(T13)Generic 泛型。比較 Array、Collection。比較 GenericCollection(泛型集合)、NonGenericCollection

CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92

==

(T13)Generic 泛型。比較 Array、Collection。比較 GenericCollection(泛型集合)、NonGenericCollection

==

==

0. Summary

1.
Array V.S. Collection(E.g. ArrayList, Hashtable ...etc.)
1.1.
Array is strongly typed and type safe.
Index starts from 0 and
the size can not be increased once initialized.
1.2.
Collection is not Strongly Type and not Type safe.
But the size can be increased by using Add()
or can be decreased by using Remove().

2.
Generic Collection V.S. Non-Generic Collection.
2.1.
Generic Collections (System.Collections.Generic)
includes List<T>, Dictionary<TKey, TValue>, Stack<T>, Queue<T>
2.2.
Non-Generic Collection (System.Collections)
includes ArrayList, Hashtable, Stack, Queue.
2.3.
Generic Collections is always better.
Generic Collections is type safe,
and the size can be changeable by using Add(), Remove ...etc.

==

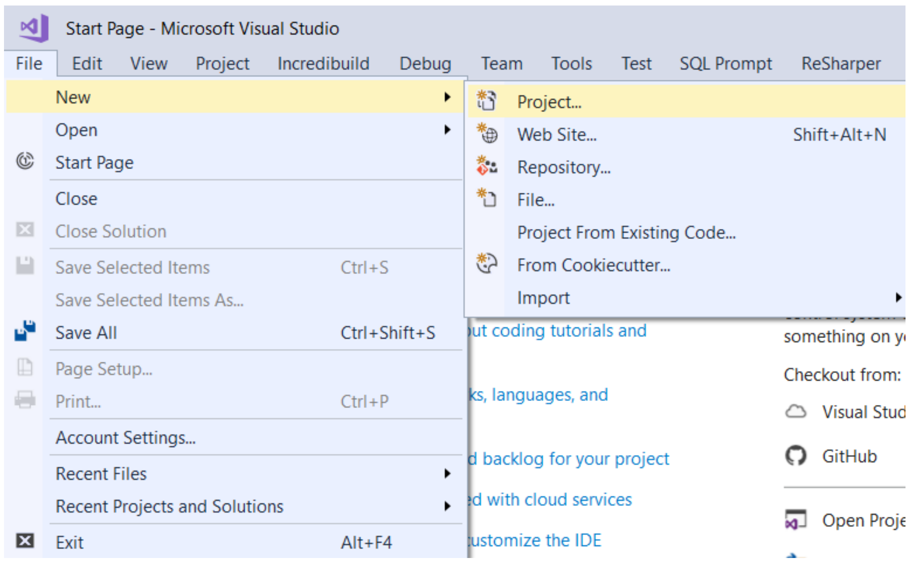
1. New Project

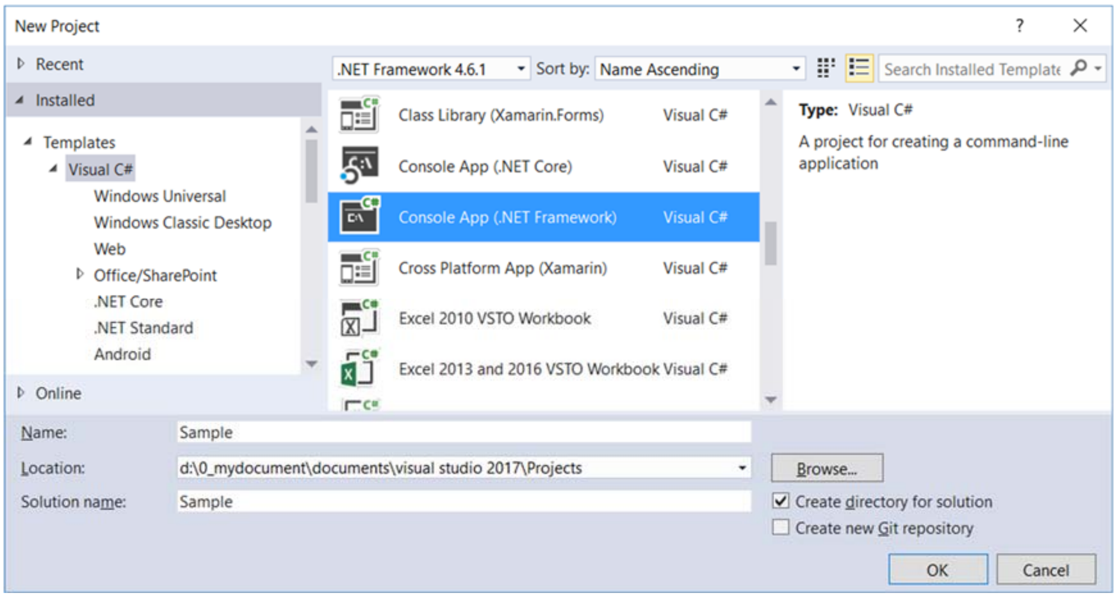
1.1. Create New Project

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
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#
Search Installed Templates
Browse...
OK
Cancel

.NET Framework 4.6.1

Sort by: Name Ascending

Type: Visual C#

A project for creating a command-line application

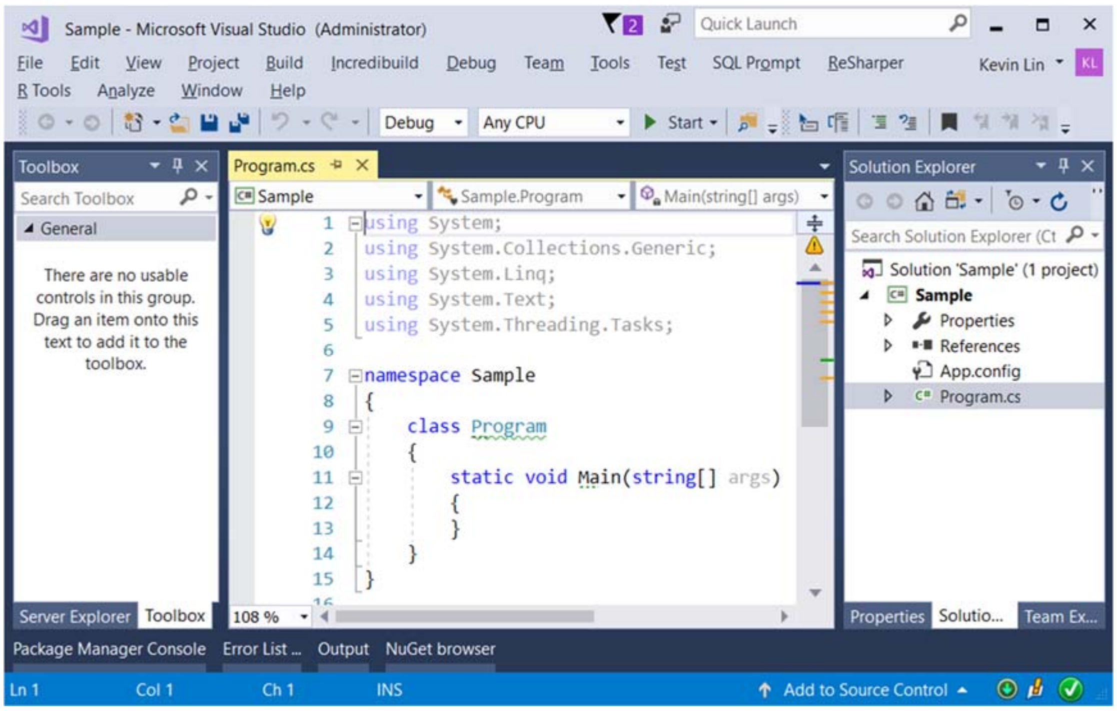
Name: Sample

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

Solution name: Sample

☑ Create directory for solution

☐ Create new Git repository



```
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)
        {
        }
    }
}
```

==

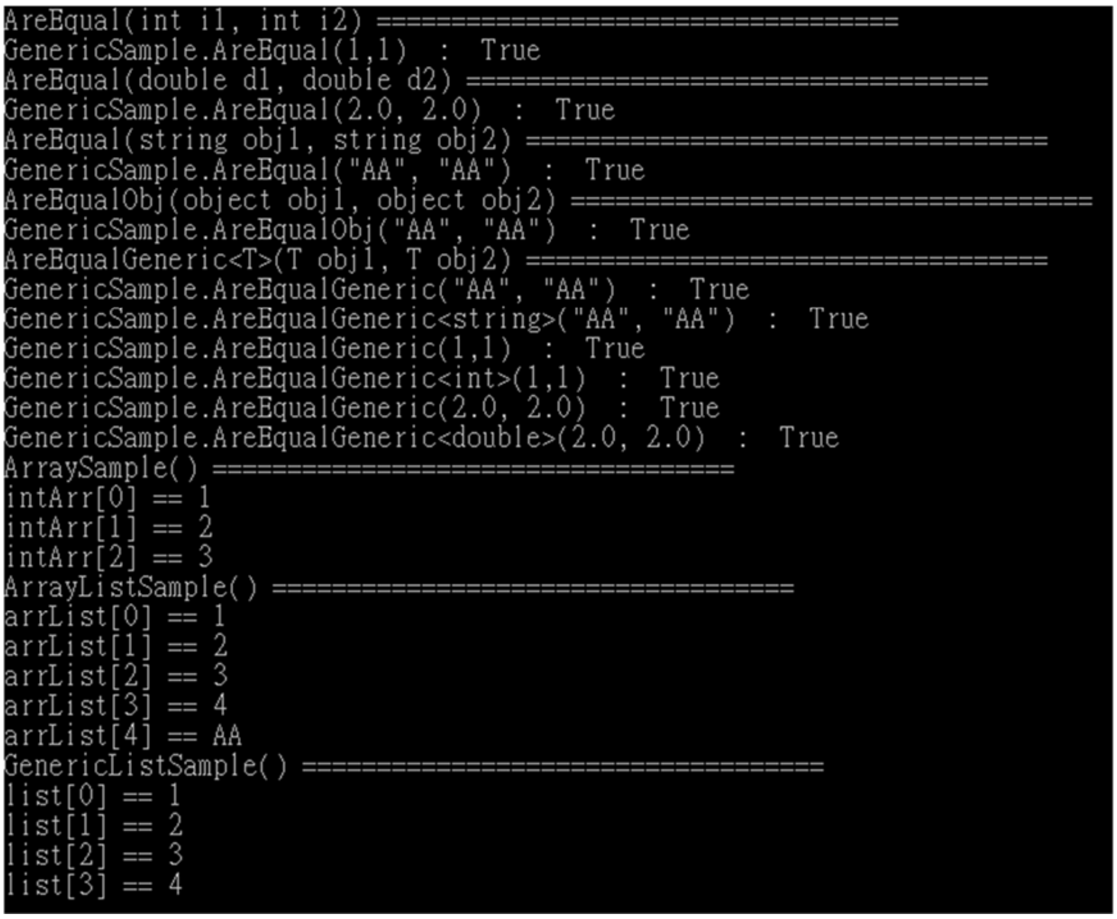
2. Program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1. -----------------------------------
            Console.WriteLine("AreEqual(int i1, int i2) ===================================");
            Console.WriteLine($"GenericSample.AreEqual(1,1)  :  {GenericSample.AreEqual(1, 1)}");
            Console.WriteLine("AreEqual(double d1, double d2) ===================================");
            Console.WriteLine($"GenericSample.AreEqual(2.0, 2.0)  :  {GenericSample.AreEqual(2.0, 2.0)}");
            Console.WriteLine("AreEqual(string obj1, string obj2) ===================================");
            Console.WriteLine($"GenericSample.AreEqual(\"AA\", \"AA\")  :  {GenericSample.AreEqual("AA", "A
A")}");
            // It need different AreEqual() methods to handle different type of inputs.
            // The logic of AreEqual() method are not reusable.
            // 2. -----------------------------------
            Console.WriteLine("AreEqualObj(object obj1, object obj2)
===================================");
            Console.WriteLine($"GenericSample.AreEqualObj(\"AA\", \"AA\")  :  {GenericSample.AreEqualObj("A
A", "AA")}");
            // Using "object" type can reuse the logic of AreEqualObj(object obj1, object obj2) method,
            // but it allows users to pass int to obj1 and string to obj2.
            // Thus, AreEqualObj() method is not type safe.
            // In addition, AreEqualObj() reduce the performance,
            // because it needs an extra action
            // that coverting object type to other type in order to do comparation.
            // 3. -----------------------------------
            Console.WriteLine(@"AreEqualGeneric<T>(T obj1, T obj2) ===================================");
            Console.WriteLine($"GenericSample.AreEqualGeneric(\"AA\", \"AA\")  :  " +
                              $"{GenericSample.AreEqualGeneric("AA", "AA")}");
            Console.WriteLine($"GenericSample.AreEqualGeneric<string>(\"AA\", \"AA\")  :  " +
                              $"{GenericSample.AreEqualGeneric<string>("AA", "AA")}");
            Console.WriteLine($"GenericSample.AreEqualGeneric(1,1)  :  " +
                              $"{GenericSample.AreEqualGeneric(1, 1)}");
            Console.WriteLine($"GenericSample.AreEqualGeneric<int>(1,1)  :  " +
                              $"{GenericSample.AreEqualGeneric<int>(1, 1)}");
            Console.WriteLine($"GenericSample.AreEqualGeneric(2.0, 2.0)  :  " +
                              $"{GenericSample.AreEqualGeneric(2.0, 2.0)}");
            Console.WriteLine($"GenericSample.AreEqualGeneric<double>(2.0, 2.0)  :  " +
                              $"{GenericSample.AreEqualGeneric<double>(2.0, 2.0)}");
            // Using Generic type can reuse the logic of AreEqualGeneric<T>(T obj1, T obj2) method,
            // The users have to pass the type to T in order to use generic method.
            // It makes parameters become type safe,
            // and also doesn't need to an extra converting.
            // T can be Class type or Interface type.
            // 4. ------------------------------------
            Console.WriteLine(@"ArraySample() ===================================");
            GenericSample.ArraySample();
            // 5. ------------------------------------
            Console.WriteLine(@"ArrayListSample() ===================================");
            GenericSample.ArrayListSample();
            // 6. ------------------------------------
            Console.WriteLine(@"GenericListSample() ===================================");
            GenericSample.GenericListSample();
            Console.ReadLine();
        }
    }

public class GenericSample
    {
        // 1. ------------------------------------
        public static bool AreEqual(int i1, int i2)
        {
            return i1 == i2;
        }
        public static bool AreEqual(double d1, double d2)
        {
            return d1.Equals(d2);
        }
        public static bool AreEqual(string str1, string str2)
        {
            return str1.Equals(str2);
        }
        // 2. ------------------------------------
        public static bool AreEqualObj(object obj1, object obj2)
        {
            return obj1.Equals(obj2);
        }
        // 3. ------------------------------------
        public static bool AreEqualGeneric<T>(T obj1, T obj2)
        {
            return obj1.Equals(obj2);
        }
        // 4. ------------------------------------
        public static void ArraySample()
        {
            int[] intArr = new int[3];
             intArr[0] = 1;
             intArr[1] = 2;
             intArr[2] = 3;
            //intArr[3] = 4;  // RunTime Error
            for (int i = 0; i<intArr.Length ; i++)
             {
                Console.WriteLine($"intArr[{i}] == {intArr[i]}");
             }
            //Array is strongly typed and type safe.
            //Index starts from 0 and
            //the size can not be increased once initialized.
        }

       // 5. ------------------------------------
        public static void ArrayListSample()
        {
            ArrayList arrList = new ArrayList();
            arrList.Add(1);
            arrList.Add(2);
            arrList.Add(3);
            arrList.Add(4);
            arrList.Add("AA");  // not type safe
            for (int i = 0; i < arrList.Count; i++)
            {
                Console.WriteLine($"arrList[{i}] == {arrList[i]}");
            }
            //Collection is not Strongly Type and not Type safe.
            //But the size can be increased by using Add()
            //or can be decreased by using Remove().
        }
        // 6. ------------------------------------
        public static void GenericListSample()
        {
            List<int> list = new List<int>();
            list.Add(1);
            list.Add(2);
            list.Add(3);
            list.Add(4);
            //list.Add("AA"); // compiler error, generic is type safe.
            for (int i = 0; i < list.Count; i++)
            {
                Console.WriteLine($"list[{i}] == {list[i]}");
            }
            //Generic Collections is always better.
            //Generic Collections is type safe,
            //and the size can be changeable by using Add(), Remove...etc.
        }
    }
}

/*
1.
Array V.S. Collection(E.g. ArrayList, Hashtable ...etc.)
1.1.
Array is strongly typed and type safe.
Index starts from 0 and
the size can not be increased once initialized.
1.2.
Collection is not Strongly Type and not Type safe.
But the size can be increased by using Add()
or can be decreased by using Remove().
---------------------------------------------------
2.
Generic Collection V.S. Non-Generic Collection.
2.1.
Generic Collections (System.Collections.Generic)
includes List<T>, Dictionary<TKey, TValue>, Stack<T>, Queue<T>
2.2.
Non-Generic Collection (System.Collections)
includes ArrayList, Hashtable, Stack, Queue.
2.3.
Generic Collections is always better.
Generic Collections is type safe,
and the size can be changeable by using Add(), Remove ...etc.
*/
```



```
AreEqual(int i1, int i2) ===================================
GenericSample.AreEqual(1,1)  :  True
AreEqual(double d1, double d2) ===================================
GenericSample.AreEqual(2.0, 2.0)  :  True
AreEqual(string obj1, string obj2) ===================================
GenericSample.AreEqual("AA", "AA")  :  True
AreEqualObj(object obj1, object obj2) ===================================
GenericSample.AreEqualObj("AA", "AA")  :  True
AreEqualGeneric<T>(T obj1, T obj2) ===================================
GenericSample.AreEqualGeneric("AA", "AA")  :  True
GenericSample.AreEqualGeneric<string>("AA", "AA")  :  True
GenericSample.AreEqualGeneric(1,1)  :  True
GenericSample.AreEqualGeneric<int>(1,1)  :  True
GenericSample.AreEqualGeneric(2.0, 2.0)  :  True
GenericSample.AreEqualGeneric<double>(2.0, 2.0)  :  True
ArraySample() ===================================
intArr[0] == 1
intArr[1] == 2
intArr[2] == 3
ArrayListSample() ===================================
arrList[0] == 1
arrList[1] == 2
arrList[2] == 3
arrList[3] == 4
arrList[4] == AA
GenericListSample() ===================================
list[0] == 1
list[1] == 2
list[2] == 3
list[3] == 4
```