**Chuẩn viết code**

# Mục đích

Đưa ra các quy ước khi coding với ngôn ngữ lập trình C#, với các quy tắc này giúp tiết kiệm thời gian rất lớn trong tiến trình phát triển phần mềm và cả trong quá trình bảo trì sản phẩm. Giúp cho quá trình làm việc nhóm có thể sử dụng, đọc hiểu code của các thành viên khác dễ dàng hơn.

Tài liệu mô tả các chức năng, kỹ thuật cơ bản để sử dụng các thành phần cơ bản của C#.

# Mô tả thành phần

1. Namespace

Một **namespace** trong C# được thiết kế để cung cấp một cách để giữ một tập hợp các tên được phân biệt riêng rẽ nhau. Các tên lớp được khai báo trong một namespace không xung đột với cùng tên đó của lớp được khai báo trong namespace khác.

1. Class

Class là một cấu trúc dữ liệu chứa các thành phần dữ liệu, các thành phần chức năng (hàm, thuộc tính, sự kiện, hoạt động, các hàm hủy, khởi tạo) và các class lồng nhau.

* 1. Abstract classes
* **Abstract class** hay còn gọi là lớp trừu tượng. Nếu dùng từ trừu tượng (dịch nghĩa của từ abstract) thì sẽ gây cho bạn khó hiểu, có gì đó mơ hồ khi mới tiếp cận, do đó từ bên dưới mình sẽ gọi lớp trừu tượng là **abstract class**phương thức trừu tượng là **abstract method**. Bạn cứ hiểu nó cũng chỉ là một tên gọi class hay phương thức như bình thường thôi, chủ yếu là bạn hiểu được cách dùng của nó.
* **Abstract class** được xem như là một lớp cha(base class) cho tất cả các lớp con(derived class) có cùng bản chất (kiểu, loại, nhiệm vụ của lớp).
* Dưới đây là ví dụ về abstract class mà mình sưu tầm được.
* using System;
* namespace ConsoleApp1
* {
* class Program
* {
* public abstract class Animal {
* //abstract method
* public abstract void AnimailSound();
* public virtual void Eat()
* {
* Console.WriteLine("Dong Vat An Co");
* }
* public void Run()
* {
* Console.WriteLine("Di Chuyen Bang 4 Chan");
* }
* // Đúng
* //public static abstract void AnimailSound(); // Sai
* }
* public class Cat : Animal
* {
* // abstract method:bắt buộc lớp con phải override lại từ lớp cha.
* // Khi override thì phạm vi truy cập phải giống của lớp cha: public
* public override void AnimailSound()
* {
* Console.WriteLine("Meo Said :'Meow Mow'");
* // virtual method: việc phải override thì không bắt buộc.
* // Lớp con có thể dùng luôn định nghĩa từ lớp cha nếu phù hợp,
* // hoặc override (định nghĩa lại) nếu không phù hợp
* // Khi override thì phạm vi truy cập phải giống của lớp cha: public
* }
* public override void Eat()
* {
* Console.WriteLine("Meo An Ca.");
* }
* // normal method Run() thì dùng luông của lớp cha không phải làm gì.
* }
* public class Dog : Animal
* {
* public override void AnimailSound()
* {
* Console.WriteLine("Gau Go Gau.");
* }
* }
* static void Main(string[] args)
* {
* // abstract class không thể tạo ra được đối tượng
* // mà phải thông qua lớp con kế thừa từ nó
* // Sai
* // Animal animal= new Animal();
* // Error Cannot create an instance of the abstract class or interface 'Animal'
* // Đúng
* Animal cat = new Cat();
* // hoặc:
* Cat cat1 = new Cat();
* Dog dog = new Dog();
* // Gọi các phương thức
* cat.Eat();
* dog.Eat();
* cat.AnimailSound();
* dog.AnimailSound();
* cat.Run();
* dog.Run();
* Console.ReadKey();
* }
* }
* }
  1. Sealed classes

Sealed class là những lớp không thể kế thừa, khi bạn tạo một class kèm thêm từ khóa sealed nó sẽ không thể là lớp cha của bất kì lớp con nào (vô sinh).

Mục đích của sealed class là ngăn chặn sự kế thừa, không muốn cho ai can thiệp vào lớp này. Như Microsoft có thể cung các các thư viện, các lớp có thể tác động đến hệ thống, họ thường để lớp này là sealed class để ngăn chặn bên thứ ba chỉnh sửa lóp này.

* 1. Static classes

Static class: class sẽ không thể tạo ra một đối tượng mà được gọi trực tiếp bằng tên class  
class nếu không được khai báo Access modifier thì sẽ mặc định là internal nó khác với publish là nó không thể truy xuất từ các assembly khác nhau.

* 1. Generic classes
* Trong link dưới đã nói rất rõ về generic trong C# rồi. <http://www.v1study.com/csharp-tao-va-su-dung-generic-a631.html>.

https://toidicodedao.com/2015/03/05/series-c-hay-ho-generic-la-cai-thu-chi-chi/.

1. Interface

* Về interface đại loại nó là 1 cái khung sườn được dựng sẵn. Gần tương tự như abstract class. Nói bóng gió nó như 1 con manocanh khi 1 class implement nó đó là giai đoạn mặc quần áo cho nó, phải đảm bảo tất cả các hàm phải được implement.

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Cho phép khai báo field | Không cho phép |
| Các phương thức có thể có thân hàm hoặc không có thân hàm. | Chỉ khai báo không có thân hàm |
| Class dẫn xuất chỉ kế thừa được từ 1 abstract class và nhiều interface. | Class triển khai có thể triển khai nhiều interface. |
| Có chứa constructor | Không có |
| Các phương thức có từ khóa access modifier | Không có |

1. Struct

Trong C#, một cấu trúc (structure) là một kiểu dữ liệu. Nó giúp bạn tạo một biến đơn mà giữ dữ liệu liên quan của các kiểu dữ liệu đa dạng. Từ khóa **struct** trong C# được sử dụng để tạo một cấu trúc (structure).

**Phân biệt Class và Structure trong C#**

Lớp và Cấu trúc trong C# có một số điểm khác nhau cơ bản sau:

Các Lớp là các kiểu tham chiếu, còn cấu trúc là các kiểu giá trị.

Cấu trúc không hỗ trợ tính kế thừa.

Cấu trúc không có constructor mặc định.

1. Attribute

Một **attribute** trong C#, là một thẻ tường thuật, được sử dụng để truyền thông tin tới runtime về các hành vi của các phần tử đa dạng như các lớp, phương thức, cấu trúc, enum, assembly, … trong chương trình của bạn. Bạn có thể thêm thông tin tường thuật tới một chương trình bởi việc sử dụng một **Attribute**. Một thẻ tường thuật được miêu tả bởi các dấu ngoặc móc vuông ([]) được đặt bên trên phần tử mà nó được sử dụng cho

Ở đây bạn có thể custom những hàm authorize, security.

1. Các kỹ thuật IOC, DI

Sự khác biệt giữa 3 khái niệm trên:

* **Dependency Inversion**: Đây là một nguyên lý để thiết kế và viết code.
* **Inversion of Control**: Đây là một [design pattern](https://toidicodedao.com/2016/03/01/nhap-mon-design-pattern-phong-cach-kiem-hiep/) được tạo ra để code có thể tuân thủ nguyên lý Dependency Inversion. Có nhiều cách hiện thực pattern này: ServiceLocator, Event, [Delegate](https://toidicodedao.com/2015/02/10/series-c-hay-ho-callback-trong-c-delegate-action-predicate-func/), … Dependency Injection là một trong các cách đó.
* **Dependency Injection**: Đây là một cách để hiện thực Inversion of Control Pattern (Có thể coi nó là một design pattern riêng cũng được). Các **module phụ thuộc (dependency) sẽ được inject** vào module cấp cao.

Có thể hiểu Dependency Injection một cách đơn giản như sau:

1. **Các module không giao tiếp trực tiếp với nhau, mà thông qua interface***. Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp thông qua interface.*

*Ví dụ: Để giao tiếp với database, ta có interface*IDatabase*, các module cấp thấp là*XMLDatabase*,*SQLDatabase*. Module cấp cao là*CustomerBusiness*sẽ chỉ sử dụng interface*IDatabase*.*

1. **Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện***. Ví dụ: Trong module*CustomerBusiness*, ta sẽ không khởi tạo IDatabase db = new XMLDatabase(), việc này sẽ do DI Container thực hiện. Module CustomerBusiness sẽ không biết gì về module XMLDatabase hay SQLDatabase.*
2. **Việc Module nào gắn với interface nào sẽ được config trong code hoặc trong file XML***.*
3. **DI được dùng để làm giảm sự phụ thuộc giữa các module***, dễ dàng hơn trong việc thay đổi module, bảo trì code và*[*testing*](https://toidicodedao.com/2015/08/25/tutorial-viet-unit-test-trong-c-voi-nunit/)*.*

**Các dạng DI**

Có 3 dạng Dependency Injection:

1. **Constructor Injection**: Các dependency sẽ được container **truyền vào (inject vào)** 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.
2. **Setter Injection**: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
3. **Interface Injection**: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó. Đây là cách rườm rà và ít được sử dụng nhất.

- Mình có tìm hiểu được 1 seri hay viết về vấn đề này

<https://toidicodedao.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/>

Theo mình đây là một kỹ thuật quan trọng nếu bạn muốn làm một lập trình viên với C# hay bất kỳ ngôn ngữ lập trình hướng đối tượng khác… Theo kinh nghiệm làm nhiều dự án ngoài của mình thì gần như dự án nào mình cũng phải sử dụng đến kỹ thuật này.

# Các nội dung quy ước

1. Các kiểu quy ức viết hoa.

* Tên biến thường,biến thành viên có ký tự đầu tiên viết thường, từng từ tiếp theo có ký tự đầu viết hoa. VD: typeName, tempData
* Tất cả các phần: tên lớp, tên thuộc tính, tên phương thức, tên không gian, sự kiện, biến tĩnh, enum… đều viết hoa từng ký tự đầu
* Tên phương thức, sự kiện hàm bắt đầu bằng một động từ
* Tên Interface bắt đầu bằng chữ I
* Tên không gian, lớp, enum, thuộc tính là một cụm danh từ
* Không dùng ký tự \_ khi đặt tên biến
* Không dùng từ “enum” trong đặt tên của enum

1. Viết comment

* Không cần comment những đoạn mã đã rõ nghĩa
* VD: *// This is a for loop that prints the 1 million times*
* for (int i = 0; i < 1000000; i++) Console.WriteLine(i);
* Bản thân việc đặt tên hàm đã phải đủ nghĩa cho chức năng hoạt động chính của hàm đó, chỉ cần comment để lý giải các trường hợp hoạt động của hàm mà thôi.
* Viết comment trong trường hợp lý giải từ đầu phương thức hoạt động, các trường hợp đặc biệt cần xử lý của hàm, những đoạn chưa chắc chắn cần note lại với ký hiệu quy ước đặc biệt để tìm và sửa sau.

1. Tổ Chức Hàm

* Hàm cần thực hiện duy nhất một tính năng
* Một hàm chỉ nên có từ 20-25 dòng code
* Không nên truyền quá nhiều biến vào một hàm, mà hãy đóng gói các biến cần truyền vào một biến cấu trúc hoặc biến đối tượng
* void HàmA() 🡪 void HàmB() 🡪 void HàmA()  
  { { {

//Line 1 HàmB()

//Line 2 //Line 1,2,3 HàmC()

//Line 3 } }

//Line 4 🡪 void HàmC()

//Line 5 {

//Line 6 //Line 4,5,6,7,8

//Line 7 }

//Line 8

}

1. Cách tổ chức file trong 1 project.
2. File source code C#.

Giữ file nguồn/ các class không quá dài(không được vượt quá 2000 LOC (Lines of Code), phân chia code thành những đơn vị code nhỏ tạo nên các cấu trúc code sáng sủa hơn. Đặt mỗi class vào mỗi file source riêng biệt và tên file cũng là tên class(với phần đuôi mở rộng .cs).

1. Thứ tự trong file.

* Các lệnh **using** ( Khai báo các namespace cần sử dụng trong file source đó).
* Lệnh khai báo **namespace**.
* Các khai báo **class** và **interface**.

1. Bố trí thư mục sourcecode C#

Tạo 1 thư mục cho mỗi namespace trong project. (Với namespace MyProject.TestSuite.TestTier ta sẽ có 1 đường dẫn thư mục tương ứng MyProject/

TestSuite/TestTier). Và ta dễ dàng ánh xạ các namespace sang cấu trúc thư mục trong Project Explorer.

1. Tổ chức code

* Tránh những logic phức tạp mà có thể đóng gói những logic đó vào các hàm trả về bool
* VD: if(product.Price>500 && !product.IsDeleted && !product.IsFeatured && product.IsExported)
* Tránh và cần xóa những biến được khai báo nhưng không sử dụng đến
* Tên biến cần đặt tên là danh từ có nghĩa phục vụ cho việc xử dụng
* Nên đặt tiền tố trước tên biến theo quy ước: i cho integer, d cho double, dt cho datetime...
* Nên gán giá trị thực trả về của hàm vào một biến trước khi sử dụng vào việc tiếp theo.
* Nếu trong vòng lặp, có một vài trường hợp cần kiểm tra đặc biệt thì hãy để các trường hợp đó vào hàm con

Tránh các vòng lặp lồng trong cùng một hàm, có thể chia nhỏ những vòng lặp con vào hàm nhỏ hơn

* Suy nghĩ kỹ để sử dụng kiểu dữ liệu phù hợp để lưu trữ hay xử lý trong hàm: biến, đối tượng, mảng, danh sách, từ điển, ngăn xếp, hàng đợi, cây...
* Tránh khai báo, khởi tạo lại đối tượng trong vòng lặp nếu không cần thiết.
* Tránh gọi lại hàm có xử lý lớn trong vòng lặp nếu không cần thiết.
* Tránh thay đổi kích thước của các cấu trúc danh sách ngay trong vòng lặp duyệt nó.