**NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG C++**

1. **Tính đóng gói**

Mọi dữ liệu và phương thức có liên quan đều sẽ được đóng gói thành các lớp để tiện lợi hơn cho quá trình sử dụng và quản lý. Nhờ vậy mà mỗi lớp đều sẽ được xây dựng để thực hiện một nhóm chức năng đặc trưng riêng của lớp đó. Việc đóng gói sẽ giúp che dấu một số thông tin và chi tiết cài đặt nội bộ khiến bên ngoài không thể nhìn thấy.

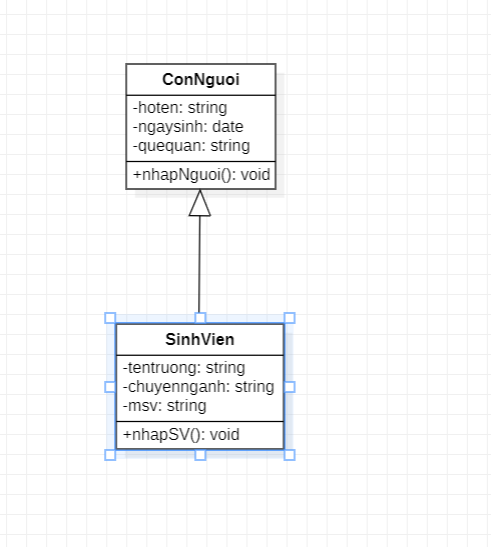
1. **Tính trừu tượng**

Tính trừu tượng thể hiện tổng quát hóa lên một cái gì đó mà không cần chú ý các chi tiết bên trong. Nó sẽ không màng đến những chi tiết bên trong là gì mà người ta vẫn có thể hiểu nó ngay mỗi khi nó được nhắc đến.

Trong lập trình OOP thì tính trừu tượng có nghĩa là chọn ra các phương thức, các thuộc tính của đối tượng cần cho việc giải quyết các bài toán lập trình. Bởi vì đối tượng sẽ có rất nhiều thuộc tính phương thức nhưng với bài toán cụ thể thì bạn không nhất thiết phải chọn toàn bộ.

1. **Tính kế thừa**

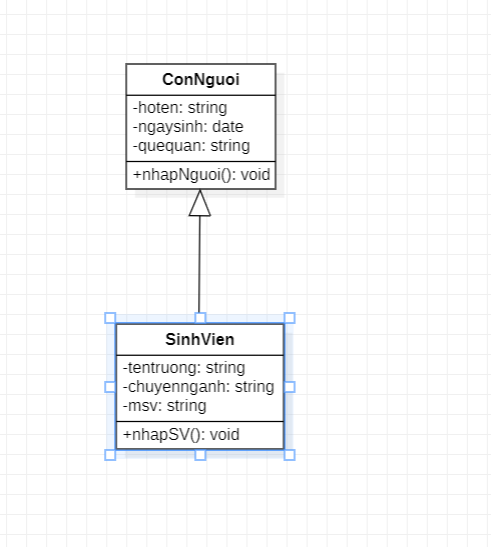
* ***Khái niệm:*** Kế thừa (inheritance) là một tính chất đặc trưng của lập trình hướng đối tượng. Nó có nghĩa là một class thừa hưởng lại tất cả các thuộc tính, phương thức của class mà nó kế thừa.
* Class kế thừa từ một class khác gọi là ***lớp con*** (child class hay subclass) hay lớp dẫn xuất (derived class). Class được lớp khác kế thừa được gọi là lớp cha (parent class hay superclass) hay ***lớp cơ sở*** (base class).



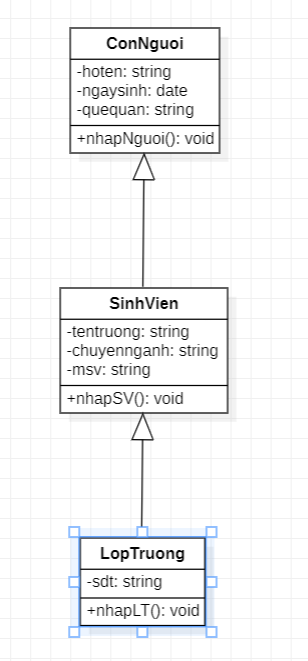
* ***Tại sao cần có kế thừa?***

Trong lập trình, chắc chắn các bạn sẽ gặp phải rất nhiều hành động lặp đi lặp lại, và chắc chắn một điều rằng sẽ không ai code lại chúng từ đầu cả. Thay vào đó, họ sẽ sử dụng lại các đoạn code đã có để thực hiện công việc tương tự. Kế thừa giúp ta có thể tái sử dụng lại những đoạn code đã có, tránh việc giải quyết lại các bài toán con đã có lời giải trước đó, gây lãng phí thời gian. Nếu chỉ là một, hai hoặc ba hành động cần giải quyết thì không thành vấn đề, nhưng nếu là một trăm thì đó lại là chuyện khác, sẽ tốn thời gian nếu bạn không biết tận dụng những gì đã có.

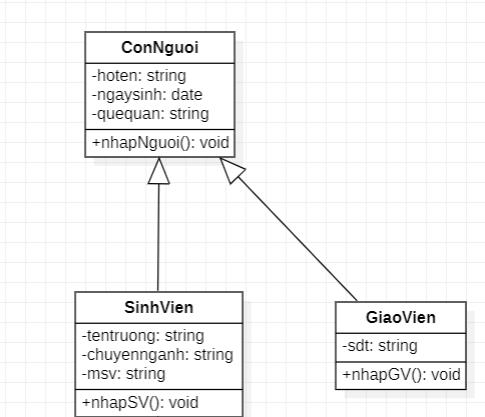
* ***Các loại kế thừa***
* Kế thừa đơn: Kế thừa đơn (single inheritance) là một class con kế thừa duy nhất từ một class cha



* Kế thừa nhiều cấp: là một class con kế thừa từ một class cha, class cha đó lại kết thừa từ một lớp khác



* Kế thừa phân cấp: Kế thừa phân cấp (hierarchical inheritance) là khi có nhiều hơn một class con kế thừa từ class cha



* Cú pháp:

class <tên\_class\_con> : <phạm\_vi\_truy\_cập> <tên\_class\_cha>

{

// code

};

Phạm vi truy cập: private, protected, public

* Phạm vi truy cập public
* Các thuộc tính **public của A** sẽ trở thành thuộc tính **public của B**
* Các thuộc tính **protected của A** sẽ trở thành **protected của B**
* Phạm vi truy cập protected
* Các thuộc tính **public của A** sẽ trở thành thuộc tính **protected của B**
* Các thuộc tính **protected của A** sẽ trở thành thuộc tính **protected của B**
* Phạm vi truy cập private
* Các thuộc tính **public của A** sẽ trở thành thuộc tính **private của B**
* Các thuộc tính **protected của A** sẽ trở thành **private của B**

1. **Tính đa hình**

* Đa hình (polymorphism) là hiện tượng mà các đối tượng thuộc các class khác nhau có thể biểu diễn cùng một thông điệp theo các cách khác nhau.
* Vấn đề đặt ra: Ví dụ hai con vật là con chó và con mèo, hai con vật này đều có thể phát ra tiếng nhưng con mèo sẽ kêu “meo meo” còn con chó lại sủa “gâu gâu”. Hành động phát ra tiếng này tuy là một hành động nhưng khi được 2 đối tượng khác nhau là chó và mèo thực hiện thì lại khác nhau.
* Có hai loại đa hình, một là **Compile time Polymorphism** và **Runtime Polymorphism**.
* ***Compile time Polymorphism*** nghĩa là cách mà đối tượng thực hiện thông điệp được xác định ngay lúc biên dịch chương trình. Overriding và Overloading.
* Overriding (Ghi đè)
* Overloading các hàm đó sẽ có cùng tên nhưng khác tham số, đó chính là đa hình, cùng là một hàm nhưng với các đối tượng là các tham số khác nhau thì hành động thực hiện cũng sẽ khác nhau
* ***Compile time Polymorphism*** cách mà đối tượng thực hiện thông điệp không được xác định lúc biên dịch mà nó chỉ được xác định khi chương trình được thực thi. Sử dụng lại ví dụ ban đầu, vẫn là hành động phát ra tiếng của chó và mèo, tuy nhiên, bạn sẽ không thể biết thực hiện như thế nào cho đến khi biết đối tượng là con chó hay mèo để thực hiện kêu “meo meo” hay là sủa “gâu gâu”.
* **Phương thức ảo** là cách thể hiện tính đa hình trong lập trình hướng đối tượng của C++, các phương thức ở class cơ sở có tính đa hình phải được định nghĩa là một phương thức ảo.
* Chúng ta đã có các phương thức ảo, vậy vấn đề hiện tại là làm sao để chương trình biết phương thức của đối tượng nào để thực hiện cho đúng? Thì trong C++, tính đa hình được thể hiện thông qua tham chiếu và con trỏ

**Lưu ý:** tính đa hình chỉ sử dụng được thông qua tham số là tham chiếu hoặc con trỏ, nếu bạn sử dụng một tham số bình thường, phương thức của class cơ sở sẽ được gọi vì về bản chất nó là một đối tượng thuộc kiểu class cơ sở.

* Phương thức thuần ảo là phương thức ảo không có phần định nghĩa và bắt buộc phải được override ở class dẫn xuất. Để khai báo phương thức thuần ảo, ta cũng khai báo tương tự như phương thức ảo nhưng thay vì định nghĩa phần thân hàm, ta sẽ cho nó bằng 0.