1. **Lập trình hướng đối tượng là gì?**

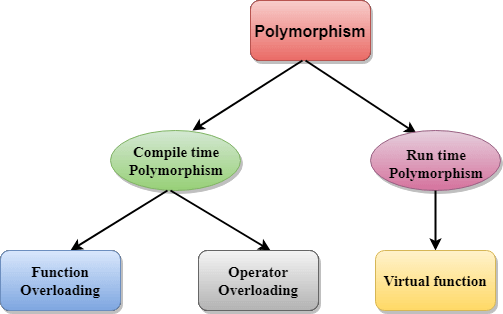
Là phương pháp lập trình *lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình.*

Lập trình hướng đối tượng là phương pháp lập trình dựa trên kiến trúc lớp (class) và đối tượng (object).

1. **Trình bày các đặc điểm quan trọng của lập trình hướng đối tượng**

* Trừu tượng hóa – Abstraction 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).
* Đóng gói – Encapsulation Nhóm những gì có liên quan với nhau vào làm một, để sau này có thể dùng một cái tên để gọi đến. Vd: các hàm/ thủ tục đóng gói các câu lệnh, các đối tượng đóng gói dữ liệu của chúng và các thủ tục có liên quan.
* Thừa kế - Inheritance cho phép một lớp D có được các thuộc tính và thao tác của lớp C, như thể các thuộc tính và thao tác đó đã được định nghĩa tại lớp D. Cho phép cài đặt nhiều quan hệ giữa các đối tượng: Đặc biệt hóa – Tổng quát hóa

- Đa hình – Polymorphism Là cơ chế cho phép một tên thao tác hoặc thuộc tính có thể được định nghĩa tại nhiều lớp và có thể có nhiều cài đặt khác nhau tại mỗi lớp trong các lớp đó.**-**Xảy ra khi có sự kế thừa giữa các lớp



* **Các Ưu điểm của OOP**
  + Nguyên lý kế thừa: tránh lặp, tái sử dụng.
  + Nguyên lý đóng gói – che dấu thông tin: chương trình an toàn không bị thay đổi bới những đoạn chương trình khác
  + Dễ mở rộng, nâng cấp
  + Mô phỏng thế giới thực tốt hơn.

1. **Đối tượng là gì?**

Một đối tượng là 1 thực thể bao gồm thuộc tính và hành động

Ví dụ đối tượng: Một người

* + Một người có các thuộc tính: *tên, tuổi, địa chỉ, màu mắt…*
  + Các hành động: *đi, nói, thở…*

**Lớp đối tượng là gì?**

* + Các đối tượng có các đặc tính tương tự nhau được gom chung thành lớp đối tượng. Một lớp đối tượng đặc trưng bằng các thuộc tính, và các hành động (hành vi, thao tác).
  + Thuộc tính (Attribute): Một thành phần của đối tượng, có giá trị nhất định cho mỗi đối tượng tại mỗi thời điểm trong hệ thống.
  + Thao tác (Operation): Thể hiện hành vi của một đối tượng tác động qua lại với các đối tượng khác hoặc với chính nó.

1. **Phân biệt các phạm vi truy xuất private, protected, public? và cho ví dụ minh họa**

Public: các thuộc tính hoặc các phương thức có thể truy xuất ra bên ngoài class

Protected: các thuộc tính các phương thức không thể truy xuất ra bên ngoài class, nó chỉ được gọi trong class và class kế thừa

Private: các thuộc tính các phương thức không thể truy xuất ra bên ngoài class, nó chỉ được gọi trong phạm vi class

**Ví dụ**:

class TamGiac{

private:

float a,b,c;*/\*độ dài ba cạnh\*/*

public:

void Nhap();*/\*nhập vào độ dài ba cạnh\*/*

void In();*/\*in ra các thông tin liên quan đến tam giác\*/*

private:

int Loaitg();*//cho biết kiểu của tam giác: 1-d,2-vc,3-c,4-v,5-t*

float DienTich();*/\*tính diện tích của tam giác\*/*

};

1. **Constructor là gì? Constructor mặc định? Phân biệt Constructor mặc định và Constructor khác**

**Constructor**

* Phương thức thiết lập hay còn gọi là constructor, là một loại phương thức đặc biệt dùng để khởi tạo thể hiện của lớp.
* Bất kỳ một đối tượng nào được khai báo đều phải sử dụng một hàm thiết lập để khởi tạo các giá trị thành phần của đối tượng.
* Hàm thiết lập được khai báo giống như một phương thức với tên phương thức trùng với tên lớp và không có giá trị trả về (kể cả void).
* Constructor phải có thuộc tính public
* Constructor có thể được khai báo chồng như các hàm C++ thông thường khác hay không?
* Constructor có thể được khai báo với các tham số có giá trị ngầm định
* **Constructor mặc định**
* Constructor mặc định (default constructor) là constructor được gọi khi thể hiện được khai báo mà không có đối số nào được cung cấp

+ MyClass x = MyClass;

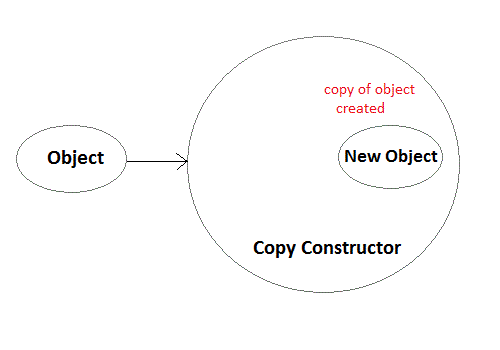
+ MyClass\* p = new MyClass;

* Ngược lại, nếu tham số được cung cấp tại khai báo thể hiện, trình biên dịch sẽ gọi constructor khác (overload)

+ MyClass x(5);

+ MyClass\* p = new MyClass(5);

* Đối với constructor mặc định, nếu ta không cung cấp bất kỳ constructor nào, C++ sẽ tự sinh constructor mặc định là một phương thức rỗng.
* Tuy nhiên, nếu ta không định nghĩa constructor mặc định nhưng lại có các constructor khác, trình biên dịch sẽ báo lỗi không tìm thấy constructor mặc định nếu ta không cung cấp tham số khi tạo thể hiện.
* **Copy Constructor**
* Chúng ta có thể **tạo đối tượng mới giống đối tượng cũ** một số đặc điểm, không phải hoàn toàn như phép gán bình thường, hình thức “giống nhau” được định nghĩa theo quan niệm của người lập trình. Để làm được vấn đề này, trong các ngôn ngữ OOP cho phép ta xây dựng **phương thức thiết lập sao chép**.
* Đây là phương thức thiết lập có tham số là tham chiếu đến đối tượng thuộc chính lớp này.
* Trong phương thức thiết lập sao chép có thể ta chỉ sử dụng một số thành phần nào đó của đối tượng ta tham chiếu 🡪 “gần giống nhau”





1. **Destructor**

* Phương thức hủy bỏ hay còn gọi là destructor, được gọi ngay trước khi một đối tượng bị thu hồi.
* Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.
* Một lớp chỉ có duy nhất một Destructor
* Cú pháp:
  + Phương thức **Destructor** có tên trùng tên với tên lớp và có dấu **~** đặt trước
  + Không có giá trị trả về
* Được **tự động gọi** thực hiện khi đối tượng hết phạm vi sử dụng.
* Destructor phải có thuộc tính **public**



1. **Kế thừa** là một đặc điểm của ngôn ngữ dùng để biểu diễn mối quan hệ đặc biệt hóa – tổng quát hóa giữa các lớp. Các lớp được trừu tượng hóa và được tổ chức thành một sơ đồ phân cấp lớp.

**Sự kế thừa** là một mức cao hơn của trừu tượng hóa, cung cấp một cơ chế gom chung các lớp có liên quan với nhau thành một mức khái quát hóa đặc trưng cho toàn bộ các lớp nói trên.

Các loại kế thừa: Kế thừa ảo, kế thừa phân cấp (Hierarchical Inheritance) , kế thừa đa cấp (Multi-level inheritance) , đơn kế thừa (Single inheritance), đa kế thừa (Multiple inheritance)

Quan hệ: 1-1 (một - một), 1-n (một-nhiều) , n-n (nhiều-nhiều), đặc biệt hoá- tổng quát hoá.

**Ưu điểm** của kế thừa:

* Kế thừa cho phép xây dựng lớp mới từ lớp đã có.
* Kế thừa cho phép tổ chức các lớp chia sẻ mã chương trình chung, nhờ vậy có thể dễ dàng sửa chữa, nâng cấp hệ thống.
* Trong C++, kế thừa còn định nghĩa sự tương thích, nhờ đó ta có cơ chế chuyển kiểu tự động.

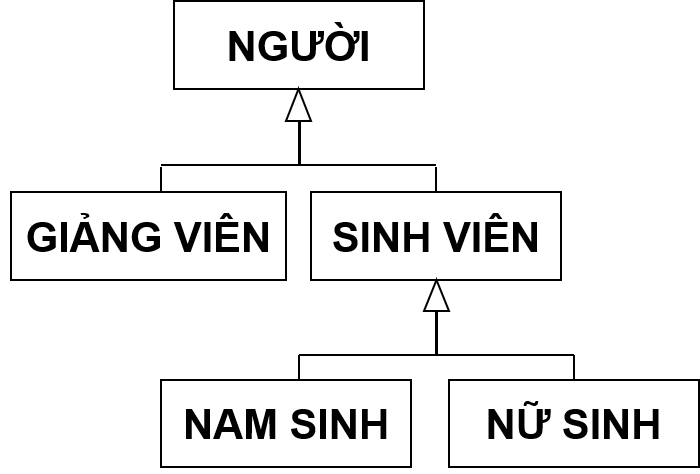
a.Mối quan hệ liên kết Has-A: quan hệ một một (1-1), quan hệmột nhiều (1-n), quan hệ

nhiều nhiều (n-n).

b.Mối quan hệ kế thừa Is-A: quan hệ đặc biệt hóa –tổng quát hóa

**Ví dụ minh họa**:

* Xét hai khái niệm Người và Sinh viên với mối quan hệ tự nhiên: *Một Sinh viên là một Người*. Trong C++, ta có thể biểu diễn khái niệm trên, một sinh viên là một người có thêm một số thông tin và một số thao tác (riêng biệt của sinh viên).
* Như vậy, ta tổ chức lớp Sinh viên kế thừa từ lớp Người.
* Ta có thể tổ chức hai lớp Nam sinh và Nữ sinh là hai lớp con (lớp dẫn xuất) của lớp Sinh viên. Trường hợp này, lớp Sinh viên trở thành lớp cha (lớp cơ sở) của hai lớp trên.



* Khai báo

class SinhVien : public Nguoi {

//...

};

* + Cho biết lớp Sinh viên kế thừa từ lớp Người. Khi đó Sinh viên *thừa hưởng các đặc tính* của lớp Người.
* Về mặt dữ liệu: Mỗi đối tượng Sinh viên *tự động có* thành phần dữ liệu họ tên, năm sinh của người.
* Về mặt thao tác: Lớp Sinh viên được tự động kế thừa các thao tác của lớp cha. Đây chính là khả năng sử dụng lại mã chương trình.
* Riêng phương thức thiết lập không được kế thừa.
* Khả năng thừa hưởng các thao tác của lớp cơ sở có thể được truyền qua “vô hạn mức”.

1. **Phân biệt các kiểu kế thừa private, protected, public**

|  |  |  |  |
| --- | --- | --- | --- |
| Từ khóa dẫn xuất |  |  |  |
| Ph  ạ  m v  i    t  ruy  c  ậ  p | Private | Protected | public |
| Private | X | X | X |
| Protected | Private | Protected | Protected |
| Public | Private | Protected | Public |

* Thành phần private ở lớp cha thì không truy xuất được ở lớp con
* Kế thừa public: Lớp con kế thừa public từ lớp cha thì các thành phần protected của lớp cha trở thành protected của lớp con, các thành phần public của lớp cha trở thành public của lớp con.
* Kế thừa private: Lớp con kế thừa private từ lớp cha thì các thành phần protected va public của lớp cha trở thành private của lớp con.
* Kế thừa protected: Lớp con kế thừa protected từ lớp cha thì các thành phần protected va public của lớp cha trở thành protected của lớp con.

1. **Hàm thuần ảo là gì? Lớp trừu tượng là gì? Cho ví dụ minh họa**.

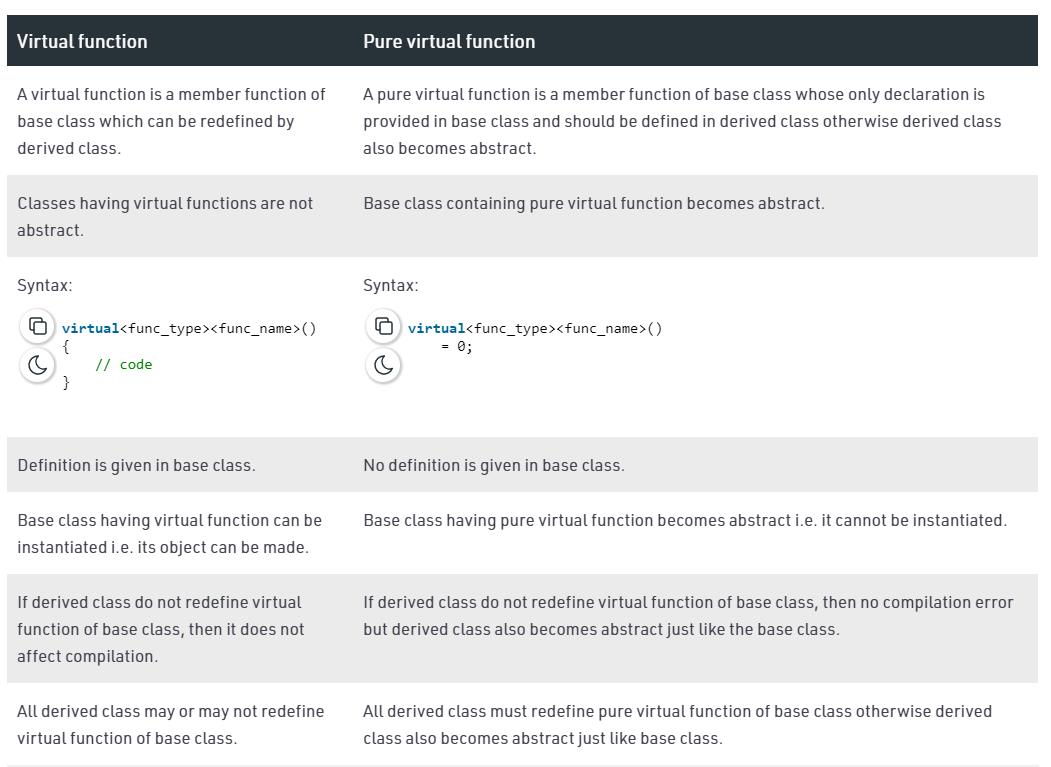
Hàm thuần ảo (Phương thức ảo thuần tuý) có ý nghĩa cho việc tổ chức sơ đồ phân cấp các lớp, nó đóng vai trò chừa sẵn chỗ trống cho các lớp con điền vào với phiên bản phù hợp. Phương thức ảo thuần tuý là phương thức ảo không có nội dung, được khai báo với từ khóa virtual và được gán giá trị =0

Khi lớp có phương thức ảo thuần tuý, lớp trở thành lớp cơ sở trừu tượng.

Lớp cơ sở trừu tượng không có đối tượng nào thuộc chính nó, không thể tạo đối tượng.

Phương thức ảo thuần tuý là phương thức ảo không có nội dung.

- Khác nhau giữa ảo và thuần ảo:



class Shape //Abstract

{

public :

//Pure virtual Function

virtual void draw() = 0;

}

Trong ví dụ trên, các hàm thành phần trong lớp Shape là phương thức ảo thuần tuý và lớp Shape là lớp cơ sở trừu tượng. Nó bảo đảm không thể tạo được đối tượng thuộc lớp Shape.

1. **Phân biệt khái niệm overload (tải chồng) và override (ghi đè)**

|  |  |  |
| --- | --- | --- |
|  | **Override** | **Overload** |
| Khái niệm | Là một tính năng cho phép một lớp con hoặc lớp con cung cấp một triển khai cụ thể của một phương thức đã được cung cấp bởi một trong các lớp siêu hoặc các lớp cha của nó. Nói cách khác, nếu lớp con cung cấp trình triển khai cụ thể của phương thức mà đã được cung cấp bởi một trong các lớp cha của nó, thì đó là ghi đè phương thức. | Nạp chồng phương thức đơn giản là có vài phương thức trùng tên nhưng khác nhau về đối số. Cài chồng phương thức cho phép ta tạo nhiều phiên bản của một phương thức, mỗi phiên bản chấp nhận một danh sách đối số khác nhau, nhằm tạo thuận lợi cho việc gọi phương thức. |
| Hành vi | Thay đổi hành vi hiện tại của phương thức. | Thêm hoặc mở rộng cho hành vi của phương thức. |
| Đa hình | Thể hiện tính đa hình tại run time. | Thể hiện tính đa hình tại compile time. |
| Danh sách tham số | Danh sách tham số phải giống nhau. | Danh sách tham số khác nhau (số lượng, thứ tự, kiểu dữ liệu) |
| Giá trị trả về | Kiểu trả về bắt buộc phải giống nhau. | Kiểu trả về có thể khác nhau. |
| Phạm vi | Xảy ra giữa 2 class có quan hệ kế thừa | Xảy ra trong phạm vi cùng 1 class. |

1. **Phương thức get/set**

**Phương thức truy vấn**

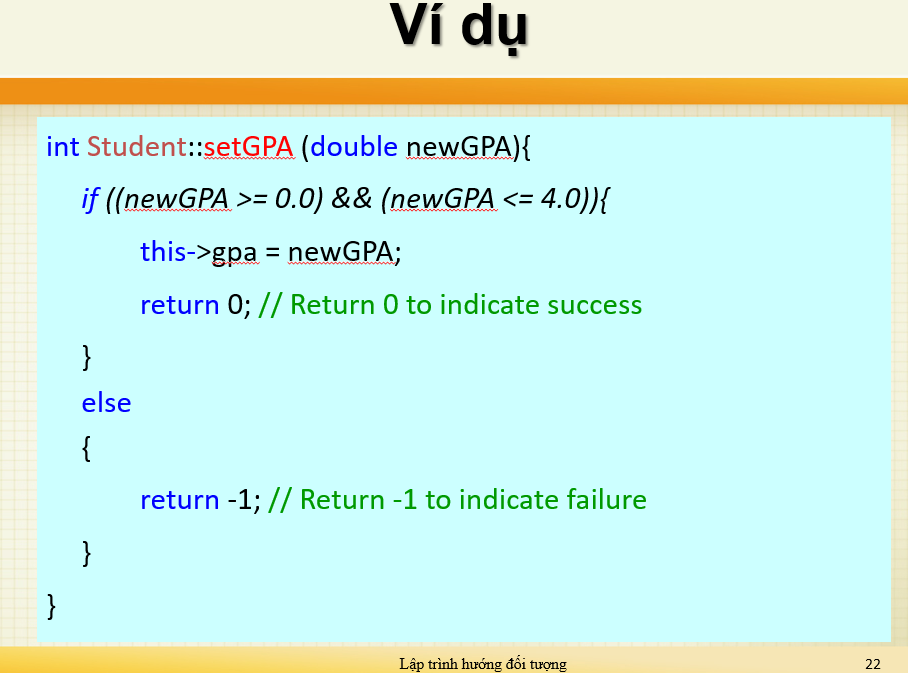
* Có nhiều loại câu hỏi truy vấn có thể:
  + Truy vấn đơn giản (“giá trị của x là bao nhiêu?”)
  + Truy vấn điều kiện (“thành viên x có > 10 không?”)
  + Truy vấn dẫn xuất (“tổng giá trị của các thành viên x và y là bao nhiêu?”)
* Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng
* Đối với các truy vấn đơn giản, quy ước đặt tên phương thức như sau: Tiền tố “get”, tiếp theo là tên của thành viên cần truy vấn
  + int getX();
  + int getSize();
* Các loại truy vấn khác nên có tên có tính mô tả
* Truy vấn điều kiện nên có tiền tố “is”

**Phương thức Cập nhật**

* Thường để thay đổi trạng thái của đối tượng bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó.
* Dạng đơn giản nhất là gán một giá trị nào đó cho một thành viên dữ liệu.
* Đối với dạng cập nhật đơn giản, quy ước đặt tên như sau: Dùng tiền tố “set” kèm theo tên thành viên cần sửa
  + int setX(int);

**Truy vấn và Cập nhật**

* Nếu phương thức get/set chỉ có nhiệm vụ cho ta đọc/ghi giá trị cho các thành viên dữ liệu 🡪Quy định các thành viên private để được ích lợi gì?
  + Ngoài việc bảo vệ các nguyên tắc đóng gói, ta cần kiểm tra xem giá trị mới cho thành viên dữ liệu có hợp lệ hay không.
  + Sử dụng phương thức truy vấn cho phép ta thực hiện việc kiểm tra trước khi thực sự thay đổi giá trị của thành viên.
  + Chỉ cho phép các dữ liệu có thể truy vấn hay thay đổi mới được truy cập đến.



**Liên kết trong oop:**

**Tĩnh:**

-Binding là liên kết giữa một cuộc gọi phương thức và định nghĩa phương thức.

-Trong liên kết tĩnh, liên kết được giải quyết tại thời gian biên dịch bởi trình biên dịch. Nó còn được gọi là ****ràng buộc sớm****. Binding xảy ra trước khi một chương trình thực sự chạy. Liĩnh xảy ra trong quá tải phương thứcên kết t

**Động:**

- Liên kết động là quá trình một hàm thông qua con trỏ lớp cơ sở lúc runtime quyết định gọi hàm ảo lớp dẫn xuất nào

**-**Trong Dynamic Binding, trình biên dịch không giải quyết ràng buộc tại thời gian biên dịch. Liên kết xảy ra trong thời gian chạy. Nó còn được gọi là ****Ràng buộc muộn****. Binding động xảy ra trong phương thức ghi đè.

* =>Giống nhau : Cả hai đều liên quan đến đa hình cho phép một đối tượng hành xử theo nhiều cách.

**Phương thức ảo:**

**-** Là cách thể hiện tính đa hình trong ngôn ngữ

- Phương thức ảo chỉ hoạt dộng thông qua lớp cơ sở

- Muốn trở thành phương thức ảo có 2 cách: sử dụng từ khoá virtual hoặc kế thừa ( phương thức tương ứng ở lớp cơ sở đã là phương thức ảo)

- chỉ hoạt động khi các phương thức ở lớp cơ sở và lớp con có nghi thức giao tiếp giống nhau

- Nếu định nghĩa phương thức ảo ở lớp con lại thì sẽ gọi phương thức ở lớp cơ sở gần nhất.

- Mỗi đối tượng sẽ có thêm 1 bảng phương thức ảo chứa các địa chỉ phương thức ảo được trình biên dịch khởi tạo ngầm

- Phương thức ảo không thể là các hàm thành viên tĩnh

- Thể hiện sự liên kết động phương thức ảo đc định nghĩa thông nhất từ lớp cha tới con

- Nếu định nghĩa phương thức ảo ở lớp con khác đi một chút thì trình biên dịch sẽ xem nó là 1 phương thức khác.

**Toán tử pham vi:**

Là toán tử cho phép class truy cập thuộc tính và phương thức bên ngoài class để định nghĩa, truy cập các biến toàn cục khi có biến cục bộ trùng tên.

Cerr: Luồng báo lỗi chuẩn

**Tham số mặc nhiên:**

- Gán giá trị mặc nhiên cho các tham số của hàm, phải là 1 dãy tham số liên tục nằm ở cuối hàm

- Dùng tham số truyền vào (đủ), dùng tham số mặc nhiên .

**Tái định nghĩa hàm:**

- Các hàm trùng tên phải khác nhau về tham số: số lượng, kiểu dữ liệu, thứ tự.

- Gọi hàm: tìm hàm có kiểu tham số phù hợp, ép kiểu, hàm gần đúng.

**Hàm inline:**

- Là hàm nội tuyến yêu cầu trình biên dịch copy code vào trong chương trình ( dùng với kích thước nhỏ)

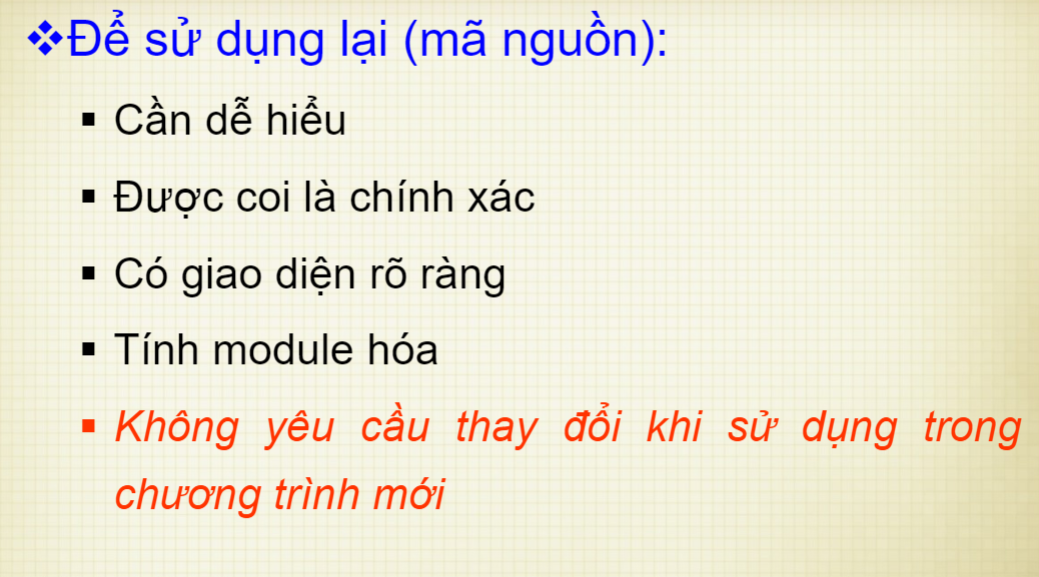
**Static:**

**Static Variables :** Variables in a function, Variables in a class

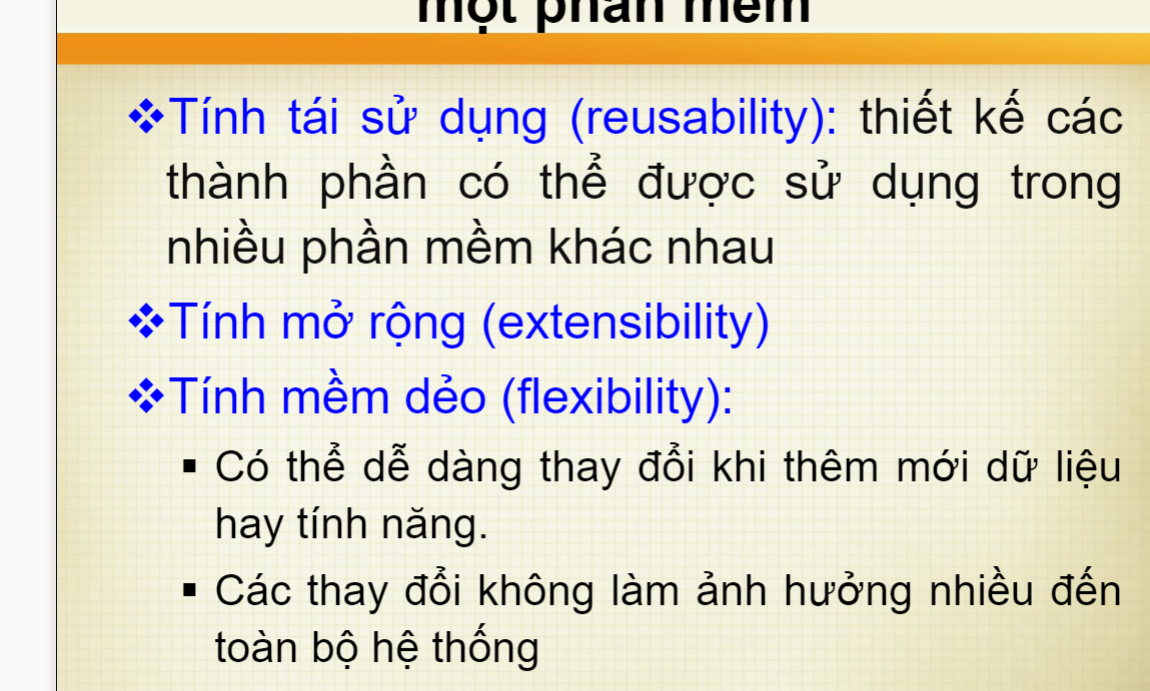
**- Static variables in a Function**: When a variable is declared as static, space for **it gets allocated for the lifetime of the program**.  
**Static Members of Class :**Class objects and Functions in a class

**- Class objects as static**: Just like variables, objects also when declared as static have a scope till the lifetime of program.

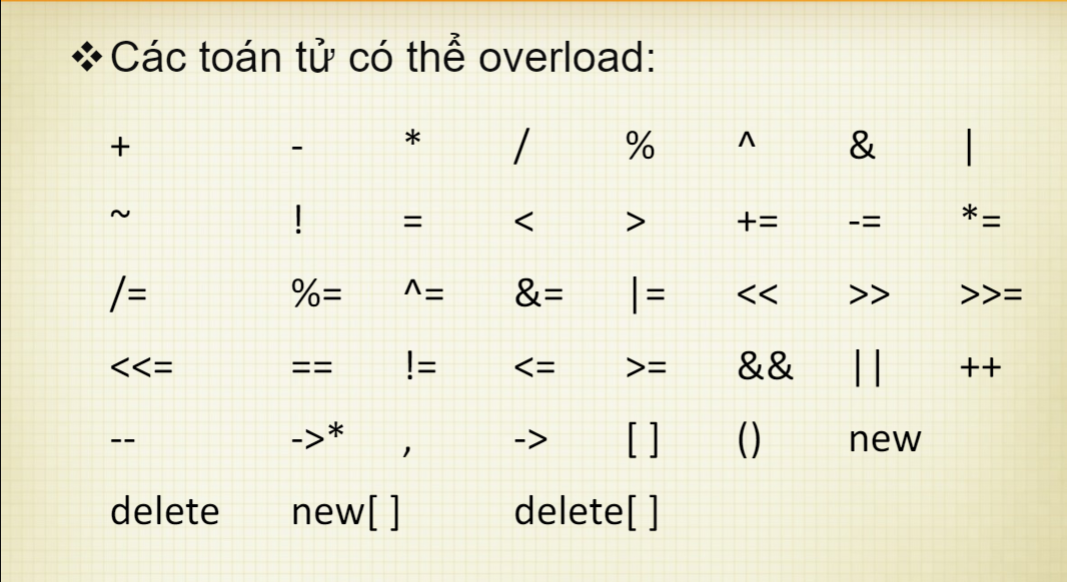
- Có hai cách truy cập vào static: hàm thành viên tĩnh, thành viên dữ liệu tĩnh.



**\*Mục tiêu của thiết kế phần mềm:**



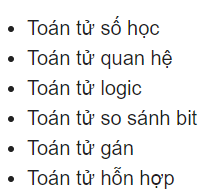
**Toán tử:**



Hàm friend và non-member không thể overloading , ( ) , [ ] , ->

Các toán tử không thể nạp chồng: . .\* :: ?:

Có cac loại toán tử:

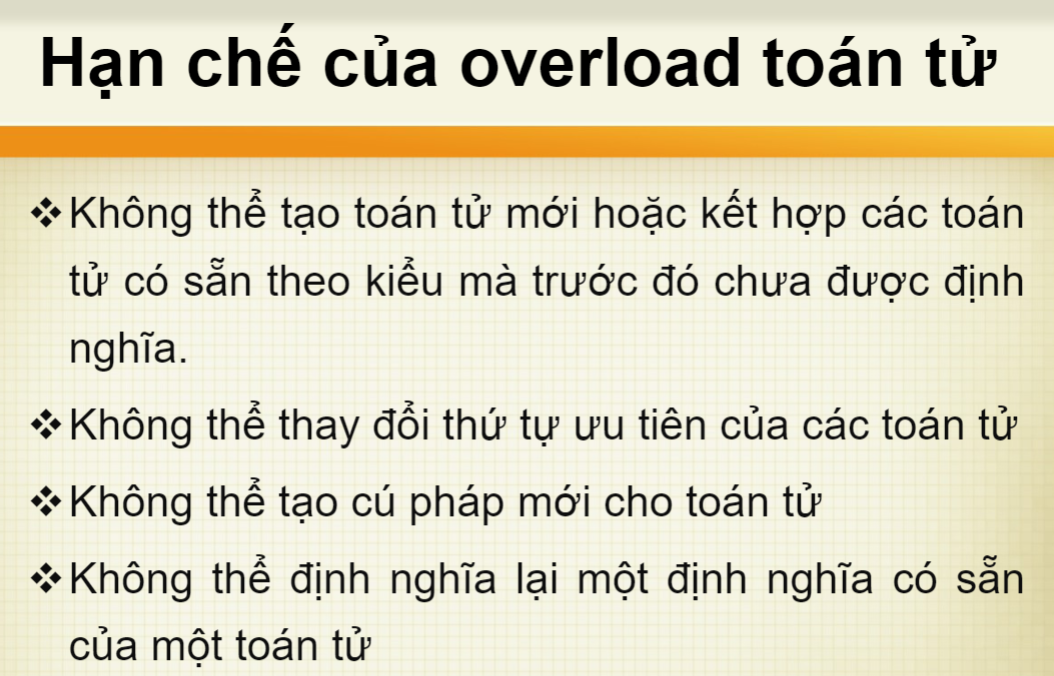


- Toán tử 1 ngôi: + - ++ --

- Toan tử 2 ngôi: + - \* / %

- Toán tử 3 ngồi: conditional operator, ?:…..

- Toán tử ép kiểu:const\_cast, dynamic\_cast, reinterpret\_cast, static\_cast

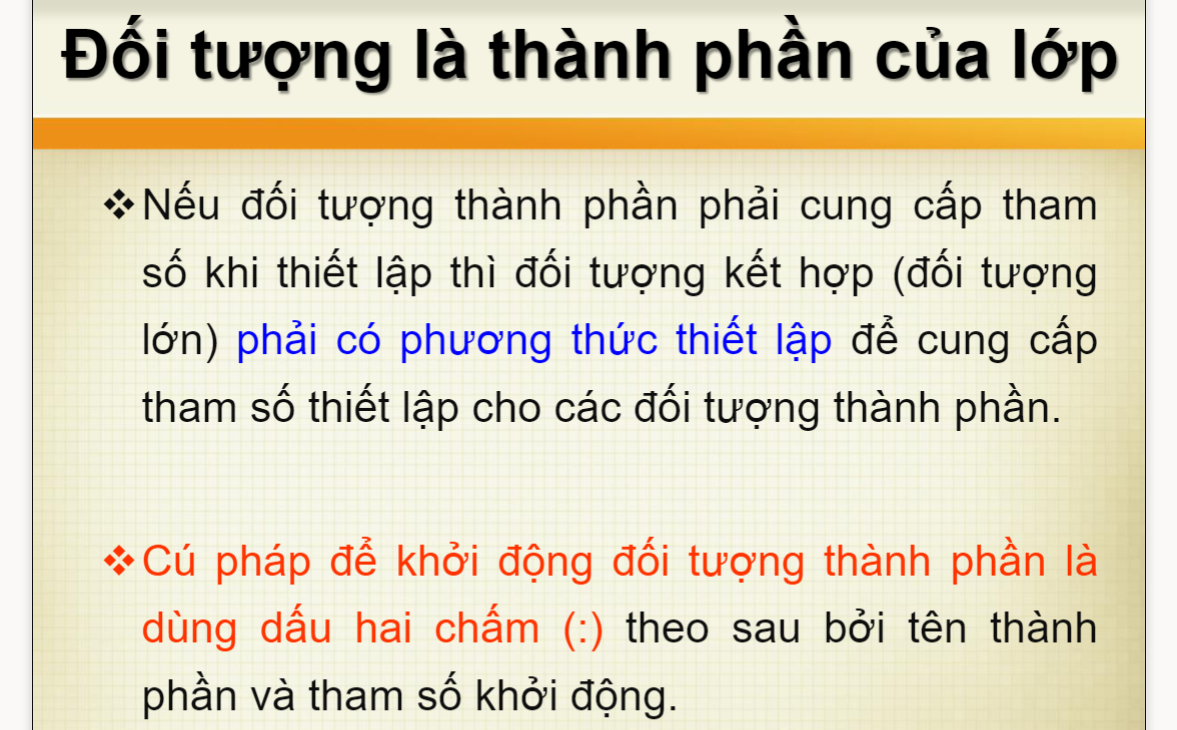


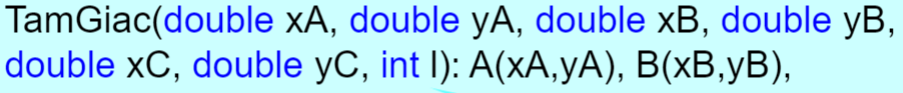
-Phép << >> được định nghĩa như một hàm toàn cục.

**Chuyển kiểu:**

- Có thể chuyển kiểu bằng constructor, phép toán chuyển kiểu

=> sự nhập nhằng: là khi trình biên dịch tìm ra được ít nhất 2 cách chuyển kiểu để thực hiện 1 việc tính toán nào đó.





**\*Đối tượng mảng**:

- Nếu 1 mảng được tạo ra -> các phần tử của nó cũng được tạo ra -> phương thức thiết lập sẽ được gọi cho từng phần tử. Vì không thể cũng cấp tham số cho toàn bộ phần tử mảng nên mỗi phần tử sẽ tự thiết lập không cần tham số.

- Đối tượng có khả năng tự khởi động trong trường hợp: lớp không có phương thức thiết lập, thiết lập không tham số, thiết lập bằng tham số mặc nhiên.

**\*Friend:**

- Có thể khai báo ở private

- Phải được cho - không được công nhận, không đối xứng, không bắc cầu

**\* Nhâp là insertion, xuất là extraction, default là mặc định**

**\*Con trỏ this:**

Dùng để xác định của đối tượng và là tham số ngâm định cho hàm thành phần. Các hàm friend không có **con trỏ** this, bởi vì friend không phải là các thành viên của một lớp.

**\*Cơ chế Upcasting và Downcasting:**

**Upcasting** là gán object của subclass cho biến tham chiếu supperclass, trong khi **Downcasting** gán object của supperclass cho biến tham chiếu subclass. Khi thực hiện **downcasting** chúng ta cần kiểm tra kiểu dữ liệu của object trước khi gán giá trị để tránh lỗi ClassCastException

**\*Hàm hằng:**

Ví dụ get(void) const { return ….};