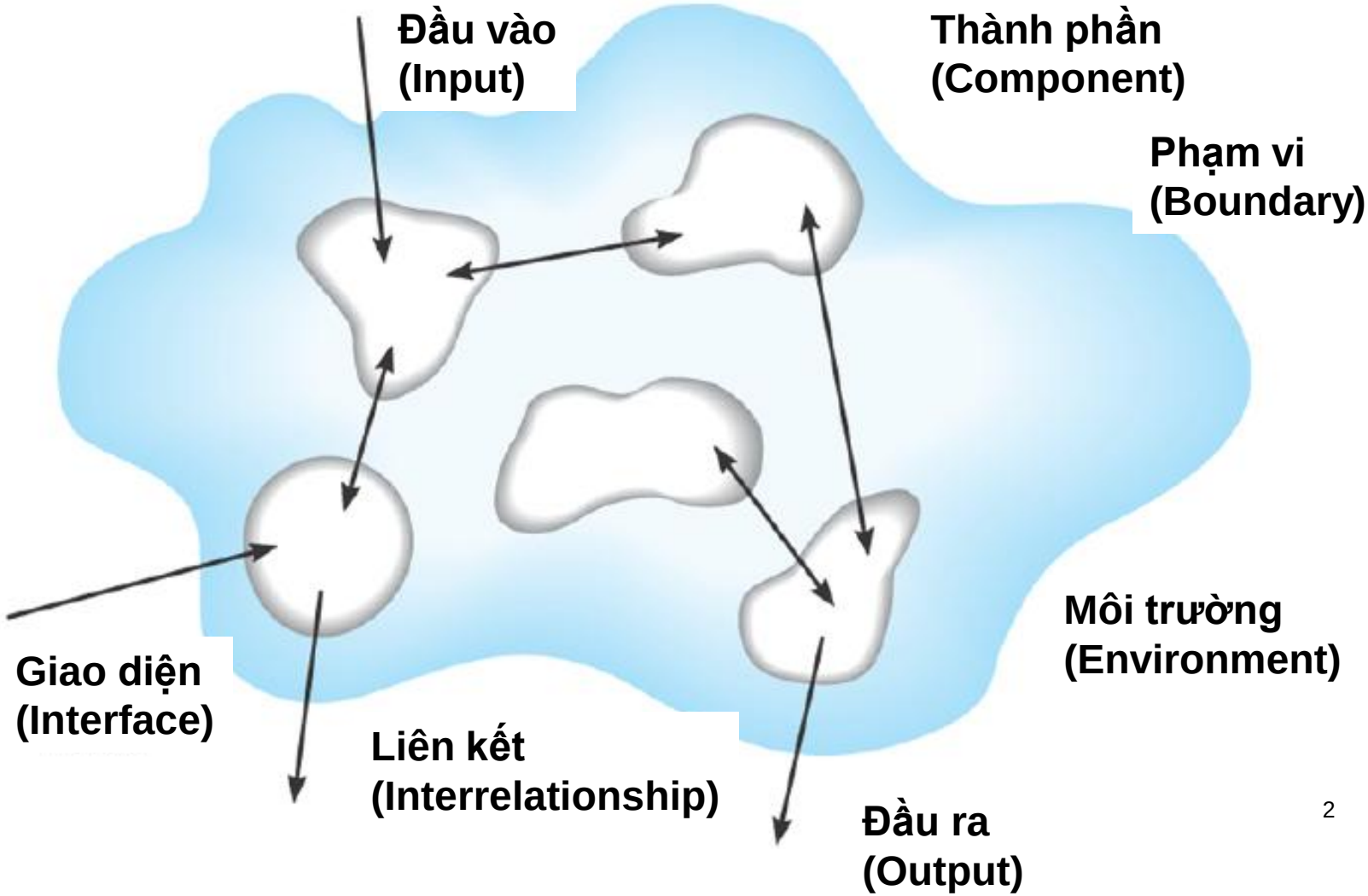
BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

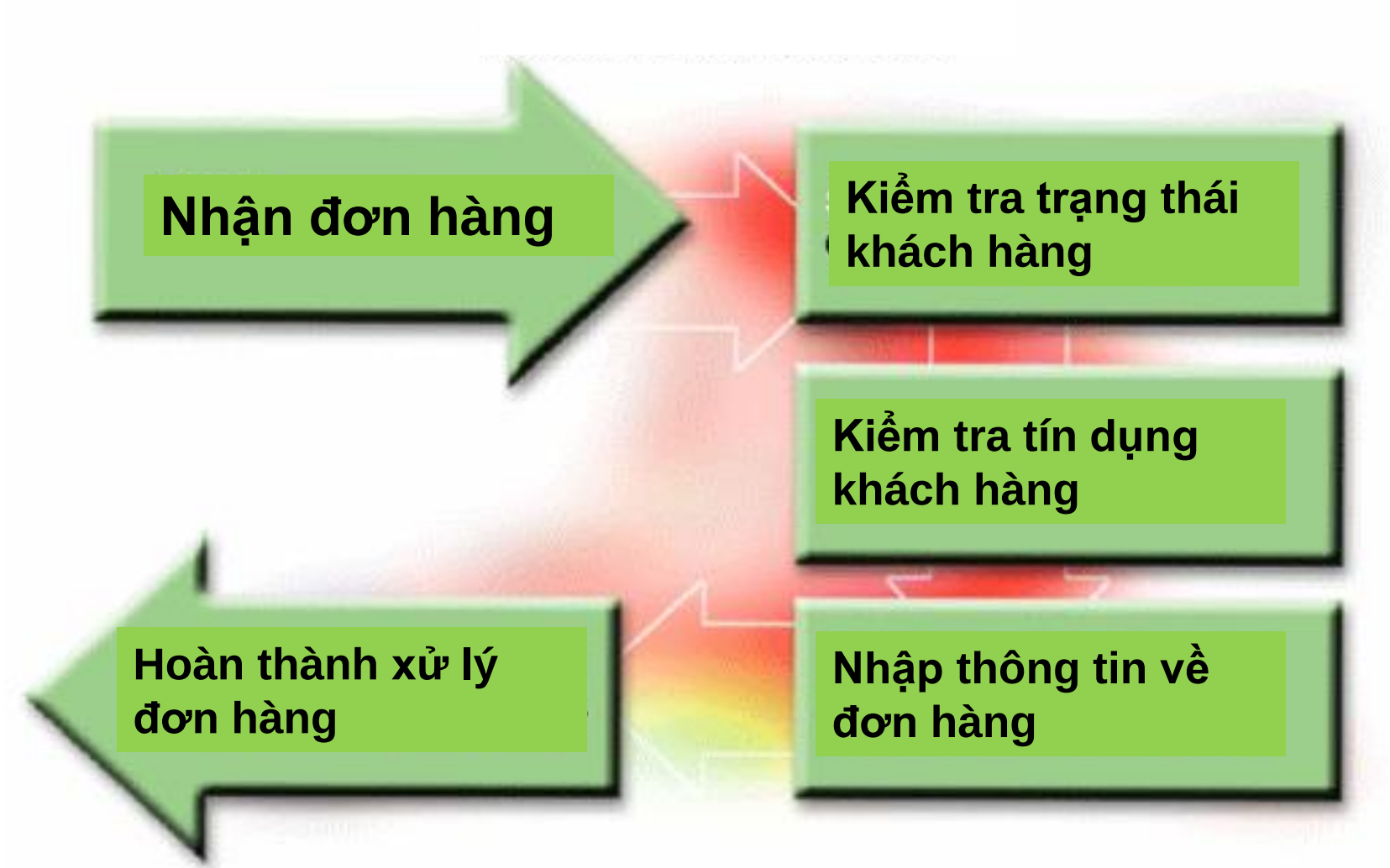
OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 01: Tổng quan về XD phần mềm

Các đặc điểm của hệ thống



Ví dụ: Quy trình xử lý đơn hàng



Vòng đời phát triển hệ thống

(*Systems Development Life Cycle* – SDLC)

yêu cầu hệ thống

Pha 1:
Lập kế hoạch

Pha
Phâ

Vận hành, bảo trì

Lập kế hoạch

- Giải quyết các vấn đề, câu hỏi
  - Tại sao phải xây dựng HTTT ?
  - Nhóm dự án phát triển hệ thống thông tin như thế nào?
- Các công việc cụ thể
  - Tìm hiểu dự án được bắt đầu và được đánh giá ban đầu như thế nào
  - Xác định các vấn đề, cơ hội, mục tiêu
  - Phân tích SWOT (Strength – Weakness – Opportunity – Threat)

Phân tích hệ thống

- Giải quyết các vấn đề, câu hỏi
  - Ai sẽ sử dụng hệ thống?
  - Hệ thống sẽ thực hiện gì, khi nào, ở đâu?
- Các công việc cụ thể
  - Phân tích chiến lược: phân tích hiện trạng, phương pháp sử dụng
  - Thu thập yêu cầu: mô hình hóa và phân tích các yêu cầu
  - Đề xuất mô hình hệ thống

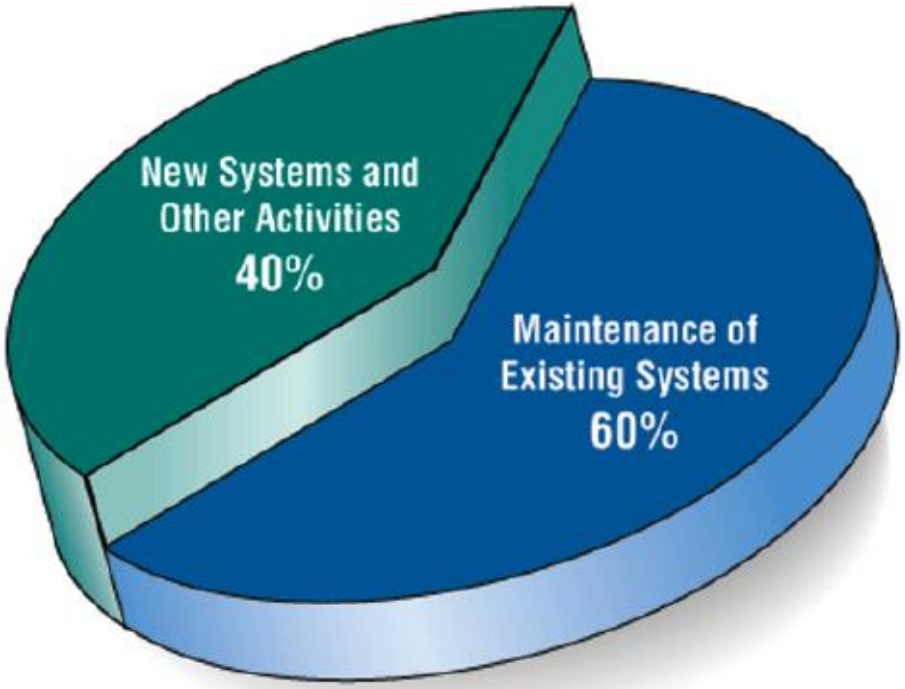
Thiết kế hệ thống

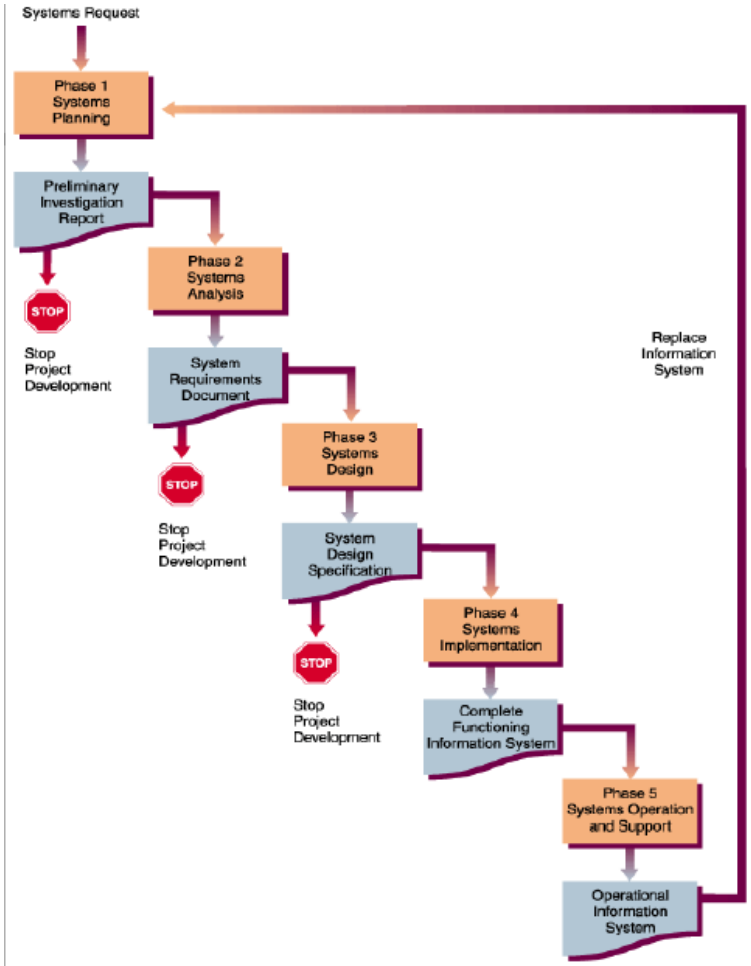
- Giải quyết các vấn đề, câu hỏi
  - Hệ thống sẽ hoạt động như thế nào (phần cứng, phần mềm, mạng, giao diện người dùng, modun chương trình, CSDL, tệp, …)
- Các công việc cụ thể
  - Chiến lược thực hiện
  - Kiến trúc hệ thống: phần cứng, phần mềm, mạng
  - Thiết kế dữ liệu
  - Thiết kế chương trình
  - Thiết kế giao diện

Cài đặt hệ thống

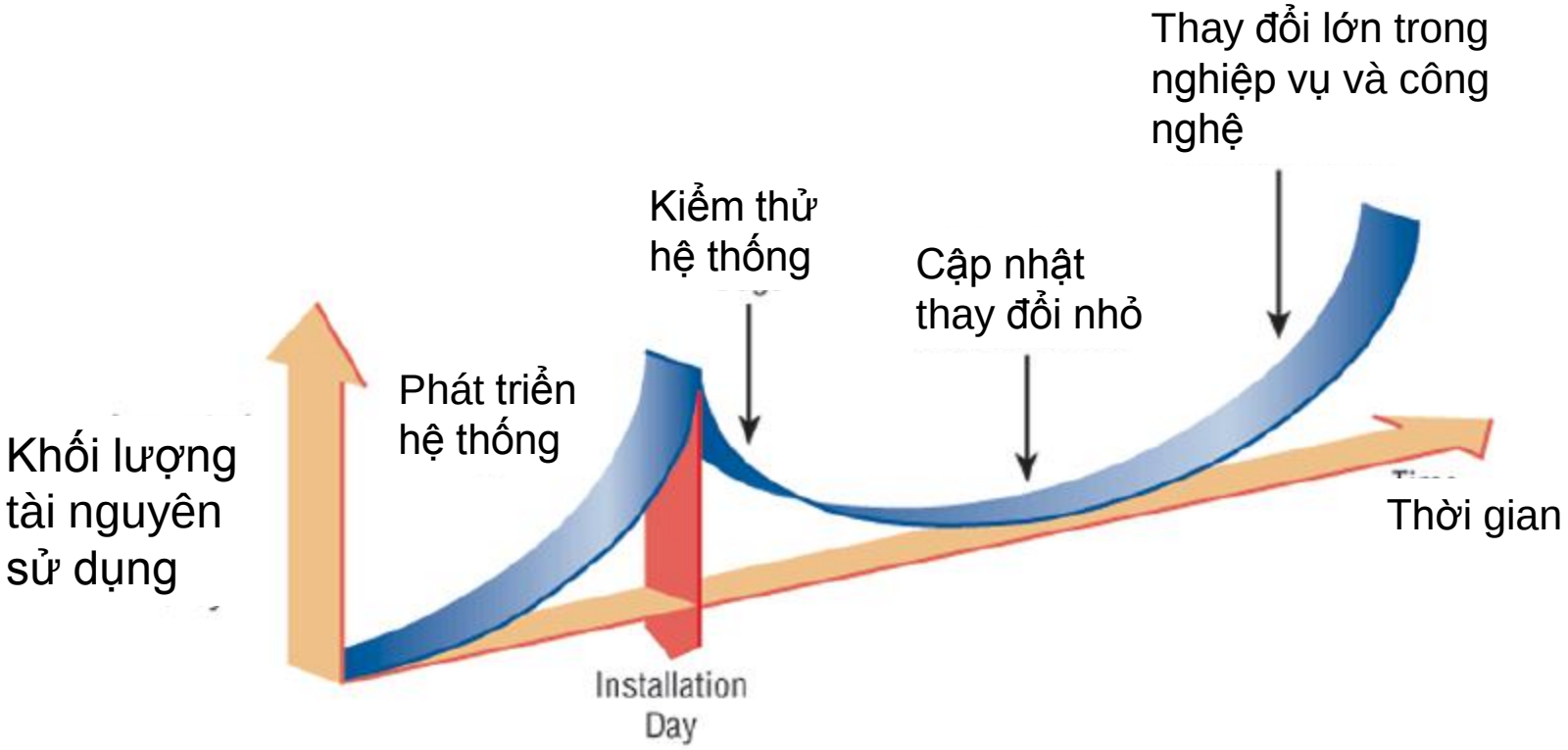
- Giải quyết các vấn đề, câu hỏi
  - Lập trình, kiểm thử
- Các công việc cụ thể
  - Phát triển ứng dụng: lập trình, kiểm thử, lập tài liệu cho các modun chương trình
  - Cài đặt và đánh giá
  - Xây dựng kế hoạch hỗ trợ và bảo trì hệ thống

Phân bổ chi phí cho các hoạt động





Sử dụng tài nguyên



Một số phương pháp phát triển HT

- Phương pháp ~ một cách thực hiện chu trình phát triển hệ thống
- 3 nhóm phương pháp
  - Các phương pháp hướng quy trình
    - Tập trung định nghĩa các hoạt động gắn với hệ thống
    - Mô hình hóa các quy trình với luồng vào/ra
  - Các phương pháp hướng dữ liệu
    - Tập trung định nghĩa nội dung dữ liệu lưu trữ
    - Mô hình hóa dữ liệu
  - Các phương pháp hướng đối tượng
    - Cân bằng giưa dữ liệu và quy trình
    - UML là một ngôn ngữ mô hình hóa

Một số kiểu phát triển hệ thống

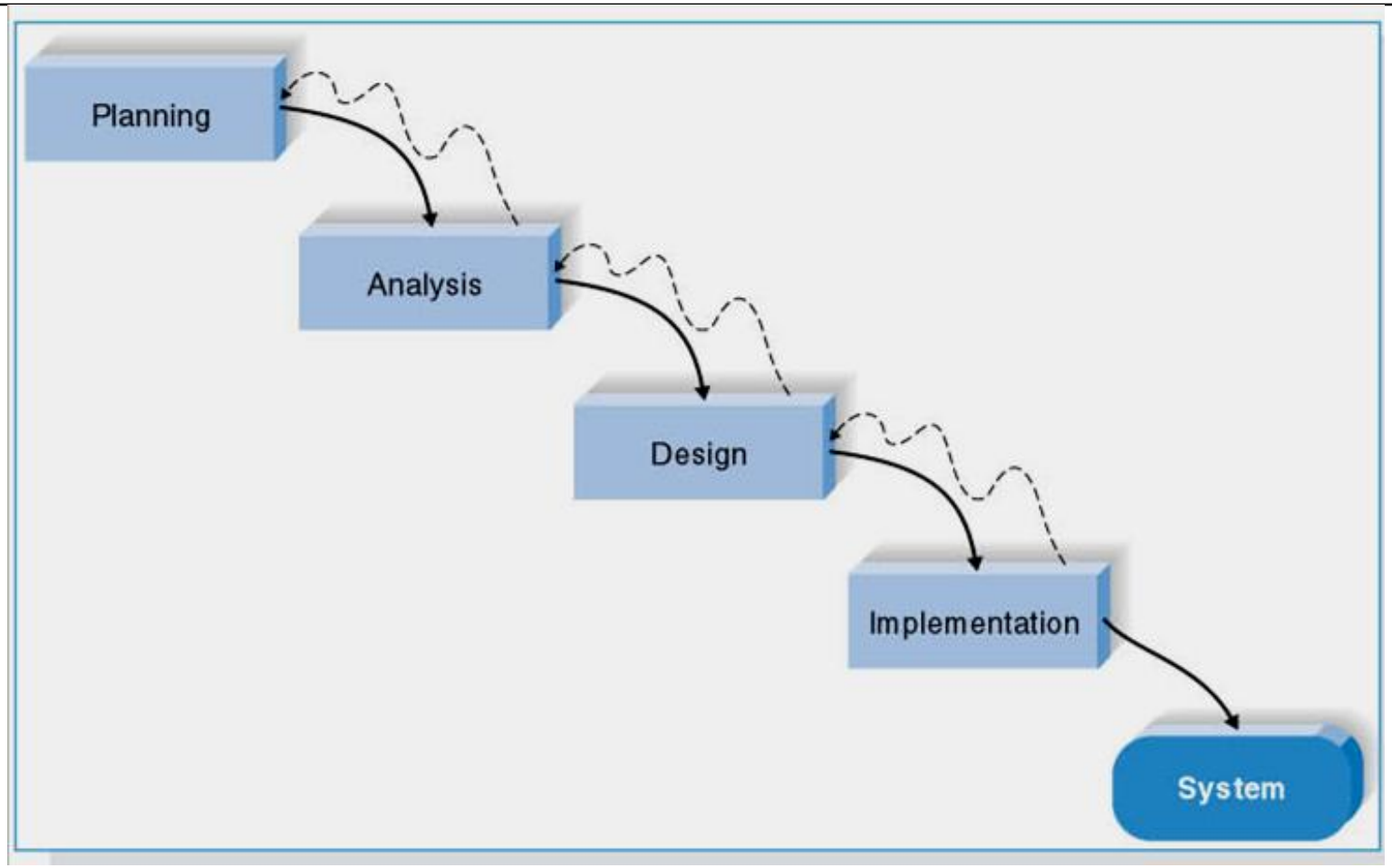
- Phân tích thiết kế có cấu trúc

  (Structured Design)
  - Chu trình thác nước
  - Chu trình tăng trưởng / chu trình song song
- Phát triển nhanh ứng dụng

  (Rapid Application Development - RAD)
  - Chu trình xoắn ốc
  - Làm bản mẫu
- Hướng lập trình ứng dụng

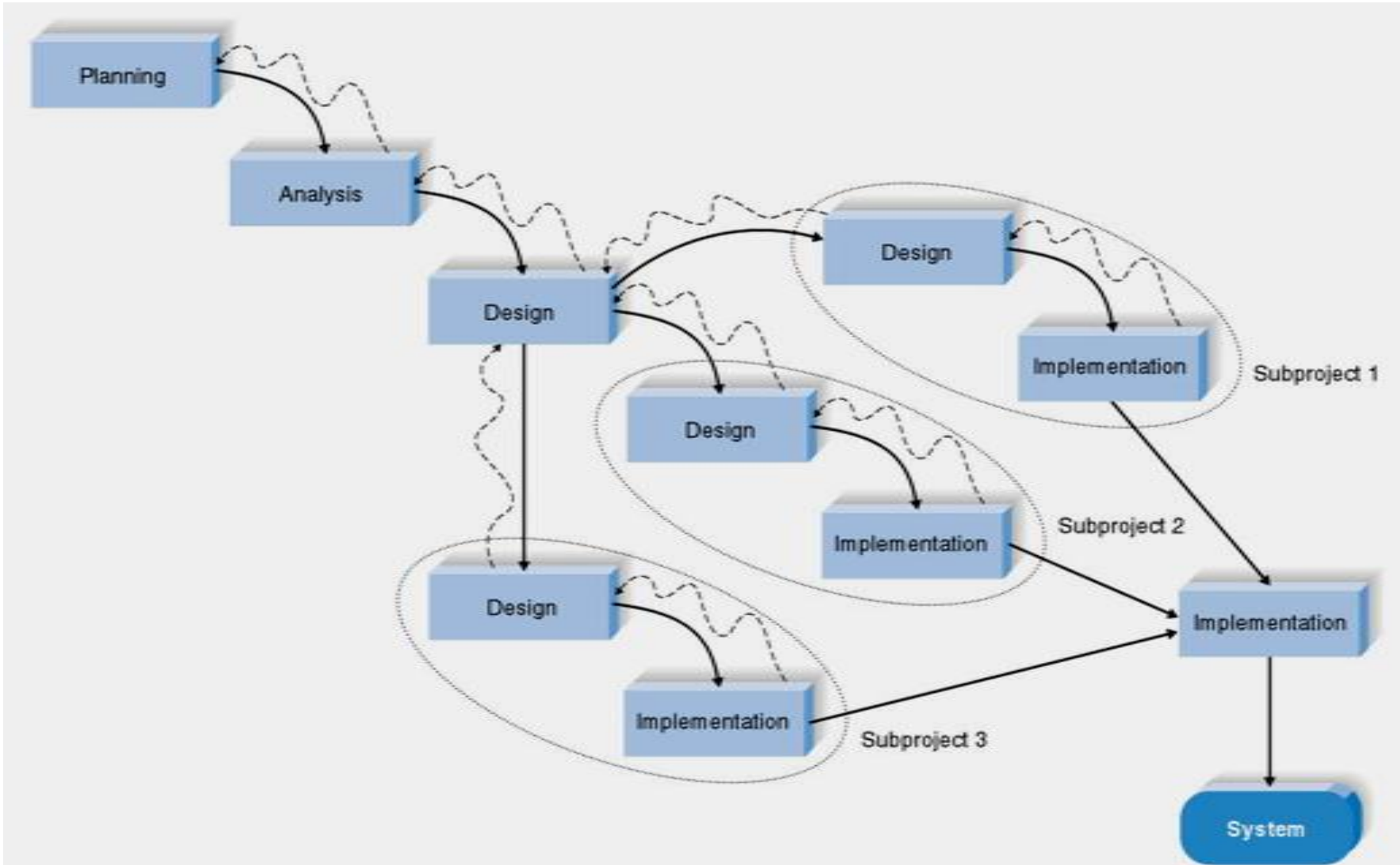
  (Agile Development)
  - eXtreme-Programming based

Mô hình phát triển thác nước
(Waterfall Development Model)



Mô hình phát triển song song

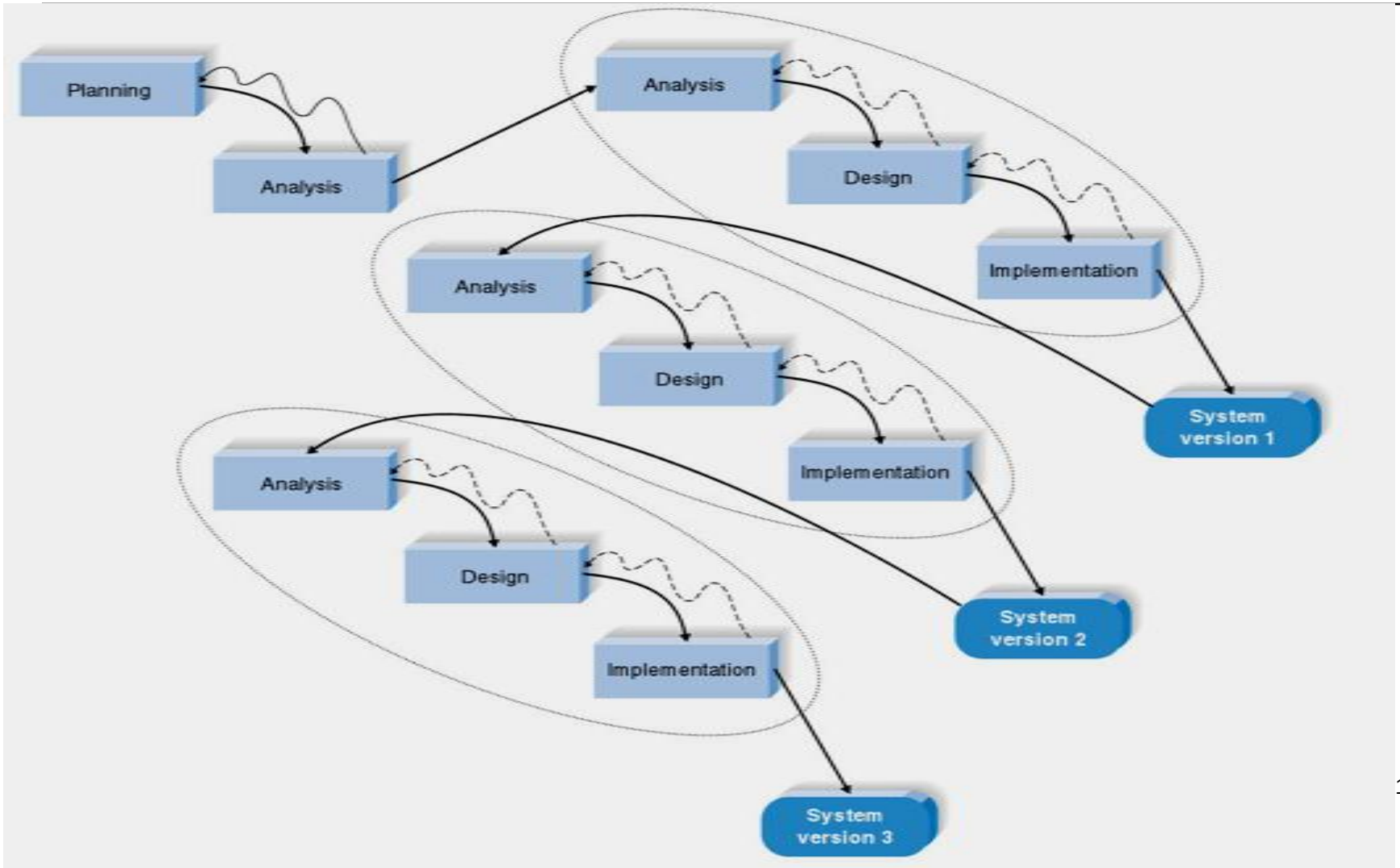
(Parallel Development Model)



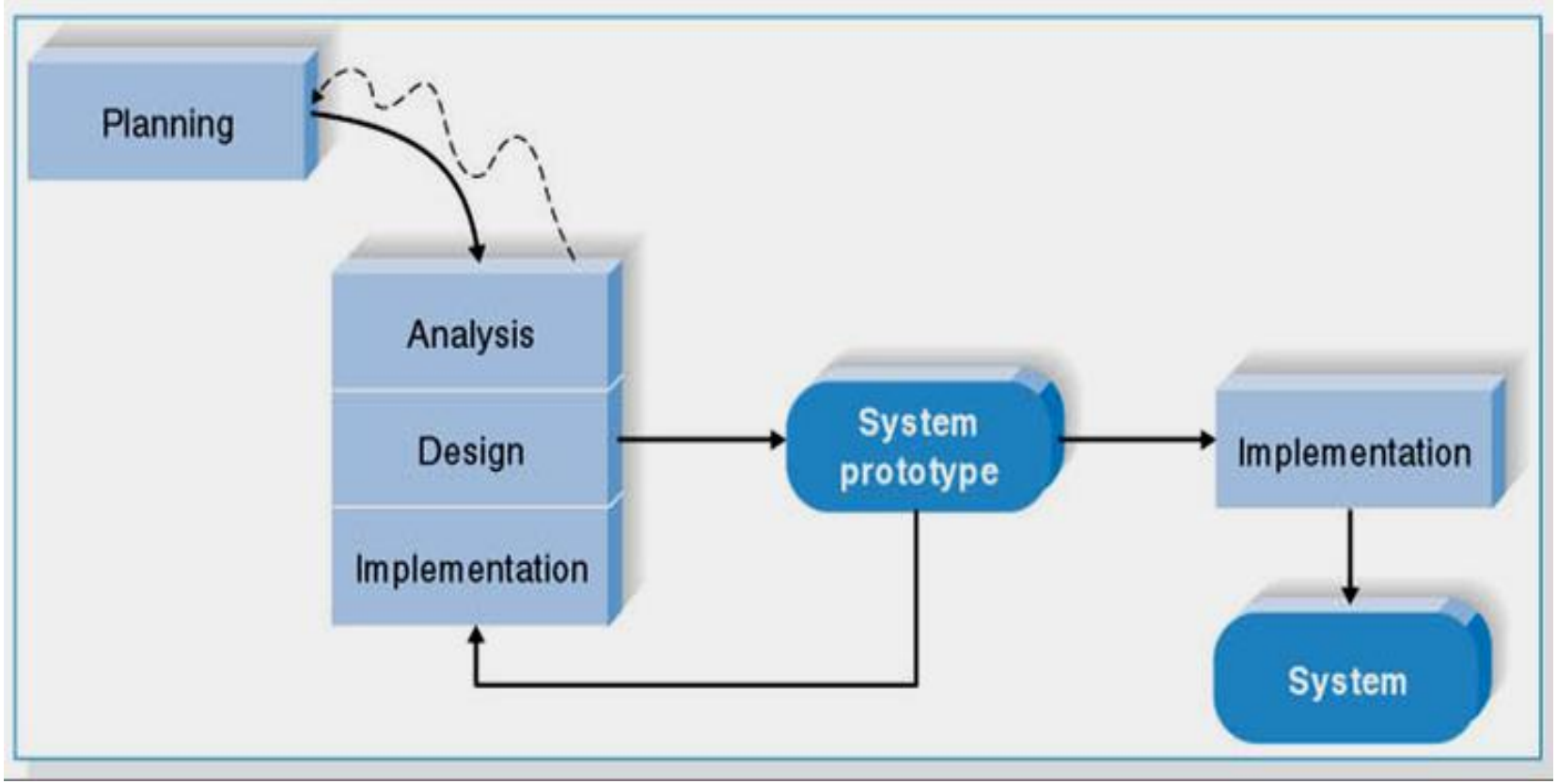
Phương pháp phát triển nhanh ƯD

- Thực hiện phát triển từng phần hệ thống với mục đích chuyển giao cho người dùng sớm
- Cần sử dụng các kỹ thuật và công cụ để tăng tốc quá trình phân tích, thiết kế và cài đặt (vd: CASE – computer-aided software engineering)

Mô hình phát triển xoắn ốc
(Spiral Development Model)



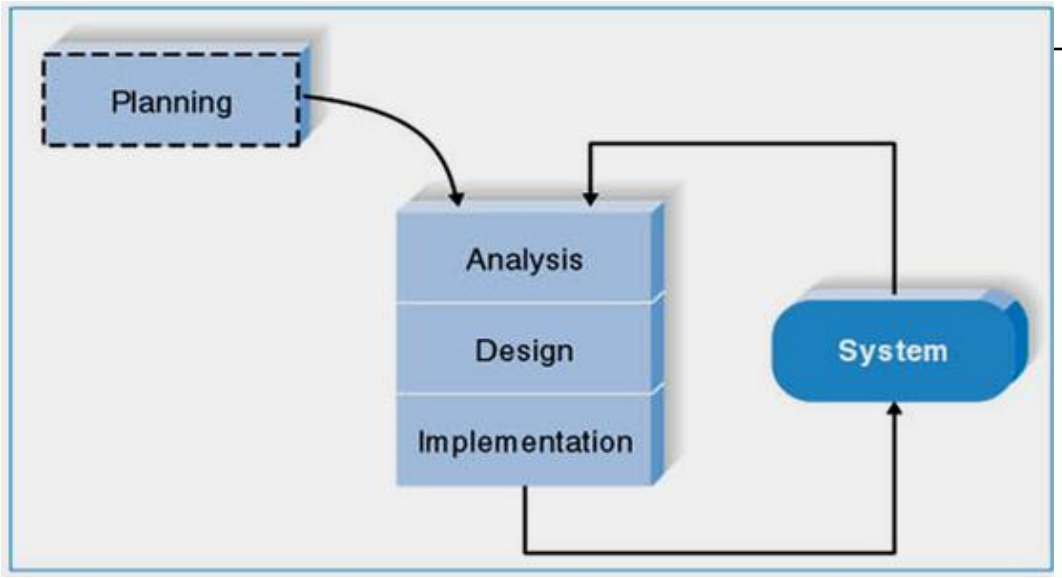
Làm bản mẫu
(prototyping-based)

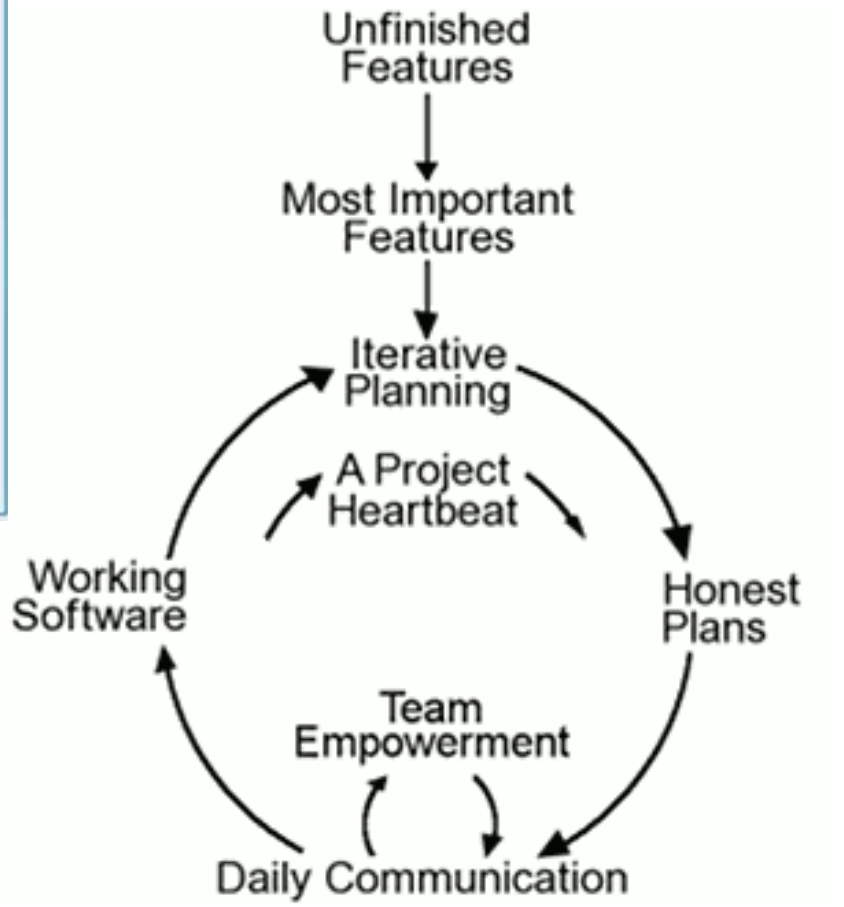


Phương pháp phát triển linh hoạt
(Agile Development)

- Tập trung vào bước của chu trình phát triển và hạn chế việc sử dụng mô hình hóa, xây dựng tài liệu của các bước trung gian
- Phát triển ứng dụng đơn giản, lăp đi lặp lại
- Sử dụng eXtreme Programming (XP)

eXtreme-Programming -based





- Trao đổi thông tin (communication)
- Đơn giản (simplicity)
- Phản hồi (feedback)
- Thế mạnh (courage)

Đặc điểm

- Tương tác liên tục
- Thiết kế đơn giản, sử dụng các nguyên lý và dạng thức thiết kế chung
- Nhóm làm việc: người lập trình, khách hàng, người quản trị - khách hàng trực diện

Các nhiệm vụ phân tích

- Phân tích nghiệp vụ
  - Phân tích các yêu cầu nghiệp vụ và vai trò của hệ thống trong việc thực hiện các yêu cầu nghiệp vụ này
  - Đưa ra các quy trình nghiệp vụ “mới” và các chính sách
- Phân tích hệ thống
  - Xác định công nghệ sử dụng
  - Thiết kế các quy trình nghiệp vụ đề xuất và HTTT theo quy chuẩn

Phân tích và thiết kế hệ thống

- Mục đích
  - Xác định các vấn đề, các cơ hội, các mục tiêu
  - Phân tích các dòng thông tin vào/ra
  - Sử dụng máy tính để xử lý tự động các thông tin
- Yêu cầu
  - Mô hình hóa quy trình nghiệp vụ
  - Xây dựng
    - Bản tóm lược nghiệp vụ
    - Mô hình nghiệp vụ
    - Quá trình nghiệp vụ

Các phương thức khảo sát

- Phỏng vấn *(Interviews)*
- *JAD – Joint Application Design*
- Bản câu hỏi *(Questionnaires)*
- Phân tích tài liệu *(Document Analysis)*
- Quan sát *(Observation)*

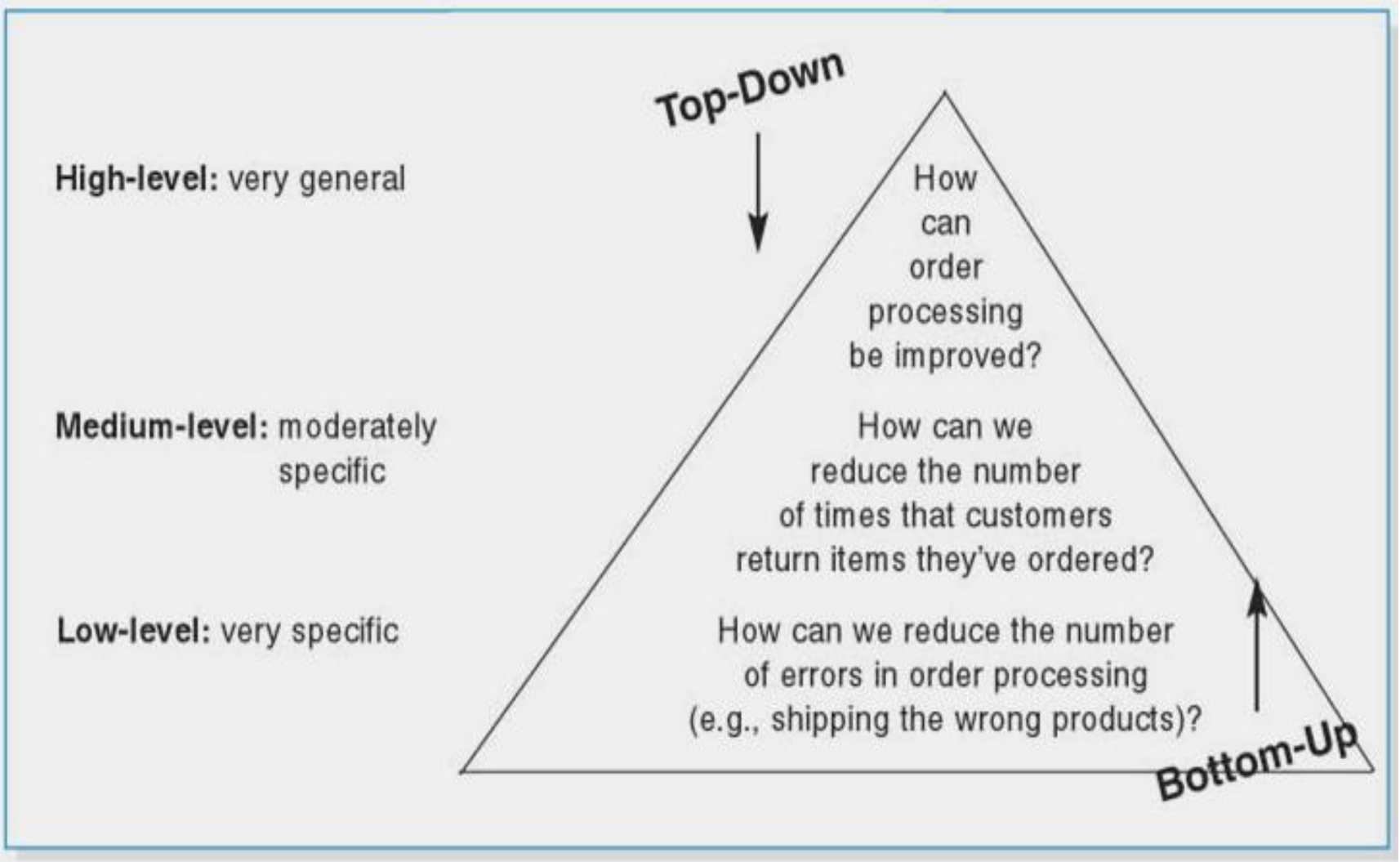
Phỏng vấn

- Mục đích:
  - Hiểu được nghiệp vụ thực hiện và ý kiến của những người đang thực hiện
  - Nắm bắt được hiện trạng hệ thống
  - Nắm bắt được mục tiêu, tổ chức nhân sự và các yêu cầu đối với hệ thống
- Các bước cơ bản
  - Xác định mục tiêu phỏng vấn & lựa chọn người để phỏng vấn
  - Thiết kế câu hỏi phỏng vấn
  - Chuẩn bị cho cuộc phỏng vấn
  - Tiến hành phỏng vấn
  - Lập tài liệu và đánh giá cuộc phỏng vấn

Lựa chọn người để phỏng vấn

- Mục tiêu của phỏng vấn
  - Xác định các lĩnh vực cần tìm hiểu
  - Liệt kê các sự kiện muốn thu thập
  - Đặt được các ý tưởng, các đề nghị và các ý kiến
- Chọn người để phỏng vấn
  - Dựa trên những thông tin cần thu thập
  - Có vai trò khác nhau: người thực hiện trực tiếp, người quản lý
  - Bám sát cơ cấu tổ chức

Chiến lược đặt câu hỏi

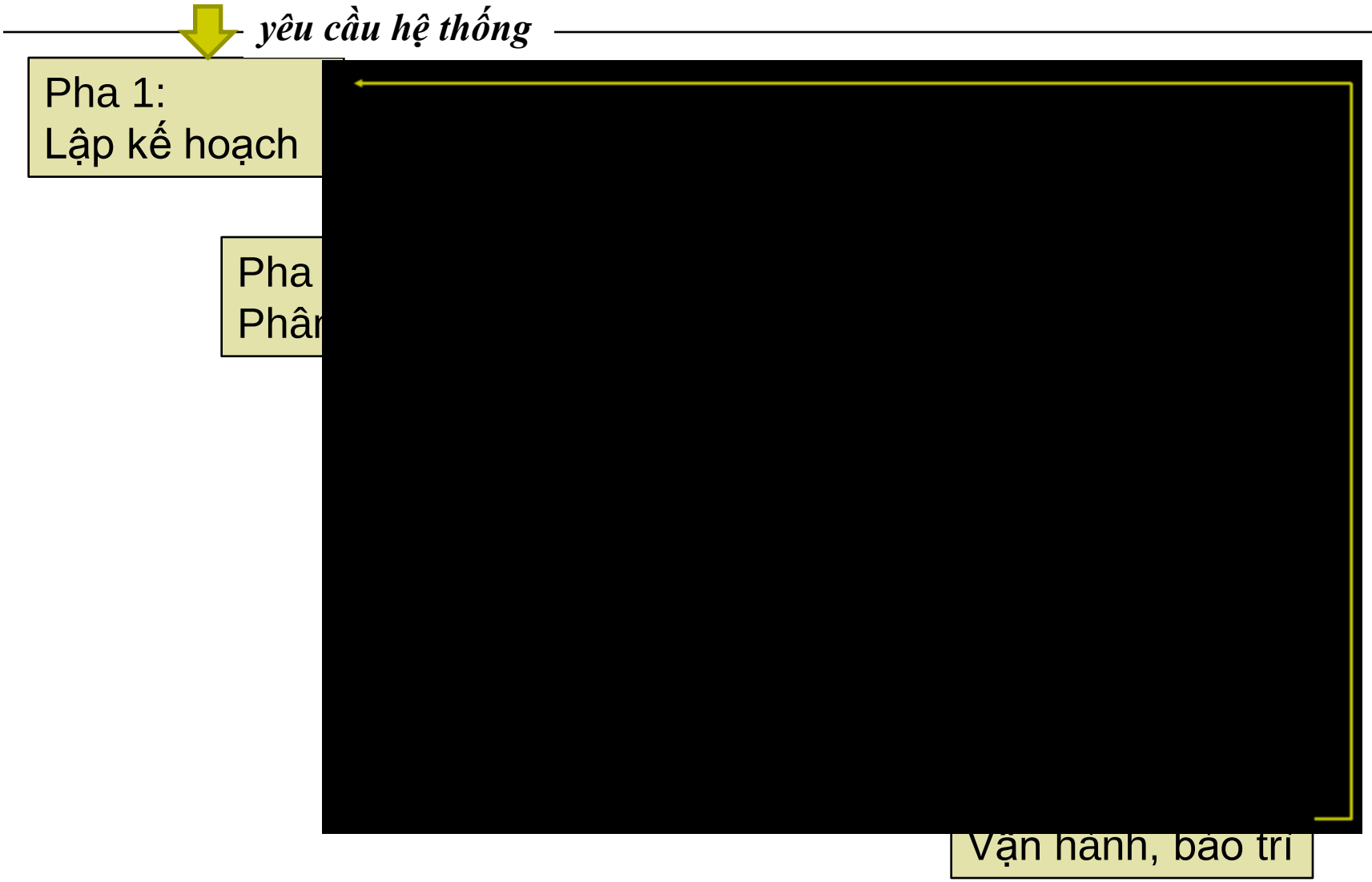


Xác định và phân tích yêu cầu

- Xác định hệ thống phải làm gì
  - yêu cầu chức năng và yêu cầu phi chức năng
- 3 kỹ thuật phân tích yêu cầu: BPA, BPI và BPR
  - Giá trị thực tế
  - Chi phí thực hiện
  - Rủi ro
- 5 kỹ thuật thu thập yêu cầu
  - *Phỏng vấn, JAD, bản câu hỏi, xem xét tài liệu, quan sát*
- Cần lựa chọn và phối hợp các kỹ thuật phù hợp

Vòng đời phát triển hệ thống

(*Systems Development Life Cycle* – SDLC)



Nhiệm vụ cụ thể của pha PTHT

- Xác định yêu cầu
- Phân tích yêu cầu
- Mô hình hóa dữ liệu và nghiệp vụ
- Chuyển sang thiết kế hệ thống

Yêu cầu là gì ?

- Gồm những chức năng hệ thống phải thực hiện
- Gồm những đặc điểm hệ thống phải có
- Tập trung vào nghiệp vụ của người dùng
- Có thể thay đổi trong quá trình phát triển hệ thống (qua các pha khác nhau)

Phân loại

- Yêu cầu chức năng (*Functional Requirements*)
  - Quy trình hệ thống phải thực hiện
  - Thông tin hệ thống phải lưu trữ và xử lý
- Yêu cầu phi chức năng (*Nonfunctional Requirements*)
  - Về vận hành
  - Về hiệu năng
  - Về an toàn bảo mật
  - Về thói quen, tập tục, các ràng buộc, …

5 loại yêu cầu

- Xuất (*output*)
- Nhập (*input*)
- Các quá trình (*process*)
- Hiệu suất (*performance*)
- Điều khiển (*control*)

Tài liệu hóa

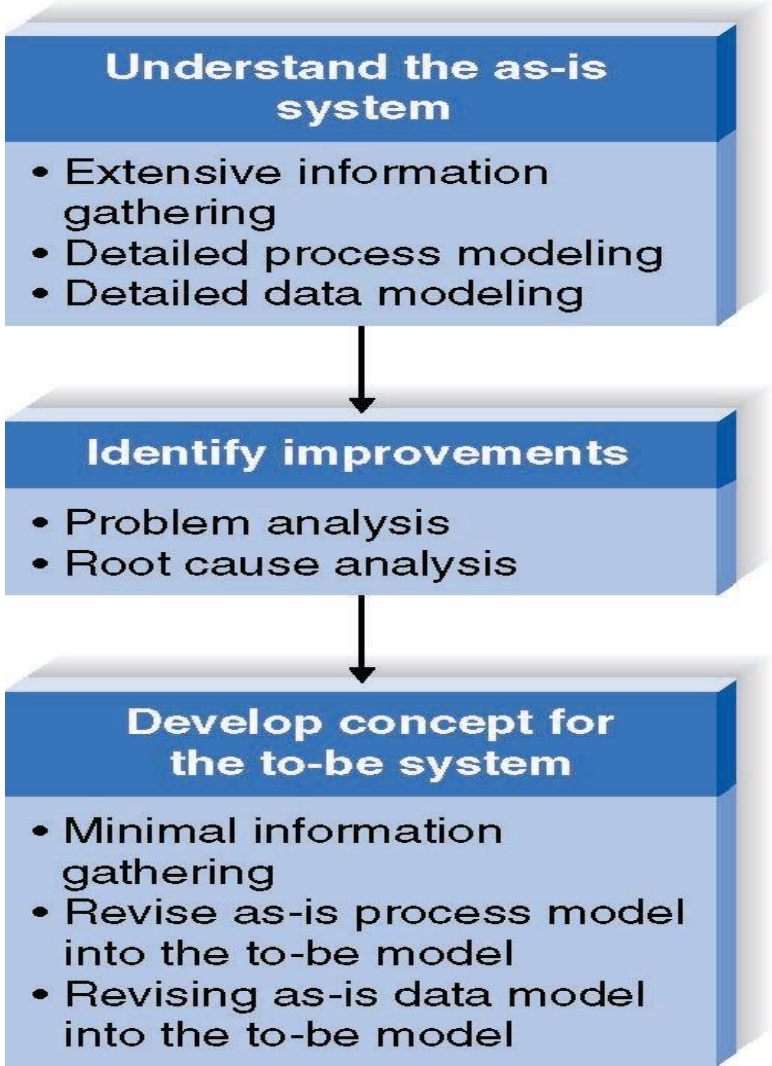
- Tài liệu đặc tả yêu cầu
  - Văn bản liệt kê danh sách các yêu cầu
  - Các đặc điểm, tính chất cần có
- Xác định phạm vi hệ thống
  - Những gì thuộc hệ thống
  - Những gì không thuộc hệ thống

Xác định yêu cầu

- Hiểu hệ thống hiện tại
- Xác định các thay đổi mong muốn thực hiện
- Xác định hệ thống cần xây dựng
- Phát hiện các yêu cầu cần có trong hệ thống mới
  - Business Process Automation (BPA)
    - Thích hợp với những thay đổi nhỏ
  - Business Process Improvement (BPI)
    - Thích hợp với thay đổi trung bình
  - Business Process Reengineering (BPR)
    - Thích hợp với thay đổi lớn

Tự động hóa quy trình nghiệp vụ (BPA)

Giúp hoạt động của người dùng trở nên hiệu quả hơn (*efficiency*)

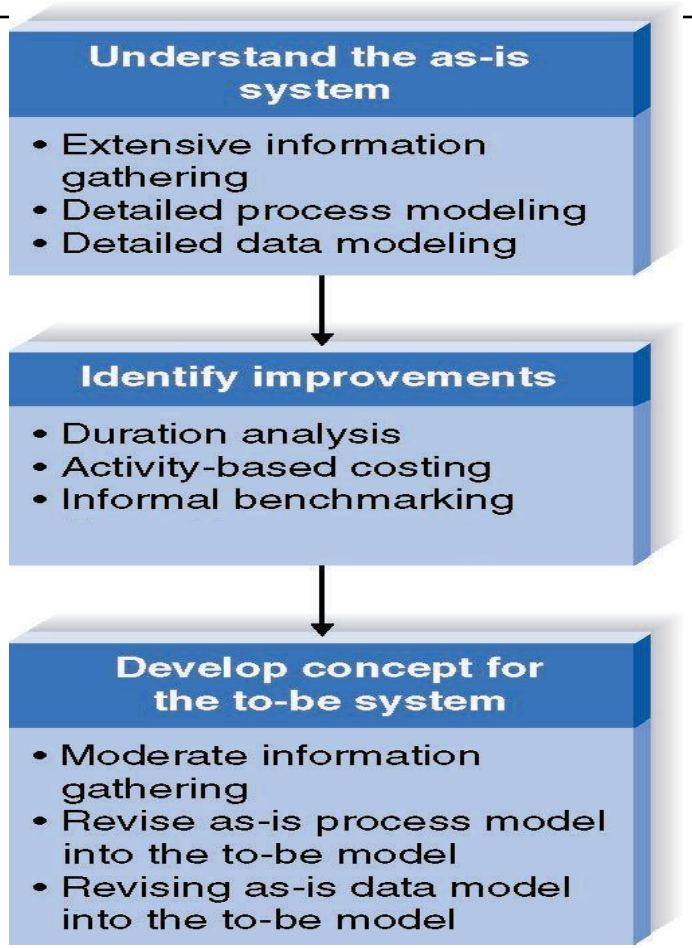


Xác định các khả năng cải thiện hệ thống hiện tại

- **Phân tích vấn đề**
  - Phỏng vấn người dùng
  - Xác định các vấn đề
  - Tìm giải pháp
- **Phân tích nguyên nhân**
  - Đặt giả thiết về lý do vấn đề tồn tại
  - Tìm hiểu các vấn đề ẩn sau
- ✓ Ví dụ

Cải thiện quy trình nghiệp vụ (BPI)

Giúp quy trình xử lý trở nên hiệu quả hơn (*efficiency & effectiveness*)



Phân tích về thời gian

- Tính toán thời gian cần cho mỗi bước xử lý
- Tính toán thời gian cần cho toàn bộ quy trình
- So sánh 2 chỉ số thời gian ➔ chênh lệch càng lớn thì khả năng cải thiện càng cao
- Giải pháp tiềm năng
  - Nhóm/tích hợp các bước xử lý ➔ giảm số lượng các bước xử lý trong quy trình
  - Song song hóa ➔ thay đổi quy trình để tăng các bước có thể xử lý song song

Phân tích về chi phí thực hiện

- Tính toán chi phí thực hiện mỗi bước xử lý
- Xem xét cả chi phí trực tiếp và chi phí gián tiếp
- Tập trung vảo việc giảm chi phí cho các bước có chi phí cao

Benchmarking

- Tìm hiểu cách thực hiện cùng 1 nghiệp vụ của các tổ chức khác nhau
- Đề xuất các thay đổi trong quy trình nghiệp vụ

Phân tích kết quả

- Xem xét những kết quả (*outcome*) mong muốn từ phía khách hàng
- Xem xét những "dịch vụ" đem lại cho khách hàng

Phân tích công nghệ

- Tìm hiểu các công nghệ liên quan
- Phân tích về khả năng áp dụng các công nghệ trong quy trình nghiệp vụ yêu cầu
- Phân tích về lợi ích của việc áp dụng các công nghệ này trong nghiệp vụ yêu cầu

Loại bỏ các xử lý “dư thừa”

- Xem xét ảnh hưởng của từng xử lý đối với hệ thống
  - Có thể loại bỏ một (bước) xử lý nào không?
- Kiểm tra các khả năng có thể

So sánh các kỹ thuật

- Giá trị thực tế (đối với tổ chức sử dụng, khai thác)
- Chi phí dự án
- Mức độ chi tiết của phân tích
- Rủi ro

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 2

CÔNG NGHỆ HƯỚNG ĐỐI TƯỢNG

2.1 Các khái niệm hướng đối tượng (nhắc lại)

Mục tiêu

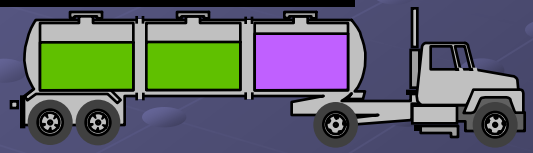
- Mô tả các khái niệm trừu tượng hóa, đóng gói, mô-đun hóa và phân cấp
- Mô tả cấu trúc vật lý của một lớp
- Mô tả mối quan hệ giữa lớp và đối tượng
- Hiểu về đa hình và tổng quát hóa

Nội dung

1. Đối tượng là gì?
2. Bốn nguyên lý của OO
3. Lớp là gì?
4. Đa hình và Tổng quát hóa
5. Tổ chức các phần tử mô hình

1. Đối tượng là gì?

- Một đối tượng biểu diễn một thực thể, có thể là thực thể vật lý, thực thể trừu tượng hoặc thực thể phần mềm.
  - Thực thể vật lý



Xe tải

  - Thực thể trừu tượng

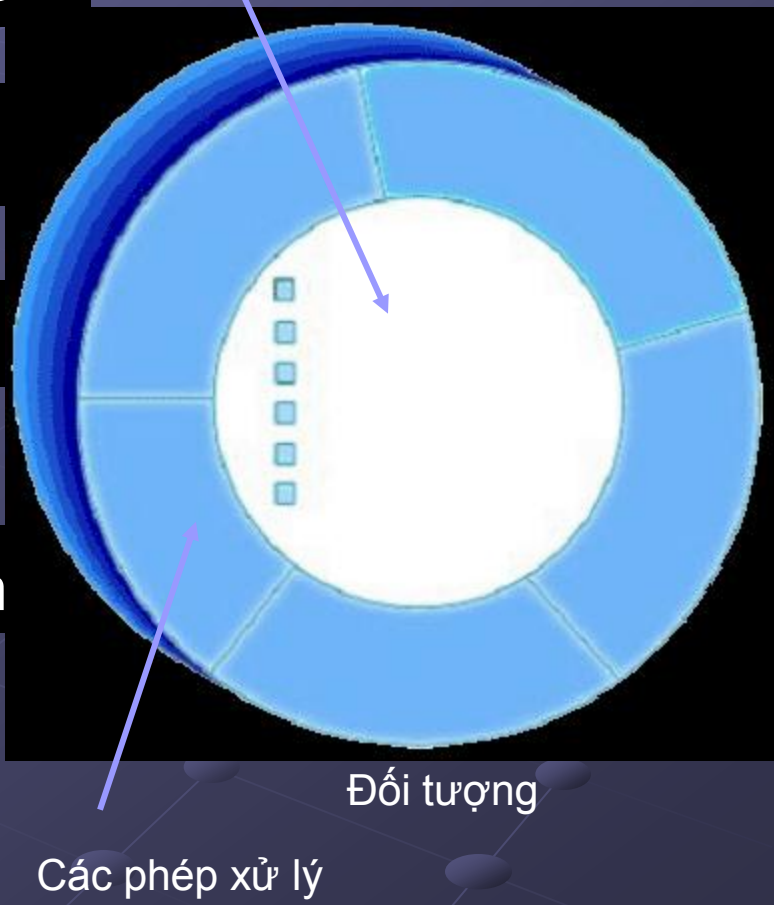
Phản ứng hóa học

  - Thực thể phần mềm

Danh sách liên kết

Định nghĩa

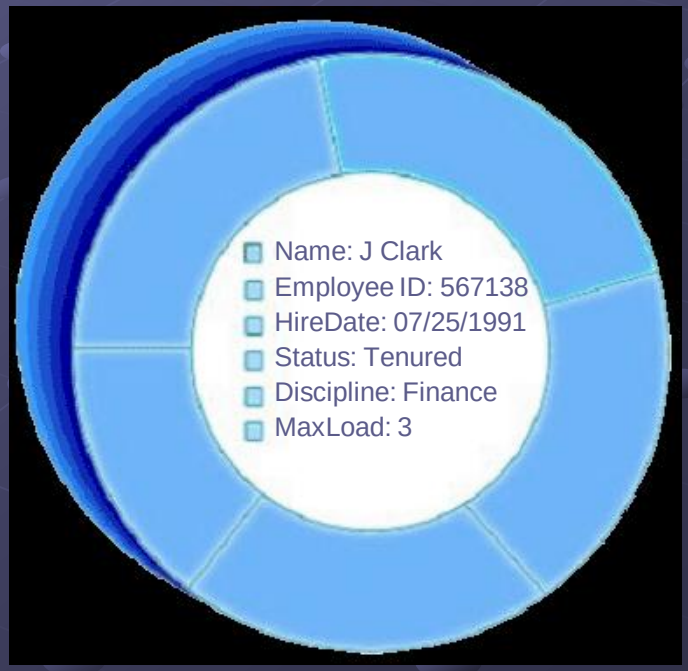
- Một đối tượng là một thực thể có biên và định danh được xác định rõ ràng để đóng gói trạng thái và hành vi.
  - Trạng thái biểu diễn thuộc tính và các mối quan hệ.
  - Hành vi là các thao tác, các phương thức và cơ chế chuyển trạng thái.



Một đối tượng có trạng thái

- Trạng thái là một điều kiện hay một tình huống trong suốt quá trình sống của đối tượng.
- Trạng thái của một đối tượng thường thay đổi theo thời gian.

Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes



Giáo sư Clark

Một đối tượng có hành vi

- Hành vi xác định cách mà một đối tượng hành động và phản ứng.
- Một hành vi có thể thấy được của một đối tượng được mô hình hóa bởi một tập các thông điệp mà nó có thể đáp ứng (các thao tác mà đối tượng có thể thực hiện).

Các hành vi của giáo sư Clark
- Submit Final Grades
- Accept Course Offering
- Take Sabbatical
- Set Max Load

Giáo sư Clark

Một đối tượng có định danh

- Mỗi đối tượng có một tên riêng để phân biệt đối tượng với các đối tượng khác mặc dù trạng thái của chúng có thể giống hệt nhau.

Giáo sư “J Clark”
dạy Sinh học

Giáo sư “J Clark”
dạy Sinh học

Nội dung?

1. Đối tượng là gì?
2. Bốn nguyên lý của OO
3. Lớp là gì?
4. Đa hình và Tổng quát hóa
5. Tổ chức các phần tử mô hình

2. Các nguyên lý cơ bản của OO

2.1. Trừu tượng hóa (Abstraction)

- Những đặc điểm cơ bản của một thực thể phân biệt nó với các loại thực thể khác.
- Xác định một biên giới liên quan đến góc độ của người quan sát.
- Nó không phải là một biểu hiện cụ thể, nó biểu thị bản chất của thực thể.

Ví dụ: Trừu tượng hóa

Sinh viên

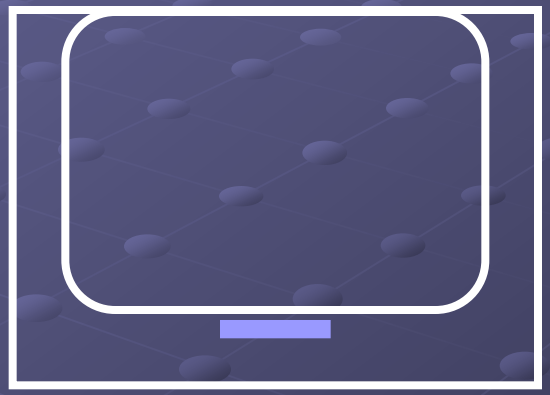
Giáo viên

Khóa học diễn ra lúc 9:00 sáng các ngày thứ 3, 5, 7

Khóa học (ví dụ đại số)

2.2. Đóng gói (Encapsulation)

- Che giấu sự thực thi bên trong
  - Client sử dụng giao diện được cung cấp



Tăng cường tính mềm dẻo

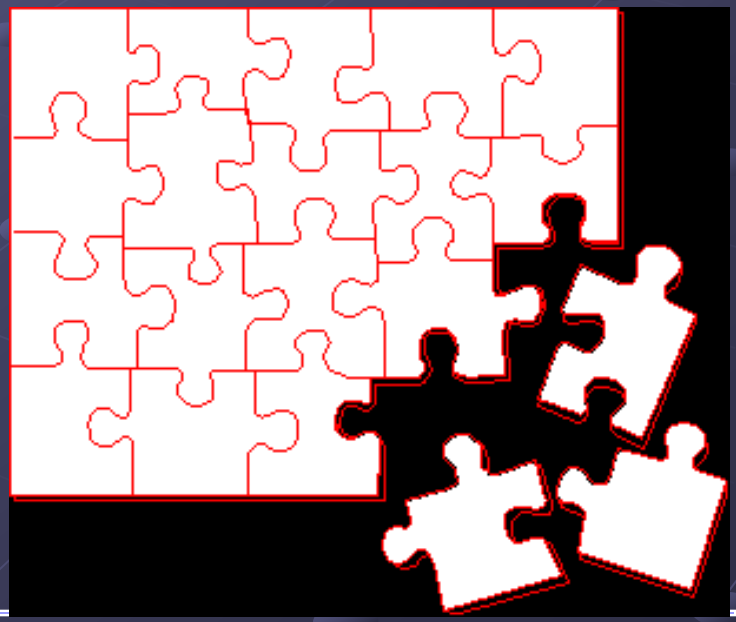
Minh họa việc đóng gói

- Giáo sư Clark được yêu cầu dạy 4 lớp tháng tới

SetMaxLoad(4)

2.3. Mô đun hóa (Modularity)

- Chia nhỏ hệ thống phức tạp thành những thành phần nhỏ có thể quản lý được.
- Cho phép người dùng hiểu biết về hệ thống.



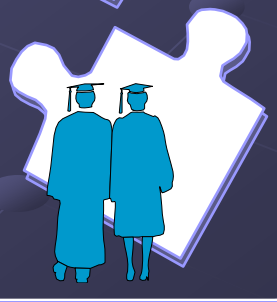
Ví dụ: Mô đun hóa

- Ví dụ, chia nhỏ một hệ thống phức tạp thành các mô đun nhỏ hơn.

Hệ thống quản lý siêu thị sách

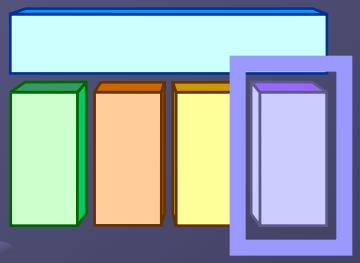
Hệ thống quản lý xuất nhập sách

Hệ thống quản lý thông tin sách



Hệ thống quản lý nhân viên

2.4. Phân cấp (Hierarchy)



Các phần tử ở cùng cấp trong sơ đồ phân cấp thì có cùng mức trừu tượng hóa

Nội dung?

1. Đối tượng là gì?
2. Bốn nguyên lý của OO
3. Lớp là gì?
4. Đa hình và Tổng quát hóa
5. Tổ chức các phần tử mô hình

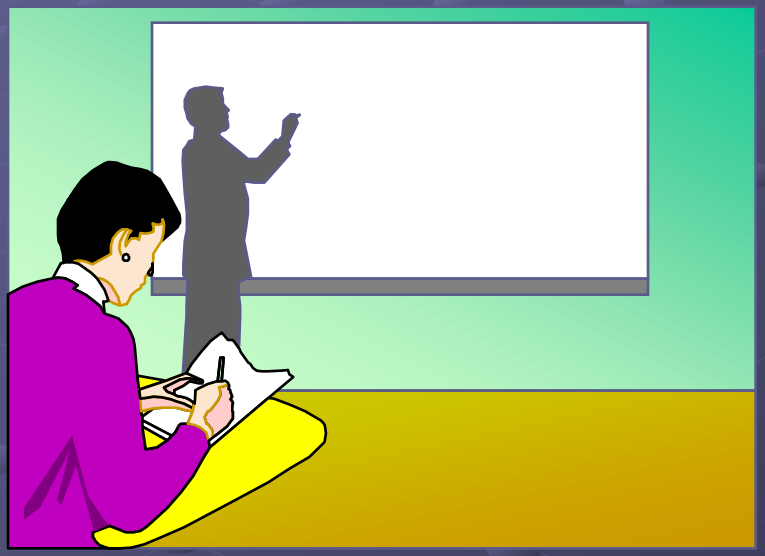
3. Lớp là gì?

- Lớp đại diện cho một tập các đối tượng. Các đối tượng này chung nhau các thuộc tính, hành vi, mối quan hệ và ngữ nghĩa
  - Một đối tượng là một thể hiện của lớp.
- Một lớp là một sự trừu tượng hóa, trong đó nó:
  - Tập trung vào các đặc tính chung
  - Bỏ đi các đặc tính khác

Ví dụ về một lớp

Lớp
Khóa học

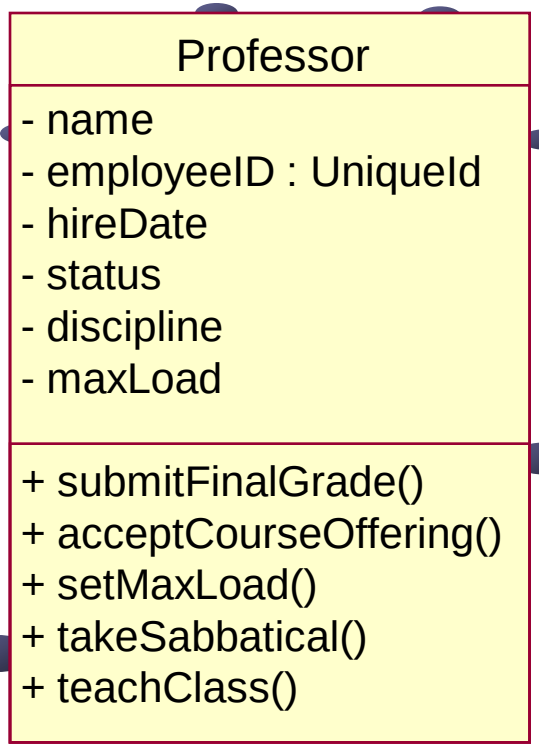
Thuộc tính
Tên
Địa điểm diễn ra
Số ngày dự kiến
Số đơn vị học trình
Thời gian bắt đầu
Thời gian kết thúc



Hành vi
Thêm sinh viên
Xóa sinh viên
Xem lịch học
Kiểm tra số thành viên

Biểu diễn lớp trong UML

- Trong UML, một lớp được biểu diễn bằng một hình chữ nhật, chia làm 3 phần.
  - Tên lớp
  - Cấu trúc (Các thuộc tính)
  - Hành vi (các phương thức)



Biểu diễn đối tượng trong UML

- Trong UML, một đối tượng được biểu diễn bằng một hình chữ nhật, với tên đối tượng được gạch chân.

Giáo sư J Clark

J Clark : Professor

Đối tượng J Clark thuộc lớp Professor

: Professor

Đối tượng nặc danh

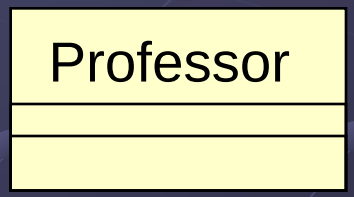
Mối quan hệ gữa lớp và đối tượng

- Lớp là sự trừu tượng hóa của đối tượng
  - Một lớp định nghĩa cấu trúc và hành vi cho tất cả các đối tượng thuộc lớp đó
  - Nó có chức năng như là khuôn mẫu cho việc tạo đối tượng
- Lớp không phải là tập hợp các đối tượng

Professor Torpie

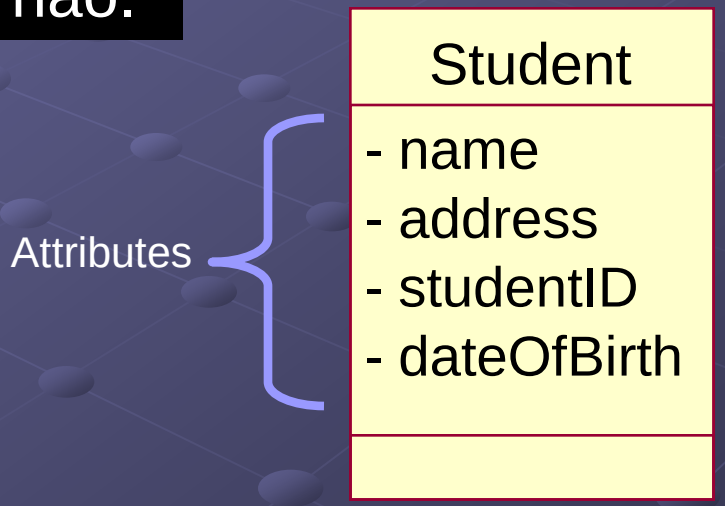
Professor Meijer

Professor Allen

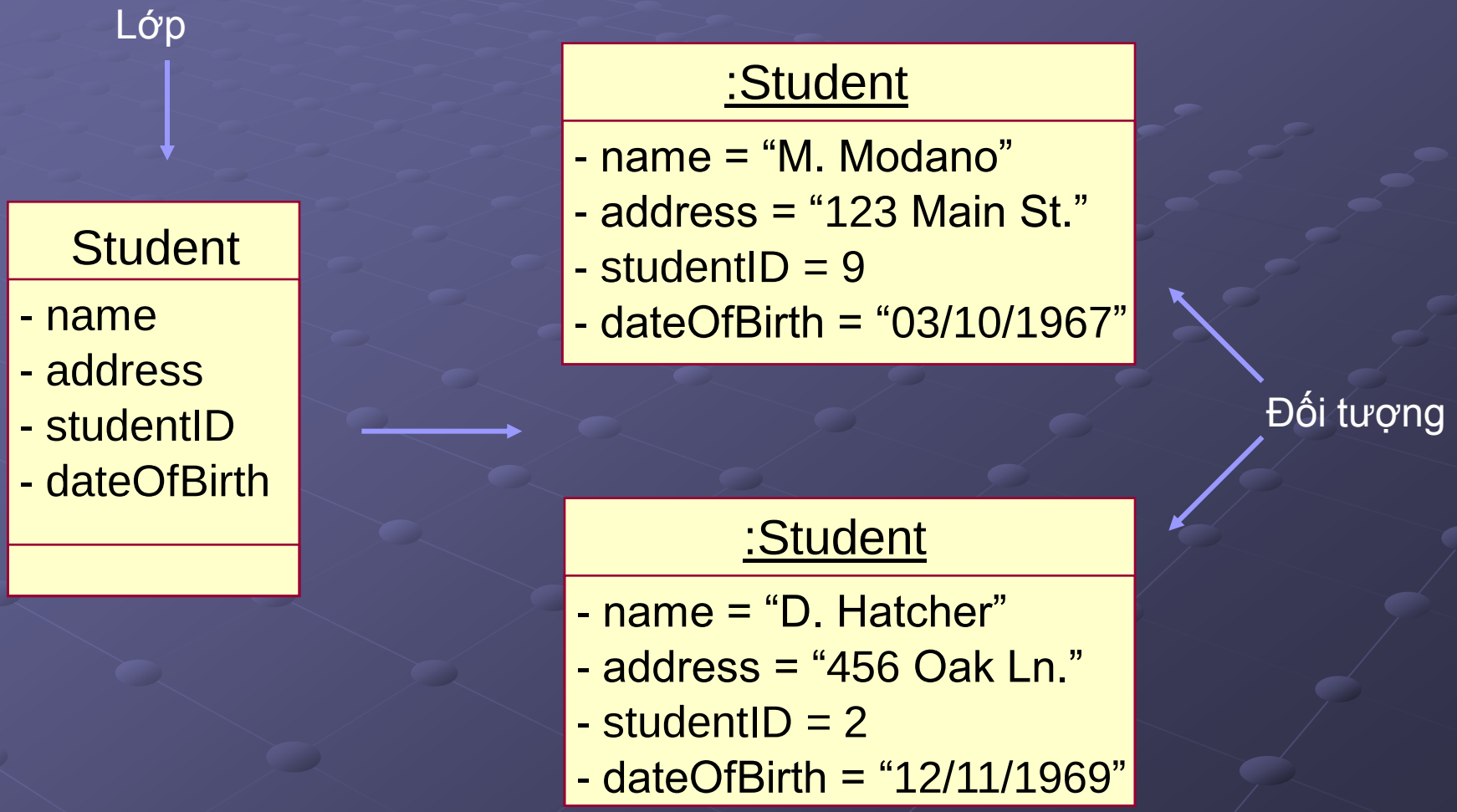


Thuộc tính (attribute) là gì?

- Một thuộc tính là một đặc tính (property) có tên của một lớp mô tả phạm vi các giá trị mà các thể hiện của đặc tính đó có thể giữ
  - Một lớp có thể có một số thuộc tính hoặc không có thuộc tính nào.

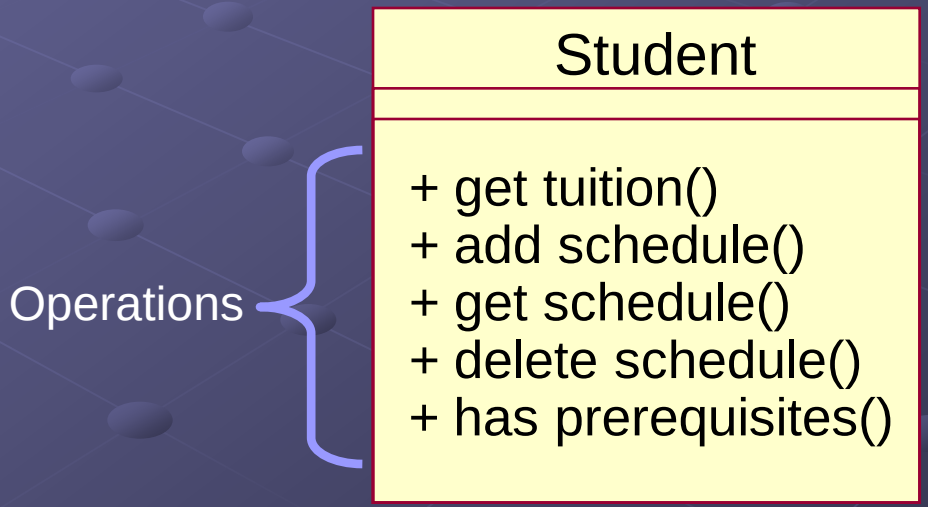


Thuộc tính trong lớp và trong đối tượng



Thao tác là gì?

- Một dịch vụ có thể được yêu cầu từ một đối tượng để thực hiện hành vi. Một thao tác có một chữ ký, có thể giới hạn các tham số thực tế có thể
- Một lớp có thể có nhiều thao tác hoặc không có thao tác nào.

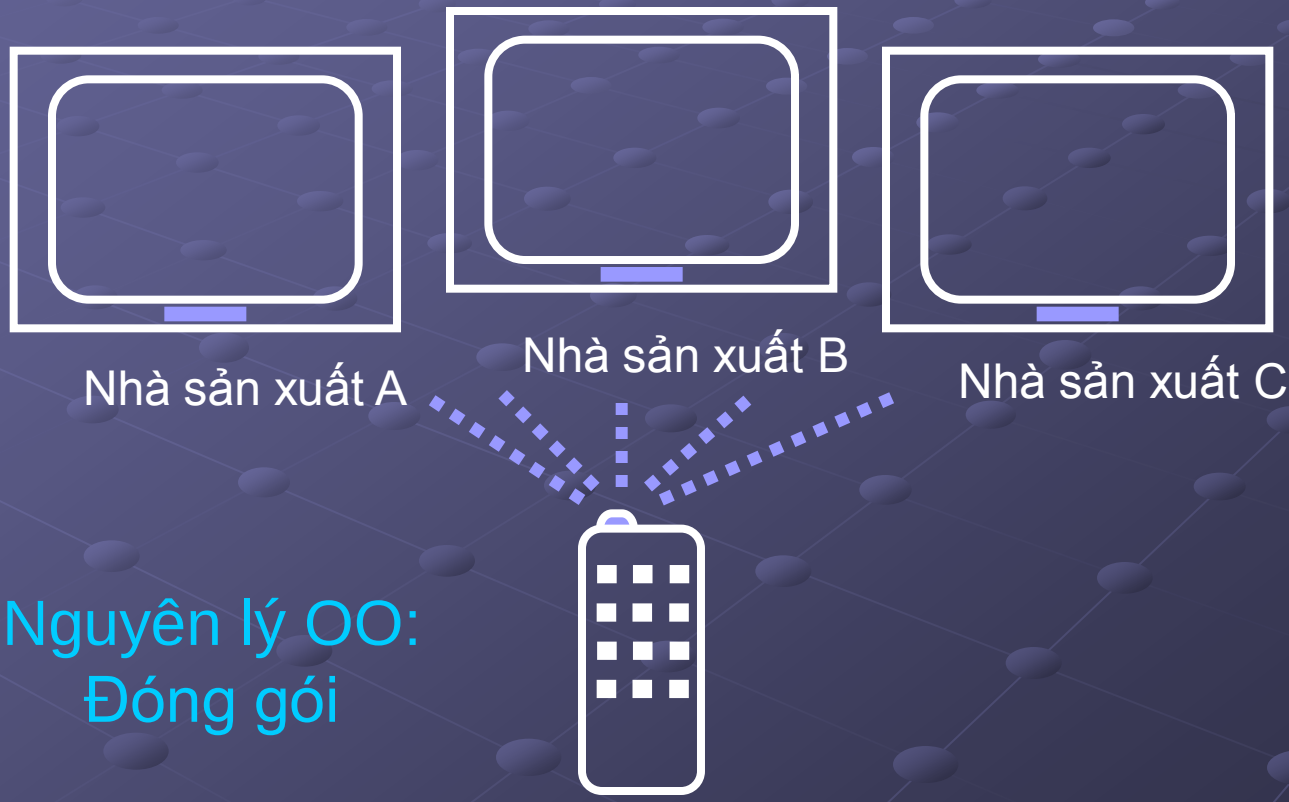


Nội dung?

1. Đối tượng là gì?
2. Bốn nguyên lý của OO
3. Lớp là gì?
4. Đa hình và Tổng quát hóa
5. Tổ chức các phần tử mô hình

4.1. Đa hình là gì?

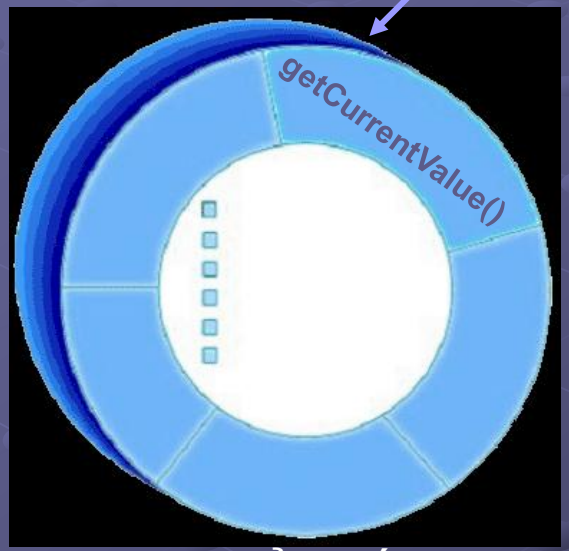
- Khả năng che giấu rất nhiều sự thực thi thông qua một giao diện.



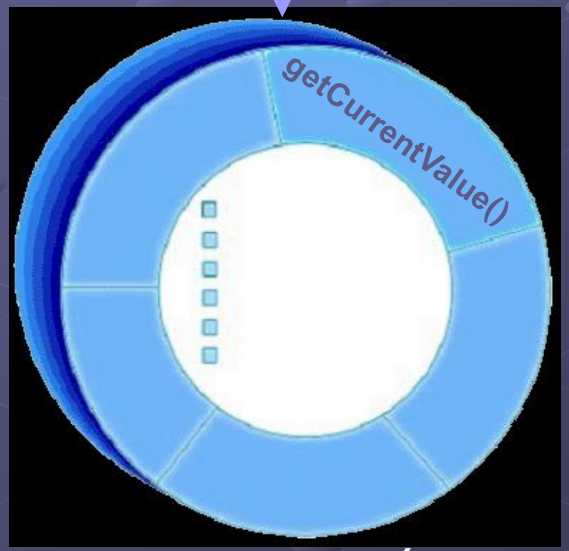
Nguyên lý OO:
Đóng gói

Ví dụ: Đa hình

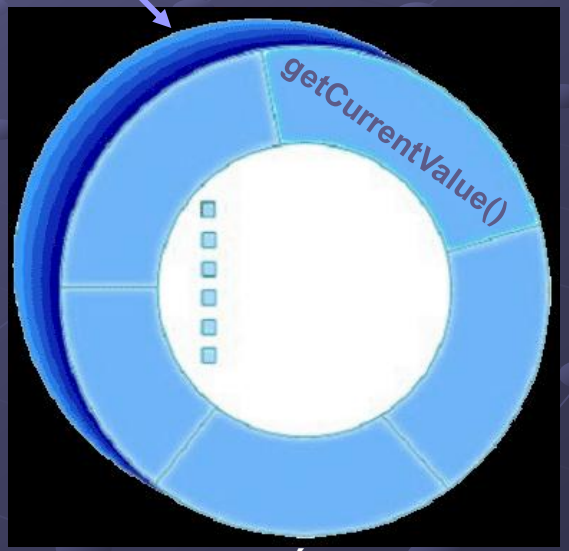
financialInstrument.getCurrentValue()



Cổ phiếu



Trái phiếu



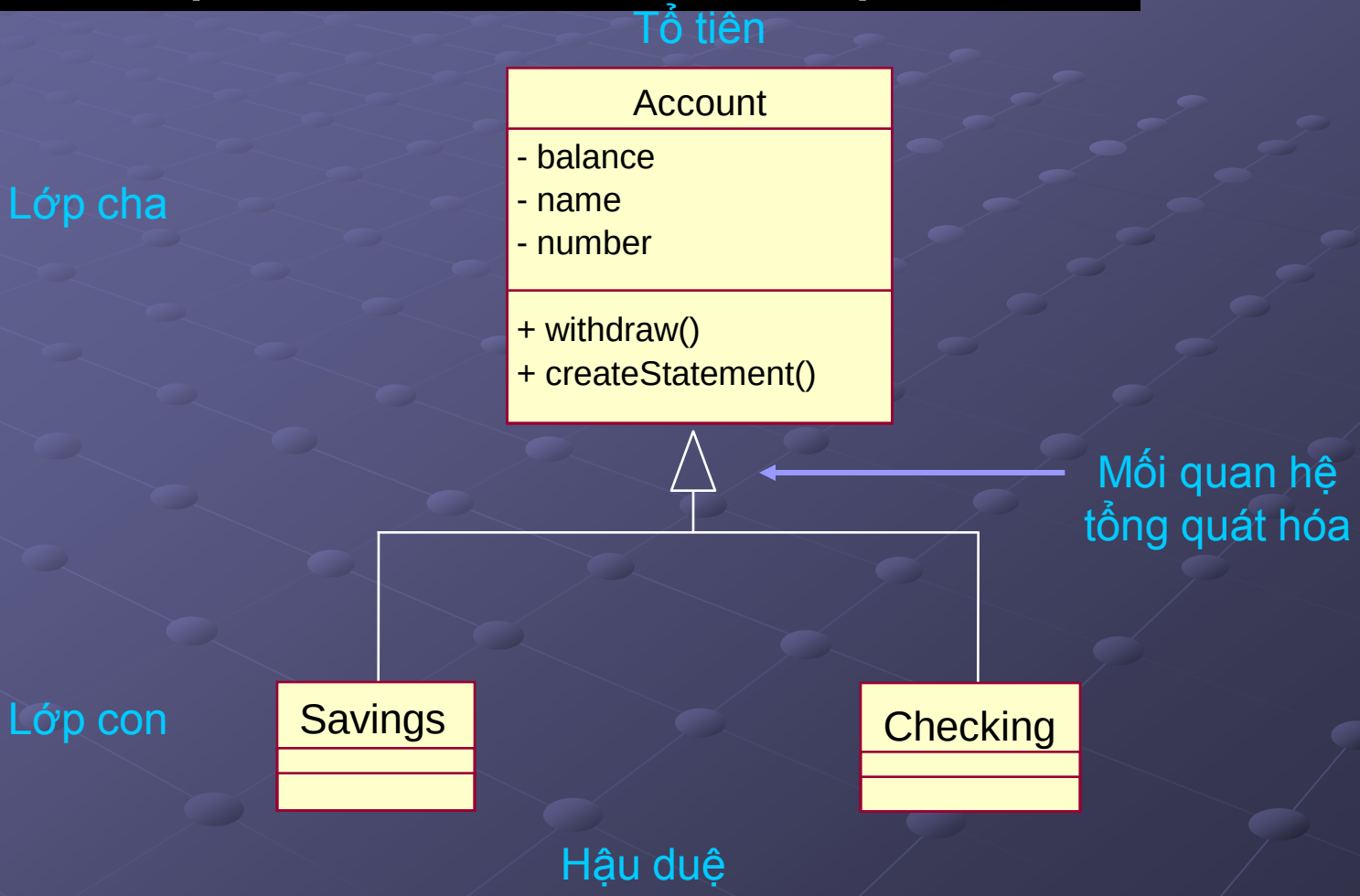
Quỹ cố định

4.2. Tổng quát hóa là gì?

- Mối quan hệ giữa các lớp trong đó một lớp chia sẻ cấu trúc và hành vi của nó cho một hoặc một số lớp khác.
- Định nghĩa một cấu trúc phân cấp của các mức trừu tượng trong đó, các lớp con kế thừa từ một hoặc một số lớp cha.
  - Đơn kế thừa
  - Đa kế thừa
- Tổng quát hóa là một loại quan hệ

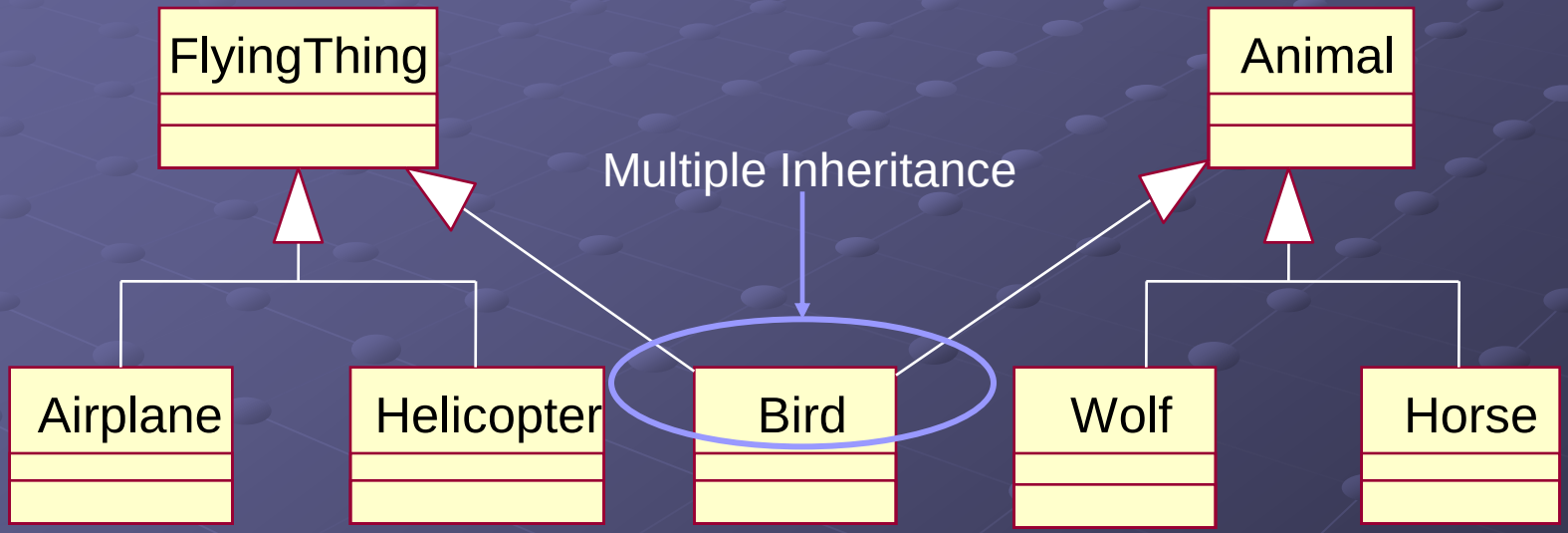
Ví dụ: Đơn kế thừa

- Một lớp kế thừa từ một lớp khác



Ví dụ: Đa kế thừa

- Một lớp có thể kế thừa từ nhiều lớp khác.



Sử dụng đa kế thừa chỉ khi nào thực sự cần và phải luôn cảnh giác!

Kế thừa những gì?

- Một lớp kế thừa từ lớp cha các thuộc tính, phương thức và các mối quan hệ.
- Một lớp con có thể :
  - Thêm các thuộc tính , phương thức và các mối quan hệ mới.
  - Định nghĩa lại các phương thức (cẩn thận)
- Các thuộc tính, phương thức và các mối quan hệ chung được chỉ ra ở mức cao nhất trong cây phân cấp.

Inheritance leverages the similarities among classes.

Nội dung?

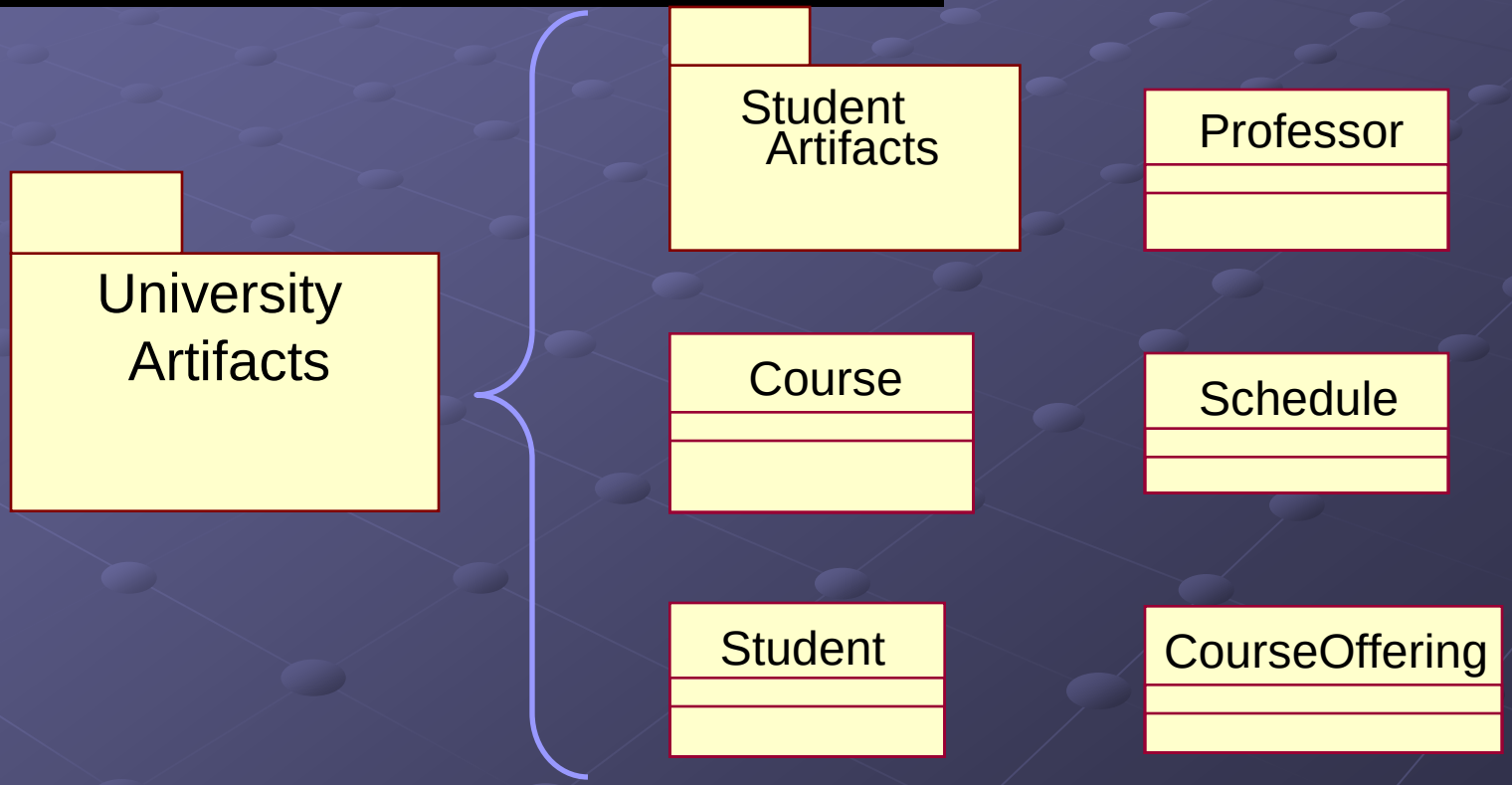
1. Đối tượng là gì?
2. Bốn nguyên lý của OO
3. Lớp là gì?
4. Đa hình và Tổng quát hóa
5. Tổ chức các phần tử trong mô hình

Package là gì?

- Là một công cụ để tổ chức các phần tử trong mô hình vào thành từng nhóm.
- Các phần tử trong mô hình có thể chứa đựng các phần tử khác.
- Một package có thể được sử dụng để:
  - Tổ chức các mô hình đang phát triển.
  - Tạo ra một khối để quản lý cấu hình.

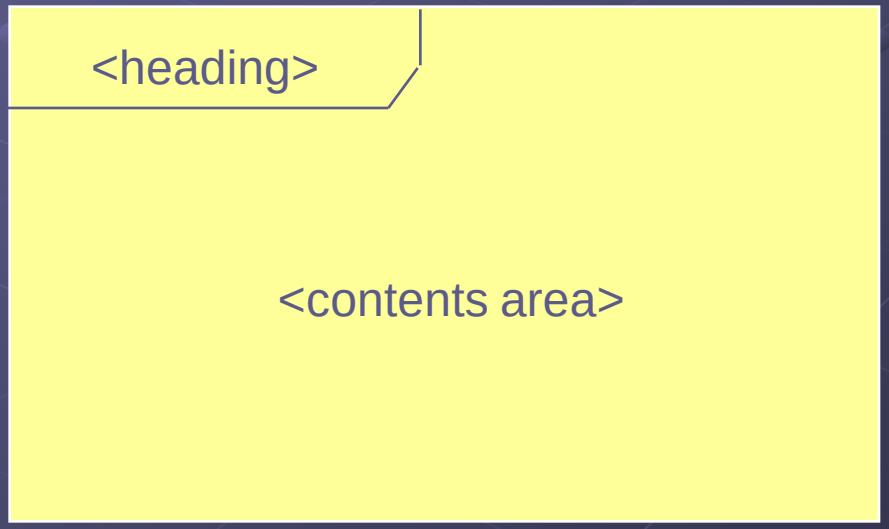
Một package có thể chứa đựng các lớp

- Trong package University Artifacts, chứa một package và 5 lớp.



Vẽ biểu đồ

- Mỗi biểu đồ có một khung (frame), một ngăn tiêu đề ở phía góc trên bên trái và một vùng nội dung
  - Nếu khung đó không cung cấp thêm giá trị gì thì nó có thể bỏ qua.



Thảo luận

- Đối tượng là gì?
- 4 nguyên lý của hướng đối tượng? Giải thích.
- Lớp là gì? Lớp và đối tượng liên quan đến nhau như thế nào?
- Thuộc tính là gì? Phương thức?
- Định nghĩa đa hình.Cung cấp một ví dụ về đa hình.
- Tổng quát hóa là gì?
- Tại sao phải sử dụng package?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

$6-

Bài 2 (tiếp)

2.1 Công nghệ đối tượng và UML

Unified Modeling Laguage

Mục tiêu

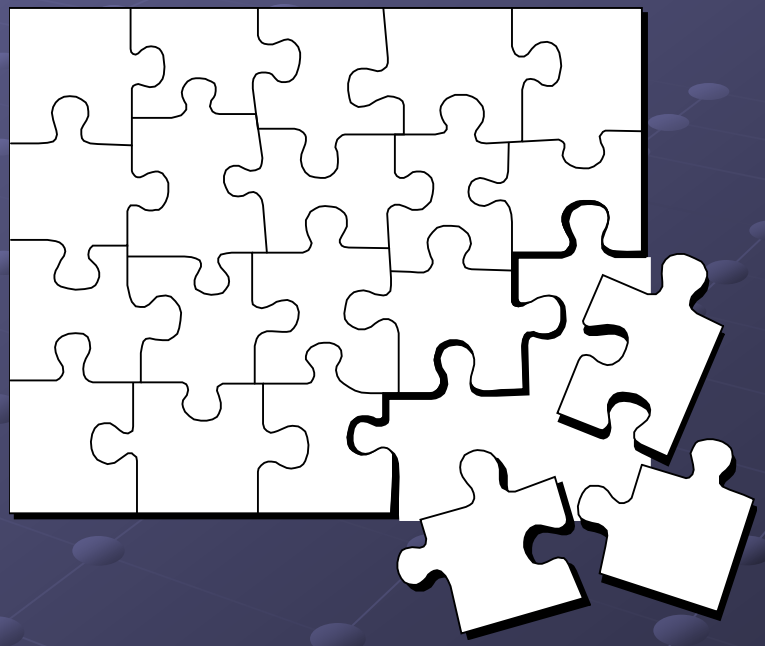
- Hiểu được về công nghệ đối tượng và chỉ ra được các ưu thế của công nghệ này.
- Nắm được lịch sử của công nghệ đối tượng và xu thế sử dụng hế công nghệ đối tượng
- Mô tả tầm quan trọng của mô hình hóa trực quan và vai trò của Kiến trúc hướng mô hình (Model Driven Architecture)
- Xác định 4 nguyên tắc của mô hình hóa trực quan
- Nắm được vai trò của UML
- Xác định loại quy trình phù hợp nhất với UML

Nội dung

1. Công nghệ đối tượng
2. Các nguyên tắc mô hình hóa trực quan
3. Ngôn ngữ mô hình hóa thống nhất UML

1.1. Công nghệ đối tượng là gì?

- Một tập các quy tắc (trừu tượng hóa, đóng gói, đa hình), hướng dẫn để xây dựng phần mềm, cùng với ngôn ngữ, cơ sở dữ liệu và các công cụ khác hỗ trợ các quy tắc này (*Object Technology - A Manager's Guide*, Taylor, 1997)

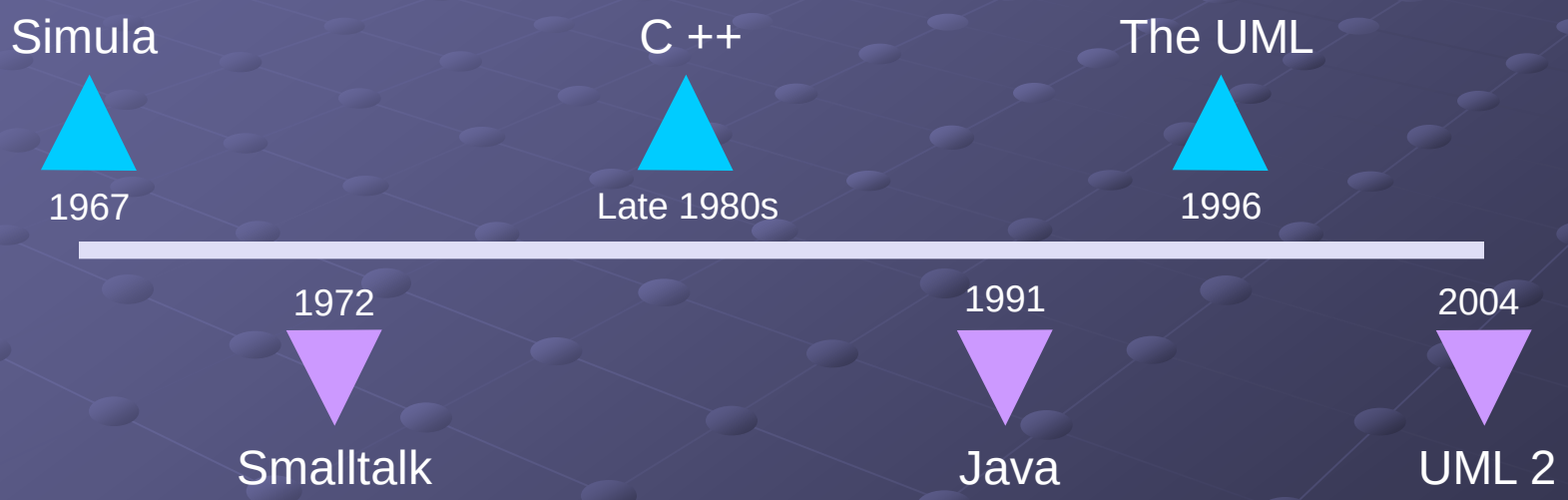


Lợi điểm của Công nghệ đối tượng

- Giúp tái sử dụng mã nguồn và kiến trúc
- Phản ánh gần hơn các mô hình trong thế giới thực
- Ổn định hơn
- Có khả năng thích ứng với thay đổi

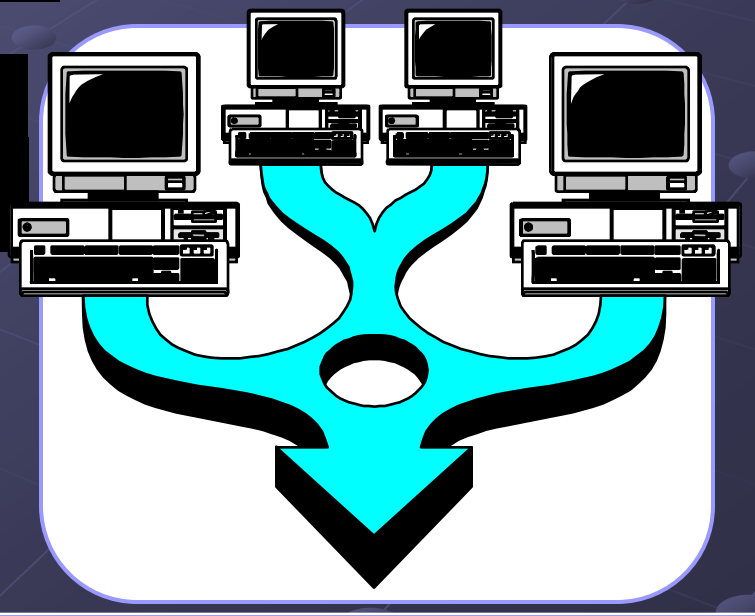
Lịch sử của Công nghệ đối tượng

- Các mốc chính của công nghệ đối tượng



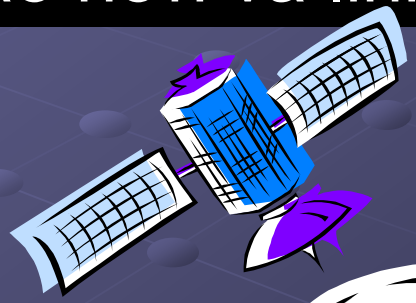
1.2. Công nghệ đối tượng được sử dụng ở đâu?

- Các hệ thống Client/Server và phát triển Web
  - Công nghệ đối tượng cho phép các công ty đóng gói thông tin doanh nghiệp trong các đối tượng và giúp phân phối quá trình xử lý qua mạng Internet hoặc một mạng máy tính.



1.2. Công nghệ đối tượng được sử dụng ở đâu? (2)

- Các hệ thống thời gian thực (real-time)
  - Công nghệ đối tượng cho phép các hệ thống thời gian thực có thể phát triển với chất lượng cao hơn và linh hoạt hơn



Hướng đối tượng và Thiết kế cấu trúc

- Hướng đối tượng (OO)
  - Kết hợp sớm dữ liệu và xử lý luồng dữ liệu cùng nhau trong vòng đời phần mềm
  - Có mức đóng gói cao
  - Đẩy mạnh khả năng tái sử dụng mã nguồn một cách hiệu quả
  - Cho phép mở rộng phần mềm hơn nữa

Nội dung

1. Công nghệ đối tượng
2. Các nguyên tắc mô hình hóa trực quan
3. Ngôn ngữ mô hình hóa thống nhất UML

2.1. Mô hình là gì?

- Mô hình là sự đơn giản hóa các vật thể, các đối tượng trong thế giới thực.

Tai sao phải mô hình hóa?

- Mô hình hóa thực hiện 4 mục tiêu:
  - Giúp chúng ta có một cái nhìn trực quan về hệ thống.
  - Cho phép chúng ta chỉ rõ cấu trúc hoặc hành vi của hệ thống.
  - Mô hình cho chúng ta một khuôn mẫu để xây dựng hệ thống.
  - Ghi lại các quyết định của chúng ta.

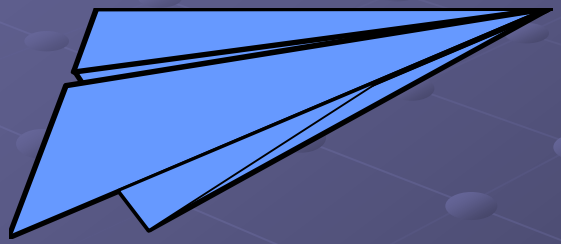
Tai sao phải mô hình hóa? (2)

- Chúng ta phải xây dựng các mô hình cho một hệ thống phức tạp vì chúng ta không thể hiểu toàn bộ hệ thống đó.
- Xây dựng các mô hình giúp chúng ta hiểu biết sâu sắc hơn về hệ thống.

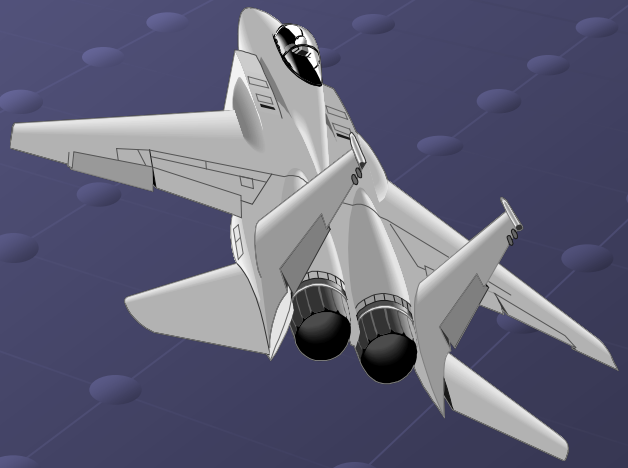
Sự quan trọng của mô hình hóa

Mức độ quan trọng thấp

Mức độ quan trọng cao hơn



Máy bay giấy



Máy bay phản lực

Đội dự án thường không mô hình hóa

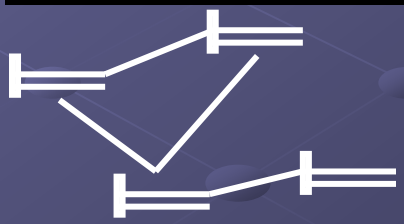
- Rất nhiều đội dự án tiến hành xây dựng ứng dụng theo hướng tiếp cận của việc gấp máy bay giấy.
  - Bắt đầu code ngay khi có được yêu cầu.
  - Mất rất nhiều thời gian và tạo ra rất nhiều mã nguồn.
  - Không có bất kỳ một kiến trúc nào.
  - Phải chịu khổ với những lỗi phát sinh.
- Mô hình hóa là một con đường dẫn đến thành công của dự án.

2.2. Bốn nguyên tắc của mô hình hóa

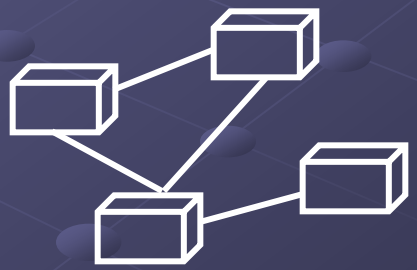
- Mô hình tạo ra chi phối cách thức vấn đề bắt đầu và cách thức hình thành giải pháp
- Mỗi mô hình diễn tả hệ thống với một mức độ chi tiết khác nhau.
- Các mô hình tốt nhất phải bám sát vào thực tế.
- Một mô hình đơn lẻ không đủ để diễn tả toàn bộ hệ thống.

Nguyên tắc 1: Việc lựa chọn mô hình rất quan trọng

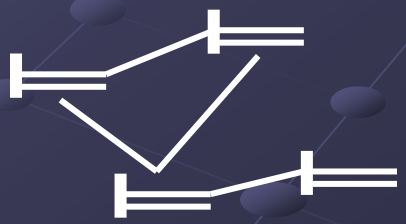
- Các mô hình giúp chúng ta hiểu biết sâu sắc về vấn đề, cách thức vấn đề được giải quyết và giải pháp đưa ra là gì.
  - Trong phần mềm, các mô hình chịu ảnh hưởng lớn bởi góc nhìn của bạn.
  - Mỗi góc nhìn sẽ dẫn đến một loại hệ thống khác nhau.



Process Model



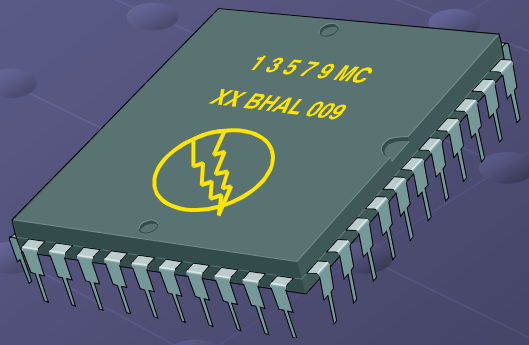
Deployment Model



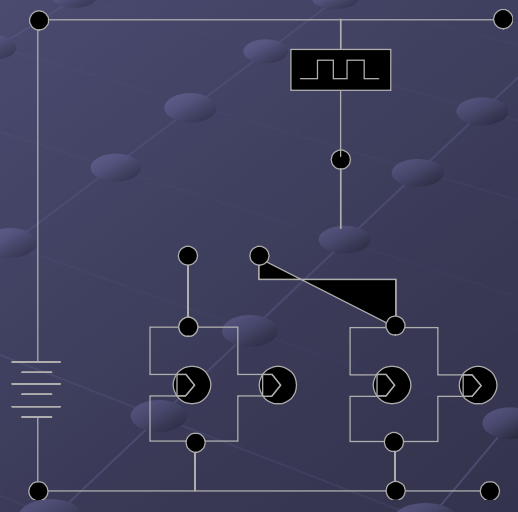
Design Model

Nguyên tắc 2: Mức độ chi tiết có thể khác nhau

- Mỗi mô hình diễn tả hệ thống với những mức độ chi tiết khác nhau.
  - Mức độ chi tiết của mô hình mà bạn lựa chọn phụ thuộc vào việc trả lời câu hỏi:
    - Ai là người sẽ xem mô hình.
    - Tại sao họ lại cần xem nó.



Mô hình cho khách hàng



Mô hình cho nhà thiết kế

Nguyên tắc 3: Các mô hình tốt nhất phải bám sát thực tế

- Tất cả các mô hình đều đơn giản hóa đối tượng trong thế giới thật.
- Một mô hình tốt phải phản ánh được những điểm ẩn chứa rủi ro.

Nguyên tắc 4: Một mô hình không đủ

- Không có mô hình đơn lẻ nào lại có thể mô tả đầy đủ về hệ thống.
- Cách tiếp cận tốt nhất đến các hệ thống không tầm thường là sử dụng một số mô hình gần như độc lập với nhau.
  - Tạo ra các mô hình có thể xây dựng và nghiên cứu độc lập nhưng vẫn có mối quan hệ tương quan lẫn nhau.

Nội dung

1. Công nghệ đối tượng
2. Các nguyên tắc mô hình hóa trực quan
3. Ngôn ngữ mô hình hóa thống nhất UML

3.1. UML là gì?

Unified Modeling Laguage

- UML là ngôn ngữ để:
  - Trực quan hóa (Visualizing)
  - Xác định rõ (Đặc tả - Specifying)
  - Xây dựng (Constructing)
  - Tài liệu hóa (Documenting)

các cấu phần (artifact) của một hệ thống phần mềm.

UML là ngôn ngữ trực quan

- UML là ngôn ngữ thống nhất trực quan giúp công việc được xử lý nhất quán, giảm thiểu lỗi xảy ra
  - Có những thứ mà nếu không mô hình hóa thì không hoặc khó có thể hiểu được
  - Mô hình trợ giúp hiệu quả trong việc liên lạc, trao đổi
    - Trong tổ chức
    - Bên ngoài tổ chức

UML là ngôn ngữ để đặc tả

- UML xây dựng các mô hình chính xác, rõ ràng và đầy đủ.

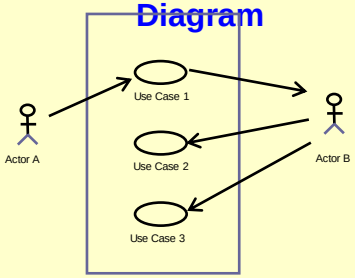
UML là ngôn ngữ để xây dựng HT

- Các mô hình UML có thể kết nối trực tiếp với rất nhiều ngôn ngữ lập trình.
  - Ánh xạ sang Java, C++, Visual Basic…
  - Các bảng trong RDBMS hoặc kho lưu trữ trong OODBMS
  - Cho phép các kỹ nghệ xuôi (chuyển UML thành mã nguồn)
  - Cho phép kỹ nghệ ngược (xây dựng mô hình hệ thống từ mã nguồn)

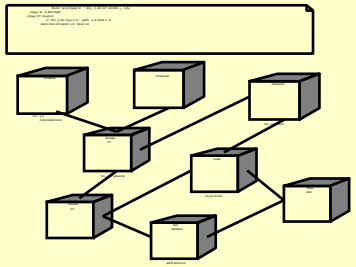
UML là ngôn ngữ để tài liệu hóa

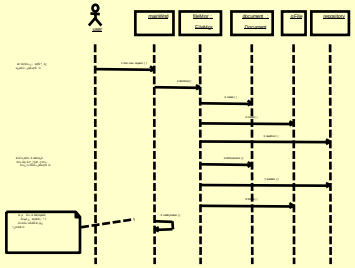
- UML giúp tài liệu hóa về kiến trúc, yêu cầu, kiểm thử, lập kế hoạch dự án, và quản lý việc bàn giao phần mềm
- Các biểu đồ khác nhau, các ghi chú, ràng buộc được đặc tả trong tài liệu

Use Case Diagram

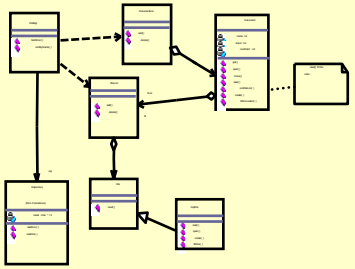


Deployment Diagram





Sequence Diagram



Class Diagram

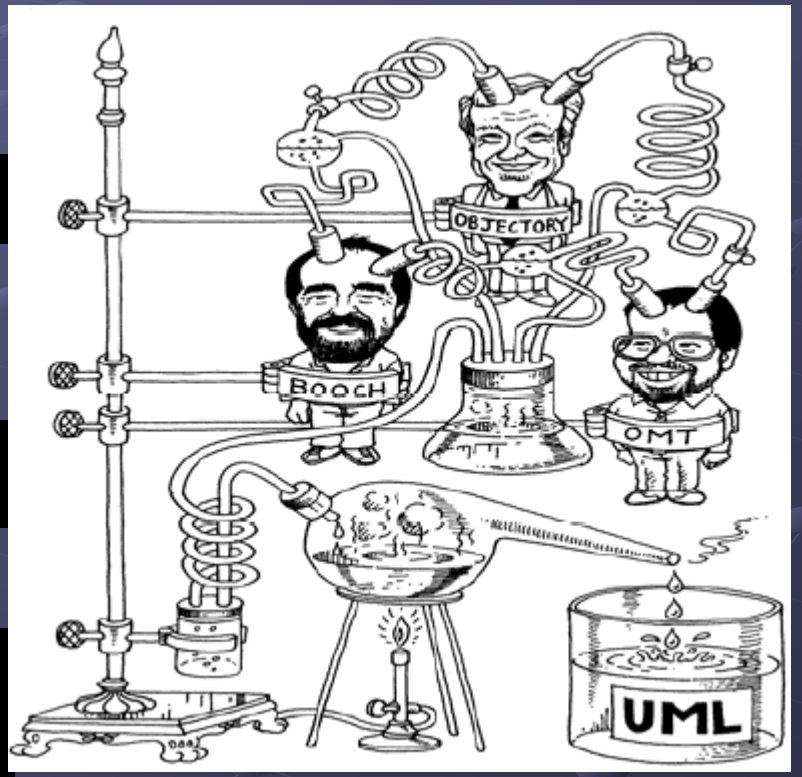
3.2. Lịch sử phát triển của UML

- Vào 1994, có hơn 50 phương pháp mô hình hóa hướng đối tượng:
  - Fusion, Shlaer-Mellor, ROOM, Class-Relation,Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS …
- "Meta-models" tương đồng với nhau
- Các ký pháp đồ họa khác nhau
- Quy trình khác nhau hoặc không rõ ràng

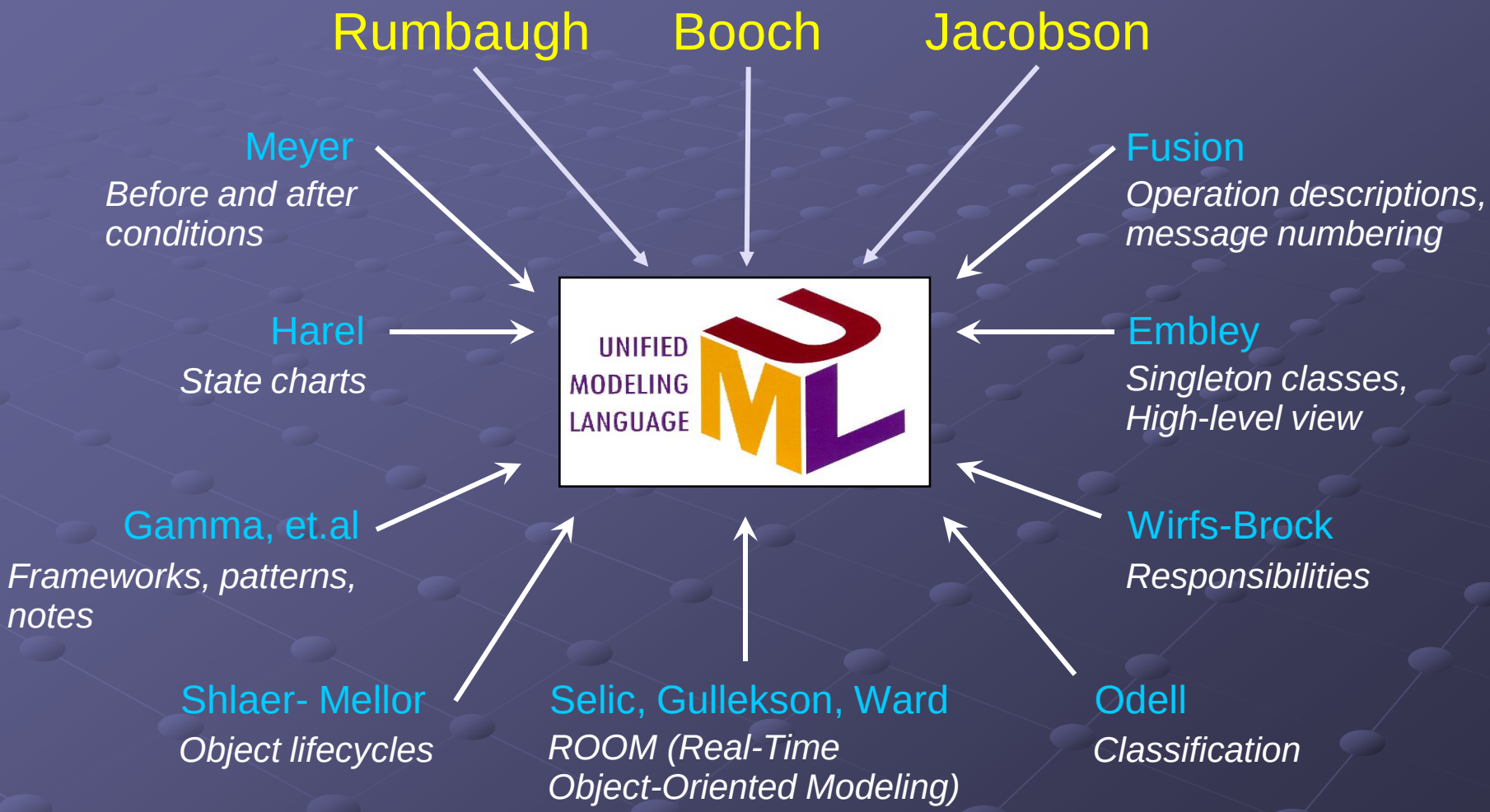
→ Cần chuẩn hóa và thống nhất các phương pháp

3.2. Lịch sử phát triển của UML (2)

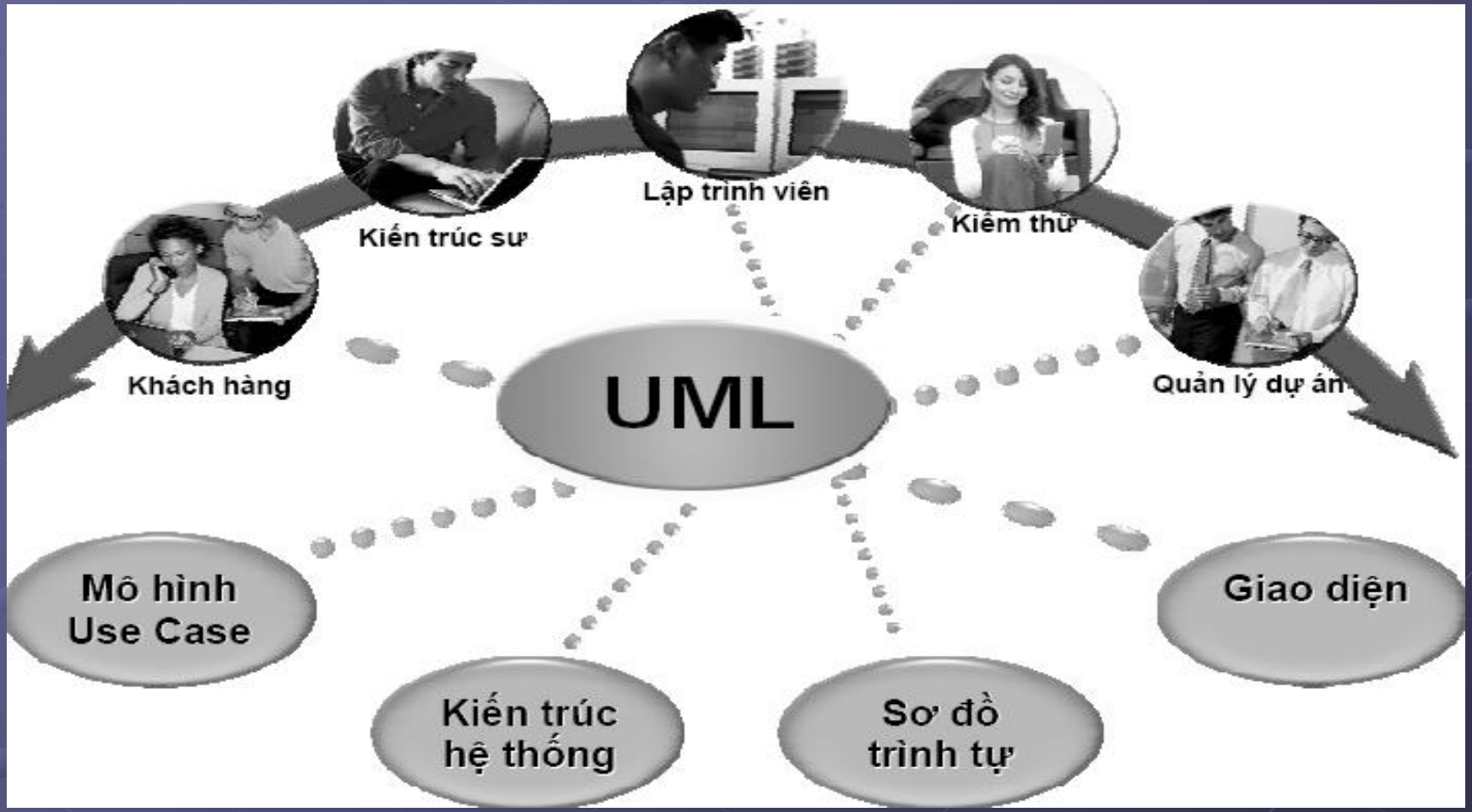
- UML được 3 chuyên gia hướng đối tượng hợp nhất các kỹ thuật của họ vào năm 1994:
  - Booch91 (Grady Booch): Conception, Architecture
  - OOSE (Ivar Jacobson): Use cases
  - OMT (Jim Rumbaugh): Analysis
- Thiết lập một phương thức thống nhất để xây dựng và "vẽ" ra các yêu cầu và thiết kế hướng đối tượng trong quá trình PTTK phần mềm → UML được công nhận là chuẩn chung vào năm 1997.



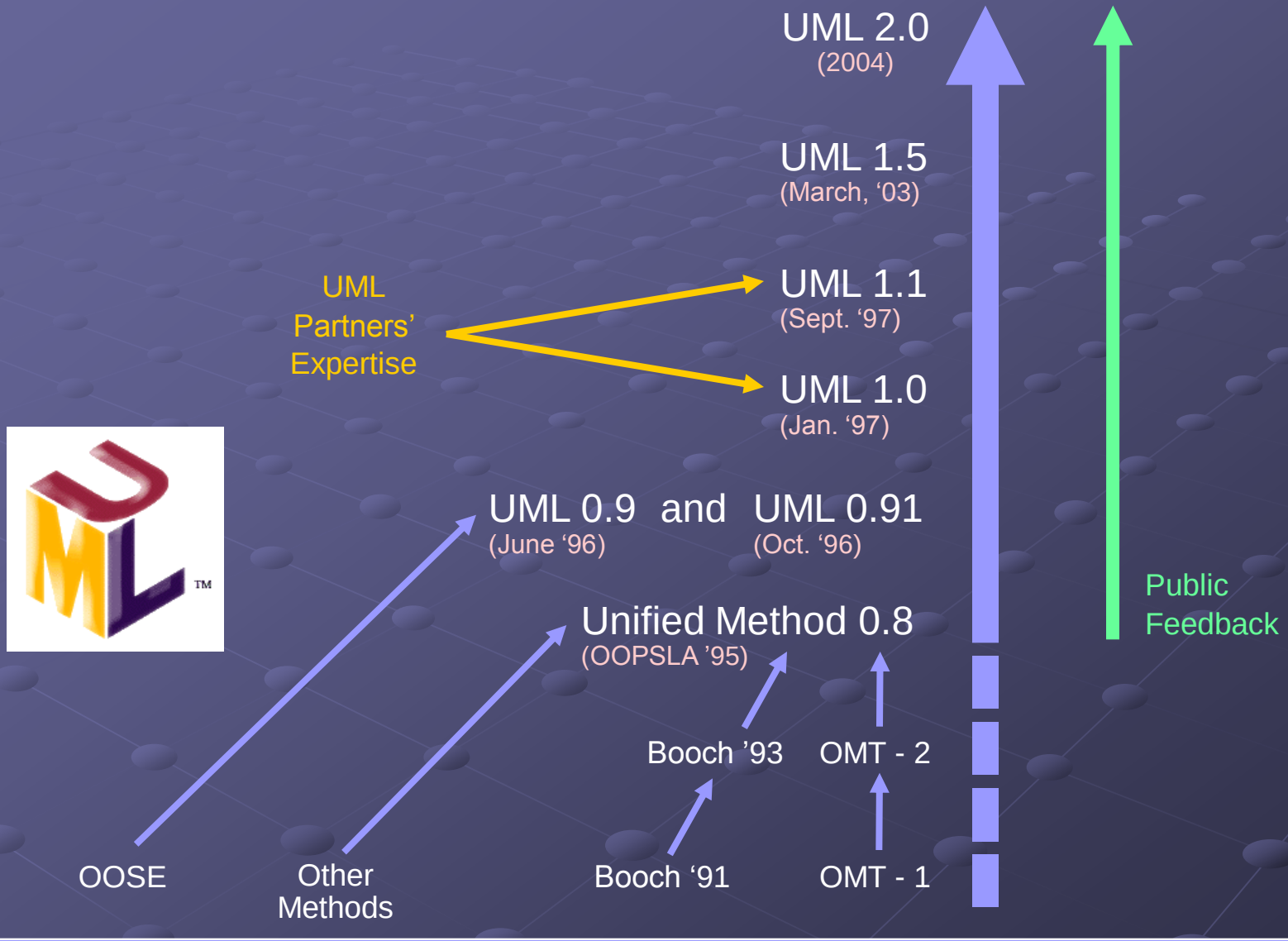
UML là một ngôn ngữ hợp nhất



UML là một ngôn ngữ thống nhất

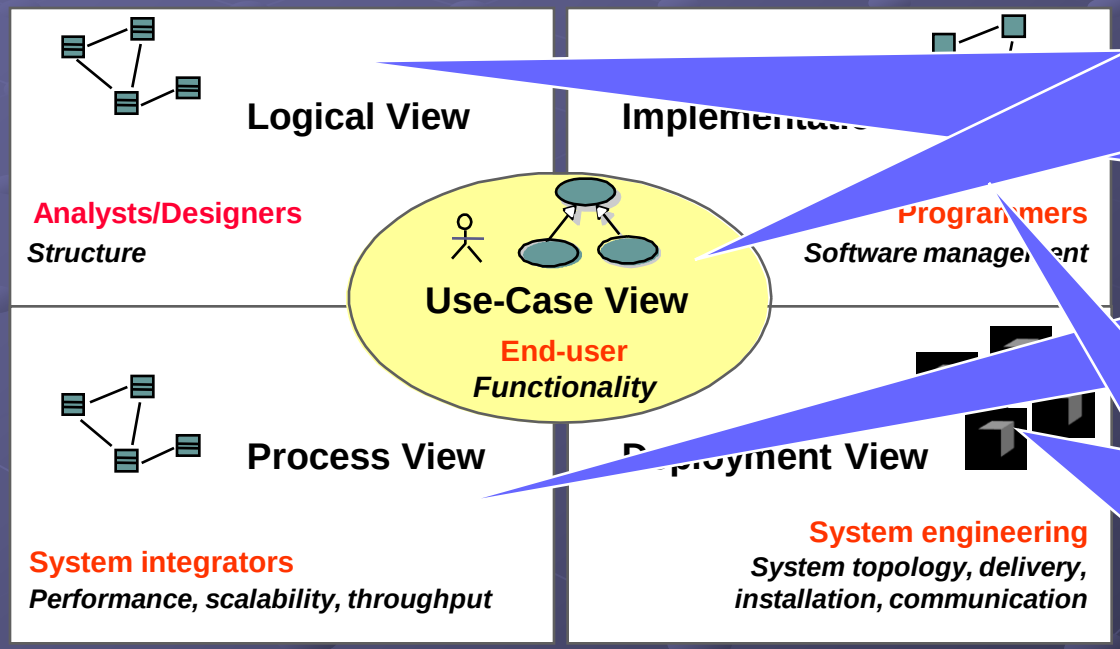


3.2. Lịch sử phát triển của UML (2)



3.3. Các khung nhìn của UML

- Khung nhìn của mô hình có ý nghĩa với những người tham gia nào đó
- 4 + 1 Architectural View



Biểu diễn các chức năng và môi trường dự kiến của hệ thống dưới góc nhìn của người dùng

Mô tả các nút vật

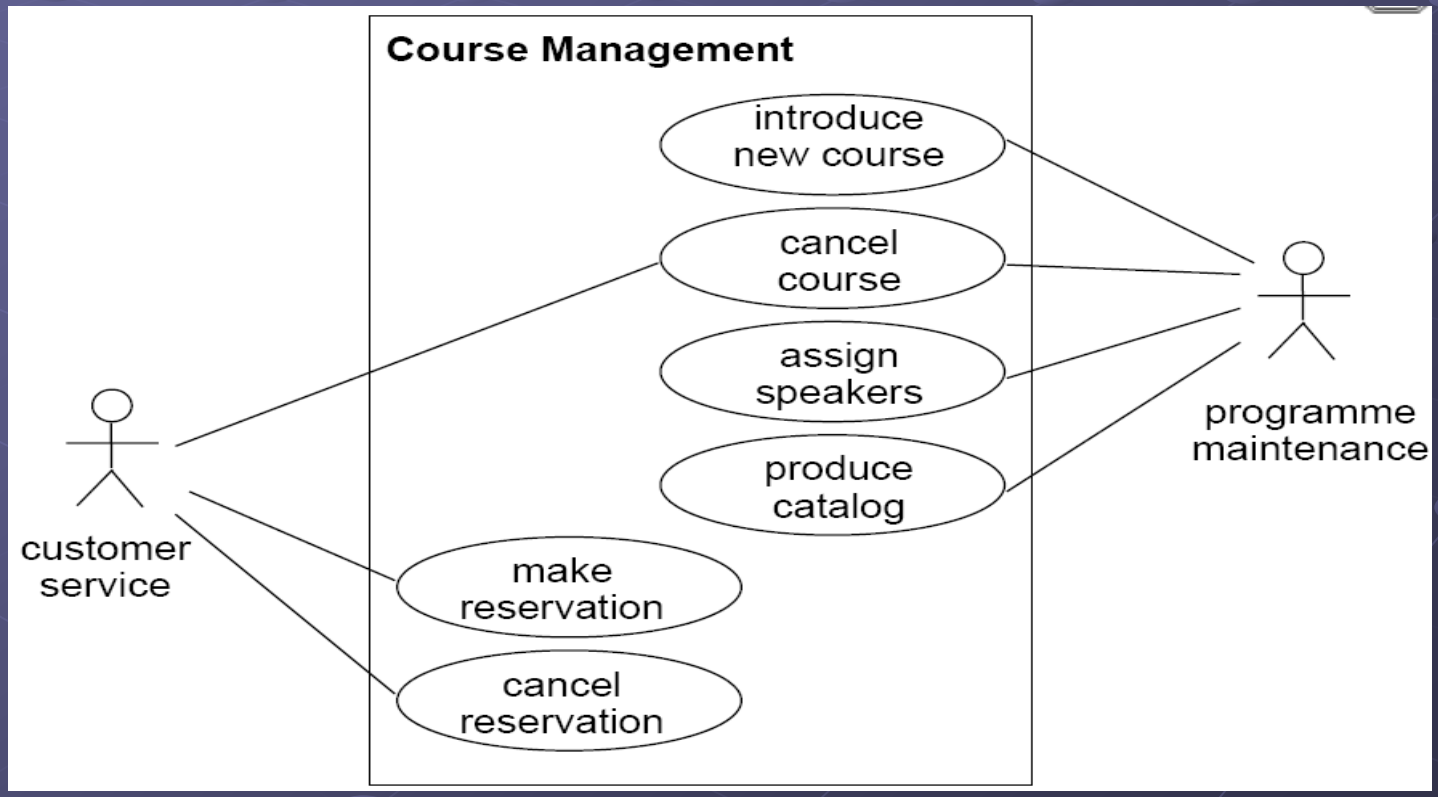
Mô tả việc tổ chức các mô-đun phần mềm tĩnh nhằm chia thành package, phân lớp và quản lý cấu hình

Các biểu đồ UML

- Biểu đồ ca sử dụng (Use Case Diagram)
- Biểu đồ tương tác (Interaction Diagrams)
  - Biểu đồ trình tự (Sequence Diagram)
  - Biểu đồ giao tiếp/cộng tác (Communication/Collaboration Diagram)
- Biểu đồ trạng thái (Statechart Diagram)
- Biểu đồ cấu trúc tĩnh (Static Structure Diagrams)
  - Class Diagram
  - Object Diagram
- Biểu đồ hoạt động (Activity Diagram)
- Biểu đồ thực thi (Implementation Diagrams)
  - Biểu đồ thành phần (Component Diagram)
  - Biểu đồ triển khai (Deployment Diagram)

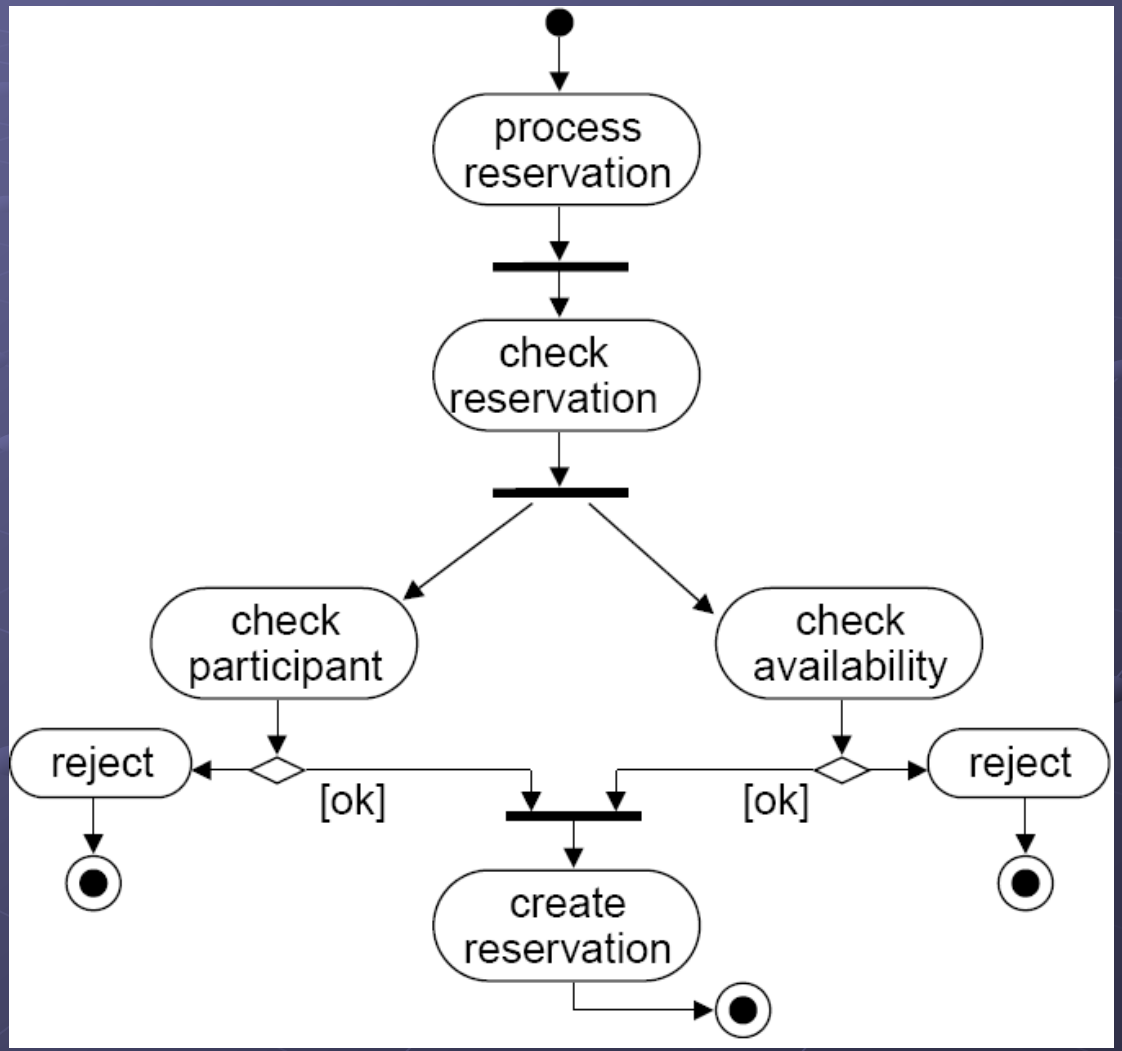
Biểu đồ use case

- Mô tả tương tác của hệ thống với thế giới bên ngoài.



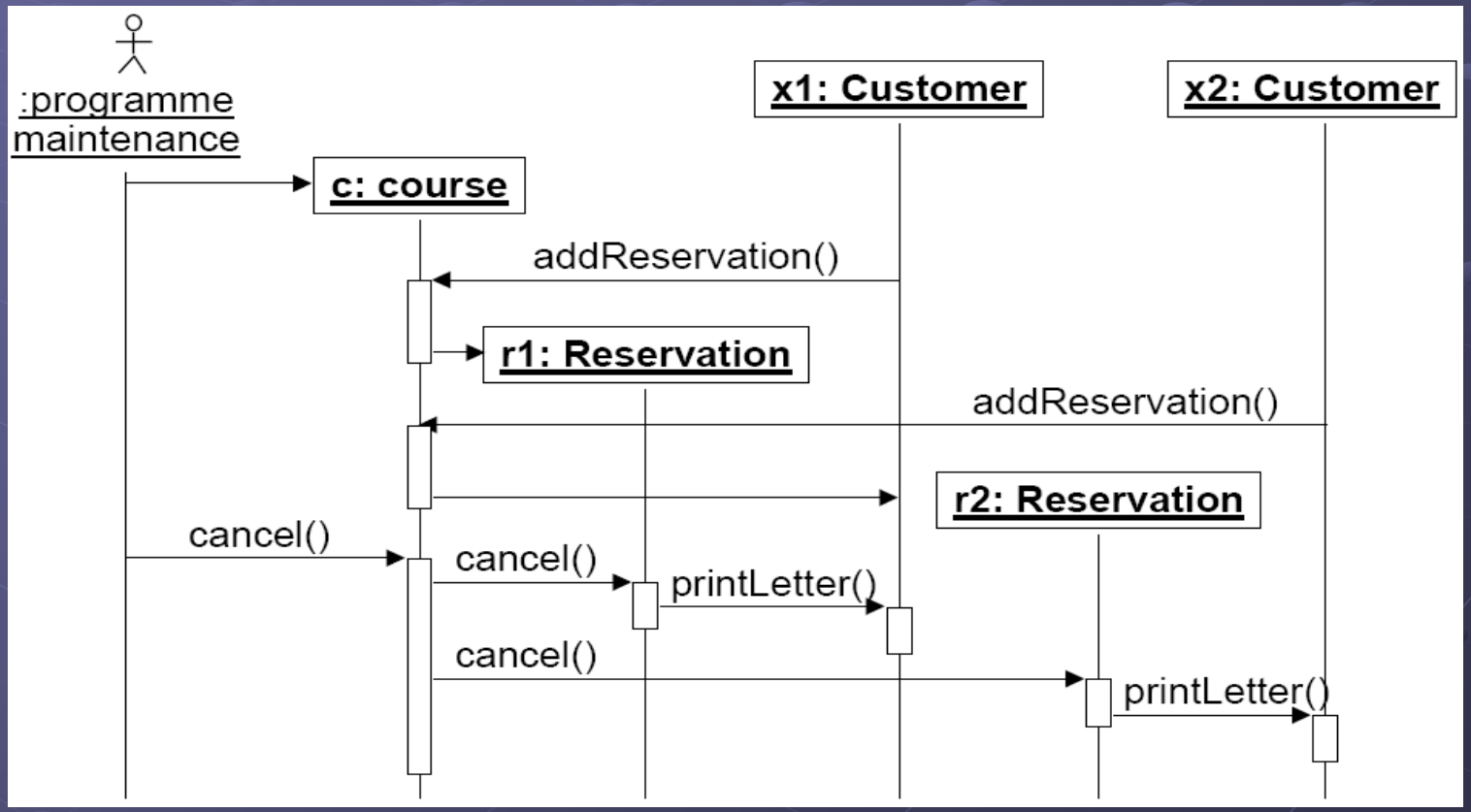
Biểu đồ hoạt động

- Chỉ ra luồng sự kiện bên trong hệ thống



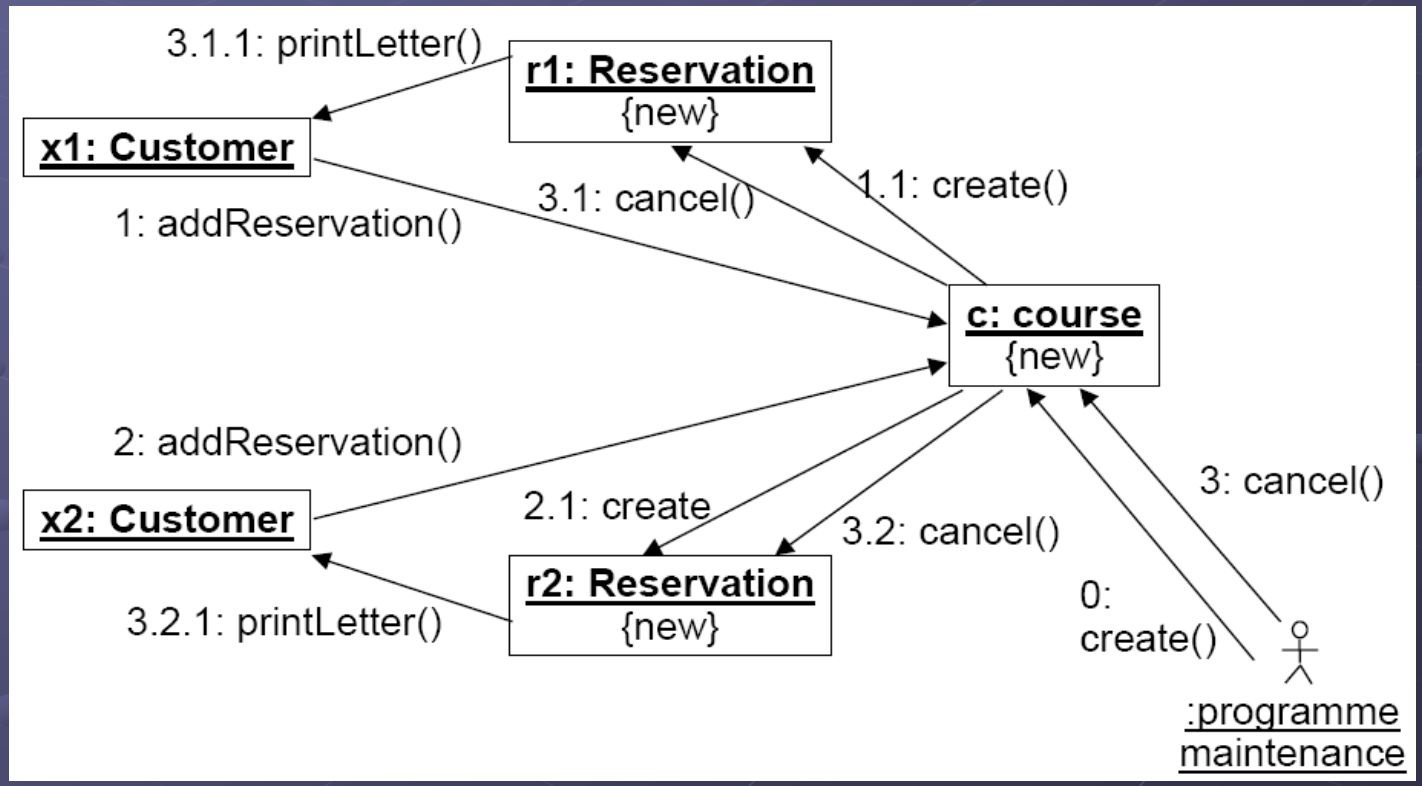
Biểu đồ trình tự

- Chỉ ra từng bước cần thực hiện để đạt được một chức năng nào đó của hệ thống



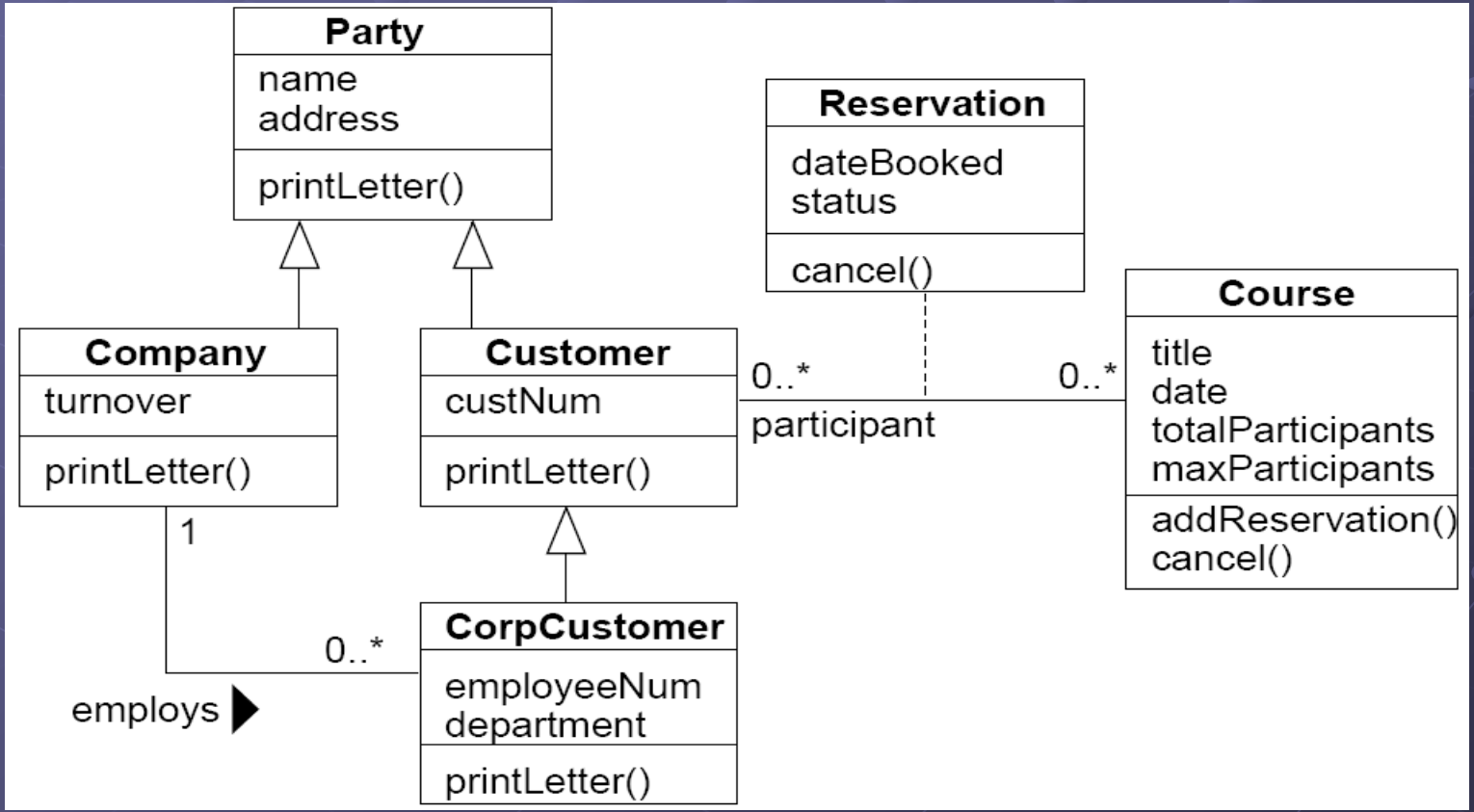
Biểu đồ giao tiếp/cộng tác

- Mô tả sự tương tác giữa các đối tượng được tổ chức xung quanh các đối tượng và liên kết giữa chúng

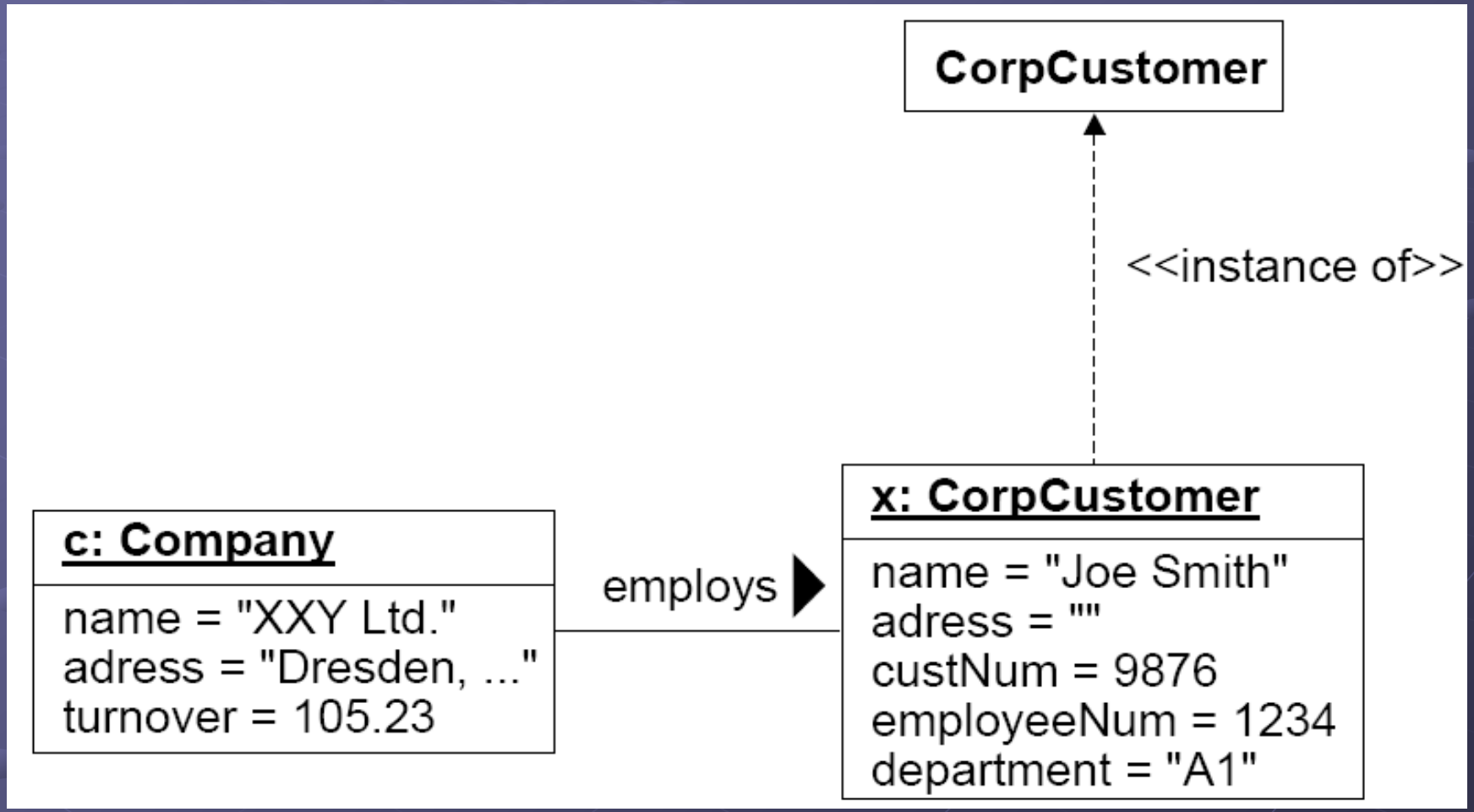


Biểu đồ lớp

- Mô tả cấu trúc của phần mềm

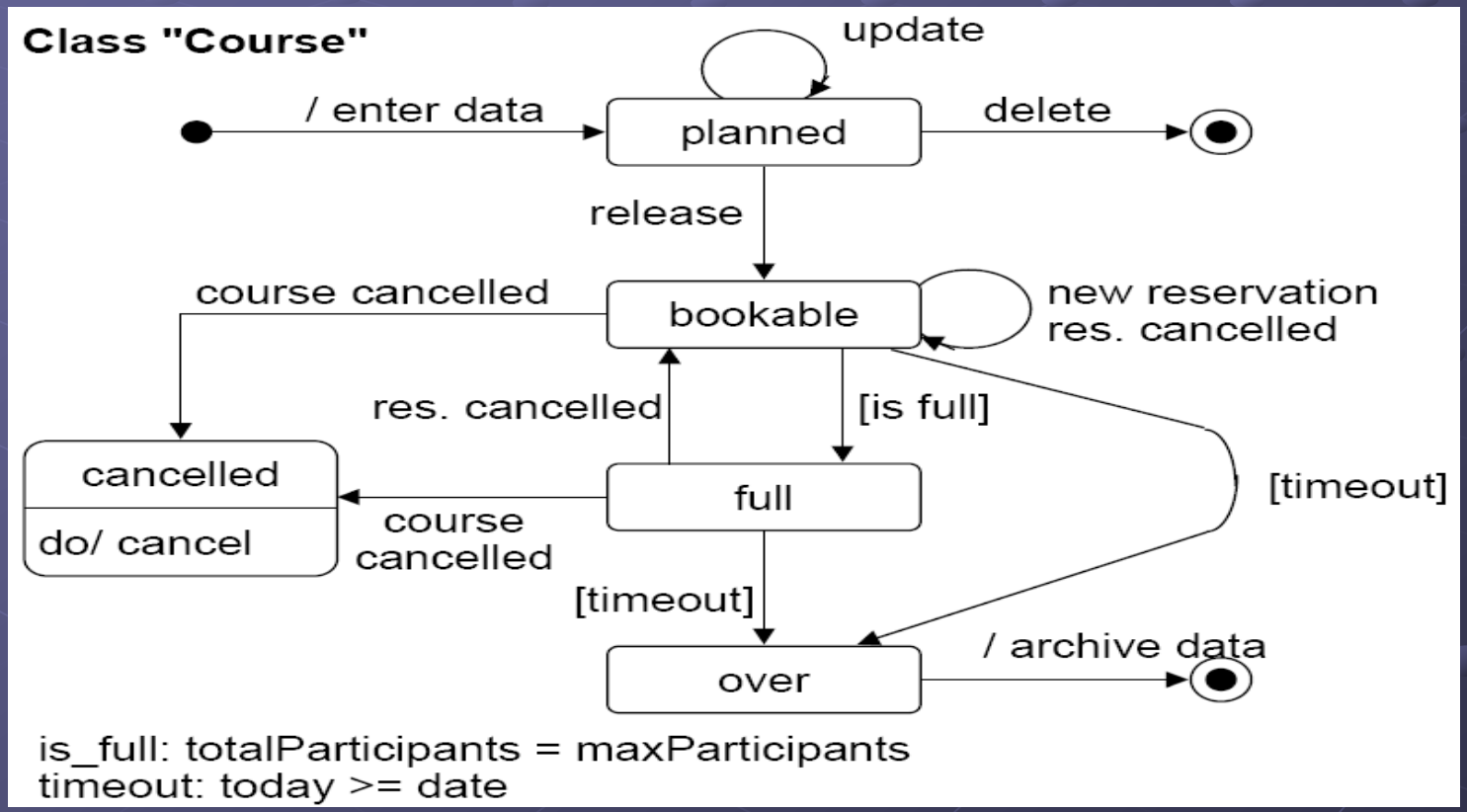


Biểu đồ đối tượng



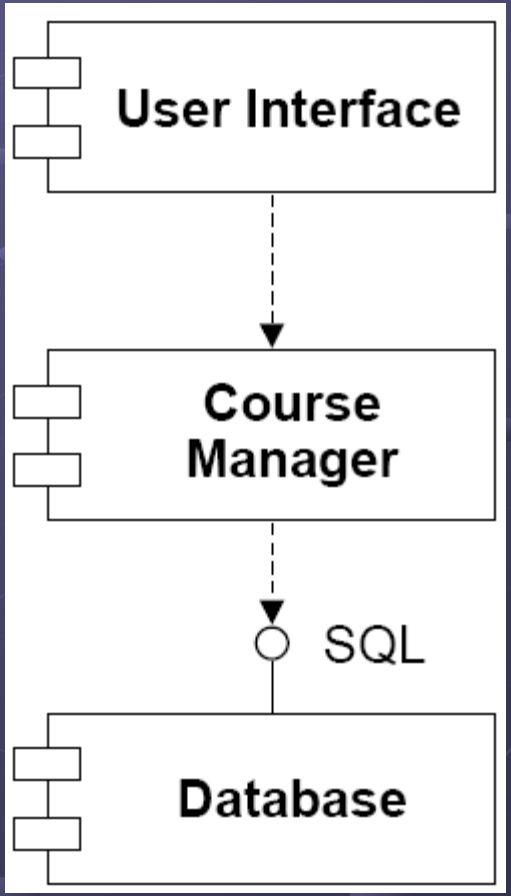
Biểu đồ trạng thái

- Mô tả vòng đời của một p nào đó



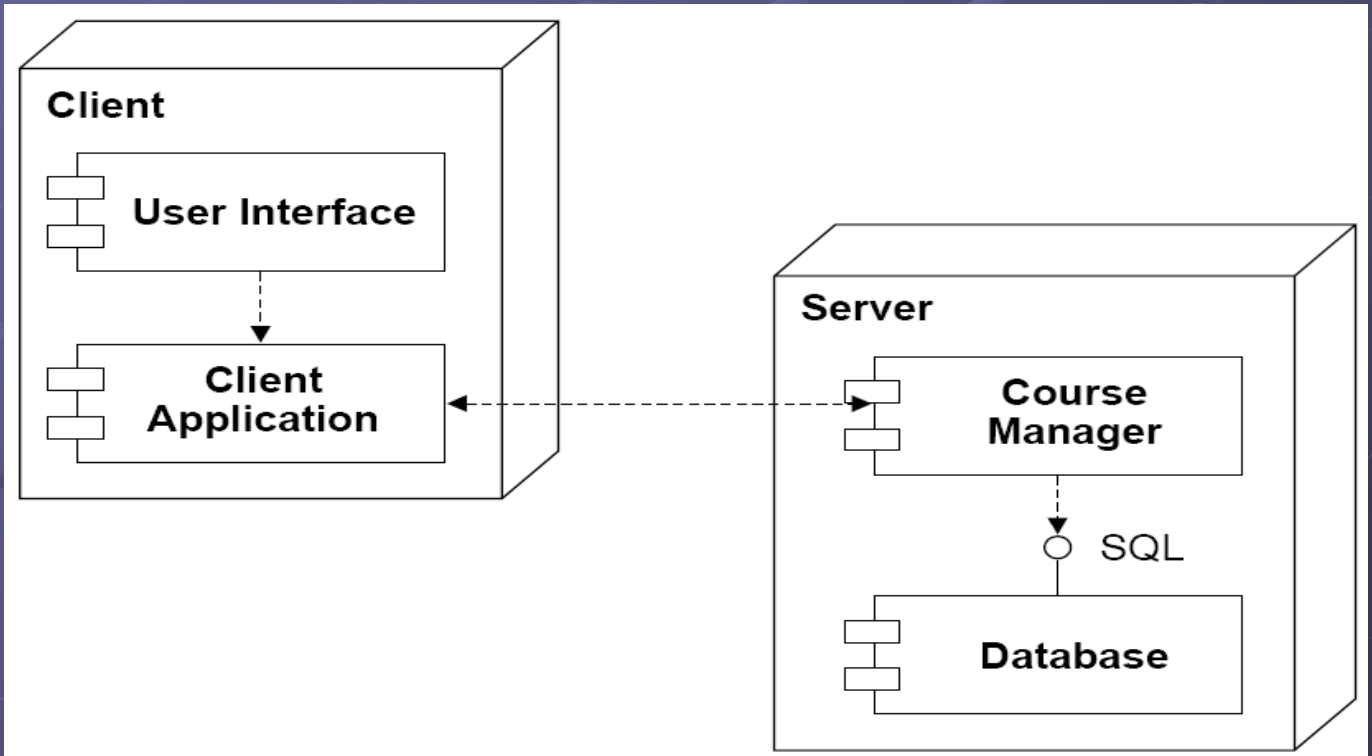
Biểu đồ thành phần

- Mô tả cách tổ chức và sự phụ thuộc giữa các thành phần phần mềm



Biểu đồ triển khai

- Mô tả sự phân phối các thành phần trong doanh nghiệp



Khung nhìn Use case

- Nắm bắt các chức năng của hệ thống
- Cần thiết cho các hoạt động phân tích, thiết kế và kiểm thử
- Hợp đồng giữa khách hàng và người phát triển
- Hành vi của hệ thống – các chức năng mà hệ thống cần cung cấp – được lưu trong một mô hình use case
  - Biểu đồ use case
  - Luồng sự kiện use case
  - Biểu đồ hoạt động
  - Các tài liệu phụ trợ

Khung nhìn Logic

- Cấu phần chính là mô hình thiết kế
  - Cung cấp mô tả cụ thể về hành vi chức năng của hệ thống.
  - Xuất phát từ mô hình phân tích
    - Mô tả vắn tắt về hành vi của hệ thống dựa trên mô hình use case
  - Tập hợp lớp, tổ chức vào các hệ thống con
  - Bao gồm:
    - Biểu đồ lớp, biểu đồ tương tác, biểu đồ trạng thái
    - Hệ thống con và giao diện của chúng

Khung nhìn thực thi (implementation)

- Tổ chức các mô-đun phần mềm tĩnh (mã nguồn, tệp dữ liệu, các thành phần thực thi, tài liệu…) trong môi trường dưới dạng:
  - Chia thành các package và phân lớp (layer)
  - Quản lý cấu hình (quyền sở hữu, kế hoạch bàn giao…)
- Được mô hình hóa trong các biểu đồ thành phần

Khung nhìn tiến trình (process)

- Bao gồm các thread và các process tạo nên các cơ chế đồng thời và đồng bộ của hệ thống
- Giải quyết về các vấn đề:
  - Đồng thời và song song (đồng bộ, deadlock…)
  - Dung thứ lỗi (cô lập chức năng và lỗi, độ tin cậy)
  - Khởi động và tắt hệ thống
  - Phân phối đối tượng và dữ liệu
  - Hiệu năng (thời gian đáp ứng, thông lượng) và tính co dãn
- Không cần thiết đối với môi trường xử lý đơn lẻ
- Mô hình hóa bằng biểu đồ lớp, biểu đồ tương tác và biểu đồ trạng thái

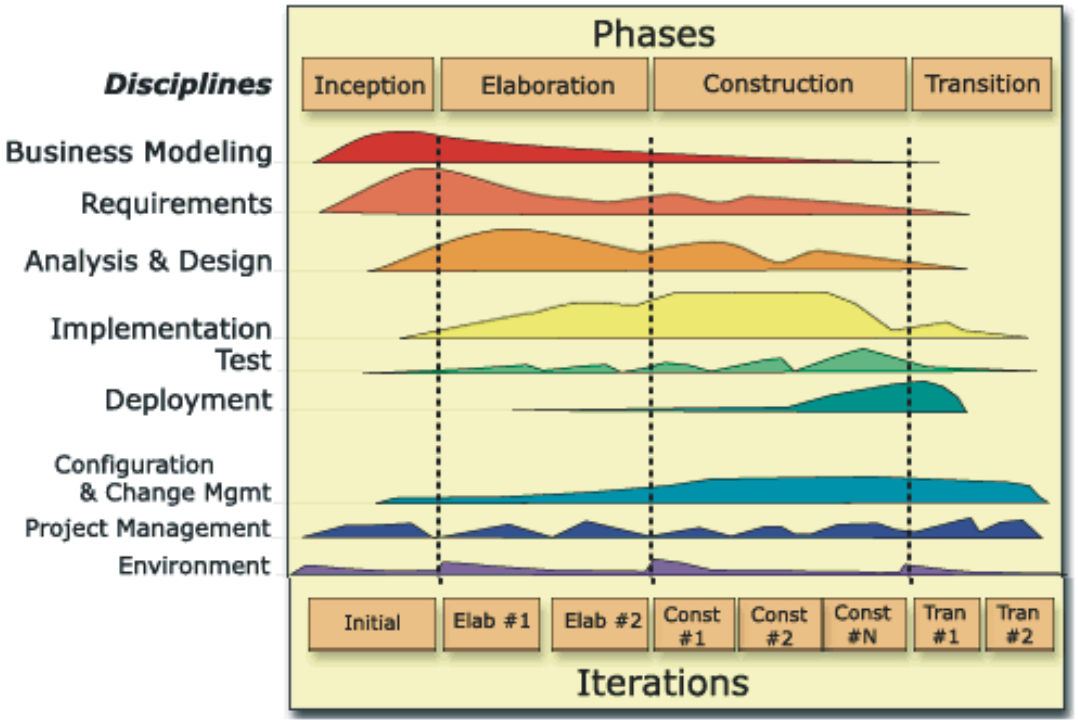
Khung nhìn triển khai (deployment)

- Mô tả các nút vật lý khác nhau và các kết nối lẫn nhau giữa chúng cho các cấu hình nền tảng điển hình nhất
- Giải quyết các vấn đề:
  - Triển khai
  - Cài đặt
  - Bảo trì
- Được mô hình hóa bằng biểu đồ triển khai

Quy trình và UML

- UML là ký pháp chứ không phải là phương pháp
  - UML có thể áp dụng cho tất cả các pha của quy trình phát triển phần mềm
  - "Rational Unified Process" - quy trình phát triển cho UML

RUP®
Rational Unified Process®



BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 03: Tổng quan về OOAD

Nội dung

1. Mục đích của OOAD
2. p OOAD
3. Case study
4. UML

1.1. Tầm quan trọng của OOAD

- Nhiều người phát triển dự án
  - Cho rằng phần mềm chủ yếu được xây dựng bằng cách gõ “code” từ bàn phím
  - Không dành đủ thời gian cho quá trình phân tích và thiết kế phần mềm

→ Họ phải “cày bừa” để hoàn thành chương trình vì

  - Không hiểu hoặc hiểu sai yêu cầu
  - Giao tiếp với các thành viên không tốt
  - Không tích hợp được với module của đồng nghiệp…

→ Họ nhận ra rằng “Phân tích” và “Thiết kế” cần được coi trọng hơn, nhưng đã quá muộn

1.1. Tầm quan trọng của OOAD (2)

- Cần thiết lập một cơ chế hiệu quả để nắm bắt yêu cầu, phân tích thiết kế
- Cơ chế này phải như là một "ngôn ngữ thống nhất" giúp cho quá trình hợp tác hiệu quả giữa các thành viên trong nhóm phát triển phần mềm.

→ OOAD

1.2. Mục đích của OOAD

- Chuyển các yêu cầu của bài toán thành một bản thiết kế của hệ thống sẽ được xây dựng
- Tập trung vào quá trình phân tích các YÊU CẦU của hệ thống và thiết kế các MÔ HÌNH cho hệ thống đó trước giai đoạn lập trình
- Được thực hiện nhằm đảm bảo mục đích và yêu cầu của hệ thống được ghi lại một cách hợp lý trước khi hệ thống được xây dựng

1.2. Mục đích của OOAD (2)

- Cung cấp cho người dùng, khách hàng, kỹ sư phân tích, thiết kế nhiều cái nhìn khác nhau về cùng một hệ thống
- Chọn lựa thiết kế để phù hợp với môi trường phát triển
  - Là thiết kế khả thi cho từng hệ thống con, thành phần của kiến trúc
  - Ở mức chi tiết, thiết kế sẽ phụ thuộc vào nền tảng, ngôn ngữ lập trình, hay cơ sở dữ liệu.

Nội dung

1. Mục đích của OOAD
2. Ph p OOAD
3. Case study
4. UML

2. p OOAD

- OOAD được chia thành 2 giai đoạn
  - Phân tích hướng đối tượng (OOA)
  - Thiết kế hướng đối tượng (OOD)
- OOA là giai đoạn nhằm tạo ra các mô hình cơ bản (mô hình khái niệm) của hệ thống dựa theo những gì khách hàng yêu cầu về hệ thống của họ
- OOD sẽ bổ sung thêm các thông tin thiết kế chi tiết cho các mô hình nói trên

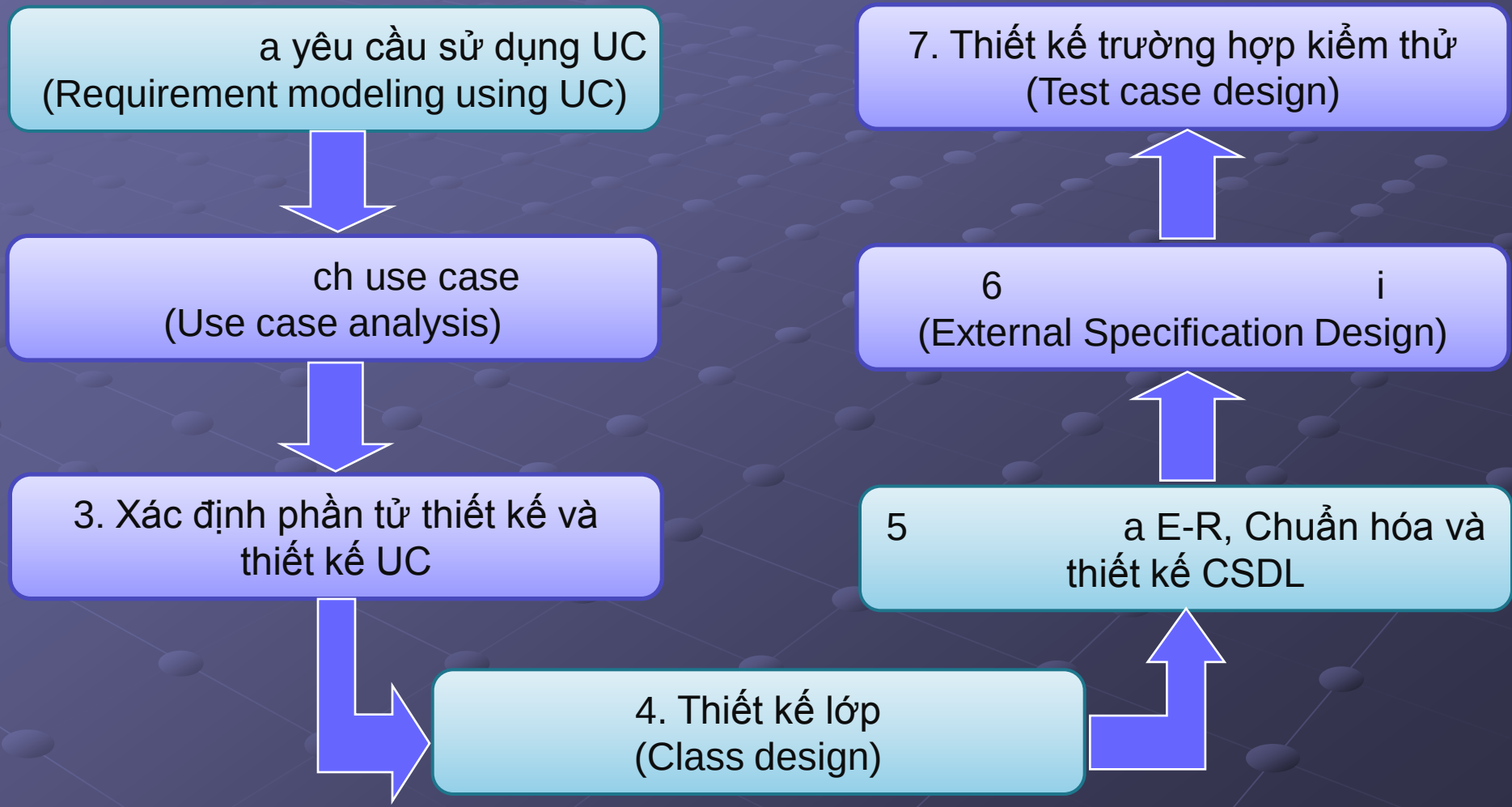
Mô hình Phân tích

- Phân tích tập trung vào việc trừu tượng hóa các vấn đề nghiệp vụ
- Xây dựng mô hình bằng cách tìm kiếm các lớp, các đối tượng chính có trong hệ thống
  - Các lớp, đối tượng này chỉ là những khái niệm nghiệp vụ cơ bản nhằm tìm hiểu hệ thống
  - Tránh cung cấp các khái ni m, thông tin cài đặt quá chi tiết (cho dù có thể đã tìm được tại thời điểm đó)

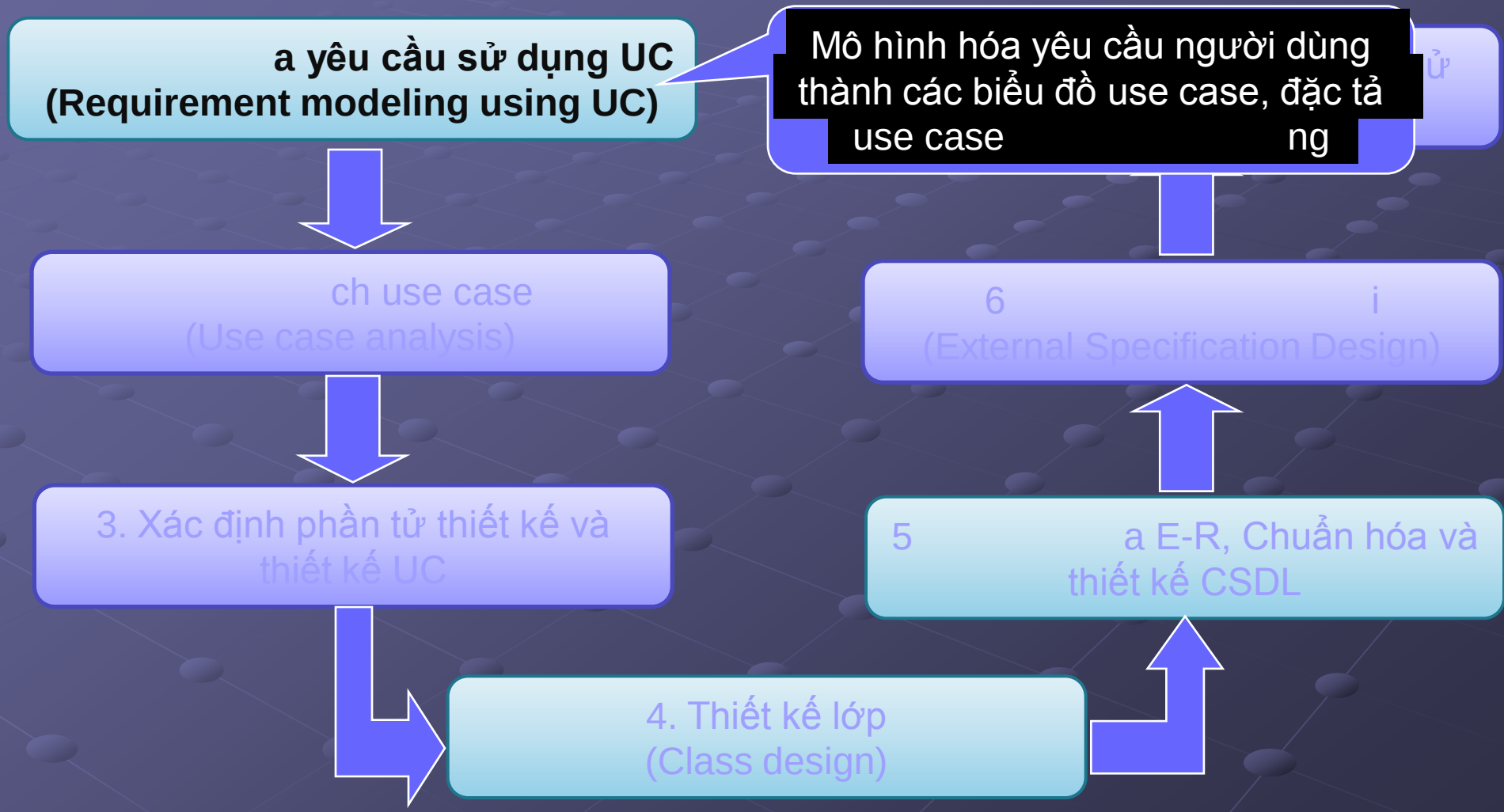
Mô hình Thiết kế

- Bao gồm các lớp, đối tượng ở mức cao
  - Nên đóng thành các gói
  - Chỉ rõ và mô hình hóa mối quan hệ tương tác giữa chúng
  - Chỉ rõ hành động và thuộc tính của các đối tượng
- Đơn giản hoá của việc phát triển mã nguồn
  - Ví dụ như thuật toán, sơ đồ khối…
- Thể hiện được bản thiết kế mã nguồn sẽ được cấu trúc và phát triển như thế nào

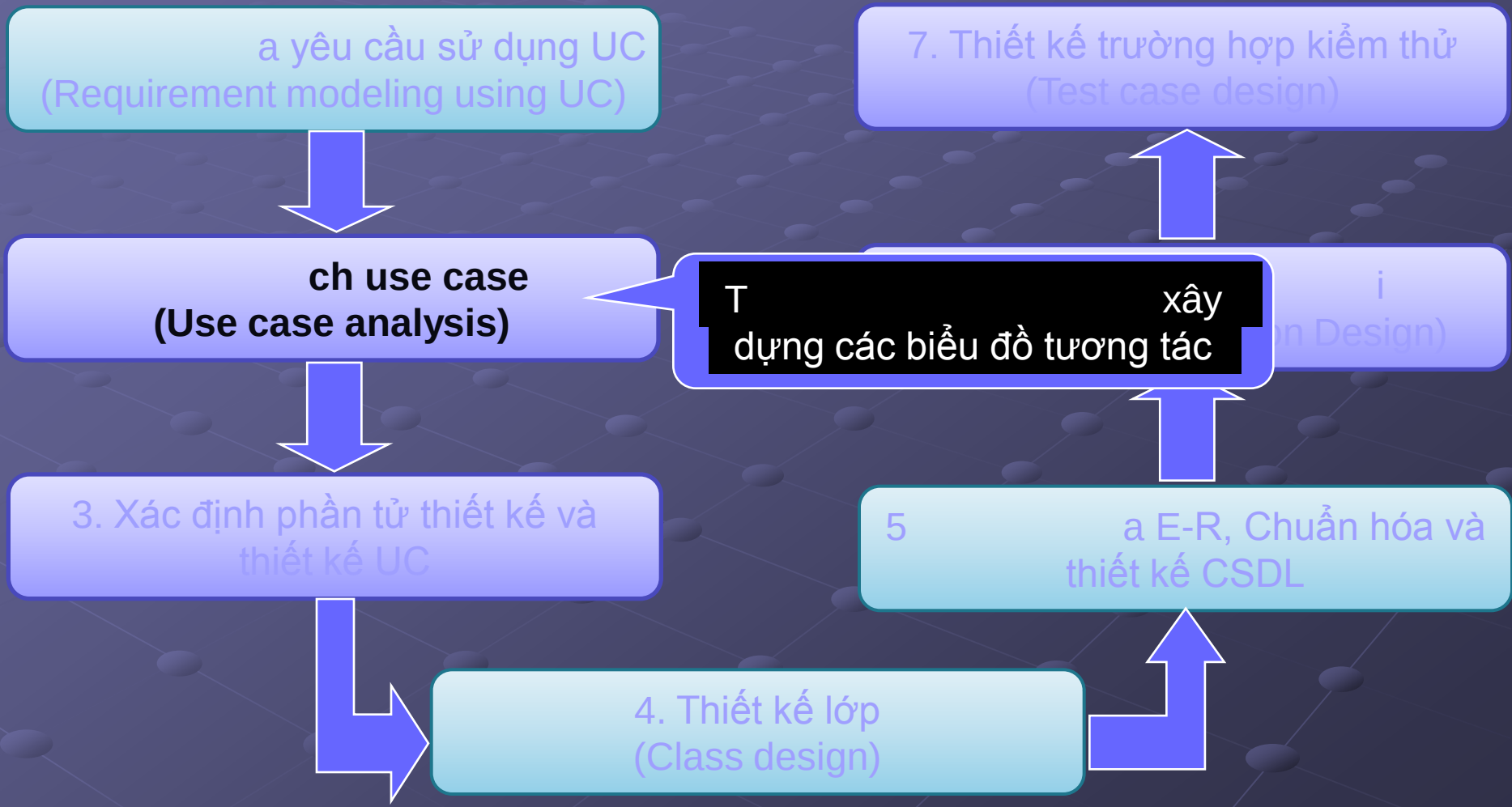
Các c trong OOAD



Các c trong OOAD



Các c trong OOAD



Các c trong OOAD

Các c trong OOAD

Các c trong OOAD

Các c trong OOAD

Các c trong OOAD

Chương 2. Tổng quan về phân tích thiết kế hướng đối tượng

1. Mục đích của OOAD
2. p OOAD
3. Case study
4. UML

3. Case study

- Case study mẫu: Course Registration
- Case study bài tập: Library System

Course Registration CS

- The system manages course registration in university. It manages course information for semesters of past one year and next semester.
- Course registration can be done for only next semester. Course information for past one year is only for viewing.
- Registrars, lecturers, and students have to login to use the system. After completing their work, they can log out the system because of security.
- Admistrators can manage users in the system (including search, add, delete and update).

Course Registration CS

- Registrars manage course information through registrar GUI. Registrars add, delete, and update basic information of courses for preparing course information of the next semester.
- Information about each course, such as lecturers, department and prerequisites will be included to help students make informed decisions. A course will have a maximum of 30 students and minimum 3 students

Course Registration CS

- After completion of preparation for all course information of the next semester, registrars open course information to students for registration.
- After that, students can register for courses.
- At the end of course registration period, registrars close course registration. After closing course registration, students cannot register nor cancel registration.
- A course with fewer than 3 students will be canceled. If a course is canceled, system must notifies students who registered the course about this cancellation.

Course Registration CS

- Lecturers make input and update detail of course information of corresponding course through lecturer GUI.
- Registrars and lecturers can view lists of students who registered for courses after registration period.
  - Registrars can view all lists.
  - Lecturers can view only lists for corresponding courses.

Course Registration CS

- Students make registration for courses through student GUI.
  - Students browse course list for the next semester and register for the course after choosing it.
  - If the course to be registered is full or its time is the same as the course he/she has registered or he/she does not satisty the prerequisites, the student must be notified.

Course Registration CS

- Students can view the list of own registration courses.
- Students can cancel course registration from registration list GUI.
- Registrars, lecturers, students can view course list for all semesters and search course information using keyword.

Course Registration CS

- Multiple users must be able to perform their work concurrently. The system shall support up to 500 simultaneous users at any one time.
- The system shall be available 24 hours a day, 7 days a week, with no more than 10% down time.

Nội dung

1. Mục đích của OOAD
2. p OOAD
3. Case study
4. UML

UML

- t trên:
  http://en.wikipedia.org/wiki/List_of_UML_tools
- :
  - EclipseUML
  - UmlDesigner
  - ArgoUML...
- i:
  - **Enterprise Architect**
  - IBM Rational Software Architect
  - Microsoft Visio
  - Visual Paradigm for UML
  - SmartDraw...

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 4. Mô hình hóa yêu cầu s ng use case

Nội dung

1. Mô hình hóa yêu cầu
2. Biểu đồ use case
3. Đặc tả use case
4.
5. trợ

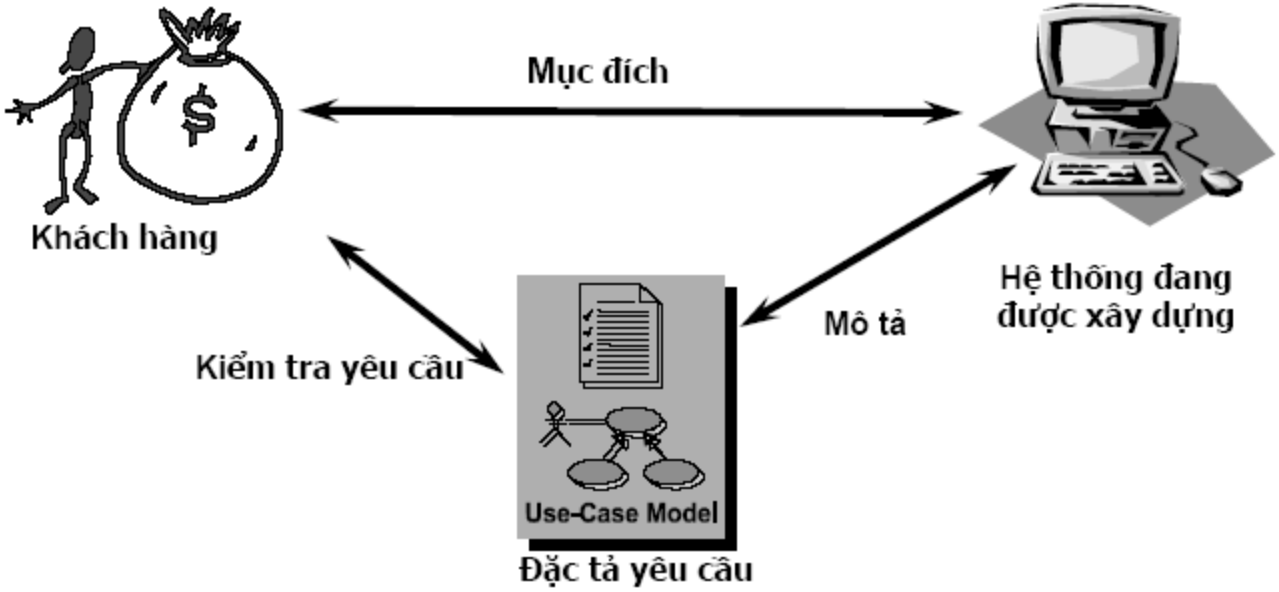
1.1. Mục đích

- Thiết lập và duy trì sự thoả thuận giữa khách hàng và người tham gia dự án về việc hệ thống sẽ làm được những gì
  - Không nói làm như thế nào để đạt được điều đó
- Giúp cho những người phát triển hệ thống một sự hiểu biết rõ hơn về những yêu cầu của hệ thống
  - Đưa ra những giới hạn mà hệ thống sẽ thực hiện và KHÔNG thực hiện
  - Cung cấp các thông tin cơ bản để lập kế hoạch phát triển dự án

1.1. Mục đích (2)

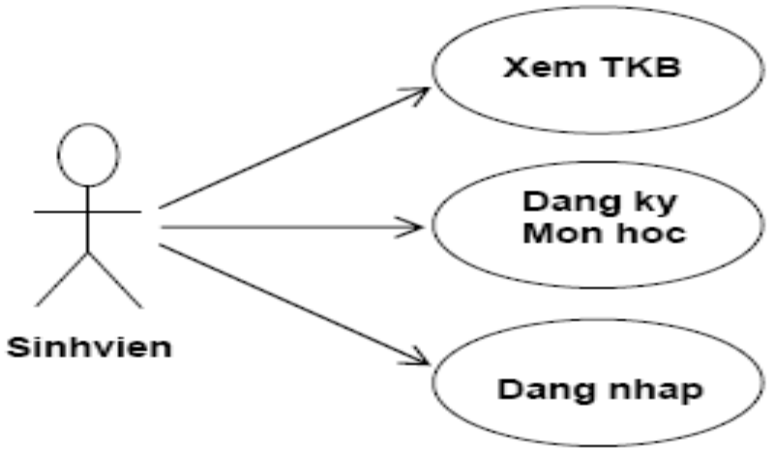
- Cung cấp những cơ sở để ước lượng giá thành và thời gian để phát triển hệ thống
- Nắm bắt được những yêu cầu và mục đích của người sử dụng

1.1. Mục đích (3)


Mục đích
Khách hàng
Hệ thống đang
được xây dựng
Kiểm tra yêu cầu
Mô tả
Use-Case Model
Đặc tả yêu cầu

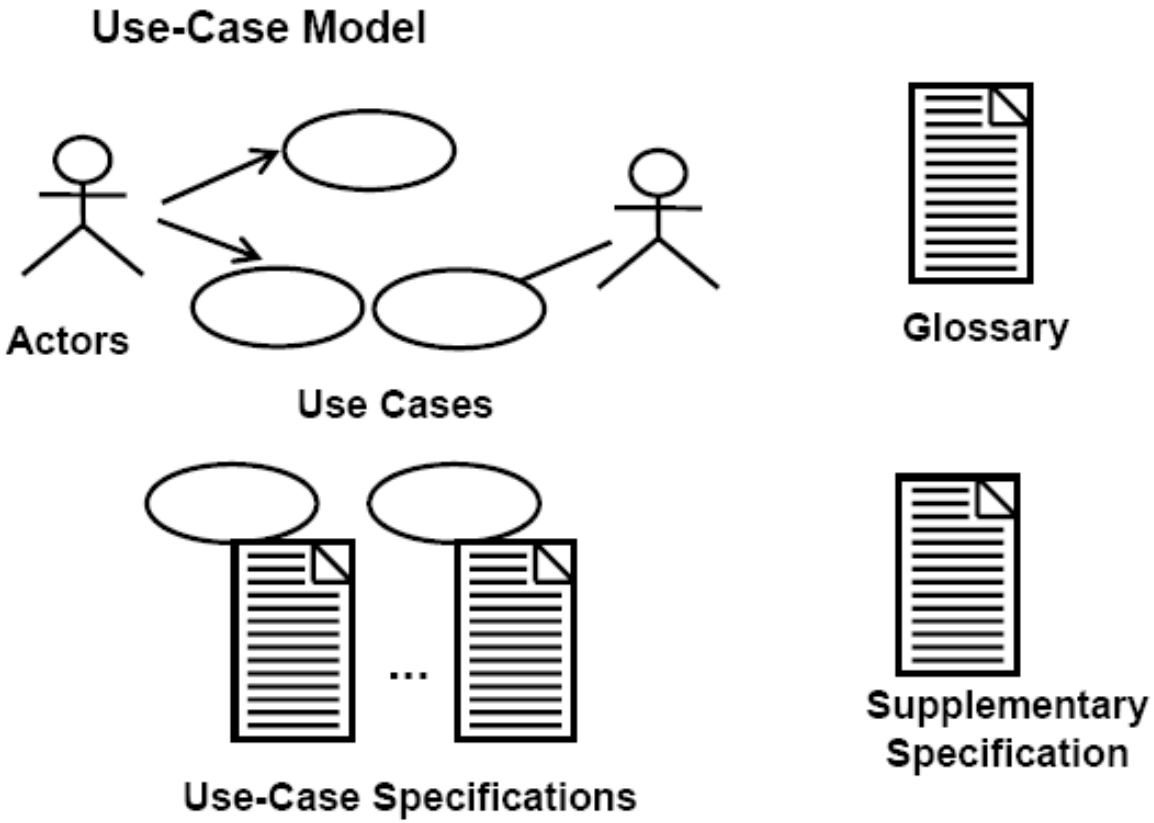
1.2. Mô hình hóa yêu cầu

- Mô hình hóa các chức năng mà hệ thống sẽ thực thi
  - Mô hình bao gồm các chức năng định trước của hệ thống
  - Sử dụng khái niệm Use Case



1.2. Mô hình hóa yêu cầu (3)

Các thành phần chính:

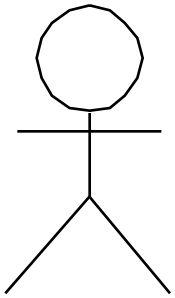


Nội dung

1. Mô hình hóa yêu cầu
2. Biểu đồ use case
3. Đặc tả use case
4.
5. trợ

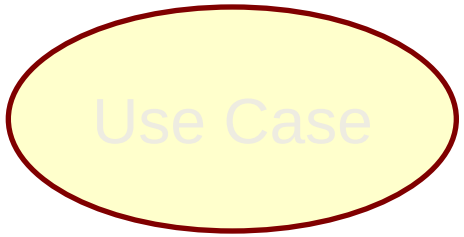
2.1. Actor và use case

- Tác nhân (actor) biểu diễn bất cứ thứ gì tương tác với hệ thống.



Actor

- Use case (Chức năng)
  - Mô tả chức năng mà hệ thống có
  - Mục đích là để PHÂN TÍCH yêu cầu nghiệp vụ của bài toán chứ không phải để THIẾT KẾ phần mềm

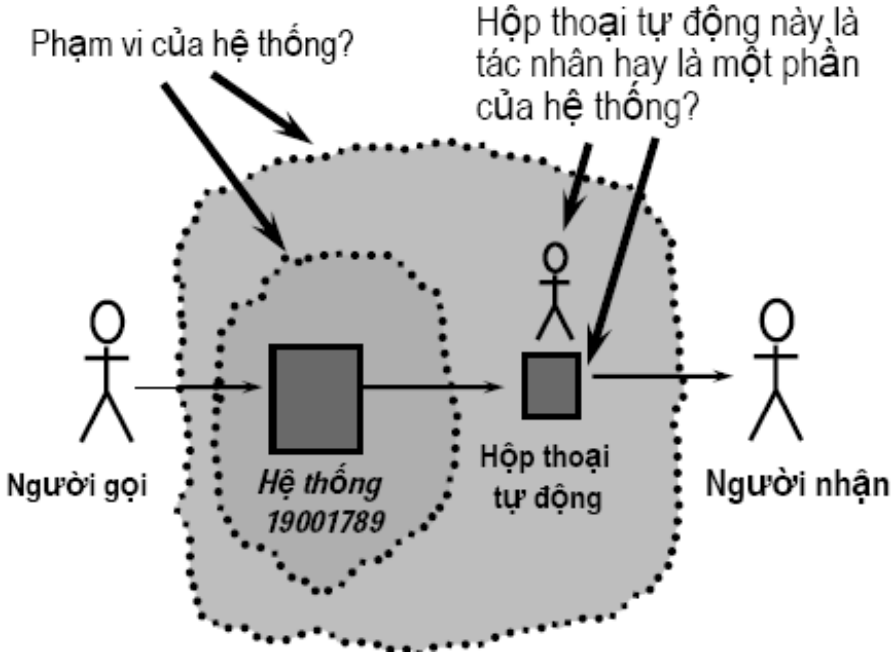


2.1.1. Tác nhân

Tìm kiếm tác nhân của hệ thống

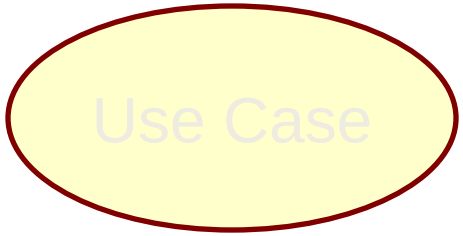
- Đặt các câu hỏi sau để tìm ra tác nhân:
  - Nhóm người nào yêu cầu hệ thống làm việc giúp họ?
  - Nhóm người nào kích hoạt chức năng của hệ thống?
  - Nhóm người nào sẽ duy trì và quản trị hệ thống hoạt động?
  - Hệ thống có tương tác với các thiết bị hay phần mềm ngoại vi nào khác hay không?
- Thông tin về tác nhân:
  - Tên tác nhân phải mô tả vai trò của tác nhân đó một cách rõ ràng
  - Tên nên là danh từ
  - Cần mô tả khái quát khả năng của tác nhân đó

Ví dụ về tác nhân



- Tác nhân trao đổi thông tin với hệ thống:
  - Gửi thông tin tới hệ thống
  - Nhận thông tin từ hệ thống

2.1.2. Use case



Tìm use case của hệ thống

- Xem các yêu cầu chức năng để tìm ra các UC
- Đối với mỗi tác nhân tìm được, đặt các câu hỏi:
  - Các tác nhân yêu cầu những gì từ hệ thống
  - Các công việc chính mà tác nhân đó muốn HT thực thi?
  - Tác nhân đó có tạo ra hay thay đổi dữ liệu gì của HT?
  - Tác nhân đó có phải thông báo gì cho HT?
  - Tác nhân đó có cần thông tin thông báo gì từ HT?
- Thông tin về use case:
  - Tên của UC nên chỉ rõ kết quả của quá trình tương tác với tác nhân
  - Tên nên là động từ
  - Mô tả ngắn gọn về mục đích của UC

Những điều nên tránh khi tạo UC

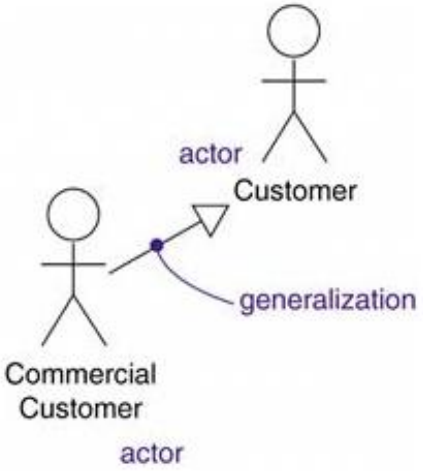
- Tạo ra các UC quá nhỏ
  - Hành động quá đơn giản mà chỉ cần mô tả bởi vài dòng
- Tạo ra quá nhiều Use case (hàng chục)
  - Nhóm các Use case liên quan thành một Use case tổng quát (mức 1)
  - Mô tả các Use Case tổng quát ở một sơ đồ khác (mức 2)
    - Ví dụ: “Quản lý sách” bao gồm “Nhập sách”, “Xuất sách”, “…”
- Sử dụng các Use-case quá cụ thể, hoặc làm việc với dữ liệu quá cụ thể. Ví dụ:
  - “Tìm sách theo tên” (nên là “Tìm sách”)
  - “Nhập Pin vào máy ATM” (nên là “Nhập PIN”)
  - “Thêm sách” (nên là “Quản lý sách” bao gồm “Thêm sách”)

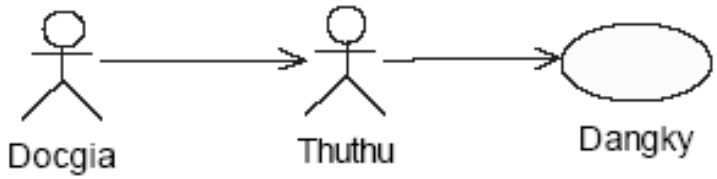
2.2. Mối liên hệ (relationship)

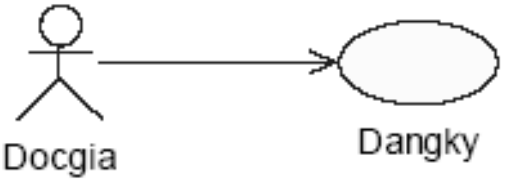
- Mối liên hệ giữa các actor với nhau
  - Khái quát hóa (Generalization)
  - Giao tiếp
- Mối liên hệ giữa actor và use case
  - Giao tiếp
- Mối liên hệ giữa các use case với nhau
  - Generalization: Khái quát hóa
  - Include: Bao hàm
  - Extend: Mở rộng

2.2.1. Mối liên hệ giữa các actor với nhau

- Khái quát hóa (Generalization)
  - Tác nhân con kế thừa tính chất và hành vi của tác nhân cha
- Giao tiếp
  - Xét sự khác nhau giữa hai biểu đồ sau

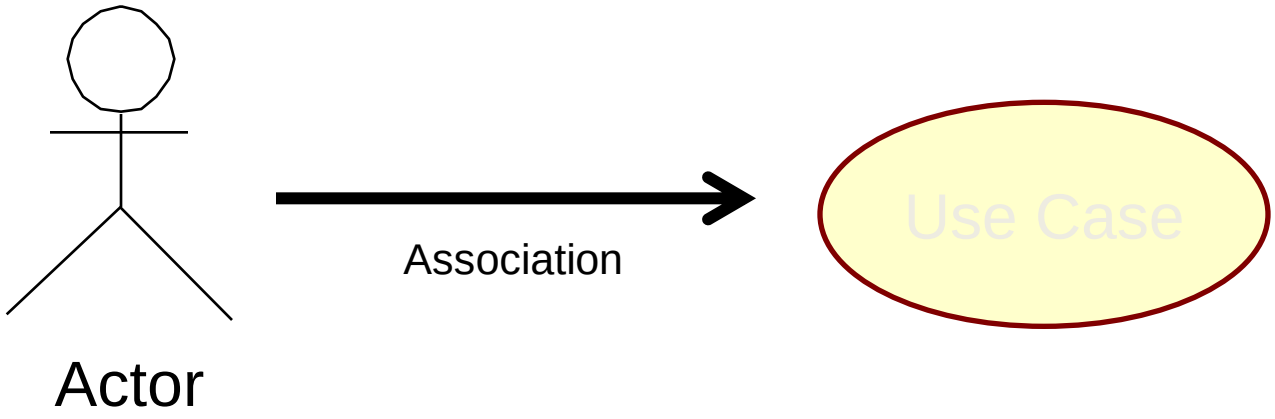






2.2.2. Mối liên hệ giữa actor với use case

- Thiết lập quan hệ giữa Tác nhân và Use Case
  - Chúng tương tác bằng cách gửi các tín hiệu cho nhau
- Một use case mô hình hóa một hội thoại giữa các tác nhân và hệ thống
- Một use case được bắt đầu bởi một tác nhân để gọi một chức năng nào đó trong hệ thống.



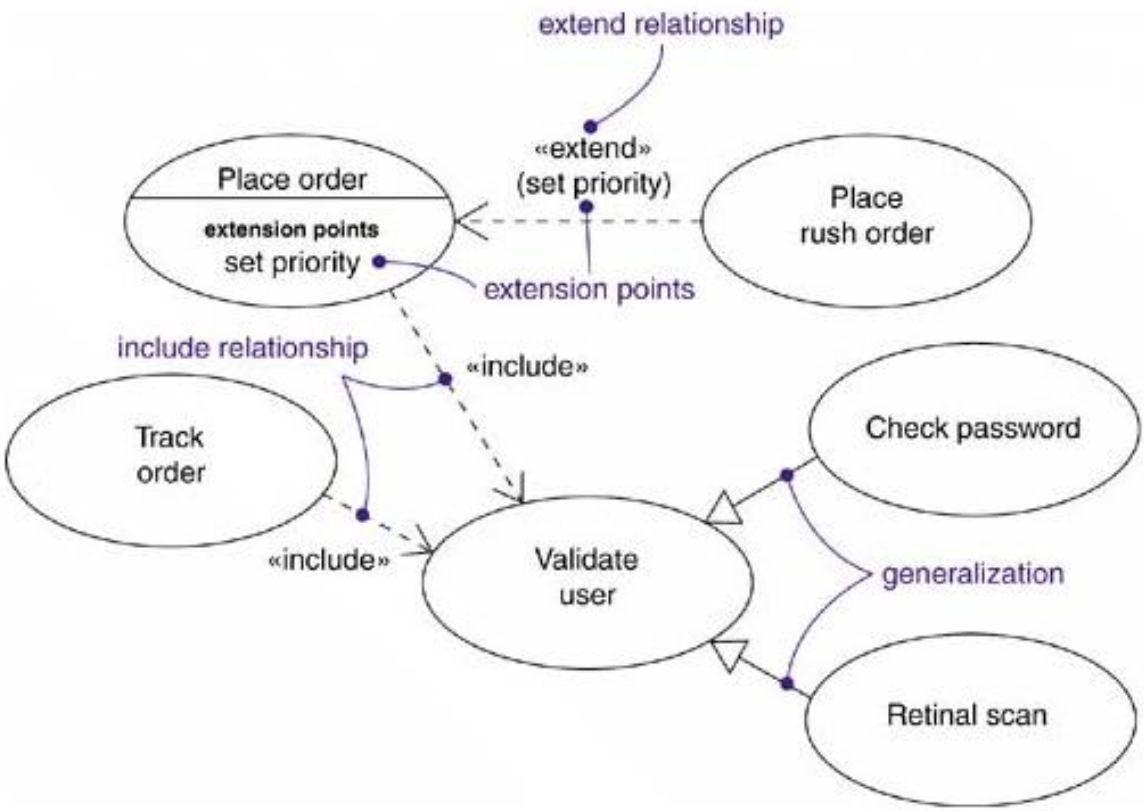
2.2.2. Mối liên hệ giữa actor với use case (2)

Chiều của quan hệ chính là chiều của tín hiệu gửi đi

- Từ tác nhân tới Use Case
  - Kích hoạt Use case
  - Hỏi thông tin nào đó trong hệ thống
  - Thay đổi thông tin nào đó trong hệ thống
  - Thông báo cho UC về một sự kiện đặt biệt nào đó xảy ra với hệ thống
- Từ Use Case tới tác nhân:
  - Nếu như có một điều gì đó xảy ra với HT và tác nhân đó cần được biết sự kiện đó
  - UC đôi khi cần hỏi thông tin nào đó từ một tác nhân trước khi UC đó đưa ra một quyết định

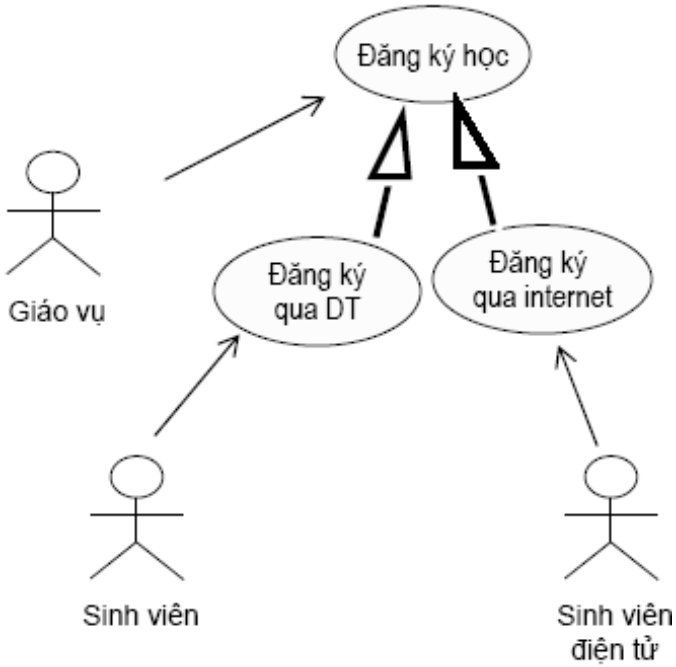
2.2.3. Mối liên hệ giữa các use case với nhau

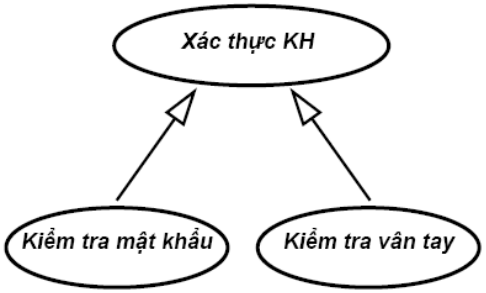
- Generalization
- <<include>>
  - always use
- <<extend>>
  - sometime use



Quan hệ generalization

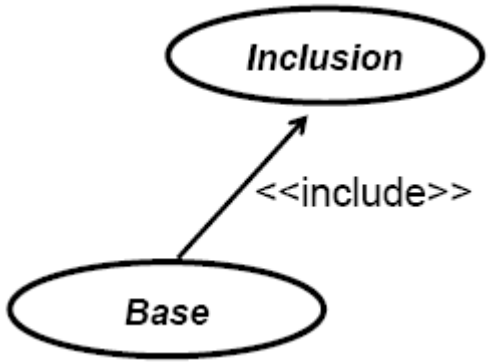
- Được sử dụng để chỉ ra một vài tính chất chung của một nhóm tác nhân hoặc UC
- Sử dụng khái niệm kế thừa
  - Mô tả hành vi chung (chia sẻ) trong UC cha
  - Mô tả hành vi riêng trong (các) UC con

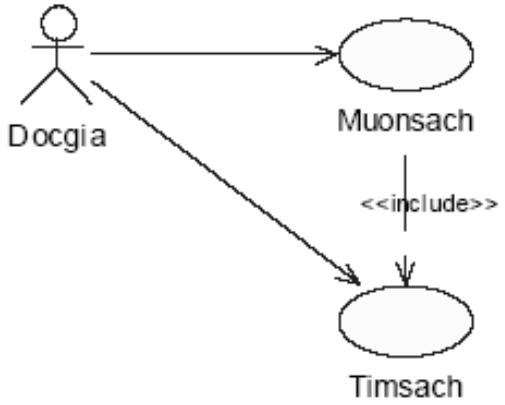




<<include>>

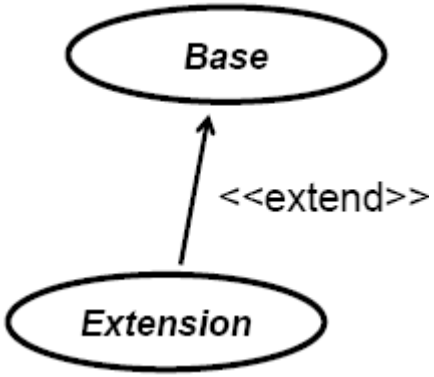
- Cho phép một UC sử dụng chức năng của UC khác
- Chức năng của UC Inclusion sẽ được gọi trong UC Base
- Sử dụng stereotype là <<include>>

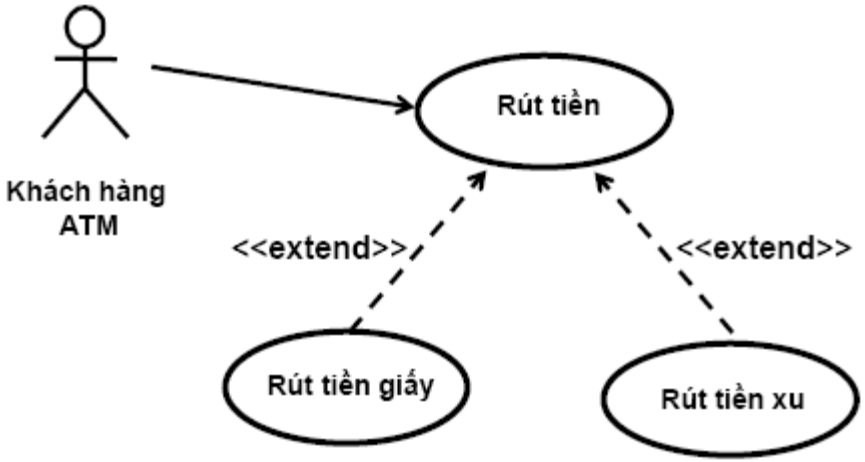




<<extend>>

- Cho phép mở rộng chức năng của một UC
- Chèn hành vi của UC Extension vào UC Base
- Chỉ chèn khi điều kiện extend đúng (mở rộng, phát sinh)
- Chèn vào lớp cơ sở tại điểm phát sinh (extension point)
- Sử dụng stereotype là <<extend>>





2.3. Các thành phần khác

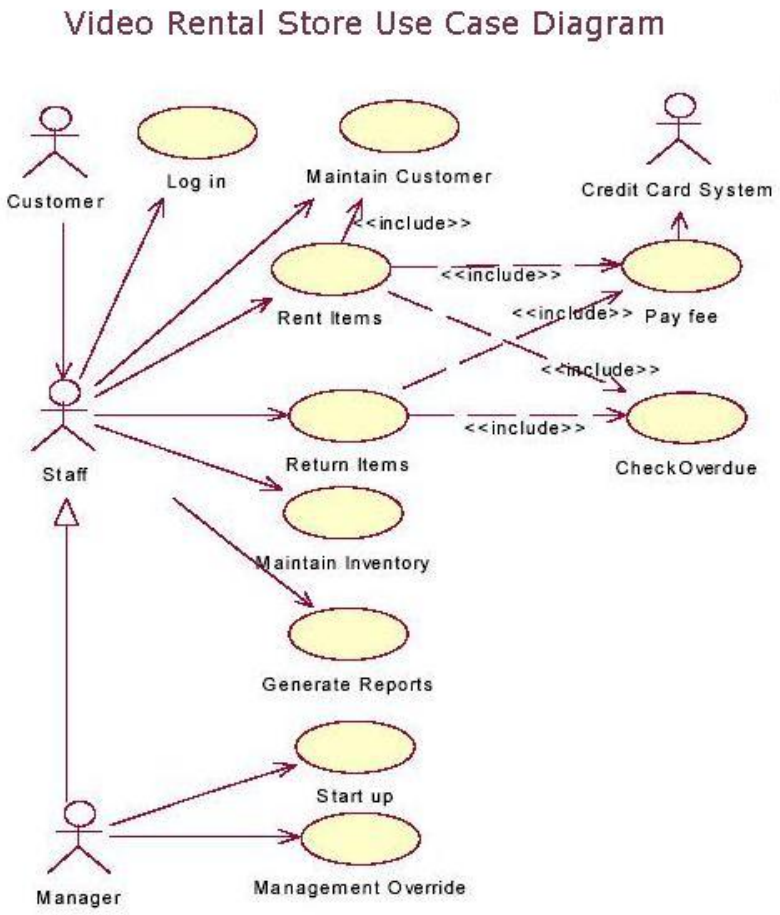
- Ghi chú (Notes)
  - Thêm các ghi chú để sơ đồ rõ ràng hơn

2.3. Các thành phần khác (2)

- Có thể nhóm các thành phần (Use-Case, Actor, quan hệ hoặc các sơ đồ khác) thành một nhóm chung (package)
- Nếu số lượng Use Case quá lớn, nên chia chúng vào các nhóm
  - Dễ hiểu mô hình tổng thể hơn
  - Dễ bảo trì mô hình Use Case
  - Dễ giao việc cho các thành viên
- Xem xét khả năng gộp nhóm
  - Tương tác với cùng một tác nhân
  - Nhóm Use Case hợp thành một quy trình (module) tương đối hoàn thiện

2.4. Biểu đồ use case

- Biểu đồ use case (use case diagram)
  - Là tập hợp các actor và các use case lại; bổ sung các mối liên quan (association) giữa chúng và lập thành biểu đồ use case



Nội dung

1. Mô hình hóa yêu cầu
2. Biểu đồ use case
3. Đặc tả use case
4.
5. trợ

Đặc tả Use Case

- Tên:
  - Tên của Use case
- Mô tả ngắn gọn:
  - Mô tả về vai trò và mục đích của use case, tránh kiểu diễn xuôi tên Use Case
- Điều kiện
  - Tiền điều kiện
  - Hậu điều kiện
- Luồng sự kiện (kịch bản):
  - Mô tả bằng lời những gì mà hệ thống sẽ làm thể hiện trên use-case
- Biểu đồ hoạt động
  - Minh họa luồng sự kiện bằng mô hình
- Các yêu cầu đặc biệt…

Luồng sự kiện của use-case

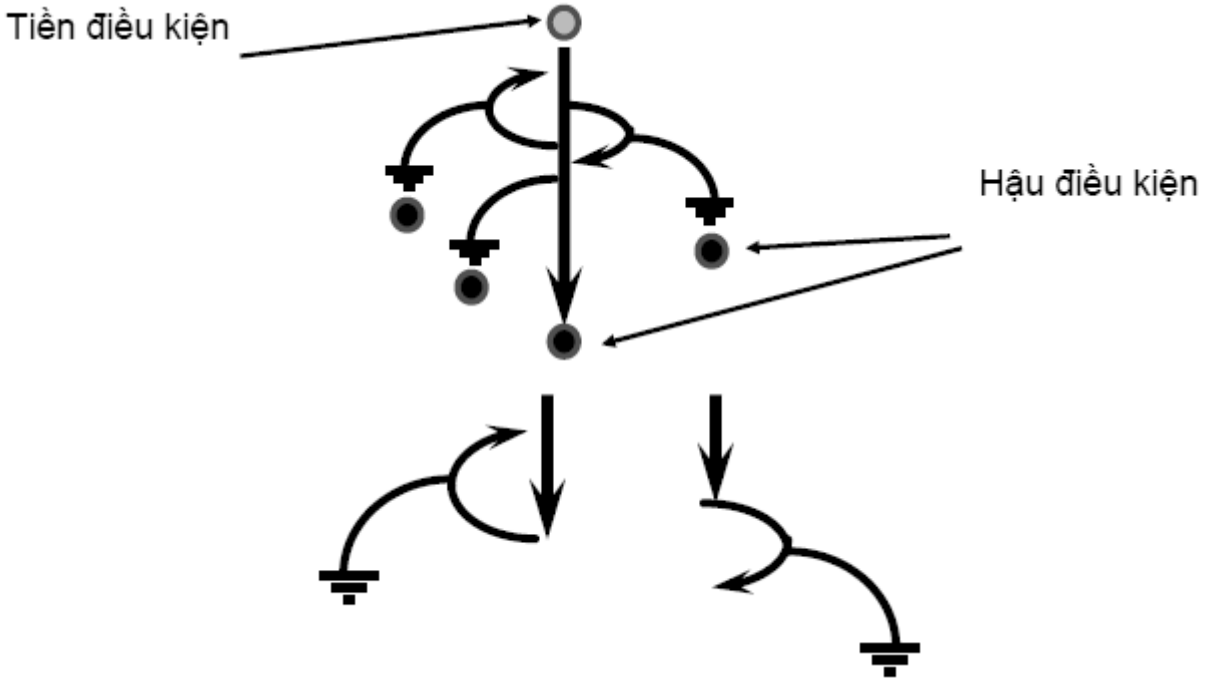
- Trả lời được quá trình từ khi bắt đầu đến khi kết thúc của một use-case
  - Chỉ mô tả chi tiết các sự kiện thuộc use-case đó
  - Nếu có sự liên hệ với Use Case khác, nên có sự phân tích và tham khảo ngắn gọn
- Mô tả dữ liệu được trao đổi giữa tác nhân và use-case đó
  - Cấu trúc: Ai làm gì, khi nào, với dữ liệu gì, [vì mục đích gì]
  - Cần phân tích rõ hệ thống cần phải làm gì để đáp ứng được yêu cầu của tác nhân đó. Không được mặc định cho rằng hệ thống tự biết làm điều đó
  - Tránh mô tả chức năng hoặc GUI (Graphic User Interface)
  - Tránh mô tả chung chung, hoặc lúc nào cũng đúng

Luồng sự kiện của use-case (2)

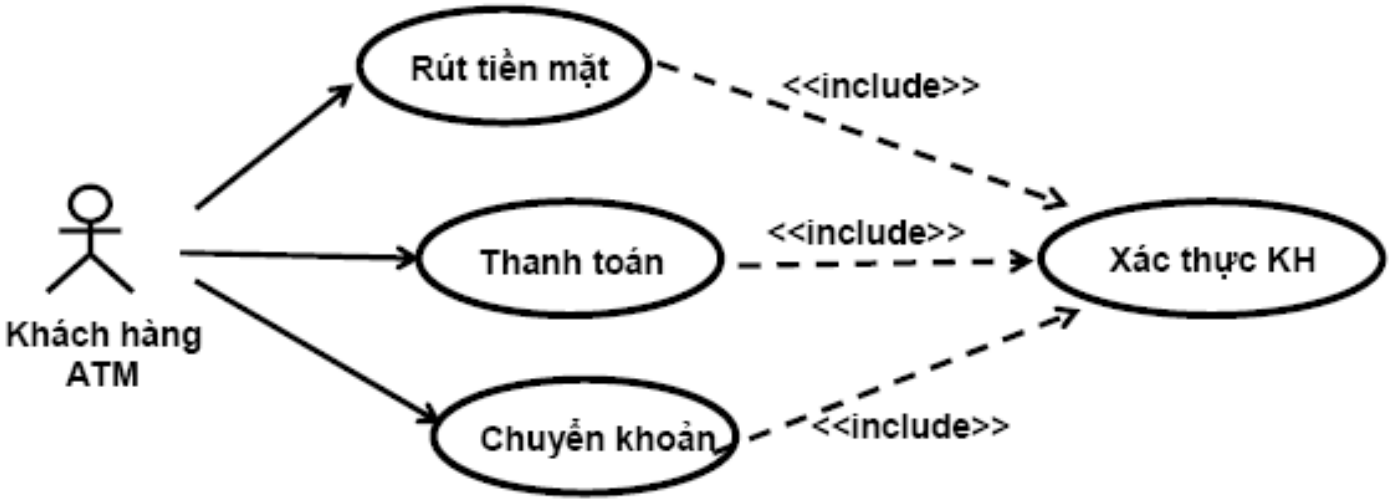
- Luồng chính (Basic flow)
  - Luồng lý tưởng mà Use case thường hoạt động
- Luồng phát sinh (Alternative flow)
  - Sử dụng nhiều lần trong luồng chính
  - Các trường hợp đặc biệt (vd nhấn mạnh một tính năng của HT)
  - Gây ra lỗi, cách xử lý lỗi trong tình huống đó
- Chú ý
  - Chỉ cần luồng chính là có thể hiểu được tác vụ chính mà Use Case đó sẽ thực thi
  - Phải có lời gọi luồng phát sinh từ luồng chính
  - Tránh viết luồng phát sinh dài hơn luồng chính
  - Tránh viết luồng phát sinh quá dài
  - Tránh tách quá nhiều luồng phát sinh

Luồng sự kiện của use-case (3)

- Kịch bản là một thể hiện của UC đó
  - Một Use Case có nhiều kịch bản tùy thuộc vào ngữ cảnh cụ thể mà nó phát sinh



Ví dụ



UC Rút tiền

1. Gọi UC "Xác thực KH"
2. Hiển thị menu.
3. KH chọn chức năng "Rút tiền"
4. ...

UC Xác thực KH

1. Đưa thẻ vào máy
2. Kiểm tra thẻ
3. KH nhập PIN
4. Hệ thống kiểm tra PIN

E1: Thẻ sai

E2: Sai PIN

E3: ...

Ví dụ đặc tả UC Rút tiền mặt

- **Luồng chính:**
  1. Gọi UC "Xác thực KH" để ktra KH
  2. Hiển thị menu
  3. KH chọn chức năng "Rút tiền"
  4. Nhập số tiền cần rút
  5. HT gửi giao dịch tới ngân hàng chờ chấp thuận
  6. Máy ATM xuất tiền (tiền giấy)
  7. Trả lại thẻ
  8. In biên lai

- **Luồng phát sinh:**
  - Luồng phát sinh "Rút tiền xu" phát sinh tại bước 6 trong luồng chính
  - (Có thể có luồng phát sinh khác như Hết tiền…)
- **Luồng phát sinh Rút tiền xu:**

  (Phần này có thể viết chung với UC Rút tiền, hoặc có thể viết riêng như một UC nếu nó tương đối phức tạp)
  1. Nếu KH chọn thể loại tiền xu
  2. KH nhập số lượng xu
  3. Hệ thống tính ra tổng số tiền cần rút
  4. KH chấp nhận
  5. Khay tiền xu mở để xuất tiền
  6. UC Rút tiền tiếp tục thực hiện

Biểu đồ hoạt động

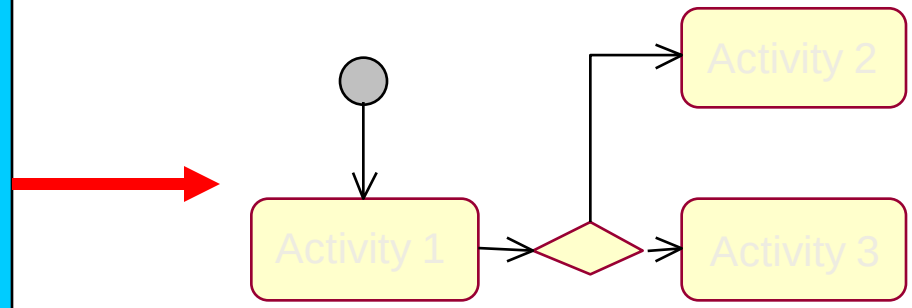
- Biểu đồ hoạt động trong mô hình use case được sử dụng để lưu lại các hoạt động và các hành động được thực hiện trong một use case → Minh họa luồng sự kiện
  - Biểu đồ luồng (flow chart): Chỉ ra luồng điều khiển từ hoạt động hoặc hành động này đến hoạt động/hành động khác.

Flow of Events

This use case starts when the Registrar requests that the system close registration.

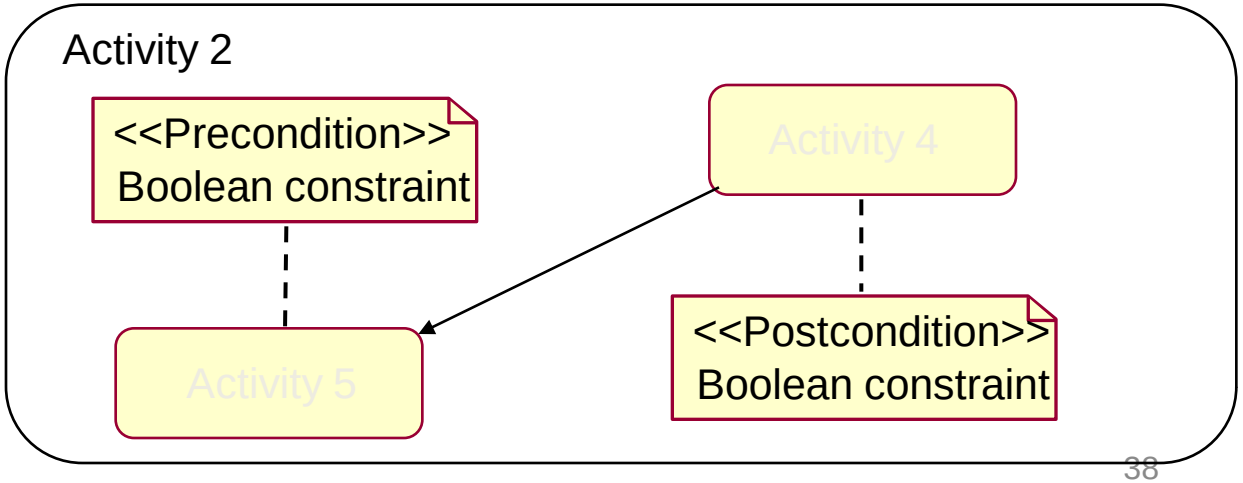
1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.



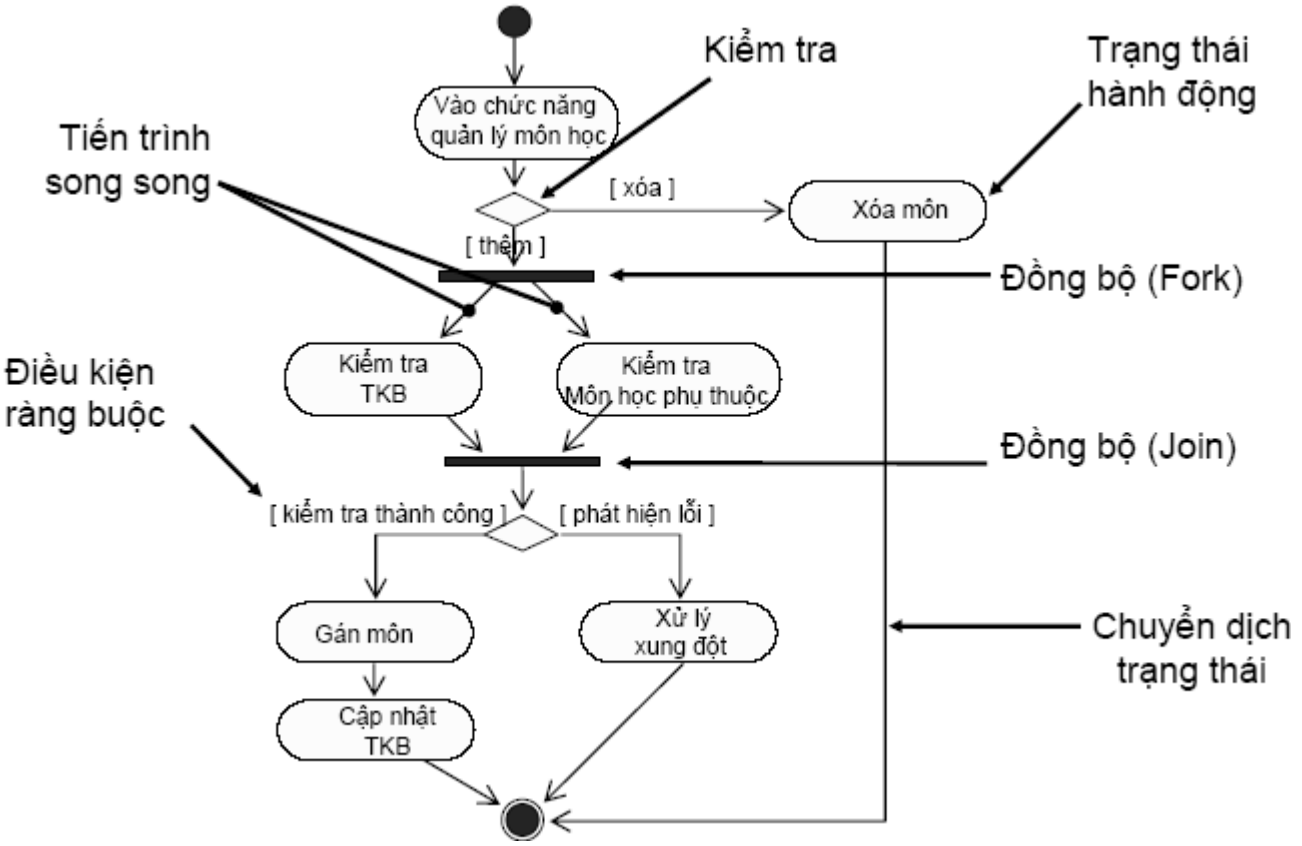
Biểu đồ hoạt động (2)

- Hoạt động
  - Đặc tả cho hành vi được diễn tả như một luồng thực thi thông qua sự sắp xếp thứ tự của các đơn vị nhỏ hơn.
  - Các đơn vị nhỏ hơn bao gồm các hoạt động lồng nhau và các hành động riêng lẻ cơ bản
  - Có thể chứa các ràng buộc biểu thức logic khi hoạt động được gọi hoặc kết thúc



Biểu đồ hoạt động

- Được sử dụng để minh hoạ luồng sự kiện



Nội dung

1. Mô hình hóa yêu cầu
2. Biểu đồ use case
3. Đặc tả use case
4.
5. trợ

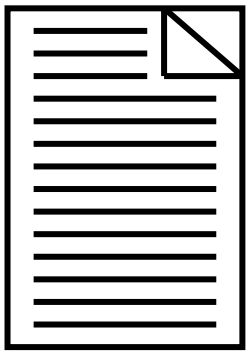
(Glossary)

- n.
  - ng.
  - n.
  - n.
- n
  Inception Elaboratio
  n.

(2)

- u
  - m
  - n.
- n:
  - **Introduction**: .
  - **Terms**: ng.

(3)



Glossary

Course Registration System Glossary

1. Introduction

This document is used to define terminology specific to the problem domain, explaining terms, which may be unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal *data dictionary*, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

2. Definitions

The glossary contains the working definitions for the key concepts in the Course Registration System.

2.1 Course: A class offered by the university.

2.2 Course Offering: A specific delivery of the course for a specific semester – you could run the same course in parallel sessions in the semester. Includes the days of the week and times it is offered.

2.3 Course Catalog: The unabridged catalog of all courses offered by the university.

Nội dung

1. Mô hình hóa yêu cầu
2. Biểu đồ use case
3. Đặc tả use case
4.
5.

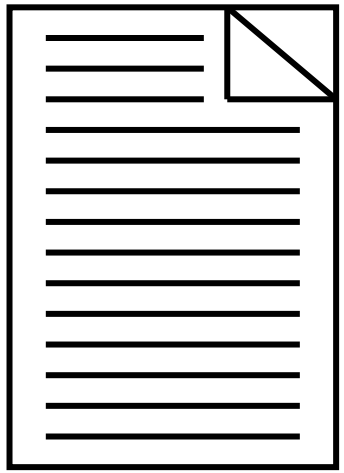
(Supplementary Specification)

-

ng

-

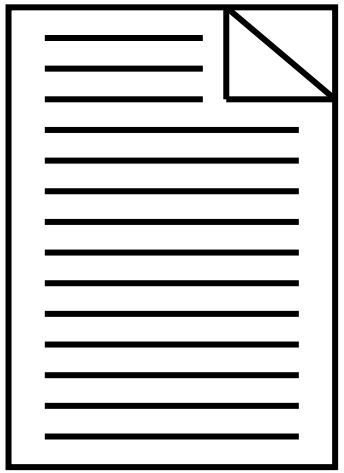
Construction.



Supplementary Specification

(Supplementary Specification)

- o
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability
  - Design constraints



Supplementary Specification

(2)

- c năng (Functionality)
  - c use case
- ng (Usability)
  - ng.
- y (Reliability)
  - .

(3)

- u năng (Performance)
  - –
    i,...
- (Supportability)
  - –
    ng
- (design constraints)

Tổng kết

- Đặc tả yêu cầu
  - Đóng vai trò như một thỏa thuận giữa khách hàng, người sử dụng và những người phát triển hệ thống
    - Cho phép khách hàng và người sử dụng kiểm tra những chức năng mà họ mong đợi hệ thống sẽ thực hiện
    - Người phát triển có thể hiểu được cần phải làm gì
  - Cần tuân thủ theo một quy trình hợp lý
- Một số chú ý khi đặc tả yêu cầu với mô hình UC
  - Sử dụng mẫu tài liệu, hướng dẫn
  - Phát triển các luồng sự kiện đơn giản, rõ ràng về mặt nghiệp vụ (trách đưa ra các thông tin quá chi tiết, kỹ thuật)
  - Xây dựng biểu đồ hoạt động cho những vấn đề phức tạp

Case study – Course Registration

- Actor?
- Use case?
- Relationship?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

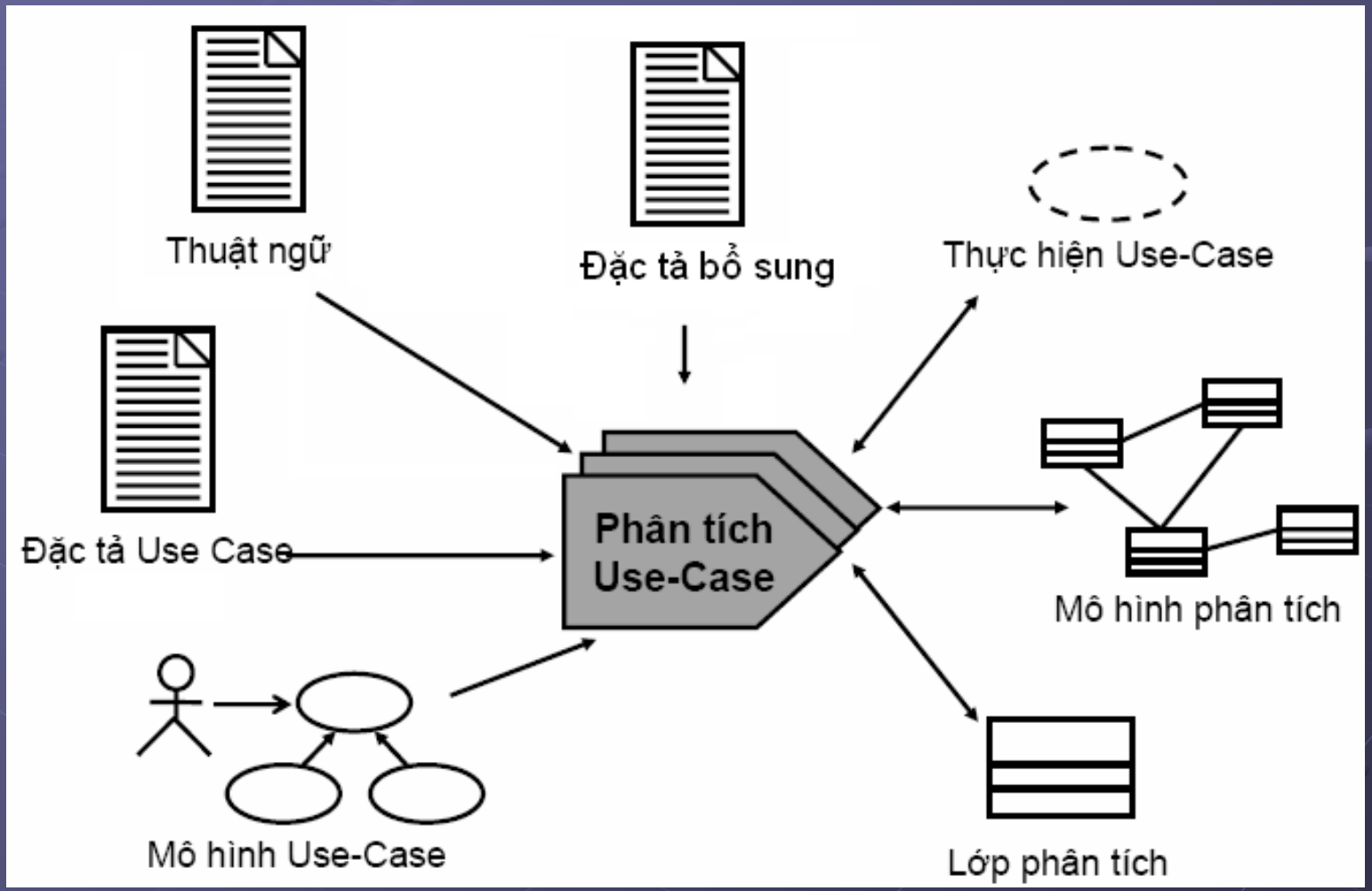
OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 5. Phân tích use case

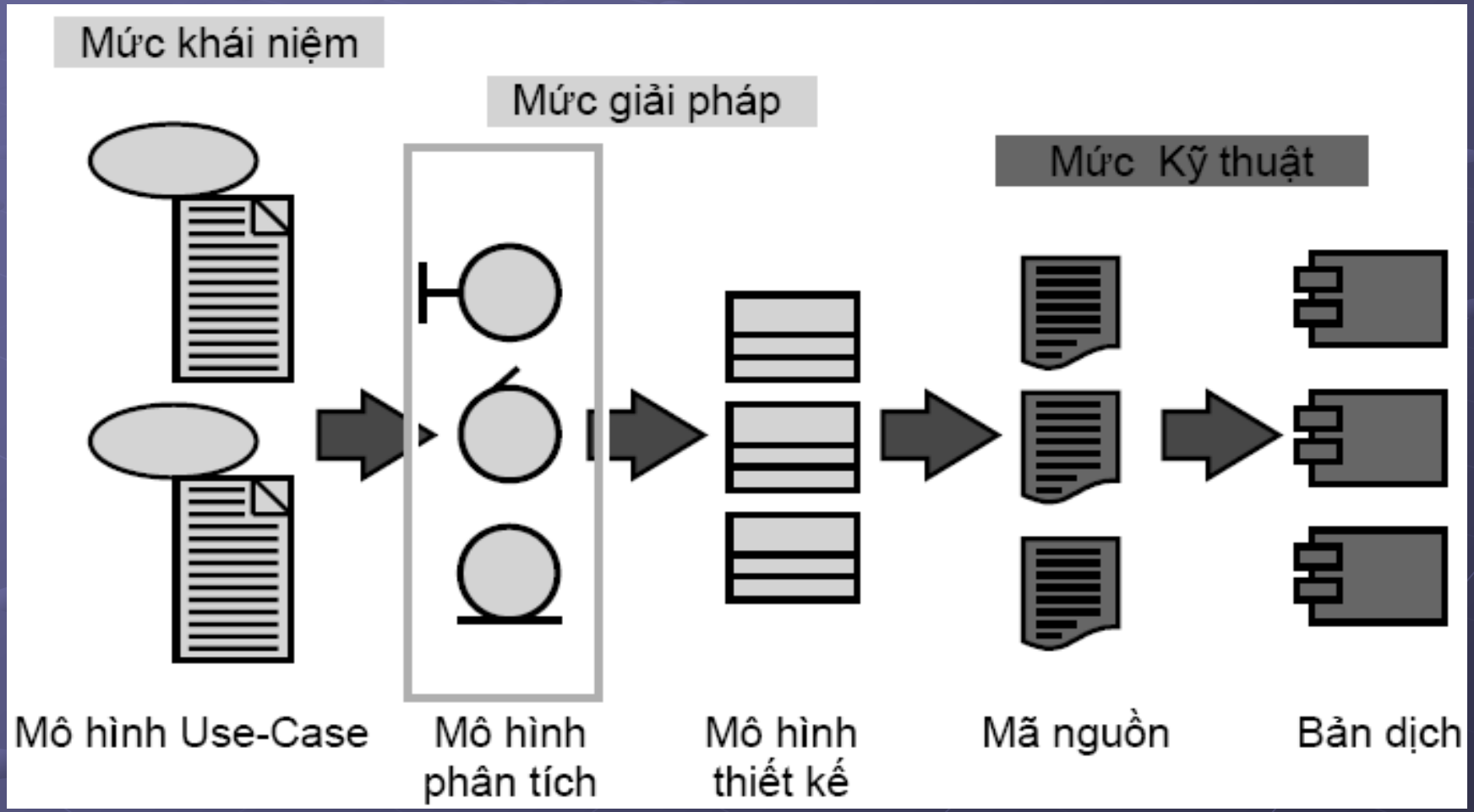
Nội dung

1. Tổng quan về phân tích use case
2. c lớp phân tích
3. Các biểu đồ tương tác
4. ch

1. Tổng quan về phân tích UC



Mô hình phân tích trong quá trình phát triển



Mô hình phân tích là quá trình trung gian

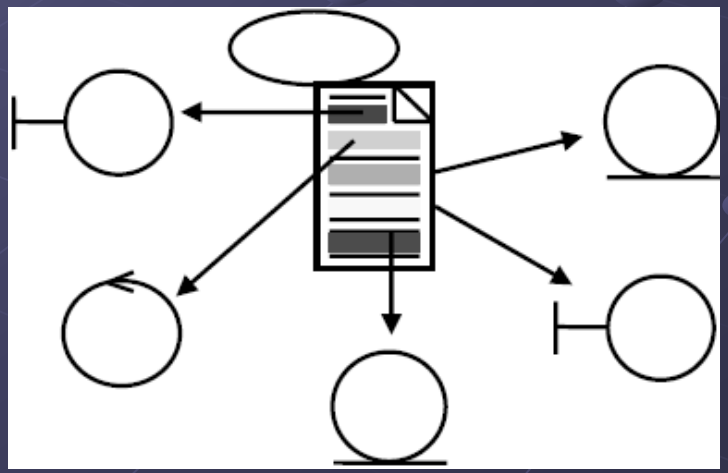
- Mô hình phân tích là mô hình ở mức khái niệm về hệ thống sẽ làm gì
  - Được phát triển và tiến triển nhanh tới giai đoạn tiếp theo
  - Dễ bị thay đổi để đảm bảo mức độ hoàn thiện hơn khi phát triển HT
- Các lớp phân tích thường sẽ "mất đi" khi giai đoạn thiết kế hoàn thành
  - Có thể coi lớp phân tích như là các lớp "non" để thể hiện một hành vi nào đó
- Không nên dành quá nhiều thời gian để tạo ra các mô hình này một cách quá chi tiết, nó sẽ bị thay đổi tại giai đoạn thiết kế

Nội dung

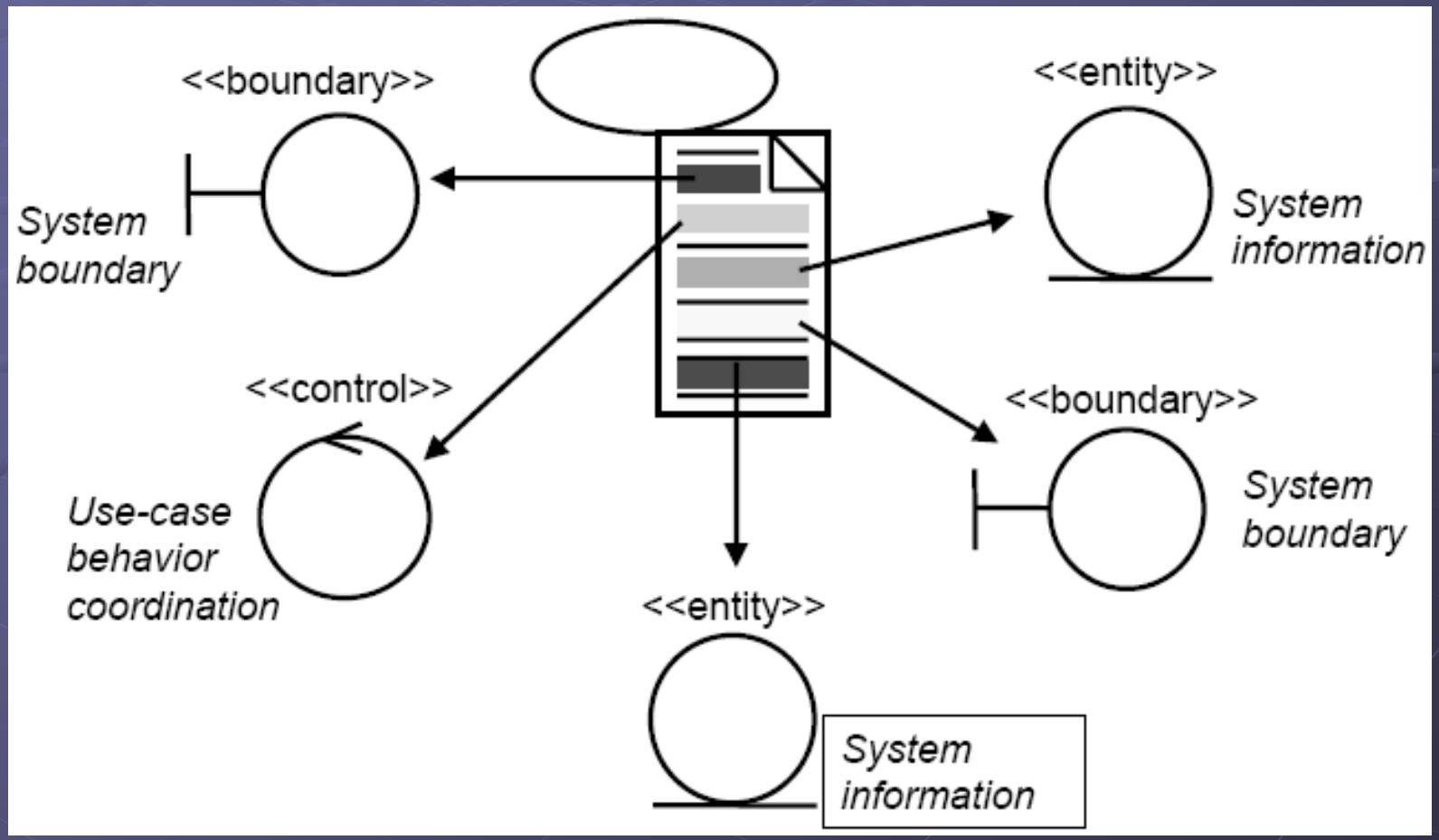
1. Tổng quan về phân tích use case
2. c lớp phân tích
3. Các biểu đồ tương tác
4. ch

Tìm các lớp phân tích trong UC

- Tìm các lớp phân tích trong Use Case
  - Mỗi lớp phải có nhiệm vụ và trách nhiệm cụ thể
  - Lớp phân tích chỉ là kết quả của quá trình trừu tượng hóa
    - Thực tế nó có thể là một hệ thống con
    - Hoặc Kết tập nhiều lớp trong bước thiết kế tiếp theo
- Chức năng tổng thể của Use-Case đó phải được phản ánh đầy đủ trong các lớp phân tích

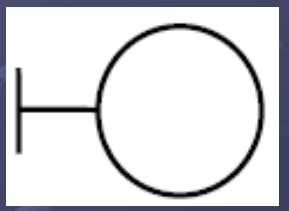


Các loại lớp phân tích



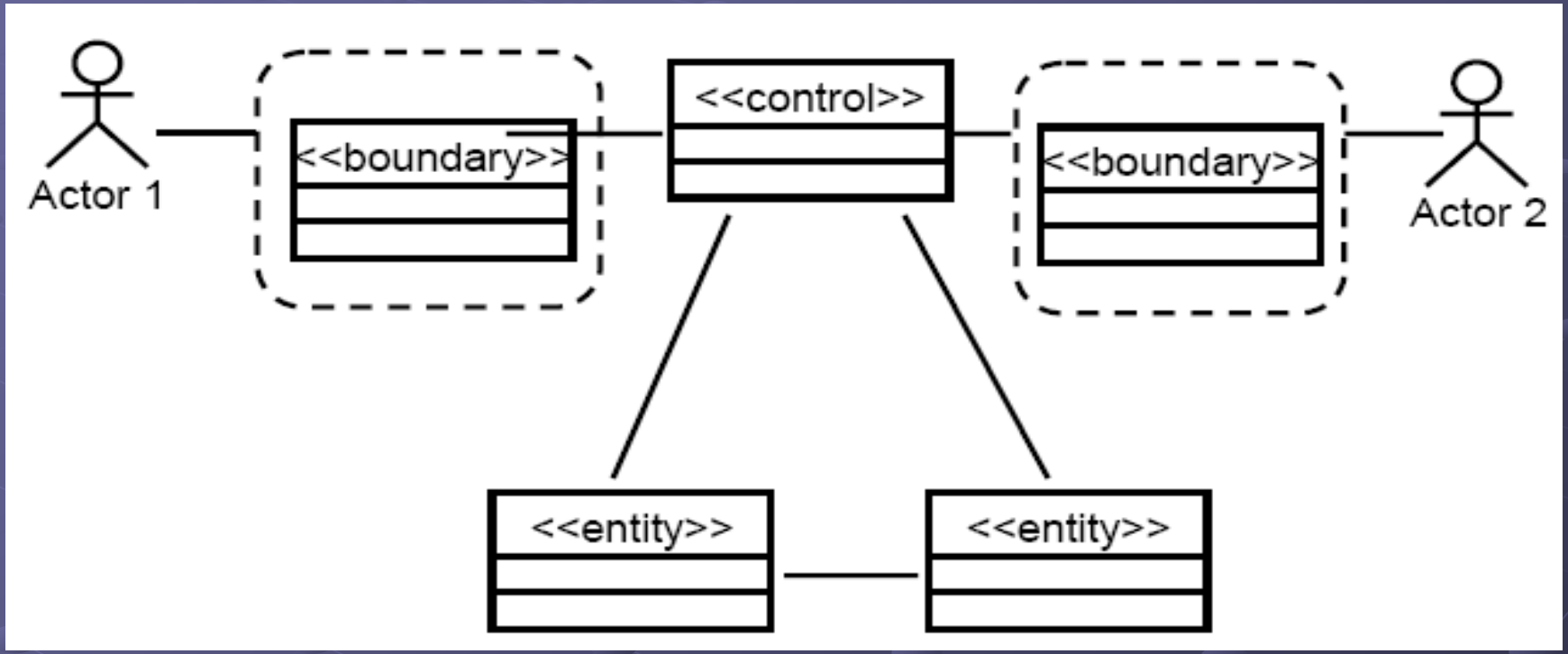
2.1. Lớp biên (Boundary class)

- Là lớp trung gian thể hiện sự tương tác giữa hệ thống và những gì bên ngoài hệ thống



- Các lớp biên:
  - Lớp giao diện giữa người dùng và hệ thống
  - Lớp giữa hệ thống và các hệ thống bên ngoài
    - Ví dụ giao dịch với “Hệ thống tài vụ”
  - Lớp giữa hệ thống và thiết bị ngoại vi
    - Ví dụ “Thiết bị giải mã vạch”
- Với mỗi cặp Actor/Use-Case bao giờ cũng có 1 lớp biên

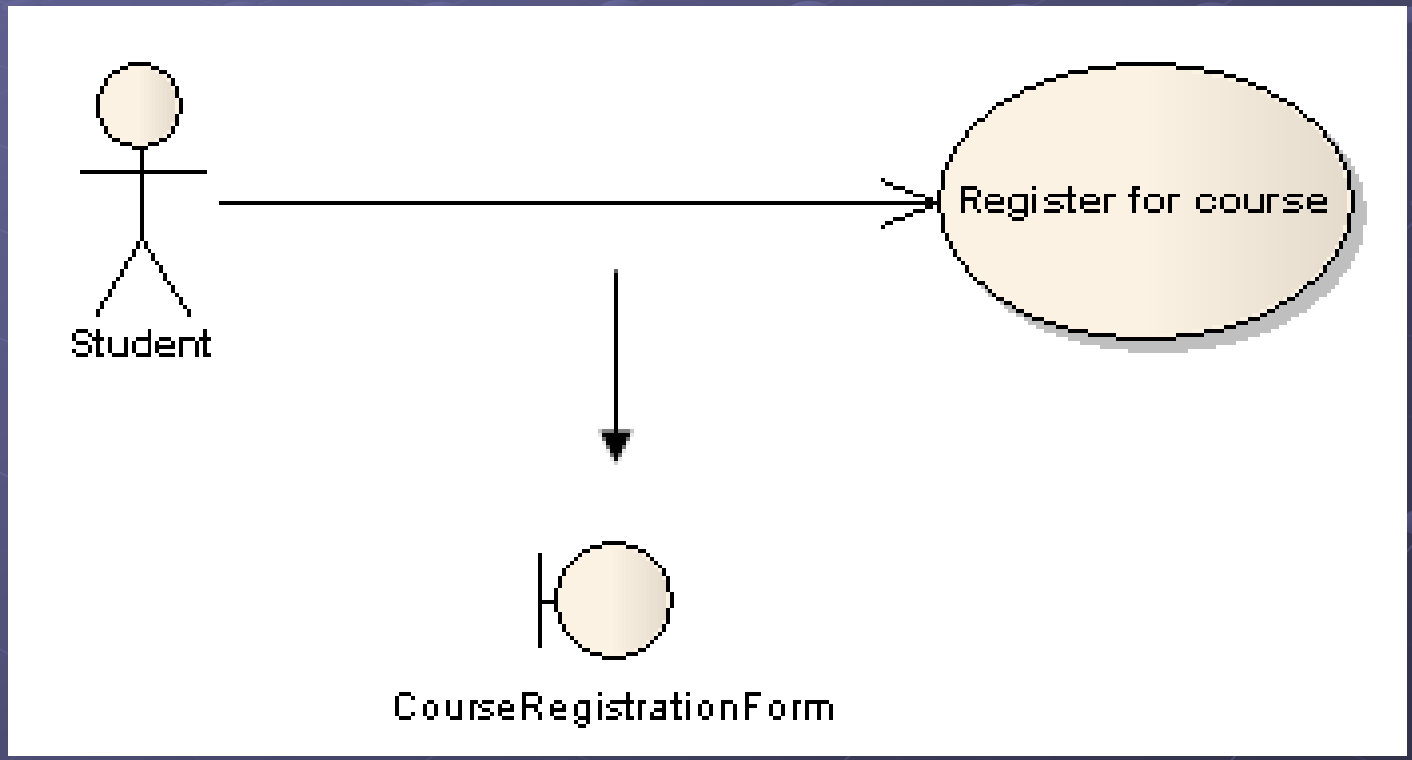
Vai trò của lớp biên



Mô hình hoá sự tương tác giữa hệ thống và môi trường bao quanh nó

UC Dangkyhoc: Tìm lớp biên

- Ít nhất một lớp biên cho mỗi cặp actor/use case
  - Ví dụ:

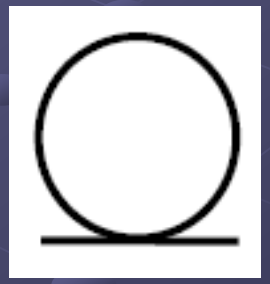


Một số chú ý với lớp biên

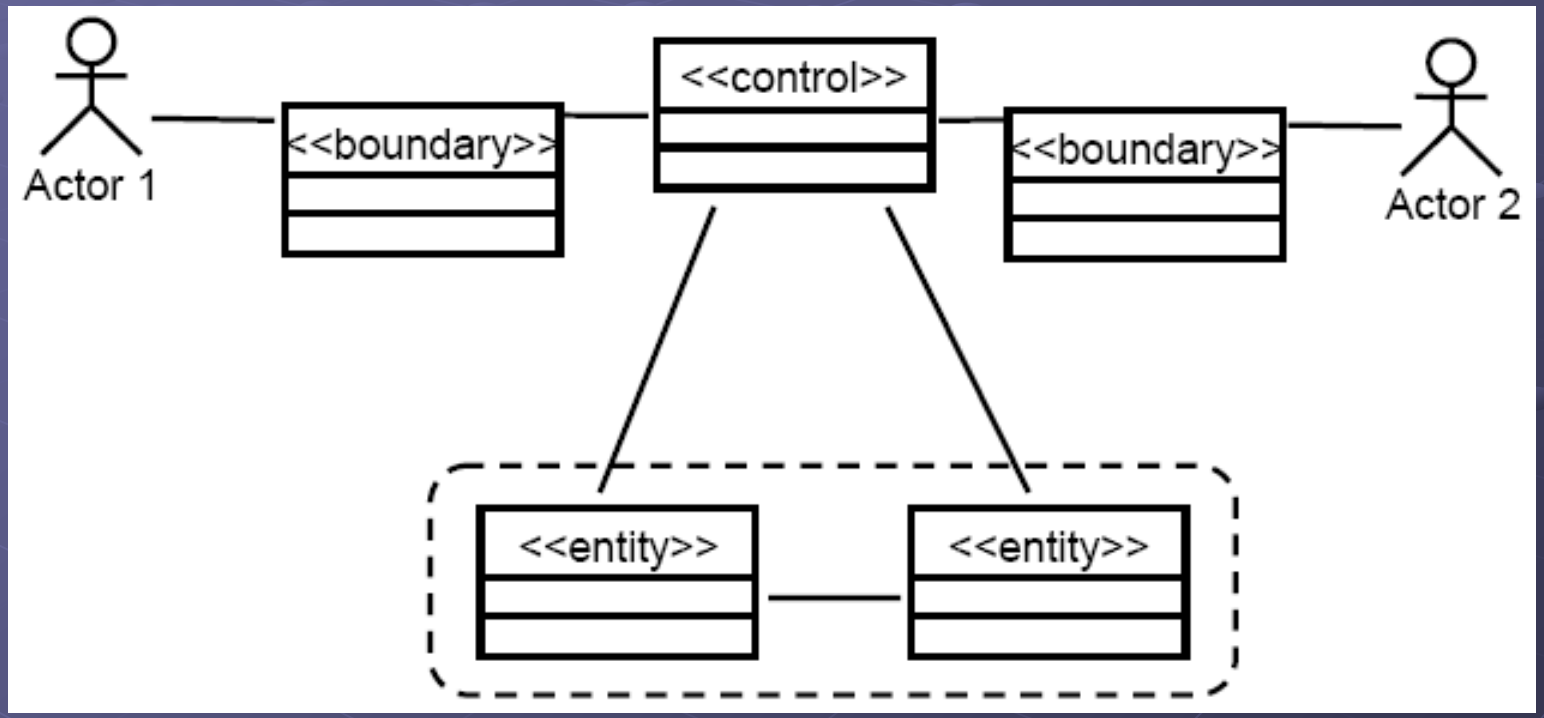
- Các lớp giao diện người dùng (GUI)
  - Tập trung vào cấu trúc thông tin cần thiết cho người dùng
  - Không tập trung vào chi tiết giao diện người dùng
- Các lớp giao diện hệ thống và thiết bị ngoại vi (API)
  - Tập trung vào cấu trúc dữ liệu trao đổi giữa chúng
  - Tập trung vào giao thức tương tác giữa chúng với hệ thống ở mức cao
  - Không quan tâm đến việc giao thức được thực thi thế nào và dữ liệu được truyền đi thế nào

2.2. Lớp thực thể (Entity class)

- Là các lớp mô tả những thực thể chính xuất hiện trong hệ thống
- Thực thể là những thông tin tồn tại và được lưu trữ lâu dài trong hệ thống
- Chỉ mô tả ở mức trừu tượng, không mô tả quá chi tiết các thuộc tính của thực thể này



Vai trò của lớp thực thể



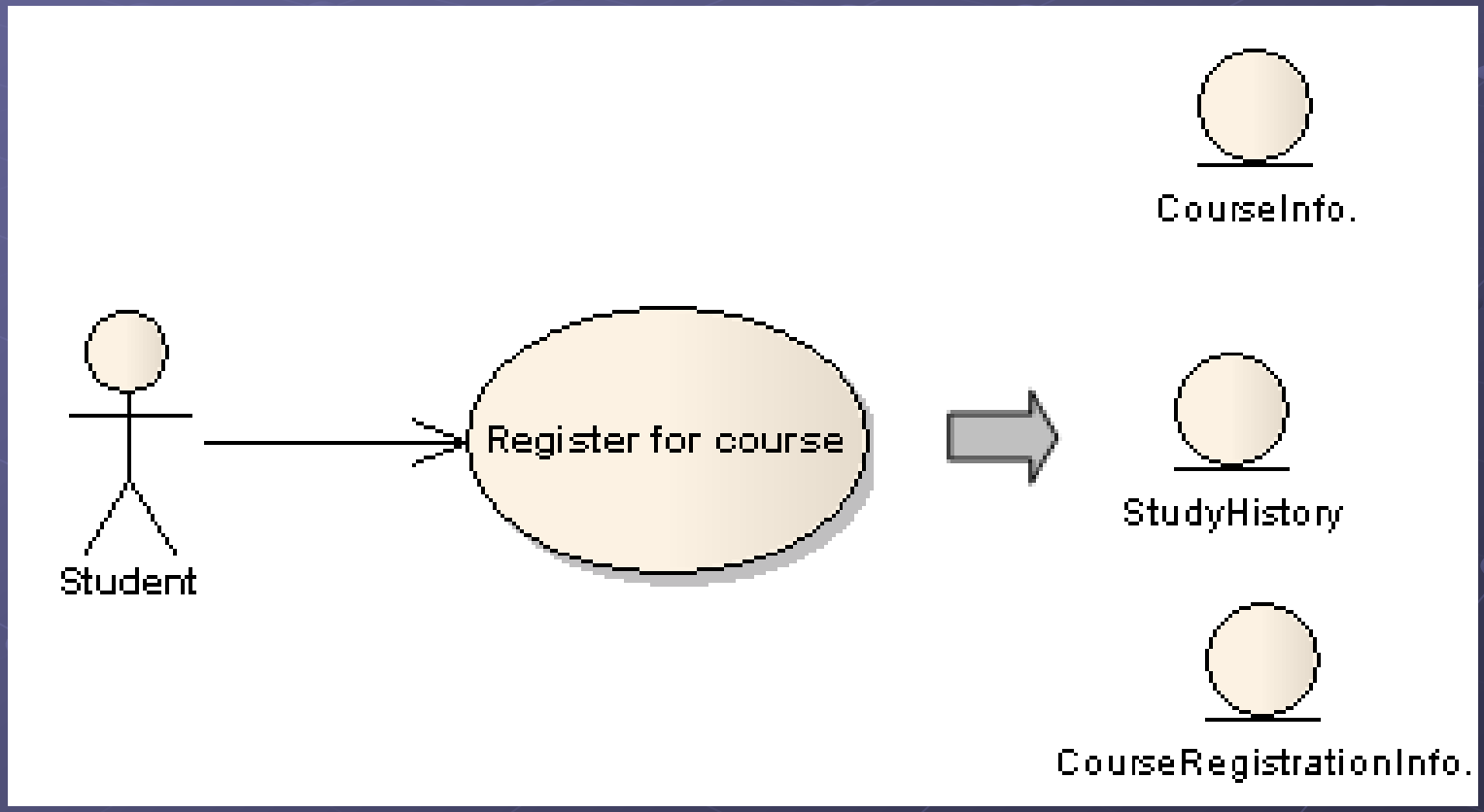
Lưu trữ và quản lý thông tin trong hệ thống

Tìm các lớp thực thể

- Sử dụng luồng sự kiện của Use-Case là đầu vào
- Lọc các danh từ
  - Tìm các mệnh đề danh từ trong luồng sự kiện
  - Loại bỏ một số thành phần không cần thiết
    - Thừa, lặp, không rõ ràng
  - Loại bỏ các từ mô tả cụ thể một thuộc tính thông tin nào đó, nhưng lưu lại để sau này có thể sử dụng cho:
    - Thuộc tính
    - Thao tác

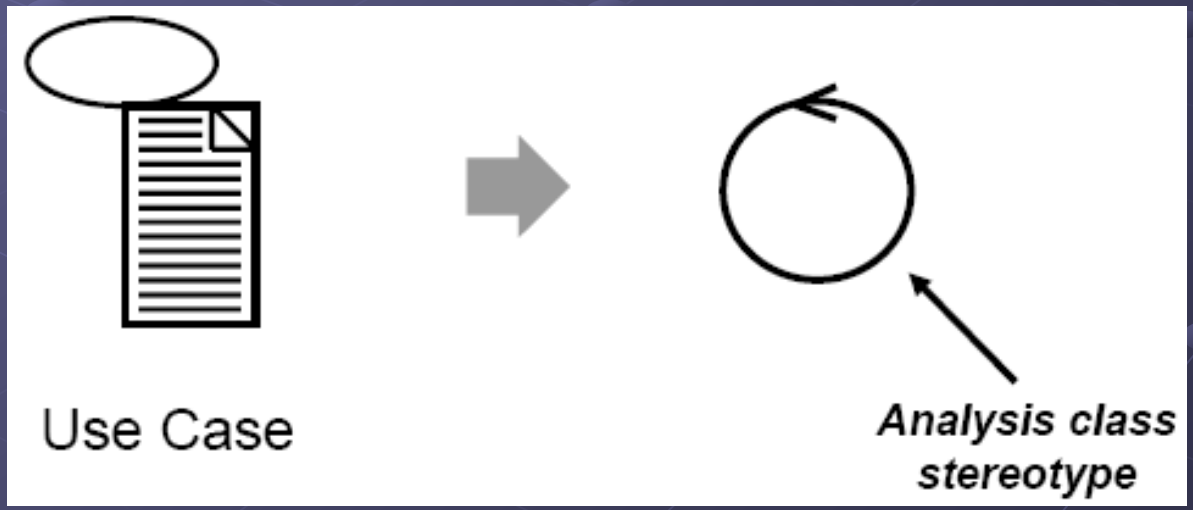
UC Dangkyhoc: Tìm các lớp thực thể

- Chức năng đăng ký học (Tạo thời khóa biểu)

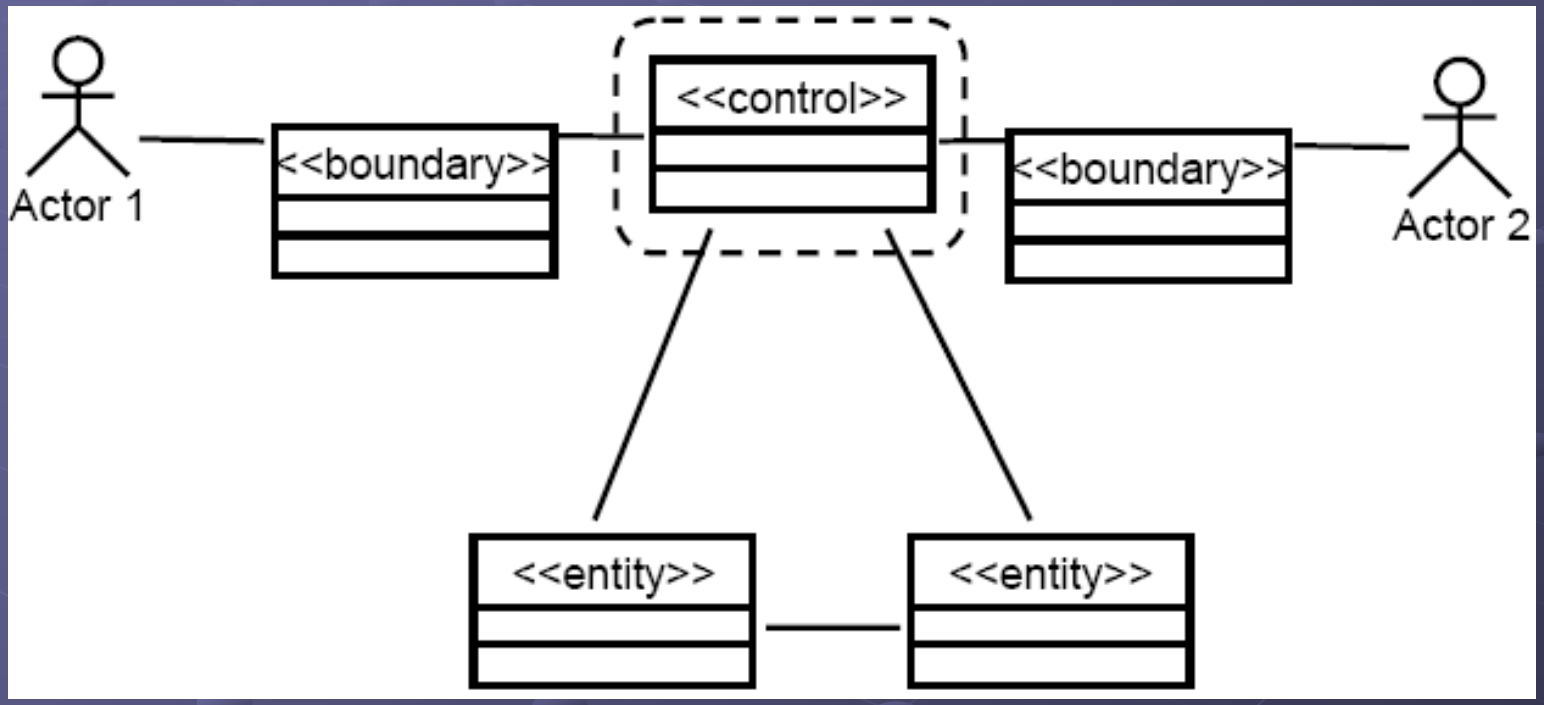


2.3. Lớp điều khiển (Control class)

- Được sử dụng để thực hiện một hoặc nhiều hành động nào đó trong hệ thống
  - Là lớp thực hiện chức năng chính trong các UC
  - Với những Use Case phức tạp, có thể có nhiều hơn một lớp điều khiển



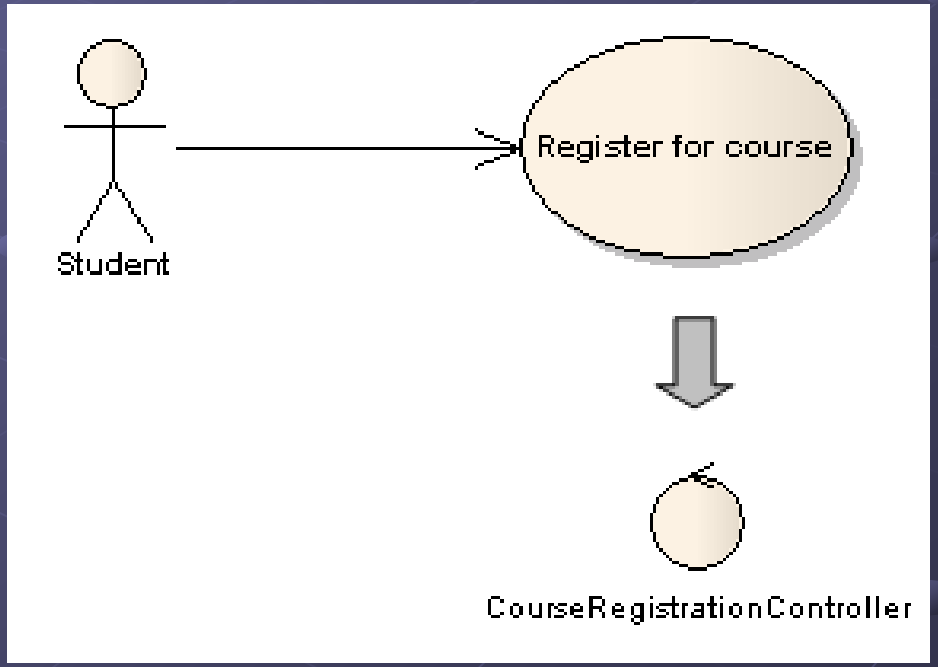
Vai trò của lớp điều khiển



Thể hiện hành động, chức năng của từng Use Case

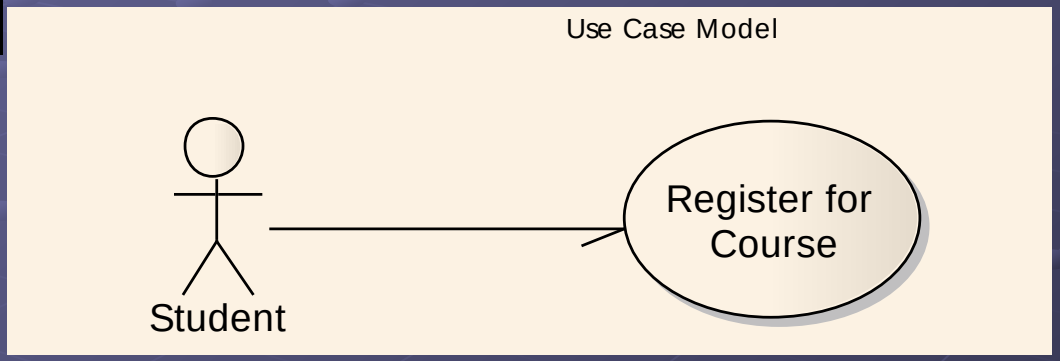
Tìm các lớp điều khiển

- Đơn giản nhất phải tìm được một lớp điều khiển cho một Use-Case
  - Với các Use-Case phức tạp có thể yêu cầu nhiều lớp điều khiển

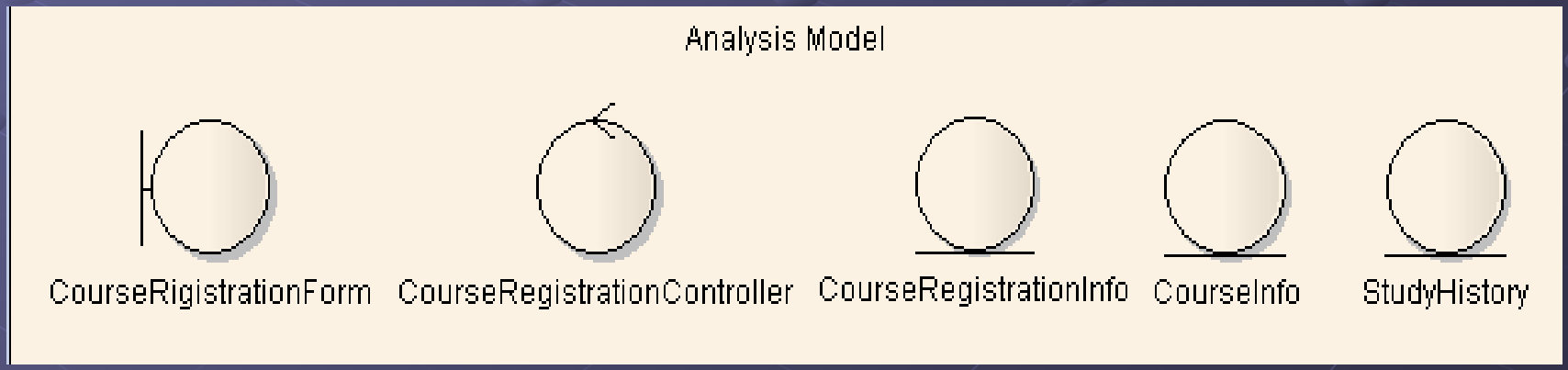


UC Register for Course: Lớp phân tích

- Mô hình use case

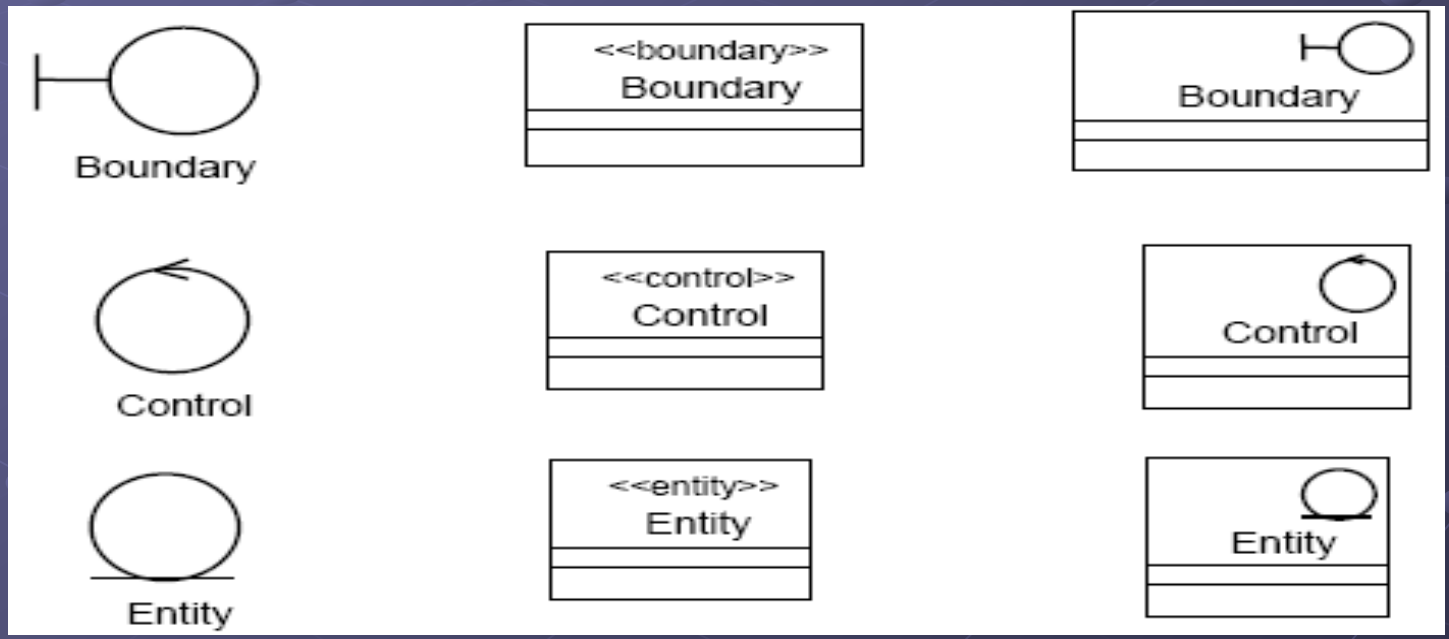


- Mô hình phân tích và thiết kế



Các biểu tượng cho lớp phân tích trong UML

- UML cho phép sử dụng một số biểu tượng khác nhau cho các lớp phân tích
  - Các lớp phân tích được biểu thị cùng với stereotype

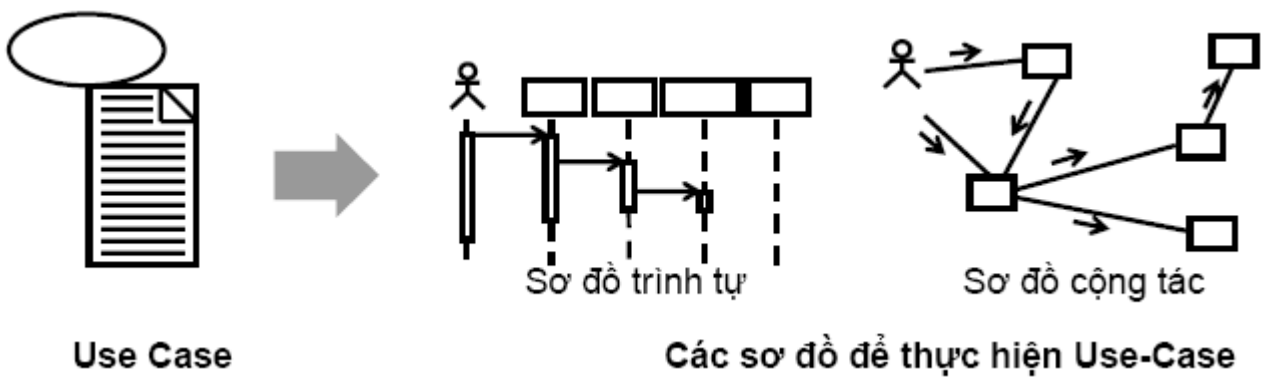


Nội dung

1. Tổng quan về phân tích use case
2. c lớp phân tích
3. Các biểu đồ tương tác
4. ch

Phân bổ các hành vi của Use Case vào các lớp

- Trong từng luồng sự kiện của từng UC
  - Tìm ra các lớp phân tích
  - Phân bố chức năng (hành vi) của Use Case này vào các lớp tìm được
- Thể hiện tương tác giữa các lớp và hành vi của chúng bằng các mô hình tương tác

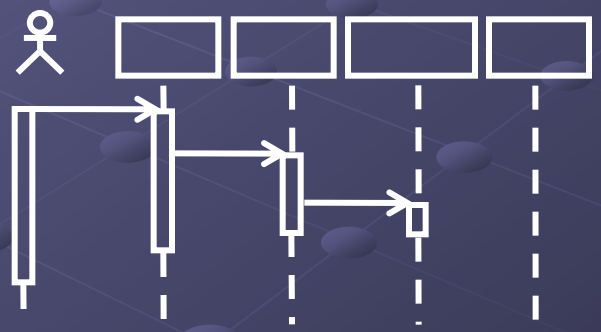


Trách nhiệm của các lớp phân tích

- Lớp biên
  - Chịu trách nhiệm thể hiện sự tương tác giữa hệ thống và tác nhân bên ngoài
  - Chịu trách nhiệm kiểm tra dữ liệu qua lại trong quá trình tương tác
- Lớp thực thể
  - Chịu trách nhiệm quản lý thông tin của nó
  - Đóng gói thông tin, và thay đổi trạng thái của nó
- Lớp điều khiển
  - Chịu trách nhiệm chính cho một Use Case nào đó
  - Tránh để lớp điều khiển làm quá ít việc

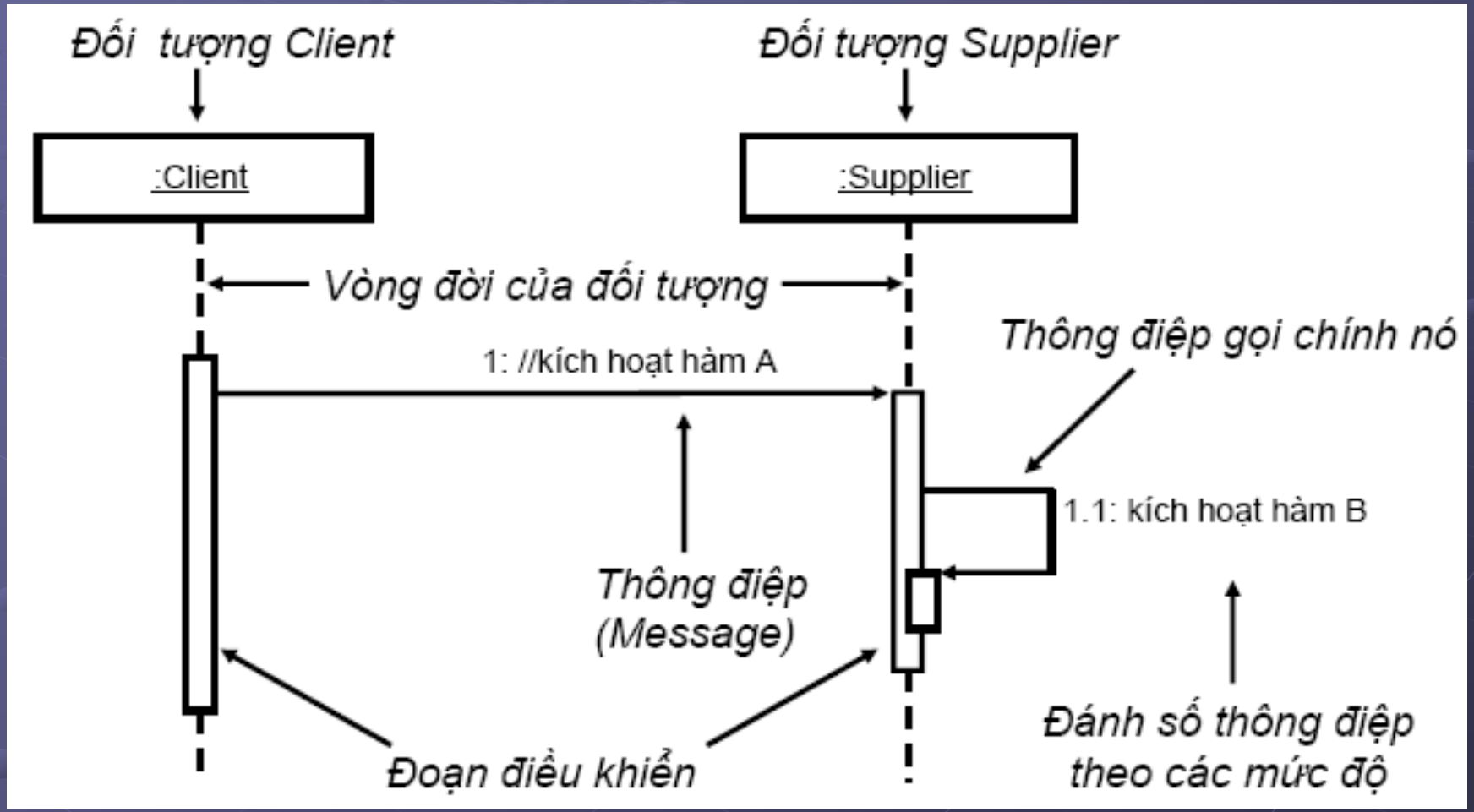
3.1. Biểu đồ trình tự (Sequence diagram)

- Là biểu đồ tương tác tập trung vào thứ tự trao đổi các thông điệp theo thời gian
- Chỉ ra:
  - Các đối tượng tham gia tương tác
  - Trình tự các thông điệp trao đổi với nhau

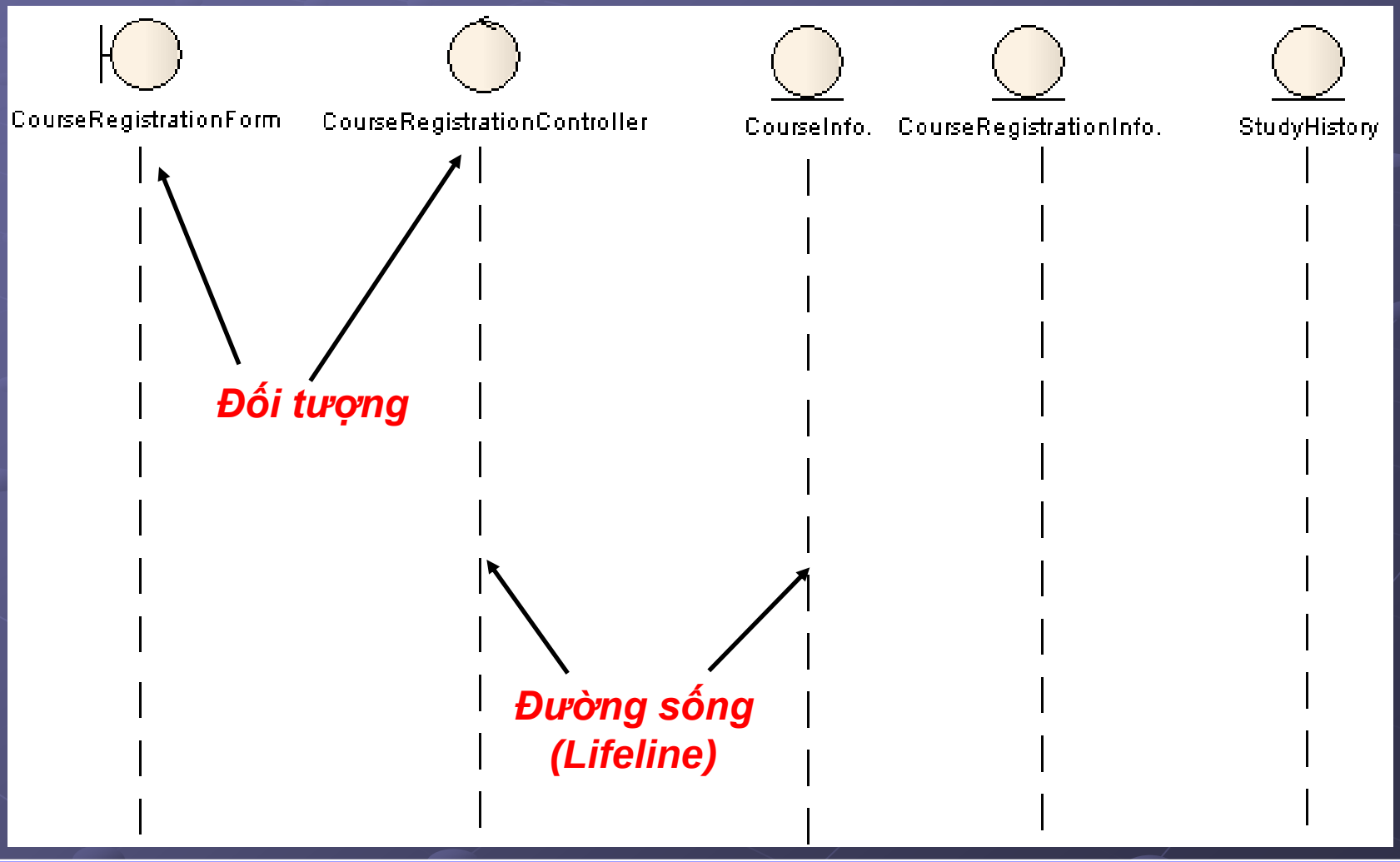


Sequence Diagram

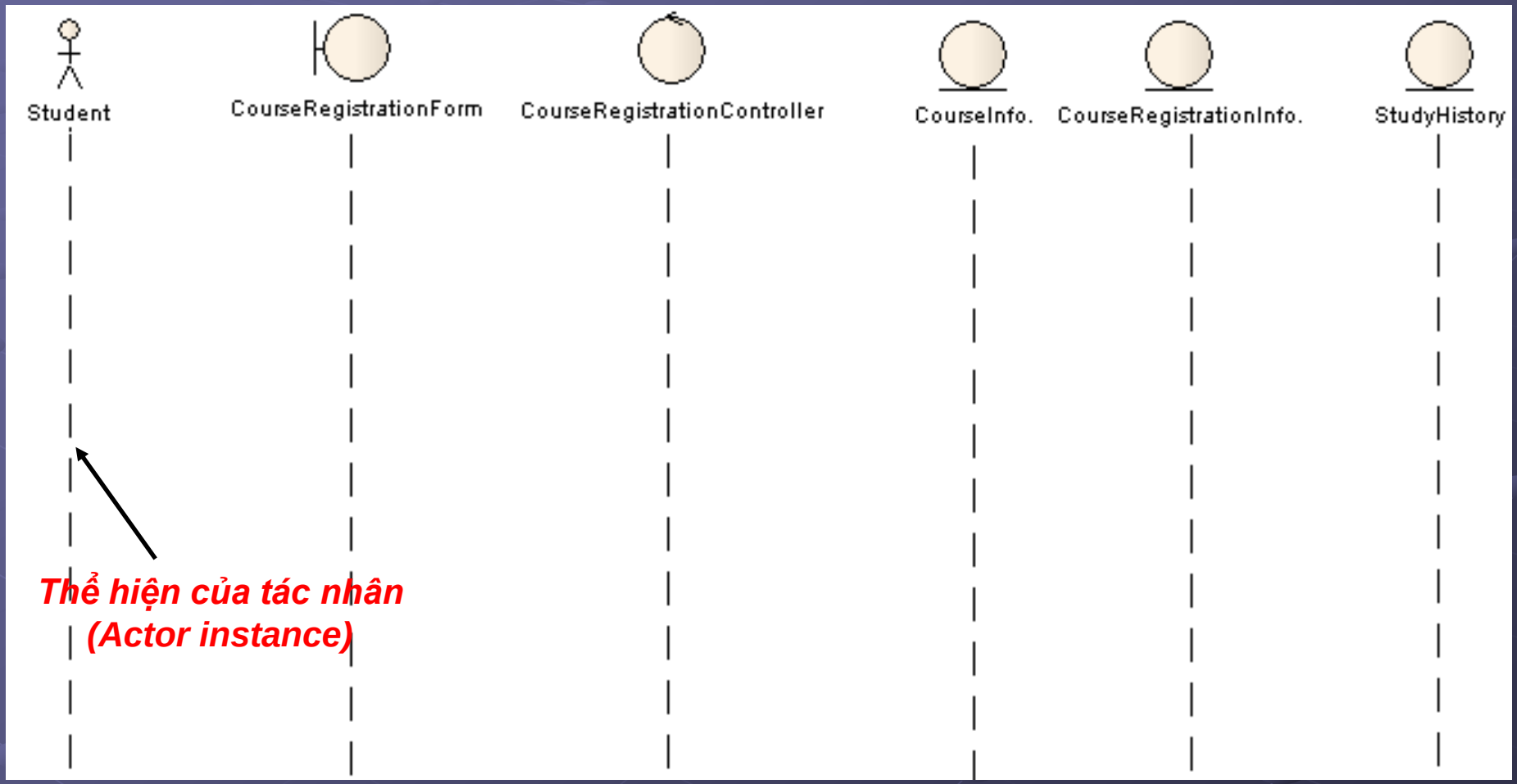
Biểu đồ trình tự



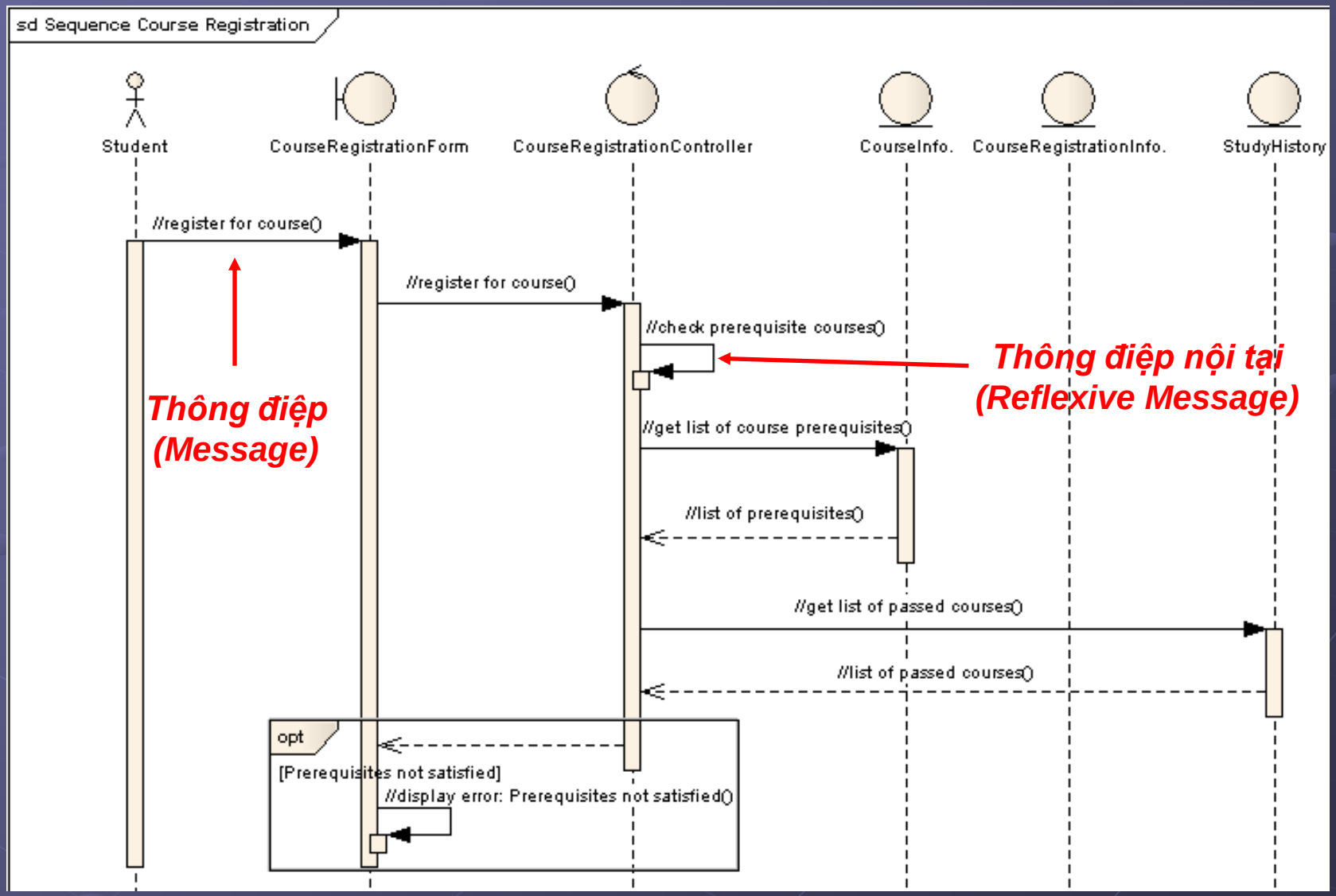
Biểu đồ trình tự - đối tượng



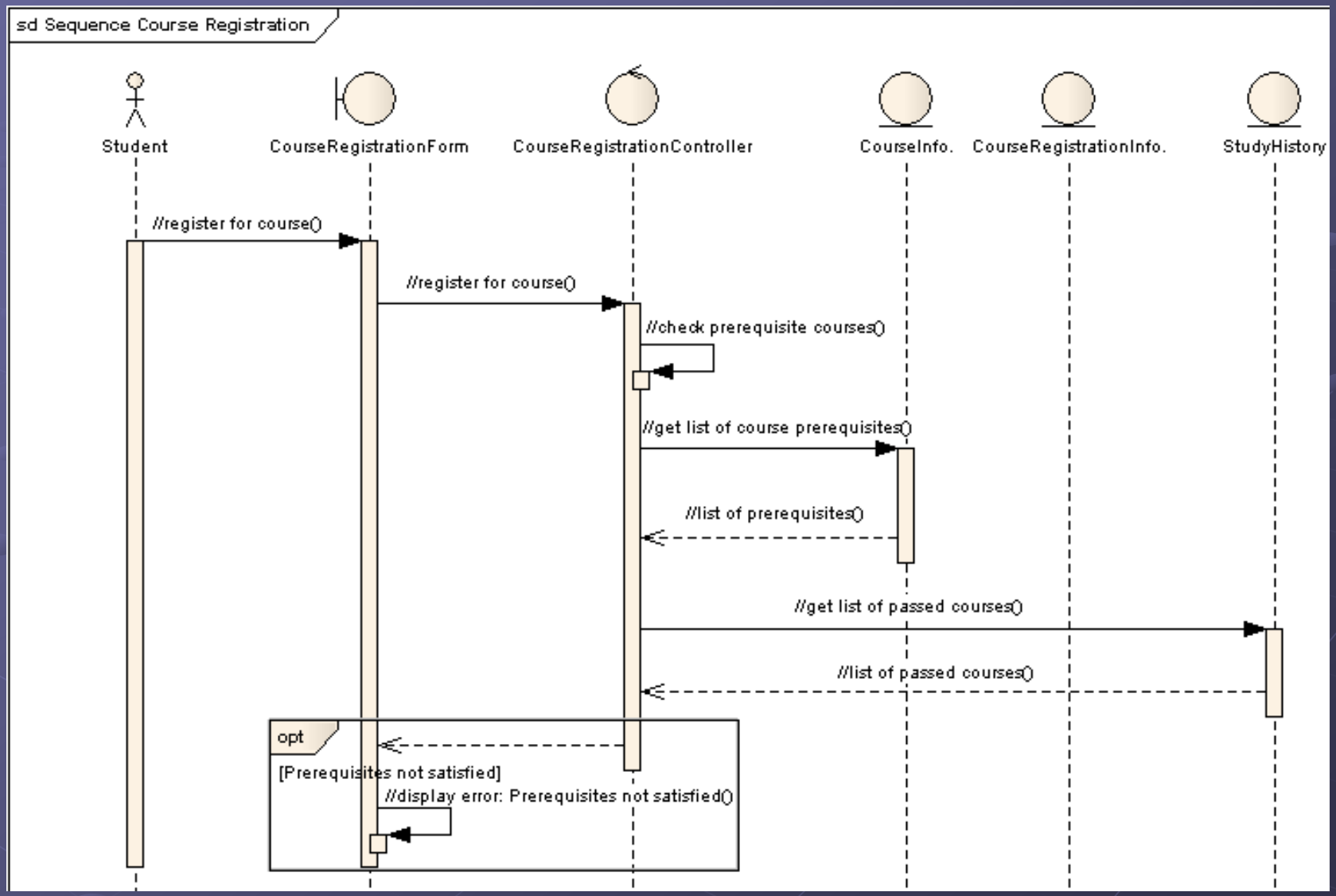
Biểu đồ trình tự - tác nhân



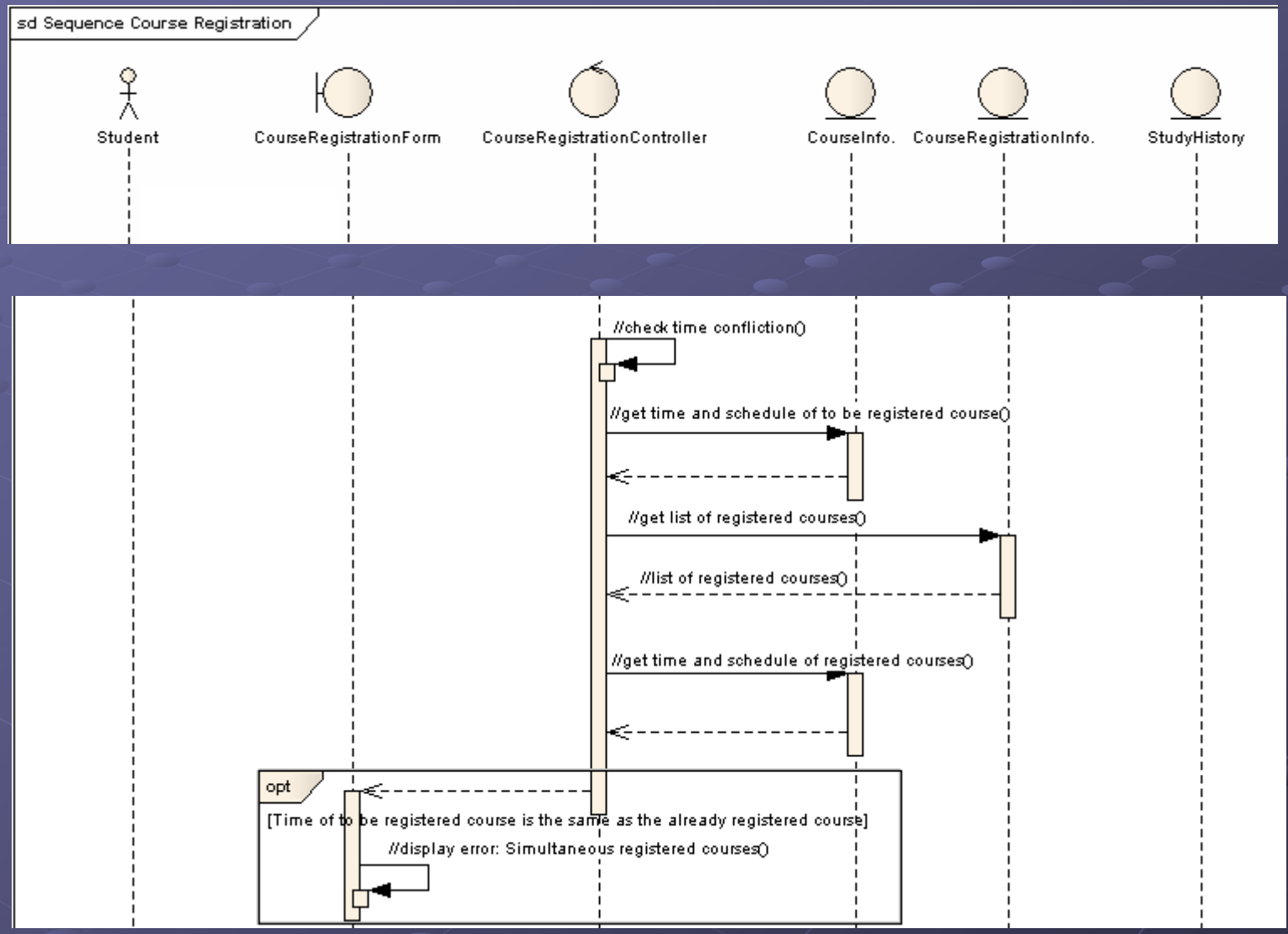
Biểu đồ trình tự - Thông điệp



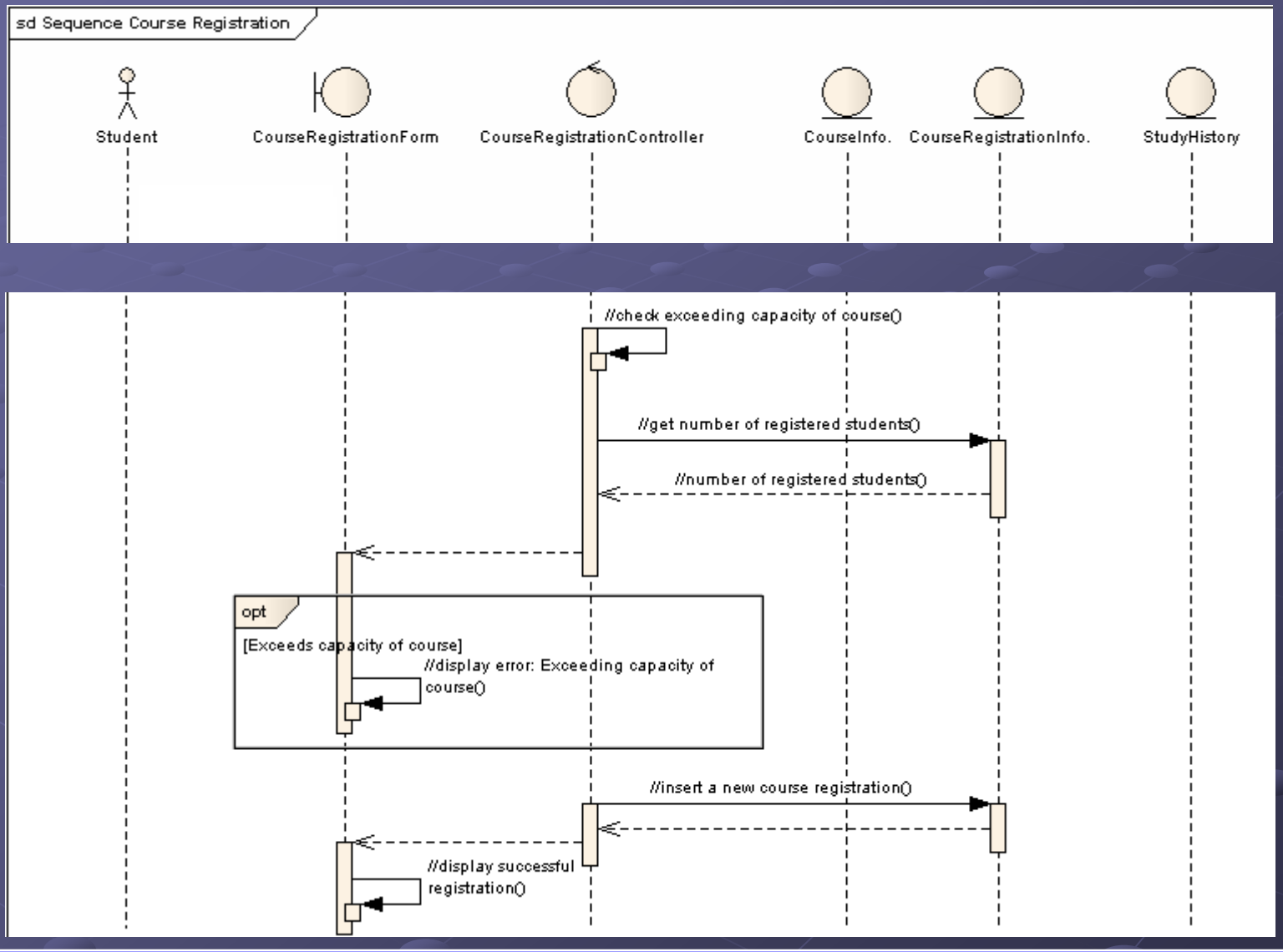
UC Register ForCourse: Sequence Diagram



UC Register ForCourse: Sequence Diagram

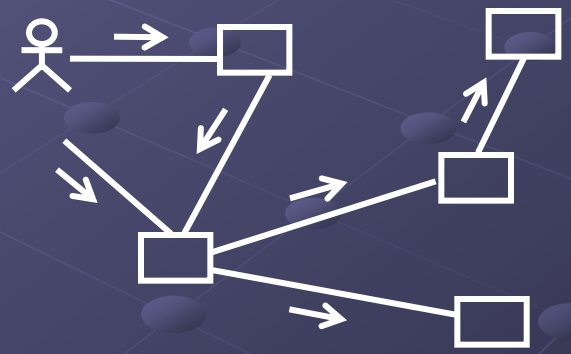


UC Register ForCourse: Sequence Diagram



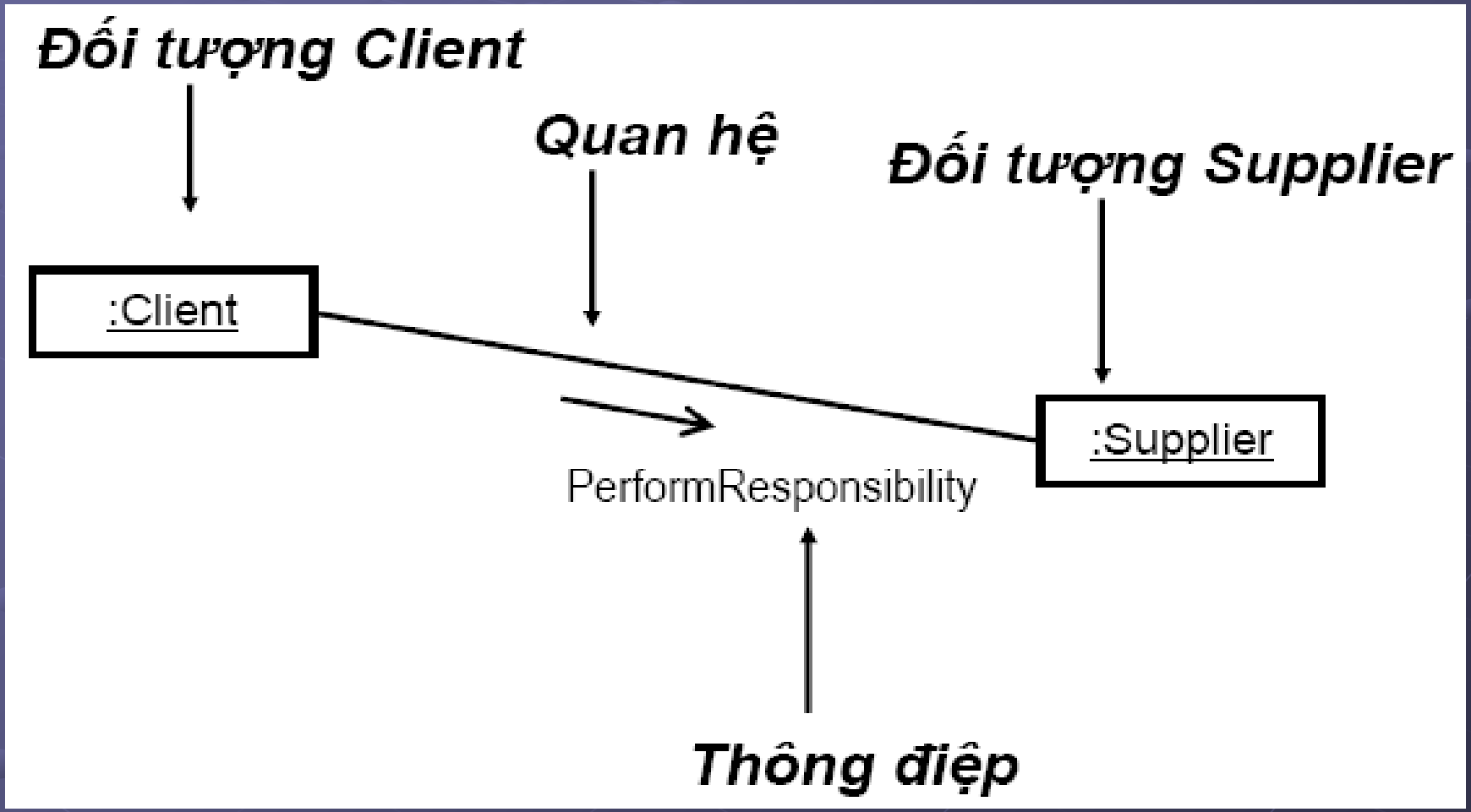
3.2. Biểu đồ giao tiếp

- Là biểu đồ tương tác tập trung vào tổ chức các đối tượng tham gia tương tác.
- Chỉ ra:
  - Các đối tượng tham gia tương tác.
  - Đường liên kết giữa các đối tượng.
  - Thông điệp trao chuyển giữa các đối tượng.

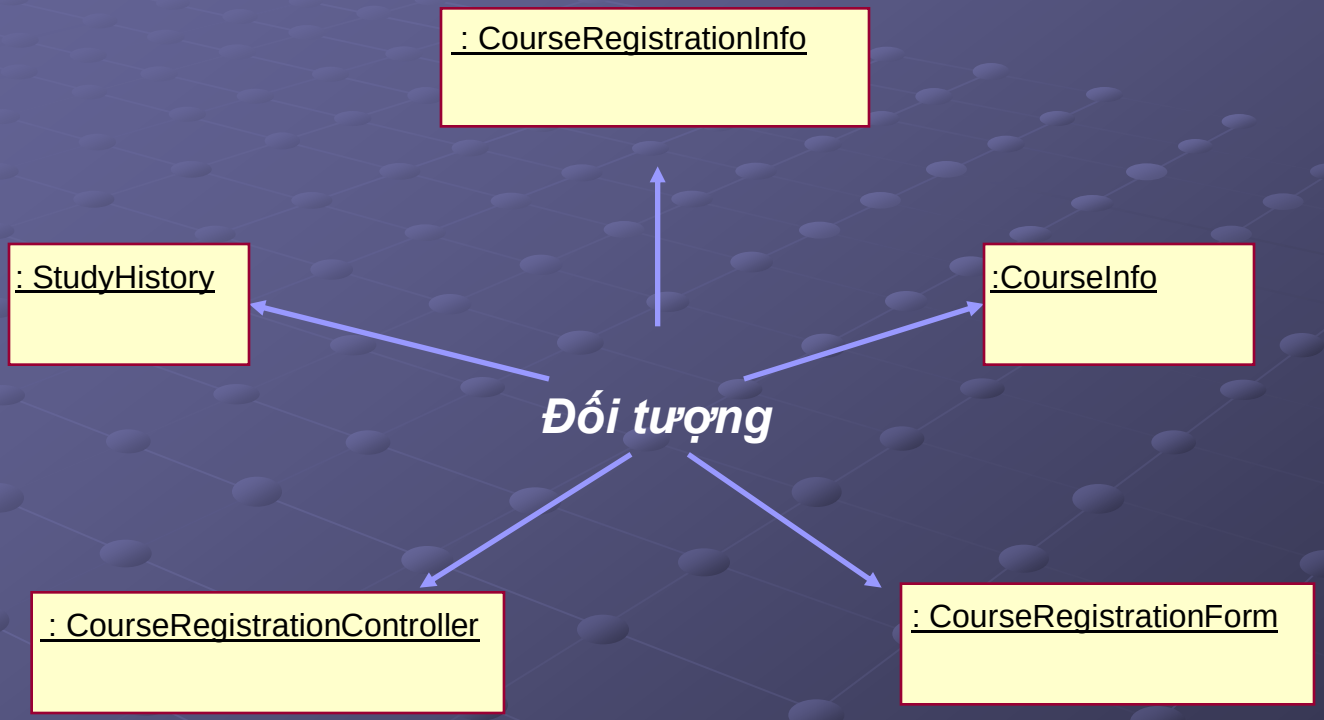


Communication Diagrams

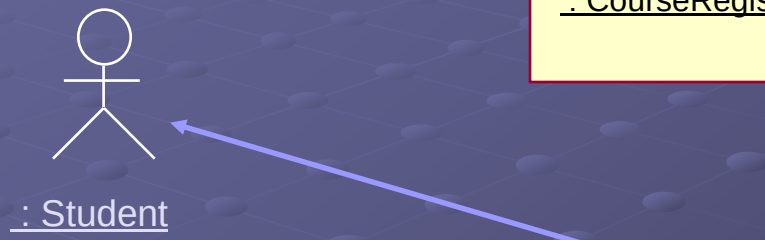
Biểu đồ giao tiếp/cộng tác



Biểu đồ giao tiếp – Đối tượng



Biểu đồ giao tiếp – Tác nhân



: Student

Actors

: CourseRegistrationInfo

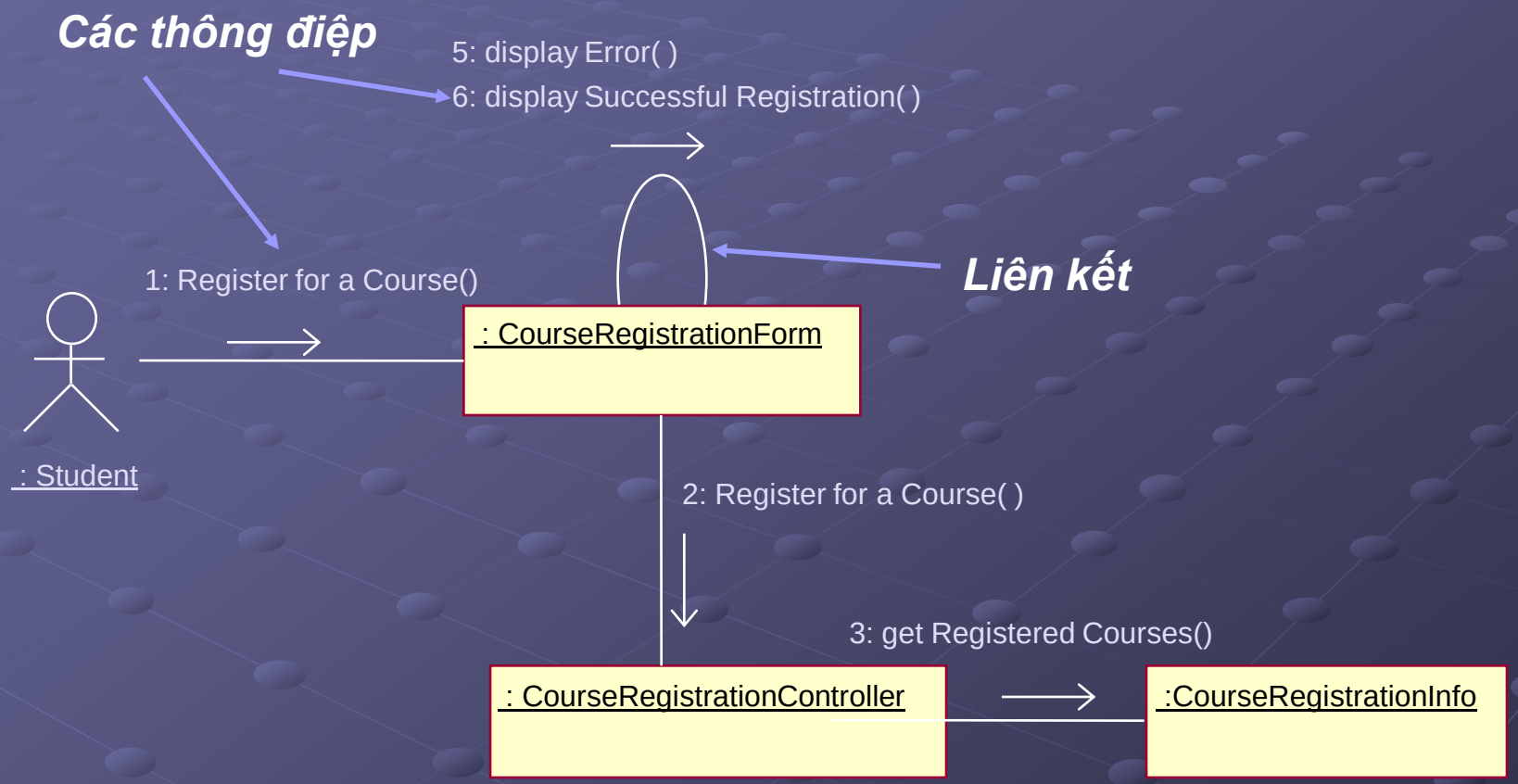
: StudyHistory

:CourseInfo

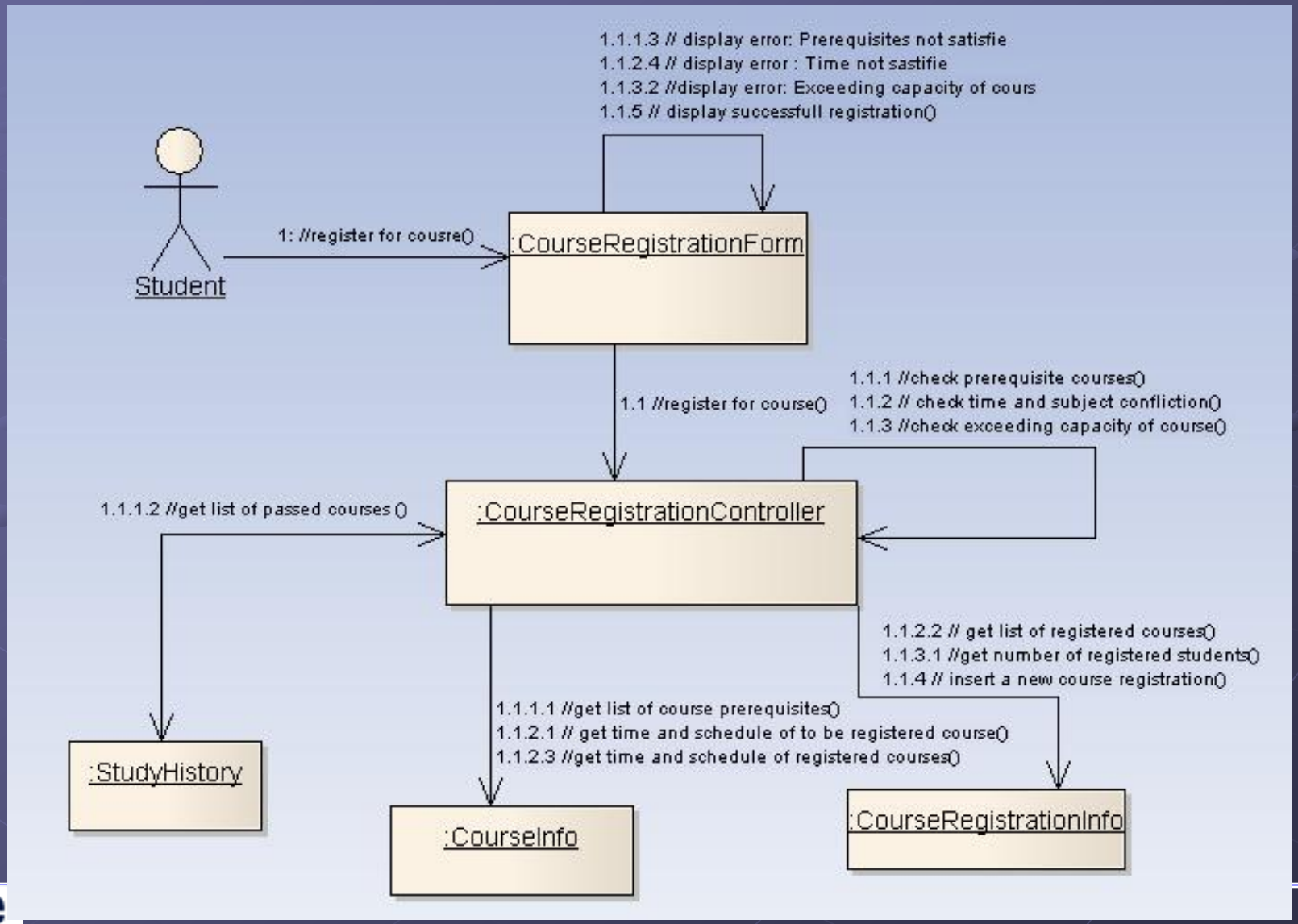
: CourseRegistrationForm

: CourseRegistrationController

Biểu đồ giao tiếp – Liên kết và thông điệp



UC Register for course: Biểu đồ giao tiếp



SD & CD: Giống nhau

- Tương đương nhau về mặt ngữ nghĩa
  - Có thể chuyển đổi biểu đồ này sang biểu đồ kia mà không mất mát bất cứ thông tin nào.
- Mô hình hóa các khía cạnh động của một hệ thống.
- Mô hình hóa một kịch bản use case.

Biểu đồ giao tiếp và biểu đồ trình tự

Biểu đồ trình tự

- Thể hiện rõ trình tự của quá trình tương tác
- Thể hiện tốt hơn luồng công việc
- Thể hiện tốt hơn quá trình mô tả các luồng sự kiện phức tạp trên phương diện thời gian thực

Biểu đồ giao tiếp

- Thể hiện mối quan hệ rõ ràng trong quá trình tương tác
- Thể hiện tốt hơn quá trình c ng tác
- Thể hiện rõ hơn hiệu quả của quá trình tương tác trên từng đối tượng.

Nội dung

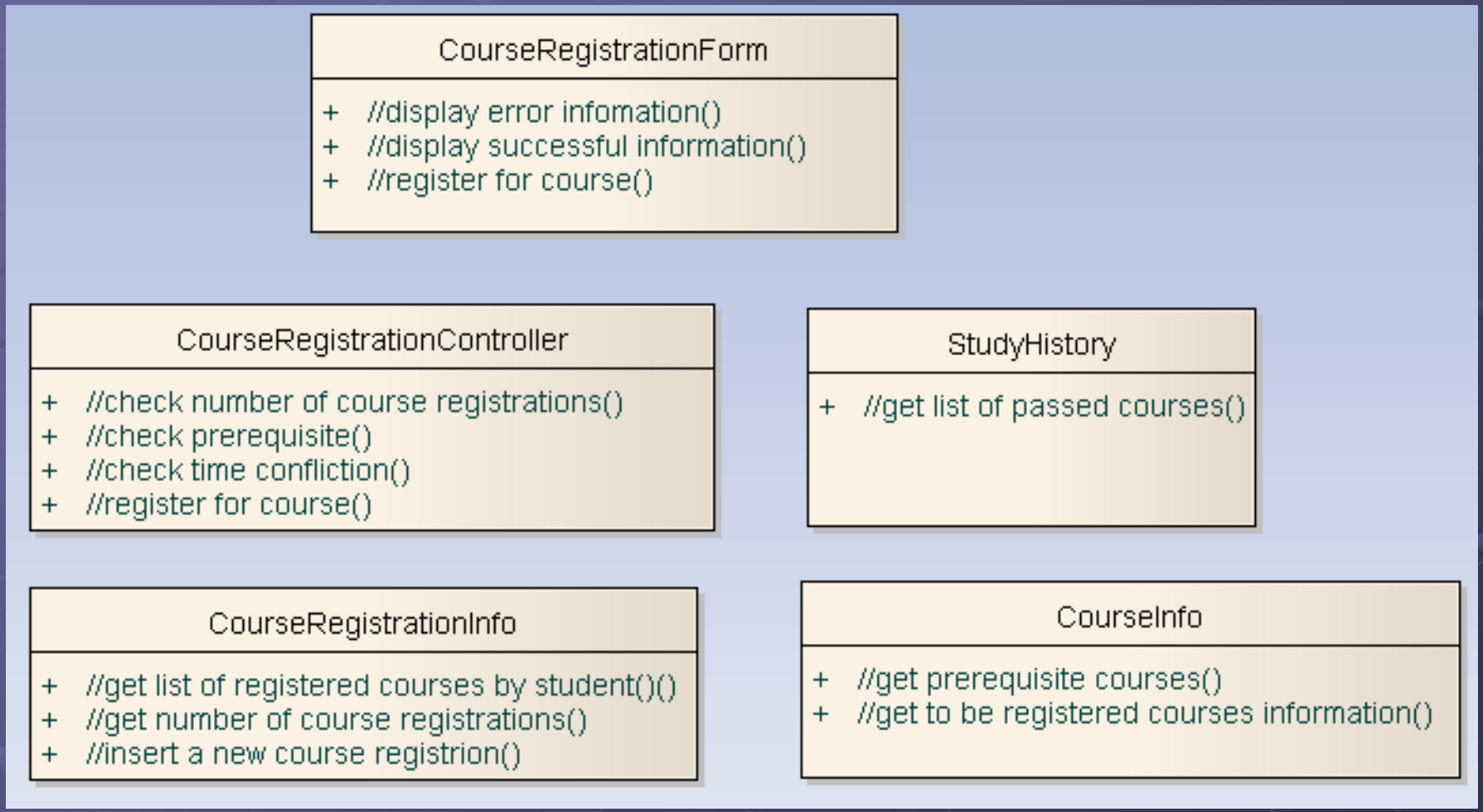
1. Tổng quan về phân tích use case
2. c lớp phân tích
3. Các biểu đồ tương tác
4. ch

4.1. Mô tả nhiệm vụ

- Biểu đồ tương tác

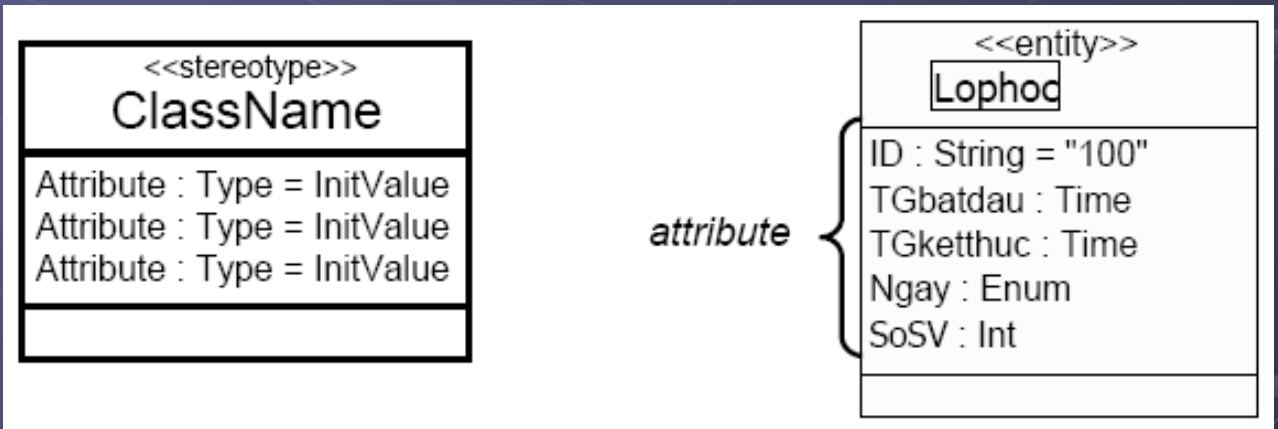
- Sơ đồ lớp

UC Register for Course: Biểu đồ lớp chi tiết

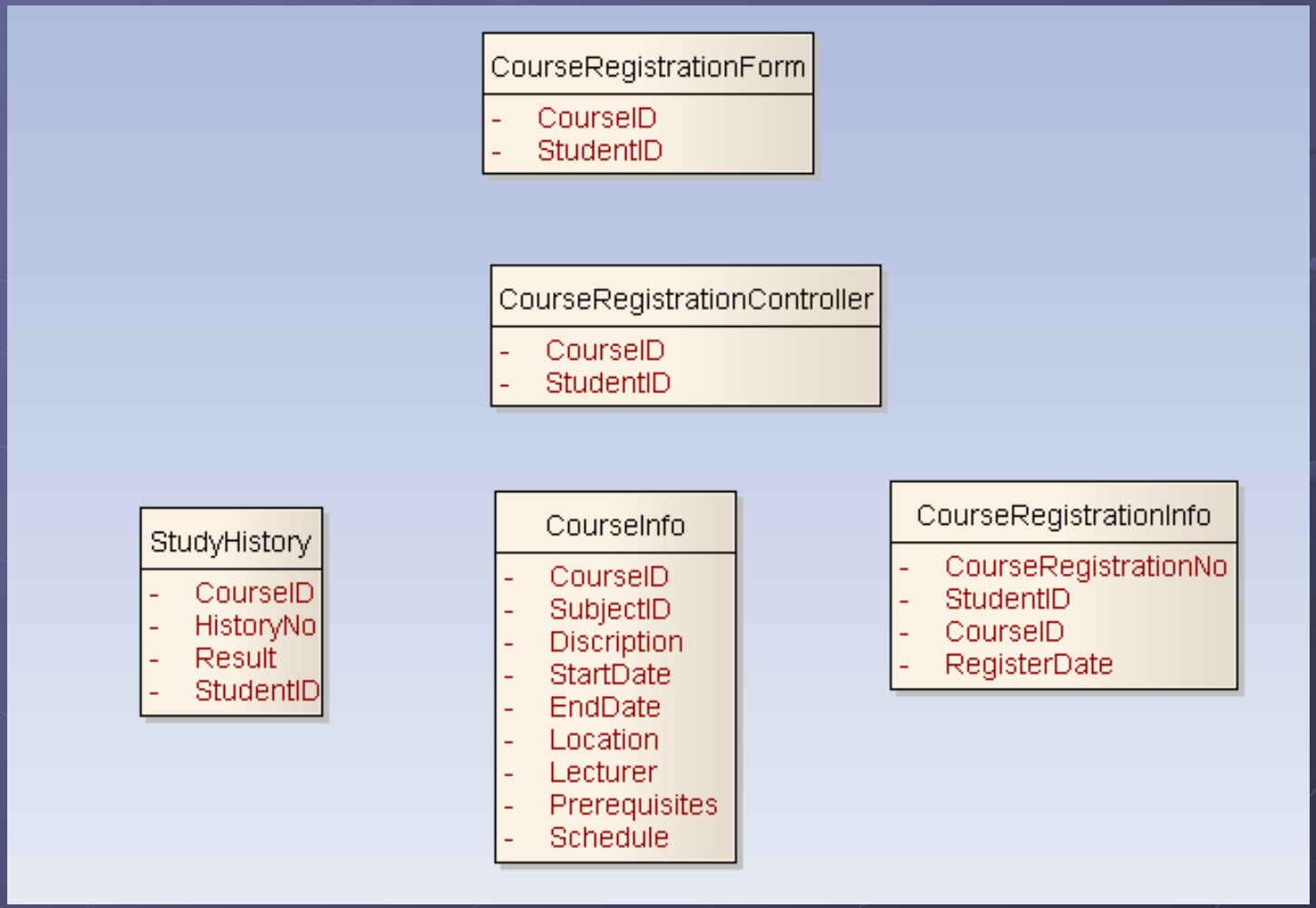


4.2. Tìm kiếm thuộc tính

- Thể hiện thuộc tính, đặc tính của một lớp
- Thông tin cần thiết để để lớp có thể thực thi các trách nhiệm (chức năng, hàm) của nó
- Chú ý đến các “danh từ” mà không đủ để trở thành lớp trong quá trình tìm kiếm lớp phân tích

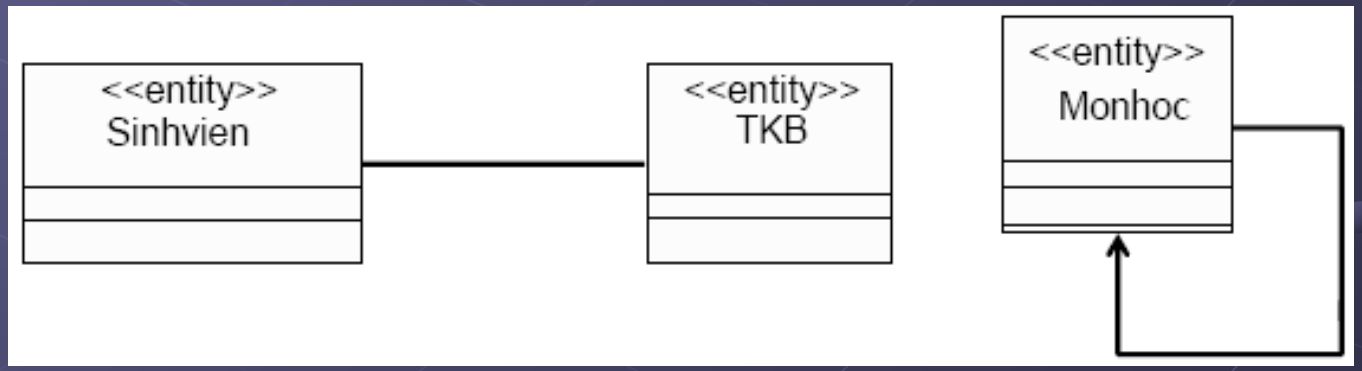


Ví dụ cho UC Register for Course

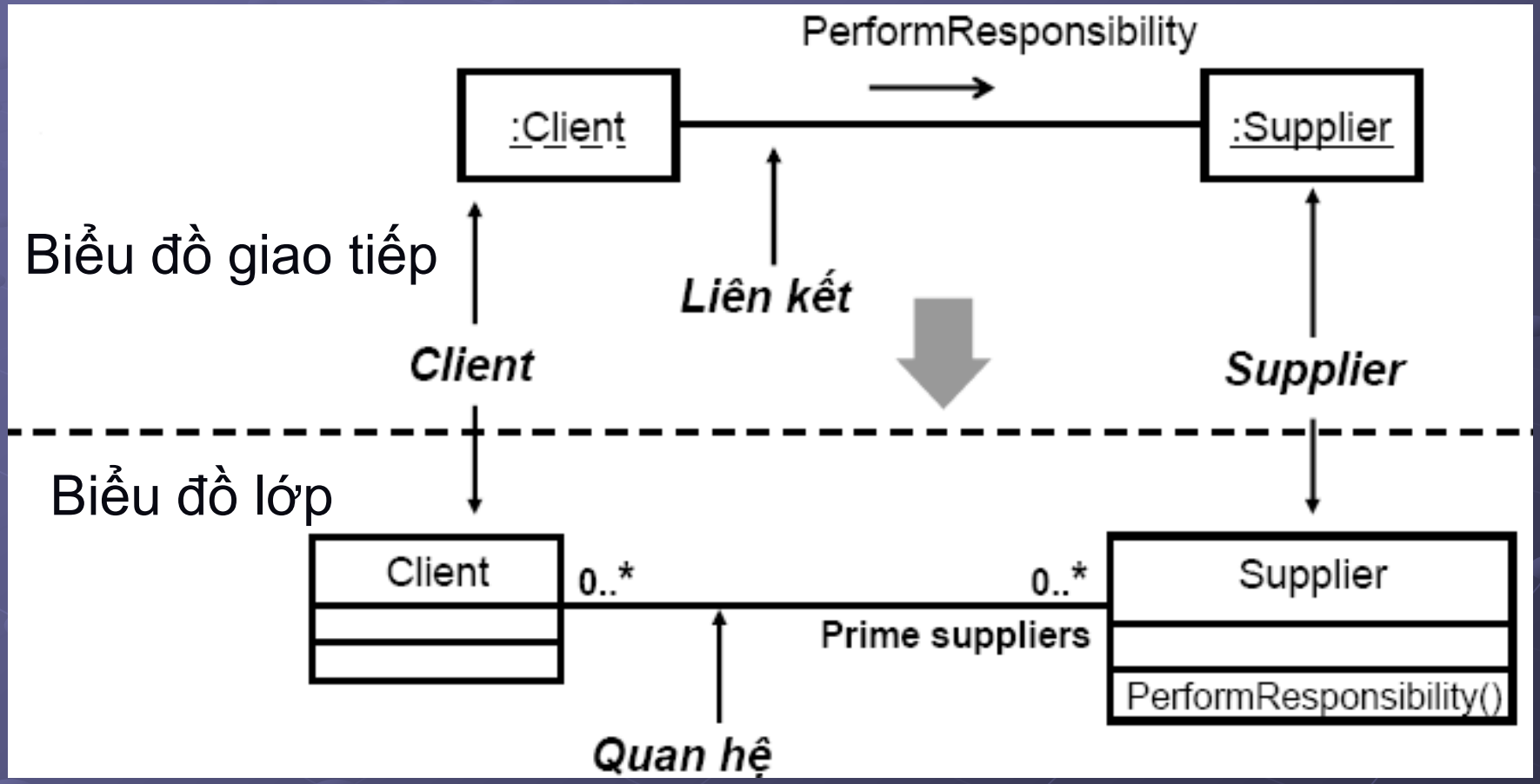


4.3. Quan hệ giữa các lớp

- Quan hệ giữa các lớp chỉ ra rằng đối tượng của lớp này có thể gửi thông điệp đến đối tượng của lớp kia
- Quan hệ có thể có 1 chiều hoặc 2 chiều
- Trong UML thể hiện bằng đường vẽ không mũi tên hoặc có mũi tên



Tìm kiếm quan hệ



Thể hiện mối quan hệ trong tất cả các liên kết

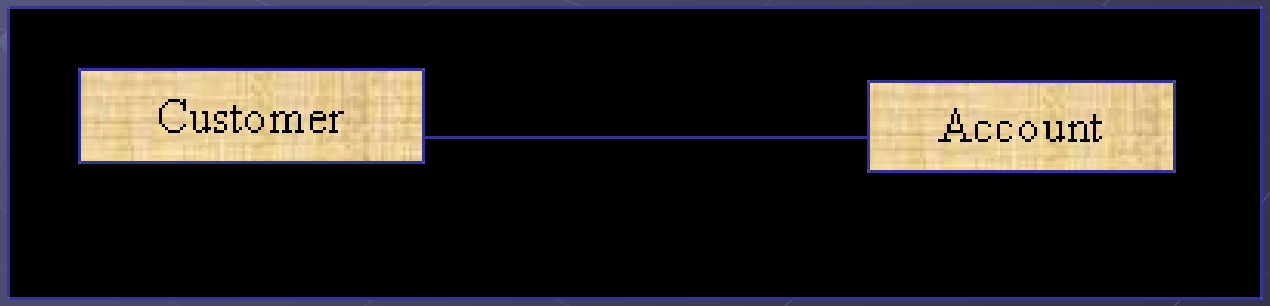
Kiểu quan hệ

- Một số kiểu quan hệ
  - Liên kết (association)
  - Kết tập (aggregation/composition)
  - Phụ thuộc (dependencies)
  - Tổng quát hoá (generalization)

Liên kết (Association)

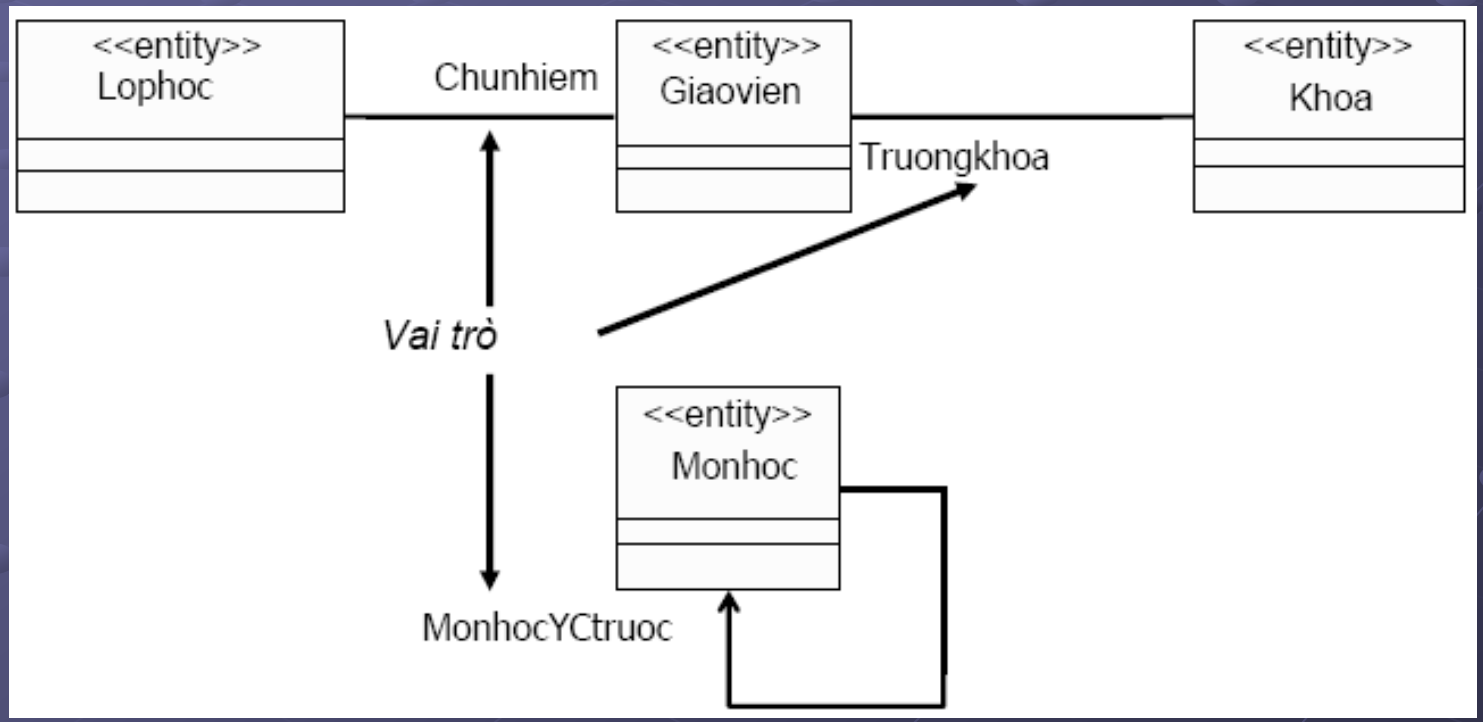
- Đây là hình thức hai lớp, đối tượng quan hệ với nhau theo hình thức Liên kết

- Một Liên kết có thể có các vai trò (**Roles**)



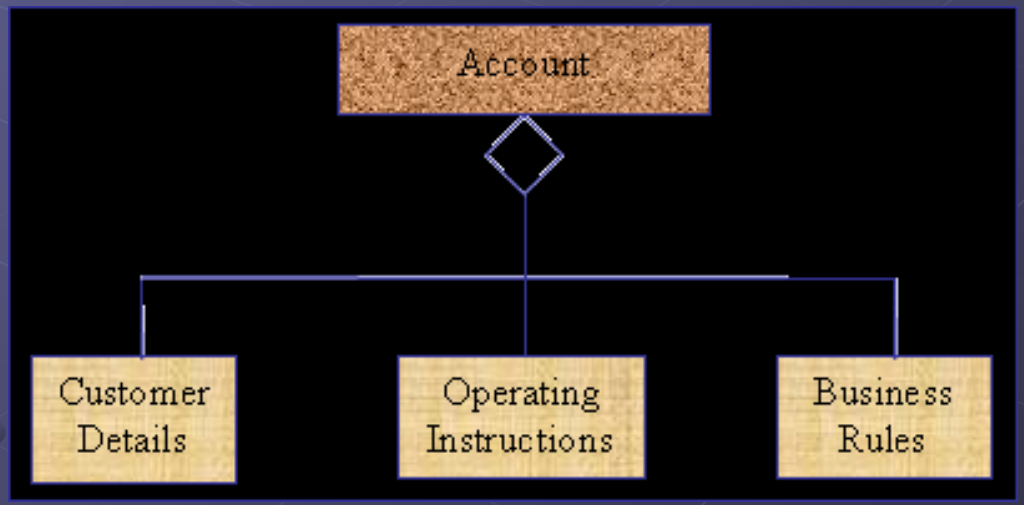
Vai trò của lớp trong mối quan hệ

- Thể hiện rõ vai trò của một lớp trong mối quan hệ đó



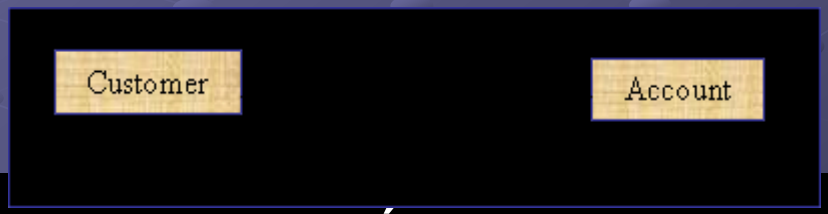
Kết tập/Thành phần

- Là một hình thức mạnh của Liên kết
- Đây là quan hệ mang tính thành phần, lớp thành phần sẽ bị mất đi nếu như lớp chứa của nó mất đi



Liên kết hay Kết tập

- Nếu 2 đối tượng thường được xem xét độc lập, mặc dù chúng có quan hệ với nhau
  - Mối quan hệ là một Liên kết



- Nếu 2 đối tượng có mối quan hệ toàn thể và thành phần
  - Mối quan hệ là một Kết tập/thành phần

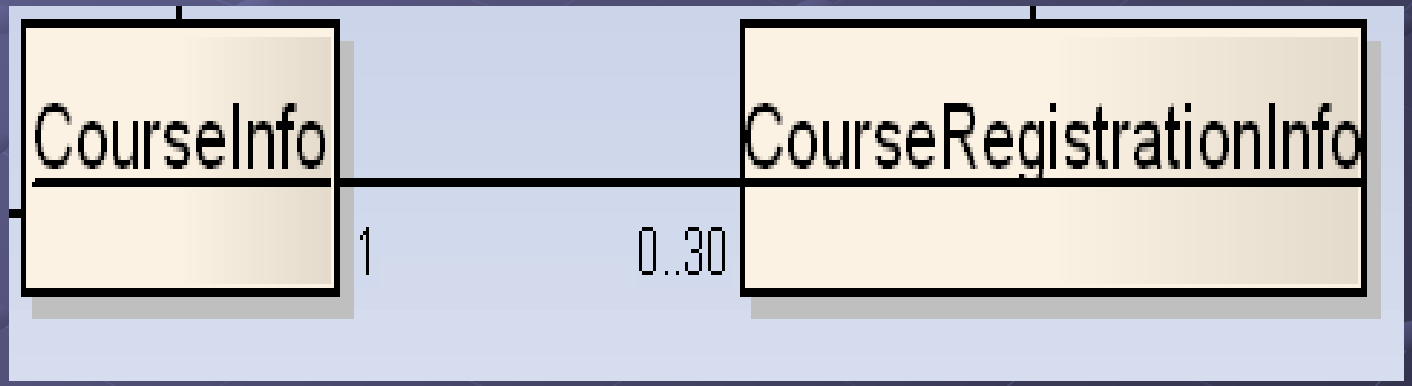
Bội số của quan hệ (Multiplicity)

- Bội số cho phép chỉ ra số lượng của 1 đối tượng cần thiết để quan hệ với số lượng của 1 đối tượng khác

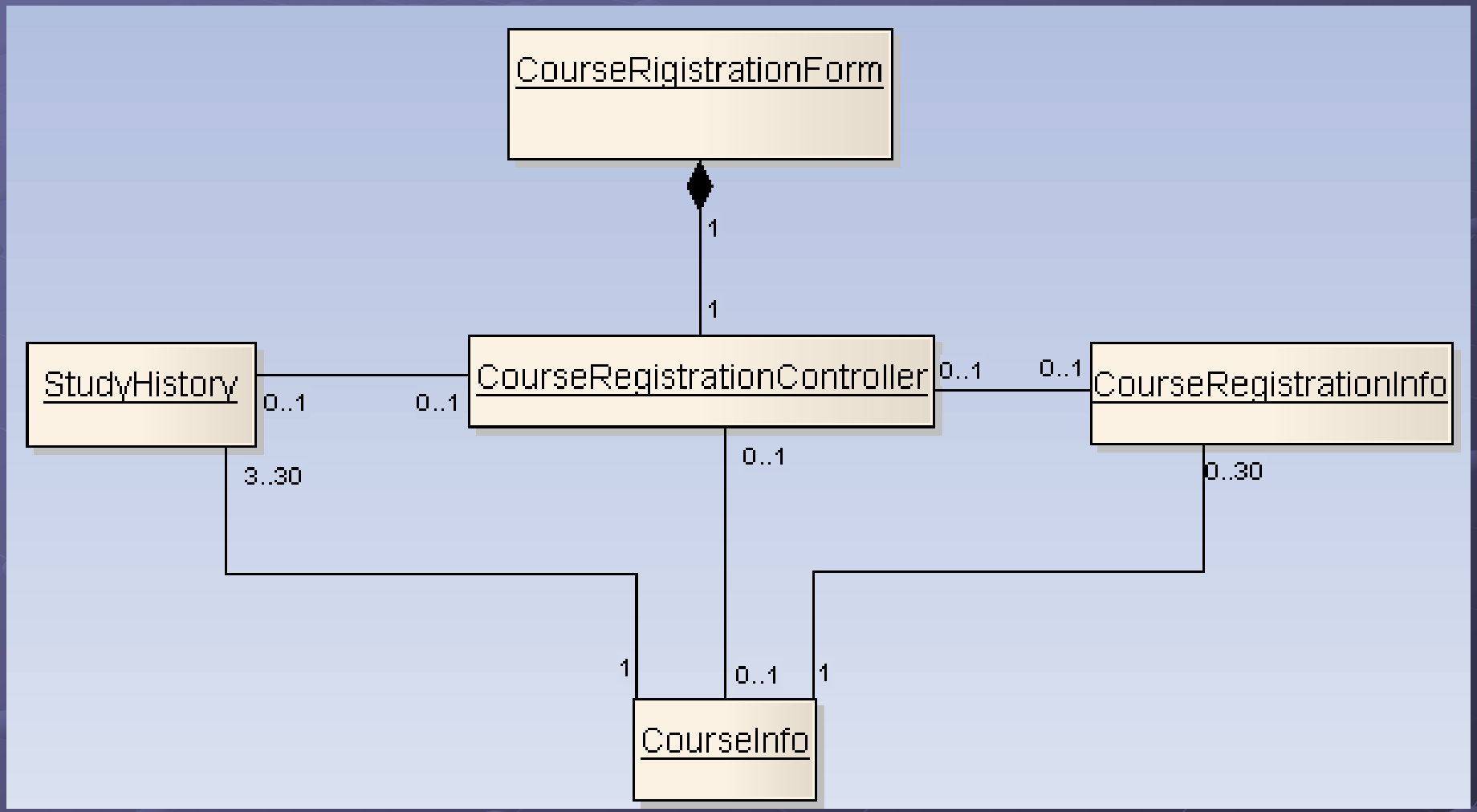
Không cụ thể	
Một	1
Không hoặc nhiều	0..*
Không hoặc nhiều	*
Một hoặc nhiều	1..*
Không hoặc một	0..1
Khoảng cách cụ thể	2..4
Khoảng cách cụ thể	2, 4..6

Bội số của quan hệ

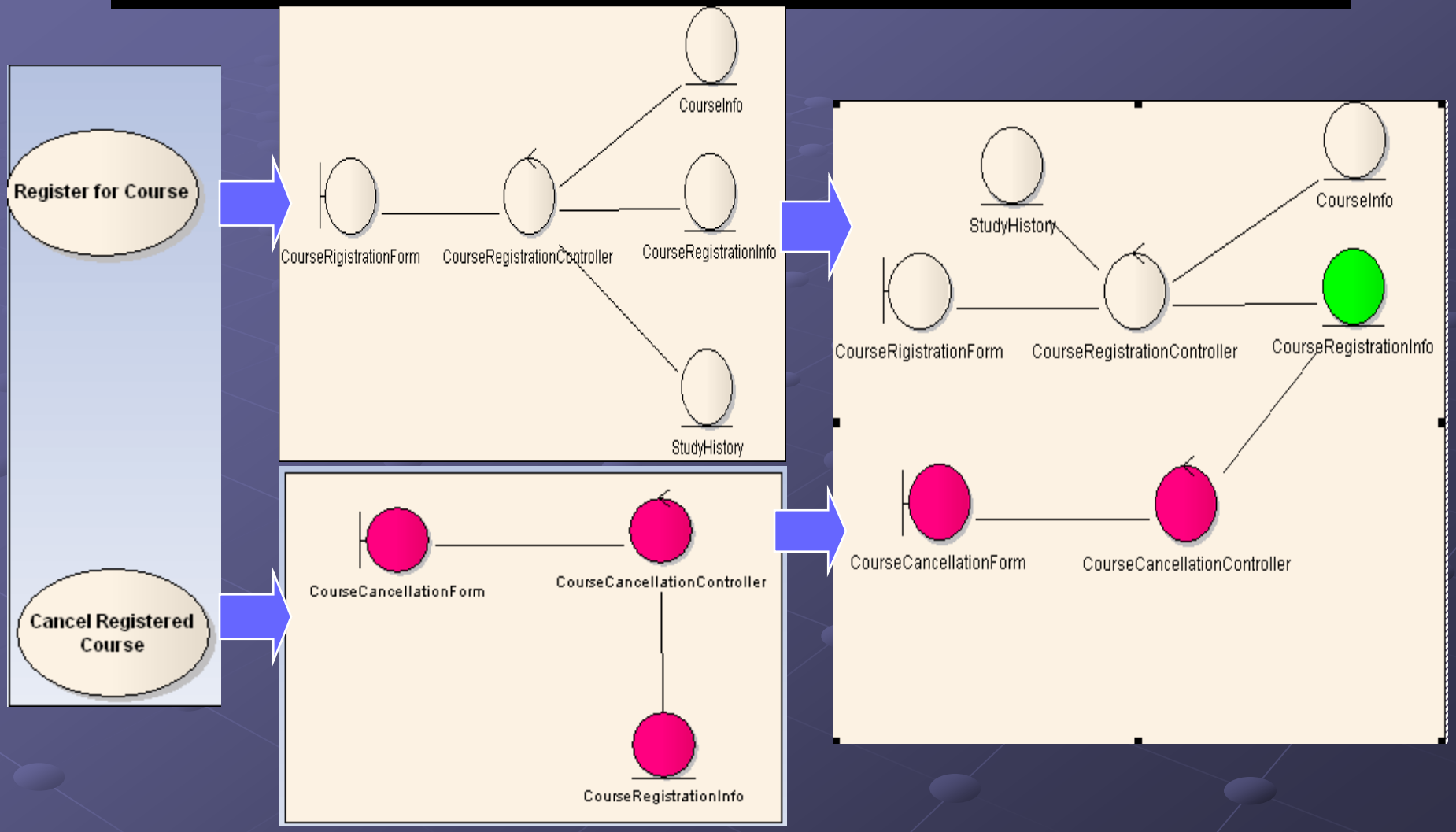
- Bội số của quan hệ trả lời 2 câu hỏi
  - Sự liên kết là bắt buộc hay tuỳ chọn
  - Số lượng nhỏ nhất và lớn nhất các đối tượng có thể kết nối với các đối tượng của lớp khác



UC Register for course: Quan hệ trong sơ đồ lớp



Hợp nhất các lớp phân tích



Ví dụ về UC Mua hàng trên mạng

- Mô tả:
  - Giả sử có một hệ thống của hàng ảo trên mạng
  - UC Bán hàng cho phép khách hàng (KH) mua được các mặt hàng mong muốn
  - Ví dụ này yêu cầu KH phải thành toán trực tuyến
- Tiền điều kiện:
  - KH muốn mua hàng trên cửa hàng ảo
  - KH có thể thanh toán điện tử tới ngân hàng mà cửa hàng hỗ trợ
- Hậu điều kiện:
  - Thành công khi KH chấp nhận mua hàng và quá trình thanh toán với ngân hàng thực hiện thành công. Hóa đơn được lập, hàng hóa được dành riêng cho KH đó
  - Nếu quá trình thanh toán với ngân hàng không thành công, hóa đơn sẽ không được lập, hàng cũng không được bán ra
- Thực thể:
  - Mặt hàng, Giỏ hàng, Đơn hàng
- Use case liên quan:
  - Tìm kiếm hàng, quản lý đơn hàng (Giao hàng)

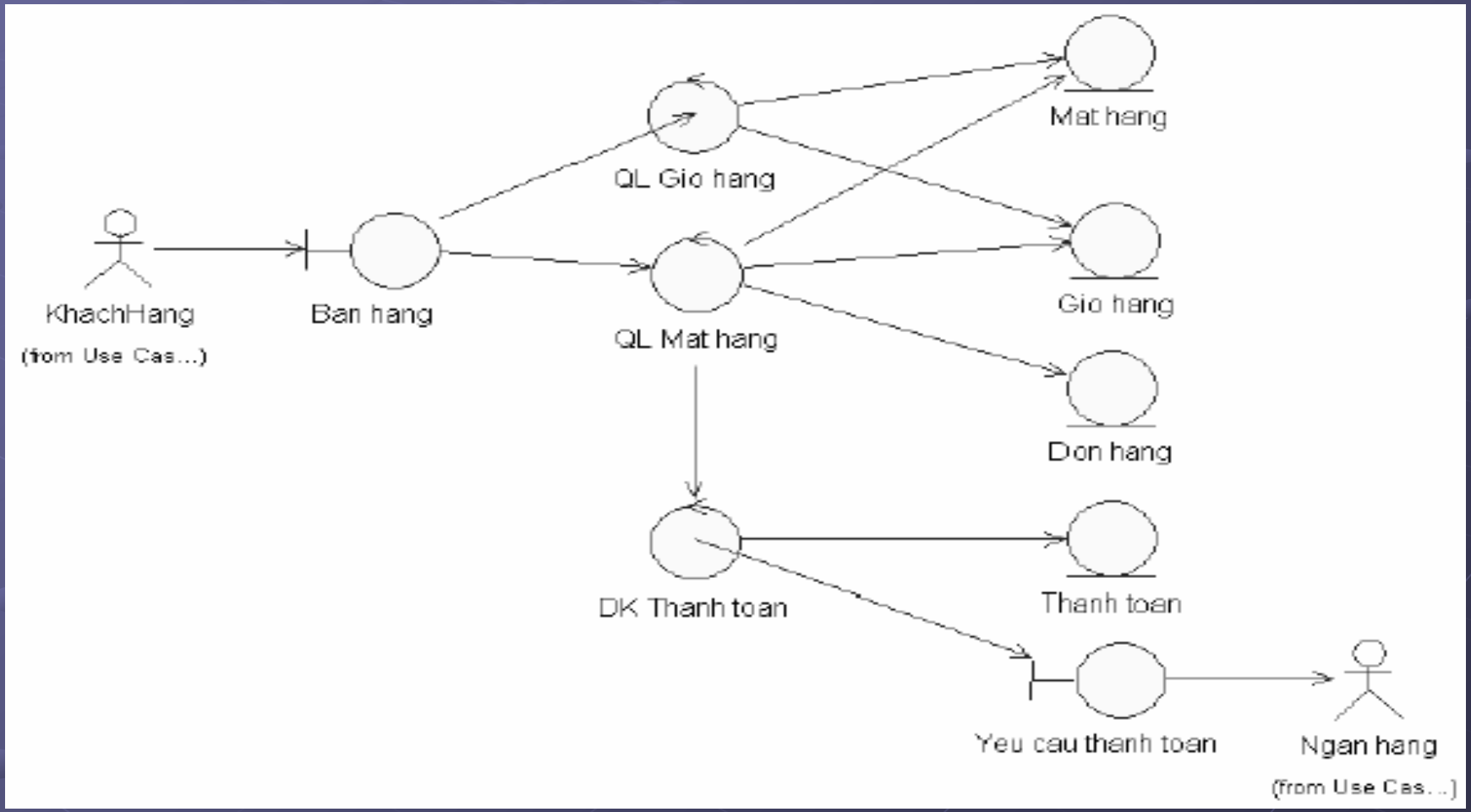
Luồng sự kiện cho Use Case

1. KH duyệt, tìm kiếm và xem thông tin các mặt hàng muốn mua (xem UC xem hàng)
    1. KH có thể chọn chức năng “Đưa hàng vào giỏ hàng”
    2. Hệ thống sẽ đưa mặt hàng này vào giỏ
    3. KH có thể nhập số lượng muốn mua (mặc định là 1)
    4. Hệ thống sẽ tự động cập nhật giá của giỏ hàng hiện tại
2. KH có thể lặp lại quá trình này để mua tiếp các mặt hàng khác
    1. 1. Giỏ hàng sẽ không mất đi trong quá trình KH tìm/mua mặt hàng khác
    2. 2. Nếu giỏ hàng đã có mặt hàng này, hệ thống sẽ báo lại cho KH…
3. Quản lý giỏ hàng
    1. Mỗi một KH có một giỏ hàng riêng rẽ và không ai nhìn thấy thông tin của nhau
    2. KH có thể chọn chức năng “Xem giỏ hàng” bất kỳ lúc nào cần
    3. Hệ thống sẽ hiển thị giỏ hàng với đầy đủ các mặt hàng KH đã chọn, cùng số lượng và giá cả từng loại
    4. KH có thể thay đổi số lượng, hoặc bỏ đi mặt hàng mà KH không muốn mua
4. KH có thể chọn chức năng thành toán, xem luồng phụ “Thanh toán”

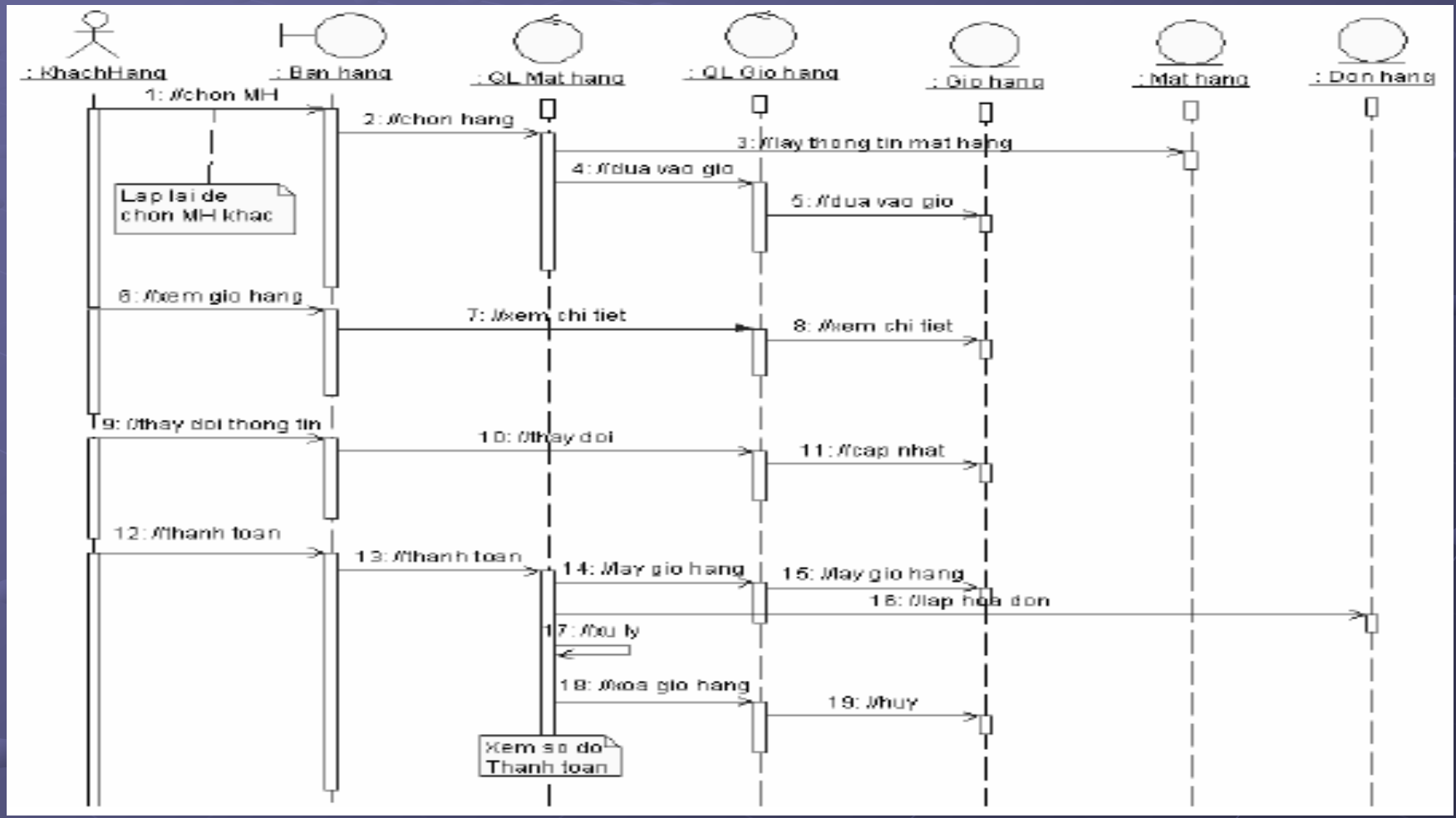
Luồng phụ: Thanh toán

1. KH có thể chọn chức năng thanh toán
2. KH được yêu cầu nhập thẻ thanh toán và địa chỉ giao hàng
3. Thông tin thanh toán được đưa tới ngân hàng, hệ thống sẽ chờ kết quả từ ngân hàng đó
    1. (Quá trình xử lý giao dịch là do ngân hàng quyết định)
4. Nếu ngân hàng không chập nhận giao dịch
    1. Hệ thống sẽ thông báo kết quả tới KH, yêu cầu nhập lại thông tin
5. Nếu ngân hàng chấp nhận
    1. (Số tiền tương ứng của KH được chuyển sang tài khoản của cửa hàng)
    2. Hệ thống sẽ lập Đơn hàng và lưu lại (xem UC quản lý đơn hàng)
    3. Số lượng hàng tồn kho sẽ được giảm tương ứng
    4. Hệ thống thông báo thành công cho KH trên trang web và gửi thông tin đơn hàng qua mail của KH
    5. Giỏ hàng sẽ bị xóa đi (nếu mua tiếp, giỏ hàng sẽ được tạo mới)

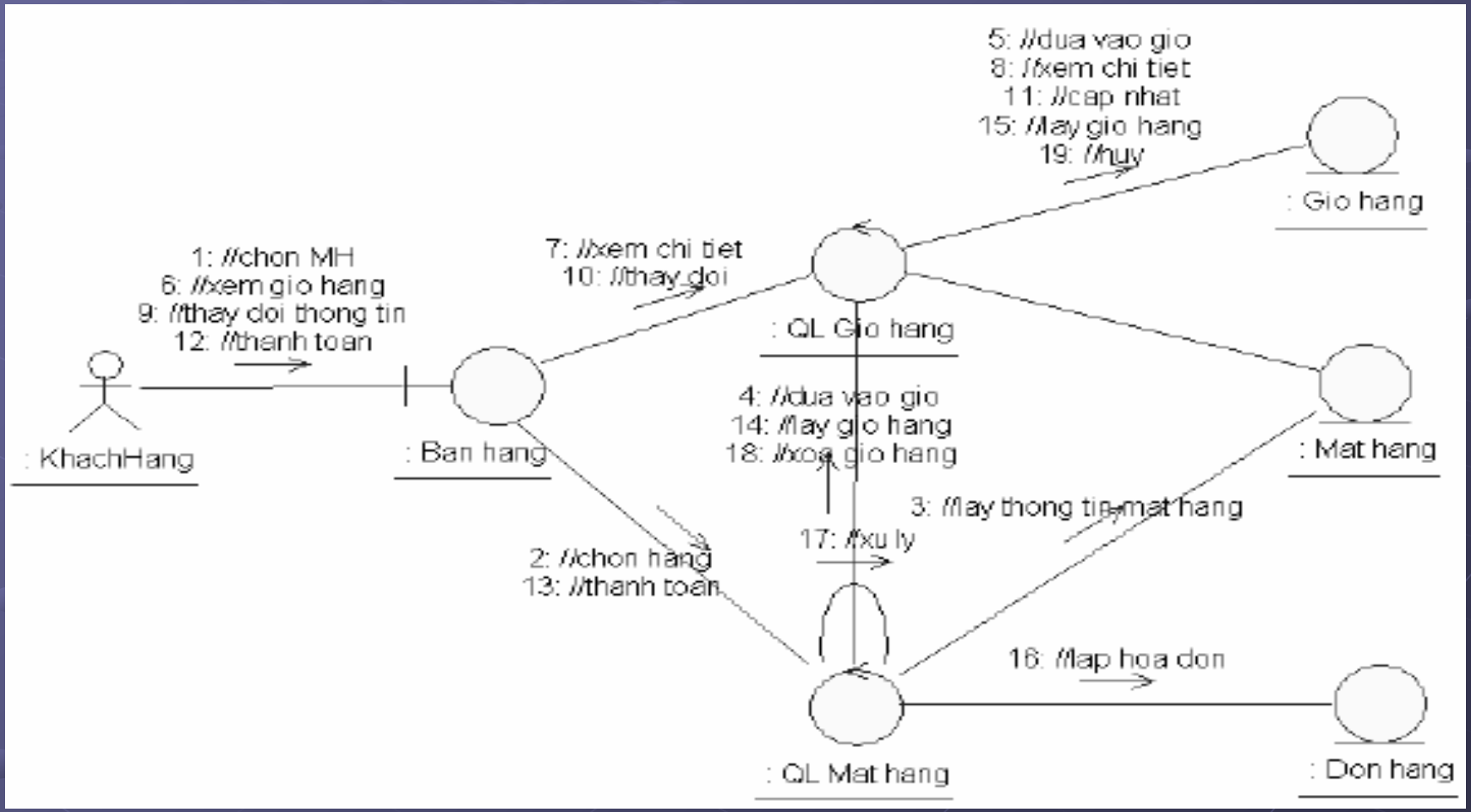
Biểu đồ lớp phân tích



Biểu đồ trình tự



Biểu đồ giao tiếp



Tổng kết

- Quy trình phân tích Use Case
  - Vai trò của các mô hình phân tích
  - Sự phát triển của mô hình phân tích tới mô hình thiết kế
- Biểu đồ lớp phân tích (Analysis Class Diagram)
  - Tìm kiếm lớp phân tích
  - Biểu đồ lớp phân tích
- Các biểu đồ tương tác (Interaction Diagram)
  - Biểu đồ trình tự, giao tiếp

- Biểu đồ tương tác:
  - Trình tự
  - Giao tiếp
- Biểu đồ lớp phân tích
- Biểu đồ lớp phân tích hợp nhất của các use case liên quan đến Student

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 6. Xác định các phần tử thiết kế

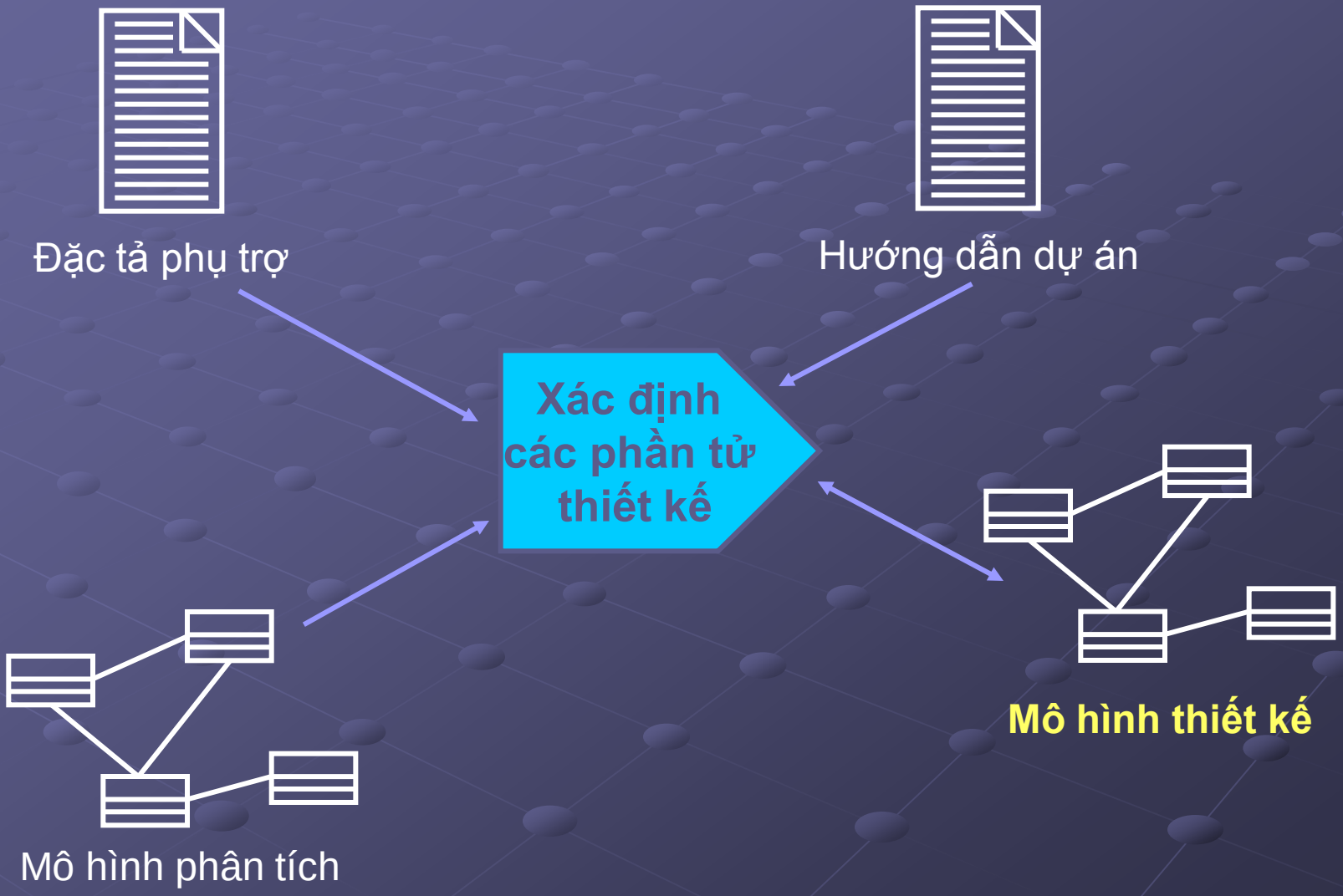
Nội dung

1. Xác định các phần tử thiết kế
2. Hệ thống con (Subsystem)
3. Tính tái sử dụng lại (Reusability)
4. Mô hình phân tầng trong quá trình thiết kế

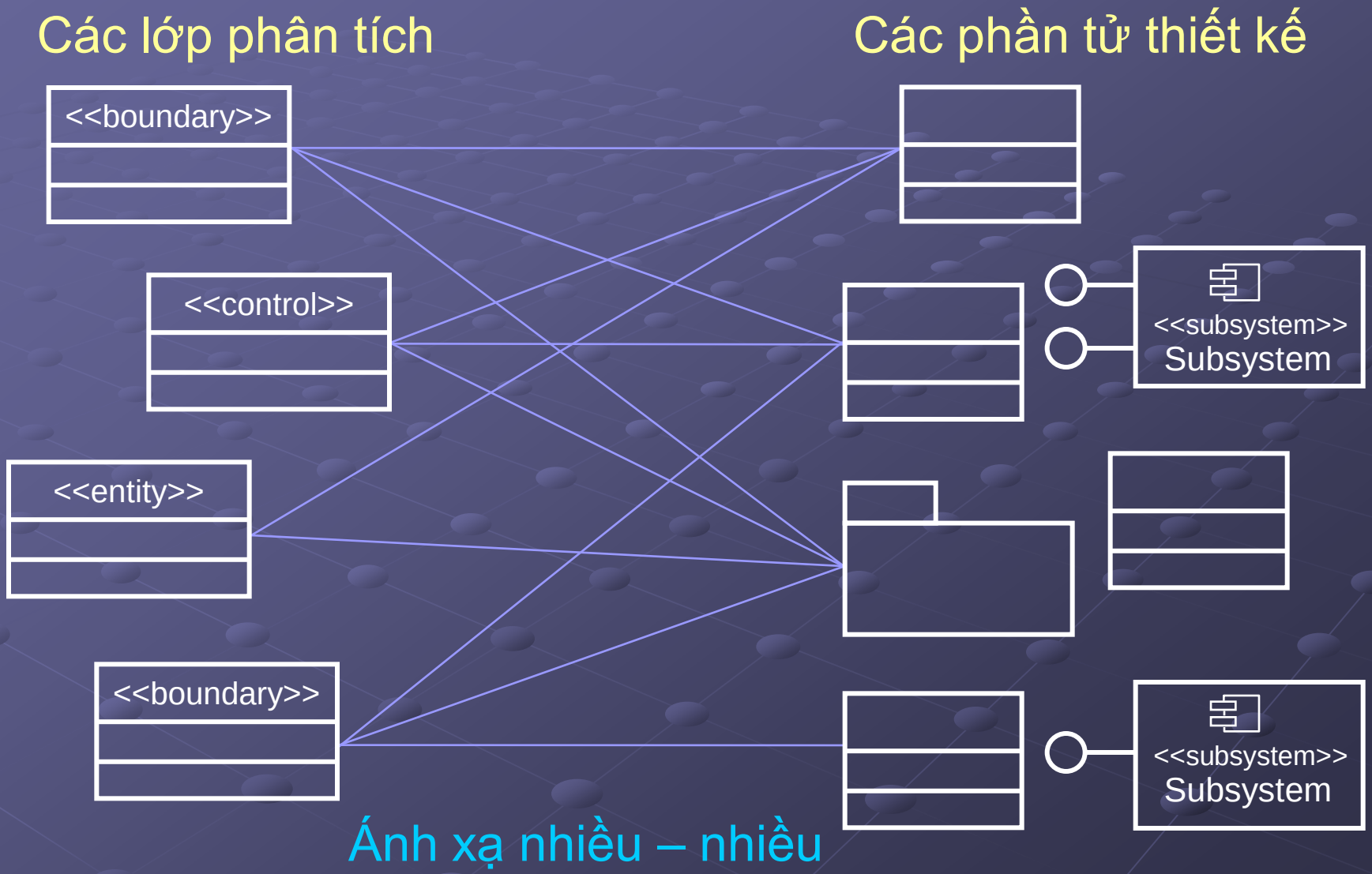
Mục đích

- Phân tích sự tương tác của các lớp phân tích và xác định các thành phần trong mô hình thiết kế
  - Lớp thiết kế (design class)
  - Hệ thống con (Subsystem)
  - Giao diện hệ thống con (Subsystem interface)

Xác định các phần tử thiết kế



Chuyển đổi lớp phân tích thành các phần tử thiết kế

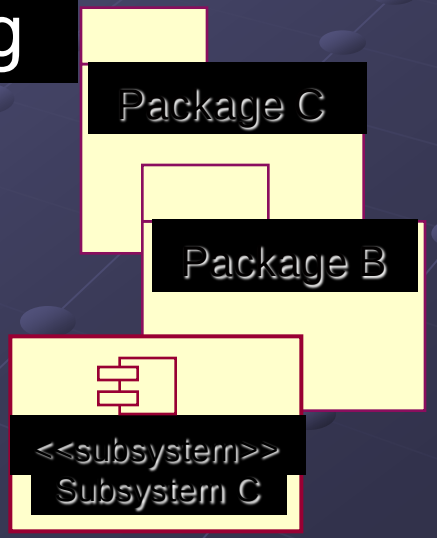


Tìm kiếm các lớp thiết kế

- Một lớp phân tích ánh xạ trực tiếp thành một lớp thiết kế nếu
  - Nó là một lớp đơn giản
  - Mức độ trừu tượng hóa đơn giản
- Các lớp phân tích phức tạp hơn có
  - Tách ra thành nhiều lớp
  - Trở thành một package
  - Trở thành một hệ thống con
  - Bất kỳ hình thức kết hợp nào

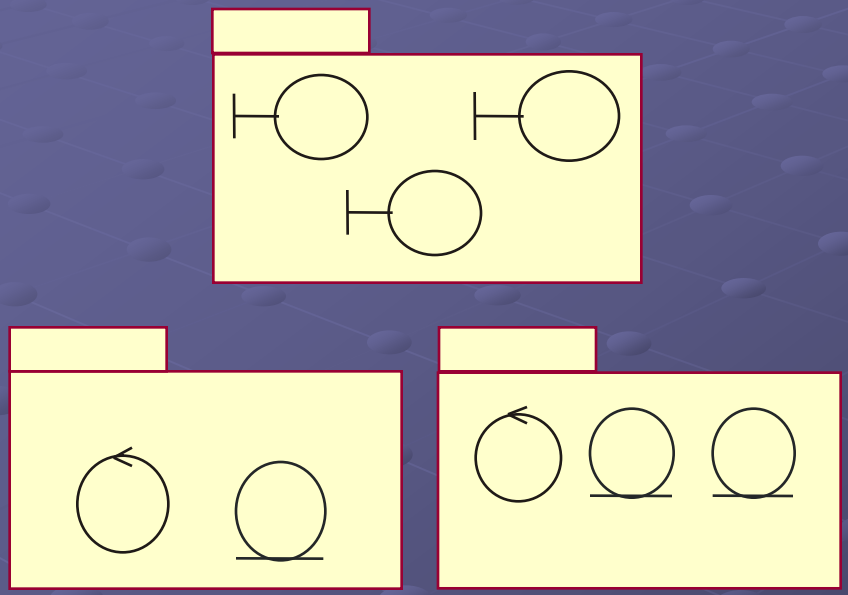
Nhóm các lớp thiết kế

- Nhóm các lớp dựa trên nhiều yếu tố:
  - Phân bổ nguồn lực trong các đội phát triển
  - Tương ứng với từng loại người dùng
  - Hệ thống con đại diện cho các sản phẩm và dịch vụ đã có mà hệ thống sử dụng
- Việc gộp nhóm hiệu quả giúp
  - Quản lý khả năng sử dụng lại
  - Bảo dưỡng hệ thống



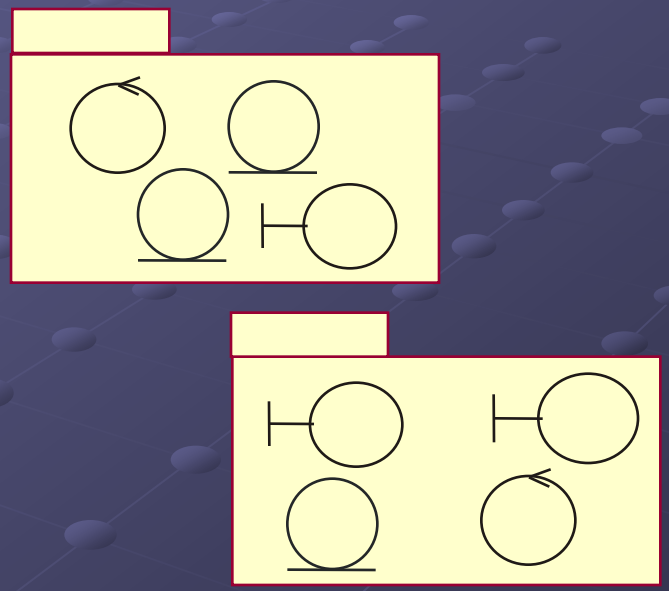
Nhóm các lớp biên

Nếu giao diện của hệ thống sẽ chắc chắn có các thay đổi đáng kể



Các lớp biên được đặt trong các package riêng biệt

Nếu giao diện của hệ thống không chắc chắn có các thay đổi đáng kể



Các lớp biên được gom nhóm với các lớp liên quan về mặt chức năng

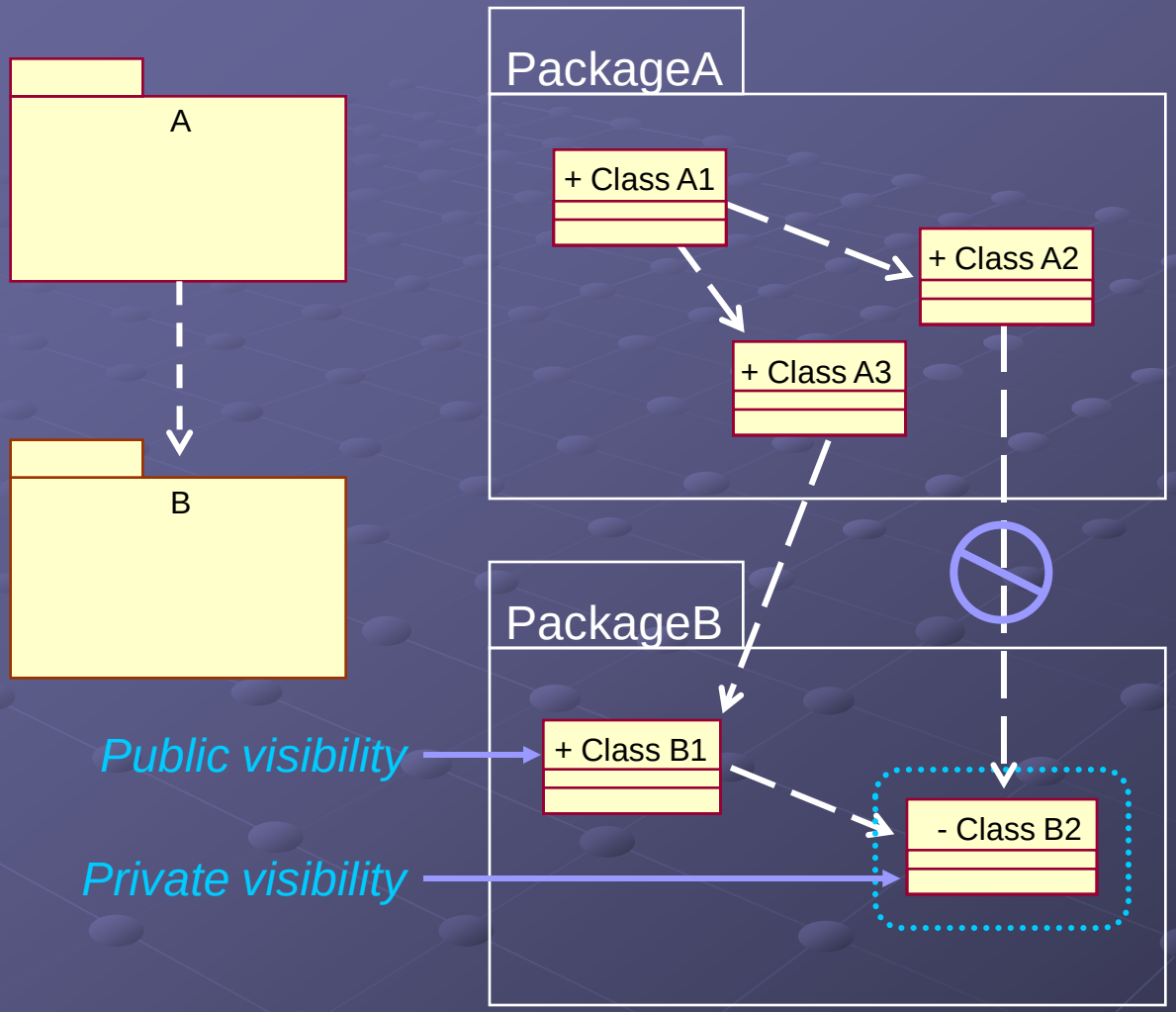
Nhóm các lớp liên quan về mặt chức năng

- Các tiêu chí – liên quan về mặt chức năng:
  - Thay đổi/xóa bỏ một lớp làm ảnh hưởng tới các lớp khác
  - Hai đối tượng tương tác với nhau bằng một lượng lớn thông điệp hoặc có mối giao tiếp phức tạp
  - Lớp biên có thể có liên quan về mặt chức năng đến một lớp thực thể nào đó nếu lớp biên biểu diễn lớp thực thể đó
  - Hai lớp tương tác hoặc cùng bị ảnh hưởng bởi thay đổi trong cùng một tác nhân
  - Một lớp tạo ra thể hiện của lớp khác

Nhóm các lớp liên quan về mặt chức năng

- Các tiêu chí – KHÔNG nên đặt hai lớp vào cùng một package:
  - Hai lớp liên quan đến các tác nhân khác nhau
  - Một lớp bắt buộc và một lớp không bắt buộc

Sự phụ thuộc package: Element Visibility



Chỉ có lớp public (+) có thể được tham chiếu bên ngoài package chứa nó

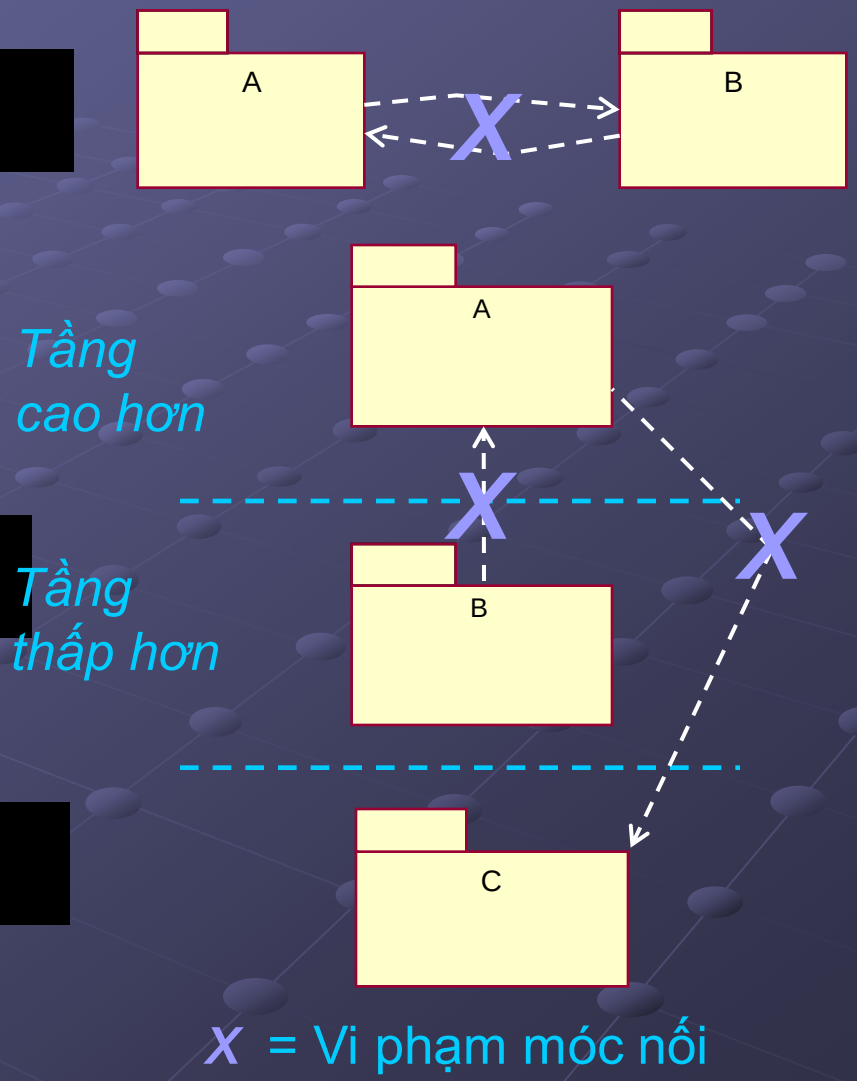
Lớp protected (#) chỉ được truy cập trong gói chứa nó và gói kế thừa gói chứa nó

Lớp private (-) chỉ được truy cập trong gói chứa nó

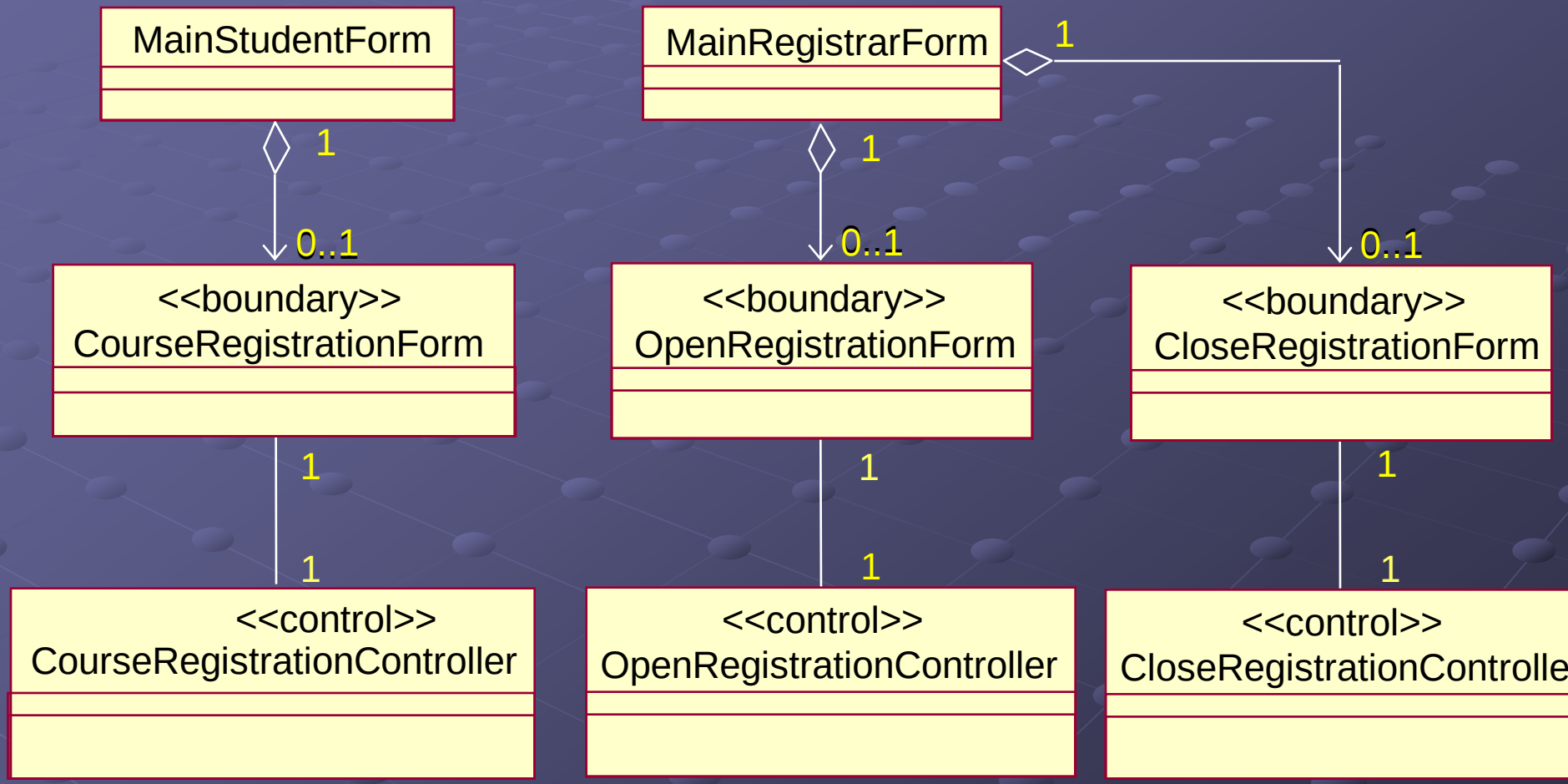
Quy tắc của OO: Đóng gói (Encapsulation)

Phụ thuộc giữa các package

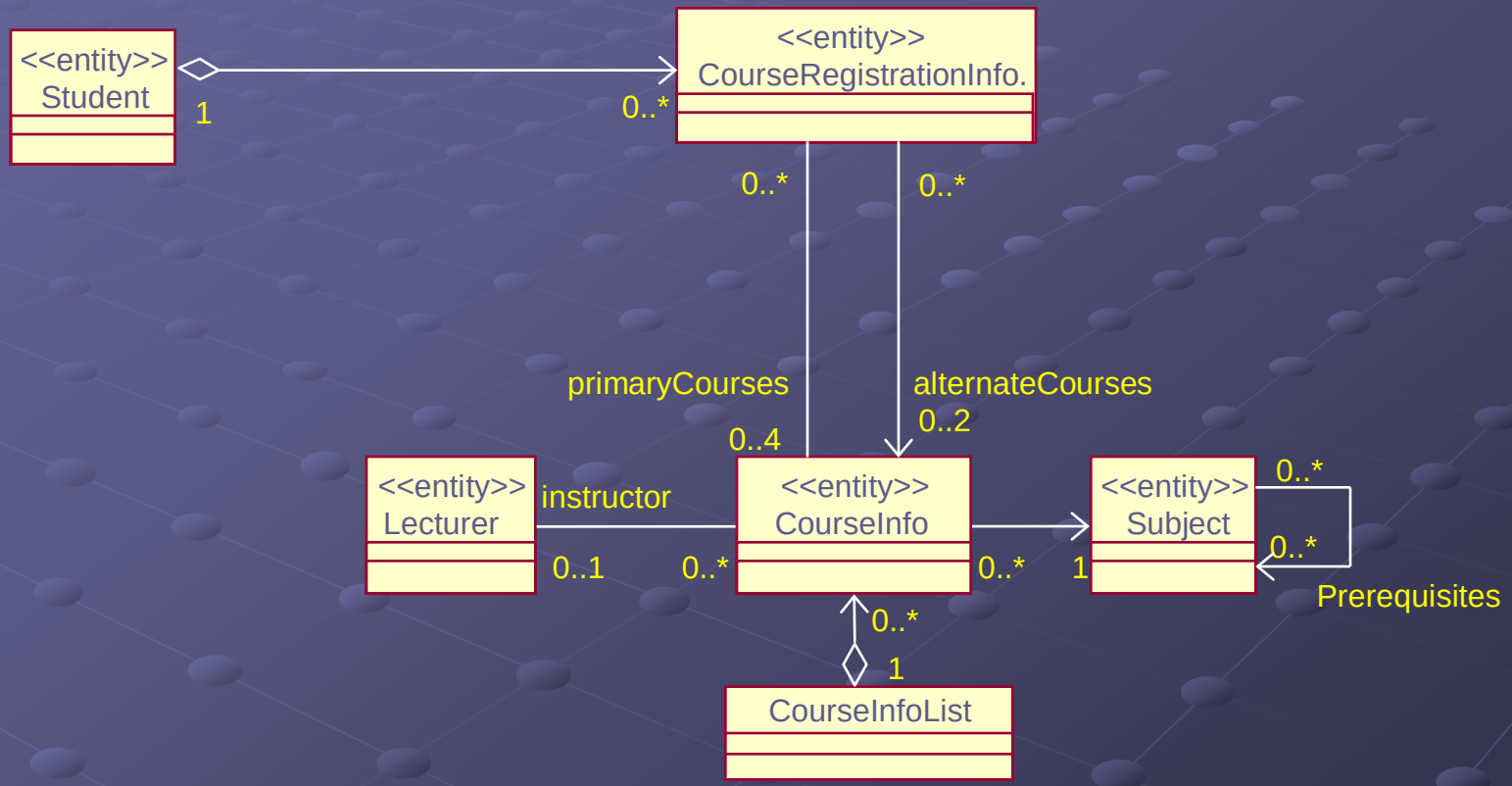
- Các package không nên phụ thuộc lẫn nhau (cross-coupling)
- Package ở tầng thấp hơn không nên phụ thuộc vào các package ở tầng trên
- Nhìn chung, các phụ thuộc không nên bỏ qua các tầng ở giữa



Ví dụ: Registration Package



Ví dụ: University Artifacts Package

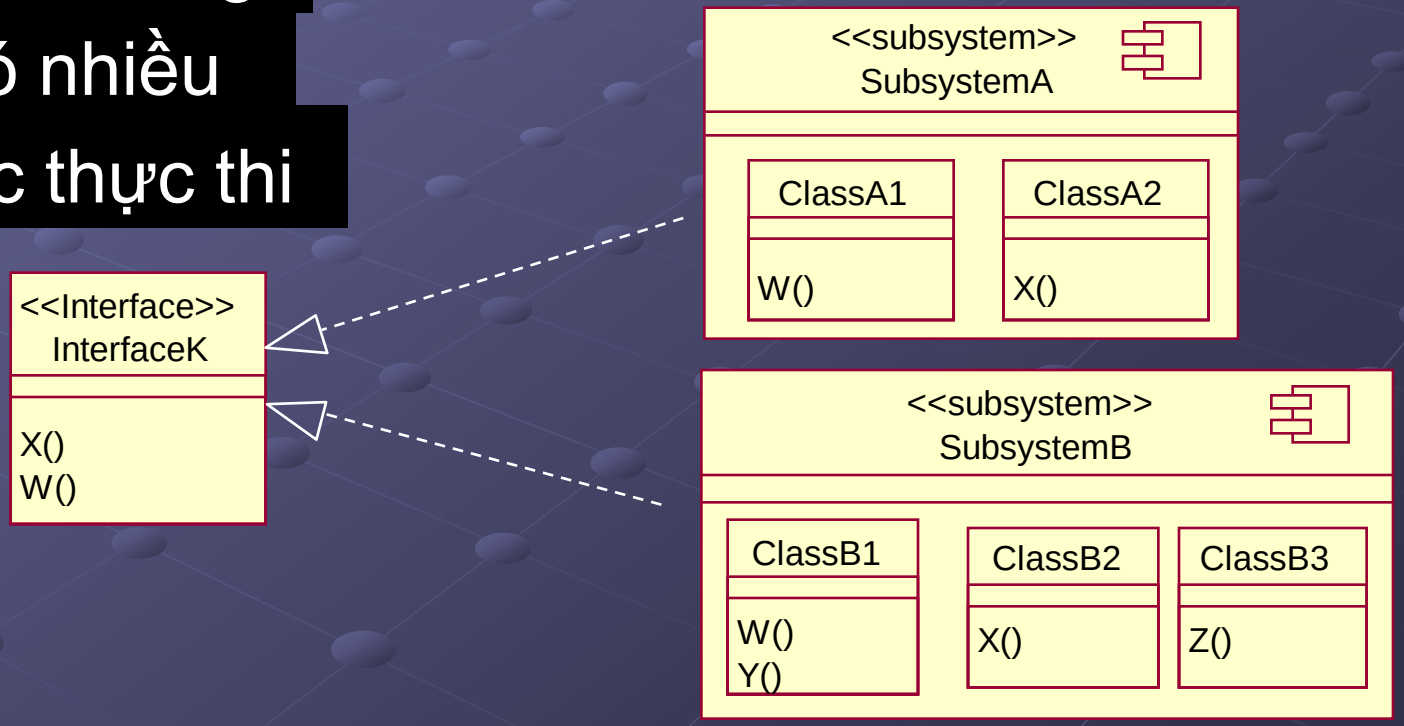


Nội dung

1. Xác định các phần tử thiết kế
2. Hệ thống con (Subsystem)
3. Tính tái sử dụng lại (Reusability)
4. Mô hình phân tầng trong quá trình thiết kế

Hệ thống con (Subsystems)

- Đóng gói hoàn chỉnh một hành vi nào đó
- Thể hiện khả năng độc lập sử dụng các giao diện một cách rõ ràng
- Có thể có nhiều hình thức thực thi

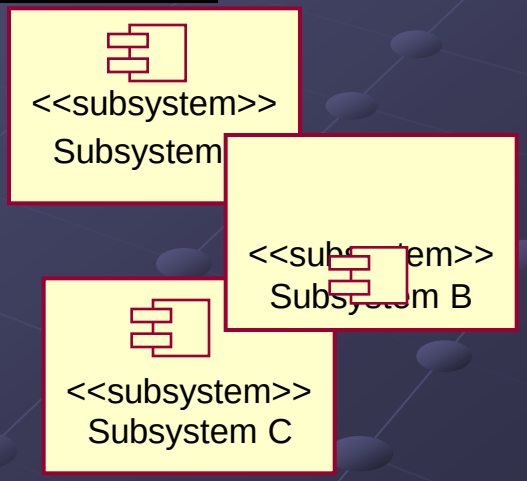


Sử dụng hệ thống con

- Phân chia hệ thống thành nhiều phần hoạt động tương đối độc lập
  - Thay đổi một phần không ảnh hưởng tới các phần còn lại
- Hệ thống con trong mô hình thiết kế sẽ trở thành thành phần trong quá trình cài đặt (components)
- Hệ thống con có thể được sử dụng để thể hiện một sản phẩm có sẵn, hoặc một hệ thống ngoại vi trong quá trình thiết kế

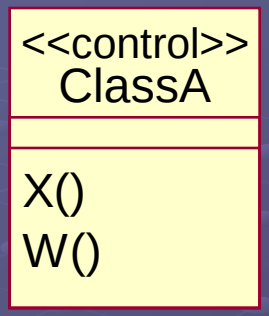
Tìm kiếm hệ thống con

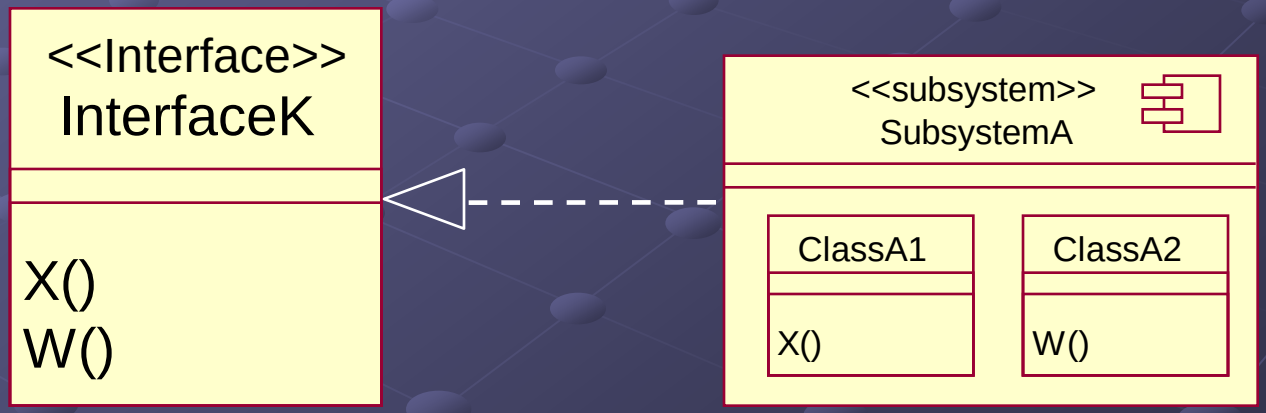
- Các phân tích có thể trở thành hệ thống con nếu:
  - Cung cấp chức năng phức tạp
  - Các lớp biên (giao diện với hệ thống bên ngoài)
- Các sản phẩm có sẵn hoặc các hệ thống bên ngoài trong quá trình thiết kế (ví dụ thành phần):
  - Phần mềm giao tiếp
  - Hỗ trợ truy cập CSDL
  - Các kiểu và cấu trúc dữ liệu
  - Các tiện ích chung
  - Các sản phẩm theo ứng dụng



Xác định hệ thống con

Superman Class”





Giao diện cho hệ thống con (Subsystem Interface)

- Mỗi hệ thống con nên có một hoặc nhiều giao diện
- Mô hình hóa các giao diện
  - Ánh xạ giao diện vào hệ thống con
  - Chỉ ra sự phụ thuộc của nó tới các lớp khác
  - Chỉ ra các hành động của giao diện
    - Tham số và kết quả
    - Kiểu dữ liệu
- Đóng gói các giao diện

Một giao diện rõ ràng, ổn định
là giải pháp tốt cho việc tạo ra một kiến trúc hiệu quả

Nội dung

1. Xác định các phần tử thiết kế
2. Hệ thống con (Subsystem)
3. Tính tái sử dụng lại (Reusability)
4. Mô hình phân tầng trong quá trình thiết kế

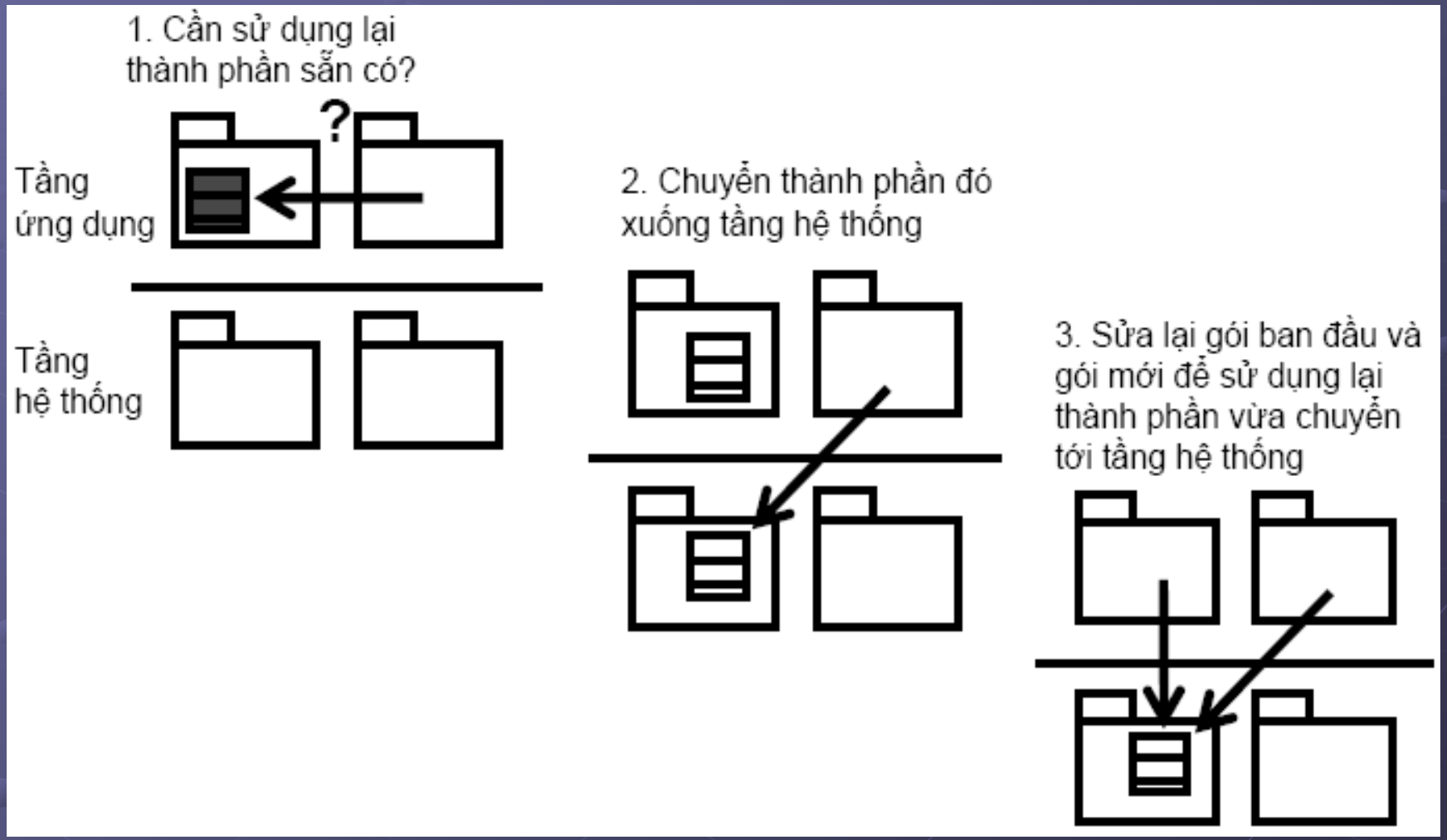
3. Tính sử dụng lại

- Mục đích
  - Sử dụng các giao diện để tìm cách sử dụng lại các hệ thống con hoặc các thành phần sẵn có trong hệ thống
- Hướng dẫn
  - Tìm kiếm các gần giao diện giống nhau
  - Sửa giao diện cho phù hợp với giao diện sẽ sử dụng lại
  - Thay thế giao diện có khả năng sử dụng lại với giao diện sẵn có (sử dụng lại)
  - Ánh xạ hệ thống con của giao diện vừa bị thay thế vào thành phần có sẵn đó để sử dụng lại

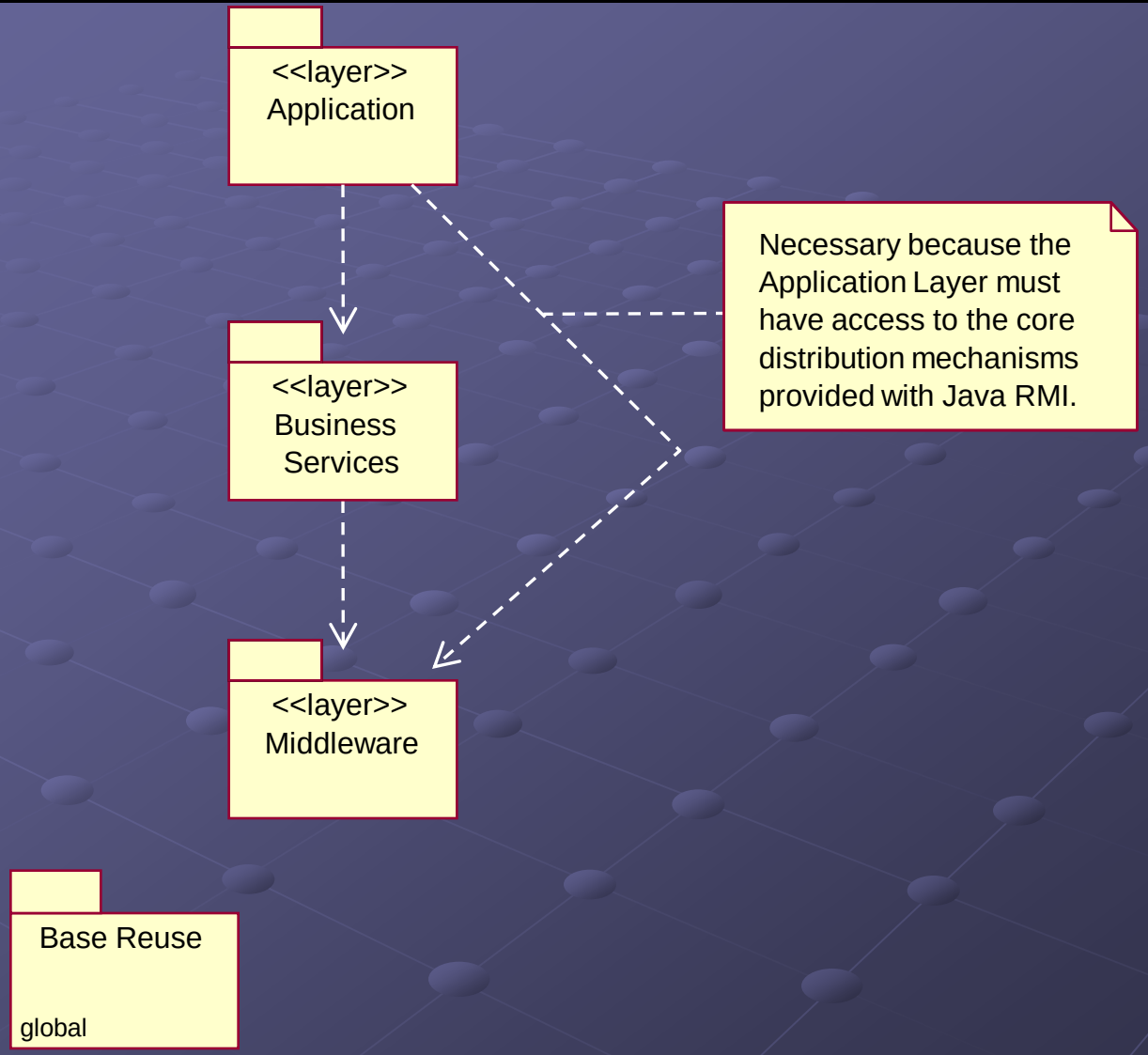
Các khả năng sử dụng lại

- Bên trong hệ thống
  - Tìm ra những điểm chung giữa các gói hoặc hệ thống con
- Bên ngoài hệ thống
  - Sử dụng các thành phần sẵn có (thương mại, miễn phí)
  - Thành phần từ hệ thống phát triển trước đây
  - Phát triển lại một thành phần có sẵn (sử dụng lại thiết kế)

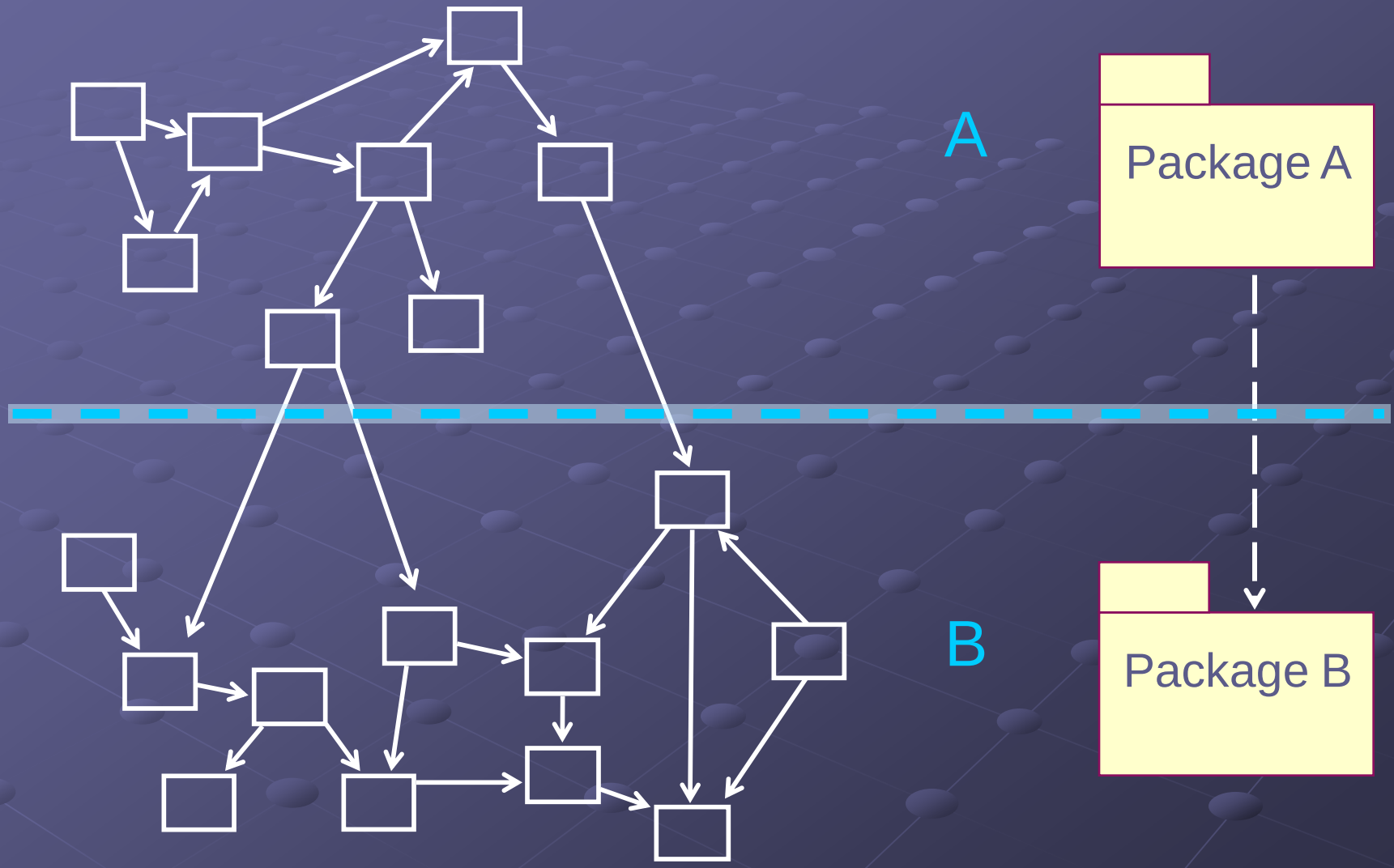
Khả năng sử dụng lại bên trong hệ thống



Ví dụ: Các tầng kiến trúc

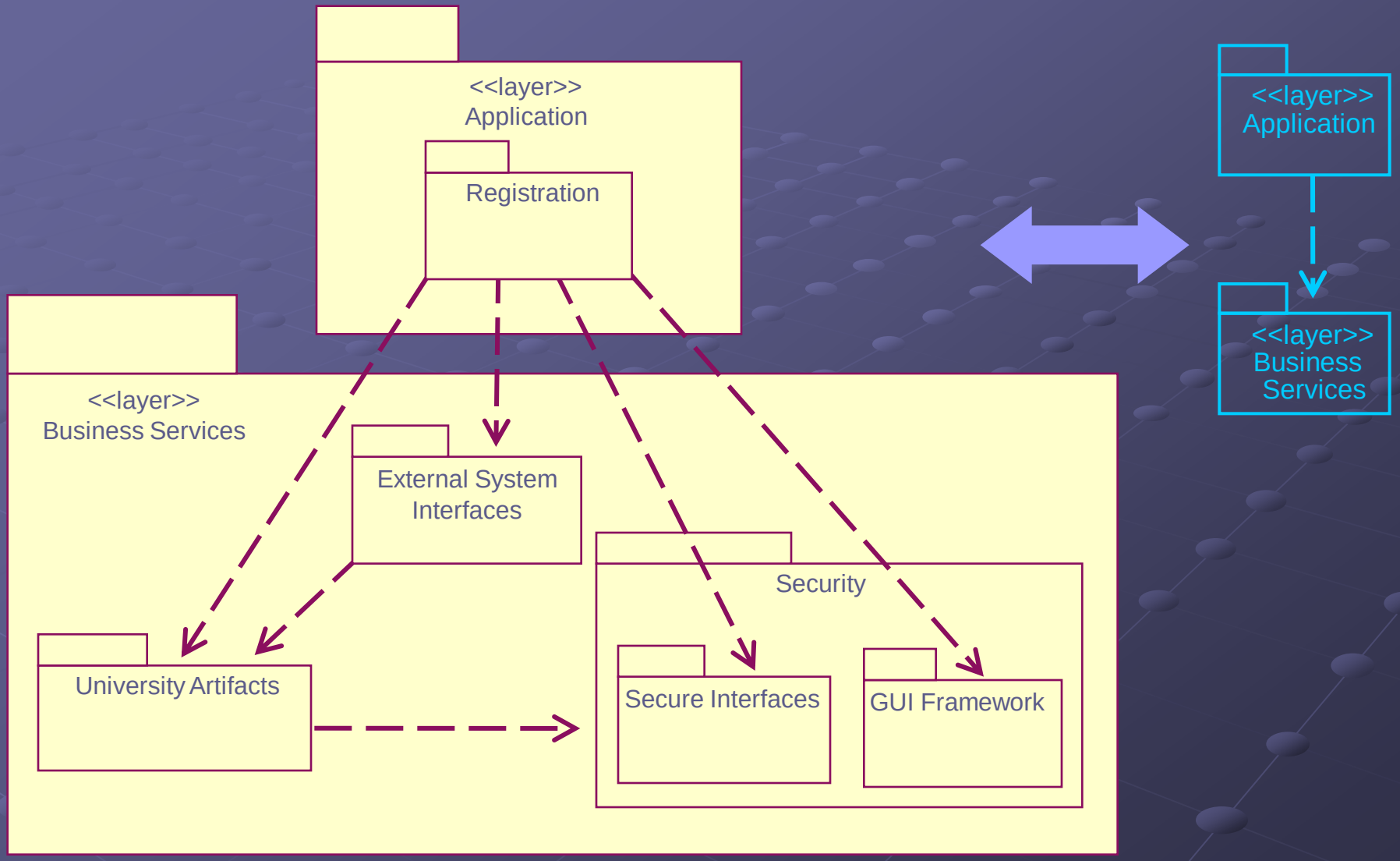


Quan hệ giữa các lớp tạo nên sự phụ thuộc

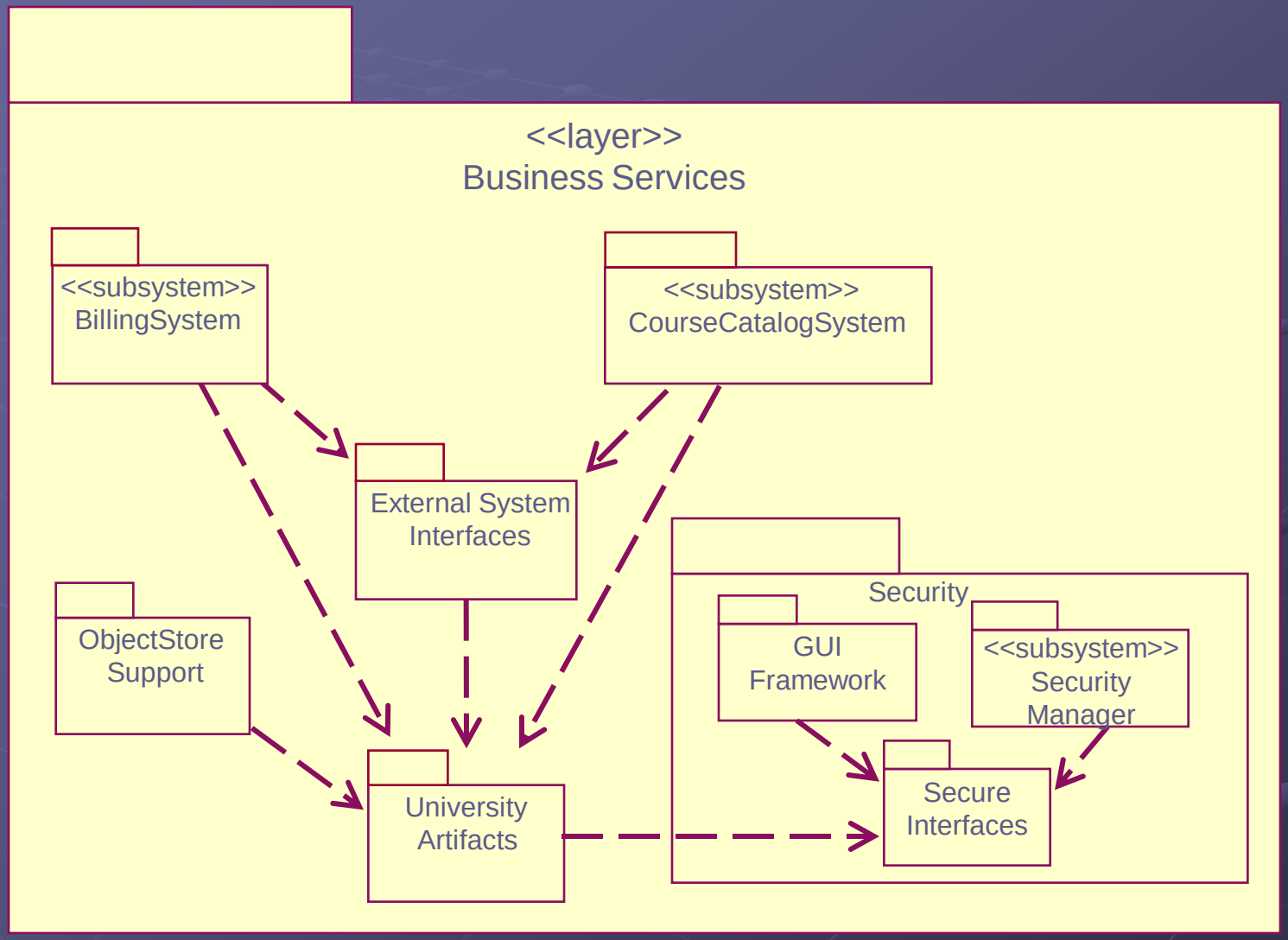


Ví dụ: Tầng ứng dụng

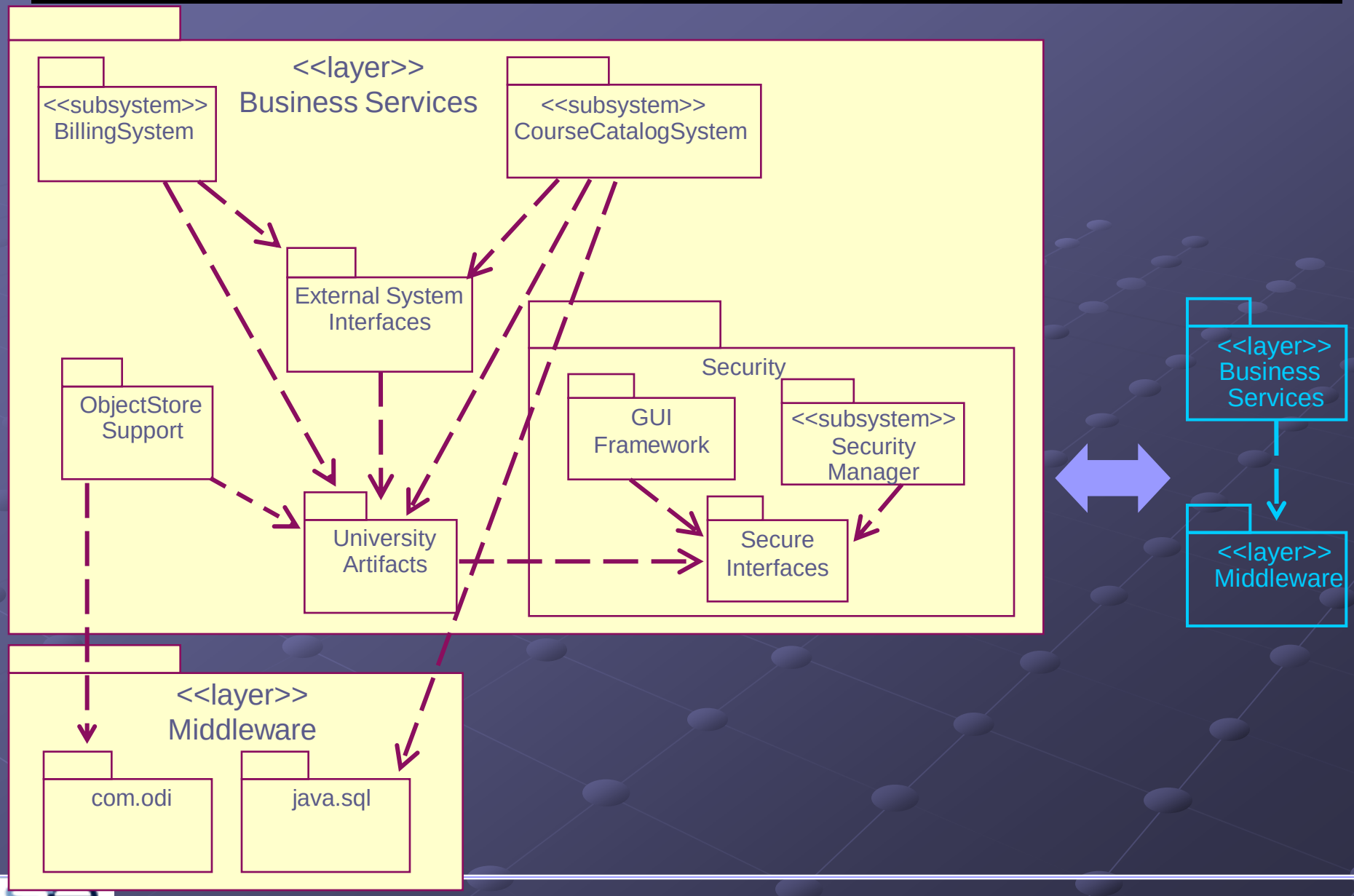
Ví dụ: Ngữ cảnh của tầng ứng dụng



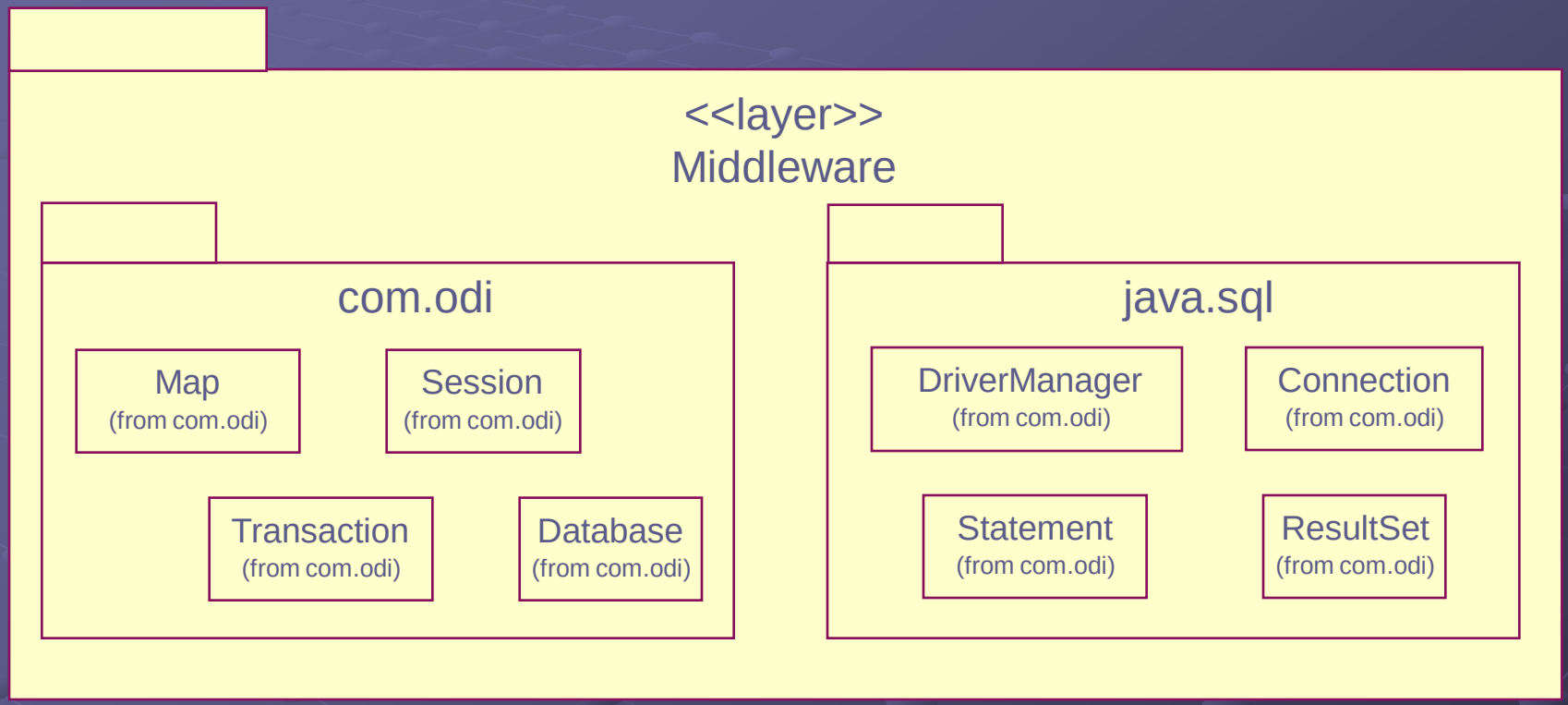
Ví dụ tầng Business Service



Ví dụ: Ngữ cảnh tầng Business Service



Ví dụ tầng Middle Layer



Tổng kết

- Các mô hình thiết kế được xây dựng trực tiếp từ các mô hình phân tích
  - Là hình thức chi tiết hóa từ các lớp trừu tượng hóa trong mô hình phân tích
    - Ánh xạ 1-1 cho những lớp phân tích đơn giản
  - Ánhxạ thành nhiều lớp thiết kế nếu lớp phân tích đó quá phức tạp
- Lớp phân tích có mức độ phức tạp cao có thể được phát triển thành hệ thống con (subsystem)
  - Sử dụng các giao diện
  - Đảm bảo hệ thống con có tính độc lập tối đa với các thành phần còn lại
- Tìm cách sử dụng lại các hệ thống con, gói hoặc các thư viện có sẵn

BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

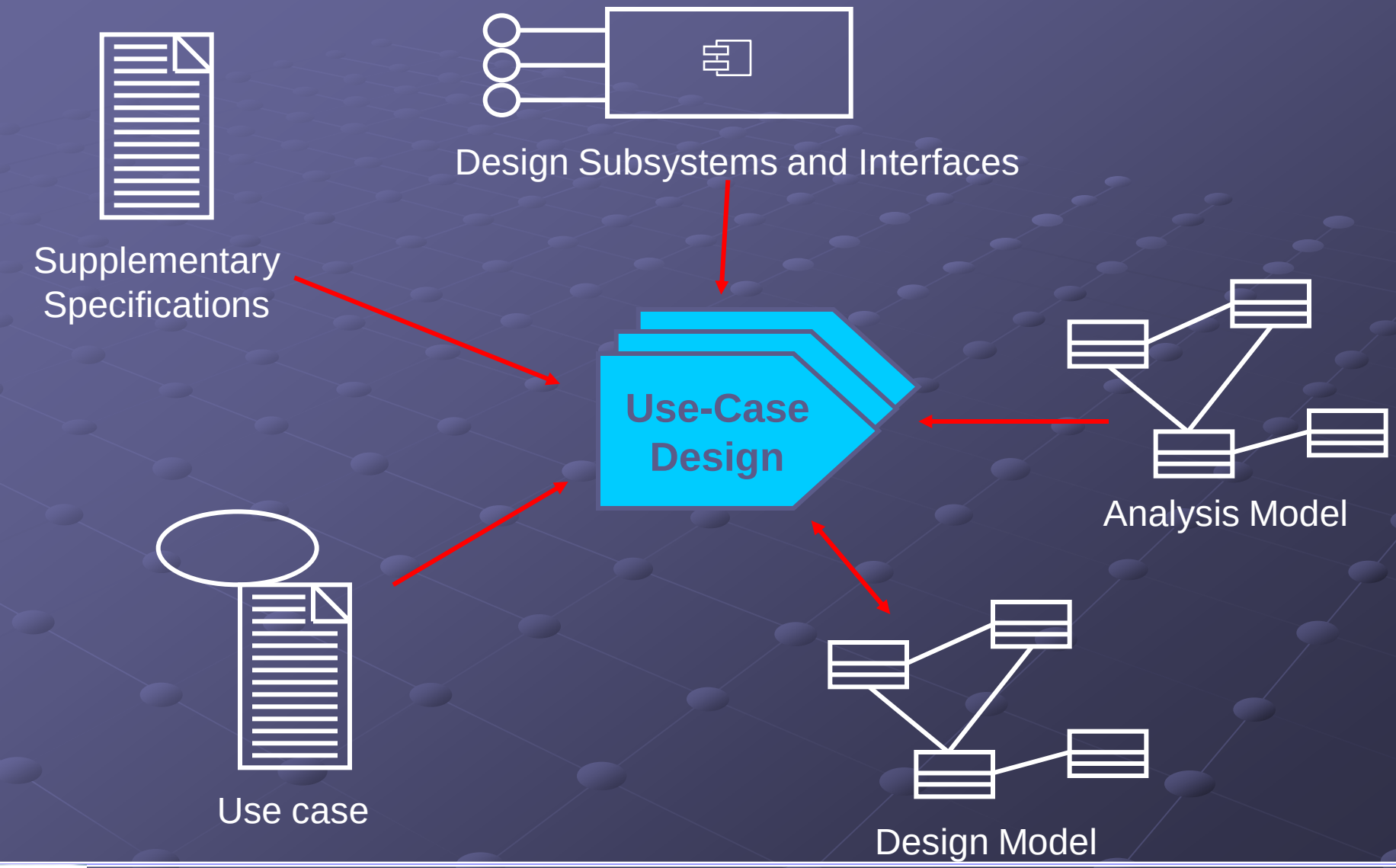
OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 7. Thiết kế use case

Mục đích

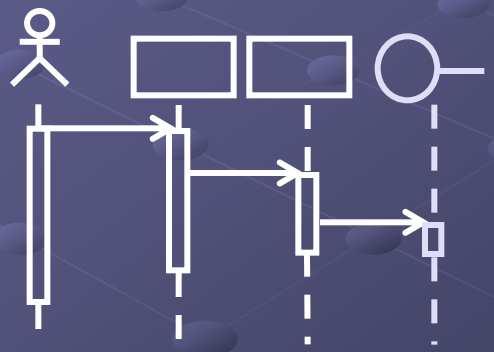
- Kiểm tra sự nhất quán trong quá trình thực hiện use case
- Tinh chỉnh sự hiện thực hóa use case từ mô hình phân tích sử dụng các thành phần thiết kế

Tổng quan về thiết kế use case



Tinh chỉnh việc hiện thực hóa use case

- Xác định các đối tượng tham gia vào luồng use case sử dụng các thành phần thiết kế
- Mô hình hóa thông điệp giữa các đối tượng vào biểu đồ tương tác
- Tinh chỉnh biểu đồ lớp

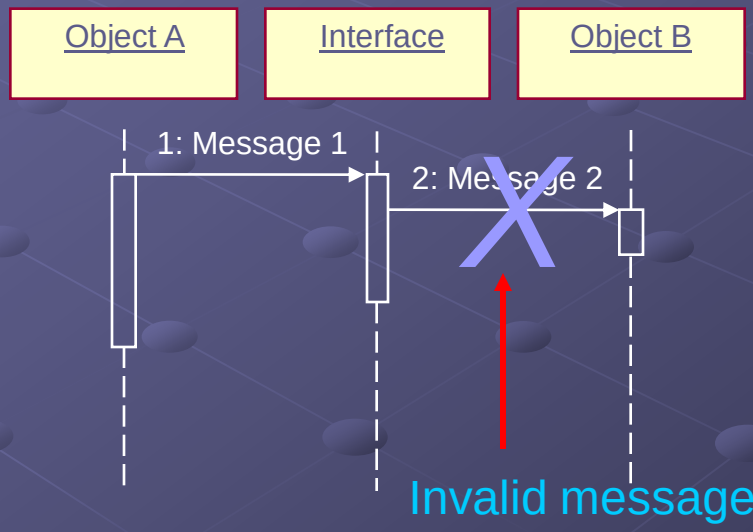


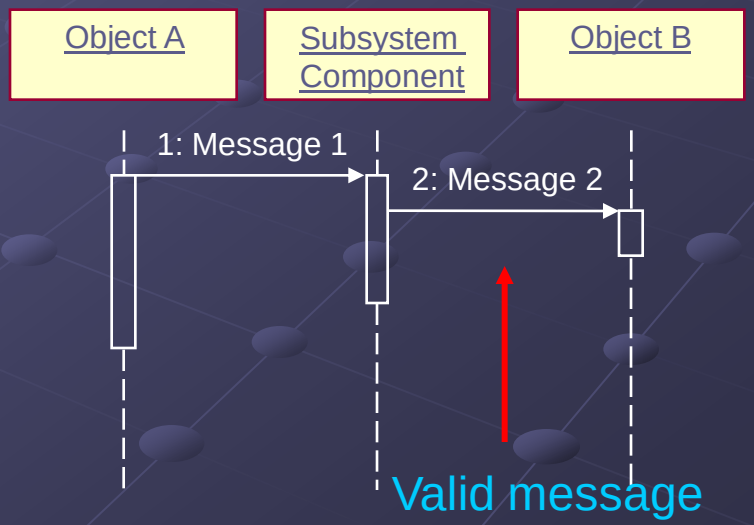
Sequence Diagrams

Class Diagrams

Biểu diễn hệ thống con vào biểu đồ trình tự

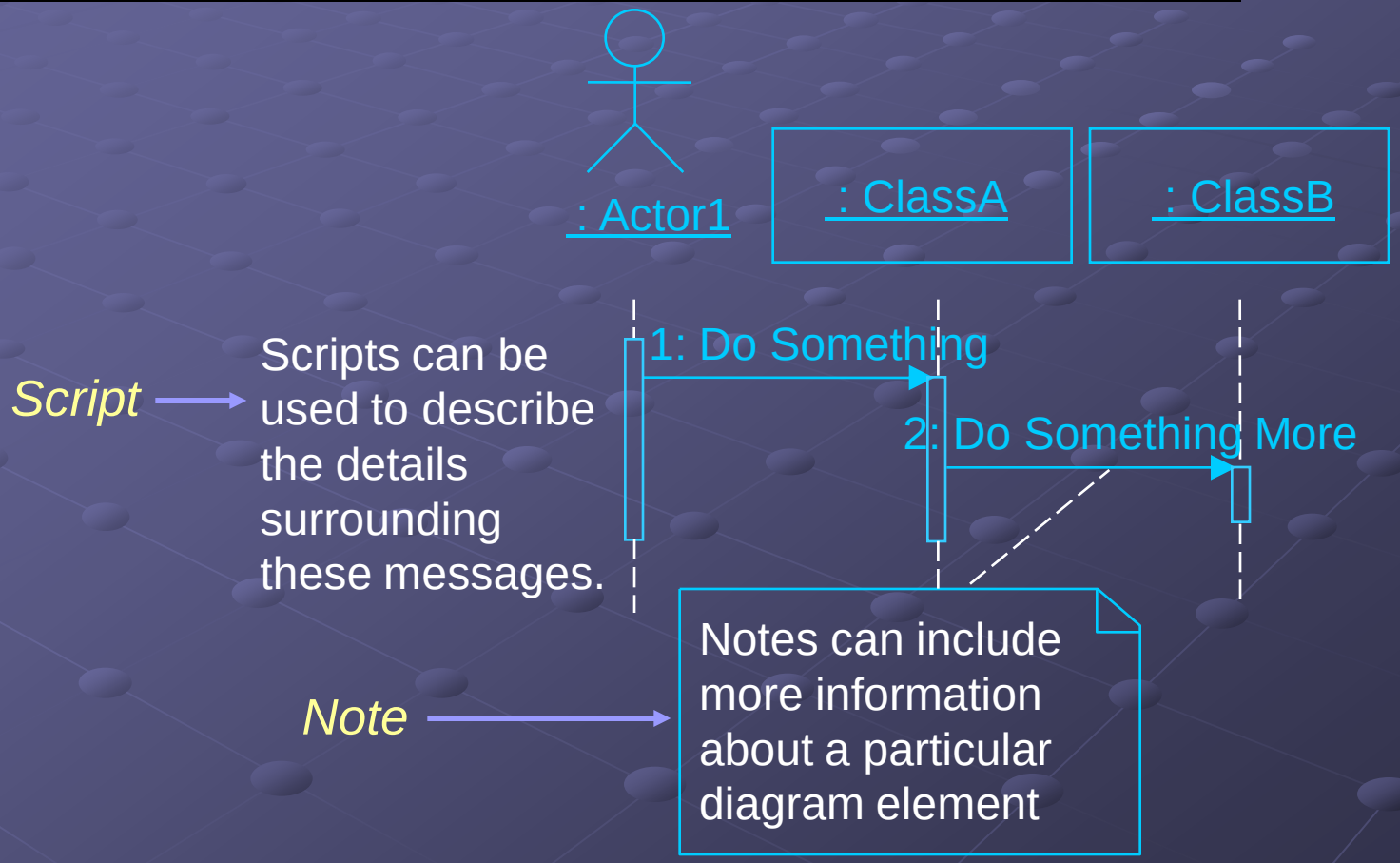
- Giao diện
  - Biểu diễn bất kỳ phần tử nào thực thi giao diện
  - Không có thông điệp nào được vẽ từ giao diện
- Thành phần hệ thống con
  - Biểu diễn một hệ thống con cụ thể
  - Thông điệp có thể vẽ từ giao diện





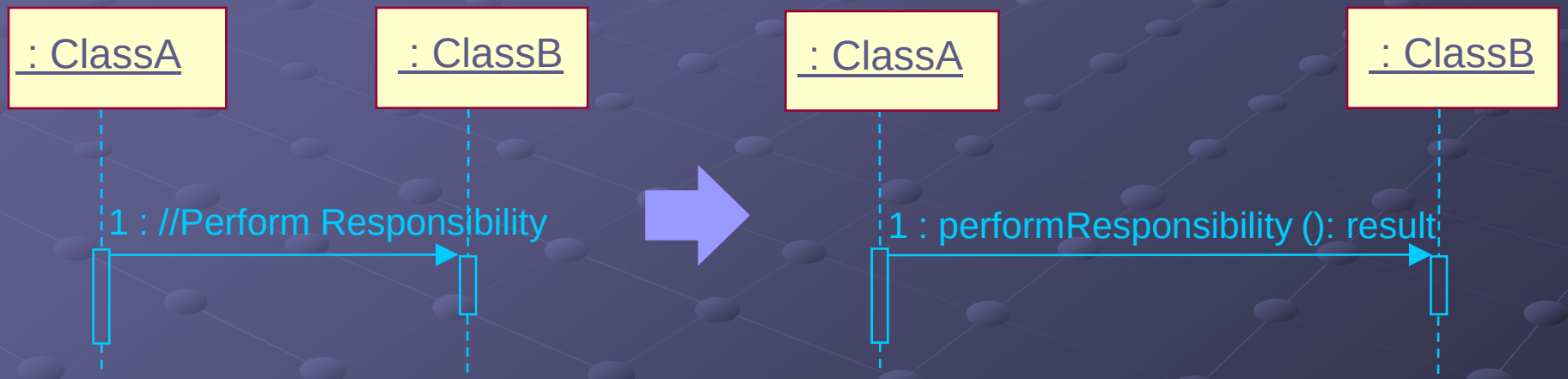
Mô tả chi tiết cho luồng sự kiện của use case

Chú thích cho biểu đồ tương tác

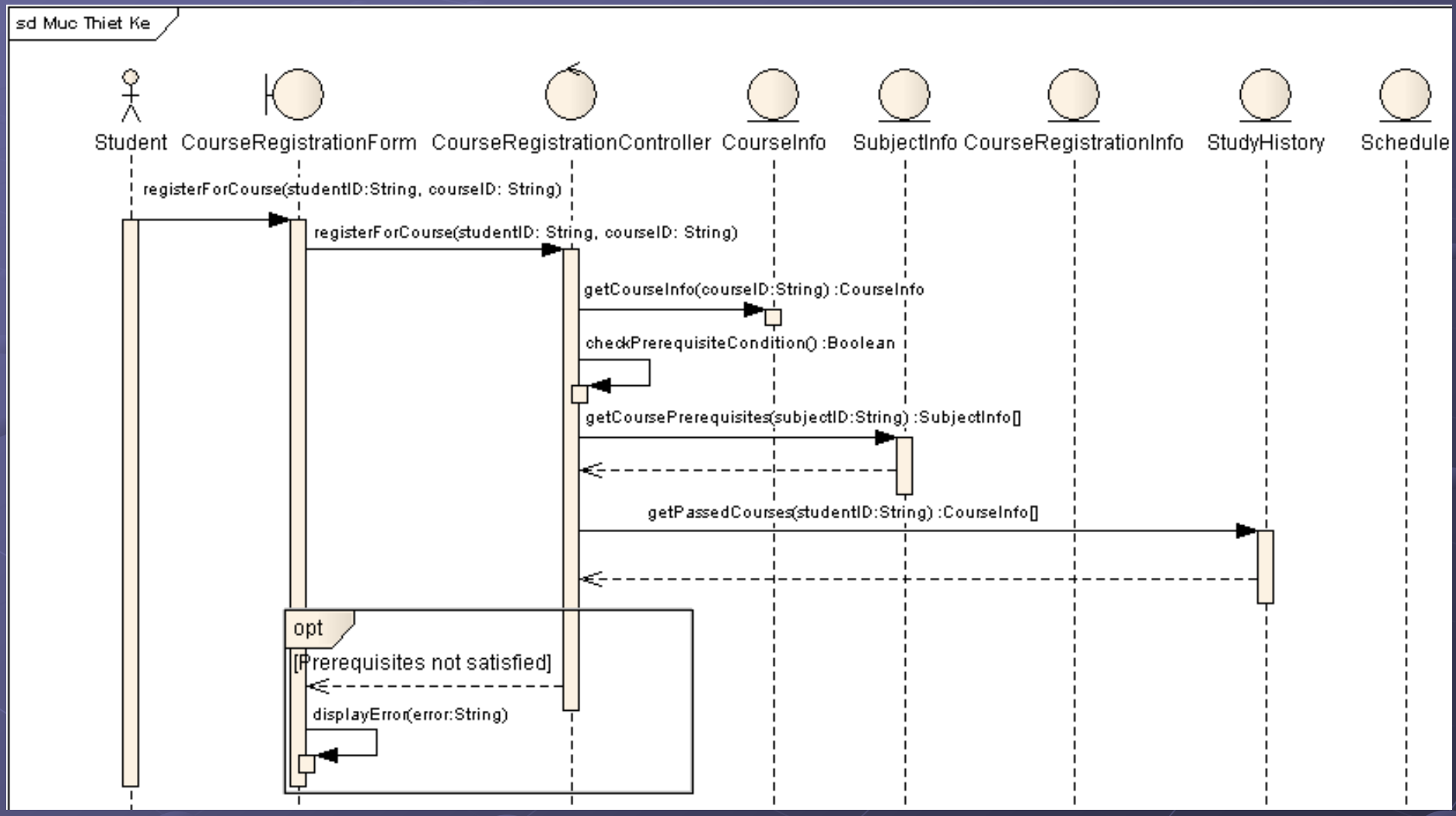


Tinh chỉnh các thông điệp thành c

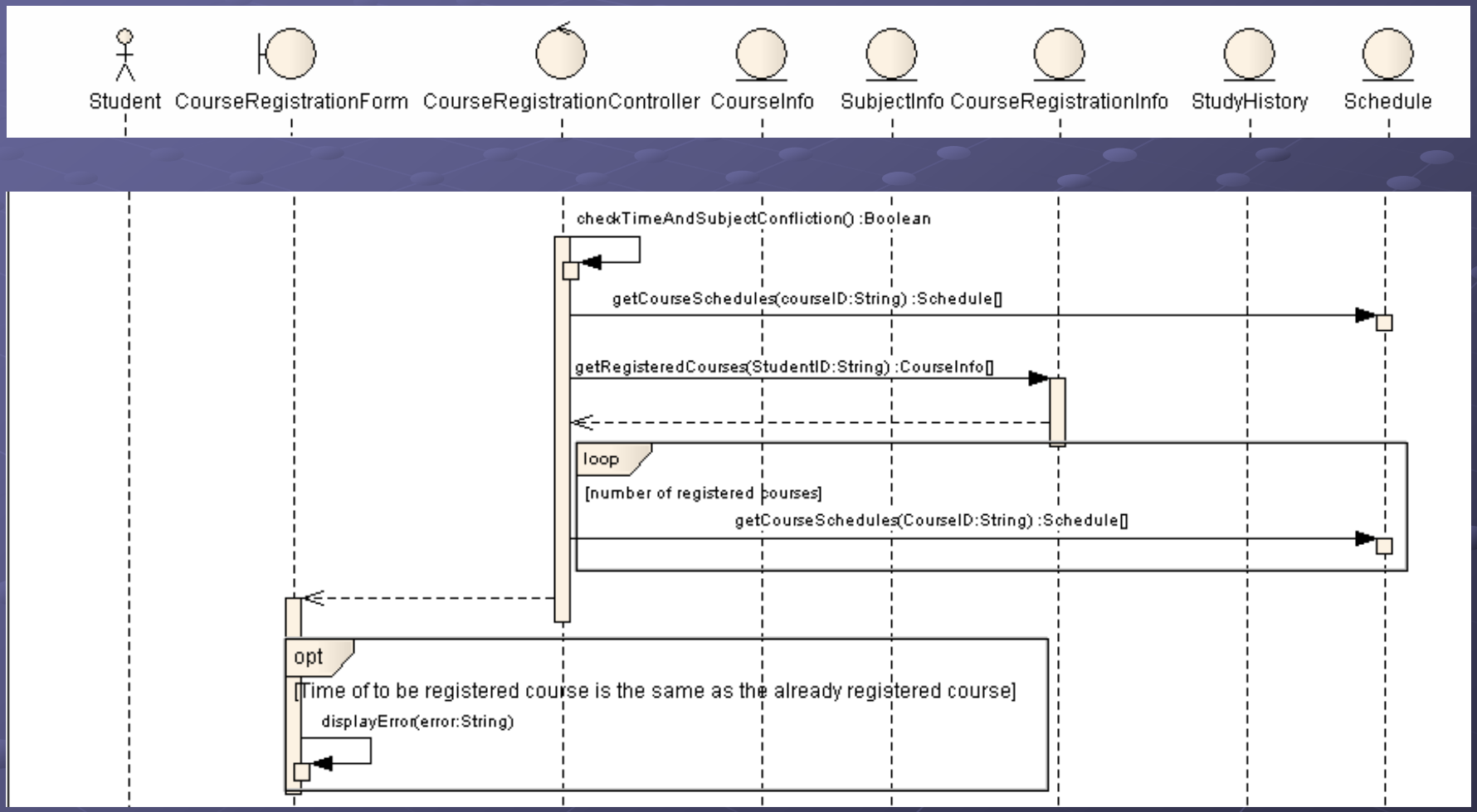
- Các thông điệp được hiển thị trong các biểu đồ tương tác



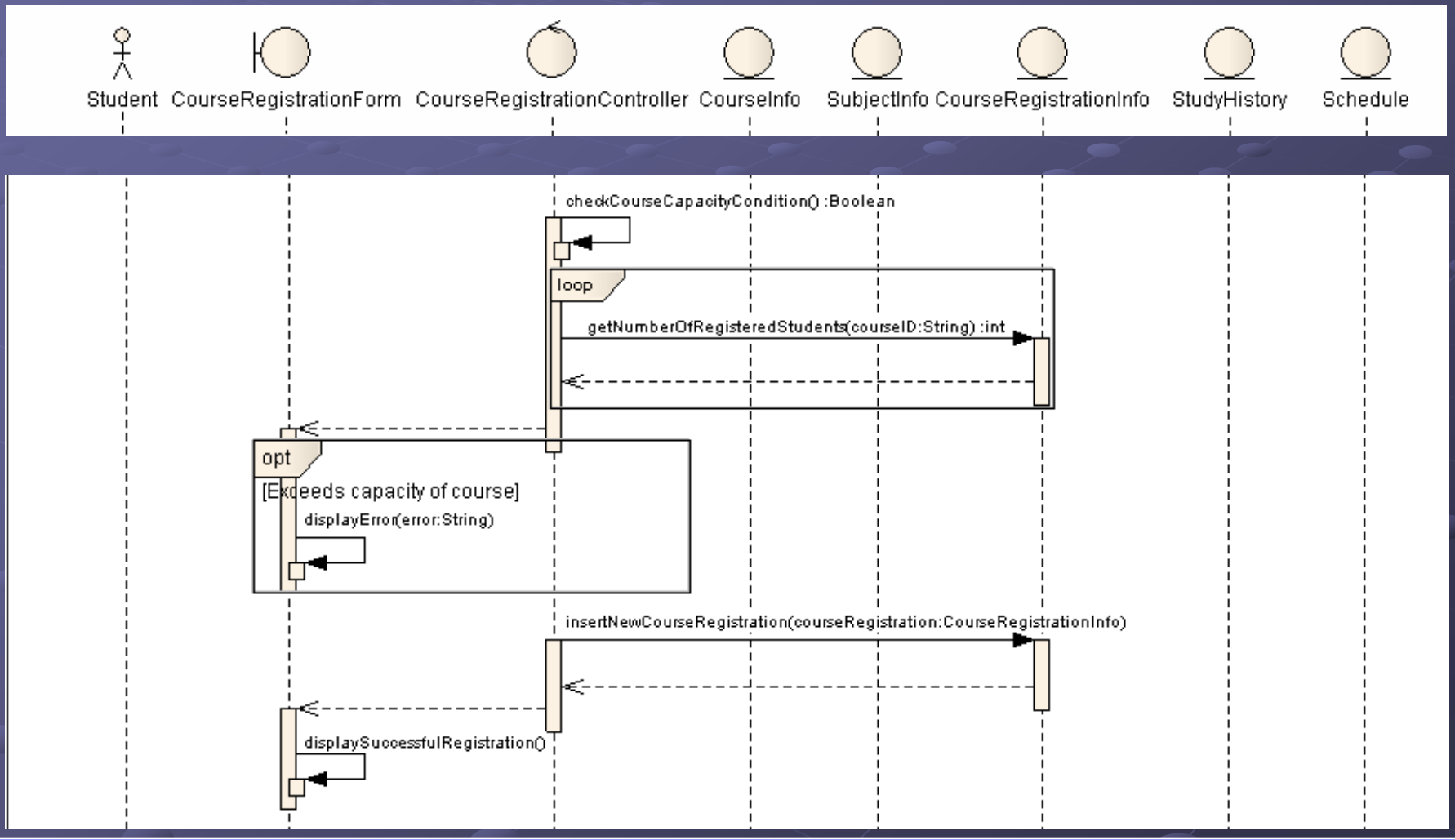
Biểu đồ trình tự mức thiết kế



Biểu đồ trình tự mức thiết kế (2)

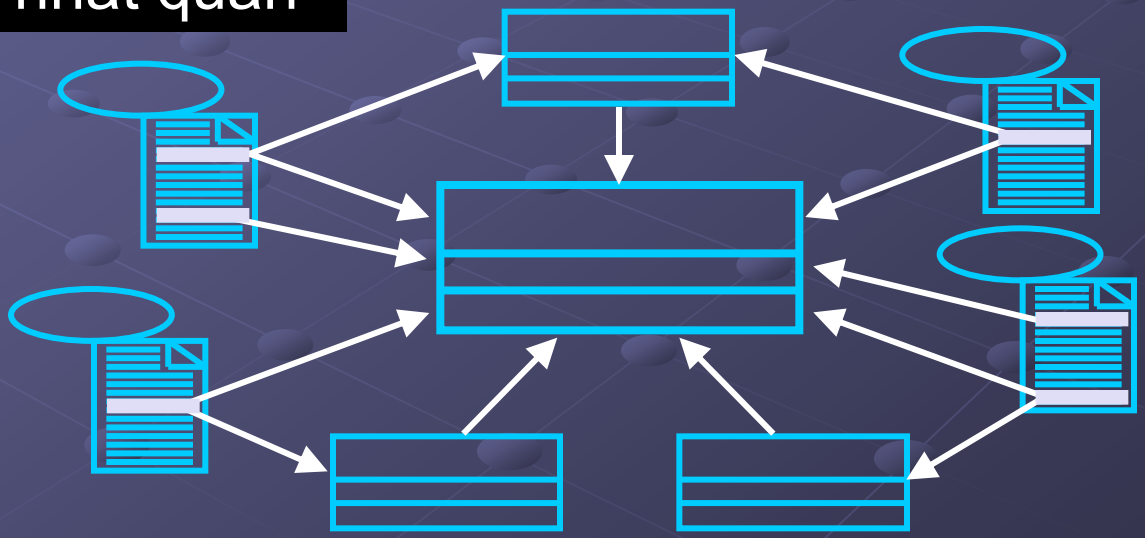


Biểu đồ trình tự mức thiết kế (3)

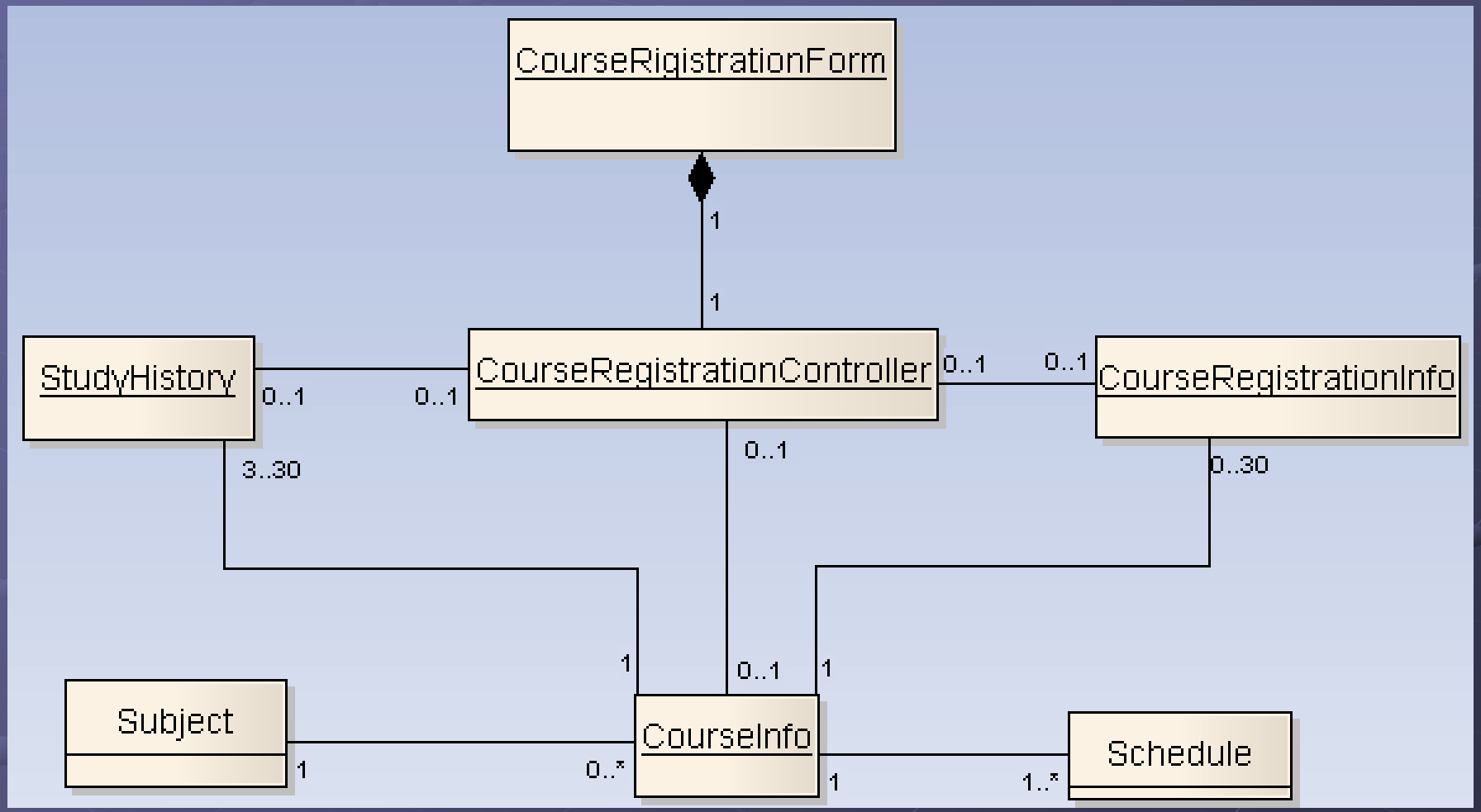


Thống nhất mô hình thiết kế

- Tên của các thành phần trong mô hình phải mô tả chức năng của chúng
- Kết hợp các thành phần tương tự nhau
- Sử dụng kế thừa để trừu tượng hóa các phần tử mô hình
- Đảm bảo cho các thành phần mô hình và luồng sự kiện được nhất quán



Thống nhất biểu đồ lớp

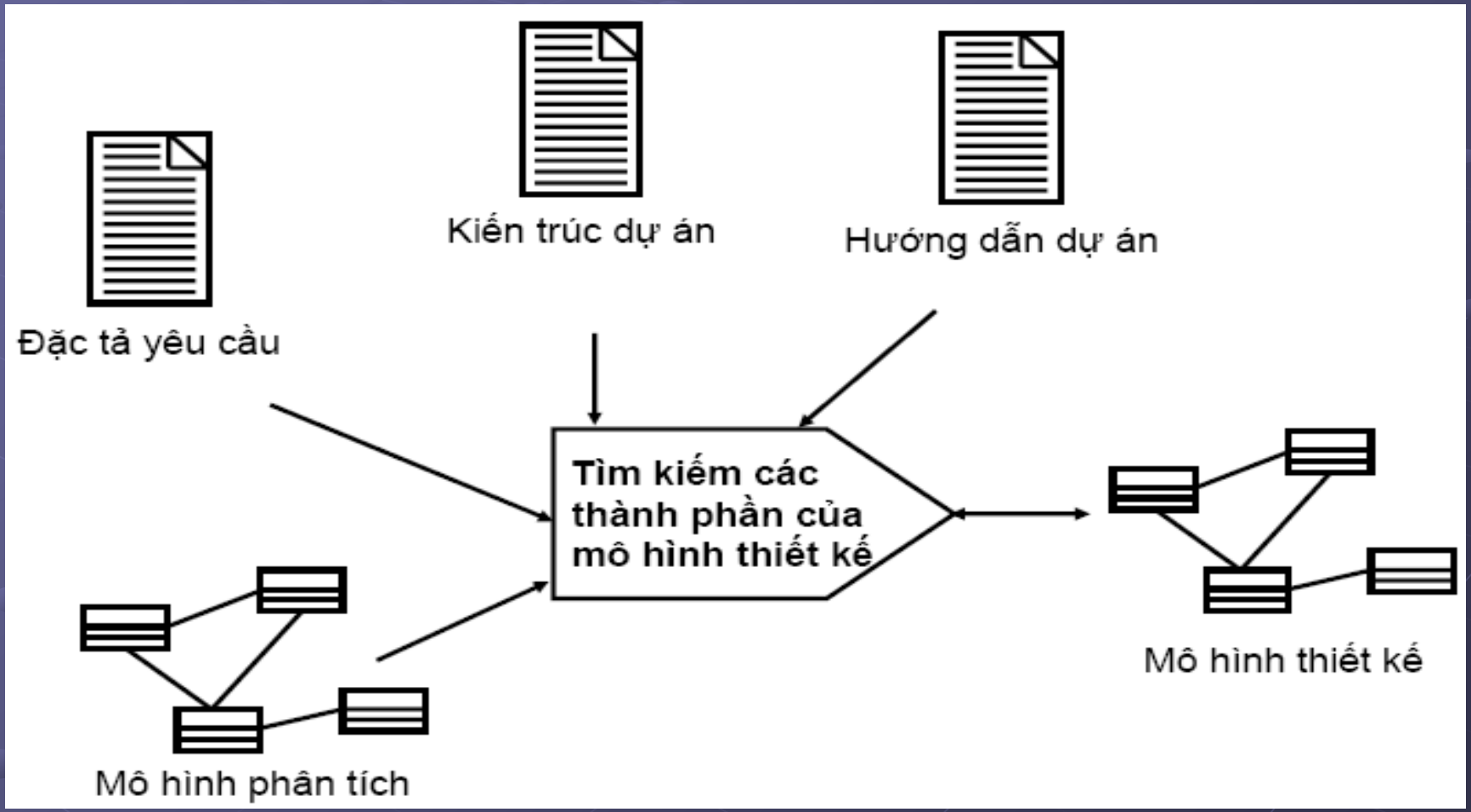


BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 8. Thiết kế lớp

Mô hình thiết kế

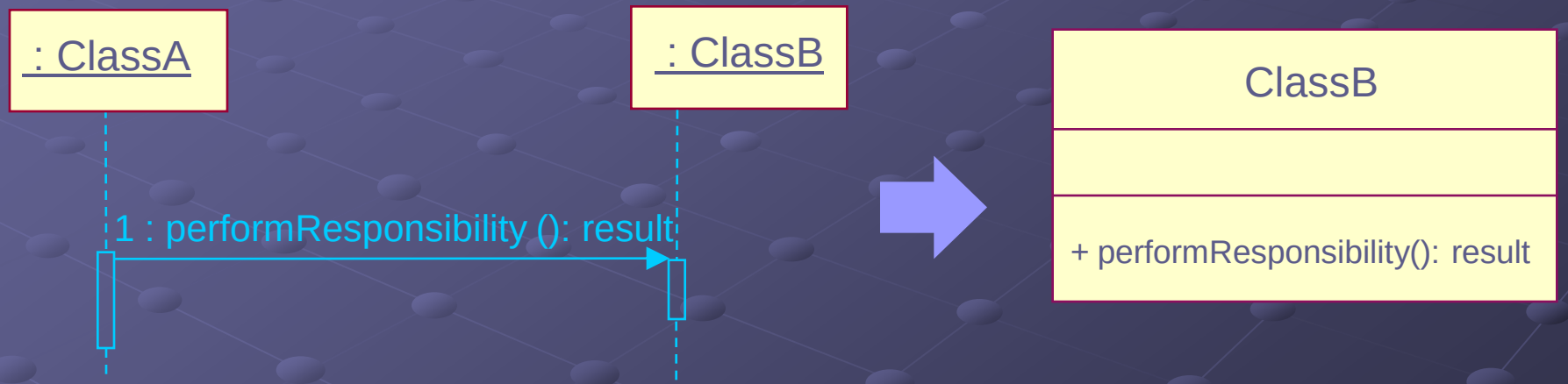


Nội dung

1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

1 c

- Ánh xạ các thông điệp trong biểu đồ tương tác thành các thao tác của các lớp



C

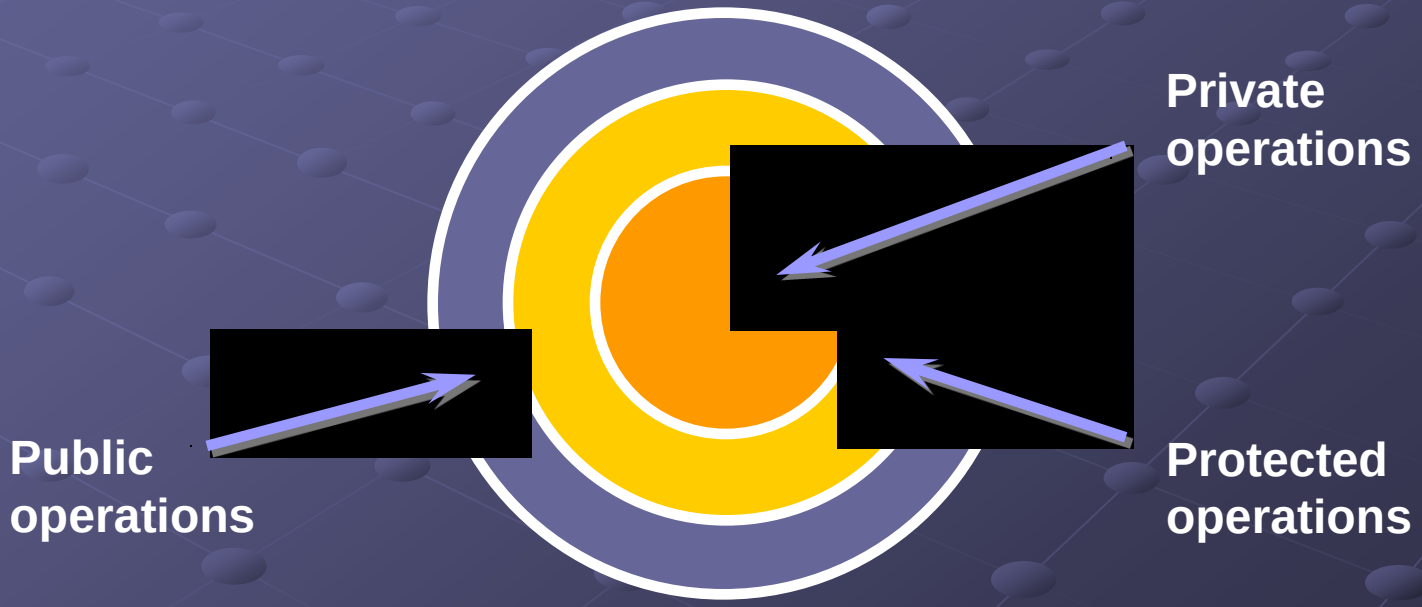
- Tạo ra các tên thao tác thích hợp
  - Mô tả kết quả
  - Sử dụng góc nhìn của đối tượng khách (gọi)
  - Nhất quán giữa các lớp
- Xác định chữ ký của thao tác
  - **operationName([direction]parameter : class,..) : returnType**
    - Direction: **in** (mặc định), **out** hoặc **inout**
- Đưa ra mô tả ngắn gọn, bao gồm ý nghĩa của tất cả các tham số

C

- Khi thiết kế chữ ký của thao tác, cần xem xét liệu tham số có:
  - Được truyền theo tham trị hay tham biến
  - Có bị thay đổi bởi thao tác hay không
  - Có tùy chọn không
  - Thiết lập các giá trị mặc định
  - Các khoảng tham số không hợp lệ
- Càng ít tham số, càng tốt
- Truyền các đối tượng thay vì hàng loạt các dữ liệu.

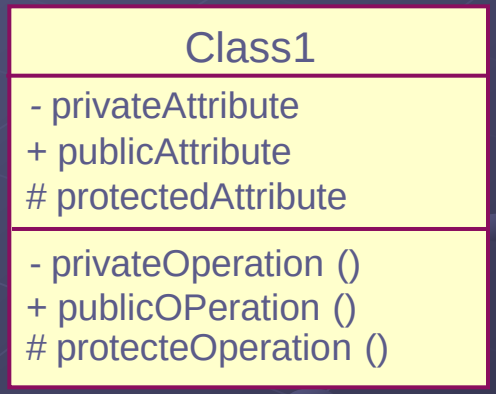
Phạm vi truy cập của thao tác (Operation Visibility)

- Phạm vi truy cập được sử dụng để thực hiện khả năng đóng gói
- Có thể là public, protected, hoặc private



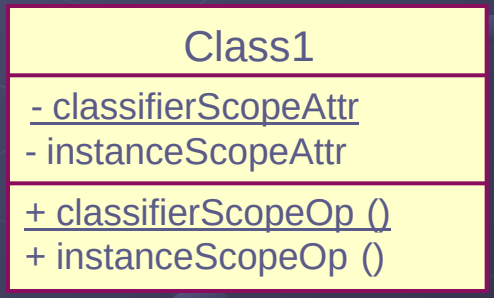
Phạm vi truy cập được biểu diễn như thế nào?

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

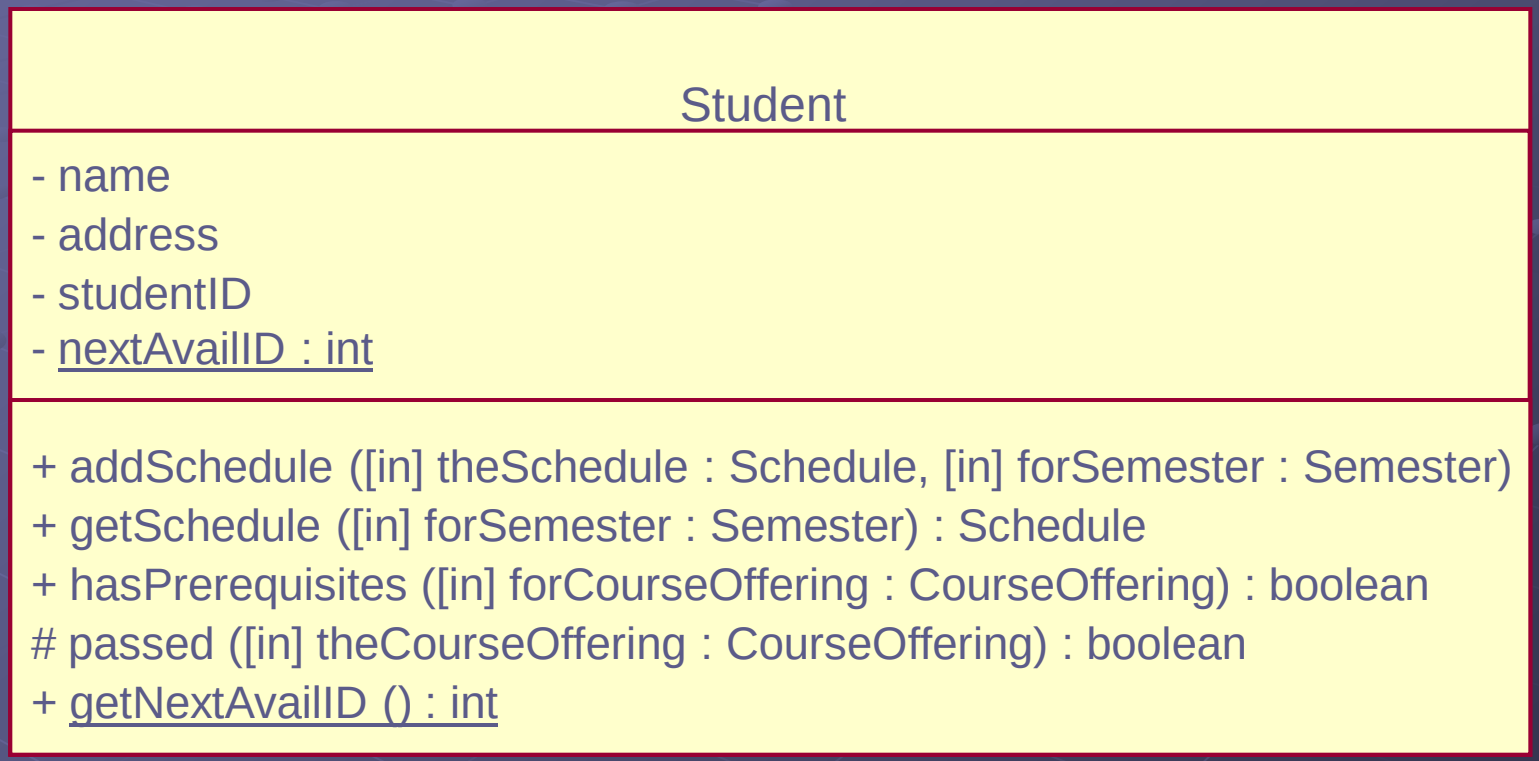


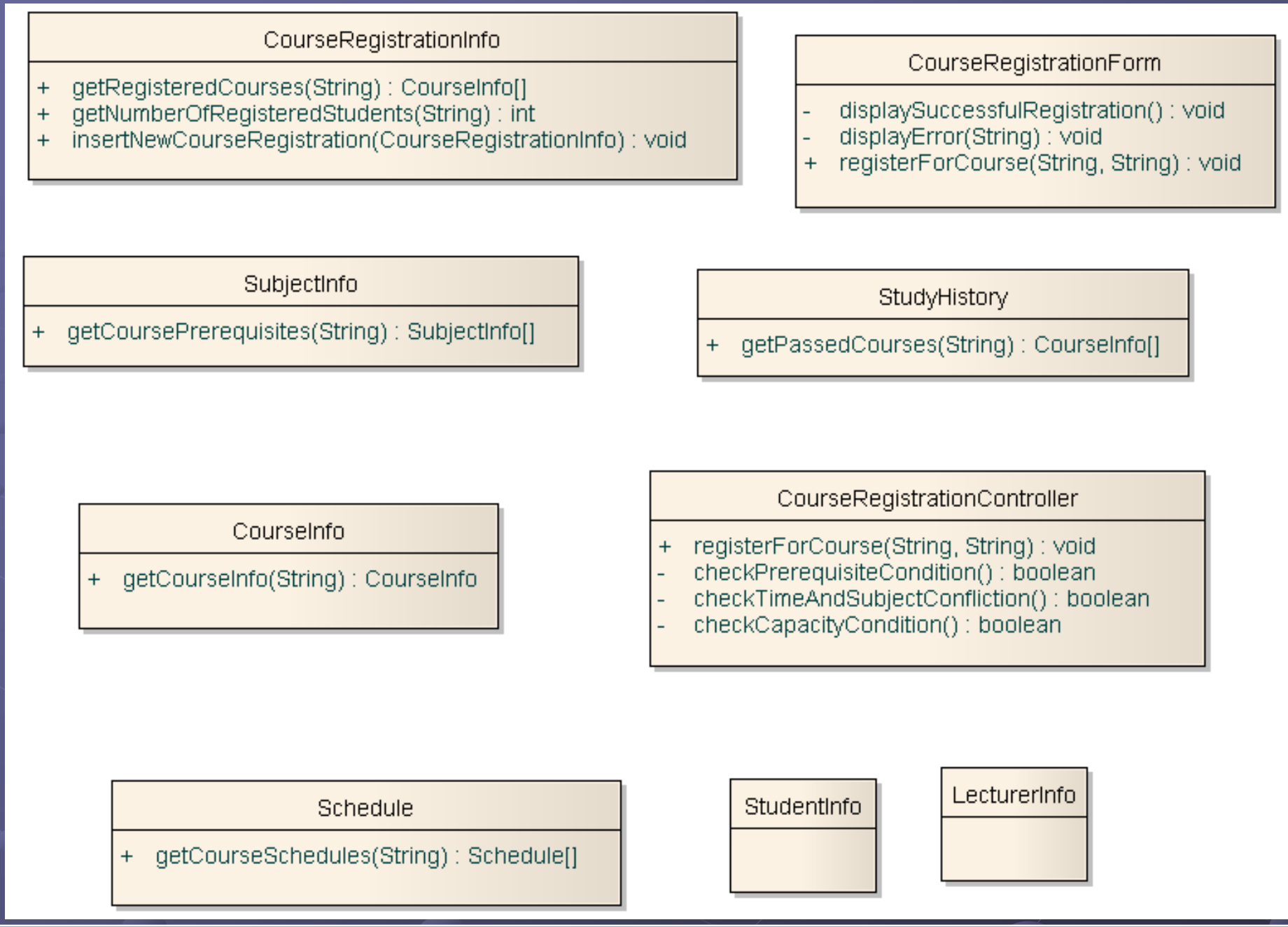
Phạm vi (Scope)

- Xác định số lượng thể hiện của thuộc tính/thao tác:
  - Instance: Một thể hiện cho mỗi thể hiện của mỗi lớp
  - Classifier: Một thể hiện cho tất cả các thể hiện của lớp
- Phạm vi Classifier được ký hiệu bằng cách gạch dưới tên thuộc tính/thao tác.



Ví dụ: Scope


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


CourseRegistrationInfo
+ getRegisteredCourses(String) : CourseInfo[]
+ getNumberOfRegisteredStudents(String) : int
+ insertNewCourseRegistration(CourseRegistrationInfo) : void
CourseRegistrationForm
- displaySuccessfulRegistration() : void
- displayError(String) : void
+ registerForCourse(String, String) : void
SubjectInfo
+ getCoursePrerequisites(String) : SubjectInfo[]
StudyHistory
+ getPassedCourses(String) : CourseInfo[]
CourseInfo
+ getCourseInfo(String) : CourseInfo
CourseRegistrationController
+ registerForCourse(String, String) : void
- checkPrerequisiteCondition() : boolean
- checkTimeAndSubjectConfliction() : boolean
- checkCapacityCondition() : boolean
Schedule
+ getCourseSchedules(String) : Schedule[]
StudentInfo
LecturerInfo

Nội dung

1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

2 c

- ?
- c (operation)
- ch
- c vấn đề riêng cho c
- t:
- t
- ng
- ng
- ng

Nội dung

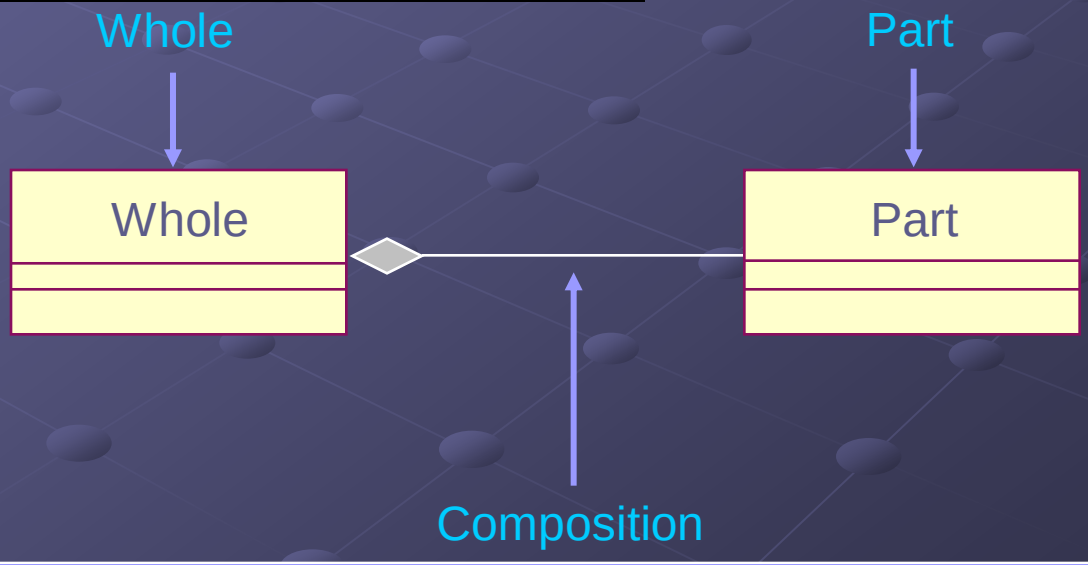
1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

3. Xác định các liên kết

- Mục đích
  - Tinh chỉnh các kết hợp còn lại
- Các vấn đề cần xem xét:
  - Association vs. Aggregation
  - Aggregation vs. Composition
  - Attribute vs. Association
  - Navigability
  - Multiplicity design

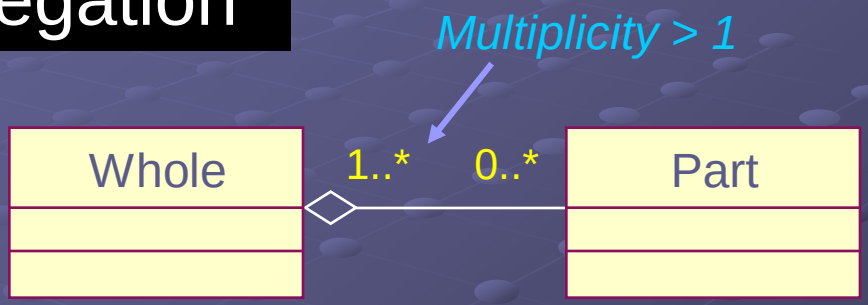
Composition là gì?

- Một dạng của aggregation với quyền sở hữu mạnh và các vòng đời trùng khớp.
  - Whole sở hữu Part, tạo và hủy Part.
  - Part bị bỏ đi khi Whole bị bỏ, Part không thể tồn tại nếu Whole không tồn tại.

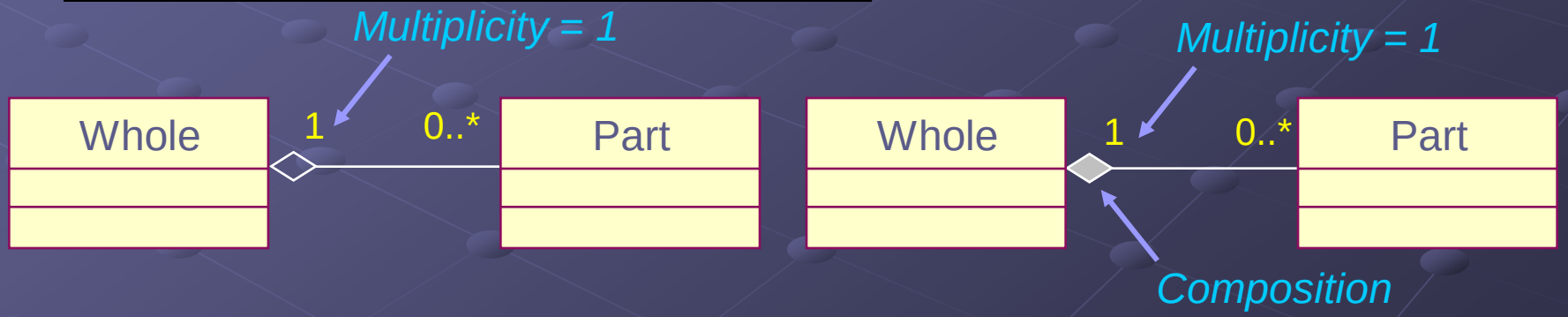


Aggregation: Shared vs. Non-shared

Shared Aggregation



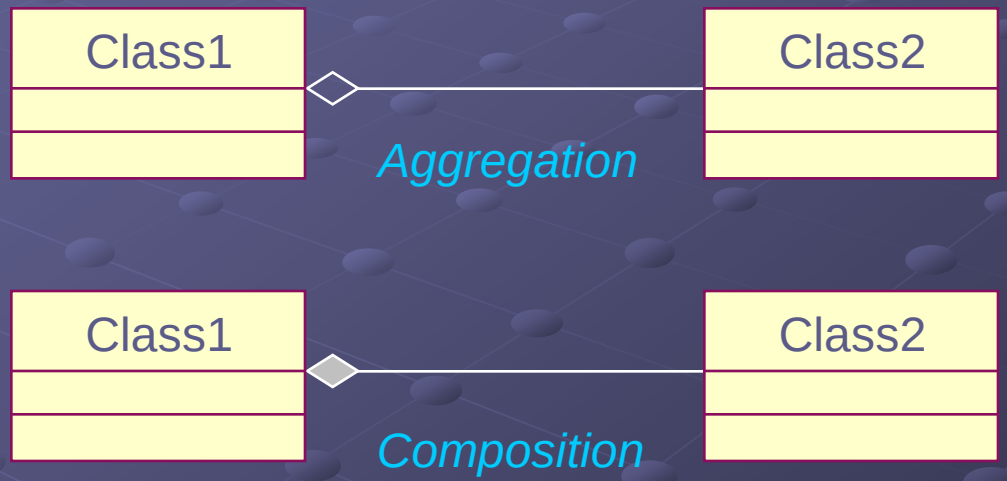
Non-shared Aggregation



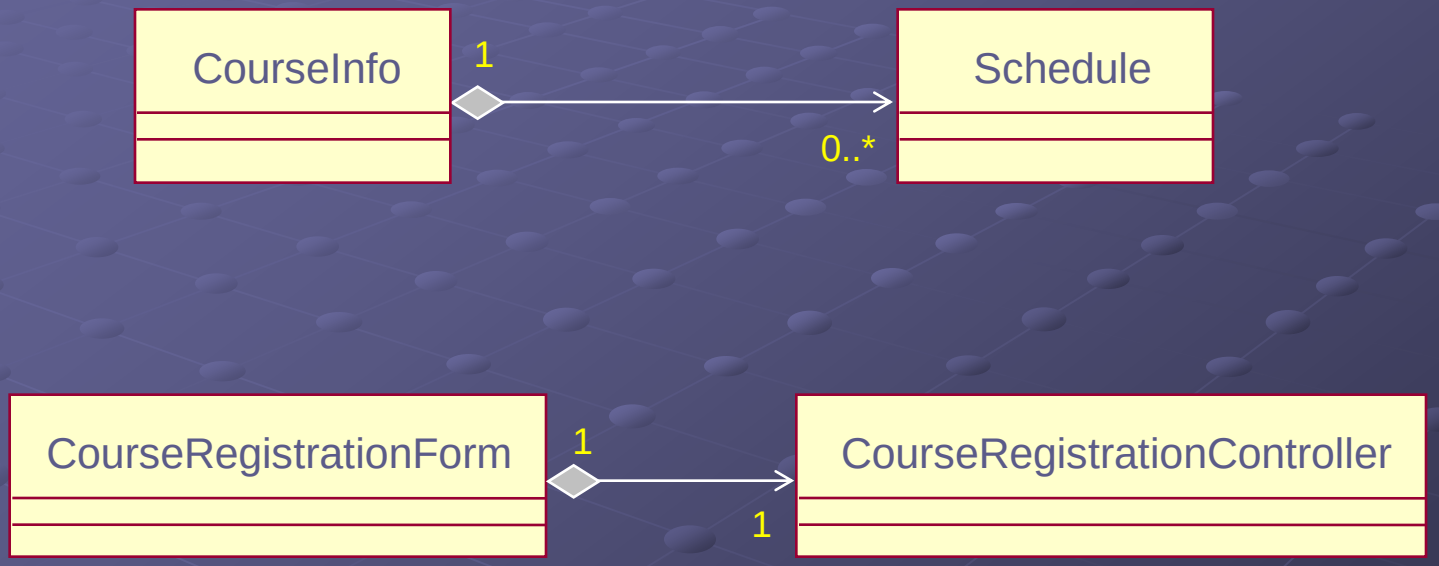
By definition, composition is non-shared aggregation.

Aggregation hay Composition?

- Xem xét vòng đời của Class1 và Class2



Ví dụ: Composition

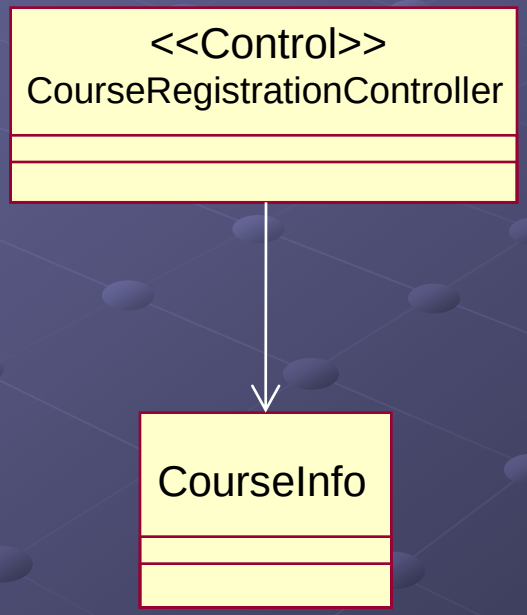


Attributes và Composition

- Sử dụng composition khi:
  - Các đặc tính (property) cần định danh độc lập
  - Nhiều lớp có cùng các đặc tính
  - Các đặc tính có một cấu trúc phức tạp và các đặc tính của riêng chúng
  - Các đặc tính phải có hành vi phức tạp của riêng chúng
  - Các đặc tính có mối quan hệ của riêng chúng
- Còn lại sử dụng thuộc tính (attribute)

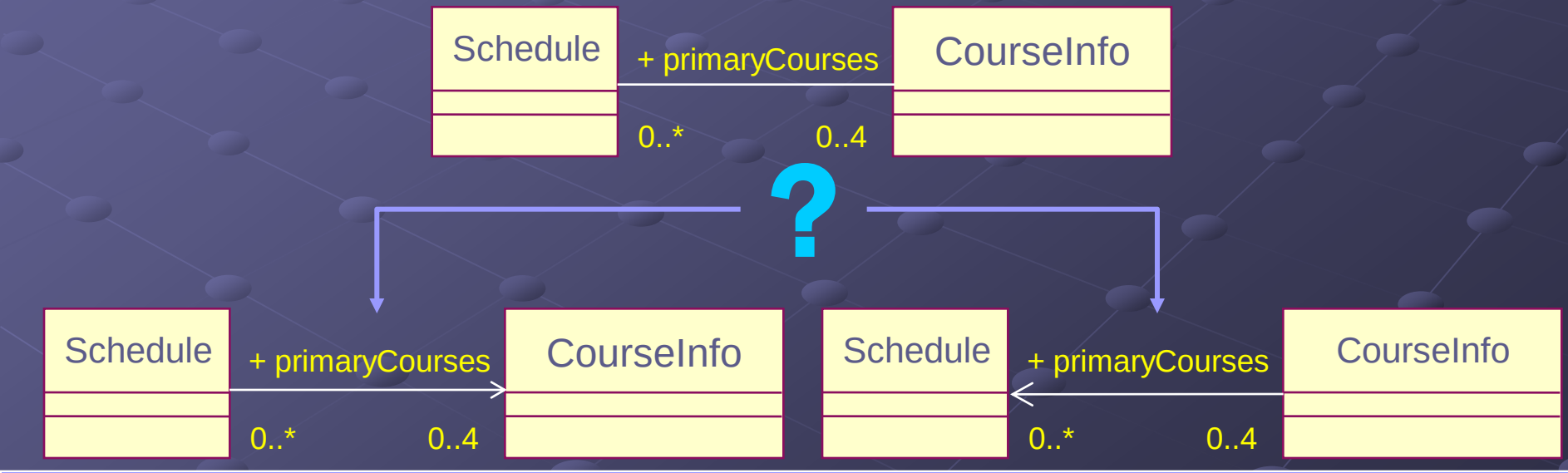
Điều hướng (Navigability) là gì?

- Điều hướng từ một lớp kết hợp đến lớp đích sử dụng association.



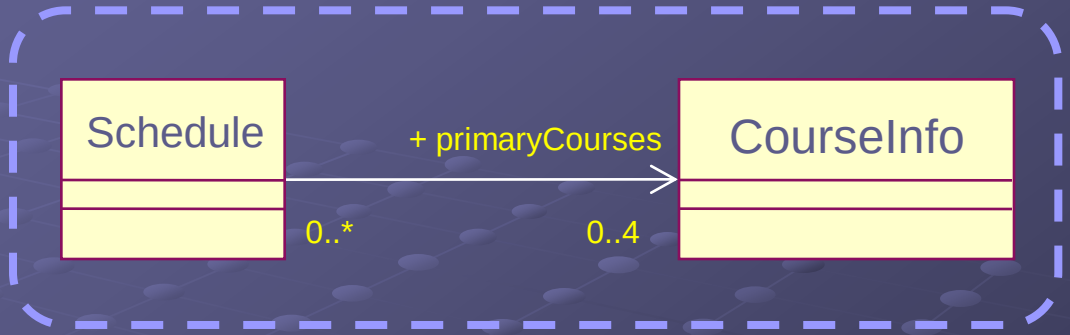
Navigability: Hướng nào thực sự cần?

- Xem xét các biểu đồ tương tác
- Thậm chí khi cả hai hướng đều có vẻ cần nhưng một hướng lại vẫn hoạt động tốt
  - Navigability theo 1 hướng ít xảy ra
  - Số lượng thể hiện của một lớp là nhỏ



Ví dụ: Tinh chỉnh điều hướng

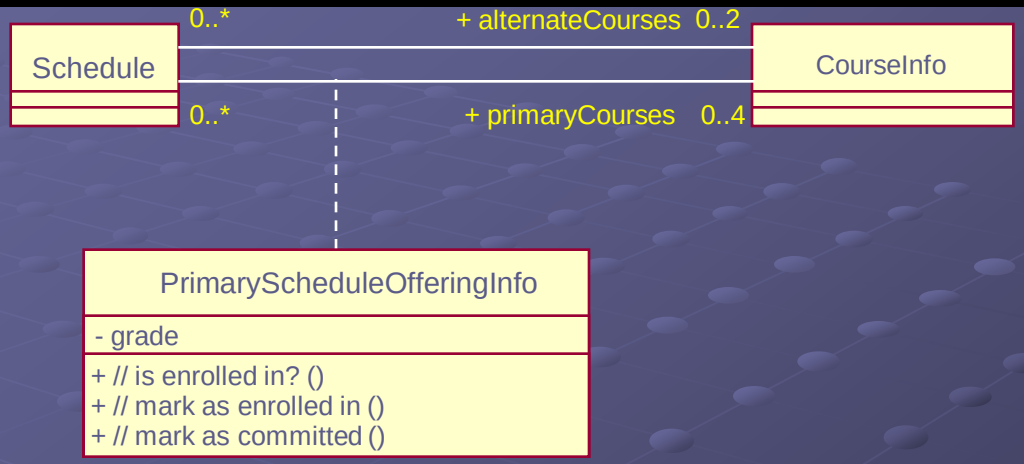
- Total number of Schedule is small, or
- Never need a list of the Schedule on which the CourseInfo appears



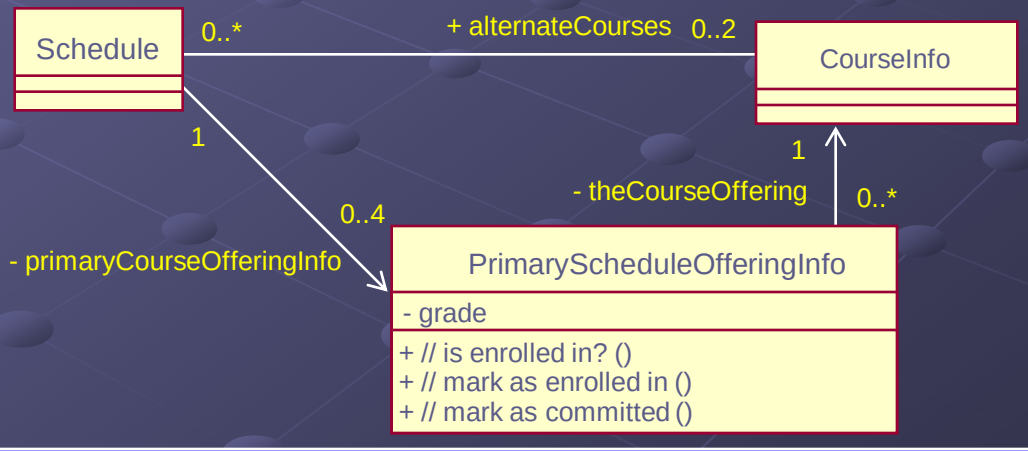
- Total number of CourseInfo is small, or
- Never need a list of CourseInfo on a Schedule

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

Example: Association Class Design



Design Decisions

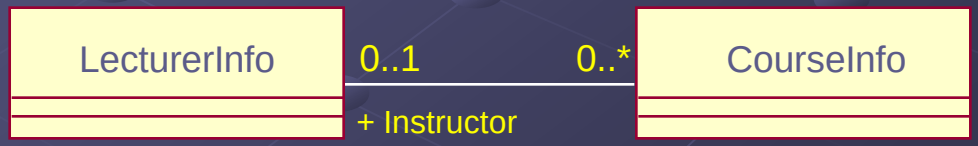


Thiết kế bội số quan hệ

- Multiplicity = 1, hoặc Multiplicity = 0..1
  - Có thể được thực hiện trực tiếp bằng một giá trị đơn hoặc con trỏ
  - Không cần thiết kế thêm

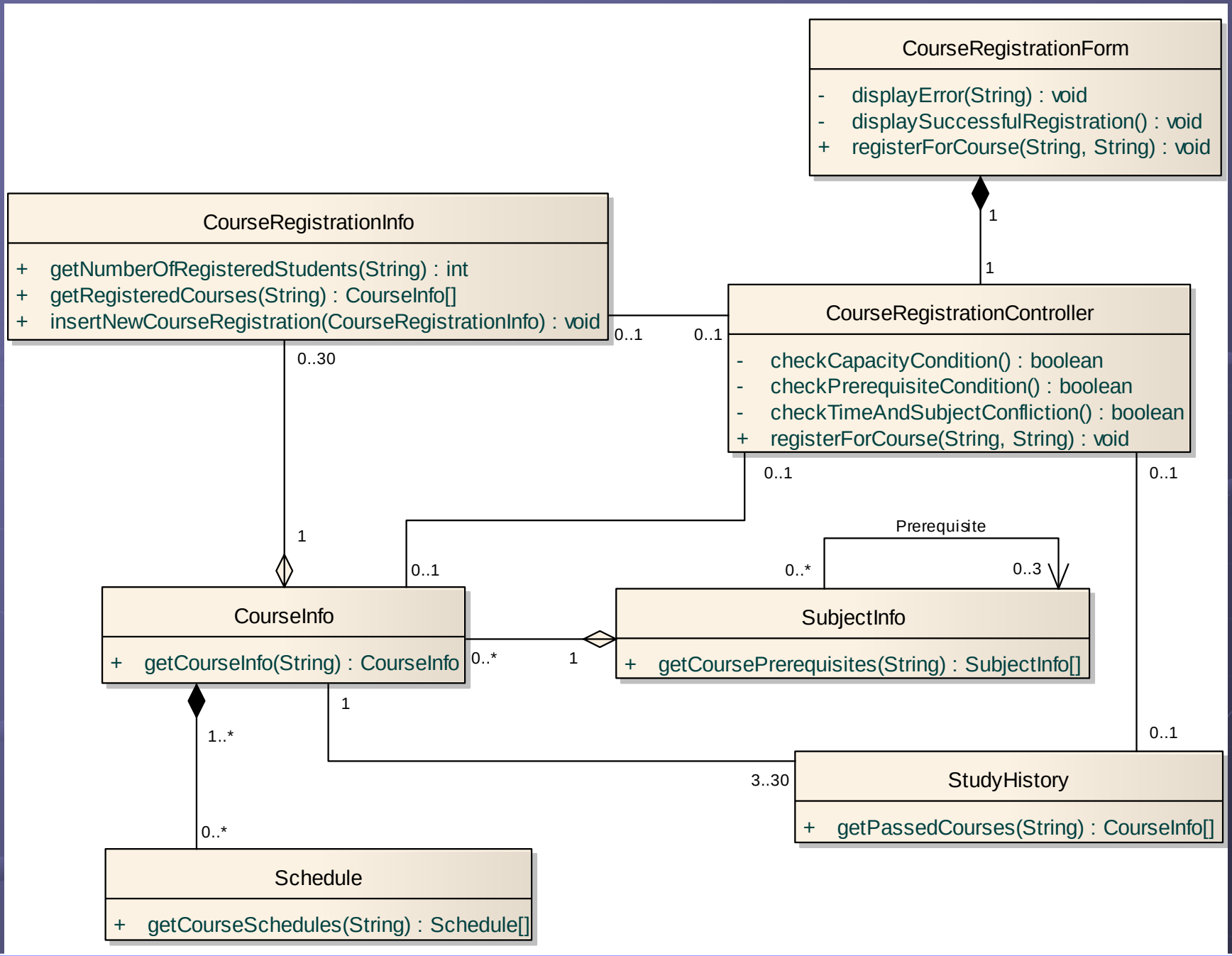
- Multiplicity > 1
  - Không thể sử dụng giá trị đơn hoặc con trỏ
  - Cần thiết kế thêm nữa

Cần một cho CourseInfo



Multiplicity Design: Optionality

- If a link is optional, make sure to include an operation to test for the existence of the link


CourseRegistrationForm
- displayError(String) : void
- displaySuccessfulRegistration() : void
+ registerForCourse(String, String) : void
1
1
CourseRegistrationInfo
+ getNumberOfRegisteredStudents(String) : int
+ getRegisteredCourses(String) : CourseInfo[]
+ insertNewCourseRegistration(CourseRegistrationInfo) : void
0..1
0..1
CourseRegistrationController
- checkCapacityCondition() : boolean
- checkPrerequisiteCondition() : boolean
- checkTimeAndSubjectConfliction() : boolean
+ registerForCourse(String, String) : void
0..30
0..1
0..1
1
0..1
Prerequisite
0..*
0..3
CourseInfo
+ getCourseInfo(String) : CourseInfo
SubjectInfo
+ getCoursePrerequisites(String) : SubjectInfo[]
0..*
1
1
1..*
0..1
3..30
StudyHistory
+ getPassedCourses(String) : CourseInfo[]
0..*
Schedule
+ getCourseSchedules(String) : Schedule[]

Chương 5. Thiết kế lớp

1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

4 nh

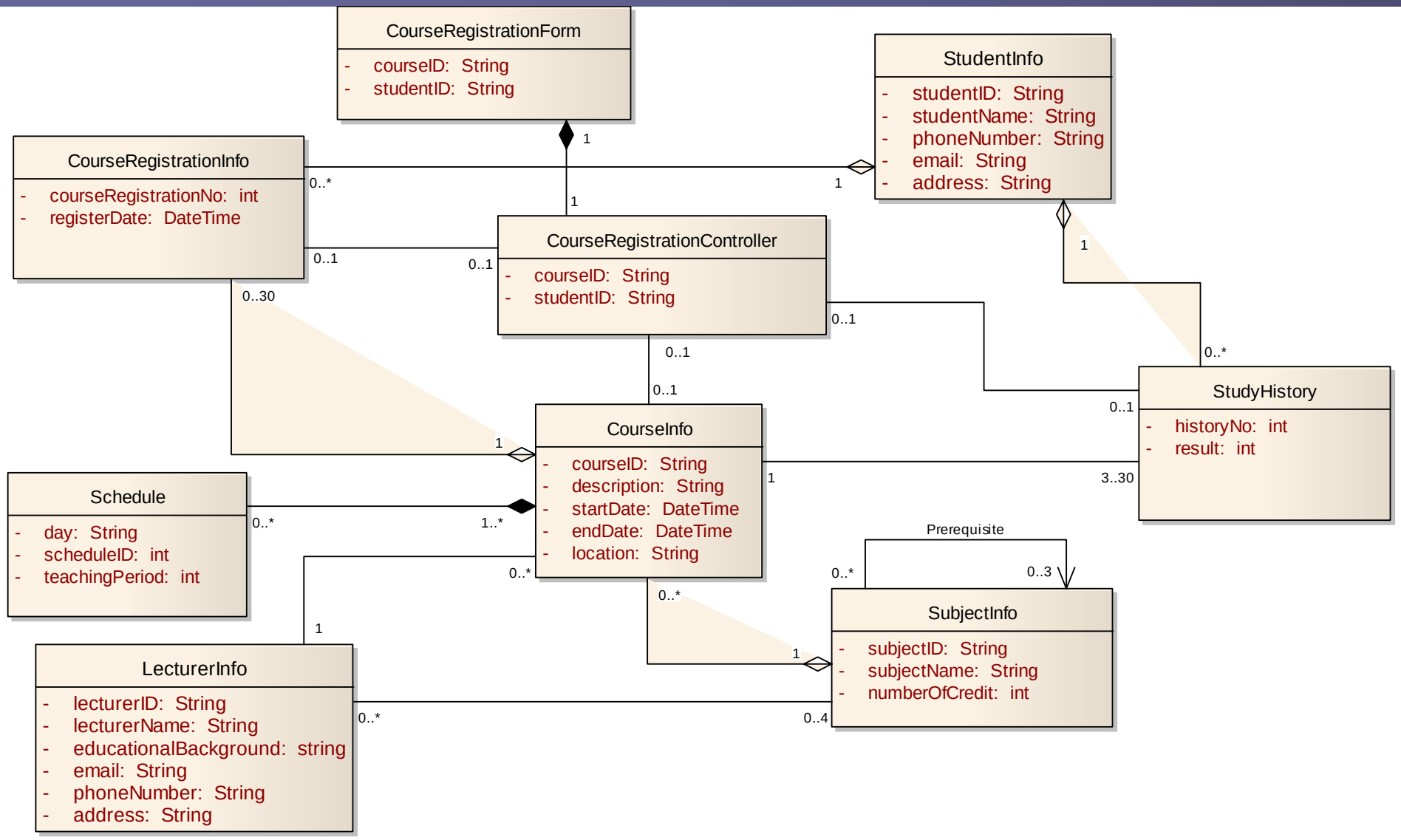
- Xem xét các mô tả phương thức
- Xem xét các trạng thái
- Xem xét bất kỳ thông tin nào mà lớp đó cần lưu giữ, duy trì.

nh

- Chỉ ra tên, kiểu và giá trị mặc định nếu có
  - `attributeName : Type = Default`
- Tuân theo quy ước đặt tên của ngôn ngữ cài đặt và của dự án.
- Kiểu (type) nên là kiểu dữ liệu cơ bản trong ngôn ngữ thực thi
  - Kiểu dữ liệu có sẵn, kiểu dữ liệu người dùng định nghĩa, hoặc lớp tự định nghĩa.
- Xác định phạm vi truy cập
  - Public: +
  - Private: -
  - Protected: #

Các thuộc tính dẫn xuất (derived)

- Thuộc tính dẫn xuất là gì?
  - Là thuộc tính có giá trị có thể được tính toán dựa trên các thuộc tính khác.
- Khi nào thì sử dụng?
  - Khi không đủ thời gian để tính toán lại giá trị mỗi khi cần
  - Khi bạn phải cân đối giữa hiệu năng thời gian chạy với bộ nhớ yêu cầu.


CourseRegistrationForm
- courseID: String
- studentID: String
StudentInfo
- studentID: String
- studentName: String
- phoneNumber: String
- email: String
- address: String
CourseRegistrationInfo
- courseRegistrationNo: int
- registerDate: DateTime
CourseRegistrationController
- courseID: String
- studentID: String
StudyHistory
- historyNo: int
- result: int
CourseInfo
- courseID: String
- description: String
- startDate: DateTime
- endDate: DateTime
- location: String
Schedule
- day: String
- scheduleID: int
- teachingPeriod: int
Prerequisite
SubjectInfo
- subjectID: String
- subjectName: String
- numberOfCredit: int
LecturerInfo
- lecturerID: String
- lecturerName: String
- educationalBackground: string
- email: String
- phoneNumber: String
- address: String
1
0..*
0..1
0..30
1..*
0..3
0..4

Nội dung

1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

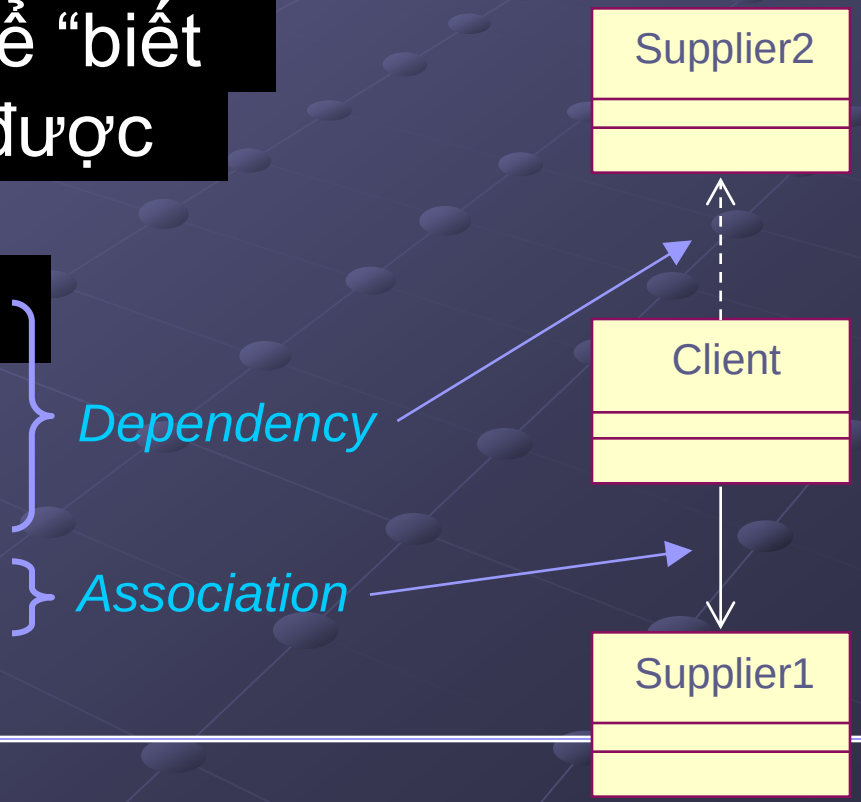
5 c (Dependency)

- Một phụ thuộc là gì?
  - Là mối quan hệ ngữ nghĩa giữa hai đối tượng, trong đó một sự thay đổi trong supplier có thể gây ra thay đổi cho client.

- Mục đ
  - Xác [illegible] ociation hoặc aggregation) không cần đến.
- Cần xem xét:
  - Cái gì làm cho supplier có thể được nhìn thấy client?

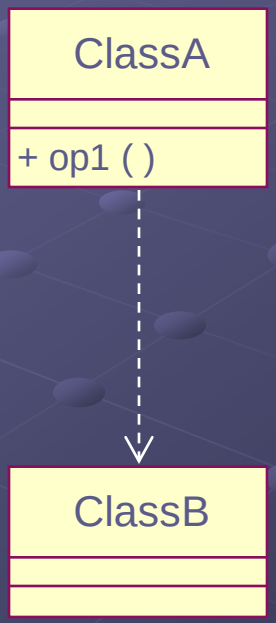
Liên kết và Phụ thuộc

- Kết hợp là mối quan hệ cấu trúc
- Phụ thuộc là mối quan hệ phi-cấu trúc
- Để các đối tượng có thể "biết lẫn nhau", chúng phải được nhìn thấy
  - Local variable reference
  - Parameter reference
  - Global reference
  - Field reference



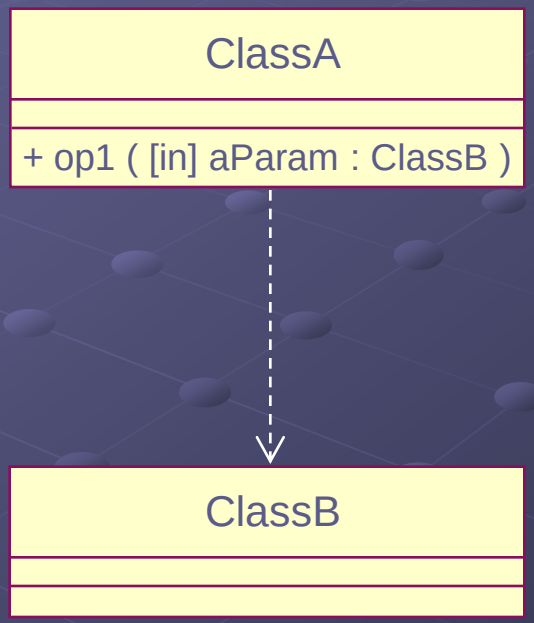
Phạm vi biến địa phương

- Thao tác op1() chứa một biến địa phương của ClassB



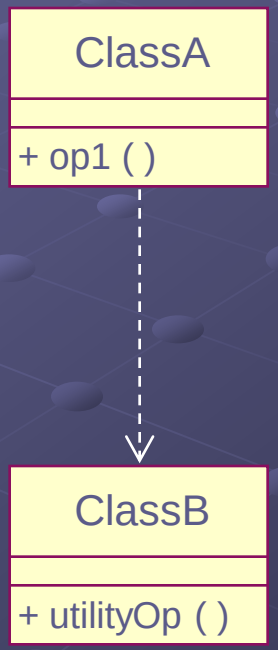
Phạm vi tham số

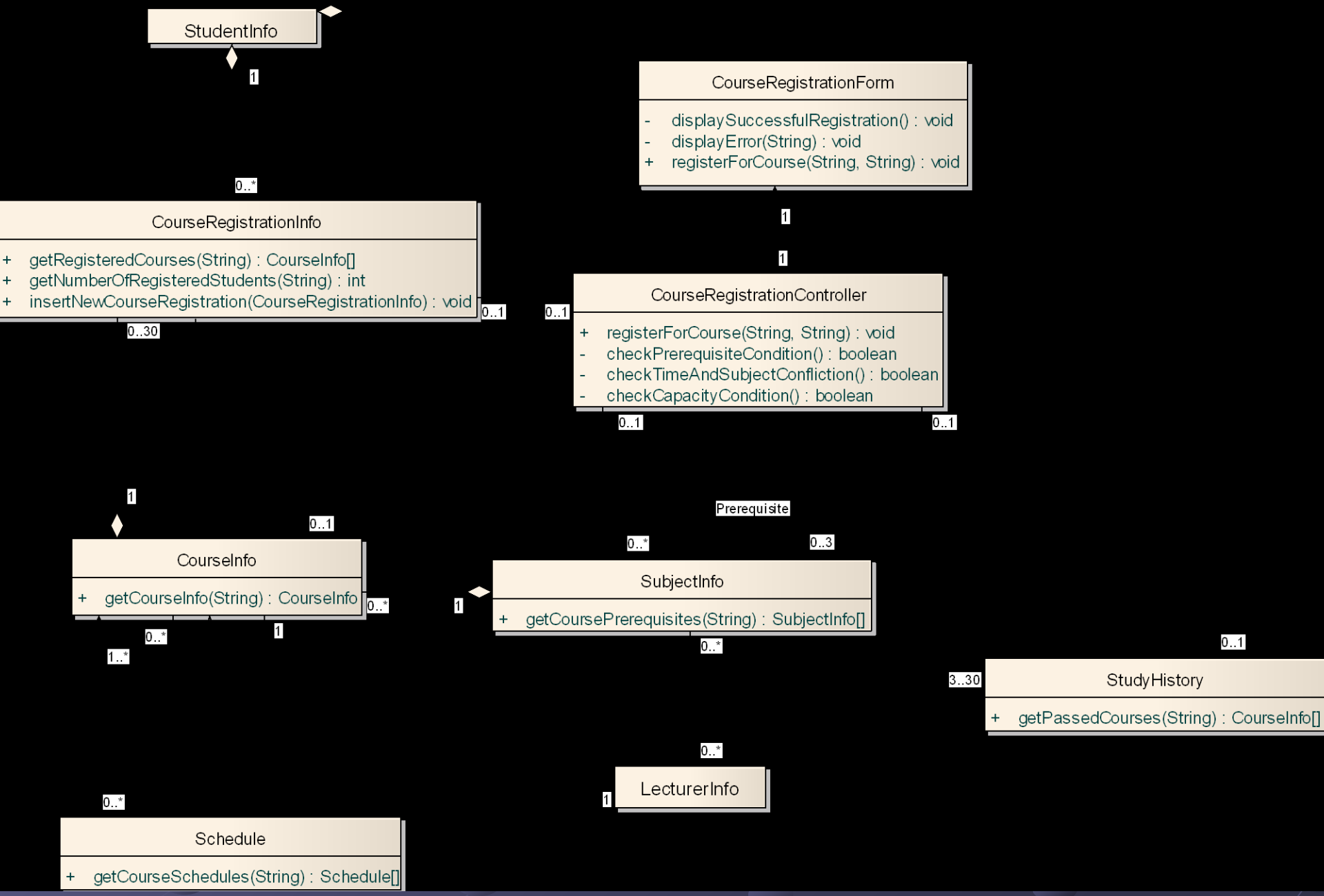
- Thể hiện của ClassB được truyền tham số đến thể hiện của ClassA.



Phạm vi toàn cục

- Thể hiện ClassUtility có thể nhìn thấy vì nó là toàn cục



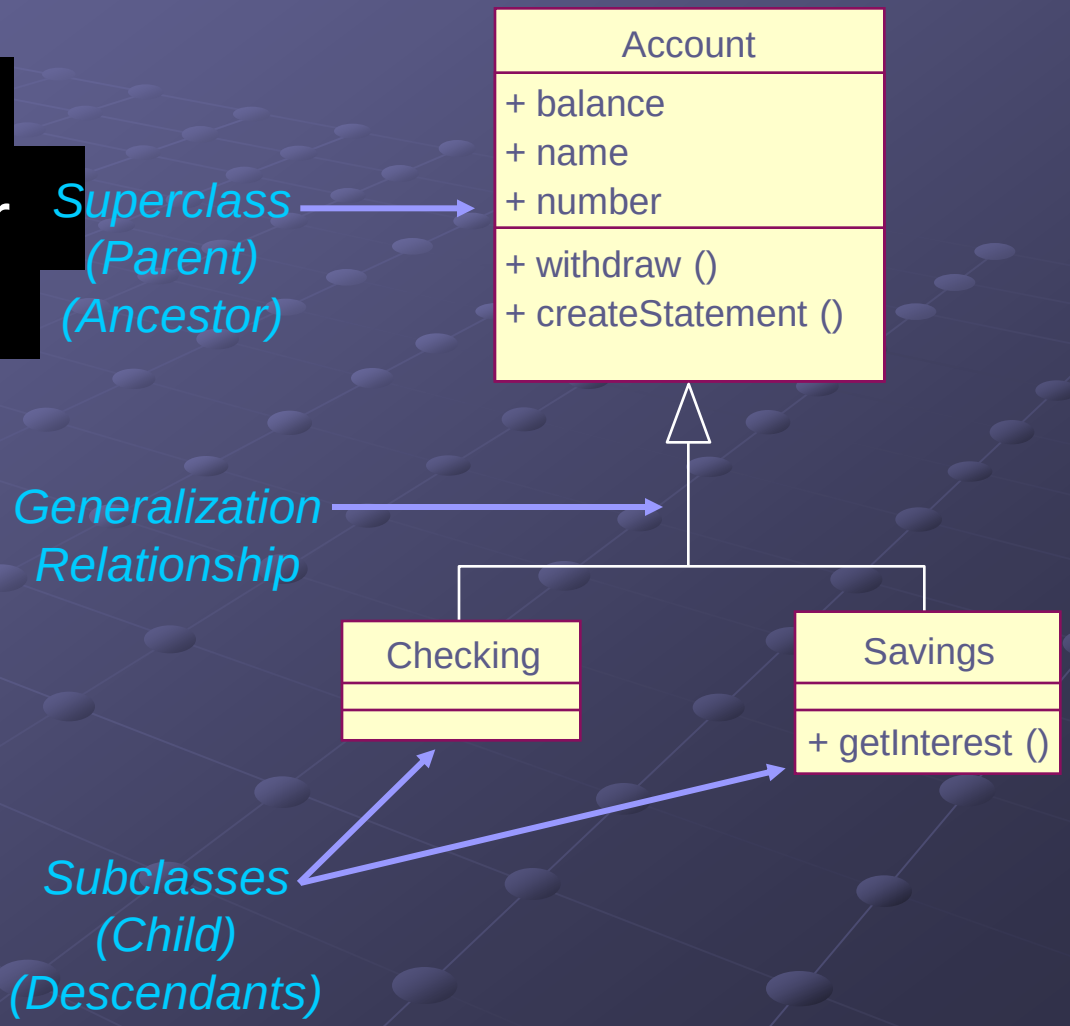

StudentInfo
1
CourseRegistrationForm
- displaySuccessfulRegistration() : void
- displayError(String) : void
+ registerForCourse(String, String) : void
0..*
CourseRegistrationInfo
+ getRegisteredCourses(String) : CourseInfo[]
+ getNumberOfRegisteredStudents(String) : int
+ insertNewCourseRegistration(CourseRegistrationInfo) : void
0..1
0..30
1
1
0..1
CourseRegistrationController
+ registerForCourse(String, String) : void
- checkPrerequisiteCondition() : boolean
- checkTimeAndSubjectConfliction() : boolean
- checkCapacityCondition() : boolean
0..1
0..1
1
0..1
CourseInfo
+ getCourseInfo(String) : CourseInfo
0..*
0..*
1
1..*
Prerequisite
0..*
0..3
SubjectInfo
+ getCoursePrerequisites(String) : SubjectInfo[]
1
0..*
0..1
3..30
StudyHistory
+ getPassedCourses(String) : CourseInfo[]
0..*
1
LecturerInfo
0..*
Schedule
+ getCourseSchedules(String) : Schedule[]

Nội dung

1. c (Operation)
2. c (Method)
3. Xác định các liên kết (Association)
4. nh (Attribute)
5. c (Dependency)
6. Xác định tổng quát hóa (Generalization)

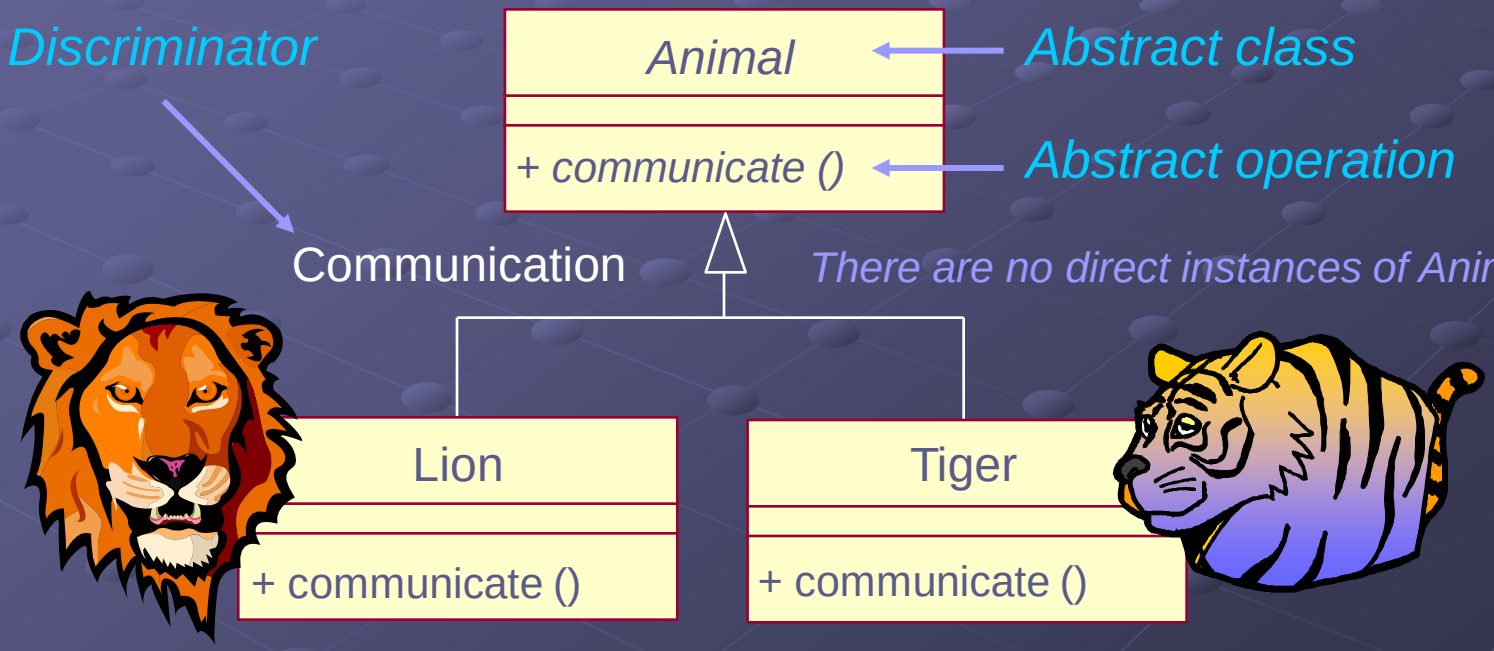
6. Generalization

- One class shares the structure and/or behavior of one or more classes
- “Is a kind of” relationship
- In Analysis, use sparingly



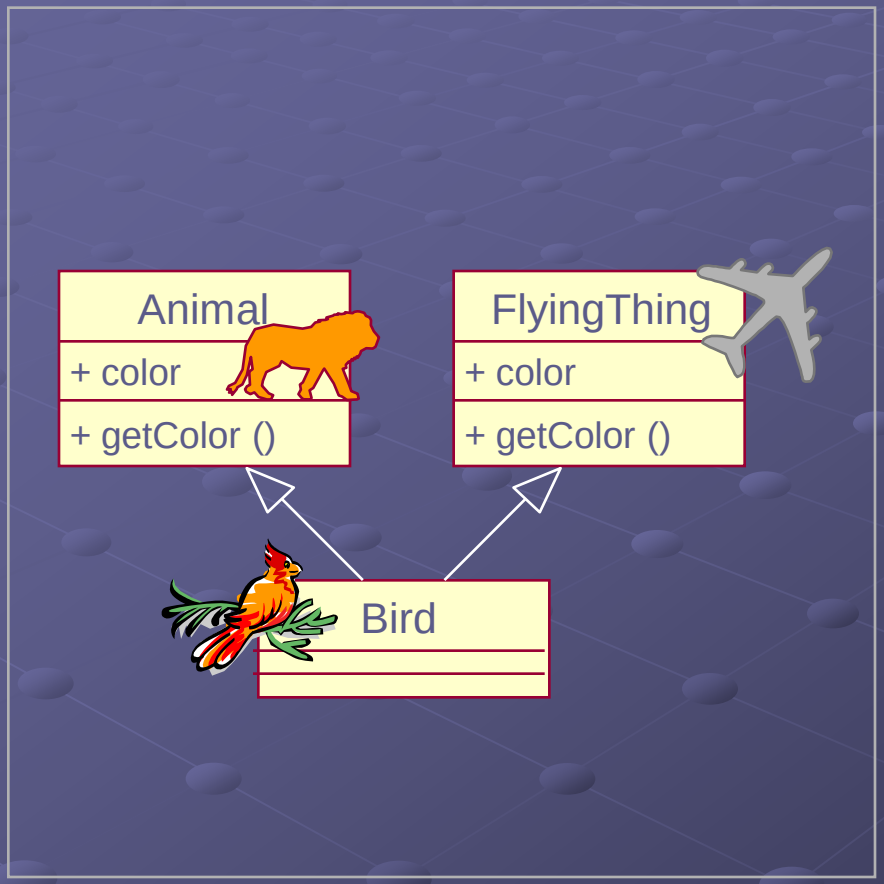
Abstract and Concrete Classes

- Abstract classes cannot have any objects
- Concrete classes can have objects

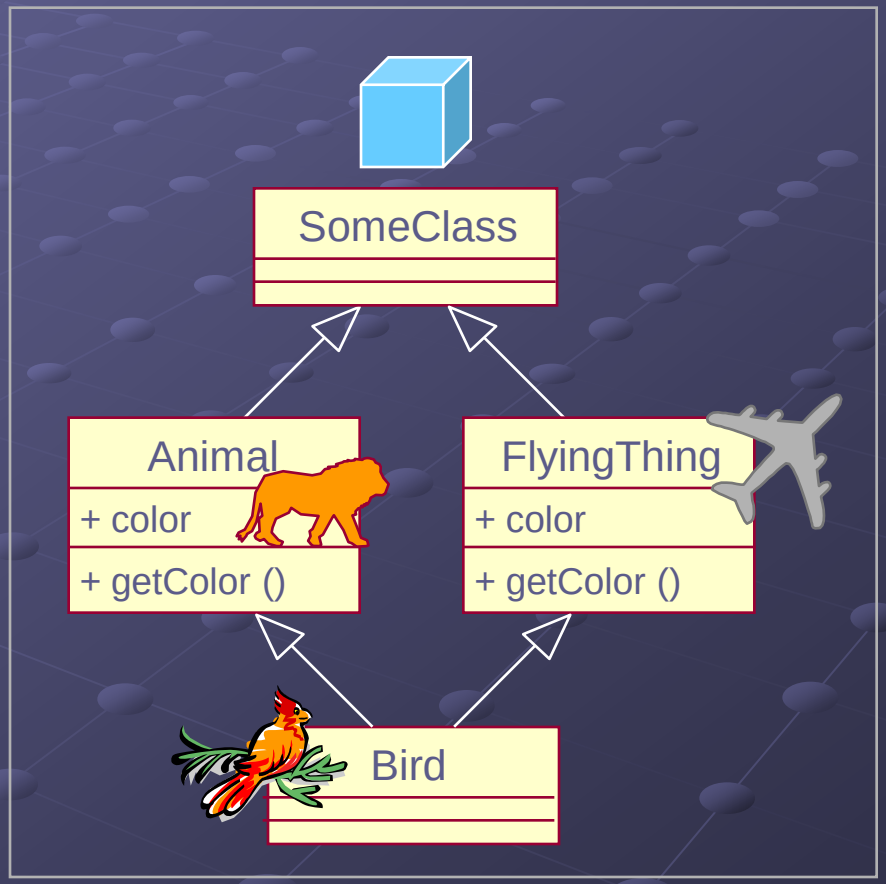


Multiple Inheritance: Problems

Name clashes on attributes or operations



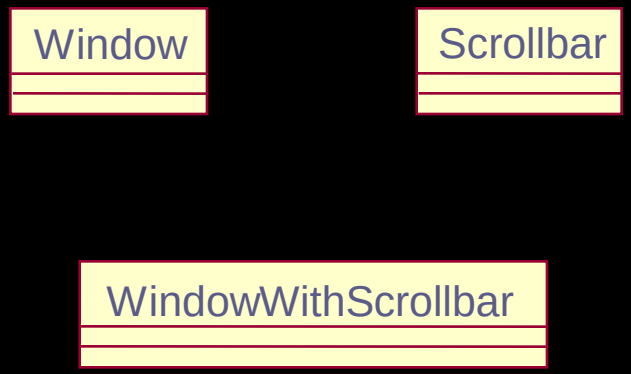
Repeated inheritance



Resolution of these problems is implementation-dependent.

Generalization vs. Aggregation

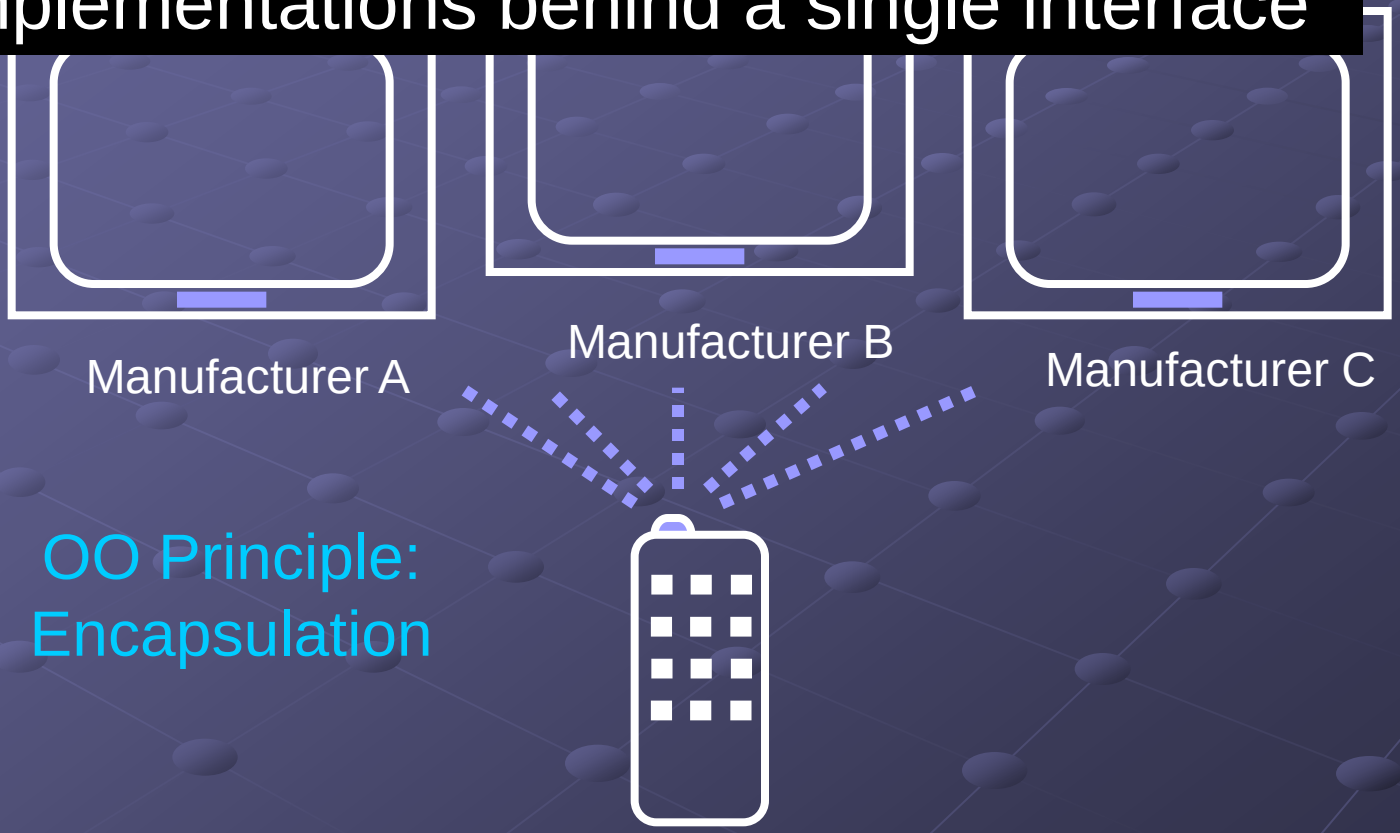
- Generalization and aggregation are often confused
  - Generalization represents an “is a” or “kind-of” relationship
  - Aggregation represents a “part-of” relationship



Is this correct?

Review: What Is Polymorphism?

- The ability to hide many different implementations behind a single interface



OO Principle: Encapsulation

Generalization: Implement Polymorphism

Animal

+ communicate ()

Lion

+ communicate ()

Tiger

+ communicate ()

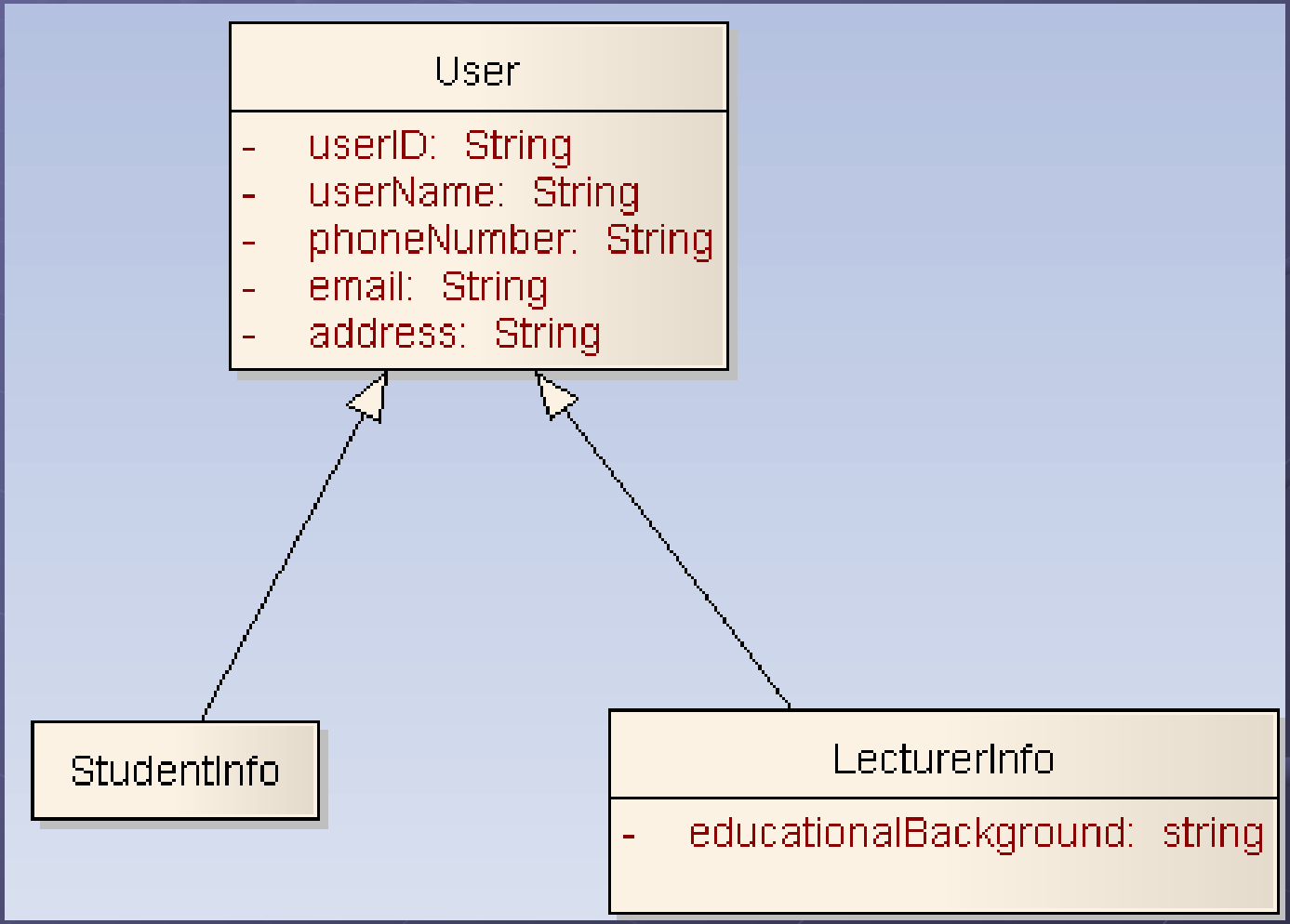
Without Polymorphism

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

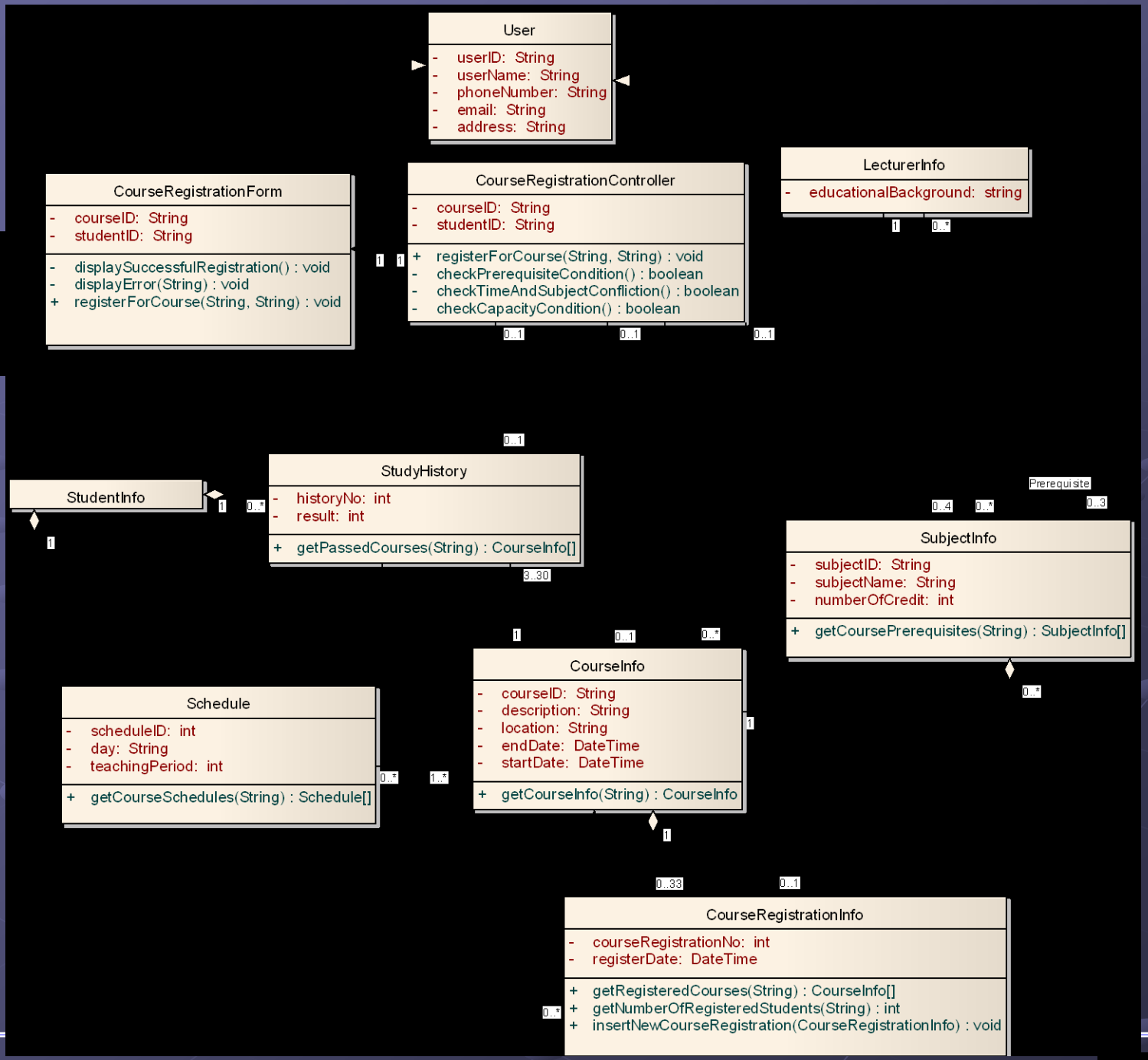
With Polymorphism

Animal communicate

Ví dụ Generalization



Biểu đồ lớp đầy đủ



Tổng kết

- Các mô hình thiết kế được xây dựng trực tiếp từ các mô hình phân tích
  - Là hình thức chi tiết hóa từ các lớp trừu tượng hóa trong mô hình phân tích
    - Ánh xạ 1-1 cho những lớp phân tích đơn giản
  - Ánhxạ thành nhiều lớp thiết kế nếu lớp phân tích đó quá phức tạp
- Lớp phân tích có mức độ phức tạp cao có thể được phát triển thành hệ thống con (subsystem)
  - Sử dụng các giao diện
  - Đảm bảo hệ thống con có tính độc lập tối đa với các thành phần còn lại
- Tìm cách sử dụng lại các hệ thống con, gói hoặc các thư viện có sẵn

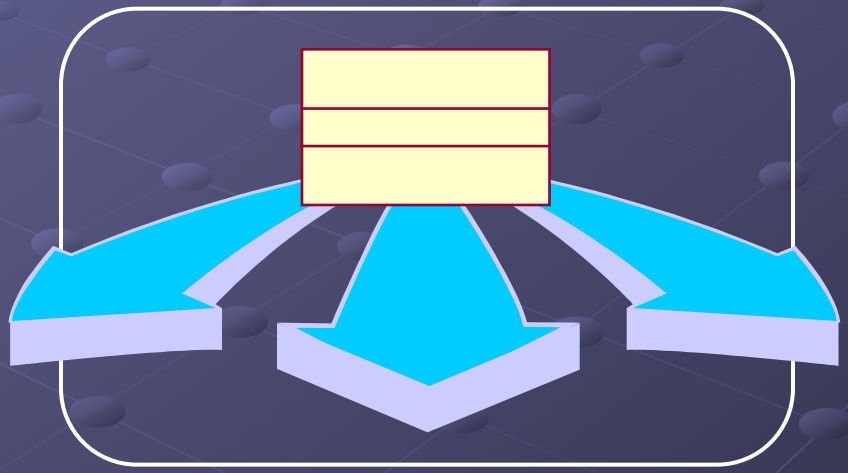
BỘ MÔN CÔNG NGHỆ PHẦN MỀM
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

OBJECT-ORIENTED ANALYSIS AND DESIGN WITH UML 2.0

Bài 10. Thiết kế CSDL

Nội dung

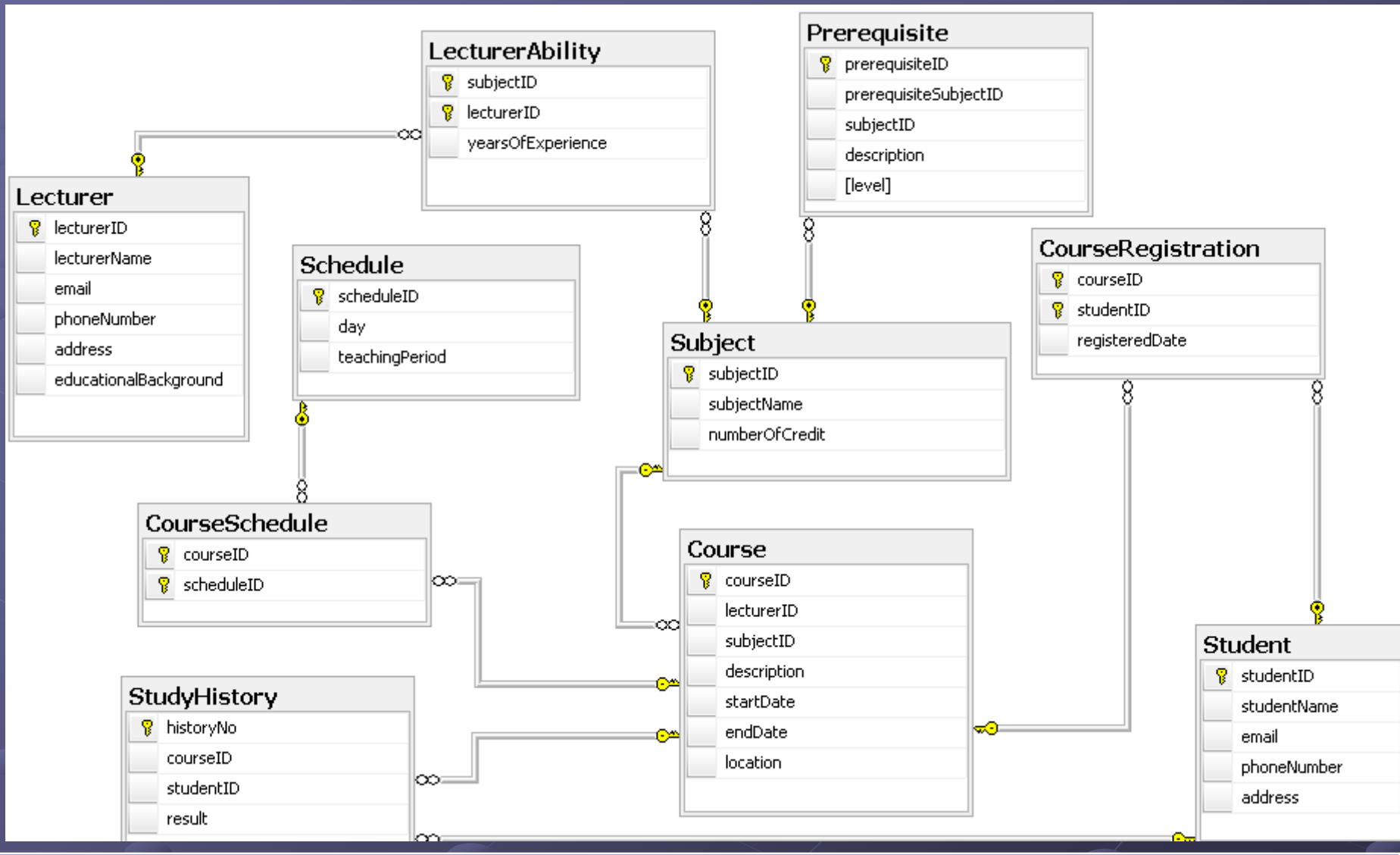
1. Thiết kế mức vật lý cho biểu đồ E-R
2. Phân phối hành vi của lớp vào cơ sở dữ liệu



1. Thiết kế mức vật lý cho biểu đồ E-R

- Thiết kế trên một hệ quản trị CSDL nào đó
  - Các thực thể → Các bảng
  - Mối liên hệ → Ràng buộc tham chiếu
  - Các kiểu dữ liệu của lớp → Các kiểu dữ liệu của hệ quản trị CSDL
- Thiết kế biểu đồ CSDL mức vật lý

Ví dụ


LecturerAbility
subjectID
lecturerID
yearsOfExperience
Prerequisite
prerequisiteID
prerequisiteSubjectID
subjectID
description
[level]
Lecturer
lecturerID
lecturerName
email
phoneNumber
address
educationalBackground
Schedule
scheduleID
day
teachingPeriod
CourseRegistration
courseID
studentID
registeredDate
Subject
subjectID
subjectName
numberOfCredit
CourseSchedule
courseID
scheduleID
Course
courseID
lecturerID
subjectID
description
startDate
endDate
location
Student
studentID
studentName
email
phoneNumber
address
StudyHistory
historyNo
courseID
studentID
result

2. Phân phối hành vi của lớp vào cơ sở dữ liệu

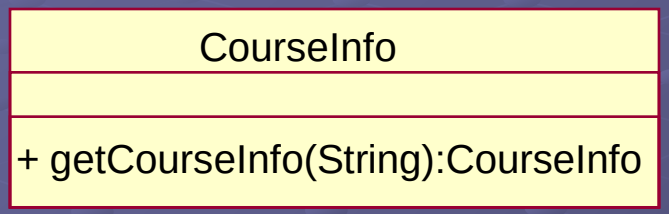
- Quyết định các thao tác nào cần được thực thi thành một thủ tục lưu trữ
- Cần xem xét:
  - Các thao tác giải quyết với dữ liệu persistent
  - Các thao tác trong đó có câu truy vấn liên quan đến tính toán
  - Các thao tác cần truy cập dữ liệu để kiểm tra tính hợp lệ của dữ liệu

Thủ tục lưu trữ là gì?

- Là đoạn mã có thể thực thi được chạy trong RDBMS
- Có 2 loại thủ tục được lưu trữ:
  - Thủ tục (procedure): Được thực thi rõ ràng bởi ứng dụng
  - Trigger: Được gọi ngầm định khi một sự kiện nào đó của CSDL xảy ra

Ví dụ

Class



Candidate Operations

- getCourseInfo

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VỚI UML 2.0

Bài 11. Thiết kế ngoài

Nội dung

1. Thiết kế giao diện người dùng (Screen design)
2. Thiết kế biểu đồ chuyển giữa các giao diện người dùng (Display transition diagram)
3. Đặc tả giao diện người dùng (Screen specification)

Từ use case

- Dựa vào các use case và các lớp biên là giao diện với người dùng ánh xạ thành các screen để thiết kế.
- Dựa vào các dữ liệu vào, ra được xác định trong giai đoạn đặc tả use case.

Screen design

- Chuẩn hóa cấu trúc, cách thức tổ chức, hình thức biểu diễn của các thành phần trên màn hình.
- Có thể sử dụng Front page (HTML), MS Word,… để thiết kế giao diện

Chuẩn hóa

- Display
  - Kích thước thực tế
  - Độ phân giải, số lượng màu hỗ trợ
- Screen: Được chia thành các đối tượng hiển thị gọi là Cửa sổ (Window)
  - Vị trí của các nút chuẩn (OK, Cancel, Register, Search,…)
  - Vị trí hiển thị các message,…
  - Title và menu của màn hình
  - Sự thống nhất trong việc biểu diễn các ký tự
  - Biểu diễn các câu vào chi tiết các item
  - Phối màu

Chuẩn hóa

- Control
  - Kiểu, kích thước, màu sắc và các ký tự hiển thị
  - Quy trình kiểm tra đầu vào
  - Trình tự di chuyển focus (ví dụ định nghĩa trình tự tab)
- Menu
  - Thiết kế menu với sự cân nhắc về chuẩn hóa của các thành phần chung của màn hình
- Nhập dữ liệu đầu vào tự bàn phím
  - Chú ý sự nhất quán giữa các phím tắt, tab

Chuẩn hóa

- Message
  - Xác định các message được hiển thị như thế nào trong thời gian thực hiện
- Error
  - Đưa ra quy trình xử lý lỗi thống nhất khi gặp sự cố
- Help
  - Phát triển chi tiết các thông tin trợ giúp: Phù hợp, nhất quán về thuật ngữ, mô tả,…

Nội dung

1. Thiết kế giao diện người dùng (Screen design)
2. Thiết kế biểu đồ chuyển giữa các giao diện người dùng (Display transition diagram)
3. Đặc tả giao diện người dùng (Screen specification)

Display transition diagram

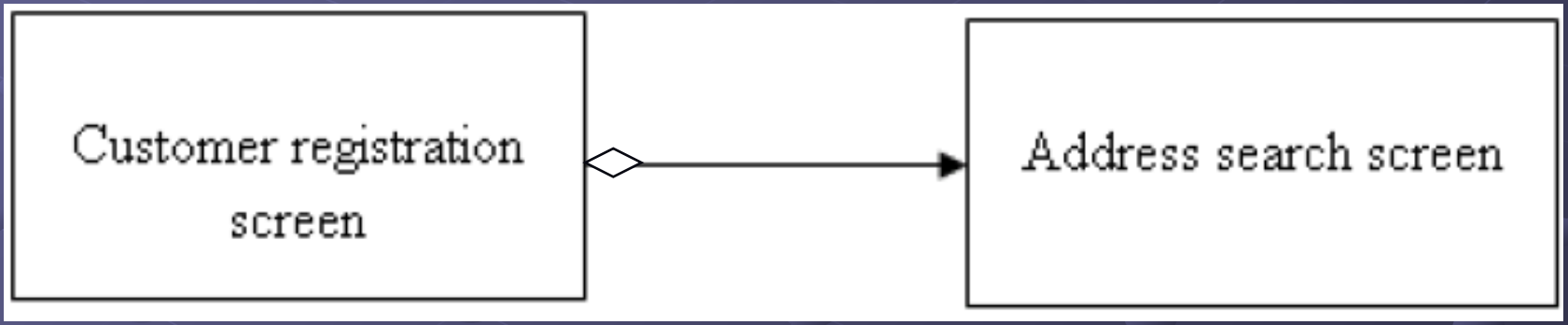
- Tổng hợp mối liên hệ giữa các màn hình
  - Phân loại các màn hình theo 4 mẫu chuẩn
  - Liên kết các màn hình theo sự phân loại trên

4 transition patterns

- 1. Simple screen transition: A conventional simple transition

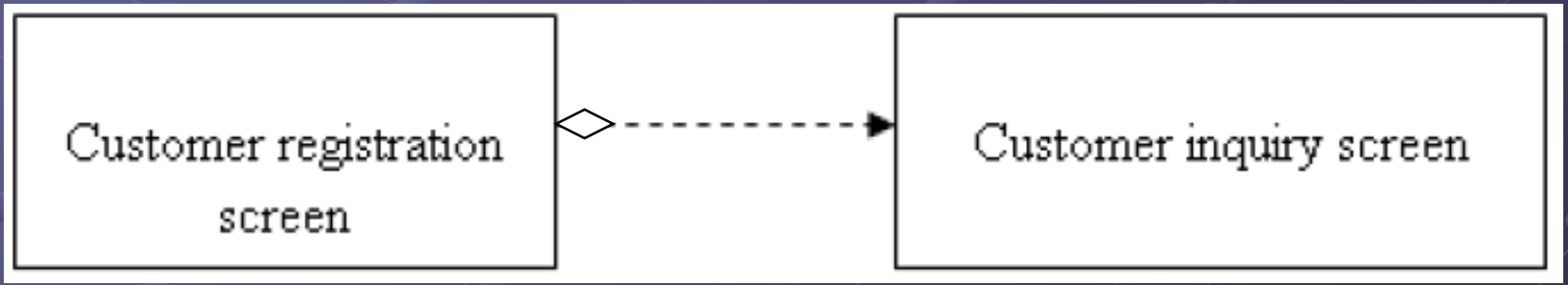
4 transition patterns

- 2. Transition to a child: Khi màn hình con hiển thị trên màn hình cha, màn hình cha ở dưới ko thao tác được



4 transition patterns

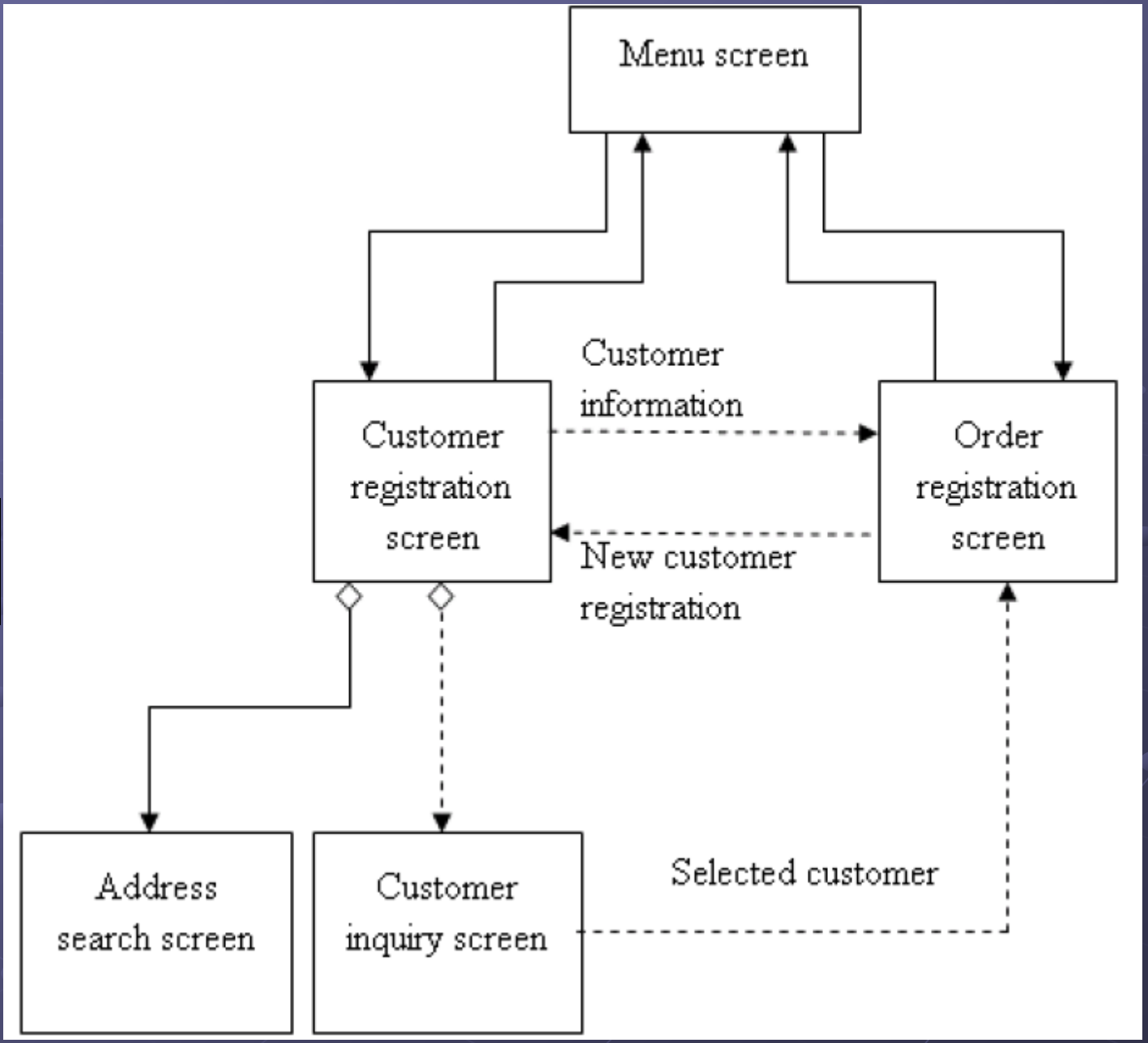
- 3. Transition to an independent child screen: Chuyển đến một màn hình pop-up từ màn hình cha, màn hình con và màn hình cha đều cùng thao tác được.



4 transition patterns

- 4. Transition to an independent screen: Khởi tạo một màn hình độc lập mới.

Liên kết các màn hình



Nội dung

1. Thiết kế giao diện người dùng (Screen design)
2. Thiết kế biểu đồ chuyển giữa các giao diện người dùng (Display transition diagram)
3. Đặc tả giao diện người dùng (Screen specification)

Screen specification

- Đặc tả chi tiết cho các thành phần trên màn hình

Screen name	Order entry			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Transaction category	3	Numeral	Green (blink)	Error items blink.
Customer code	5	Numeral	Green (blink)	Error items blink.
Customer name	30	Character	White	15 characters, left-justified
Product code	8	Numeral	Green (blink)	Error items blink.
Product name	22	Character	White	11 characters, left-justified
Quantity	6	Numeral	Green (blink)	Error items blink.
Unit price	7	Numeral	White	
Amount	9	Numeral	White	
Quantity in stock	10	Numeral, special character	White	Displayed in the format of ZZZ, ZZZ, ZZ9

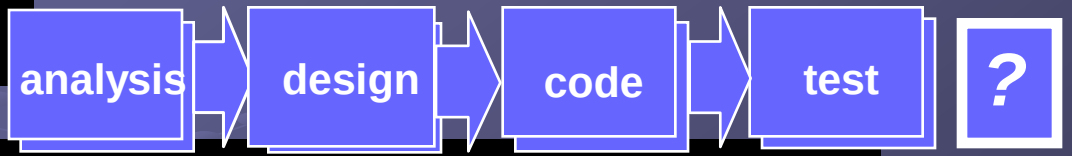
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VỚI UML 2.0

Bài 12. Thiết kế trường hợp kiểm thử

Kiểm thử phần mềm

- Giai đoạn nào?



- Tuy nhiên, kiểm thử cần được thực hiện xuyên suốt các giai đoạn nhằm tăng chất lượng phần mềm
  - Kiểm tra quá trình phân tích thiết kế
  - Xem xét lớp (Class)
  - Kiểm thử đơn vị (Unit Test)
  - Kiểm thử tích hợp (Integration Test)
  - Kiểm thử thẩm định (Validation Tests)

➔ Cần lập kế hoạch kiểm thử

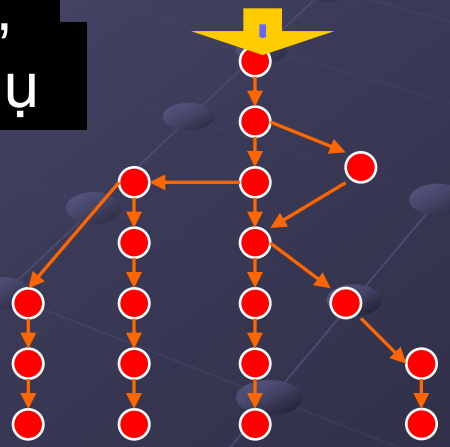
➔ Thiết kế các trường hợp kiểm thử

Use case

- Biểu đồ use case đưa ra tương tác giữa các tác nhân và hệ thống.
- Trong một biểu đồ use case, hệ thống được xem như:
  - Đầu vào (Input)
  - Đầu ra (Output)
  - Các vấn đề về chức năng
- Mục đích của một UC:
  - Tập trung vào giao tiếp
  - Hiểu yêu cầu khách hàng
  - Giúp xác định các biên để đóng gói dữ liệu
  - Tập trung vào “What” chứ không phải là “How”
  - Cung cấp các Prototype Test Cases

Tạo các Test case từ UC

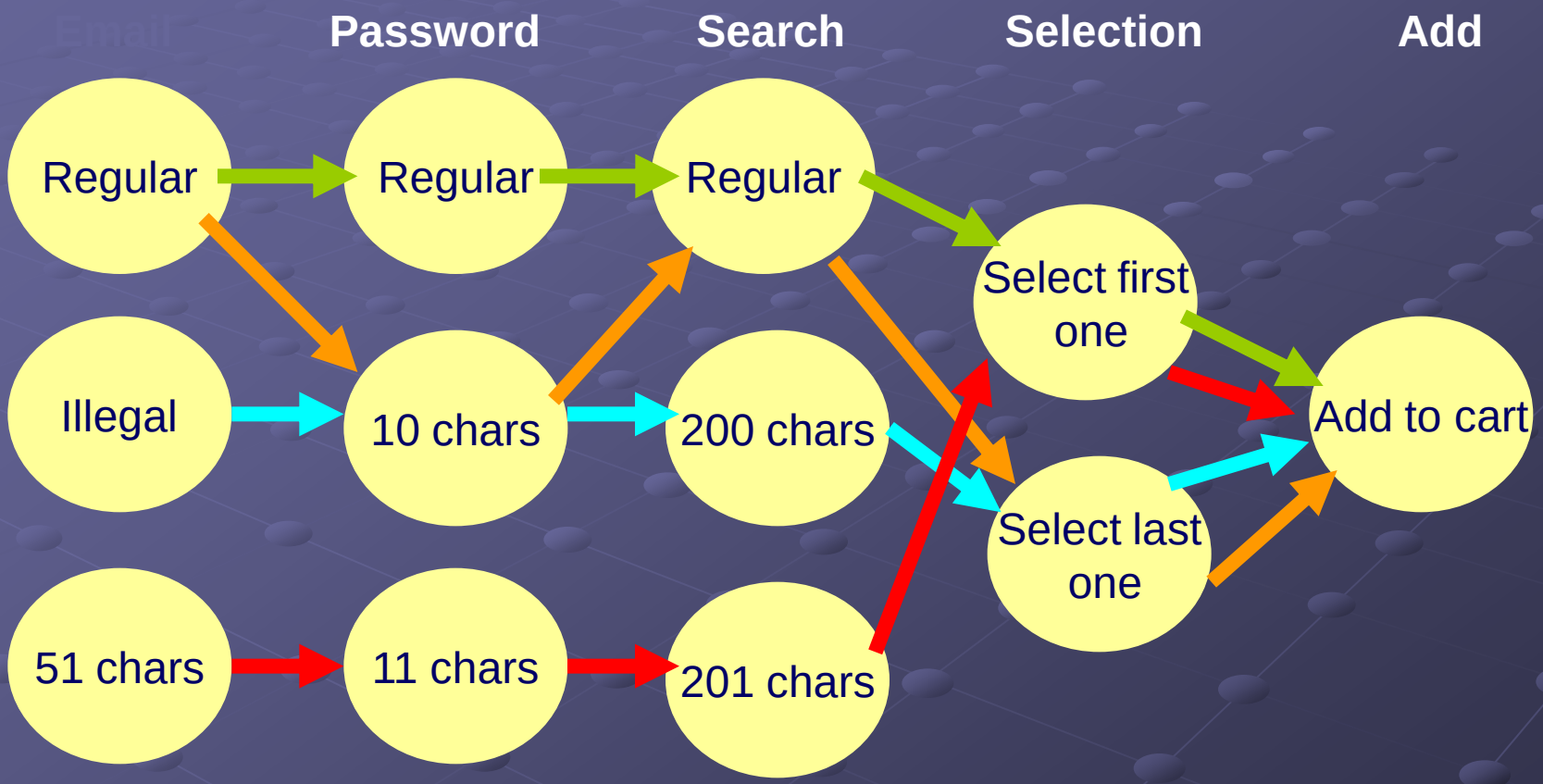
- Xác định tất cả các kịch bản của một use case nào đó
- Các kịch bản phụ cũng cần được vẽ ra trong một biểu đồ cho mỗi hành động
- Tạo kịch bản cho:
  - Luồng cơ bản,
  - Luồng kịch bản bao gồm mỗi luồng phụ,
  - Và một vài sự kết hợp của các luồng phụ
- Tạo ra các vòng lặp vô hạn



Tạo các kịch bản

- Xác định các dữ liệu đầu vào cho mỗi UC
- Xác định các tùy chọn quan trọng khác cho mối dữ liệu (VD: mật khẩu quá dài, quá ngắn, các ký tự đặc biệt…)
- Kết hợp các tùy chọn có thể kiểm thử đưa vào test cases
- Đưa các ví dụ về giá trị cho các đầu vào

USE CASE (4)


Email
Password
Search
Selection
Add
Regular
Regular
Regular
Select first one
Illegal
10 chars
200 chars
Add to cart
Select last one
51 chars
11 chars
201 chars

Ví dụ

- UC Register for course
  - Luồng chính: Đăng ký khóa học thành công → Tạo 1 test case cho luồng chính với các vài dữ liệu hợp lệ
  - Luồng phụ
    - 4a: Sinh viên không đủ điều kiện tiên quyết để đăng ký khóa học
    - 5a: Thời gian hoặc môn học của khóa học đăng ký trùng với các khóa học đã đăng ký
      - Xem xét nhiều kiểu trùng đủ tách ra thành các test case nhỏ hơn
      - Đăng ký môn học đã đăng ký rồi, trong kỳ đó
    - 6a: Đăng ký vào lớp học đã có đủ sinh viên đăng ký

Ví dụ

- UC Register for course
  - Có thể xét thêm trường hợp hai người dùng đồng thời cùng đăng ký một khóa học gần đầy (chỉ thiếu 1 người là đầy)
  - Có thể kết hợp 1 số luồng phụ với nhau (nếu cần) hoặc đưa thêm luồng phát sinh
    - Vừa đăng ký vào lớp đầy, vừa trùng thời gian…
    - Đã đóng đăng ký, liệu 1 sinh viên có đăng ký được nữa không (dù các điều kiện khác đều hợp lệ)