OOAD

Thiết kế hệ thống
(Phần I)

Nguyễn Anh Hào
0913609730 – nahao@ptithcm.edu.vn

Thiết kế hệ thống

- Thiết kế hệ thống là quá trình xác định kiến trúc, thành phần, giao diện và dữ liệu của hệ thống để thể hiện giải pháp khả thi sẵn sàng triển khai.
  - Thiết kế: ý tưởng của giải pháp.
- Thiết kế hướng đối tượng là xây dựng hệ thống bằng cách sử dụng khái niệm & nguyên lý hướng đối tượng
  - Tất cả các vấn đề trong thực tế đều cần có giải pháp từ (tình huống) thực tế đó.
  - Mô phỏng thực tế là một cách thiết kế tốt để giải quyết các vấn đề trong thế giới thực.

2015_Object-Oriented Analysis, Design and Implementation,2nd edition, Brahma Dathan&Sarnath Ramnath,Springer 2015 (chp 7, Page 159)

Component, Package, SubSystem

- **Component** (thành phần)
  - Mỗi thành phần của hệ thống, theo nghĩa thiết kế, là đơn vị nhỏ nhất (lớp đối tượng) của hệ thống.
  - Mỗi thành phần chỉ giải quyết một vấn đề cơ bản, phổ biến và thực tế

- **Package** (gói - thư viện mã được viết sẵn)
  - Là một file/thư mục chứa các thành phần đã được cài đặt
  - Gói giải quyết các vấn đề phổ biến và tạo điều kiện tái sử dụng mã của các thành phần đã được thiết kế và cài đặt.

- **SubSystem** (hệ thống con)
  - Là một tập hợp các thành phần có cộng tác nhau
  - SubSystem giải quyết vấn đề của một usecase

Mục tiêu của thiết kế hệ thống

Một thiết kế tốt là thiết kế giảm thiểu tổng chi phí khởi tạo, sử dụng và nâng cấp hệ thống trong suốt vòng đời của nó.

1 Mỗi thành phần của hệ thống đều dễ hiểu
  - Không cần tham khảo tài liệu bổ sung
  - Cần cân bằng giữa khả năng hiểu và hiệu quả.

2 Các thành phần (hoặc gói) đều dễ xây dựng

3 Việc sửa đổi chỉ cần thực hiện trong phạm vi nhỏ
  - Để giảm thiểu phát sinh lỗi từ các thành phần bị sửa sang các thành phần khác.

4 Các thành phần sau khi được tạo ra có thể tái sử dụng.

1) Coupling & Cohession

- **Coupling**: mức độ phụ thuộc lẫn nhau giữa các thành phần khác nhau trong một hệ thống.
  - Sự phụ thuộc lẫn nhau càng cao thì một thay đổi ở một thành phần sẽ dẫn đến nhiều phần khác phải thay đổi theo.

- **Cohesion**: mức độ gắn kết của các phần tử trong một thành phần, và chúng hoạt động cùng nhau như một thể thống nhất cho một mục đích.
  - Làm giảm bớt nguy cơ thay đổi trên các phần tử sau khi đã thiết kế

Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Design criteria (p.286)

Coupling: Encapsulation

1 Nếu một phương thức có tham chiếu đến một thuộc tính trong lớp của nó, nó sẽ bị phụ thuộc vào thuộc tính đó.

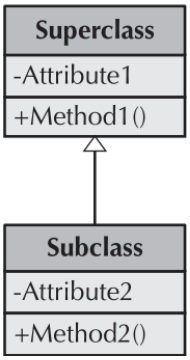
2 Nếu một lớp có thuộc tính kiểu A, nó sẽ bị phụ thuộc vào kiểu A của thuộc tính đó.

3 Một lớp có thuộc tính mà miền giá trị có ý nghĩa: Nếu miền giá trị này bị thay đổi, thì mọi phương thức sử dụng thuộc tính đó sẽ phải được sửa đổi.

- Ví dụ: Giới tính = {nam, nữ} → Giới tính {nam, nữ, song tính} thì dịch vụ xếp phòng phải thay đổi.

Coupling: Inheritance

- Khi lớp con B kế thừa từ lớp A: Nếu B sử dụng nội dung kế thừa từ A, thì B phụ thuộc vào nội dung kế thừa này.
  - *Một phương thức được định nghĩa trong một lớp con có cần gọi một phương thức của lớp cha không?*
  - *Một phương thức được định nghĩa trong một lớp con có cần tham chiếu đến một thuộc tính của lớp cha không?*



Coupling: Interaction

Coupling Interaction là phụ thuộc giữa phương thức và đối tượng trong thông điệp.

1 Phụ thuộc bản thân (một đối tượng gửi thông điệp cho chính nó): Phương thức gọi phụ thuộc vào phương thức được gọi trong lớp của nó.

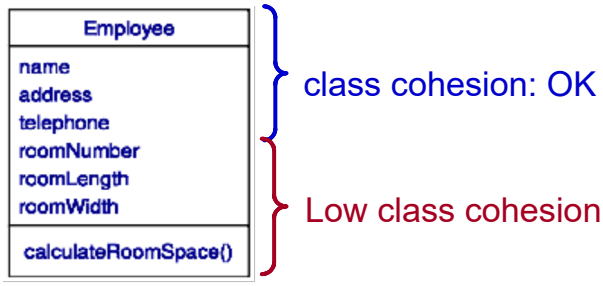
2 Khi lớp A có một thuộc tính là một đối tượng thuộc lớp B: A phụ thuộc vào đối tượng đó vì bất kỳ thay đổi nào đối với public interface của lớp B sẽ ảnh hưởng đến cách mà lớp A sử dụng thuộc tính (đối tượng) đó.

3 Một đối tượng được truyền dưới dạng tham số cho phương thức: phương thức sử dụng tham số đó sẽ phụ thuộc vào đối tượng.

4 Một đối tượng được tạo ra bởi một phương thức: Phương thức gọi phụ thuộc vào đối tượng được trả về.

Cohesion: Class

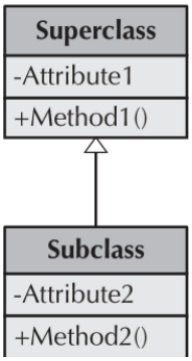
1 Một lớp chỉ nên đại diện cho **một** thứ gì đó đã biết rõ.
2 Tất cả các thuộc tính và phương thức của một lớp phải diễn tả được (vai trò, nhiệm vụ, khả năng) của lớp.
3 Không nên có các thuộc tính hoặc phương thức dư thừa
4 Sự gắn kết của một lớp là mức độ gắn kết giữa các thuộc tính và phương thức của lớp đó



Cohesion: Inheritance

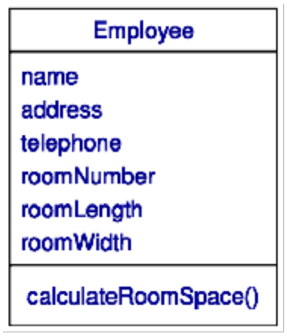
Liskov Substitution : Tất cả các lớp dẫn xuất đều **phải thực sự cần thiết** để làm tròn vai trò/trách nhiệm của lớp cơ sở.

Các lớp trong quan hệ thừa kế có liên quan như thế nào? Chúng có liên quan chặt chẽ với nhau qua ý nghĩa của quan hệ tổng quát hóa / chuyên môn hóa không ?



Cohesion: Method

1 Một phương thức chỉ dùng để giải quyết một nhiệm vụ.
  – Một phương pháp thực hiện nhiều chức năng sẽ khó hiểu và khó tái sử dụng hơn.
2 Mọi thành phần của một phương thức (dữ liệu, mã) đều cần thiết cho phương thức đó.



2) 5 nguyên tắc đầu tiên của OOAD

1. **S**ingle Responsibility: Một lớp chỉ nên có một và chỉ một lý do để thay đổi, ie, một lớp chỉ nên có một nhiệm vụ.
2. **O**pen/Closed: Một lớp nên mở cho yêu cầu thêm mới, nhưng đóng đối với yêu cầu sửa đổi.
3. **L**iskov Substitution : mọi lớp dẫn xuất đều có thể thay thế cho lớp cơ sở của chúng.
4. **I**nterface Segregation: Không nên buộc 'client' phải phụ thuộc vào dữ liệu/chức năng mà nó không sử dụng (tránh **giao diện dư thừa dữ liệu & chức năng**).
5. **D**ependency Inversion: Mô-đun cấp cao không nên phụ thuộc vào mô-đun cấp thấp.

Dependency Inversion: Bad example

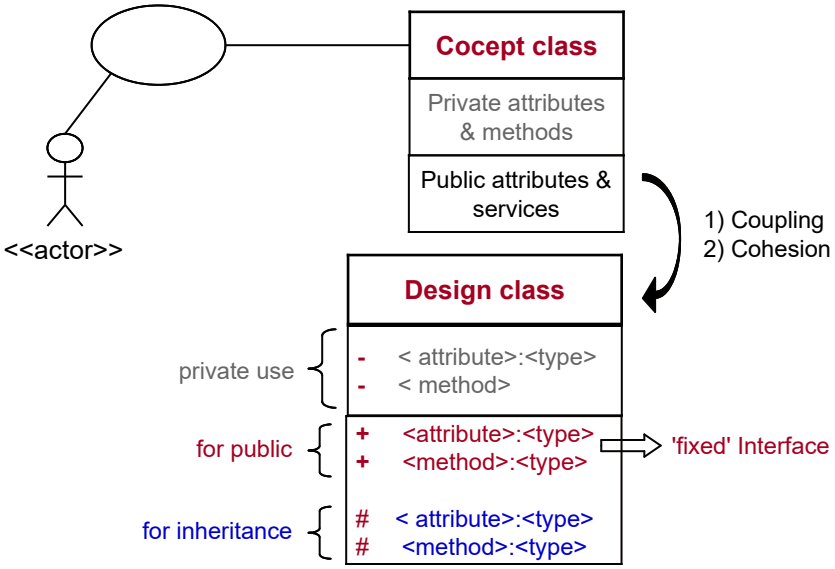

```
class Manager {
        Worker worker;
        public void setWorker( Worker w) { //or SuperWorker. How?
                worker = w;
        }
        public void manage() {   worker.work();
        }
  }
class Worker {
        public void work() {   // ....working
        }
  }
class SuperWorker {       // ADDED.
        public void work() {  //.... working much more
        }
  }
```

Dependency Inversion: Good example

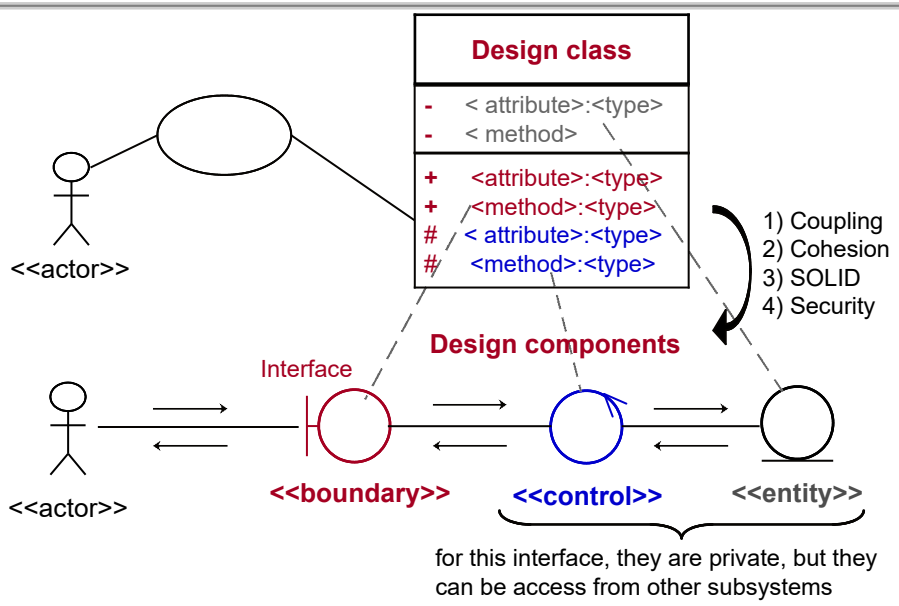

```
class Manager {
        IWorker worker;
        public void setWorker( IWorker w) {     worker = w;   }
        public void manage() {    worker.work();   }
}

interface  IWorker {                    // Abstract interface
        public void work();
}
class Worker implements IWorker{
        public void work() { … }      // Concrete class 1
}
class SuperWorker implements IWorker{
        public void work() { … }      // Concrete class 2
}
```

3) Từ phân tích đến thiết kế



Từ phân tích đến thiết kế



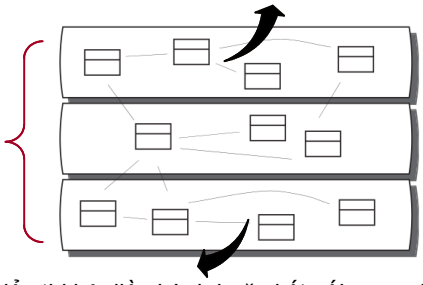
a) LAYERING

Mỗi lớp là một nhóm (cluster) các đối tượng cộng tác, phụ thuộc vào các tiện ích từ các lớp thấp hơn cung cấp.

- Giúp giảm độ phức tạp
- Tăng khả năng tái sử dụng

Lớp trên cùng biểu thị giao diện người dùng (menu, hộp thoại, khả năng sử dụng, trực quan,..)

Các lớp khác nhau có trọng tâm khác nhau.



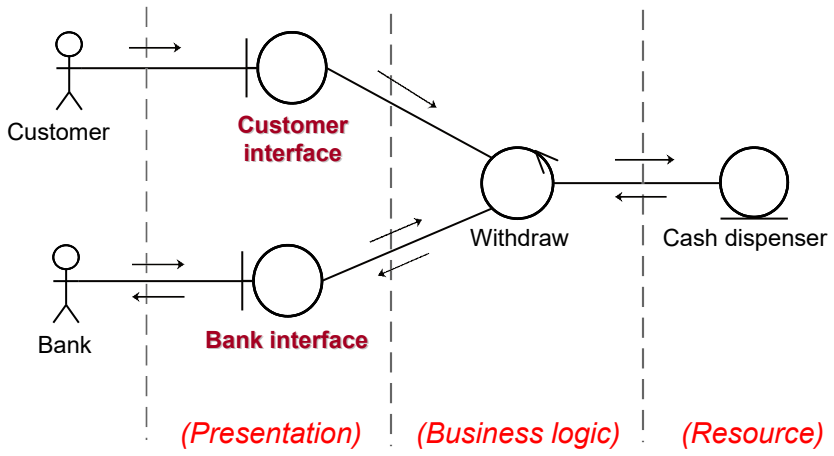
Lớp giữa: trừu tượng hóa, thuộc tính, cơ sở dữ liệu, thao tác, đa hình, tái sử dụng...

Lớp dưới cùng biểu thị hệ điều hành hoặc kết nối mạng (giao thức, băng thông và các loại máy chủ khác nhau...)

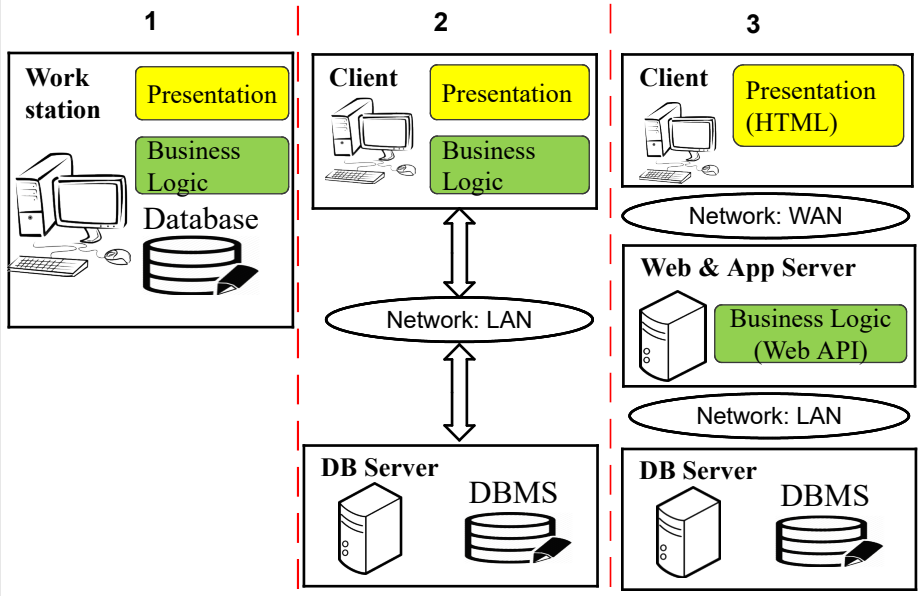
Object Oriented Analysis And Design, Mike O'Docherty, Wiley 2005: Layer (P248)

Layering example

ATM: withdraw money usecase



"Thinner client, thicker server"



b) PARTITIONING

- Phân cụm: nhóm các môđun/lớp vào một cụm để **giảm thiểu phụ thuộc giữa các cụm**
- Phân cụm dựa trên sự hợp tác được mô hình hóa trong lược giao tiếp UML cho các usecase
- Cần xem xét lược đồ lớp để thấy các lớp có mối quan hệ như thế nào trong sự cộng tác.
  - Sự kết hợp giữa các lớp càng lớn thì khả năng các lớp này được nhóm lại thành một cụm càng cao.
  - Việc kết hợp lược đồ lớp với lược đồ giao tiếp có thể rất hữu ích để xem xét mức độ phụ thuộc giữa các lớp để đưa vào 1 cụm.

Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Partitions and Collaborations (p.258)

c) FACTORING (phân tích nhân tử)

- FACTORING là quá trình tách một mô-đun thành các mô-đun độc lập để dùng chung ("thừa số chung").
  - Chúng ta có thể nhận ra rằng một số lớp thiết kế có một số thuộc tính và phương thức chung
  - Khi đó, những điểm giống nhau giữa các lớp sẽ được tách ra thành một lớp mới được dùng chung
  - Lớp mới có thể liên quan đến các lớp hiện có qua mối quan hệ tổng quát (A-Kind-Of) hoặc kết tập (Has-Parts).

- *Abstraction & Refinement* là hai quá trình có liên quan chặt chẽ đến phân tích nhân tử.

Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Factoring (p.257)

Factoring: Abstraction & Refinement

- ***Abstraction*** liên quan đến việc tạo ra một ý niệm "cấp cao hơn" từ một tập hợp các ý niệm.
  - Việc xác định lớp Nhân viên là một ví dụ về việc trừu tượng hóa từ một tập hợp các lớp chi tiết: Y tá , Nhân viên hành chính , Bác sĩ ,..
  - Quá trình trừu tượng hóa sẽ xác định các lớp trừu tượng

- ***Refinement*** là quá trình ngược lại với quá trình trừu tượng, tạo ra các lớp chi tiết.
  - Xác định thêm các lớp con của lớp Nhân viên hành chính , chẳng hạn như Nhân viên lễ tân , Thư ký ,...
  - Chỉ thêm các lớp con mới nếu có đủ sự khác biệt giữa chúng.

d) Security consideration

1 **Privacy (quyền riêng tư)** : Ẩn thông tin, chỉ cho phép những người được phép đọc hoặc thay đổi thông tin của họ.

2 **Authentication (xác thực)** : Cần biết thông tin đến từ đâu để quyết định có nên tin tưởng thông tin đó hay không.

3 **Irrefutability (tính không thể chối cãi)** : Đây là mặt trái của tính xác thực, đảm bảo rằng người tạo ra thông tin không thể phủ nhận họ là nguồn.

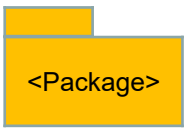
4 **Integrity (tính toàn vẹn)**: Đảm bảo thông tin không bị hư hại (vô tình hoặc cố ý) trên đường truyền từ nguồn đến đích .

5 **Safety (an toàn)** : Kiểm soát quyền truy cập vào tài nguyên (như máy móc, quy trình, cơ sở dữ liệu và tệp). An toàn còn được gọi là ủy quyền.

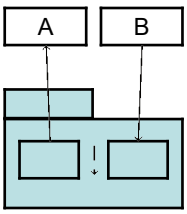
Object Oriented Analysis And Design,...:8.6 DESIGNING FOR SECURITY (P222)

4) Design Packages

1 Thiết kế các gói: sắp xếp các lớp (mã, dữ liệu) thành từng gói để dễ phát triển, tái sử dụng và bảo trì, bằng cách nhóm các thành phần thiết kế cơ bản (lớp, giao diện) thành một cấu trúc vật lý (“mô-đun”).

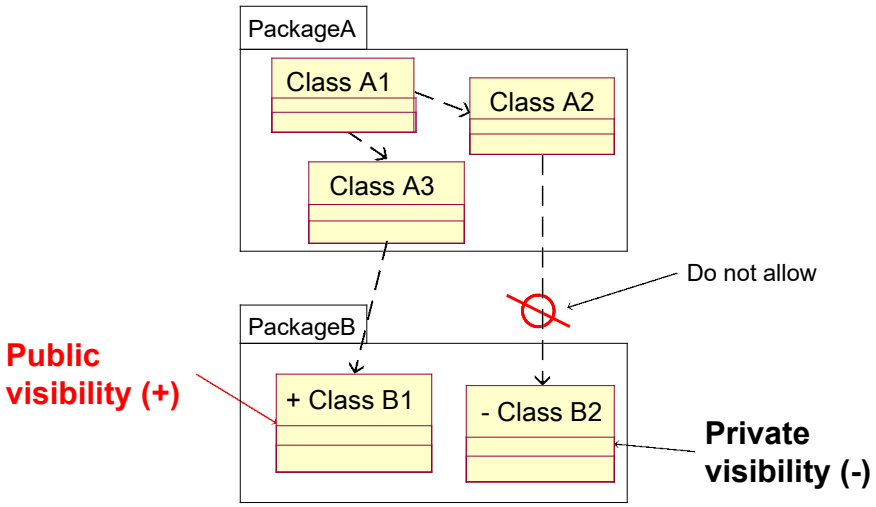


2 Nội dung của một gói có thể là:
- Một thư viện gồm các thành phần có thể tái sử dụng
- Hoặc các lớp cần được cài đặt cùng nhau (chúng phụ thuộc vào nhau)

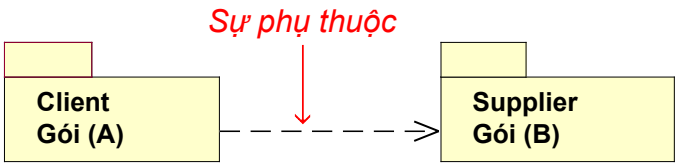


Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Package and package diagram (p.262)

Package interface: Visibility & Access



Package dependency



1 Gói A phụ thuộc vào gói B nếu có một lớp trong A phụ thuộc vào một lớp trong B.

2 Những hậu quả không thể tránh khỏi là:
- Những thay đổi đối với Gói B có thể ảnh hưởng đến Gói A
- Gói A không thể được cài đặt hoặc sử dụng riêng biệt mà không có gói B

Hướng dẫn tạo lược đồ package

1. Xác định bối cảnh cho lược đồ là miền vấn đề liên kết với một usecase.

2. Nhóm các lớp lại với nhau khi có mối quan hệ kế thừa, tổng hợp hoặc hợp thành giữa chúng, hoặc khi các lớp này tạo thành sự cộng tác.
   - Các lớp càng phụ thuộc vào nhau thì khả năng chúng nằm trong cùng một gói càng cao.
   - Hướng của sự phụ thuộc thường là từ lớp con đến lớp cha, từ kết tập đến thành phần và từ máy khách đến máy chủ.
   - Mỗi nhóm là một gói.

3. Xác định mối quan hệ phụ thuộc giữa các gói
   - Các gói không nên phụ thuộc chéo (cross coupled)