THUỘC TÍNH

Property (đặc tính) là một loại thành viên đặc biệt trong class C#. Property trong C# có nhiệm vụ hỗ trợ xuất nhập dữ liệu cho biến thành viên hoặc trực tiếp lưu trữ dữ liệu. Trong C#, property được sử dụng đặc biệt phổ biến do nó có thể kiểm soát được việc đọc/ghi dữ liệu. Ngoài ra, property được sử dụng trong khởi tạo object của class. Một số thư viện class (như của windows forms) hoàn toàn sử dụng property.

Bài học sẽ giúp bạn hiểu chi tiết về loại thành viên này của class. Nắm vững cách sử dụng Property bạn mới có thể xây dựng được class "theo kiểu C#".

Property là gì?

Trong bài học về biến thành viên bạn đã biết rằng có thể sử dụng biến public để lưu trữ và truy xuất dữ liệu cho class. Tuy nhiên, class xây dựng với biến thành viên public có một số nhược điểm:

1. Người sử dụng class có thể trực tiếp truy xuất giá trị của các trường. Việc trực tiếp truy xuất này là khó chấp nhận nếu chúng ta cần giới hạn một phần việc truy xuất thông tin. Ví dụ, có trường thông tin chúng ta chỉ muốn cho đọc mà không cho ghi giá trị.
2. Giá trị nhập vào không được kiểm soát hoặc biến đổi phù hợp. Ví dụ, năm không thể nhận giá trị âm.

Trong lập trình hướng đối tượng nên hạn chế tối đa việc sử dụng biến thành viên public.

Thông thường, để giải quyết vấn đề này, người lập trình thường xây dựng một căp phương thức để gán/lấy giá trị cho mỗi biến thành viên. Cặp phương thức như vậy thường được gọi là setter/getter. Phương pháp này rất phổ biến khi lập trình PHP hay Java.

Tuy nhiên, cách viết như vậy làm code trở nên dài dòng.

C# cung cấp một công cụ đặc biệt để giải quyết vấn đề này, giúp viết code class đơn giản hơn, an toàn hơn, cũng như tiện lợi hơn về sau (khi khởi tạo object của class): property (đặc tính).

Property là một loại thành viên đặc biệt có vai trò lai giữa biến thành viên (lưu trữ và truy xuất dữ liệu) nhưng cho phép kiểm soát dữ liệu (gán vào hoặc xuất ra) giống như phương thức, cũng như cho phép kiểm soát riêng rẽ từng chiều truy xuất. Property được sử dụng đặc biệt phổ biến trong class C#.

Thực tế, property trong C# chỉ là một dạng viết tắt và viết gộp (syntactic sugar) của hai phương thức get và set.

Cùng thực hiện ví dụ sau để hiểu cách thức làm việc với property.

Tạo blank solution S07_Property và project P00_PropertyWithBackedField. Viết code cho Program.cs như sau:

```
namespace P00_PropertyWithBackedField
{
    using static System.Console;
    /// <summary>
```

```
    /// sách điện tử
    /// </summary>
    internal class Book
    {
        private int _id = 1;
        private string _authors = "Unknown author";
        private string _title = "A new book";
        private string _publisher = "Unknown publisher";
        private int _year = 2018;
        private string _description;
        public int Id
        {
            get { return _id; }
            protected set {
                _id = value;
            }
        }
        /// <summary>
        /// tên tác giả/ nhóm tác giả
        /// </summary>
        public string Authors
        {
            get { return _authors; }
            set {
                _authors = value;
            }
        }
        /// <summary>
        /// tiêu đề
        /// </summary>
        public string Title
        {
            get { return _title; }
            set {
                _title = value;
            }
        }
        /// <summary>
        /// nhà xuất bản
        /// </summary>
        public string Publisher
        {
            get { return _publisher; }
            set {
                _publisher = value;
            }
        }
        /// <summary>
        /// năm xuất bản
        /// </summary>
        public int Year
```

```
        {
            get { return _year; }
            set {
                _year = value;
            }
        }
        /// <summary>
        /// thông tin mô tả
        /// </summary>
        public string Description
        {
            get { return _description; }
            set {
                _description = value;
            }
        }
    }
    internal class Program
    {
        private static void Main(string[] args)
        {
            var book = new Book();
            // lệnh này lỗi, vì setter của Id là protected
            // chỉ có thể gán giá trị cho Id từ trong class
            // không thể gán giá trị từ ngoài class
            //book.Id = 2;
            book.Authors = "Christian Nagel";
            book.Title = "Professional C# 7 and .NET Core";
            book.Publisher = "Wrox";
            book.Year = 2018;
            book.Description = "The best book ever about the new C# 7 and the .NET Core";
            WriteLine($"{book.Authors}, {book.Title}, - {book.Publisher}, {book.Year}");
            ReadKey();
        }
    }
}
```

Trong class Book bạn đã khai báo (và khởi tạo) một loạt biến thành viên private (_id, _authors, _title, v.v.). Tiếp theo bạn lại viết một số cấu trúc lạ như

```
public string Authors
{
    get { return _authors; }
    set {
        _authors = value;
    }
}
```

Đây chính là cách khai báo property trong C#. Property này có tên là Authors.

Như đã nói, property trong C# thực chất chỉ là một dạng viết tắt của hai phương thức get và set. Phương thức get dùng để trả lại giá trị của biến _authors; Phương thức set dùng để gán giá trị cho biến _authors. Hai phương thức này hoạt động không có gì khác biệt phương thức thông thường, ngoại trừ chúng không có danh sách tham số.

Biến _authors có tên gọi là **backed field** cho property Authors. Đó là nơi lưu dữ liệu thực sự. Còn property Authors đóng vai trò kiểm soát xuất/nhập cho biến backed field này.

Từ khóa value có vai trò đặc biệt. Trong client code, phép gán

```
var book = new Book();
book.Authors = "Christian Nagel";
```

Giá trị "Christian Nagel" cung cấp vào cho property Authors thông qua từ khóa value. Nói cách khác, bạn có thể xem value là tham số đầu vào của phương thức get.

Bạn cũng nhìn thấy trong phương thức Main, việc sử dụng các property không khác biệt gì so với sử dụng biến thành viên public.

```
book.Authors = "Christian Nagel";
book.Title = "Professional C# 7 and .NET Core";
book.Publisher = "Wrox";
book.Year = 2018;
book.Description = "The best book ever about the new C# 7 and the .NET Core";
```

Riêng đối với Id có chút khác biệt. Do setter của nó đặt là protected, bạn không gán giá trị cho nó được. Đây là một property chỉ đọc (đối với client code).

Auto property

Nếu không cần kiểm soát giá trị xuất/nhập, bạn có thể thu gọn code bằng cách sử dụng auto-property.

Auto-property là loại property trong đó trường backed field được compiler sinh tự động. Bạn không biết tên của backed field khi viết code, do đó cũng không trực tiếp sử dụng được nó.

Cấu trúc khai báo auto-property (kết hợp gán giá trị đầu) trong C# như sau:

```
[public|protected|private] <tên-kiểu> <tên-thuộc-tính>
{
    [public|protected|private] get;
    [public|protected|private] set;
} [= <giá-trị>];
```

Trong đó, tên property được đặt theo quy tắc đặt định danh và quy ước giống như biến thành viên public (sử dụng PascalCase).

Hai phương thức *get* và *set* được gọi chung là *accessor*, đôi khi cũng được gọi là getter và setter. Getter hoặc setter có thể sử dụng từ khóa điều khiển truy cập của riêng mình, giúp property đó biến thành loại:

- chỉ đọc (read-only): public get, protected/private set;
- chỉ gán (assign-only): protected/private get, public set;
- truy cập tự do (full access): public get, public set (mặc định).

Modifier mặc định cho getter và setter là "public", do đó cấu trúc khai báo ngắn gọn nhất là:

```
public <tên-kiểu> <tên-thuộc-tính> { get; set; }
```

Hãy cùng thực hiện một ví dụ để hiểu rõ cách khai báo và sử dụng của auto property. Thêm project P01_AutoProperty kiểu Console App vào solution. Viết code cho Program.cs như sau:

```
namespace P01_AutoProperty
{
    using static System.Console;
    /// <summary>
    /// sách điện tử
    /// </summary>
    class Book
    {
        public int Id { get; protected set; } = 1;
        /// <summary>
        /// tên tác giả/ nhóm tác giả
        /// </summary>
        public string Authors { get; set; } = "Unknown author";
        /// <summary>
        /// tiêu đề
        /// </summary>
        public string Title { get; set; } = "A new book";
        /// <summary>
        /// nhà xuất bản
        /// </summary>
        public string Publisher { get; set; } = "Unknown publisher";
        /// <summary>
        /// năm xuất bản
        /// </summary>
        public int Year { get; set; } = 2018;
        /// <summary>
        /// thông tin mô tả
        /// </summary>
        public string Description { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            var book = new Book();
```

```
            // lệnh này lỗi, vì setter của Id là protected
            // chỉ có thể gán giá trị cho Id từ trong class
            // không thể gán giá trị từ ngoài class
            //book.Id = 2;
            book.Authors = "Christian Nagel";
            book.Title = "Professional C# 7 and .NET Core";
            book.Publisher = "Wrox";
            book.Year = 2018;
            book.Description = "The best book ever about the new C# 7 and the .NET Core";
            WriteLine($"{book.Authors}, {book.Title}, - {book.Publisher}, {book.Year}");
            ReadKey();
        }
    }
}
```

Trong ví dụ trên bạn đã khai báo một class hoàn toàn sử dụng auto property mà không có biến thành viên.

Bạn dễ dàng để ý thấy, việc khai báo auto property không khác biệt nhiều so với biến, ngoại trừ cặp {get; set;} đứng sau tên.

Id, Authors, Title, Publisher và Year khi khai báo được gán sẵn giá trị đầu. Description không được gán sẵn giá trị nên sẽ nhận giá trị mặc định (null) lúc khởi tạo object. Tất cả các property này đều có thể tự do truy xuất trong client code (phương thức Main).

Riêng Id đặc biệt hơn một chút là setter của nó để mức truy cập là protected. Điều này dẫn tới là trong client code không thể gán giá trị cho Id, nhưng vẫn có thể đọc giá trị của Id.

Bạn có thể thấy, mỗi property lúc khai báo sẽ chứa hai phương thức get và set. Tuy nhiên, khi truy xuất qua tên object sẽ chỉ nhìn thấy duy nhất tên property tương tự như một biến thành viên public bình thường.

Property sử dụng cấu trúc này thường được sử dụng để thay thế cho biến public. Trong thân phương thức thành viên có thể sử dụng auto property giống hệt như sử dụng biến thành viên.

Full property

Full property đủ có cách khai báo khác với auto property. Full property phải đi kèm với backed field, nơi thực sự lưu trữ thông tin. Property sẽ kiểm soát giá trị dữ liệu trước khi gán giá trị đó vào biến.

Cấu trúc để khai báo thuộc tính đầy đủ như sau:

```
[private|protected] <tên-kiểu> <tên-biến-hỗ-trợ>;
[public|protected|private] <tên-kiểu> <tên-thuộc-tính>
{
     [public|protected|private] get { [thân-phương-thức]; return <tên-biến-hỗ-trợ>;}
     [public|protected|private] set { [thân-phương-thức]; <tên-biến-hỗ-trợ> = value;}
}
```

Khi sử dụng thuộc tính đầy đủ chúng ta gặp từ khóa *value*. Từ khóa này được sử dụng như một biến chứa giá trị đang cần gán cho thuộc tính.

Để sử dụng cấu trúc đầy đủ, thông thường mỗi thuộc tính sẽ được tạo ra cùng một biến thành viên private. Biến private lưu trữ thông tin, property làm nhiệm vụ kiểm soát thông tin cho biến này.

Phương thức get sẽ trả giá trị của biến qua tên property; phương thức set cho phép property nhận giá trị và gán lại vào biến. Biến thành viên private này được gọi là *trường/biến hỗ trợ* (backed field).

Để dễ hiểu, hãy cùng thực hiện một ví dụ. Thêm project P02_FullProperty kiểu ConsoleApp vào solution và viết code như sau:

```
namespace P02_FullProperty
{
    using static System.Console;
    /// <summary>
    /// sách điện tử
    /// </summary>
    internal class Book
    {
        public int Id { get; protected set; } = 1;
        private string _authors = "Unknown author";
        private string _title = "A new book";
        private int _year = 2018;
        /// <summary>
        /// tên tác giả/ nhóm tác giả
        /// </summary>
        public string Authors
        {
            get { return _authors; }
            set { if (!string.IsNullOrEmpty(value)) { _authors = value; } }
        }
        /// <summary>
        /// tiêu đề
        /// </summary>
        public string Title
        {
            get { return _title; }
            set { if (!string.IsNullOrEmpty(value)) { _title = value; } }
        }
        /// <summary>
        /// nhà xuất bản
        /// </summary>
        public string Publisher { get; set; } = "Unknown publisher";
        /// <summary>
        /// năm xuất bản
        /// </summary>
        public int Year
        {
            get { return _year; }
```

```
            set { if (value > 0) _year = value; }
        }
        /// <summary>
        /// thông tin mô tả
        /// </summary>
        public string Description { get; set; }
    }
    internal class Program
    {
        private static void Main(string[] args)
        {
            var book = new Book();
            // lệnh này lỗi, vì setter của Id là protected
            // chỉ có thể gán giá trị cho Id từ trong class
            // không thể gán giá trị từ ngoài class
            //book.Id = 2;
            book.Authors = "Christian Nagel";
            book.Title = "Professional C# 7 and .NET Core";
            book.Publisher = "Wrox";
            book.Year = 2018;
            book.Description = "The best book ever about the new C# 7 and the .NET Core";
            WriteLine($"{book.Authors}, {book.Title}, - {book.Publisher}, {book.Year}");
            ReadKey();
        }
    }
}
```

Trong ví dụ trên chúng ta đã điều chỉnh lớp Book để sử dụng full property. Hãy để ý các trường Authors, Title và Year. Đây là 3 full property.

Ví dụ, property Authors giờ đây phải sử dụng kết hợp với biến backed field _authors. Đây là nơi thực sự lưu giữ thông tin. Property Authors giờ đóng vai trò xuất/nhập/kiểm soát dữ liệu cho biến _authors. Nhìn từ client code thì sẽ không thấy _authors mà chỉ thấy Authors duy nhất. Thông qua Authors có thể đọc thông tin. Tuy nhiên, nếu gán chuỗi trống hoặc chuỗi null cho Authors thì giá trị này không được gán cho _authors. Setter của Authors kiểm soát việc gán thông tin này.

Full property không thể gán giá trị đầu như auto-property mà chỉ có thể gán giá trị đầu cho biến backed field.

Tình huống tương tự cũng diễn ra với Title: không chấp nhận giá trị rỗng hoặc null. Đối với Year: không chấp nhận giá trị âm.

Như vậy, full property nên sử dụng cho các trường thông tin cần kiểm soát dữ liệu, và sử dụng auto-property cho các trường thông tin cho phép truy xuất tự do.

Trong thân phương thức thành viên có thể sử dụng full property hoặc biến backed field cho cùng mục đích. Hai cách sử dụng này không có gì khác biệt.

Thực tế, auto-property chỉ là một dạng viết tắt của full-property, trong đó biến backed field được sinh tự động.

Để cho ngắn gọn, nếu thân của getter hoặc setter chỉ có 1 lệnh duy nhất thì có thể sử dụng lối viết **expression body** như sau:

```
public string Authors
{
    get => _authors;// expression body, tương đương với get { return _authors;}
    set { if (!string.IsNullOrEmpty(value)) { _authors = value; } }
}
```

Lưu ý về sử dụng property

Property là một hoặc một tổ hợp phương thức giúp xuất nhập dữ liệu cho một biến thành viên cụ thể.

Trên thực tế, bạn có thể xem property là phương thức thông thường. Chỉ có điều C# thay đổi cú pháp đi một chút để tiện lợi hơn khi sử dụng, giúp bạn dùng property như là dùng một biến.

Nếu tất cả các biến thành viên của class được sử dụng hoàn toàn cục bộ (không cần tương tác với bên ngoài), bạn không cần đến property.

Nếu biến chỉ cần trả giá trị (để bên ngoài object có thể sử dụng), bạn cần tạo ra phương thức getter cho nó – chính là tạo ra readonly property.

Nếu biến chỉ cần nhận giá trị (tức là cho phép code bên ngoài class gán giá trị cho biến), bạn phải tạo ra setter cho nó – chính là tạo ra write-only property.

Nếu cần xuất nhập (cả hai chiều), bạn cần tạo ra đủ bộ getter và setter.

Nếu trong quá trình xuất nhập không cần kiểm soát biến (tức là cho xuất nhập tự do), bạn dùng auto property cho nhanh gọn (đỡ mất công code, lại có thể dùng code snippet prop).

Nếu cần kiểm soát giá trị biến khi nhập/xuất, khi đó bạn cần dùng full-property.

Auto property thực chất cũng là full-property. Tuy nhiên, khi này C# compiler tự động giúp bạn sinh ra backing field. Còn trong full-property, backing field là do bạn tự tạo ra và kiểm soát.

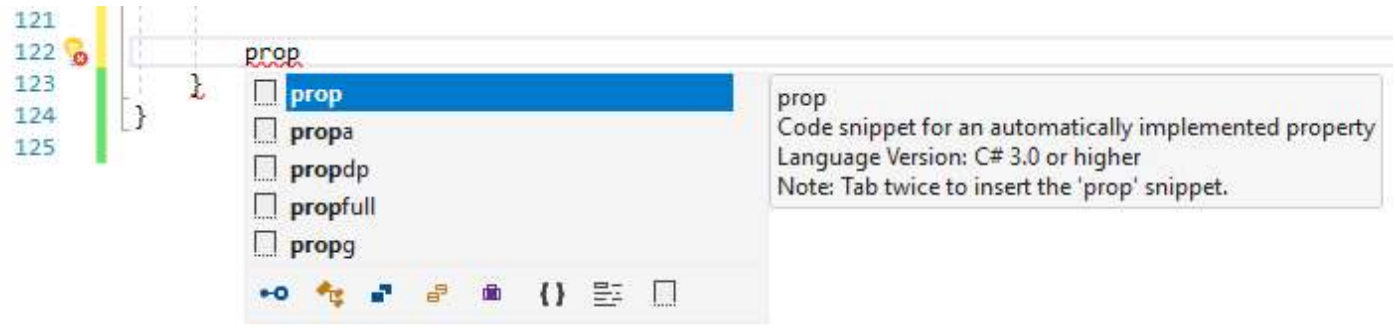
Hỗ trợ property trong Visual Studio

Code snippet

Để tăng tốc độ code, Visual Studio cung cấp một tính năng gọi là "code snippet". *Code snippet* là một khối code mẫu được xây dựng sẵn mà người lập trình có thể gọi ra thông qua một cụm ký tự viết tắt.

Để nhanh chóng tạo ra một auto property có thể sử dụng snippet **prop**: nhập cụm **prop** => trong danh sách lựa chọn của Visual Studio IntelliSense xuất hiện mục **prop** => chọn mục này và bấm phím Tab hai lần.

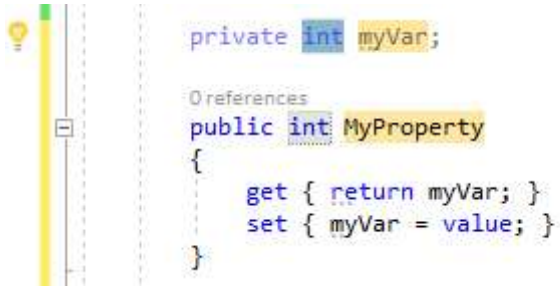
Trong snippet vừa tạo chỉnh sửa các thông tin cần thiết (phần code được bôi vàng). Di chuyển giữa các vùng bôi vàng bằng phím Tab. Kết thúc chỉnh sửa snippet bằng phím Enter.



Sử dụng snippet "prop" để tạo property

Sử dụng snippet prop

Tương tự, để tạo ra full property có thể dùng snippet **propfull**.



Kết quả thực hiện của snippet "propfull"

Kết quả thực hiện của snippet "propfull"

Quick Action

Tính năng Quick Action của Visual Studio giúp thực hiện tự động nhiều loại công việc khác nhau trong quá trình code.

Để sử dụng tính năng Quick Action có hai cách khác nhau:

1. Đặt con trỏ chuột vào đối tượng (trong trường hợp này là tên của property), ấn tổ hợp Ctrl + . (dấu chấm). Một menu sẽ xuất hiện ở rìa trái trình soạn thảo. Trong menu này chỉ chứa các lệnh khả dụng trong ngữ cảnh.
2. Click vào biểu tượng bóng đèn hoặc tô-vít xuất hiện khi đặt con trỏ vào đối tượng nào đó. Click biểu tượng này tương đương với tổ hợp *Ctrl* + . (dấu chấm).

Chuyển đổi giữa full và auto property

Một trong những công việc đầu tiên chúng ta có thể vận dụng Quick Action là chuyển đổi từ auto-property sang full property và ngược lại.

Đặt con trỏ văn bản vào property cần điều chỉnh và kích hoạt Quick Action (dùng tổ hợp Ctrl + . hoặc bấm nút tương ứng). Trong menu Quick Action sẽ xuất hiện lệnh chuyển đổi tương ứng.

Sử dụng quick action trong Visual Studio

Sử dụng quick action trong Visual Studio