(T5)比較 public、protected、private。比較 AbstractClass、Interface
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==

(T5)比較 public、protected、private。比較 AbstractClass、Interface

==

==

0. Summary

1.
Interface
1.1.
Interface is like an product booklet which contains
the standard actions that this product must be able to do.
Interface only contains the method signature without its body.
E.g.
IAircraft must be able to take off and land.

1.2.
Interface can contain declarations signature head of properties, methods, delegates, and events,
then the Class or the Struct which implement the interface, has to implement all Interface members
in order to avoid the compiler error.
In addition, Interface can not contain any field.

1.3.
Interface can not be use to create as object instance.
E.g.
//IAircraft aircraft = new IAircraft() //Complier Error

1.4.
Interface can be used as the Reference Variable Type(LHS).
E.g.
//IBase b1 = new SubClass();

1.5.
The prefix of interface is "I".

1.6.
By default, Interface is Public, and it does not allow explicit access modifiers.
A class can extend only one class and implement many Interfaces,
and the base Class must be front of Interface.
E.g.
//public class ClassA : ClassB, InterfaceA, InterfaceB

1.7.
Interfaces can inherit from other interfaces.
E.g.
//public Interface IBase1 : IBase2

1.8.
//public void IBase3.BaseSame1()   //Error
Accessibility Levels is invalid for explicit interface implementation.
Explicit interface implementation fix

the isse of the same methods signature heads from 2 interfaces.
Explicit interface implementation can not be used in Class Reference Variable(LHS).
It can only be used in Interface Reference Variable(LHS).
Non-explicit interface implementation can be used in Class Reference Variable(LHS).

2.
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.

3.
Abstract Class V.S. Interface

3.1.
An abstract class is incomplete, so it cannot be instantiated.
In addition, an Abstract Class can not be a Sub-Class of other Class.
Abstract Class can only be a Base-Class,
so it must not be Sealed type.

3.2.
Interface:
3.2.1.
An Abstract Class is very similar to Interface.
Interface can contain declarations signature head of properties, methods, delegates, and events,
then the Class or the Struct which implement the interface, must implement all Interface members
in order to avoid the compiler error.
3.2.2.
Interface can not have implementation for any of its members.
3.2.3.
Interface can not have fields.
3.2.4.
Interface members can not have access modifier such as public, private or protected.

3.3.
Abstract Class:
3.3.1.
An Abstract Class may or may not contain abstract members
which includes methods, properties, indexers, and events,
then its Sub-Class may implement some of or all abstract members
in order to avoid the compiler error.
Non-abstract Sub-Class of the Abstract Class must implement all inherited abstract members.
Abstract Sub-Class of the Abstract Class may just implement some inherited abstract members.
3.3.2.
Abstract Class can have implementation for some of its members such as methods.
3.3.3.
Abstract Class can have fields.
3.3.4.
Abstract class members can have access modifier such as public, private or protected.

3.4.
One non-Abstract Class can extend one non-Abstract Class or Abstract Class
and implement many Interfaces.
The base case class must be in front of Interfaces.
E.g. public class ClassA : ClassB, InterfaceA, InterfaceB

3.5.

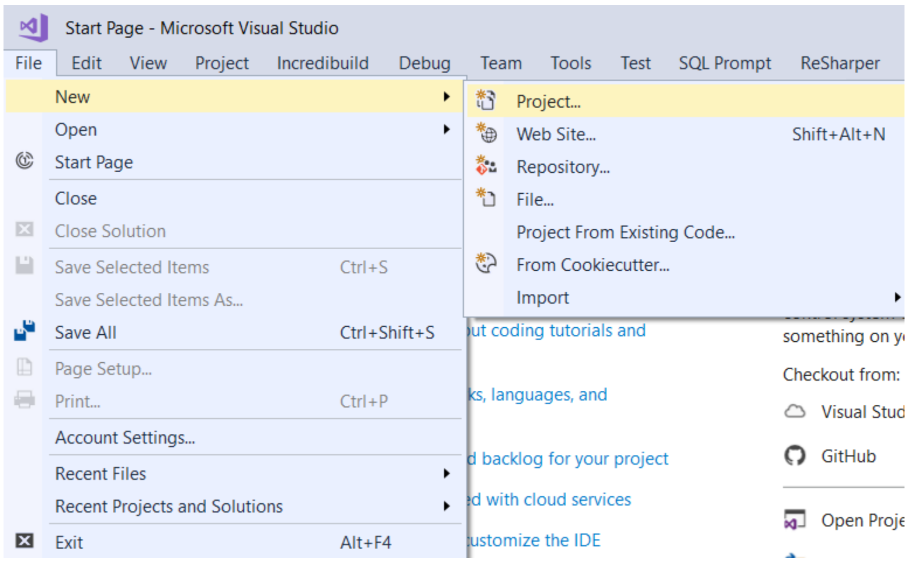
Abstract Class can extend anther Abstract Class
and implement many Interfaces.
The base case class must be in front of Interfaces.
E.g. public class ClassA : ClassB, InterfaceA, InterfaceB

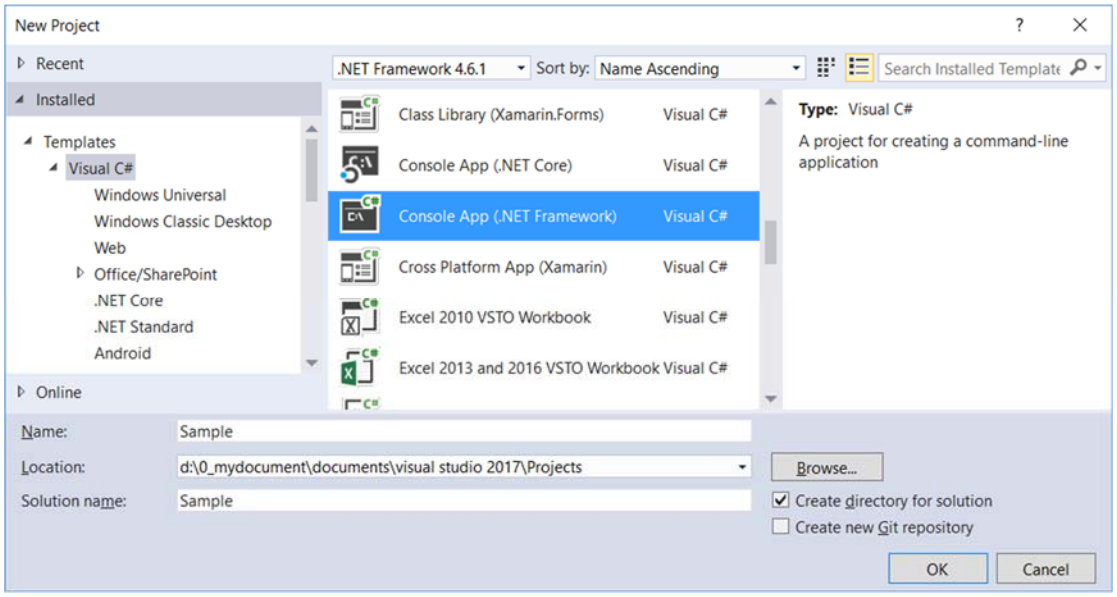
3.6.
Interface can implement many other Interfaces
but can not extend an Abstract Class.
E.g. public Interface Interface01 : InterfaceA, InterfaceB

==

1. Create New Project

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

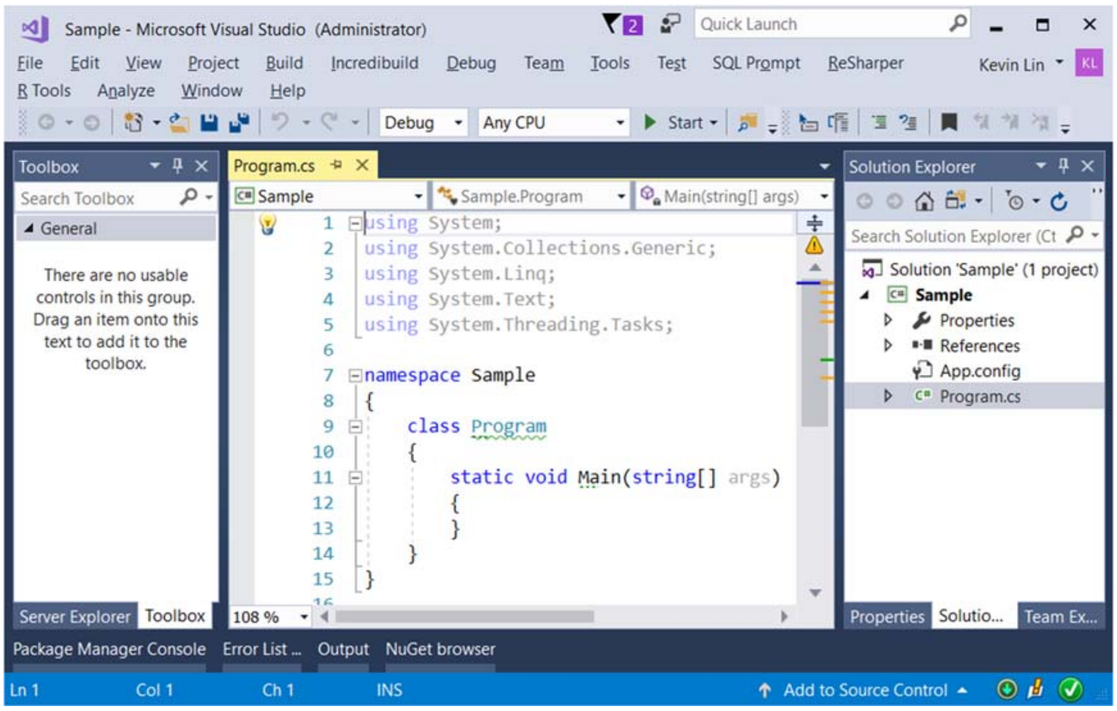




Class Library (Xamarin.Forms)	Visual C#
Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Cross Platform App (Xamarin)	Visual C#
Excel 2010 VSTO Workbook	Visual C#
Excel 2013 and 2016 VSTO Workbook	Visual C#

Type: Visual C#
A project for creating a command-line application

Name: Sample
Location: d:\0_mydocument\documents\visual studio 2017\Projects
Solution name: Sample
[✓] Create directory for solution
[ ] Create new Git repository



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

==

2. Program

```
using System;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1.
            // Interfaces can inherit from other interfaces.
            Console.WriteLine("-------------------------------------");
            ClassA a1 = new ClassA();
             a1.Base1Print();
             a1.Base2Print();
            //Base1Print
            //Base2Print
            IBase2 a2 = new ClassA();
             a2.Base1Print();
             a2.Base2Print();
            //Base1Print
            //Base2Print
            IBase1 a3 = new ClassA();
             a3.Base1Print();
            //Base1Print
            //--------------------------------------------------------------------------------
            // 2.
            // Implement 2 Interfaces with same methods signature.
            Console.WriteLine("-------------------------------------");
            ClassB b1 = new ClassB();
```

```
             b1.BaseSame2();
            //display "BaseSame2   comes from IBase3"
            //Non-explicit interface implementation can be used in Class Reference Variable(LHS).
            //Explicit interface implementation can not be used in Class Reference Variable(LHS)
            Console.WriteLine("-------------------------------------");
            IBase3 b3 = new ClassB();
             b3.BaseSame1();
             b3.BaseSame2();
            //display "IBase3.BaseSame1"
            //display "BaseSame2 comes from IBase3"
            Console.WriteLine("-------------------------------------");
            IBase4 b4 = new ClassB();
             b4.BaseSame1();
             b4.BaseSame2();
            //display "IBase4.BaseSame1"
            //display "IBase4.BaseSame2"
            //------------------------------------------------------------------------------
            // 3.
            // Abstract Class
            Console.WriteLine("-------------------------------------");
            AbstractC abstractC1 = new ClassC();
             abstractC1.AbstractCPrint();
             abstractC1.Print1();
            //AbstractCPrint
            //ClassC
            ClassC abstractC2 = new ClassC();
             abstractC2.AbstractCPrint();
             abstractC2.Print1();
            //AbstractCPrint
            //ClassC
            //------------------------------------------------------------------------------
            // 4.
            // Abstract Class
            Console.WriteLine("-------------------------------------");
            //AbstractD abstractD1 = new ClassD();
            ////Error! An abstract class is incomplete, so it cannot be instantiated.
            AbstractD abstractD1 = new ClassE();
             abstractD1.AbstractDPrint();
             abstractD1.Print1();
             abstractD1.Print2();
            //AbstractDPrint
            //Print1
            //Print2
            ClassE classE1 = new ClassE();
             classE1.AbstractDPrint();
             classE1.Print1();
             classE1.Print2();
            //AbstractDPrint
            //Print1
            //Print2
            Console.ReadLine();
        }
    }
}

//------------------------------------------------------------------
// 1.
```

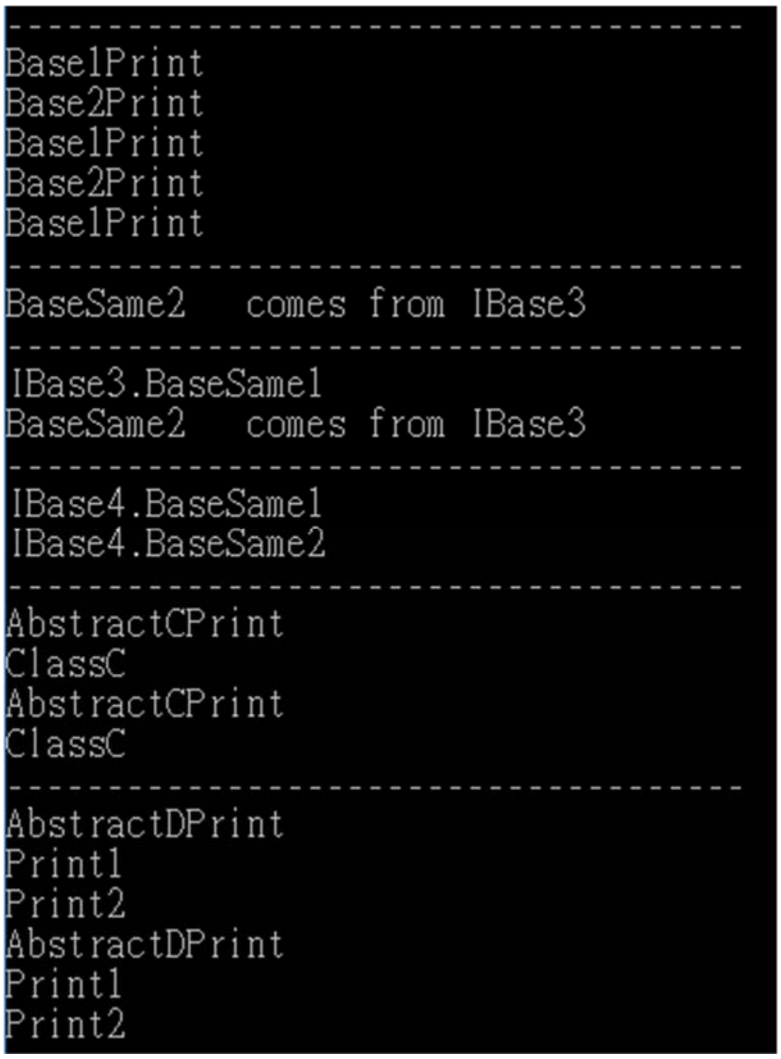
```csharp
// Interfaces can inherit from other interfaces.
public interface IBase1
{
    void Base1Print();
}
public interface IBase2 : IBase1
{
    void Base2Print();
}
public class ClassA : IBase2
{
    public void Base1Print()
    {
        Console.WriteLine("Base1Print");
    }
    public void Base2Print()
    {
        Console.WriteLine("Base2Print");
    }
}
//-----------------------------------------------------------------
// 2.
// Implement 2 Interfaces with same methods signature.
public interface IBase3
{
    void BaseSame1();
    void BaseSame2();
}
public interface IBase4
{
    void BaseSame1();
    void BaseSame2();
}
public class ClassB : IBase3, IBase4
{
    //1.
    //Accessibility Levels is invalid for explicit interface implementation.
    ////public void IBase3.BaseSame1()    //Error
    //2.
    //Explicit interface implementation fix
    //the isse of the same methods signature heads from 2 interfaces,
    //but can not be used in Class Reference Variable(LHS).
    //It can only be used in Interface Reference Variable(LHS).
    void IBase3.BaseSame1()
    {
        Console.WriteLine("IBase3.BaseSame1");
    }
    //// Accessibility Levels is invalid for explicit interface implementation.
    //public void IBase4.BaseSame1()    //Error
    void IBase4.BaseSame1()
    {
        Console.WriteLine("IBase4.BaseSame1");
    }
    // Non-explicit interface implementation can be used in Class Reference Variable(LHS).
    public void BaseSame2()
    {
        Console.WriteLine("BaseSame2   comes from IBase3");
    }
    //// Accessibility Levels is invalid for explicit interface implementation.
    //public void IBase4.BaseSame2()    //Error
    void IBase4.BaseSame2()
    {
        Console.WriteLine("IBase4.BaseSame2");
    }
}

//----------------------------------------------------------------
// 3.
// Abstract Class
public abstract class AbstractC
{
    public abstract void Print1();
    public void AbstractCPrint()
    {
        Console.WriteLine("AbstractCPrint");
    }
}
public class ClassC : AbstractC
{
    // "override" keyword to override the abstract method from base class
    public override void Print1()
    {
        Console.WriteLine("ClassC");
    }
}
//----------------------------------------------------------------
// 4.
// Abstract Class
public abstract class AbstractD
{
    public abstract void Print1();
    public abstract void Print2();
    public void AbstractDPrint()
    {
        Console.WriteLine("AbstractDPrint");
    }
}
//Abstract Class does not has to implement all Base Abstract Class members such as methods.
public abstract class ClassD : AbstractD
{
    // "override" keyword to override the abstract method from base class
    public override void Print1()
    {
        Console.WriteLine("Print1");
    }
}
public class ClassE : ClassD
{
    public override void Print2()
    {
        Console.WriteLine("Print2");
    }
}


/*
1.
Interface
```

```
1.1.
Interface is like an product booklet which contains
the standard actions that this product must be able to do.
Interface only contains the method signature without its body.
E.g.
IAircraft must be able to take off and land.
------------------------------------------
1.2.
Interface can contain declarations signature head of properties, methods, delegates, and events,
then the Class or the Struct which implement the interface, has to implement all Interface members
in order to avoid the compiler error.
In addition, Interface can not contain any field.
------------------------------------------
1.3.
Interface can not be use to create as object instance.
E.g.
//IAircraft aircraft = new IAircraft()  //Complier Error
------------------------------------------
1.4.
Interface can be used as the Reference Variable Type(LHS).
E.g.
//IBase b1 = new SubClass();
------------------------------------------
1.5.
The prefix of interface is "I".
------------------------------------------
1.6.
By default, Interface is Public, and it does not allow explicit access modifiers.
A class can extend only one class and implement many Interfaces,
and the base Class must be front of Interface.
E.g.
//public class ClassA : ClassB, InterfaceA, InterfaceB
------------------------------------------
1.7.
Interfaces can inherit from other interfaces.
E.g.
//public Interface IBase1 : IBase2
------------------------------------------
1.8.
//public void IBase3.BaseSame1()    //Error
Accessibility Levels is invalid for explicit interface implementation.
Explicit interface implementation fix
the isse of the same methods signature heads from 2 interfaces.
Explicit interface implementation can not be used in Class Reference Variable(LHS).
It can only be used in Interface Reference Variable(LHS).
Non-explicit interface implementation can be used in Class Reference Variable(LHS).
-------------------------------------------------------------------
2.
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.
-------------------------------------------------------------------
3.
Abstract Class V.S. Interface
------------------------------------------
3.1.
An abstract class is incomplete, so it cannot be instantiated.
In addition, an Abstract Class can not be a Sub-Class of other Class.
Abstract Class can only be a Base-Class,
so it must not be Sealed type.
------------------------------------------
3.2.
```

```
Interface:
3.2.1.
An Abstract Class is very similar to Interface.
Interface can contain declarations signature head of properties, methods, delegates, and events,
then the Class or the Struct which implement the interface, must implement all Interface members
in order to avoid the compiler error.
3.2.2.
Interface can not have implementation for any of its members.
3.2.3.
Interface can not have fields.
3.2.4.
Interface members can not have access modifier such as public, private or protected.
------------------------------------------
3.3.
Abstract Class:
3.3.1.
An Abstract Class may or may not contain abstract members
which includes methods, properties, indexers, and events,
then its Sub-Class may implement some of or all abstract members
in order to avoid the compiler error.
Non-abstract Sub-Class of the Abstract Class must implement all inherited abstract members.
Abstract Sub-Class of the Abstract Class may just implement some inherited abstract members.
3.3.2.
Abstract Class does not has to implement all Base Abstract Class members such as methods.
3.3.3.
Abstract Class can have fields.
3.3.4.
Abstract class members can have access modifier such as public, private or protected.
------------------------------------------
3.4.
One non-Abstract Class can extend one non-Abstract Class or Abstract Class
and implement many Interfaces.
The base case class must be in front of Interfaces.
E.g. public class ClassA : ClassB, InterfaceA, InterfaceB
------------------------------------------
3.5.
Abstract Class can extend anther Abstract Class
and implement many Interfaces.
The base case class must be in front of Interfaces.
E.g. public class abstract ClassA : ClassB, InterfaceA, InterfaceB
------------------------------------------
3.6.
Interface can implement many other Interfaces
but can not extend an Abstract Class.
E.g. public Interface Interface01 : InterfaceA, InterfaceB
*/
```



```
-------------------------------------
Base1Print
Base2Print
Base1Print
Base2Print
Base1Print
-------------------------------------
BaseSame2   comes from IBase3
-------------------------------------
IBase3.BaseSame1
BaseSame2   comes from IBase3
-------------------------------------
IBase4.BaseSame1
IBase4.BaseSame2
-------------------------------------
AbstractCPrint
ClassC
AbstractCPrint
ClassC
-------------------------------------
AbstractDPrint
Print1
Print2
AbstractDPrint
Print1
Print2
```