THÀNH VIÊN TĨNH

Biến tĩnh (static field)

Trước khi đi vào nhu cầu tạo ra biến tĩnh, hãy cùng xây dựng một class như sau:

```
class Employee
{
   public Employee(string firstName, string lastName)
   {
      FirstName = firstName;
      LastName = lastName;
   }
   public int Id { get; set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
   public string Salary { get; set; } = "Not Enough";
}
```

Đây là một class đơn giản mô tả cho nhân viên công ty.

Bài toán đặt ra là, mỗi khi khởi tạo một object của Employee, object đó sẽ được gán một giá trị Id độc nhất. Ngoài ra, nếu liên tục tạo ra một loạt object mới (ví dụ, để lưu trong một mảng), giá trị của Id sẽ tăng dần. Nghĩa là object tạo sau có Id bằng Id của object vừa tạo trước cộng thêm 1. Nhu cầu tạo ra Id tự động như vậy rất giống với cách tăng tự động giá trị khóa chính của bảng dữ liệu Sql Server.

Một giải pháp bạn chắc chắn sẽ nghĩ tới là tạo ra một biến đếm (counter) độc lập nào đó (nằm ngoài class). Mỗi lần tạo ra object mới của Employee thì gán giá trị hiện thời của counter cho Id, sau đó tăng counter thêm 1. Vấn đề khi đó nằm ở chỗ, bạn phải liên tục truyền biến counter này đến cho bất kỳ chỗ nào thực hiện khởi tạo object.

Bây giờ hãy thực hiện thay đổi nhỏ sau trong class Employee:

```
class Employee
{
    public Employee(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
        Id = NextId;
        NextId++;
    }
    public static int NextId;
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
```

```
    public string Salary { get; set; } = "Not Enough";
}
```

Bạn chỉ thêm lời khai báo:

```
public static int NextId;
```

Sau đó bổ sung thêm hai lệnh vào hàm dựng:

```
Id = NextId;
NextId++;
```

Khai báo biến NextId ở trên là một lệnh **khai báo biến tĩnh** (static field). Dễ thấy, biến tĩnh về mặt hình thức khác biệt với biến thành viên ở mỗi từ khóa `static`. Tuy nhiên, tác dụng của nó lại hoàn toàn khác.

Biến tĩnh NextId được dùng chung trong mọi object của Employee. Nghĩa là dù bạn khởi tạo bao nhiêu object đi nữa, tất cả các object đó đều có chung biến NextId này. Tức là sự tồn tại của NextId thực sự độc lập với các object của Employee.

Đối với biến thành viên, nếu bạn khởi tạo một object, một biến thành viên mới sẽ được tạo ra (có vùng nhớ riêng của nó). Biến thành viên đó sẽ được khởi tạo giá trị của riêng nó.

Đối với biến tĩnh, trong lần truy xuất đầu tiên sẽ tạo ra vùng nhớ cho biến. Tất cả các object tạo ra sau đó đều sử dụng chung vùng nhớ này. Do đó, biến tĩnh chỉ có 1 bản duy nhất.

Bạn đã thấy sự tương đồng với ý tưởng tạo biến counter độc lập chưa ạ! Sự khác biệt là, biến counter độc lập đó (NextId) giờ được đặt thẳng trong class. Tất cả các thành viên của Employee tự động sử dụng được NextId. Bạn không cần phải mất công truyền qua truyền lại nữa, cũng không cần tác động gì từ bên ngoài. Bằng cách này, bạn thậm chí còn đếm được là hiện tại có bao nhiêu object của Employee đã được tạo ra.

Giờ hãy viết một số client code để thử nghiệm lớp Employee với trường `static NextId`:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Employee.NextId = 1000000;
        Employee employee1 = new Employee("Inigo", "Montoya");
        Employee employee2 = new Employee("Princess", "Buttercup");
        Console.WriteLine("{0} {1} ({2})", employee1.FirstName, employee1.LastName, employee1.Id);
        Console.WriteLine("{0} {1} ({2})", employee2.FirstName, employee2.LastName, employee2.Id);
        Console.WriteLine($"NextId = { Employee.NextId }");
        Console.ReadKey();
    }
    // ... khai báo của lớp Employee nằm đây, bỏ qua cho gọn code
}
```

Chạy thử chương trình bạn sẽ thấy object đầu tiên có Id = 1000000. Object thứ hai Id = 1000001. Sau đó NextId lại tăng lên 1000002.

Về mặt cú pháp khai báo, biến `static` chỉ khác biệt duy nhất ở từ khóa `static` nằm trước tên kiểu.

Khi sử dụng trong client code, bạn chỉ có thể truy xuất biến static thông qua tên class, không phải thông qua tên object như đối với biến thành viên. Như trong ví dụ, bạn truy xuất biến static `NextId` trong phương thức `Main` là `Employee.NextId`. Bạn không thể truy xuất theo kiểu `employee2.NextId` được.

Chỉ khi nào bạn sử dụng biến tĩnh bên trong class, bạn có thể bỏ qua tên class. Khi đó, về mặt hình thức, nó không có gì khác biệt với biến thành viên. Đây là tình huống bạn đã sử dụng `NextId` bên trong constructor.

Lưu ý rằng, trong cùng một class không thể khai báo biến static và biến thành viên trùng tên. Bạn hẳn sẽ thấy ngay, nếu dùng trong class, compiler sẽ không hiểu bạn muốn dùng biến nào.

Trong cuộc sống bạn cũng có thể thấy những ví dụ tương đồng với biến thành viên và biến static. Lấy ví dụ về khuôn đúc và sản phẩm đúc từ khuôn. Dễ hình dung, khuôn đúc tương đương với class, còn sản phẩm đúc tương đương với object.

Mỗi sản phẩm đúc có những thông tin của riêng nó. Những thông tin riêng biệt này tương tự như biến thành viên, vốn chứa thông tin về trạng thái và đặc điểm riêng của object (sản phẩm đúc).

Tuy nhiên, lại có những thông tin phải liên kết với chính khuôn đúc. Ví dụ số lượng sản phẩm đã đúc từ khuôn, số series của sản phẩm tiếp theo, công suất đúc (bao nhiêu sản phẩm mỗi giờ). Rõ ràng những thông tin này không liên quan đến từng sản phẩm cụ thể, mà liên quan đến chính khuôn đúc.

Như vậy, dữ liệu không nhất thiết chỉ liên kết với object mà còn có thể liên kết với chính class. Biến static chính là loại thông tin liên kết với bản thân class.

Về hằng thành viên của class
Hằng thành viên của class C# có chút đặc biệt mà ít người để ý. Khi khai báo một hằng thành viên, hằng đó sẽ được C# tự động coi là một thành viên static giống như biến static. Sự khác biệt duy nhất là giá trị của nó phải được xác định ngay từ lúc khai báo và sau này không thể thay đổi được nữa. Hằng thành viên khai báo giống hệt như biến thành viên, ngoại trừ từ khóa `const` ở trước:

```
public const int MaxValue = 12345;
```

Phương thức tĩnh (static method)

Tương tự như biến tĩnh, phương thức tĩnh liên kết với bản thân class chứ không liên kết với object của class đó. Do đó, cũng chỉ có 1 phiên bản duy nhất của phương thức đó được tạo ra.

Về cú pháp khai báo, phương thức tĩnh khác biệt với phương thức thành viên duy nhất ở từ khóa static đặt trước tên kiểu trả về.

Hãy cùng thực hiện ví dụ sau:

```
class ConsoleHelper
{
    public ConsoleColor BackgroundColor { get; set; } = ConsoleColor.Black;
    public ConsoleColor ForegroundColor { get; set; } = ConsoleColor.White;
    public void WriteLine(object message)
    {
        WriteLine(message, ForegroundColor, BackgroundColor);
    }
    public static void WriteLine(object message,
                            ConsoleColor fgColor = ConsoleColor.White,
                            ConsoleColor bgColor = ConsoleColor.Black,
                            bool reset = true)
    {
        Console.ForegroundColor = fgColor;
        Console.BackgroundColor = bgColor;
        Console.WriteLine(message);
        if (reset)
            Console.ResetColor();
    }
}
```

Ví dụ trên khai báo `class ConsoleHelper` với một phương thức `static WriteLine`, một phương thức thành viên cùng tên `WriteLine`. Hai phương thức này cùng hỗ trợ viết thông tin ra màn hình console nhưng có thể thiết lập màu nền và màu văn bản. Trong class này cũng chứa hai property thành viên `BackgroundColor` và `ForegroundColor`.

Phương thức static được gọi trực tiếp từ tên class, so với phương thức thành viên được gọi từ tên object. Do đó, để gọi phương thức static bạn không cần khởi tạo object.

```
static void Main(string[] args)
{
    // gọi phương thức tĩnh WriteLine
    ConsoleHelper.WriteLine("Hello world!", ConsoleColor.Magenta, ConsoleColor.White);
    // khởi tạo object và gọi phương thức thành viên WriteLine
    var helper = new ConsoleHelper { BackgroundColor = ConsoleColor.Black, ForegroundColor = Conso
    helper.WriteLine("Hello again!");
    Console.ReadKey();
}
```

Bởi vì phương thức static không được sử dụng qua object (và cũng không liên quan gì đến object), bạn không được sử dụng từ khóa **this** bên trong phương thức static. Cũng vì lý do này, phương thức static không thể sử dụng được các thành viên bình thường (như biến thành viên, phương thức thành viên) khác bên trong class.

Trong ví dụ trên, phương thức tĩnh WriteLine không thể sử dụng được `BackgroundColor` và `ForegroundColor` vì đây là hai property thành viên.

Tuy nhiên, phương thức static vẫn có thể khởi tạo và sử dụng object của chính class chứa nó. Ví dụ, lệnh khởi tạo sau là hoàn toàn hợp lệ bên trong thân của WriteLine:

```
var helper = new ConsoleHelper();
```

Ở chiều ngược lại, các phương thức thành viên hoàn toàn có thể sử dụng phương thức tĩnh. Như trong ví dụ trên, phương thức thành viên WriteLine đã gọi phương thức static WriteLine.

Nhìn chung, phương thức tĩnh có vai trò rất gần với phương thức toàn cục trong C/C++. Sự khác biệt duy nhất là nó được quản lý tốt hơn.

Nếu một phương thức không sử dụng bất kỳ thành viên nào khác của class, bạn nên đặt nó làm phương thức tĩnh. Cách làm này sẽ hiệu quả hơn so với đặt nó làm phương thức thành viên vì sẽ chỉ có duy nhất 1 phiên bản của phương thức được tạo ra và quản lý. Nếu để phương thức tương tự làm phương thức thành viên, mỗi object sẽ phải chứa một phiên bản riêng của nó.

Intellisense của Visual studio có thể tự động phát hiện và gợi ý chuyển đổi phương thức thành dạng static theo logic trên.

Hàm dựng tĩnh (static constructor)

Quay trở lại ví dụ minh họa của phần biến static. Bài toán đặt ra là: làm sao để gán giá trị đầu của biến tĩnh NextId là một giá trị ngẫu nhiên nhưng không dùng client code? Để tạo giá trị ngẫu nhiên, bạn có thể sử dụng class Random như sau:

```
Random randomGenerator = new Random();
NextId = randomGenerator.Next(101, 999);
```

tức là bạn cần đến vài lệnh tính toán thì mới tạo ra được một giá trị ngẫu nhiên.

Hãy nghĩ đến một tình huống phức tạp hơn. Giả sử bạn lưu dữ liệu employee vào file. Khi chương trình hoạt động, bạn cần lấy giá trị lớn nhất của Id đã lưu trong file để tiếp tục tăng giá trị của Id theo quy luật chứ không muốn gán giá trị đầu ngẫu nhiên.

Để giải quyết tình huống khởi tạo giá trị đầu (mà phải tính toán phức tạp) cho biến static, C# sử dụng **hàm dựng tĩnh** (static constructor).

Hãy bổ sung hàm dựng sau vào lớp Employee:

```
class Employee
{
    static Employee()
    {
        Random randomGenerator = new Random();
        NextId = randomGenerator.Next(101, 999);
    }
    // ...
```

```
    public static int NextId = 42;
    // ...
}
```

Đây là khai báo cho hàm dựng static của lớp Employee. Về cú pháp, nó khác hàm dựng (constructor) bình thường ở từ khóa static đứng tên.

Hàm dựng static khác biệt với hàm dựng thông thường ở một số điểm:

- Hàm dựng không cần dùng từ khóa điều khiển truy cập. Lý do là vì hàm dựng static không thể gọi từ client code. Hàm dựng này được chương trình tự động gọi khi cần thiết. Chính xác hơn là nó được gọi khi sử dụng class lần đầu tiên trong chương trình.
- Hàm dựng static không chấp nhận tham số. Lý do giống như ở trên: static constructor được gọi tự động.

Hàm dựng static được sử dụng để khởi tạo giá trị cho các trường static của class, đặc biệt là khi phải thực hiện những tính toán phức tạp.

Nếu bạn đồng thời gán giá trị cho trường static lúc khởi tạo và trong hàm dựng static, giá trị trong hàm dựng static sẽ là giá trị chính thức của trường static.

Nếu có thể, hãy gán giá trị đầu cho trường static lúc khai báo thay vì gán trong constructor.

Đặc tính tĩnh (static property)

Như bạn đã biết, property thực chất là tổ hợp hai phương thức get-set để kiểm soát xuất nhập giá trị cho một trường dữ liệu. Như vậy, nếu đã có biến static và phương thức static thì cũng có khái niệm static property với cùng ý nghĩa: tổ hợp hai phương thức tĩnh get – set để kiểm soát xuất nhập giá trị cho biến static.

Hãy bổ sung thêm đoạn code sau vào class Employee mà bạn đã xây dựng từ trước:

```
class Employee
{
    // ...
    public static int NextId
    {
        get {
            return _nextId;
        }
        private set {
            _nextId = value;
        }
    }
    private static int _nextId = 42;
    // ...
}
```

Đây là cách khai báo static property NextId để xuất nhập dữ liệu cho biến static _nextId. Cách khai báo static property không có gì khác biệt so với property thành viên, ngoại trừ từ khóa static.

Bạn thậm chí có thể khai báo auto static property như sau:

```
public static int NextId { get; private set; } = 42;
```

Static property được sử dụng qua tên class giống như biến static và phương thức static.

Tương tự như đặc tính thành viên, bạn nên sử dụng static property thay cho biến static public.

Lớp tĩnh (static class)

Một số class được tạo ra nhưng không chứa bất kỳ biến thành viên nào. Lấy ví dụ, nếu bạn muốn xây dựng một class chuyên thực hiện các hàm tính toán số học, bạn chẳng cần biến thành viên nào cả. Hãy cùng thực hiện một class như vậy:

```
public static class MyMath
{
    public static int Max(params int[] numbers)
    {
        if (numbers.Length == 0)
        {
            throw new ArgumentException("Không có giá trị để so sánh", "numbers");
        }
        int max = numbers[0];
        foreach (var number in numbers)
        {
            if (number > max)
            {
                max = number;
            }
        }
        return max;
    }
    public static int Min(params int[] numbers)
    {
        if (numbers.Length == 0)
        {
            throw new ArgumentException("Không có giá trị để so sánh", "numbers");
        }
        int min = numbers[0];
        foreach (var number in numbers)
        {
            if (number < min)
            {
                min = number;
            }
```

```
        }
        return min;
    }
}
```

Trong class này khai báo hai phương thức static Min và Max để tìm giá trị nhỏ nhất và lớn nhất trong một mảng số nguyên.

Lớp MyMath không chứa bất kỳ biến hoặc phương thức thành viên thường nào. Do đó, việc khởi tạo object của nó khá vô nghĩa. Do vậy nó được khai báo làm lớp static với từ khóa static đứng trước từ khóa class như bạn đã thấy.

Từ khóa static đứng trước khai báo class có mấy tác dụng:

- Thứ nhất nó không cho phép khởi tạo object từ class này.
- Thứ hai, nó không cho phép khai báo bất kỳ thành viên thông thường nào mà chỉ có thể khai báo các thành viên tĩnh.

Khi một class được đánh dấu là static, C# compiler tự động đánh dấu nó là abstract và sealed, nghĩa là cấm khởi tạo object và không cho phép kế thừa.

Một đặc điểm nữa của lớp static là bạn có thể sử dụng cấu trúc using static để trực tiếp truy xuất các thành viên static của class này mà không cần chỉ rõ tên class, nghĩa là nếu ở đầu file code có lệnh `using static MyMath;` thì bạn có thể gọi các phương thức của class này theo cách ngắn gọn `Max(numbers);` thay cho `MyMath.Max(numbers);`.

```
using static Console;
using static MyMath;
class Program
{
    static void Main(string[] args)
    {
        int[] numbers = new[] { 1, 2, 3, 4, 5, 6 };
        // có thể gọi Max và Min theo cách ngắn gọn vì đã có using static MyMath; ở trên
        int max = Max(numbers);
        int min = Min(numbers);
        // có thể gọi WriteLine ngắn gọn vì đã có using static Console; ở trên
        WriteLine($"Max value: {max}");
        WriteLine($"Min value: {min}");
        ReadKey();
    }
}
```

Lớp System.Console mà bạn đã biết cũng là một static class. Vì vậy, nếu đặt using `static System.Console;` ở đầu file code thì có thể gọi các phương thức trong đó một cách ngắn gọn: `WriteLine("Hello world");` thay vì `Console.WriteLine("Hello world");`

Extension method

Extension method (phương thức mở rộng) là một tính năng rất thú vị của C# cho phép "chèn" một phương thức của class này vào làm phương thức thành viên một class khác.

Để cho dễ hiểu, hãy tưởng tượng thế này. Lớp string mà bạn đã biết mặc dù cung cấp rất nhiều phương thức hữu ích nhưng nếu bạn làm việc nhiều với giao diện console, hẳn bạn sẽ muốn class này có một phương thức giúp xuất trực tiếp chuỗi ra console, thay vì phải liên tục gọi đến Console.Write/WriteLine. Hoặc bạn cũng có thể muốn lớp này hỗ trợ luôn việc chuyển đổi chuỗi thành các kiểu dữ liệu cơ sở quen thuộc, thay vì phải gọi Parse/TryParse của kiểu đích.

Bắt đầu từ C# 3 bạn đã có thể thực hiện mong muốn trên bằng cách sử dụng extension method. Extension method thực chất chỉ là một static method nằm trong một static class cùng với một thay đổi nhỏ trong danh sách tham số.

Hãy cùng thực hiện một ví dụ nhỏ sau cho dễ hiểu.

```
static class ExtensionMethods
{
    public static void ToConsole(this string message)
    {
        Console.WriteLine(message);
    }
    public static void ToConsole(this string message, ConsoleColor fgColor = ConsoleColor.White, C
    {
        Console.ForegroundColor = fgColor;
        Console.BackgroundColor = bgColor;
        Console.WriteLine(message);
        if (reset) Console.ResetColor();
    }
    public static double ToDouble(this string number)
    {
        return double.TryParse(number, out double d) ? d : double.NaN;
    }
    public static int ToInt(this string number)
    {
        return int.Parse(number);
    }
}
```

Ví dụ trên xây dựng một `static class ExtensionMethods` với hai static method cùng tên `ToConsole`. Hai phương thức này cùng thực hiện in một thông báo ra màn hình. Overload thứ hai in ra có màu nền và màu văn bản. `ToDouble` và `ToInt` thực hiện chuyển đổi chuỗi thành kiểu `double` và `int`.

Hãy để ý tham số thứ nhất của cả bốn phương thức. Chúng cùng có dang `this string <tên-biến>`. Để ý thấy rằng, tham số này khác thường một chút ở từ khóa `this`. Chỉ một từ khóa đó làm cho hai phương thức có năng lực đặc biệt: có thể gọi chúng từ một `string` bất kỳ. Hãy cùng xem client code:

```
static void Main(string[] args)
{
    "Hello world!".ToConsole();
    "Hello again!".ToConsole(ConsoleColor.Magenta);
    int i = "2000".ToInt();
    double d = "2000.0001".ToDouble();
    Console.ReadKey();
}
```

Bạn đã thấy, ToConsole(), ToInt(), ToDouble() giờ được gọi thẳng từ object của string, giống hệt như gọi các phương thức thành viên khác của string. Nếu chỉ nhìn lời gọi phương thức ở client code, bạn không phân biệt được đâu là phương thức thành viên "xịn" của class, đâu là extension method.

Dĩ nhiên, bạn hoàn toàn có thể gọi extension method như các static method thông thường:

```
ExtensionMethods.ToConsole("Hiiiiiiiiiiiiiiiiiiii!");
```

Lưu ý, nếu bạn tạo một extension method có signature giống hệt như một phương thức thành viên có sẵn của class, bạn sẽ không gọi nó qua object được mà chỉ có thể gọi như phương thức static thông thường.

Nếu muốn tạo extension method cho kiểu nào, bạn đặt tên kiểu đó sau từ khóa this của tham số đầu tiên. Trong ví dụ trên, bạn muốn tạo extension method cho lớp string thì tham số đầu tiên phải là `this string <tên-tham-số>`. Bạn có thể sử dụng tham số này trong thân method như bất kỳ tham số bình thường nào.

Bạn có thể tạo extension method cho cả các class có sẵn của C# cũng như class tự xây dựng. Cách thực hiện không có gì khác biệt nhau.

Mặc dù extension method là một tính năng rất thú vị, bạn không nên lạm dụng nó, nhất là khi áp dụng cho các class không phải do bạn xây dựng. Extension method có vấn đề trong việc quản lý phiên bản (versioning). Đặc biệt **không** nên tạo extension method cho kiểu `object`.