(T6)比較 Public、Protected、Private。比較 Delegate、MulticastDelegates
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==========
(T6)比較 Public、Protected、Private。比較 Delegate、MulticastDelegates
==========

==========

0. Summary

1.
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.

2.
Delegate

2.1.
Syntax:
//[AccessibilityLevels] delegate [void|ReturnType] DelegateName ([ParametersList...]);
E.g.
//delegate void MessageDelegate(string str);
The syntax is similar to method signature head.
Just like adding "delegate" keyword in front of Method.
2.1.1.
Accessibility Levels can be public, protected, private, or ...etc.
For more details, please visit
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels
2.1.2.
delegate is the keyword.
ReturnType can be string, int, or .... etc.
Void means return nothing.
DelegateName is the name of this delegate.
ParametersList here is optional and it is the list of parameters.

2.2.
Delegate is a function pointer which stores reference to a function.
The signature of the delegate must match the signature of the function.
Using Delegate is similar to use Class,
the Delegate instance must be initiated before using it.
The target function name must be passed as a parameter to the delegate constructor.
Invoke the delegate, which will invoke the target function.

2.3.
Multicast Delegates

2.3.1.
Multicast Delegates means a delegate has references to many functions.
When the Delegates is invoked,
all functions will be invoked in the same order

in which they are added to invocation list.
"+" or "+=" to register a method with the delegate.
"-" or "-=" to un-registera method with the delegate.

2.3.2.
If the Multicast Delegate has return type,
then only the last invoked method will be return.

2.3.3.
If the Multicast Delegate has output parameter,
then only the last invoked method will assign the value of output parameter.

2.3.4.
Multicast Delegate allows other programmers to add some extra logic without changing the original code.
This can make the publish/subscribe pattern such as your customized SDK become more flexible.

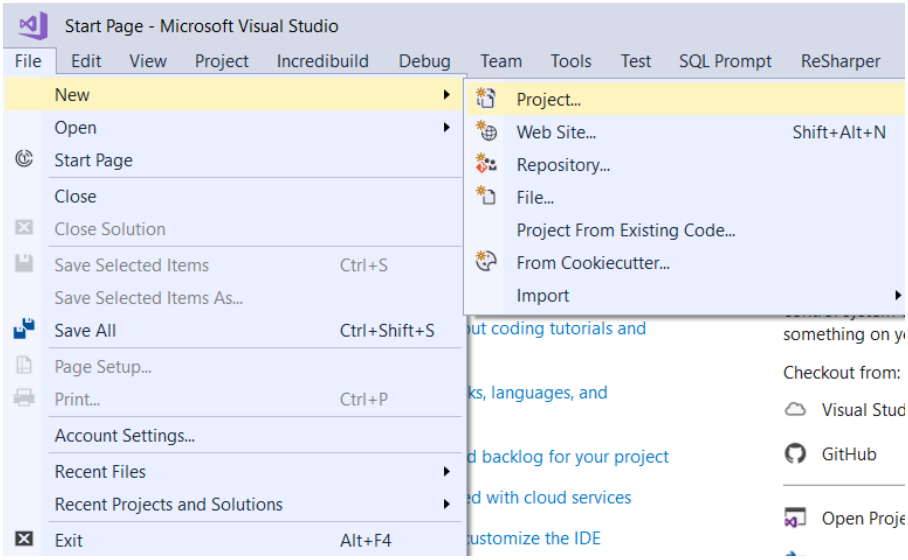
==========

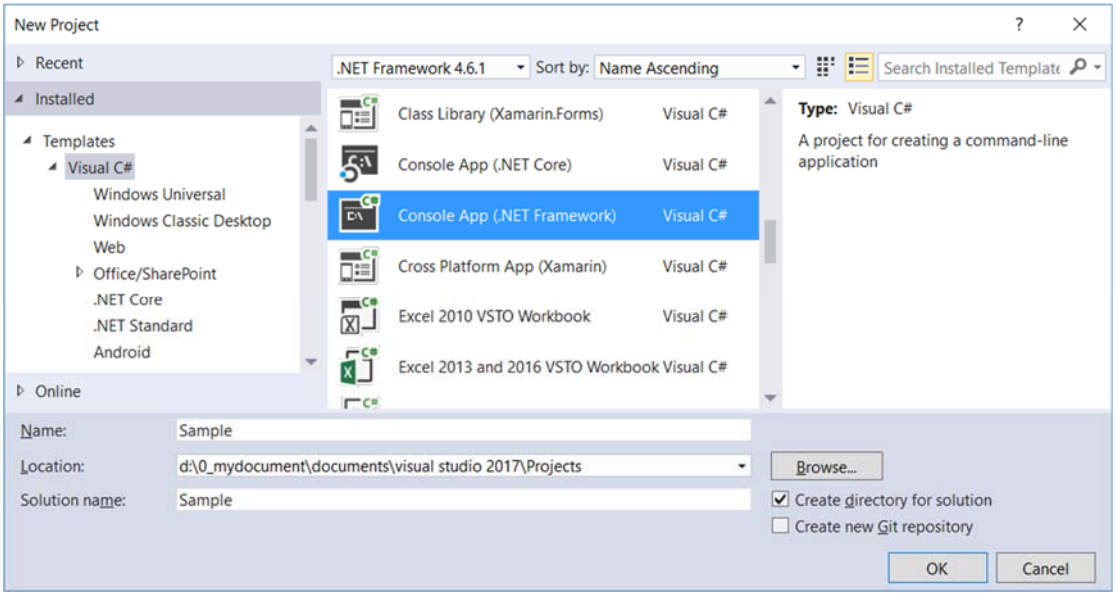
1. Create New Project

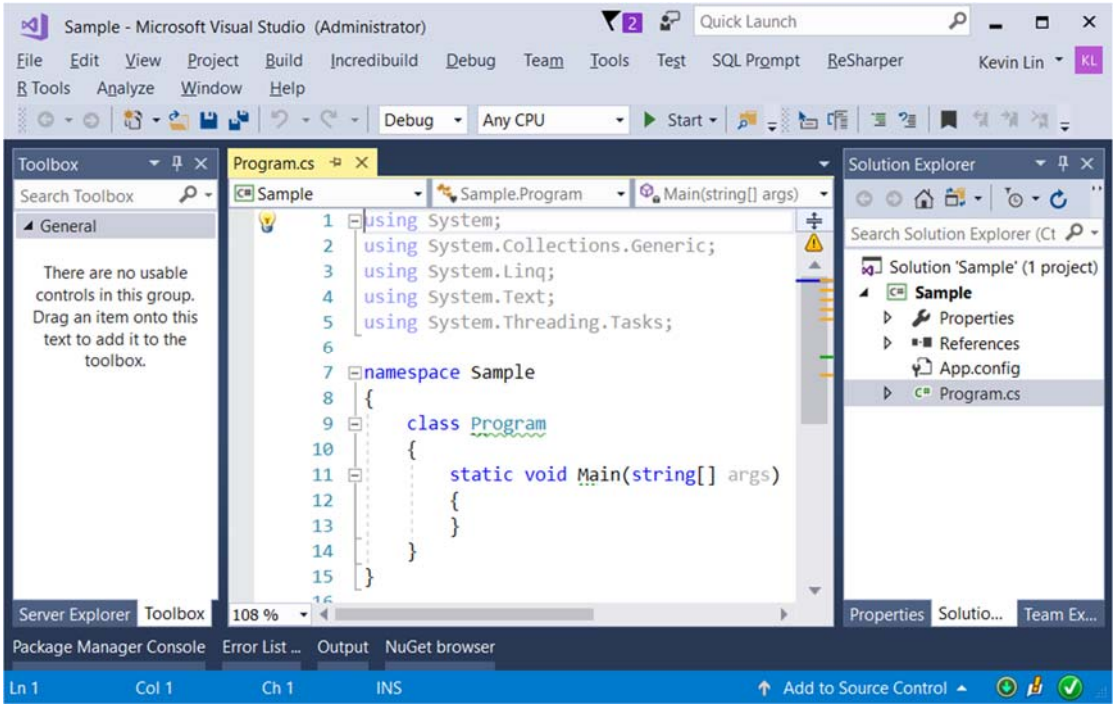
File --> New --> Project... -->

Visual C# --> **Console App (.Net Framework)** -->

Name: **Sample**







==========

2. Program

```
using System;
using System.Collections.Generic;
using OnlineGame;
namespace Sample
{
    // 1. -------------------------------------------------
```

```
// Delegate 1st Sample
// 1.
//Delegate
//1.1.
//Syntax:
////[AccessibilityLevels] delegate [void|ReturnType] DelegateName ([ParametersList...]);
//E.g.
////delegate void MessageDelegate(string str);
//The syntax is similar to method signature head.
//Just like adding "delegate" keyword in front of Method.
//1.1.1.
//Accessibility Levels can be public, protected, private, or...etc.
//For more details, please visit
//https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels
//1.1.2.
//delegate is the keyword.
//ReturnType can be string, int, or....etc.
//Void means return nothing.
//DelegateName is the name of this delegate.
//ParametersList here is optional and it is the list of parameters.
//1.2.
//Delegate is a function pointer which stores reference to a function.
//The signature of the delegate must match the signature of the function.
//Using Delegate is similar to use Class,
//the Delegate instance must be initiated before using it.
//The target function name must be passed as a parameter to the delegate constructor.
//Invoke the delegate, which will invoke the target function.
// Delegate Declaration.
public delegate void Delegate1(string str);
// 5. ------------------------------------------------
public delegate void MulticastVoidDelegate();
// 6. ------------------------------------------------
public delegate int IntDelegate();
// 7. ------------------------------------------------
public delegate void outIntDelegate(out int i);

class Program
 {
     static void Main(string[] args)
     {
         // 1. ------------------------------------------------
         // Delegate 1st Sample
         Console.WriteLine("1 ================================================");
         Program.Print("Program.Print");
         //Delegate is a function pointer which stores reference to a function.
         //The signature of the delegate must match the signature of the function.
         //Using Delegate is similar to use Class,
         //the Delegate instance must be initiated before using it.
         //The target function name must be passed as a parameter to the delegate constructor.
         //Invoke the delegate, which will invoke the target function.
         Delegate1 del1 = new Delegate1(Print);
          del1(@"Delegate1 del1 = new Delegate1(Print);  del1(...);");
         // 2. ------------------------------------------------
```

```
            //Delegate allows other programmers
            //to add some extra logic without changing the original code.
            Console.WriteLine("2 ================================================");
            List<Gamer> gamerList = new List<Gamer>();
             gamerList.Add(new Gamer (1, "Name01"));
            //Using   public Gamer(int id, string name, int hp=1, int mp=1, int gameScore=0, int
experience=0, int level=1)
             gamerList.Add(new Gamer { Id = 2, Name = "Name02", Level = 20, Experience = 5000 });
            // using  public Gamer()
             gamerList.Add(new Gamer { Id = 3, Name = "Name03", Level = 14, Experience = 4500 });
             gamerList.Add(new Gamer { Id = 4, Name = "Name04", Level = 19, Experience = 6000 });
             gamerList.Add(new Gamer { Id = 5, Name = "Name05", Level = 32, Experience = 8000 });
            foreach (Gamer gamer in gamerList)
            {
                gamer.toString();
            }
            Console.WriteLine("Gamer.LevelUp1(gamerList); ----------------------------------------");
            Gamer.LevelUp1(gamerList);
            foreach (Gamer gamer in gamerList)
            {
                gamer.toString();
            }
            // 3. ------------------------------------------------
            //Delegate allows other programmers
            //to add some extra logic without changing the original code.
            Console.WriteLine("3 ================================================");
            IsLevelUp isLvUp = new IsLevelUp(CanLevelUp);
            DoingLevelUp doingLvUp = new DoingLevelUp(LevelIncreased);
            Gamer.LevelUp(gamerList, isLvUp, doingLvUp);
            foreach (Gamer gamer in gamerList)
            {
                gamer.toString();
            }
            // 4. ------------------------------------------------
            //Using lambda expression if the delegate function only return one line.
            Console.WriteLine("4 ================================================");
            Gamer.LevelUp(gamerList, gamer => gamer.Experience >= 1000 && gamer.Hp >= 1, gamer =>
gamer.Level++);
            foreach (Gamer gamer in gamerList)
            {
                gamer.toString();
            }
            // 5. ------------------------------------------------
            //Multicast Void Delegates
            //Multicast Delegates means a delegate has references to many functions.
            //When the Delegates is invoked,
            //all functions will be invoked in the same order
            //in which they are added to invocation list.
            //"+" or "+=" to register a method with the delegate.
            //"-" or "-=" to un-registera method with the delegate.
            Console.WriteLine("5 ================================================");
            //5.1. ---------------------------------------
            Console.WriteLine("5.1. ---------------------------------------");
            MulticastVoidDelegate mvd1 = new MulticastVoidDelegate(VoidMethod1);
            MulticastVoidDelegate mvd2 = VoidMethod2;
            MulticastVoidDelegate mvd3 = VoidMethod3;
```

```
        MulticastVoidDelegate mvd4 = mvd1 + mvd2 + mvd3 - mvd2;
        mvd4();
        //5.2. ---------------------------------------
        Console.WriteLine("5.2. ---------------------------------------");
        MulticastVoidDelegate mvd = new MulticastVoidDelegate(VoidMethod1);
        mvd += VoidMethod2;
        mvd += VoidMethod3;
        mvd -= VoidMethod2;
        mvd();
        // 6. ------------------------------------------------
        //Multicast Delegate with return type
        //If the Multicast Delegate has return type,
        //then only the last invoked method will be return.
        Console.WriteLine("6 ================================================");
        IntDelegate iDel = new IntDelegate(intMethod1);
        iDel += intMethod2;
        iDel += intMethod3;
        int i1 = iDel();
        Console.WriteLine("Multicast Delegate with return type, i1 = {0}", i1);
        // 7. ------------------------------------------------
        //Multicast Delegate with output parameter
        //If the Multicast Delegate has output parameter,
        //then only the last invoked method will assign the value of output parameter.
        Console.WriteLine("7 ================================================");
        outIntDelegate outIntDel = new outIntDelegate(outMethod1);
        outIntDel += outMethod2;
        outIntDel += outMethod3;
        int i2;
        outIntDel(out i2);
        Console.WriteLine("Multicast Delegate with output parameter, i2 = {0}", i2);
        Console.ReadLine();
    }

    // 1. ------------------------------------------------
    // Delegate 1st Sample
    public static void Print(string strMsg)
    {
        Console.WriteLine(strMsg);
    }
    // 3. ------------------------------------------------
    public static bool CanLevelUp(Gamer gamer)
    {
        // add some extra condtion to level up.
        return gamer.Experience >= 1000 && gamer.Hp >=1;
    }
    public static void LevelIncreased(Gamer gamer)
    {
        gamer.Level++;
        gamer.Experience -= 1000;
        gamer.Hp += 100;    // some extra reward
        gamer.Mp += 100;    // some extra reward
    }
    // 5. ------------------------------------------------
    //Multicast Void Delegates
    public static void VoidMethod1()
    {
        Console.WriteLine("VoidMethod1");
```

```
        }
        public static void VoidMethod2()
        {
            Console.WriteLine("VoidMethod2");
        }
        public static void VoidMethod3()
        {
            Console.WriteLine("VoidMethod3");
        }

       // 6. ------------------------------------------------
        //Multicast Delegate with return type
        public static int intMethod1()
        {
            return 1;
        }
        public static int intMethod2()
        {
            return 2;
        }
        public static int intMethod3()
        {
            return 3;
        }
        // 7. ------------------------------------------------
        //Multicast Delegate with output parameter
        public static void outMethod1(out int i)
        {
            i = 1;
        }
        public static void outMethod2(out int i)
        {
            i = 2;
        }
        public static void outMethod3(out int i)
        {
            i = 3;
        }
    }
}


// 2. ------------------------------------------------
//Delegate allows other programmers
//to add some extra logic without changing the original code.
namespace OnlineGame
{
    // 3. ------------------------------------------------
    // Delegate 1st Sample
    //Syntax:
    ////[AccessibilityLevels] delegate [void|ReturnType] DelegateName ([ParametersList...]);
    //E.g.
    ////delegate void MessageDelegate(string str);
    //The syntax is similar to method signature head.
    //Just like adding "delegate" keyword in front of Method.
    public delegate bool IsLevelUp(Gamer gamer);
    public delegate void DoingLevelUp(Gamer gamer);
    // 2. ------------------------------------------------
```

```
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Hp { get; set; }
        public int Mp { get; set; }
        public int GameScore { get; set; }
        public int Experience { get; set; }
        public int Level { get; set; }
        ////public Gamer(int id, string name, int hp=1, int mp=1, int gameScore=0, int experience=0, int
level=1)
        //Except id and name, rest of parametes has default value.
        //when the parameter has the default value,
        //it make the parameter the become optional.
        public Gamer(int id, string name, int hp=1, int mp=1, int gameScore=0, int experience=0, int level=1)
        {
            Id = id;
            Name = name;
            Hp = hp;
            Mp = mp;
            GameScore = gameScore;
            Experience = experience;
            Level = level;
        }
        ////List<Gamer> gamerList = new List<Gamer>();
        ////gamerList.Add(new Gamer{Id=1, Name = "Name01"});
        // need the parameterless constructor.
        // otherwise return compile error.
        public Gamer()
        {
            Id = 0;
            Name = "";
            Hp = 1;
            Mp = 1;
            GameScore = 0;
            Experience = 0;
            Level = 1;
        }
        public void toString()
        {
            Console.WriteLine("id={0}, name={1}, hp={2}, mp={3}, gameScore={4}, experience={5},
level={6}", Id, Name, Hp, Mp, GameScore, Experience, Level);
        }
        //Level up when Experience >= 1000
        public static void LevelUp1(List<Gamer> gamerList)
        {
            foreach (Gamer gamer in gamerList)
            {
                if (gamer.Experience >= 1000)
                {
                    gamer.Level++;
                    gamer.Experience -= 1000;
                }
            }
        }
        // 3. ------------------------------------------------
        public static void LevelUp(List<Gamer> gamerList, IsLevelUp isLevelUp, DoingLevelUp doingLevelUp)
        {
            foreach (Gamer gamer in gamerList)
            {
```

```
                if (isLevelUp(gamer))
                {
                    doingLevelUp(gamer);
                }
            }
        }
    }
}



/*
1.
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.
-----------------------------------------------------------------------------------------
2.
Delegate
-----------------------------------------------------------------------------
2.1.
Syntax:
//[AccessibilityLevels] delegate [void|ReturnType] DelegateName ([ParametersList...]);
E.g.
//delegate void MessageDelegate(string str);
The syntax is similar to method signature head.
Just like adding "delegate" keyword in front of Method.
2.1.1.
Accessibility Levels can be public, protected, private, or ...etc.
For more details, please visit
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels
2.1.2.
delegate is the keyword.
ReturnType can be string, int, or .... etc.
Void means return nothing.
DelegateName is the name of this delegate.
ParametersList here is optional and it is the list of parameters.
-----------------------------------------------------------------------------
2.2.
Delegate is a function pointer which stores reference to a function.
The signature of the delegate must match the signature of the function.
Using Delegate is similar to use Class,
the Delegate instance must be initiated before using it.
The target function name must be passed as a parameter to the delegate constructor.
Invoke the delegate, which will invoke the target function.
-----------------------------------------------------------------------------
2.3.
Multicast Delegates
-----------------------------------------------------
2.3.1.
Multicast Delegates means a delegate has references to many functions.
When the Delegates is invoked,
all functions will be invoked in the same order
in which they are added to invocation list.
"+" or "+=" to register a method with the delegate.
"-" or "-=" to un-registera method with the delegate.
-----------------------------------------------------
2.3.2.
If the Multicast Delegate has return type,
then only the last invoked method will be return.
-----------------------------------------------------
2.3.3.
```

```
If the Multicast Delegate has output parameter,
then only the last invoked method will assign the value of output parameter.
-----------------------------------------------------
2.3.4.
Multicast Delegate allows other programmers to add some extra logic without changing the original code.
This can make the publish/subscribe pattern such as your customized SDK become more flexible.

 */
```

```
1 ================================================
Program.Print
Delegate1 del1 = new Delegate1(Print);  del1(...);
2 ================================================
id=1, name=Name01, hp=1, mp=1, gameScore=0, experience=0, level=1
id=2, name=Name02, hp=1, mp=1, gameScore=0, experience=5000, level=20
id=3, name=Name03, hp=1, mp=1, gameScore=0, experience=4500, level=14
id=4, name=Name04, hp=1, mp=1, gameScore=0, experience=6000, level=19
id=5, name=Name05, hp=1, mp=1, gameScore=0, experience=8000, level=32
Gamer.LevelUp1(gamerList); ----------------------------------------
id=1, name=Name01, hp=1, mp=1, gameScore=0, experience=0, level=1
id=2, name=Name02, hp=1, mp=1, gameScore=0, experience=4000, level=21
id=3, name=Name03, hp=1, mp=1, gameScore=0, experience=3500, level=15
id=4, name=Name04, hp=1, mp=1, gameScore=0, experience=5000, level=20
id=5, name=Name05, hp=1, mp=1, gameScore=0, experience=7000, level=33
3 ================================================
id=1, name=Name01, hp=1, mp=1, gameScore=0, experience=0, level=1
id=2, name=Name02, hp=101, mp=101, gameScore=0, experience=3000, level=22
id=3, name=Name03, hp=101, mp=101, gameScore=0, experience=2500, level=16
id=4, name=Name04, hp=101, mp=101, gameScore=0, experience=4000, level=21
id=5, name=Name05, hp=101, mp=101, gameScore=0, experience=6000, level=34
4 ================================================
id=1, name=Name01, hp=1, mp=1, gameScore=0, experience=0, level=1
id=2, name=Name02, hp=101, mp=101, gameScore=0, experience=3000, level=23
id=3, name=Name03, hp=101, mp=101, gameScore=0, experience=2500, level=17
id=4, name=Name04, hp=101, mp=101, gameScore=0, experience=4000, level=22
id=5, name=Name05, hp=101, mp=101, gameScore=0, experience=6000, level=35
5 ================================================
5.1. ------------------------------------
VoidMethod1
VoidMethod3
5.2. ------------------------------------
VoidMethod1
VoidMethod3
6 ================================================
Multicast Delegate with return type, i1 = 3
7 ================================================
Multicast Delegate with output parameter, i2 = 3
```