TRƯỜNG ĐẠI HỌC KINH TẾ QUỐC DÂN
VIỆN CÔNG NGHỆ THÔNG TIN & KINH TẾ SỐ

CHƯƠNG IV

THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

MỤC ĐÍCH

- **Phân tích** là để trả lời câu hỏi "là gì/làm cái gì" ("what") – tập trung vào các **yêu cầu (chức năng và phi chức năng)** đối với hệ thống
  - Gồm 6 bước đầu tiên (trong 10 bước) của quy trình RUP
- **Thiết kế** là để trả lời câu hỏi "làm thế nào" ("how") – tập trung nghiên cứu **sự thực thi** của hệ thống
  - Đưa ra những quyết định thiết kế phù hợp với các công nghệ được lựa chọn
  - Đáp ứng các yêu cầu phi chức năng (vd: giao diện, hiệu năng, tính sẵn sàng, tính bảo mật,…)

MỤC ĐÍCH

- Thiết kế kiến trúc tổng thể của hệ thống
- Thiết kế giao diện sử dụng
- Thiết kế lớp chi tiết
- Thiết kế việc lưu trữ dữ liệu

4.1 THIẾT KẾ KIẾN TRÚC TỔNG THỂ

- Mục đích của thiết kế kiến trúc tổng thể
- Phân rã hệ thống thành các hệ thống con
- Mô tả các thành phần vật lý của hệ thống
- Bố trí các thành phần khả thi vào các nút phần cứng

MỤC ĐÍCH CỦA THIẾT KẾ KIẾN TRÚC TỔNG THỂ

- Mục đích là **thiết kế kiến trúc tổng thể** của hệ thống
- Các thành phần tạo nên kiến trúc là gì phụ thuộc vào từng cách nhìn đối với hệ thống
- Kiến trúc tổng thể hệ thống có thể được nhìn theo 3 góc nhìn: Theo *hệ con*, Theo *thành phần phần mềm*, Theo *đơn vị phần cứng*
  - Phân rã hệ thống thành các hệ thống con (các gói)
    - Biểu đồ gói (Package diagram)
  - Mô tả các thành phần vật lý của hệ thống
    - Biểu đồ thành phần (Component diagram)
  - Bố trí các thành phần khả thi vào các nút phần cứng
    - Biểu đồ triển khai (Deployment diagram)

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Khái niệm về hệ thống con (subsystem)
  - Các lớp là những thực thể cấu trúc rất nhỏ so với một hệ thống thực. Bởi vậy, khi số các lớp trong hệ thống đã lên tới hàng chục, ta nên gom các lớp liên quan với nhau thành từng nhóm gọi là các hệ thống con.
  - **Hệ thống con (subsystem)** là sự gom nhóm một cách logic (hợp lý) các lớp có sự **gắn kết mạnh bên trong** (của hệ thống con) và sự **liên kết yếu bên ngoài** (giữa các hệ thống con)
  - UML dùng thuật ngữ **gói (package)**, cho nên ta cũng sẽ biểu diễn hệ con dưới dạng gói, mang theo khuôn dập <<subsystem>>

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Nội dung của một hệ thống con (gồm các lớp và các mối liên quan giữa chúng) được UML 2.0 diễn tả trong một khung (frame), với một tựa đề viết trong một hình chữ nhật cắt góc theo khuôn dạng:

[<loại>] Tên [<tham số>]

- Số các lớp trong một hệ thống con không nên ít quá hay nhiều quá

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

Sự **gắn kết** mạnh của các lớp trong cùng một hệ thống con thể hiện:

- ***Về mục đích***: Các lớp phải cung cấp các dịch vụ có cùng bản chất cho người dùng. Như vậy chúng phải thuộc vào cùng một lĩnh vực và đề cập một số thuật ngữ chung (chẳng hạn hệ thống con giao diện đề cập các thuật ngữ như: cửa sổ, thực đơn, nút nhấn,...)
- ***Về xu thế phát triển***: Người ta tách các lớp bền vững với các lớp có nhiều khả năng thay đổi. Đặc biệt, thường tách các lớp nghiệp vụ với các lớp ứng dụng, và xếp chúng vào các hệ con khác nhau.
- ***Về ứng dụng các công nghệ***: Để tận dụng các công nghệ có sẵn, như các thư viện chương trình (lớp/thành phần), các GUI, các hệ quản trị cơ sở dữ liệu,…, ta thường tách các hệ thống con giao tiếp, hệ thống con quản trị dữ liệu ra khỏi phần lõi (ứng dụng và nghiệp vụ) của hệ thống

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Sự **liên kết** giữa các hệ thống con thể hiện ở mối liên quan **phụ thuộc** giữa chúng
  - Sự phụ thuộc giữa 2 hệ thống con phản ánh **các mối liên quan tĩnh** (thừa kế, liên kết, …) và **các mối liên quan động** (trao đổi thông điệp) giữa các lớp thuộc 2 hệ thống con đó
  - Sự phụ thuộc giữa các hệ thống con phải càng đơn giản, lỏng lẻo thì càng tốt. Để đảm bảo tính liên kết yếu này, khi thành lập hệ thống con, áp dụng các quy tắc sau:
    - Các lớp thuộc vào cùng một cấu trúc thừa kế (inheritance hierarchy) nên được xếp vào cùng một hệ thống con
    - Các lớp có mối liên quan kết nhập và hợp thành với nhau thường được xếp vào cùng một hệ thống con
    - Các lớp cộng tác với nhau nhiều, trao đổi thông tin nhiều, thể hiện qua các biểu đồ tương tác, thì nên đặt chung vào một hệ thống con
    - Nên tránh sự phụ thuộc vòng quanh giữa các hệ thống con

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Kiến trúc phân tầng
  - Một hệ thống con thường được định nghĩa bởi các dịch vụ mà nó cung cấp
    - Mối liên quan giữa một hệ thống con với phần còn lại của hệ thống có thể là ngang hàng hay là khách/chủ
  - Trong mối **liên quan ngang hàng (peer-to-peer)** thì mỗi bên đều có thể truy cập các dịch vụ của bên kia
  - Còn mối **liên quan khách/chủ (client/server)** thì đơn giản hơn: bên khách (client) gọi bên chủ (server) và bên chủ thực hiện một dịch vụ theo yêu cầu và trả kết quả cho bên khách
    - Bên khách phải biết giao diện của bên chủ
    - Nhưng bên chủ thì không cần biết giao diện của bên khách

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Từ 2 hình thức giao tiếp đó, ta có hai cách để chia hệ thống thành các hệ thống con:
  - Tổ chức hệ thống thành các **tầng**, với mối quan hệ khách/chủ luôn luôn hướng từ tầng trên xuống (các) tầng dưới
    - Tầng trên đóng vai trò khách, tầng dưới đóng vai trò chủ
    - Ví dụ: Hệ thống tạo cửa sổ trong giao diện người dùng của máy tính
  - Tổ chức hệ thống thành các **lát**, với mối quan hệ ngang hàng giữa các lát; tuy nhiên các lát là khá độc lập hoặc liên kết yếu với nhau
    - Ví dụ: Hệ điều hành thường gồm các hệ con như là các hệ quản lý tệp, hệ điều khiển thiết bị, hệ quản lý sự kiện và ngắt...

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Tổ chức phân tầng là đáng được ưu tiên hơn, vì nó mang lại nhiều ưu thế trong thiết kế, trong cài đặt, cũng như trong việc sử dụng lại
- **Đối với các hệ thống lớn, thì ta thường phải phối hợp cả 2 cách tổ chức phân tầng và phân lát**
  - Phân hệ thống thành các tầng (architectural building layers)
  - Trong mỗi tầng, thì lại phân thành các lát (architectural building blocks)

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

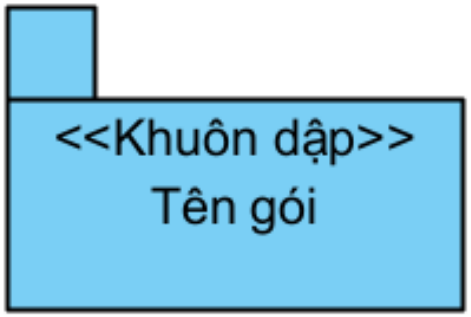
- Khi thực hiện phân tầng, thì số tầng là tuỳ thuộc sự phức tạp của hệ thống:
  - Trong một hệ thống đơn giản, thì số tầng có thể chỉ là hai (2 tiers): Tầng khách (client) thì quản lý giao diện người dùng và các quá trình khai thác, còn tầng máy chủ (server) thì xử lý việc lưu giữ các dữ liệu
  - Trong một hệ thống phức tạp hơn, thì người ta tách tầng trên thành 2 tầng: *giao diện* và *ứng dụng*, và ở dưới nó là tầng *nghiệp vụ* (hay tầng lĩnh vực), bền vững hơn và có nhiều khả năng sử dụng lại hơn
  - Đó là một kiến trúc khách/chủ ba tầng (3 tiers)

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

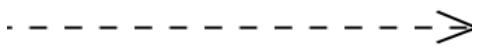
- Trong các hệ thống lớn, số tầng còn có thể nhiều hơn (n tiers)
- Điển hình là kiến trúc 5 tầng (từ trên xuống):
  - **Tầng trình bày:** Chuyển các dữ liệu cho người dùng và biến đổi các hành động của người dùng thành các sự kiện vào của hệ thống
  - **Tầng ứng dụng:** Bao gồm các đối tượng điều khiển và dẫn dắt các quy luật của ứng dụng
  - **Tầng nghiệp vụ:** Bao gồm các đối tượng nghiệp vụ (hay lĩnh vực), và việc cài đặt các quy tắc quản lý chúng
  - **Tầng truy cập dữ liệu:** Quản lý việc truy cập (đọc/viết) các đối tượng nghiệp vụ từ các phương tiện lưu trữ dữ liệu
  - **Tầng lưu trữ dữ liệu:** Bảo đảm sự lưu giữ lâu dài các dữ liệu

PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Biểu đồ gói: kí pháp:
  - Gói:

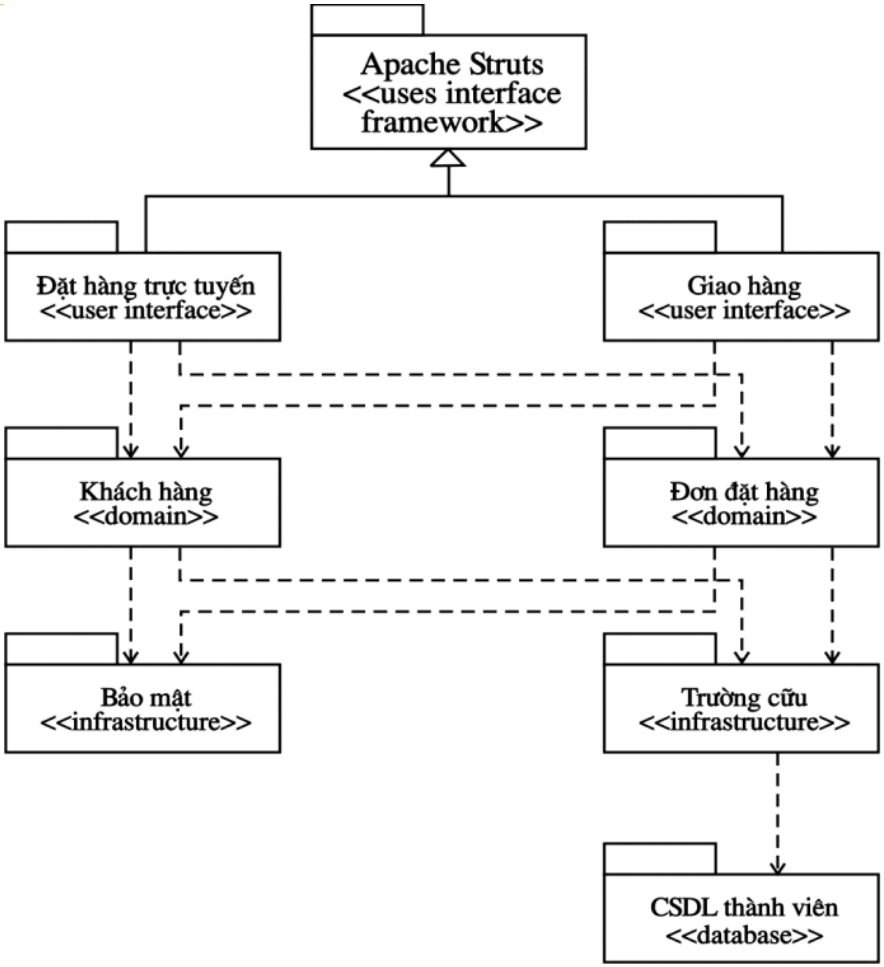


  - Mối quan hệ:



PHÂN RÃ HỆ THỐNG THÀNH CÁC HỆ THỐNG CON

- Hình vẽ bên minh hoạ về kiến trúc khách/chủ gồm 5 tầng, trong đó mỗi gói (hệ thống con) đều có mang khuôn dập thích hợp:
  - <<user interface framework>>: khung giao diện người dùng
  - <<user interface>>: giao diện người dùng
  - <<domain>>: lĩnh vực
  - <<infrastructure>>: cơ sở hạ tầng
  - <<database>>: cơ sở dữ liệu



MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Thành phần (Components) và biểu đồ thành phần (Component diagram)
  - Nếu như biểu đồ gói (hệ thống con) mà ta nói ở phần trên phản ánh cho góc nhìn về cấu trúc logic của hệ thống (ở mức cao so với biểu đồ lớp), thì biểu đồ thành phần cho ta một cách nhìn về **cấu trúc vật lý của hệ thống**
  - Chữ "vật lý" ở đây được hiểu theo nghĩa là sự mô tả hướng tới các sản phẩm phần mềm, là kết quả của sự cài đặt và thực sự tồn tại
    - Chứ không phải là các sản phẩm logic, là kết quả của quá trình phân tích
    - Tuy nhiên, ở đây ta chưa đề cập tới phần cứng, mặc dù tính vật lý của nó cũng là đương nhiên

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- UML định nghĩa **thành phần (component)** là một bộ phận vật lý và thay thế được của hệ thống, cung cấp sự thực hiện (implementation) cho một tập các giao diện (interfaces)
- Nói cách khác, thì thành phần là một cài đặt của một tập hợp các phần tử logic (vd: các lớp hay các hợp tác)

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Có 3 loại thành phần:
  - Các **thành phần triển khai (deployment components)**: Đó là các thành phần cần và đủ để tạo nên một hệ thống khả thi, như là các thư viện động (DLL) và các mã thực thi (executable). Định nghĩa thành phần của UML là đủ rộng để bao hàm các mô hình đối tượng kinh điển (vd: COM+, CORBA, và EJB - Enterprise Java Beans), cũng như các mô hình đối tượng khác như là các trang Web động, các bảng cơ sở dữ liệu, và các mã thực thi sử dụng những cơ chế truyền thông riêng

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

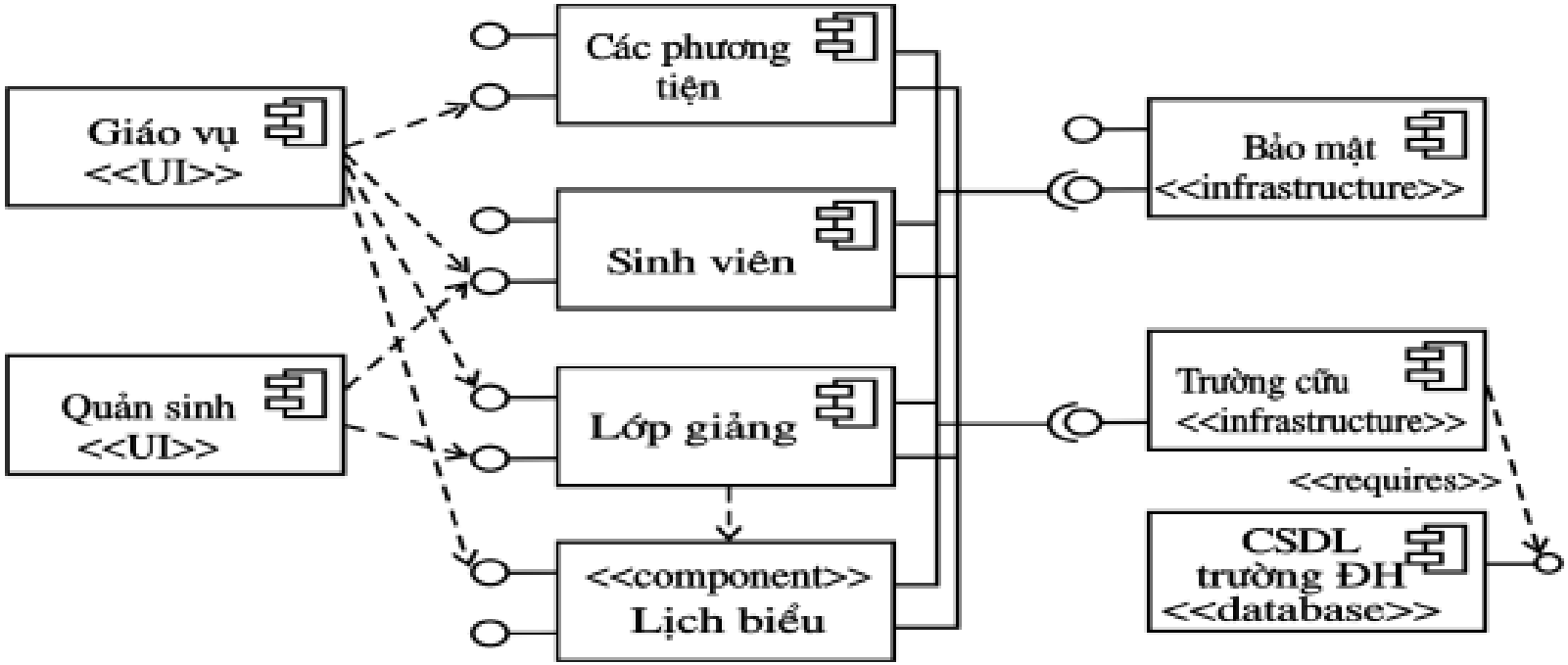
- Các **thành phần sản phẩm làm việc (work product components)**: Đó là các thành phần có từ quá trình phát triển hệ thống, bao gồm các tệp mã nguồn, các tệp dữ liệu, từ đó mà ta đã tạo lập ra các thành phần triển khai. Các thành phần này không trực tiếp tham gia vào hệ thống thực thi, nhưng không có chúng thì không tạo được hệ thống thực thi
- Các **thành phần thực hiện (execution components)**: Đó là các thành phần được tạo nên như là một kết quả của một hệ thực hiện (vd: một thành phần .JAR)

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- 5 khuôn dập chuẩn của UML
  - <<executable>>: Một thành phần có thể thực hiện trên 1 nút
  - <<library>>: Một thư viện đối tượng tĩnh hoặc động
  - <<table>>: Một bảng trong cơ sở dữ liệu
  - <<file>>: Một tập tin chứa mã nguồn hoặc dữ liệu
  - <<document>>: Một tài liệu
- Có thể dùng các khuôn dập không chuẩn
  - <<application>>: Một ứng dụng
  - <<database>>: Một cơ sở dữ liệu
  - <<infrastructure>>: Một thành phần cơ sở hạ tầng (vd: một dịch vụ lưu trữ, một kiểm soát đăng nhập,…)
  - <<source code>>: Một tập tin mã nguồn
  - <<web service>>: Một dịch vụ Web

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Để tổ chức các thành phần lại với nhau, ta có hai cách:
  - Gom các thành phần vào các gói, nghĩa là đưa chúng vào các hệ thống con; hoặc
  - Thiết lập các mối liên quan phụ thuộc giữa chúng, và như thế ta có một Biểu đồ thành phần



MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Các mục đích mô hình hoá của Biểu đồ thành phần
  - Như đã trình bày, có nhiều loại thành phần: thành phần triển khai, thành phần sản phẩm làm việc, và thành phần thực hiện
  - Vì vậy, Biểu đồ thành phần lập ra phải có mục đích ***mô tả loại thành phần nào***

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

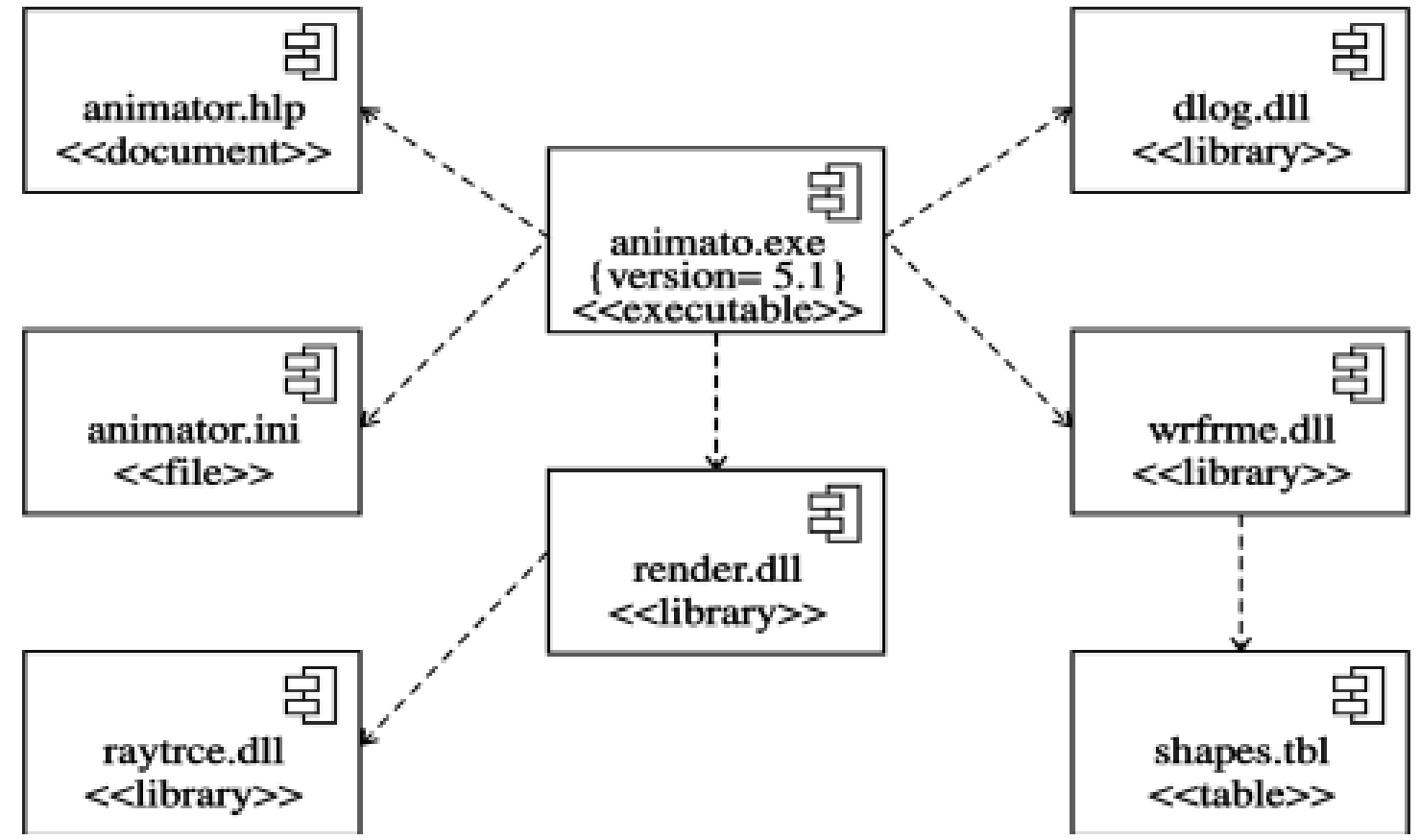
- **Mô hình hoá các thành phần khả thi và thư viện**: Mục đích chính của việc sử dụng Biểu đồ thành phần là để mô hình hoá các thành phần triển khai, tạo nên sự cài đặt của hệ thống
  - Nếu ta muốn cài đặt một hệ thống chỉ gồm đúng một tệp khả thi (.EXE), thì ta chẳng cần dùng tới thành phần
  - Nhưng nếu hệ thống gồm nhiều tệp khả thi và liên kết với các thư viện đối tượng, thì ta cần dùng biểu đồ thành phần để giúp hiển thị, đặc tả, thành lập và tư liệu hoá các quyết định đối với hệ thống vật lý

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- **Mô hình hoá các bảng, các tệp và các tài liệu**
  - Bên cạnh các tệp khả thi và thư viện tạo nên phần chạy được của hệ thống, thì còn nhiều thành phần bố trí khác, gọi là các thành phần phụ trợ, cần cho việc cài đặt hệ thống
  - Ví dụ: Để cài đặt, ta vẫn cần các tệp dữ liệu, các tài liệu trợ giúp, các scripts, các tệp log, các tệp khởi tạo tham số cấu hình,…
  - Mô hình hoá các thành phần này cũng là một phần quan trọng để diễn tả hình trạng của hệ thống

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- *Ví dụ*:

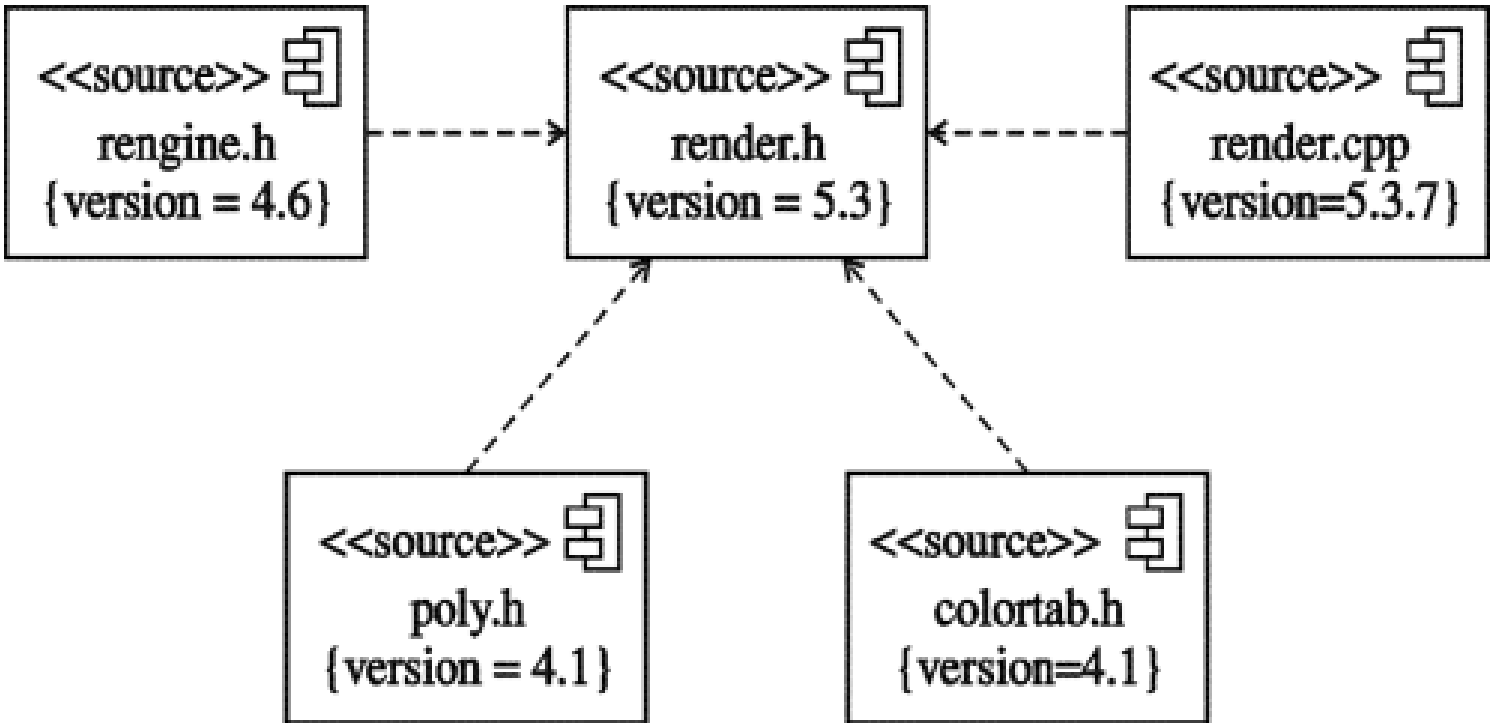


MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- **Mô hình hoá mã nguồn**
  - Mô hình hoá mã nguồn cũng là một mục đích của việc tạo lập biểu đồ thành phần
  - Các tệp mã nguồn dùng để chứa các chi tiết về các lớp, các giao diện, các hợp tác và các phần tử logic khác
  - Các tệp mã nguồn tạo nên một bước trung gian để tạo lập các thành phần vật lý/thực thi
  - Các công cụ (vd: chương trình biên dịch) thường đòi hỏi các tệp mã nguồn phải được tổ chức theo một quy cách nhất định nào đó (vd: tệp header .h và tệp mã nguồn .cpp). Các mối liên quan phụ thuộc giữa các tệp này phản ánh một trật tự biên dịch

MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Mô hình hoá các tệp mã nguồn dùng để xây dựng thư viện render.dll từ ví dụ trước



MÔ TẢ CÁC THÀNH PHẦN VẬT LÍ CỦA HỆ THỐNG

- Biểu đồ thành phần: kí pháp
  - Thành phần

  - Liên kết:

BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

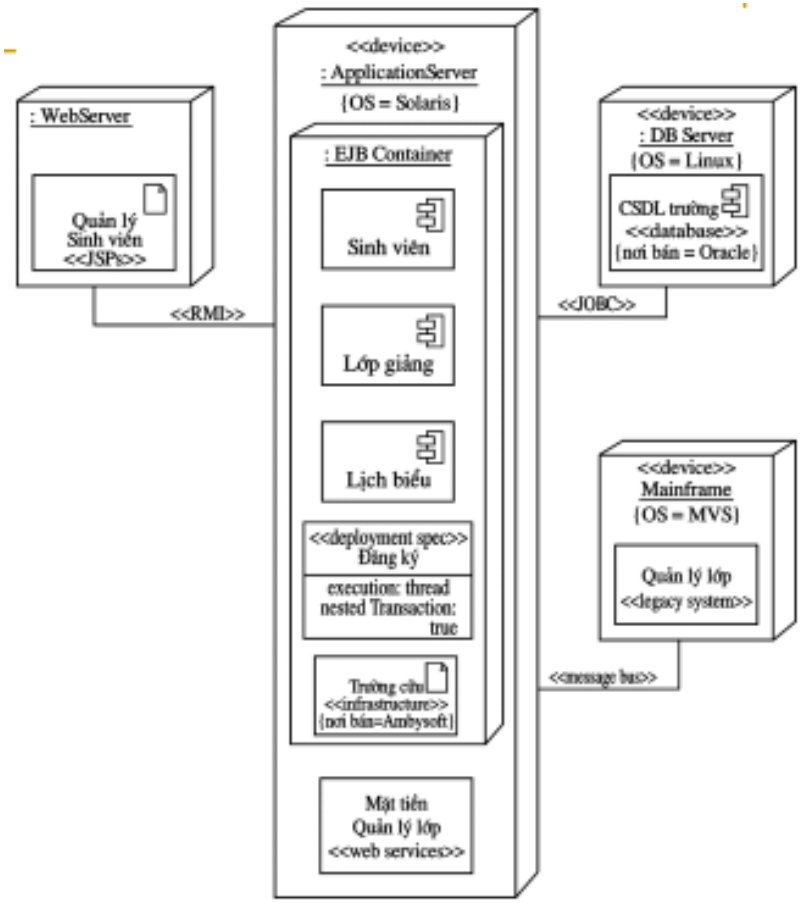
- Để bố trí các thành phần phần mềm vào các nút phần cứng, ta dùng Biểu đồ triển khai
- **Biểu đồ triển khai (Deployment diagram)** là một biểu đồ diễn tả sự bố trí các đối tượng thực thi (executable artifacts) trên nền tảng thực thi (underlying platform). Biểu đồ triển khai gồm các nút và các kết nối giữa các nút đó
- Một **nút (node)** là một phần tử vật lý tồn tại vào lúc chạy và biểu diễn cho một **tài nguyên tính toán (computational resource)**. Một nút được biểu diễn bởi một hình hộp, có mang tên.
  - Nếu tên này không gạch dưới, thì nút thể hiện một **lớp các tài nguyên**
  - Nếu tên được gạch dưới, thì nút thể hiện một **cá thể tài nguyên**

BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

- Có hai mức diễn tả của Biểu đồ triển khai: mức lớp (tương ứng với Biểu đồ lớp) và mức cá thể (tương ứng với Biểu đồ đối tượng)
  - Nút có thể là:
    - Một thiết bị (nút phần cứng), thường mang khuôn dập <<device>> (hoặc cụ thể hơn là <<processor>>, <<console>>, <<kiosk>>, <<printer>>,…)
    - Một môi trường thực hiện (nút phần mềm), thường mang khuôn dập <<execution env>> (vd: <<EJB Container>>, <<J2EEServer>>)

BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

- Hình vẽ bên thể hiện một biểu đồ triển khai (ở mức cá thể) biểu diễn cho hình trạng vật lý của Hệ thống thông tin của 1 trường đại học
  - Nút **WebServer** chưa có khuôn dập, đó là vì người phát triển hệ thống chưa có quyết định – Nút này có thể là một loại phần mềm hoặc là một thiết bị vật lý
  - Các nút có thể chứa các nút khác hoặc các phần mềm. Chẳng hạn, nút **ApplicationServer** chứa nút **EJBContainer** (một nút phần mềm), và nút này lại chứa ba thành phần phần mềm, một đặc tả bố trí, và một phần mềm



BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

- Các **kết nối (connections)** là các mối liên quan giao tiếp giữa các cặp nút, thể hiện về mặt vật lý bằng một đường truyền (vd: một kết nối Ethernet, một đường truyền tuần tự, hay một kênh truyền dùng chung)
- Mỗi kết nối hỗ trợ cho một hay nhiều giao thức truyền thông, mà ta cần chỉ rõ bằng các khuôn dập thích hợp

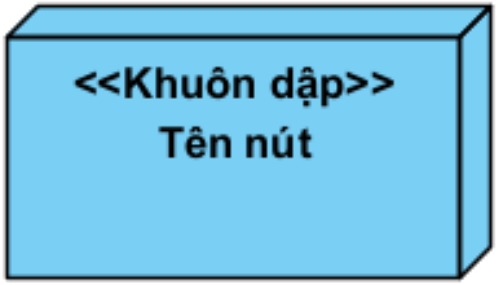
BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

- Một số khuôn dập dùng cho các kết nối

Khuôn dập	Ý nghĩa
Asynchronous	Một kết nối không đồng bộ,
HTTP	HyperText Transport Protocol, một giao thức Internet
JDBC	Java Database Connectivity, công nghệ Java để kết nối và truy cập CSDL
ODBC	Open Database Connectivity, kết nối và truy cập CSDL
RMI	Remote Method Invocation, một giao thức hỗ trợ lời gọi từ xa
RPC	Remote Procedure Call, lời gọi thủ tục từ xa
Synchronous	Một kết nối đồng bộ, trong đó bên gửi chờ trả lời từ bên nhận
Web services	Liên lạc qua các giao thức web services (vd: SOAP)

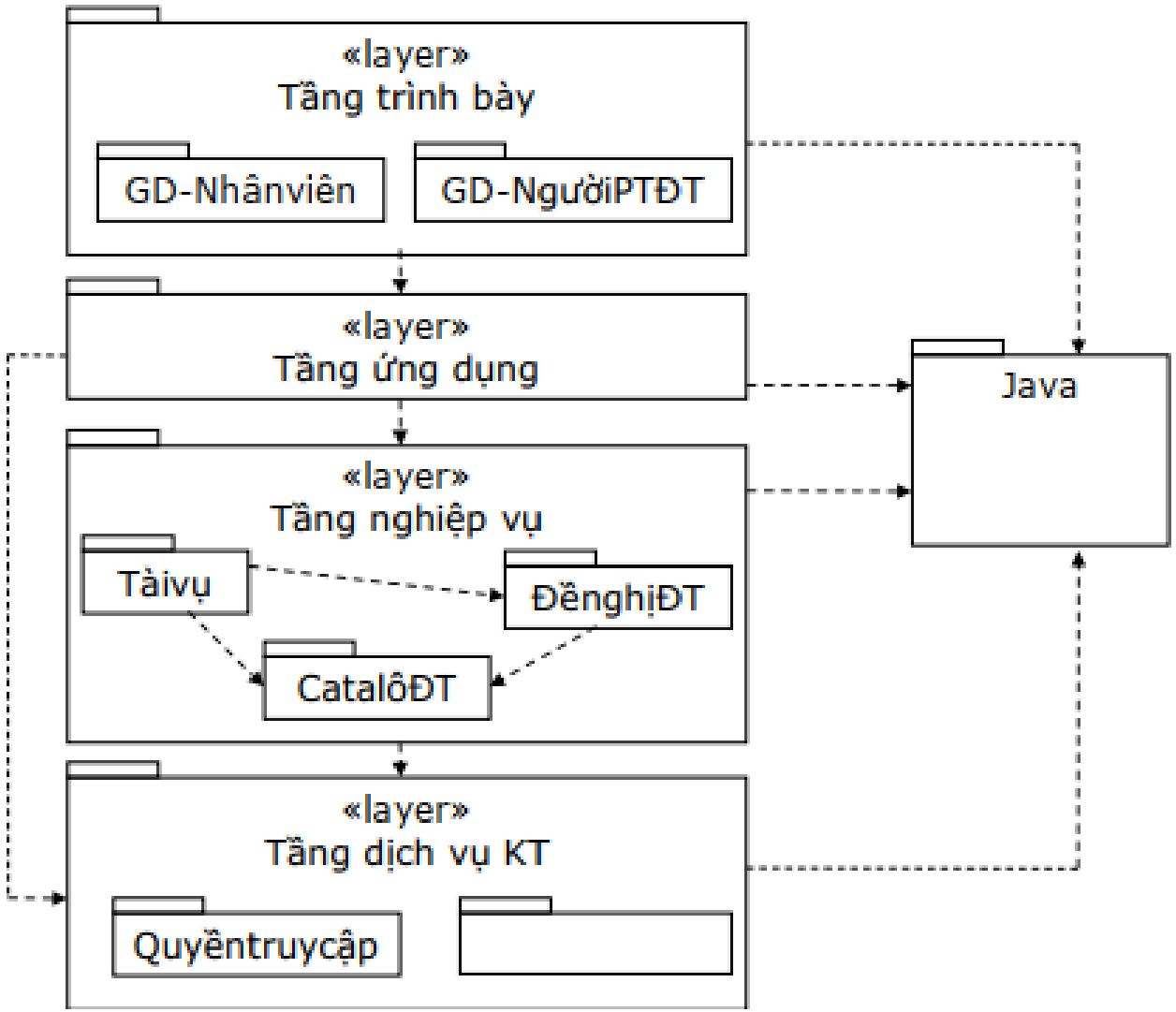
BỐ TRÍ CÁC THÀNH PHẦN KHẢ THI VÀO CÁC NÚT PHẦN CỨNG

- Biểu đồ triển khai: kí pháp
  - Nút – node

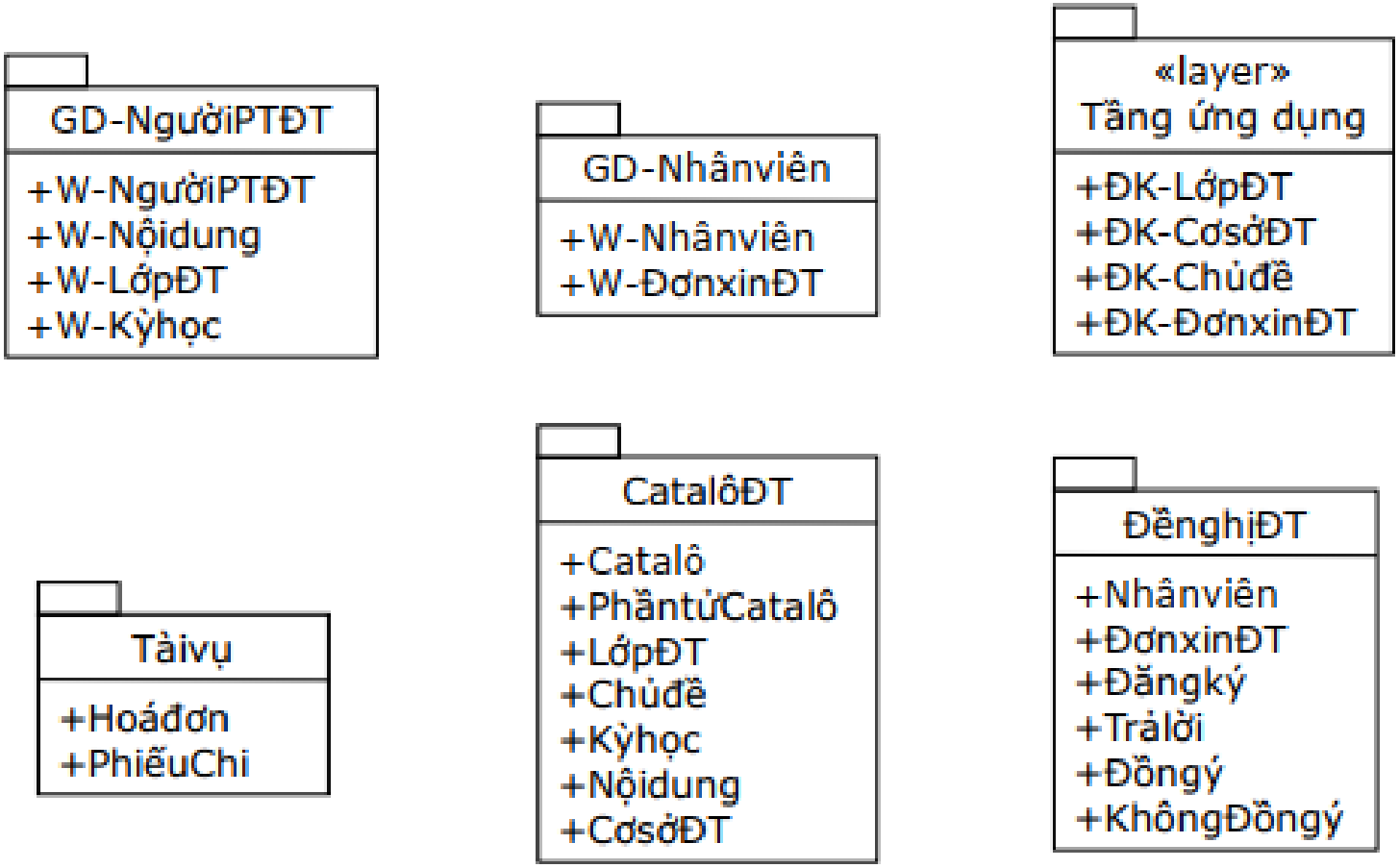


  - Liên kết:

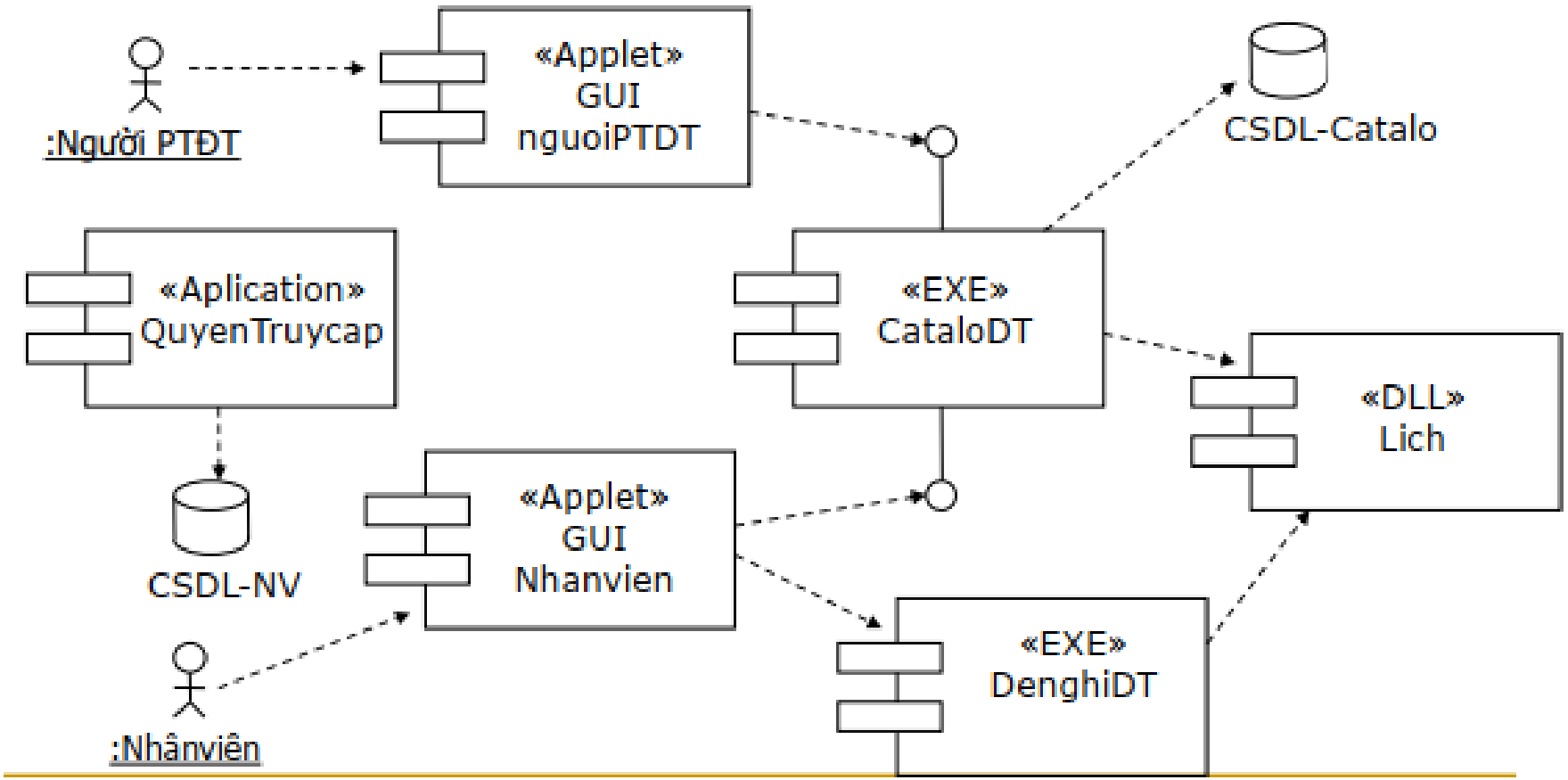
VÍ DỤ


«layer»
Tầng trình bày
GD-Nhânviên
GD-NgườiPTĐT
«layer»
Tầng ứng dụng
Java
«layer»
Tầng nghiệp vụ
Tàivụ
ĐềnghịĐT
CatalôĐT
«layer»
Tầng dịch vụ KT
Quyềntruycập

VÍ DỤ


GD-NgườiPTĐT
+W-NgườiPTĐT
+W-Nộidung
+W-LớpĐT
+W-Kỳhọc
GD-Nhânviên
+W-Nhânviên
+W-ĐơnxinĐT
«layer»
Tầng ứng dụng
+ĐK-LớpĐT
+ĐK-CơsởĐT
+ĐK-Chủđề
+ĐK-ĐơnxinĐT
Tàivụ
+Hoáđơn
+PhiếuChi
CatalôĐT
+Catalô
+PhầntửCatalô
+LớpĐT
+Chủđề
+Kỳhọc
+Nộidung
+CơsởĐT
ĐềnghịĐT
+Nhânviên
+ĐơnxinĐT
+Đăngký
+Trảlời
+Đồngý
+KhôngĐồngý

VÍ DỤ


:Người PTĐT
«Applet»
GUI
nguoiPTDT
CSDL-Catalo
«Aplication»
QuyenTruycap
«EXE»
CataloDT
«DLL»
Lich
CSDL-NV
«Applet»
GUI
Nhanvien
«EXE»
DenghiDT
:Nhânviên

THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

- Mục đích
- Mô tả các giao diện của hệ thống
- Làm nguyên mẫu

MỤC ĐÍCH LÀM NGUYÊN MẪU GIAO DIỆN NGƯỜI DÙNG

- Dựa vào các công cụ tạo lập giao diện người dùng (GUI – Graphical user interface – design tools) ta thành lập sớm và nhanh một nguyên mẫu giao diện người dùng (GUI prototype), có tính thăm dò sự phù hợp, nhằm các mục đích:
  - Tạo ra một môi trường làm việc cụ thể, dễ tiếp xúc, dễ làm thử, làm cho người dùng trở nên yên tâm hơn, và năng động hơn trong việc đóng góp cho việc phát triển hệ thống
  - Qua quá trình dùng thử, ta thu thập được nhiều ý kiến phản hồi có ích từ phía người dùng
  - Sớm phát hiện được các yêu cầu hay chức năng bị bỏ sót, sớm nhìn thấy các điểm yếu, chỗ khó khăn nhất của hệ thống
- **Thiết kế giao diện không phải là các ảnh chụp màn hình hệ thống sau cài đặt!**

MÔ TẢ GIAO DIỆN CỦA HỆ THỐNG

- Như đã biết, cứ mỗi cặp (**tác nhân, ca sử dụng**) liên quan, có ít nhất một lớp biên để chuyển đổi các thông tin vào/ra
  - Thể hiện của lớp biên chính là giao diện mà bây giờ ta cần phải mô tả
- Muốn mô tả các lớp biên, ta xem lại từng bước trong kịch bản của mỗi ca sử dụng – để xác định các phần tử của giao diện:
  - Xét nội dung của tương tác giữa tác nhân và hệ thống
  - Các thông tin vào và Các thông tin ra
  - Các hành động được yêu cầu

MÔ TẢ GIAO DIỆN CỦA HỆ THỐNG

- Bản mô tả một giao diện thường chứa các điểm sau:
  - Tên của giao diện;
  - Mô tả tóm tắt nội dung của giao diện;
  - Mức độ phức tạp của giao diện (phức tạp/chuẩn/đơn giản);
  - Ghi chú thêm (nếu có);
  - Thiết kế chi tiết của giao diện, tuỳ thuộc vào 4 loại giao diện sau:
    - Các giao diện hội thoại (với người dùng)
    - Các giao diện hiển thị thông tin (kết quả xử lý, báo cáo,…)
    - Các giao diện dữ liệu từ/đến các hệ thống ngoài
    - Các giao diện chức năng đến các hệ thống ngoài

LÀM NGUYÊN MẪU GIAO DIỆN

- Làm nguyên mẫu giao diện nên được bắt đầu càng sớm càng tốt
  - Có thể bắt đầu làm nguyên mẫu giao diện ngay khi xong phân tích các ca sử dụng
- Có nhiều bộ tạo lập giao diện người dùng (GUI builders) cho phép làm các nguyên mẫu giao diện mà không tốn mấy công sức
- **Đối với phiên bản đầu của nguyên mẫu giao diện, thì các trường là rỗng hoặc mang giá trị minh họa, các nút và các phần tử tương tác có thể chưa cần có hiệu ứng tương tác rõ ràng**
- **Qua các vòng lặp tiếp theo (của việc làm nguyên mẫu), thì giao diện dần trở nên sinh động hơn (dần đi tới phương án cuối)**
- Như vậy, người dùng có thể sớm làm việc thử với các nguyên mẫu giao diện

LÀM NGUYÊN MẪU GIAO DIỆN

- Nguyên mẫu giao diện chỉ có ý nghĩa thăm dò => nên làm nhanh và không nên cầu toàn
  - Chưa nên chú ý nhiều về trình bày và mỹ thuật, mà cần chú ý đến ***nội dung*** (các trường và các điều khiển của giao diện) và ***luồng dẫn dắt*** *từ phần tử giao diện này sang phần tử giao diện khác* (screens flow)
- Quá trình phát triển nguyên mẫu giao diện nên được làm song song với quá trình phân tích và thiết kế, và nên hỗ trợ cho phân tích và thiết kế
- Nguyên mẫu giao diện nói chung chỉ là mặt ngoài của hệ thống, không phản ánh hết tầm sâu (các chi tiết hoạt động) của hệ thống
  - Cần nói rõ với người dùng là nguyên mẫu không phải là hệ thống hoàn chỉnh
- **Làm nguyên mẫu chỉ là một sự hỗ trợ tốt, chứ không thể là sự thay thế cho các bước phân tích và thiết kế hệ thống một cách nghiêm túc được**

4.3 THIẾT KẾ CHI TIẾT LỚP

- Mục đích của thiết kế chi tiết
- Quy trình thiết kế chi tiết
- Thiết kế chi tiết các tầng

MỤC ĐÍCH CỦA THIẾT KẾ CHI TIẾT LỚP

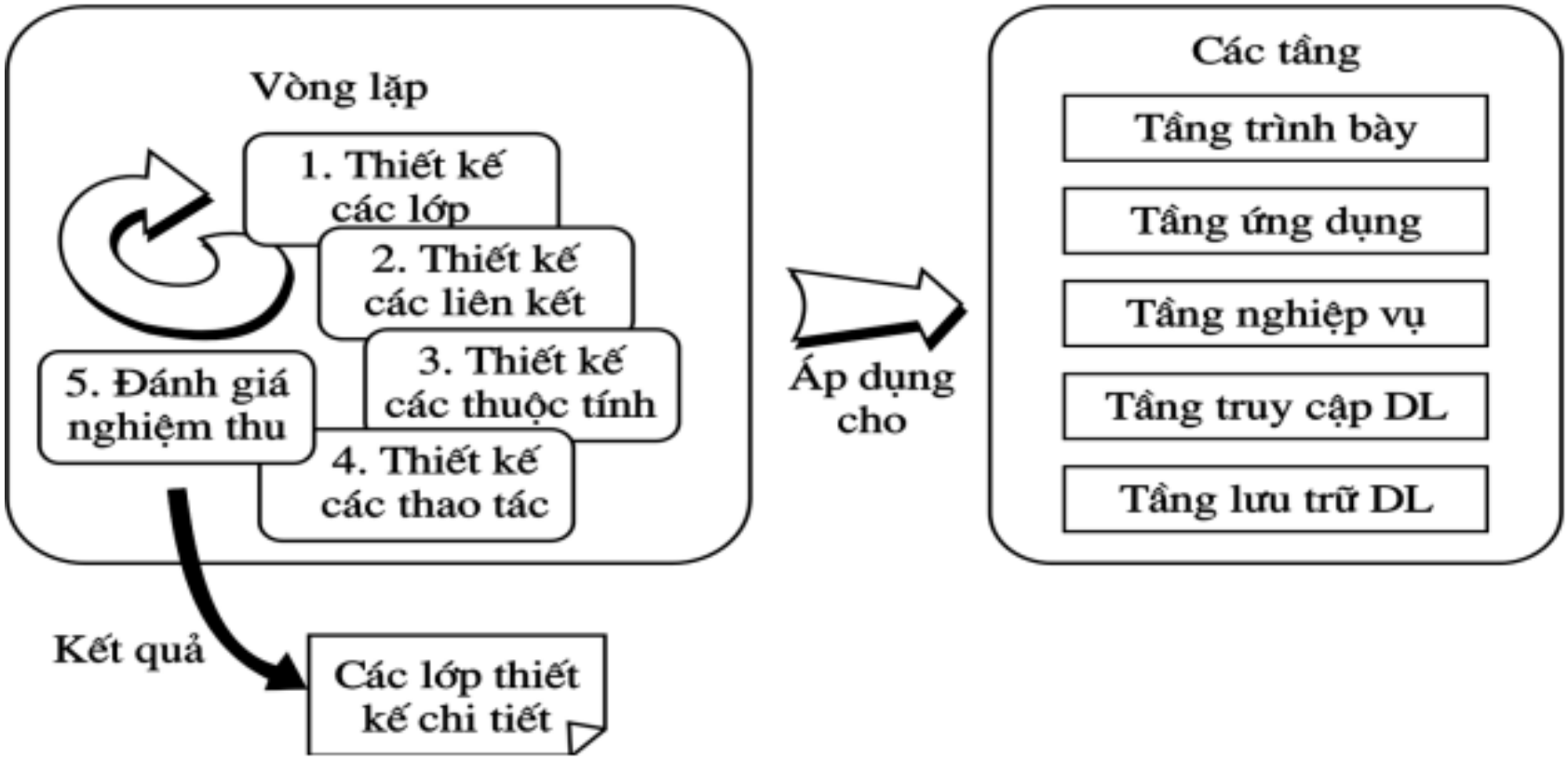
- Trong phân tích, ta tập trung nghiên cứu cấu trúc logic của các thông tin, cần thiết cho việc xây dựng một giải pháp nghiệp vụ
  - Phân tích luôn được triển khai theo nhãn quan ứng dụng, và chưa tính tới các điều kiện về công nghệ
- Trong thiết kế, đặc biệt là **thiết kế chi tiết**, ta cần nghiên cứu các cách tốt nhất để cài đặt cấu trúc logic nói trên, nhằm tối ưu hoá hiệu năng của ứng dụng
  - Thiết kế phải được triển khai theo nhãn quan kỹ thuật và phụ thuộc vào các khả năng và các hạn chế của tài nguyên tính toán và công nghệ lập trình phát triển ứng dụng

MỤC ĐÍCH CỦA THIẾT KẾ CHI TIẾT LỚP

- Đầu vào của việc thiết kế chi tiết là các mô hình đã được thiết lập từ các bước phân tích và thiết kế hệ thống trước đây, bao gồm:
  - Các mô hình về cấu trúc (các biểu đồ lớp, thành phần, hệ thống con, kiến trúc phân tầng), và
  - Các mô hình động thái (các biểu đồ ca sử dụng, tương tác, máy trạng thái và giao diện người dùng)
- Kết quả đầu ra của thiết kế chi tiết phải là một **mô hình sẵn sàng cho lập trình**, bao gồm mọi quyết định về cài đặt, **thích hợp với ngôn ngữ lập trình và môi trường cài đặt** đã được lựa chọn (hay sẵn có)
  - Mô hình này có thể vẫn được diễn tả thông qua các biểu đồ của UML (kết hợp với các tài liệu diễn giải)

QUY TRÌNH THIẾT KẾ

- Việc thiết kế chi tiết được tiến hành theo một **quy trình lặp đ**ược diễn tả như hình sau:



THIẾT KẾ CÁC LỚP

- Thiết kế lớp đưa ra nhiều sự thay đổi đối với các lớp phân tích. Đó có thể là:

a) Phân bổ lại hay giải phóng bớt trách nhiệm cho các lớp phân tích

- Khi phân tích, thì các lớp được đưa ra theo nhu cầu, mà chưa tính đến hiệu năng hay sự thích ứng với điều kiện kỹ thuật. Vì vậy trong thiết kế chi tiết, cần có sự phân bổ lại trách nhiệm, theo nhiều mục đích khác nhau, như:
  - Rút các nhiệm vụ có tính chất kỹ thuật ra khỏi các lớp phân tích
  - Phân bổ lại trách nhiệm, và đặc biệt là các sự trao đổi thông điệp và sự kiện, cho phù hợp với yêu cầu của kiến trúc phân tầng
  - Chuyển việc lưu giữ các kết quả trung gian hay việc thực hiện các nhiệm vụ cấp thấp cho các lớp mới

THIẾT KẾ CÁC LỚP

b) Thêm các lớp mới để cài đặt các cấu trúc dữ liệu

- Khi phân tích, thì nhiều cấu trúc dữ liệu được xem là mặc định. Trong thiết kế chi tiết, thì các cấu trúc dữ liệu đó phải được xem xét việc cài đặt cụ thể (chi tiết)
- Thường thì nhiều cấu trúc dữ liệu là có sẵn trong thư viện của ngôn ngữ lập trình, vd: mảng (array), danh sách (list), hàng đợi (queue), ngăn xếp (stack), tập hợp (set), (bag), từ điển (dictionary), cây (tree), …

THIẾT KẾ CÁC LỚP

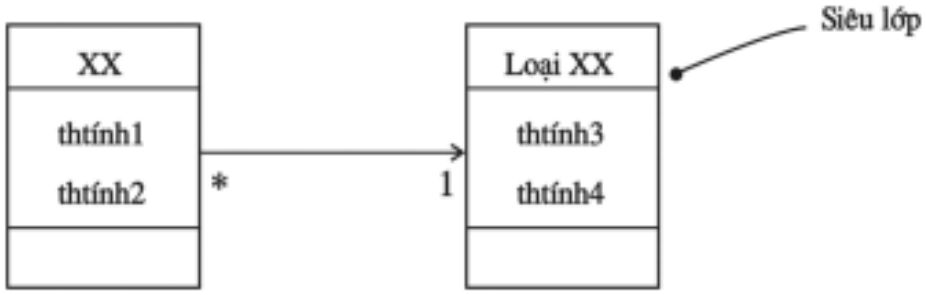
c) Thêm các lớp mới để cài đặt các khái niệm phân tích

- Có nhiều khái niệm dùng trong phân tích, nhưng không có trong các ngôn ngữ lập trình, cần phải tìm cách để cài đặt chúng bằng các lớp
- Ví dụ điển hình: Máy trạng thái
  - Có nhiều cách để cài đặt máy trạng thái
  - Trong một ngôn ngữ lập trình hướng đối tượng, thì ta có thể cài đặt nó theo mẫu thiết kế trạng thái (state)

THIẾT KẾ CÁC LỚP

d) Thêm các lớp mới vì mục đích tối ưu hoá

- Các **lớp mô tả**: Ta gọi lớp mô tả (meta-class) là một lớp mà đối tượng của nó lại là một lớp khác
  - Khi xem xét một lớp XX, mà ta thấy trong các trách nhiệm của nó, có những trách nhiệm không thuộc riêng từng đối tượng, mà thuộc vào từng nhóm (loại) đối tượng => Ta thêm lớp LoạiXX (hay KiểuXX) và phân bố lại các thuộc tính và liên kết trên hai lớp
  - Kết nối lớp XX với lớp mới LoạiXX bằng một liên kết **nhiều-1**, thường là một chiều, hướng tới LoạiXX. Lớp LoạiXX là một lớp mô tả (meta-class), vì mỗi đối tượng của nó (một loại) lại là một tập hợp các đối tượng
  - Ưu điểm của giải pháp trên là đã tránh được sự lặp lại nhiều lần giá trị của các thuộc tính của siêu lớp trên mỗi đối tượng của lớp XX



THIẾT KẾ CÁC LỚP

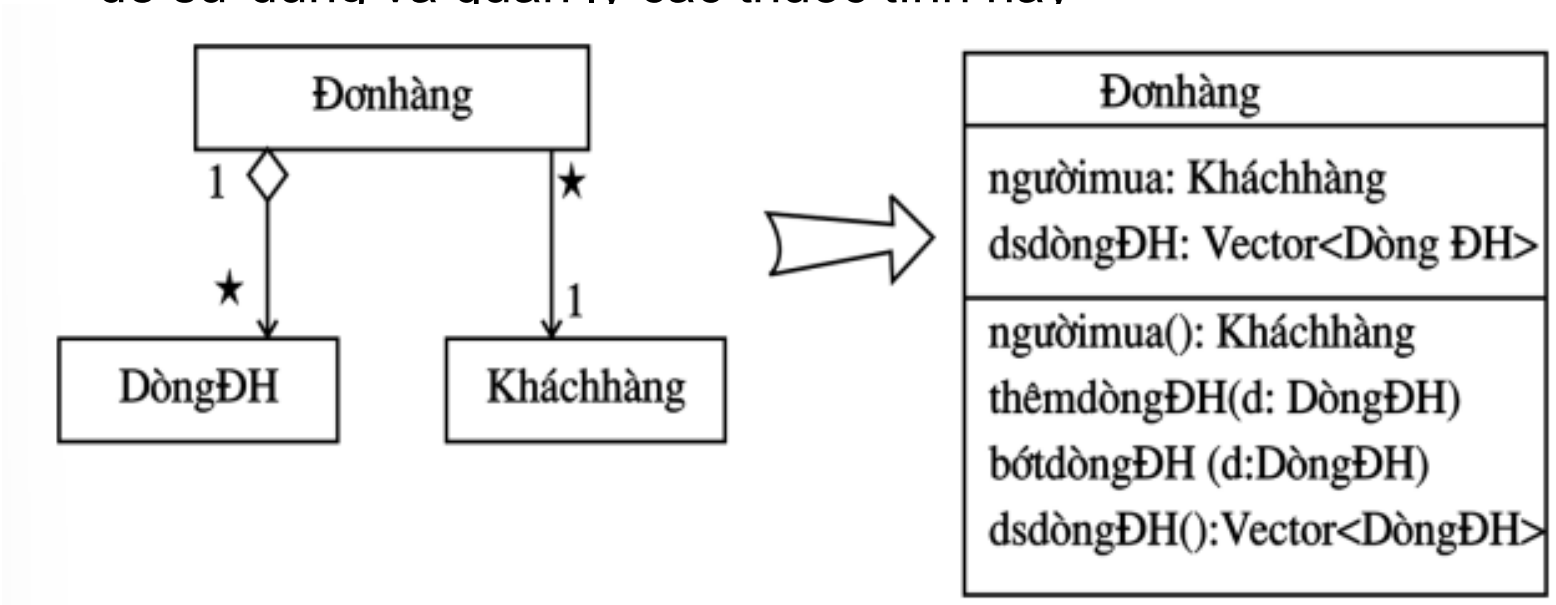
- Các **lớp dẫn xuất**: Nếu ta muốn loại bỏ sự tính toán lại, để tăng hiệu năng, thì ta có thể lập ra các lớp mới để lưu giữ các thuộc tính (dữ liệu) dẫn xuất. Tuy nhiên, cần phải cập nhật các thuộc tính dẫn xuất mỗi khi các thuộc tính cơ sở thay đổi. Việc này có thể thực hiện bằng các cách sau:
  - *Mã tường minh*: Đưa thêm mã lệnh vào các phương thức cập nhật thuộc tính của các lớp cơ sở. Các mã lệnh mới thêm này sẽ cập nhật một cách tường minh các thuộc tính dẫn xuất, khi thuộc tính cơ sở được cập nhật
  - *Tính toán lại một cách định kỳ:* Khi các giá trị cơ sở thay đổi đồng loạt (không lẻ tẻ), thì ta có thể thực hiện tính toán lại một cách định kỳ sau khi các giá trị cơ sở đều đã thay đổi
  - *Thủ tục tự động (Trigger)*: Khi một thuộc tính cơ sở có nhiều thuộc tính dẫn xuất từ nó được cập nhật, thì một trigger (một thủ tục tự khởi động khi một sự kiện định trước xảy ra) sẽ được kích hoạt và báo cho các đối tượng chứa các thuộc tính dẫn xuất biết là thuộc tính cơ sở đã thay đổi giá trị. Sau đó thì chính đối tượng chứa thuộc tính dẫn xuất có trách nhiệm cập nhật thuộc tính dẫn xuất của mình

THIẾT KẾ CÁC LỚP

- Các **lớp cha**: Khi có *cùng một số thao tác hay một số thuộc tính* được phát hiện trên nhiều lớp khác nhau, thì có thể đưa chúng vào một lớp cha để các lớp đó sử dụng lớp cha theo cách thừa kế
  - Tuy nhiên, các thao tác trong các lớp khác nhau thường chỉ là tương tự nhau, chứ chưa thực sự là đồng nhất. Bấy giờ ta phải điều chỉnh tiêu đề của thao tác ít nhiều (đổi tên, thêm tham số,…) thì mới quy được các dịch vụ tương tự nhau vào một, để tổ chức thành thừa kế
  - Các biện pháp tổ chức thừa kế nói trên là có ích theo mục đích tối ưu hoá. Tuy nhiên ta không nên lạm dụng nó như là những biện pháp kỹ thuật thuần tuý để "cưỡng bức" các lớp xa lạ với nhau về ngữ nghĩa vào trong cùng một phân cấp kế thừa (vì có thể dẫn tới các hiệu ứng phụ khó lường trước được)

THIẾT KẾ CÁC LIÊN KẾT

- Khái niệm liên kết không có sẵn trong các ngôn ngữ lập trình
  - Chuyển đổi mỗi liên kết thành thuộc tính hay mảng các thuộc tính, tuỳ theo cơ số của nó
  - Cùng với các thuộc tính đưa thêm đó, cần có thêm các thao tác để sử dụng và quản lý các thuộc tính này



THIẾT KẾ CÁC LIÊN KẾT

- Nếu liên kết là 1 chiều (như trên hình vừa rồi), thì thuộc tính chỉ đưa vào một phía (phía gốc của sự lưu hành)
- Nhưng nếu liên kết là 2 chiều, thì thuộc tính cài đặt nó phải đưa vào cả 2 phía (nhưng cũng có thể chỉ đưa vào một phía và thực hiện tìm kiếm ở phía kia)
- Khi đưa thuộc tính thể hiện liên kết vào cả 2 phía, thì phải bảo đảm tính đồng bộ:
  - Ví dụ, với liên kết 2 chiều giữa lớp Sinhviên và lớp Trường ĐH, thì ta phải bảo đảm rằng khi một sinh viên biết trường mà mình đang học, thì trường này phải có sinh viên đó trong danh sách sinh viên của trường
  - Tính đồng bộ đó phải được thực hiện bởi các thao tác quản lý thuộc tính thể hiện liên kết

THIẾT KẾ CÁC LIÊN KẾT

- Các liên kết lại thường kèm theo nhiều ràng buộc:
  - *Cơ số* (quy định số lượng tối đa và tối thiểu), *Hợp thành* (ràng buộc sự tồn tại đồng thời của toàn thể và bộ phận)
  - Ví dụ: Khi cài đặt liên kết kiểu hợp thành, thì phải chú ý là mỗi lần huỷ bỏ cái toàn thể thì phải huỷ bỏ cái bộ phận
  - *Các ràng buộc này phải được thực hiện bởi các thao tác quản lý các thuộc tính* thể hiện liên kết

THIẾT KẾ CÁC LIÊN KẾT

- Khi một liên kết có các thuộc tính, nhưng không có thao tác, thì liên kết đó có thể cài đặt như sau:
  - Nếu liên kết là một-một, thì các thuộc tính của liên kết có thể đưa vào một trong hai lớp tham gia liên kết;
  - Nếu liên kết là nhiều-một, thì các thuộc tính của liên kết có thể đưa vào lớp ở đầu nhiều;
  - Nếu liên kết là nhiều-nhiều, thì tốt nhất là lập thêm một lớp chứa các thuộc tính của liên kết.
- Nếu liên kết 2 ngôi có cả thuộc tính và thao tác, hoặc liên kết nhiều ngôi, thì ta phải lập thêm 1 lớp liên kết
  - Một lớp liên kết là một tập hợp các bộ 2 (hay bộ n) các đối tượng thuộc các lớp tham gia liên kết

THIẾT KẾ THUỘC TÍNH

- Việc thiết kế các thuộc tính chủ yếu là định nghĩa các kiểu cho các thuộc tính đã được đưa ra trong phân tích
  - Phần lớn các thuộc tính là thuộc các kiểu cơ sở có sẵn trong ngôn ngữ lập trình
  - Nhưng có các thuộc tính có kiểu là các cấu trúc dữ liệu cần được làm rõ
  - Các cấu trúc dữ liệu này được diễn tả như là một lớp
    - Để phân biệt chúng với các lớp bình thường, ta đưa thêm khuôn dập <<struct>>
    - Với kiểu liệt kê, thì ta dùng khuôn dập <<enum>>
    - Trong các cấu trúc dữ liệu này, thì các thuộc tính đều là công cộng (public), vì vậy thường là không cần có thêm các thao tác liên quan

THIẾT KẾ THUỘC TÍNH

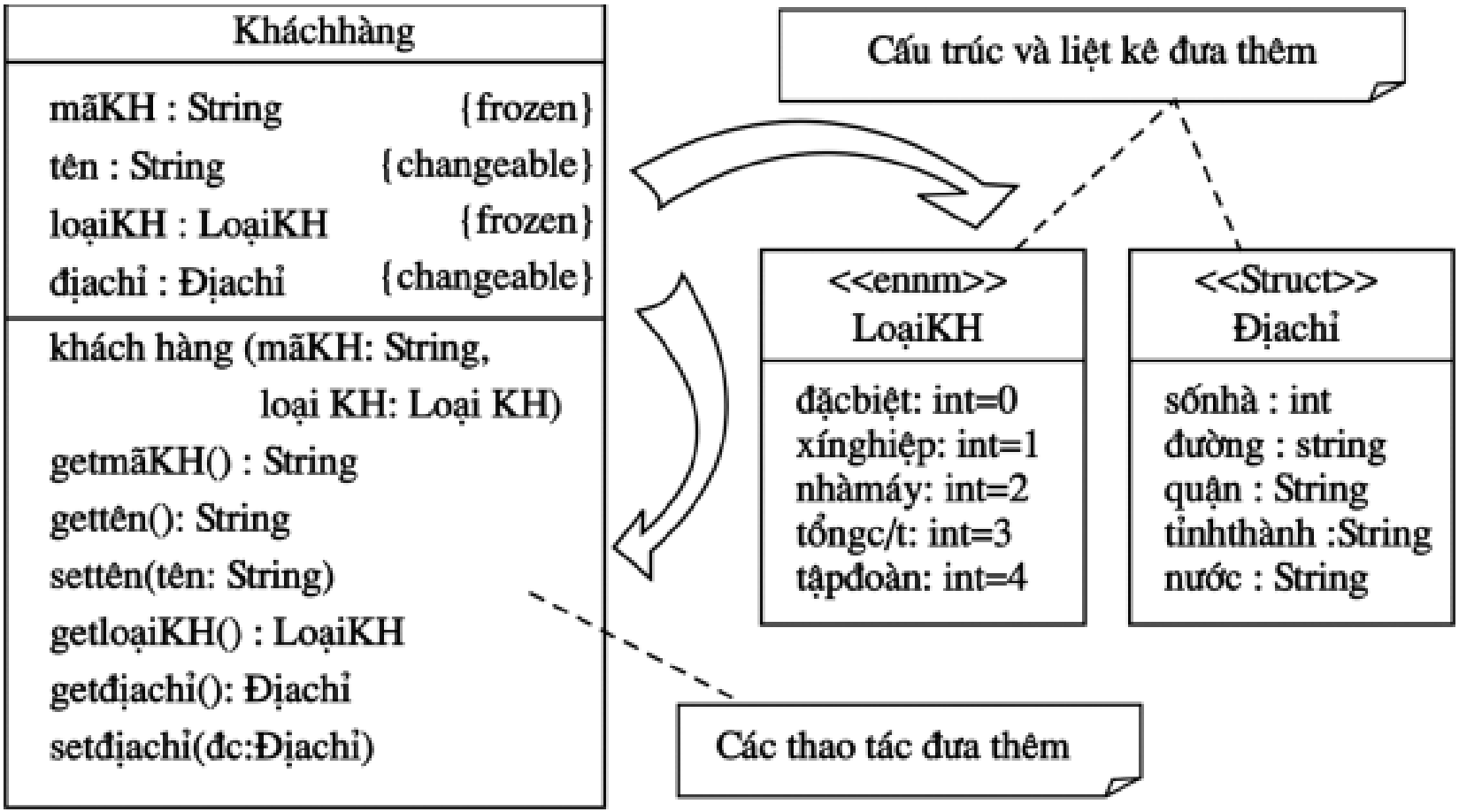
- Thiết kế các thuộc tính còn cần xác định tầm nhìn và cách truy cập chúng
  - Mặc định, thì các thuộc tính là riêng tư {*private*}
  - Các tính chất UML gắn cho thuộc tính như là {*frozen*}, {*changeable*} hay {*readOnly*} cần được thực hiện ngầm bởi các thao tác truy cập:
    - Các thuộc tính {*changeable*} cần có các thao tác truy cập *set<tên thuộc tính>* và *get<tên thuộc tính>*
    - Tính chất {*frozen*} đòi hỏi phải thiết lập giá trị ban đầu cho thuộc tính trong lời gọi khởi tạo (*constructor*) của lớp. Điều này thường được thực hiện thông qua một tham số khởi tạo.
    - Với thuộc tính {*readOnly*} thì chỉ có thao tác *get*, mà không có thao tác *set*

THIẾT KẾ THUỘC TÍNH

- Thiết kế các thuộc tính còn phải chỉ rõ các phương thức dùng để cập nhật các thuộc tính dẫn xuất
  - Các kỹ thuật sử dụng cho mục đích này đã được đề cập trong mục thiết kế lớp

THIẾT KẾ THUỘC TÍNH

- VD



THIẾT KẾ CÁC THAO TÁC

- Thiết kế các thao tác là bước cuối cùng trong vòng lặp thiết kế chi tiết
  - Đây là công việc tốn nhiều sức lực và thời gian nhất
  - Nội dung thiết kế các thao tác là đưa ra một hình ảnh chi tiết cho các phương thức
  - Để thiết kế và tài liệu hoá (logic xử lý của) các phương thức, UML đã cung cấp các biểu đồ mô hình hoá động thái, như là:
    - Biểu đồ hoạt động, cho phép mô tả một phương thức mà trong đó **có ít các lớp** tham gia. Biểu đồ này là rất hữu hiệu khi mô tả một giải thuật **có nhiều giai đoạn và có nhiều phép chọn lựa tình huống**.
    - Biểu đồ tương tác, cho phép trình bày một phương thức mà trong đó **có nhiều lớp tham gia**, nhưng lại **ít sự chọn lựa tình huống**

THIẾT KẾ CHI TIẾT CÁC TẦNG

- Thiết kế tầng trình bày
- Thiết kế tầng ứng dụng
- Thiết kế tầng nghiệp vụ
- Thiết kế việc lưu trữ dữ liệu

THIẾT KẾ TẦNG TRÌNH BÀY

- Vòng lặp của quy trình thiết kế chi tiết được áp dụng cho từng tầng trong kiến trúc khách/chủ (tầng trình bày, tầng ứng dụng, tầng nghiệp vụ, tầng truy cập dữ liệu và tầng lưu trữ dữ liệu)
- Tầng trình bày giới hạn ở phần trông thấy được của ứng dụng; đó là phần thực hiện giao diện giữa người và máy tính
  - Đây là phần dễ thay đổi nhất của hệ thống, bởi vì kỹ thuật làm giao diện tiến bộ rất nhanh, mà người dùng thì sau một thời gian khai thác hệ thống lại thường muốn có những đổi mới trong hình thức giao tiếp

THIẾT KẾ TẦNG TRÌNH BÀY

- Hình thức giao diện người dùng phổ biến nhất hiện nay là giao tiếp qua các cửa sổ (window). Hình thức giao diện cửa sổ có 3 khái niệm chính:
  - Các cửa sổ và nội dung bên trong đó, mà người dùng có thể trông thấy, dịch chuyển, thay đổi kích cỡ... Đó là phần nhìn thấy hay là phần **cấu trúc tĩnh (hiển thị)** của giao diện.
  - Các hành động mà người dùng có thể kích hoạt tạo nên sự thay đổi trạng thái của giao diện. Đó là phần **hành vi** của giao diện.
  - Các luồng thông tin mà hệ thống đưa ra cho người dùng hay người dùng đưa vào hệ thống thông qua các điều khiển kiểu listbox, textbox,... Đó là phần trao đổi thông tin với ứng dụng, tức là phần **chức năng** của giao diện

THIẾT KẾ TẦNG TRÌNH BÀY

- Thiết kế và tài liệu hoá tầng trình bày trở thành việc xem xét ba khái niệm nói trên (sự hiển thị, hành vi và chức năng)
- Sự trợ giúp dễ dàng và thuận lợi của các công cụ GUI là rất dễ bị lạm dụng, dẫn tới sự vi phạm nghiêm trọng **nguyên tắc cố kết cao giữa các lớp trong một tầng và liên kết yếu giữa các lớp thuộc các tầng khác nhau**.
  - Dễ có xu hướng là từ giao diện người dùng ta cho gọi trực tiếp tới các đối tượng thực thể để lấy thông tin, và điều đó sẽ tạo ra một sự liên kết chằng chịt, khó kiểm soát giữa tầng trình bày với các tầng dưới nó
  - Để tránh nguy cơ này, trong thiết kế tầng trình bày, ta nên áp dụng mô hình **MVC (Model-View-Controller)**, nhằm tạo ra một sự phân công tách biệt giữa 3 thành phần MVC như sau:

THIẾT KẾ TẦNG TRÌNH BÀY

- **Model (Mô hình)**, đó là lớp thực thể (ví dụ: Khách hàng, Đơn hàng) thuộc tầng nghiệp vụ
  - Đây là nguồn gốc của thông tin được hiển thị trên giao diện người dùng
  - Tuy nhiên, giao diện không truy cập trực tiếp vào đối tượng thực thể, mà tầng ứng dụng sẽ gom (tạo) thông tin từ các đối tượng thực thể vào các "**tài liệu**" (document), và giao diện người dùng truy cập vào tài liệu để lấy thông tin
  - Thường thì có sự tương ứng 1-1 giữa khung nhìn (view) và tài liệu, song một tài liệu có thể tương ứng với một hay nhiều đối tượng thực thể

THIẾT KẾ TẦNG TRÌNH BÀY

- **View (Khung nhìn)**, đó là phần hiển thị của giao diện mà người dùng trông thấy
  - Khung nhìn phải quản lý các đối tượng đồ hoạ (cấu trúc tĩnh) của mình, và có thể truy cập vào các tài liệu (documents) (được quản lý bởi tầng ứng dụng) để lấy thông tin hiển thị hay để cập nhật thông tin
- **Controller (Bộ điều khiển)**, đó là phần đảm trách việc quản lý động thái (hành vi) của giao diện, thực hiện sự chuyển đổi trạng thái của giao diện theo các sự kiện kích hoạt từ phía người dùng
  - Có sự tương ứng 1-1 giữa View và Controller
  - ***Thông thường thì (View + Controller) tạo nên tầng trình bày***

THIẾT KẾ TẦNG ỨNG DỤNG

- Tầng ứng dụng bao gồm các lớp điều khiển
  - Đã được chỉ ra trong phân tích, trên mô hình cấu trúc tĩnh của hệ thống
  - Có nhiệm vụ dẫn dắt các quá trình của ứng dụng, cũng như thực hiện các quy tắc cần bảo đảm của ứng dụng

THIẾT KẾ TẦNG ỨNG DỤNG

- Có ba loại quy tắc nghiệp vụ, hay còn gọi là các **luật nghiệp vụ (business rules)**:
  - Các **luật toàn vẹn** phát biểu rằng điều gì đó phải luôn đúng (Ví dụ: Một thuộc tính phải có giá trị nguyên từ 1 đến 5)
  - Các **luật dẫn xuất** phát biểu rằng một giá trị (hay một tập các giá trị) được tính như thế nào (Ví dụ: thànhtiền = đơngiá x sốlượng)
  - Các **luật hành vi** mô tả sắc thái động của hành vi, như là các điều kiện nào phải đúng khi bắt đầu thực hiện một hành động (Ví dụ: Khi cửa sổ lò vi sóng mở ra thì đèn phải bật sáng)

THIẾT KẾ TẦNG ỨNG DỤNG

- 3 luật nêu trên đã được nghiên cứu trong giai đoạn phân tích và đã được ánh xạ vào các lớp điều khiển
- Tuy nhiên, còn có các luật (các yêu cầu kỹ thuật) trong thiết kế chi tiết ta phải ý chú đến:
  - Sự **đồng bộ hoá** của nhiều cửa sổ trên các dữ liệu chung
  - Sự **tối ưu hoá** việc nạp tải các dữ liệu (data load) tại các trạm người dùng
  - Sự **kiểm soát** chặt chẽ quá trình tương tác với người dùng
- Để đáp ứng các yêu cầu trên, thì việc thiết kế chi tiết đối với tầng ứng dụng được thực hiện dựa trên sự định nghĩa các "tài liệu“ (documents), biểu diễn cho hình ảnh bộ nhớ khi trao đổi dữ liệu với các cửa sổ. Theo định nghĩa, thì có một tài liệu cho mỗi cửa sổ. Trên cơ sở của khuôn khổ cài đặt này, ta sẽ lần lượt xem xét ở sau đây ba yêu cầu về kỹ thuật vừa nói trên.

THIẾT KẾ TẦNG ỨNG DỤNG

a) Đồng bộ hoá các cửa sổ

- Đồng bộ hoá các cửa sổ chính là việc làm cho các cửa sổ khác nhau là không mâu thuẫn với nhau khi chúng trình bày về dữ liệu có chung nguồn gốc
  - Ví dụ: Cùng một số thông tin địa lý, nhưng được trình bày trên 2 cửa sổ: cửa sổ văn bản và cửa sổ bản đồ
- Để đảm bảo tính đồng bộ của các thông tin trình bày, ta phải *thiết lập các mối liên quan kết nhập giữa các tài liệu tương ứng với các cửa sổ (ở tầng ứng dụng) để chỉ rõ sự chia sẻ thông tin giữa chúng*

THIẾT KẾ TẦNG ỨNG DỤNG

b) Tối ưu hoá việc tải nạp thông tin từ các thực thể

- Trong một hệ phân tán, thì các đối tượng thực thể thường cư trú một cách phân tán, làm cho việc truy cập vào chúng trở nên tốn kém
  - Để tối ưu hoá việc tải nạp thông tin từ các thực thể tới tầng ứng dụng (cho tài liệu), ta phải tìm cách giảm thiểu số lần truy cập
  - Muốn thế, ta phải ghi nhớ các thông tin đã tải nạp, để tránh việc truy cập lặp đi lặp lại trên cùng một thực thể

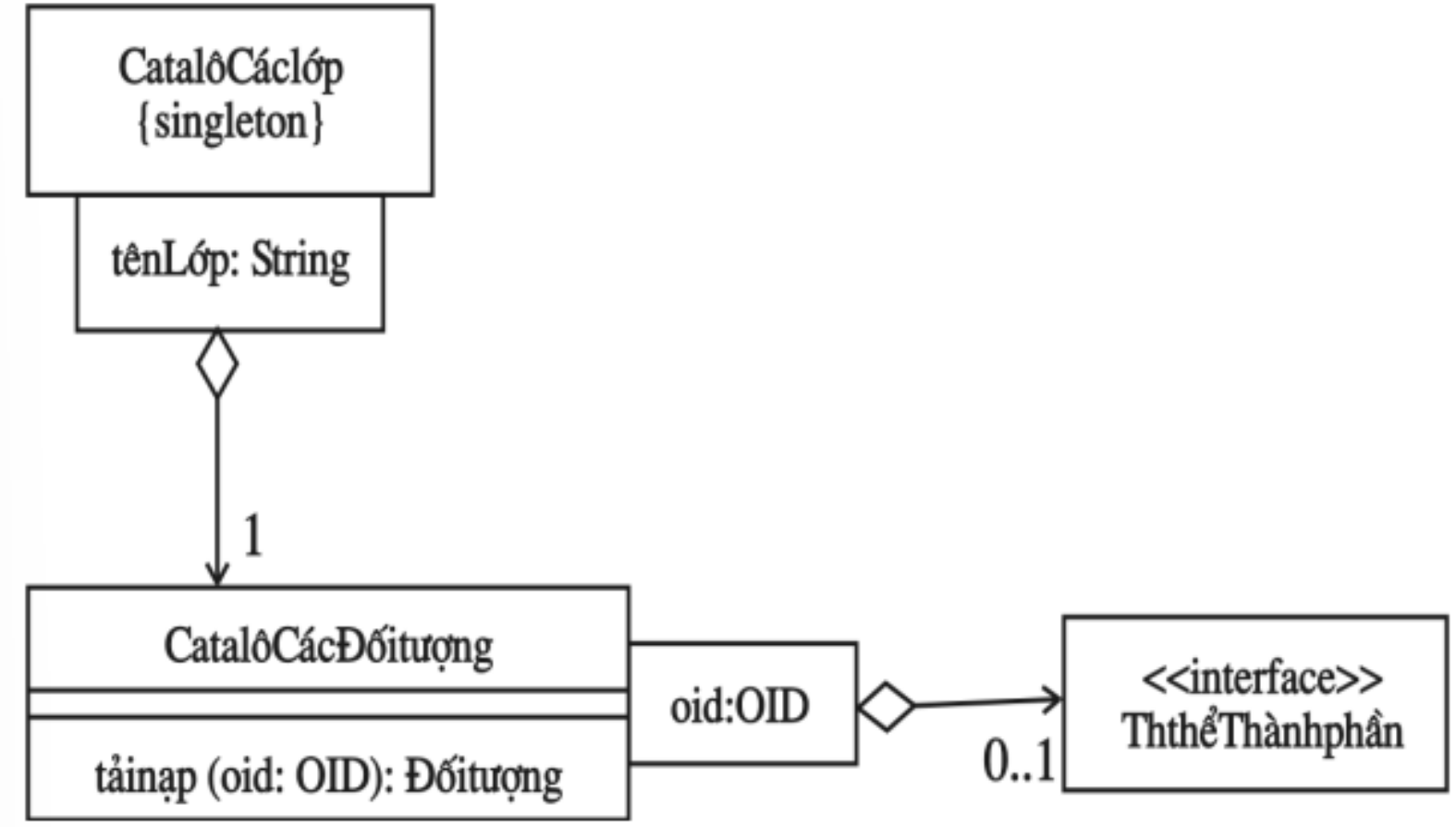
THIẾT KẾ TẦNG ỨNG DỤNG

b)Tối ưu hoá việc tải nạp thông tin từ các thực thể (tiếp)

- Người ta dùng một **catalô tham chiếu** các đối tượng thực thể theo OID (Object identifier) của chúng
  - Để tối ưu hoá, catalô được tổ chức thành catalô con, sắp xếp theo tên của lớp, để từ đó tìm đến các đối tượng theo OID
  - Giao diện "ThthểThànhphần" biểu diễn cho các thực thể đến từ các thành phần phân tán. Khi tải nạp mới một thực thể, thì một đối tượng của giao diện ThthểThànhphần được tạo lập để ghi nhận các thông tin đã tải nạp.
  - Nhờ đó những lần tải nạp sau sẽ tìm lại được các thông tin này (trong catalô) mà không phải truy cập trực tiếp vào thực thể nữa

THIẾT KẾ TẦNG ỨNG DỤNG

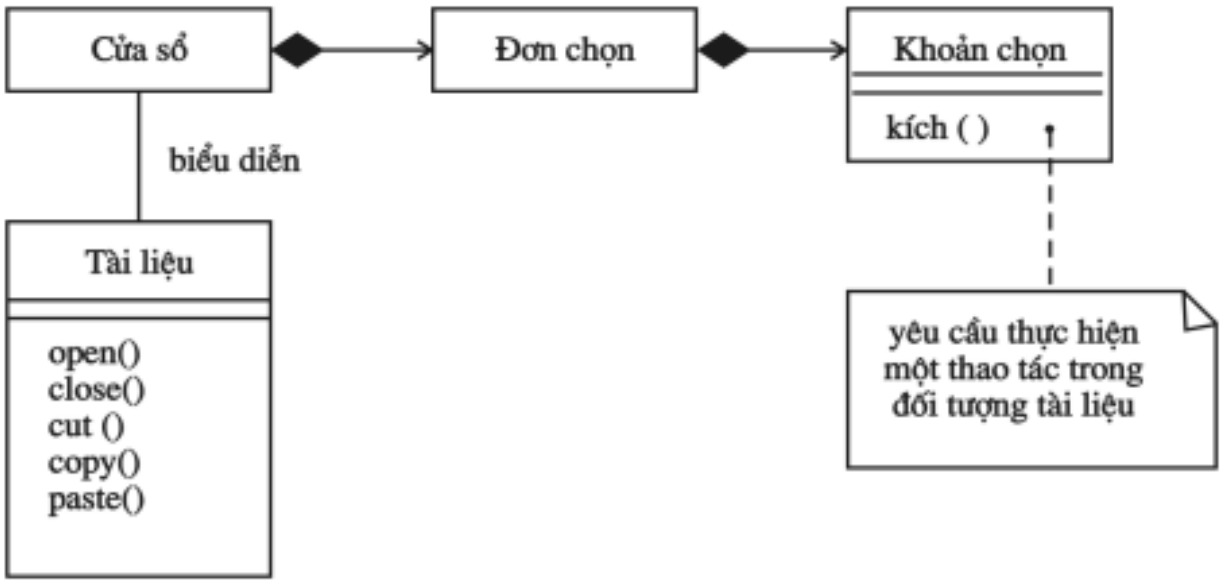
- Tổ chức catalô để lưu giữ các thông tin đã tải nạp:



THIẾT KẾ TẦNG ỨNG DỤNG

c) Tổ chức sự kiểm soát chặt chẽ quá trình tương tác với người dùng

- Khi làm việc với một cửa sổ, người dùng có thể phát ra một lệnh (command), ví dụ: bằng cách kích vào một nút (button) hay kích vào một lựa chọn (listbox). Lệnh đó có mục đích gọi một thao tác nào đó trong đối tượng tài liệu, ứng với cửa sổ đó:



THIẾT KẾ TẦNG ỨNG DỤNG

- Tuy nhiên, lệnh phát ra từ cửa sổ không thể truyền lập tức và trực tiếp tới tài liệu (nơi nhận lệnh), vì nhiều lý do:
  - Lệnh có thể cần được chặn lại để kiểm tra quyền sử dụng;
  - Lệnh phải được lưu giữ lại, để đưa vào hàng đợi, đến lượt mới được thực hiện;
  - Dãy các lệnh thực hiện liên tiếp cần phải quản lý và ghi nhớ để khi cần có thể quay trở lại (undo), hay thực hiện lại (redo);
- Tất cả các yêu cầu về quản lý lệnh nêu trên không thể giao cho cửa sổ phụ trách được, vì các cửa sổ được thiết kế theo các bộ công cụ (GUI toolkits), phải giữ tính độc lập với ứng dụng
- Vì vậy, chính ứng dụng phải tự tổ chức việc quản lý các lệnh này một cách chặt chẽ và sát với yêu cầu của mình

THIẾT KẾ TẦNG NGHIỆP VỤ

- Tầng nghiệp vụ chứa các lớp thực thể, phản ánh lĩnh vực ứng dụng
  - Đây là tầng ổn định nhất và có khả năng tái sử dụng cao
- Bước đầu, thì mỗi lớp thực thể bao gồm các thuộc tính và thao tác như đã phát hiện ở giai đoạn trước. Tuy nhiên, sau đó thì ta phải đưa thêm:
  - Các thao tác lấy giá trị (get) và thiết lập giá trị (set) cho mỗi thuộc tính, vì các thuộc tính đều được khai báo là riêng tư;
  - Các dịch vụ mà mỗi lớp thực thể phải cung cấp, nhằm đáp ứng các yêu cầu tìm kiếm thông tin theo những trạng thái của hệ thống;
  - Các thao tác đáp lại các yêu cầu cập nhật, bằng cách chuyển các yêu cầu cập nhật xuống tầng dưới (tầng lưu trữ dữ liệu).

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- Có những đối tượng cần phải lưu trữ lại một cách lâu dài (trên bộ nhớ ngoài), chứ không thể để mất đi cùng với sự kết thúc chương trình
- Đó là các đối tượng trường cửu (persistent data objects)
  - Thường thì đó là các đối tượng thực thể
  - Hiếm khi là các đối tượng biên (giao diện) hay đối tượng điều khiển

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

a) Chọn lựa cách lưu trữ dữ liệu

- Việc mô hình hoá hệ thống bằng các lớp và đối tượng tạo nhiều thuận lợi cho việc lưu trữ dữ liệu. Có một số cách lưu trữ dữ liệu phổ biến sau đây:
  - Các **hệ thống tệp** (tập tin): Đó là phương tiện lưu trữ đơn giản nhất
    - Tuy cho phép đọc và viết các đối tượng, nhưng không thích hợp cho các truy vấn dữ liệu phức tạp
    - Ít được dùng với các hệ thống lớn, có tầm quan trọng
  - Các **cơ sở dữ liệu quan hệ** (RDBMS): Cho phép quản lý dữ liệu có quan hệ (relational data)
    - Tồn tại nhiều hệ quản trị CSDL phù hợp cho các yêu cầu khác nhau về khối lượng, sự phân tán, và hệ điều hành
    - Đây là cách lưu trữ dữ liệu được dùng phổ biến nhất

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- Các **cơ sở dữ liệu hướng đối tượng** (OODBMS): Cho phép lưu trữ và quản lý các đối tượng một cách trực tiếp
  - Nhờ vậy mà có thể nói khâu thiết kế cho việc lưu trữ dữ liệu hầu như chẳng còn việc gì phải làm
  - Tuy nhiên các hệ quản trị CSDL đối tượng còn chưa chiếm được thị phần lớn trên thực tế
- Các **cơ sở dữ liệu phi quan hệ** (non-relational/noSQL DBMS): Cho phép quản lý dữ liệu phi quan hệ (non-relational data)
  - Có 4 nhóm chính: Key-value, Column, Document, Graph
  - Phù hợp khi: Kiểu dữ liệu phi quan hệ, Dữ liệu ở nhiều khuôn dạng (vd: dữ liệu mạng xã hội, hình ảnh,..), Kích thước lưu trữ rất lớn (vd: terabytes), Tốc độ cập nhật/bổ sung dữ liệu rất lớn
- Phần tiếp theo đây, ta đề cập việc thiết kế CSDL với (giả sử) sự chọn lựa một hệ quản trị CSDL quan hệ

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

b) Ánh xạ các lớp sang bảng

- Mỗi lớp trường cửu tương ứng với một bảng trong mô hình quan hệ:
  - Mỗi thuộc tính của lớp tương ứng với một cột của bảng,
  - Mỗi cá thể (đối tượng) của lớp tương ứng với một dòng (một bộ-n) của bảng, trong đó OID của đối tượng đóng vai trò là khoá chính (primary key) trong bảng.
- Có một số thuộc tính của lớp có thể có kiểu phức tạp (các cấu trúc dữ liệu) và không tương ứng được với các kiểu của SQL. Vậy phải có sự biến đổi các kiểu phức tạp đó:
  - Hoặc là bằng nhiều cột, mỗi cột tương ứng với một trường trong cấu trúc dữ liệu.
  - Hoặc là bằng một bảng riêng biệt, liên hệ với bảng chính bằng một khoá ngoài, cho phép kết nối các đối tượng với các giá trị của thuộc tính phức tạp.

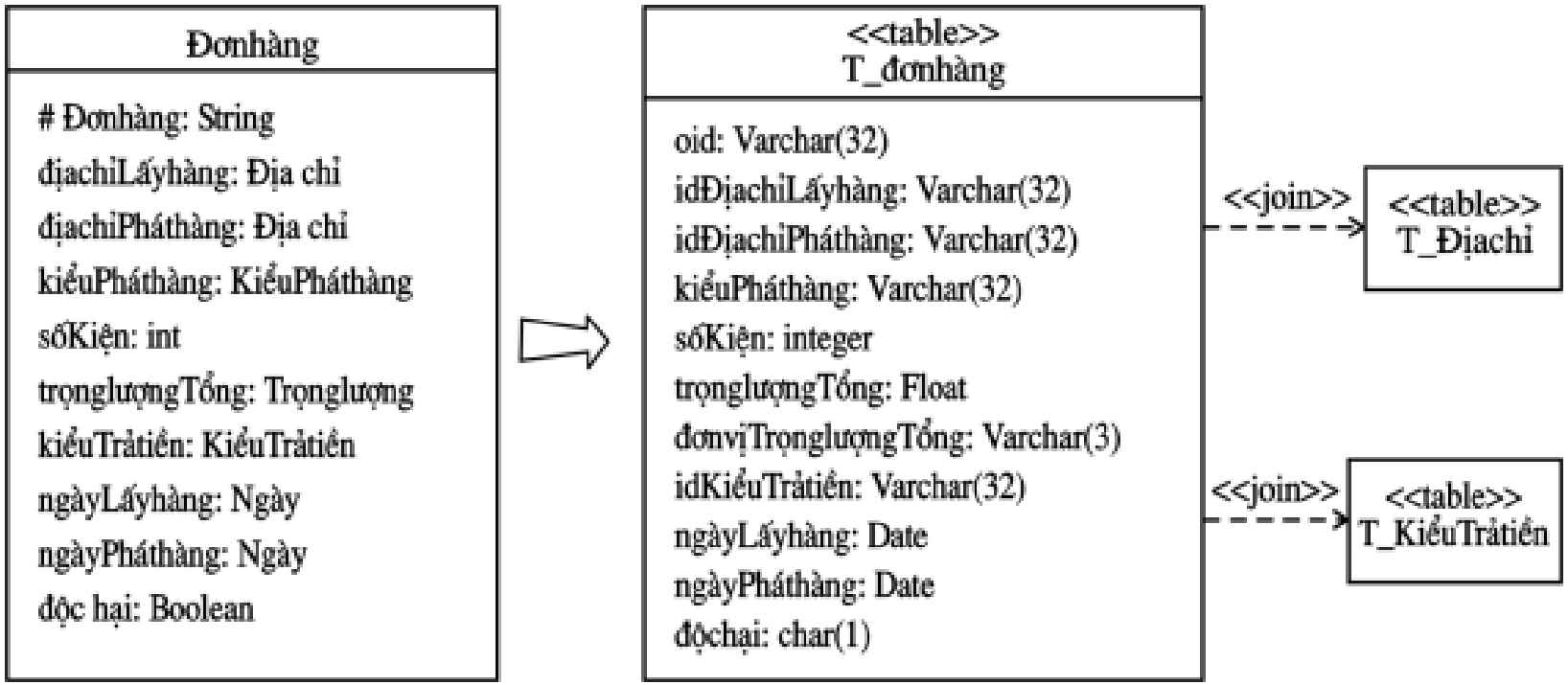
4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- Sự tương đương giữa lớp và bảng

Mô hình đối tượng	**Mô hình quanhệ**
Lớp	Bảng
Thuộc tính có kiểu đơn	Cột
Thuộc tính có kiểu phức tạp	Các cột, hay Bảng và khoá ngoài
Đối tượng	Bộ-n
OID	Khoá chính
Liên kết và Kết nối	Khoá ngoài hay bảng
Thừa kế	Khoá chính đồng nhất trên nhiều bảng

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Ví dụ*: Ánh xạ lớp Đơnhàng sang bảng. Kết quả của sự ánh xạ là một biểu đồ lớp (trong UML), diễn tả cấu trúc lưu trữ, trong đó mỗi lớp đều mang khuôn dập <<table>>, và tên lớp được gắn thêm tiền tố T_. Một công cụ trợ giúp (ví dụ: Rational Rose) sẽ biến đổi biểu đồ này thành ngôn ngữ định nghĩa dữ liệu (DDL), cho phép tạo lập các bảng trong CSDL.

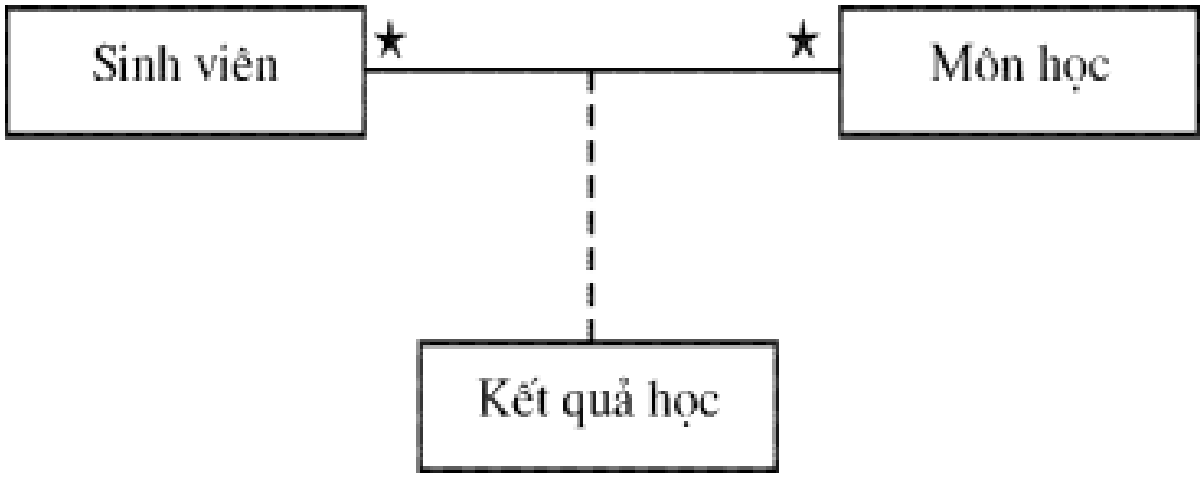


4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- Ánh xạ từ lớp sang bảng lại không nên hiểu máy móc là ánh xạ 1-1, vì ánh xạ 1-1 thường dẫn tới các hiện tượng không tốt như sau:
  - *Quá nhiều bảng và quá nhiều kết nối phải thực hiện*: Thông thường thì biểu đồ lớp có thể chứa rất nhiều lớp, nếu cứ ánh xạ 1-1 thành các bảng, thì ta sẽ có rất nhiều bảng. Mà khi đã có nhiều bảng, thì để tìm kiếm thông tin trong CSDL, ta lại phải áp dụng nhiều phép kết nối (join), ảnh hưởng nặng nề tới tốc độ truy cập. Vì vậy nên cố gắng **gom các bảng thường đi liền với nhau trong truy xuất dữ liệu thành các bảng lớn**.

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Thiếu các bảng*: Như ta đã thấy trong mục thiết kế các liên kết, thì nếu giữa hai lớp có một liên kết nhiều-nhiều mà liên kết đó lại có thuộc tính, thì phải định nghĩa một lớp liên kết chứa thuộc tính đó. Chẳng hạn giữa Sinh viên và Môn học có một liên kết nhiều-nhiều ("Sinh viên hoàn thành môn học"); liên kết này lại có thuộc tính (như là năm học, kết quả thi), do đó trong mô hình đối tượng, phải thêm một lớp liên kết là lớp Kết quả học. Chuyển sang mô hình quan hệ, thì ngoài hai bảng Sinh viên và Môn học, ta cũng có bảng Kết quả học.



4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Mất khả năng thừa kế*: Mô hình quan hệ không sẵn sàng hỗ trợ cho sự thừa kế. Vì vậy một ánh xạ đơn giản các lớp sang các bảng, mà không có các xử lý đặc biệt (xem ở dưới) thì không còn giữ được mối liên quan khái quát hoá và sự thừa kế.
- *Suy giảm hiệu năng*: Một ánh xạ 1-1 thường chuyển biểu đồ lớp thành mô hình quan hệ ở dạng chuẩn 3. Tuy nhiên có nhiều ứng dụng hướng tới một số báo cáo nhất định (reporting functionality), đòi hỏi phải hạ chuẩn đối với dữ liệu (chẳng hạn phải lặp lại cùng một dữ liệu ở nhiều bảng, hay sáp nhập một số bảng) nhằm tạo khả năng truy cập thuận lợi nhất cho những mục đích định sẵn (vd: tạo báo cáo thống kê) của ứng dụng đó

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

c) Ánh xạ các liên kết

- Trong phần trên, ta đã đề cập việc thiết kế chi tiết các liên kết, vốn không có sẵn trong các ngôn ngữ lập trình hướng đối tượng. Nay ta ánh xạ các liên kết giữa các lớp trường cửu vào các bảng trong mô hình quan hệ, ta cũng phải chuyển các liên kết thành các khoá ngoài hay bảng liên kết. Sự khác biệt chỉ là ở chỗ các thuộc tính trong các bảng của mô hình quan hệ đều phải là thuộc tính đơn.
  - *Với liên kết một-một*: Lập một bảng cho mỗi lớp (bảng A, B). Khoá chính của mỗi bảng cũng là khoá ngoài đối với bảng kia. Thực ra lập thành hai bảng là chỉ thực sự cần thiết khi liên kết là một ánh xạ vào (1 đến 0..1), còn khi liên kết là một đối một, thì cách tốt nhất là chỉ lập một bảng

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Với liên kết một-nhiều*: Lập một bảng cho mỗi lớp (bảng A, B). Khoá chính của bảng A (đầu Một) là khoá ngoài trong bảng B (đầu Nhiều).
- *Với liên kết nhiều-nhiều*: Lập một bảng cho mỗi lớp (bảng A, B). Lập thêm một bảng kết nối, hay còn gọi là bảng giao (bảng C). Khoá chính của mỗi bảng A, B được định nghĩa là khoá ngoài trong bảng C. Khoá chính của C có thể là một cột riêng song cũng có thể là khoá bội hợp thành từ hai khoá ngoài.

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Với kết nhập và hợp thành*: Trong mô hình quan hệ thì kết nhập và hợp thành cũng chỉ được mô hình hoá như một liên kết bình thường. Hợp thành khi có cơ số là 1-1, thì nên cài đặt thành một bảng duy nhất. Nếu hợp thành được cài đặt thành hai bảng riêng biệt, thì việc lớp tổng thể tạo mới/loại bỏ lớp bộ phần phải được xem xét và cài đặt trong CSDL vật lý. Còn kết nhập, thì cứ nên để thành hai bảng, vì các đối tượng bị kết nhập vẫn có thể tồn tại một cách độc lập.
- *Với các liên kết đệ qui*: Đó là loại liên kết giữa một lớp với chính nó, cho ta các kết nối giữa các cặp đối tượng của lớp. Liên kết đệ qui được cài đặt bằng cách thêm vào một cột, mà giá trị là các giá trị khoá chính của lớp đó, xem như là một khoá ngoài.

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

d) Ánh xạ mối liên quan khái quát hoá

- Nếu hầu hết các ngôn ngữ lập trình hướng đối tượng hỗ trợ cho mối liên quan này giữa các lớp (sự thừa kế), thì mô hình quan hệ không hỗ trợ. Tuy nhiên, có một số cách để thể hiện mối liên quan khát quát hóa trong mô hình quan hệ:
- *Lập một bảng cho mỗi lớp* (lớp cha cũng như lớp con). Mối liên quan giữa các bảng được giải quyết theo hai cách:
  - Dùng một OID chung cho mọi bảng trong cùng một phả hệ thừa kế. Như vậy, để truy cập dữ liệu của một đối tượng phải thực hiện phép kết nối (join) giữa lớp cha và lớp con;
  - Lập một khung nhìn SQL (view) cho mỗi cặp lớp cha/lớp con. Như vậy, số bảng tăng thêm, và thực chất cũng phải dùng phép kết nối (join) khi truy cập dữ liệu.
- Cả 2 cách làm này đều cho phép tăng thêm các lớp con trong tương lai, mà không làm xáo trộn các lớp cũ.

4.4 THIẾT KẾ VIỆC LƯU TRỮ DỮ LIỆU

- *Chỉ lập một bảng* (ứng với lớp cha), mọi thuộc tính ở các lớp con đều dồn về bảng trên. Như vậy số bảng là tối thiểu (=1), nhưng số cột tăng thêm, và với mỗi cá thể, có những cột là không dùng tới. Vậy chỉ **thích hợp khi số các thuộc tính trong các lớp con là ít**. Không thuận lợi cho việc thêm mới các lớp con.
- *Lập một bảng cho mỗi lớp con*, lặp lại các thuộc tính của lớp cha vào bảng của mỗi lớp con (hạ chuẩn). Như vậy số bảng rút bớt (không có bảng cho lớp cha), và việc tăng thêm lớp con trong tương lai không gây ảnh hưởng, nhưng việc điều chỉnh lớp cha buộc phải điều chỉnh lại các bảng. **Thích hợp khi số các thuộc tính trong lớp cha là ít**.