(T14)Reflection(反射)、讀取 DLL。比較 EarlyBinding、LateBinding
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==
(T14)Reflection(反射)、讀取 DLL。比較 EarlyBinding、LateBinding
==

==

0. Summary

1.
Reflection
1.1.
Reflection can find Types in an assembly by giving the string value of Type Name.
In addition, Reflection can use the Type to dynamically
create an object instance of a Type by late binding at run time.
Furthermore, Reflection can dynamically
invoke its methods or access its fields and properties.
1.2.
E.g.
The popular way of using Reflection is to dynamically load DLLs from XML file.
For example, Create several DLLs into a folder.
Write a XML to contain the DLLs Name which you want to load.
Using string value of DLLs Name and using Reflection to
load dynamically DLLs into your project.
Reflection will allows users to dynamically create object instance of the Type from DLLs.
It also allow users to dynamically invoke its methods or access its fields and properties.
This will not cover in this tutorial.

2.
Early binding V.S. Late binding:
2.1.
Early binding is better for performance and can flag errors at compile time.
2.2.
Late binding performance is worse than Early binding.
In addition, Late binding has a risk of run time exceptions
if the string value of Type Name or Method name is incorrect.
However, Late binding is good when working with onjects
that are not available at compile time.
E.g.
The popular way of using Reflection is to dynamically load DLLs from XML file.
2.3.
Load DLLs

Reference:
https://stackoverflow.com/questions/18483354/get-assembly-of-program-from-a-dll
https://stackoverflow.com/questions/43318419/get-dll-file-extension-by-system-reflection-assembly

==

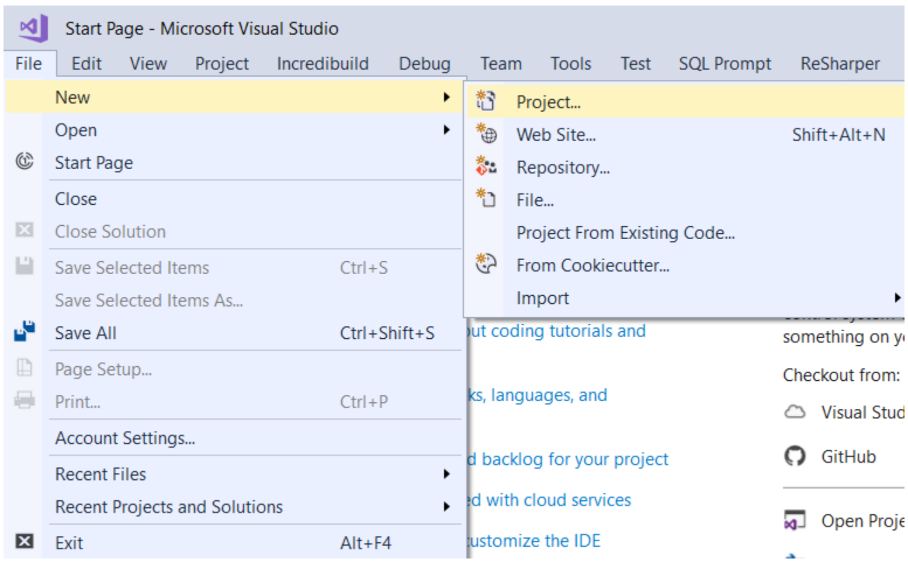
1. New Project

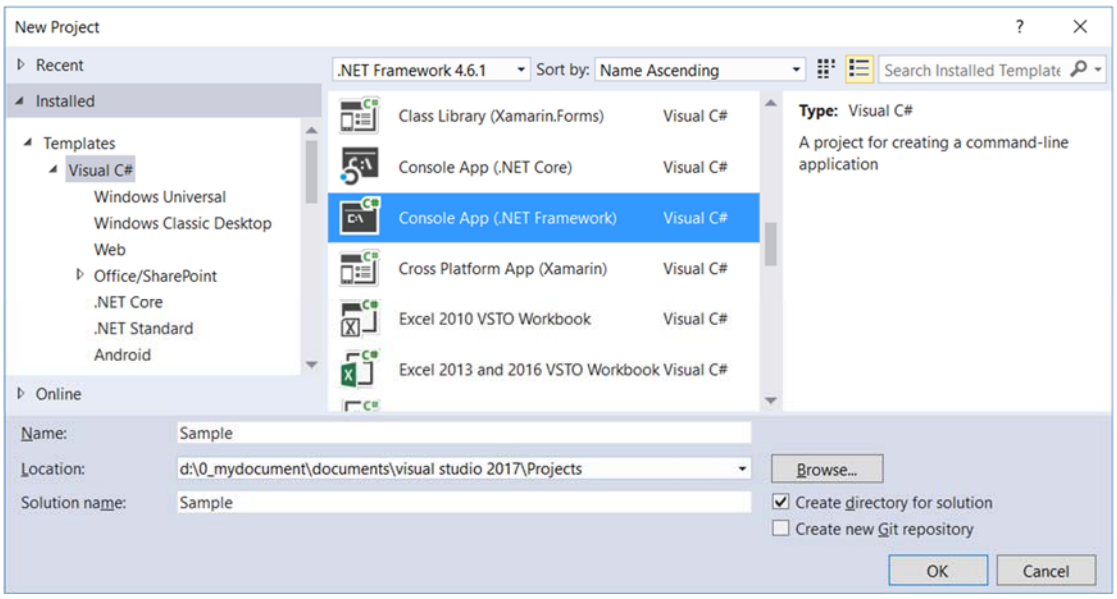
1.1. Create New Project

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

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

Name: **Sample**


Start Page - Microsoft Visual Studio
File Edit View Project Incredibuild Debug Team Tools Test SQL Prompt ReSharper
New
Open
Start Page
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Page Setup...
Print... Ctrl+P
Account Settings...
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project...
Web Site... Shift+Alt+N
Repository...
File...
Project From Existing Code...
From Cookiecutter...
Import


New Project
Recent
Installed
Templates
Visual C#
Windows Universal
Windows Classic Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Online
.NET Framework 4.6.1
Sort by: Name Ascending
Search Installed Templates
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#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

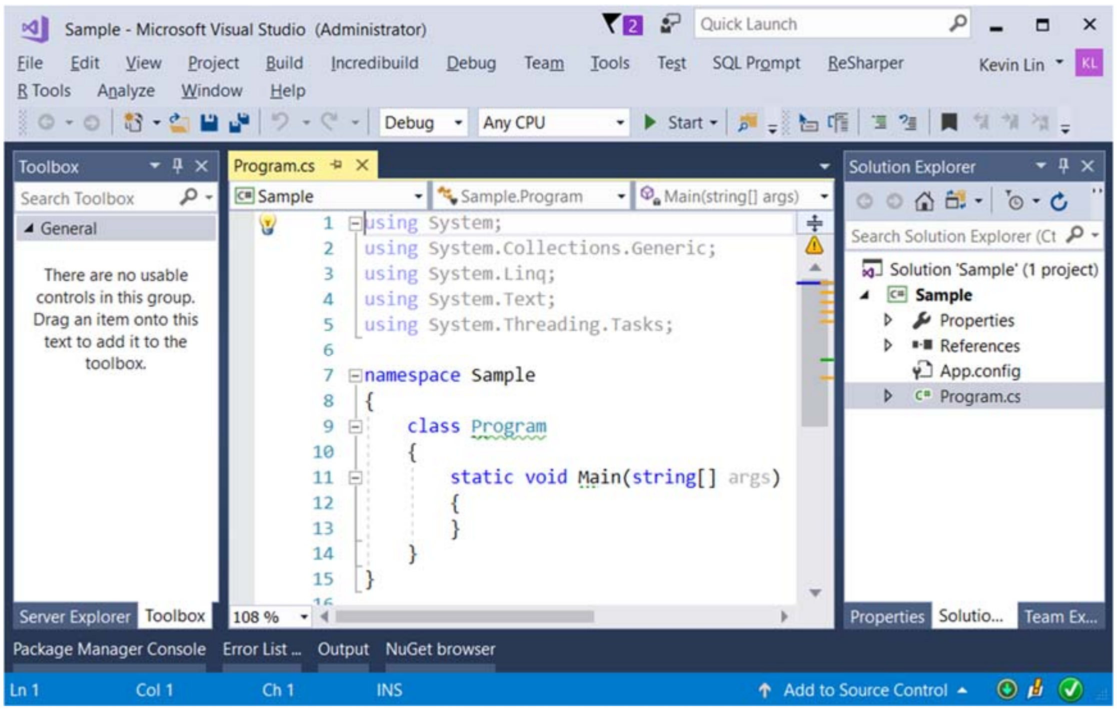
Name: Sample

Location: d:\0_mydocument\documents\visual studio 2017\Projects

Solution name: Sample

[✓] Create directory for solution

[ ] Create new Git repository

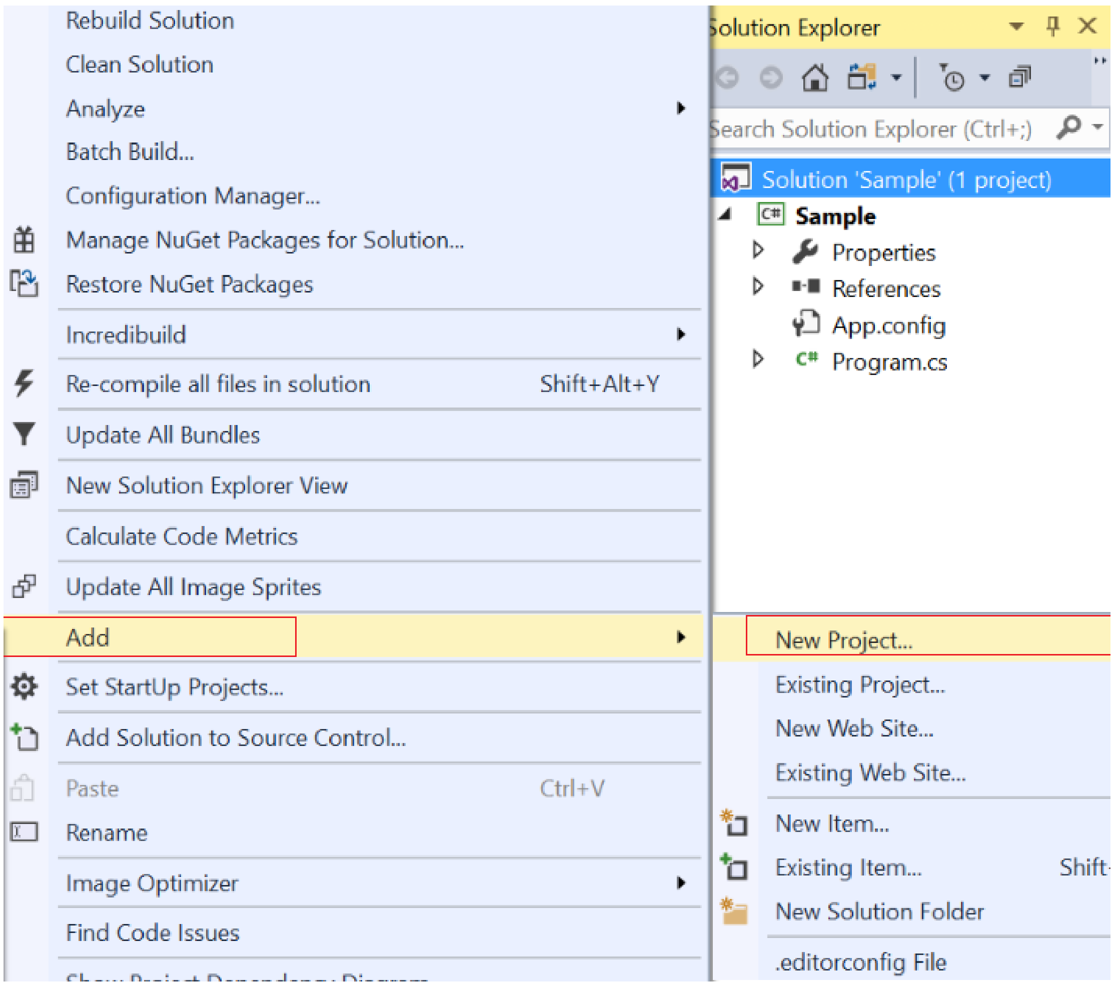


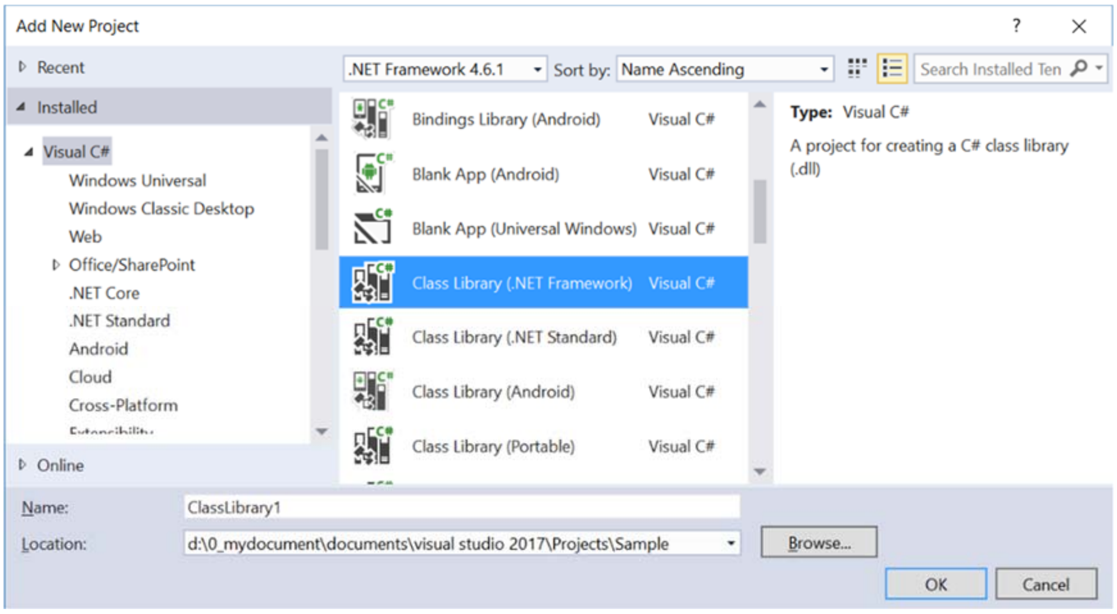
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
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)
        {
        }
    }
}
```

1.2. Add New Project

Solution Name --> Right Click --> Add --> New Project --> Class Library (.Net Framework)
-->
Project Name :
ClassLibrary1




Add New Project
Recent
Installed
Visual C#
Windows Universal
Windows Classic Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Cloud
Cross-Platform
Online
Bindings Library (Android) Visual C#
Blank App (Android) Visual C#
Blank App (Universal Windows) Visual C#
Class Library (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (Android) Visual C#
Class Library (Portable) Visual C#
Search Installed Ten
Browse...
OK
Cancel

.NET Framework 4.6.1 Sort by: Name Ascending

Type: Visual C#

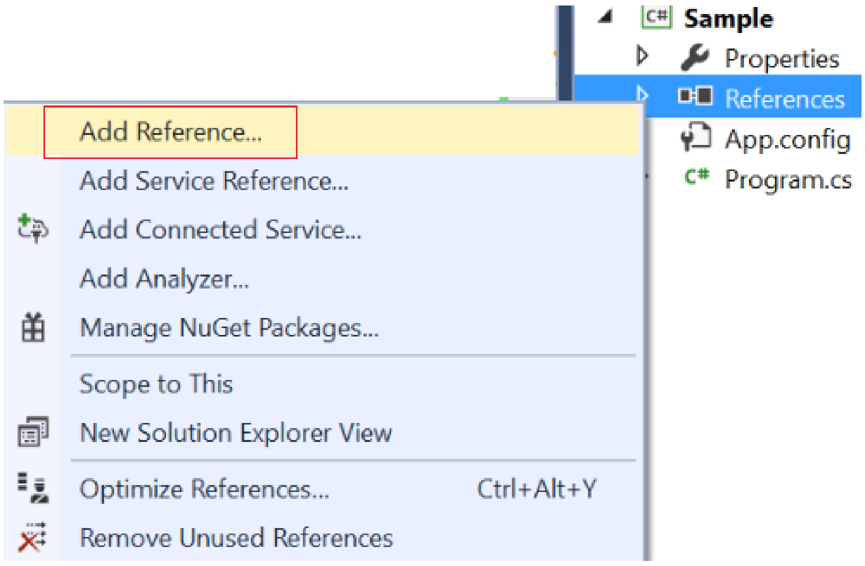
A project for creating a C# class library (.dll)

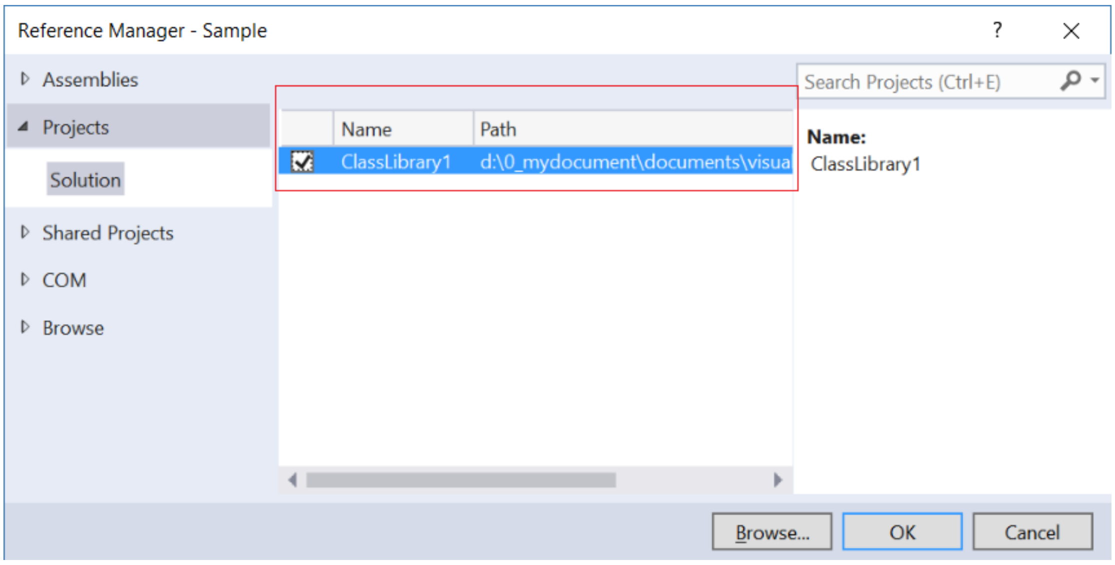
Name: ClassLibrary1

Location: d:\0_mydocument\documents\visual studio 2017\Projects\Sample

1.3. Add Reference

Project Name --> References --> Add Reference
--> Select the reference you want to add.





	Name	Path
✓	ClassLibrary1	d:\0_mydocument\documents\visua

Name:
ClassLibrary1

===

2. Code

2.1. ClassLibrary1/GamerA.cs

```
using System;
namespace OnLineGameA
{
    // This class has 2 constructors, 2 properties, and 4 methods
    public class GamerA
     {
        // Properties ---------------------------
        public string Name { get; set; }
        public int GameScore { get; set; }
        // Constructor ---------------------------
        public GamerA(string name, int gameScore)
         {
             GameScore = gameScore;
```

```
            Name = name;
        }
        public GamerA()
        {
            GameScore = -1;
            Name = string.Empty;
        }
        // Methods ----------------------------
        public void PrintName()
        {
            Console.WriteLine($"Name == {Name}");
        }
        public void PrintGameScore()
        {
            Console.WriteLine($"GameScore == {GameScore}");
        }
        public override string ToString()
        {
            return $"Name : {Name} ; GameScore : {GameScore}";
        }
        public string SetNameAndGameScore(string name, int gameScore)
        {
            Name = name;
            GameScore = gameScore;
            return ToString();
        }
    }
}
```

2.2. Sample/Program.cs

```
using System;
using System.Reflection;
using OnLineGame;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            //1-4 --------------------------------------------
            Console.WriteLine("1-4. ReflectionBasic(); ===========================");
            ReflectionBasic();
            //5. --------------------------------------------
            Console.WriteLine("5. EarlyBinding(); ===========================");
            EarlyBinding();
            //6. --------------------------------------------
            Console.WriteLine("6. LateBinding(); ===========================");
            LateBinding();
            //7. --------------------------------------------
            Console.WriteLine("7. LateBinding2(); ===========================");
            LateBinding2();
            //8. --------------------------------------------
            Console.WriteLine("8. LateBinding3(); ===========================");
            LateBinding3();
            Console.ReadLine();
        }
```

```
//1-4 --------------------------------------------
static void ReflectionBasic()
{
    // 0. -----------------------------------------------
    // There are 3 ways to get Type
    // 0.1. ---------------------------
    // Type.GetType(string TypeFullName) is a static method
    // The parameter is string value of TypeFullName
    // which is "NameSpace.ClassName".
    // It will return the Type of TypeFullName.
    Type t1 = Type.GetType("OnLineGame.Gamer");
    // 0.2. ---------------------------
    // typeof keyword parameter is ClassName,
    // and it need   using ItsNameSpace
    // E.g. using OnLineGame;
    Type t2 = typeof(Gamer);
    // 0.2. ---------------------------
    // object.GetType() can get the Type of object.
    Gamer g1 = new Gamer();
    Type t3 = g1.GetType();
    if (t1 != null)
    {
        // 1. -----------------------------------------------
        Console.WriteLine("1. Type details -------------------------------------");
        Console.WriteLine($"t1.FullName == {t1.FullName} " +
                          $"which is NameSpace.ClassName");
        Console.WriteLine($"t1.Name == {t1.Name} which is ClassName");
        Console.WriteLine($"t1.Namespace = {t1.Namespace} which is NameSpace");
        //1.Type details------------------------------------ -
        //t1.FullName == OnLineGame.Gamer which is NameSpace.ClassName
        //t1.Name == Gamer which is ClassName
        //t1.Namespace = OnLineGame which is NameSpace
        // 2. -----------------------------------------------
        Console.WriteLine("2. Methods -------------------------------------");
        Console.WriteLine($"Methods in {t1.FullName} Class");
        MethodInfo[] methods = t1.GetMethods();
        foreach (MethodInfo method in methods)
        {
            // MethodReturnType + MethodName
            Console.WriteLine($"{method.ReturnType.Name} {method.Name}");
        }
        //2.Methods------------------------------------ -
        //Methods in OnLineGame.Gamer Class
        //String get_Name
        //Void set_Name
        //Int32 get_GameScore
        //Void set_GameScore
        //Void PrintName
        //Void PrintGameScore
        //String ToString
        //String SetNameAndGameScore
        //Boolean Equals
        //Int32 GetHashCode
```

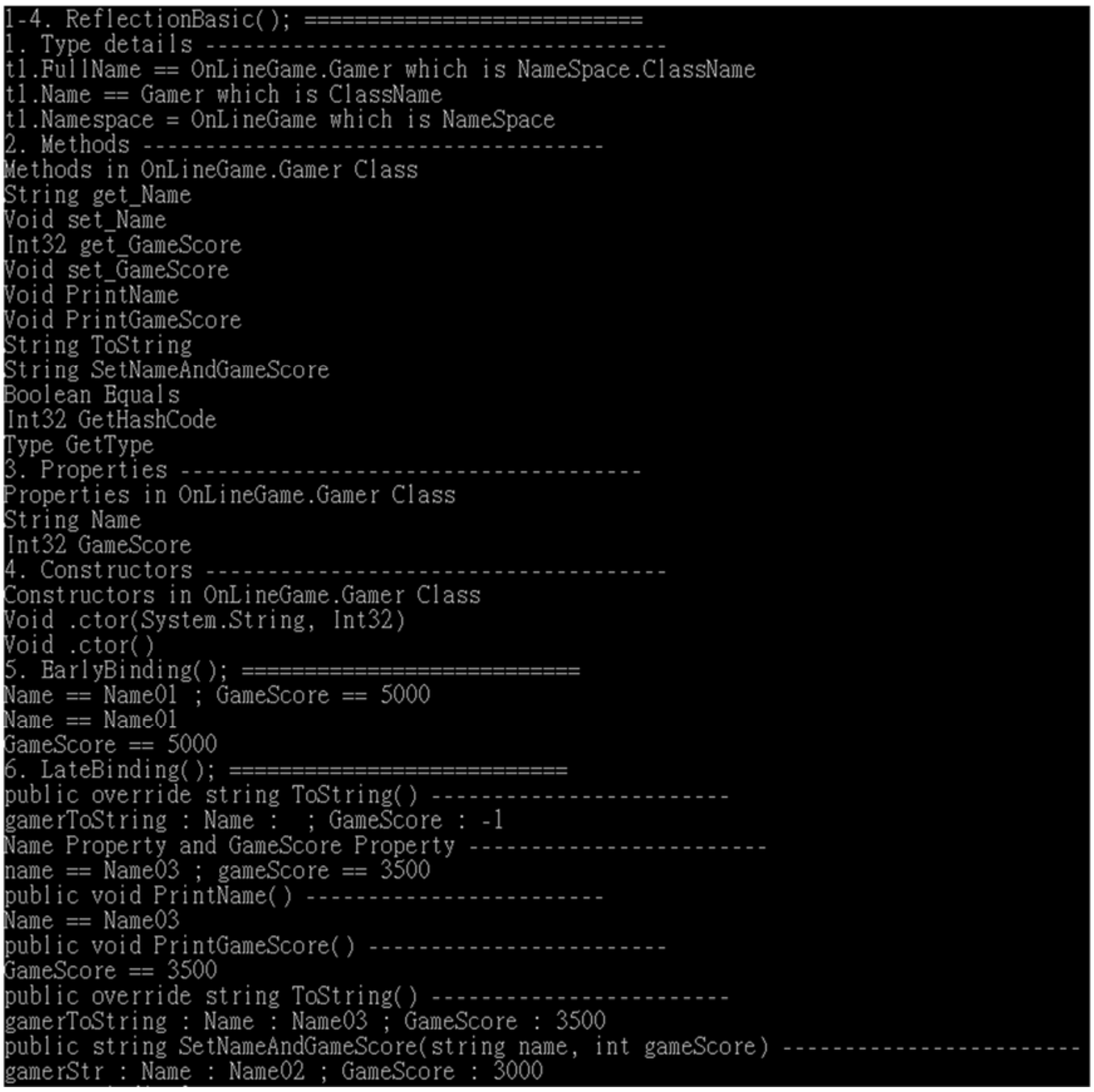
```csharp
            //Type GetType
            // Each Property will automaticly generate SetMethod and GetMethod.
            // ToString(), Equals(), GetHashCode(), GetType()
            // is inherited from System.Object.
            // 3. -----------------------------------------------
            Console.WriteLine("3. Properties --------------------------------------");
            Console.WriteLine($"Properties in {t1.FullName} Class");
            PropertyInfo[] properties = t1.GetProperties();
            foreach (PropertyInfo property in properties)
            {
                // PropertyType + PropertyName
                Console.WriteLine(property.PropertyType.Name + " " + property.Name);
            }
            //3.Properties------------------------------------ -
            //Properties in OnLineGame.Gamer Class
            //String Name
            //Int32 GameScore
            // 4. -----------------------------------------------
            Console.WriteLine("4. Constructors --------------------------------------");
            Console.WriteLine($"Constructors in {t1.FullName} Class");
            ConstructorInfo[] constructors = t1.GetConstructors();
            foreach (ConstructorInfo constructor in constructors)
            {
                Console.WriteLine(constructor.ToString());
            }
            //4.Constructors------------------------------------ -
            //Constructors in OnLineGame.Gamer Class
            //Void.ctor(System.String, Int32)
            //Void.ctor()
        }
    }


   //5. -----------------------------------------------
    static void EarlyBinding()
    {
        Gamer g1 = new Gamer();
        g1.Name = "Name01";
        g1.GameScore = 5000;
        Console.WriteLine($"Name == {g1.Name} ; GameScore == {g1.GameScore}");
        g1.PrintName();
        g1.PrintGameScore();
        //Name == Name01; GameScore == 5000
        //Name == Name01
        //GameScore == 5000
    }


    //6. -----------------------------------------------
    static void LateBinding()
    {
        // 6.1.
        // Load the current executing assembly (DLL or Exe)
        // Because Gamer class is in current assembly.
        Assembly executingAssembly = Assembly.GetExecutingAssembly();
```

```
            // 6.2.
            // Type.GetType(string TypeFullName) is a static method
            // The parameter is string value of TypeFullName
            // which is "NameSpace.ClassName".
            // It will return the Type of TypeFullName.
            // Type can be used to create an instance object dynamically.
            Type gamerType = executingAssembly.GetType("OnLineGame.Gamer");
            // 6.3.
            // Activator.CreateInstance(Type)
            // creates the instance of the Type,OnLineGame.Gamer.
            object gamerInstance = Activator.CreateInstance(gamerType);
            // 6.4.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public override string ToString() ------------------------");
            MethodInfo toStringMethodInfo = gamerType.GetMethod("ToString");
            if (toStringMethodInfo != null)
            {
                //6.5.
                // TypeObject.GetMethod(string MethodName) can get the MethodInfo
                // by the string value of MethodName.
                string gamerToString = (string)toStringMethodInfo.Invoke(gamerInstance, null);
                Console.WriteLine($"gamerToString : {gamerToString}");
                //public override string ToString() ------------------------
                //gamerToString : Name :  ; GameScore : -1
                // 6.6.
                // TypeObject.GetProperty(string PropertyName) can get the PropertyInfo
                // by using string value of PropertyName.
                PropertyInfo namePropertyInfo = gamerType.GetProperty("Name");
                if (namePropertyInfo != null)
                {
                    MethodInfo namePropertySetMethodInfo = namePropertyInfo.SetMethod;
                    MethodInfo namePropertyGetMethodInfo = namePropertyInfo.GetMethod;
                    object[] namePropertySetMethodParameters = new object[1];
                    namePropertySetMethodParameters[0] = "Name03"; //Name
                    namePropertySetMethodInfo.Invoke(gamerInstance, namePropertySetMethodParameters);
                    object name = namePropertyGetMethodInfo.Invoke(gamerInstance, null);
                    // 6.7.
                    PropertyInfo gameScorePropertyInfo = gamerType.GetProperty("GameScore");
                    if (gameScorePropertyInfo != null)
                    {
                        MethodInfo gameScorePropertySetMethodInfo = gameScorePropertyInfo.SetMethod;
                        MethodInfo gameScorePropertyGetMethodInfo = gameScorePropertyInfo.GetMethod;
                        object[] gameScorePropertySetMethodParameters = new object[1];
                        gameScorePropertySetMethodParameters[0] = 3500; //gameScore
                        gameScorePropertySetMethodInfo.Invoke(gamerInstance,
gameScorePropertySetMethodParameters);
                        object gameScore = gameScorePropertyGetMethodInfo.Invoke(gamerInstance, null);
                        // 6.8.
                        Console.WriteLine("Name Property and GameScore Property ------------------------
");
                        Console.WriteLine($"name == {name} ; gameScore == {gameScore}");
                    }
                }
                //Name Property and GameScore Property------------------------
                //name == Name03; gameScore == 3500
                // 6.9.
                // TypeObject.GetMethod(string MethodName) can get the MethodInfo
                // by the string value of MethodName.
                Console.WriteLine("public void PrintName() ------------------------");
                MethodInfo printNameMethodInfo = gamerType.GetMethod("PrintName");
                if (printNameMethodInfo != null) printNameMethodInfo.Invoke(gamerInstance, null);
                //public void PrintName() ------------------------
                //Name == Name03
                // 6.10.
                // TypeObject.GetMethod(string MethodName) can get the MethodInfo
                // by the string value of MethodName.
                Console.WriteLine("public void PrintGameScore() ------------------------");
                MethodInfo printGameScoreMethodInfo = gamerType.GetMethod("PrintGameScore");
                if (printGameScoreMethodInfo != null) printGameScoreMethodInfo.Invoke(gamerInstance, null);
                //public void PrintGameScore() ------------------------
                //GameScore == 3500
                // 6.11.
                // TypeObject.GetMethod(string MethodName) can get the MethodInfo
                // by the string value of MethodName.
                Console.WriteLine("public override string ToString() ------------------------");
                gamerToString = (string)toStringMethodInfo.Invoke(gamerInstance, null);
                Console.WriteLine($"gamerToString : {gamerToString}");
            }
            //public override string ToString() ------------------------
            //gamerToString : Name : Name03 ; GameScore : 3500
            // 6.12.
            // 6.12.1.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public string SetNameAndGameScore(string name, int gameScore) ------------
-----------");
            MethodInfo setNameAndGameScoreMethodInfo = gamerType.GetMethod("SetNameAndGameScore");
            // 6.12.2.
            // Create object[] array for the parameters of the method.
            object[] methodParameters = new object[2];
            methodParameters[0] = "Name02"; //Name
            methodParameters[1] = 3000;     //GameScore
            // 6.12.3.
            // MethodInfoObject.Invoke(object InstanceObject, object[] methodParametersObjectArr)
            // invoke the method and get the return value.
            if (setNameAndGameScoreMethodInfo == null) return;
            string gamerStr = (string)setNameAndGameScoreMethodInfo.Invoke(gamerInstance,
methodParameters);
            Console.WriteLine("gamerStr : {0}", gamerStr);
            //public string SetNameAndGameScore(string name, int gameScore) ------------------------
            //gamerStr : Name: Name02; GameScore: 3000
        }


          //7. --------------------------------------------
           static void LateBinding2()
           {
            // 7.1.
            // Load the executing assembly (DLL or Exe)
            // Because GamerA class is in that assembly.
            Assembly executingAssembly = Assembly.LoadFrom("ClassLibrary1.dll");
            // 7.2.
            // Type.GetType(string TypeFullName) is a static method
            // The parameter is string value of TypeFullName
            // which is "NameSpace.ClassName".
            // It will return the Type of TypeFullName.
            // Type can be used to create an instance object dynamically.
            Type gamerType = executingAssembly.GetType("OnLineGameA.GamerA");
            //E.g.
            //The popular way of using Reflection is to dynamically load DLLs from XML file.
            //For example, Create several DLLs into a folder.
            //Write a XML to contain the DLLs Name which you want to load.
            //Using string value of DLLs Name and using Reflection to
            //load dynamically DLLs into your project.
            //Reflection will allows users to dynamically create object instance of the Type from DLLs.
            //It also allow users to dynamically invoke its methods or access its fields and properties.
            //This will not cover in this tutorial.
            // 7.3.
            // Activator.CreateInstance(Type)
            // creates the instance of the Type,OnLineGame.Gamer.
            object gamerInstance = Activator.CreateInstance(gamerType);
            // 7.4.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public override string ToString() ------------------------");
            MethodInfo toStringMethodInfo = gamerType.GetMethod("ToString");
            if (toStringMethodInfo != null)
            {
                //7.5.
                // TypeObject.GetMethod(string MethodName) can get the MethodInfo
                // by the string value of MethodName.
                string gamerToString = (string)toStringMethodInfo.Invoke(gamerInstance, null);
                Console.WriteLine($"gamerToString : {gamerToString}");
                //public override string ToString() ------------------------
                //gamerToString : Name :  ; GameScore : -1
                // 7.6.
                // TypeObject.GetProperty(string PropertyName) can get the PropertyInfo
                // by using string value of PropertyName.
                PropertyInfo namePropertyInfo = gamerType.GetProperty("Name");
                if (namePropertyInfo != null)
                {
                   MethodInfo namePropertySetMethodInfo = namePropertyInfo.SetMethod;
                   MethodInfo namePropertyGetMethodInfo = namePropertyInfo.GetMethod;
                   object[] namePropertySetMethodParameters = new object[1];
                    namePropertySetMethodParameters[0] = "Name03"; //Name
                    namePropertySetMethodInfo.Invoke(gamerInstance, namePropertySetMethodParameters);
                   object name = namePropertyGetMethodInfo.Invoke(gamerInstance, null);
                   // 7.7.
                   PropertyInfo gameScorePropertyInfo = gamerType.GetProperty("GameScore");
                   if (gameScorePropertyInfo != null)
                    {
                       MethodInfo gameScorePropertySetMethodInfo = gameScorePropertyInfo.SetMethod;
```

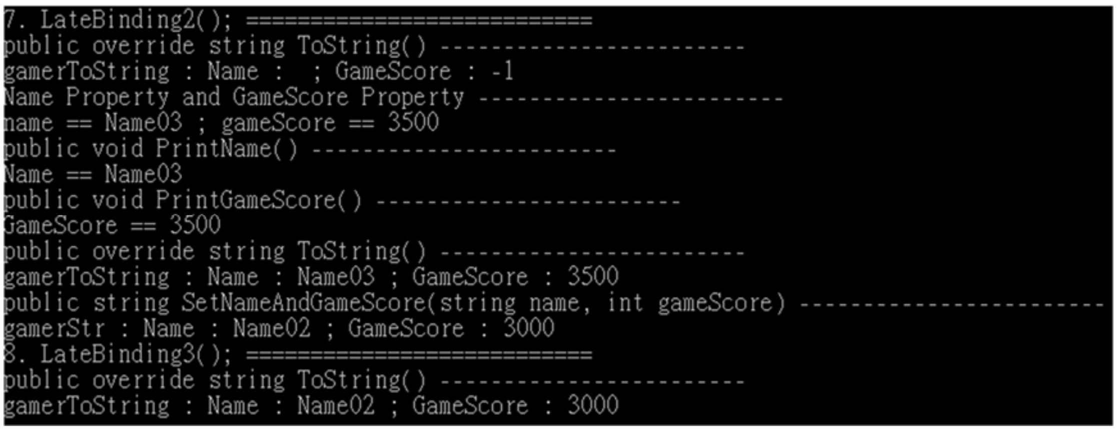
```
                    MethodInfo gameScorePropertyGetMethodInfo = gameScorePropertyInfo.GetMethod;
                    object[] gameScorePropertySetMethodParameters = new object[1];
                    gameScorePropertySetMethodParameters[0] = 3500; //gameScore
                    gameScorePropertySetMethodInfo.Invoke(gamerInstance,
gameScorePropertySetMethodParameters);
                    object gameScore = gameScorePropertyGetMethodInfo.Invoke(gamerInstance, null);
                    // 7.8.
                    Console.WriteLine("Name Property and GameScore Property ------------------------
");

                    Console.WriteLine($"name == {name} ; gameScore == {gameScore}");
                }
            }
            //Name Property and GameScore Property------------------------
            //name == Name03; gameScore == 3500
            // 7.9.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public void PrintName() ------------------------");
            MethodInfo printNameMethodInfo = gamerType.GetMethod("PrintName");
            if (printNameMethodInfo != null) printNameMethodInfo.Invoke(gamerInstance, null);
            //public void PrintName() ------------------------
            //Name == Name03
            // 7.10.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public void PrintGameScore() ------------------------");
            MethodInfo printGameScoreMethodInfo = gamerType.GetMethod("PrintGameScore");
            if (printGameScoreMethodInfo != null) printGameScoreMethodInfo.Invoke(gamerInstance, null);
            //public void PrintGameScore() ------------------------
            //GameScore == 3500
            // 7.11.
            // TypeObject.GetMethod(string MethodName) can get the MethodInfo
            // by the string value of MethodName.
            Console.WriteLine("public override string ToString() ------------------------");
             gamerToString = (string)toStringMethodInfo.Invoke(gamerInstance, null);
            Console.WriteLine($"gamerToString : {gamerToString}");
        }
        //public override string ToString() ------------------------
        //gamerToString : Name : Name03 ; GameScore : 3500
        // 7.12.
        // 7.12.1.
        // TypeObject.GetMethod(string MethodName) can get the MethodInfo
        // by the string value of MethodName.
        Console.WriteLine("public string SetNameAndGameScore(string name, int gameScore) -------------
-----------");
        MethodInfo setNameAndGameScoreMethodInfo = gamerType.GetMethod("SetNameAndGameScore");
        // 7.12.2.
        // Create object[] array for the parameters of the method.
        object[] methodParameters = new object[2];
         methodParameters[0] = "Name02"; //Name
         methodParameters[1] = 3000;     //GameScore
        // 7.12.3.
        // MethodInfoObject.Invoke(object InstanceObject, object[] methodParametersObjectArr)
        // invoke the method and get the return value.
```

```
            if (setNameAndGameScoreMethodInfo == null) return;
            string gamerStr = (string)setNameAndGameScoreMethodInfo.Invoke(gamerInstance,
methodParameters);
            Console.WriteLine("gamerStr : {0}", gamerStr);
            //public string SetNameAndGameScore(string name, int gameScore) ------------------------
            //gamerStr : Name: Name02; GameScore: 3000
        }


        //8. --------------------------------------------
        static void LateBinding3()
        {
            // 8.1.
            // Load the current executing assembly (DLL or Exe)
            // Because Gamer class is in current assembly.
            Assembly executingAssembly = Assembly.GetExecutingAssembly();
            // 8.2.
            // Type.GetType(string TypeFullName) is a static method
            // The parameter is string value of TypeFullName
            // which is "NameSpace.ClassName".
            // It will return the Type of TypeFullName.
            // Type can be used to create an instance object dynamically.
            Type gamerType = executingAssembly.GetType("OnLineGame.Gamer");
            // 8.3.
            // Consturctor
            // 8.3.1.
            // Create object[] array for the parameters of the Consturctor.
            object[] consturctorParameters = new object[2];
             consturctorParameters[0] = "Name02"; //Name
             consturctorParameters[1] = 3000;     //GameScore
            // 8.3.2.
            // Activator.CreateInstance(Type)
            // creates the instance of the Type,OnLineGame.Gamer.
            object gamerInstance = Activator.CreateInstance(gamerType, consturctorParameters);
            // 8.4.
            Console.WriteLine("public override string ToString() ------------------------");
            MethodInfo toStringMethodInfo = gamerType.GetMethod("ToString");
            if (toStringMethodInfo != null)
             {
                string gamerToString = (string) toStringMethodInfo.Invoke(gamerInstance, null);
                Console.WriteLine($"gamerToString : {gamerToString}");
                //public override string ToString() ------------------------
                //gamerToString : Name : Name02 ; GameScore : 3000
             }
        }
    }
}


namespace OnLineGame
{
    // This class has 2 constructors, 2 properties, and 4 methods
    public class Gamer
     {
        // Properties ----------------------------
        public string Name { get; set; }
        public int GameScore { get; set; }
        // Constructor ----------------------------
        public Gamer(string name, int gameScore)
        {
            GameScore = gameScore;
            Name = name;
        }
        public Gamer()
        {
            GameScore = -1;
            Name = string.Empty;
        }
        // Methods ----------------------------
        public void PrintName()
        {
            Console.WriteLine($"Name == {Name}");
        }
        public void PrintGameScore()
        {
            Console.WriteLine($"GameScore == {GameScore}");
        }
        public override string ToString()
        {
            return $"Name : {Name} ; GameScore : {GameScore}";
        }
        public string SetNameAndGameScore(string name, int gameScore)
        {
            Name = name;
            GameScore = gameScore;
            return ToString();
        }
    }
}


/*
1.
Reflection
1.1.
Reflection can find Types in an assembly by giving the string value of Type Name.
In addition, Reflection can use the Type to dynamically
create an object instance of a Type by late binding at run time.
Furthermore, Reflection can dynamically
invoke its methods or access its fields and properties.
1.2.
E.g.
The popular way of using Reflection is to dynamically load DLLs from XML file.
For example, Create several DLLs into a folder.
Write a XML to contain the DLLs Name which you want to load.
Using string value of DLLs Name and using Reflection to
load dynamically DLLs into your project.
Reflection will allows users to dynamically create object instance of the Type from DLLs.
It also allow users to dynamically invoke its methods or access its fields and properties.
This will not cover in this tutorial.
---------------------------------------------------------------------
2.
Early binding V.S. Late binding:
2.1.
Early binding is better for performance and can flag errors at compile time.
2.2.
Late binding performance is worse than Early binding.
In addition, Late binding has a risk of run time exceptions
```

```
if the string value of Type Name or Method name is incorrect.
However, Late binding is good when working with onjects
that are not available at compile time.
E.g.
The popular way of using Reflection is to dynamically load DLLs from XML file.
2.3.
Load DLLs
Reference:
https://stackoverflow.com/questions/18483354/get-assembly-of-program-from-a-dll
https://stackoverflow.com/questions/43318419/get-dll-file-extension-by-system-reflection-assembly
*/
```



```
1-4. ReflectionBasic(); ===========================
1. Type details -----------------------------------
t1.FullName == OnLineGame.Gamer which is NameSpace.ClassName
t1.Name == Gamer which is ClassName
t1.Namespace = OnLineGame which is NameSpace
2. Methods -----------------------------------
Methods in OnLineGame.Gamer Class
String get_Name
Void set_Name
Int32 get_GameScore
Void set_GameScore
Void PrintName
Void PrintGameScore
String ToString
String SetNameAndGameScore
Boolean Equals
Int32 GetHashCode
Type GetType
3. Properties -----------------------------------
Properties in OnLineGame.Gamer Class
String Name
Int32 GameScore
4. Constructors -----------------------------------
Constructors in OnLineGame.Gamer Class
Void .ctor(System.String, Int32)
Void .ctor()
5. EarlyBinding(); ===========================
Name == Name01 ; GameScore == 5000
Name == Name01
GameScore == 5000
6. LateBinding(); ===========================
public override string ToString() -----------------------
gamerToString : Name :  ; GameScore : -1
Name Property and GameScore Property -----------------------
name == Name03 ; gameScore == 3500
public void PrintName() -----------------------
Name == Name03
public void PrintGameScore() -----------------------
GameScore == 3500
public override string ToString() -----------------------
gamerToString : Name : Name03 ; GameScore : 3500
public string SetNameAndGameScore(string name, int gameScore) -----------------------
gamerStr : Name : Name02 ; GameScore : 3000
```



```
7. LateBinding2(); ===========================
public override string ToString() -----------------------
gamerToString : Name :  ; GameScore : -1
Name Property and GameScore Property -----------------------
name == Name03 ; gameScore == 3500
public void PrintName() -----------------------
Name == Name03
public void PrintGameScore() -----------------------
GameScore == 3500
public override string ToString() -----------------------
gamerToString : Name : Name03 ; GameScore : 3500
public string SetNameAndGameScore(string name, int gameScore) -----------------------
gamerStr : Name : Name02 ; GameScore : 3000
8. LateBinding3(); ===========================
public override string ToString() -----------------------
gamerToString : Name : Name02 ; GameScore : 3000
```