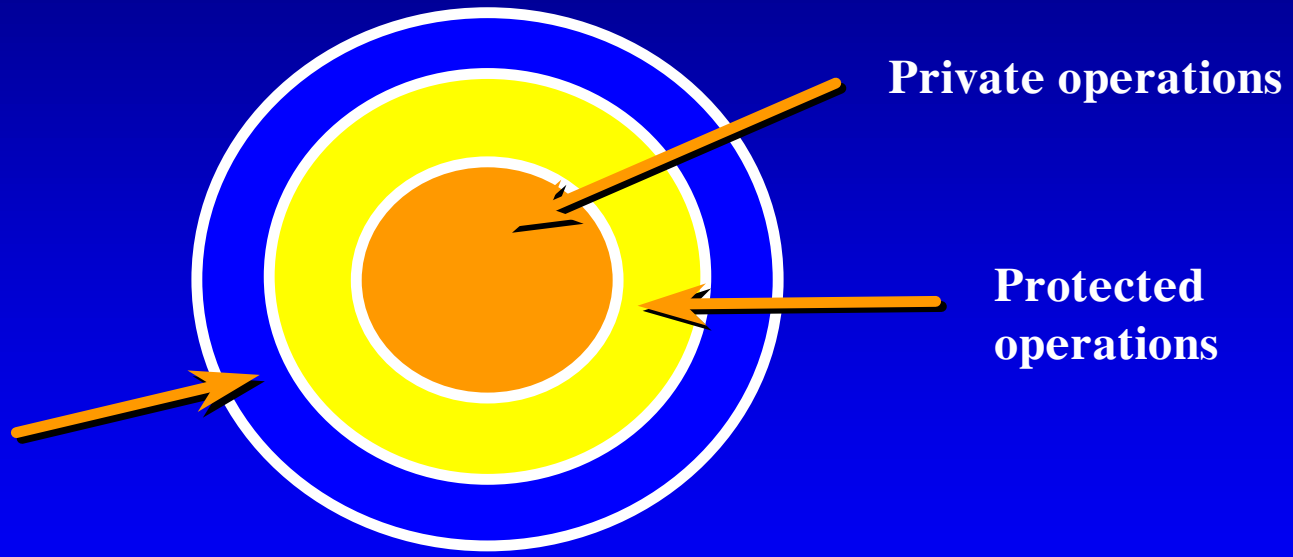
Phát hiện Additional Classes và Relationships



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

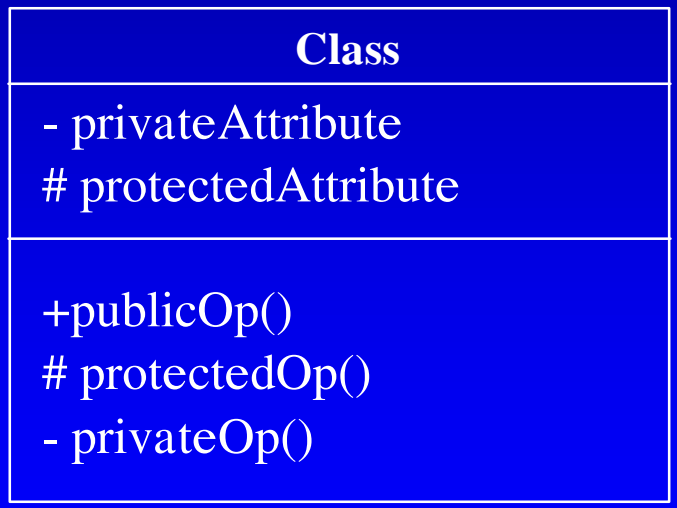
Operation Visibility

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



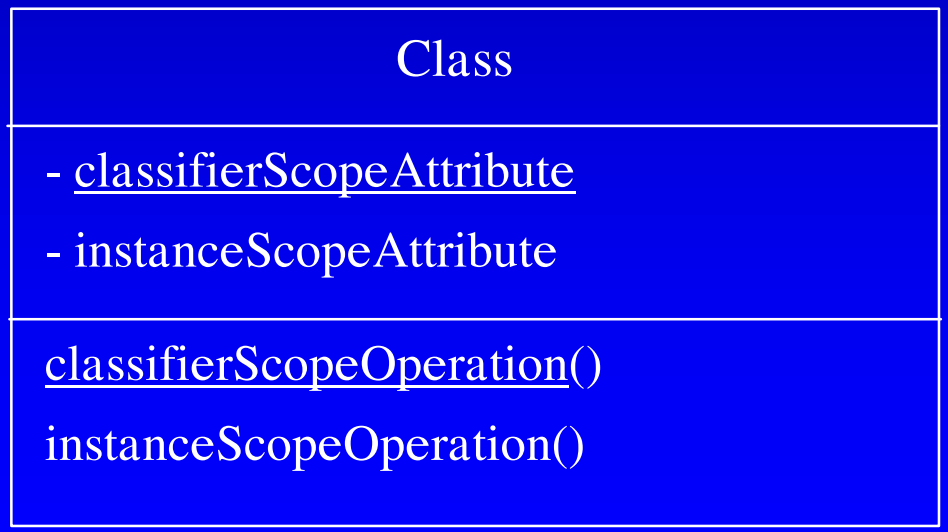
Ký hiệu tính khả kiến?

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

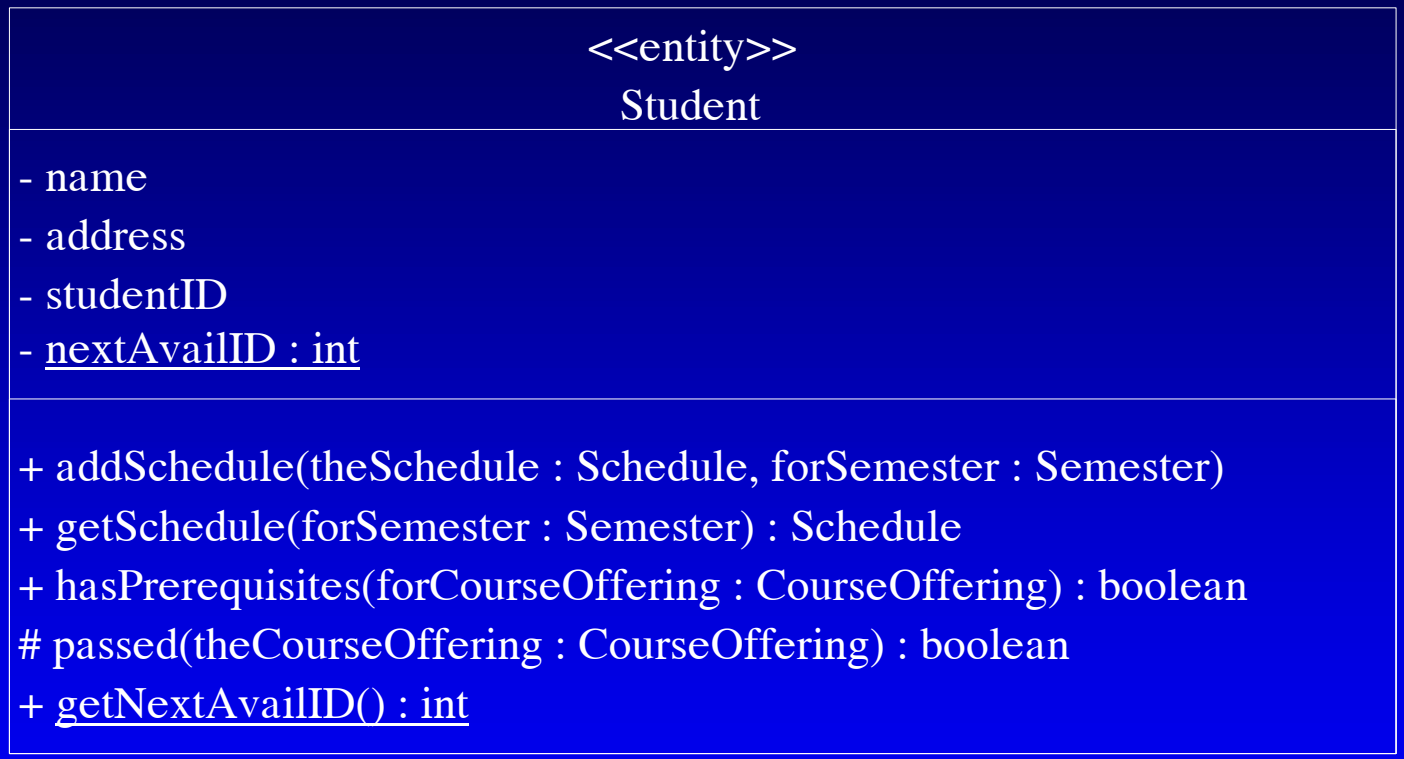


Tầm vực

- 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

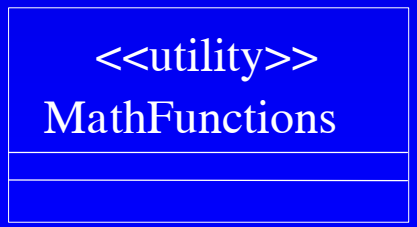


Ví dụ: Scope

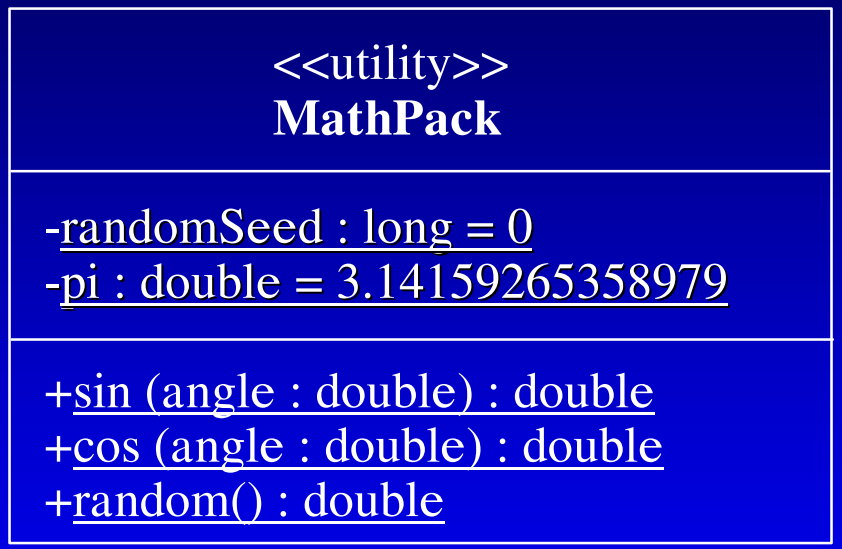

<<entity>>
Student
- name
- address
- studentID
- nextAvailID : int
+ addSchedule(theSchedule : Schedule, forSemester : Semester)
+ getSchedule(forSemester : Semester) : Schedule
+ hasPrerequisites(forCourseOffering : CourseOffering) : boolean
# passed(theCourseOffering : CourseOffering) : boolean
+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

Utility Classes

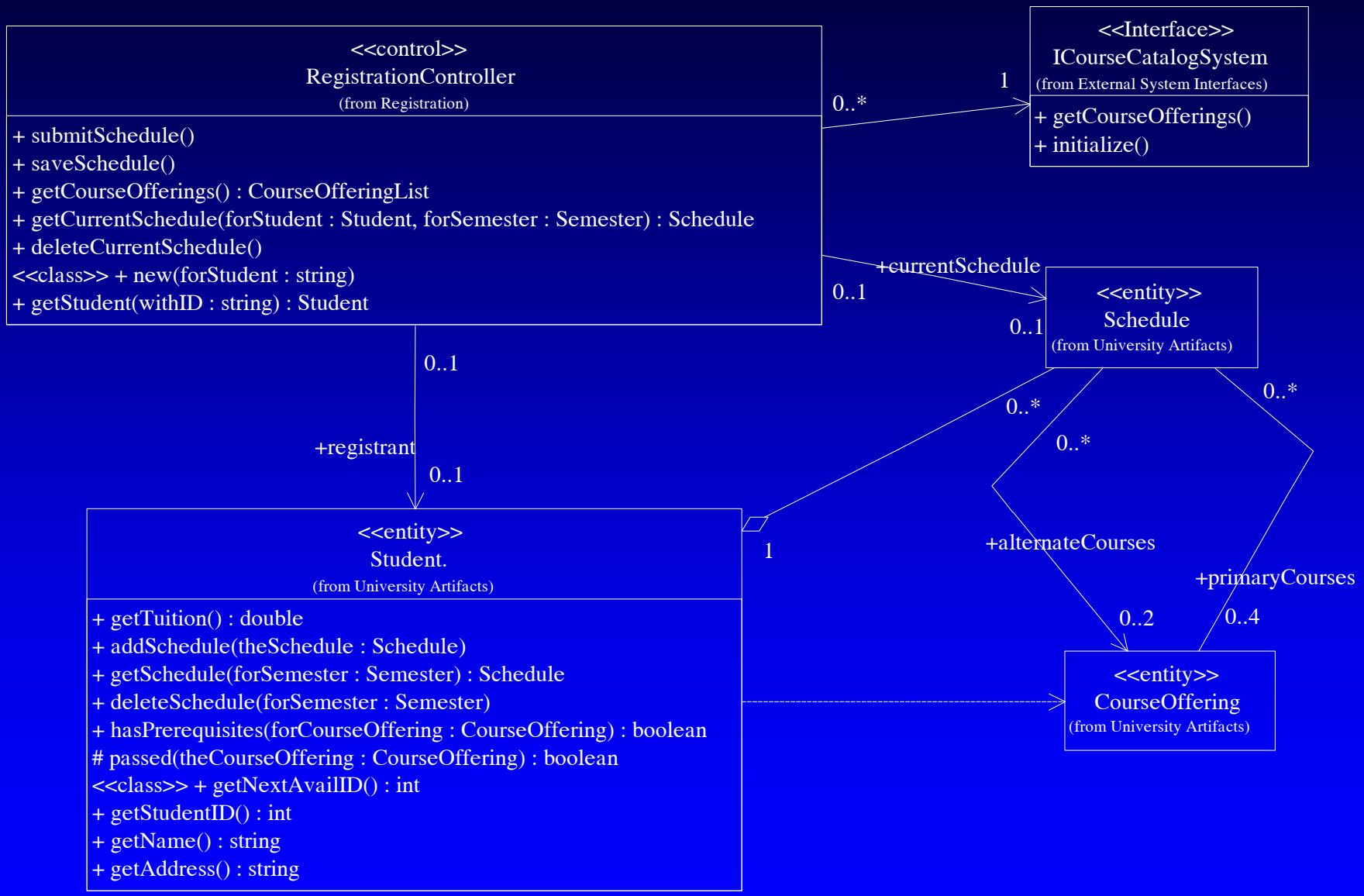
- Thế nào là một Utility Class?
  - Utility là một class stereotype
  - Dùng để chỉ các class chứa một bộ các chương trình con miễn phí
- Tại sao lại dùng chúng?
  - Để cung cấp các dịch vụ có thể hữu dụng trong các ngữ cảnh khác nhau
  - Để gói các hàm thư viện hay các ứng dụng phi đối tượng



Ví dụ: Utility Classes



Ví dụ: Định nghĩa các Operation



Bài tập: Định nghĩa các Operation

- Hãy cho biết:
  - Các architectural layers, các package và các phụ thuộc của chúng
  - Các Design class cho một use case cụ thể

(còn tiếp)

Bài tập: Define Operations (tt.)

- Với các design class, hãy xác định:
  - Các Operation và mô tả hoàn chỉnh của chúng
  - Operation scope và visibility
  - Mọi mối quan hệ và các class bổ sung để hỗ trợ cho các operation đã định nghĩa

(còn tiếp)

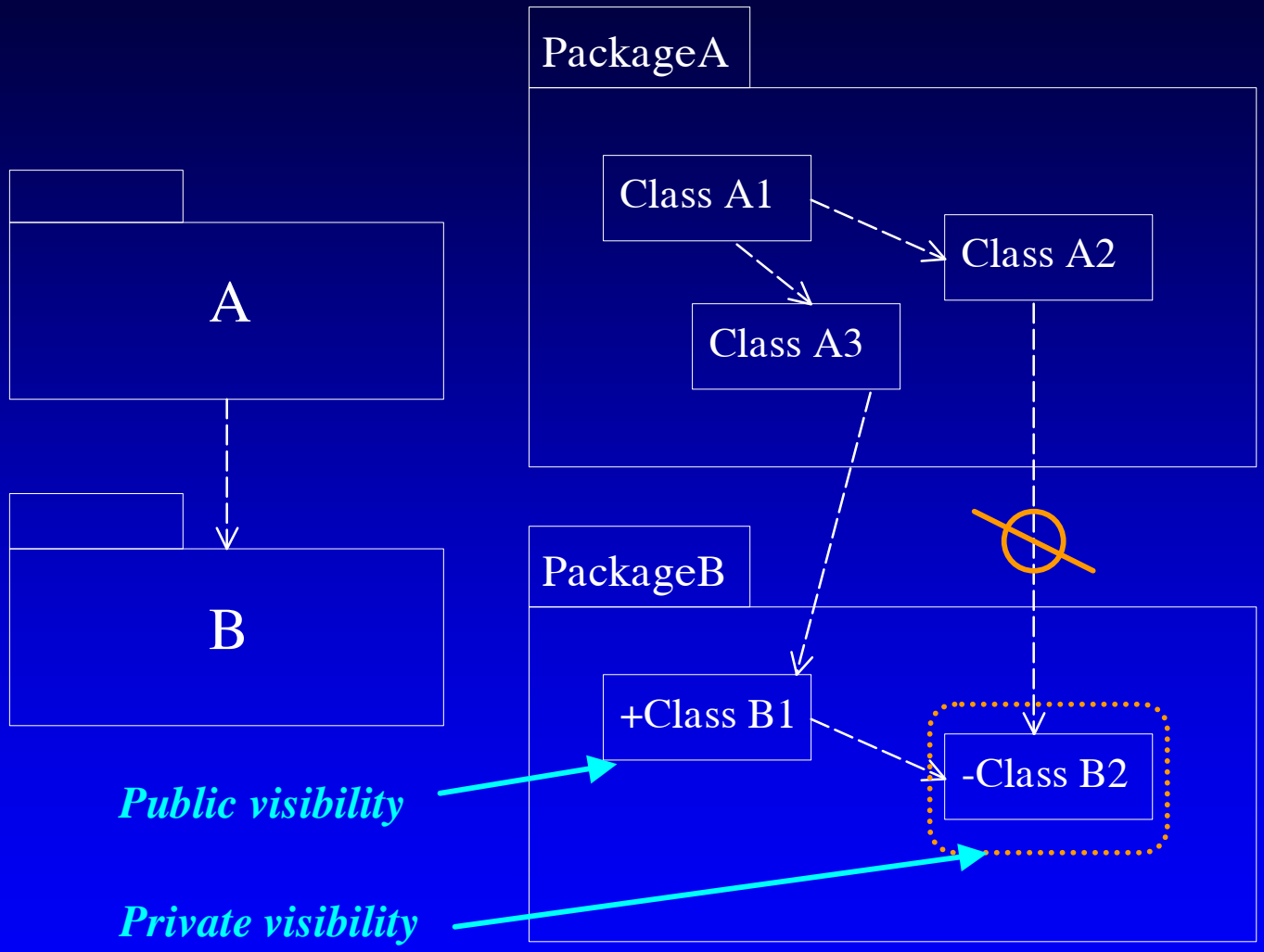
Bài tập: Định nghĩa các Operation (tt.)

- Xây dựng lược đồ sau:
  - VOPC class diagram, chứa tất cả các operation, operation signature, và các quan hệ

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Nhắc lại: Package Element Visibility



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

OO Principle: Encapsulation

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

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

Các bước thiết kế Class

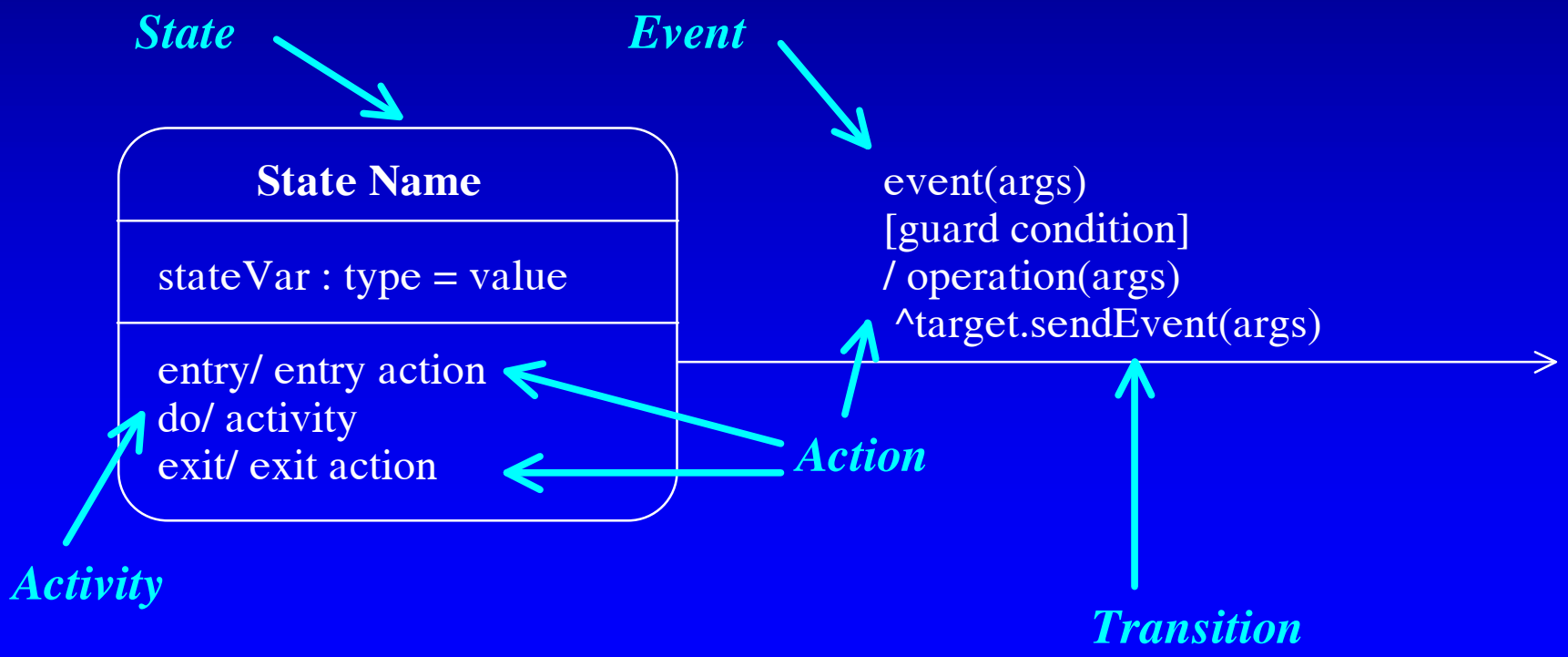
- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Định nghĩa các trạng thái

- Mục đích
  - Thiết kế ảnh hưởng của trạng thái đối tượng lên hành vi của nó
  - Phát triển statecharts để mô hình các hành vi này
- Những gì cần xem xét:
  - Những object nào có trạng thái đáng kể?
  - Cách xác định các trạng thái của một object?
  - Cách ánh xạ statechart với phần còn lại của mô hình?

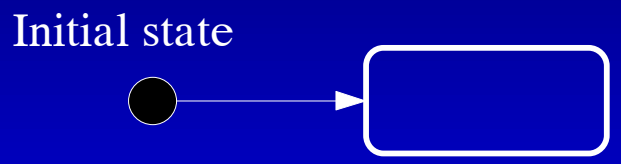
Statechart là gì?

- Là 1 đồ thị có hướng với các node là các trạng thái nối với nhau bởi các transition
- Mô tả lịch sử đời sống của đối tượng

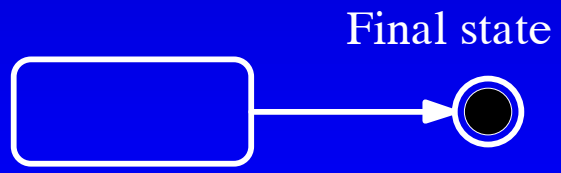


Các trạng thái đặc biệt

- Trang thái bắt đầu (Initial state)
  - Là trạng thái khi mới được khởi tạo của object
  - Mang tính bắt buộc
  - Chỉ có thể có 1 initial state



- Trang thái kết thúc (Final state)
  - Chỉ vị trí kết thúc đời sống của object
  - Optional
  - Có thể có nhiều



Qui trình suy dẫn ra Statecharts

- Xác định và định nghĩa các trạng thái
- Xác định các event
- Xác định các transition (hồi đáp lại các event)
- Thêm các activity và các action

Xác định và định nghĩa các trạng thái

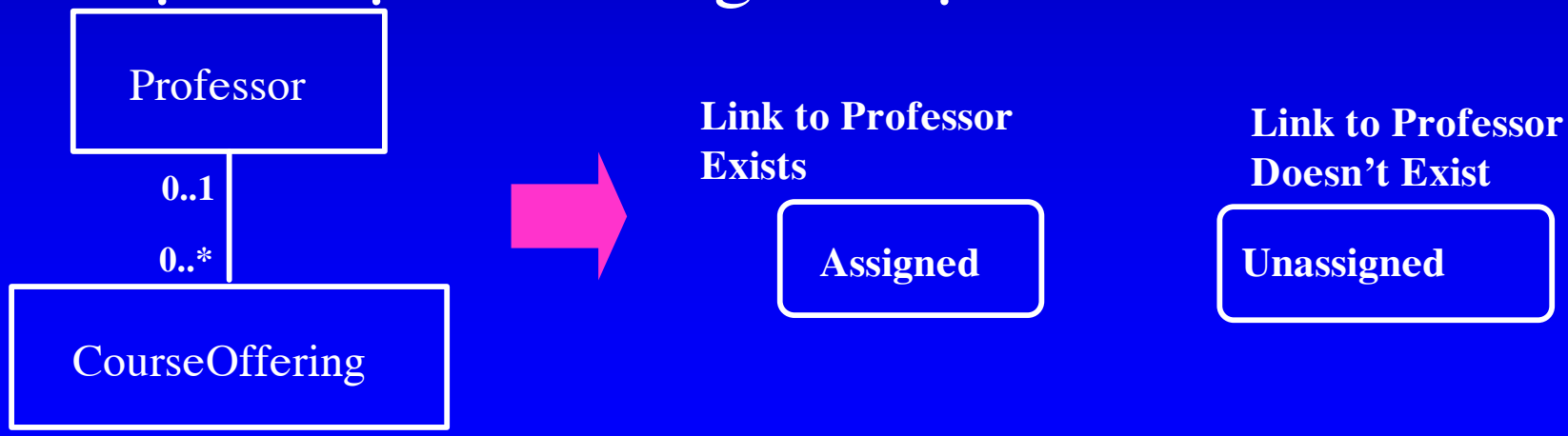
- Significant, dynamic attributes

Số sinh viên tối đa trong 1 lớp là 100

numStudents < 100 | **numStudents >= 100**

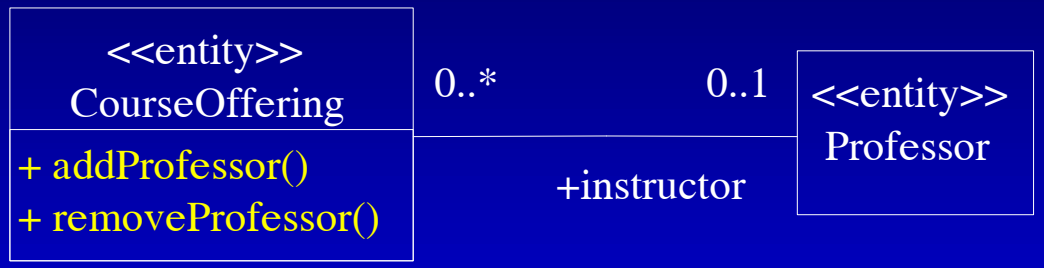
Open | **Closed**

- Sự tồn tại và không tồn tại của các link



Xác định các Event

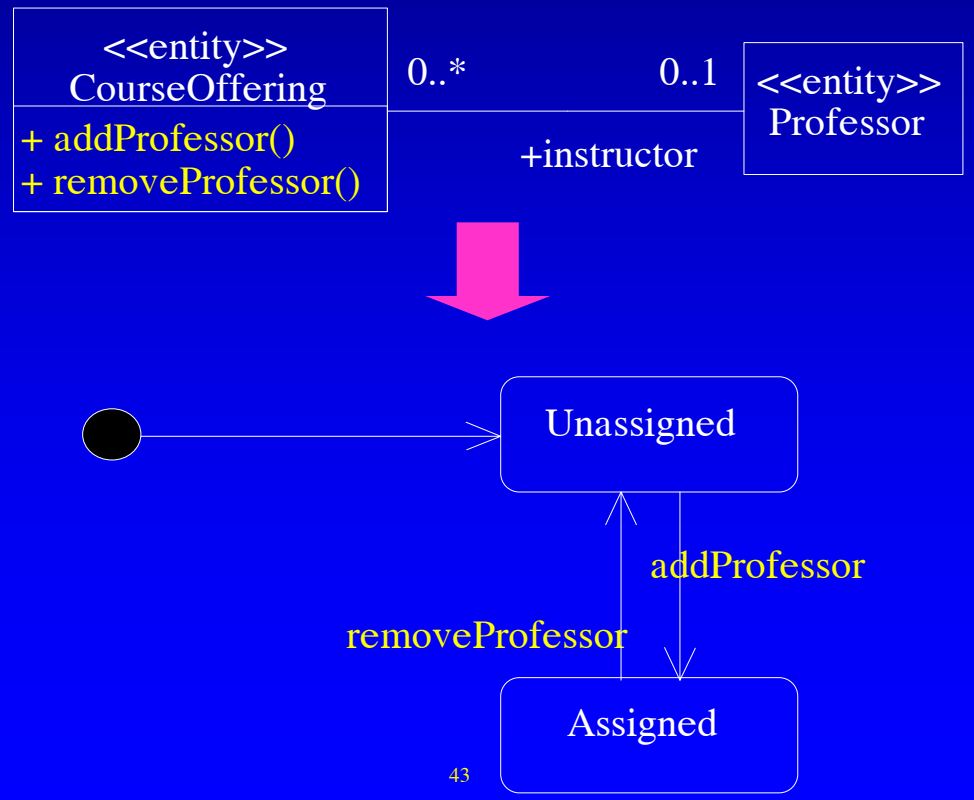
- Xem xét các class interface operation



Events: addProfessor, removeProfessor

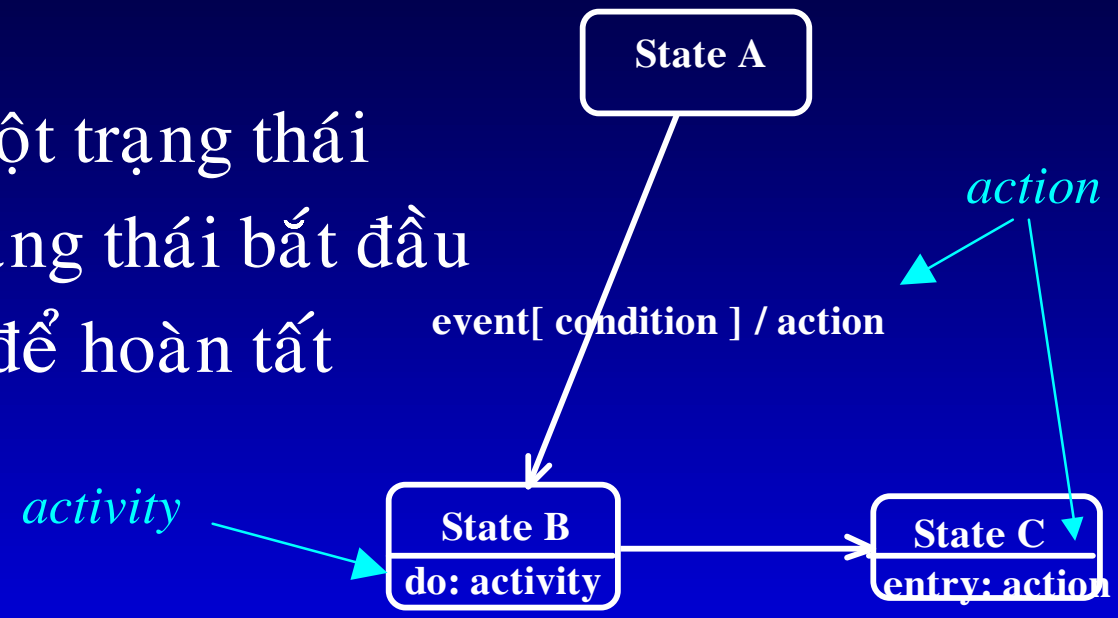
Xác định các Transition

- Với mỗi trạng thái, xác định events nào gây ra transitions đến trạng thái nào, bao gồm các điều kiện kiểm soát, nếu cần
- Transitions mô tả điều gì xảy ra khi đối tượng hồi đáp lại một event nhân được



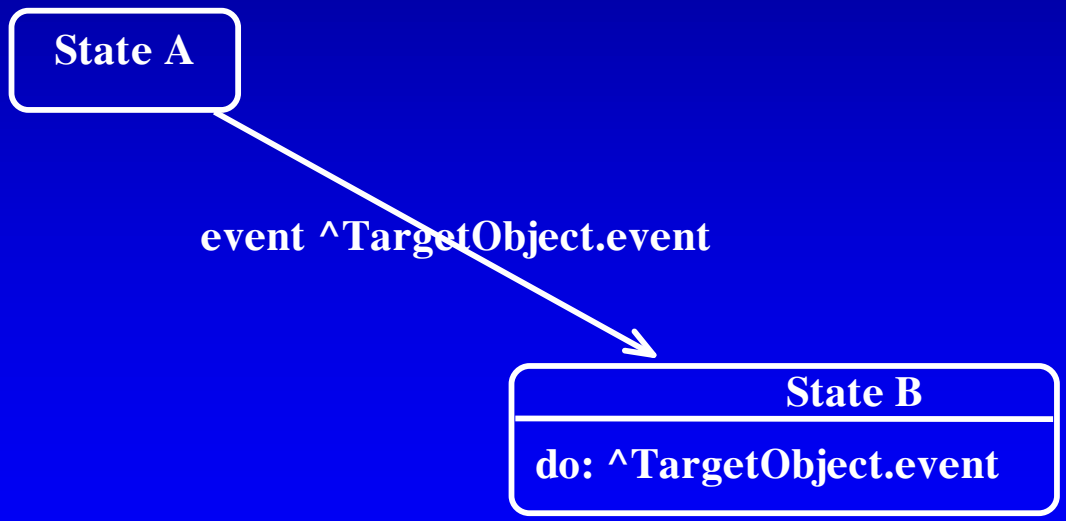
Thêm Activities và Actions

- Activities
  - Kết hợp với một trạng thái
  - Bắt dầu khi trạng thái bắt đầu
  - Cần thời gian để hoàn tất
  - Có thể ngắt
- Actions
  - Kết hợp với 1 transition
  - Cần thời gian không đáng kể để hoàn tất
  - Không thể ngắt ngang

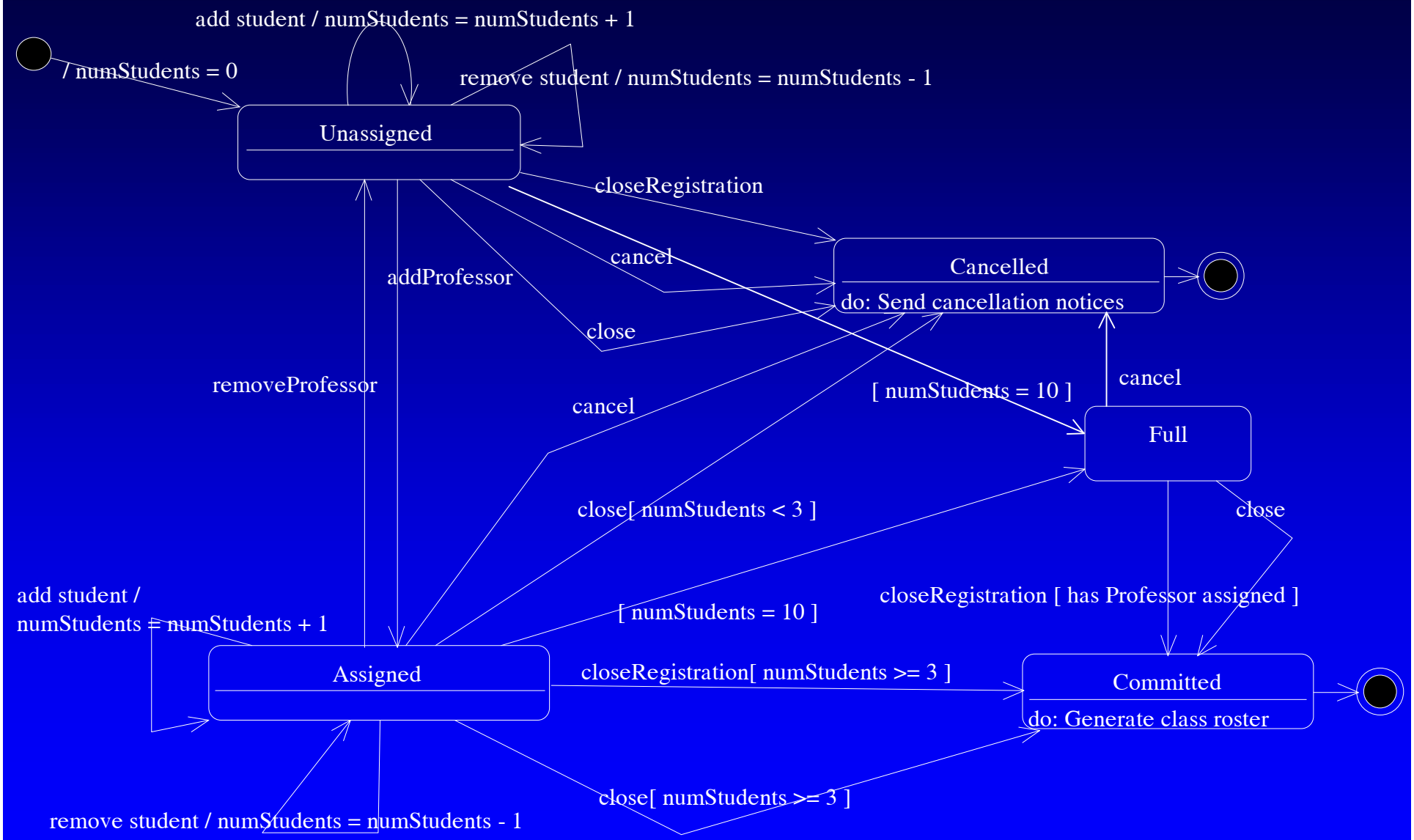


Gửi Events

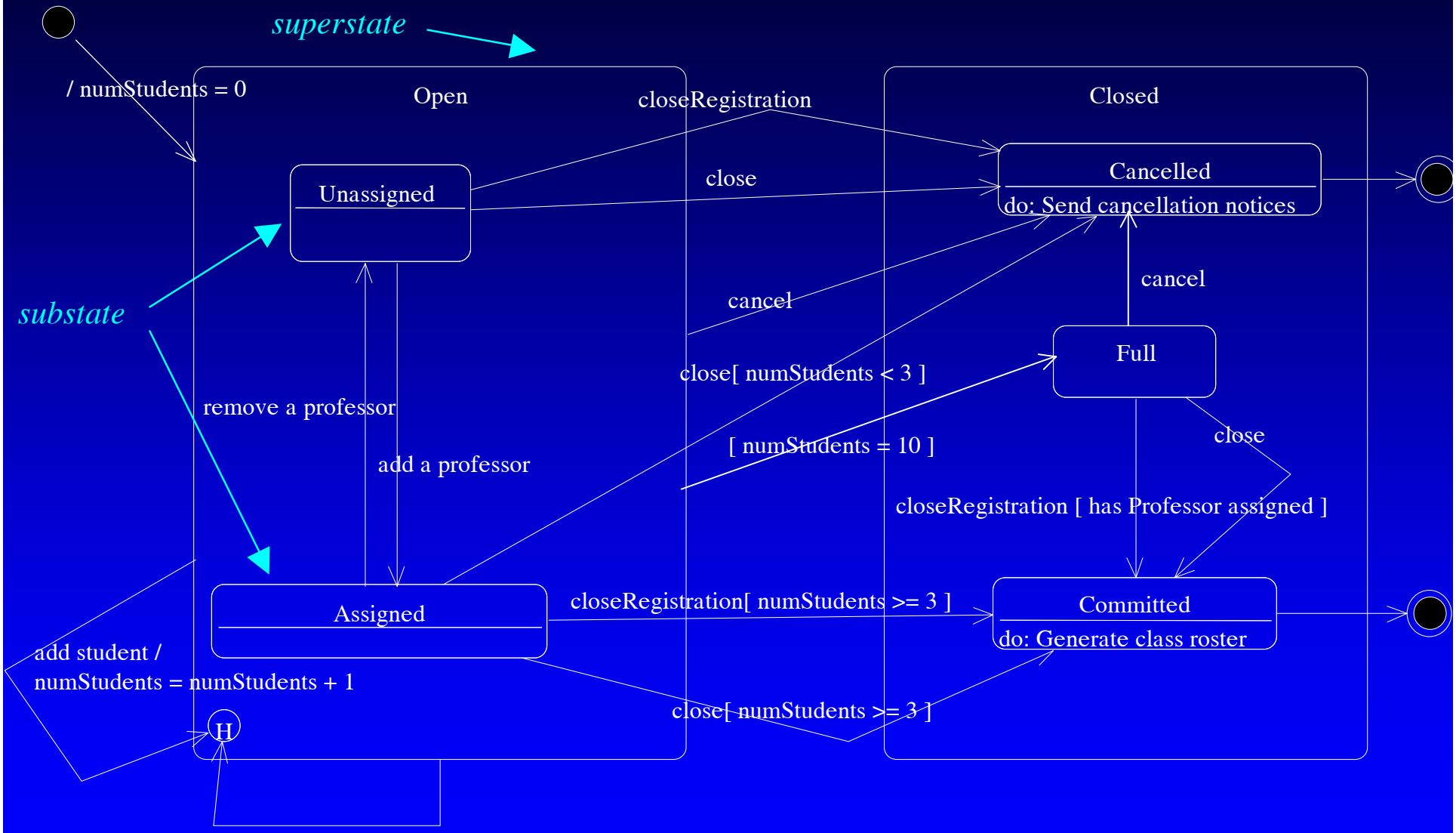
- Một event có thể gây ra việc gửi một event khác
- Một activity cùng có thể gửi event đến object khác



Ví dụ: Statechart



Ví dụ: Statechart với các trạng thái lồng nhau

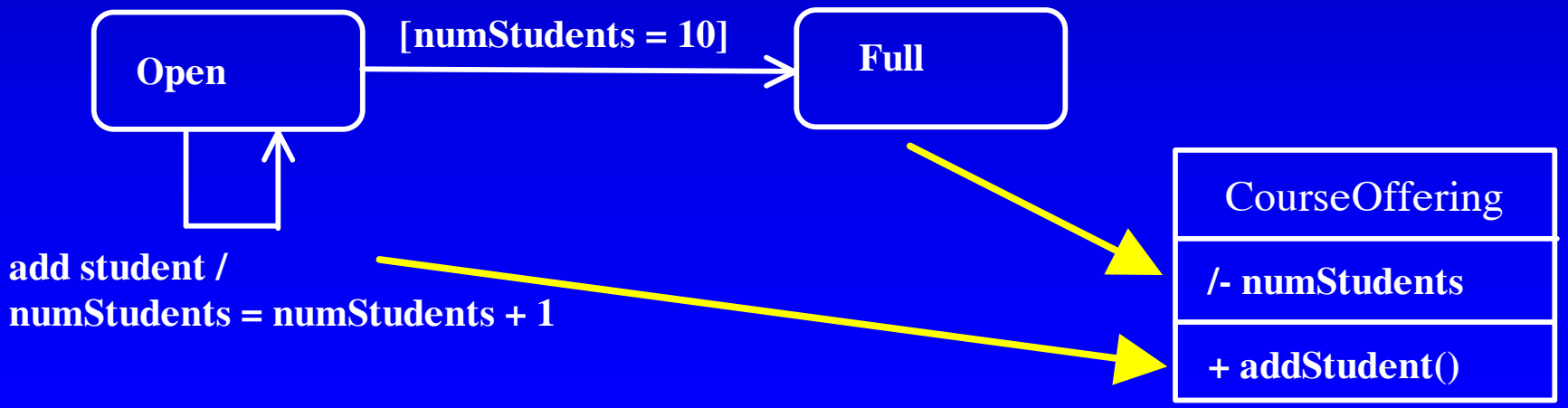


Những Object có Significant State?

- Các Object có vai trò thể hiện rõ bởi state transitions
- Các use case phức tạp là state-controlled
- Không cần mô hình hóa tất cả các object
  - Các Object dễ dàng cài đặt
  - Các Object không thuộc loại state-controlled
  - Các Object chỉ với một trạng thái

Cách Statecharts gắn với phần còn lại?

- Các Event biến thành các operation
- Các Method phải được cập nhật với các thông tin đặc thù cho các trạng thái
- Các trạng thái được biểu diễn bởi attributes
  - Chúng là input cho bước định nghĩa Attribute



(Stay tuned for derived attributes)

Bài tập: Định nghĩa States (optional)

- Hãy cho biết:
  - Tất cả các design classe
- Hãy xác định:
  - Các Class vứi significant state-controlled behavior
  - Các trạng thái và transitions quan trong của class
- Hãy xây dựng lược đồ:
  - Statechart của một class

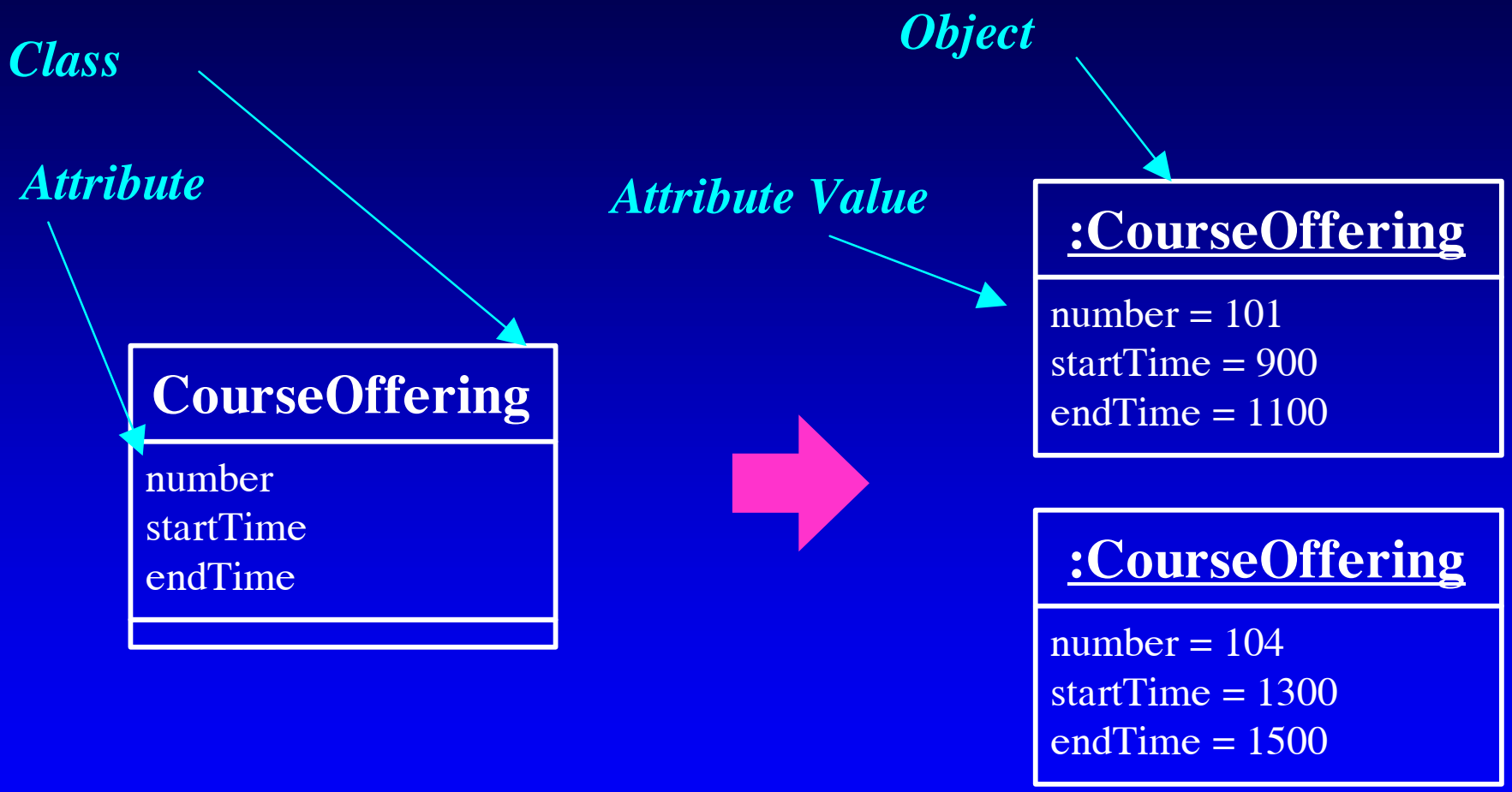
Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Định nghĩa Attributes

- Mục đích
  - Formalize definition of attributes
- 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

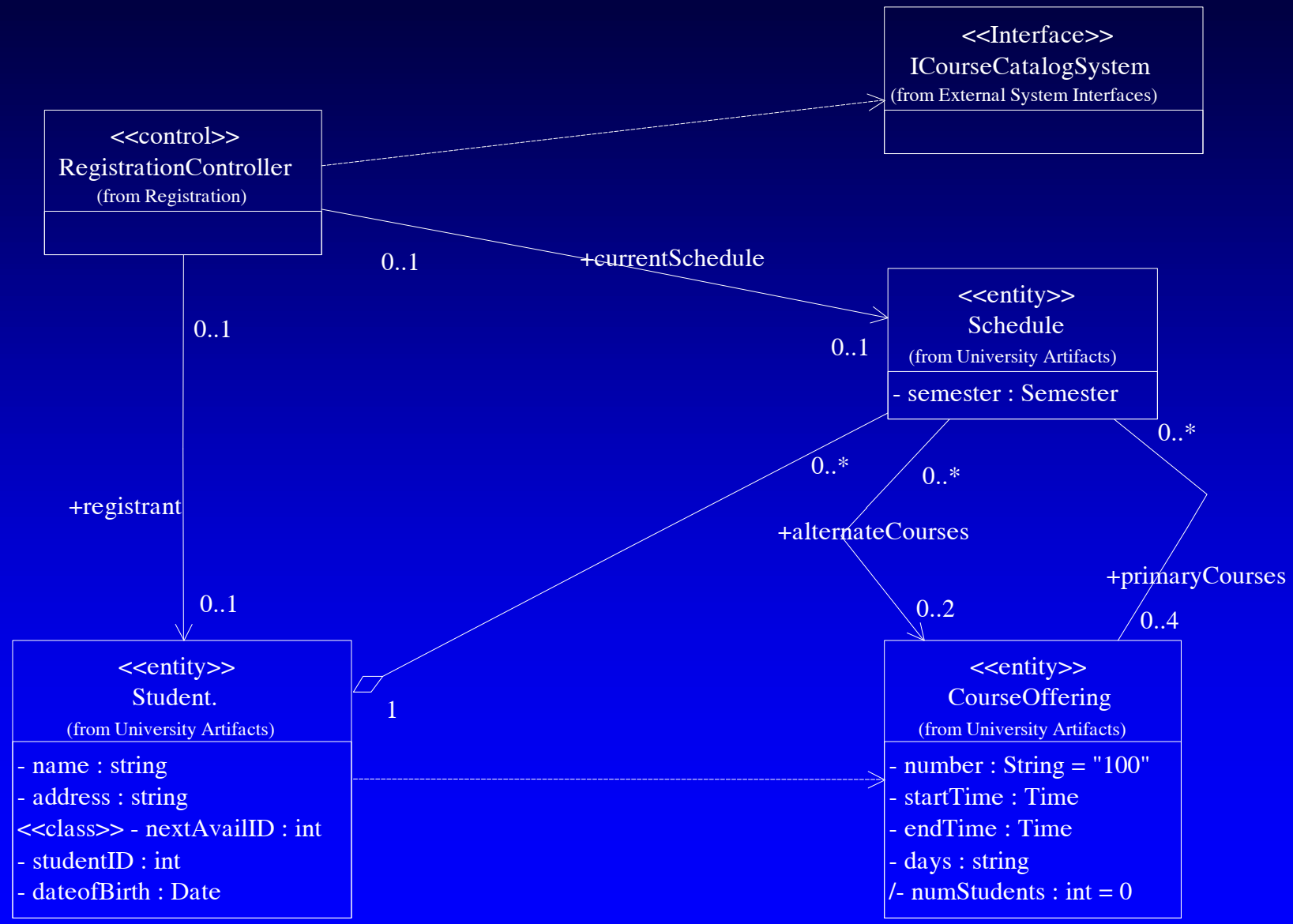
Biểu diễn Attribute

- 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 tình khả kiến
  - Public: ‘+’
  - Private: ‘-’
  - Protected: ‘#’

Các Derived Attribute

- Thế nào là derived attribute?
  - Một attribute mà giá trị co 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: Define Attributes



Bài tập: Define Attributes (optional)

- Hãy cho biết:
  - Các architectural layers, các package và phụ thuộc của chúng
  - Các Design class của một use case cụ thể

(còn tiếp)

Bài tập: Define Attributes (tt.)

- Với các design class hãy xác định:
  - Các Attribute và mô tả đầy đủ của nó
  - Tầm vực và tính khả kiến của Attribute
  - Mọi mối quan hệ và class bổ sung để hỗ trợ cho việc định nghĩa các attribute và attribute signatures

(còn tiếp)

Bài tập: Define Attributes (cont.)

- Xây dựng lược đồ:
  - VOPC class diagram, chứa tất cả các attribute và mối quan hệ

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

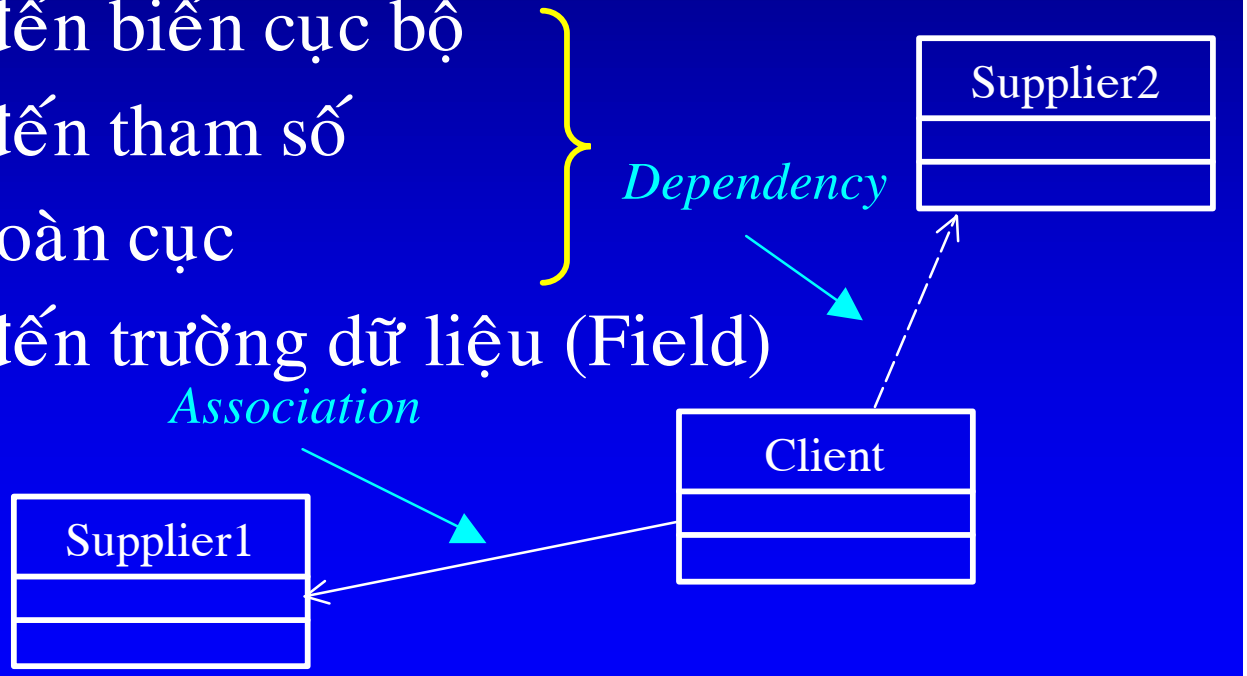
Định nghĩa Dependency

- 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

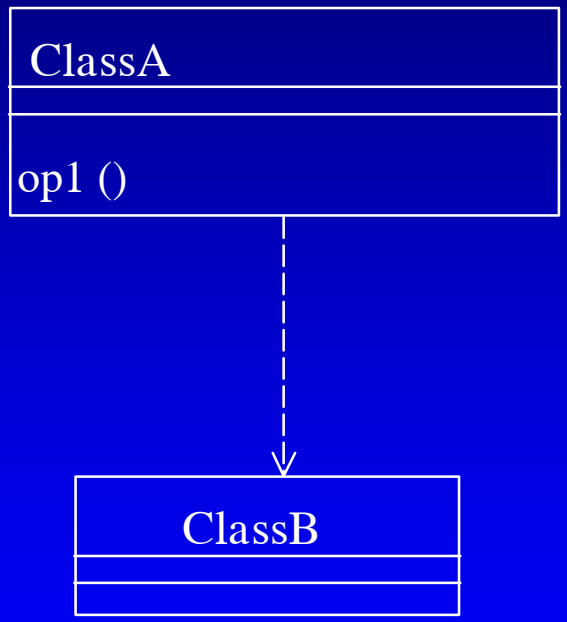
So sánh Dependency và Association

- Association là quan hệ câu 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)



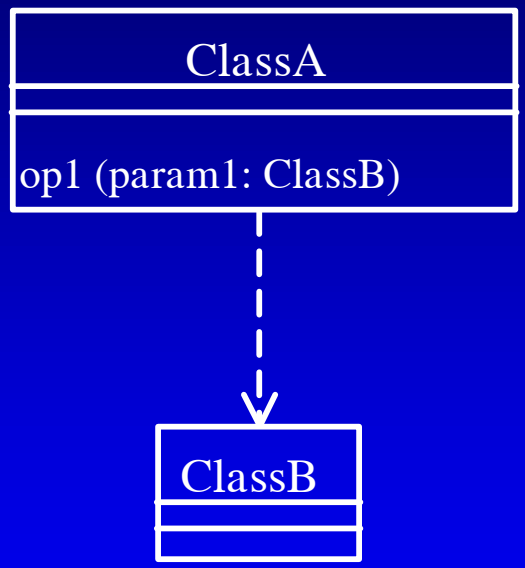
Local Variable Visibility

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



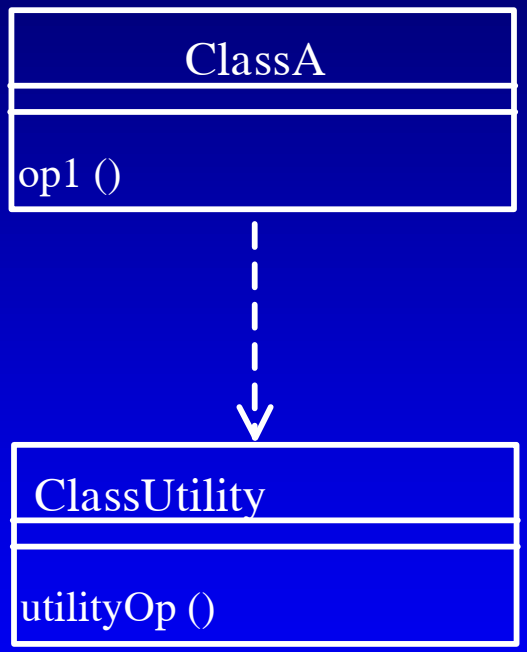
Parameter Visibility

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



Global Visibility

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

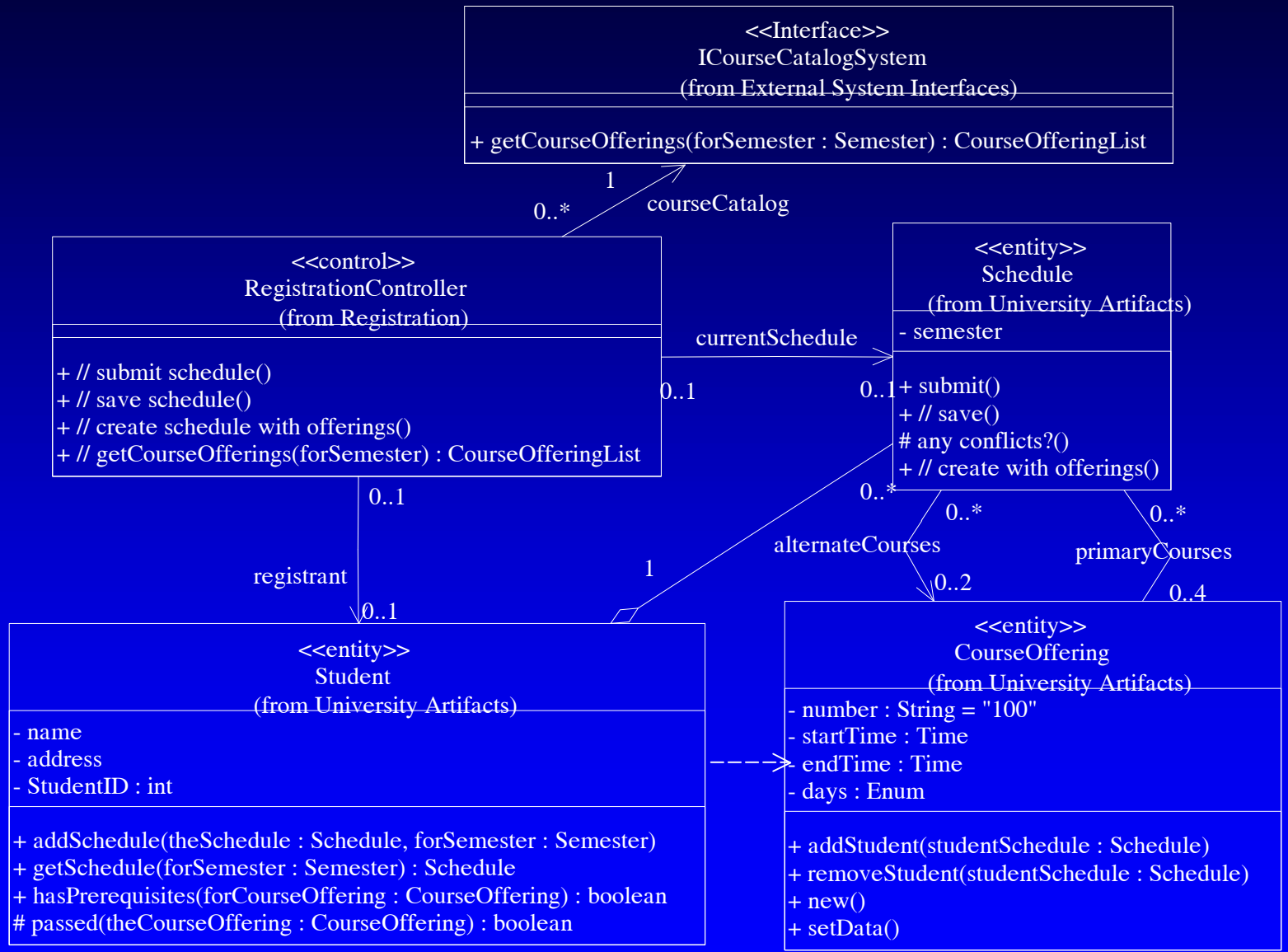


Xác định Dependency

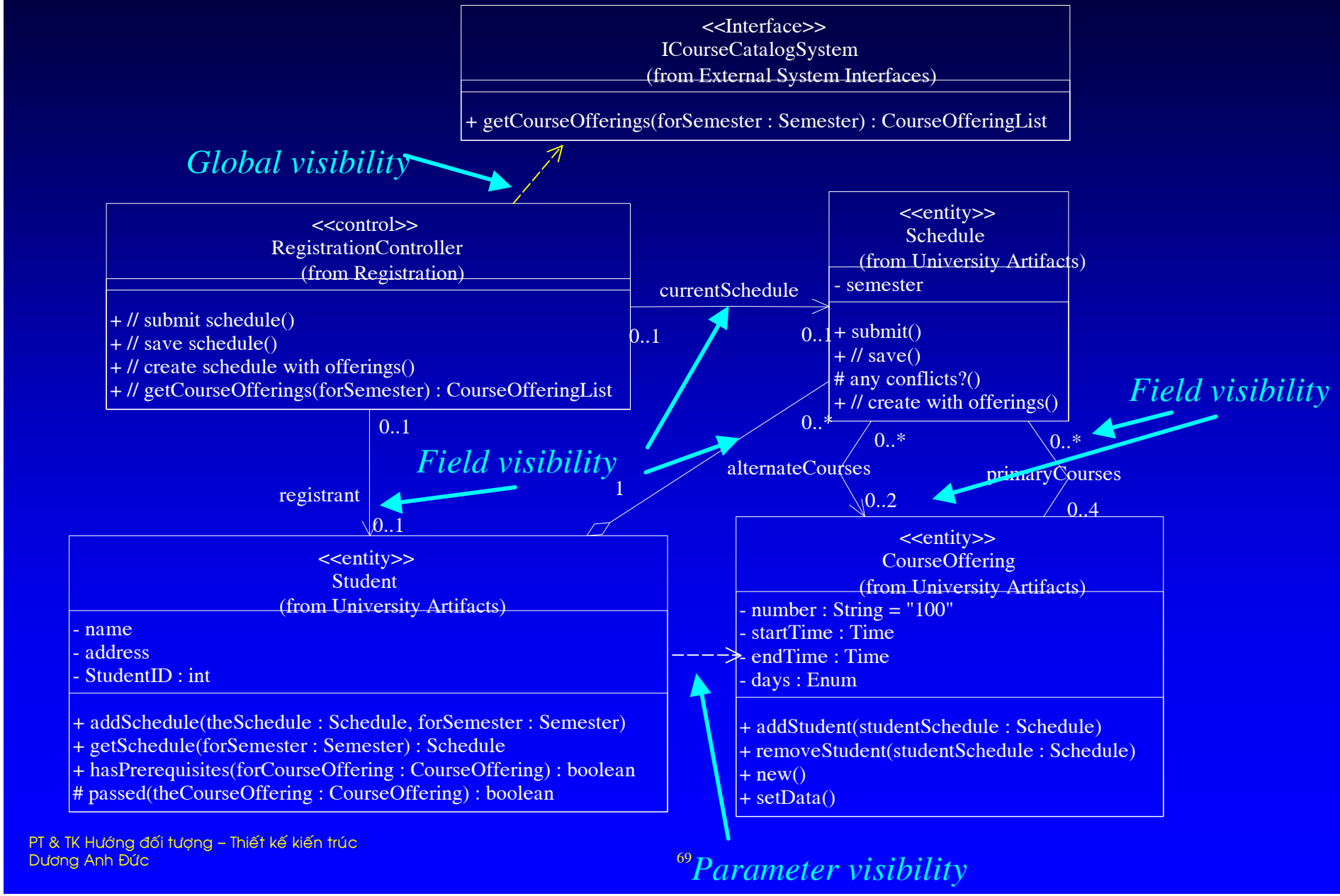
- Bám vào các quan hệ trong thế giới thực
- Bám váo các quan hệ yếu nhất (dependency)
- Quan hệ là "thường trực"? Dùng association (field visibility)
- Quan hệ là "nhất thời"? Dùng dependency
  - Nhiều object chia sẻ chung 1 instance
    - Truyền instance như tham số (parameter visibility)
    - Đặt 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

Chọn lựa nào đúng? CÒN TÙY

Ví dụ: Trước khi định nghĩa Dependency



Ví dụ: Sau khi định nghĩa Dependency



Các bước thiết kế Class

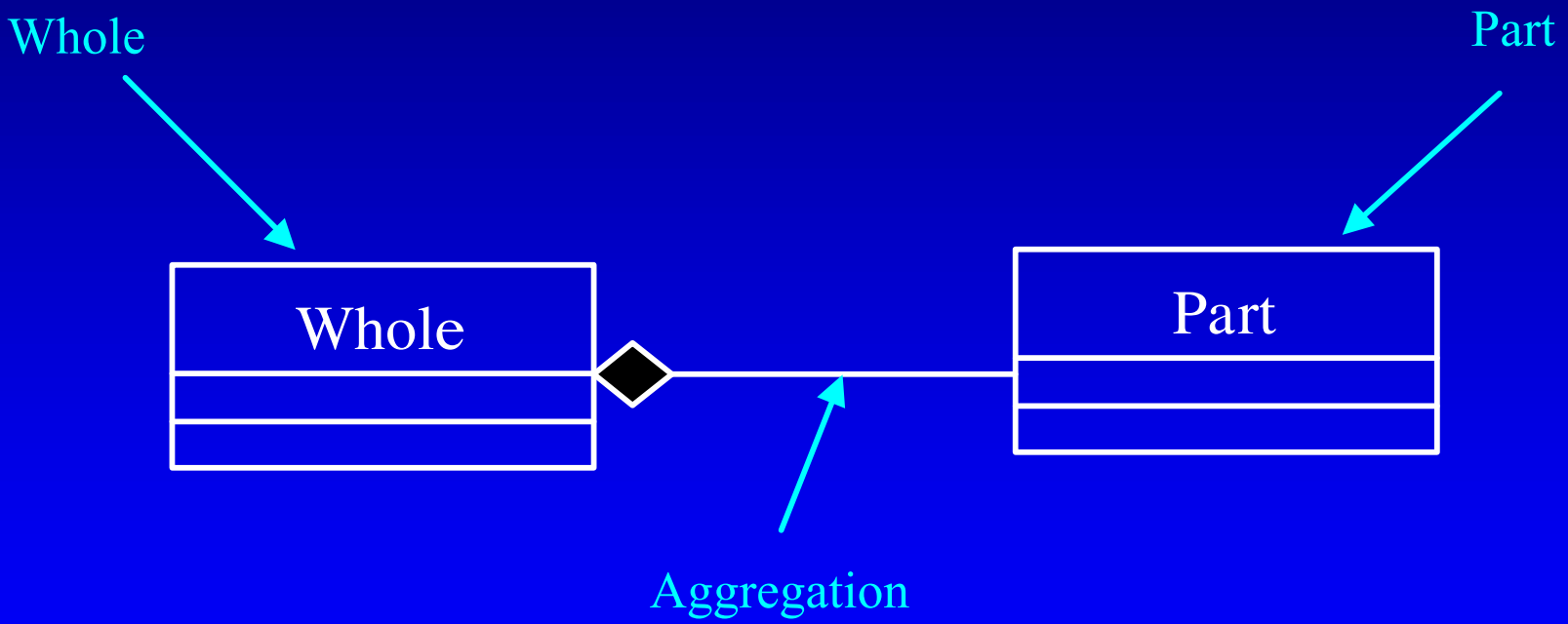
- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Định nghĩa Associations

- 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)

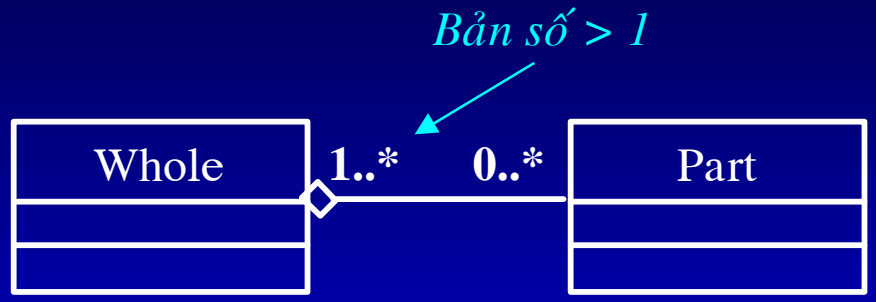
Nhắc lại: Composition là gì ?

- 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"

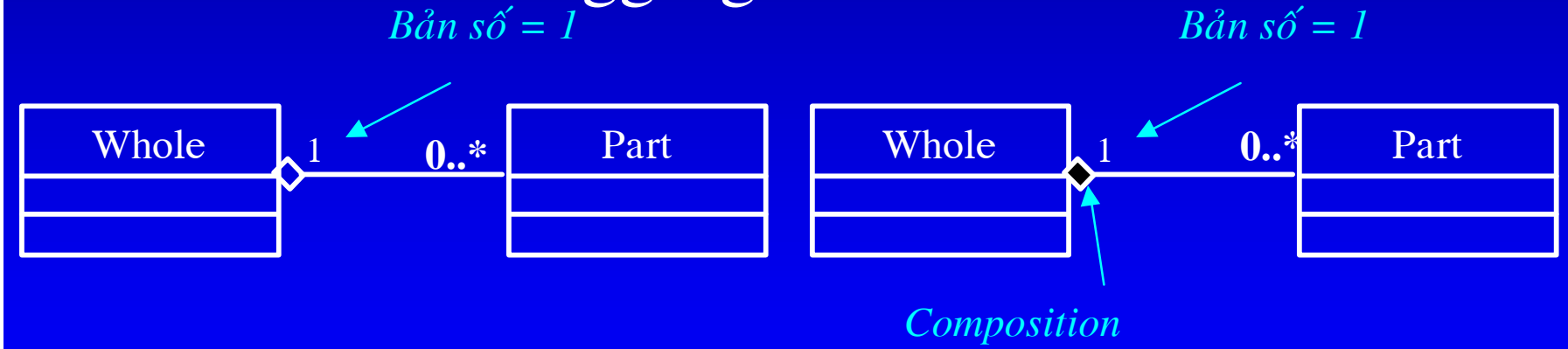


Aggregation: Shared hay không shared

- Shared Aggregation



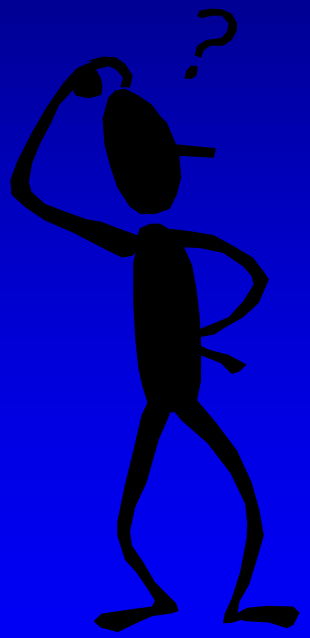
- Non-shared Aggregation

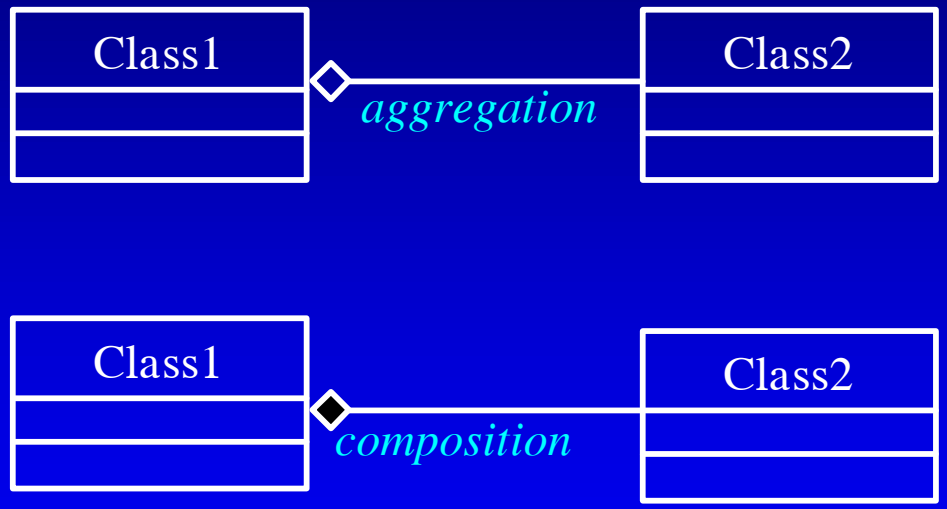


Theo định nghĩa, composition là non-shared aggregation

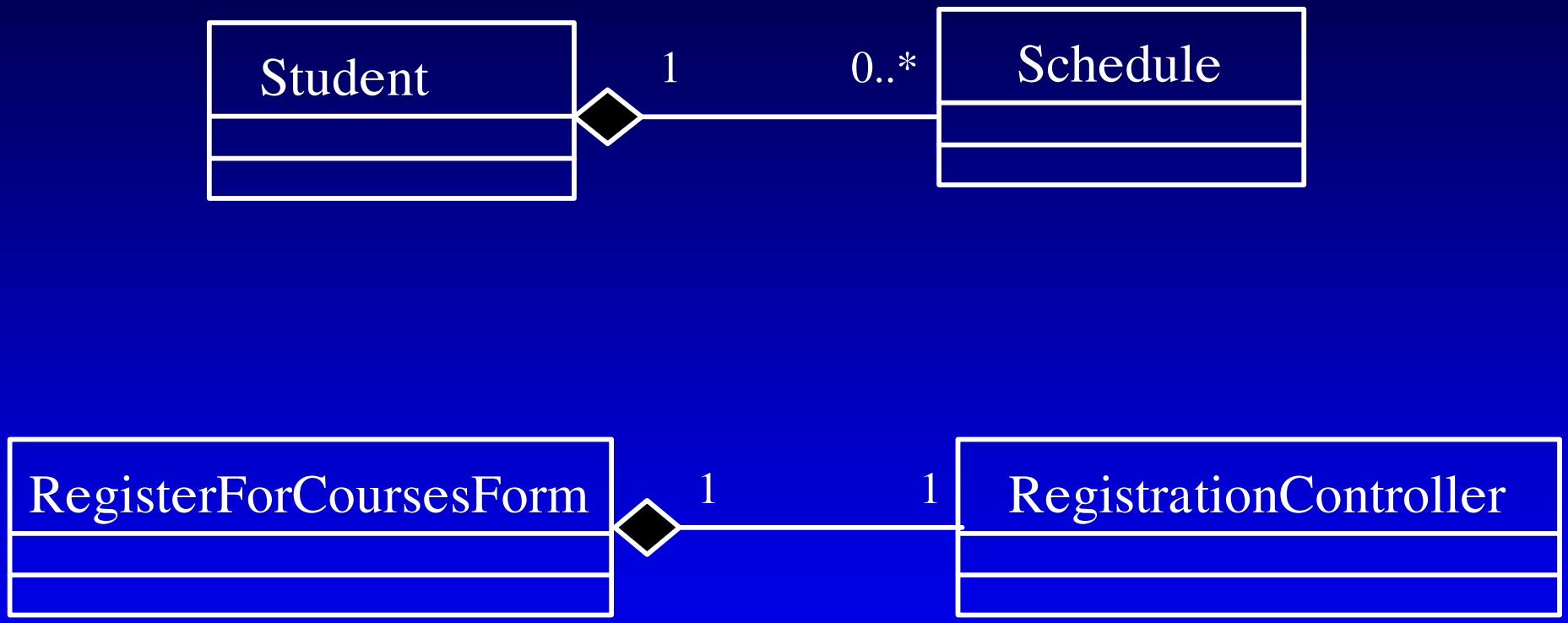
Aggregation hay Composition?

- Xem xét
  - Chu kỳ sống của Class1 và Class2





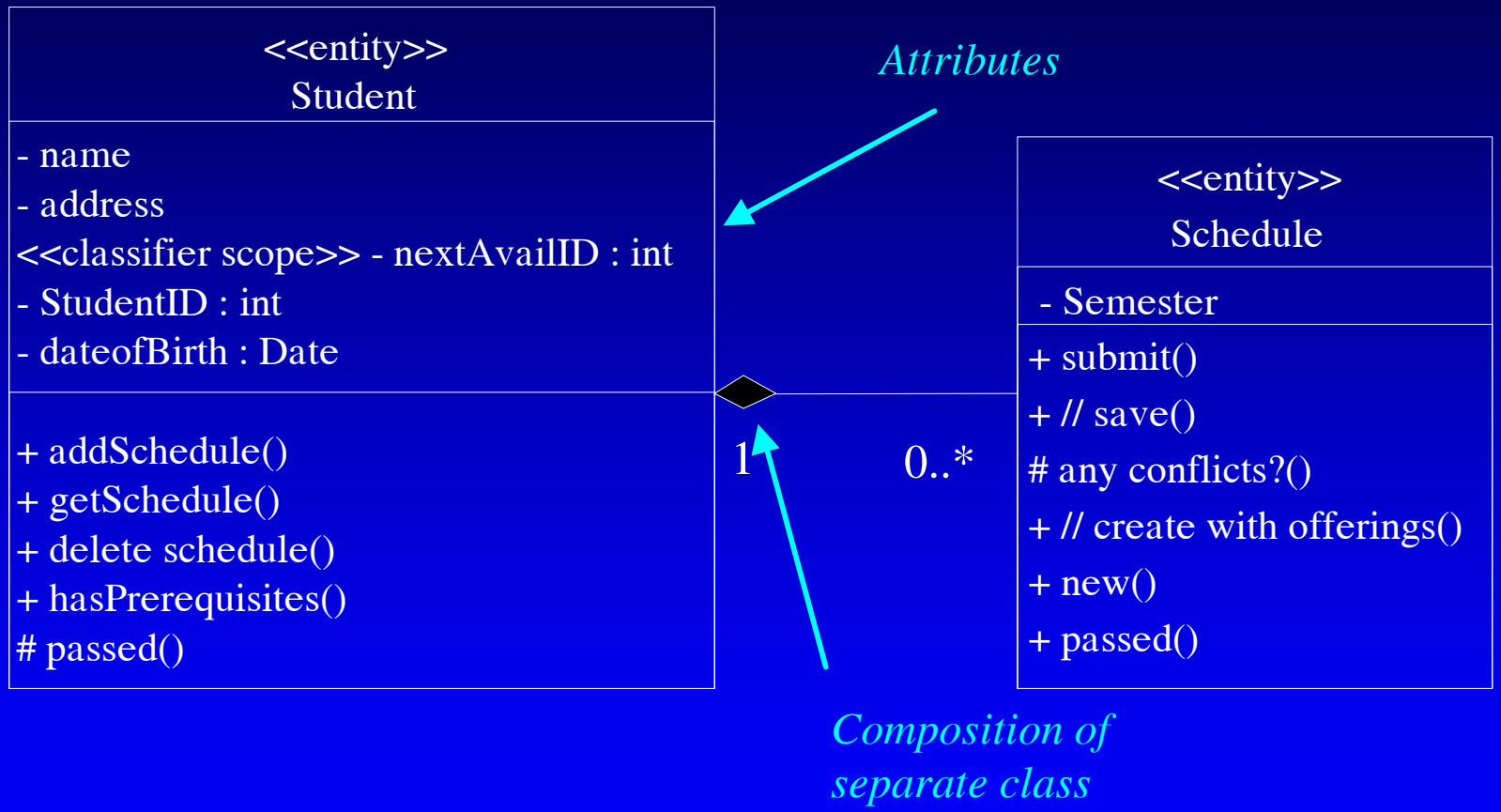
Ví dụ: Composition



Cân nhắc giữa Attributes và Composition

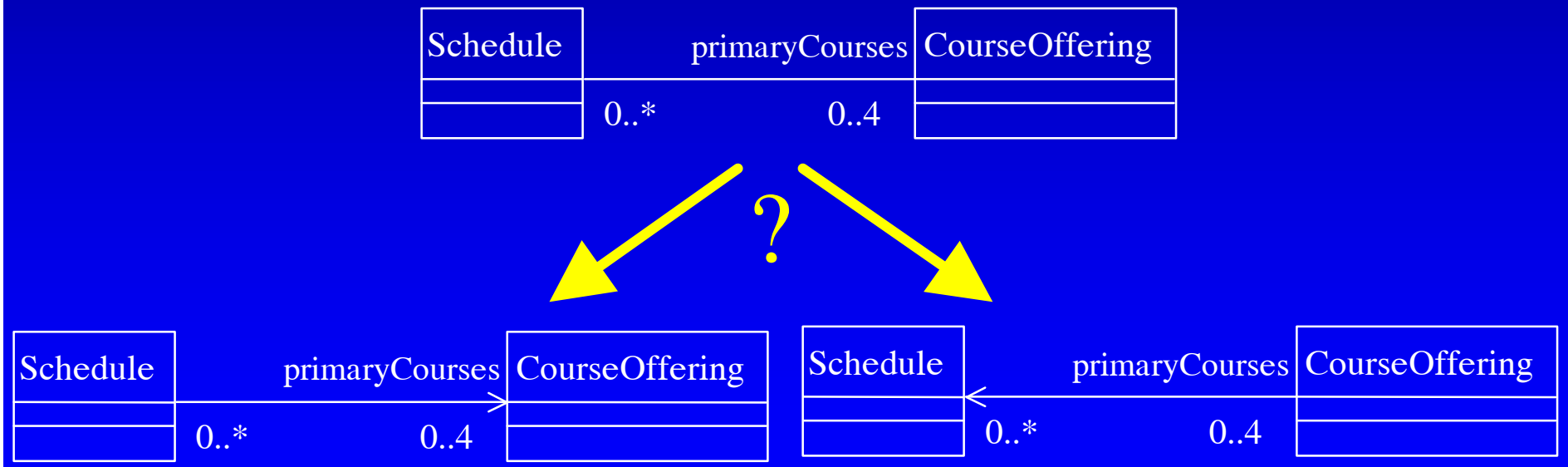
- Dùng composition khi
  - Các thuộc tính cần được nhận dạng độc lập
  - Nhiều class có chung các thuộc tính
  - Các thuộc tính có cấu trúc phức tạp và bản thân chúng cũng có thuộc tính riêng
  - Các thuộc tính có hành vi riêng (phức tạp)
  - Các thuộc tính có quan hệ riêng
- Các trường hợp còn lại dùng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

Ví dụ: Attributes/Composition



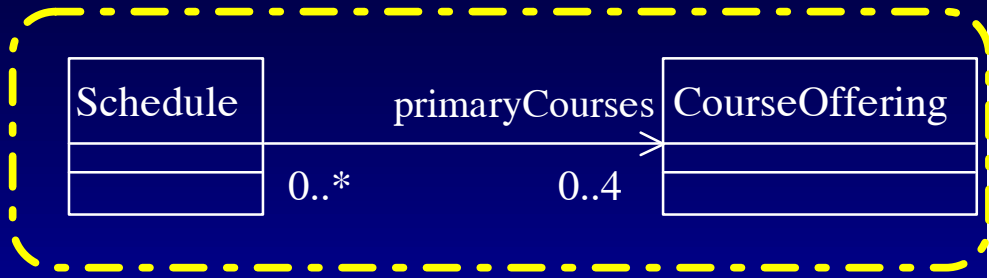
Chiều của quan hệ

- 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

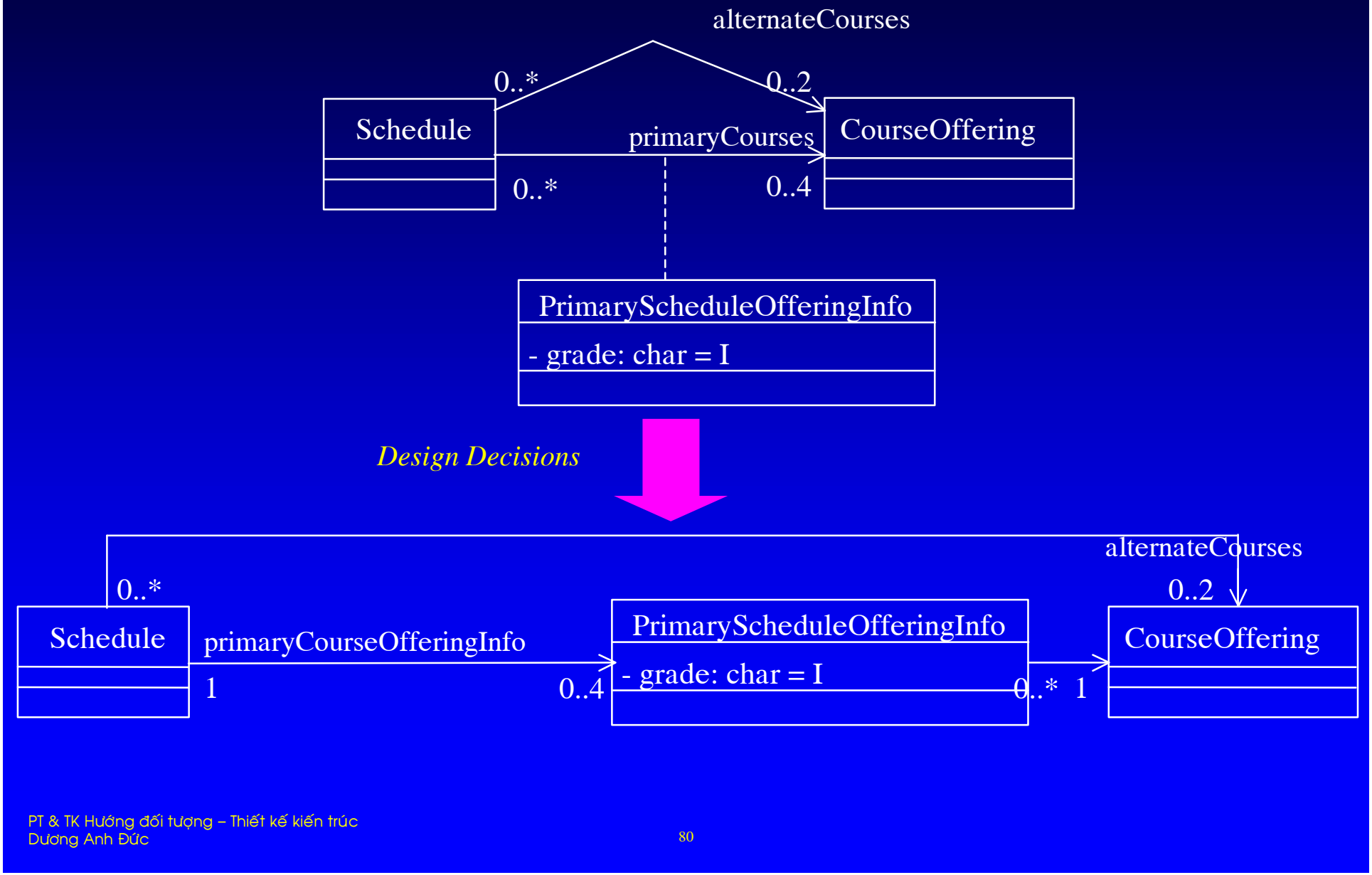
- Tổng số Schedule nhỏ, hay
- Không bao giờ cần một list Schedule có CourseOffering xuất hiện trên đó



- 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

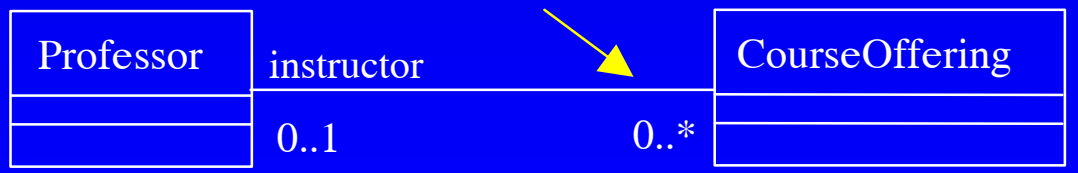
Ví dụ: Thiết kế Association Class



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

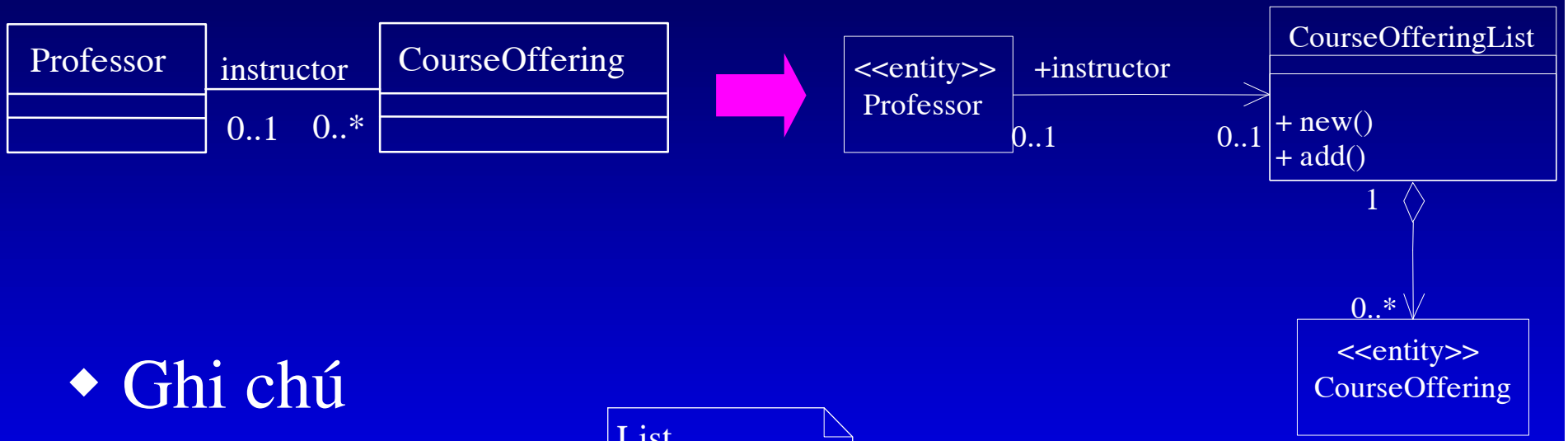
- Bản số > 1
  - Không thể dùng giá trị đơn hay pointer
  - Cần thực hiện thêm một số công việc thiết kế



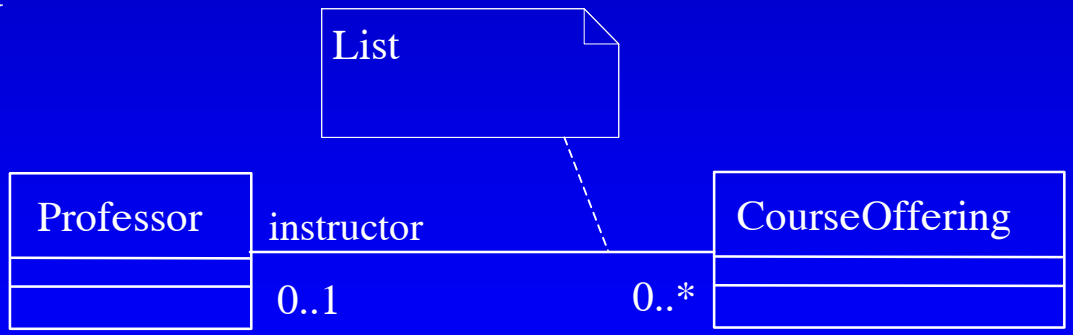
Cần một container

Multiplicity Design Options

- Mô hình hóa tường minh một container class

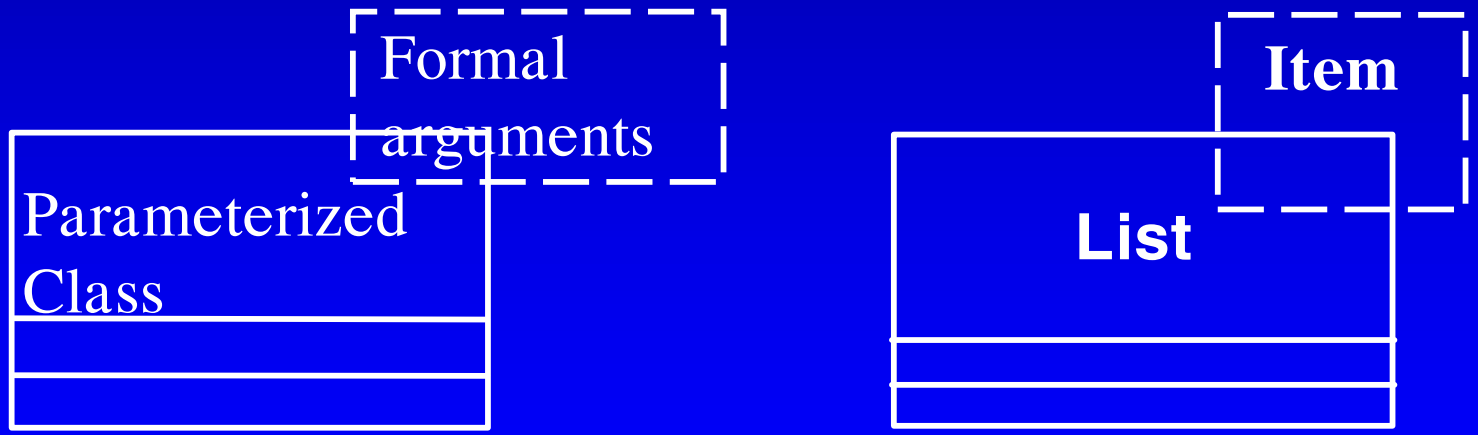


- Ghi chú



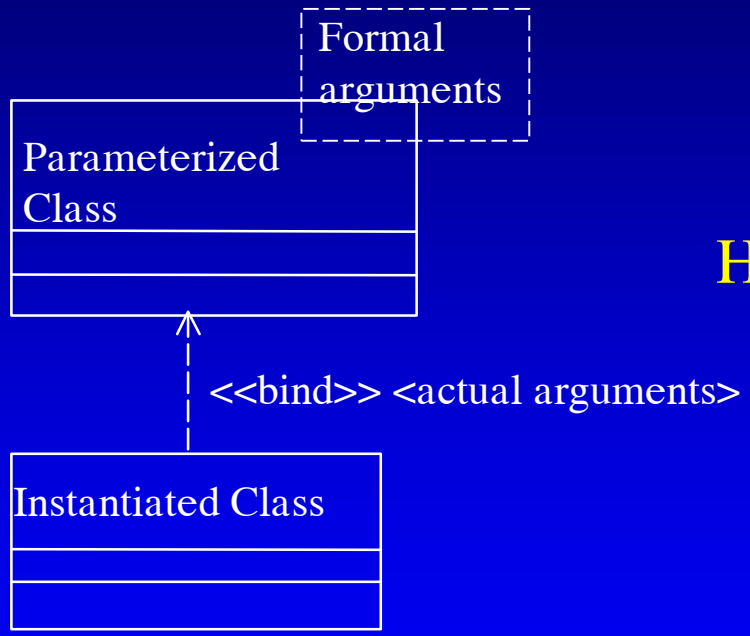
Parameterized Class (template) là gì?

- Là một class dùng để định nghĩa các class khác
- Thường dùng cho các container class
  - Một số container class thông dụng:
    - Set, list, dictionary, stack, queue, …



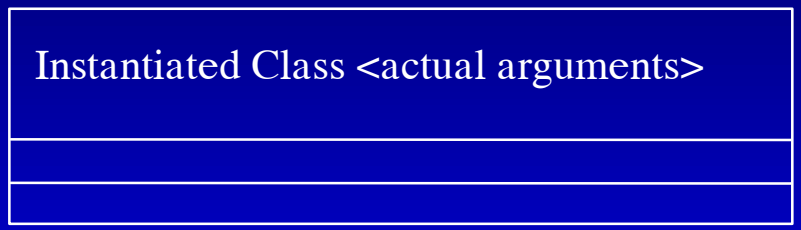
Thể hiện của Parameterized Class

Kết buộc tường minh



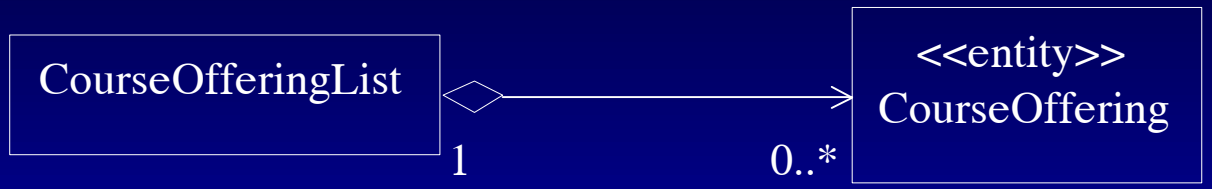
HAY

Kết buộc ẩn



Ví dụ: Thể hiện của Parameterized Class

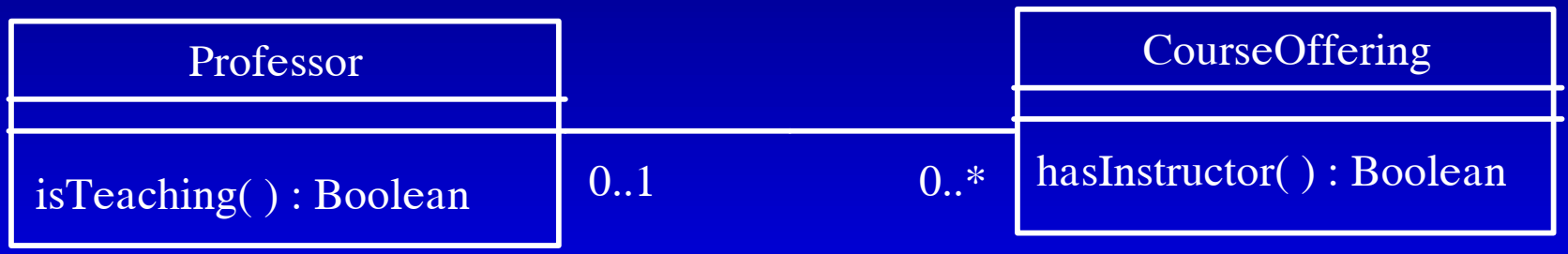
Trước



Sau

Multiplicity Design: Optionality

- Nếu một link là tùy chọn, hãy cèn thêm một operation để kiểm tra sự tồn tại của link



Bài tập: Đ/n Dependency và Association

- Hãy cho biết:
  - Use-case realization của 1 use case và chi tiết thiết kế của 1 subsystem
  - Thiết kế của tất cả các design element

(còn tiếp)

Bài tập: Đ/n Dependenc và Association (tt.)

- Hãy xác định:
  - Chiều của mỗi quan hệ
  - Mọi class cần bổ sung để hỗ trợ cho việc thiết kế quan hệ
  - Mọi association được tinh chỉnh thành dependency
  - Mọi association được tinh chỉnh thành aggregation hoặc composition
  - Mọi tinh chỉnh liên quan đến bản số

(còn tiếp)

Bài tập: Đ/n Dependency và Association (tt.)

- Xây dựng lược đồ
  - Một bản cập nhật của VOPC, bao gồm cả các tinh chỉnh quan hệ

Các bước thiết kế Class

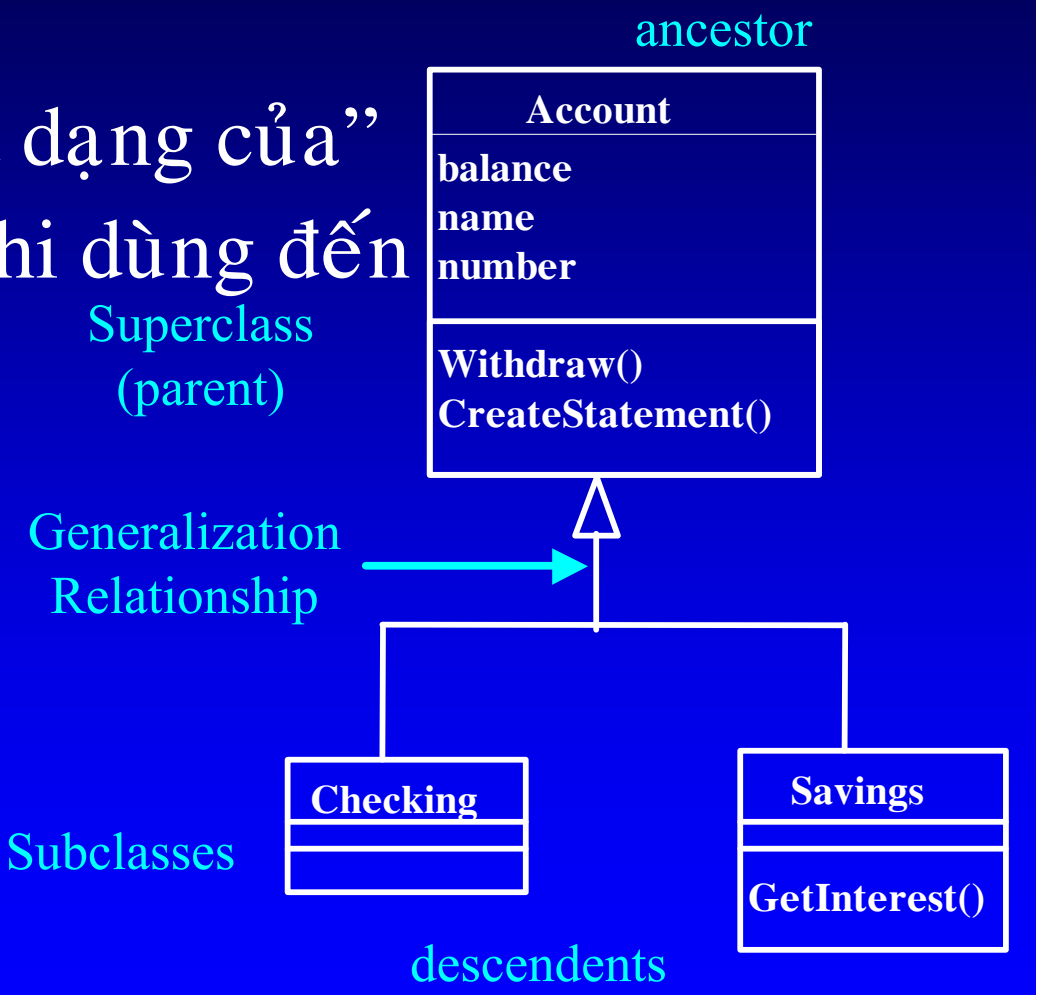
- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

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

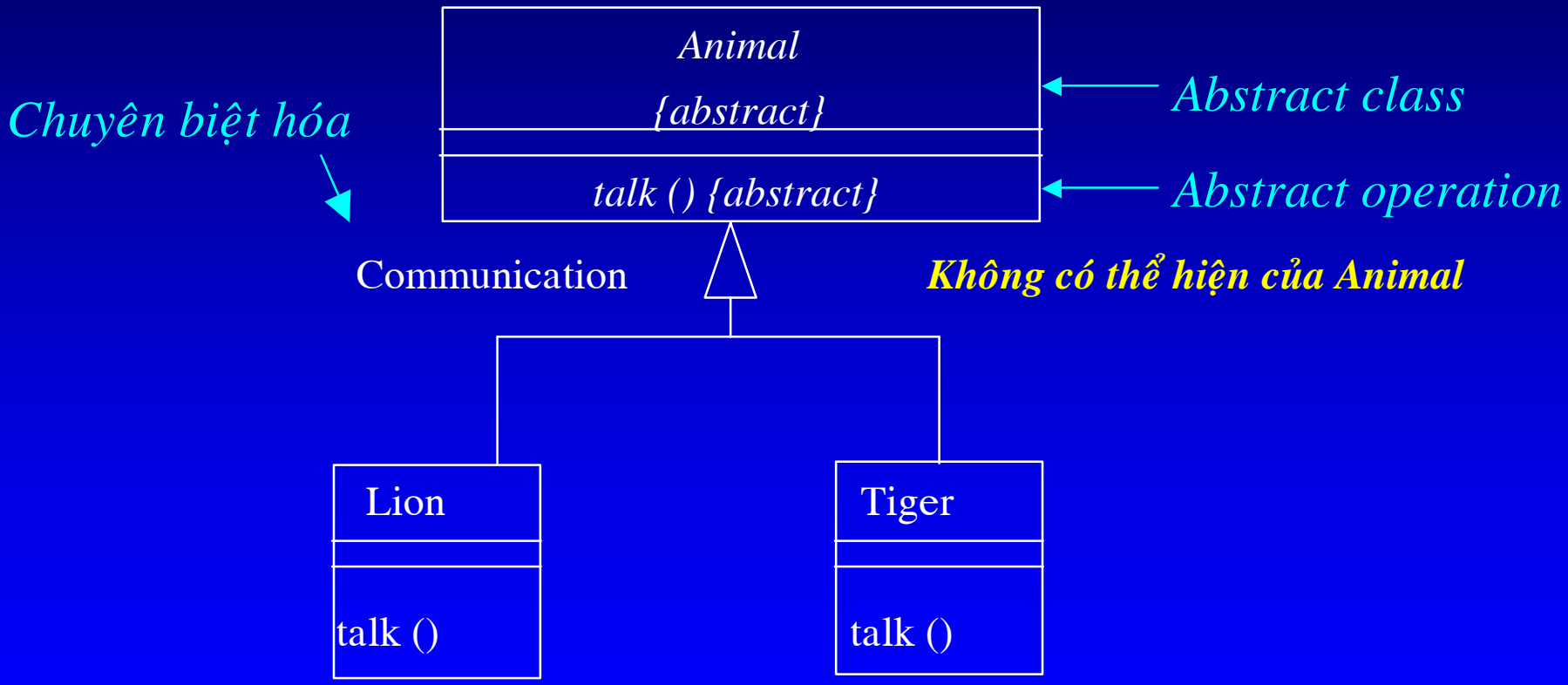
Nhắc lại: Generalization

- Một class chi 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



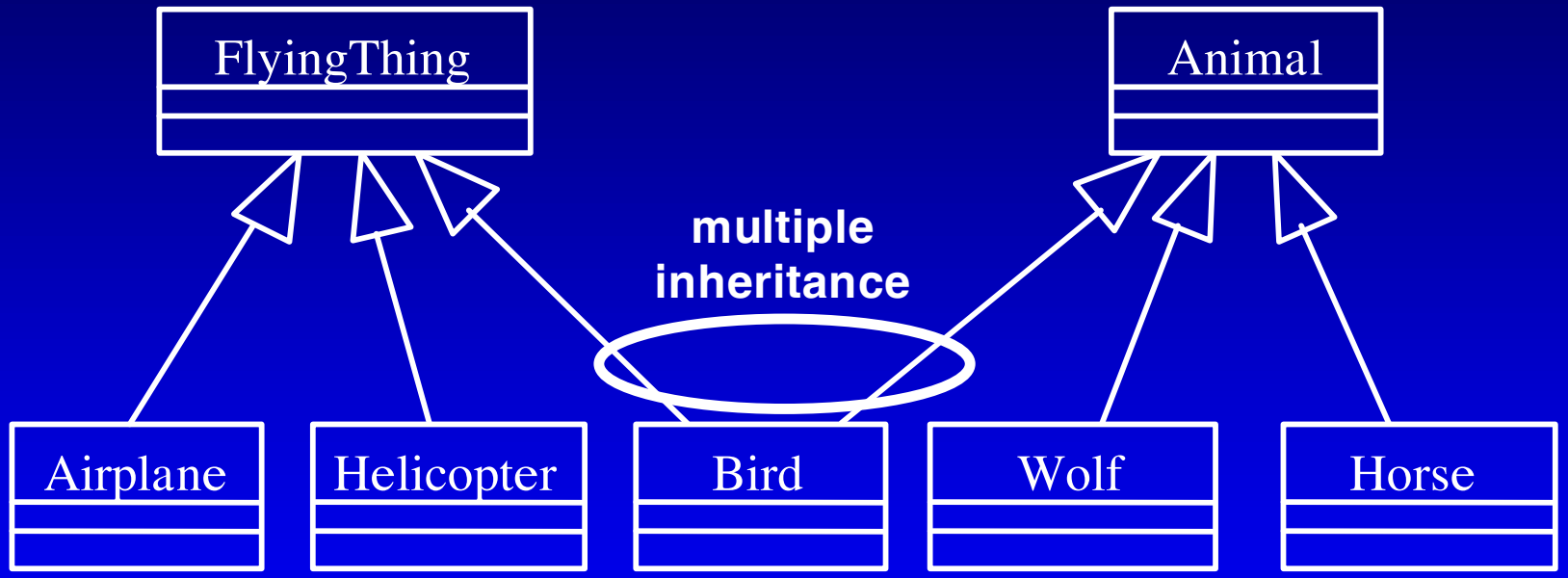
Abstract và Concrete Class

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



Nhắc lại: Đa kế thừa

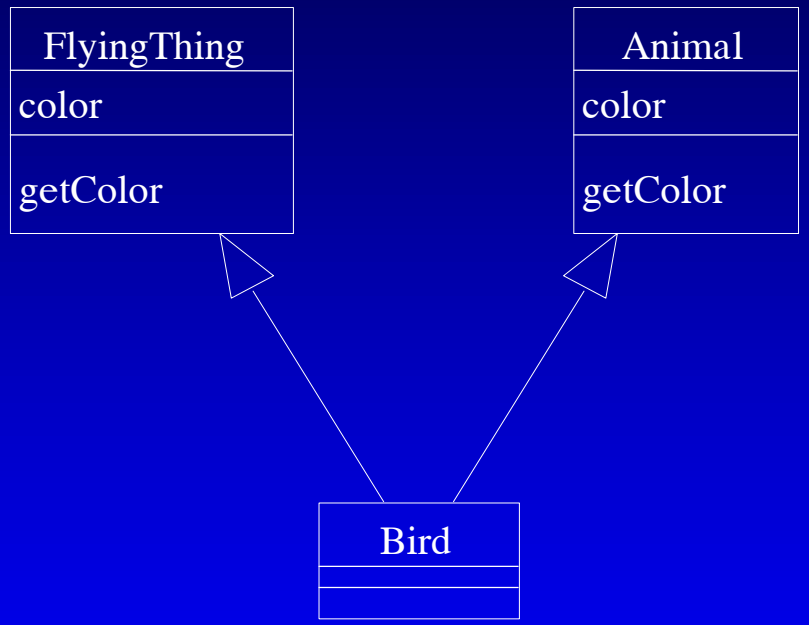
- Một class có thể kế thừa từ nhiều class



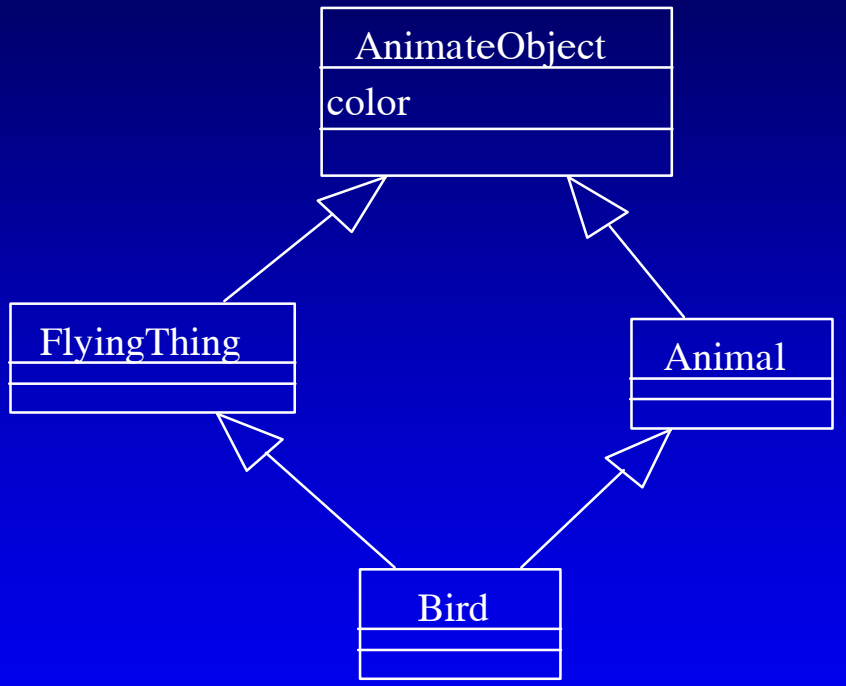
Dùng đa kế thức chỉ khi thật cần thiết, và phải luôn cẩn thận !

Các vấn đề của đa kế thừa

Tên của attribute hay operation bị trùng



Lặp lại việc kế thừa

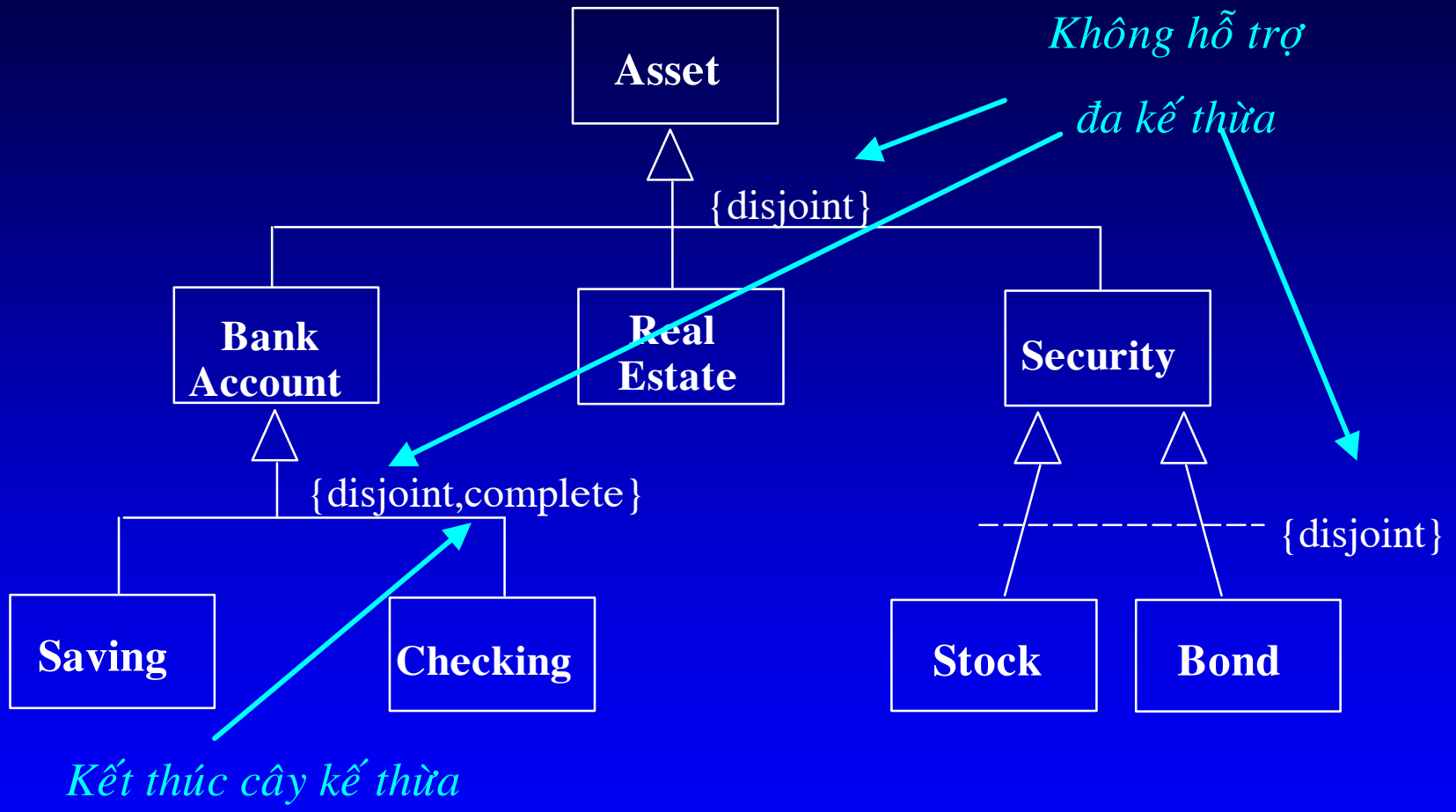


Lời giải của các vấn đề trên phụ thuộc cài đặt cụ thể

Các ràng buộc của quan hệ tổng quát hóa

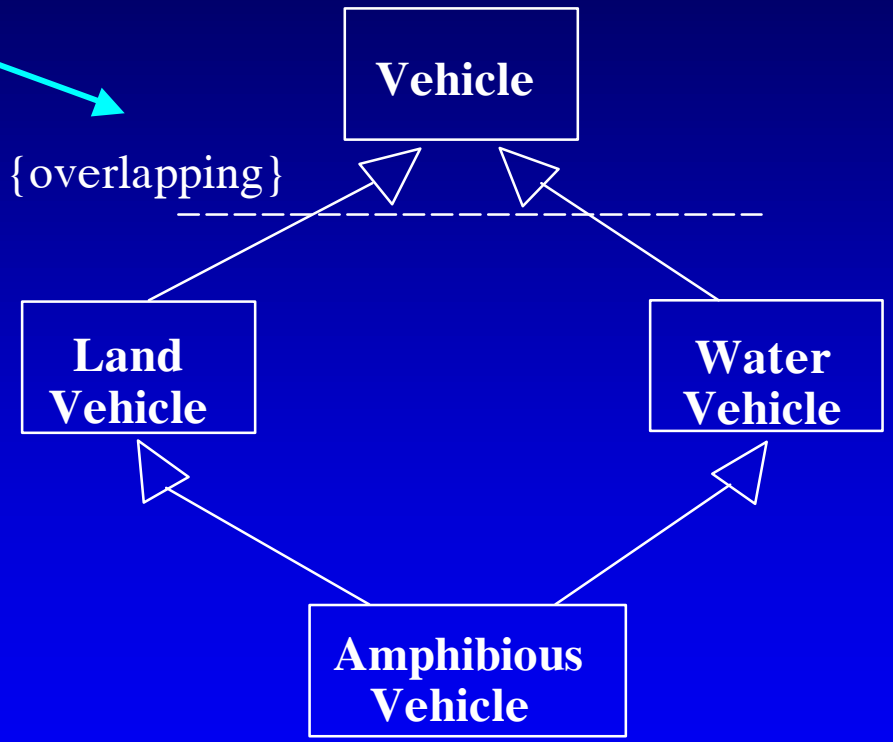
- Complete (Hoàn chỉnh)
  - Kết thúc toàn bộ cây kế thừa trong thiết kế
- Incomplete (Không hoàn chỉnh)
  - Cây kế thừa có thể mở rộng
- Disjoint (Phân tách)
  - Các Subclass loại trừ lần nhau
  - Không hỗ trợ đa kế thừa
- Overlapping (Chồng lắp)
  - Các Subclass không loại trừ lẫn nhau
  - Hỗ trợ đa kế thừa

Ví dụ: Generalization Constraints



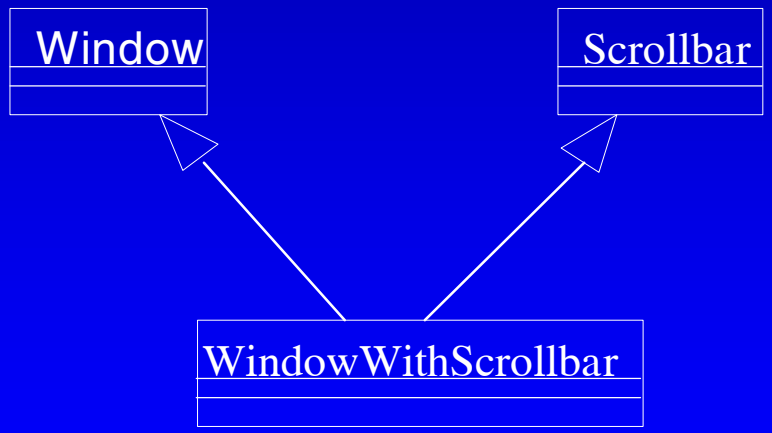
Ví dụ: Generalization Constraints (tt.)

Hỗ trợ đa kế thừa



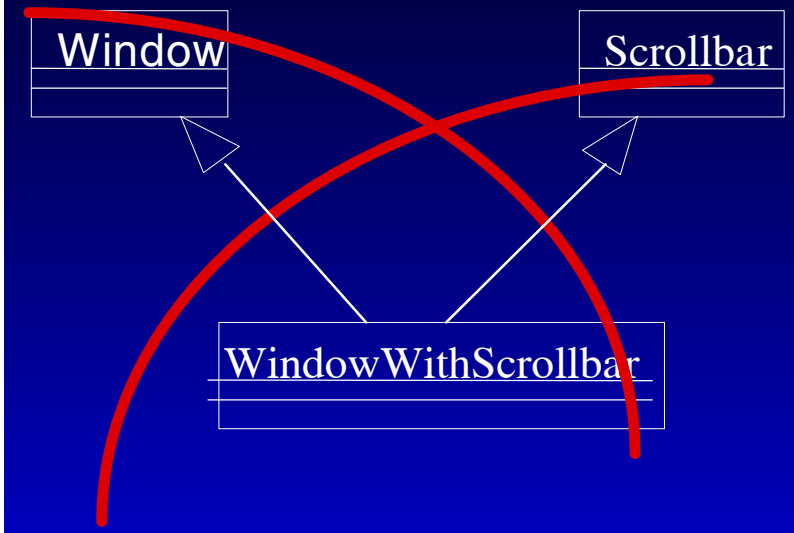
Chọn Generalization hay Aggregation

- 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

Sử dụng quan hệ tổng quát hóa

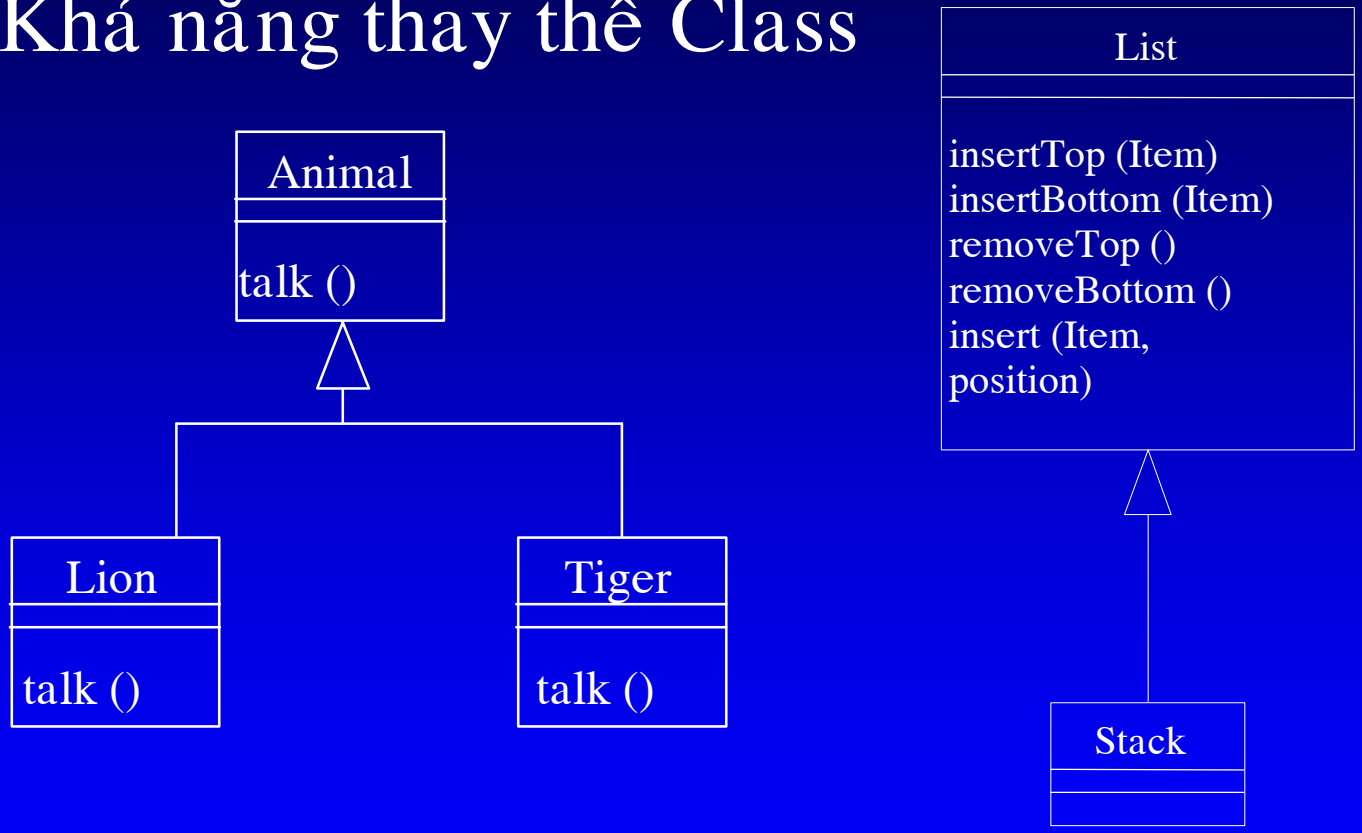
- Chia sẻ các thuộc tính và hành vi chung
- Chia sẻ cài đặt
- Cài đặt cơ chế Polymorphism
- Cài đặt cơ chế Metamorphosis

Sử dụng quan hệ tổng quát hóa

- Chia sẻ các thuộc tính và hành vi chung
- Chia sẻ cài đặt
- Cài đặt cơ chế Polymorphism
- Cài đặt cơ chế Metamorphosis

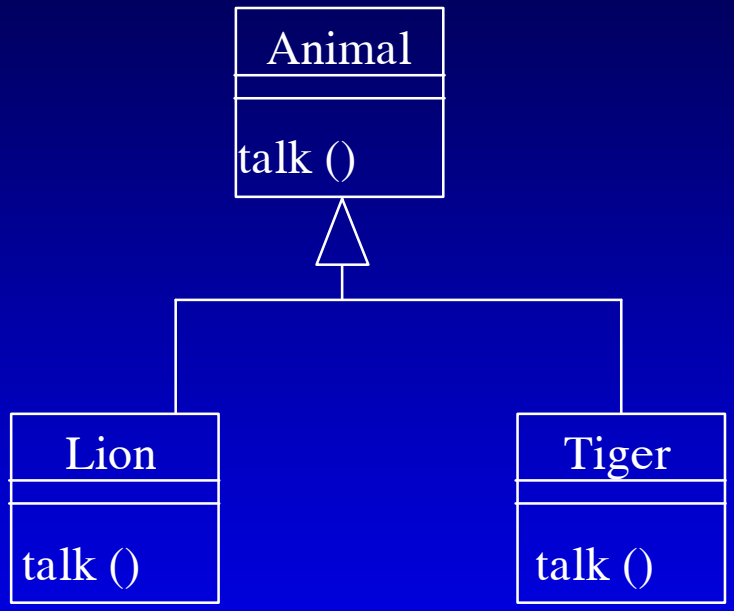
Chia sẻ các thuộc tính và hành vi chung

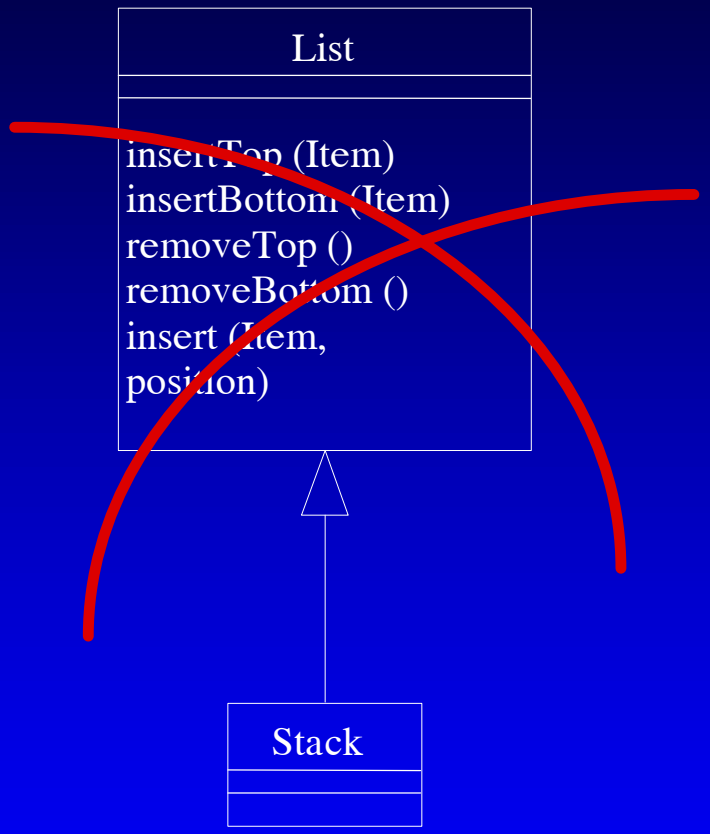
- Tuân thủ qui tắc lập trình "Là một dạng của"
- Khả năng thay thể Class



Các class trên có tuân thủ IS-A style of programming?

Chia sẻ các thuộc tính và hành vi chung (tt)



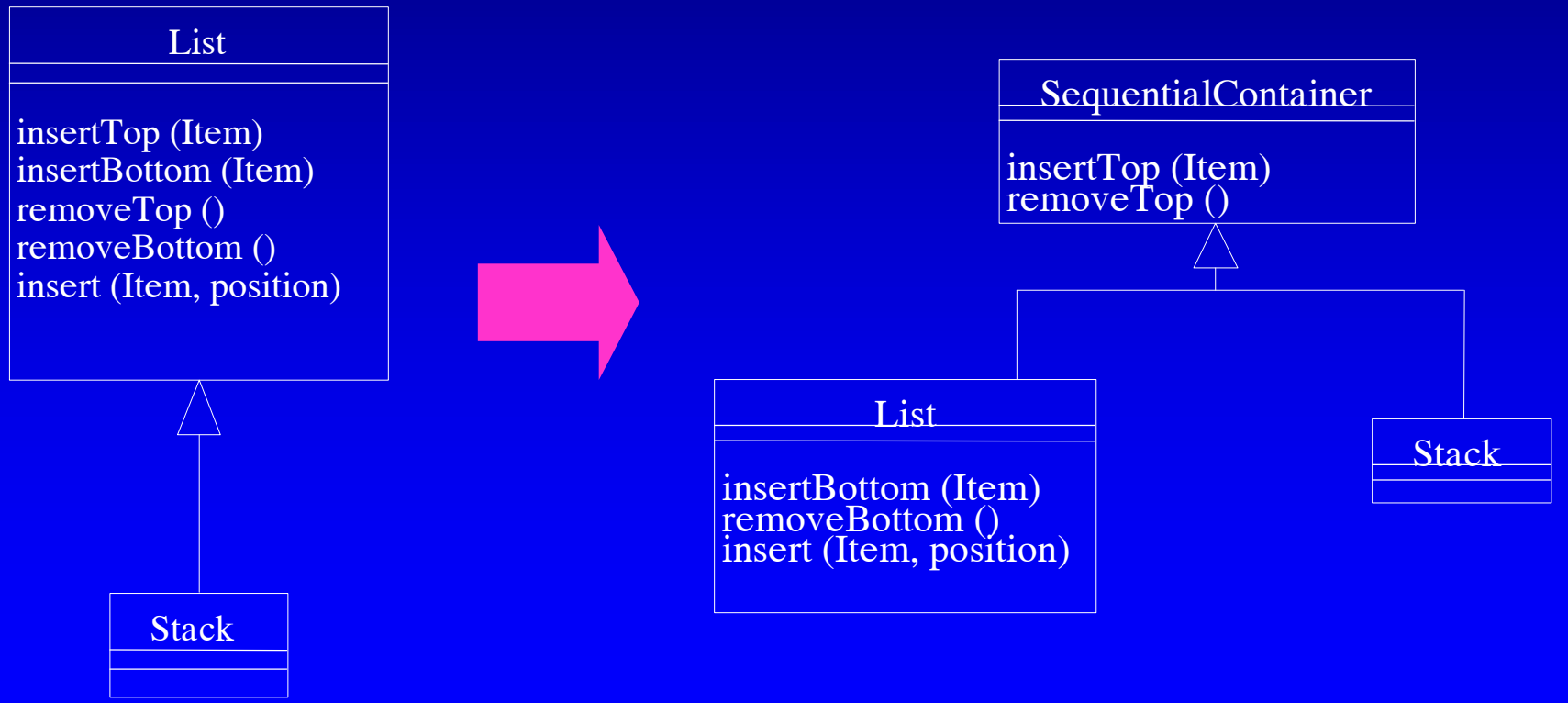


Sử dụng quan hệ tổng quát hóa

- Chia sẻ các thuộc tính và hành vi chung
- Chia sẻ cài đặt
- Cài đặt cơ chế Polymorphism
- Cài đặt cơ chế Metamorphosis

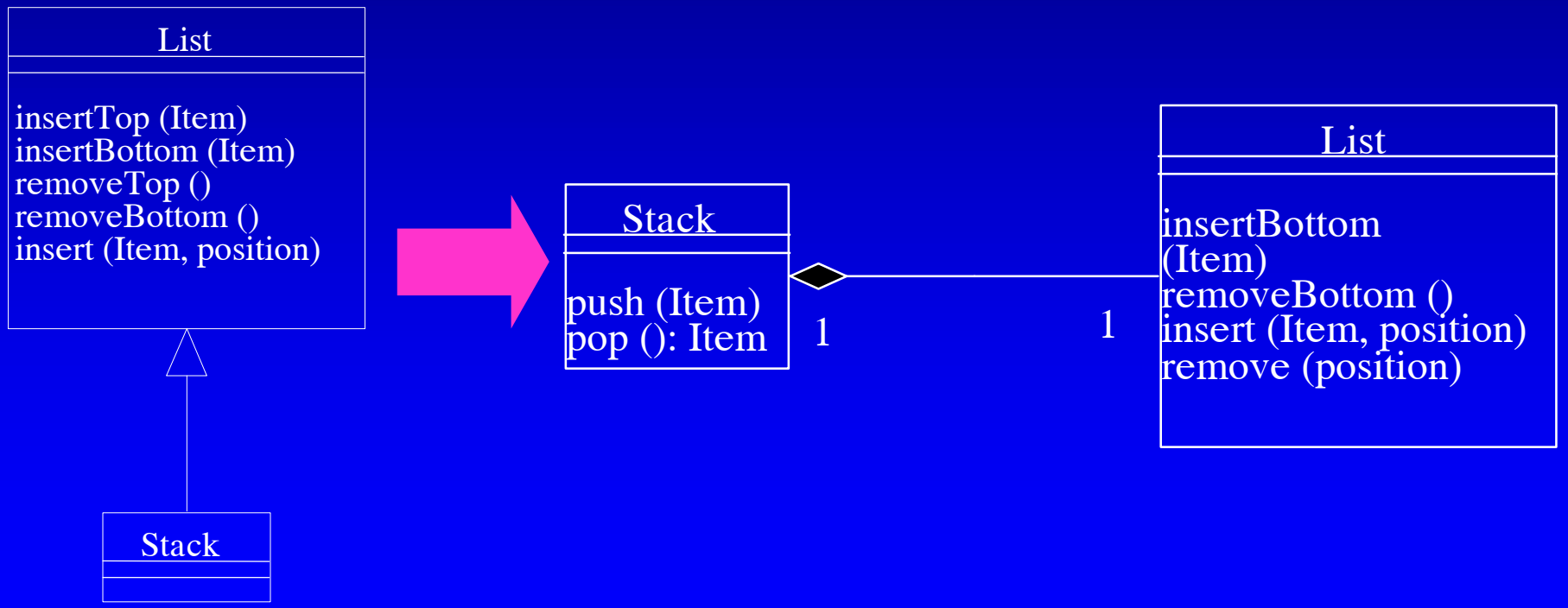
Chia sẻ cài đặt: Factoring (phân chia)

- Hỗ trợ khả năng dùng lại khi cài đặt class khác
- Không thể dùng nếu class bạn muốn "dùng lại" không thể thay đổi



Chia sẻ cài đặt: Delegation (đại diện)

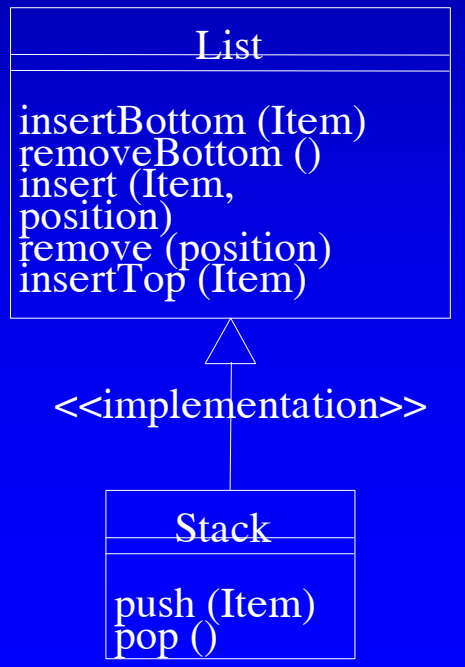
- Hổ trợ khả năng dùng lại khi cài đặt class khác
- Không thể dùng nếu class bạn muốn "dùng lại" không thể thay đổi



Quan hệ kế thừa dạng <<implementation>>

- Các public operation, attribute và relationship của tổ tiên không nhìn thấy được bởi các client của các thể hiện của các class con cháu
- Các class con cháu phải định nghĩa các truy cập đến operations, attributes, và relationships của tổ tiên

push() and pop() có thể truy cập đến các method của List nhưng các thể hiện của Stack thì không

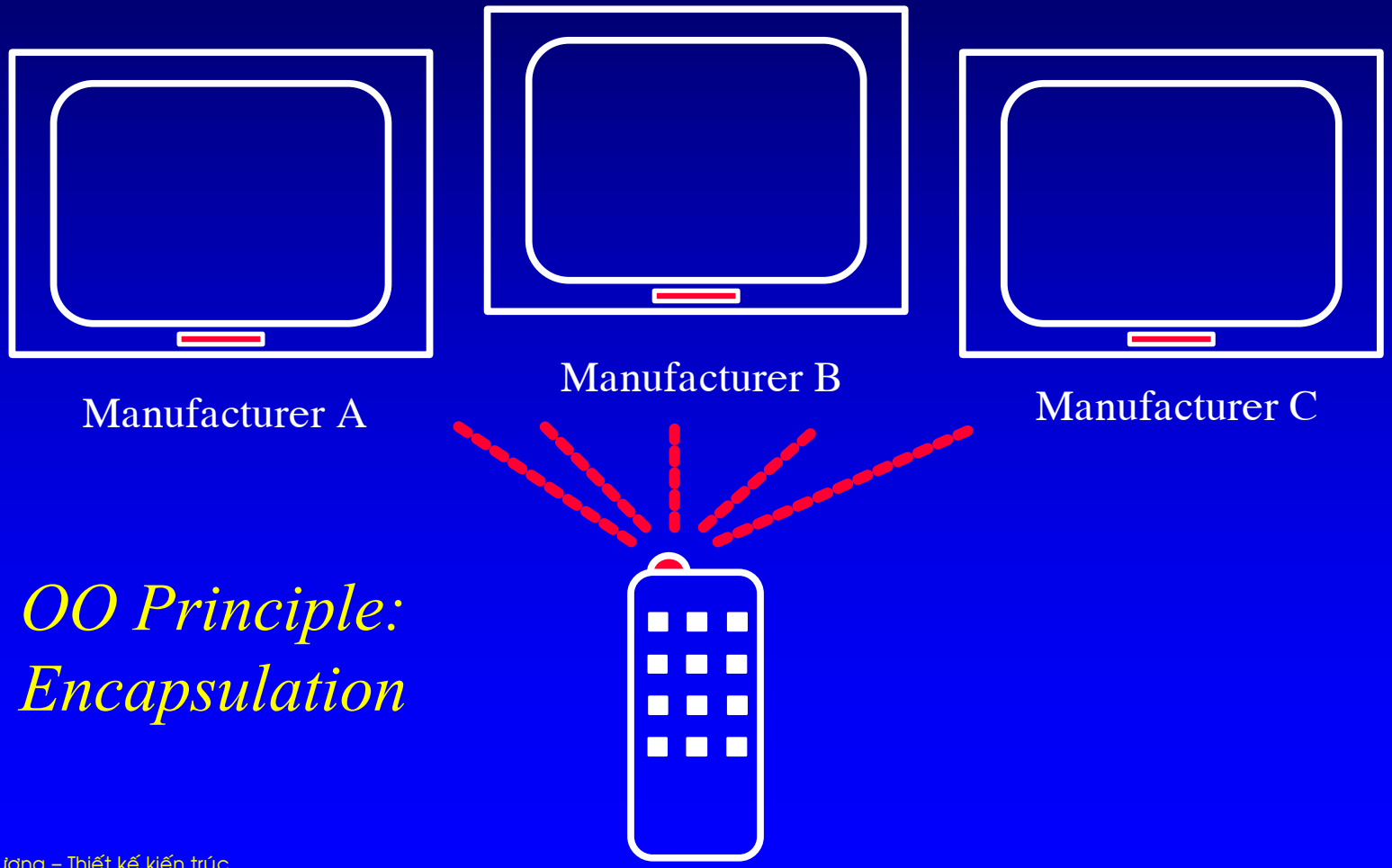


Sử dụng quan hệ tổng quát hóa

- Chia sẻ các thuộc tính và hành vi chung
- Chia sẻ cài đặt
- Cài đặt cơ chế Polymorphism
- Cài đặt cơ chế Metamorphosis

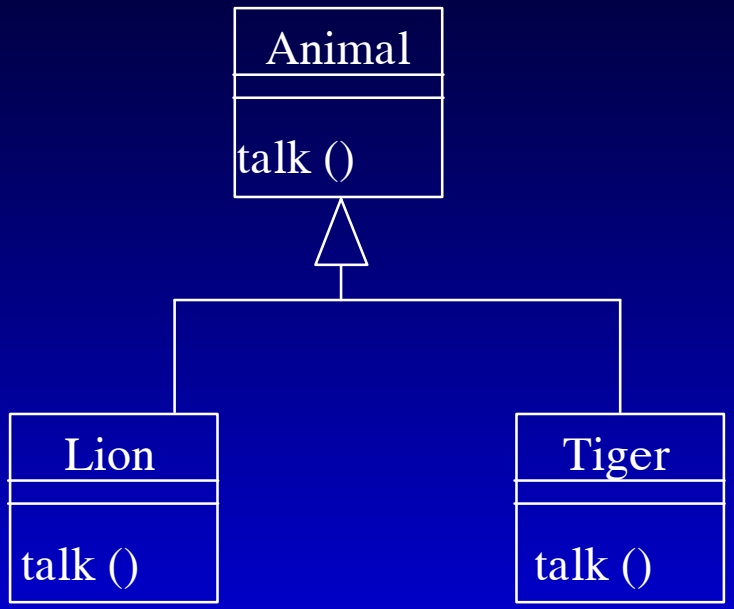
Nhắc lại: Polymorphism là gì ?

- 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

So sánh Interface và Generalization

- Các Interface hỗ trợ biểu diễn độc lập với cài đặt của polymorphism
  - Realization relationships có thể băng ngang qua cấu trúc phân cấp của quan hệ tổng quát hóa
- Các Interface chỉ thuần là đặc tả, không có hành vi
  - Abstract base class có thể định nghĩa attributes và associations
- Các Interface hoàn toàn độc lập với quan hệ kế thừa
  - Generalization thường dùng để cài đặt việc dùng lại
  - Interfaces thường dùng để đặc tả việc tái sử dụng các hành vi
- Generalization cung cấp một cách cài đặt polymorphism

Dùng QH tổng quát hóa để cài Polymorphism

- Chỉ cung cấp interface cho các class con cháu?
  - Thiết kế tổ tiên như một abstract class
  - Mọi method cài đặt ở các class con cháu
- Cung cấp interface và behavior mặc định cho các class con cháu?
  - Thiết kế tổ tiên như một concrete class với các method mặc định
  - Cho phép dùng các polymorphic operation
- Cung cấp interface và behavior bắt buộc cho các class con cháu?
  - Thiết kế tổ tiên là concrete class
  - Không cho phép các polymorphic operation

Sử dụng quan hệ tổng quát hóa

- Chia sẻ các thuộc tính và hành vi chung
- Chia sẻ cài đặt
- Cài đặt cơ chế Polymorphism
- Cài đặt cơ chế Metamorphosis

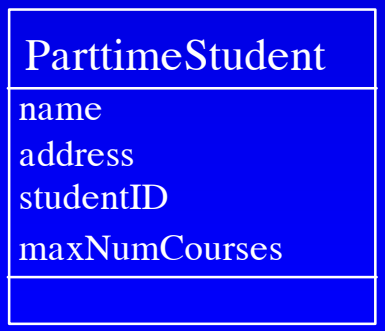
Metamorphosis là gì?

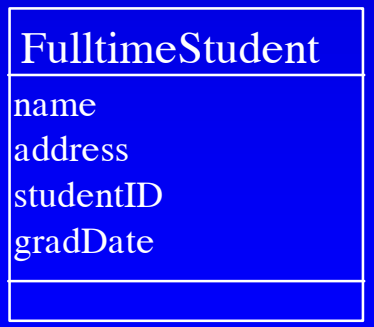
- Metamorphosis
  - 1. Một thay đổi trong hình dạng, cấu trúc, hay chức năng; đặc biệt là các thay đổi vật lý mà các động vật phải trải qua, như con nòng nọc biến thành con ếc
  - 2. Mọi thay đổi được ghi nhận, như trong các ký tự, thể hiện, hoặc điều kiện
  - Xem thêm Webster's New World Dictionary, Simon & Schuster, Inc., 1979

Metamorphosis tồn tại trong thế giới thực
Làm sao mô hình hóa chúng?

Ví dụ: Metamorphosis

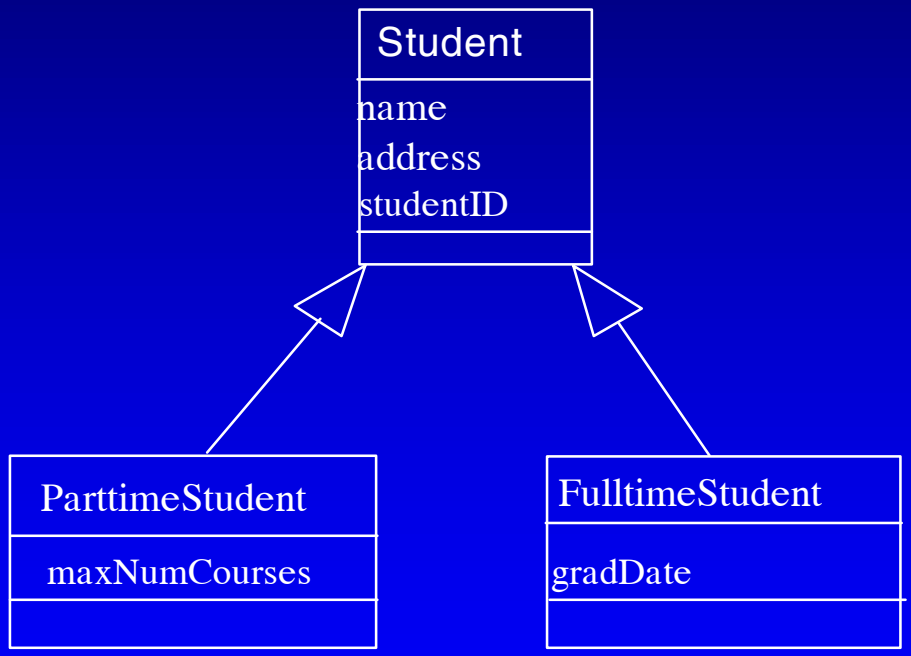
- Trong trường đại học, có full time students và part time students
  - Full time students có ngày tốt nghiệp dự kiến còn part time students lại không
  - Part time students có thể đăng ký học tối đa 3 môn trong khi full time students không bị giới hạn





Một hướng tiếp cận Modeling Metamorphosis

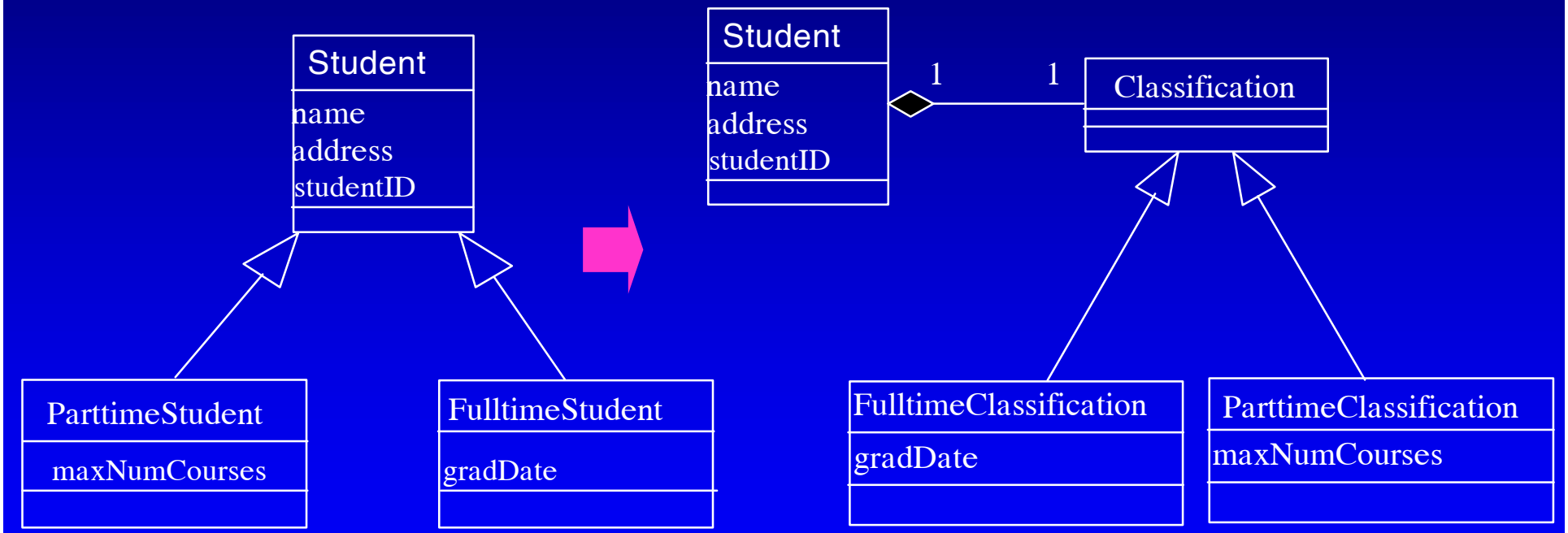
- Có thể tạo một quan hệ tổng quát hóa



Chuyện gì xảy ra nếu một part-time student trở thành full-time student?

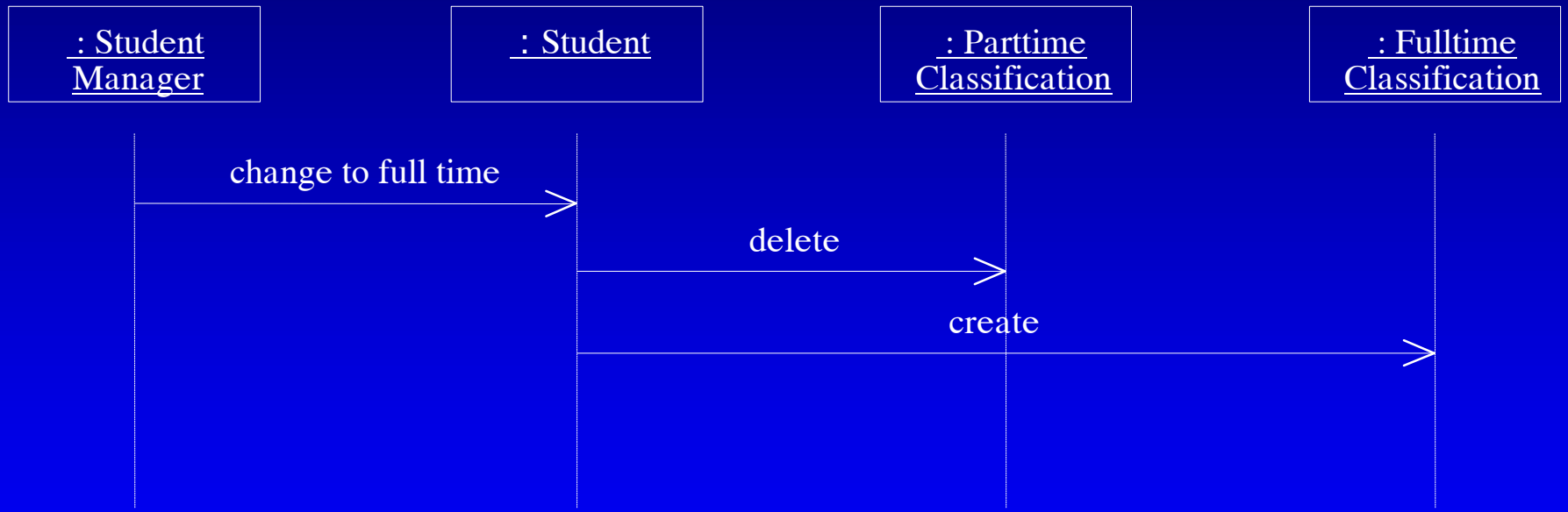
Một hướng tiếp cận khác

- Quan hệ kế thừa có thể dùng để mô hình hóa cấu trúc, hành vi và quan hệ chung và tạo quan hệ với phần "thay đổi"



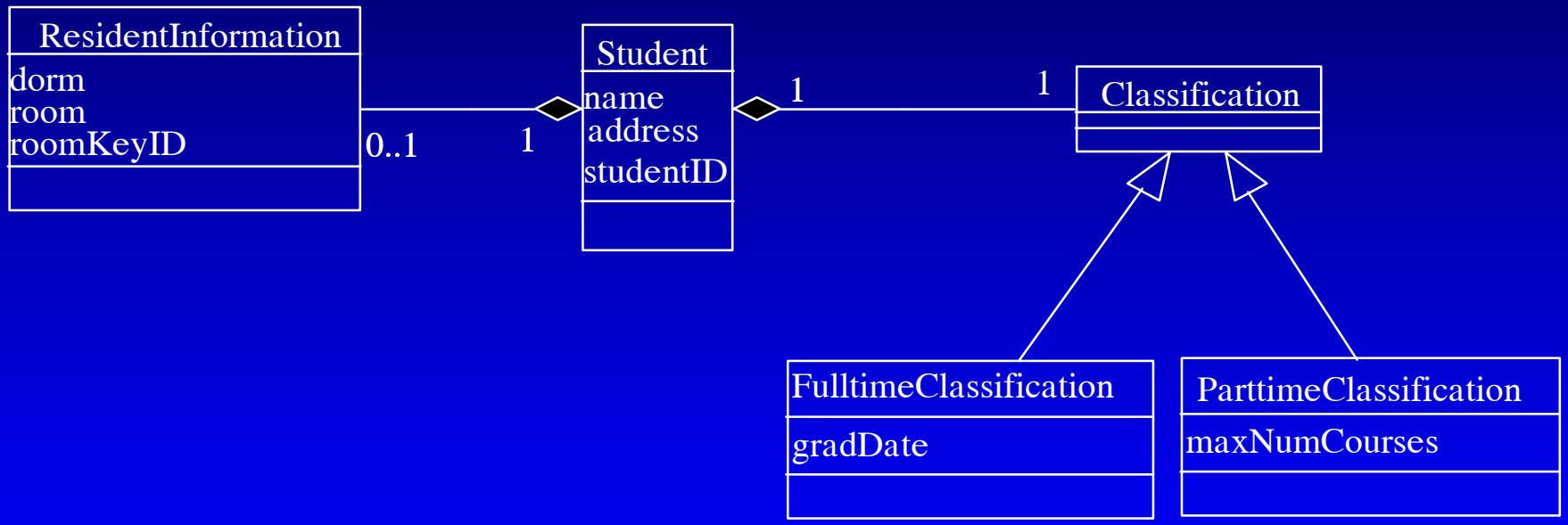
Một hướng tiếp cận khác (tt)

- Metamorphosis được hoàn tất bởi object "nói chuyện với" phần "thay đổi"



Metamorphosis và tính mềm dẻo

- Kỹ thuật này thêm tính mềm dẻo cho mô hình



Bài tập: Định nghĩa Generalizations

- Hãy cho biết:
  - Tất cả các design class
- Hãy xác định:
  - Tất cả các tinh chỉnh liên quan đên generalizations có sẵn
  - Mọi ứng dụng generalization mới
    - Kiểm tra là đã xem xétmọi metamorphosis
- Xây dựng các lược đồ:
  - Class diagram chứa mọi quan hệ tổng quát hóa mới (hay đã tính chỉnh) giữa các class
  - Tinh chỉnh use-case realizations (optional)

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Giải quyết đụng độ giữa các Use-Case

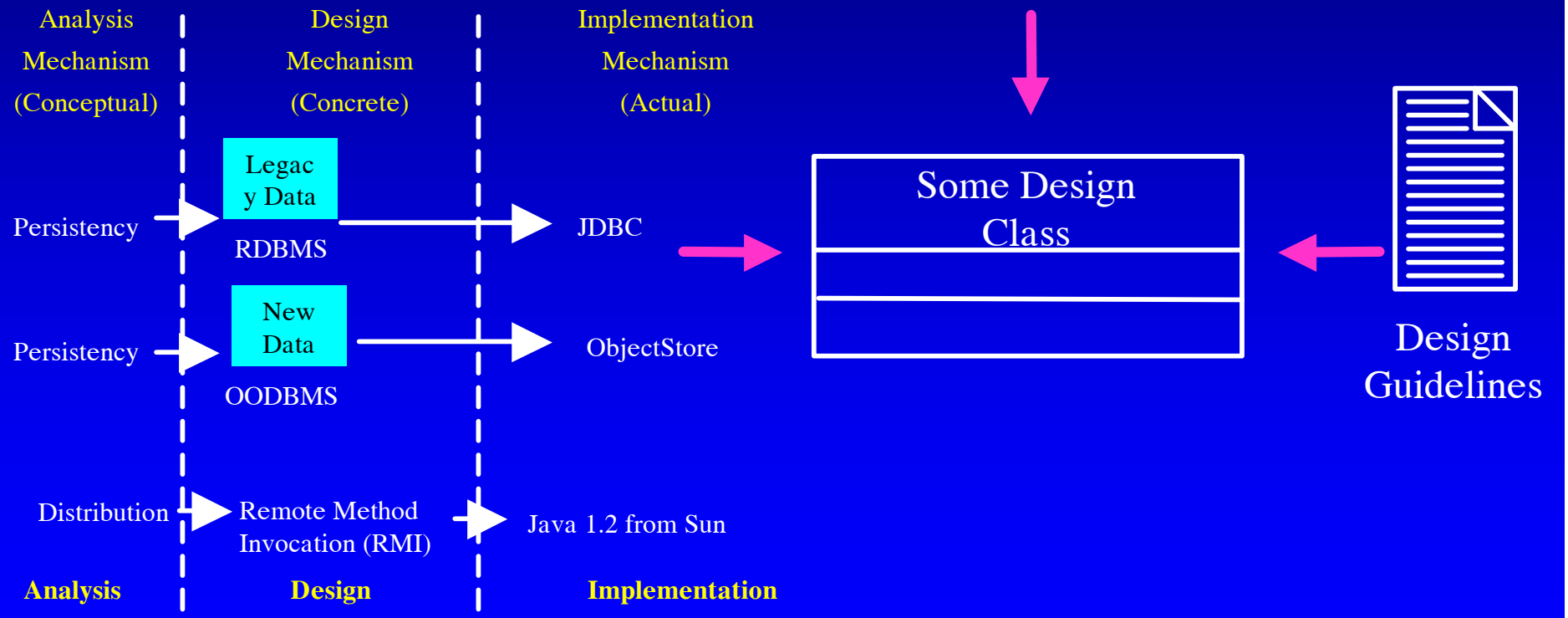
- Nhiều use case có thể truy cập riêng rẽ đến các design object
- Options
  - Dùng cơ chế truyền message đồng bộ => đến trước được xử lý trước
  - Xác định các operation (hay code) cần protect
  - Áp dụng cơ chế access control
    - Lập hàng đợi Message
    - Semaphores (hoặc 'tokens')
    - Các cơ chế khóa khác
- Lời giải phụ thuộc nhiều vào môi trường cài đặt

Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- Checkpoints

Xử lý các yêu cầu phi chức năng nói chung

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



Các bước thiết kế Class

- Tạo các Design Class ban đầu
- Xác định các Persistent Class
- Định nghĩa các Operation
- Định nghĩa Class Visibility
- Định nghĩa các Method
- Định nghĩa các trạng thái
- Định nghĩa các thuộc tính
- Định nghĩa các phụ thuộc
- Định nghĩa các mỗi kết hợp
- Định nghĩa các quan hệ tổng quát hóa
- Giải quyết đụng độ giữa các Use-Case
- Xử lý các yêu cầu phi chức năng nói chung
- 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 NNLT 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?