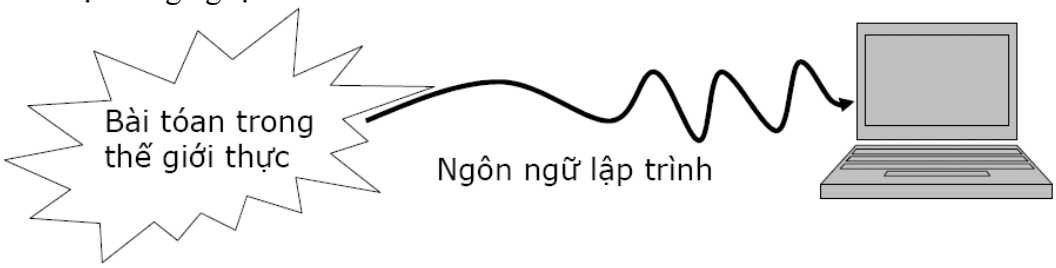
Chương I. TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

I. Giới thiệu

- Mục tiêu của kỹ sư lập trình:
    + Tạo ra sản phẩm tốt một cách có hiệu quả
    + Nắm bắt được công nghệ



- Độ phức tạp và độ lớn ngày càng cao:
    + Một số hệ Unix chứa khoảng 4M dòng lệnh
    + MS Windows chứa hàng chục triệu dòng lệnh
    + Người dùng ngày càng đòi hỏi nhiều chức năng, đặc biệt là chức năng thông minh
    + Phần mềm luôn cần được sửa đổi
    + …
- Cần kiểm soát chi phí: Chi phí phát triển, Chi phí bảo trì
- Giải pháp chính là sử dụng lại (tái sử dụng): Giảm chi phí và thời gian phát triển; Nâng cao chất lượng
- Để sử dụng lại (mã nguồn): Cần dễ hiểu; Được coi là chính xác; Có giao diện rõ ràng; Tính module hóa; *Không yêu cầu thay đổi khi sử dụng trong chương trình mới*

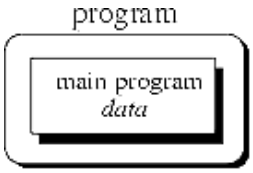
- Mục tiêu của việc thiết kế một phần mềm
    + Tính tái sử dụng (reusability): thiết kế các thành phần có thể được sử dụng trong nhiều phần mềm khác nhau
    + Tính mở rộng (extensibility)
    + Tính mềm dẻo (flexibility):
        . Có thể dễ dàng thay đổi khi thêm mới dữ liệu hay tính năng.
        . Các thay đổi không làm ảnh hưởng nhiều đến toàn bộ hệ thống

https://www.desy.de/gna/html/cc/Tutorial/node3.htm

II. Các phương pháp lập trình (Các trường phái lập trình)

    + Lập trình tuyến tính
    + Lập trình thủ tục/ cấu trúc
    + Lập trình module
    + Lập trình hướng đối tượng (*) *(Vượt trội, nhắc đến lập trình là nhắc đến OOP)*

1. Lập trình tuyến tính



- Giải quyết các bài toán nhỏ, đơn giản
- Chỉ gồm một chương trình main *(một chuỗi các lệnh hoặc câu lệnh sửa đổi dữ liệu mang tính tổng thể trong toàn bộ chương trình).*
- Chương trình gồm 1 chuỗi tuần tự các câu lệnh
- Kỹ thuật lập trình này gây ra những bất lợi to lớn khi chương trình đủ lớn:
    + Lặp lại những đoạn code giống nhau (Copy – paste) → Khó bảo trì, chỉnh sửa, nâng cấp…
    + Bất tiện khi viết chương trình lớn (khó tách thành nhiều chương trình, project nhỏ hơn)
    + Tất cả dữ liệu trong chương trình là toàn cục → không kiểm soát được phạm vi truy xuất dữ liệu
    + Code repetition:
        . Lỗi cú pháp → 1 chỗ lỗi – nhiều chỗ lỗi
        . Lỗi ngữ nghĩa: Chạy không có lỗi, nhưng kết quả đưa ra là không mong muốn → Khó khăn trong việc tìm lỗi (Do quá nhiều nơi giống nhau…).

2. Lập trình thủ tục/ cấu trúc (Procedure)

- Chương trình được tổ chức thành những *chương trình con* (Chia nhỏ)
- Chương trình con:
    + Hàm (những code có khả năng dùng lại nhiều lần, tách ra một hàm, khi dùng ta gọi hàm, tránh việc copy code), thủ tục
    + Độc lập nhau, thực hiện một tác vụ cụ thể

+ Khó khăn liên quan đến việc chia sẻ code trong team:

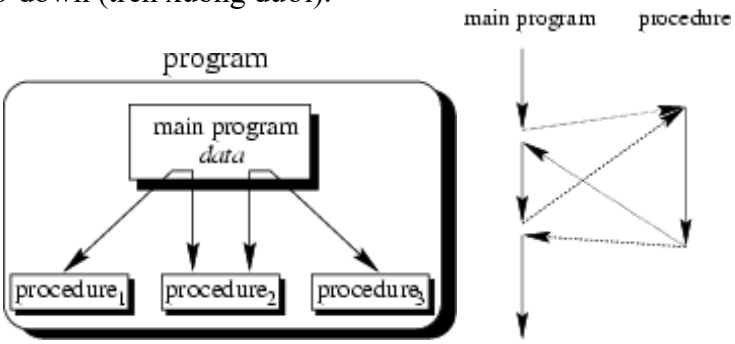
. 1 project trung bình, lớn sẽ share cho nhiều thành viên làm, sau đó gộp lại thành 1 chương trình.

. Nếu không có lỗi, hoàn tất thì không sao.

. Nếu có lỗi, sẽ khó khăn trong việc xác định lỗi là của ai (của bản thân mình, hay của người khác làm, copy lại bị sai, thiếu…).

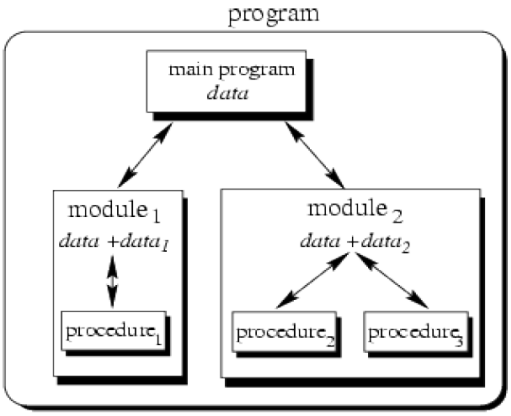
. Khi 1, 2 phần nhỏ cần nâng cấp, người phụ trách phần đó sẽ làm rồi copy đưa lại vào chương trình, nhưng không chắc chắn rằng chương trình sẽ chạy tốt như lúc đầu…

+ Viết theo dạng top-down (trên xuống dưới).



3. Lập trình module

- Các thủ tục/hàm được nhóm với nhau trong 1 module riêng biệt
- Các module truy xuất trên các dữ liệu, có thể được biên dịch một cách riêng lẻ (của ai người đó test, fix bug…)
- Chương trình gồm nhiều phần nhỏ
- Các phần tương tác thông qua việc gọi thủ tục



- Mỗi mô-đun có thể có dữ liệu riêng. Điều này cho phép mỗi mô-đun quản lý trạng thái bằng cách gọi các thủ tục của mô-đun này. *Tuy nhiên, chỉ có một trạng thái cho mỗi mô-đun và mỗi mô-đun tồn tại tối đa một lần trong toàn bộ chương trình.*

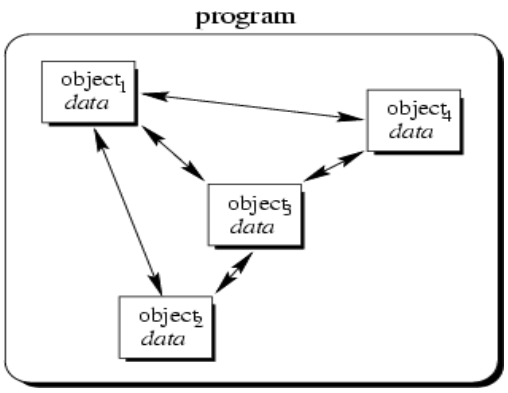
**** Nhược điểm của lập trình thủ tục & module***

+ Thay đổi dữ liệu dùng chung → thay đổi tất cả hàm/thủ tục liên quan
+ Chương trình khó kiểm soát (do dữ liệu dùng chung)
+ Khó mở rộng

VD: Bạn cần tích hợp module nhận dạng khuôn mặt AI, bạn mua module này từ hãng khác. Một thời gian sau, công ty họ đưa ra một module mới chạy nhanh hơn, hiệu quả hơn,… và bạn muốn nâng cấp lên, lúc đó bạn tích hợp module mới vào, và vấn đề xảy ra liên quan đến dữ liệu: đầu vào, kích thước, độ phân giải, kiểu dữ liệu sẽ khác… khi đó bạn cần phải chỉnh sửa lại code, làm mới các code… gây ra bất tiện.

+ Khi dùng dữ liệu chung thì độ bảo mật sẽ thấp.
+ …

4. Lập trình hướng đối tượng



- Hình dung chương trình như một môi trường, bạn tạo ra một môi trường, trong môi trường đó, bạn tạo ra các đối tượng (Object).
- Khi chương trình này chạy, các object này sẽ tương tác với nhau.

VD1: Một chương trình mô phỏng mạng lưới giao thông gồm các đối tượng: bản đồ, đường sá, các xe…

Khi "thả" các đối tượng này vào chương trình, mỗi đối tượng sẽ được lập trình một cách khác nhau, hành xử khác nhau (Mỗi xe có cách chạy nhanh – chậm, kích thước… khác nhau, xe này đụng vào xe khác [tương tác]). Mỗi object này sẽ hoạt động độc lập với nhau.

VD2: Trong game, các nhân vật (~object) sẽ hành xử khác nhau (chiến đấu…) độc lập nhau.

- Nếu dùng Kỹ thuật lập trình code các bài toán VD này, bạn sẽ phải tưởng tượng, xây dựng… nhưng sẽ khá/ rất vất vả.

- Nếu ta tạo các đối tượng một cách độc lập, riêng biệt, mỗi đối tượng có mỗi cách viết code khác nhau, sau đó thả vào trong một môi trường (một chương trình) và cho thực thi thì sẽ dễ dàng hơn.

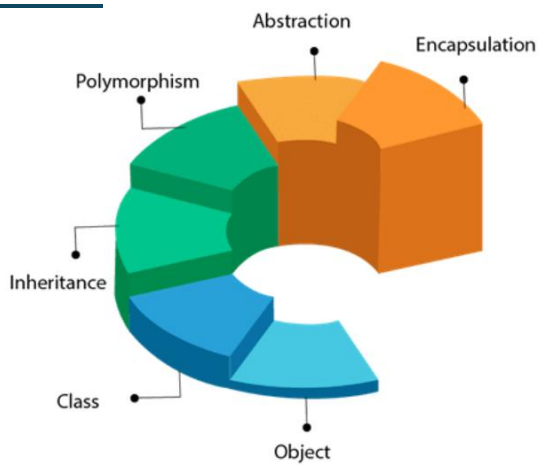
→ *Đây là tư duy về lập trình hướng đối tượng.*

- *Đặc tính chủ yếu:*

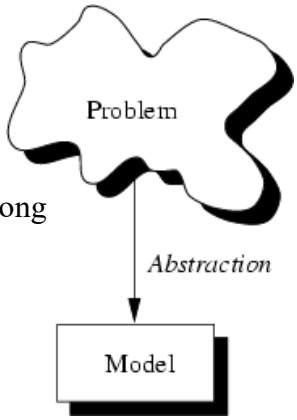
+ Chương trình được chia thành các đối tượng
+ Đặt trọng tâm vào đối tượng.
+ Dữ liệu gắn chặt với các hàm, được đóng gói, che dấu, bảo mật
+ Các đối tượng tương tác với nhau qua message (thông điệp)
+ Chương trình thiết kế theo hướng **bottom-up** (Xây dựng từ những đối tượng nhỏ, từ đó xây dựng lên thành một chương trình).

- Cách lập trình này hiệu quả, giúp giải quyết những bài toán lớn, ngoài OOP thì gần như không cách nào khác giải quyết được.

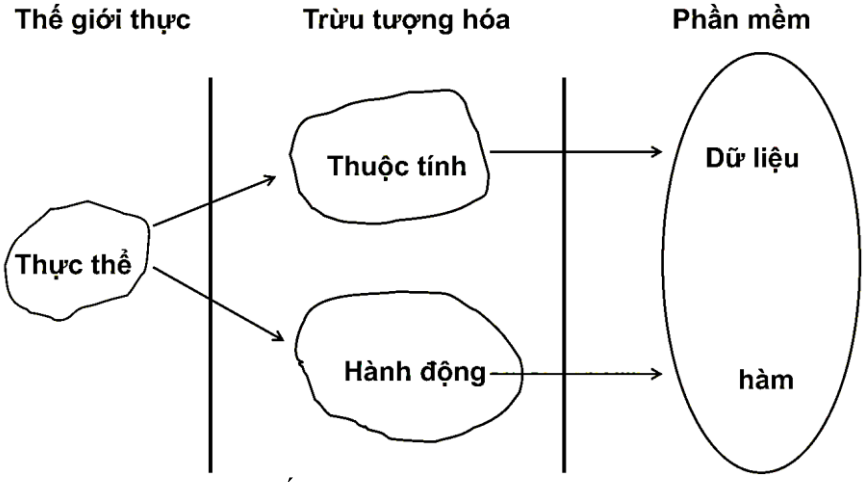
III. Các khái niệm cơ bản LTHĐT



1. Trừu tượng hóa (Abstraction)



- Là **sự đơn giản hóa, làm cho vấn đề dễ hiểu, sáng sủa hơn** *(làm giảm mức độ phức tạp của bài toán) (khác với nghĩa đời thường).*
- Cách nhìn ***khái quát hóa*** về một tập các đối tượng có chung các đặc điểm được ***quan tâm*** (và bỏ qua những chi tiết không cần thiết).
- Bài toán ngoài thế giới thực: phức tạp, gồ ghề, xấu xí → Mô hình hóa thành vấn đề trong máy tính: cắt gọt, sửa chữa,… thành "vuông vức".
- Che dấu chi tiết, làm nổi bật cái tổng thể.



- 02 loại trừu tượng hóa (2 dạng cắt gọt):

+ *Trừu tượng hóa dữ liệu*

. *Mô hình hóa các thuộc tính của lớp dựa trên thuộc tính của các đối tượng tương ứng.*

. **Ví dụ:** Hệ thống quản lý thông tin sinh viên trường:

1 sinh viên: tên, năm sinh, địa chỉ… → 1 SV ngoài thế giới thực được mô hình hóa trong không gian máy tính (thế giới ảo).

Nếu như lưu quá nhiều thông tin của một sinh viên (Đưa các thông tin dư thừa) → Dữ liệu phức tạp → Xử lý bài toán sẽ rất khó khăn.

+ *Trừu tượng hóa chương trình*

. *Mô hình hóa các phương thức của lớp dựa trên hành động của đối tượng.*

. Đơn giản hóa quy trình

VD: 1 sinh viên sau khi đăng ký học phần, tín chỉ có thể xem môn, giờ… đăng ký xét tốt nghiệp…

. Phân chia các chương trình thành các chương trình con, mỗi CTC sẽ có 1 cái tên gợi nhớ. Ở CT gọi (có thể là CT chính), ta chỉ thấy *lời gọi* các CTC thông qua các tên.

. Che dấu tất cả các lệnh cài đặt chi tiết trong CTC (CTC sẽ viết trong một file riêng)

**** Chương trình không trừu tượng hóa:***

```
int main () {
if ( ..... ) { ..... }else { ..... }
while( ..... ) { ..... }
printf ……
scanf ……
for ( ..... ) { ..... }
```

**** Chương trình trừu tượng hóa:***

```
int main () {
      Nhap (Lop);
      Xu Ly (Lop);
      Xuat (Lop);
      return 0;
}
```

- Tùy theo trường hợp, môi trường mà các nhà SX sẽ dùng các thông tin, dữ liệu nào cần thiết để mô hình hóa, cắt gọt chúng, với mục đích cuối cùng là làm cho bài toán đơn giản và dễ xử lý hơn.

Ví dụ:

+ *Hệ thống quản lý sinh viên cần các thông tin/ dữ liệu liên quan đến sinh viên:* Họ và tên, ngày tháng năm sinh, địa chỉ, CCCD, MSSV, … (1) mà không cần các thông tin khác, chẳng hạn mối quan hệ của sinh viên này với xã hội (bạn bè, người yêu, họ hàng…) (2)

→ *Đối với những thông tin (1) thì cần mô hình hóa, còn những thông tin (2) thì không cần.*

+ Ứng dụng mạng xã hội (Facebook), những thông tin như (1) thì những người tạo nên ứng dụng họ cảm thấy không đáng giá bằng những thông tin (2) (Mối quan hệ bạn bè…, sở thích về các lĩnh vực, sở trường…); họ không cần họ tên thật của người dùng (người dùng có thể dùng bất cứ tên nào, kể cả tên giả…).

2. Kiểu dữ liệu trừu tượng (ADT)

2.1 Data type (Kiểu dữ liệu)

- Tập hợp dữ liệu và các phép toán trên dữ liệu

Ví dụ kiểu int trong ngôn ngữ C:

+ Tập các giá trị: từ -32768 đến 32767 (2 bytes)
+ từ -2.147.483.648 đến 2.147.483.647 (4 bytes)
+ Tập các phép toán: +, -, *, /, %, ...

· Hai loại:

* *Kiểu dữ liệu sơ cấp:* giá trị dữ liệu của nó là đơn nhất. **Ví dụ:** int, float, char

* Kiểu dữ liệu có cấu trúc (cấu trúc dữ liệu): giá trị dữ liệu của nó là sự kết hợp của các giá trị khác. **Ví dụ:** mảng, chuỗi ký tự, cấu trúc, ...

- Tách biệt đặc tả các tính chất logic với cài đặt bên trong

Ví dụ: Cổng USB (Interface). Cắm thiết bị (Chuột, bàn phím) của bất kỳ hãng/ loại nào vào cổng USB đó. Muốn hệ thống nhận diện được, NSX phần cứng (Client) đã phải tuân theo một tiêu chuẩn nào đó của cổng USB (mạch điện…) (depends) để thiết bị hoạt động và truyền được "thông điệp" mà người sử dụng mong muốn (Inplementation satisfies Interface).

2.2 Abstract Data type (ADT)

- Kiểu dữ liệu chưa có sẵn mà phải mô hình hóa đối tượng mới có được.
- Kiểu dữ liệu mà biểu diễn bên trong của nó bị che dấu
- Bản chất tương tự kiểu dữ liệu của các NNLT (Họ và tên: kiểu chuỗi; Năm sinh: kiểu số nguyên…; Sinh viên: họ và tên, ngày tháng năm sinh, MSSV…)
- Client không được phép chỉnh sửa dữ liệu trực tiếp mà qua **tập các phép toán**
- Các phép toán chỉ được cho phép thông qua interface

- Ví dụ 1:

Stack ("last in first out" or LIFO).

+ Push: thêm phần tử vào stack
+ Pop: xóa phần tử khỏi stack
+ Initialize: khởi tạo stack
+ Empty: kiểm tra stack rỗng

Stack Interface (~ Khai báo nguyên mẫu hàm)	**Stack Implementation (~ Định nghĩa hàm)**	**Stack Client**

STACK.h

```
void STACKinit(void);
int  STACKisempty(void);
void STACKpush(int);
int  STACKpop(void);
```

stackarray.c

```
#include "STACK.h"
#define MAX_SIZE 1000
static int s[MAX_SIZE];
static int N;

void STACKinit(void) {
  N = 0;
}

int STACKisempty(void) {
  return N == 0;
}

void STACKpush(int item) {
  s[N++] = item;
}

int STACKpop(void) {
  return s[--N];
}
```

big enough?

posfix.c

```
#include <stdio.h>
#include <ctype.h>
#include "STACK.h"

int main(void) {
  int c;
  STACKinit();
  while ((c = getchar()) != EOF) {
    if ('+' == c)
      STACKpush(STACKpop() + STACKpop());
    else if ('*' == c)
      STACKpush(STACKpop() * STACKpop());
    else if (isdigit(c))
      STACKpush(c - '0');
  }

  printf("top of stack = %d\n", STACKpop());
  return 0;
}
```

pop 2 elements and push sum

convert char to integer and push

Chỉ cần dùng hàm đã tạo, không cần quan tâm phần cài đặt (tên biến, kiểu dữ liệu…). Nếu hàm chưa hiệu quả, có thể dùng hàm này trong file khác cũng được định nghĩa.

- Ví dụ 2: Chương trình quản lý sinh viên:

+ **Login** → Tạo ra đối tượng sinh viên (~ Đối tượng Avatar của bạn).
+ User sẽ *không thực hiện, thao tác trực tiếp* với hệ thống mà thông qua Avatar.
+ **Xem điểm:** Click vào 1 nút trên menu (~ Đối tượng giao diện: nút); Đối tượng giao diện đó sẽ gửi thông điệp cho Avatar của user

→ Đối tượng Avatar sẽ send message, truy vấn đến đối tượng liên quan đến database để lấy thông tin về điểm số

→ Database truy vấn thông tin…

→ Trả ngược thông tin lại cho Avatar. Nhưng ta không thể nhìn trực tiếp database mà Avatar mới vừa nhận, vì vậy cần thông qua một "report"

→ Avatar truy vấn đối tượng "report", khi đó "report" được sinh ra và truy vấn tiếp đến đối tượng giao diện

→ Thông tin điểm số được hiện lên màn hình.

=> Cách các object tương tác với nhau qua các thao tác chạy ngầm bên trong (Cách OOP tiếp cận).

+ Việc tương tác giữa các đối tượng với nhau được thực hiện một cách "kịch bản", sắp xếp trước.

+ Khi bạn click vào một đường link, nhấp vào một nút, nó sẽ được send message theo một kịch bản và có được kết quả như mong đợi.

+ Khi lập trình, client không quan tâm bạn dùng struct SV với kiểu dữ liệu nào, chỉ quan tâm đến các thuật toán, thao tác trên đó.

```
{/*Họ tên, ngày tháng năm sinh,…*/};
```

- Các đối tượng được dẫn xuất từ các ADT
- Chương trình hoạt động dựa trên cơ chế các đối tượng tương tác bằng cách gửi thông điệp cho nhau

3. Đối tượng (object)

- Là khái niệm trừu tượng *phản ánh các thực thể trong thế giới thực*
- **ADT là "khuôn", "đúc ra" các object.**

ADT Sinhvien tạo ra nhiều đối tượng sinh viên khác nhau, object của SV này sẽ khác với SV kia (điểm bạn này khác với điểm bạn kia…)

- **Object = data (attributes) + methods (behavior)**

** Được xác định bằng ba yếu tố:*

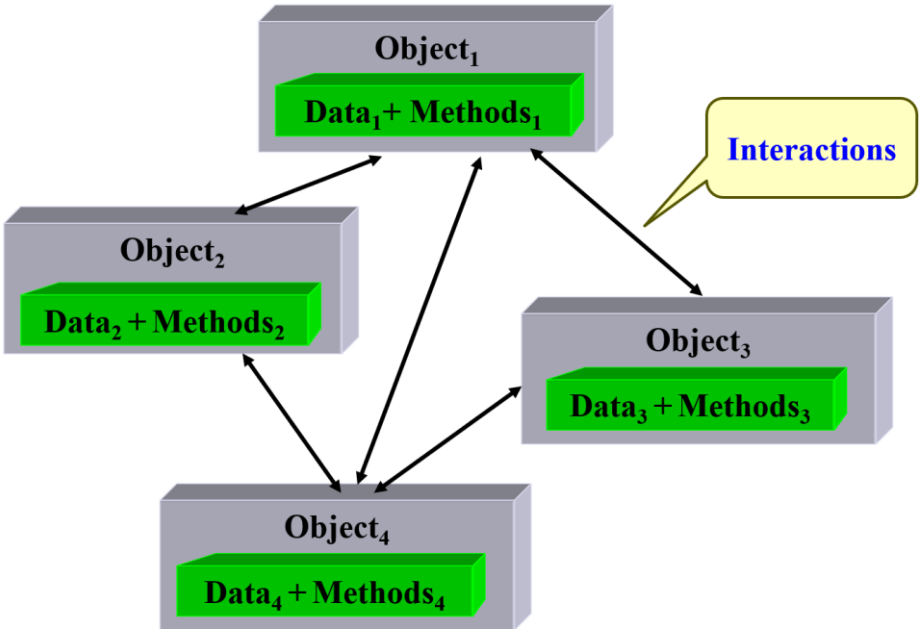
o *Định danh đối tượng:* xác định duy nhất cho mỗi đối tượng trong hệ thống, nhằm phân biệt các đối tượng với nhau;

o *Trạng thái của đối tượng:* sự tổ hợp của các giá trị của các thuộc tính mà đối tượng đang có;

o *Hoạt động của đối tượng:* là các hành động mà đối tượng có khả năng thực hiện được.

	Trạng thái	Hành động
Chiếc xe	Nhãn hiệu: "Ford" Màu sơn: Trắng Giá bán: 5000$	Khởi động Dừng lại Chạy

- Mỗi đối tượng bất kể đang ở trạng thái nào đều có định danh và được đối xử như một thực thể riêng biệt:
  + Mỗi đối tượng có một handle (trong C++ là địa chỉ);
  + Hai đối tượng có thể có giá trị giống nhau nhưng handle khác nhau.
- Là thể hiện (instance) của 1 lớp (Đại diện Lớp là Đối tượng)
- Các đối tượng là thể hiện của cùng 1 lớp có dữ liệu ở trạng thái khác nhau



Ví dụ 1: thiết kế và xây dựng chương trình game hockey

+ Object: Hockey player
+ Attributes: Position, height, weight, salary,…
+ Behaviors: shoot, punch another player,…

Ví dụ 2: Viết một chương trình mô phỏng sự phát triển của quần thể virus trong cơ thể con người theo thời gian. Mỗi tế bào virus có thể sinh sản trong một khoảng thời gian. Bệnh nhân có thể trải qua điều trị bằng thuốc để ức chế quá trình sinh sản, và hủy diệt các tế bào virus từ cơ thể của họ. Tuy nhiên, một số tế bào có khả năng kháng thuốc và có thể tồn tại.

Bệnh nhân	Virus
- *Attributes* + Quần thể virus + Miễn dịch với virus (%) - *Behaviors* + Dùng thuốc	- *Attributes* + Tỉ lệ sinh sản (%) + Kháng thuốc (%) - *Behaviors* + Sinh sản + Tồn tại

4. Thuộc tính & phương thức (Data & Methods)

- Một đối tượng là 1 thực thể bao gồm thuộc tính và hành động (phương thức)
- Thuộc tính (~Data):
    + Dữ liệu trình bày các đặc điểm về một đối tượng;
    + Đặc trưng, phân biệt đối tượng
    + Là dữ liệu được lưu trữ trong đối tượng
    + Nên được đặt ở *mức cao nhất trong phân cấp kế thừa*
    + Hằng, biến, tham số…
    + Thuộc kiểu dữ liệu cơ bản hoặc kiểu do người dùng định nghĩa

- Phương thức (~Cách hoạt động):
    + Có liên quan tới những thứ mà đối tượng có thể làm.
    + Một phương thức đáp ứng một chức năng tác động lên dữ liệu của đối tượng (thuộc tính).
    + Hàm nội tại được ứng dụng cho đối tượng (Hàm thành viên) (Hàm tự xử lý bên trong/ Interface).
    + Thao tác trên dữ liệu được lưu trữ trong đối tượng
    + Operations, behaviors, member functions

5. Lớp (class)

- Đối tượng là một thực thể cụ thể, tồn tại trong hệ thống.
- Lớp là một khái niệm trừu tượng, dùng để chỉ một tập hợp các đối tượng có mặt trong hệ thống.
- Là một *tập các objects cùng loại*

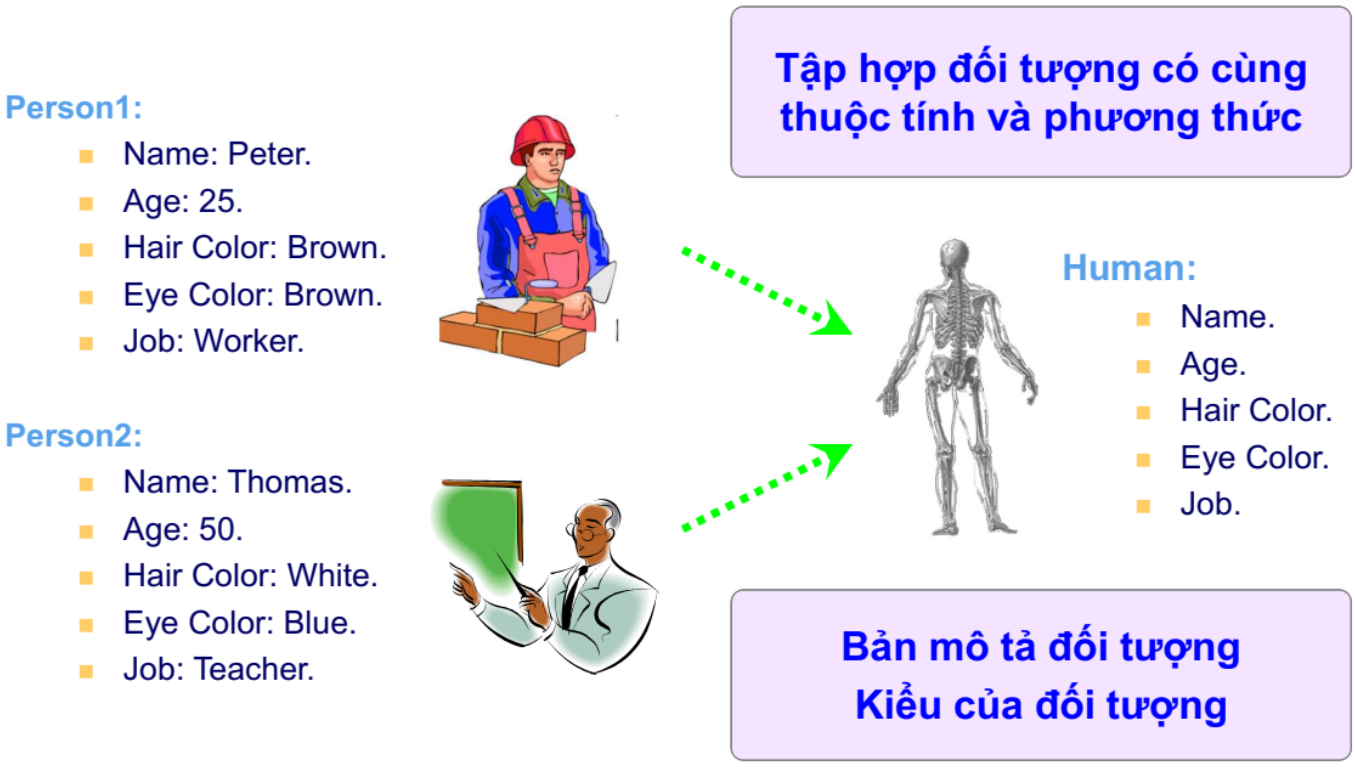
    Các đối tượng sẽ có các loại khác nhau, đặc điểm, … khác nhau và được phân vào các nhóm. Các nhóm này được gọi là class.

    VD: * lớp SV tạo đối tượng SV, lớp GV tạo đối tượng GV.

    * Mỗi chiếc xe có trong cửa hàng là một đối tượng, nhưng khái niệm "xe hơi" là một lớp đối tượng dùng để chỉ tất cả các loại xe có trong cửa hàng.
- *Là một đại diện của ADT (ADT > Lớp)*

    Trong OOP, ADT chính là lớp. Nhưng không có nghĩa là ADT chỉ cài đặt theo hướng OOP.
- Được sử dụng như *kiểu dữ liệu do người dùng định nghĩa*
- Là bản mẫu để tạo ra thể hiện của object

    Lớp là khuôn, object là sản phẩm được đúc ra từ khuôn. Khi lập trình, ta không lập trình sản phẩm, mà ta lập trình cho khuôn. Khi chạy chương trình, khuôn bắt đầu mới làm việc và tạo ra các đối tượng.

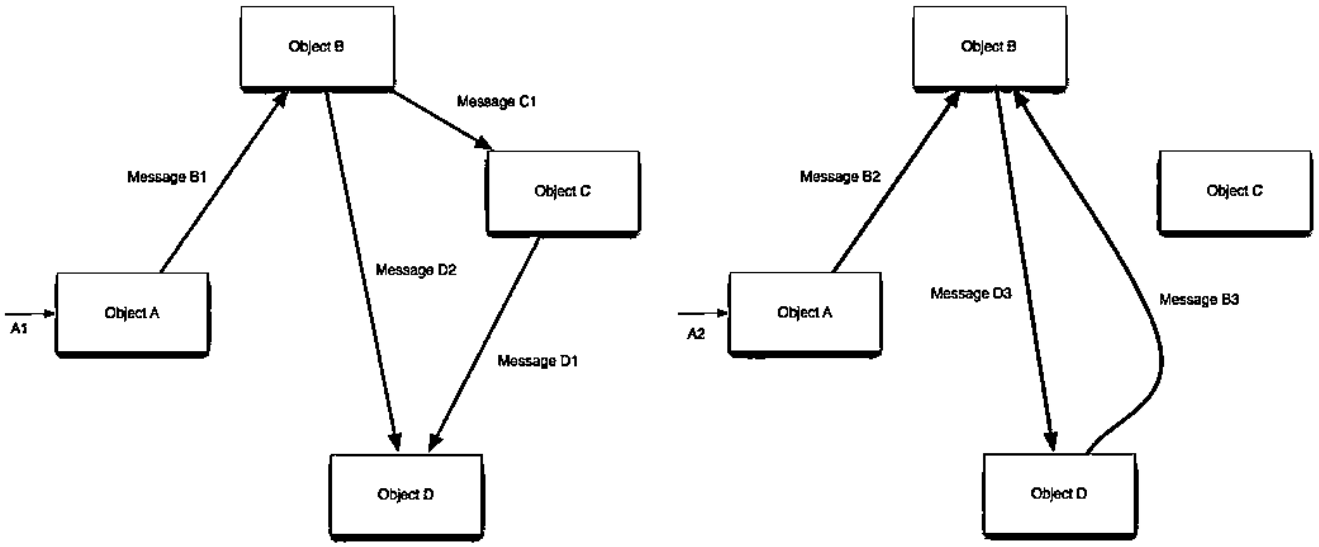


- Do *đối tượng lúc thực thi chương trình mới được tạo ra,* nên bộ nhớ được cấp phát cho object, không phải cho class. Class này được thể hiện khi biên dịch chương trình. *Nếu tạo nhiều đối tượng thì bộ nhớ bị chiếm dụng lớn, và ngược lại.*

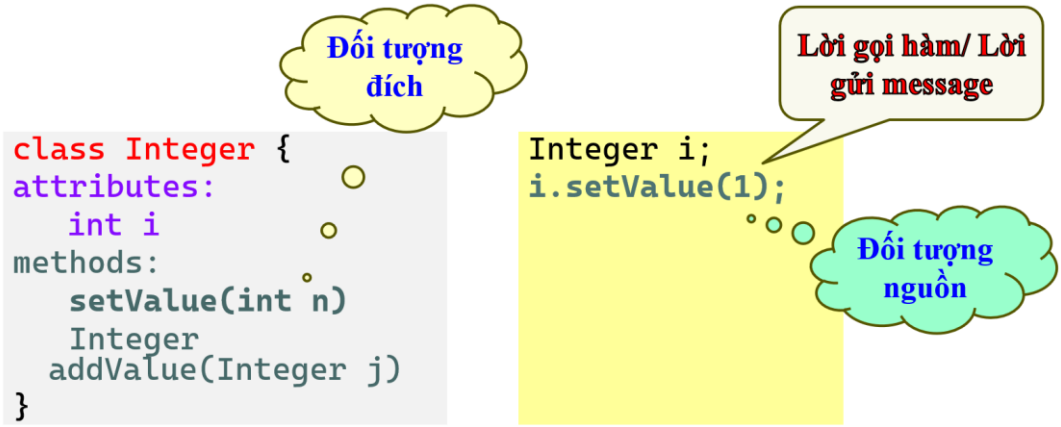
- Một lớp có thể có một trong các khả năng sau:
  + Hoặc chỉ có thuộc tính, không có phương thức;
  + Hoặc chỉ có phương thức, không có thuộc tính;
  + Hoặc có cả thuộc tính, phương thức (phổ biến);
  + Đặc biệt, lớp không có thuộc tính, phương thức nào, gọi là lớp trừu tượng, các lớp này không có đối tượng.
- Lớp và đối tượng mặc dù có mối liên hệ tương ứng lẫn nhau, nhưng lại khác nhau về bản chất.

Lớp	Đối tượng
Sự trừu tượng hóa của đối tượng	Là thể hiện của lớp
Là một khái niệm trừu tượng, chỉ tồn tại ở dạng khái niệm mô tả đặc tính chung của một số đối tượng	Là một thực thể cụ thể, có thực, tồn tại trong bộ nhớ
Là nguyên mẫu cho các đối tượng, xác định các hành vi và các thuộc tính cần thiết cho một nhóm các đối tượng cụ thể	Tất cả các đối tượng thuộc cùng một lớp có cùng các thuộc tính và hành động

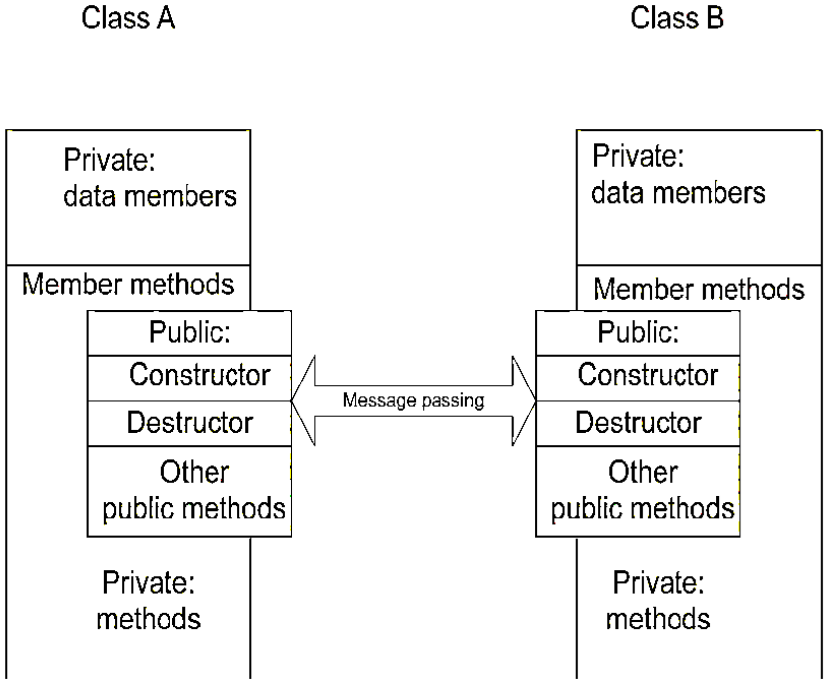
6. Thông điệp (Message)



- Là phương tiện để đối tượng này chuyển yêu cầu tới đối tượng khác.
- Các đối tượng tương tác, giao tiếp với nhau sử dụng thông điệp (message).
- ***Hiểu đơn giản, thông điệp là một lời gọi hàm.***
- ***Message bao gồm:*** Đối tượng đích, Tên của phương thức cần thực hiện, Các tham số cần thiết



- Truyền thông điệp: Là cách một đối tượng triệu gọi một hay nhiều phương thức của đối tượng khác để yêu cầu thông tin;
- Hệ thống yêu cầu đối tượng thực hiện phương thức:
  + Gửi thông báo và tham số cho đối tượng;
  + Kiểm tra tính hợp lệ của thông báo;
  + Gọi thực hiện hàm tương ứng phương thức.



7. Sơ đồ đối tượng

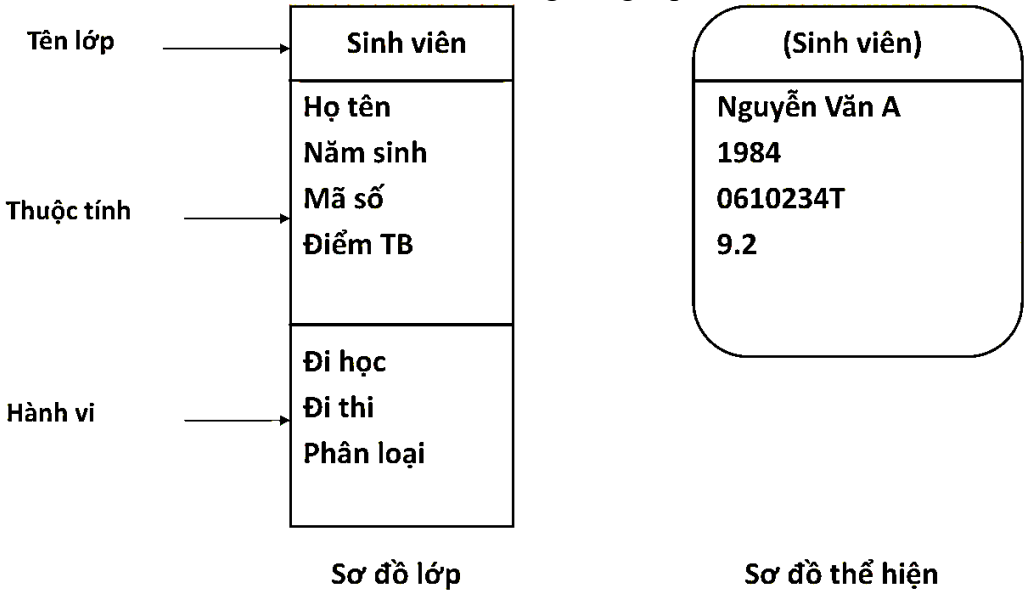
- Ta dùng sơ đồ đối tượng để mô tả các lớp đối tượng. Sơ đồ đối tượng bao gồm sơ đồ lớp và sơ đồ thể hiện.

+ Sơ đồ lớp mô tả các lớp đối tượng trong hệ thống, một lớp đối tượng được diễn tả bằng một hình chữ nhật gồm 3 phần:

. Phần đầu chỉ tên lớp

. Phần 2 mô tả các thuộc tính

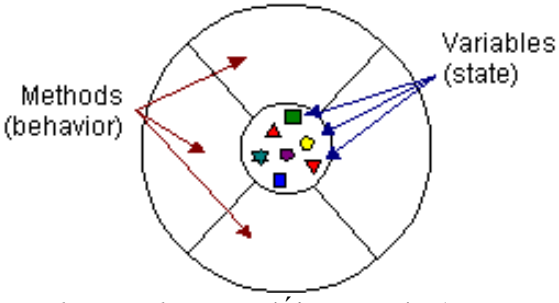
. Phần 3 mô tả các thao tác của các đối tượng trong lớp



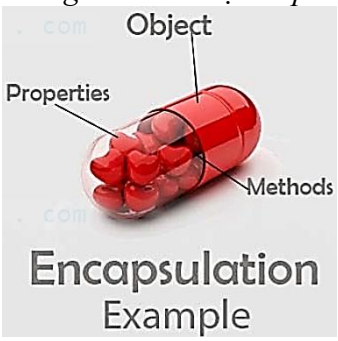
Sơ đồ lớp Sơ đồ thể hiện

8. Đóng gói, kế thừa, đa hình

8.1 Tính đóng gói (Tính bao đóng) (Encapsulation)



- Là việc bao gói các thuộc tính của đối tượng bởi các phương thức
- "Thuộc tính của ai người đó giữ, dùng" → Độc lập giữa các đối tượng (Máy tính của ai thì người đó sử dụng, dùng chung gây sự rắc rối).
- Thực tế đôi khi cần phải "phơi bày" một vài thuộc tính và "che dấu một vài phương thức".
- Muốn sử dụng các thuộc tính của đối tượng thì phải *thông qua những phương thức* của đối tượng đó (**VD:** get, set…)
- Che dấu thông tin: việc sử dụng đối tượng *không cần thiết phải biết chi tiết về dữ liệu và toán tử bên trong (Do không thao tác trực tiếp đến đối tượng);* Ngăn chặn các thao tác không được phép từ bên ngoài



- Ưu điểm: Cô lập được bài toán, xử lý nội bộ nếu có lỗi, Bảo vệ dữ liệu, Kiểm soát được sự thay đổi

+ Cho phép che dấu sự cài đặt chi tiết bên trong:

. Chỉ cần gọi các phương thức theo một cách thống nhất;

. Phương thức có thể cài đặt khác nhau cho các trường hợp khác nhau.

+ Cho phép che dấu dữ liệu bên trong đối tượng:

. Khi sử dụng, không biết thực sự bên trong đối tượng có những gì;

. Chỉ thấy được những gì đối tượng cho phép truy nhập vào.

+ Cho phép tối đa hạn chế việc sửa lại mã chương trình.

8.2 Các mối quan hệ của lớp (Relationship)

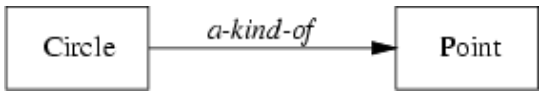
https://www.desy.de/gna/html/cc/Tutorial/node6.htm

a/ Quan hệ A-Kind-Of

Bài toán: Viết chương trình vẽ các đối tượng points, circles, rectangles, triangles,…

class Point { **attributes:** **int x, y** **methods:** **setX(int newX)** **getX()** **setY(int newY)** **getY()** **}**	**class Circle {** **attributes:** **int x, y, radius** **methods:** **setX(int newX)** **getX()** **setY(int newY)** **getY()** **setRadius(newRadius)** **getRadius()** **}**

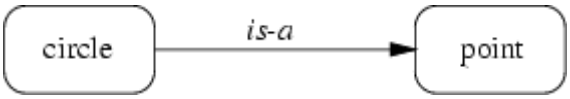
- Cả hai lớp đều có hai phần tử dữ liệu x và y. Trong lớp Point, các phần tử này mô tả vị trí của điểm, trong trường hợp lớp Circle, chúng mô tả tâm của đường tròn. Do đó, x và y có cùng ý nghĩa trong cả hai lớp: Chúng mô tả *vị trí của đối tượng liên quan bằng cách xác định một điểm.*
- Cả hai lớp đều cung cấp cùng một bộ phương thức để lấy và đặt giá trị của hai phần tử dữ liệu x và y.
- Lớp Circle thêm phần tử dữ liệu mới "radius" và các phương thức truy cập tương ứng.
- Biết được các thuộc tính của lớp Point chúng ta có thể mô tả một đường tròn như là một điểm + với bán kính và các phương thức để truy cập nó. Như vậy, đường tròn là "một loại" (a-kind-of) điểm. Tuy nhiên, hình tròn có phần riêng biệt hơn.



Circle is a-kind-of Point.

b/ Quan hệ Is-A

- Mối quan hệ ở trên được sử dụng *ở cấp độ lớp* để mô tả *mối quan hệ giữa hai lớp* tương tự. Nếu chúng ta *tạo các đối tượng của hai lớp như vậy thì chúng ta gọi mối quan hệ của chúng là mối quan hệ "is-a".*
- Vì lớp Circle là một loại của lớp Point, nên một thể hiện của Circle, chẳng hạn như đường tròn (acircle), là một điểm. Do đó, mỗi vòng tròn hoạt động như một điểm.
- Ví dụ: bạn có thể di chuyển các điểm theo hướng x bằng cách thay đổi giá trị của x. Tương tự, bạn di chuyển vòng tròn theo hướng này bằng cách thay đổi giá trị x của chúng.
- Ở hình dưới, các đối tượng được vẽ bằng các hình chữ nhật có góc tròn. Tên của chúng chỉ bao gồm các chữ cái viết thường.



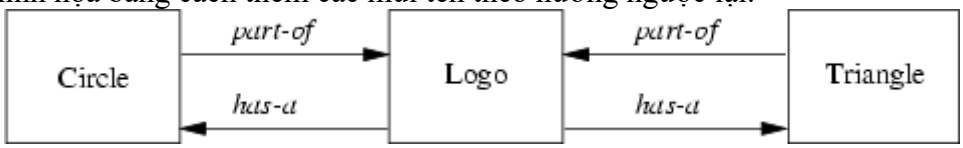
c/ Mối quan hệ Part-Of

- Đôi khi bạn cần có khả năng xây dựng các đối tượng bằng cách kết hợp chúng với những đối tượng khác. - Bạn đã tạo một số lớp cho các số liệu có sẵn. Bây giờ bạn quyết định rằng bạn muốn có một hình đặc biệt đại diện cho logo của riêng bạn, bao gồm hình tròn và hình tam giác. (Giả sử rằng bạn đã xác định một lớp Tam giác.)
- Do đó, logo của bạn bao gồm hai phần hoặc hình tròn và hình tam giác là một phần *(part-of)* của logo của bạn:

class Logo { **attributes:** **Circle circle** **Triangle triangle** **methods:** **set(Point where)** **}**	

d/ Mối quan hệ Has-A

Mối quan hệ này chỉ là sự nghịch đảo của mối quan hệ *part-of*. Vì vậy, chúng ta có thể dễ dàng thêm mối quan hệ này vào phần minh họa bằng cách thêm các mũi tên theo hướng ngược lại.

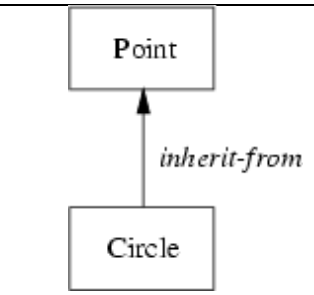


8.3 Tính kế thừa (Inheritance) (Chương V)

- Khái niệm:
  - + Khả năng cho phép xây dựng lớp mới được thừa hưởng các thuộc tính của lớp đã có;
  - + Các phương thức & thuộc tính được định nghĩa trong một lớp có thể được sử dụng lại bởi lớp khác.
- Đặc điểm:
  - + Lớp nhận được có thể bổ sung các thành phần;
  - + Hoặc định nghĩa là các thuộc tính của lớp cha.
  - + Cung cấp cơ chế rất mạnh trong việc tổ chức và cấu trúc chương trình

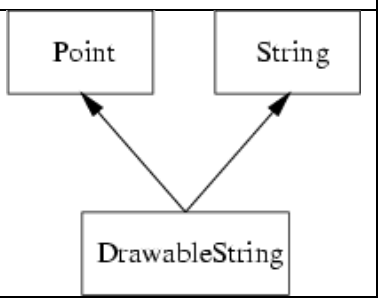
class Circle inherits from Point {
attributes:
int radius

methods:
setRadius(int newRadius)
getRadius()
}



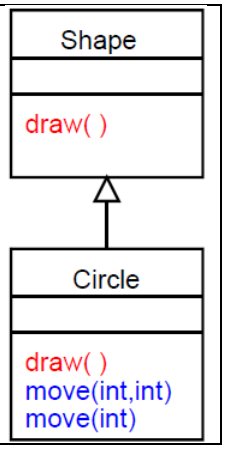
- Các loại kế thừa:
  + Đơn kế thừa
  + Đa kế thừa

- Lợi ích:
+ Tái sử dụng
+ Có thể cài đặt lớp cha là lớp trừu tượng



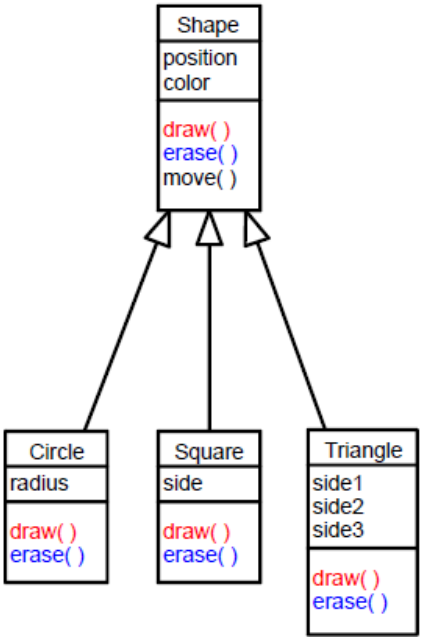
**** Lớp trừu tượng:***

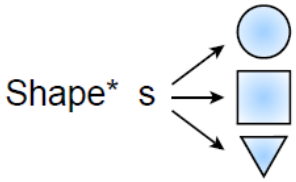
Định nghĩa:
- Class A được gọi là lớp trừu tượng nếu nó *chỉ được sử dụng như là SuperClass* của lớp khác
- Các thành phần trong class được chỉ định nhưng chưa được định nghĩa đầy đủ
- *Không được dùng để tạo ra đối tượng*



8.4 Tính đa hình (Polymorphism) (Chương VI)

- "Nhiều hình thức" – Cùng 1 code nhưng tùy ngữ cảnh nó sẽ chạy khác nhau.
- Các phương thức cùng tên thực hiện khác nhau với các đối tượng/lớp khác nhau
- Được thực hiện bởi: Quá tải, Nạp chồng

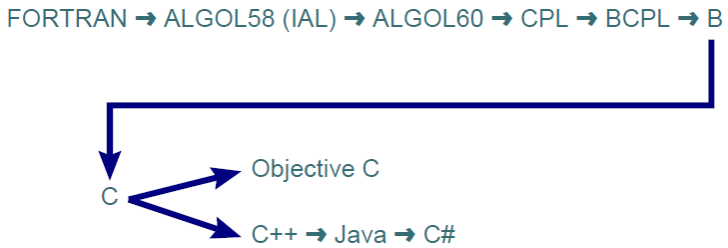




9. Ưu điểm của LTHĐT

- Cho phép phát triển các hệ thống lớn dễ dàng hơn
- Hợp nhất quy trình thiết kế và cài đặt mã
- Chia nhỏ code thành các phần có tính logic, loại bỏ các code lặp
- Hỗ trợ tái sử dụng: off-the-shelf code libraries
- Hỗ trợ sự phát triển code theo thời gian: code bên trong của một lớp có thể được viết lại mà không ảnh hưởng đến toàn hệ thống nếu interfaces được đảm bảo.

10. Các ngôn ngữ LTHĐT



Ngôn ngữ OOP thuần khiết: Smalltalk, C#, Java,…
Ngôn ngữ lai hỗ trợ OOP: C++, Objective-C, Object-Pascal, Delphi,…