

PHÂN TÍCH THIẾT KẾ HỆ THỐNG (VỚI UML)

Bài 8. Thiết kế hệ thống

► Mục tiêu bài học

- Kiến thức:
 - Các bước thiết kế hệ thống
 - Lựa chọn topo mạng
 - Thiết kế đồng thời và an toàn
 - Phân rã phần mềm
- Kết quả cần đạt được:
 - Hiểu được các kiểu hệ phân tán và thiết kế hệ thống theo các dạng phân tán đó
 - Biết cách phân rã hệ phần mềm thành các hệ thống con

► Nội dung

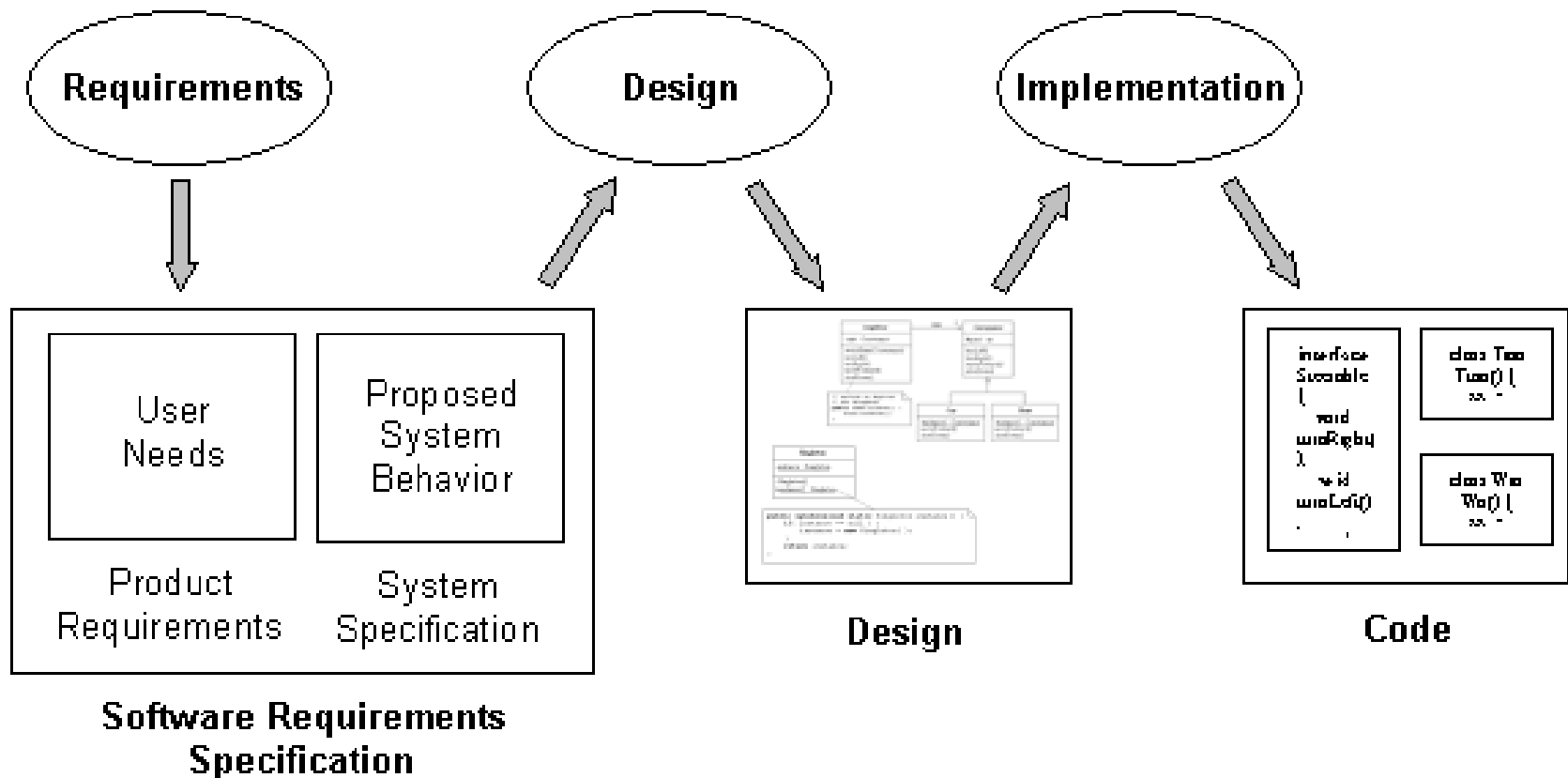
1. Tổng quan quá trình thiết kế
2. Lựa chọn kiến trúc
3. Phân rã các hệ thống con
4. Thiết kế chi tiết

1. Tổng quan quá trình thiết kế

- Mục đích của **pha phân tích** là hình dung ra **nghiệp vụ cần gì**, trong khi mục đích của **pha thiết kế** là quyết định **cách xây dựng hệ thống**.
 - Hay nói một cách khác, phân tích là nhằm trả lời câu hỏi "**cái gì**", còn thiết kế là để trả lời câu hỏi "**như thế nào**".
- Hoạt động chính của pha thiết kế là **tiến hóa tập biểu diễn phân tích thành tập biểu diễn thiết kế**
 - Có thể không quan tâm về việc có chăng sự tương thích giữa những đối tượng phân tích và những đối tượng thiết kế, mà chú trọng hơn vào vấn đề thiết kế có dẫn tới một giải pháp hiệu quả không.
- Đề xuất công nghệ lựa chọn cho thiết kế

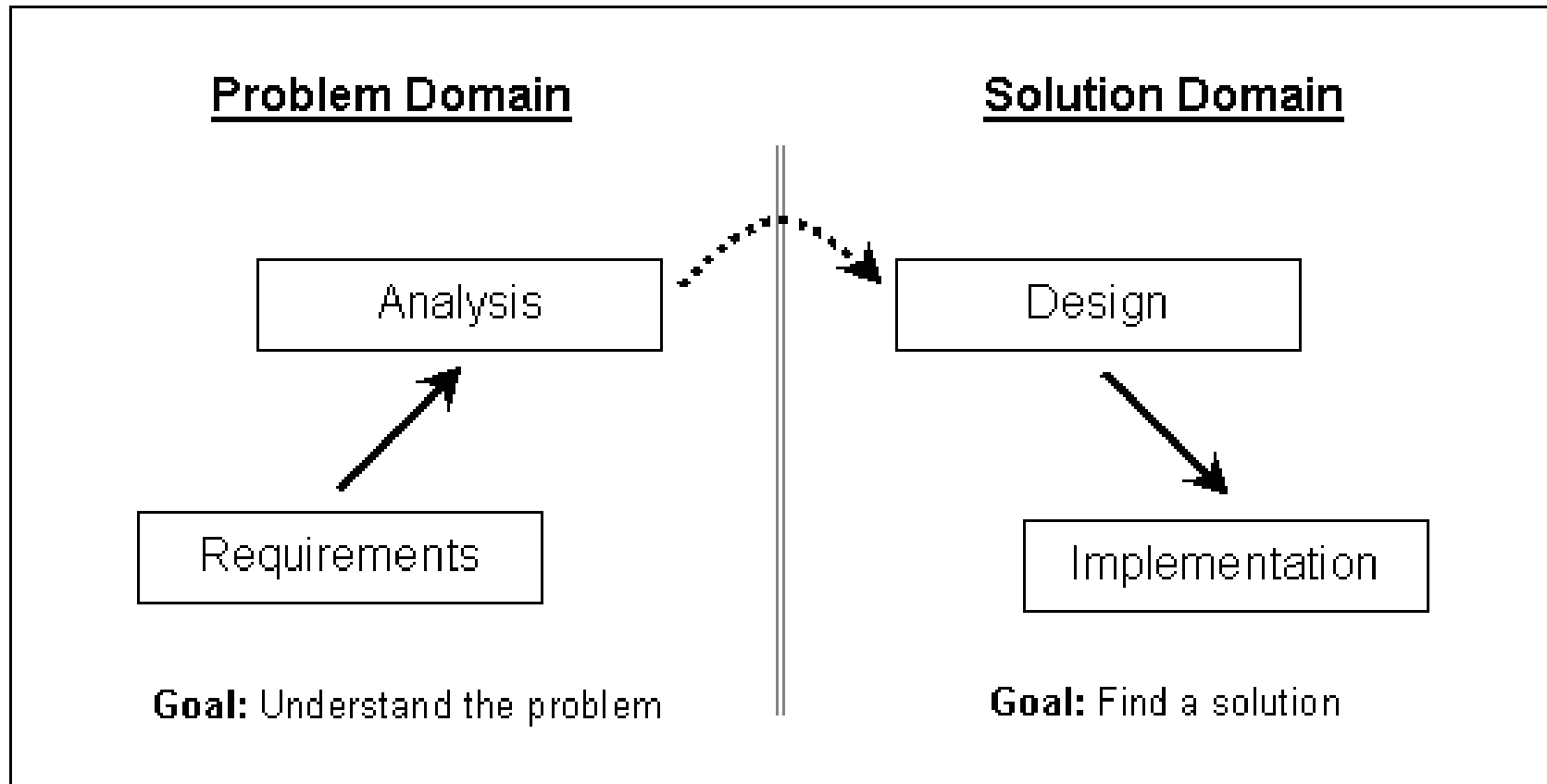
1. Tổng quan quá trình thiết kế

- Design translates system specifications into solution models



1. Tổng quan quá trình thiết kế

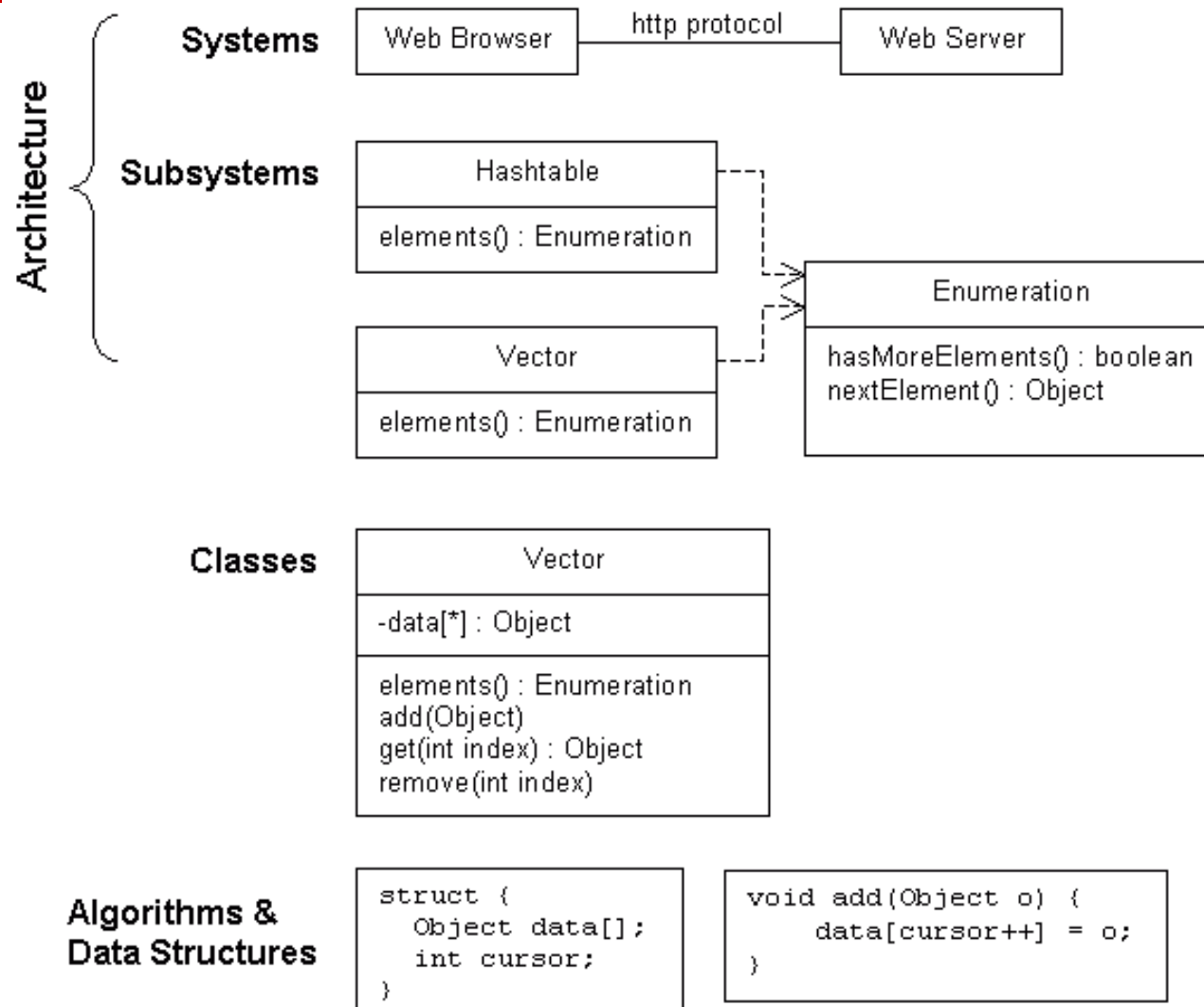
- ❑ Analysis Models are abstractions of the problem domain.
- ❑ Design models are abstractions of the implementation.



1. Tổng quan quá trình thiết kế

- ❑ Các bước trong pha thiết kế gồm hai giai đoạn:
 - **Thiết kế hệ thống** (hay Thiết kế kiến trúc hay Thiết kế tổng thể)
 - ❑ 1. Lựa chọn công nghệ mạng cho hệ thống
 - ❑ 2. Thiết kế tương tranh và an toàn-bảo mật
 - ❑ 3. Phân rã hệ thống thành các hệ thống con
 - ❑ 4. Xây dựng biểu đồ gói
 - **Thiết kế hệ thống con** (hay Thiết kế chi tiết)
 - ❑ 1. Xây dựng biểu đồ lớp thiết kế
 - ❑ 2. Xây dựng biểu đồ tuần tự
 - ❑ 3. Xây dựng lược đồ cơ sở dữ liệu
 - ❑ 4. Thiết kế giao diện

► Design levels

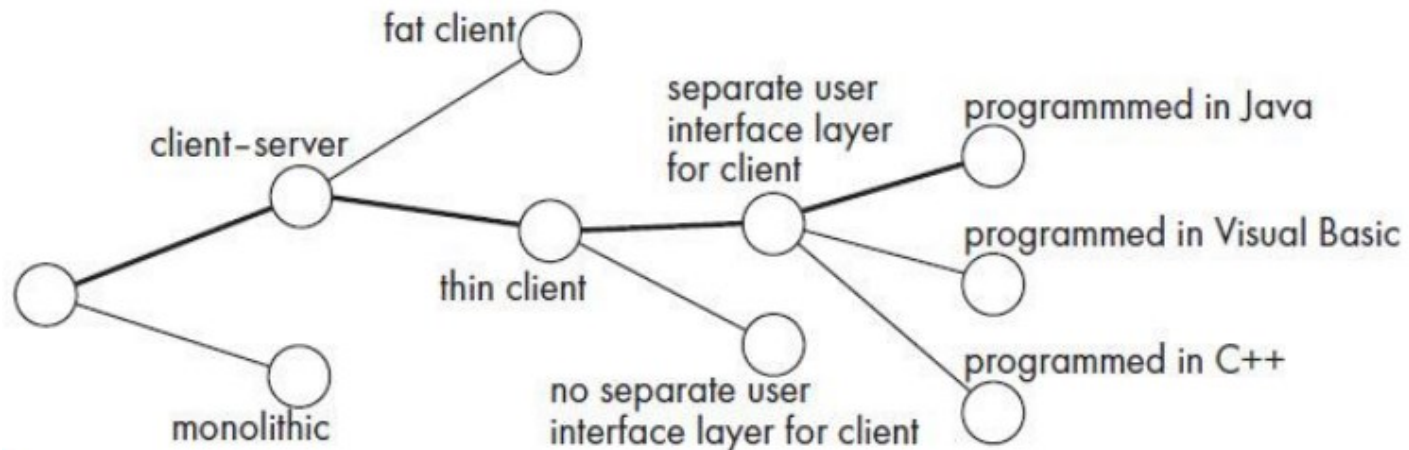


1. Tổng quan quá trình thiết kế

- ❑ Thiết kế tạo ra một biểu diễn hay mô hình của phần mềm, nhưng không giống như mô hình phân tích (tập trung vào việc mô tả dữ liệu, chức năng và hành vi)
- ❑ Mô hình thiết kế cung cấp chi tiết cần thiết để cài đặt phần mềm :
 - Kiến trúc (architecture),
 - Giao diện (interfaces) và
 - Thành phần (component)
- ❑ Sản phẩm công tác (work product):
 - Biểu diễn kiến trúc (Cơ sở dữ liệu, giao tiếp với hệ thống khác...),
 - Giao diện người dùng (GUI),
 - Thành phần (giao tiếp các thành phần, cấu trúc dữ liệu, giải thuật dưới dạng mã giả...)

1. Tổng quan quá trình thiết kế

- ❑ Thiết kế dùng để chỉ ra hệ thống sẽ làm như thế nào (how), các yêu cầu sẽ được hiện thực hóa (realize) ra sao?
- ❑ Kết quả của quá trình thiết kế là **Software Design Document (SDD)**.
- ❑ Thiết kế là một chuỗi các quyết định: các thiết kế thay thế từ những lựa chọn khác nhau

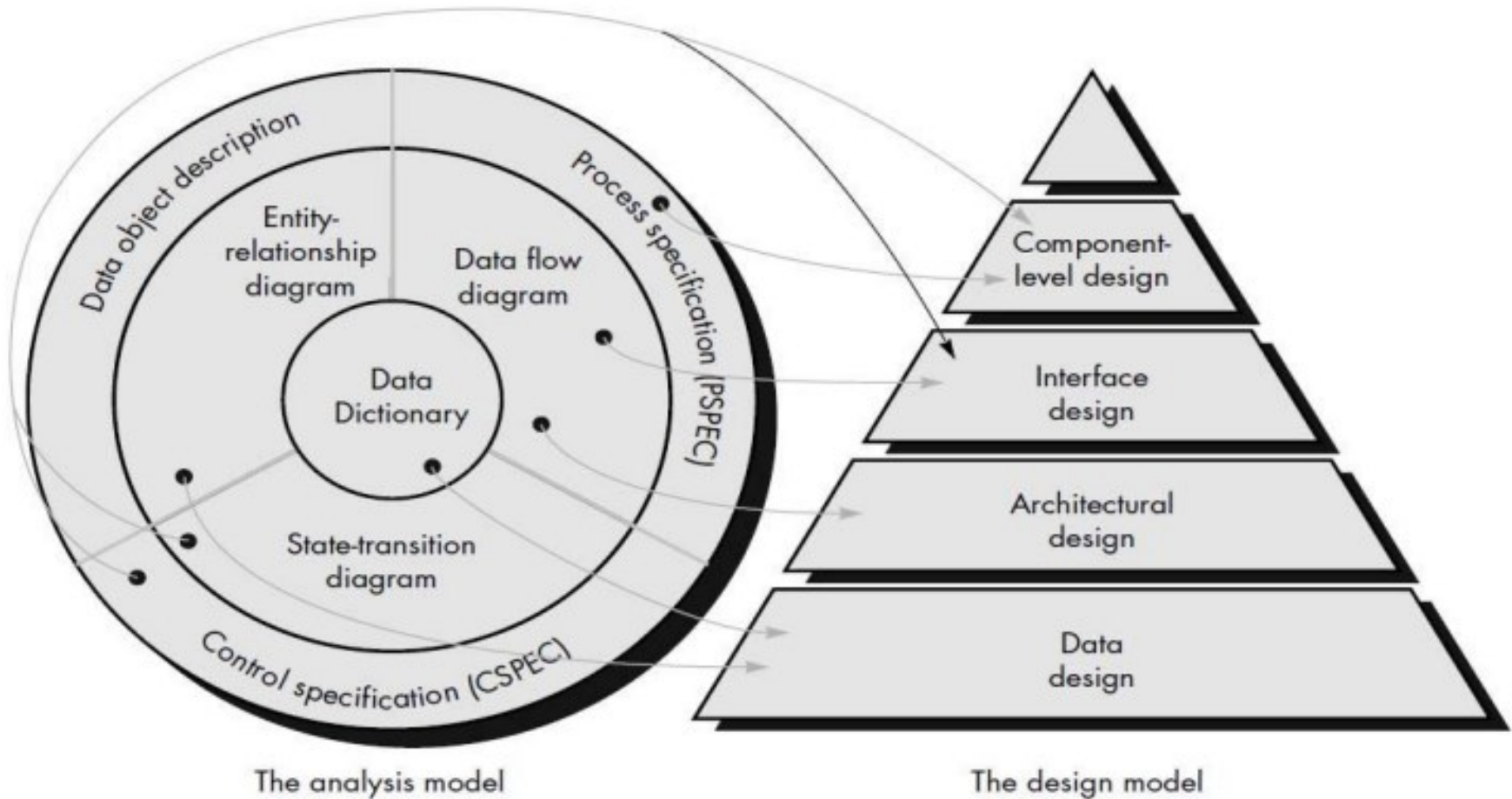


1. Tổng quan quá trình thiết kế

- Chuyển đổi sang mô hình thiết kế:
 - Mỗi một phần tử trong mô hình yêu cầu (requirements model) cung cấp thông tin cần thiết để tạo ra bốn mô hình thiết kế (design model) cho một đặc tả thiết kế hoàn chỉnh.
 - Thiết kế dữ liệu/lớp (data/class design)
 - Thiết kế kiến trúc (architectural design)
 - Thiết kế giao diện (interface design)
 - Thiết kế thành phần (component design)

1. Tổng quan quá trình thiết kế

□ Từ **phân tích** sang **thiết kế**



1. Tổng quan quá trình thiết kế

□ Nguyên tắc thiết kế:

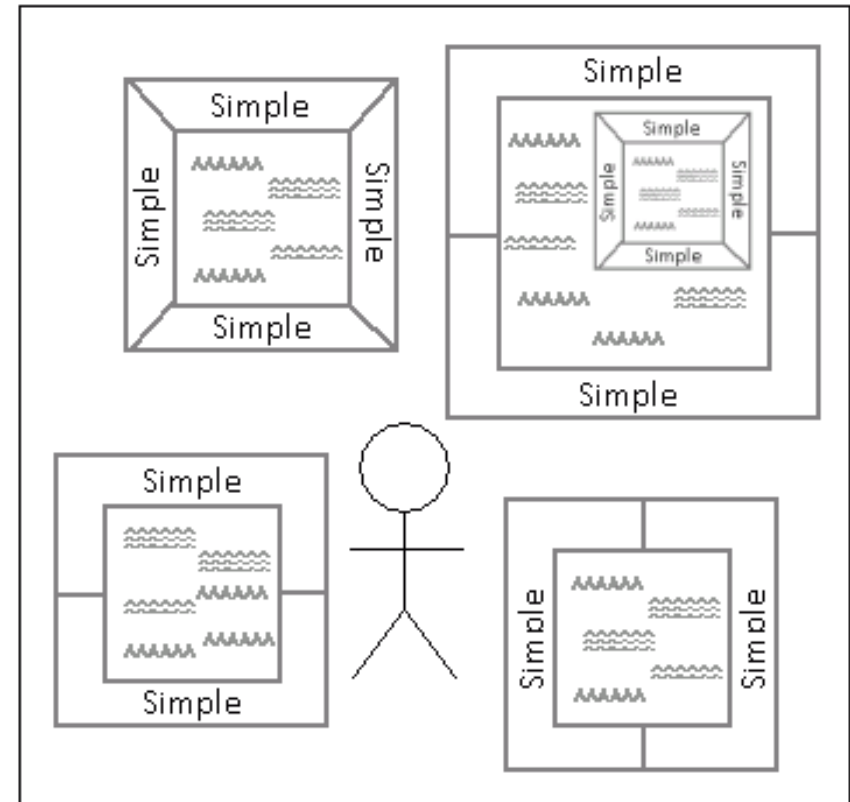
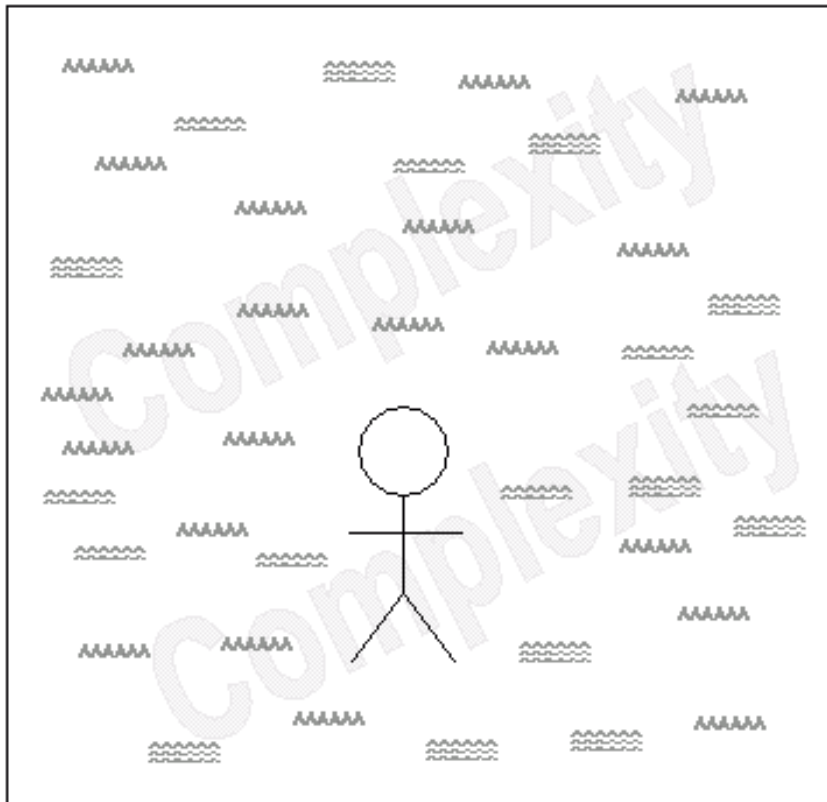
- Quá trình thiết kế không nên bị "bó hẹp tầm nhìn"
- Thiết kế phải lần vết được đến mô hình phân tích
- Thiết kế không được mất thời gian vào những cái đã có
- Thiết kế nên "giảm thiểu khoảng cách trí tuệ" giữa phần mềm và các vấn đề như nó đã tồn tại trong thế giới thực
- Thiết kế phải thể hiện tính thống nhất và tích hợp / toàn vẹn
- Thiết kế phải được cấu trúc để thích ứng với sự thay đổi và giảm thiểu lỗi hoặc điều kiện hoạt động bất thường
- Thiết kế không phải là viết mã lệnh, viết mã lệnh không phải là thiết kế

1. Tổng quan quá trình thiết kế

- 4 đặc tính của một thiết kế tốt
 - Hoàn chỉnh và đầy đủ (complete and sufficient)
 - Nguyên thủy (primitiveness)
 - Gắn kết cao (high cohesion)
 - Nối kết thấp (low coupling)

1. Tổng quan quá trình thiết kế

- ❑ Một thiết kế tốt sẽ ẩn giấu sự phức tạp bên dưới các giao diện đơn giản



► Nội dung

1. Tổng quan quá trình thiết kế
2. Lựa chọn kiến trúc
3. Phân rã các hệ thống con
4. Thiết kế chi tiết

▶ 2. Lựa chọn kiến trúc

- ❑ Thiết kế hệ thống chính là **thiết kế kiến trúc tổng thể** của nó.
- ❑ Các thành phần tạo nên kiến trúc là gì phụ thuộc vào từng cách nhìn đối với hệ thống.
- ❑ Có thể tiếp cận kiến trúc theo 3 góc nhìn (theo hệ con, theo thành phần phần mềm, theo các đơn vị phần cứng):
 - Phân rã hệ thống thành các hệ con (các gói).
 - Mô tả các thành phần vật lý của hệ thống.
 - Bố trí các thành phần khả thi vào các nút phần cứng.

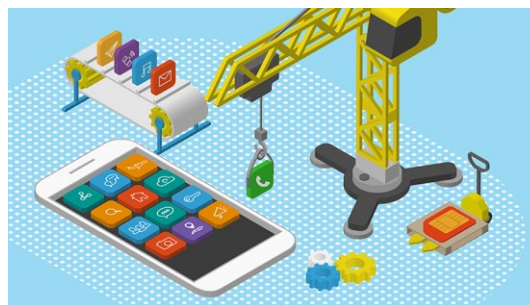
► Why Architecture?

- Các kiến trúc không phải là phần mềm hoạt động. Thay vào đó, nó là một đại diện cho phép một kỹ sư phần mềm:
 - (1) phân tích hiệu quả của thiết kế trong việc đáp ứng các yêu cầu đề ra,
 - (2) xem xét lựa chọn thay thế kiến trúc khi thay đổi thiết kế vẫn tương đối dễ dàng, và
 - (3) giảm thiểu rủi ro gắn liền với việc xây dựng các phần mềm.

▶ 2. Lựa chọn kiến trúc

□ Kiến trúc – Architecture

- **Kiến trúc** liên quan đến việc xác định
 - Các thành phần chính của hệ thống
 - Cách mà các thành phần này liên kết với nhau
- Kiến trúc thể hiện
 - Tổ chức cấu trúc của hệ thống từ các thành phần của nó
 - Cách các phần tử tương tác với nhau để cung cấp các hành vi tổng thể của hệ thống hoặc yêu cầu chức năng



▶ 2. Lựa chọn kiến trúc

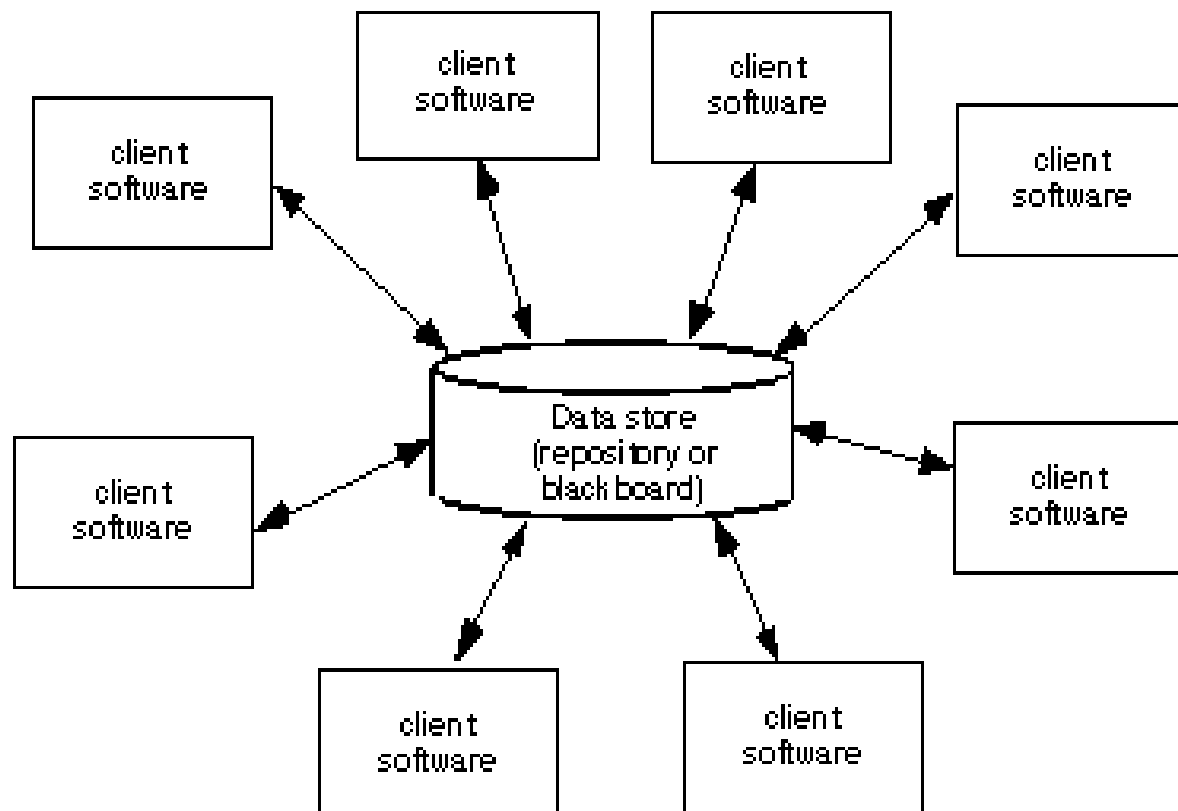
□ Kiến trúc – Architecture

- Kiến trúc là một tập hợp các quyết định quan trọng về việc tổ chức một hệ thống phần mềm. Quyết định này bao gồm:
- Việc lựa chọn các yếu tố **cấu trúc** và **giao diện** của hệ thống
- Sự kết hợp của yếu tố cấu trúc và hành vi thành subsystem
- Phong cách kiến trúc bao gồm các phần tử, giao diện, và sự hợp tác của chúng.

- ❑ Mỗi phong cách mô tả một loại hệ thống bao gồm:
 - (1) một tập hợp các thành phần (ví dụ, một cơ sở dữ liệu, mô đun tính toán) thực hiện một chức năng cần thiết của một hệ thống,
 - (2) một tập hợp các kết nối cho phép "truyền thông, phối hợp và hợp tác" giữa các thành phần,
 - (3) khó khăn để xác định cách các thành phần có thể được tích hợp để tạo thành hệ thống, và
 - (4) các mô hình ngữ nghĩa cho phép một nhà thiết kế phải hiểu được tính chất tổng thể của hệ thống bằng cách phân tích các đặc tính được biết đến của các bộ phận cấu thành của nó.

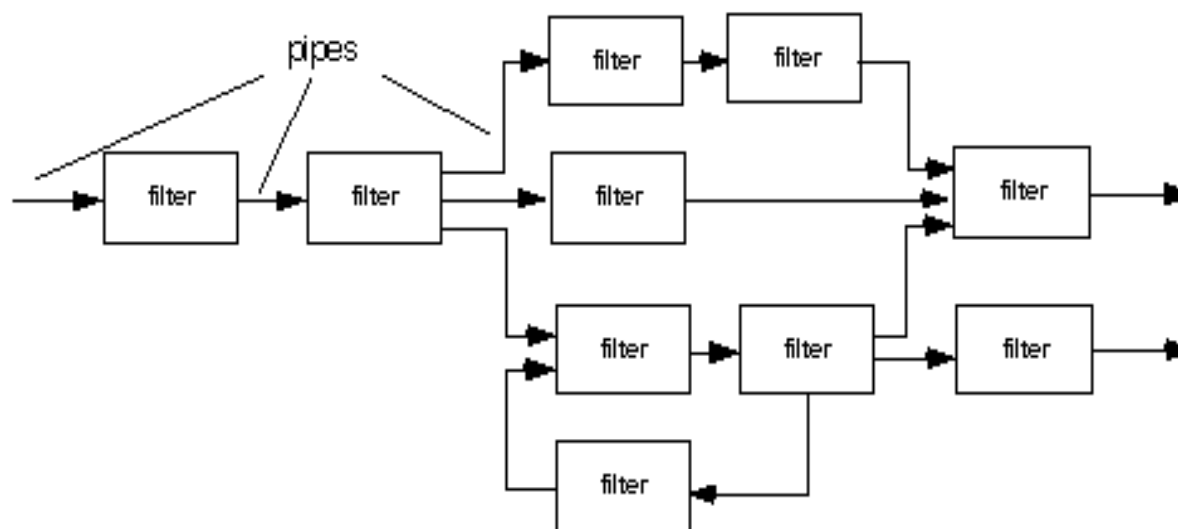
► Kiểu kiến trúc

❑ Kiến trúc lấy dữ liệu làm trung tâm

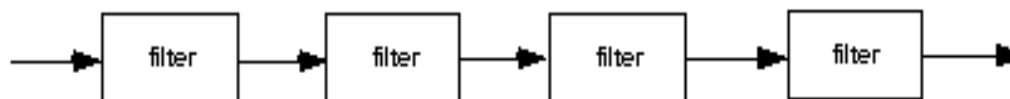


► Kiểu kiến trúc

□ Kiến trúc luồng dữ liệu



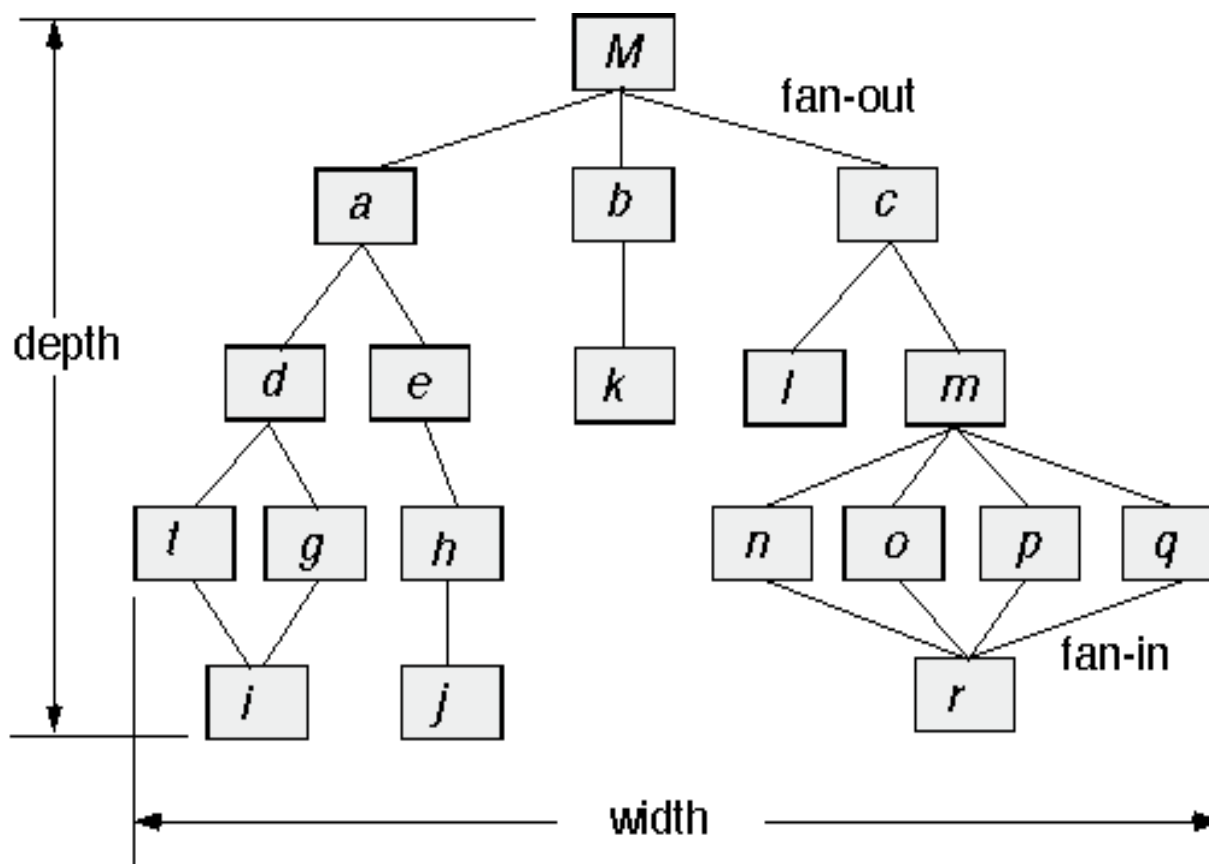
(a) pipes and filters



(b) batch sequential

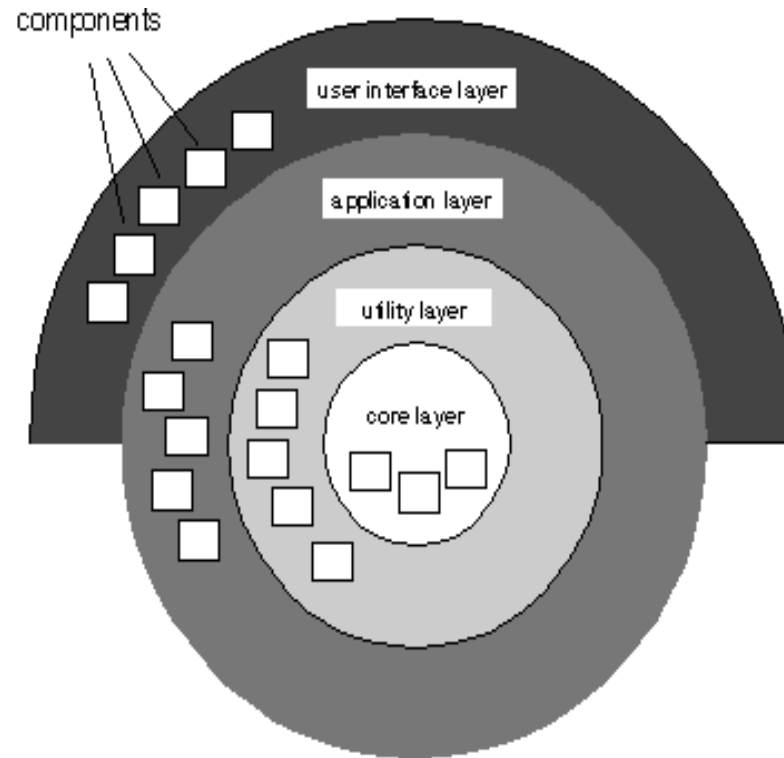
► Kiểu kiến trúc

□ Kiến trúc gọi và trả về



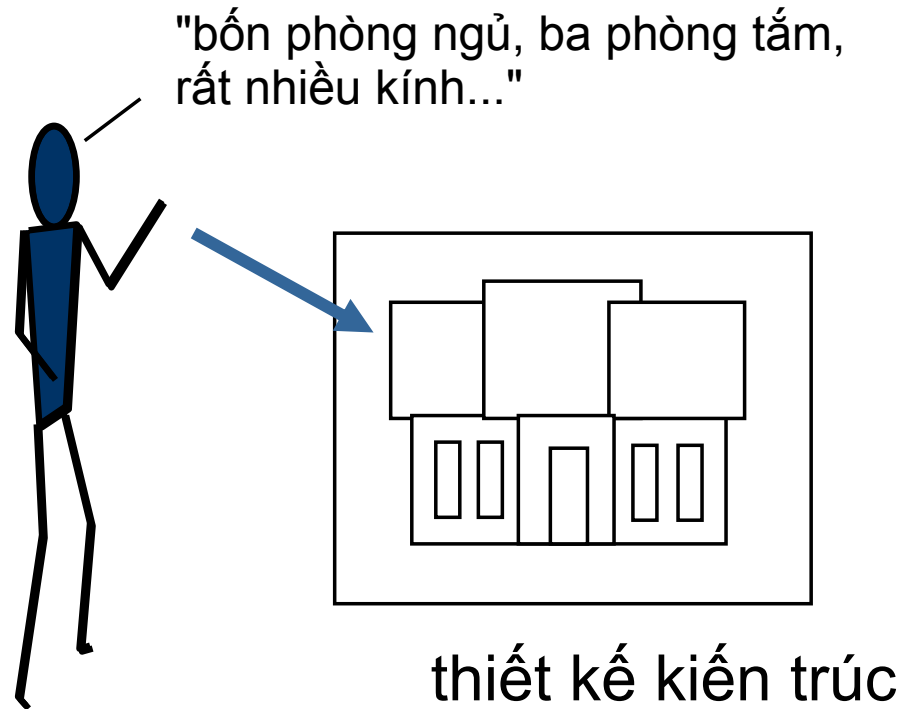
► Kiểu kiến trúc

□ Kiến trúc phân lớp



► Thiết kế kiến trúc

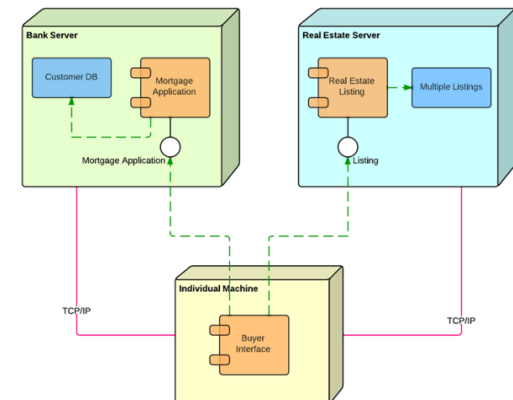
□ Từ yêu cầu khách hàng



▶ 2. Lựa chọn kiến trúc

□ Lựa chọn công nghệ mạng

- Chỉ ra cách hệ thống được phân rã thành các thành phần logic và vật lý riêng biệt như thế nào?
- Việc phân rã được gọi là hình trạng hệ thống (system topology)
- Biểu diễn hình trạng hệ thống bằng đồ triển khai trong UML

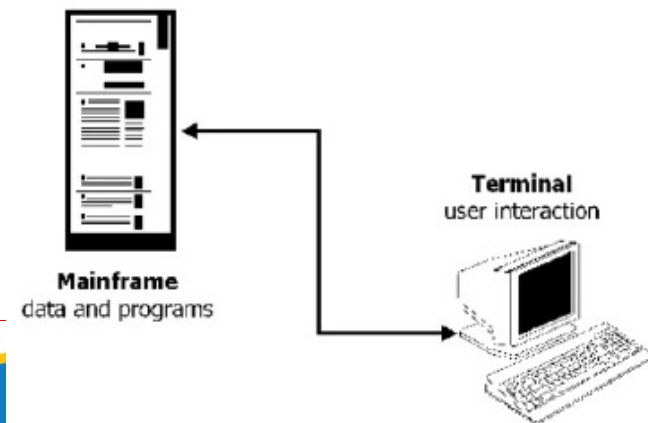


▶ 2. Lựa chọn kiến trúc

❑ Kiến trúc đơn tầng

■ **Mô hình mainframe** kiến trúc một tầng (one-tier Architecture)

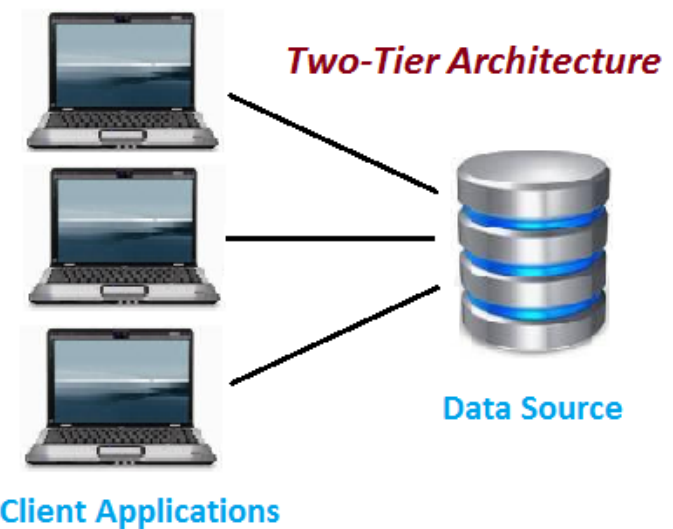
- ❑ Bất cứ chương trình nào được đưa vào, chỉ có duy nhất một mức hoạt động tính toán chạy trên một máy
- ❑ Ưu điểm: Cài đặt đơn giản
- ❑ Hạn chế: chỉ có thể tăng khả năng tính toán bằng cách mua thêm mainframe mới, hoặc nâng cấp cái cũ



▶ 2. Lựa chọn kiến trúc

□ Kiến trúc hai tầng

- Phổ biến vào những năm 1970
- **Ý tưởng**: tăng cường khả năng xử lý trên mỗi máy khách để cho các máy tính trung tâm không phải thực hiện tất cả các tiến trình
- Có thể thêm hay thay thế các máy khách rẻ hơn các máy tính trung tâm



▶ 2. Lựa chọn kiến trúc

□ Kiến trúc hai tầng

- Máy khách (Client) và máy chủ (Server) trao đổi thông tin với nhau dưới dạng các thông điệp (Message).
 - Thông điệp gửi từ máy khách sang máy chủ gọi là các thông điệp yêu cầu (Request Message) mô tả công việc mà phần máy khách muốn máy chủ thực hiện.
 - Mỗi khi máy chủ nhận được một thông điệp yêu cầu, máy chủ sẽ phân tích yêu cầu, thực thi công việc theo yêu cầu và gửi kết quả về máy khách trong một thông điệp trả lời (Reply Message).
 - Mô hình máy khách - máy chủ (client-server) phải có một Giao thức (Protocol) riêng để trao đổi thông tin



▶ 2. Lựa chọn kiến trúc

□ Kiến trúc Client-Server

- **Server:** một máy đơn hoặc một ứng dụng mà nó cung cấp các dịch vụ cho nhiều clients
 - Có thể là **IIS** (Internet Information Services) dựa trên máy chủ Web
 - Có thể là WCF (Windows Communication Foundation)
 - Có thể là một dịch vụ trong đám mây,...
- **Clients:** phần mềm ứng dụng cung cấp giao diện người dùng cuối UI để truy cập các dịch vụ từ server
 - WPF, HTML5, Silverlight, ASP.NET, ...

▶ 2. Lựa chọn kiến trúc

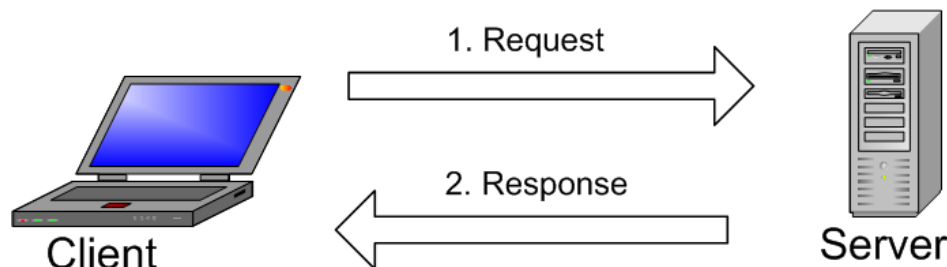
□ Kiến trúc Client-Server



▶ 2. Lựa chọn kiến trúc

□ Ví dụ

- Web server (IIS) – Web browser (Firefox)
- FTP server (ftpd) – FTP client (FileZilla)
- Email server (qmail) – email client (Outlook)
- SQL Server – SQL Server Management Studio
- Bit Torrent Tracker – Torrent client (µTorrent)
- DNS server (bind) – DNS client (resolver)
- DHCP server (wireless router firmware) – DHCP client (mobile phone /Android DHCP client/)
- SMB server (Windows) – SMB client (Windows)



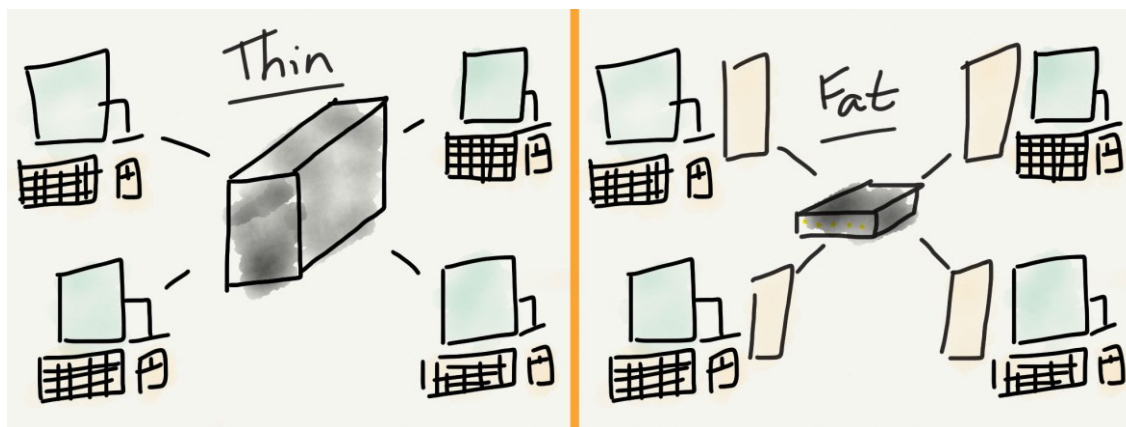
▶ 2. Lựa chọn kiến trúc

□ Kiến trúc hai tầng

- Thông thường phần máy khách đảm nhận các chức năng:
 - Giao diện người dùng: như tạo các form nhập liệu, các thông báo, các biểu báo giao,...
- Phần máy chủ này đảm nhận các chức năng về xử lý và lưu trữ dữ liệu.

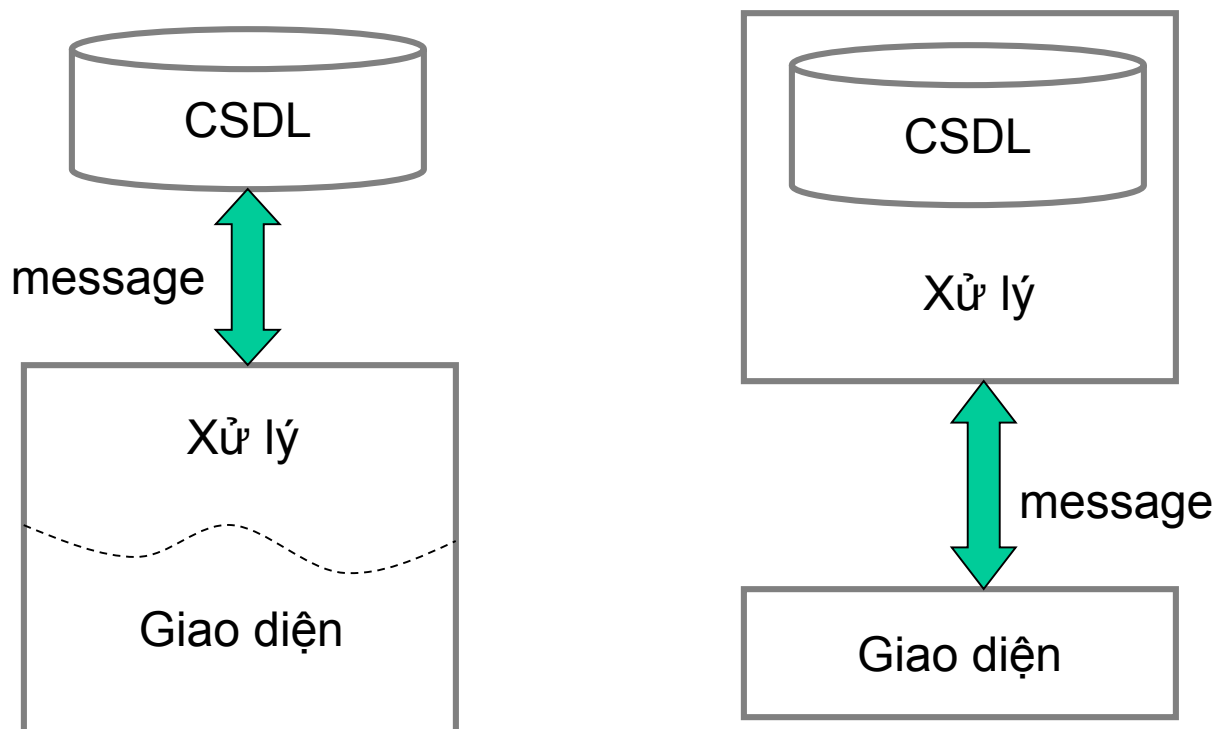
► 2. Lựa chọn kiến trúc

- ❑ Máy khách nhẹ (Thin Client)
 - Máy khách đều có khả năng xử lý (thường là yếu hơn máy chủ) và lưu trữ riêng để phục vụ cho các tác vụ không liên quan đến mạng
- ❑ Máy khách nặng (Fat Client)
 - Các hoạt động nghiệp vụ được cài đặt bên phía máy khách và phần máy chủ thực hiện chức năng chủ yếu về truy vấn và lưu trữ thông tin



▶ 2. Lựa chọn kiến trúc

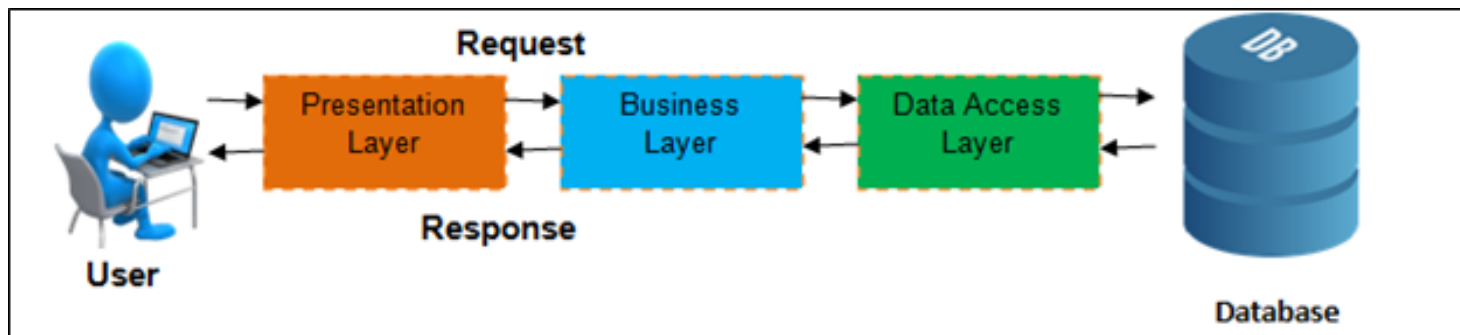
□ (Thin Client) & (Fat Client)



▶ 2. Lựa chọn kiến trúc

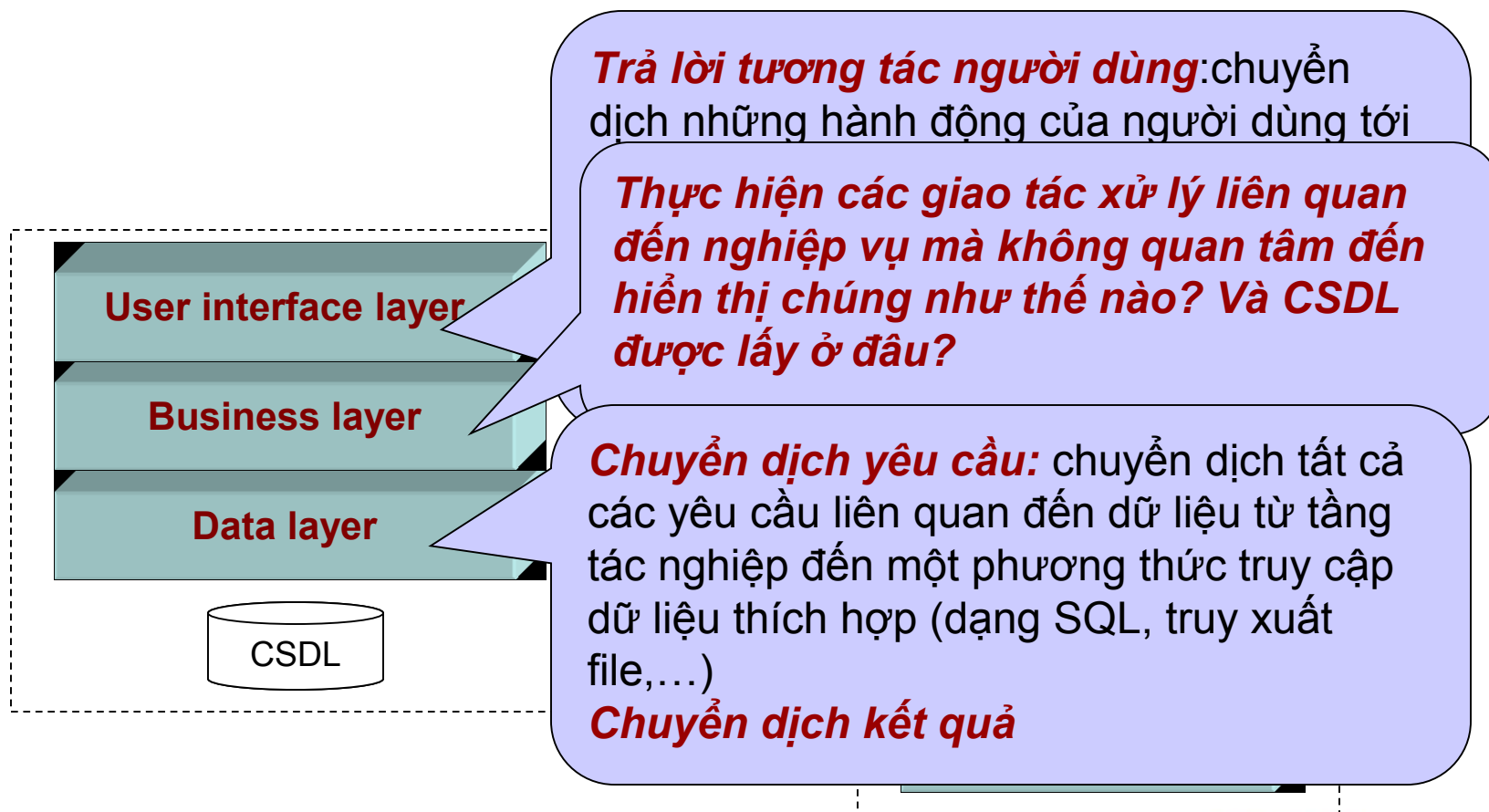
□ Kiến trúc ba tầng

- Phổ biến vào những năm 1990 bằng cách chia hệ thống thành ba phần:
 - giao diện người dùng,
 - logic chương trình
 - lưu trữ dữ liệu.



▶ 2. Lựa chọn kiến trúc

□ Kiến trúc ba tầng



▶ 2. Lựa chọn kiến trúc

☐ Kiến trúc 3-Tier / Multi-Tier

■ **Front-end (client layer)**

- ☐ Client software: cung cấp giao diện (UI) của hệ thống

■ **Middle tier (business layer)**

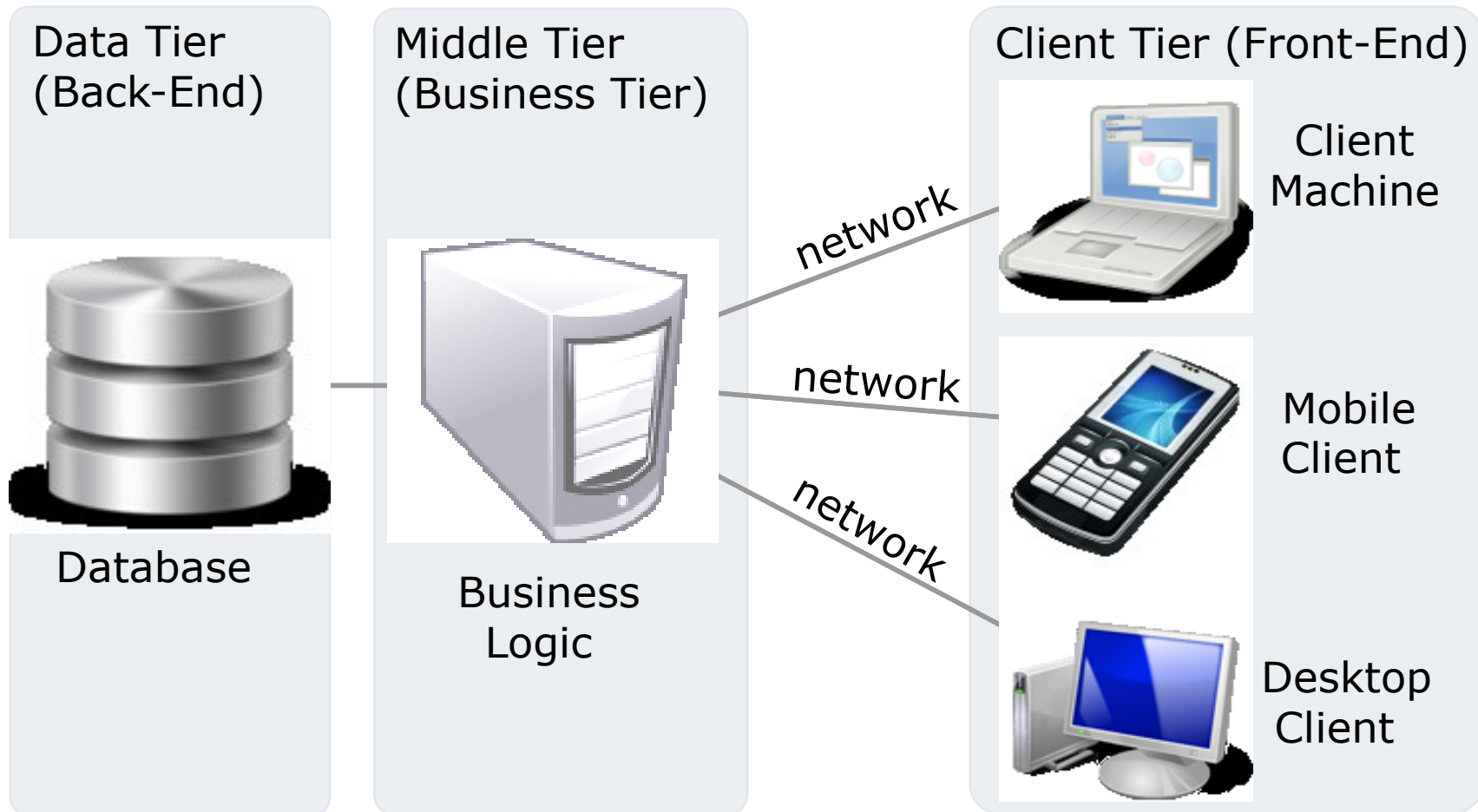
- ☐ Server software: cung cấp lõi logic của hệ thống.
- ☐ Hiện thực các quy trình nghiệp vụ hoặc các dịch vụ

■ **Back-end (data layer)**

- ☐ Quản lý dữ liệu của hệ thống(database / cloud)

▶ 2. Lựa chọn kiến trúc

❑ Mô hình kiến trúc 3-Tier / Multi-Tier

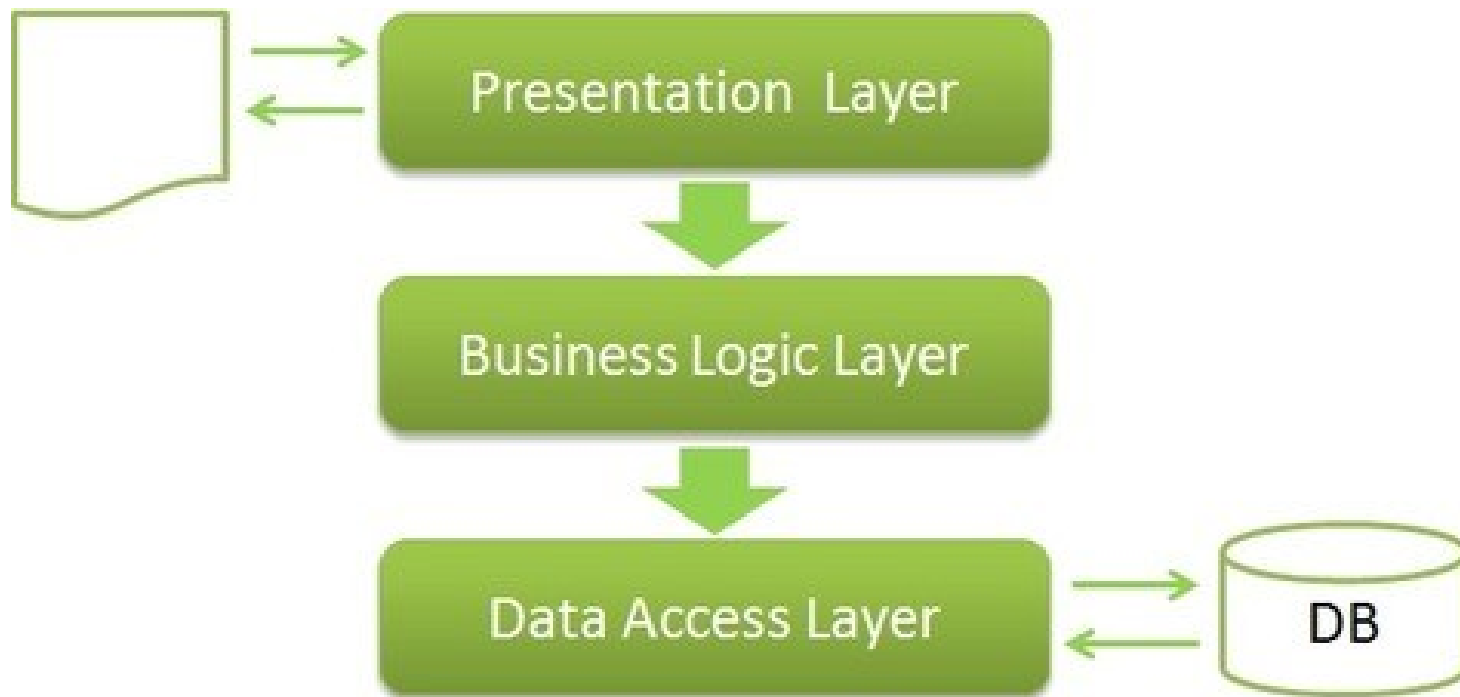


▶ 2. Lựa chọn kiến trúc

- ❑ Kiến trúc ba tầng kết hợp với phân rã các lớp (layer) trong ứng dụng
 - **Layer giao diện người dùng (User Interface)** hay máy khách thể hiện giao diện mà người sử dụng có thể nhập yêu cầu, dữ liệu và xem kết quả.
 - **Layer ứng dụng (Application Server, Business Rule)** được biết như tầng logic nghiệp vụ hay tầng dịch vụ, thực hiện các chức năng chính và độc lập với cách thiết kế cũng như cài đặt giao diện
 - **Layer dữ liệu (Database Server, Data Storage)** nhằm lưu trữ dữ liệu và cung cấp cơ chế an toàn cho việc truy nhập đồng thời với sự giúp đỡ của hệ quản trị cơ sở dữ liệu.

▶ 2. Lựa chọn kiến trúc

- ❑ Kiến trúc ba tầng kết hợp với phân rã các lớp (layer) trong ứng dụng



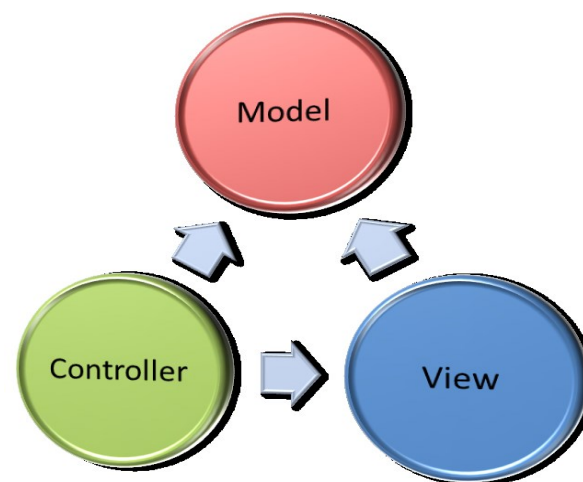
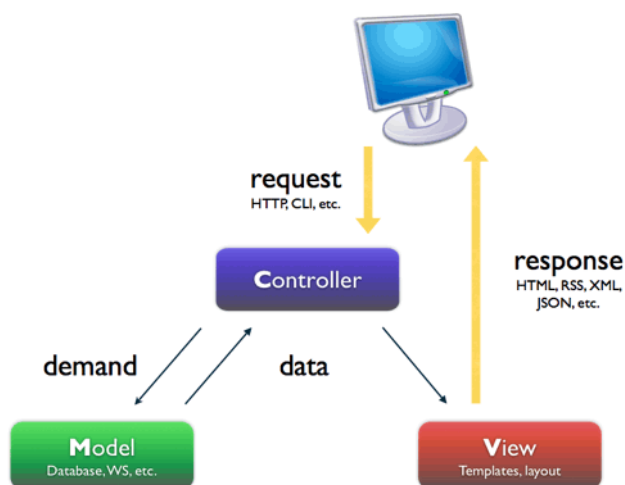
▶ 2. Lựa chọn kiến trúc

- ❑ Các ưu điểm của kiến trúc 3 tầng:
 - Tạo điều kiện dễ dàng khi phát triển
 - Sử dụng máy tính hiệu quả hơn
 - Cải tiến hiệu năng
 - Nâng cao tính bảo mật
 - Hạn chế đầu tư
 - Tính linh hoạt
 - Đa dạng kiểu dáng máy client

▶ 2. Lựa chọn kiến trúc

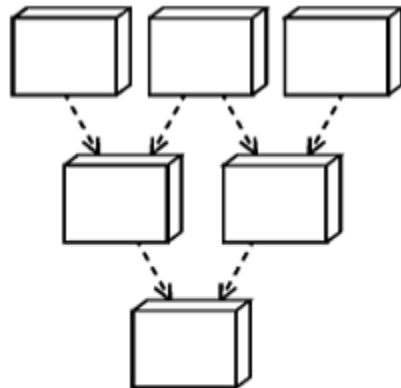
❑ Kiến trúc Model-View-Controller

- ❑ Tách riêng logic nghiệp vụ với dữ liệu của ứng dụng và cách trình bày
 - Model: Giữ trạng thái của ứng dụng (data)
 - View: Hiển thị dữ liệu cho người sử dụng (UI)
 - Controller: Điều khiển tương tác người sử dụng

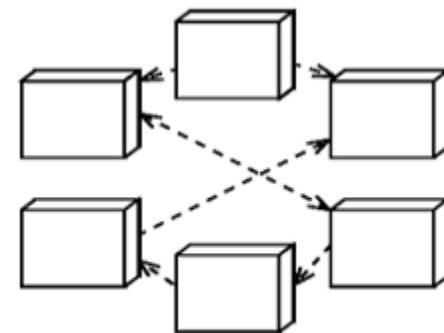


▶ 2. Lựa chọn kiến trúc

- ❑ Kiến trúc Client – Server và phân tán:
 - Kiến trúc phân tán (hay ngang hàng) được đặc trưng bằng một tập các máy tự chủ, truyền thông nhau theo bất kì một hướng nào khi có nhu cầu phát sinh
 - Thực hiện công việc tính toán lớn trải rộng ra trên nhiều máy tính trong mạng



Client–Server Architecture

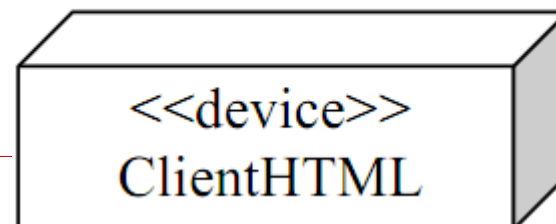


Distributed Architecture

▶ 2. Lựa chọn kiến trúc

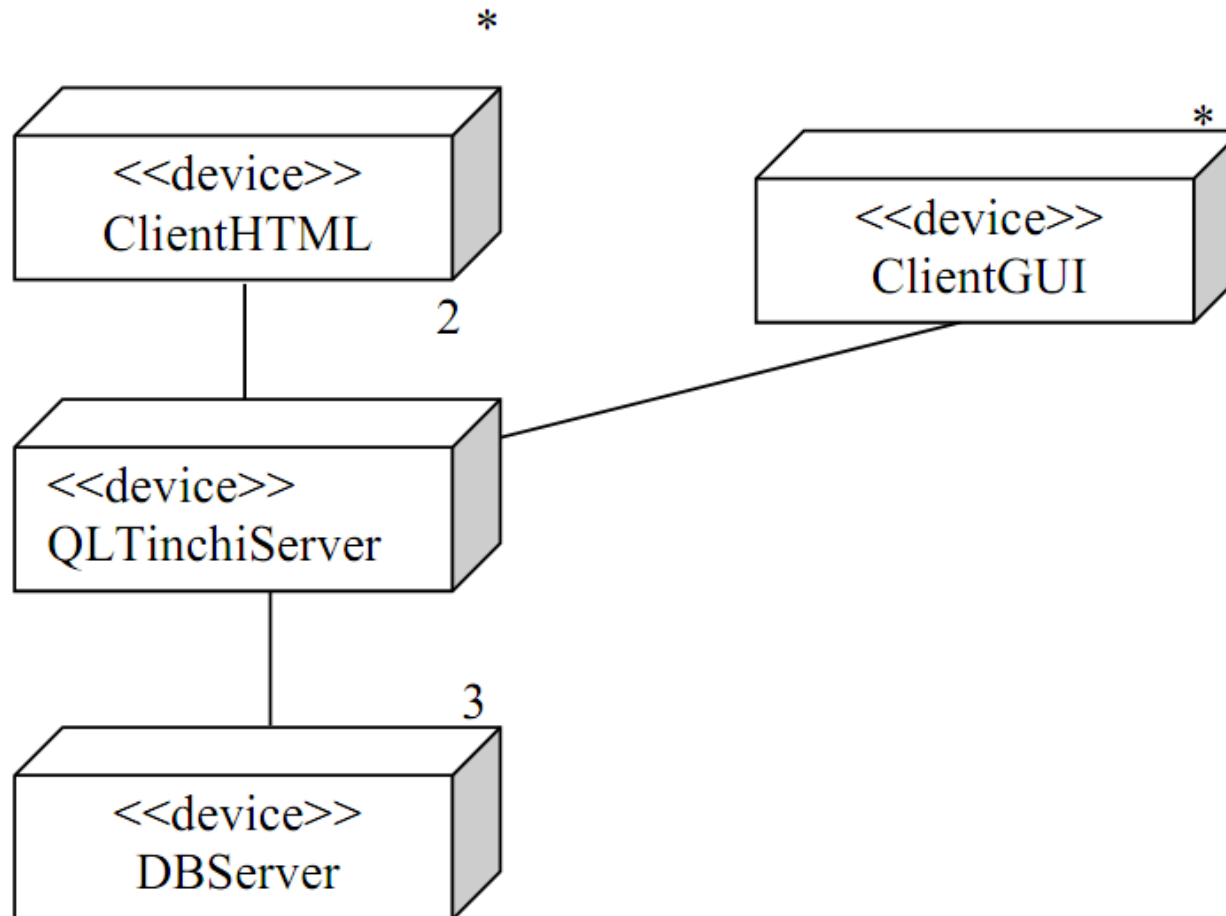
- Mô hình hoá kiến trúc với các biểu đồ UML
 - Kiến trúc hệ thống có thể được mô tả bằng **biểu đồ triển khai** trong UML
 - Biểu đồ triển khai đơn giản chỉ nêu ra các nút, đường truyền thông và vô số các điểm
 - Mỗi nút trong biểu đồ này thể hiện một máy trạm (thể hiện với từ khóa UML <<device>>)
 - Các nút có dấu * thể hiện có rất nhiều nút có thể tồn tại trong thời gian chạy
 - Khi chỉ ra các thể hiện nút, giống như các đối tượng trong biểu đồ đối tượng, nhãn nút có dạng tên:Kiểu và nên được gạch chân

*

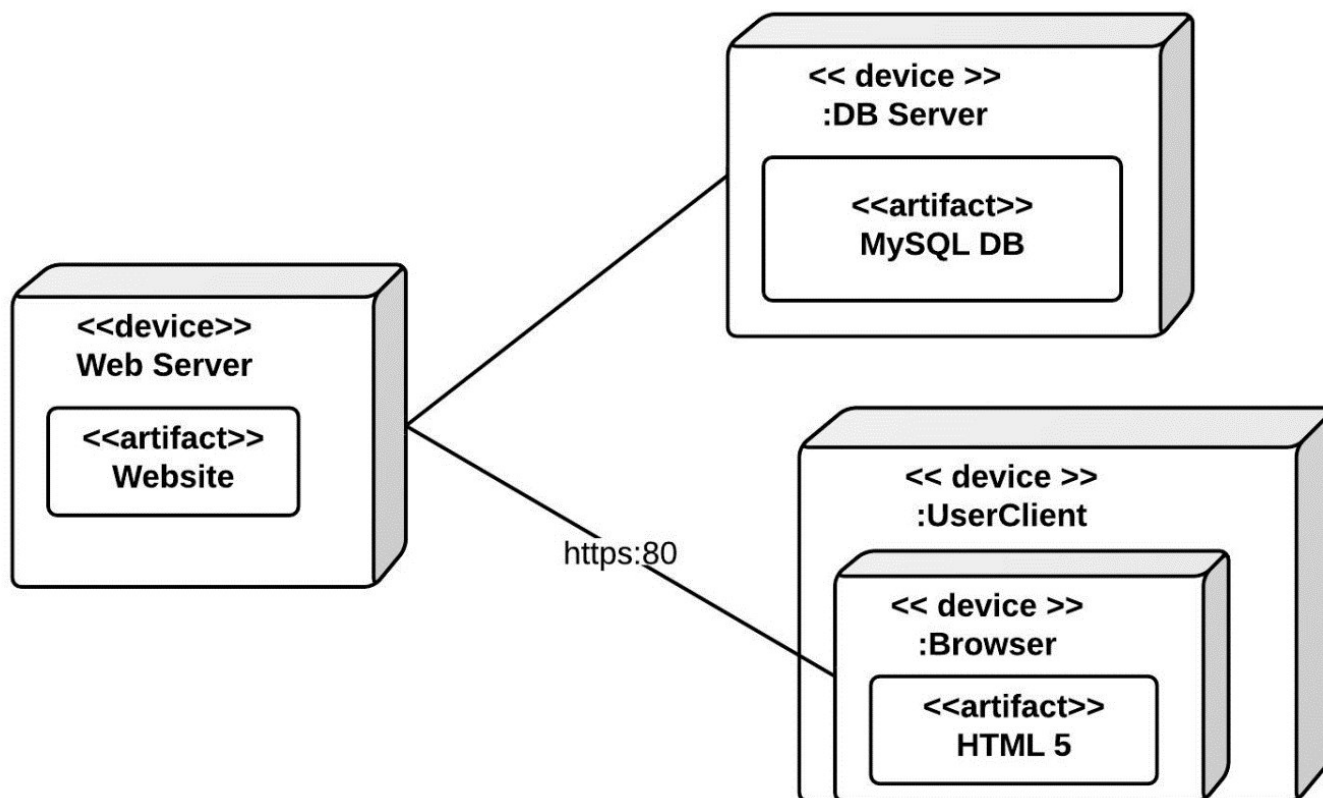


▶ 2. Lựa chọn kiến trúc

□ Ví dụ biểu đồ triển khai



- ❑ Xây dựng biểu đồ triển khai sau trên công cụ Visual Paradigm



► Nội dung

1. Tổng quan quá trình thiết kế
2. Lựa chọn kiến trúc
3. Phân rã các hệ thống con
4. Thiết kế chi tiết

▶ 3. Phân rã các hệ thống con

☐ Thiết kế tương tranh:

- **Hệ thống tương tranh (concurrent system)** hay đồng thời: nhiều hoạt động thường xảy ra cùng một lúc như trong một thể thống nhất
- Các vấn đề:
 - ☐ Làm sao đảm bảo được rằng thông tin được cập nhật đầy đủ trước khi người nào thao tác trên cập nhật đó
 - ☐ Làm sao đảm bảo được rằng thông tin không được cập nhật trong khi nó đang được đọc
 - ☐ ...
- Hai mức độ:
 - ☐ **Mức thấp:** các giao dịch cơ sở dữ liệu và điều khiển luồng để bảo vệ dữ liệu bên trong các tiến trình riêng lẻ
 - ☐ **Mức cao:** sử dụng các **quy tắc của hệ thống** và **quy tắc nghiệp vụ** để điều khiển hoạt động tương tranh

▶ 3. Phân rã các hệ thống con

□ Thiết kế an toàn-bảo mật:

- Một hệ thống được gọi là an toàn khi hệ thống được bảo vệ khỏi sự lạm dụng dù là vô tình hay cố ý
- Các khía cạnh:
 - **Sự riêng tư (Privacy)**: Thông tin có thể được che dấu, và chỉ ở trạng thái sẵn sàng với những người dùng được phép tác động vào nó như xem hoặc chỉnh sửa.
 - **Xác thực (Authentication)**: quyết định xem phần thông tin đó có đáng tin cậy hay không
 - **Tính không thể bác bỏ được (Irrefutability)**: người tạo ra thông tin không thể phủ nhận họ chính là người tạo ra nó
 - **Tính toàn vẹn (Integrity)**: Đảm bảo rằng thông tin không bị mất mát trên đường đi, bảo đảm sự nhất quán của dữ liệu trong hệ thống
 - **Tính an toàn (Safety)**: phải có thể điều khiển việc truy cập tài nguyên

▶ 3. Phân rã các hệ thống con

□ Quy luật chung:

- Ngăn chặn việc xâm nhập máy chủ khi không được phép
- Các thông tin nhạy cảm phải được đảm bảo không bị truyền ra ngoài
- Đảm bảo thông tin khi đi ra bên ngoài không bị “nghe lén” trong quá trình truyền và chỉ có đúng người nhận mới có thể đọc được
- Bảo vệ mật mã (password)
- Bảo vệ tài nguyên hệ thống của client

▶ 3. Phân rã các hệ thống con

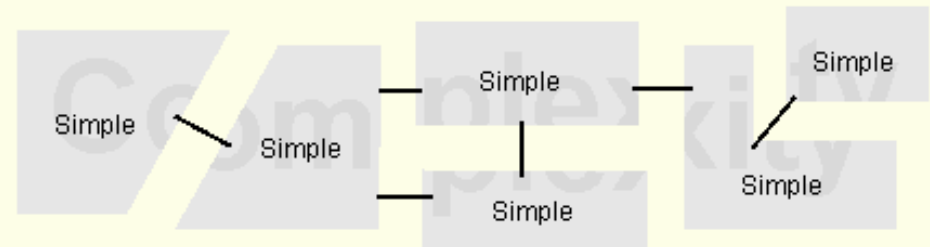
- Hệ thống (system) và hệ thống con (subsystem)
 - Hệ thống nghiệp vụ nên có một số hệ thống tách biệt nhau,
 - Mỗi hệ thống được cài đặt bởi các nhóm phát triển khác nhau để việc sử dụng lại các đối tượng không tương thích được giảm bớt.
 - Thông tin nào cần được truyền giữa các hệ thống phải được truyền theo một cách xác định, được điều khiển qua những **giao diện** xác định trước.
 - Để giảm độ phức tạp hơn nữa, mỗi hệ thống nên được chia nhỏ hơn nữa thành các **hệ thống con** riêng biệt

Phân chia hệ thống thành hệ thống con

- Mục tiêu:
 - Giảm thiểu sự phức tạp, sự chồng chéo của một hệ thống lớn
 - Tạo thuận lợi cho công việc thiết kế: bởi vì phải chi tiết hoá các nội dung đặt được ở phần tích tích
 - Dễ dàng hơn cho quá trình bảo dưỡng hệ thống sau này

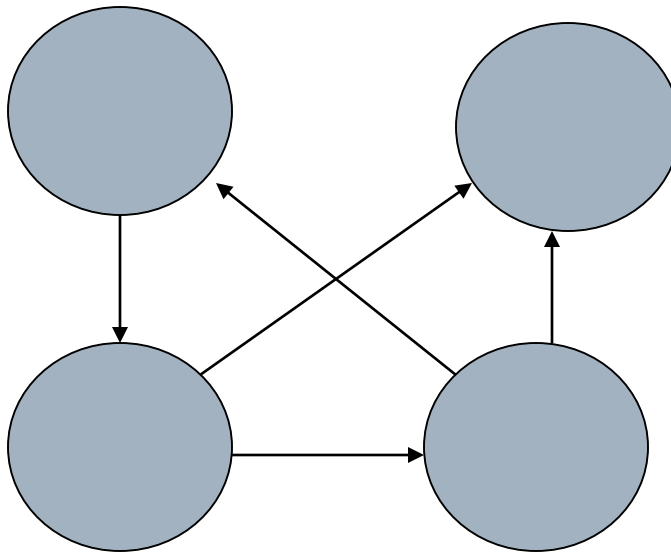
Complexity

=>

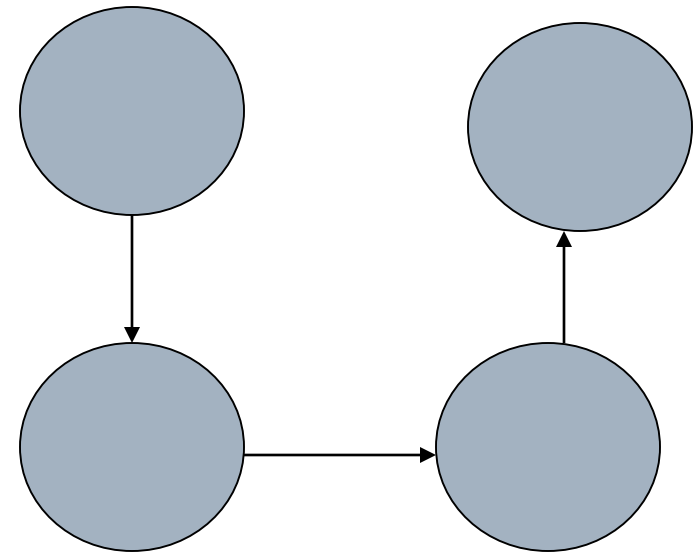


Phân chia hệ thống thành hệ thống con

□ Duy trì tính độc lập của các thành phần thiết kế



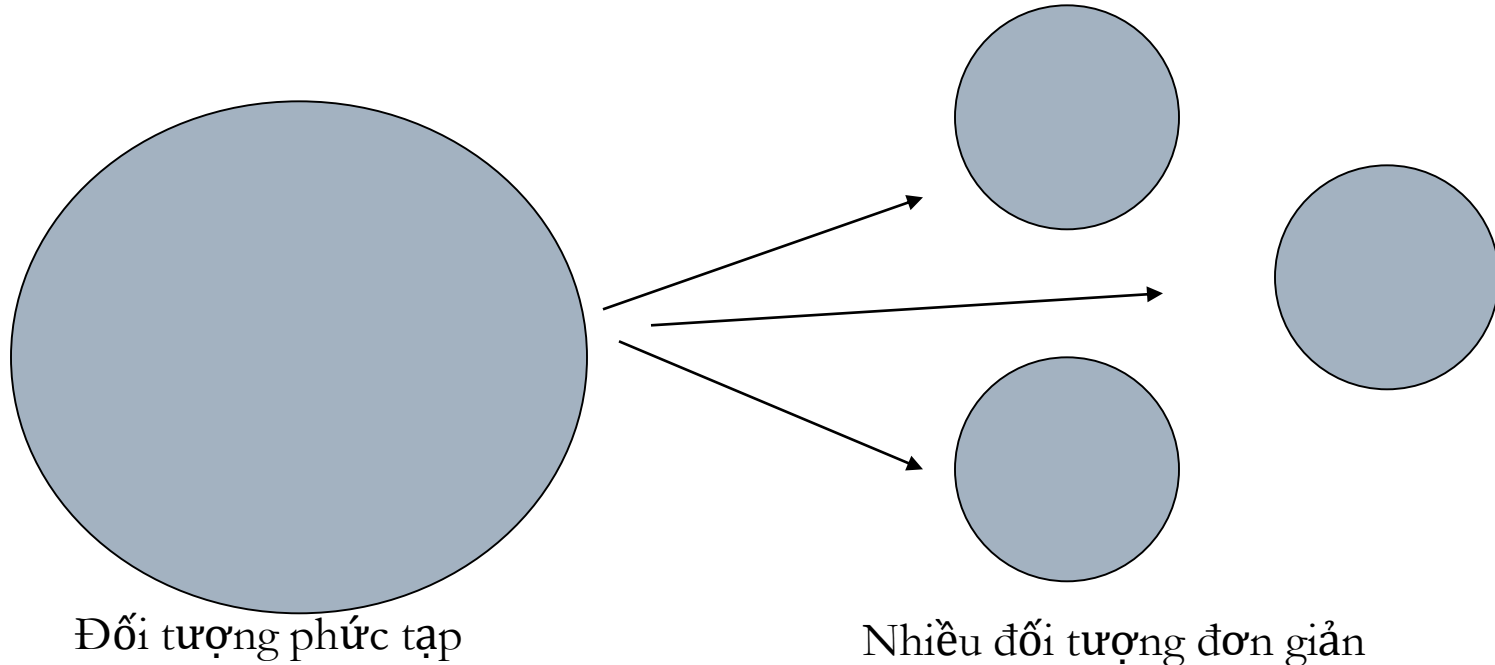
Tính độc lập của các thành phần không cao



Tính độc lập của các thành phần cao

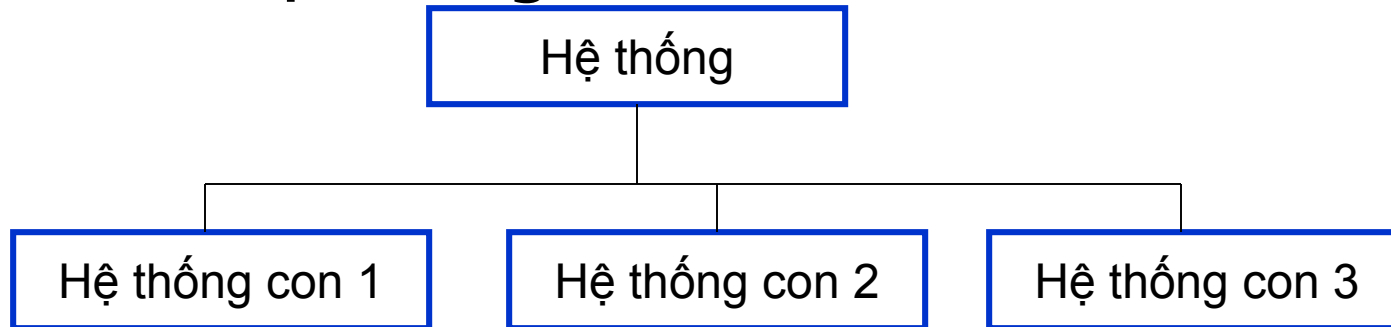
Phân chia hệ thống thành hệ thống con

- Giảm tối đa thông tin các đối tượng thiết kế
 - Tách thành nhiều đối tượng đơn giản hơn
 - Tách thành cấu trúc phân cấp kế thừa



Phân chia hệ thống thành hệ thống con

■ Mô tả hệ thống con:



Hệ thống: ABC

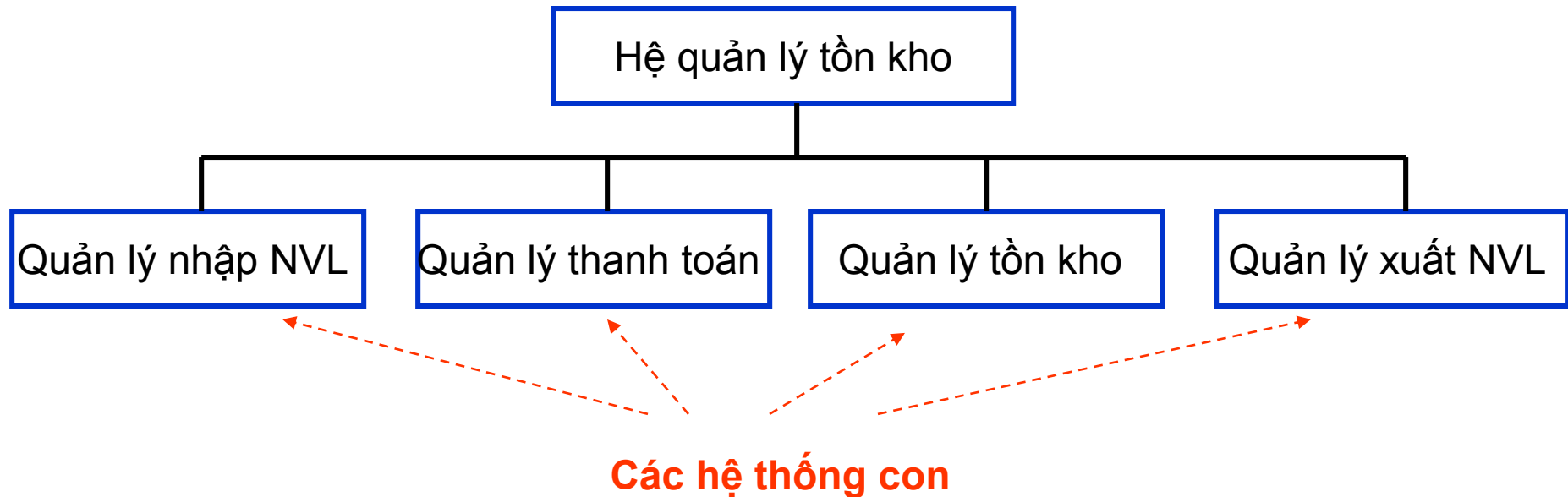
Dòng dữ liệu vào:

Dòng dữ liệu ra:

| STT | Hệ thống con | Xử lý | Kho dữ liệu |
|-----|--------------|-------|-------------|
| | | | |

Phân chia hệ thống thành hệ thống con

- Ví dụ: hệ thống quản lý tồn kho và các hệ thống con

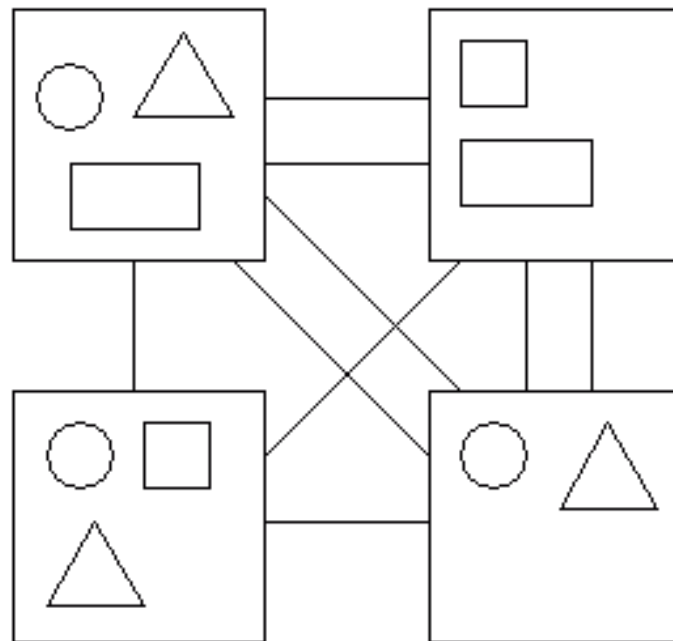


Phân chia hệ thống thành hệ thống con

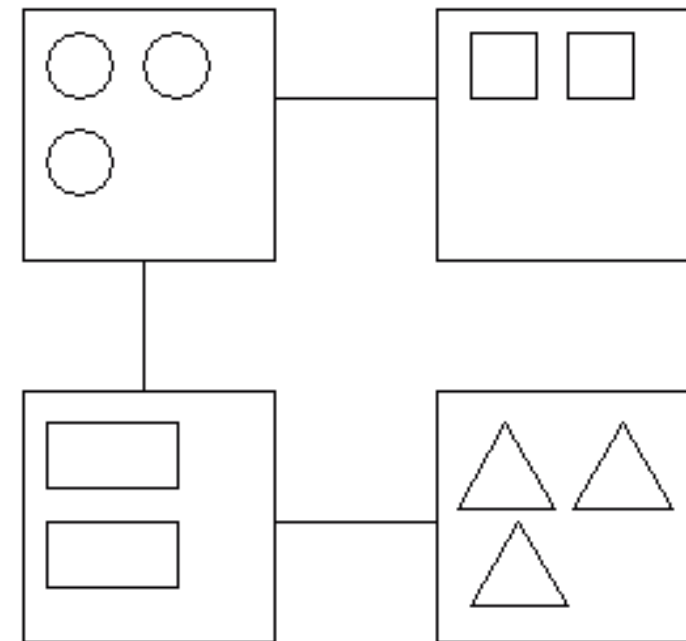
- Tiêu chí phân chia:
 - Sự gắn kết / cố kết (cohesion): sự gắn bó về luận lý hoặc mục đích của các xử lý trong một hệ thống con.
 - Tính cố kết càng cao thì càng tốt
 - Sự liên kết / ghép nối (coupling): sự trao đổi thông tin và tác động lẫn nhau giữa các hệ thống con.
 - Sự liên kết này càng lỏng lẻo, càng đơn giản càng tốt

Phân chia hệ thống thành hệ thống con

- **The best designs** have **high cohesion** (also called strong cohesion) within a module and **low coupling** (also called weak coupling) between modules.



Low cohesion and high coupling



High cohesion and low coupling

Phân chia hệ thống thành hệ thống con

□ Sự gắn kết (cohesion)

■ Góc nhìn quy ước:

□ Một “tư duy đơn lẻ” của một mô-đun.

■ Góc nhìn hướng đối tượng:

□ *Sự gắn kết nghĩa là một thành phần hoặc một lớp chỉ đóng gói các thuộc tính và hoạt động liên quan chặt chẽ với nhau và với bản thân thành phần hoặc lớp đó.*

■ Các mức của sự gắn kết:

□ Chức năng, Tầng, Truyền thông, Liên tục, Thủ tục, Phụ thuộc thời gian, Lợi ích

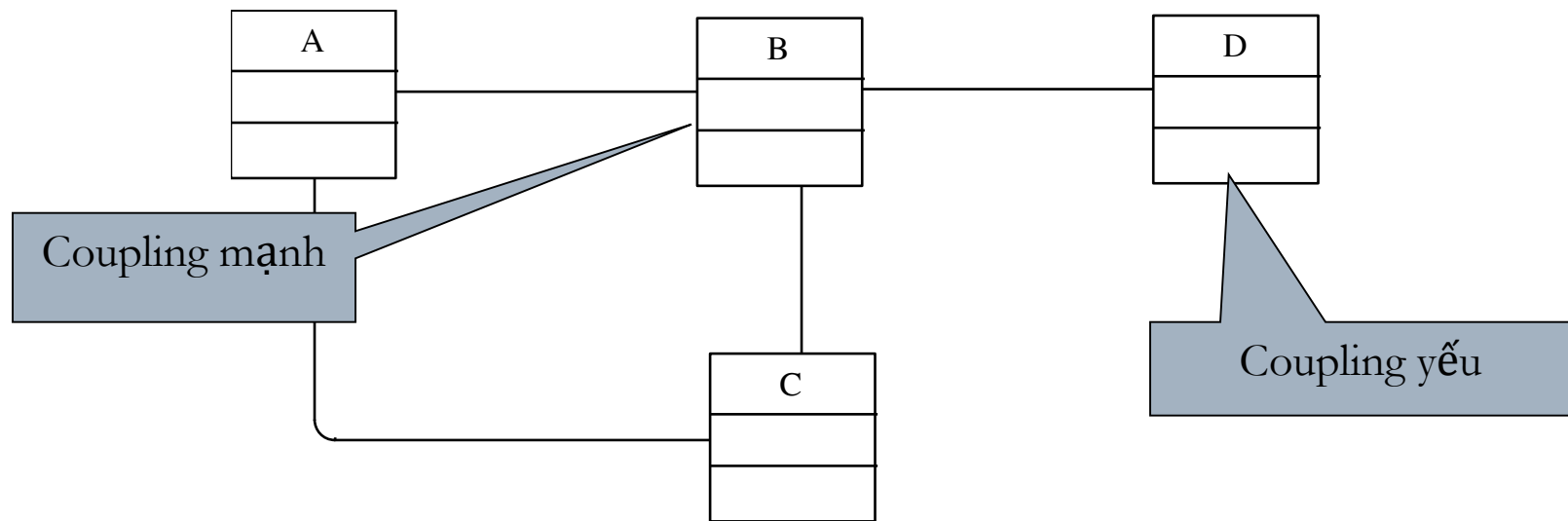
Phân chia hệ thống thành hệ thống con

☐ Sự liên kết (coupling)

- Dùng để đo mức độ liên quan lẫn nhau giữa các đối tượng hoặc giữa thành phần phần mềm
- Biểu diễn thông qua kết hợp nhị phân
- Coupling càng mạnh → liên hệ giữa các đối tượng càng phức tạp
- Đánh giá coupling dựa vào:
 - ☐ Mức độ phức tạp của kết nối
 - ☐ Kết nối tham chiếu đến chính bản thân đối tượng hoặc bên ngoài đối tượng
 - ☐ Các thông điệp nhận và gửi đi

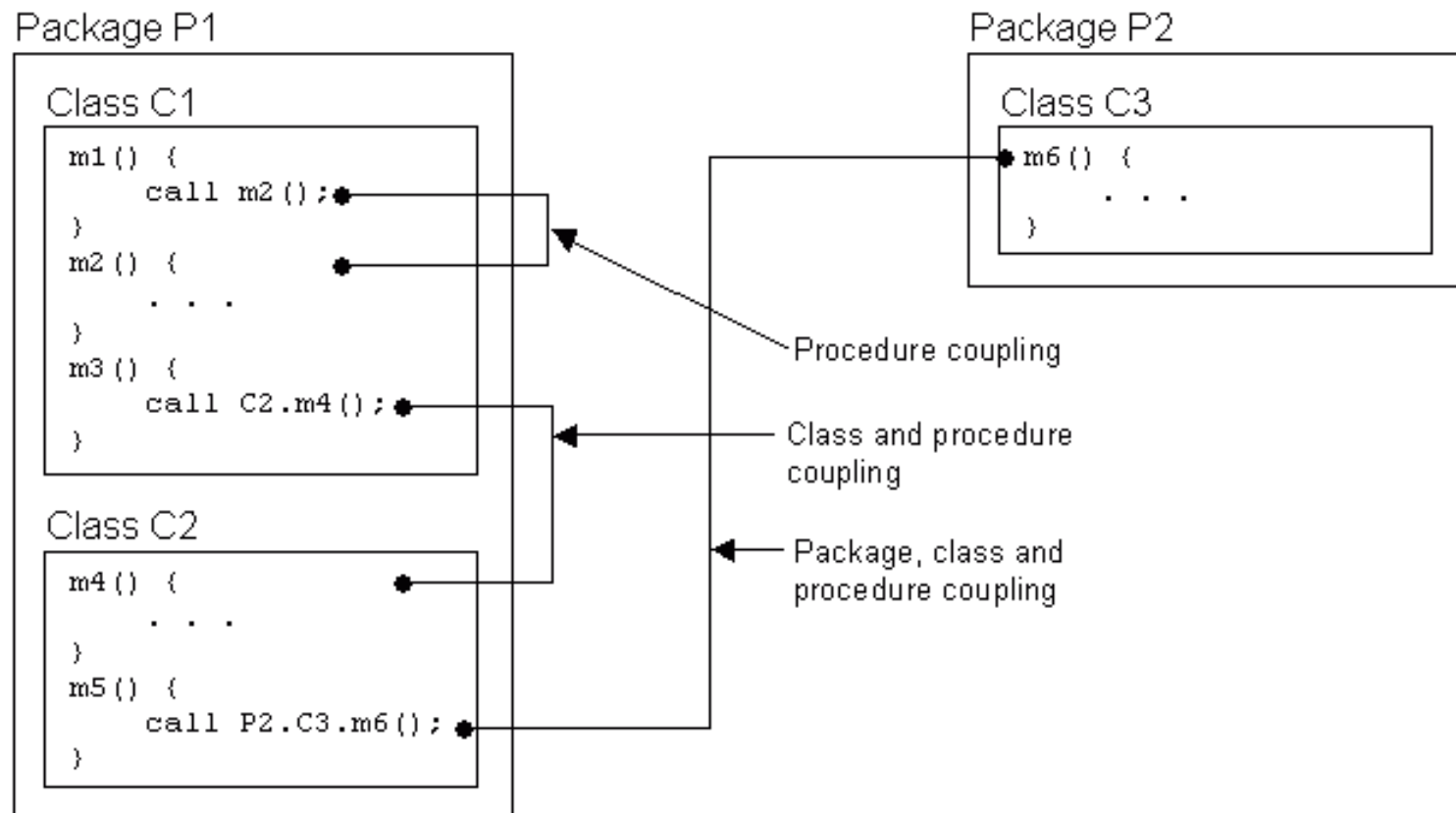
Phân chia hệ thống thành hệ thống con

□ Coupling – Ví dụ:



Phân chia hệ thống thành hệ thống con

- ❑ Coupling có thể ở nhiều mức độ: giữa các phương thức, các lớp hoặc các package



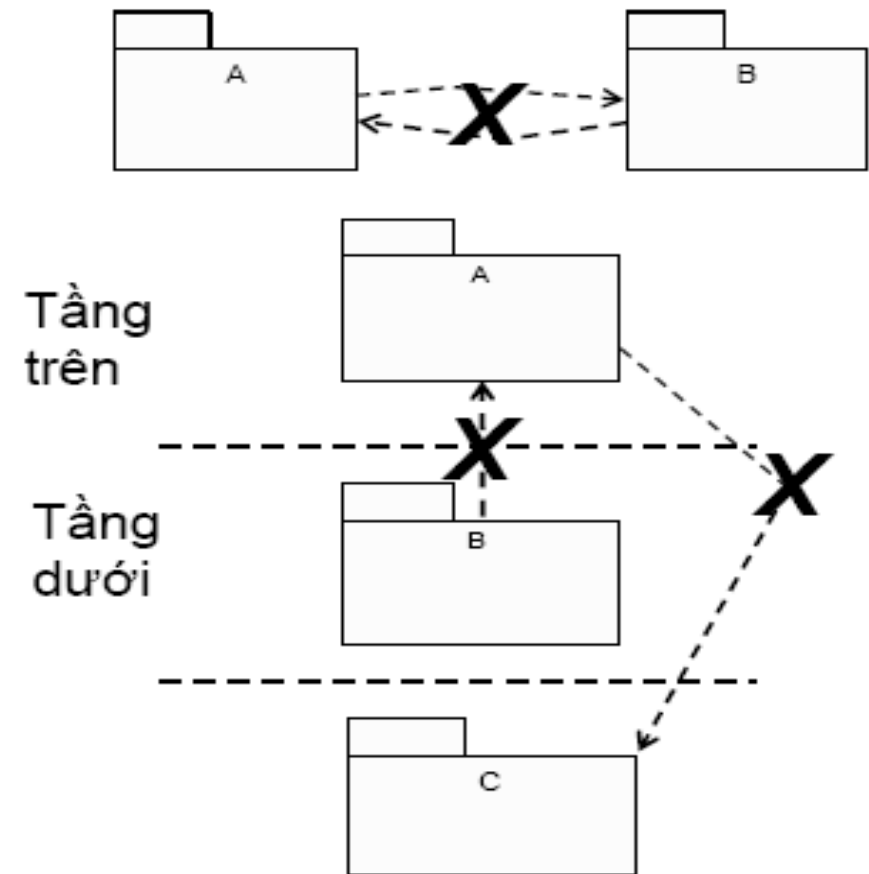
▶ 3. Phân rã các hệ thống con

☐ Nhóm các lớp thiết kế (package)

- Nhóm các lớp trong cùng một gói nếu
 - ☐ Thay đổi một lớp làm ảnh hưởng tới các lớp còn lại
 - ☐ Đối tượng của lớp này tương tác tới các đối tượng của lớp khác
 - ☐ Chúng tương tác với cùng một tác nhân
 - ☐ Chúng có quan hệ chặt chẽ với nhau
 - ☐ Lớp này tạo ra lớp kia
- Việc gộp nhóm hiệu quả cho phép chúng ta có thể quản lý khả năng sử dụng lại và bảo trì hệ thống.

▶ 3. Phân rã các hệ thống con

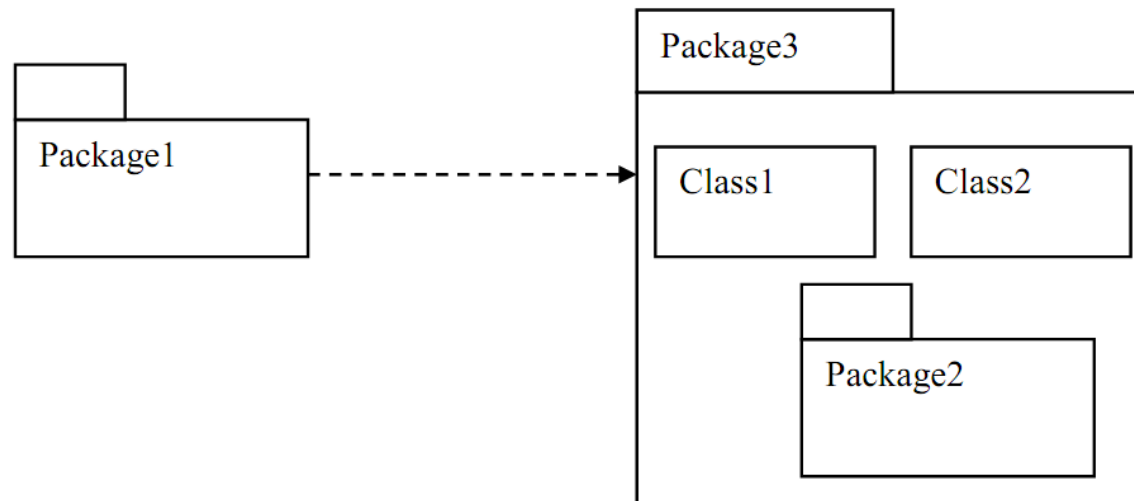
- ❑ Nhóm các lớp thiết kế (package)
 - Các gói không nên quan hệ chéo lẫn nhau
 - Nhóm ở tầng dưới không nên phụ thuộc vào tầng trên
 - Tránh sự phụ thuộc bỏ qua các tầng ở giữa



X = Vi phạm nguyên tắc quan hệ

▶ 3. Phân rã các hệ thống con

- ❑ Xây dựng **biểu đồ gói** (package diagram)
 - Một cụm (layer)
 - Một hệ thống con (subsystem)
 - Thư viện dùng lại (resusable library)
 - Khung (framework)
 - Các lớp cần triển khai cùng nhau



▶ 3. Phân rã các hệ thống con

- ❑ Xây dựng **biểu đồ gói** (package diagram)

■ **Packages**

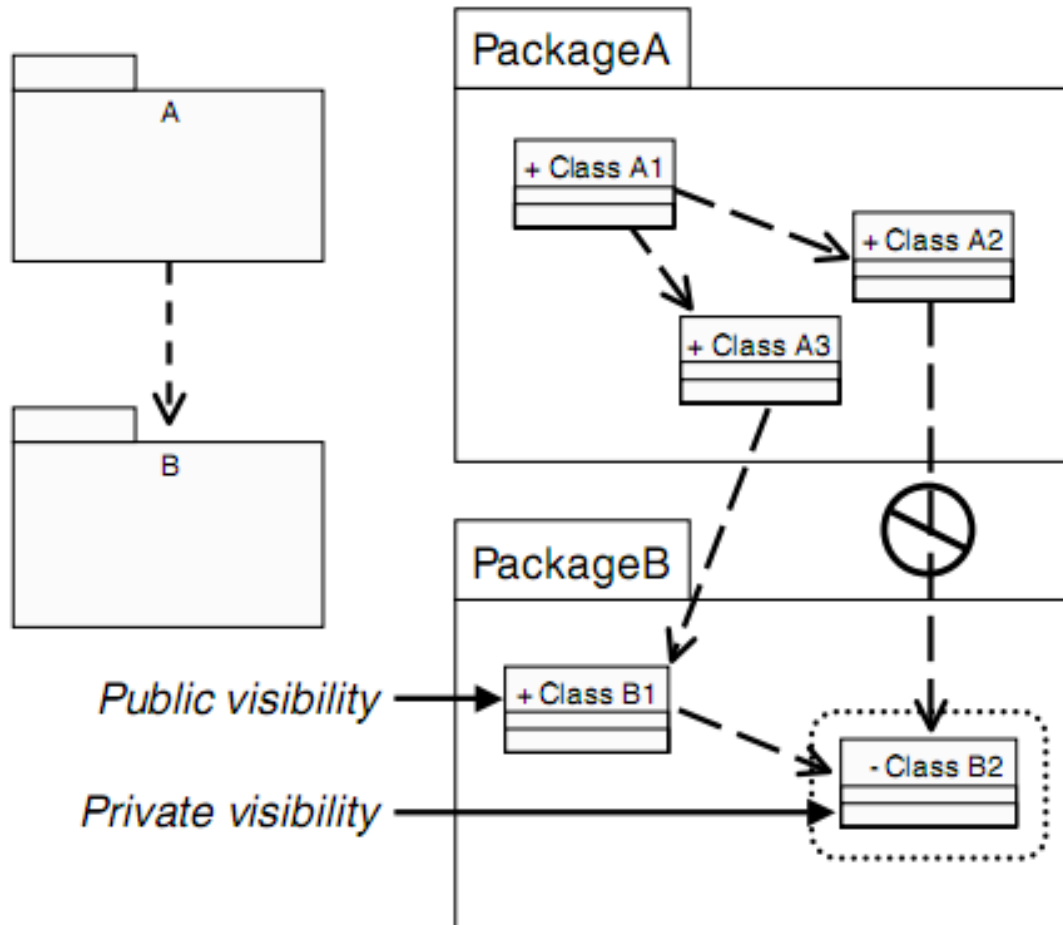
- ❑ Là cơ chế tổ chức các phần tử mô hình thành các nhóm
- ❑ Nhóm các phần tử có ngữ nghĩa gần nhau và có xu hướng thay đổi cùng nhau
- ❑ Package nên low coupling và high cohesive, kiểm soát truy cập nội dung

■ **Các phần tử bên trong Package**

- ❑ Package có thể chứa các phần tử: classes, interfaces, components, nodes, collaborations, use cases, diagrams và packages khác.
- ❑ Một tạo thành một không gian tên, do đó các phần tử cùng loại phải có tên duy nhất.
- ❑ Ví dụ, không thể có hai lớp trong cùng một package có cùng tên.

▶ 3. Phân rã các hệ thống con

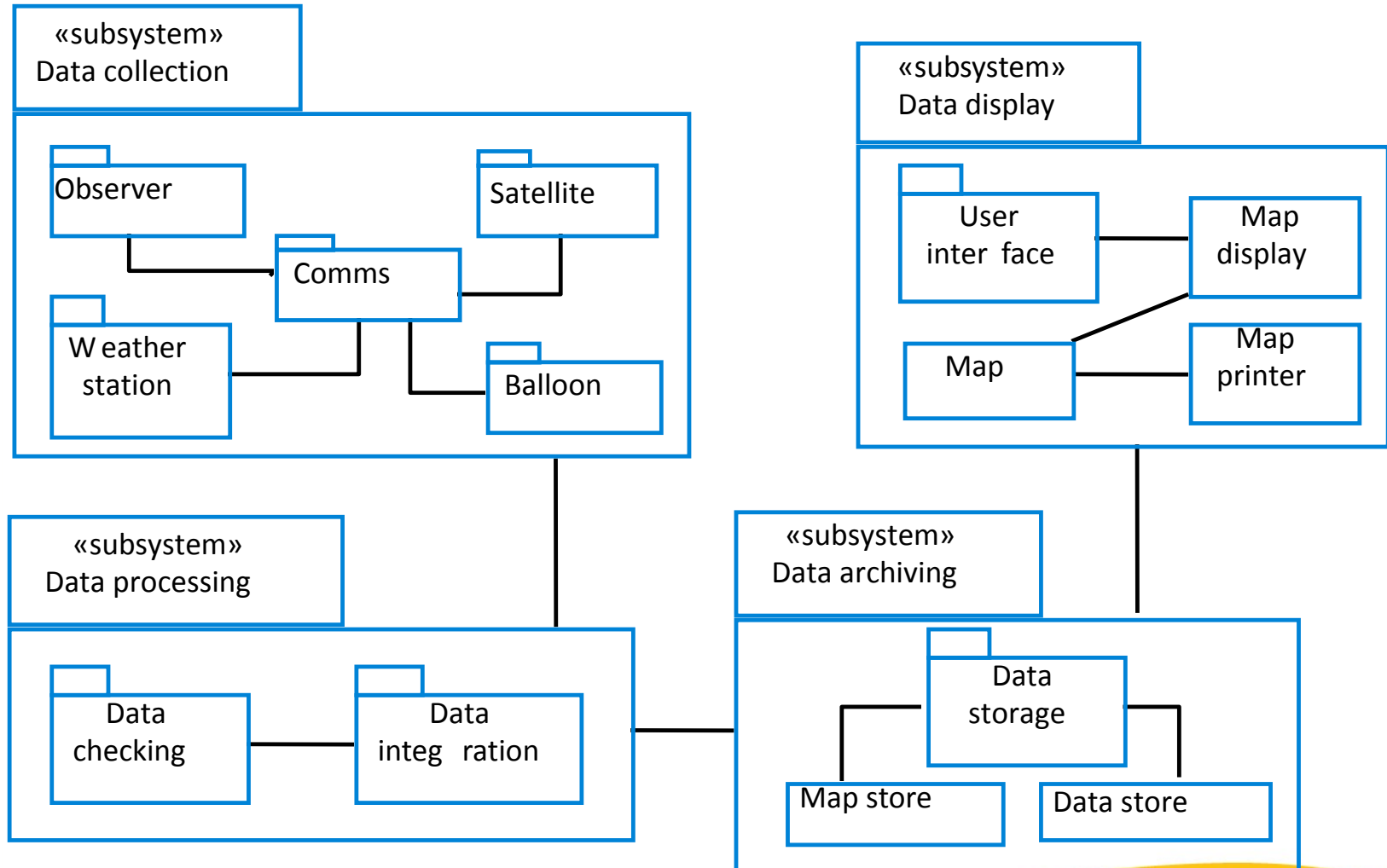
□ Ví dụ



*Only public classes
can be referenced
outside of the
owning package*

3. Phân rã các hệ thống con

- ❑ Ví dụ: Các hệ thống con trong hệ thống bản đồ thời tiết

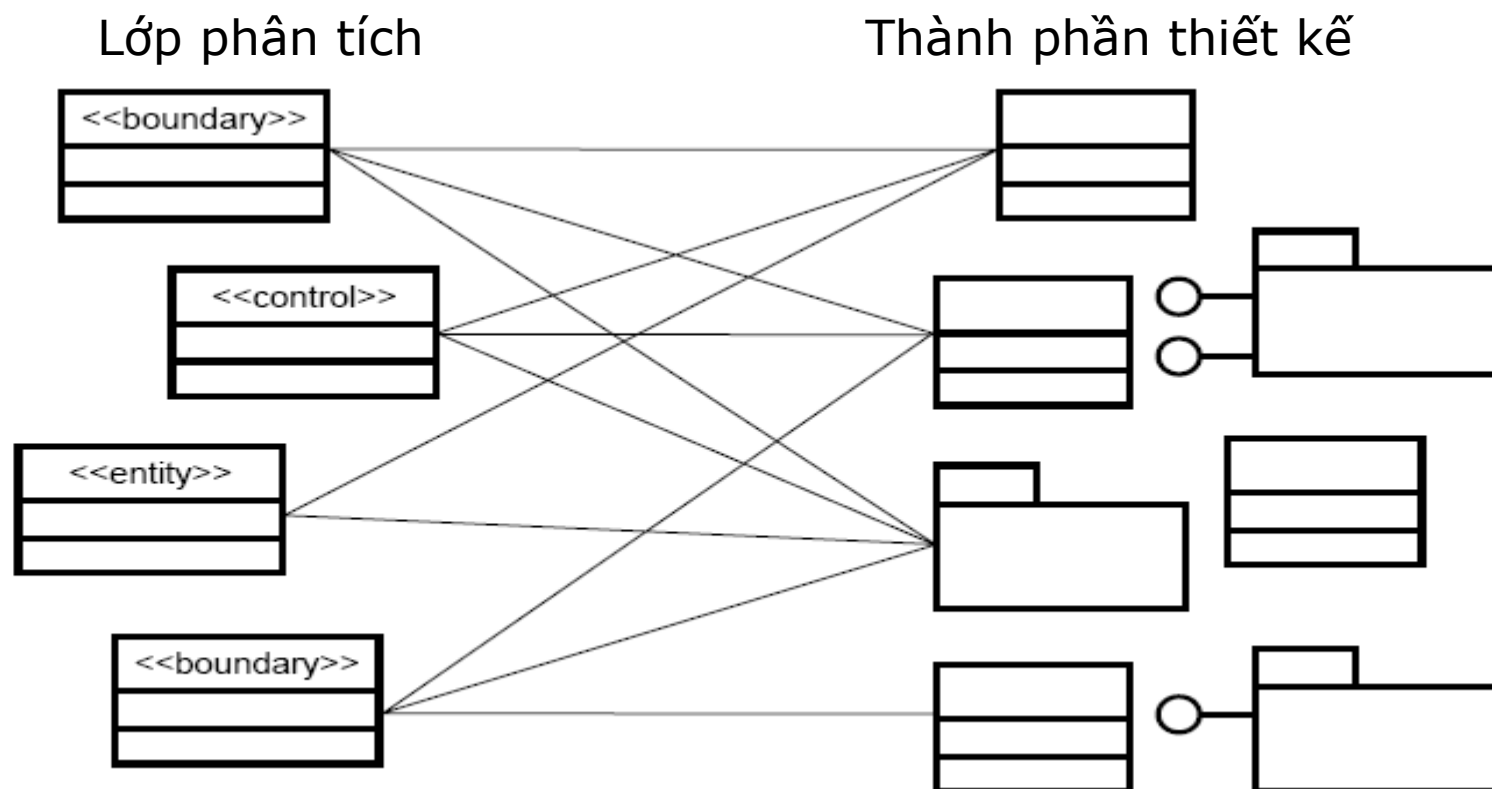


► Nội dung

1. Tổng quan quá trình thiết kế
2. Lựa chọn kiến trúc
3. Phân rã các hệ thống con
4. Thiết kế chi tiết

▶ 4. Thiết kế chi tiết

- ❑ Chuyển đổi lớp phân tích thành các thành phần thiết kế



► Thiết kế chi tiết các lớp

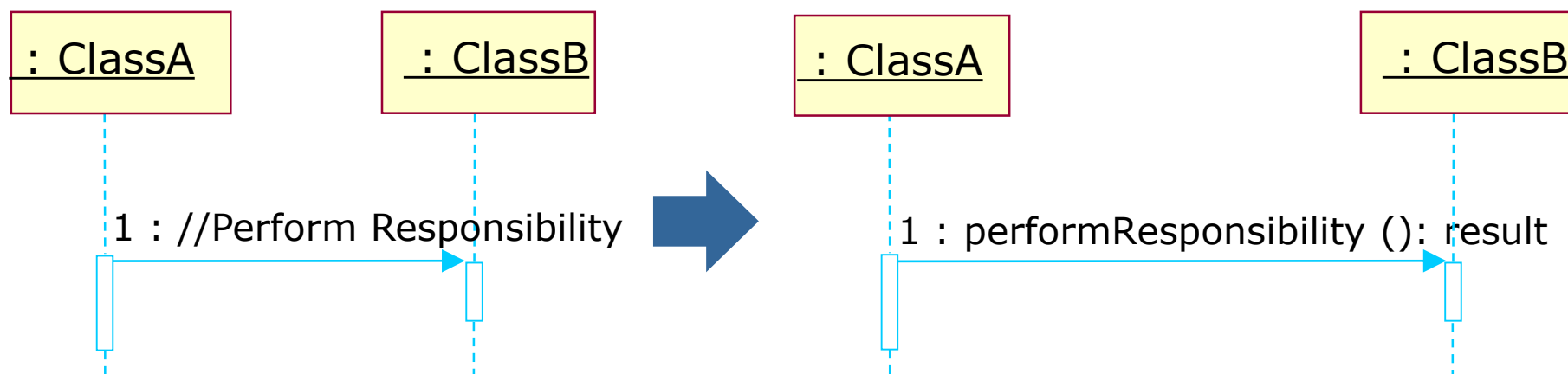
□ Biểu đồ lớp (class diagram)

- Độc lập với ngôn ngữ
- Những ký hiệu cho phép đặc tả lớp, dữ liệu hay thuộc tính của chúng (private) và phương thức (method) sự thừa kế...
- Những sơ đồ đưa ra những hình ảnh về quan hệ cấu trúc và những ứng xử về chức năng của các lớp

Thiết kế chi tiết các lớp

□ 4.1. Xác định các thao tác

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



Các chức năng khác phụ thuộc vào việc thực thi:

Có cần tạo ra một bản sao cho một thể hiện của lớp hay không?

Có cần kiểm tra hai thể hiện của lớp có bằng nhau hay không?

...

► Thiết kế chi tiết các lớp

- ❑ Tên và mô tả thao tác
- ❑ 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

► Thiết kế chi tiết các lớp

☐ Hướng dẫn thiết kế chữ ký thao tác

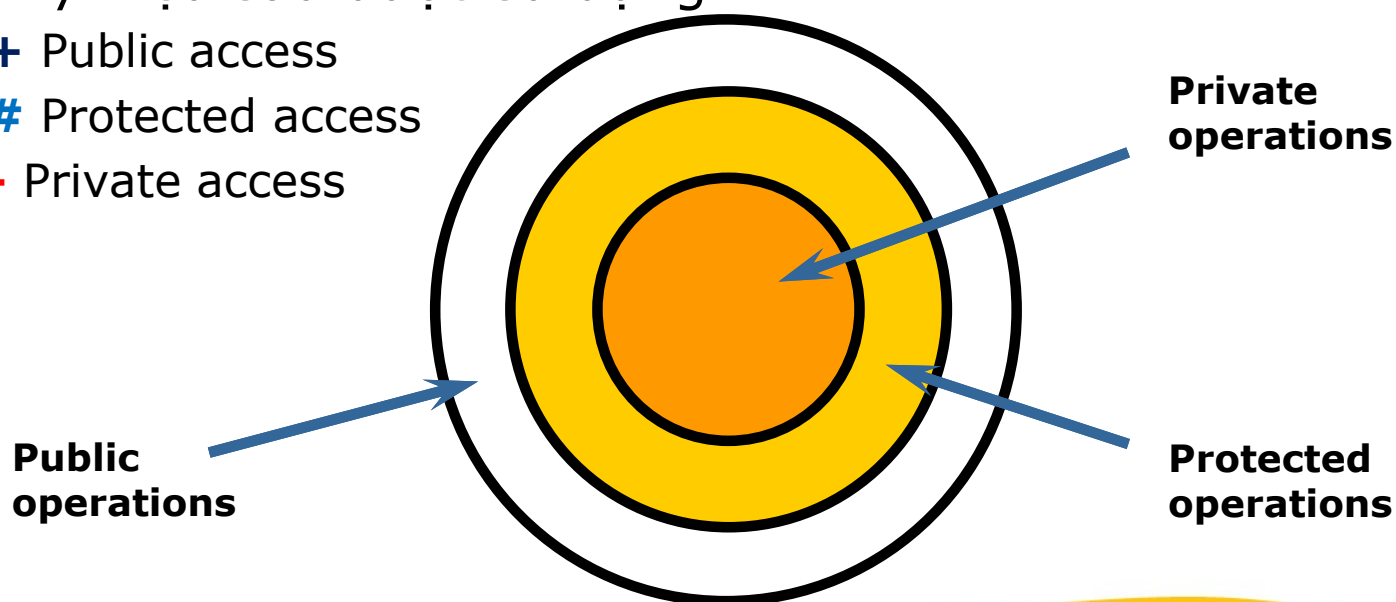
- Khi thiết kế chữ ký của thao tác, cần xem xét liệu tham số có:
 - ☐ Được truyền theo tham trị hay tham biến
 - ☐ Có bị thay đổi bởi thao tác hay không
 - ☐ Có tùy chọn không
 - ☐ Thiết lập các giá trị mặc định
 - ☐ Các khoảng tham số không hợp lệ
- Càng ít tham số, càng tốt
- Truyền các đối tượng thay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

► Thiết kế chi tiết các lớp

❑ 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**
- Các ký hiệu sau được sử dụng:

- ❑ **+** Public access
- ❑ **#** Protected access
- ❑ **-** Private access



Class1

- privateAttribute
+ publicAttribute
protectedAttribute

- privateOperation ()
+ publicOperation ()
protecteOperation ()

► Thiết kế chi tiết các lớp

□ 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.

| Class1 |
|---|
| <u>- classifierScopeAttr</u> - instanceScopeAttr |
| <u>+ classifierScopeOp ()</u> + instanceScopeOp () |

► Thiết kế chi tiết các lớp

□ 4.2. Xác định các thuộc tính

- 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.

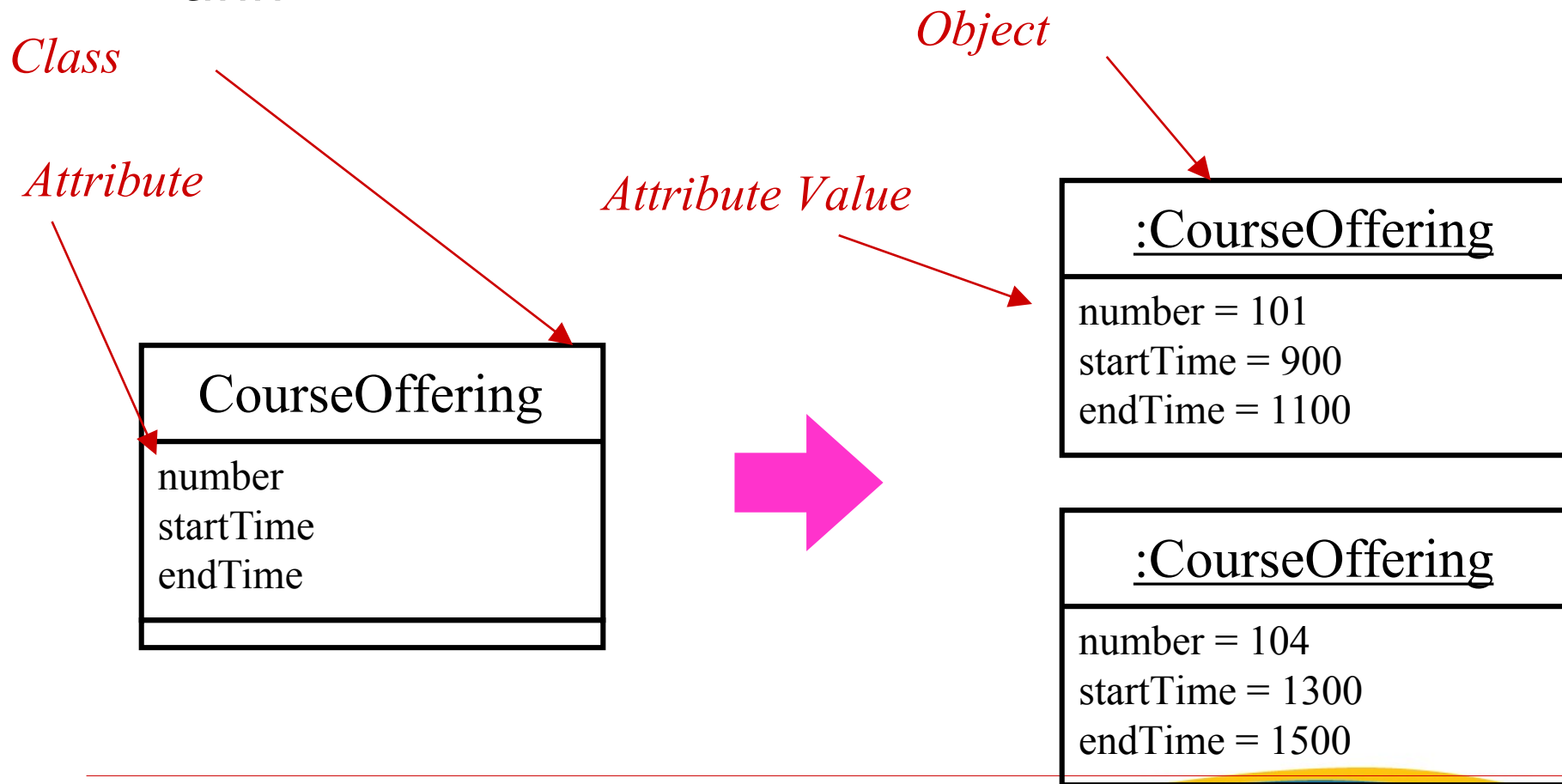


► Thiết kế chi tiết các lớp

- ❑ 4.2. Xác định các thuộc tính - Biểu diễn các thuộc tính
 - 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: #

Thiết kế chi tiết các lớp

4.2. Xác định các thuộc tính - Biểu diễn các thuộc tính



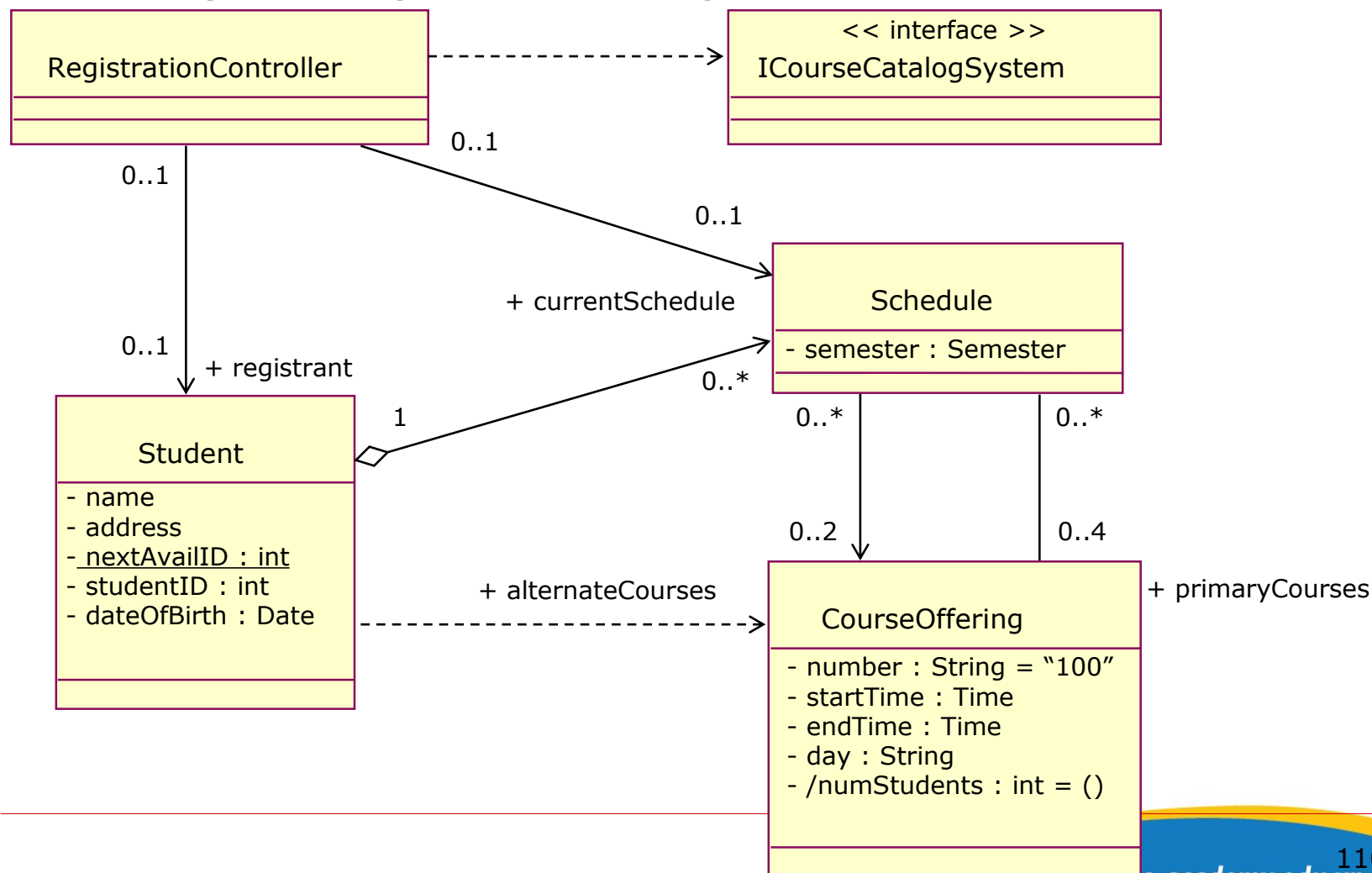
► Thiết kế chi tiết các lớp

- ❑ 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.












Thiêt kế chi tiết các lớp

□ Ví dụ xác định các thuộc tính



► Thiết kế chi tiết các lớp

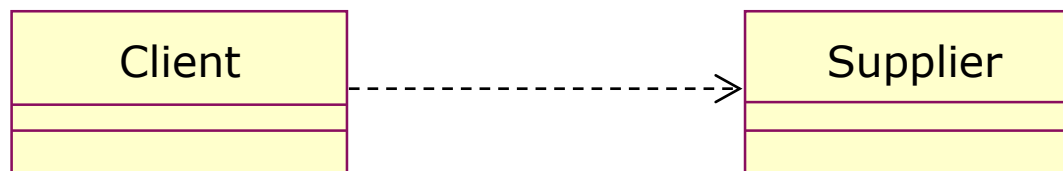
□ 4.3. Xác định quan hệ giữa các lớp

| | | | |
|------------------|--|----|---|
| ■ Association |  | OR |  |
| ■ Aggregation |  | OR |  |
| ■ Composition |  | OR |  |
| ■ Generalization |  | | |
| ■ Dependency |  | | |
| ■ Realization |  | | |

► Thiết kế chi tiết các lớp

□ Phụ thuộc (Dependency)

- 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.



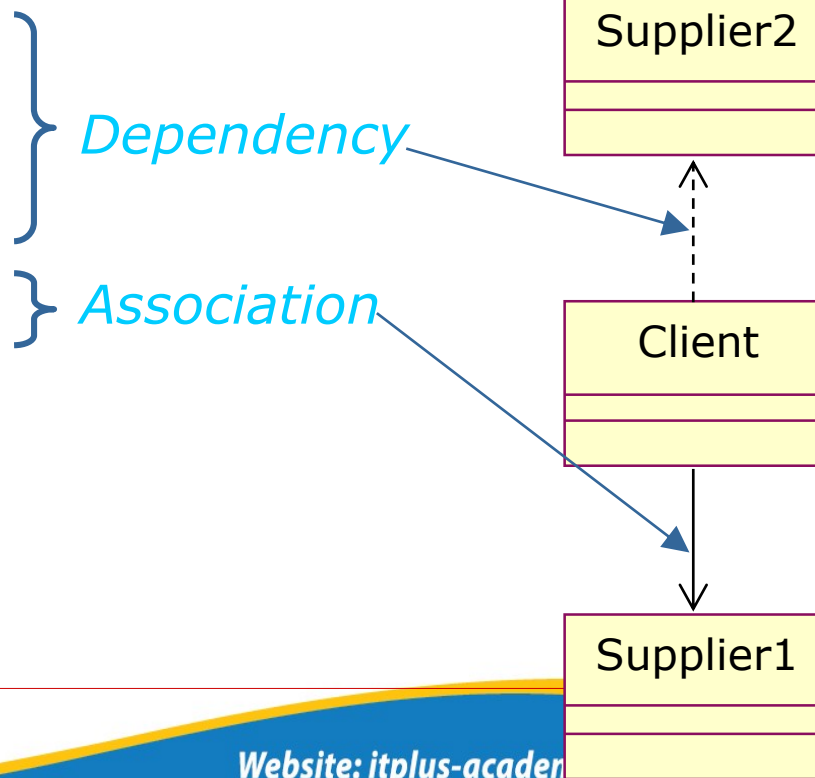
- Mục đích
 - Xác định khi các mối quan hệ cấu trúc (association hoặc aggregation) không cần đến.
- Cần xem xét:
 - Cái gì làm cho supplier có thể được nhìn thấy client?

► Thiết kế chi tiết các lớp

□ Phụ thuộc (Dependency)

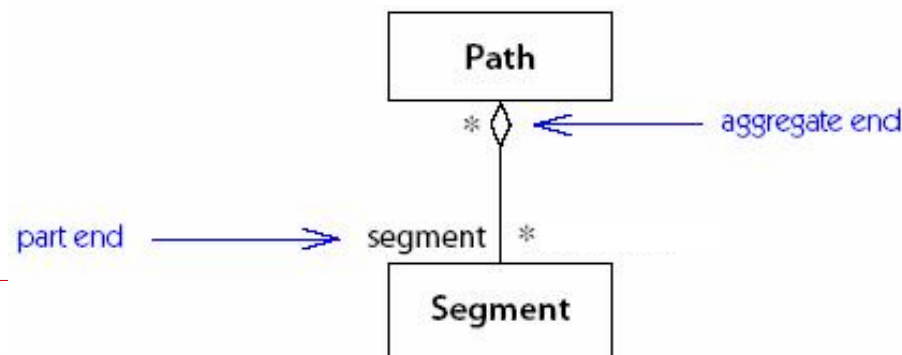
- Kết hợp là mối quan hệ cấu trúc
- Phụ thuộc là mối quan hệ phi-cấu trúc
- Để các đối tượng có thể “biết lẫn nhau”, chúng phải được nhìn thấy

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



► Thiết kế chi tiết các lớp

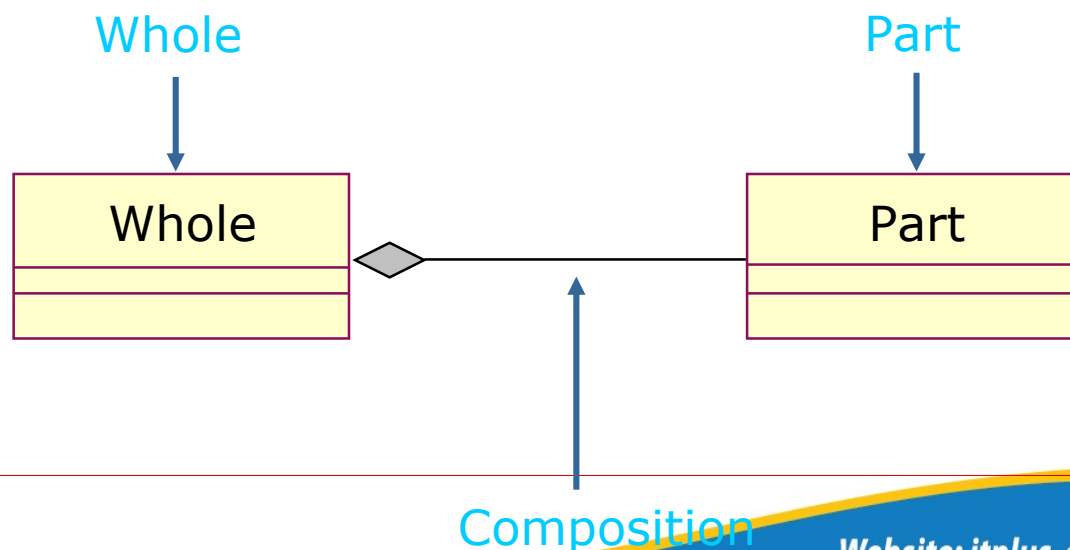
- ❑ Kết tập / Kết hợp (Aggregation) và Hợp thành (Composition)
- Kết tập (aggregation)
 - ❑ Tạo ra các đối tượng của các lớp có sẵn trong lớp mới → thành viên của lớp mới.
 - ❑ Kết tập tái sử dụng thông qua *đối tượng*
 - ❑ Sử dụng “hình thoi” tại đầu của lớp toàn thể
- Composition là gì? Một dạng của 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.



► Thiết kế chi tiết các lớp

□ Sử dụng composition khi:

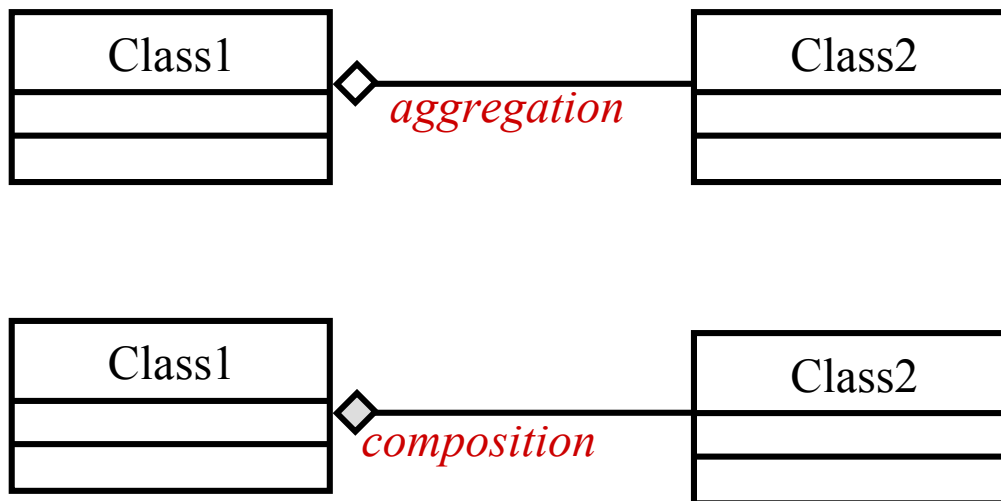
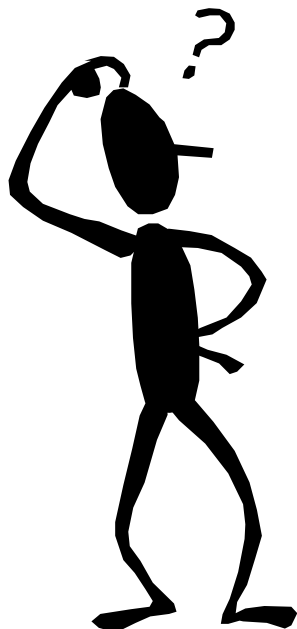
- Các đặc tính (property) cần định danh độc lập
- Nhiều lớp có cùng các đặc tính
- Các đặc tính có một cấu trúc phức tạp và các đặc tính của riêng chúng
- Các đặc tính phải có hành vi phức tạp của riêng chúng
- Các đặc tính có mối quan hệ của riêng chúng



► Thiết kế chi tiết các lớp

□ Aggregation hay Composition?

- Xem xét chu kỳ sống của Class1 và Class2



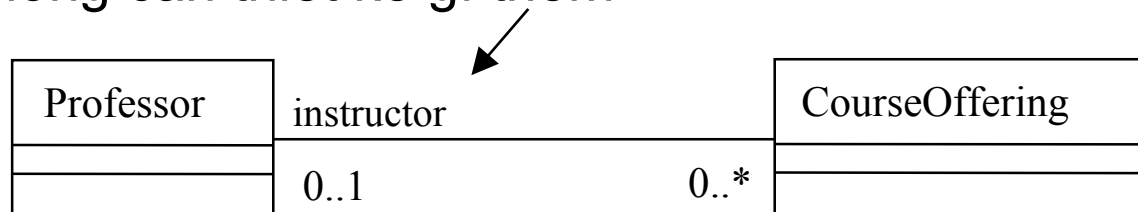
► Thiết kế chi tiết các lớp

□ Thiết kế bản số / bội số của quan hệ

■ Bản số = 1, hay = 0..1

□ Có thể cài đặt đơn giản như một giá trị hay pointer

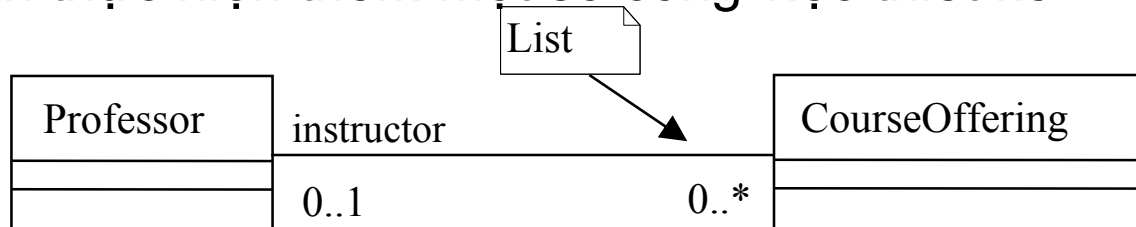
□ Không cần thiết kế gì thêm



■ Bản số > 1

□ Không thể dùng giá trị đơn hay pointer

□ Cần thực hiện thêm một số công việc thiết kế

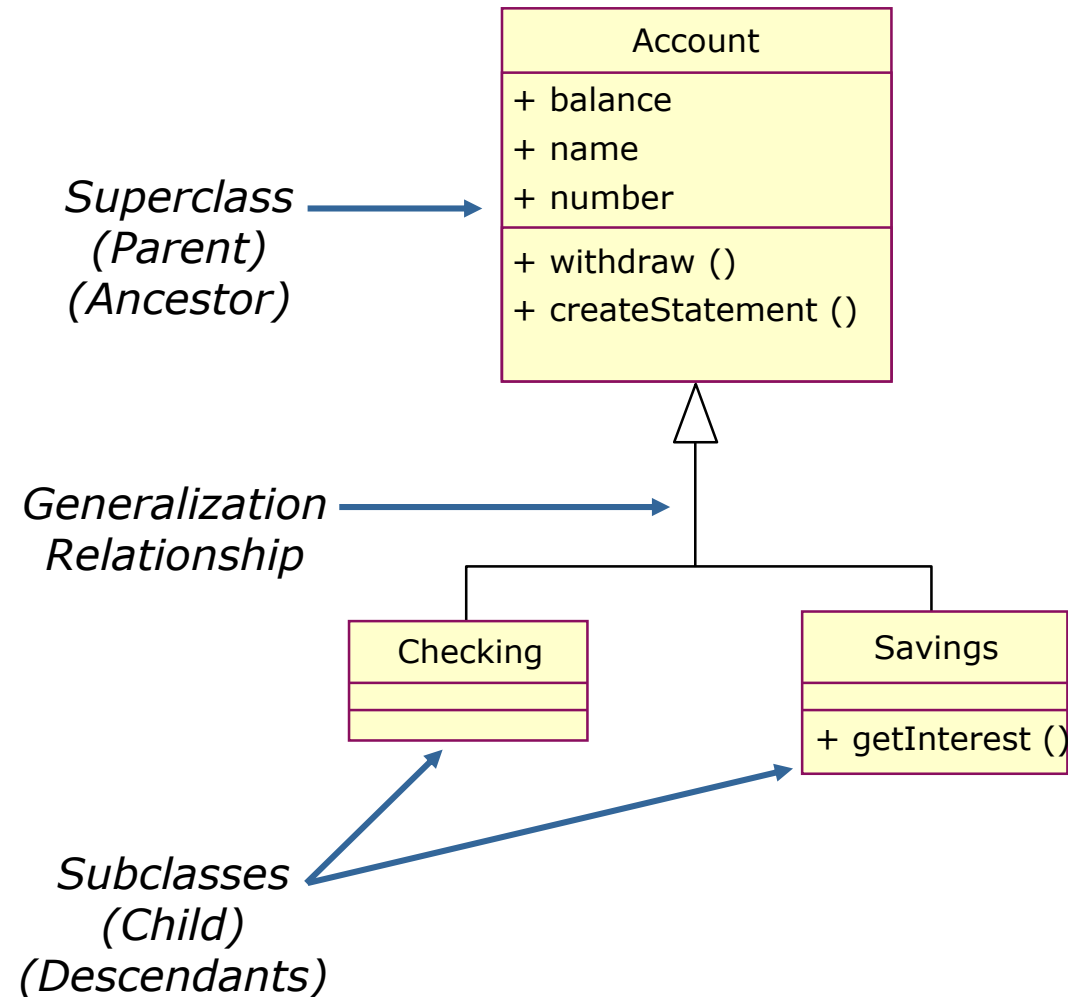


*Cần một
container*

Thiết kế chi tiết các lớp

□ Kế thừa / Tổng quát hoá (Generalization):

- Một lớp chia sẻ thuộc tính và phương thức với 1 hoặc nhiều lớp khác
- Quan hệ "is kind of"



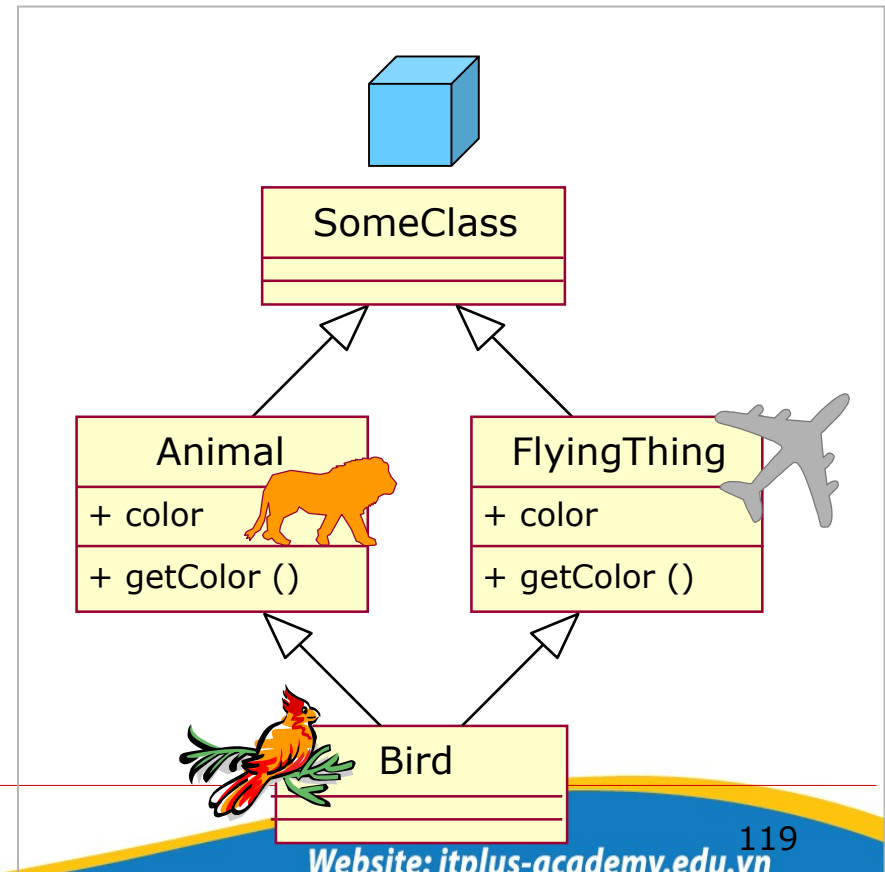
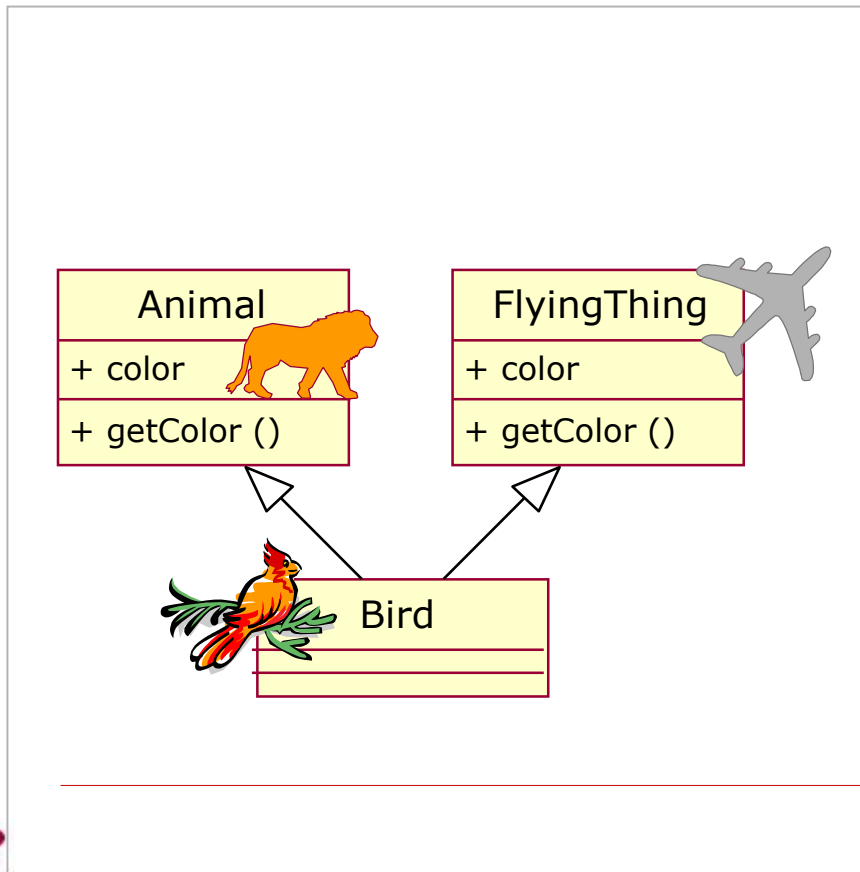
Thiết kế chi tiết các lớp

□ Kế thừa / Tổng quát hoá (Generalization):

■ Đơn kế thừa và đa kế thừa

Name clashes

Repeated inheritance



► Thiết kế chi tiết các lớp

□ Mô tả Sơ đồ lớp

- Danh sách các lớp đối tượng và quan hệ
- Mô tả chi tiết từng lớp đối tượng và quan hệ

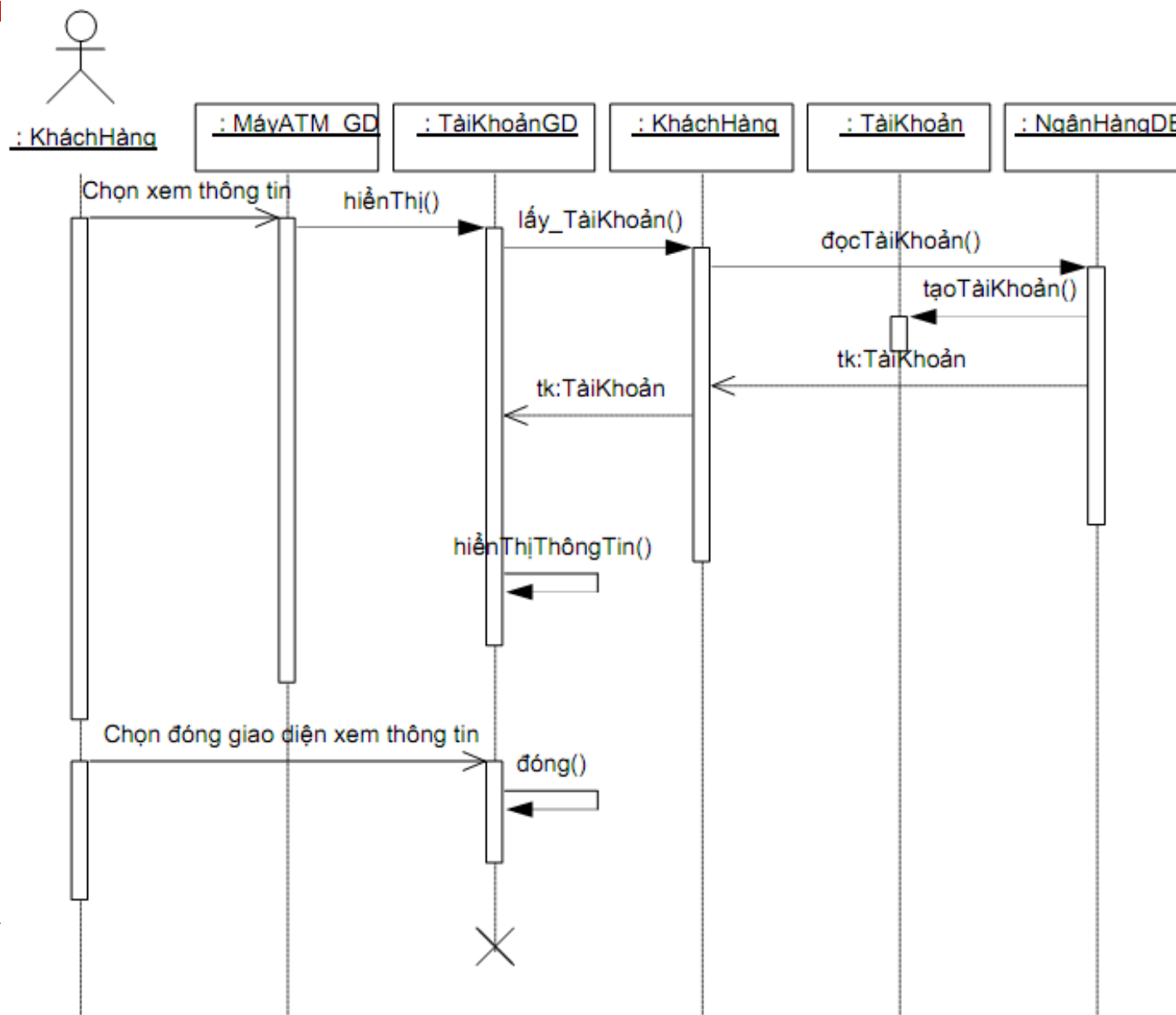
| STT | Tên lớp / Quan hệ / Thuộc tính / Phương thức | Kiểu | Ràng buộc | Ý nghĩa / Ghi chú |
|-----|---|------|-----------|----------------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

- 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
 - Ánh xạ 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

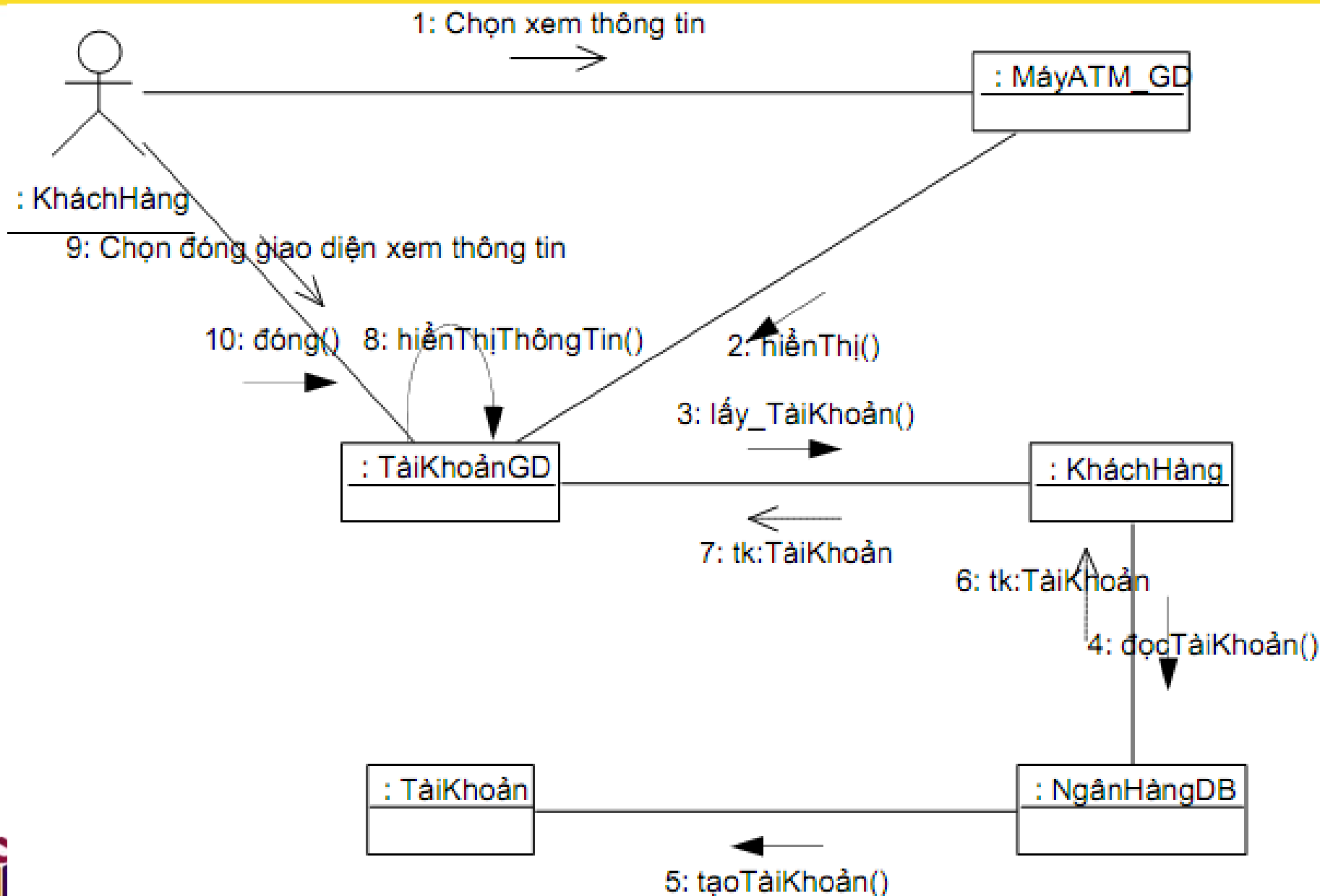
▶ Bài tập 1

- Mô tả hiện thực hoá của use case “Truy vấn thông tin tài khoản” trong Hệ thống AMT
 - Đặc tả kịch bản thực thi của ca sử dụng
 - Phân tích các thành phần tham gia trong ca sử dụng (biểu đồ trình tự / biểu đồ giao tiếp)
 - Xác định các lớp phân tích → Các lớp thiết kế

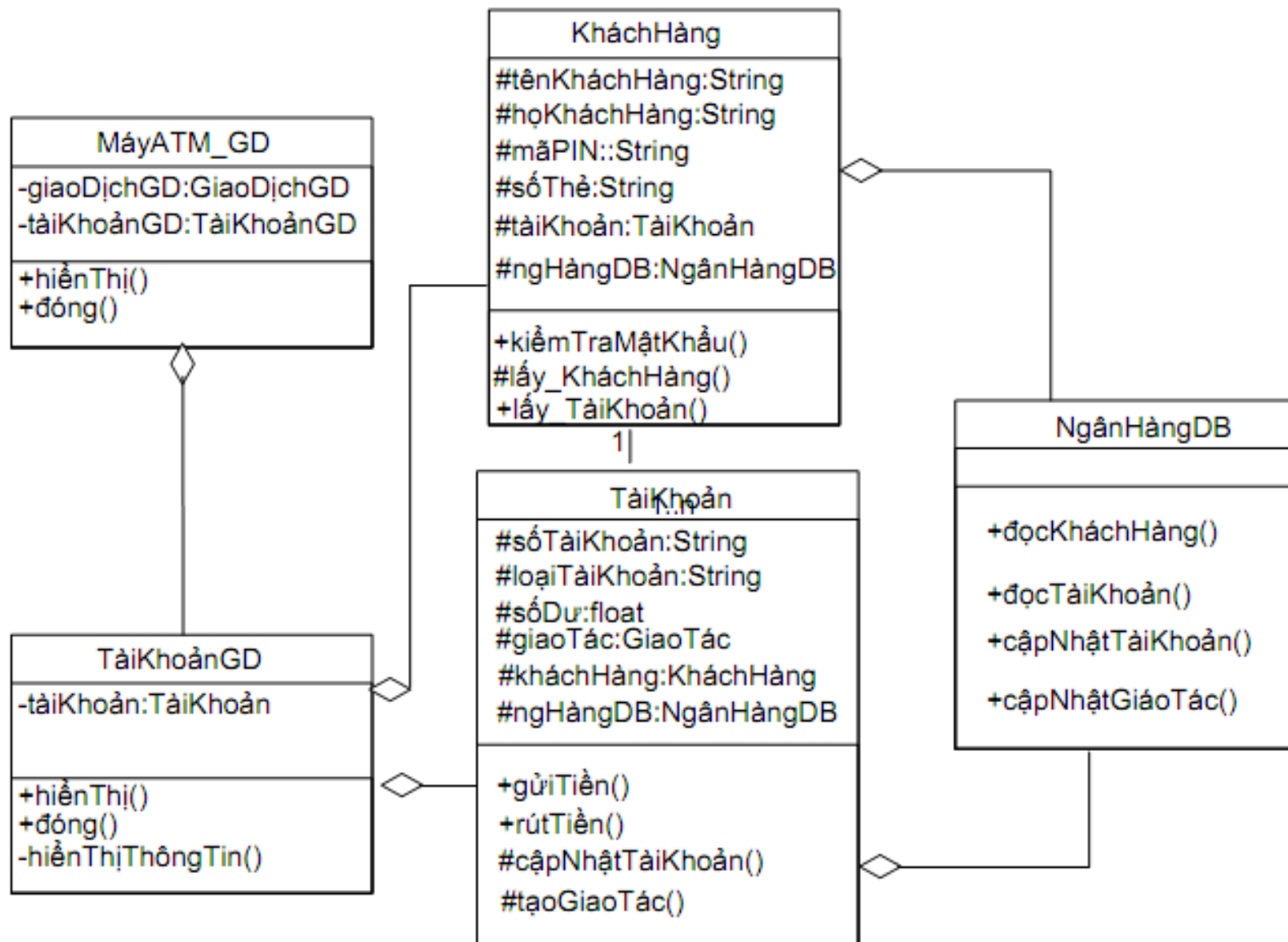
► Biểu đồ trình tự



▶ Biểu đồ giao tiếp



► Biểu đồ lớp



```

classDiagram
    class MayATM_GD {
        +GiaoDienGD: GiaoDienGD
        +TaiKhoanGD: TaiKhoanGD
    }
    class GiaoDienGD {
        +TaiKhoanGD: TaiKhoanGD
        +KhoanHangGD: KhoanHangGD
    }
    class TaiKhoanGD {
        +TaiKhoanGD: TaiKhoanGD
        +KhoanHangGD: KhoanHangGD
    }
    class KhoanHangGD {
        +TaiKhoanGD: TaiKhoanGD
        +KhoanHangGD: KhoanHangGD
    }
    class MayATMKhongDongGD {
        +MayATM: MayATM
    }
    class MayATM {
        +TaiKhoanGD: TaiKhoanGD
        +KhoanHangGD: KhoanHangGD
    }

    MayATM_GD --|> GiaoDienGD
    MayATM_GD --|> TaiKhoanGD
    GiaoDienGD --|> TaiKhoanGD
    GiaoDienGD --|> KhoanHangGD
    TaiKhoanGD --|> KhoanHangGD
    MayATMKhongDongGD --|> MayATM
    
```

The diagram illustrates the structure of the 'Tàng giao điện' (Electronic Trading) system. It features several classes and their relationships:

- MayATM_GD**: This class is a generalization of **GiaoDienGD** and **TaiKhoanGD**. It contains two attributes: **+GiaoDienGD: GiaoDienGD** and **+TaiKhoanGD: TaiKhoanGD**.
- GiaoDienGD**: This class is a generalization of **TaiKhoanGD** and **KhoanHangGD**. It contains two attributes: **+TaiKhoanGD: TaiKhoanGD** and **+KhoanHangGD: KhoanHangGD**.
- TaiKhoanGD**: This class is a generalization of **KhoanHangGD**. It contains two attributes: **+TaiKhoanGD: TaiKhoanGD** and **+KhoanHangGD: KhoanHangGD**.
- KhoanHangGD**: This class is a generalization of **MayATMKhongDongGD**. It contains two attributes: **+TaiKhoanGD: TaiKhoanGD** and **+KhoanHangGD: KhoanHangGD**.
- MayATMKhongDongGD**: This class is a generalization of **MayATM**. It contains one attribute: **+MayATM: MayATM**.
- MayATM**: This class is a generalization of **MayATMKhongDongGD**. It contains two attributes: **+TaiKhoanGD: TaiKhoanGD** and **+KhoanHangGD: KhoanHangGD**.

Relationships are indicated by lines with diamonds at the ends, representing generalization or association. The diagram shows a hierarchy where **MayATM_GD** is the root, branching into **GiaoDienGD** and **TaiKhoanGD**, which further branch into **KhoanHangGD** and **MayATM**.

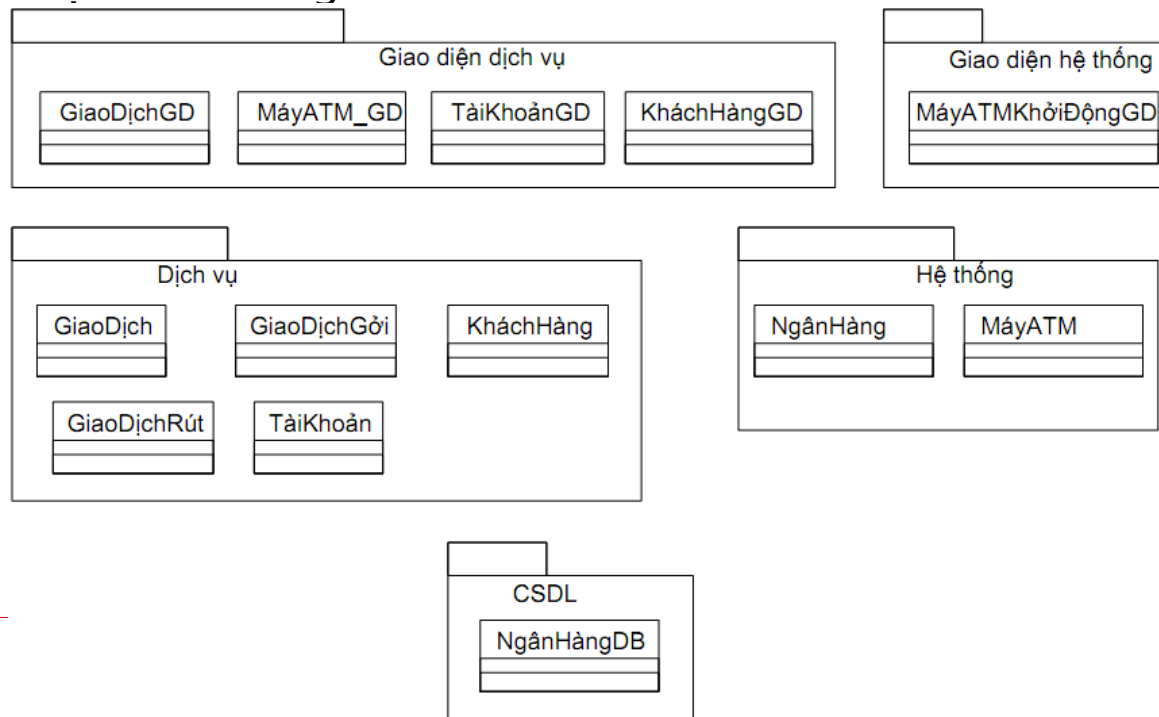
```

classDiagram
    class NgânHàng {
        +tên()
        +địaCh()
    }
    class KháchHàng {
        +tênKháchHàng()
        +sốKháchHàng()
        +sốThẻ()
        +tênKháchHàng()
        +tênNgânHàng()
    }
    class TàiKhoản {
        +sốTàiKhoản()
        +loạiTàiKhoản()
        +sốDư()
        +tênKháchHàng()
        +tênNgânHàng()
    }
    class MáyATM {
        +địaCh()
        +tênThẻ()
        +sốTênHết()
    }
    class GiaoDich {
        +giaoDichID()
        +ngàyGiaoDich()
        +thờiGianGiaoDich()
        +loạiGiaoDich()
        +sốTiền()
        +sốDư()
    }
    class GiaoDichRut {
    }
    class GiaoDichGhi {
    }
    NgânHàng "1" -- "*" KháchHàng
    KháchHàng "1" -- "1..n" TàiKhoản
    KháchHàng "1" -- "0..n" MáyATM
    TàiKhoản "1" -- "0..n" GiaoDich
    GiaoDich <|-- GiaoDichRut
    GiaoDich <|-- GiaoDichGhi
  
```

Tổng cộng cấp độ nhận

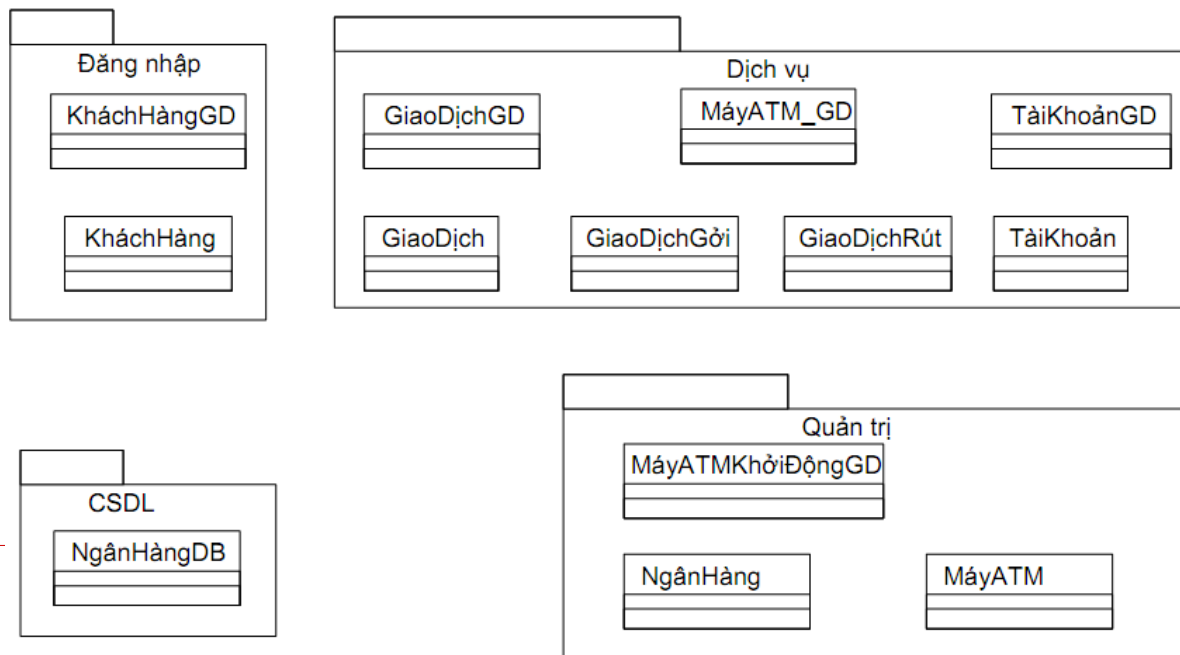
▶ Bài tập 2

- Xây dựng biểu đồ gói cho hệ thống ATM: tùy thuộc chiến lược thiết kế
 - Nếu giao diện của hệ thống sẽ bị thay thế, chịu thay đổi lớn trong tương lai thì giao diện nên được thiết kế tách biệt với các thành phần còn lại của mô hình thiết kế.
 - Do đó, khi giao diện có sự thay đổi, chỉ có các gói của giao diện bị ảnh hưởng.



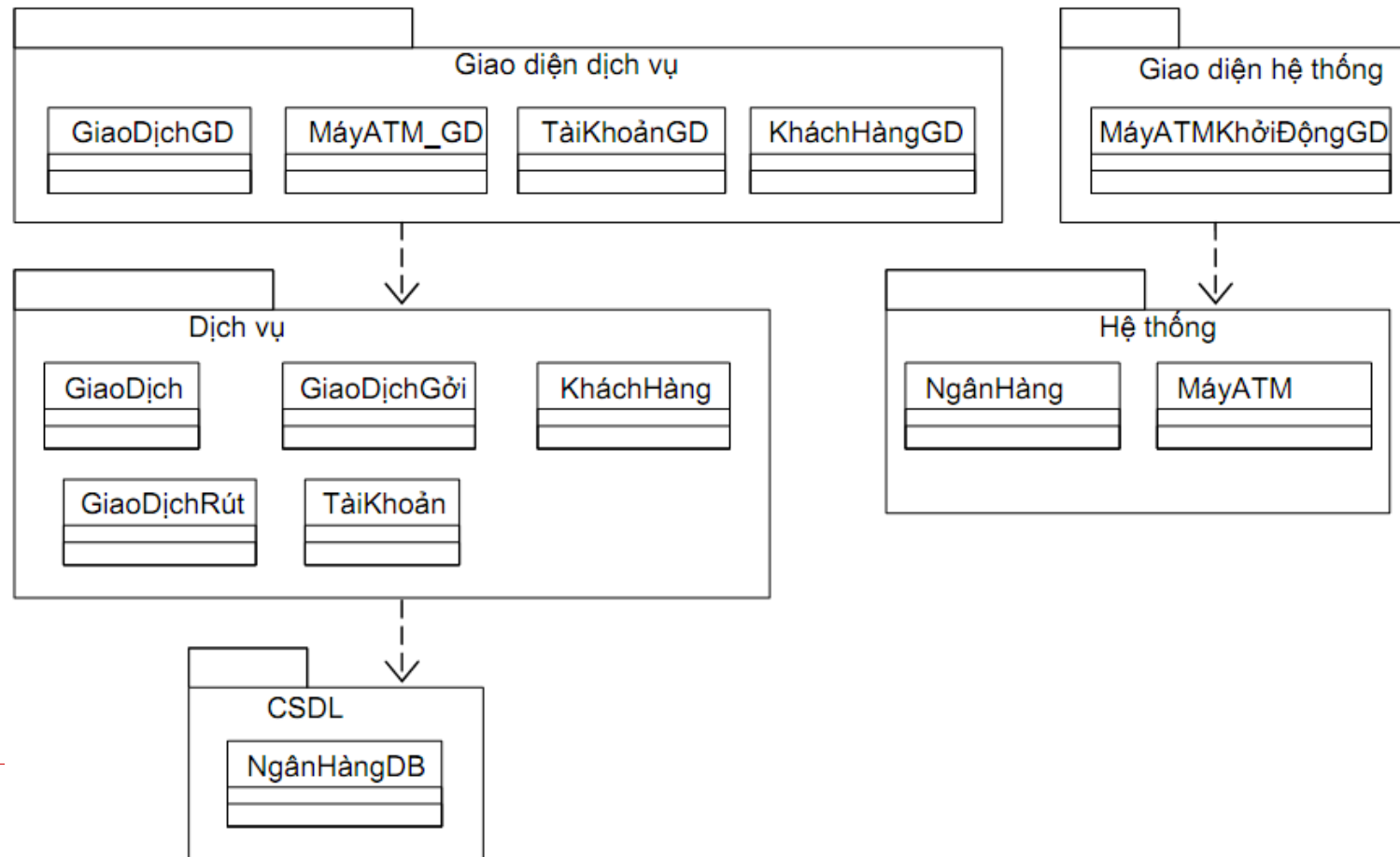
▶ Bài tập 2

- Xây dựng biểu đồ gói cho hệ thống ATM: tùy thuộc chiến lược thiết kế
 - Nếu các giao diện được xác định là sẽ không có sự thay đổi và sẽ ổn định trong tương lai.
 - Thì một thay đổi tới hệ thống nên được hiểu là một thay đổi bên trong thay vì thay đổi chỉ giao diện.
 - Do đó, các lớp ở tầng giao diện sẽ được gom nhóm chung với các lớp tầng nghiệp vụ thành một gói.



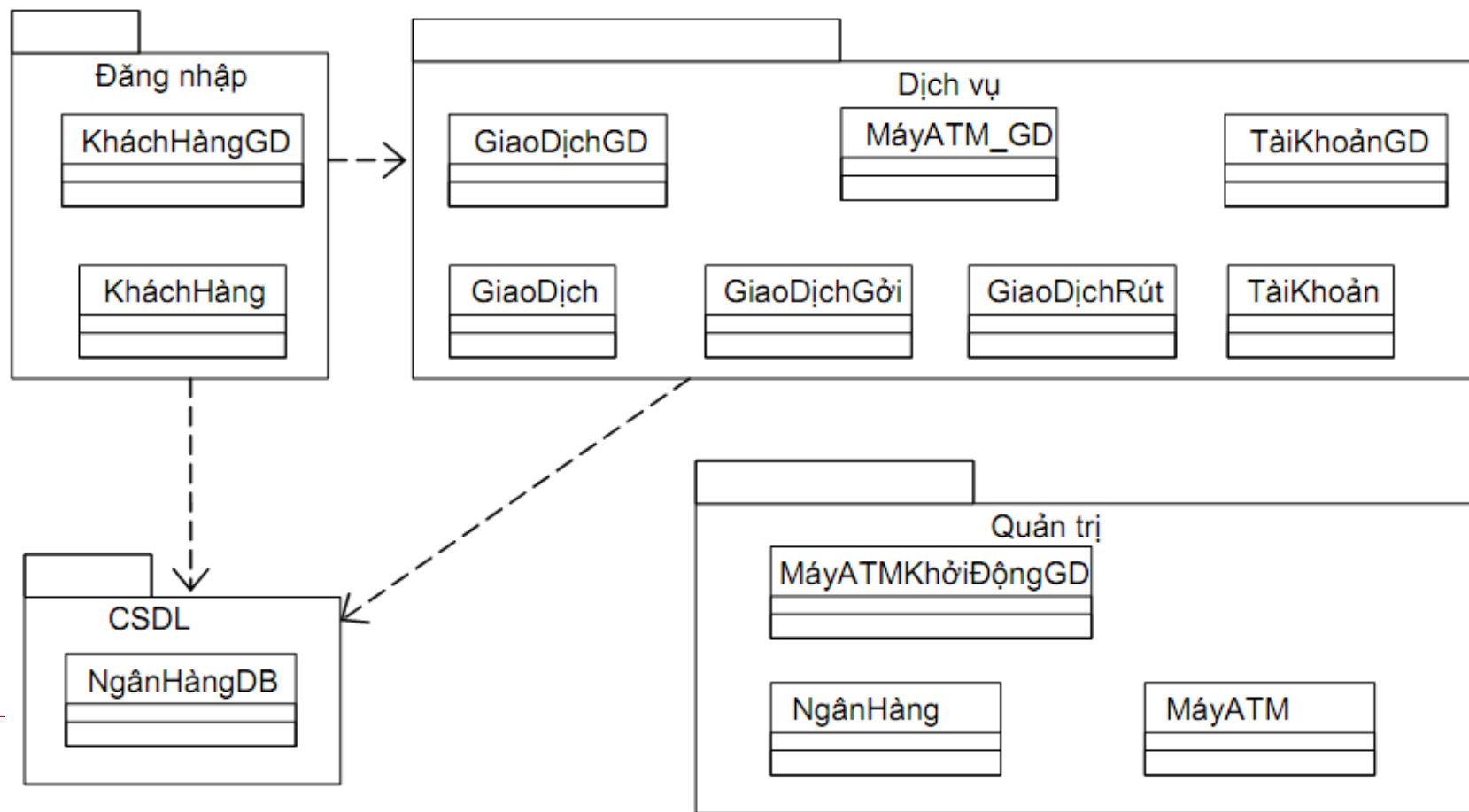
▶ Bài tập 2

- ❑ Xây dựng biểu đồ gói cho hệ thống ATM: Xác định phụ thuộc giữa các gói
- ❑ Giải pháp 1:



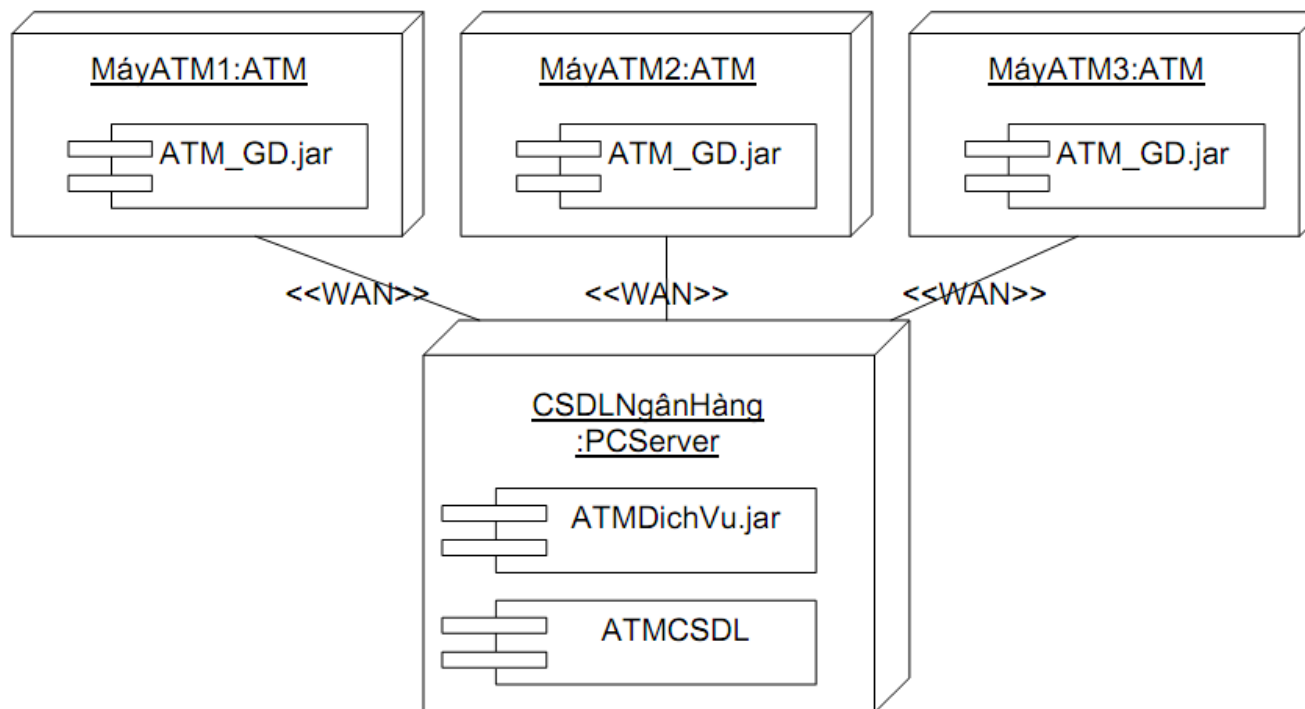
▶ Bài tập 2

- ❑ Xây dựng biểu đồ gói cho hệ thống ATM: Xác định phụ thuộc giữa các gói
- ❑ Giải pháp 2:



► Bài tập 3

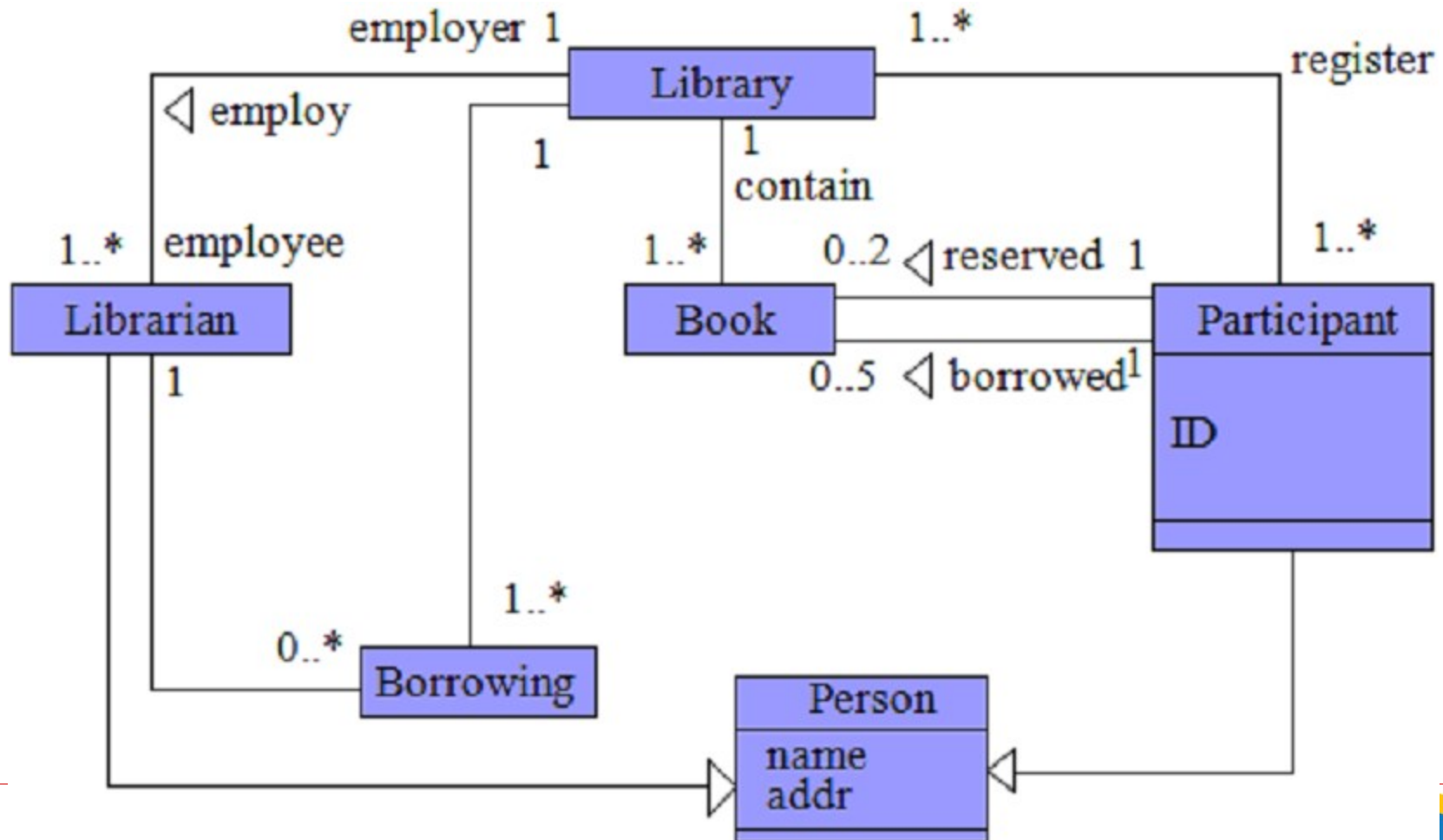
- ❑ Xây dựng mô hình triển khai trong hệ thống ATM, các địa điểm triển khai bao gồm: một ngân hàng, ba vị trí đặt máy ATM.
- Trong mô hình này, chúng ta chỉ cần biểu diễn một vị trí cho máy ATM, các vị trí ATM khác chỉ là một bản sao và giống nhau cho tất cả các máy ATM.



- Mô hình hóa biểu đồ các lớp thực thể cho hệ thống quản lý thư viện
 - Người quản lý thư viện mong muốn tự động hóa việc mượn sách
 - Họ yêu cầu một phần mềm cho phép người sử dụng biết sách hiện có, có thể đặt mượn 2 quyển sách, những người tham gia mượn sách có thể biết sách nào đã mượn hoặc đã đặt
 - Những người tham gia mượn sách sở hữu một password để truy nhập
 - Việc mượn sách được thực hiện bởi các thủ thư, sau khi xác định người mượn sách, họ biết được người này có được phép mượn hay không? (tối đa 5 quyển), người này được ưu tiên? (đã đặt trước)

▶ Bài tập 4

- ❑ Mô hình hóa biểu đồ các lớp thực thể cho hệ thống quản lý thư viện





► Tài liệu tham khảo

- ❑ Nguyễn Văn Ba, "Phát triển hệ thống hướng đối tượng với UML 2.0 và C++", NXB ĐHQG Hà Nội, 2008.
- ❑ Trần Đình Quế và Nguyễn Mạnh Sơn, "Phân tích và thiết kế hệ thống thông tin".
- ❑ Đặng Văn Đức, "Phân tích thiết kế hướng đối tượng bằng UML", NXB Giáo dục, 2002
- ❑ Martin Fowler, Kendall Scott, "UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language", Publisher: Addison Wesley, 2000.
- ❑ Kim Hamilton, Russell Miles, "Learning UML 2.0", Publisher: O'Reilly, 2006. ISBN: 978-0-59-600982-3.
- ❑ Terry Quatrani, "Visual modeling with Rational Rose 2002 and UML", Publisher: Addison Wesley, 2002. ISBN: 0-201-72932-6
- ❑ <http://www.omg.org>
- ❑ <http://www.uml.org>