Bài 4: TỪ KHÓA STATIC TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Xem bài học trên website để ủng hộ Kteam: Từ khóa Static 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 CÁC LOẠI PHẠM VI TRUY CẬP TRONG C# OOP**.** Hôm nay chúng ta sẽ cùng tìm hiểu về **Từ khoá static trong lập trình hướng đối tượng 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#
- MẢNG MỘT CHIỀU TRONG C#

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

- Đặc điểm của thành viên tĩnh.
- Biến tĩnh.
- Phương thức tĩnh.

- Lớp tĩnh.
- Phương thức khởi tạo tĩnh.

Đặc điểm của thành viên tĩnh

Bình thường các thuộc tính, phương thức sẽ có đặc điểm:

- Chỉ có thể sử dụng sau khi khởi tạo đối tượng.
- Dữ liệu thuộc về riêng mỗi đối tượng (xét cùng 1 thuộc tính thì các đối tượng khác nhau thì thuộc tính đó sẽ mang các giá trị khác nhau).
- Được gọi thông qua tên của đối tượng.

Đôi lúc người lập trình mong muốn 1 thuộc tính nào đó được dùng chung cho mọi đối tượng (chỉ được cấp phát 1 vùng nhớ duy nhất). Từ đó khái niệm **thành viên tĩnh** ra đời.

Đặc điểm của thành viên tĩnh:

- Được khởi tạo **1 lần duy nhất** ngay khi biên dịch chương trình.
- Có thể **dùng chung** cho mọi đối tượng.
- **Được gọi thông qua tên lớp**.
- Được **huỷ khi kết thúc** chương trình.

Có 4 loại thành viên tĩnh chính:

- Biến tĩnh (static variable).
- Phương thức tĩnh (static method).
- Lớp tĩnh (static class).
- Phương thức khởi tạo tĩnh (static constructor).

Để khai báo 1 **thành viên tĩnh** ta sử dụng từ khoá **static** đặt trước tên biên, tên phương thức hoặc tên lớp. Chi tiết sẽ được trình bày ngày sau đây.

Biến tĩnh

Cú pháp:

```
<phạm vi truy cập> static <kiểu dữ liệu> <tên biến> = <giá trị khởi tạo>;
```

Trong đó:

- **<phạm vi truy cập>** là các phạm vi đã trình bày trong bài **CÁC LOẠI PHẠM VI TRUY CẬP TRONG C# OOP**
- **static** là từ khoá để khai báo thành viên tĩnh.
- **<kiểu dữ liệu>** là kiểu dữ liệu của biến (đã được trình bày trong bài **KIỂU DỮ LIỆU TRONG C#**).
- **<tên biến>** là tên biến do người dùng đặt và tuân thủ các quy tắc đặt tên biến (các quy tắc đặt tên biến đã trình bày trong bài **BIẾN TRONG C#**).
- **<giá trị khởi tạo>** là giá trị ban đầu mà biến tĩnh này chứa. Nếu bạn không khai báo giá trị này thì C# thì tự gán giá trị mặc định và đưa ra 1 cảnh báo khi bạn biên dịch chương trình.

Bạn có thể hiểu **biến tĩnh** là:

- Một biến dùng chung cho mọi đối tượng thuộc lớp.
- Nó được khởi tạo vùng nhớ 1 lần duy nhất ngay khi chương trình được nạp vào bộ nhớ để thực thi và huỷ khi kết thúc chương trình.

Ngoài biến tĩnh ra thì hằng số cũng có thể được gọi thông qua tên lớp và không cần khởi tạo đối tượng nhưng biến tĩnh linh hoạt hơn đó là có thể thay đổi giá trị khi cần thiết.

Về cách sử dụng thì bạn thao tác hoàn toàn giống 1 biến bình thường chỉ cần lưu ý là phải gọi biến này thông qua tên lớp.

Ví dụ:

Ta muốn quản lý số lượng mèo đang có (giả sử 1 đối tượng được tạo ra là 1 con mèo)

```
class Cat
```

```
    {
        private int weight;
        /*
            Khai báo property tương ứng với thuộc tính.
            Mặc dù trong bài này mình không sử dụng tới nhưng mình vẫn khai báo để
nhắc các bạn nhớ tuân thủ tính đóng gói.
         */
        public int Weight
        {
            get { return weight; }
            set { weight = value; }
        }
        private int height;
        public int Height
        {
            get { return height; }
            set { height = value; }
        }

        /*
            Khai báo 1 biến static tên Count để chứa số lượng mèo hiện tại.
            Mỗi lần có 1 đối tượng được tạo ra thì ta sẽ tăng Count lên.
        */

        public static int Count = 0;

        public Cat()
        {
            weight = 20;
            height = 500;
            /* Vì constructor chỉ được gọi khi có đối tượng được tạo ra nên ta sẽ tăng
Count ở đây */
            Count++;
        }
    }
```

Trong hàm **main**:

```
Console.WriteLine(" So luong meo ban dau: " + Cat.Count);
Cat BlackCat = new Cat(); // Tạo ra con mèo đầu tiên

Console.WriteLine(" So luong meo hien tai: " + Cat.Count);
```

```
Cat WhiteCat = new Cat(); // Tạo ra con mèo thứ 2
Console.WriteLine(" So luong meo hien tai: " + Cat.Count);
```

Kết quả khi chạy đoạn code trên:

Phương thức tĩnh

Cú pháp:

```
<phạm vi truy cập> static <kiểu trả về> <tên phương thức>
{
        // nội dung phương thức
}
```

Trong đó:

- **<phạm vi truy cập>** là các phạm vi đã trình bày trong bài **CÁC LOẠI PHẠM VI TRUY CẬP TRONG C# OOP**.
- **static** là từ khoá để khai báo thành viên tĩnh.
- **<kiểu trả về>** là kiểu trả về của phương thức (đã được trình bày trong bài **HÀM TRONG C#**).
- **<tên phương thức>** là tên do người dùng đặt và tuân thủ các quy tắc đặt tên (các quy tắc đặt tên đã trình bày trong bài **BIẾN TRONG C#**).

Hàm tĩnh được sử dụng với 2 mục đích chính:

- Hàm tĩnh là 1 hàm dùng chung của lớp. Được gọi thông qua tên lớp và không cần khởi tạo bất kỳ đối tượng nào, từ đó tránh việc lãng phí bộ nhớ.
- Hỗ trợ trong việc viết các hàm tiện ích để sử dụng lại.

Về sử dụng thì bạn thao tác hoàn toàn giống 1 phương thức bình thường chỉ cần lưu ý là phải gọi phương thức này thông qua tên lớp.

Ví dụ:

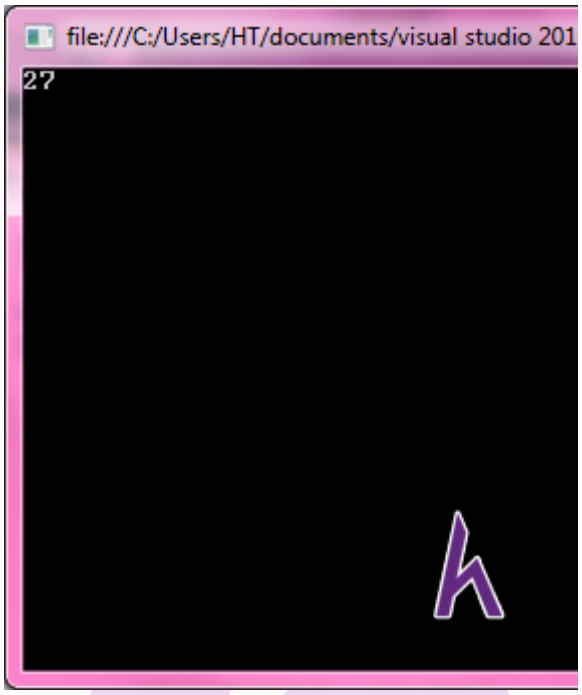
Bạn viết 1 hàm tiện ích đó là tính luỹ thừa của 1 số nguyên để hỗ trợ tính toán.

```
class TienIch
   {
      /*
         Khai báo và định nghĩa 1 phương thức tính luỹ thừa 2 số nguyên
       */

      public static long LuyThua(int CoSo, int SoMu)
      {
         long KetQua = 1;
         for (int i = 0; i < SoMu; i++)
         {
            KetQua *= CoSo;
         }

         return KetQua;
      }
   }
```

Trong hàm **main** ta thử gọi phương thức này ra dùng thử:

```
/*
Gọi phương thức thông qua tên lớp và không cần khởi tạo đối tượng.
*/
Console.WriteLine(TienIch.LuyThua(3, 3));
```

Kết quả khi chạy chương trình trên:



Lớp tĩnh

Cú pháp:

```
<phạm vi truy cập> static class <tên lớp>
{
        // các thành phần của lớp
}
```

Trong đó:

- **<phạm vi truy cập>** là các phạm vi đã trình bày trong bài **CÁC LOẠI PHẠM VI TRUY CẬP TRONG C# OOP**.
- **static** là từ khoá để khai báo thành viên tĩnh.
- **class** là từ khoá để khai báo lớp.
- **<tên lớp>** là tên do người dùng đặt và tuân thủ các quy tắc đặt tên (các quy tắc đặt tên đã trình bày trong bài **BIẾN TRONG C#**).

Lớp tĩnh có các đặc điểm

- Chỉ chứa các thành phần tĩnh (biến tĩnh, phương thức tĩnh).
- **Không** thể khai báo, khởi tạo 1 đối tượng thuộc lớp tĩnh.

Với 2 đặc điểm trên có thể thấy lớp tĩnh thường được dùng với mục đích khai báo 1 lớp tiện ích chứa các hàm tiện ích hoặc hằng số vì:

- Ràng buộc các thành phần bên trong lớp phải là **static**.
- Không cho phép tạo ra các đối tượng dư thừa làm lãng phí bộ nhớ.
- Mọi thứ đều được truy cập thông qua tên lớp.

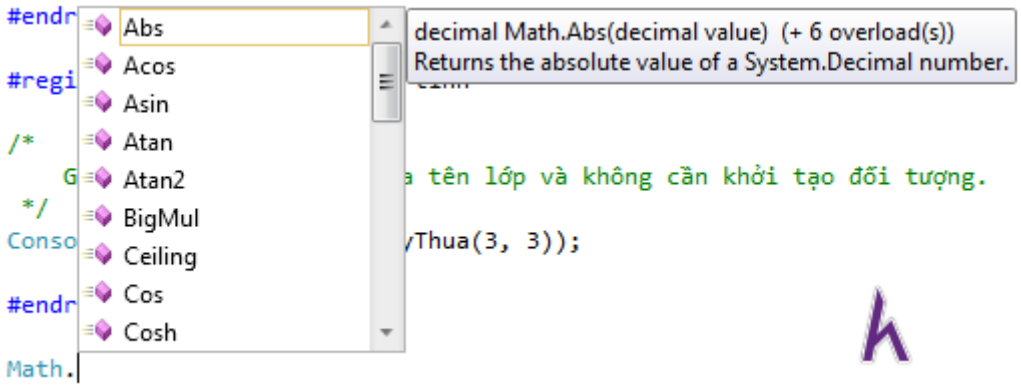
Xét lại ví dụ trong phần hàm tĩnh. Rõ ràng là người có thể vô ý tạo ra đối tượng thuộc **TienIch**. Đối tượng này khá vô nghĩa vì không có gì để sử dụng. Để tránh điều này ta thêm từ khoá **static** vào trước khai báo lớp.

Trong C# có rất nhiều lớp tiện ích sử dụng lớp tĩnh, phương thức tĩnh để khai báo. Ví dụ điển hình đó là lớp **Math**.

Lớp **Math** chứa:

- Các hằng số như **PI**, **E**.
- Các phương thức hỗ trợ tính toán như: **sin**, **cos**, **tan**, **sqrt**, **exp**, . . .

Bạn có thể tự khám phá chúng bằng cách gõ "**Math.**" để xem các thành phần của lớp **Math**.



Phương thức khởi tạo tĩnh

Cú pháp:

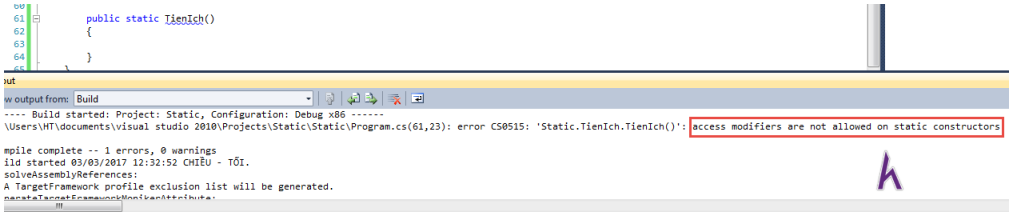
```
static <tên lớp>()
{
        // nội dung của constructor
}
```

Trong đó:

- **static** là từ khoá để khai báo thành viên tĩnh.
- **<tên lớp>** là tên của lớp chứa constructor này.

Đặc điểm của constructor tĩnh

- **Không được phép** khai báo phạm vi truy cập. Nếu cố tình làm điều này C# sẽ báo lỗi khi biên dịch.



- Constructor tĩnh sẽ **được gọi 1 lần duy nhất** khi chương trình được nạp vào bộ nhớ để thực thi như là 1 cách để ta thiết lập một số thông số theo ý muốn trước khi có bất kỳ đối tượng nào được tạo ra.
- Constructor tĩnh cũng giống phương thức tĩnh nên không thể gọi các thuộc tính không phải static.

Ví dụ:

Giả sử như bạn có 1 biến tĩnh chỉ định màu sắc chủ đạo của ngày và màu sắc chủ đạo này phụ thuộc vào hôm nay là thứ mấy.

- Thứ 2: màu xanh dương
- Thứ 3: màu đỏ
- Thứ 4: màu tím
- Thứ 5: màu hồng
- Thứ 6: màu đen
- Thứ 7: màu xanh lá
- Chủ nhật: màu vàng

Rõ ràng là bạn mong muốn khởi tạo giá trị cho biến tĩnh này nhưng:

- Không thể khởi tạo bằng cách gán trực tiếp như thế này:

```
public static string MauChuDao = "Red";
```

Vì ngoài màu đỏ ra thì còn có màu tím và màu này phụ thuộc vào ngày hiện tại.

- Không thể khởi tạo biến tĩnh này trong **constructor** bình thường được. Vì **constructor** bình thường chỉ được gọi khi có đối tượng được khởi tạo.

Trường hợp này thì **constructor tĩnh** là 1 giải pháp đơn giản nhưng hiệu quả.

```
class MauSac
   {
```

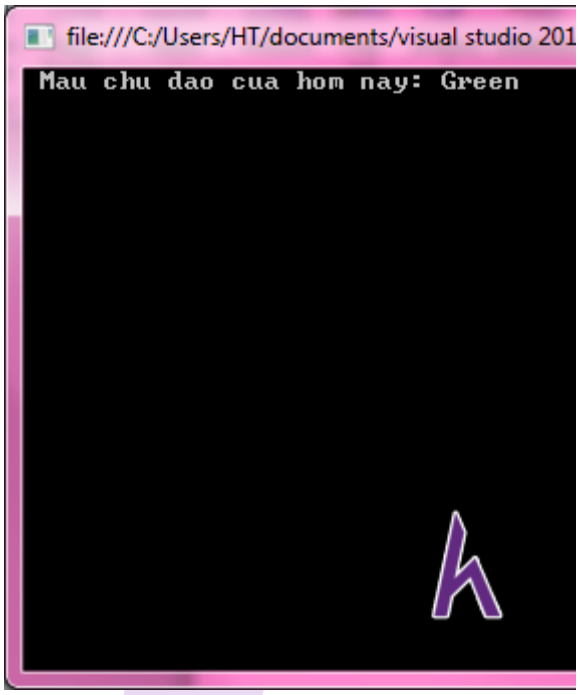
```
        /* Giả sử màu chủ đạo là 1 chuỗi ký tự lưu tên màu tương ứng */
        public static string MauChuDao;
        /* Dùng static constructor để kiểm tra ngày hiện tại và khởi tạo giá trị cho
biến tĩnh MauChuDao */
        static MauSac()
        {
            /* Khai báo đối tượng ngày giờ và lấy ngày giờ hiện tại của hệ thống */
            DateTime now = DateTime.Now;

            /* lấy ra thứ của ngày hiện tại và so sánh với 7 ngày trong tuần */
            switch (now.DayOfWeek)
            {
                case DayOfWeek.Friday:
                    MauChuDao = "Black";
                    break;
                case DayOfWeek.Monday:
                    MauChuDao = "Blue";
                    break;
                case DayOfWeek.Saturday:
                    MauChuDao = "Green";
                    break;
                case DayOfWeek.Sunday:
                    MauChuDao = "Yellow";
                    break;
                case DayOfWeek.Thursday:
                    MauChuDao = "Pink";
                    break;
                case DayOfWeek.Tuesday:
                    MauChuDao = "Red";
                    break;
                case DayOfWeek.Wednesday:
                    MauChuDao = "Purple";
                    break;
            }
        }
    }
```

Trong hàm **main** ta thử kiểm tra xem bằng cách gọi thuộc tính MauChuDao ra:

```
/* In ra màn hình giá trị của thuộc tính màu chủ đạo */
Console.WriteLine(" Mau chu dao cua hom nay: " + MauSac.MauChuDao);
```

Kết quả khi chạy đoạn code trên:

- Hôm nay đang là 04/02/2017 – Thứ 7



Kết luận

Nội dung bài này giúp các bạn nắm được:

- Đặc điểm của thành viên tĩnh.
- Biến tĩnh.
- Phương thức tĩnh.
- Lớp tĩnh.
- Phương thức khởi tạo tĩnh.

Bài sau chúng ta sẽ tìm hiểu về KẾ THỪA TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG C#.

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**".