(T15)討論SetOperators的Distinct、Union、Intersect、Except、Concat  
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c  
=======================================================================  
(T15)討論SetOperators的Distinct、Union、Intersect、Except、Concat  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project : Sample

-----------

2. Sample : 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.

Distinct, Union, Intersect, Except, and Concat are Set operators.

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

1.1.

Distinct()

Reference:

[https://msdn.microsoft.com/en-us/library/bb348436(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb348436%28v=vs.110%29.aspx)

[https://msdn.microsoft.com/en-us/library/bb338049(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb338049%28v=vs.110%29.aspx)

1.1.1.

//Enumerable.Distinct<TSource>(this IEnumerable<TSource> source)

Returns distinct elements from a sequence

by using the default equality comparer to compare values.

1.1.2.

//Enumerable.Distinct<TSource>

//(this IEnumerable<TSource> source, IEqualityComparer<TSource> comparer)

Returns distinct elements from a sequence

by using a specified IEqualityComparer<T> to compare values.

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

1.2.

Union

Reference:

[https://msdn.microsoft.com/en-us/library/bb341731(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb341731%28v=vs.110%29.aspx)

[https://msdn.microsoft.com/en-us/library/bb358407(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb358407%28v=vs.110%29.aspx)

1.2.1.

//Enumerable.Union<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second)

Produces the set union of two sequences

by using the default equality comparer.

1.2.2.

//Enumerable.Union<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer)

Produces the set union of two sequences

by using a specified IEqualityComparer<T>.

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

1.3.

Intersect

Reference:

[https://msdn.microsoft.com/en-us/library/bb460136(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb460136%28v=vs.110%29.aspx)

[https://msdn.microsoft.com/en-us/library/bb355408(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb355408%28v=vs.110%29.aspx)

1.3.1.

//Enumerable.Intersect<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second)

Produces the set intersection of two sequences

by using the default equality comparer to compare values.

1.3.2.

//Enumerable.Intersect<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer)

Produces the set intersection of two sequences

by using the specified IEqualityComparer<T> to compare values.

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

1.4.

Except

Reference:

[https://msdn.microsoft.com/en-us/library/bb300779(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb300779%28v=vs.110%29.aspx)

[https://msdn.microsoft.com/en-us/library/bb336390(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb336390%28v=vs.110%29.aspx)

1.4.1.

//Enumerable.Except<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second)

Produces the set difference of two sequences

by using the default equality comparer to compare values.

1.4.2.

//Enumerable.Except<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer) )

Produces the set difference of two sequences

by using the specified IEqualityComparer<T> to compare values.

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

1.5.

Concat

Reference:

[https://msdn.microsoft.com/en-us/library/bb302894(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/bb302894%28v=vs.110%29.aspx)

//Enumerable.Concat<TSource>

//(this IEnumerable<TSource> first, IEnumerable<TSource> second)

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

1. New Project

1.1. Create New Project : Sample

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

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

Name: **Sample**







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

2. Sample : Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using OnLieGame;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            // 1. ==================================

            //DistinctSample()

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

            DistinctSample();

            // 2. ==================================

            //UnionAndConcatSample()

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

            UnionAndConcatSample();

            // 3. ==================================

            //IntersectSample()

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

            IntersectSample();

            // 4. ==================================

            //ExceptSample()

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

            ExceptSample();

            Console.ReadLine();

        }

       // 1. ==================================

        //DistinctSample()

        static void DistinctSample()

        {

            Console.WriteLine("1.1. strArr.Distinct() ------------------------------ ");

            string[] strArr = { "Name1", "name1", "Name2", "Name2", "Name3" };

            IEnumerable<string> strArrDistinct = strArr.Distinct();

            foreach (string strArrDistinctItem in strArrDistinct)

            {

                Console.WriteLine($"strArrDistinctItem=={strArrDistinctItem}");

            }

            // strArrDistinctItem==Name1

            // strArrDistinctItem==name1

            // strArrDistinctItem==Name2

            // strArrDistinctItem==Name3

            Console.WriteLine("1.2. strArr2.Distinct(StringComparer.OrdinalIgnoreCase) -------------------- ");

            string[] strArr2 = { "Name1", "name1", "Name2", "Name2", "Name3" };

            IEnumerable<string> strArr2Distinct = strArr2.Distinct(StringComparer.OrdinalIgnoreCase);

            foreach (string strArr2DistinctItem in strArr2Distinct)

            {

                Console.WriteLine($"strArr2DistinctItem=={strArr2DistinctItem}");

            }

            // strArr2DistinctItem==Name1

            // strArr2DistinctItem==Name2

            // strArr2DistinctItem==Name3

        }

       // 2. ==================================

        //UnionAndConcatSample()

        static void UnionAndConcatSample()

        {

            //Concat operator concatenates two sequences into one sequence.

            //Union combines two collections into one collection

            //and remove the duplicate elements.

            // 2.1. -------------------------------------

            //intArrA1.Concat(intArrA2)

            Console.WriteLine("2.1. intArrA1.Concat(intArrA2) -------------- ");

            int[] intArrA1 = { 1, 2, 3, 4, 5 };

            int[] intArrA2 = { 1, 3, 5, 7, 9 };

            IEnumerable<int> intArrA1ConcatintArrA2 =

                intArrA1.Concat(intArrA2);

            foreach (int item in intArrA1ConcatintArrA2)

            {

                Console.Write($" [ {item} ] ");

            }

            Console.WriteLine();

            //  [ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]  [ 1 ]  [ 3 ]  [ 5 ]  [ 7 ]  [ 9 ]

            // 2.2. -------------------------------------

            //intArrA1.Union(intArrA2)

            Console.WriteLine("2.2. intArrA1.Union(intArrA2) -------------- ");

            IEnumerable<int> intArrA1UnionintArrA2 =

                intArrA1.Union(intArrA2);

            foreach (int item in intArrA1UnionintArrA2)

            {

                Console.Write($" [ {item} ] ");

            }

            Console.WriteLine();

            //  [ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]  [ 7 ]  [ 9 ]

            // 2.3. -------------------------------------

            //gamerList1.Concat(gamerList2)

            Console.WriteLine("2.3. gamerList1.Concat(gamerList2) -------------- ");

            List<Gamer> gamerList1 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 2, Name = "Name2", TeamId = 2 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            List<Gamer> gamerList2 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 3, Name = "Name3", TeamId = 1 },

                new Gamer { Id = 4, Name = "Name4", TeamId = 1 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            IEnumerable<Gamer> gamerList1ConcatgamerList2 =

                gamerList1.Concat(gamerList2);

            foreach (Gamer gamer in gamerList1ConcatgamerList2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==2,GamerName=Name2,TeamId=2

            // GamerId==5,GamerName=Name9,TeamId=2

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==3,GamerName=Name3,TeamId=1

            // GamerId==4,GamerName=Name4,TeamId=1

            // GamerId==5,GamerName=Name9,TeamId=2

            // 2.4. -------------------------------------

            //gamerList1.Union(gamerList2)

            Console.WriteLine("2.4. gamerList1.Union(gamerList2) -------------- ");

            IEnumerable<Gamer> gamerList1UniongamerList2 =

                gamerList1.Union(gamerList2);

            foreach (Gamer gamer in gamerList1UniongamerList2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==2,GamerName=Name2,TeamId=2

            // GamerId==5,GamerName=Name9,TeamId=2

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==3,GamerName=Name3,TeamId=1

            // GamerId==4,GamerName=Name4,TeamId=1

            // GamerId==5,GamerName=Name9,TeamId=2

            // 2.5. -------------------------------------

            //gamerList1.Union(gamerList2, new GamerHelper())

            Console.WriteLine("2.5. gamerList1.Union(gamerList2, new GamerHelper()).OrderBy(g => g.Id) -------------- ");

            IEnumerable<Gamer> gamerList1UniongamerList2V2 =

                gamerList1.Union(gamerList2, new GamerHelper())

                .OrderBy(g => g.Id);

            ////IEnumerable<Gamer> gamerList1ConcatgamerList2V2 =

            ////    gamerList1.Concat(gamerList2, new GamerHelper())

            ////Concat with IEqualityComparer is not possible.

            ////because Union without IEqualityComparer can do the same thing.

            foreach (Gamer gamer in gamerList1UniongamerList2V2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==2,GamerName=Name2,TeamId=2

            // GamerId==3,GamerName=Name3,TeamId=1

            // GamerId==4,GamerName=Name4,TeamId=1

            // GamerId==5,GamerName=Name9,TeamId=2

        }

       // 3. ==================================

        //IntersectSample()

        static void IntersectSample()

        {

            //Intersect() returns the elements which both collections have.

            // 3.1. -------------------------------------

            //intArrA1.Intersect(intArrA2)

            Console.WriteLine("3.1. intArrA1.Intersect(intArrA2) ------------------ ");

            int[] intArrA1 = { 1, 2, 3, 4, 5 };

            int[] intArrA2 = { 1, 3, 5, 7, 9 };

            IEnumerable<int> intArrA1IntersectintArrA2 = intArrA1.Intersect(intArrA2);

            foreach (int item in intArrA1IntersectintArrA2)

            {

                Console.Write($" [ {item} ] ");

            }

            Console.WriteLine();

            // [ 1 ]  [ 3 ]  [ 5 ]

            // 3.2. -------------------------------------

            //gamerList1.Intersect(gamerList2)

            Console.WriteLine("3.2. gamerList1.Intersect(gamerList2).OrderBy(g => g.Id) -------------- ");

            List<Gamer> gamerList1 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 2, Name = "Name2", TeamId = 2 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            List<Gamer> gamerList2 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 3, Name = "Name3", TeamId = 1 },

                new Gamer { Id = 4, Name = "Name4", TeamId = 1 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            IEnumerable<Gamer> gamerList1IntersectgamerList2 =

                gamerList1.Intersect(gamerList2)

                .OrderBy(g => g.Id);

            foreach (Gamer gamer in gamerList1IntersectgamerList2)

            {

                Console.WriteLine($"{gamer}");

            }

            // Return nothing,

            //because the default Equals() and GetHashCode()

            //of Gamer is not good enough to let Gamer to compare its properties.

            // 3.3. -------------------------------------

            //gamerList1.Intersect(gamerList2)

            Console.WriteLine("3.3. gamerList1.Intersect(gamerList2, new GamerHelper()).OrderBy(g => g.Id) -------------- ");

            IEnumerable<Gamer> gamerList1IntersectgamerList2V2 =

                gamerList1.Intersect(gamerList2, new GamerHelper())

                .OrderBy(g => g.Id);

            foreach (Gamer gamer in gamerList1IntersectgamerList2V2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==5,GamerName=Name9,TeamId=2

        }

       // 4. ==================================

        //ExceptSample()

        static void ExceptSample()

        {

            //Except() returns the elements

            //that are in the first collection

            //but not in the second collection.

            // 4.1. -------------------------------------

            //intArrA1.Except(intArrA2)

            Console.WriteLine("4.1. intArrA1.Except(intArrA2) ------------------ ");

            int[] intArrA1 = { 1, 2, 3, 4, 5 };

            int[] intArrA2 = { 1, 3, 5, 7, 9 };

            IEnumerable<int> intArrA1ExceptintArrA2 = intArrA1.Except(intArrA2);

            foreach (int item in intArrA1ExceptintArrA2)

            {

                Console.Write($" [ {item} ] ");

            }

            Console.WriteLine();

            // [ 2 ]  [ 4 ]

            // 4.2. -------------------------------------

            //gamerList1.Except(gamerList2)

            Console.WriteLine("4.2. gamerList1.Except(gamerList2) -------------- ");

            List<Gamer> gamerList1 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 2, Name = "Name2", TeamId = 2 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            List<Gamer> gamerList2 = new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 3, Name = "Name3", TeamId = 1 },

                new Gamer { Id = 4, Name = "Name4", TeamId = 1 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 }

            };

            IEnumerable<Gamer> gamerList1ExceptgamerList2 =

                gamerList1.Except(gamerList2);

            foreach (Gamer gamer in gamerList1ExceptgamerList2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==1,GamerName=Name1,TeamId=1

            // GamerId==2,GamerName=Name2,TeamId=2

            // GamerId==5,GamerName=Name9,TeamId=2

            // 4.3. -------------------------------------

            //gamerList1.Except(gamerList2)

            Console.WriteLine("4.3. gamerList1.Except(gamerList2, new GamerHelper()).OrderBy(g => g.Id) -------------- ");

            IEnumerable<Gamer> gamerList1ExceptgamerList2V2 =

                gamerList1.Except(gamerList2, new GamerHelper())

                .OrderBy(g => g.Id);

            foreach (Gamer gamer in gamerList1ExceptgamerList2V2)

            {

                Console.WriteLine($"{gamer}");

            }

            // GamerId==2,GamerName=Name2,TeamId=2

        }

    }

}

namespace OnLieGame

{

    public class Team

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public override string ToString()

        {

            return $"TeamId=={Id},TeamName={Name}";

        }

    }

    public class TeamHelper

    {

        public static List<Team> GetSampleTeam()

        {

            return new List<Team>

            {

                new Team { Id = 1, Name = "Team1"},

                new Team { Id = 2, Name = "Team2"},

                new Team { Id = 3, Name = "Team3"},

            };

        }

    }

    public class Gamer

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public int TeamId { get; set; }

        public override string ToString()

        {

            return $"GamerId=={Id},GamerName={Name},TeamId={TeamId}";

        }

    }

    public class GamerHelper : IEqualityComparer<Gamer>

    {

        public static List<Gamer> GetSampleGamer()

        {

            return new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },

                new Gamer { Id = 2, Name = "Name2", TeamId = 2 },

                new Gamer { Id = 3, Name = "Name3", TeamId = 1 },

                new Gamer { Id = 4, Name = "Name4", TeamId = 1 },

                new Gamer { Id = 5, Name = "Name9", TeamId = 2 },

                new Gamer { Id = 6, Name = "Name10"}

            };

        }

        public bool Equals(Gamer x, Gamer y)

        {

            return y != null && x != null &&

                x.Id == y.Id &&

                x.Name == y.Name &&

                x.TeamId == y.TeamId;

        }

        public int GetHashCode(Gamer obj)

        {

            return obj.Id.GetHashCode() ^

                obj.TeamId.GetHashCode() ^

                obj.Name.GetHashCode();

        }

    }

}

Text

Description automatically generated

Text

Description automatically generated