

C# 10

Bill Chung

https://mvc.tw

about me

- Bill Chung
- 專長是說故事
- 海角點部落 https://dotblogs.com.tw/billchung
- github https://github.com/billchungiii

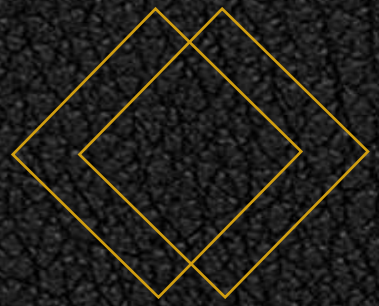
請尊重講師的著作權及智慧財產權!
此課程之教材、程式碼等、僅供課程中學習用、請不要任意自行散佈、重製、分享，謝謝



大綱

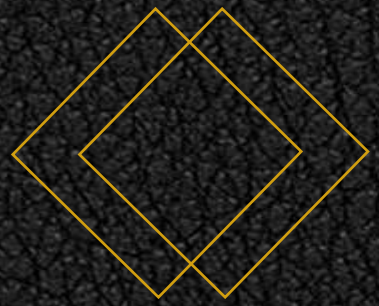
- Name Space
- Extended property patterns
- Structure
- Lambda
- String
- Caller Argument Expression

Name Space



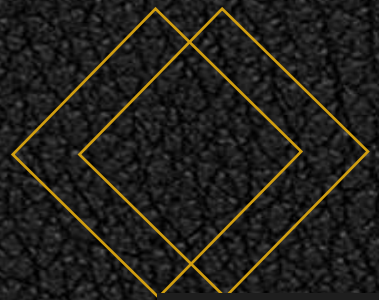
global usings

- 一次設定，專案全域引入
- global using [namespace]

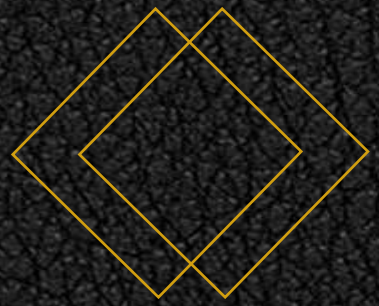


implicit usings

- .NET 6
- 編譯器會自動引入基本的命名空間
- 檔案藏在 obj 目錄下 [project name].GlobalUsings.g.cs



```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
  <ItemGroup>
     <Using Include="System.Reflection" />
  </ItemGroup>
</Project>
```



file-scoped namespace

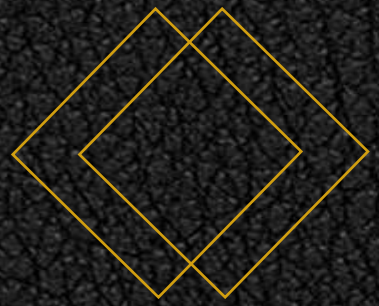
```
namespace ConsoleApp1;
internal class Class1 { }
```



```
namespace ConsoleApp1
{
    internal class Class1     { }
}
```

MVC
https://mvc.tw

Extended Property Patterns



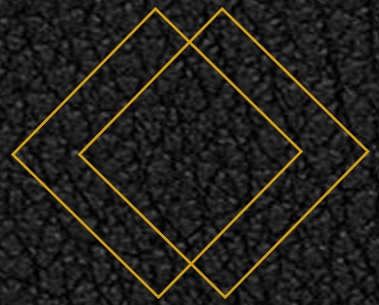
extended property patterns

```
if (p is (string firstName, "Chung", _) { Address: { City: string city, ZipCode: "100" } })
```



```
if (p is (string firstName, "Chung", _) { Address.City: string city, Address.ZipCode: "100" })
```

Structure



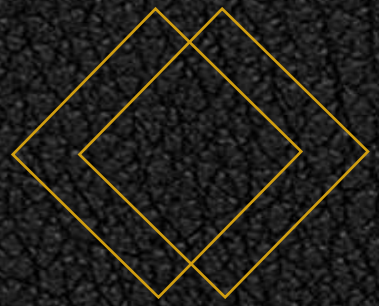
parameterless constructors

```
struct MyStructure
{
    public double Value { get; init; }
    public string Name { get; init; }

    public MyStructure()
    {
        Value = double.NaN;
        Name = string.Empty;
    }

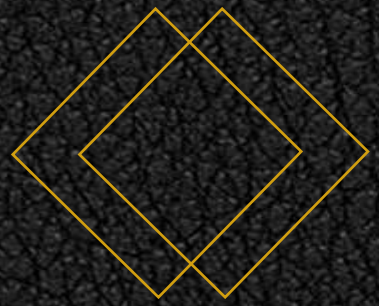
    public MyStructure(int value, string name)
    {
        Value = value;
        Name = name;
    }

    public override string ToString()
    {
        return $"{Value},{Name}";
    }
}
```



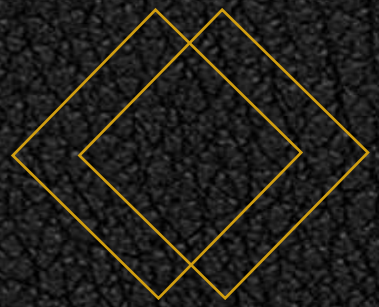
default value expressions

- 回傳 field / property 型別的預設值，而非初始化設定的值。



filed / property initializer

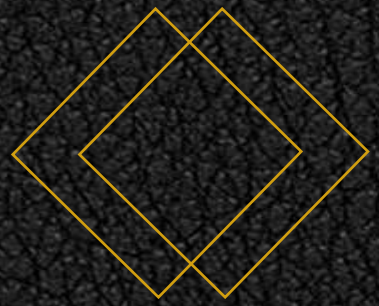
```
struct MyStructure
 {
     public double Value { get; set; }
     public string Name { get; set; } = "Bill";
```



record strcutrue

```
public readonly record struct Point(double X, double Y, double Z);

public readonly record struct Point
{
    public double X { get; init; }
    public double Y { get; init; }
    public double Z { get; init; }
}
```

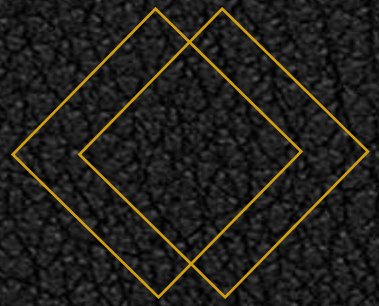


with expression

- 過去：with 敘述僅能用在 record
- 現在：可以用在 structure 和 anonymous type

```
var s1 = new MyStructure();
var s3 = s1 with { Name = "ABC", Value = 100.0 };
```

Lambda

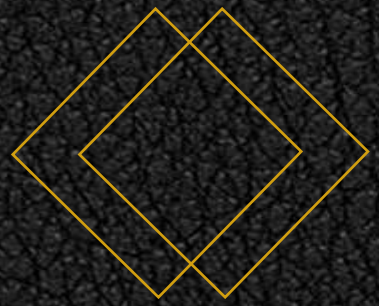


Infer delegate type

```
var parse1 = (string s) => int.Parse(s);
object parse2 = (string s) => int.Parse(s);
Delegate parse3 = (string s) => int.Parse(s);

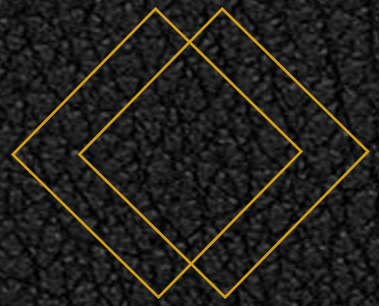
var read = Console.Read;

LambdaExpression parseExpr = (string s) => int.Parse(s);
Expression parseExpr = (string s) => int.Parse(s);
```



Declared return type

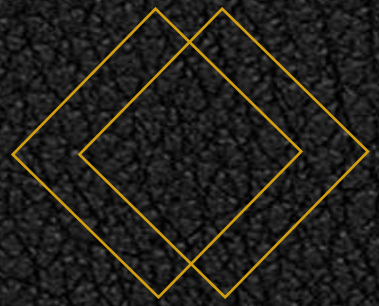
```
var choose = object(bool b) => b? 1 : "two";
```



Attributes

```
Func<string, int> parse =[Some(1)] (s) => int.Parse(s);
```

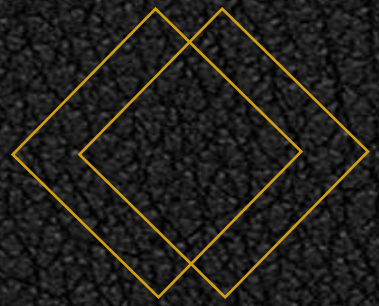
String



String Constant

- 字串常數可使用字串插值功能，但必須都是常數

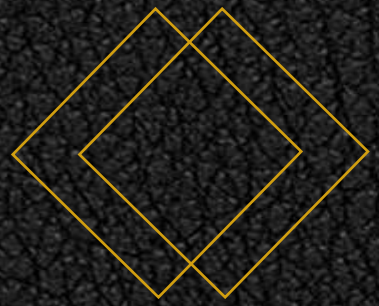
```
const string Name = "Bill";
const string ProgramName = $"{Name} program";
```



Interpolated string handlers

- 字串插補可以自定義處理常式
- 自訂處理常式需求
  - 套用 InterpolatedStringHandlerAttribute
  - 建構式需要兩個 int 型別參數 -- literalLength and formatCount
  - public void AppendLiteral(string s) method
  - public void AppendFormatted<T>(T t) method

Caller Argument Expression



CallerArgumentExpressionAttribute

- 無需使用反射就能得到呼叫端引數的表達式字串

```
public static void CheckExpression(bool condition, [CallerArgumentExpression("condition")] string? message = null)
{
    Console.WriteLine($"Condition: {message}");
}

public static void CheckDelegate(Func<int, int, int> func, [CallerArgumentExpression("func")] string? message = null)
{
    Console.WriteLine($"Func is  : {message}");
}

public static void CheckSquence(IEnumerable<int> source, [CallerArgumentExpression("source")] string? message = null)
{
    Console.WriteLine($"Squence is  : {message}");
}
```


The END
MVC
https://mvc.tw