(T12)比較Contains、Equals、SequenceEqual、GetHashCode。比較IEqualityComparer、AnonymousTypes(匿名型別)  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92  
=======================================================================  
(T12)比較Contains、Equals、SequenceEqual、GetHashCode。比較IEqualityComparer、AnonymousTypes(匿名型別)  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project

-----------

2. Program.cs  
=======================================================================

0. Summary

0.

---------------------------------------

0.1.

Three popular ways to solve the problems of Contains() and Equals() and SequenceEqual() for Reference Type, ClassA

----------------------

0.1.1.

Override Equals() and GetHashCode() methods in ClassA

----------------------

0.1.2.

If you can not access ClassA, then

Use another overloaded version of SequenceEqual(),Contains() method which can take a sub-class of IEqualityComparer as parameter.

----------------------

0.1.3.

If you can not access ClassA, then

use Select() or SelectMany() to project into a new anonymous type,

which overrides Equals() and GetHashCode() methods.

---------------------------------------

0.2.

Three popular ways to solve the problems of Compare() and Sort() for Reference Type, ClassA

----------------------

0.2.1.

ClassA implement IComparable<ClassA>

and then implement

//public int CompareTo(ClassA other)

----------------------

0.2.2.

If you can not access ClassA, then

use other class to implement IComparer<ClassA>

E.g.

//public class ClassACompareName: IComparer<ClassA >

and then implement

public int Compare(ClassA current, ClassA other)

----------------------

0.2.3.

If you can not access ClassA, then

use anonymous type to provide the method to compare.

1. New Project

1.1. Create New Project

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

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

Name: **Sample**







=============================================

2. Program.cs

using System;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

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

            Console.WriteLine("1 ValueTypeEqualSample =============================");

            ValueTypeEqualSample();

            // 2. ----------------------------------

            Console.WriteLine("2 EnumEqualSample =============================");

            EnumEqualSample();

            // 3. ----------------------------------

            Console.WriteLine("3 ReferenceTypeEqualSample =============================");

            ReferenceTypeEqualSample();

            // 4. ----------------------------------

            // Reason to override Equal method.

            Console.WriteLine("4 ReferenceTypeEqualSample2 =============================");

            ReferenceTypeEqualSample2();

            // 5. ----------------------------------

            Console.WriteLine("5 ReferenceTypeEqualSample3 =============================");

            ReferenceTypeEqualSample3();

            Console.ReadLine();

        }

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

        static void ValueTypeEqualSample()

        {

            int i = 10;

            int j = 10;

            Console.WriteLine($"i == j  :  {i == j}");

            Console.WriteLine($"i.Equals(j) : {i.Equals(j)}");

            //i == j  :  True

            //i.Equals(j) : True

            //1.

            // System.Object has Equals() virtual method.

            // int is value type which stored in the stack.

            // Thus, "==" and "Equals()" will return the same result.

        }

        // 2. ----------------------------------

        static void EnumEqualSample()

        {

            MagicType enum1 = MagicType.Metal;

            MagicType enum2 = MagicType.Metal;

            Console.WriteLine($"enum1 == enum2  :  {enum1 == enum2}");

            Console.WriteLine($"enum1.Equals(enum2)  ;  {enum1.Equals(enum2)}");

            //enum1 == enum2  :  True

            //enum1.Equals(enum2); True

            //1.

            //Enum keyword can create enumerations and

            //it is strongly value typed constants.

            //The default underlying type of an enum is int.

            //Since, enum is value type, and

            //both enums has the same underlying integer value.

            //Thus, "==" and "Equals()" will return the same result.

        }

        // 3. ----------------------------------

        static void ReferenceTypeEqualSample()

        {

            Gamer g1 = new Gamer();

            g1.FirstName = "F01";

            g1.LastName = "L01";

            Gamer g2 = g1;

            Console.WriteLine($"g1 == g2  :  {g1 == g2}");

            Console.WriteLine($"g1.Equals(g2)  :  {g1.Equals(g2)}");

            //g1 == g2  :  True

            //g1.Equals(g2)  :  True

            //1.

            //"==" operator checks for reference equality.

            //g1 and g2 are different object reference variables which stores in Stack.

            //Both g1 and g2 refer to the same memory location.

            //Thus, g1 == g2 will return true.

            //2.

            //Equals() method checks for value equality.

            //g1 and g2 variables are pointing to the same object instance which stores in Heap.

            //thus, the values are the same,

            //Therefore, g1.Equals(g2) will return true.

            //3.

            //If "obj1==obj2", that means they have reference equality,

            //then they must also have value equality which is obj1.Equals(obj2).

            //However, if obj1.Equals(obj2) that means they have value equality.

            //then "obj1==obj2" reference equality will not be guarantee.

        }

        // 4. ----------------------------------

        // Reason to override Equal method.

        static void ReferenceTypeEqualSample2()

        {

            Gamer g1 = new Gamer();

            g1.FirstName = "F01";

            g1.LastName = "L01";

            Gamer g2 = new Gamer();

            g2.FirstName = "F01";

            g2.LastName = "L01";

            Console.WriteLine($"g1 == g2  :  {g1 == g2}");

            Console.WriteLine($"g1.Equals(g2)  :  {g1.Equals(g2)}");

            //g1 == g2  :  False

            //g1.Equals(g2)  :  False

            //1.

            //"==" operator checks for reference equality.

            //g1 and g2 are different object reference variables which stores in Stack.

            //g1 and g2 refer to the different memory location.

            //Thus, g1 != g2

            //2.

            //Equals() method checks for value equality.

            //g1 and g2 variables are pointing to the different object instance which stores in Heap.

            //thus, the values are different,

            //Therefore, !g1.Equals(g2)

            //3.

            //The FirstName and LastName are the same.

            //Thus, g1.Equals(g2) should return true.

            //However, it return false.

            //Therefore, it make sense to override Equals.

        }

        // 5. ----------------------------------

        // Reason to override Equal method.

        static void ReferenceTypeEqualSample3()

        {

            GamerA gA1 = new GamerA();

            gA1.FirstName = "F01";

            gA1.LastName = "L01";

            GamerA gA2 = new GamerA();

            gA2.FirstName = "F01";

            gA2.LastName = "L01";

            Console.WriteLine($"gA1 == gA2  :  {gA1 == gA2}");

            Console.WriteLine($"gA1.Equals(gA2)  :  {gA1.Equals(gA2)}");

            //gA1 == gA2  :  False

            //gA1.Equals(gA2)  :  True

            //1.

            //"==" operator checks for reference equality.

            //g1 and g2 are different object reference variables which stores in Stack.

            //g1 and g2 refer to the different memory location.

            //Thus, g1 != g2

            //2.

            //Equals() method checks for value equality.

            //g1 and g2 variables are pointing to the different object instance which stores in Heap.

            //thus, originally values are different and !g1.Equals(g2)

            //However, we override Equals method,

            //thus g1.Equals(g2) will return true.

        }

    }

    // 2. ----------------------------------

    public enum MagicType   // : int

    {

        Wood,

        Fire,

        Earth,

        Metal,

        Water

    }

    // 3. ----------------------------------

    public class Gamer

    {

        public string FirstName { get; set; }

        public string LastName { get; set; }

    }

    // 5. ----------------------------------

    public class GamerA

    {

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public override bool Equals(object obj)

        {

            // If the passed in object is null

            if (!(obj is GamerA))

            {

                return false;

            }

            return FirstName == ((GamerA)obj).FirstName

                && LastName == ((GamerA)obj).LastName;

        }

        public override int GetHashCode()

        {

            return FirstName.GetHashCode() ^ LastName.GetHashCode();

        }

        //1.

        //Reference:

        //<https://stackoverflow.com/questions/371328/why-is-it-important-to-override-gethashcode-when-equals-method-is-overridden>

        //<https://stackoverflow.com/questions/2363143/whats-the-best-strategy-for-equals-and-gethashcode>

        //<http://www.cnblogs.com/gentlewolf/archive/2007/07/09/810815.html>

        //<https://en.wikipedia.org/wiki/Exclusive_or>

        //It is important to override GetHashCode

        //when Equals method is overridden.

        //1.1.

        //HashCode is a 32bit int.

        //If the item will be used as a key in a dictionary, or HashSet<T>, etc

        //since key is used to group items into buckets.

        //(in the absence of a custom IEqualityComparer<T>)

        //If the hash-code for two items does not match,

        //they may never be considered equal

        //(Equals will simply never be called).

        //1.2.

        //The popular way to override GetHashCode is

        //using XOR to connect all fields.

    }

}

/\*

1.

Value Type V.S. Reference Type

E.g.

//int i1 = 2;

//int i2 = i1;

//i2++;

...

//PersonClass pc1 = new PersonClass();

//pc1.Id = 1;

//pc1.Name = "Name01";

//PersonClass pc2 = pc1;

//pc2.Id = 2;

//pc2.Name = "Name02";

    Stack         |        Heap

----------------------------------------------------

int i1 = 2;       |

int i2 = 3;       |

PersonClass pc1 --|-->

                  |     PersonClass object instance

PersonClass pc2 --|-->

                  |

------------------------------------------------------------

2.

Enum

---------------------

2.1.

Using Enum keyword to create enumerations and it is strongly value typed constants.

The default underlying type of an enum is int.

You may use " : short " to set the underlying type of an enum is short.

The default value for first element is ZERO and gets incremented by 1.

---------------------

2.2.

Syntax :

//public enum EnumName  [ : underlyingType ]

//{

//    EnumValue1 [ = StarValue],

//    EnumValue2,

//    EnumValue3 [ = SpecificValue],

//    ....

//}

E.g.1.

//public enum MagicType   // : int

//{

//    Wood,

//    Fire,

//    Earth,

//    Metal,

//    Water

//}

E.g.2.

//public enum MagicType2 : short

//{

//    Wood = 5,

//    Fire,   //6

//    Earth   //7

//}

E.g.3.

//public enum MagicType4 : short

//{

//    Wood = 8,

//    Fire = 100,

//    Earth = 20

//}

---------------------

2.3.

//int woodInt = (int)MagicType.Wood;

Convert Enum to int

---------------------

2.4.

//MagicType magicType1 = (MagicType)1;

Convert int to Enum

---------------------

2.5.

Enum.GetValues list Enum underlying type values.

E.g.

int[] MagicTypeValues = (int[])Enum.GetValues(typeof(MagicType));

//MagicTypeValues == {0,1,2,3,4}

---------------------

2.6.

Enum.GetNames list Enum underlying type names.

string[] MagicTypeNames = Enum.GetNames(typeof(MagicType));

//MagicTypeNames == {"Wood","Fire","Earth","Metal","Water"}

 \*/

Text

Description automatically generated