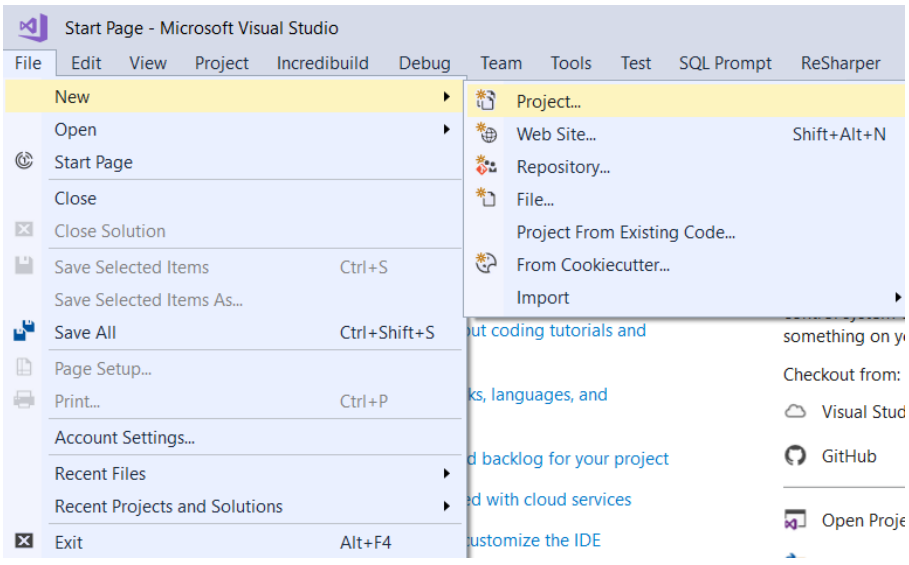
(T3)討論 Namespace。比較 StaticMethod、InstanceMethod。比較 Ref、Out、params。比較 MethodOverride，MethodHide
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==========
(T3)討論 Namespace。比較 StaticMethod、InstanceMethod。比較 Ref、Out、params。比較 MethodOverride，MethodHide
==========
1. Create New Project
2. Program.cs
==========

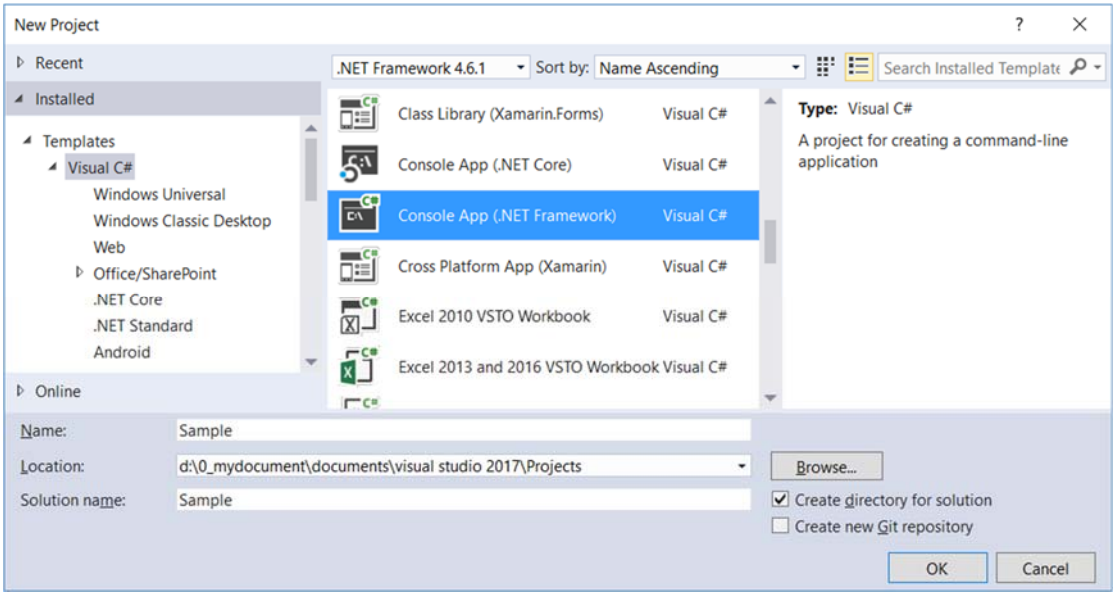
# 0. What to learn

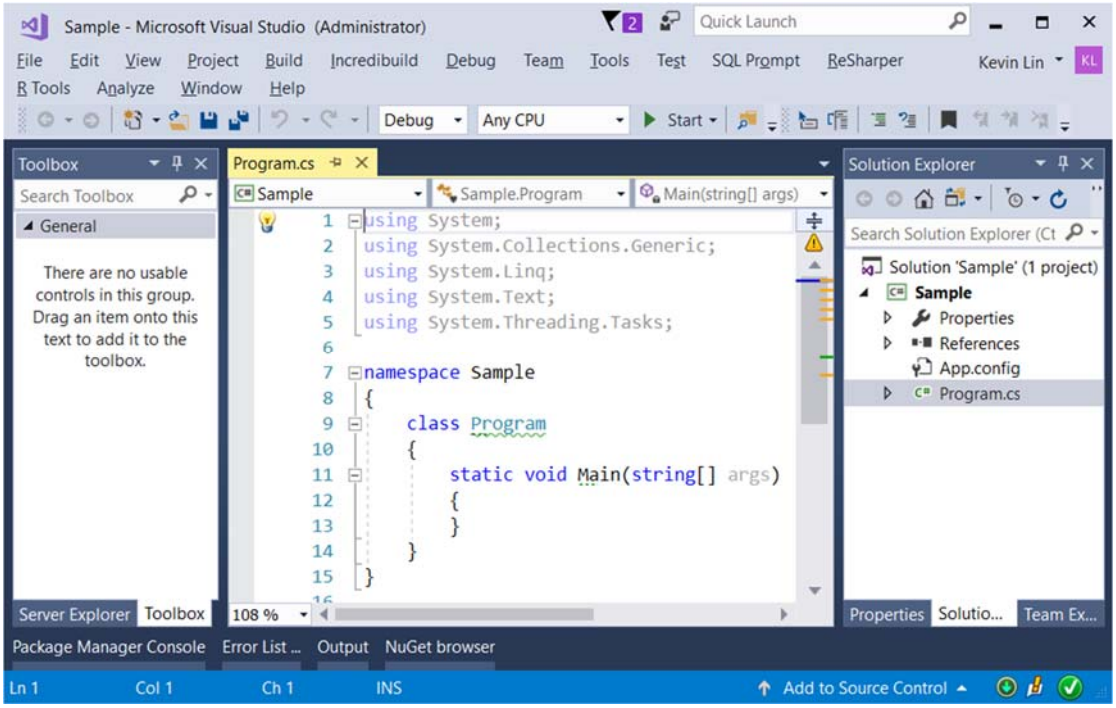
Static method V.S. Instance method
Ref keyword
Out keyword
params keyword
Method override V.S. Method Hide

# 1. Create New Project

File --> New --> Project... -->
Visual C# --> **Console App (.Net Framework)** -->
Name: **Sample**






==========

# 2. Program.cs

```
// Namespace is like a folder which contains
// another namespace, class, interface, struct, enum delegate.
using System;
using AA11 = FolderA.FolderA1.FolderA1_1;
using AA2 = FolderA.FolderA2;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1. Method
```

```
Console.WriteLine("Method ------------------------------");
// Static Method
Program.StaticMethod(15);
// Instance Method
Program p = new Program();
int sum = p.Sum(2,3);
Console.WriteLine("sum = {0}", sum);
// valued types parameter, pass by value
int i = 0;
Program.ValuedTypesParameter(i);
Console.WriteLine(i);   // Still 0
// Ref keyword, pass by reference
int i2 = 0;
Program.Refkeyword(ref i2);
Console.WriteLine(i2);   // 100
// Out keyword
int outSum = 0;
double outAvg = 0;
Program.Outkeyword(2, 3, out outSum, out outAvg);
Console.WriteLine("int outSum = {0}; double outAvg = {1}", outSum, outAvg);
//int outSum = 5; double outAvg = 2
// params keyword
int[] paramsKeywordArr = {1, 2, 3};
 ParamsKeyword();
 ParamsKeyword(paramsKeywordArr);
 ParamsKeyword(10, 9, 8);
// 2. namespace
Console.WriteLine("namespace ------------------------------");
 FolderA.FolderA1.FolderA1_1.ClassA.WriteLine();
 FolderA.FolderA2.ClassA.WriteLine();
 FolderB.ClassA.WriteLine();
// namespace alias.
 AA11.ClassA.WriteLine();
 AA2.ClassA.WriteLine();
 FolderB.ClassA.WriteLine();
// 3. Static
Console.WriteLine("Static ------------------------------");
Console.WriteLine(OnlineGame.NumberOfMonstersInAreaA);  //5
OnlineGame onlineGameSession1 = new OnlineGame();
Console.WriteLine(OnlineGame.NumberOfMonstersInAreaA);  //6
OnlineGame.KillOneMonsterInAreaA();
Console.WriteLine(OnlineGame.NumberOfMonstersInAreaA);  //5
OnlineGame onlineGameSession2 = new OnlineGame();
Console.WriteLine(OnlineGame.NumberOfMonstersInAreaA);  //6
// 4. Method override V.S. Method Hide
Console.WriteLine("Method override V.S. Method Hide ------------------------------");
Base1 b1 = new Sub1();
 b1.Print();
//display "Sub Class"
//Method override means
//when baseClass has Print() with virtual
//and subClass has Print() with override.
//RHS is "new" Sub1() object.
//LHS is the "Base1" typed variable b1.
//b1.Print() will invoke the sub class Print() method.
```

```
        Base2 b2 = new Sub2();
        b2.Print();
        // display "Base Class"
        //Method hide means
        //when baseClass has Print()
        //and subClass has Print() without override.
        //RHS is "new" Sub2() object.
        //LHS is the "Base2" typed variable b2.
        //b2.Print() will invoke the base class Print() method.
        Console.ReadLine();
    }

    // 1. Method ----------------------------------------------------
    // Instance Method
    public int Sum(int i1, int i2)
    {
        return i1 + i2;
    }
    // Static Method
    public static void StaticMethod(int parameter1)
    {
        Console.WriteLine("Static Method {0}", parameter1);
    }
    //valued types parameter, pass by value
    public static void ValuedTypesParameter(int parameter1)
    {
        parameter1 = 100;
    }
    //Ref keyword, pass by reference
    public static void Refkeyword(ref int parameter1)
    {
        parameter1 = 100;
    }
    //Out keyword
    public static void Outkeyword(int i1, int i2, out int sum, out double avg)
    {
        sum = i1 + i2;
        avg = sum / 2;
    }
    // params keyword
    public static void ParamsKeyword(params int[] numbers)
    {
        Console.WriteLine("ParamsKeyword-------------------------");
        foreach (int item in numbers)
        {
            Console.WriteLine(item);
        }
    }
    //5. Method Overloading ------------------------------------------
    //allows methods have the same name but different method signature head.
    public static void Add(int a, int b)
    {
        Console.WriteLine("a + b = {0}", a+b);
    }
    ////public static string Add(int a, int b)
    ////{
    ////    return a + b;
    ////}
```

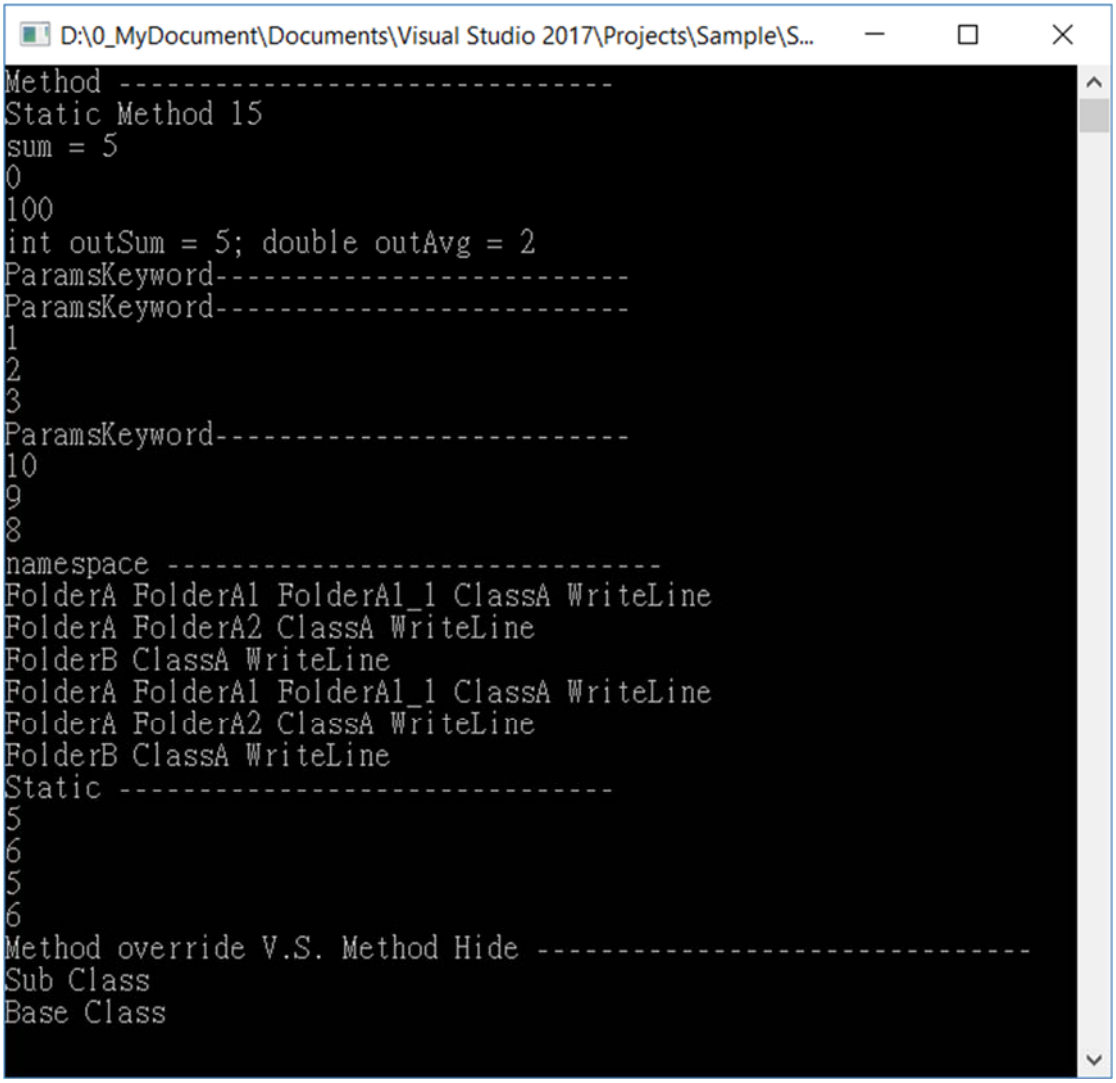
```
        //public static string Add(int a, int b)
        //and
        //public static void Add(int a, int b)
        //are actually having the same method signature.
        public static void Add(int a, int b, int c)
        {
            Console.WriteLine("a + b = {0}", a + b+c);
        }
        public static void Add(int a, int b, out int c)
        {
            Console.WriteLine("a + b = {0}", a + b);
            c = a + b;
        }
        public static void Add(int a, double b)
        {
            Console.WriteLine("a + b = {0}", (double)a + b);
        }
        public static void Add(int a, int b, int[] c)
        {
        }
        ////public static void Add(int a, int b, params int[] c)
        ////{
        ////}
        //public static void Add(int a, int b, params int[] c)
        //and
        //public static void Add(int a, int b, int[] c)
        //are actually having the same method signature.
    }
}


// 2. namespace ------------------------------------------------------
namespace FolderA
{
    namespace FolderA1
    {
        namespace FolderA1_1
        {
            public class ClassA
            {
                public static void WriteLine()
                {
                    Console.WriteLine("FolderA FolderA1 FolderA1_1 ClassA WriteLine");
                }
            }
        }
    }
    namespace FolderA2
    {
        public class ClassA
        {
            public static void WriteLine()
            {
                Console.WriteLine("FolderA FolderA2 ClassA WriteLine");
            }
        }
    }
}
namespace FolderB
```

```
{
    public class ClassA
    {
        public static void WriteLine()
        {
            Console.WriteLine("FolderB ClassA WriteLine");
        }
    }
}
// 3. Static ------------------------------------------------------
public class OnlineGame
{
    public static int NumberOfMonstersInAreaA;
    static OnlineGame()
    {
        NumberOfMonstersInAreaA = 5;
    }
    public OnlineGame()
    {
        NumberOfMonstersInAreaA = 6;
    }
    public static void KillOneMonsterInAreaA()
    {
        NumberOfMonstersInAreaA-=1;
    }
}
// 4. Method override V.S. Method Hide --------------------------------
public class Base1
{
    public virtual void Print()
    {
        Console.WriteLine("Base Class");
    }
}
public class Sub1 : Base1
{
    public override void Print()
    {
        Console.WriteLine("Sub Class");
    }
}
public class Base2
{
    public void Print()
    {
        Console.WriteLine("Base Class");
    }
}
public class Sub2 : Base2
{
    public void Print()
    {
        Console.WriteLine("Sub Class");
    }
}



/*
1.
Method (or called Function)
[attributes]
```

```
//AccessLevel (virtual) ReturnType methodName(Parameters)
//{
//    MethodBody
//}
1.1.
Method is a set of logic process
that you may repeatly perform in many place
by calling it.
1.2.
Only virtual method can be overrided in the sub-class.
1.3.
attributes can add some extra information to the method.
We will discuss later.
1.4.
//AccessLevel
//public / protected / private
Reference:
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels
Accessibility Levels includes several levels.
Here, we only discuss, public, protected, and private.
public means access is not restricted.
protected means access is limited to the containing class or types derived from the containing class.
private means access is limited to the containing type.
1.5.
//ReturnType
Return type is the valid data type this method return.
//E.g. public string Name() { return "AAA"};
if the ReturnType is string,
then the method have to return string in the end of method.
//E.g. public void Name(){}
If the ReturnType is void,
then the method does not need to return anything.
1.6.
Parameters are optional
-----------------------------------------------------------
2.
Static method V.S. Instance method
2.1.
// ClassName.StaticMethod()
You may call StaticMethod without creating the instance of the class.
2.2.
// ClassA classA = new ClassA();  classA.InstanceMethod();
You have to create an object of the class before you use InstanceMethod.
2.3.
class is like a blueprint or template.
object is a single instance of the class.
// ClassA classA = new ClassA();
Left hand side:  ClassA classA   means declare classA variable(like container)
which type is ClassA (like the container can only contain ClassA type object.)
Right hand side: new ClassA();
new is verb means create.
Use ClassA as template to create an object instance of ClassA
2.4.
InstanceMethod can only be used for that specific instance object.
All the instance object share the same StaticMethod();
E.g.
In the online game, there are 5 monster in areaA.
Now, I kill one, so only 4 left.
Thus, the game should use some kind of static method
to record the number of monsters in areaA.
-----------------------------------------------------------
3.
Method parameter types
3.1
By default, valued types parameter
creates a copy of the parameter and pass into method.
E.g.
```

```
Pass by value:
//int i = 0;
//Program.ValuedTypesParameter(i);
//Console.WriteLine(i);   // Still 0
...
//public static void ValuedTypesParameter(int parameter1)
//{
//    parameter1 = 100;
//}
i --> 0
parameter1 --> 100
i and parameter1 are pointing to different memory location.
-----------------------------
3.2.
Ref keyword
Pass the memory address of the variable into the method.
Any changes made to the parameter in the method
will be reflected back to the method caller,
because the Reference Parameters memory location
and the original caller memory location
are actully pointing to the same location.
E.g.
//int i2 = 0;
//Program.Refkeyword(ref i2);
//Console.WriteLine(i2);   // retuns 100
...
//public static void Refkeyword(ref int parameter1)
//{
//    parameter1 = 100;
//}
i2 __________
             |----> 100
parameter1 __|
i2 and parameter1 are pointing to the same memory location.
Any changes made to the parameter1 in the method
will be reflected back to the i2.
-----------------------------
3.3.
Out keyword
Out keyword allow you to return more than one value.
-----------------------------
3.4.
params keyword
allows you to optionally pass an array[].
The method can take no parameter or an array
or a comma-separated list of arguments.
You may only has one params parameter
and must be the last one in a method declaration.
E.g.
//int[] paramsKeywordArr = {1, 2, 3};
//ParamsKeyword();
//ParamsKeyword(paramsKeywordArr);
//ParamsKeyword(10, 9, 8);
-------------------------------------------------------
4.
Namespace is like a folder which contains
another namespace, class, interface, struct, enum delegate.
E.g.
//using AA11 = FolderA.FolderA1.FolderA1_1;
//using AA2 = FolderA.FolderA2;
//using B = FolderB;
...
//FolderA.FolderA1.FolderA1_1.ClassA.WriteLine();
//FolderA.FolderA2.ClassA.WriteLine();
//FolderB.ClassA.WriteLine();
//// namespace alias.
//AA11.ClassA.WriteLine();
```

```
//AA2.ClassA.WriteLine();
//B.ClassA.WriteLine();

-------------------------------------------------------
5.
Static field V.S. Instance field
Static Constructor V.S. Instance Constructor
5.1.
InstanceField/InstanceMethod/InstanceConstructor
can only be used for that specific instance object.
All the instance object share the same copy of
StaticField/StaticMethod/StaticConstructor
No matter how many instances you create,
StaticConstructor is called only once before you "new" the first instance.
Thus, StaticConstructor is called before InstanceConstructor
E.g.
In the online game, there are 5 monster in areaA.
Now, I kill one, so only 4 left.
Thus, the game should use some kind of static method
to record the number of monsters in areaA.
-------------------------------------------------------
6.
Method override V.S. Method Hide
6.1.
Method override means
when baseClass has Print() with virtual
and subClass has Print() with override.
RHS is "new" Sub1() object.
LHS is the "Base1" typed variable b1.
b1.Print() will invoke the sub class Print() method.
6.2.
Method hide means
when baseClass has Print()
and subClass has Print() without override.
RHS is "new" Sub2() object.
LHS is the "Base2" typed variable b2.
b2.Print() will invoke the base class Print() method.
*/
```


D:\0_MyDocument\Documents\Visual Studio 2017\Projects\Sample\S...
Method ------------------------------
Static Method 15
sum = 5
0
100
int outSum = 5; double outAvg = 2
ParamsKeyword-------------------------
ParamsKeyword-------------------------
1
2
3
ParamsKeyword-------------------------
10
9
8
namespace ------------------------------
FolderA FolderA1 FolderA1_1 ClassA WriteLine
FolderA FolderA2 ClassA WriteLine
FolderB ClassA WriteLine
FolderA FolderA1 FolderA1_1 ClassA WriteLine
FolderA FolderA2 ClassA WriteLine
FolderB ClassA WriteLine
Static ------------------------------
5
6
5
6
Method override V.S. Method Hide ------------------------------
Sub Class
Base Class