C# Fast Food

Thứ ba, 19 Tháng 8 2008 05:39 Trần Phạm thanh Tùng C - C++ - Visual C#

Bài viết này dành cho những bạn đã thông thạo về C++ và muốn tiếp cận C# một cách nhanh chóng. Tuy nhiên, những bạn mới bắt đầu học C# cũng có thể xem nó như là một bản tóm tắt về C# nhằm định hướng tốt hơn trong việc học ngôn ngữ này.

Giới thiệu

C# là một ngôn ngữ mang những đặc điểm của C++, có phong cách lập trình như Java và có mô hình ứng dụng như Basic. Nếu đã biết về C++ thì các bạn sẽ mất không dưới một giờ để tìm hiểu cú pháp của C#. Còn nếu các bạn đã quen thuộc với Java thì sẽ là một lợi thế khác khi học C#. Cấu trúc chương trình Java, khái niệm về gói (package), garbage collection … chắc chắn sẽ giúp bạn học C# nhanh hơn. Trong bài viết này, khi chúng ta nói về các cấu trúc của C#, tôi sẽ xem như các bạn đã biết C++.

Bài viết này sẽ nói về các cấu trúc và những đặc điểm của ngôn ngữ C#, minh họa bằng những ví dụ ngắn gọn và dễ hiểu để khi các bạn nhìn vào ví dụ, các bạn có thể hiểu rõ những khái niệm. Trong bài viết này , chúng ta sẽ bàn về những chủ đề sau:

·Cấu trúc chương trình
·Namespaces
·Kiểu dữ liệu
·Biến
·Toán tử và biểu thức
·Kiểu liệt kê
·Câu lệnh
·Class và Struct
·Modifier
·Các thuộc tính (Property)
·Interface
·Các thông số hàm
·Mảng (array)
·Indexer
·Boxing và Unboxing
·Delegate
·Thừa kế và tính đa hình

Chúng ta sẽ không bàn về các vấn đề như: C# thông dụng hơn hay C++ thông dụng hơn, các khái niệm **garbage collection**, **threading, xử lý file** … chuyển kiểu dữ liệu, thư viện .Net.

Cấu trúc chương trình

Cũng như C++, C# thuộc dạng **case – sensitive** (phân biệt chữ hoa và chữ thường). Dấu chấm phẩy (;) là ký hiệu ngăn cách các phát biểu. Không như C++, trong C# không có sự phân chia giữa phần **khai báo (header) và phần hiện thực (cpp)**. Mọi đoạn mã (khai báo class và hiện thực) đều được đặt trong một file có phần mở rộng .cs. Bây giờ chúng ta làm quen với chương trình đầu tiên của C#, chương trình Hello world:

```
 using System;
namespace MyNameSpace
{
class HelloWorld
{
static void Main(string[] args)
{
    Console.WriteLine (“Hello World”);
    }
  }
}
```

Trong C#, tất cả các câu lệnh, phương thức… được “gói” trong một class, và tất cả các class được “gói” trong một namespace (giống như file trong folder). Cũng như C++, có một chương trình chính chứa những điểm nhập cho chương

trình của bạn. Chương trình chính của C++ được bắt đầu bằng “main”, trong khi đó chương trình chính của C# bắt đầu với “Main”.
Không cần đặt dấu chấm phẩy sau một khối class hay sau định nghĩa struct. Đó là quy định của C++, C# thì không cần.

Namespaces
Mỗi class được gói lại trong một namespace. Thật ra, namespaces là một khái niệm trong C++, nhưng trong C# chúng ta dùng namespaces thường xuyên hơn. Các bạn có thể truy xuất một class trong một namespaces bằng cách dùng dấu chấm (.). MyNameSpace là namespace của chương trình Hello world ở trên.
Bây giờ chúng ta hãy viết lại chương trình **HelloWorld** bằng cách truy xuất lớp HelloWorld từ một lớp khác trong một namespaces khác:

```
 using System;
namespace AnotherNameSpace
{
class AnotherClass
{
    public void Func()
{
    Console.WriteLine (“Hello World”);
    }
 }
}
```

Bây giờ từ lớp HelloWorld, các bạn có thể truy xuất nó:

```
using System;
using AnotherNameSpace;
namespace MyNameSpace
{
class HelloWorld
{
    static void Main(string[] args)
    {
        AnotherClass obj = new AnotherClass();
        obj.Func();
    }
  }
}
```

Trong thư viện .Net, System là namespace cấp cao nhất trong các namespace. Trong C#, bằng cách mặc định tồn tại một namespace toàn cục, một class định nghĩa bên ngoài một namespace được lưu trực tiếp trong namespace toàn cục này và do đó bạn có thể truy xuất lớp này mà không cần bất kỳ một qualifier (bổ từ) nào. Bạn cũng có thể định nghĩa những namespace lồng nhau.

--Using
Chỉ thị #include trong C++ được thay thế bởi từ khóa using, theo sau đó là tên của một namespace, chẳng hạn như “using System”. System là một namespace nền trong đó chứa tất cả những namespace khác và tất cả các lớp được định nghĩa trong những namespace đó. Lớp nền cho mọi đối tượng là Object trong namespace System.

----Biến

Các biến trong C# hầu hết đều giống trong C++, ngoại trừ những khác biệt sau:

· Biến trong C# luôn cần phải khởi tạo trước khi bạn truy xuất nó, nếu không bạn sẽ bị báo lỗi trong khi biên dịch. Do đó, phải nhớ rằng không thể truy xuất một biến chưa được khởi tạo.

· Bạn không thể truy xuất một con trỏ không trỏ vào đâu cả trong C#.

· Một biểu thức không thể gọi một phần tử của mảng (array) mà chỉ số của nó vượt ra khỏi kích thước của mảng.

· Không có khái niệm biến toàn cục hay hàm toàn cục trong C# và những khái niệm về toàn cục được biểu diễn thông qua những hàm và biến static

---Kiểu dữ liệu

Tất cả những kiểu dữ liệu của C# được định nghĩa trong lớp Object. Có hai loại kiểu dữ liệu:

*Kiểu dữ liệu cơ bản (dựng sẵn)

*Kiểu dữ liệu do người dùng định nghĩa

Dưới đây là bảng liệt kê các kiểu dữ liệu định sẵn trong trong C#:

```
 Kiểu      Kích thước Mô tả
Byte          1          unsigned byte
Sbyte        1       signed byte
Short        2       signed short
Ushort      2    unsigned short
Int           4    signed integer
Uint         4    unsigned integer
Long        8    signed long
Ulong     8    unsigned long
Float      4     floating point
Double    8    double precision
Decimal   8    fixed precision
String unicode  string
Char unicode   char
Bool true, false boolean
```

Ghi chú: Kích thước của các kiểu dữ liệu trong C++ và C# không giống nhau, ví dụ như kiểu long trong C++ là 4 byte , trong C# là 8 byte. Kiểu bool và string cũng khác nhau, kiểu bool trong C# chỉ chấp nhận giá trị true hay false chứ không chấp nhận giá trị integer. Kiểu dữ liệu do người dùng định nghĩa bao gồm:

1.Class

2.Struct

3.Interface

Nếu phân chia kiểu dữ liệu theo sự cấp phát bộ nhớ thì ta có thể chia làm hai loại:

1.Kiểu giá trị

2.Kiểu tham khảo

Kiểu giá trị

Kiểu giá trị là những dữ liệu được cấp phát bộ nhớ trong stack. Các loại dữ liệu này bao gồm:

·Tất cả những kiểu dữ liệu dựng sẵn ngoại trừ kiểu string.

·Struct

·Kiểu liệt kê

Kiểu tham khảo

Kiểu tham khảo được cấp phát bộ nhớ trên heap và sẽ trở thành rác khi chúng không được sử dụng nữa. Để khai báo kiểu dữ liệu này, ta dùng từ khóa new, và không như C++, không có từ khóa delete . Trong C#, chúng tự động được gom lại bởi " bộ phận thu thập rác" (garbage collector).

--Kiểu tham khảo bao gồm:

·Class

·Interface

·Kiểu tập hợp như mảng

·String

--Kiểu liệt kê (Enumeration)

Kiểu liệt kê trong C# hoàn toàn giống như C++, chúng được định nghĩa thông qua từ khóa enum

Ví dụ như:

```
enum Weekdays
```

```
{
Saturday, Sunday, Monday
Tuesday, Wednesday, Thursday, Friday
}
```

Class và Struct

Class và Struct cũng tương tự như C++, ngoại trừ sự khác nhau về sự cấp phát bộ nhớ. Những đối tượng của class được cấp phát bộ nhớ trong heap và được tạo ra bằng cách dùng new, còn struct được cấp phát bộ nhớ trong stack. Struct trong C# là kiểu dữ liệu rất nhẹ và nhanh. Do đó đối với những kiểu dữ liệu nặng bạn nên khai báo class. Bây giờ chúng ta quan sát ví dụ sau:

```
 struct Date
{
    int day;
    int month;
    int year;
}


 class Date
{
    int day;
    int month;
    int year;
    string weekday;
    string monthName;
public int GetDay()
{
   return day;
}
public int GetMonth()
{
   return month;
}
public int GetYear()
{
  return year;
}
public void SetDay(int Day)
{
   day = Day ;
}
public void SetMonth(int Month)
{
   month = Month;
}
public void SetYear(int Year)
{
   year = Year;
}
public bool IsLeapYear()
{
  return (year/4 == 0);
```

```
}
public void SetDate (int day, int month, int year)
{
}
...
}
```

Các thuộc tính

Nếu bạn đã quen thuộc với hướng đối tượng trong C++, bạn ắt hẳn phải có chút khái niệm gì đó về thuộc tính (properties). Các thuộc tính trong ví dụ class Date ở trên là day, month và year. Nếu trong C++, bạn phải dùng phương thức Get và Set thì C# cung cấp những cách thuận tiện hơn, đơn giản hơn và trực tiếp hơn để truy xuất những thuộc tính. Đối với ví dụ trên ta có thể viết lại như sau:

```
 using System;
class Date
{
public int Day{
get {
return day;
}
set {
day = value;
}
}
int day;
public int Month{
get {
return month;
}
set {
month = value;
}
}
int month;


 public int Year{
get {
return year;
}
set {
year = value;
  }
}
int year;


 public bool IsLeapYear(int year)
{
return year%4== 0 ? true:false;
}
```

```
 public void SetDate (int day, int month, int year)
{
this.day = day;
this.month = month;
this.year = year;
}
}
 Sau đây là cách bạn lấy và thiết lập thuộc tính:
 class User
{
public static void Main()
   {
Date date = new Date();
date.Day = 27;
date.Month = 6;
date.Year = 2003;
Console.WriteLine("Date: {0}/{1}/{2}",date.Day, date.Month, date.Year);
  }
}
```

Bổ từ (Modifier)
Trong C++, chúng ta đã gặp các bổ từ thông dụng như **public, private và protected**. Sau đây chúng ta sẽ nói về một số bổ từ mới trong C#.
Readonly
readonly chỉ được dùng cho những dữ liệu thành viên (member) của class. Dữ liệu readonly chỉ có thể đọc được khi chúng đã được khởi tạo trực tiếp hay gán giá trị cho chúng trong **constructor**. Sự khác nhau giữa dữ liệu **readonly và const (hằng)** là khi khai báo hằng, ta phải khởi tạo giá trị cho nó một cách trực tiếp. Hãy nhìn vào ví
dụ sau:

```
 class MyClass
{
//trực tiếp
const int constInt = 100;
//không trực tiếp
readonly int myInt = 5;
readonly int myInt2;
public MyClass()
{
// không trực tiếp
myInt2 = 8;
}
public Func()
{
myInt = 7; //không hợp lệ
Console.WriteLine(myInt2.ToString());
}
}
```

Sealed
Sử dụng từ khóa sealed khi khai báo một class sẽ không cho phép bạn lấy bất kỳ class nào từ nó. Do đó bạn nên sử dung từ khóa sealed cho những lớp mà bạn không muốn những lớp con thừa kế chúng.

```
 sealed class CanNotbeTheParent
```

```
{
int a = 5;
}
```

Unsafe

Bạn có thể định nghĩa một ngữ cảnh không an toàn trong C# bằng cách dùng từ khóa unsafe.
Trong ngữ cảnh không an toàn, bạn có thể viết một đoạn mã không an toàn, ví dụ như con trỏ C++ chẳng hạn. Chúng ta hãy xem xét ví dụ sau:

```
 public unsafe MyFunction( int * pInt, double* pDouble)
{
int* pAnotherInt = new int;
*pAnotherInt = 10;
pInt = pAnotherInt;
...
*pDouble = 8.9;
}
```

Interface

Nếu bạn đã có khái niệm về COM, bạn sẽ dễ dàng hiểu được nội dung của phần này. Một interface là một lớp trừu tượng cơ bản, trong đó chỉ chứa những ký hiệu của hàm, sự hiện thực những hàm này được cung cấp bởi những lớp con. Trong C#, bạn có thể định nghĩa những class như những interface thông qua từ khóa interface. .NET có nền tảng từ nhiều interface. Trong C# bạn không thể dùng nhiều lớp thừa kế, điều mà trong C++ cho phép, nhưng thật ra bản chất của sự đa thừa kế được thực hiện thông qua interface. Những lớp con của bạn cũng có thể hiện thực đa interface.

```
 using System;
interface myDrawing
{
int originx
{
get;
set;
}
int originy
{
get;
set;
}
void Draw(object shape);
}
class Shape: myDrawing
{
int OriX;
int OriY;
public int originx
{
get{
return OriX;
}
set{
```

```
OriX = value;
}
}
public int originy
{
get{
return OriY;
}
set{
OriY = value;
}
}
public void Draw(object shape)
{
...
}
public void MoveShape(int newX, int newY)
{
.....
}
}
```

---Mảng

Mảng trong C# có nhiều tính năng vượt trội hơn so với C++. Mảng được cấp phát bộ nhớ trong heap và do đó nó được truyền bằng tham khảo. Bạn không thể truy xuất một phần tử vượt ngoài giới hạn trong một mảng (có chỉ số lớn hơn số phần tử trong mảng). Do đó C# đã khắc phục lỗi này. Ngoài ra C# còn cung cấp một số hàm trợ giúp để xử lý các phần tử trong mảng. Ta có thể thấy rõ sự khác nhau giữa cú pháp của mảng trong C++ và C# là:

·Dấu ngoặc vuông được đặt sau tên kiểu chứ không phải sau tên biến.

·Bạn có thể tạo vùng nhớ cho phần tử trong mảng bằng cách dùng từ khóa new.

Ngoài ra C# còn hỗ trợ việc hiện thực mảng một chiều (single dimensional), đa chiều (multi dimensional) và jagged array (mảng của mảng). Ví dụ như:

```
 // mảng một chiều đơn giản
int[] array = new int[10];
for (int i = 0;i < array.Length; i++)
array[i] = i;
// mảng hai chiều
int[,] array2 = new int[5,10];
array2[1,2] = 5;
// mảng 3 chiều
int[,,] array3 = new int[5,10,5];
array3[0,2,4] = 9;
// mảng của mảng
int[][] arrayOfarray = = new int[2];
arrayOfarray[0] = new int[4];
arrayOfarray[0] = new int[] {1,2,15};
```

----Indexer

Indexer được dùng để viết một phương thức truy xuất trực tiếp một phần tử từ một tập hợp bằng cách dùng dấu [ ], như trong mảng. Việc bạn cần làm chỉ là chỉ rõ chỉ số để truy xuất một phần tử. Cú pháp của một indexer cũng giống như của thuộc tính một class, ngoại trừ chúng cần một thông số nhập, đó chính là chỉ số của phần tử cần truy xuất.

Bây giờ chúng ta tiếp tục tham khảo một ví dụ cho vấn đề này. Trong ví dụ sau bạn sẽ bắt gặp lớp CollectionBase, đó là một lớp thư viện dùng để tạo ra những tập hợp. Danh sách (list) là một **protected member** của lớp **CollectionBase,**

trong đó lưu trữ tập hợp các danh sách.

```
 class Shapes: CollectionBase
{
public void add(Shape shp)
{
List.Add(shp);
}
//indexer
public Shape this[int index]
{
get {
return (Shape) List[index];
}
set {
List[index] = value ;
}
}
}
```

Boxing/Unboxing
Boxing là một khái niệm mới trong C#. Như đã đề cập ở trên, mọi kiểu dữ liệu, dựng sẵn hay do người dùng định nghĩa, đều được lấy từ một lớp cơ bản là Object trong namespace System.
Do đó việc đóng gói những kiểu căn bản hay nguyên thủy vào trong class Object được gọi là boxing, và thao tác ngược lại được gọi là unboxing. Ví dụ như:

```
 class Test
{
static void Main()
{
int myInt = 12;
// boxing
object obj = myInt ;
// unboxing
int myInt2 = (int) obj;
}
}
```

Ví dụ trên cho ta thấy cả hai thao tác boxing và unboxing. Một giá trị int có thể được chuyển đổi thành đối tượng và chuyển đổi trở lại thành int. Khi kiểu dữ liệu của một biến cần được chuyển thành một kiểu truyền bằng tham khảo, một object box được tạo ra để chứa giá trị, và giá trị được lưu vào box. **Unboxing** chỉ là quá trình ngược lại. Khi một object box được trả lại thành kiểu nguyên thủy , giá trị sẽ được chuyển từ box sang ô nhớ lưu trữ ban đầu.
---Thông số của hàm
Trong C# có 3 loại thông số:
1.Thông số in/ truyền bằng trị.
2.Thông số in– out/truyền bằng tham khảo.
3.Thông số out.
Nếu bạn đã nắm rõ về COM interface và những kiểu thông số của nó, bạn sẽ dễ dàng hiểu được các kiểu thông số của C#.
--Thông số in/ truyền bằng trị
Khái niệm thông số truyền bằng trị cũng tương tự như trong C++. Giá trị truyền được chép vào một ô nhớ và được truyền vào hàm.
Ví dụ:

```
SetDay(5);
...
void SetDay(int day)
{
....
}
```

---Thông số in – out/truyềnbằng tham khảo
Thông số truyền bằng tham khảo trong C++ được truyền thông qua một con trỏ hay toán tử &. Trong C#, thông số truyền bằng tham khảo còn được gọi là thông số in – out, vì khi bạn truyền một địa chỉ tham khảo của một ô nhớ, bạn đã truyền một giá trị nhập và lấy một giá trị xuất từ hàm đó.
Bạn không thể truyền một thông số chưa khởi tạo vào một hàm. C# dùng từ khóa ref để chỉ thông số truyền bằng tham khảo. Bạn cũng có thể dùng từ khóa ref cho một đối số trong khi truyền nó vào một hàm có thông số truyền bằng tham khảo. Ví dụ:

```
 int a= 5;
FunctionA(ref a);
Console.WriteLine(a);
void FunctionA(ref int Val)
{
    int x= Val;
    Val = x* 4;
}
```

----Thông số out
Là thông số chỉ trả về giá trị là kết quả của một hàm, không đòi hỏi giá trị nhập. C# dùng từ khóa out cho loại tham số này.

```
 int Val;
GetNodeValue(Val);
bool GetNodeValue(out int Val)
{
Val = value;
return true;
}
```

---Số lượng các thông số và mảng
Để truyền thông số là một mảng trong C#, người ta dùng từ khóa **params**. Chỉ có thể có một đối số kiểu mảng. Bạn có thể truyền phần tử như là một đối số của mảng đó. Tốt nhất, ta hãy xem ví dụ sau:

```
 void Func(params int[] array)
{
    Console.WriteLine("number of elements {0}", array.Length);
}
    Func();
    Func(5);
    Func(7,9);
    Func(new int[] {3,8,10});
    int[] array = new int[8] {1,3,4,5,5,6,7,5};
    Func(array);
```

---Toán tử và biểu thức
Hầu hết các toán tử trong C# hoàn toàn giống như trong C++, do đó biểu thức cũng vậy. Tuy nhiên C# còn bổ sung thêm

một số toán tử mới và hữu ích. Chúng ta sẽ xem xét vài toán tử trong số đó.

---Toán tử is

Toán tử này được dùng để kiểm tra xem kiểu của các toán hạng có tương đương hay không. Toán tử is thường được sử dụng trong kịch bản đa ngữ cảnh. Toán tử này có hai toán hạng và kết quả trả về có kiểu **bool.** Chúng ta xem ví dụ sau:

```
void function(object param)
{
    if(param is ClassA)
// something
        else if(param is MyStruct)
//something
}
}
```

---Toán tử as

Toán tử as kiểm tra xem kiểu của các toán hạng có khả đổi hay không, nếu có thì kết quả trả về là một đối tượng đã được chuyển đổi hay được box. Nếu đối tượng không chuyển đổi hay box được, kết quả trả về null. Chúng ta xem ví dụ sau để hiểu rõ hơn về khái niệm này:

```
 Shape shp = new Shape();
//kết quả là null, không chuyển đổi
//kiểu được
Vehicle veh = shp as Vehicle;
Circle cir = new Circle();
Shape shp = cir;
//sẽ được chuyển đổi
Circle cir2 = shp as Circle;
object[] objects = new object[2];
objects[0] = “Aisha”;
object[1] = new Shape();
string str;
for(int i=0; i&< objects.Length; i++)
{
    str = objects[i] as string;
    if(str == null)
    Console.WriteLine(“can not be converted”);
    else
        Console.WriteLine(“{0}”,str);
}
```

----Câu lệnh

Đa số các câu lệnh trong C# đều tương tự như trong C++, ngoài ra còn có một số câu lệnh mới được bổ sung và có một vài sự sửa đổi trong một số câu lệnh cũ. Sau đây là một số câu lệnh mới:
foreach: dùng cho việc thực hiện vòng lặp cho một tập hợp như mảng… Ví dụ:

```
 foreach (string s in array)
Console.WriteLine(s);
```

lock: dùng để bao một đoạn code thành một section.
checked/unchecked: dùng để kiểm tra tràn trong các toán tử có đối số là số. Ví dụ như:

```
 int x = Int32.MaxValue; x++;
```

```
{
    x++;
}
unchecked
{
    x++; }
}
```

---Switch
Trong C#, câu lệnh switch được thay đổi như sau:
1.Sau khi thực thi một câu lệnh case, chương trình sẽ không nhảy đến câu lệnh case kế tiếp. Ví dụ:

```
int var = 100;
switch (var)
{
    case 100:
    Console.WriteLine(“<Value is 100>”); // không dùng break
    case 200:
    Console.WriteLine(“<Value is 200>”); break;
}
```

Trong C++, kết quả sẽ là: <Value is 100><Value is 200>
Trong C#, bạn sẽ nhận được thông báo lỗi sau:
error CS0163: Control cannot fall through from one case label (‘case 100:’) to another

Tuy nhiên, bạn có thể làm như sau:

```
 switch (var)
{
     case 100:
     case 200:
     Console.WriteLine(“100 or 200<VALUE is 200>”);
     break;
}
```

2.Bạn cũng có thể dùng hằng trong giá trị của case. Ví dụ như:

```
 const string WeekEnd = “Sunday”;
const string WeekDay1 = “Monday”;
....
string WeekDay =
Console.ReadLine();
switch (WeekDay )
{
    case WeekEnd:
    Console.WriteLine(“It’s weekend!!”);
    break;
    case WeekDay1:
```

```
    Console.WriteLine("It's Monday");
    break;
}
```

----Delegate

Delegate cho phép chúng ta lưu sự tham khảo hàm vào một biến. Trong C++, việc này giống như dùng và lưu con trỏ hàm và chúng ta hay dùng **typedef**. Ví dụ:

```
 delegate int Operation(int val1, int val2);
public int Add(int val1, int val2)
{
  return val1 + val2;
}


public int Subtract (int val1, int val2)
{
  return val1- val2;
}


public void Perform()
{
Operation Oper;
Console.WriteLine("Enter + or - ");
string optor = Console.ReadLine();
Console.WriteLine("Enter 2 operands");
string opnd1 = Console.ReadLine();
string opnd2 = Console.ReadLine();
int val1 = Convert.ToInt32 (opnd1);
int val2 = Convert.ToInt32 (opnd2);
if (optor == "+")
Oper = new Operation(Add);
else
Oper = new Operation(Subtract);
Console.WriteLine(" Result ={0}", Oper(val1, val2));
}
```

Tính thừa kế và tính đa hình

Trong C# chỉ cho phép sự thừa kế đơn. Nếu bạn muốn thực hiện đa thừa kế, bạn có thể dùng interface. Ví dụ:

```
 class Parent{
}
class Child : Parent
```

Hàm ảo

Từ khái niệm hàm ảo đến hiện thực khái niệm đa hình trong C# là như nhau, ngoại trừ việc bạn dùng từ khóa override đối với việc hiện thực hàm ảo trong lớp con. Lớp cha vẫn sử dụng từ khóa **virtual.** Lớp nào **override** phương thức ảo cũng sử dụng từ khóa override.

```
class Shape
{
     public virtual void Draw()
     {
     Console.WriteLine(“Shape.Draw”);
      }
  }


class Rectangle : Shape
{
     public override void Draw()
   {
Console.WriteLine(“Rectangle.Draw”);
   }
}


class Square : Rectangle
{
    public override void Draw()
   {
    Console.WriteLine(“Square.Draw”);
   }
}


class MainClass
{
   static void Main(string[] args)
   {
Shape[] shp = new Shape[3];
Rectangle rect =
new Rectangle();
shp[0] = new Shape();
shp[1] = rect;
shp[2] = new Square();
shp[0].Draw();
shp[1].Draw();
shp[2].Draw();
   }
} Xuất ra kết quả như sau:
```

Shape.Draw
Rectangle.Draw
Square.Draw

Ẩn lớp cha bằng cách dùng “new” Trong một lớp con, bạn có thể định nghĩa một hàm mới, ẩn với lớp cha, bằng cách dùng từ khóa new. Trong ví dụ dưới đây, là bản sửa đổi của ví dụ trên, tôi thay thế từ khóa **override** bằng từ khóa new trong lớp **Rectangle.**

```
 class Shape
```

```
{
   public virtual void Draw()
   {
  Console.WriteLine(“Shape.Draw”);
   }
}
   class Rectangle : Shape

{
 public new void Draw()
   {
    Console.WriteLine(“Rectangle.Draw”);
   }
}


 class Square : Rectangle
{
public new void Draw()
  {
   Console.WriteLine(“Square.Draw”);
   }
 }
class MainClass
{
static void Main(string[] args)
     {
Console.WriteLine(“Using Polymorphism:”);
Shape[] shp = new Shape[3];
Rectangle rect = new Rectangle();
shp[0] = new Shape();
shp[1] = rect;
shp[2] = new Square();
shp[0].Draw();
shp[1].Draw();
shp[2].Draw();
Console.WriteLine(“Using without Polymorphism:”);
rect.Draw();
Square sqr = new Square();
sqr.Draw();
    }
}
```

Kết quả xuất ra như sau:
Using Polymorphism
Shape.Draw
Shape.Draw
Shape.Draw
Using without Polymorphism:
Rectangle.Draw
Square.Draw

Trong ví dụ trên, phương thức Draw của lớp Rectangle không phải là dạng đa hình của phương thức Draw trong lớp Shape. Thay vì vậy, nó được xem như là một phương thức khác. Do đó, để tránh sự trùng tên giữa lớp cha và lớp con, ta

nên dùng từ khóa new.
Lưu ý: bạn không thể dùng hai phương thức cùng tên trong cùng một lớp nếu một phương thức dùng từ khóa new, phương thức kia dùng **override hay virtual.**
Do đó trong lớp Square, tôi không thể override phương thức **Draw của lớp Shape**. Gọi những member của lớp cha Nếu lớp con có dữ liệu member cùng tên với dữ liệu đó trong lớp cha, để tránh bị trùng tên, dữ liệu và hàm member của lớp cha được truy xuất thông qua từ khóa base. Trong ví dụ sau, hãy xem cách **constructor** của lớp cha được gọi và dữ liệu member được dùng:

```
 public Child(int val) :base(val)
  {
   myVar = 5;
   base.myVar;
}
```

Hay

```
 public Child(int val)
{
   base(val);
   myVar = 5 ;
   base.myVar;
}
```

Bài viết này chỉ giới thiệu một cách rất tổng quát về ngôn ngữ C# để các bạn có thể quen với những đặc trưng cơ bản của ngôn ngữ. Mặc dù đã cố gắng đề cập đến hầu hết những khái niệm chính trong C#, nhưng tôi nghĩ vẫn còn rất nhiều thứ khác cần thêm vào và bàn đến.

(Theo: *Trần Phạm thanh Tùng - Báo Học Lập Trình*)
Nguồn: tincntt.com