Bài 5: KẾ THỪA TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Xem bài học trên website để ủng hộ Kteam: Kế thừa trong Lập trình hướng đối tượng

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về TỪ KHÓA STATIC TRONG OOP C#. Hôm nay chúng ta sẽ cùng tìm hiểu về **Kế thừa trong C#**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- BIẾN và KIỂU DỮ LIỆU trong C#
- TOÁN TỬ TRONG C#
- CÂU ĐIỀU KIỆN TRONG C#
- CẤU TRÚC CƠ BẢN CỦA VÒNG LẶP TRONG C#
- CẤU TRÚC HÀM CƠ BẢN TRONG C#
- TỔNG QUAN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
- CLASS TRONG C#
- CÁC LOẠI PHẠM VI TRUY CẬP TRONG C#

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Khái niệm kế thừa.
- Khai báo và sử dụng kế thừa.

- Các vấn đề trong kế thừa.

Khái niệm kế thừa

Trong thực tế, **kế thừa** là việc thừa hưởng lại những gì mà người khác để lại. Ví dụ: con kế thừa tài sản của cha, . . .

Trong lập trình cũng vậy, **kế thừa trong lập trình** là cách 1 lớp có thể **thừa hưởng** lại những **thuộc tính**, **phương thức từ 1 lớp khác** và sử dụng chúng như là của bản thân mình.

Một định nghĩa trừu tượng hơn về kế thừa: là một đặc điểm của ngôn ngữ hướng đối tượng dùng để biểu diễn mối quan hệ **đặc biệt hoá – tổng quát hoá** giữa các lớp.

Ví dụ: Giả sử ta có lớp TamGiac chứa thông tin toạ độ của 3 điểm A, B, C. Ta biết rằng tam giác cân là 1 trường hợp **đặc biệt** của tam giác (ngược lại tam giác là trường hợp **tổng quát** của tam giác cân).

Từ đó ta có thể cho lớp TamGiacCan kế thừa lại lớp TamGiac để có thể sử dụng lại các thông tin như toạ độ 3 điểm A, B, C mà không cần phải khai báo.

Ngoài ra, ta có thể xét các lớp với các đặc điểm tương tự nhau, sau đó rút trích những thành phần chung và tạo thành 1 lớp. Cuối cùng cho các lớp kế thừa từ lớp cha đó để sử dụng lại thông tin mà không cần khai báo.

Dĩ nhiên là việc kế thừa không nên thực hiện 1 cách lung tung, lạm dụng.

Ưu điểm của kế thừa

- Cho phép xây dựng 1 lớp mới từ lớp đã có.
  - Lớp mới gọi là **lớp con (subclass)** hay **lớp dẫn xuất (derived class)**.
  - Lớp đã có gọi là **lớp cha (superclass)** hay **lớp cơ sở (base class)**.
- Cho phép chia sẽ các thông tin chung nhằm tái sử dụng và đồng thời giúp ta dễ dàng nâng cấp, dễ dàng bảo trì.

- Định nghĩa sự tương thích giữa các lớp, nhờ đó ta có thể chuyển kiểu tự động (sẽ được trình bày trong bài ĐA HÌNH TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG).

Khai báo và sử dụng kế thừa

Cú pháp

```
class <tên lớp con> : <tên lớp cha>
{
}
```

Trong đó:

- **class** là từ khoá để khai báo lớp.
- **<tên lớp con>** là tên do người dùng đặt và tuân theo các quy tắc đặt tên (quy tắc đặt tên đã trình bày trong bài **BIẾN TRONG C#**).
- **<tên lớp cha>** là tên lớp mà ta muốn kế thừa các đặc tính của nó.

Sử dụng

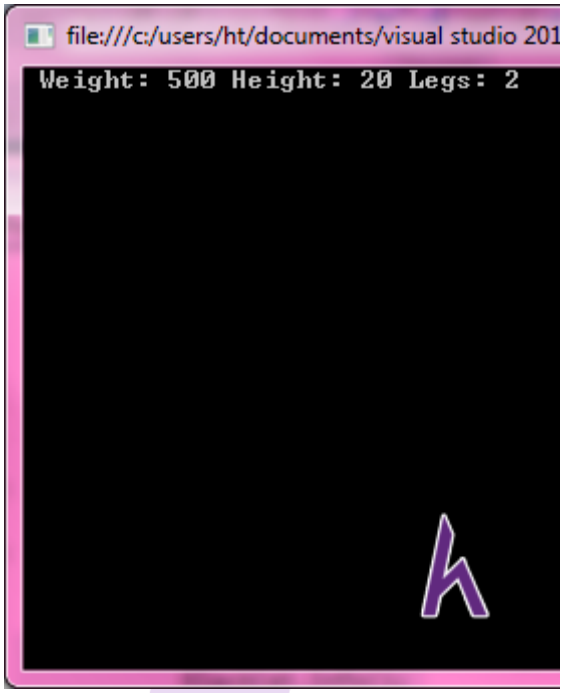
```
class Animal
   {
      protected double Weight;
      protected double Height;
      protected static int Legs;

      public void Info()
      {
         Console.WriteLine(" Weight: " + Weight + " Height: " + Height + " Legs: " +
Legs);
      }
   }
```

```
class Cat : Animal
   {
      public Cat()
      {
         /*
            Lớp Cat kế thừa lớp Animal
            mà các thuộc tính Weight, Height, Legs có phạm vi là protected nên
được phép kế thừa
            Từ đó lớp Cat có thể sử dụng mà không cần phải khai báo

          */
         Weight = 500;
         Height = 20;
         Legs = 2;
      }
   }
```

Trong hàm **main** ta thử gọi phương thức Info xem có được không nhé:

```
Cat BlackCat = new Cat();

         /* Lớp Cat kế thừa phương thức Info từ lớp Animal nên đối tượng thuộc lớp
Cat có thể gọi phương thức Info() */

         BlackCat.Info();
```

Kết quả khi chạy chương trình trên:



Bạn có thể thấy nhờ kế thừa mà lớp con trở nên ngắn gọn hơn rất nhiều.

Lưu ý:

Trong C#, **không hỗ trợ** đa kế thừa (1 lớp kế thừa từ nhiều lớp) những lại hỗ trợ thực thi nhiều interface (khái niệm về interface sẽ được trình bày trong bài INTERFACE TRONG C#).

Các thành phần của lớp cha có được kế thừa xuống lớp con hay không là do phạm vi truy cập của thành phần đó là gì.

- Thành phần có phạm vi là private thì không được kế thừa.
- Thành phần có phạm vi là protected, public thì được phép kế thừa.

Phương thức khởi tạo và phương thức huỷ bỏ không được kế thừa.

Các vấn đề trong kế thừa

Vấn đề về phương thức khởi tạo, phương thức huỷ bỏ

Phương thức khởi tạo mặc định của lớp cha luôn luôn được gọi mỗi khi có 1 đối tượng thuộc lớp con khởi tạo. Và được gọi trước phương thức khởi tạo của lớp con.

Nếu như lớp cha có phương thức khởi tạo có tham số thì đòi hỏi lớp con phải có phương thức khởi tạo tương ứng và thực hiện gọi phương thức khởi tạo của lớp cha thông qua từ khoá base.

Ví dụ: Giả sử lớp Animal có thêm 1 **constructor** có tham số như sau:

```
class Animal
   {
      public Animal(double w, double h, int l)
      {
         Weight = w;
         Height = h;
         Legs = l;
      }
   }
```

Khi đó lớp con kế thừa từ lớp Animal phải có **constructor** có các tham số tương ứng:

```
class Cat : Animal
   {
      /*
         Cách gọi constructor của lớp cha thông qua từ khoá base
       */

      public Cat()
      {
         Weight = 500;
         Height = 20;
         Legs = 2;
      }


      /*
```

```
        Cách gọi constructor của lớp cha thông qua từ khoá base
    */

    public Cat(double w, double h, int l) : base(w, h, l)
    {


    }
}
```

Qua ví dụ trên bạn dễ dàng thấy được cú pháp để gọi **constructor** của lớp cha như sau:

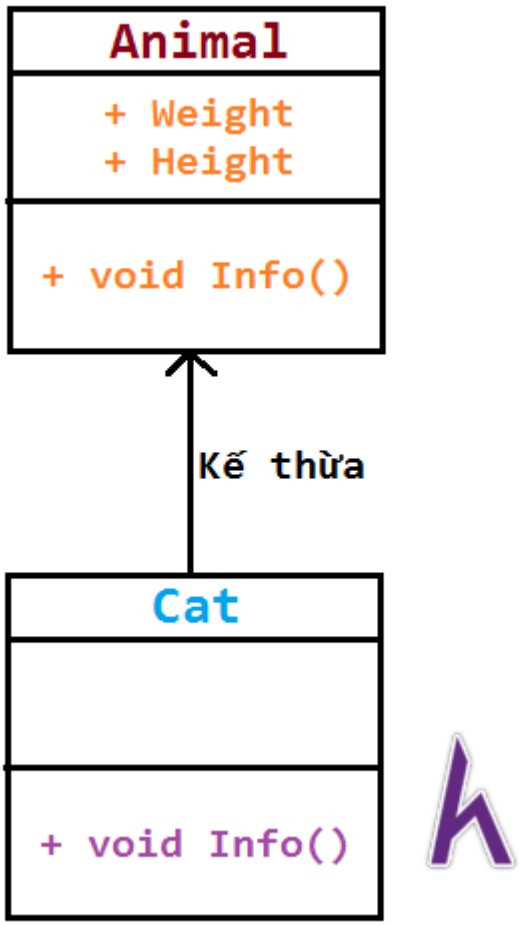
```
public <tên lớp>(<danh sách tham số của lớp con>) : base(<danh sách tham số>)

{

}
```

Trong đó:

- **<tên lớp>** là tên lớp con (lớp dẫn xuất).
- **<danh sách tham số của lớp con>** là danh sách tham số của constructor của lớp con.
- **base** là từ khoá để gọi đến constructor của lớp cha.
- **<danh sách tham số>** là danh sách tham số tương ứng với constructor của lớp cha.

Khi đối tượng của lớp con bị huỷ thì phương thức huỷ bỏ của nó sẽ được gọi trước sau đó mới gọi phương thức huỷ bỏ của lớp cha để huỷ những gì lớp con không huỷ được.

Vấn đề hàm trùng tên và cách gọi phương thức của lớp cha



Xét lại ví dụ phía trên. Giả sử lớp Animal có phương phức tên **Info()**. Lớp Cat kế thừa lớp Animal nên cũng sẽ nhận được phương thức này.

Bây giờ trong lớp Cat ta cũng định nghĩa 1 phương thức tên **Info()**, có kiểu trả về là void và không có tham số truyền vào. **Vậy câu lệnh sau sẽ gọi phương thức Info() nào?**

```
Cat BlackCat = new Cat();

BlackCat.Info();
```

Câu trả lời là C# sẽ gọi phương thức **Info()** của lớp Cat định nghĩa. Đồng thời cũng đưa ra 1 cảnh báo khi biên dịch.

Trong C# có hỗ trợ từ khoá new nhằm đánh dấu đây là 1 hàm mới và hàm kế thừa từ lớp cha sẽ bị che đi khiến **bên ngoài** không thể gọi được.

Cụ thể bạn sẽ thêm từ khoá new vào trước khai báo hàm Info() trong lớp Cat.

```
public new void Info()
{
}
```

Khi đó hàm **Info()** của lớp cha sẽ bị che giấu đi. Và mọi đối tượng **bên ngoài** chỉ gọi được hàm **Info()** của lớp Cat.

Từ khoá này chỉ làm tường minh khai báo của hàm **Info()** chứ về kết quả khi chạy chương trình sẽ không có thay đổi.

Đến đây 1 câu hỏi nữa được đặt ra: **Vậy có cách nào gọi hàm Info() của lớp cha được nữa không?**

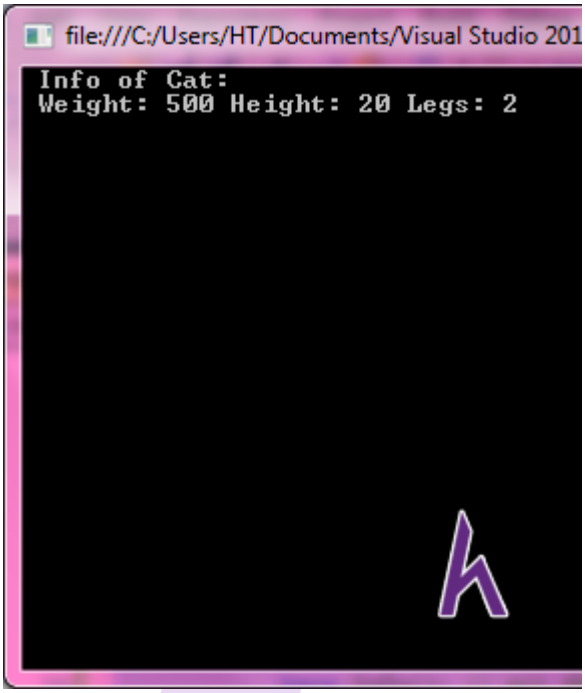
Câu trả lời là có nhưng chỉ có thể gọi trong nội bộ của lớp Cat mà thôi.

Bạn có thể sử dụng từ khoá base để đại diện cho lớp cha và gọi đến các thành phần của lớp cha.

Ví dụ: gọi hàm **Info()** của lớp cha:

```
/* Từ khoá new chỉ định đây là 1 hàm Info mới của lớp Cat */

    public new void Info()
    {

        Console.WriteLine(" Info of Cat: ");
        base.Info(); // gọi đến hàm Info() của lớp cha

    }
```

Kết quả khi gọi hàm **Info()** của lớp Cat là:



Vấn đề cấp phát vùng nhớ cho đối tượng

Bình thường nếu như 1 đối tượng kiểu Animal không thể khởi tạo vùng nhớ có kiểu Cat được.

```
Animal cat = new Cat();
```

Câu lệnh này sẽ báo lỗi: "không thể chuyển từ kiểu Cat sang kiểu Animal".

Nhưng nếu như 2 lớp này có quan hệ kế thừa thì điều này hoàn toàn được.

Tính chất này được phát biểu như sau:

"**Một đối tượng thuộc lớp cha có thể tham chiếu đến vùng nhớ của đối tượng thuộc lớp con nhưng ngược lại thì không**".

Có nghĩa là nếu lớp Cat kế thừa từ lớp Animal thì câu lệnh **Animal cat = new Cat();** hoàn toàn đúng nhưng ngược lại **Cat cat = new Animal ();** sẽ báo lỗi.

Bạn cần lưu ý điều này. Vì muốn thể hiện tính đa hình trong lập trình ta phải sử dụng tính chất này.

Kết luận

Nội dung bài này giúp các bạn nắm được:

- Khái niệm kế thừa.
- Khai báo và sử dụng kế thừa.
- Các vấn đề trong kế thừa.

Bài sau chúng ta sẽ tìm hiểu về ĐA HÌNH TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG .

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".