Lecture 08:
Mô hình hướng đối tượng

- **Phân tích hướng đối tượng**
  - Phân tích cơ sở
  - Định nghĩa lớp (Classes)
  - Các thuộc tính (Attributes) và phương thức (Operations)
- **Biểu đồ lớp UML (Class Diagrams)**
  - Quan hệ kết hợp (Associations)
  - Tính bội/Bản số (Multiplicity)
  - Quan hệ tập hợp (Aggregation)
  - Quan hệ hợp thành (Composition)
  - Quan hệ thừa kế (Generalization)

Phân tích hướng đối tượng

Background

- Mô hình yêu cầu trong thuật ngữ của các đối tượng và dịch vụ mà chúng cung cấp
- Phát sinh thiết kế hướng đối tượng
  - Được áp dụng để mô hình hóa lĩnh vực ứng dụng hơn là chương trình

Động cơ

- OO (được kiến nghị) là quá 'tự nhiên'
  - Khi triển khai một hệ thống, các chức năng thực thi của nó cần được thay đổi thường hơn là các đối tượng đang hoạt động trên nó…
  - …một mô hình dựa trên các đối tượng (hơn là các chức năng) sẽ rất ổn định…
  - …vì thế, có kiến nghị rằng các thiết kế hướng đối tượng có thể sẽ còn tiếp tục được duy trì
- OO nhấn mạnh sự quan trọng của giao tiếp giữa các đối tượng một cách rõ ràng.
  - đã được so sánh với sự mơ hồ của các quan hệ dòng dữ liệu

NOTE: Áp dụng OO cho kỹ nghệ yêu cầu vì nó là một công cụ mô hình hóa. Song, chúng ta đang mô hình hóa các thực thể trong lĩnh vực chứ không phải thiết kế hệ thống mới.

UML là gì ?

UML (Unified Modeling Language)

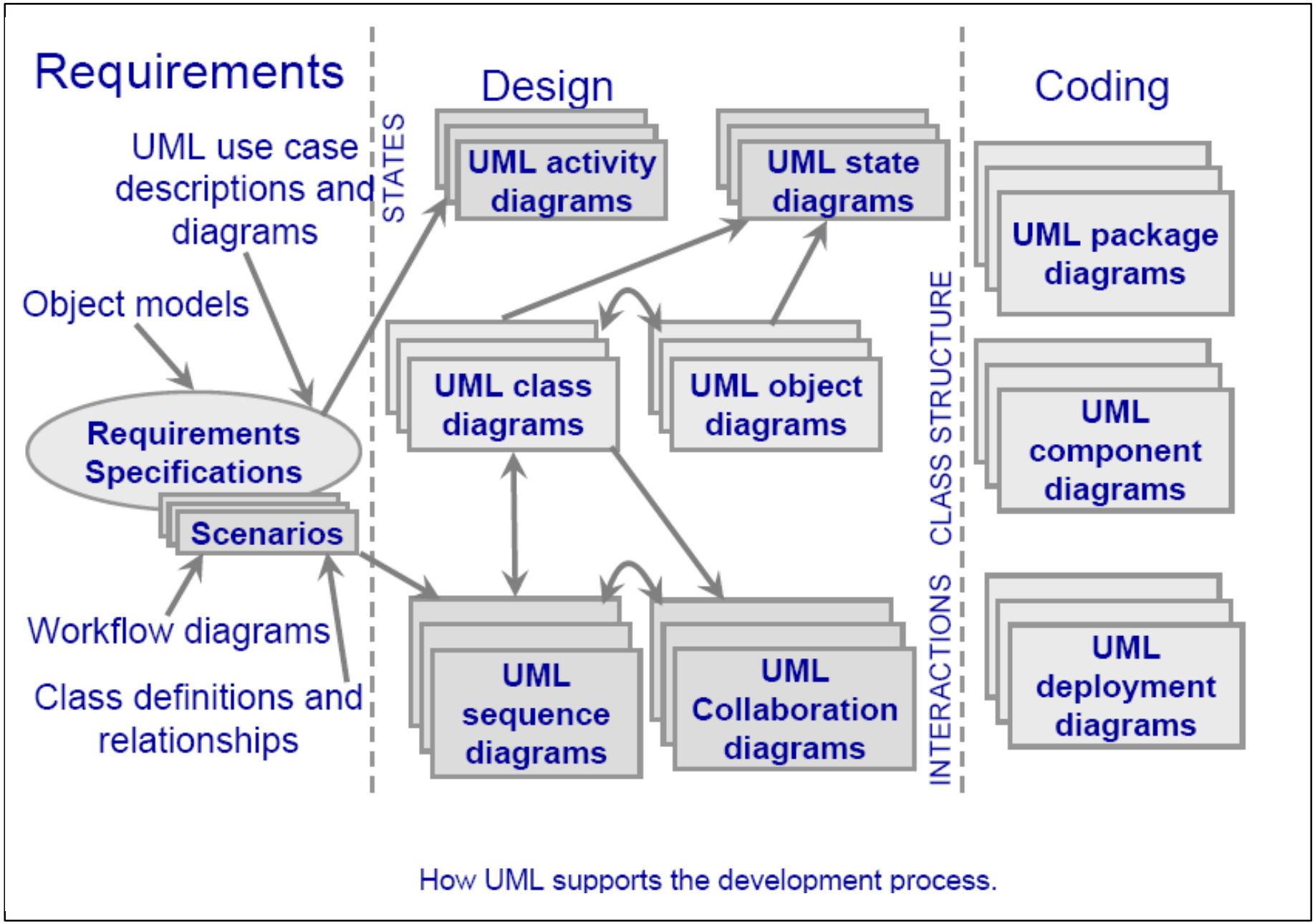
- **Một công nghệ chuẩn cho việc mô hình hóa phần mềm hướng đối tượng**
- **Là kết quả của sự thống nhất hệ thống ký hiệu của 3 phương pháp hướng đối tượng tiêu biểu :**
  - **OMT (James Rumbaugh) -** Object Modeling Technique
  - **OOSE (Ivar Jacobson)** - Object-Oriented Software Enginering
  - **Booch91 (Grady Booch)**

Được hỗ trợ bởi một số CASE tools:

- **Rational ROSE**
- **TogetherJ**

Bạn có thể mô hình 80% của hầu hết vấn đề bằng cách dùng chỉ khoảng 20% UML

Chúng ta học 20% đó



How UML supports the development process.

Gần như mọi thứ đều có thể là object…

Source: *Adapted from Pressman, 1994, p242*

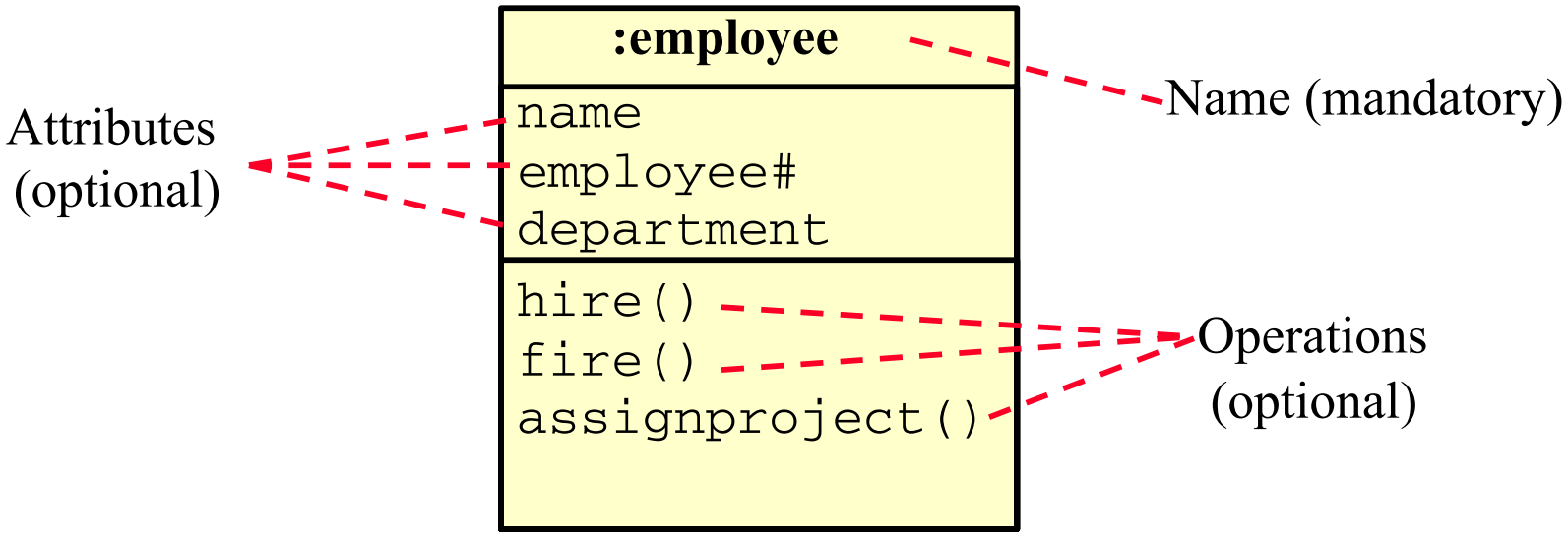
- **Các thực thể bên ngoài**
  - **…tương tác với hệ thống đang được mô hình hóa**
    - **E.g. people, devices, other systems**
- **Các vật thể**
  - **…là những phần của lĩnh vực đang được mô hình hóa**
    - **E.g. báo cáo, màn hình, tín hiệu, etc.**
- **Việc xảy ra hoặc sự kiện**
  - **…xuất hiện trong ngữ cảnh của hệ thống**
    - **E.g. chuyển giao tài nguyên, hành động kiểm soát, etc.**
- **Vai diễn**
  - **được đóng bởi những người đang tương tác với hệ thống**
- **Các thành phần tổ chức**
  - **có liên quan tới ứng dụng**
    - **E.g. phân chia, nhóm, đội, etc.**
- **Nơi chốn**
  - **…thiết lập ngữ cảnh của vấn đề đang được mô hình hóa**
    - **E.g. nhà máy, bến tàu, etc.**
- **Các cấu trúc**
  - **định nghĩa một lớp hay nhóm các objects**
    - **E.g. bộ cảm biến, bộ bánh xe, máy tính, etc.**

Một số thứ không thể là objects:

- **các thủ tục (e.g. in ấn, đảo ngược, etc)**
- **các thuộc tính (e.g. màu xanh, 50Mb, etc)**

Lớp (classes) là gì ?

- **Một *lớp* mô tả một nhóm các đối tượng (objects) với :**
  - Các đặc tính tương tự (thuộc tính - attributes),
  - Cùng hành vi ứng xử (phương thức - operations),
  - Quan hệ như nhau đối với các object khác.
  - Và có chung ngữ nghĩa ("semantics").
- **Ví dụ**
  - **Nhân viên (employee): có 1 tên (name), mã số nhân viên (employee#) và bộ phận trực thuộc (department); một nhân viên thì có thể được thuê (hired), và bị sa thải (fired); mỗi nhân viên làm việc trong một hay nhiều dự án.**



Tìm lớp (Classes)

- **Tìm *lớp* từ dữ liệu nguồn:**
  - **Tìm các danh từ và cụm danh từ trong mô tả vấn đề của các đối tác**
    - **chứa trong mô hình nếu họ giải thích một cách tự nhiên hoặc cấu trúc thông tin trong ứng dụng.**
- **Tìm *lớp* từ các nguồn khác:**
  - **Xem xét các thông tin nền tảng;**
  - **Những người dùng và các đối tác khác;**
  - **Các mẫu phân tích;**
- **Sẽ rất tốt nếu ban đầu có nhiều ứng viên cho *lớp***
  - **Bạn có thể bỏ chúng ngay sau đó nếu chúng hóa ra không hữu ích**
  - **Quyết định dứt khoát loại bỏ các *lớp* thì tốt hơn là chỉ suy nghĩ về điều đó.**

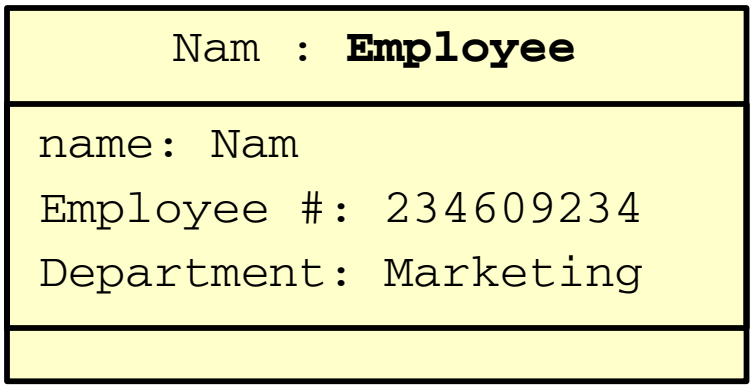
Chọn lựa lớp

Loại bỏ *lớp* là một khái niệm khi :

- Không nằm trong phạm vi phân tích;
- Tham chiếu đến toàn bộ hệ thống;
- Trùng với các lớp khác;
- Có quá nhiều mơ hồ hoặc quá chi tiết
  - e.g. có quá nhiều hoặc quá ít thể hiện (instances)
- Tiêu chuẩn của Coad & Yourdon's:
  - Giữ lại thông tin : Hệ thống sẽ còn nhớ thông tin về các lớp này của objects?
  - Các dịch vụ cần thiết : Objects trong lớp này có nhận biết các phương thức làm thay đổi giá trị các thuộc tính của chúng ?
  - Đa thuộc tính (Multiple Atributes): Nếu lớp chỉ có duy nhất một thuộc tính, nó có thể đại diện tốt hơn một thuộc tính của lớp khác
  - Thuộc tính chung (Common Attributes): Lớp có chứa các thuộc tính mà có thể chia sẻ với tất cả các thể hiện của đối tượng đó không ?
  - Phương thức chung (Common Operations): Lớp có chứa các phương thức mà có thể chia sẻ với tất cả các thể hiện của đối tượng đó không ?
- Các thực thể bên ngoài phát sinh hoặc thu nhận các thông tin chủ yếu cho hệ thống nên được đặt thành *lớp*.

Objects vs. Classes

- **Các thể hiện của một lớp được gọi là đối tượng**
  - **Một đối tượng được trình bày như sau:**



  - **Hai đối tượng khác nhau có thể có các giá trị thuộc tính giống nhau (như hai người với tên và địa chỉ giống nhau)**
- **Đối tượng có quan hệ kết hợp (associations) với đối tượng khác**
  - **E.g. Nam :employee thì có quan hệ kết hợp với đối tượng Mekong1000 :project**
  - **Nhưng chúng ta sẽ tìm hiểu quan hệ này ở cấp độ *lớp !***
  - **Chú ý : Phải bảo đảm rằng thuộc tính thì kết hợp với đúng lớp**

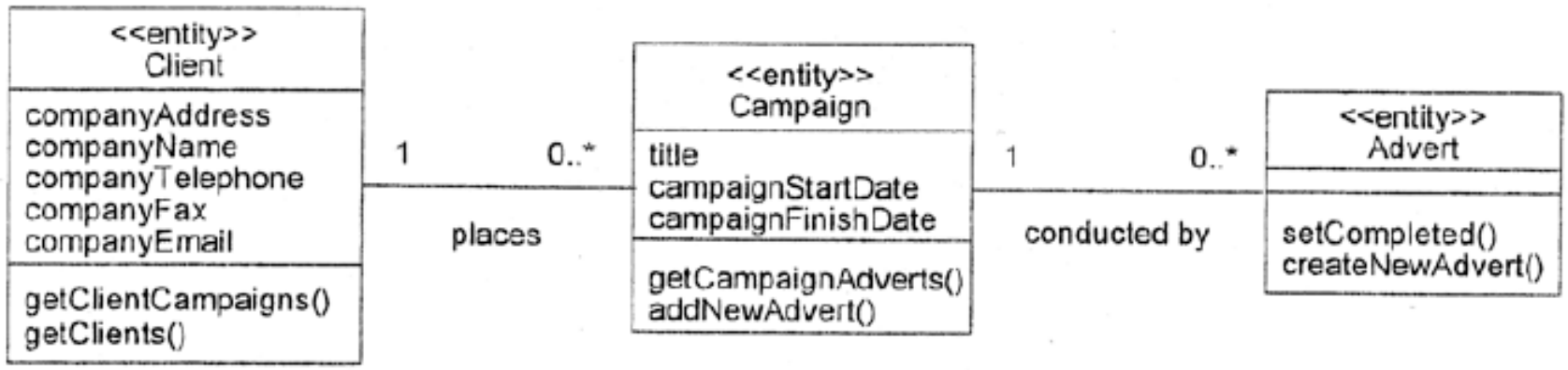
➢**E.g. bạn không muốn cả managerName và manager# đều là thuộc tính của Project !?**

Quan hệ kết hợp (Associations)

Đối tượng không tồn tại độc lập với những cái khác

- Một quan hệ (relationship) diễn tả một sự kết hợp trong số những cái khác.
- Trong UML, có nhiều kiểu quan hệ khác nhau:
  - Quan hệ kết hợp (Association)
  - Quan hệ tập hợp (Aggregation) và Quan hệ hợp thành (Composition)
  - Quan hệ thừa kế (Generalization)
  - Quan hệ phụ thuộc (Dependency)
  - Quan hệ hiện thực hóa (Realization)
- Chú ý : Hai quan hệ cuối không hữu ích trong quá trình phân tích yêu cầu

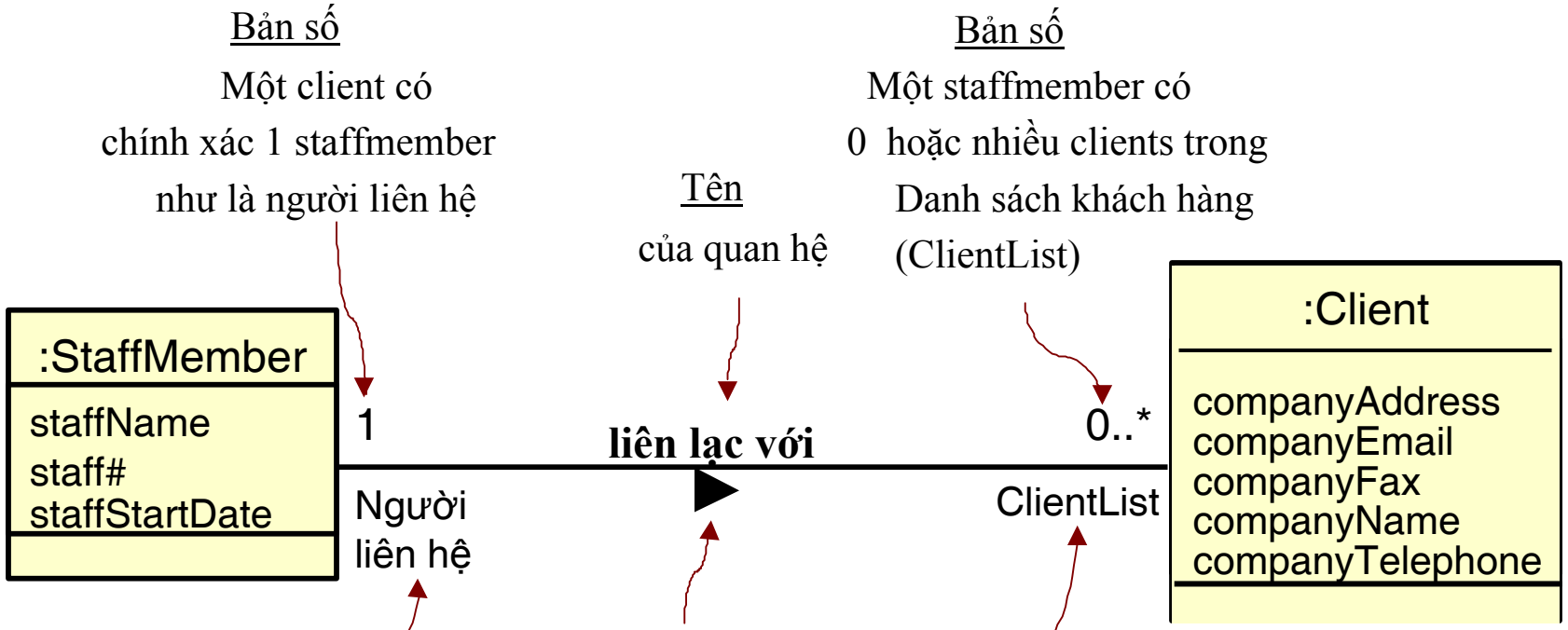
Sơ đồ lớp chỉ rõ các lớp và mối quan hệ giữa chúng



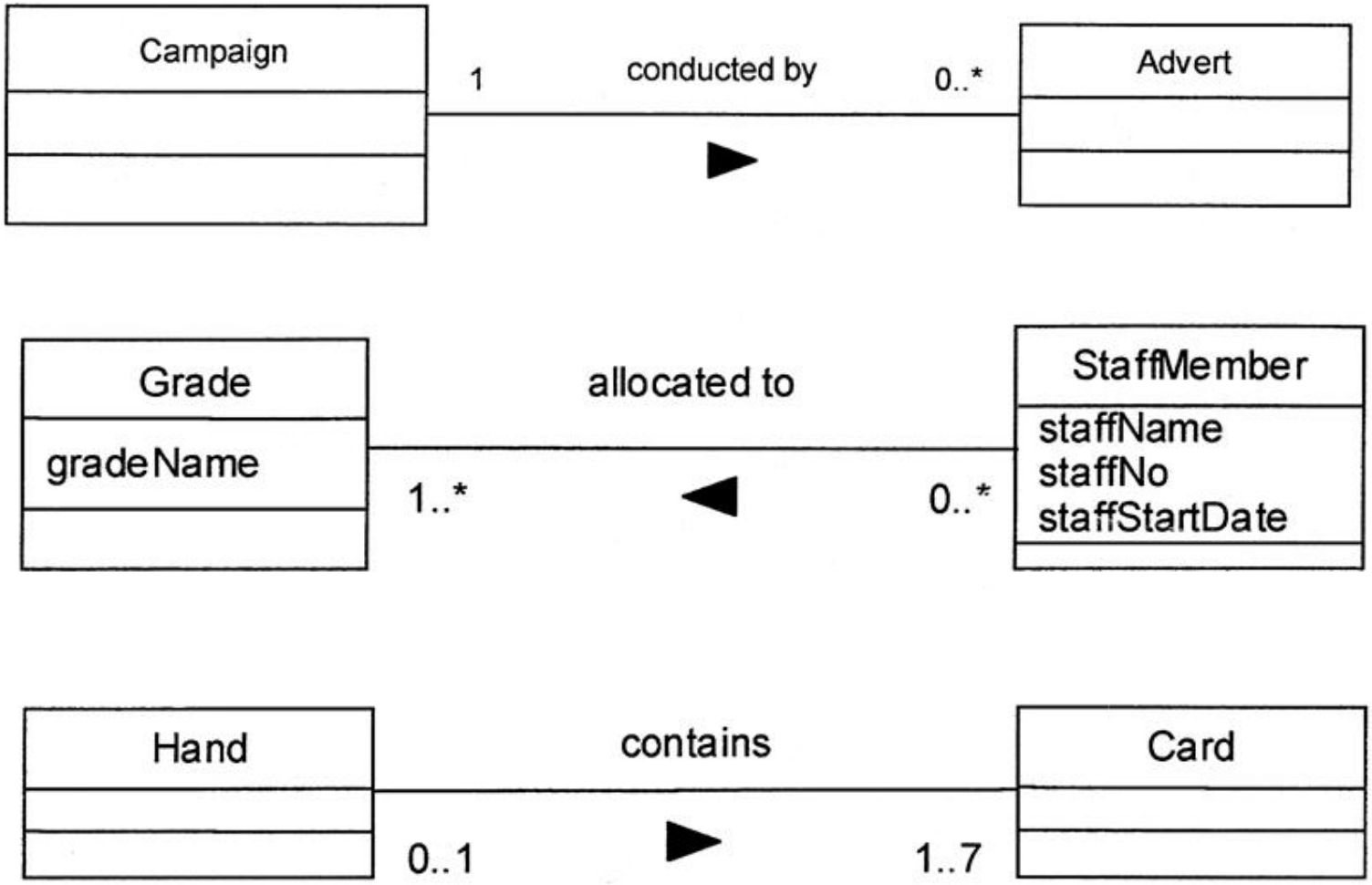
Bản số (Multiplicity) của Quan hệ kết hợp

- **Hỏi các câu hỏi về quan hệ kết hợp:**
  - **Một cuộc vận động (campaign) có thể tồn tại mà không có thành viên trong Ban quản lý hay không ?**
    - **Nếu có, thì quan hệ kết hợp này sẽ là một tùy chọn trong Ban quản lý - 0 hoặc nhiều (0..*)**
    - **Nếu không, thì nó là tùy chọn – 1 hoặc nhiều (1..*)**
    - **Nếu nó cần được quản lý bởi 1 và chỉ 1 thành viên trong Ban – chính xác 1 (1)**
  - **Một câu hỏi khác của quan hệ kết hợp?**
    - **Mỗi thành viên trong Ban quản lý có cần phải quản lý chính xác chỉ một cuộc vận động không?**
    - **Không. Vì thế bản số chính xác là 0 hoặc nhiều (0..*)**
- **Một số ví dụ biểu diễn của bản số:**
  - **Tùy chọn (0 hoặc 1)** **0..1**
  - **Chính xác 1** **1** **= 1..1**
  - **0 hoặc nhiều** **0..*** **= ***
  - **1 hoặc nhiều** **1..***
  - **Một vùng giá trị** **2..6**

Quan hệ kết hợp lớp

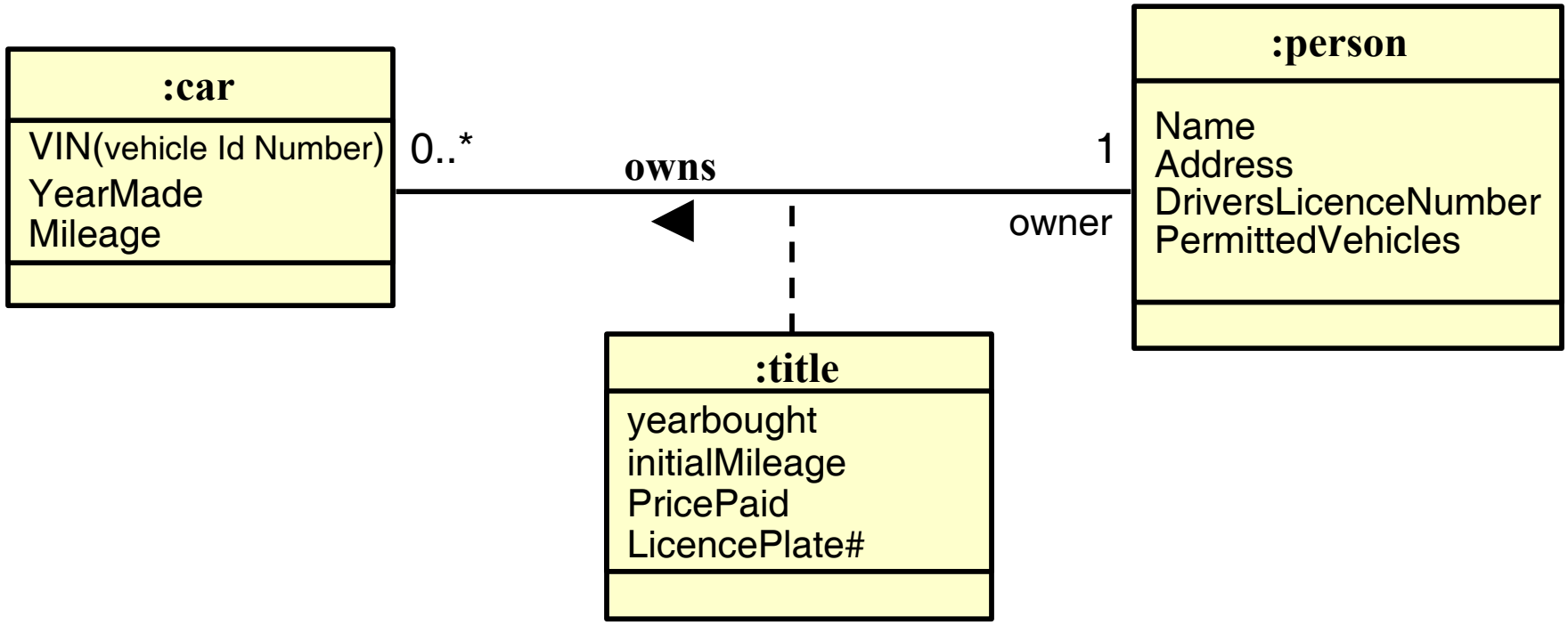


Các ví dụ khác



Các lớp quan hệ kết hợp

- **Đôi khi có quan hệ kết hợp của một lớp với chính quan hệ**
  - **… bởi vì chúng ta cần giữ lại thông tin về quan hệ kết hợp**
  - **… và những thông tin này thì không còn tồn tại trong các lớp vào cuối của quan hệ kết hợp**
    - **E.g. "Chủ quyền" (title) là một đối tượng dùng mô tả thông tin về mối quan hệ giữa người chủ và chiếc xe của cô ấy**



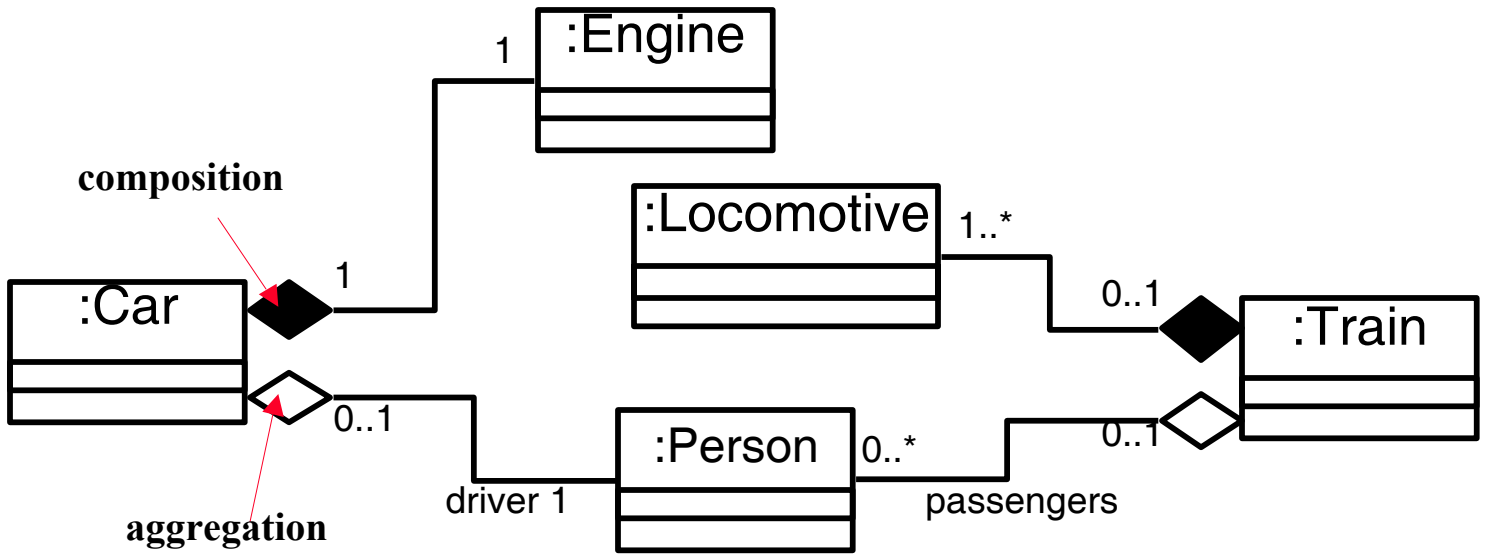
Quan hệ tập hợp và Quan hệ hợp thành

Quan hệ tập hợp (Aggregation)

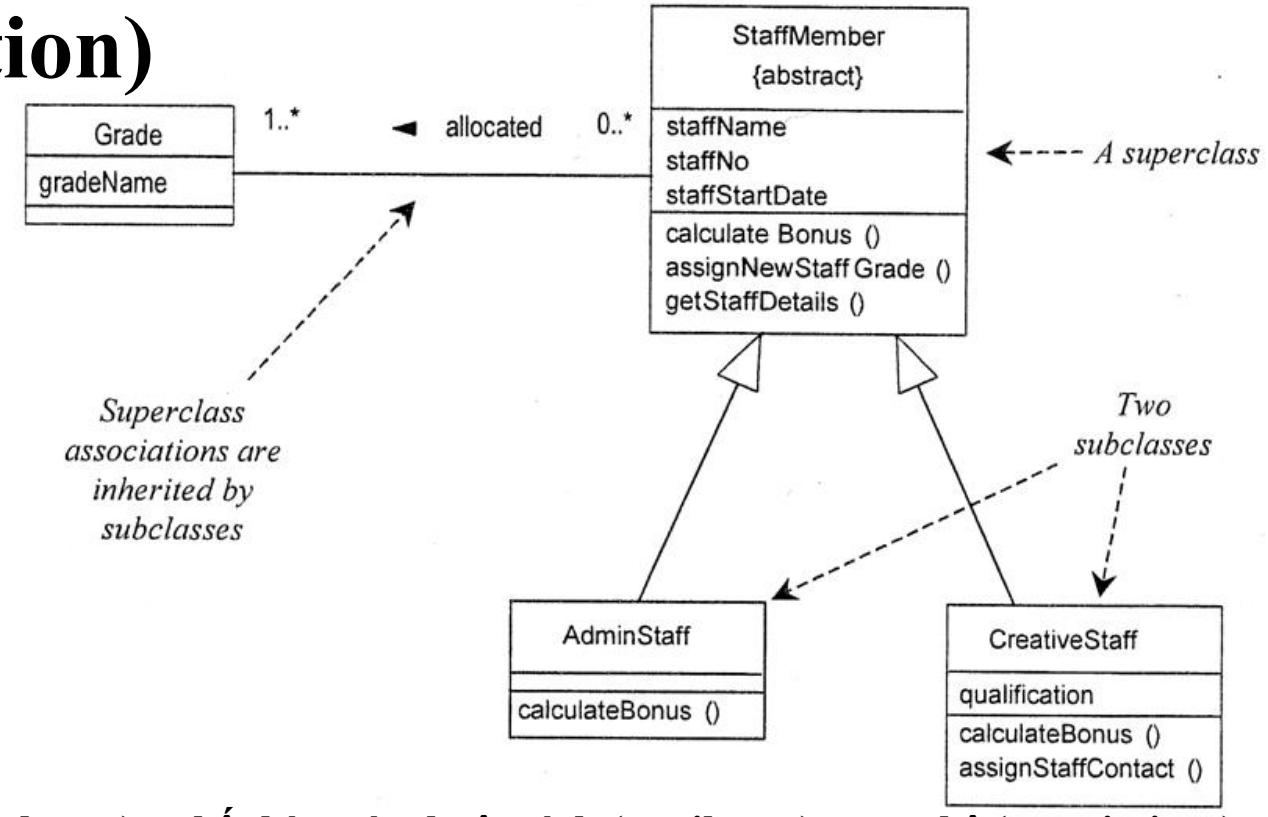
- Đây là một quan hệ "Có một (Has-a)" hay "Toàn thể/Bộ phận (Whole/part)"

Quan hệ hợp thành (Composition)

- Là dạng mạnh hơn của quan hệ tập hợp dùng chứng tỏ quyền sở hữu:
  - nếu đối tượng toàn thể bị hủy, đối tượng bộ phận cũng bị hủy theo.
  - đối tượng toàn thể chịu trách nhiệm về sự sắp xếp các thành phần của nó.



Quan hệ kế thừa (Generalization)



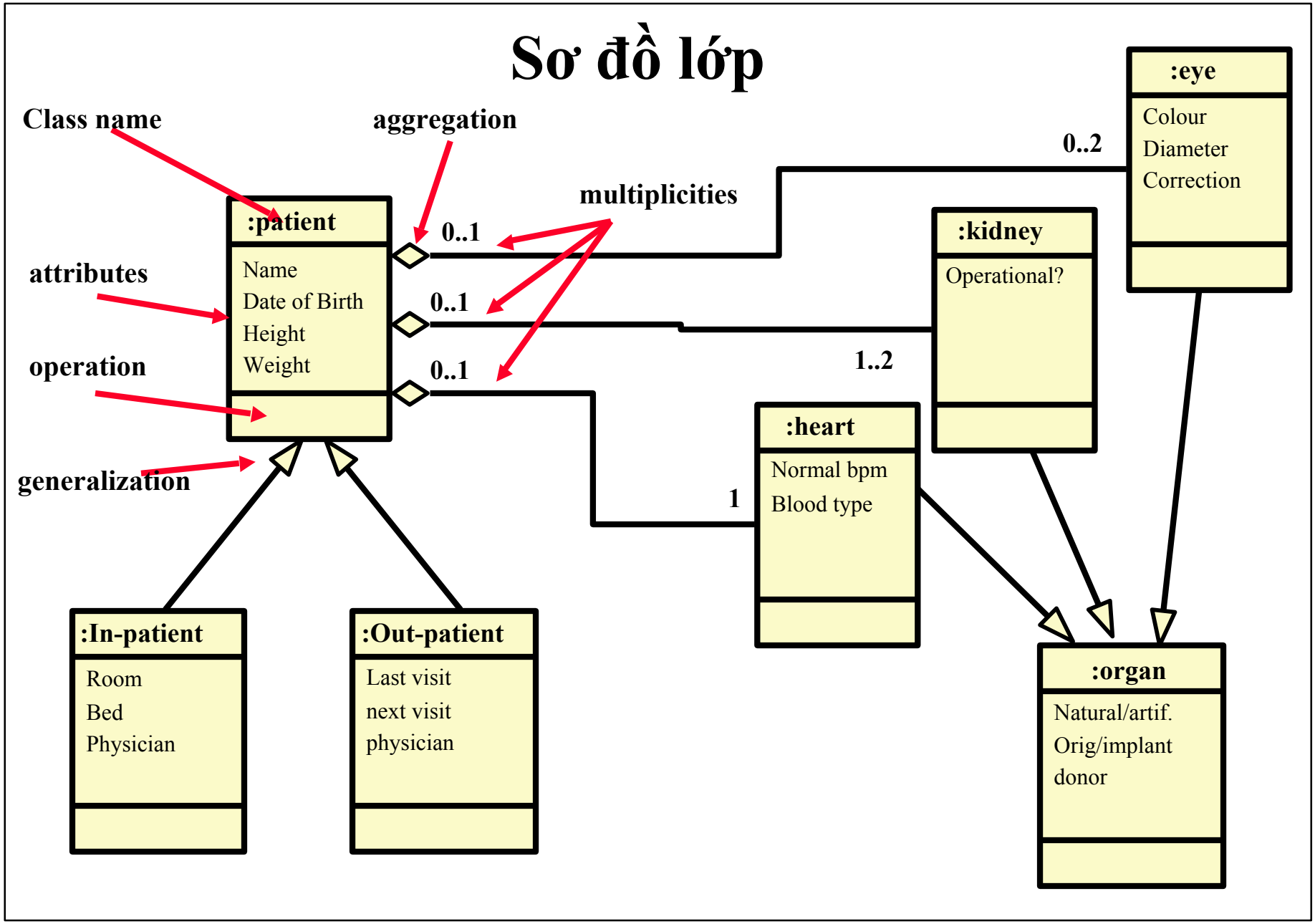
Chú ý :

- Các lớp con (subclasses) sẽ kế thừa các thuộc tính (attributes), quan hệ (associations) và phương thức (operations) từ lớp cha (superclass)
- Một lớp con có thể bỏ qua một khía cạnh kế thừa
  - e.g. AdminStaff & CreativeStaff có các phương pháp khác nhau cho cách tính điểm thưởng
- Các lớp cha có thể khai báo {abstract}, nghĩa là chúng không có thể hiện (instances)
  - Chứng tỏ rằng các lớp con bao phủ tất cả
  - e.g. không có bộ phận nào khác hơn AdminStaff và CreativeStaff

Nói thêm về Quan hệ kế thừa

- **Ích lợi của quan hệ kế thừa**
  - Có thể dễ dàng thêm vào các lớp con mới khi có thay đổi tổ chức
- **Tìm kiếm quan hệ kế thừa theo 2 cách:**
  - Trên xuống (Top Down)
    - Bạn có một lớp, và việc khảo sát nó có thể được chia nhỏ ra
    - Hoặc bạn có một quan hệ kết hợp diễn tả một "loại của (kind of)" quan hệ
    - E.g. "Hầu hết công việc của chúng ta là quảng cáo cho báo chí – các tờ báo và tạp chí, cũng như là trên các pano quảng cáo và videos"
  - Dưới lên (Bottom Up)
    - Bạn cần lưu ý sự tương tự giữa các lớp mà bạn định nghĩa
    - E.g. "Chúng ta có nhiều sách và dĩa CD trong Thư viện, nhưng tất cả chúng đã được ghi số phân loại theo hệ thống Dewey, và tất cả đều có thể được cho mượn và dặn trước"
- **Nhưng đừng kế thừa chỉ những ích lợi của nó**
  - Bảo đảm rằng mọi thứ trong lớp cha được áp dụng vào lớp con
  - Bảo đảm rằng lớp cha thì hữu ích khi một lớp trong nó được sở hữu hoàn toàn
  - Đừng thêm các lớp con hoặc lớp cha không có liên quan vào phân tích của bạn

Sơ đồ lớp


Class name
aggregation
multiplicities
attributes
operation
generalization
:patient
Name
Date of Birth
Height
Weight
0..1
0..1
0..1
0..2
:eye
Colour
Diameter
Correction
:kidney
Operational?
1..2
:heart
Normal bpm
Blood type
1
:In-patient
Room
Bed
Physician
:Out-patient
Last visit
next visit
physician
:organ
Natural/artif.
Orig/implant
donor

Kết luận

- **Hiểu rõ các đối tượng trong lĩnh vực ứng dụng**
  - **Định nghĩa tất cả các đối tượng mà đối tác nêu ra**
  - **Quyết định đối tượng nào quan trọng cho thiết kế của bạn**
  - **Sơ đồ lớp thiết kế tốt khi:**
    - **Có quan hệ trực quan giữa các đối tượng trong lĩnh vực**
    - **Khảo sát các quy tắc nghiệp vụ và giả thiết thông qua bản số**
    - **Đặc tả cấu trúc của thông tin để (cuối cùng) lưu trữ**
- **OO là một cách thức tốt để khảo sát các chi tiết của vấn đề**
  - **Tránh những chấp vá tự nhiên của cấu trúc phân tích**
  - **Cung cấp một phương thức chặt chẽ để hiểu rõ thực tế**
- **Nhưng cẩn thận…**
  - **Nó lôi cuốn để thiết kế hơn là phân tích vấn đề**
    - **Trong RE, sơ đồ lớp KHÔNG phản ánh các lớp chương trình (e.g. Java)**
  - **Đối với các nhà phân tích, dùng các lược đồ UML như là sự phác họa chứ không phải bản thiết kế**
    - **Tuy nhiên vẫn có thể trở thành bản thiết kế khi được dùng trong một sự đặc tả**

Kết luận: UML vs ERD

Sơ đồ ER tương tự như sơ đồ lớp trong UML

- Sơ đồ lớp nhấn mạnh thứ bậc lớp (class hierarchies) và các phương thức (operaions)
- Sơ đồ R nhấn mạnh các mối quan hệ (relationships) và khóa xác định (identity)

Nhưng chỉ cần một cho việc phân tích mọi vấn đề cho trước !

ER cung cấp nhiều ký hiệu hơn cho khái niệm cơ sở dữ liệu:

- Sơ đồ ER cho phép các quan hệ đa chiều (N-ary relationships)
  - (Sơ đồ lớp UML chỉ cho phép quan hệ hai chiều (binary relationships))
- Sơ đồ ER cho phép các thuộc tính đa giá trị.
- Sơ đồ ER cho phép đặc tả các khóa xác định.

Sự lựa chọn tùy thuộc vào mục tiêu cài đặt:

- Sơ đồ lớp UML cho kiến trúc hướng đối tượng (Object Oriented Architecture)
- Sơ đồ ER cho CSDL quan hệ (Relational Databases)
- Nhưng điều này chỉ quan trọng khi bạn dùng chúng cho bản thiết kế
  - Đối với bản phác thảo, sự tương tự với ký hiệu thì quan trọng hơn