Kế thừa là gì?

Kế thừa là một trong các tính chất đặc trưng của lập trình hướng đối tượng, bên cạnh tính đóng gói (encapsulation), che giấu thông tin (hiding information), tính đa hình (polymorphism) và tính trừu tượng (abstraction). Vậy thì kế thừa là gì?

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).

Ví dụ như bạn có một class con người, có các thuộc tính cơ bản như họ tên, ngày sinh, quê quán, mình khai báo thêm một class sinh viên kế thừa từ class con người. Vậy, class sinh viên sẽ có các thuộc tính họ tên, ngày sinh, quê quán từ class con người mà không cần phải khai báo. Class con người sẽ là lớp cha và class sinh viên là lớp con.

Ngoài các thuộc tính của class cha, class con còn có thể có thêm các thuộc tính, phương thức của riêng mình. Ví dụ như sinh viên thì có thêm các thuộc tính như MSSV, tên trường, chuyên ngành…

Tính kế thừa

Chúng ta đã hiểu được kế thừa là gì rồi. Vậy thì câu hỏi đặt ra lúc này là tại sao cần có kế thừa? Hãy cùng tìm hiểu tiếp nhé.

Tại sao cần có kế thừa?

Trong lập trình, chắc hẳn các bạn sẽ gặp phải rất nhiều hành động lặp đi lặp lại, và chắc chắn một điều rằng sẽ không ai rảnh mà ngồi 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. Nói đến đây chắc các bạn cũng đã hiểu kế thừa có tác dụng gì rồi đúng không nào!

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 của bạn kha khá thời gian nếu bạn không biết tận dụng những gì đã có đấy!

Trong ví dụ ở đầu bài viết, các bạn có để ý thấy mối quan hệ giữa class con người và sinh viên không? Đúng vậy, mối quan hệ đó là “sinh viên là một con người”, sinh viên cũng có các thông tin như họ tên, ngày sinh, quê quán… và được biểu diễn bằng các thuộc tính trong class. Vậy thì tính kế thừa giúp cho chúng ta có thể dễ dàng thể hiện mối quan hệ giữa các đối tượng hơn (sẽ được trình bày bên dưới).

Sinh viên ngoài có những đặc điểm của class con người ra, còn có thể có thêm các đặc điểm của riêng mình ví dụ như tên trường, tên ngành, năm học… Đây là mối quan hệ tổng quát hóa – đặc biệt hóa trong kế thừa. Class con người chính là tổng quát hóa và class con sinh viên chính là đặc biệt hóa.

Mối quan hệ giữa các đối tượng

Có hai mối quan hệ giữa các đối tượng trong C++, quan hệ “Has-A” và quan hệ “Is-A”.

Quan hệ Has-A

Có 3 loại quan hệ “has a” giữa các đối tượng, mối quan hệ một – một, mối quan hệ một – nhiều và mối quan hệ nhiều – nhiều.

Quan hệ một – một (1 – 1) là mối quan hệ giữa hai đối tượng thuộc hai class khác nhau, mỗi đối tượng thuộc class này có quan hệ duy nhất với một đối tượng thuộc class kia và tương tự mỗi đối tượng của class kia cũng chỉ có quan hệ duy nhất với một đối tượng thuộc class này.

Ví dụ như mỗi lớp chỉ có một giáo viên chủ nhiệm, mỗi giáo viên chỉ được chủ nhiệm một lớp, mỗi một quốc gia chỉ có một thủ đô, một thành phố chỉ có thể là thủ đô của một quốc gia…

Quan hệ một – nhiều (1 – n) là mối quan hệ giữa đối tượng thuộc hai class khác nhay, mỗi đối tượng thuộc class này có quan hệ duy nhất với một đối tượng thuộc class kia, nhưng mỗi đối tượng thuộc class kia có thể có quan hệ với nhiều đối tượng thuộc class này.

Ví dụ như mỗi học sinh chỉ có một lớp học, nhưng mỗi lớp học lại có thể có nhiều học sinh, hay mỗi nhân viên chỉ có thể làm cho một công ty nhưng một công ty lại có thể có nhiều nhân viên…

Quan hệ nhiều – nhiều (n – n) là mối quan hệ giữa đối tượng thuộc hai class khác nhau, mỗi đối tượng thuộc class này có thể quan hệ với nhiều đối tượng thuộc class kia, và mỗi đối tượng thuộc class kia cũng có thể có quan hệ với nhiều đối tượng thuộc lớp này.

Ví dụ như mỗi bệnh nhân có thể thăm khám bệnh ở nhiều bác sĩ khác nhau, và mỗi bác sĩ lại có thể chữa cho nhiều bệnh nhân…

Quan hệ tổng quát – đặc biệt hóa là mối quan hệ giữa đối tượng thuộc hai class khác nhau khi đối tượng thuộc class này là một trường hợp đặc biệt của class kia, và đối tượng thuộc class kia là trường hợp tổng quát của đối tượng thuộc class này.

Ví dụ như con người là tổng quát hóa của sinh viên, còn sinh viên là đặc biệt hóa của con người, xe máy hiệu exciter là đặc biệt hóa của xe máy và xe máy là tổng quát của xe máy hiệu exciter…

Quan hệ Is-A

Mối quan hệ Is-A giữa các đối tượng chính là class này là một class kia. Cũng ví dụ từ đầu bài viết đến giờ, sinh viên là một con người (student **is a** human). Đó chính là mối quan hệ có được do thực hiện kế thừa.

Các loại kế thừa

Chúng ta đã cùng tìm hiểu qua về kế thừa là gì và tại sao cần có kế thừa, tiếp theo, hãy cùng tìm hiểu xem có những loại kế thừa nào nha.

Kế thừa đơn

Kế thừa đơn (single inheritance) là một class con kế thừa duy nhất từ một class cha. Ví dụ như class sinh viên chỉ kế thừa duy nhất từ class con người, đó là một kế thừa đơn.

Đơn kế thừa

Kế thừa đa cấp

Kế thừa đa cấp (multilevel inheritance) là một class con kế thừa từ một class cha, class cha đó lại kết thừa từ một lớp khác. Ví dụ như class sinh viên kế thừa từ class học sinh, class học sinh lại kế thừa từ class con người, đó là một kế thừa đa cấp.

Kế thừa đa cấp

Kế thừa phân cấp

Kế thừa phân cấp (hierarchical inheritance) là khi có nhiều hơn một class con kế thừa từ class cha. Ví dụ như class sinh viên và class công nhân đều kế thừa từ class cha là con người.

Kế thừa phân cấp

Một sơ đồ class không chỉ có một trong 3 loại kế thừa trên, nó thậm chí có thể có cả 3 và điều này hoàn toàn bình thường và gặp rất thường xuyên.

Sơ đồ Class

Vậy là chúng ta đã tìm hiểu xong các loại kế thừa, giờ hãy xem cú pháp thực hiện kế thừa trong C++ như thế nào.

Cú pháp

Cú pháp để khai báo một lớp kế thừa từ một lớp như sau:

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

{

// code goes here

};

Trong đó, class cha và class con đã được trình bày ở bên trên, phạm vi truy cập sẽ được mình trình bày ngay bến dưới.

Giả sử mình có một class A với các thuộc tính, phương thức bên trong. Mình muốn khai báo một class B kế thừa từ class A, mình sẽ có cú pháp như sau:

class B : public/private/protected A

{

// code goes here

};

Khi này, các thuộc tính trong class A sẽ được class B kế thừa. public/private/protected là phạm vi truy cập của các thuộc tính sẽ được kế thừa. Hãy cùng tìm hiểu xem cụ thể phạm vi truy cập là như thế nào ngay bên dưới.

Phạm vi truy cập

Trong bài [Cơ bản về Class trong C++](https://khiemle.dev/co-ban-ve-class-trong-cpp/) mình đã có giới thiệu qua về phạm vi truy cập (access modifier). Tuy nhiên, phạm vi truy cập trong bài đó khác với phạm vi truy cập trong kế thừa.

Có hai loại phạm vi truy cập, truy cập các thành phần từ bên ngoài đối tượng được gọi là truy cập theo chiều ngang. Truy cập các thành phần của class cha từ class con được gọi là truy cập theo chiều dọc. Phạm vi truy cập trong bài [Cơ bản về Class trong C++](https://khiemle.dev/co-ban-ve-class-trong-cpp/) chính là truy cập theo chiều ngang.

Đối với phạm vi truy cập trong kế thừa, đó là truy cập theo chiều dọc. Giả sử mình có class B kế thừa từ class A, cụ thể các phạm vi truy cập đó là như sau:

* 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
* 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
* 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 trong kế thừa C++

Đối với một class, thuộc tính protected cũng tương tự như private, chỉ có khác ở chỗ class con có thể truy cập thành phần protected, còn private lại không cho phép truy cập.

Phạm vi truy cập giúp đảm bảo tính đóng gói và che giấu thông tin của đối tượng. Ví dụ như khi thuộc tính public ở class cha, được kế thừa private sang class con, thì nó cũng trở thành private ở class con và không thể được truy cập từ bên ngoài class con. Ví dụ:

class A

{

public:

int publicMethod;

};

class B : private A

{

};

// bên trong hàm main

B b;

cout << b.publicMethod; // lỗi vì lúc này publicMethod trở thành private method của B rồi

Cài đặt

Từ đầu bài viết đến giờ, mình nói khá nhiều về lý thuyết mà chưa có đoạn code nào cả. Để cho các bạn dễ hiểu hơn, mình sẽ sử dụng lại các ví dụ trong bài viết và code chúng bằng C++. Hãy bắt đầu với các quan hệ Has-A trước.

Quan hệ 1 – 1:

class GVCN

{

public:

LopHoc lopHoc;

};

class LopHoc

{

public:

GVCN gvcn;

};

Bạn nào chưa biết về vector có thể xem lại bài viết [Vector trong C++](https://khiemle.dev/vector-trong-cpp/) của mình nha.

Quan hệ 1 – n:

class HocSinh

{

public:

LopHoc lopHoc;

};

class LopHoc

{

public:

vector<HocSinh> hocSinh;

};

Quan hệ n – n:

class BenhNhan

{

public:

vector<BacSi> bacSi;

};

class BacSi

{

public:

vector<BenhNhan> benhNhan;

};

Quan hệ Is-A:

class ConNguoi

{

public:

string hoTen;

string ngaySinh;

string queQuan;

};

class SinhVien : public ConNguoi

{

public:

string MSSV;

string tenTruong;

string lop;

string chuyenNganh;

};

Truy xuất thành phần từ lớp cơ sở

Khi class con kế thừa thành phần từ class cơ sở, các thành phần của nó cũng sẽ được thừa hưởng từ class cơ sở. Điều này có nghĩa là, bạn có thể sử dụng con trỏ this để thao tác với thành phần của class hiện tại như bình thường. Ví dụ:

class A

{

public:

int count;

void sayHello()

{

cout << "Hello, welcome to KhiemLe.Dev!";

}

};

class B : public A

{

public:

B() // constructor

{

this->count = 0; // thuộc tính count được kế thừa và có thể truy cập qua con trỏ this

this->sayHello(); // phương thức cũng được kế thừa

}

};

Override phương thức từ lớp cơ sở

Trong khi sử dụng tính kế thừa, bạn sẽ gặp phải trường hợp hai phương thức ở class con và class cha trùng tên nhau. Trong trường hợp này, phương thức được khai báo và định nghĩa ở class con sẽ ghi đè lên phương thức ở class cha và thay thế hoàn toàn nó. Ví dụ:

class A

{

public:

void sayHello()

{

cout << "Hello, welcome to khiemle.dev!";

}

};

class B : public A

{

public:

void sayHello()

{

cout << "Chao mung ban den voi trang web cua Khiem Le!";

}

};

// trong hàm main

B b;

b.sayHello(); // Chao mung ban den voi trang web cua Khiem Le!

Vậy thì bạn có thể thấy được là, nếu như mà phương thức bị trùng tên thì nó bị ghi đè, vậy thì không còn đảm bảo được tính tái sử dụng code của tính kế thừa. Vậy làm sao chúng ta có thể tái sử dụng lại đoạn code của phương thức ở class cha?

Sử dụng toán tử phân giải phạm vi (::), chúng ta có thể gọi được phương thức từ class cha, ngoài ra chúng ta còn có thể bổ sung được những đoạn code của riêng phương thức ở class con. Ví dụ:

class A

{

public:

void sayHello()

{

cout << "Hello, welcome to khiemle.dev!";

}

};

class B : public A

{

public:

void sayHello()

{

A::sayHello(); // Gọi lại phương thức ở class A

// Thêm các đoạn code các bạn muốn

cout << endl;

cout << "Nho like va share nha ^^!";

}

};

// trong hàm main

B b;

b.sayHello();

// Hello, welcome to khiemle.dev!

// Nho like va share nha ^^!