CONSTRUCTOR

Constructor (hàm tạo/ hàm dựng) là một loại thành viên đặc biệt trong class C#. Nhiệm vụ của nó là khởi tạo object của class. Mỗi khi gọi lệnh khởi tạo, thực tế là bạn đang gọi tới constructor.

Bài học này sẽ hướng dẫn bạn cách viết hàm tạo khi xây dựng class và một vài cách khởi tạo đối tượng cho class trong C#.

Constructor và khởi tạo object

Các class bạn xây dựng trong các bài học trước tự bản thân nó không có nhiều giá trị với chương trình bởi vì class chỉ đơn thuần là mô tả kiểu dữ liệu. Để sử dụng class trong chương trình C#, bạn cần khởi tạo đối tượng của nó.

Khởi tạo đối tượng trong C# là quá trình yêu cầu tạo ra một object của class tương ứng trên vùng nhớ heap và lấy địa chỉ của object gán cho một biến.

Sau khi object được khởi tạo, bạn có thể truy xuất các thành viên của nó để phục vụ cho mục đích của chương trình.

Để *khởi tạo object* trong C# sử dụng từ khóa `new` và lời gọi tới một trong số các **hàm tạo** (constructor) của class tương tự như đối với struct.

Xây dựng constructor

Hàm tạo, về mặt hình thức, luôn có cùng tên với class và không có kiểu ra. Danh sách tham số và thân hàm tương tự như các phương thức thành viên.

Hãy cùng thực hiện ví dụ sau để hiểu cách xây dựng hàm tạo của class.

Tạo một blank solution S08_ObjectInstantiation rồi thêm project P01_DefaultConstructor. Viết code cho Program.cs như sau:

```
namespace P01_DefaultConstructor
{
    using static System.Console;
    internal class Book
    {
        private string _authors;
        private string _title;
        private int _year;
        private string _publisher;
        public Book() // đây là một hàm tạo của class Book
        {
            _authors = "Unknown author";
            _title = "A new book";
            _publisher = "Unknown publisher";
            _year = 2019;
```

```
        }
        public Book(string author, string title, int year, string publisher) // đây là hàm tạo có
        {
            _authors = author;
            _title = title;
            _year = year;
            _publisher = publisher;
        }
        public string Print()
        {
            return $"{_authors}, \"{_title}\", -{_publisher}, {_year}";
        }
    }
    internal class Program
    {
        private static void Main(string[] args)
        {
            ReadKey();
        }
    }
}
```

Trong ví dụ trên bạn đã xây dựng một class Book đơn giản. Trong class này chỉ có 4 biến thành viên private (_authors, _title, _year, _publisher), 1 phương thức thành viên Print().

Bạn có thể để ý hai thành viên đặc biệt:

```
public Book() // đây là một hàm tạo của class Book
{
    _authors = "Unknown author";
    _title = "A new book";
    _publisher = "Unknown publisher";
    _year = 2019;
}
public Book(string author, string title, int year, string publisher) // đây là hàm tạo có tham số
{
    _authors = author;
    _title = title;
    _year = year;
    _publisher = publisher;
}
```

Đây là hai constructor của class Book. Khi khởi tạo object với lệnh `new`, thực tế bạn sẽ gọi tới một trong hai constructor này. Đây cũng là khối code đầu tiên được thực thi khi khởi tạo object.

Mỗi constructor có thể chứa một access modifier (public, private, protected) như các thành viên khác. Điều đặc biệt là tên của constructor phải trùng với tên class. Phía sau tên của constructor là danh sách tham số, tương tự như đối với phương thức.

Với vai trò đó, trong constructor thường đặt các lệnh để khởi tạo giá trị cho các thành viên (như bạn đã làm).

Trong mỗi class C# không giới hạn số lượng constructor. Tuy nhiên, các constructor không được phép có danh sách tham số trùng nhau. Nếu có nhiều constructor trong một class, mỗi constructor được gọi là một **overload** (nạp chồng hàm tạo).

Danh sách tham số được gọi là trùng nhau nếu thứ tự các tham số theo kiểu (không theo tên gọi) và số lượng tham số giống nhau. Bạn sẽ quay lại vấn đề này khi học về nạp chồng phương thức và delegate.

Bạn có thể để ý hàm tạo và phương thức được Visual Studio hiển thị với cùng một biểu tượng.

Visual Studio sử dụng snippet *ctor* để sinh đoạn code khung cho hàm tạo.

Khởi tạo object với constructor

Bây giờ trong phương thức Main hãy viết một số lệnh như sau:

```
private static void Main(string[] args)
{
    var book1 = new Book();
    WriteLine(book1.Print());
    var book2 = new Book("Christian Nagel", "Professional C# 7 and the .NET core 2.0", 2018, "Wrox
    WriteLine(book2.Print());
    ReadKey();
}
```

Trong đó,

```
var book1 = new Book();
var book2 = new Book("Christian Nagel", "Professional C# 7 and the .NET core 2.0", 2018, "Wrox");
```

là hai lệnh **khởi tạo object** của lớp Book, sử dụng hai constructor đã xây dựng.

Như vậy có thể thấy, lệnh khởi tạo object cần có từ khóa new và gọi tới một trong số các constructor của class. Kết quả khởi tạo có thể gán cho một biến để sau tái sử dụng.

Khởi tạo object với property

C# cung cấp một cách khởi tạo object khác: sử dụng *bộ khởi tạo* (object initializer). Cú pháp khởi tạo này sử dụng property và được đưa vào từ C# 3 (.NET framework 3.5).

Hãy cùng thực hiện một ví dụ trước.

Thêm các property sau vào class Book:

```
public string Authors { get => _authors; set => _authors = value; }
public string Title { get => _title; set => _title = value; }
```

```
public int Year { get => _year; set => _year = value; }
```

Trong phương thức Main bổ sung các lệnh sau:

```
var book3 = new Book
 {
     Authors = "Christian Nagel",
     Title = "Professional C# 7 and the .NET core 2.0",
     Year = 2018
 };
 WriteLine(book3.Print());
```

Đây là cách khởi tạo với object initializer sử dụng property.

Trong cách khởi tạo này, chúng ta vẫn sử dụng lời gọi tới hàm tạo như bình thường, tuy nhiên, chúng ta có thể kết hợp luôn việc gán giá trị cho các property trong cùng lệnh khởi tạo theo quy tắc:

- Tất cả lệnh gán giá trị cho property phải đặt trong cặp dấu ngoặc nhọn;
- Mỗi lệnh gán viết tách nhau bởi một dấu phẩy;
- Phải kết thúc bằng dấu chấm phẩy, vì đây thực chất là một lệnh, không phải một khối lệnh (code block) như bình thường;
- Không bắt buộc phải gán giá trị cho tất cả các thuộc tính.

Cách thức khởi tạo này đặc biệt phù hợp với các class chứa dữ liệu sử dụng property cũng như khởi tạo danh sách. Hãy tưởng tượng nếu không có cách thức khởi tạo này, hàm tạo phải có rất nhiều tham số đầu vào để có thể gán giá trị cho tất cả các thành viên. Một hàm tạo với danh sách tham số quá dài nhìn rất cồng kềnh, khó nhớ thứ tự các tham số, cũng như dễ gây lỗi khi truyền tham số.

Nếu sử dụng hàm tạo mặc định hoặc hàm tạo không tham số có thể bỏ cả cặp dấu ngoặc tròn sau tên constructor.

Một số vấn đề khác với constructor

Default constructor

Hàm tạo là bắt buộc khi định nghĩa class. Tuy nhiên chương trình dịch của C# có khả năng **tự sinh hàm tạo** cho class nếu nó không nhìn thấy định nghĩa hàm tạo nào trong class. Loại hàm tạo này có tên gọi là **hàm tạo mặc định** (default constructor). Hàm tạo mặc định không có tham số đầu vào.

Nếu trong khai báo class chúng ta tự viết một hàm tạo không có tham số đầu vào, hàm tạo này không được gọi là hàm tạo mặc định nữa mà được gọi là **hàm tạo không tham số** (parameter-less/zero-parameter constructor), vì nó không phải do chương trình dịch của C# sinh ra.

Trong ví dụ trên, `public Book() {...}` là một hàm tạo không tham số nhưng nó không phải là hàm tạo mặc định.

Một khi đã định nghĩa hàm tạo riêng trong class, C# compiler sẽ không tự sinh ra hàm tạo mặc định nữa. Nghĩa là nếu bạn muốn gọi hàm tạo không tham số, bạn phải tự viết thêm hàm tạo đó. Nếu không, quá trình dịch sẽ báo lỗi.

Chuỗi constructor, constructor initializer

Hãy điều chỉnh lại class Book như sau:

```
internal class Book
{
    private string _authors = "Unknown author";;
    private string _title = "A new book";
    private int _year = 2019;
    private string _publisher = "Unknown publisher";
    public Book()
    {
        _authors = "Unknown author";
        _title = "A new book";
        _publisher = "Unknown publisher";
        _year = 2019;
    }
    public Book(string author)
    {
        _authors = author;
    }
    public Book(string author, string title) : this(author)
    {
        _title = title;
    }
    public Book(string author, string title, int year): this(author, title)
    {
        _year = year;
    }
    public Book(string author, string title, int year, string publisher) : this(author, title, ye
    {
        _publisher = publisher;
    }
    public string Print()
    {
        return $"{_authors}, \"{_title}\", -{_publisher}, {_year}";
    }
}
```

Những điều chỉnh trên sử dụng một khả năng đặc biệt của C#: constructor gọi lẫn nhau. Khi cho các constructor gọi lẫn nhau như trên bạn có thể tạo ra một chuỗi constructor với số lượng tham số tăng dần, đồng thời tận dụng được code của constructor xây dựng trước đó.

Cấu trúc : `this (...)` như trên có tên gọi là **constructor initializer**, là loại cấu trúc đặc biệt cho phép gọi đến constructor khác và truyền tham số phù hợp cho nó.

Trong ví dụ trên, `this(author)` là lời gọi đến constructor `Book(string author)` trước đó; `this(author, title)` là lời gọi đến `Book(string author, string title)`; v.v..

Constructor initializer luôn luôn thực thi trước constructor gọi nó. Khi kết hợp tốt các constructor initializer như trên, bạn có thể tạo ra một chuỗi constructor với số lượng tham số tăng dần mà không cần viết lặp lại code của các constructor trước đó.

Khi tạo chuỗi constructor như trên, bạn có thể khởi tạo object với bất kỳ constructor nào:

```
var b1 = new Book(); // dùng hàm tạo không tham số
var b2 = new Book("Donald Trump"); // gọi hàm tạo Book(string author)
var b3 = new Book("Donald Trump", "C# for dummy"); // gọi hàm tạo Book(string author, string titl
var b4 = new Book("Donald Trump", "C# for dummy", 2020); // gọi hàm tạo Book(string author, strin
```

Lưu ý: Nếu trong class sử dụng public property thay cho biến thành viên, bạn nên khởi tạo object với property theo cú pháp object initializer (đã xem xét ở trên), và do đó không cần tạo chuỗi constructor. Object initializer là cú pháp được khuyến khích (và yêu thích) trong C# đối với các domain class.

Vấn đề khởi tạo và sử dụng object

Quan hệ class và object

Có thể hình dung class giống như một bản thiết kế trên giấy của một ngôi nhà. Tự bản thân bản thiết kế này không phải ngôi nhà. Và có bản thiết kế không có nghĩa là chúng ta có ngôi nhà.

Chỉ khi bạn sử dụng bản thiết kế này để xây dựng được một/một số ngôi nhà cụ thể, bản thiết kế đó mới có giá trị.

Quá trình sử dụng bản thiết kế để xây dựng ngôi nhà có thể xem như tương đương với quá trình **khởi tạo đối tượng** (object initialization/instantiation) trong C#. Sau khi có ngôi nhà, chúng ta mới có thể ở. Quá trình sử dụng ngôi nhà này tương đương với việc *sử dụng object* để giải quyết các vấn đề của chương trình.

Trong lập trình hướng đối tượng có thể phân biệt ba giai đoạn:

1. *Xây dựng class*: định nghĩa kiểu dữ liệu, tương tự như tạo bản thiết kế ngôi nhà;
2. *Khởi tạo object*: khai báo và gán giá trị đầu cho biến, tương tự giai đoạn xây nhà theo thiết kế;
3. *Sử dụng object*: sử dụng biến trong các lệnh và biểu thức, tương tự khai thác ngôi nhà.

Khai báo và khởi tạo object

Việc *khai báo một object* thực hiện tương tự như khai báo biến thuộc các kiểu dữ liệu cơ sở mà bạn đã biết. Tuy nhiên, việc khai báo đơn thuần như vậy không đủ để sử dụng object, vì khi đó C# đơn giản gán cho object giá trị null – giá trị mặc định của object, mà không thực sự cấp phát bộ nhớ cho object.

Việc truy xuất một object có giá trị null luôn luôn gây lỗi `NullReferenceException ("Object reference not set to an instance of an object.")`. Ngoài ra, trình biên dịch của C# luôn bắt buộc *các biến cục bộ* **phải** được *khởi tạo* (instantiation, initialization) trước khi sử dụng.

Khi khởi tạo, một object sẽ được tạo ra trong vùng nhớ heap. Nếu kết quả khởi tạo gán cho một biến, địa chỉ của object sẽ được gán cho biến này. Bản thân địa chỉ của object chỉ là một con số. Con số này lại được lưu trong stack của phương thức.

Do khởi tạo object thực chất là lời gọi tới hàm tạo, C# bắt buộc mỗi class phải có hàm tạo.

Truy xuất các thành viên của object

Trong định nghĩa class, chúng ta đã biết ba loại thành viên là biến thành viên, đặc tính, và phương thức. Khi một object được khai báo và khởi tạo, chúng ta có thể sử dụng các thành viên này để thực sự chứa dữ liệu hoặc xử lý dữ liệu.

Việc truy xuất các thành viên chỉ có thể thực hiện thông qua tên object, không thể thực hiện qua tên class (trừ thành viên static sẽ học sau).

Để phân biệt, người ta sử dụng thuật ngữ *instance members* (bao gồm *instance method*, *instance variable*, *instance property*) để mô tả các thành viên của class mà chỉ thực sự tồn tại sau khi khởi tạo object. Sự tồn tại của các thành viên này phụ thuộc vào object (vốn cũng được gọi là một *instance* của class).

Để truy xuất thành viên của object chúng ta sử dụng phép toán "." với tên object. Truy xuất phương thức thành viên đơn giản là một lời gọi phương thức từ một object nào đó. Việc truy xuất phương thức thành viên cũng sử dụng cấu trúc tương tự:

Có sự khác biệt khi truy xuất thành viên của một object từ client code với việc truy xuất trong nội bộ một class.

Client code là đoạn code nơi thực hiện khởi tạo và sử dụng object.

Nếu trong định nghĩa class, một thành viên được xác định là protected hoặc private sẽ không thể truy xuất được từ client code mà chỉ có thể được truy xuất trong nội bộ class.

Chỉ những thành viên được xác định là public mới có thể được truy xuất từ client code.

Khi truy xuất thành viên từ trong nội bộ của class thì không cần sử dụng phép toán truy xuất thành viên. Tuy nhiên có một số tình huống đặc thù sẽ phải sử dụng đến từ khóa `this` và phép toán truy xuất thành viên.

Từ khóa this

Giả sử trong constructor bạn đặt tên cho tham số như sau:

```
    public Book(string _authors)
    {
```

```
        _authors = _authors; // làm sao phân biệt _authors nào là member class, _authors nào là tl
    }
```

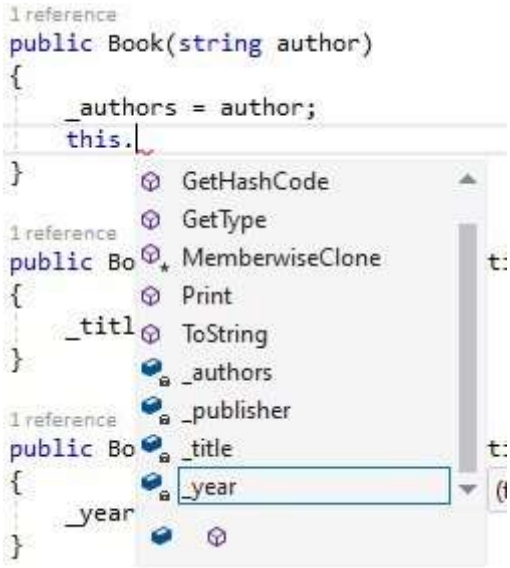
Tham số _authors giờ trùng tên với biến thành viên _authors. Cách đặt tên này không vi phạm gì trong C#. Vấn đề bây giờ là, làm sao để phân biệt _authors nào là member class, _authors nào là tham số???

Trong những tình huống thế này, bạn có thể sử dụng từ khóa `this` để chỉ rõ đâu là thành viên của class:

```
this._authors = _authors;
```

Từ khóa `this` cho phép chỉ định chính bản thân object nơi đang thực thi code. Trong ví dụ trên, `this._authors` báo hiệu rằng cần dùng chính biến thành viên `_authors` của object đó. Điều này cũng có nghĩa là bạn có thể dùng this trước mọi thành viên của class. Tuy nhiên, bạn nên hạn chế sử dụng `this` nếu có thể vì nó làm code nhìn phức tạp hơn.

Từ khóa this cũng có một tác dụng phụ khác khá tốt. Nếu bạn không nhớ hết các thành viên của class, bạn có thể gõ this, dấu chấm, và chờ intellisense giúp liệt kê hết các thành viên (non-static) của class đó.



từ khóa this giúp liệt kê thành viên của class

Từ khóa this giúp liệt kê thành viên của class

Từ khóa this chỉ có tác dụng với các thành viên bình thường (không có từ khóa static).