Contents

[**Các tính chất trong lập trình hướng đối tượng** 2](#_Toc111645028)

[**I. Tính đóng gói(Encapsulation)** 2](#_Toc111645029)

[**II. Tính trừu tượng(Apstraction)** 2](#_Toc111645030)

[**III. Tính kế thừa:** 3](#_Toc111645031)

[3.1 Vấn đề Constructor trong kế thừa: 3](#_Toc111645032)

[3.2 Vấn đề Access Modifier khi kế thừa 3](#_Toc111645033)

[3.3 Ghi đè(Overriding) 5](#_Toc111645034)

[3.4 Đa kế thừa trong OOP(Multiple Inheritance) 6](#_Toc111645035)

[3.5 Sự mập mờ trong đa kế thừa 8](#_Toc111645036)

[**4.1** Các loại đa hình: 9](#_Toc111645037)

[**4.1.1** Compile time Polymorphism 9](#_Toc111645038)

[4.1.2 Runtime Polymorphism: 11](#_Toc111645039)

# **Các tính chất trong lập trình hướng đối tượng**

1. Tính đóng gói

2. Tính trừu tượng

3. Tính kế thừa

4. Tính kế thừa

## **I. Tính đóng gói(Encapsulation)**

Tính đóng gói có nghĩa là gì: Trong ngôn ngữ lập trình máy tính hướng đối tượng (OOP), khái niệm đóng gói (hay OOP Encapsulation) đề cập đến việc dữ liệu, cùng với các phương thức hoạt động trên dữ liệu đó thành một đơn vị duy nhất. Nhiều ngôn ngữ lập trình sử dụng tính năng đóng gói dưới dạng các class.

Tính đóng gói cũng có thể đề cập đến cơ chế hạn chế quyền truy cập trực tiếp vào một số thành phần của một đối tượng, như vậy **người dùng không thể truy cập các giá trị trạng thái cho tất cả các biến của một đối tượng cụ thể. Đóng gói có thể được sử dụng để ẩn cả thành viên dữ liệu và các hàm hoặc phương thức dữ liệu được liên kết với một lớp hoặc đối tượng được khởi tạo.**

* **Lợi ích của tính đóng gói:**
* **Hiding Data:** Người dùng sẽ không biết Class đang được triển khai hoặc lưu trữ như thế nào. Tất cả những gì người dùng sẽ biết là các giá trị đang được truyền vào và khởi tạo.
* Linh hoạt hơn: Cho phép bạn đặt các biến là chỉ đọc hoặc chỉ ghi. Ví dụ bao gồm: setName (), setAge () hoặc để đặt các biến là chỉ ghi thì ta chỉ cần bỏ qua các phương thức get như getName (), getAge (), v.v.
* Dễ sử dụng lại: Với tính năng đóng gói, bạn có thể dễ dàng thay đổi và thích ứng với các yêu cầu mới.

## **II. Tính trừu tượng(Apstraction)**

Trừu tượng hóa dữ liệu là một trong những tính năng cần thiết và quan trọng nhất của lập trình hướng đối tượng trong C ++. Trừu tượng có nghĩa là chỉ hiển thị thông tin cần thiết và ẩn các chi tiết. Trừu tượng hóa dữ liệu đề cập đến việc chỉ cung cấp thông tin cần thiết về dữ liệu cho thế giới bên ngoài, che giấu các chi tiết cơ bản hoặc việc triển khai.

Hãy xem xét một ví dụ thực tế về một người đàn ông lái xe ô tô. Người đàn ông chỉ biết rằng nhấn ga sẽ tăng tốc độ xe hoặc nhấn phanh sẽ dừng xe nhưng anh ta không biết về cách nhấn ga thực sự tăng tốc độ, anh ta không biết về cơ cấu bên trong của ô tô hay thực hiện chân ga, phanh vv trên ô tô. Đây là những gì trừu tượng là.

**Tính trừu tượng sử dụng các chỉ định quyền truy cập(Access specifier) như: public, private, protected**

* **Lợi ích của tính trừu tượng:**
* Giúp tăng cường bảo mật của một ứng dụng hoặc chương trình vì chỉ những chi tiết quan trọng mới được cung cấp cho người dùng.
* Có thể thay đổi việc triển khai nội bộ của lớp một cách độc lập mà không ảnh hưởng đến người dùng

## **III. Tính kế thừa:**

Kế thừa là một tính năng hoặc một quá trình, trong đó, các lớp mới được tạo ra từ các lớp hiện có. Lớp mới được tạo được gọi là “derived class”(lớp dẫn xuất) hoặc “child class”(lớp con) và lớp hiện có được gọi là “class base”(lớp cơ sở) hoặc “parent class”(lớp cha).

* **Sub Class:** Lớp kế thừa các thuộc tính từ một lớp khác được gọi là SubClass hoặc derived class.
* **Super Class:**Lớp có các thuộc tính được kế thừa bởi một lớp con được gọi là Base Class hoặc Super Class.

### 3.1 Vấn đề Constructor trong kế thừa:

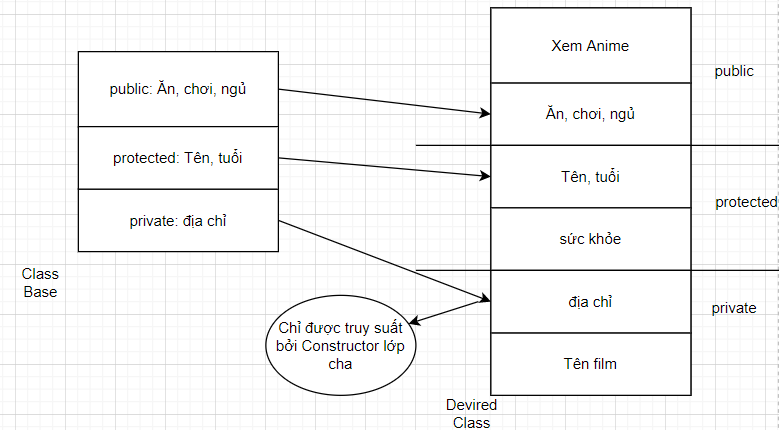
* Class con bắt buộc phải gọi 1 constructor lớp cha.
* **Nếu lớp cha không khai báo constructor thì lớp con sẽ tự động gọi constructor nguyên thủy của lớp cha**
* Nếu lớp cha khai báo các constructors không phải construct nguyên thủy của lớp cha, thì lớp con bắt buộc phải gọi 1 trong các cunstructors khai báo đó.
* Nếu lớp cha làm tường minh constructor nguyên thủy thì lớp cha không cần gọi constructor lớp cha.

### 3.2 Vấn đề Access Modifier khi kế thừa

* Ở private của Class cở sở, khi class con kế thừa được class cơ sở, class con không thể sử dụng được các thuộc tính ở phần private đó(Trừ việc gọi constructor thì được).
  + Sinh ra từ khóa “Protected”
* **Từ khóa "protected" đảm bảo cơ chế không thể truy suất dữ liệu từ bên ngoài nhưng vẫn truy suất được từ các Class dẫn suất.**
* **Trong C++ có 3 kiểu Inheritance Access Mode:**
  + **Public access mode**
  + **Protected access mode**
  + **Private access mode**

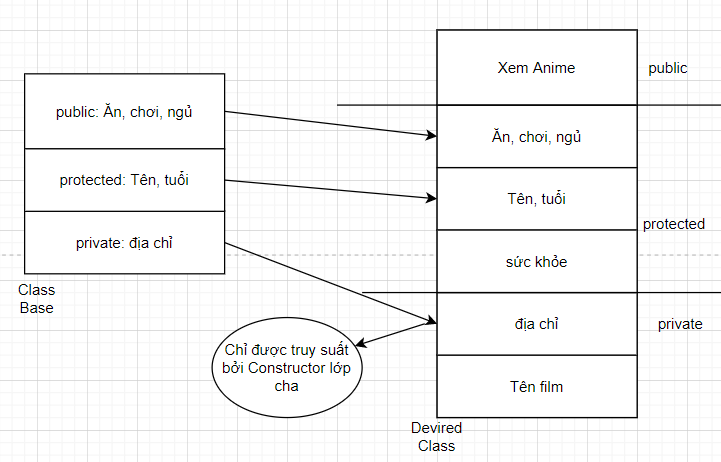
#### 3.2.1 Public Access Mode:

* **Thuộc tính public của lớp cha sẽ là public của lớp con.**
* **Thuộc tính protected của lớp cha sẽ là protected của lớp con.**
* **Thuộc tính private của lớp cha chỉ được sử dụng trong việc định nghĩa constructor lớp con.**

****

#### 3.2.2 Procectec Access Mode

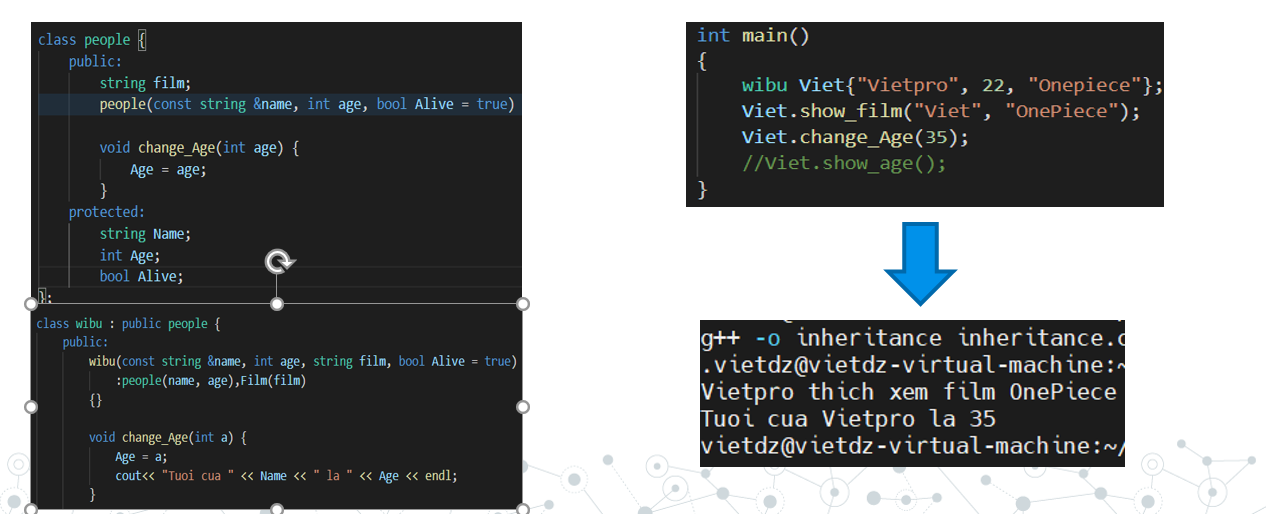
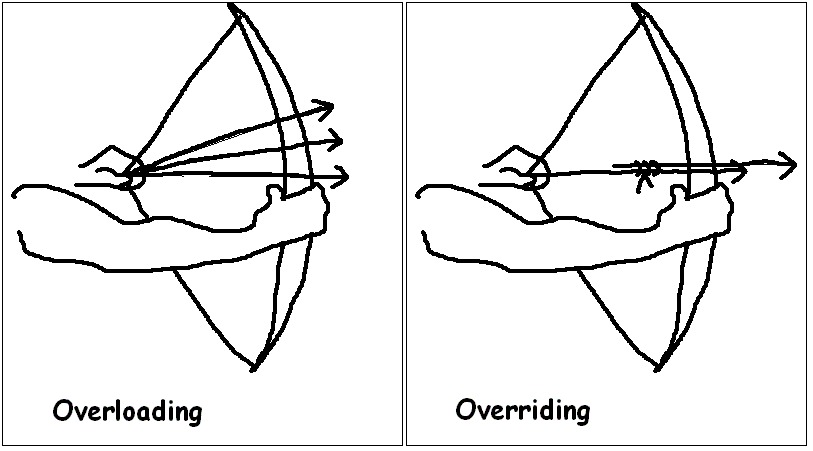
* **Thuộc tính public của lớp cha sẽ là protected của lớp con.**
* **Thuộc tính protected của lớp cha sẽ là protected của lớp con.**
* **Thuộc tính private của lớp cha chỉ được sử dụng trong việc định nghĩa constructor lớp con.**

****

#### 3.2.3 Private Access Mode:

* **Tất cả đều là Private của lớp con**

### 3.3 Ghi đè(Overriding)

* **Khi lớp con lại khai báo lại một phương thức có tên hàm và kiểu tham số giống với phương thức của lớp cha thì phương thức của lớp cha sẽ bị ghi đè.**
* ****
* 

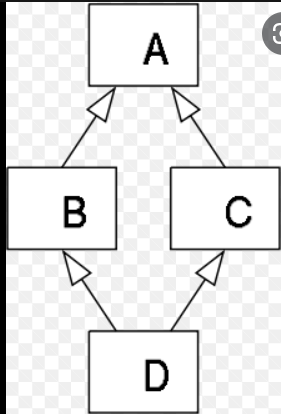
### 3.4 Đa kế thừa trong OOP(Multiple Inheritance)

* **C++ cho phép một lớp con có thể kế thừa một hoặc nhiều lớp cha.**

**3.4.1 DIAMOND INHERITANCE**

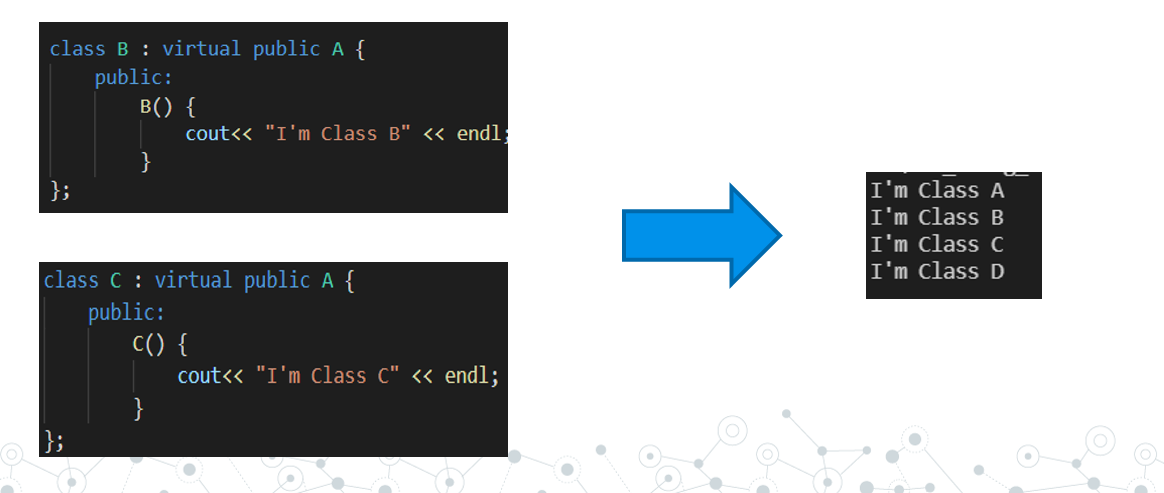
* **Class B và C kế thừa Class A**
* **Class D kế thừa Class B và Class C**

**🡪 Khi gọi Constructor, Constructor Class A được gọi 2 lần**



* **Để giải quyết vấn đề này, ta thêm từ khoá virtual khi class B và class C kế thừa class A.**

**🡪 Kỹ thuật này được gọi là “virtual inheritance”, nó sẽ đảm bảo chỉ duy nhất một thể hiện của class được kế thừa được khởi tạo.**

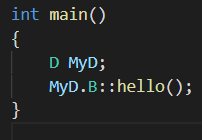
****

### 3.5 Sự mập mờ trong đa kế thừa

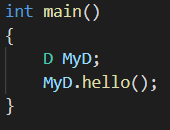
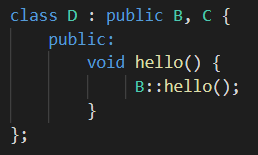
* **Khi lớp con kế thừa hai lớp cha và hai lớp cha này có một số phương thức giống nhau**
* **🡪 Sẽ gây ra sự mập mờ trong đa kế thừa, cụ thể khi đối tượng gọi tới phương thức đấy thì sẽ không biết được là muốn gọi phương thức nào**

**Vấn đề này được giải quyết bằng thêm toán hạng phân giải phạm vi**

**Cách 1:**



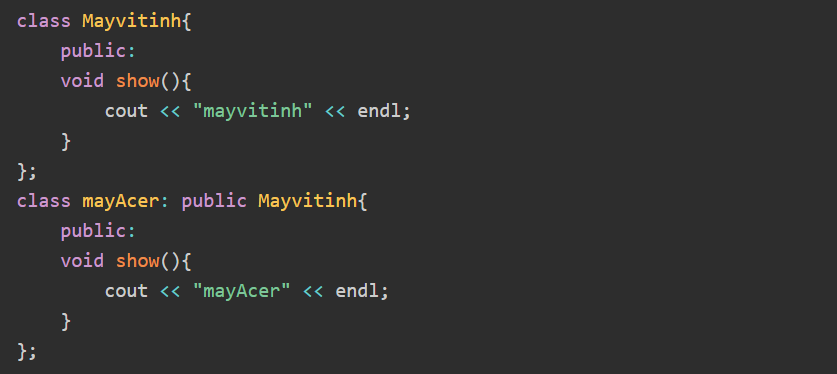
**Cách 2:**

****

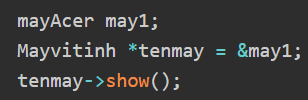
**IV. Tính đa hình(Polymorphism)**

Sự kế thừa trong C++ cho phép có sự tương ứng giữa lớp cơ sở và các lớp dẫn xuất trong sơ đồ thừa kế:

* **Một con trỏ có kiểu lớp cơ sở luôn có thể trỏ đến địa chỉ của một đối tượng của lớp dẫn xuất. (chỉ đúng khi kế thừa kiểu public).**
* Tuy nhiên, khi thực hiện lời gọi một phương thức của lớp, trình biên dịch sẽ quan tâm đến kiểu của con trỏ chứ không phải đối tượng mà con trỏ đang trỏ tới.



khi đó, nếu ta khai báo một con trỏ lớp mayAcer, nhưng lại trỏ vào địa chỉ của một đối tượng lớp Mayvitinh:



Thì chương trình sẽ gọi đến phương thức show() của lớp Mayvitinh, mà không gọi tới phương thức show() của lớp mayAcer.  
Để giải quyết vấn đề này, chúng ta cần sử dụng đến **tính đa hình** trong Lập trình hướng đối tượng. Mình sẽ giải thích chi tiết hơn ở bên dưới.

* **Đa hình (polymorphism)** nghĩa là có nhiều hình thái khác nhau. Tiêu biểu là, **đa hình** xuất hiện khi có một cấu trúc cấp bậc của các lớp và chúng liên quan với nhau bởi tính kế thừa.

## **4.1** Các loại đa hình:

**Tính đa hình** chủ yếu được chia thành hai loại:

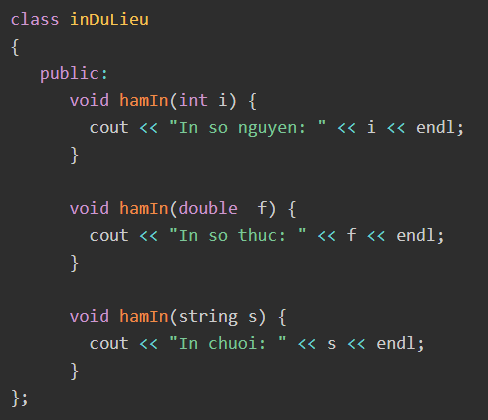
* Compile time Polymorphism.
* Runtime Polymorphism.

### **4.1.1** Compile time Polymorphism

**Tính đa hình** này được sử dụng bằng cách nạp chồng hàm hoặc nạp chồng toán tử.

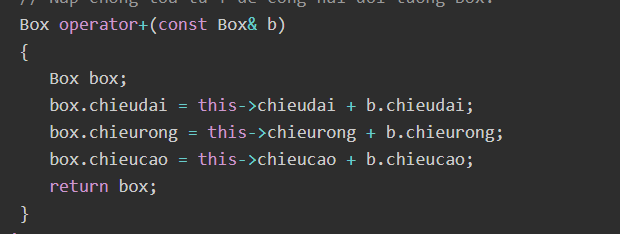
#### **4.1.1.1 Nạp chồng hàm (Function Overloading)**

Cho phép sử dụng cùng một tên gọi cho các hàm “giống nhau” (có cùng mục đích). Nhưng khác nhau về kiểu dữ liệu tham số hoặc số lượng tham số.

****

#### **4.1.1.2 Nạp chồng toán tử**

**Nạp chồng toán tử (Operator Overloading)**được dùng để định nghĩa toán tử cho có sẵn trong c++ phục vụ cho dữ liệu riêng do bạn tạo ra.



### 4.1.2 Runtime Polymorphism:

Để khắc phục ở ví dụ đầu tiên, để chương trình gọi tới phương thức show() của lớp mayAcer ta sử dụng hàm ảo virtual như sau:



Kết quả

