# Slide 1: Giới thiệu về mô hình hoá:

## Học mô hình hoá để làm gì ?

* + Biết các mô hình và tiến trình là gì.
  + Biết các công nghệ áp dụng vào trong các hệ thống lớn thông qua các mô hình.
  + Biết vẽ các diagram tạo ra mô hình dùng để diễn tả các hệ thống VD: Quá trình kinh doanh, quá trình phát triển phần mềm..
  + Biết áp dụng mô hình hoá để biết viết đọc và sử dụng mô hình thường dùng trong công nghệ phần mềm.

## Khái niệm mô hình hoá:

* + Mô hình hoá là biểu diễn những gì có trong thế giới thực hoặc tưởng tượng thành mô hình. Nó là trung tâm của mọi hoạt động xây dựng và tạo tác trong thực tế.

## Ý nghĩa mô hình hóa :

* Tất cả các phương pháp, kỹ thuật và lý thuyết đều có thể được mô hình hoá các dạng khác nhau.
* Bản thân lý thuyết mô hình hoá được tạo bởi triết học, lý thuyết hệ thống và cả những sự mường tượng thông minh.
* Mô hình hoá là sự cần thiết và không thể tách rời trong hoạt động khoa học.
* Mô hình hoá sẽ tạo các kỹ năng và kỹ thuật nhằm tạo ra các kết quả sâu sắc, đáng tin cậy, và dễ sử dụng (đây cũng là một nguyên nhân và mục đích của mô hình hoá).
* Mô hình hoá cần phải được xác nhận nếu như mô hình phản ánh thực tế, và nhận dạng sự thoả thuận khác biệt giữa lý thuyết và dữ liệu.

## Mô hình là gì?

* + Mô hình là một cách thể hiện cái nhìn cụ thể của một hệ thống. Là một phương tiện để hiểu các vấn đề liên quan trong việc xây dựng một cái gì đó. Nó hỗ trợ cho việc giao tiếp giữa những người tham gia vào dự án, đặc biệt là giữa người phân tích yêu cầu và người dùng. Là một thành phần của các phương pháp được sử dụng trong hoạt động phát triển (phân tích yêu cầu và thiết kế).

## Tính chất của mô hình:

* + Mô hình là sự trừu tượng hoá: Tập trung vào những phần cơ bản của vấn đề phức tạp, bỏ qua những chi tiết không quan trọng.
  + Mô hình là sự đơn giản hoá: Chỉ những thuộc tính liên quan mới được biểu diễn.

## Mô hình trong nghiên cứu:

* + Các mô hình chỉ biểu diễn gần đúng vì do trong thực tế thì sẽ có những sự phức tạp làm giảm tính chính xác của sự biểu diển. Nên người nghiên cứu phải tìm các nào đó để có được sự cân bằng giữa tính chính xác và sai số đó. Trong một số trường hợp phải chấp nhận sai sót. Mỗi mô hình khác nhau thể hiện một góc nhìn khác nhau về thế giới thực.

## Nguyên tắc để xây dựng một mô hình

* + Không quá phức tạp để người vẽ và các thành viên trong nhóm có thể sử dụng.
  + Đơn giản hoá các giả định và các hệ quả đều được trình bày rõ ràng.
  + Tất cả giả định đều phải chính đáng. Quá nhiều chi tiết khiến mất tập trung và khó sử dụng để hỗ trợ giao tiếp.
  + Mô hình có tính thay đổi. Phải có kiểm thử để có thể bảo trì và kiểm chứng với thực tế.

## Các loại mô hình:

* + Mô hình tĩnh: Miêu tả những phần tử và mối quan hệ tồn tại giữa chúng.
  + Mô hình động: Miêu tả hành vi của một hoặc nhiều phần tử qua thời gian.

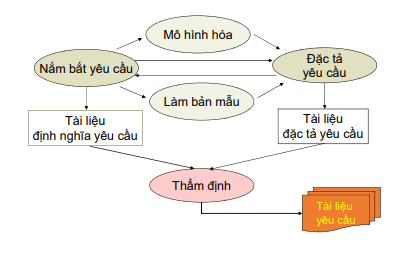
## Mô hình trong phát triển phần mềm:

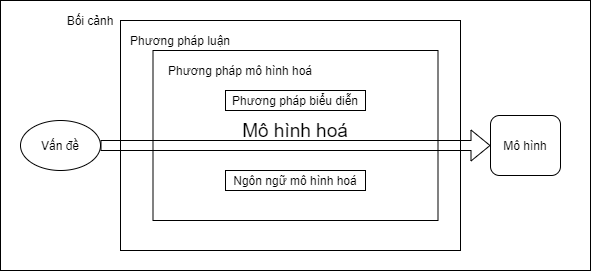
* + Mô hình liên quan đến máy móc (phần mềm làm gì khi nó được thực thi trên máy tính), mô phỏng những thứ có thật.
  + Diễn tả một domain (Một phần của thế giới thực mà bạn có hứng thú)
  + Mô hình và domain khác nhau nên chú ý đến sự khác nhau giữa những thứ đúng trên cả máy móc và domain, những chỉ đúng trên domain và những thứ chỉ đúng trên máy móc.

## Ứng dụng của mô hình hoá:

* + Trong thời tiết: Có thể tạo mô hình thời tiết, mô phỏng dựa trên dữ liệu và tạo dự báo.
    - VD: Tạo một mô hình dự báo bão dựa trên sự theo dõi cường độ và các hiện tượng liên quan xảy ra.
  + Mô phỏng ảnh hưởng của các hiện tượng thời tiếc khắc nghiệt lên xây dựng cơ sở hạ tầng để thiết kế các hệ thống chống chịu tốt hơn.
  + Tạo chương trình để mô phỏng tình huống xã hội và quan sát hành vi của các cá nhân. Dùng để cung cấp dự đoán về những gì xảy ra trong thế giới thực ví dụ như cách các chuẩn mực xã hội phát triển.
  + Mô phỏng để biết những ảnh hưởng về mặt vật lý sẽ ảnh hưởng đến sự hoạt động của hệ thống như thế nào.
    - Vd: NASA đã khám phá khí động lực học của máy bay không người lái với các mô hình logic mô phỏng áp suất không khí và dòng chảy xung quanh cánh quạt, thông tin đó giúp thiết kế hệ thống giảm nhiễu động, giúp xe vận hành êm hơn.

## Mô hình hoá là 1 phần của quá trình tạo ra tài liệu yêu cầu:



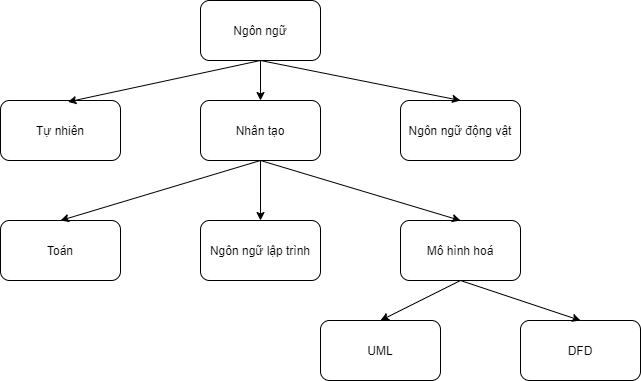


# Slide 2: Ngôn ngữ mô hình hoá

## Mô hình là gì ?

Mô hình là một cách thể hiện cái nhìn cụ thể của một hệ thống. Là một phương tiện để hiểu các vấn đề liên quan trong việc xây dựng một cái gì đó. Nó hỗ trợ cho việc giao tiếp giữa những người tham gia vào dự án, đặc biệt là giữa người phân tích yêu cầu và người dùng. Là một thành phần của các phương pháp được sử dụng trong hoạt động phát triển (phân tích yêu cầu và thiết kế).

## Các dạng ngôn ngữ



## Ngôn ngữ tự nhiên đang gây sai

Ngôn ngữ tự nhiên thì có thể gây hiểu nhầm. Mà cần thiết phải sử dụng một ngôn ngữ ít gây hiểu nhầm hoặc chỉ có một nghĩa, không thể hiểu nhầm nên người ta đã tạo ra ngôn ngữ mô hình hoá.

## Ngôn ngữ mô hình hoá là gì:

* + Tạo những mô hình thích hợp
  + Ký hiệu của ngôn ngữ mô hình hoá cho phép người phát triển tạo nên những kết nối hữu dụng giữa các mô hình.
  + Ngôn ngữ mô hình hoá được biểu diễn bằng sơ đồ.
  + Cú pháp: Một tập hợp các quy tắc xác định sơ đồ hiện tại có thể sử dụng những ký hiệu nào và những gì có thể xây dựng được bằng những ký hiệu đó.
  + Ngữ nghĩa: Xác định sơ đồ và biểu tượng đó nghĩa là gì.

## Cách lựa chọn một ngôn ngữ mô hình hoá:

* + Cung cấp đủ các cú pháp và ngữ nghĩa để có thể vẽ sơ đồ ở một tình huống cụ thể.
  + Dễ dàng sử dụng, nhưng các ký hiệu phải đủ để thể hiện những gì muốn thể hiện (có thể sẽ có hạn chế bởi vì mô hình là sự đơn giản hoá của thực tế)
  + Phải rõ ràng, để ý nghĩa của mô hình là ít nhất (chỉ có một nghĩa, một các hiểu là tốt nhất)
  + Được hỗ trợ bằng các công cụ phù hợp để có thể áp dụng kỹ năng mô hình hoá không bị phù thuộc vào kỹ năng vẽ mô hình.
  + Ngôn ngữ đó cần phải được sử dụng rộng rãi.

## Các ngôn ngữ mô hình hoá

* + Mã giả
  + Lý thuyết cấu trúc
  + DFD
  + UML
    - Class diagram
    - Use-case diagram:

## Các ngôn ngữ phổ biến:

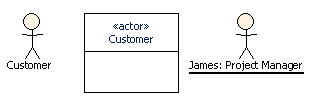
### Phương pháp Use-case

* Khái niệm :

Use Case Diagram là sơ đồ tính năng của sản phẩm cung cấp cho người dùng. Bản vẽ này sẽ cho người dùng hiểu được sản phẩm này cung cấp những tính năng gì cho người dùng, hoặc người dùng có thể làm được gì với nó.

* Ký hiệu (notation)
  + Actor

Các thực thể bên ngoài này được tham chiếu đến như các Actor. Actor giữ vai trò người dùng hệ thống, phần cứng hoặc các hệ thống khác bên ngoài tương tác với hệ thống chúng ta đang xem xét . Một Actor thường được vẽ như hình một người (hình trái), hoặc cách khác, giống như hình chữ nhật mô tả lớp với stereotype «actor» (hình giữa). Một thực thể Actor (Actor Instance) chỉ một Actor cụ thể của lớp Actor đó (hình phải).



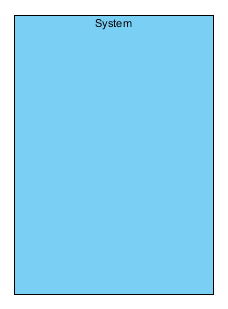
* + Use case

Là một đơn vị công việc riêng lẻ, là các chức năng của phần mềm được actor sử dụng . Nó cung cấp một cái nhìn cấp cao của người ngoài hệ thống về một hành vi có thể nhận thấy được của hệ thống. Ký hiệu mô tả một Use Case là một hình ellipse.



* + System boundary

Các Use Case thường được bao trong một khung chữ nhật biểu thị hệ thống (System Boundary). Khi đó Use Case đặt bên trong hệ thống và Actor đặt bên ngoài hệ thống.



* + Các quan hệ
    - Association: thường được dùng để mô tả mối quan hệ giữa Actor và Use Case và giữa các Use Case với nhau

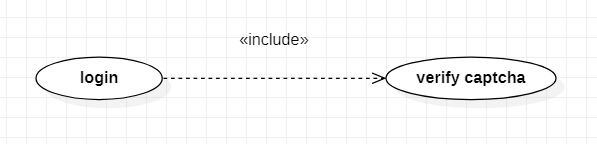


* + - Generalization

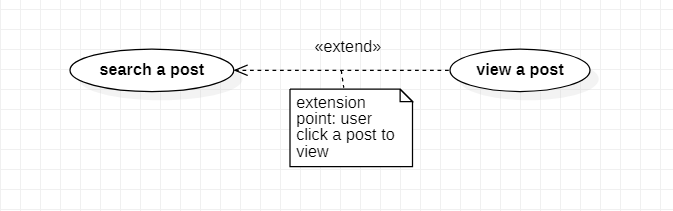
Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau.



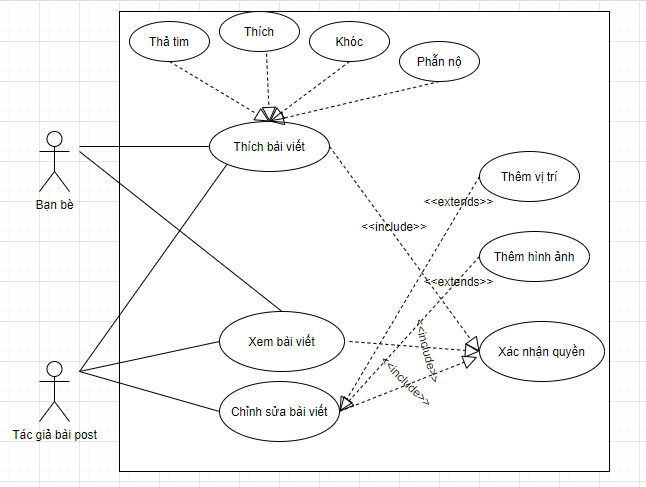
* + - Include: thường dùng giữa các Use Case để mô tả việc một Use Case lớn được chia ra thành các Use Case nhỏ để dễ cài đặt (module hóa) hoặc thể hiện sự dùng lại. Hành động ở đuôi mũi tên phải được hoàn thành trước khi thực hiện hành động ở đầu mũi tên.



* + - Extend

Extend dùng để mô tả quan hệ giữa 2 Use Case. Quan hệ Extend được sử dụng khi có một Use Case được tạo ra để bổ sung chức năng cho một Use Case có sẵn và được sử dụng trong một điều kiện nhất định nào đó. Trong Extend, hành động có thể có hoặc có thể không thực hiện cũng được.

* Ví dụ minh họa cho phương pháp use-case



### Phương pháp OOAD (phân tích và thuyết kế hướng đối tượng)

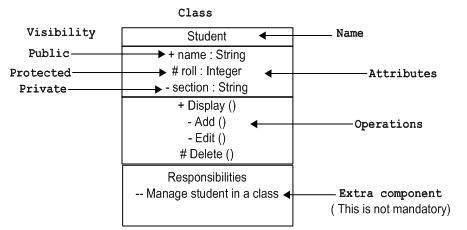
* Phương pháp OOAD
  + Khái niệm

Object Oriented Analysis And Design (OOAD) là một ngôn ngữ mô hình gồm các ký hiệu đồ họa mà các phương pháp hướng đối tượng sử dụng để thiết kế các hệ thống thông tin một cách nhanh chóng.

* + Ký hiệu (notation)
    - Class

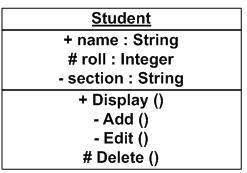
Lớp UML được biểu diễn bằng hình dưới đây. Sơ đồ được chia thành bốn phần:

* + - * Phần trên cùng được sử dụng để đặt tên cho lớp.
      * Phần thứ hai được sử dụng để hiển thị các thuộc tính của lớp.
      * Phần thứ ba được sử dụng để mô tả các hoạt động được thực hiện bởi lớp.
      * Phần thứ tư là tùy chọn để hiển thị bất kỳ thành phần bổ sung.



* + - Object (đối tượng)

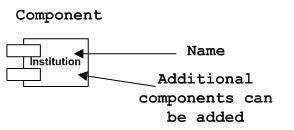
Đối tượng được đại diện theo cách giống như lớp. Sự khác biệt duy nhất là tên được gạch chân như thể hiện trong hình dưới đây.



Vì đối tượng là một thực hiện thực tế của một lớp, được gọi là thể hiện của một lớp. Do đó, nó có cùng cách sử dụng như lớp học.

* + - Component

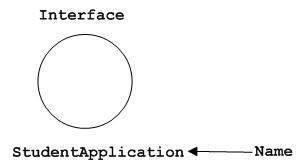
Một thành phần trong UML được thể hiện trong hình dưới đây với tên bên trong. Các yếu tố bổ sung có thể được thêm vào bất cứ nơi nào yêu cầu.



Hợp phần được sử dụng để đại diện cho bất kỳ phần nào của một hệ thống mà các sơ đồ UML được tạo ra.

* + - Interface

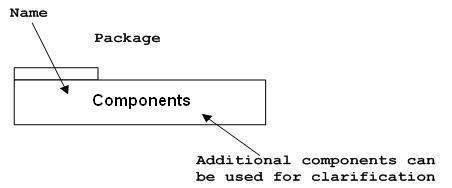
Giao diện được đại diện bởi một vòng tròn như thể hiện trong hình sau đây. Nó có một cái tên thường được viết dưới vòng tròn.



Giao diện được sử dụng để mô tả các chức năng mà không cần thực hiện. Giao diện giống như một mẫu mà bạn xác định các chức năng khác nhau chứ không phải là việc thực hiện. Khi một lớp triển khai giao diện, nó cũng thực hiện các chức năng theo yêu cầu.

* + - Package

Một gói là một nhóm các yếu tố có tổ chức. Một gói có thể chứa những thứ cấu trúc như các lớp, các thành phần và các gói khác trong đó. Khái niệm gói được thể hiện trong hình dưới đây và được sử dụng để gói các thành phần của một hệ thống.



* + - Mối quan hệ

Relationship thể hiện mối quan hệ giữa các Class với nhau. Trong UML 2.0 có các quan hệ thường sử dụng như sau:

* + - * Association
      * Aggregation
      * Composition
      * Generalization

Cụ thể:

* + - * **Association**

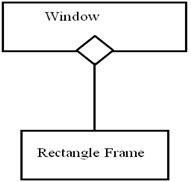
Association là quan hệ giữa hai lớp với nhau, thể hiện chúng có liên quan với nhau. Association thể hiện qua các quan hệ như “has: có”, “Own: sở hữu” v.v…

Ví dụ quan hệ trên thể hiện Khách hàng nắm giữ Tài khoản và Tài khoản được sở hữu bởi Khách hàng:

Association 1

* + - **Aggregation**

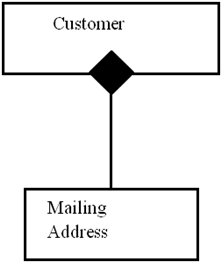
Aggregation là một loại của quan hệ Association nhưng mạnh hơn. Nó có thể cùng thời gian sống (cùng sinh ra hoặc cùng chết đi)

Ví dụ quan hệ trên thể hiện lớp Window(cửa sổ) được lắp trên Khung cửa hình chữ nhật. Nó có thể cùng sinh ra cùng lúc: 

* + - **Composition**

Composition là một loại mạnh hơn của Aggregation thể hiện quan hệ class này là một phần của class kia nên dẫn đến cùng tạo ra hoặc cùng chết đi.

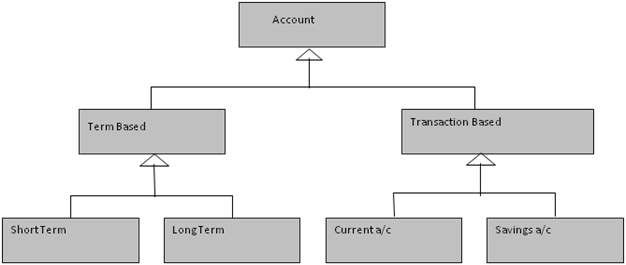
Ví dụ về class Mailing Address là một phần của class Customer nên chỉ khi nào có đối tượng Customer thì mới phát sinh đối tượng Mailing Address:



* + - **Generalization**

Generalization là quan hệ thừa kế được sử dụng rộng rãi trong lập trình hướng đối tượng.

Ví dụ về Genelization:

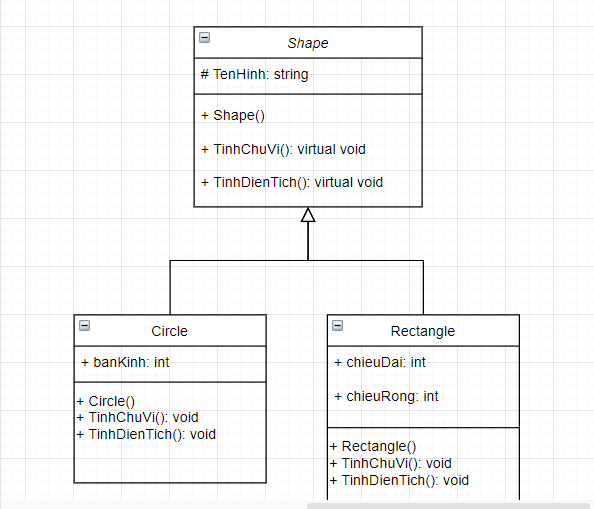


Các lớp ở cuối cùng như Short Term, Long Term, Current a/c, Savings a/c gọi là các lớp cụ thể (concrete Class). Chúng có thể tạo ra đối tượng và các đối tượng này thừa kế toàn bộ các thuộc tính, phương thức của các lớp trên.

Các lớp trên như Account, Term Based, Transaction Based là những lớp trừu tượng (Abstract Class), những lớp này không tạo ra đối tượng.

Ngoài ra, còn một số quan hệ như khác như dependence, realization nhưng ít được sử dụng.

* 1. Ví dụ minh họa



### Phương pháp DFD

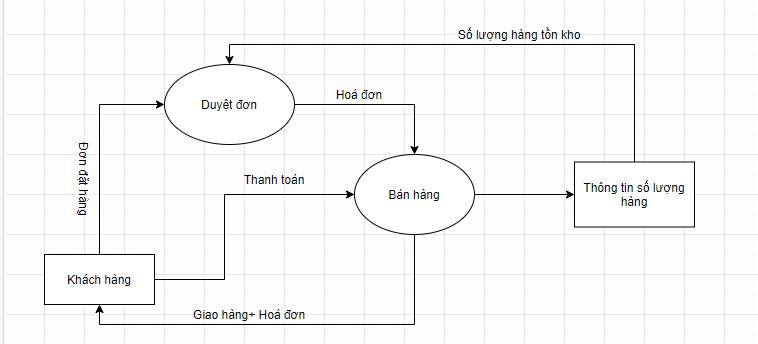
* Khái niệm

Sơ đồ luồng dữ liệu (Data Flow Diagram) là một mô hình hệ thống cân xứng cả dữ liệu và tiến trình (progress). Nó chỉ ra cách thông tin chuyển vận từ một tiến trình hoặc từ chức năng này trong hệ thống sang một tiến trình hoặc chức năng khác.

* Ký hiệu (notation)

|  |  |  |
| --- | --- | --- |
| **Khái niệm** | **Ký hiệu** | **Ý nghĩa** |
| **Xử lý (process)** |  | - Process là các bộ phận của hệ thống nhận đầu vào và tạo ra đầu ra.  - Không có bộ nhớ hoặc trạng thái riêng của chúng và được giả định là hoạt động tức thời, khi sơ đồ luồng dữ liệu được sử dụng để mô hình hóa các yêu cầu thiết yếu. |
| **Dòng dữ liệu (Data flow)** |  | - **Data flow** đại diện cho thông tin được chuyển giữa các quy trình, cửa hàng và người dùng hoặc hệ thống bên ngoài.  - **Data flow** được biểu diễn bằng các mũi tên trên sơ đồ luồng dữ liệu, được gắn nhãn bằng tên của dữ liệu có thể được truyền dọc theo chúng. |
| **Kho dữ liệu (Data store)** |  | - **Data store** là vùng chứa thông tin, tương ứng với một số phần của sơ đồ mối quan hệ-thực thể cho cùng một hệ thống. |
| **Tác nhân đầu cuối (Terminator)** |  | **- Terminator** đại diện cho người dùng của hệ thống - những người hoặc các hệ thống khác bên ngoài hệ thống đang được mô hình hóa, những người mà hệ thống đang được mô hình hóa giao tiếp bằng cách nhận đầu vào hoặc trả lại đầu ra. |

* 1. Ví dụ minh họa



# SLIDE 3: Pseudo Code – Flowchart

## Pseudocode là gì? (mã giã)

Pseudocode là một mô tả dạng plain text của một đoạn code hoặc thuật toán. Nó không thực sự là lập trình, không có script, không có file và không có chương trình. Đúng như tên gọi, nó là “code giả”.

Pseudocode không được viết bằng bất kỳ ngôn ngữ lập trình cụ thể nào. Nó viết bằng tiếng Anh và dễ hiểu.

Mặc dù pseudocode không được viết bằng [ngôn ngữ lập trình](https://quantrimang.com/ban-co-biet-ngon-ngu-lap-trinh-la-gi-165350), nhưng vẫn có những keyword được sử dụng để chỉ các khái niệm phổ biến khi viết code. Chúng được viết bằng chữ in hoa để dễ đọc hơn.

* START
* INPUT
* READ/GET
* PRINT/DISPLAY
* CALCULATE/DETERMINE
* SET
* INCREMENT/DECREMENT
* PROGRAM
* END

Dưới đây là một snippet về việc pseudocode có thể trông như thế nào đối với một chương trình yêu cầu bạn nhập màu yêu thích và xuất ra lựa chọn của bạn.

START

PROGRAM getColor

Create variable Color

Ask the user for their favorite color

READ INPUT into Color

PRINT Color

END

Đây là một thuật toán khá đơn giản được viết bằng pseudocode. Bất cứ ai cũng có thể đọc và hiểu những gì nó đang cố gắng thực hiện.

*Pseudocode viết thuật toán, ngôn ngữ lập trình viết cú pháp.*

## Pseudocode hữu ích ra sao?

Pseudocode giúp bạn lập kế hoạch cho ứng dụng của mình trước khi viết. Nó giúp bạn tạo các thuật toán theo định dạng dễ đọc hơn cú pháp code.

Ví dụ JavaScript rất dễ đọc nếu bạn biết ngôn ngữ lập trình này. Các thuật ngữ cụ thể như **window.prompt** hoặc **console.log** không tiết lộ nhiều về thuật toán.

Nếu bạn phỏng vấn để trở thành một kỹ sư phần mềm, nhà tuyển dụng sẽ không mong muốn bạn ghi nhớ cú pháp. Họ sẽ hỏi kiến ​​thức của bạn về các thuật toán và cấu trúc. Bạn sẽ viết code tốt hơn nhiều nếu xây dựng các thuật toán và cấu trúc của mình trước khi bắt đầu viết code.

## Flow charts là gì?

Flow chart thực chất là một dạng biểu đồ được hay được sử dụng để trình bày những bước ở trong một quy trình nào đó.

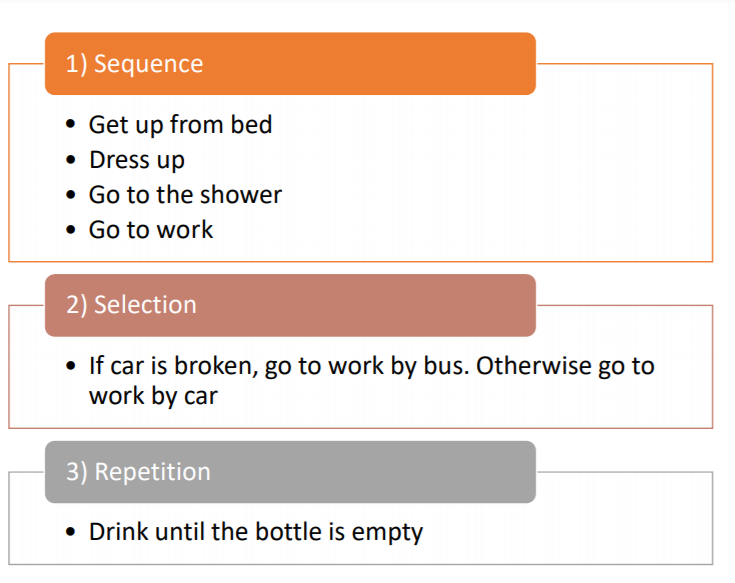
* Biểu đồ hay lưu đồ Flow chart thường hay được sử dụng để hỗ trợ cho các hoạt động đào tạo, lập kế hoạch hay tạo tài liệu. Đây là một loại biểu đồ có thể rất hữu ích dành cho bạn đối với việc ra một quyết định nào đó. Loại lưu đồ này rất thường dùng các mũi tên để minh họa cho một bước dẫn tiếp theo trong một chu trình nào đó.
* Tạo điều kiện giao tiếp giữa lập trình viên và doanh nhân.
* Đóng một vai trò quan trọng trong việc lập trình một vấn đề và khá hữu ích trong việc hiểu logic của những vấn đề phức tạp và dài dòng. Khi lưu đồ được vẽ, việc viết chương trình bằng bất kỳ ngôn ngữ cấp cao nào trở nên dễ dàng. Thông thường, chúng tôi thấy sơ đồ hữu ích như thế nào trong việc giải thích chương trình cho người khác. Do đó, đúng khi nói rằng một sơ đồ là cần thiết để tạo ra tài liệu tốt hơn cho một chương trình phức tạp.

## Một số trường hợp cụ thể nên áp dụng sơ đồ quy trình Flow chart là:

* Trường hợp sử dụng để phát triển sự hiểu biết về cách một quá trình được thực hiện
* Nghiên cứu một quá trình cải tiến
* Để truyền đạt cho người khác cách một quá trình được thực hiện
* Khi cần giao tiếp tốt hơn giữa những người có cùng quy trình
* Để ghi lại một quá trình
* Khi lập kế hoạch dự án

# SLIDE 04: Structure Theorem (định lý cấu trúc)

## Các quy tắc cơ bản (trình tự,lựa chọn, lập lại)



## Lợi ích

Đây được coi là phương pháp hay cho một sơ đồ không bao giờ vượt quá giới hạn của một trang.

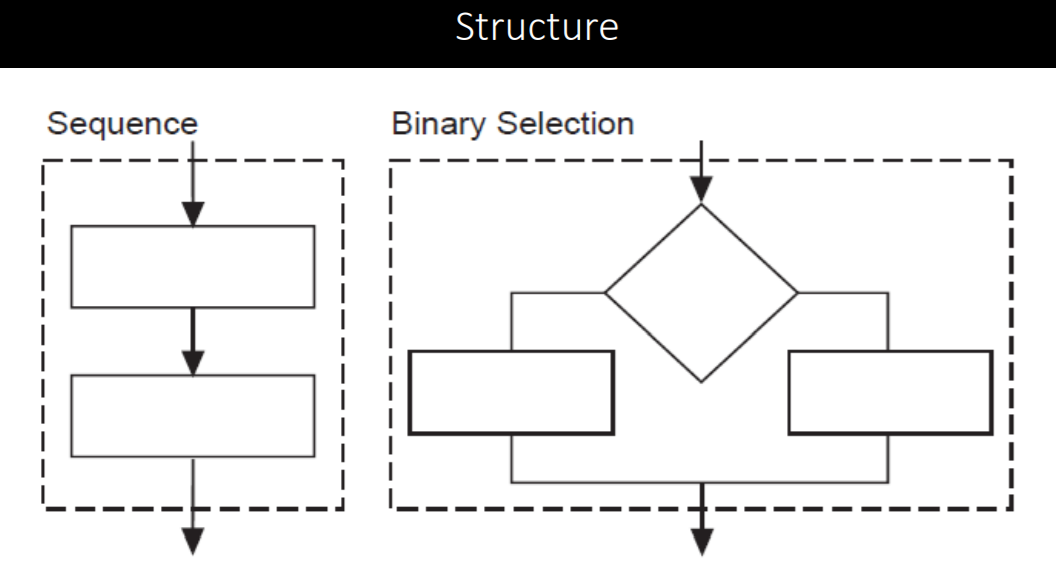
• Nếu một sơ đồ không vừa trên một trang, lọc và tạo ra các chương trình con.

• Một sơ đồ thể hiện giải pháp cho một vấn đề liên quan có thể có:

- sơ đồ chương trình chính trên một trang

- với các chương trình con tiếp tục giải quyết vấn đề trên các trang tiếp theo.

• Bất kể kích thước trang là bao nhiêu, điều quan trọng là bắt đầu bất kỳ thuật toán phức tạp nào với một dòng chính(main line) rõ ràng, gọn gàng. Điều này sẽ tham chiếu đến các chương trình con bắt buộc có chi tiết được hiển thị trong các sơ đồ riêng biệt.



## Cách dùng

• Mỗi cấu trúc trong số năm cấu trúc được chấp nhận có thể được xây dựng từ các yếu tố cơ bản như đã trình bày trước đây.

• Trong mọi trường hợp, lưu ý có:

- chỉ một điểm vào (only one entry point) và một điểm ra (one exit point) được đặt trong các dashed boxes.

- Vì mỗi cấu trúc có thể được coi là một quy trình (như được hiển thị bằng các hộp (dashed boxes) chứa cấu trúc), các thuật toán phức tạp hơn có thể được xây dựng bằng cách thay thế bất kỳ quy trình đơn lẻ nào bằng một hoặc nhiều cấu trúc khác.

\*các dùng và ví dụ (chỉ minh họa 1 dạng đặc trưng để áp dụng)

|  |  |
| --- | --- |
| Trình tự (sequence) |  |
| Lựa chọn (selection) |  |
| Lặp lại (repetition) |  |
| Chương trình con  (Subprograms) |  |
|  |  |

# SLIDE 05: MÔ HÌNH HÓA QUY TRÌNH NGHIỆP VỤ --UML—

1. Nội dung

Cập nhật sau

1. Bài tập

# SLIDE 7 : REQUIREMENT MODELLING (mô hình hóa yêu cầu)

## 1. Mô hình hóa yêu cầu là gì ?

Mô hình hóa yêu cầu là quy trình được sử dụng trong các dự án phát triển phần mềm nơi các yêu cầu và giải pháp liên tục phát triển thông qua nỗ lực hợp tác và làm việc theo nhóm. Bằng cách sử dụng phương pháp nhóm chức năng chéo và tự tổ chức này, bạn có thể đảm bảo rằng nhóm của bạn đáp ứng nhu cầu chính xác của các bên liên quan.

## 2. Nắm bắt yêu cầu

Trong hoạt động này, đội phân tích sẽ làm việc cùng với phía khách hàng để thu thập thông tin về hệ thống hiện tại của khách hàng, những khó khăn trong công việc của họ với hệ thống hiện tại. Từ đó xác định chính xác các yêu cầu cho hệ thống mới.

Tất cả các yêu cầu này được mô tả vắn tắt, dễ hiểu trong tài liệu gọi là Tài liệu định nghĩa yêu cầu. Tài liệu này được sử dụng để cam kết với khách hàng về yêu cầu cho nên thường được viết bằng ngôn ngữ tự nhiên và được mô tả đơn giản, dễ hiểu.

## 3. Đặc tả yêu cầu

Là hoạt động mô tả chi tiết về các yêu cầu và mối quan hệ giữa các yêu cầu của một hệ thống một cách chính xác và chặt chẽ sao cho người đọc không hiểu sai về yêu cầu. Ngoài ngôn ngữ tự nhiên, người ta thường dùng các ngôn ngữ chuyên dụng để đặc tả yêu cầu vì phải đảm bảo tính chặt chẽ của nội dung và vì tài liệu này phục vụ cho các đội phát triển và bảo trì phần mềm.

Do nhiều yếu tố mà việc nắm bắt yêu cầu trong thực tế gặp rất nhiều khó khăn. Một trong các lý do là người dùng khó diễn đạt mong muốn của mình còn đội phân tích khó hiểu nổi khách hàng muốn gì ở hệ thống mới.

Vì vậy, trong quá trình phân tích, người ta thường thực hiện thêm hai hoạt động bổ trợ, đó là: Làm bản mẫu và mô hình hóa.

## 4. Làm bản mẫu là việc xây dựng các mô hình chạy thử để người dùng quan sát, dùng thử và đánh giá.

Hoạt động này rất hữu ích khi đội phân tích không chắc chắn về một yêu cầu nào đó, họ làm bản mẫu và người

dùng đánh giá xem đó có đúng là cái họ cần hay không.

## 5. Mô hình hóa: khi chưa chắc chắn về một miền vấn đề hoặc một phần nào đó của hệ thống, người ta

xây dựng mô hình cho nó, nghĩa là phác họa những nét chính của miền vấn đề đó, lược bỏ những chi tiết phức

tạp, tạo ra mô hình đơn giản nhưng chứa đựng những nét cốt lõi nhất của miền vấn đề để trao đổi với khách

hàng và xác nhận xem đã hiểu đúng về miền vấn đề hay chưa.

Hai hoạt động này giúp cho đội phân tích hiểu đúng và đủ các yêu cầu cũng như các khía cạnh khác của hệ

thống như khía cạnh tổ chức, quy trình nghiệp vụ,…

Ngoài ra, để đảm bảo chất lượng của giai đoạn phân tích và giảm thiểu sự thay đổi yêu cầu sau này, người ta

thực hiện hoạt động thẩm định các tài liệu định nghĩa yêu cầu và tài liệu đặc tả yêu cầu. Việc thẩm định được

tiến hành bởi cả hai phía khách hàng và phía phát triển, trong nhiều trường hợp còn có thêm cá nhân/đơn vị

khác tham gia vào hoạt động thẩm định này. Các bên tham gia sẽ cùng kiểm tra lại các tài liệu định nghĩa và đặc

tả yêu cầu để nếu có thiếu sót về yêu cầu thì có thể sửa chữa kịp thời trước khi bàn giao tài liệu cho đội phát

triển, nhằm giảm thiểu chi phí cho sửa lỗi ở những giai đoạn sau. Các bên tham gia phải cam kết về các tài liệu

đã thẩm định, các tài liệu này được gọi chung là tài liệu yêu cầu.

Trong các mô đun sau, chúng ta sẽ bàn luận kỹ hơn về các hoạt động này, trước hết chúng ta cần hiểu yêu cầu

là gì và có những loại yêu cầu nào để việc nắm bắt yêu cầu được thực hiện tốt hơn vì mỗi loại yêu cầu có cách

thức thu thập và mô tả khác nhau

# SLIDE 8 : DATA MODEL

## Data model ( mô hình dữ liệu ) là gì ?

**Data model (mô hình dữ liệu)** là một mô hình trừu tượng tổ chức các phần tử của dữ liệu và chuẩn hóa cách chúng liên quan với nhau và với các thuộc tính của các thực thể trong thế giới thực.

Ví dụ: mô hình dữ liệu có thể chỉ định rằng phần tử dữ liệu đại diện cho một chiếc ô tô bao gồm một số phần tử khác, đến lượt nó, đại diện cho màu sắc và kích thước của chiếc ô tô và xác định chủ sở hữu của nó.

## Data modeling ( mô hình hóa dữ liệu ) là gì ?

**Mô hình hóa dữ liệu (data modelling)**là quá trình tạo ra một mô hình dữ liệu để dữ liệu được lưu trữ trong [cơ sở dữ liệu](https://a1digihub.com/database-la-gi/). Mô hình dữ liệu này là một biểu diễn khái niệm của các đối tượng dữ liệu, sự liên kết giữa các đối tượng dữ liệu khác nhau và các quy tắc.

## Các loại mô hình dữ liệu (Types of data model ) :

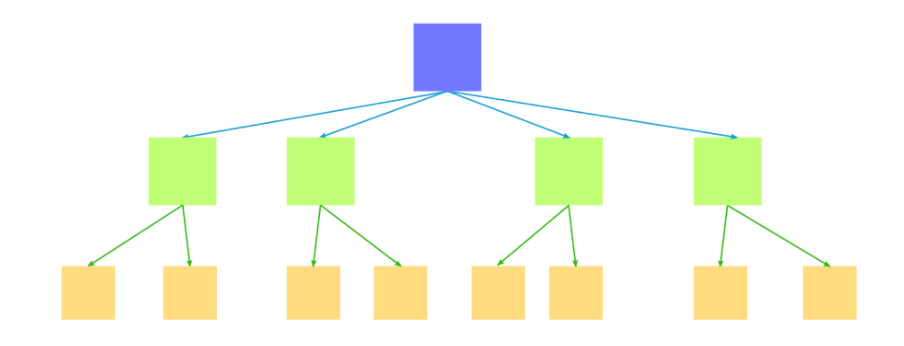
* **Conceptual data models**– Mô hình Dữ liệu Khái niệm: Mô hình dữ liệu này xác định **những gì** hệ thống chứa. Mô hình này thường được tạo bởi **các bên liên quan của doanh nghiệp** và Kiến trúc sư dữ liệu. Mục đích là để tổ chức, phạm vi và xác định các khái niệm và quy tắc kinh doanh.
* **Logical data models**– Mô hình dữ liệu logic: Xác định **cách** hệ thống sẽ được triển khai bất kể hệ quản lý cơ sở dữ liệu. Mô hình này thường được tạo bởi kiến trúc sư dữ liệu và nhà phân tích kinh doanh. Mục đích là phát triển bản đồ kỹ thuật của các quy tắc và cấu trúc dữ liệu.
* **Physical data models**– Mô hình Dữ liệu Vật lý: Mô hình dữ liệu này mô tả cách hệ thống sẽ được triển khai bằng cách sử dụng một hệ thống quản lý cơ sở dữ liệu cụ thể. Mô hình này thường được tạo bởi chuyên viên quản trị dữ liệu và các nhà phát triển. Mục đích là triển khai thực tế cơ sở dữ liệu.

## Quy trình trong mô hình hóa dữ liệu ? (Data modeling)

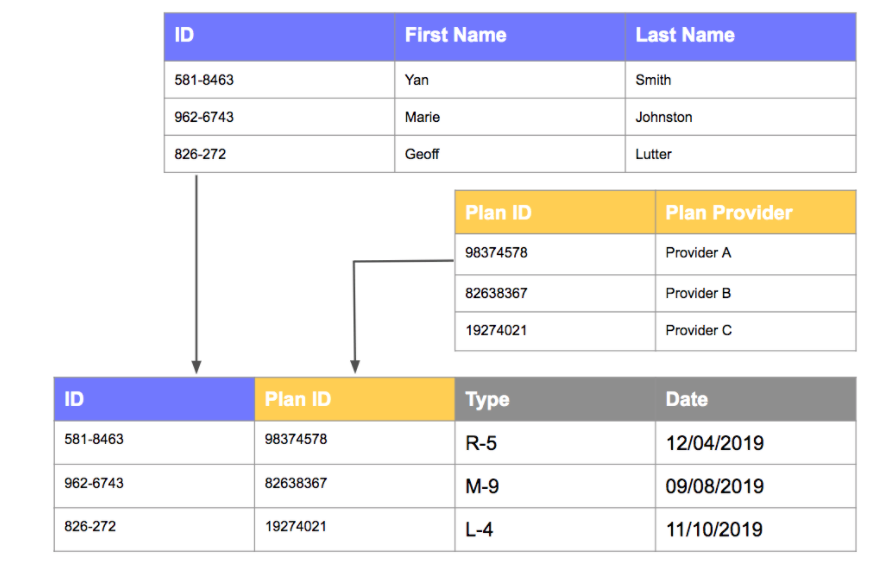
1. **Identify the entities – Xác định thực thể.**
2. **Identify key properties of each entity – Xác định thuộc tính của từng thực thể.**
3. **Identify relationships among entities – Xác định mối quan hệ giữa các thực thể**
4. **Assign keys as needed, and decide on a degree of normalization that balances the need to reduce redundancy with performance requirements : Gán các khóa khi cần thiết và quyết định mức độ chuẩn hóa cân bằng giữa nhu cầu giảm dư thừa với các yêu cầu về hiệu suất.**
5. **Finalize and validate the data model : Hoàn thiện và xác thực mô hình dữ liệu.**

## 5. Các loại mô hình hóa dữ liệu : (Types of data modeling)

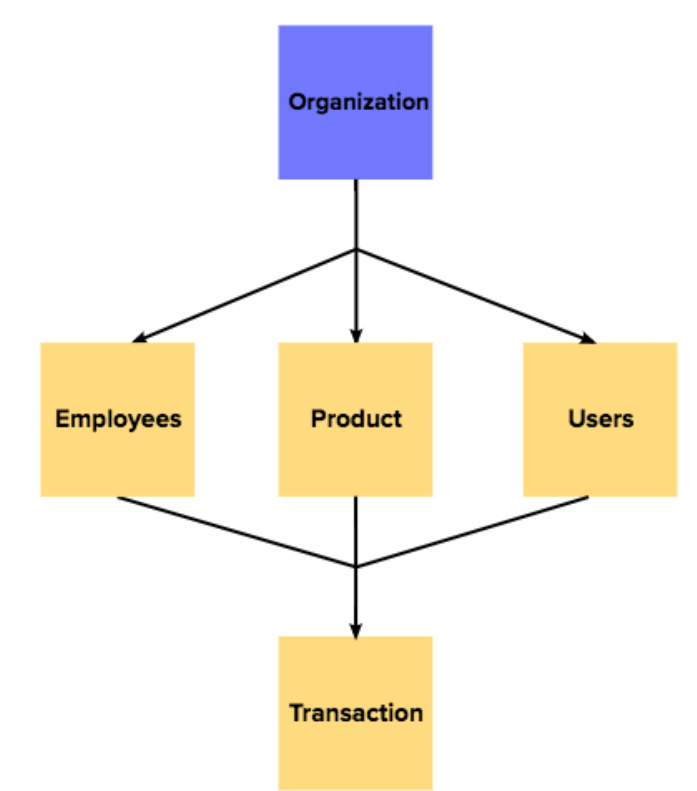
* **Mô hình phân cấp – Hierarchical model :** Mô hình dữ liệu này sử dụng hệ thống phân cấp để cấu trúc dữ liệu theo định dạng giống như mô hình cây.  Tuy nhiên, việc truy xuất và truy cập dữ liệu khá khó khăn trong cơ sở dữ liệu phân cấp.  Đây là lý do tại sao nó hiếm khi được sử dụng ngày nay.



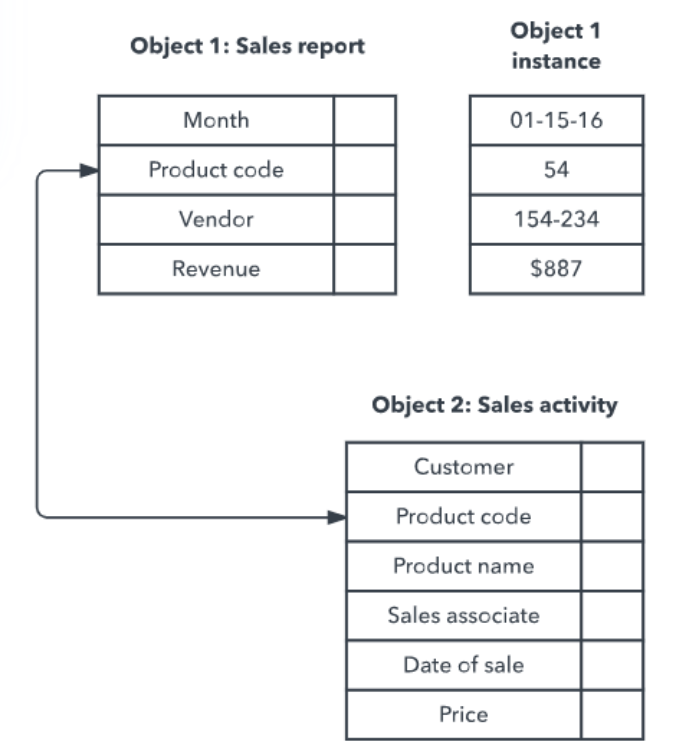
* **Mô hình quan hệ - Relation model :** Được đề xuất như là một thay thế cho mô hình phân cấp bởi một nhà nghiên cứu của IBM. Ở đây dữ liệu được biểu diễn dưới dạng bảng. Nó làm giảm sự phức tạp và cung cấp một cái nhìn tổng quan rõ ràng về dữ liệu.



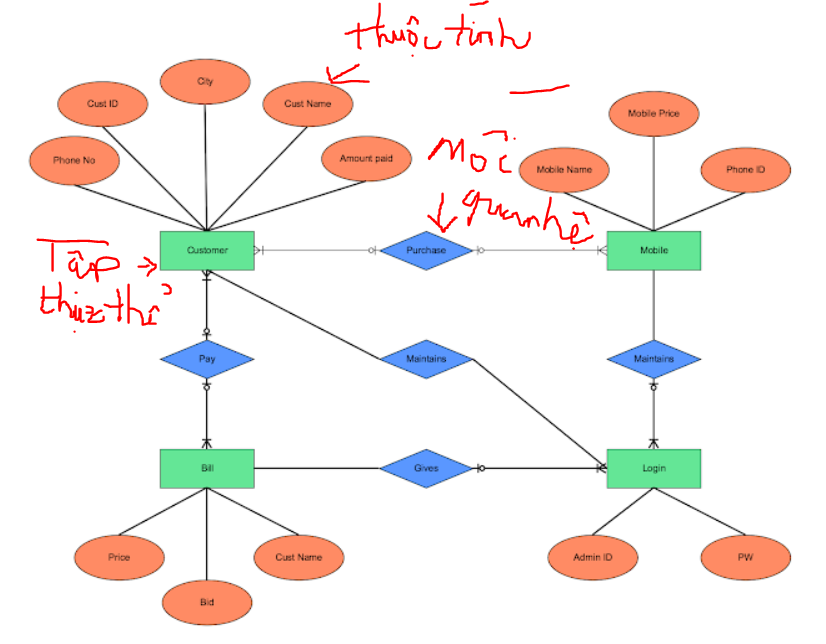
* **Mô hình mạng – Network model :** Mô hình mạng được lấy cảm hứng từ mô hình phân cấp.  Tuy nhiên, không giống như mô hình phân cấp, mô hình này giúp truyền đạt các mối quan hệ phức tạp dễ dàng hơn vì mỗi bản ghi có thể được liên kết với nhiều bản ghi khác nhau.



* **Mô hình hướng đối tượng - Object-oriented model :** Mô hình cơ sở dữ liệu này bao gồm một tập hợp các đối tượng, mỗi đối tượng có các tính năng và phương thức riêng. Kiểu mô hình cơ sở dữ liệu này còn được gọi là mô hình cơ sở dữ liệu hậu quan hệ.



* **Mô hình mối quan hệ thực thể - Entity relationship model :** Mô hình mối quan hệ thực thể, còn được gọi là mô hình ER, đại diện cho các thực thể và các mối quan hệ của chúng ở định dạng đồ họa. Một thực thể có thể là bất cứ thứ gì – một khái niệm, một phần dữ liệu hoặc một đối tượng.



* **Mô hình dữ liệu liệu ngữ nghĩa – Semantic data modeling :** Mô hình dữ liệu ngữ nghĩa (SDM) là mô tả cơ sở dữ liệu cấp cao dựa trên ngữ nghĩa và cấu trúc hình thức (mô hình cơ sở dữ liệu) cho cơ sở dữ liệu. Mô hình cơ sở dữ liệu này được thiết kế để nắm bắt nhiều ý nghĩa của môi trường ứng dụng hơn là khả năng có thể có với các mô hình cơ sở dữ liệu hiện đại.

## 6. Lợi của mô hình hóa dữ liệu : Benefits of data modeling

* Cải thiện khả năng khám phá, tiêu chuẩn hóa và tài liệu hóa các nguồn dữ liệu.
* Giúp doanh nghiệp có thể thiết kế và áp dụng database một cách hiệu quả
* Quản lý doanh nghiệp hiệu quả hơn
* Nâng cao tinh thần của các nhân viên
* Hỗ trợ nâng cấp BI của doanh nghiệp
* Tăng khả năng tích hợp trong hệ thống doanh nghiệp

## 7. Công cụ BI cho doanh nghiệp ứng dụng data modeling

* **Tableau**
* **FineReport**
* **Power BI**
* **Qlikview**
* **Sisense**

## 8. Luật nghiệp vụ:

* Xác định khái niệm dữ liệu
* Các hướng ràng BUỘC của ứng dụng : các yếu tố ràng buộc , rang buộc thành viên,
* Ví dụ :

• Để được xem như một khách hàng của của hàng, mỗi khách hàng cần phải có ít nhất một đơn đặt hàng

• Một công nhân không thể lảnh lương cao hơn người quản lý

• Một phòng thường được đặt ở nhiều hơn một văn phòng, mỗi văn phòng cần phải thuộc vào một phòng.

## 9. Chuẩn hóa thông tin :

* Thông tin dư thừa
* Cập nhật thường xuyên
* Chuẩn hoá thông tin
* Phụ thuộc hàm

# SLIDE 9: SYSTEMS MODEL OOAD

## Phân tích và thiết kế hướng đối tượng – OOAD : Object-Oriented Analysis and Design

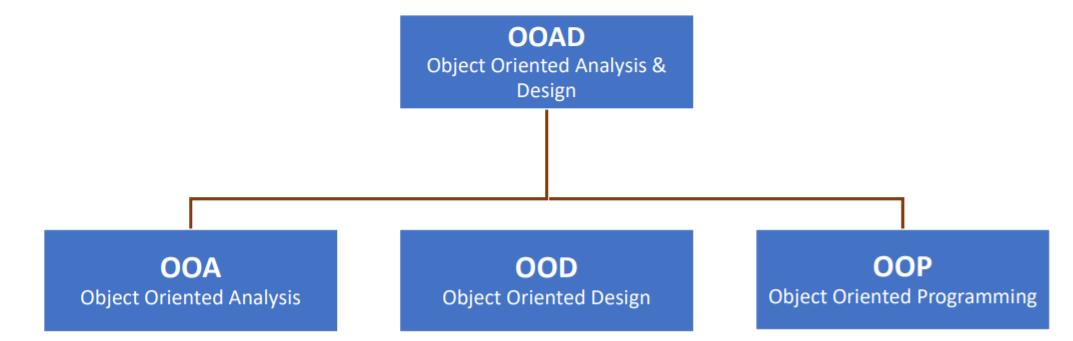
* Là một kỹ thuật tiếp cận phổ biến dùng để phân tích, thiết kế một ứng dụng, hệ thống. Nó dựa trên bộ các nguyên tắc chung, đó là một tập các hướng dẫn để giúp chúng ta tránh khỏi một thiết kế xấu.
* OOAD là một trong ba nhánh chính của phân tích và thiết kế hệ thống, bao gồm: • Phương pháp hướng đối tượng (OOAD)

• Phương pháp hướng cấu trúc

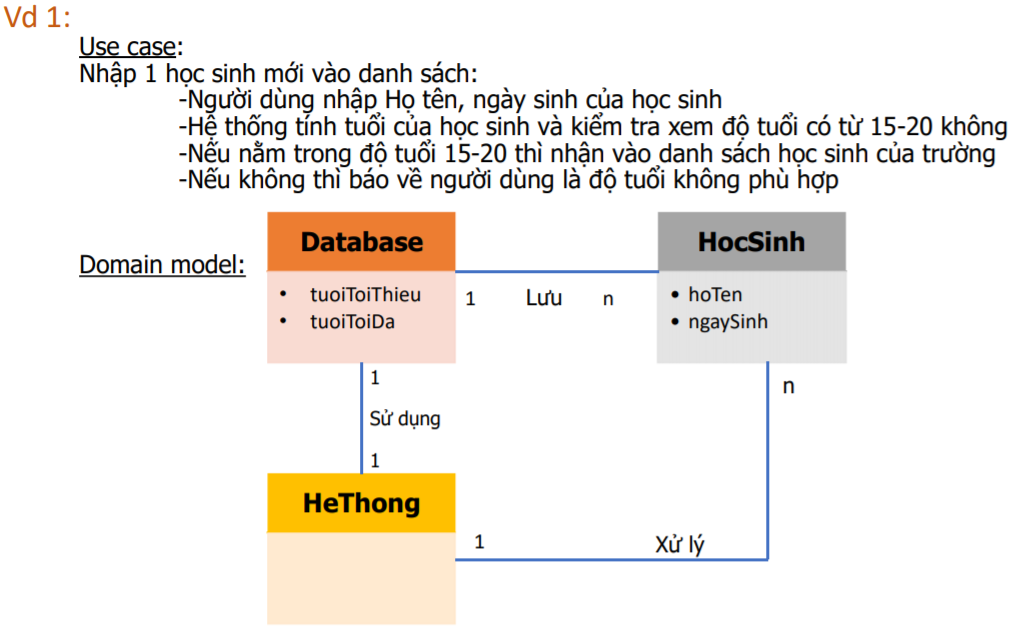
• Phương pháp hướng thích nghi

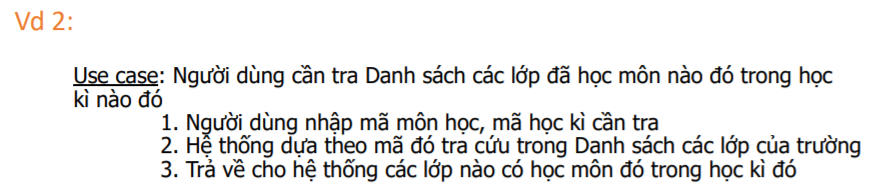
* Sử dụng bộ ngôn ngữ UML (ngôn ngữ mô hình hóa thống nhất).

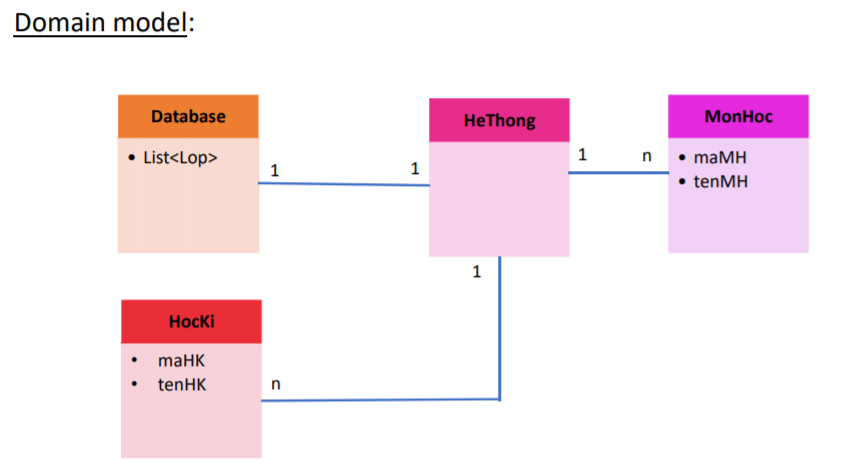
## Các bước trong phương pháp OOAD



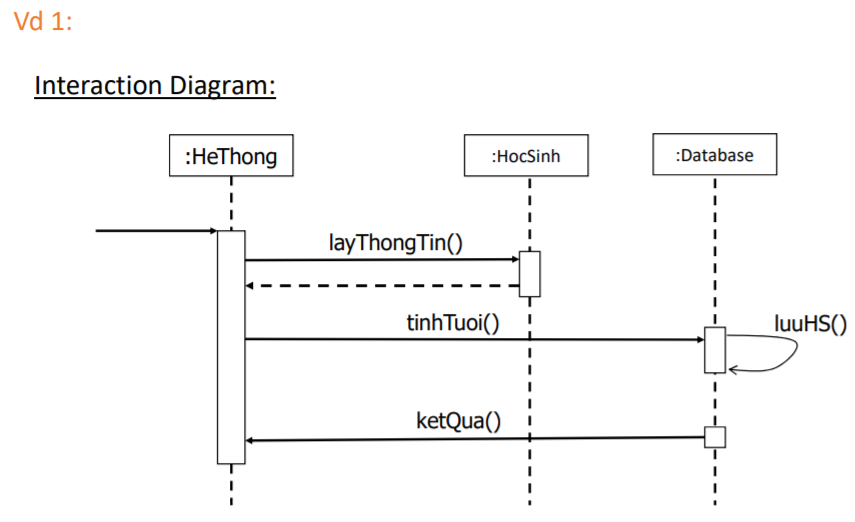
* **Object Oriented Analysis – OOA – Phân tích hướng đối tượng** : Giai đoạn này trình bày vấn đề bằng các thuật ngữ tương ứng với các đối tượng có thực, và phải được định nghĩa sao cho người không chuyên cũng có thể hiểu được, đồng thời vẫn giữ nguyên các mẫu hình về cấu trúc, quan hệ cũng như hành vi của chúng
* Xác định use case : Use case là một văn bản hoặc một câu chuyện diễn tả cách làm việc của hệ thống
* Xác định domain model : Domain model là mô hình các khái niệm, sự vật quan trọng trong miền ứng dụng và quan hệ ràng buộc giữa chúng

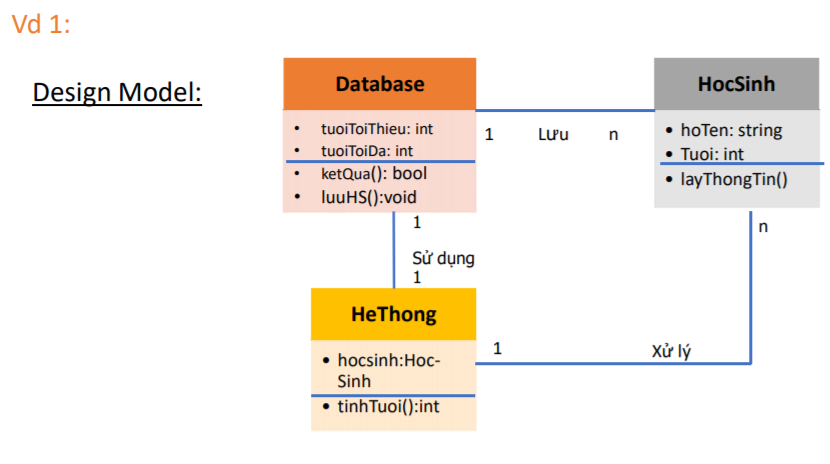






* **Object Oriented Design – OOD – Thuyết kế hướng đội tượng**: Là giai đoạn tổ chức chương trình thành các tập hợp đối tượng , mỗi đối tượng trong đó là thực thể của một lớp. Các lớp là thành viên của một cây cấu trúc với mối quan hệ thừa kế.
* Xác định Interaction Diagram : Thể hiện dòng thông điệp đi giữa các object
* Xác định Design Model : Xác định thêm các thuộc tính, phương thức nhằm giải bài toán phần mềm

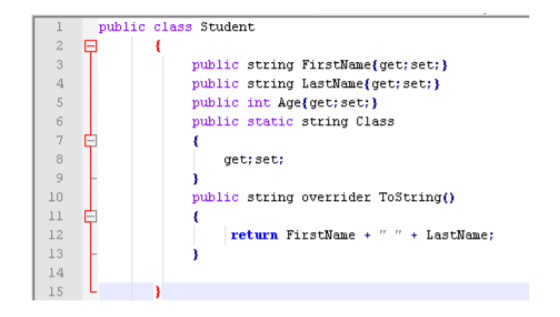




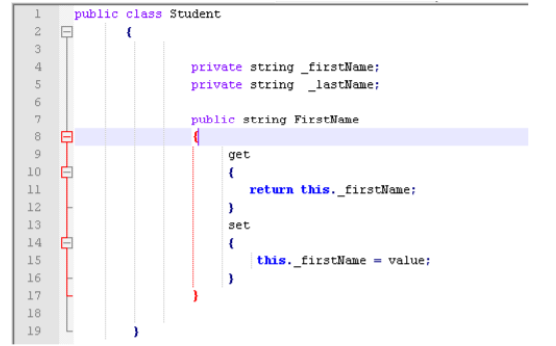
* **Object Oriented Programming – OOP – Lập trình hướng đối tượng :** Là giai đoạn hiện thực hóa phần mềm dựa trên kết quả của OOD. Sử dụng các ngôn ngữ lập trình hỗ trợ hướng đối tượng để chuyển hóa mô hình thành code.
* Một số ngôn ngữ hỗ trợ hướng đối tượng: C++ , Java , C# , Perl , Python , PHP.

## 4 nguyên tắc căn bản của OOP :

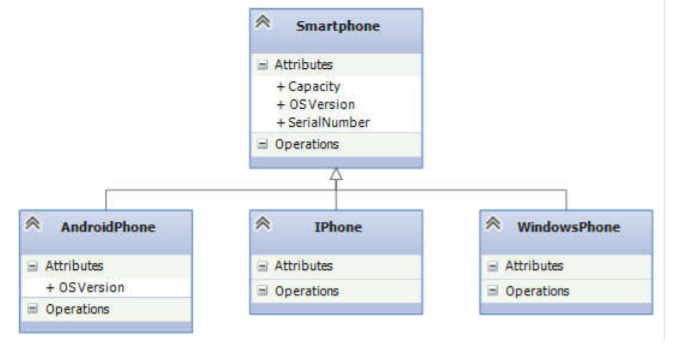
#### **• Abstraction - tính trừu tượng :** Khi thiết kế các đối tượng, ta cần rút tỉa ra những đặc trưng của chúng, rồi trừu tượng hóa thành các interface, và thiết kế xem chúng sẽ tương tác với nhau như thế nào. Nói cách khác, chúng ta định ra các interface và các contract mà chúng cần thỏa mãn.



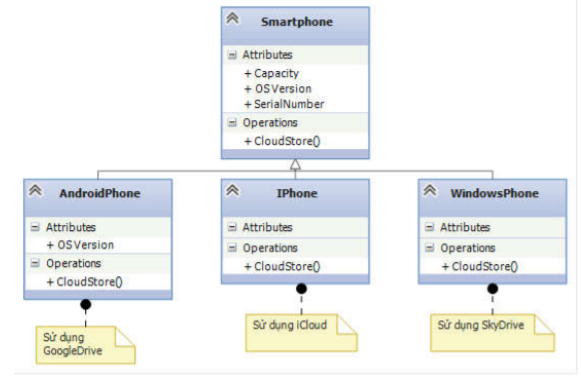
#### **• Encapsulation - tính đóng gói :** Tính chất này không cho phép người sử dụng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.



#### **• Inheritance - tính kế thừa :** Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.



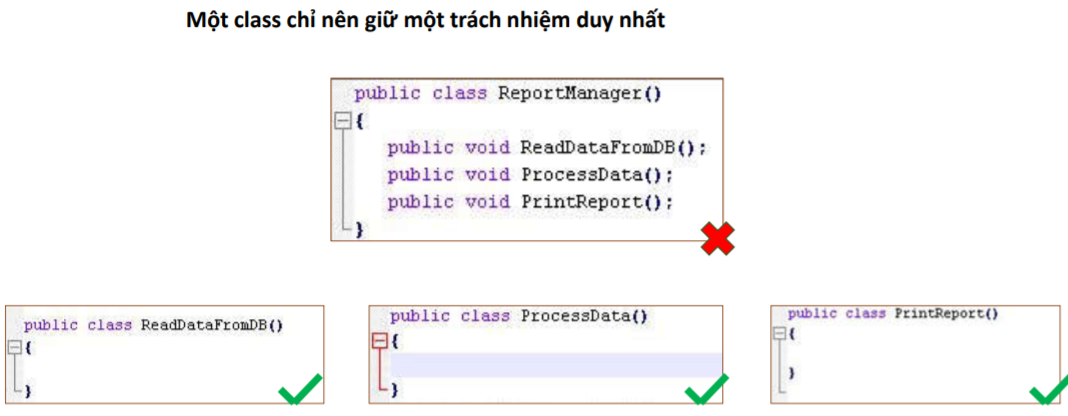
#### **• Polymorphism - tính đa hình :** Đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.



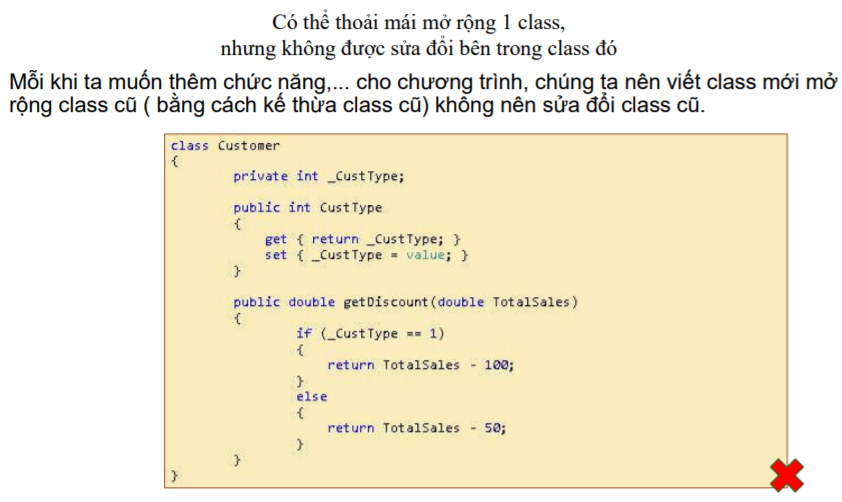
## Nguyên tắc của hướng đối tượng

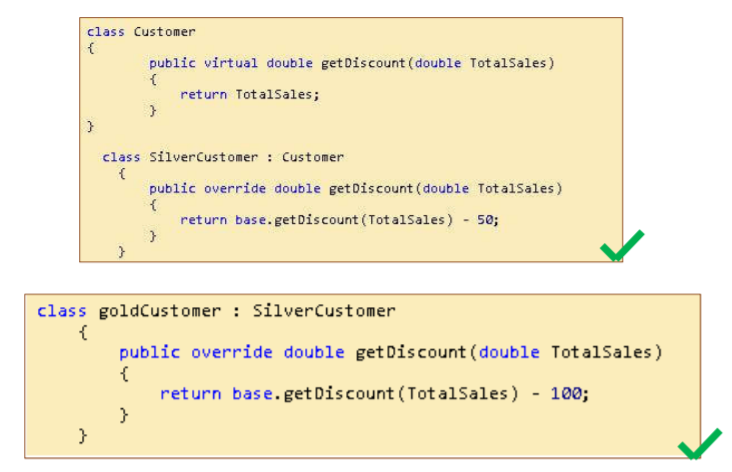
**S.O.L.I.D** là một từ viết tắt cho 5 nguyên tắc hàng đầu của thiết kế hướng đối tượng (OOD), được đề xuất bởi Robert C. Martin, thường được gọi là Bác Bob.

#### **S – Single-responsiblity principle**



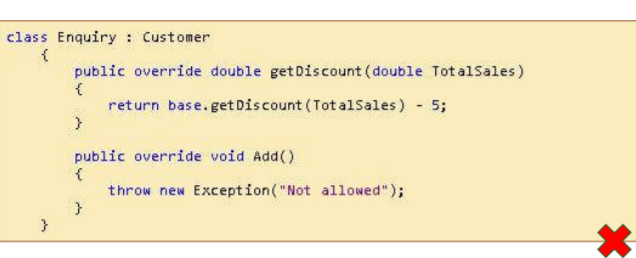
#### **O – Open-closed principle**

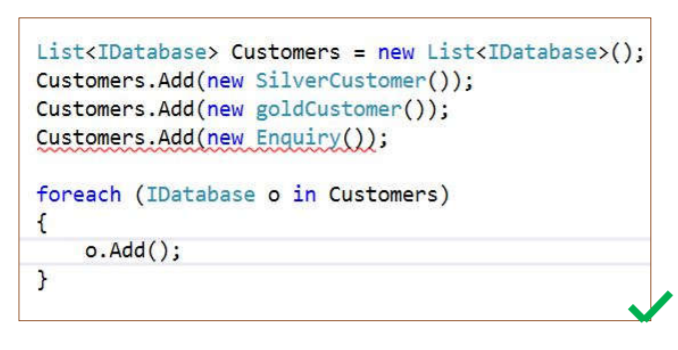




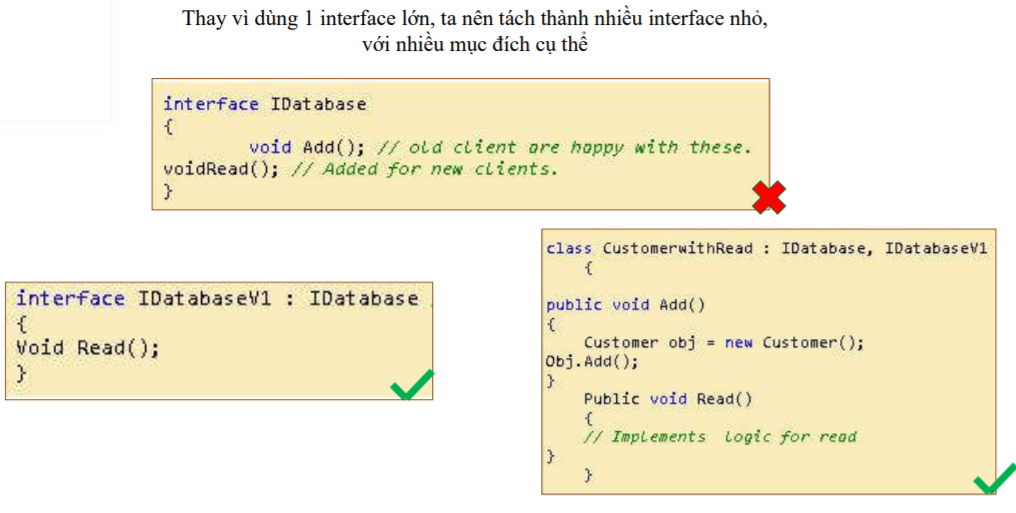
#### **L – Liskov substitution principle :**

Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình.





#### **I – Interface segregation principle**



#### **D – Dependency Inversion Principle**

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

## So sánh giữa hướng đối tượng và hướng cấu trúc

