# BIẾN THÀNH VIÊN

Như bạn đã biết, trong lớp có các thành viên chứa thông tin và thành viên chịu trách nhiệm xử lý thông tin. Biến thành viên (member variable) là loại thành viên có nhiệm vụ chứa thông tin của class. Với đặc điểm "strong và static typing" của C#, biến thành viên bắt buộc phải thuộc về một kiểu dữ liệu xác định. Học cách sử dụng biến thành viên và kiểu dữ liệu là bước đi đầu tiên của chúng ta để xây dựng các class hữu ích trong C#.

Bài học sẽ giúp bạn nắm được khái niệm và cách làm việc với biến thành viên của class. Bài học cũng sẽ giúp bạn làm quen với một số kiểu dữ liệu cơ bản của C#, vốn không thể không biết khi khai báo và sử dụng biến thành viên và thuộc tính.

## Biến thành viên của lớp

Biến thành viên (member variable) là thành phần chứa dữ liệu của class trong C#. Nó được khai báo trực tiếp trong thân của class (phải nằm trực tiếp trong khối code của thân class). Dữ liệu này có thể được sử dụng bởi bất kỳ thành phần nào khác của class. Tùy vào thiết lập, dữ liệu này cũng có thể được sử dụng bởi thành phần bên ngoài class.

Trong C#, nếu một biến được khai báo nằm trong thân của phương thức, nó được gọi là biến cục bộ (local variable). Chúng ta xem xét biến cục bộ và phương thức ở bài học khác.

Biến thành viên thường được gọi ngắn gọn là trường (field), hay trường dữ liệu (data field).

Biến thành viên được xem là nơi lưu trữ trạng thái (status) của lớp. Thay đổi giá trị biến thành viên là thay đổi trạng thái của lớp.

### Cú pháp khai báo biến thành viên

Quy tắc khai báo biến thành viên như sau:

```
[public|private|protected] <kiểu-dữ-liệu> <tên-biến> [= <giá-trị>];
```

Trong đó:

`public`, `private` và `protected` là các *từ khóa điều khiển truy cập* (access modifier) cho biến thành viên, cụ thể:

- `public`: biến có thể được truy xuất từ bên ngoài class
- `private` (mặc định): biến sử dụng hoàn toàn nội bộ, cũng như không được truyền thừa
- `protected`: biến sử dụng nội bộ và có thể truyền thừa

Nếu không ghi rõ từ khóa, C# mặc định sẽ hiểu đó biến đó là `private`, nghĩa là chỉ có thể sử dụng nội bộ trong class, bên ngoài không thể truy xuất giá trị này. Sự khác biệt giữa ba từ khóa này sẽ thể hiện rõ ràng hơn ở bài học về kế thừa.

Khái niệm "trong" và "ngoài" class có thể hiểu đơn giản như sau: "trong" class nghĩa là tất cả code nằm trong khối code thân class. Những code nào không nằm trong khối code thân class được coi là "ngoài". Code trong class đều có thể "nhìn thấy" các biến thành viên và sử dụng chúng. Code ở bên ngoài class chỉ có thể nhìn thấy và truy xuất các thành viên đặt là public.

Xem phần ví dụ về khai báo biến thành viên ở cuối bài học.

## Kiểu dữ liệu

*Kiểu dữ liệu* có thể là những kiểu dữ liệu do C# định nghĩa sẵn (int, bool, string, v.v.) hoặc do người dùng tự định nghĩa.

C# đã định nghĩa sẵn các kiểu dữ liệu cơ bản, tương tự trong nhiều ngôn ngữ lập trình khác.

C# cũng cho phép người dùng định nghĩa thêm các kiểu dữ liệu có cấu trúc của riêng mình, như mảng (array), bản ghi (struct), liệt kê (enum), lớp (như chúng ta đang làm).

Bộ thư viện .NET framework cung cấp một số lượng kiểu dữ liệu (lớp) khổng lồ có thể sử dụng trong các khai báo này. Trong quá trình học bạn đã lần lượt gặp các kiểu dữ liệu nói trên, cũng như tự định nghĩa ra các kiểu dữ liệu của riêng mình.

Các kiểu dữ liệu cơ sở của C# được Visual Studio hiển thị màu sắc tương tự như từ khóa; các kiểu dữ liệu do người dùng định nghĩa được hiển thị tương tự như class.

## Quy ước đặt tên biến thành viên

*Tên biến* do người lập trình tự chọn và tuân thủ theo quy tắc đặt định danh giống như đối với tên class. Ngoài ra có một số quy ước (không bắt buộc nhưng nên tuân thủ để thống nhất) sau về cách đặt tên biến:

1. Tên biến `public` và `protected` nên bắt đầu bằng chữ cái in hoa và tuân theo cách viết **PascalCase**;
2. Tên biến `private` nên bắt đầu bằng ký tự gạch chân và tuân theo cách viết **camelCase**;

Quy ước đặt tên biến thành viên khác với quy ước đặt tên biến cục bộ (biến khai báo trong thân phương thức). Biến cục bộ đều viết theo kiểu camelCase.

## Gán giá trị cho biến thành viên, giá trị mặc định

Khi khai báo một biến thành viên, C# sẽ tự động gán cho biến này một **giá trị mặc định** (lúc khởi tạo object) tùy thuộc vào kiểu dữ liệu: giá trị 0 đối với kiểu số nguyên, false đối với kiểu bool. Ví dụ:

```
int a; // a tự nhận giá trị 0
bool b; // b tự nhận giá trị false
```

Nếu muốn gán một giá trị khác, người dùng có thể sử dụng *biểu thức gán* (assignment operator) để gán giá trị cho biến ngay trong lúc khai báo.

```
int a = 10;
bool b = true;
```

## Ví dụ về cách khai báo biến thành viên

Trong phần này bạn sẽ thực hiện một ví dụ nhỏ về khai báo biến thành viên cho class để nắm rõ hơn các vấn đề lý thuyết trình bày ở trên.

Tạo một solution S02_MemberVariable với 1 project P01_VariableSample (Console App) và viết code như sau:

```
using System;
namespace P01_VariableSample
{
    class Program
    {
        static void Main(string[] args)
        {
            // khởi tạo biến kiểu Car
            Car bmw = new Car();
            // để ý là phương thức Main thuộc lớp Program nằm "bên ngoài" class Car,
            // do đó nó chỉ "nhìn thấy" các biến thành viên public của Car
            bmw.Color = "White";
            bmw.Seats = 2;
            // các biến thành viên private khác của Car hoàn toàn "vô hình" với Program.
            Console.WriteLine("A new " + bmw.Color + " BMW");
            Console.ReadKey();
        }
    }
    /// <summary>
    /// Lớp mô tả ô tô
    /// </summary>
    class Car
    {
        // dưới đây là các trường private (chứa thông tin nội bộ của class)
        // các trường này không được gán giá trị đầu, chúng sẽ nhận giá trị mặc định theo kiểu
        // lưu ý cách đặt tên: có dấu gạch chân ở đầu, bắt đầu là chữ cái thường, camelCase, báo
        string _make; // nhãn hiệu, giá trị mặc định theo kiểu là null
        int _yearOfProduction; // năm sản xuất, giá trị mặc định theo kiểu là 0
        bool _hasInsurance; // có mua bảo hiểm hay không, giá trị mặc định theo kiểu là false
        // dưới đây là các trường public (các class khác có thể đọc/ghi giá trị)
        // mỗi trường sẽ được gán giá trị đầu
        // lưu ý cách đặt tên: bắt đầu là chữ cái hoa
        public int Seats = 5; // số chỗ ngồi, gán giá trị đầu là 5
        public string Color = "Black"; // màu sắc, giá trị đầu là "Black"
    }
}
```

Dịch và chạy debug (F5) để xem kết quả.

Trong phần thực hành này, bạn đã xây dựng một class Car với một số biến thành viên mô tả cho một chiếc ô-tô.

Bạn đã khai báo các biến private: _make, _yearOfProduction, _hasInsurance; Các biến public Seats và Color.

Hãy lưu ý cách đặt tên các biến này. _make, _yearOfProduction va _hasInsurance là các biến private nên đặt tên theo quy ước camelCase với ký tự _ ở đầu. Seats và Color là các biến public nên đặt tên theo quy ước PascalCase.

Bạn cũng để ý rằng, trong phương thức Main (nơi khởi tạo và sử dụng object của Car) bạn chỉ có thể truy xuất biến Seats và Color. Bạn không thể truy xuất các biến private từ bên ngoài class.

Biến thành viên của Car hoàn toàn có thể tham gia vào các lệnh và biểu thức giống hệt như khi bạn sử dụng biến cục bộ.

## Một số vấn đề khi sử dụng biến thành viên

### Hạn chế sử dụng biến thành viên public

Trong lập trình hướng đối tượng, nhìn chung đều khuyến nghị hạn chế sử dụng biến thành viên public.

Các biến thành viên thường đường khai báo là private/protected, sau đó viết các phương thức xuất/nhập dữ liệu riêng cho từng biến (gọi là các hàm getter/setter). Cách thức này giúp kiểm soát được giá trị xuất nhập cho các biến.

C# cung cấp một tính năng riêng giúp giải quyết vấn đề này và chúng ta sẽ xem xét trong bài Property.

Như trong ví dụ trên bạn khai báo hai biến public Seats và Color. Ví dụ này hoàn toàn mang tính chất minh họa. Nếu xây dựng class thực sự, bạn không nên khai báo như vậy mà cần dùng đặc tính (Property) thay cho hai biến public này.

### Biến thành viên chỉ đọc, từ khóa readonly

Mặc định C# cho phép tự do thay đổi giá trị của biến thành viên. Nếu biến public, có thể thay đổi giá trị của nó ở trong hoặc ngoài class. Nếu là biến protected/private thì chỉ có thể thay đổi giá trị của nó ở bên trong class.

Tuy nhiên, có một số trường hợp bạn muốn rằng giá trị của biến sau khi khởi tạo sẽ không thay đổi được nữa. Tức là, nó biểu hiện gần giống như hằng số.

Vậy tại sao không sử dụng hằng?

Trong C#, hằng thành viên của class (và struct) có chút hơi khác biệt. Hằng thành viên được xem là một thành viên tĩnh (static) của class. Bạn sẽ học về thành viên tĩnh trong một bài học khác. Đặc điểm này khiến hằng không phù hợp với vai trò lưu trữ dữ liệu chỉ đọc cho object của class.

C# cung cấp một từ khóa riêng để tạo ra các biến chỉ đọc: từ khóa `readonly`. Từ khóa này cần đặt trước tên kiểu khi khai báo biến thành viên.

Hãy cùng thực hiện ví dụ sau để hiểu rõ hơn về biến readonly

```
using static System.Console;
namespace P02_ReadOnly
{
    class Product
    {
        // đây là các biến readonly, chỉ có thể khởi tạo giá trị trực tiếp ở đây
        public readonly double UnitPrice = 100;
        public readonly double Discount = 0.1;
        // đây là một biến thông thường, có thể tự do thay đổi giá trị
        public int Amount;
        public Product()
        {
        }
        public Product(double unitPrice, double discount)
        {
            // hoặc thay đổi giá trị trong constructor
            // một khi đã khởi tạo giá trị, biến readonly không thay đổi giá trị được nữa
            UnitPrice = unitPrice;
            Discount = discount;
            Amount = 0;
        }
        /* Phương thức này lỗi.
         * UnitPrice và Discount là biến readonly.
         * Không thể thay đổi giá trị của biến readonly trong phương thức
         *
        public void Reset()
        {
            UnitPrice = 0;
            Discount = 0;
        }
        */
    }
    class Program
    {
        static void Main(string[] args)
        {
            var product = new Product();
            WriteLine($"Unit Price = {product.UnitPrice}");
            // Lệnh này lỗi. Không thể thay đổi giá trị biến readonly
            //product.UnitPrice = 300;
            product = new Product(200, 0.2);
            WriteLine($"Unit Price = {product.UnitPrice}");
            ReadKey();
        }
    }
}
```

Biến readonly có đặc điểm:

- Chỉ có thể gán giá trị một lần duy nhất lúc khai báo, hoặc trong hàm tạo (constructor).
- Không thể gán giá trị ở bất kỳ phương thức nào khác.
- Một khi đã gán giá trị, nó không cho phép thay đổi giá trị.