(T20to22)討論 List 的 SimpleType、ReferenceType。討論 Sort、Contains、Equals、SequenceEqual
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==
(T20to22)討論 List 的 SimpleType、ReferenceType。討論 Sort、Contains、Equals、SequenceEqual
(T20)討論 List、SimpleType(簡單型別)、Sort
(T21)討論 List、ReferenceType(參考型別)
(T22)討論 List、ReferenceType(參考型別)、Sort、Contains、Equals、SequenceEqual
==

==

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.
List<T> is a collection of any type such as integer, string, or any complex type.
The size is changeable by using Add() and Remove().
The List<T> provide search() and sort() to manipulate data.

1.1.
//List<T>.Contains(T item)
Returns true if the items exists in the list, else false.

1.2.
//List<T>.Exists(Predicate<T> match)
Use the conditions defined by the lambda expression
to search for the element, then return true
if first matching item is existed, otherwise return false.

1.3.
//List<T>.Find(Predicate<T> match)
Use the conditions defined by the lambda expression
to search for the element, then return the first matching item.

1.4.
//List<T>.FindLast(Predicate<T> match)
Use the conditions defined by the lambda expression
to search for the element, then return the last matching item.
1.5.
//List<T>.FindAll(Predicate<T>)
Use the conditions defined by the lambda expression
to search for the element, then return the last matching all of them.

1.6.
1.6.1.
//List<T>.FindIndex(Predicate<T> match)
Use the conditions defined by the lambda expression

to search for the index of the element,
then return the first matching index.
Return -1 if the index can not be found.
1.6.2.
//List<T>.FindIndex(int startIndex, Predicate<T> match)
From the startIndex position,
use the conditions defined by the lambda expression
to search for the index of the element,
then return the first matching index.
Return -1 if the index can not be found.
1.6.3.
//List<T>.FindIndex(int startIndex, int count, Predicate<T> match)
From the "startIndex" position until "count" numbers of elements,
Use the conditions defined by the lambda expression
to search for the index of the element,
then return the first matching index.
Return -1 if the index can not be found.

1.7.
1.7.1.
//List<T>.FindLastIndex(Predicate<T> match)
Use the conditions defined by the lambda expression
to search for the index of the element,
then return the last matching index.
Return -1 if the index can not be found.
1.7.2.
//List<T>.FindLastIndex(int startIndex, Predicate<T> match)
From the startIndex position,
use the conditions defined by the lambda expression
to search for the index of the element,
then return the last matching index.
Return -1 if the index can not be found.
//1.7.3.
List<T>.FindLastIndex(int startIndex, int count, Predicate<T> match)
From the "startIndex" position until "count" numbers of elements,
use the conditions defined by the lambda expression
to search for the index of the element,
then return the last matching index.
Return -1 if the index can not be found.

1.8.
//Enumerable.ToList<TSource>(IEnumerable<TSource> source)
Creates a List<T> from an IEnumerable<T>.

1.9.
//List<T>.ToArray()
Copies the elements of the List<T> to a new array.

1.10.
//Enumerable.ToDictionary<TSource, TKey>
//(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector)
Creates a Dictionary<TKey, TValue> from an IEnumerable<T> according to a specified key selector function.

1.11.
1.11.1.
//List<T>.Add(T item)
Adds an object to the end of the List<T>.
1.11.2.
//List<T>.AddRange(IEnumerable<T> collection)
Adds the elements of the specified collection to the end of the List<T>.

1.12.
//List<T>.GetRange(int index, int count)
From the "index", and take "count" elements, then return a List<T>

1.13.
1.13.1.
//List<T>.Insert(int index, T item)
Inserts an element into the List<T> at the specified index.
1.13.2.
//List<T>.InsertRange(int index, IEnumerable<T> collection)
Inserts the elements of a collection into the List<T> at the specified index.

1.14.
1.14.1.
//List<T>.Remove(T item)
Removes the first occurrence of a specific object from the List<T>.
1.14.2.
//List<T>.RemoveAt(int index)
Removes the element at the specified index of the List<T>.
1.14.3.
//List<T>.RemoveRange(int index, int count)
From "index" postion and take "count" elements, then remove them from the list.

1.15.
//List<T>.Clear()
Removes all elements from the List<T>.

1.16.
//List<T>.Reverse()
Reverses the order of the elements in the entire List<T>.

1.17
//List<T>.TrueForAll(Predicate<T> match)
Determines whether every element in the List<T>
matches the conditions defined by the specified predicate.

1.18.
//List<T>.AsReadOnly()
Returns a read-only ReadOnlyCollection<T> wrapper for the current collection.
It means user can not Add() or Remove() any more.

1.19.
//List<T>.TrimExcess()
Sets the capacity to the actual number of elements in the List<T>, if that number is less than a threshold value.

2.
Value Type Sort

2.1.
//List<T>.Sort()
Sorts the elements in the entire List<T> using the default comparer.

2.2.
//List<T>.Reverse()
Reverses the order of the elements in the entire List<T>.

3.
Reference Type Sort

3.1.
//List<T>.Sort()
Sorts the elements in the entire List<T> using the default comparer.

Need to implement IComparable<T> and CompareTo() before using Sort().
//public class ClassA : IComparable<ClassA>
...
//public int CompareTo(ClassA other){...the compare logic...}
if CompareTo()==0,
it means current instance object is equal to "other" instance object.
if CompareTo()>0,
it means current instance object > "other" instance object.
if CompareTo()<0,
it means current instance object < "other" instance object.
...
//listClassA.Sort();
Sort the listClassA

3.2.
//List<T>.Sort(IComparer<T> comparer)
Sorts the elements in the entire List<T> using the specified comparer.

If you are not allowed to change ClassA code.
Then you care create other class to implement IComparer

```
//public class SortByName : IComparer<T>
//{
//    public int Compare(T currentObj, T otherObj)
//    {
//        //...Compare Logic...
//    }
//}
...
//listCutomers.Sort(new SortByName());
```

3.3.
//List<T>.Sort(Comparison<T> comparison)
Sorts the elements in the entire List<T> using the specified System.Comparison<T>.

3.3.1.
//private static int CompareClassA(ClassA c1, ClassB c2)
//{
//   return c1.ID.CompareTo(c2.ID);
//}
Create a function whose signature matches the signature of System.Comparison delegate.
which is
//public void Sort(Comparison<T> comparison)
...
//Comparison<ClassA> classAComparer = new Comparison<ClassA>(CompareClassA);
Create an instance of System.Comparison delegate and point to CompareClassA method.
...
//listClassA.Sort(classAComparer);
Pass the delegate to sort parameter.

3.3.2.
//listClassA.Sort(delegate(ClassA c1, ClassB c2)
//{
//   return (c1.ID.CompareTo(c2.ID));
//});

3.3.3.
//listClassA.Sort((x, y) => x.ID.CompareTo(y.ID));

==

1. T020_ListSimpleType

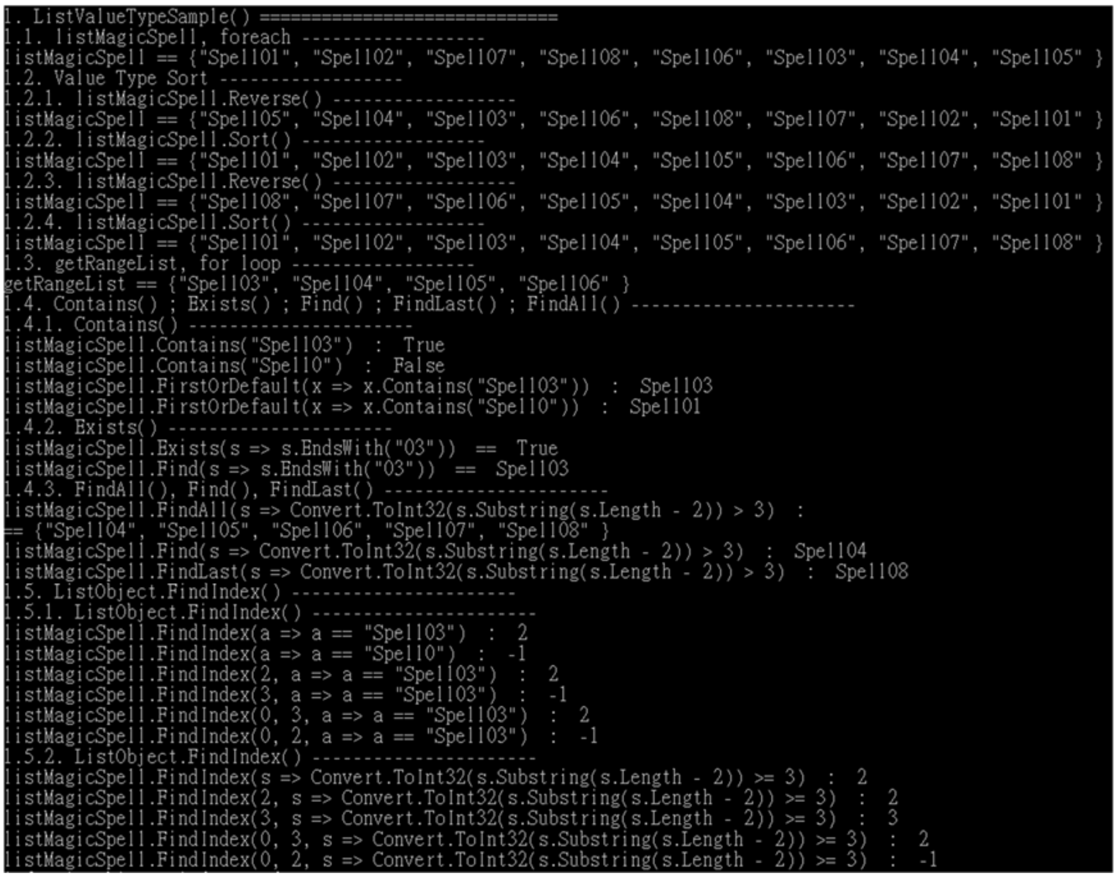
```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using OnLineGame;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1. ----------------------------------------------------
            Console.WriteLine("1. ListSimpleTypeSample() ==============================");
            ListSimpleTypeSample();
            // 2. ----------------------------------------------------
            Console.WriteLine("2. ListReferenceTypeSample() ==============================");
            ListReferenceTypeSample();
```

```csharp
    // 3. ----------------------------------------------------
    Console.WriteLine("3. ListReferenceTypeSortSample() ==============================");
    ListReferenceTypeSortSample();
    Console.ReadLine();
}

// 1. ==============================================================
static void ListSimpleTypeSample()
{
    // 1.0. -------------------------------------------------------
    //Add 2 values to List<T> when initialize.
    //1.11.
    //1.11.1.
    ////List<T>.Add(T item)
    //Adds an object to the end of the List<T>.
    //1.11.2.
    ////List<T>.AddRange(IEnumerable<T> collection)
    //Adds the elements of the specified collection to the end of the List<T>.
    //1.13.
    //1.13.1.
    ////List<T>.Insert(int index, T item)
    //Inserts an element into the List<T> at the specified index.
    //1.13.2.
    ////List<T>.InsertRange Method (int index, IEnumerable<T> collection)
    //Inserts the elements of a collection into the List<T> at the specified index.
    List<string> listMagicSpell = new List<string>
    {
        "Spell01", "Spell02"
    };
    listMagicSpell.Add("Spell03");
    listMagicSpell.AddRange(new List<string>
    {
        "Spell04", "Spell05"
    });
    listMagicSpell.Insert(2, "Spell06");
    listMagicSpell.InsertRange(2, new List<string>
    {
        "Spell07", "Spell08"
    });
    //"Spell01", "Spell02", "Spell07", "Spell08", "Spell06", "Spell03", "Spell04", "Spell05"
    // 1.1. -------------------------------------------------------
    //List of Magic Spell, foreach
    Console.WriteLine("1.1. listMagicSpell, foreach ------------------ ");
    Console.Write("listMagicSpell == {");
    foreach (string magicSpell in listMagicSpell)
    {
        if (magicSpell == listMagicSpell.Last())
        {
            Console.WriteLine($"\"{magicSpell}\"" + " }");
        }
        else
        {
            Console.Write($"\"{magicSpell}\", ");
        }
    }
    //1.1. listMagicSpell, foreach ------------------
```

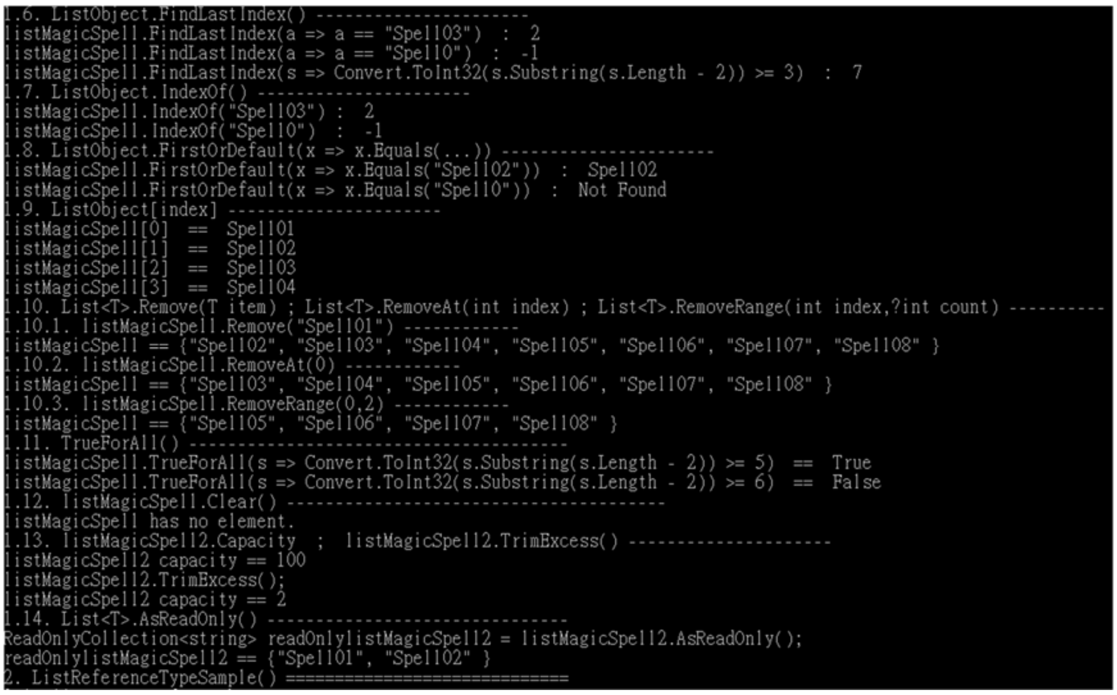
```
            //listMagicSpell == {"Spell01", "Spell02", "Spell07", "Spell08", "Spell06", "Spell03", 
"Spell04", "Spell05" }
            // 1.2. --------------------------------------------------------
            //Value Type Sort
            Console.WriteLine("1.2. Value Type Sort ------------------ ");
            //2.
            //Value Type Sort
            //----------------------
            //2.1.
            ////List<T>.Sort()
            //Sorts the elements in the entire List < T > using the default comparer.
            //   ----------------------
            //2.2.
            ////List<T>.Reverse()
            //Reverses the order of the elements in the entire List<T>.
            Console.WriteLine("1.2.1. listMagicSpell.Reverse() ------------------ ");
            listMagicSpell.Reverse();
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
            {
                if (magicSpell == listMagicSpell.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }

            Console.WriteLine("1.2.2. listMagicSpell.Sort() ------------------ ");
            listMagicSpell.Sort();
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
            {
                if (magicSpell == listMagicSpell.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }
            Console.WriteLine("1.2.3. listMagicSpell.Reverse() ------------------ ");
            listMagicSpell.Reverse();
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
            {
                if (magicSpell == listMagicSpell.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }
            Console.WriteLine("1.2.4. listMagicSpell.Sort() ------------------ ");
            listMagicSpell.Sort();
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
            {
                if (magicSpell == listMagicSpell.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }
            //1.2.Value Type Sort ------------------
            //1.2.1.listMagicSpell.Reverse()------------------
            //listMagicSpell == { "Spell05", "Spell04", "Spell03", "Spell06", "Spell08", "Spell07", 
"Spell02", "Spell01" }
            //1.2.2.listMagicSpell.Sort()------------------
            //listMagicSpell == { "Spell01", "Spell02", "Spell03", "Spell04", "Spell05", "Spell06", 
"Spell07", "Spell08" }
            //1.2.3. listMagicSpell.Reverse() ------------------
            //listMagicSpell == { "Spell08", "Spell07", "Spell06", "Spell05", "Spell04", "Spell03", 
"Spell02", "Spell01" }
            //1.2.4. listMagicSpell.Sort() ------------------
            //listMagicSpell == { "Spell01", "Spell02", "Spell03", "Spell04", "Spell05", "Spell06", 
"Spell07", "Spell08" }
            // 1.3. --------------------------------------------------------
            //List of Magic Spell, for loop

           //1.12.
            ////List<T>.GetRange(int index, int count)
            //From the "index", and take "count" elements, then return a List<T>
            List<string> getRangeList = listMagicSpell.GetRange(2, 4);
            Console.WriteLine("1.3. getRangeList, for loop ------------------ ");
            Console.Write("getRangeList == {");
            for (int index = 0; index < getRangeList.Count; index++)
            {
                string magicSpell = getRangeList[index];
                if (magicSpell == getRangeList.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }
            //1.3. getRangeList, for loop ------------------
            //getRangeList == { "Spell03", "Spell04", "Spell05", "Spell06" }
            // 1.4. --------------------------------------------------------
            Console.WriteLine("1.4. Contains() ; Exists() ; Find() ; FindLast() ; FindAll() --------------
-------- ");
            //1.1.
            ////List<T>.Contains(T item)
            //Returns true if the items exists in the list, else false.
```

```
//1.2.
////List<T>.Exists(Predicate<T> match)
//Use the conditions defined by the lambda expression
//to search for the element, then return true
//if first matching item is existed, otherwise return false.
//1.3.
////List<T>.Find(Predicate<T> match)
//Use the conditions defined by the lambda expression
//to search for the element, then return the first matching item.
//1.4.
////List<T>.FindLast(Predicate<T> match)
//Use the conditions defined by the lambda expression
//to search for the element, then return the last matching item.
//1.5.
////List<T>.FindAll(Predicate<T>)
//Use the conditions defined by the lambda expression
//to search for the element, then return the last matching all of them.
//1.4.1. Contains() --------------------------------
Console.WriteLine("1.4.1. Contains() ---------------------- ");
bool containsSpell3 = listMagicSpell.Contains("Spell03");
Console.WriteLine($"listMagicSpell.Contains(\"Spell03\")  :  " +
                  $"{containsSpell3}"); //True
bool containsSpell0 = listMagicSpell.Contains("Spell0");
Console.WriteLine($"listMagicSpell.Contains(\"Spell0\")  :  " +
                  $"{containsSpell0}"); //False
string firstContainSpell03 = listMagicSpell.FirstOrDefault(x => x.Contains("Spell03"));
Console.WriteLine($"listMagicSpell.FirstOrDefault(x => x.Contains(\"Spell03\"))  :  " +
                  $"{firstContainSpell03}");    //Spell03
string firstContainSpell0 = listMagicSpell.FirstOrDefault(x => x.Contains("Spell0"));
Console.WriteLine($"listMagicSpell.FirstOrDefault(x => x.Contains(\"Spell0\"))  :  " +
                  $"{firstContainSpell0}"); //Spell01
//1.4.1. Contains() ----------------------
//listMagicSpell.Contains("Spell03")  :  True
//listMagicSpell.Contains("Spell0")  :  False
//listMagicSpell.FirstOrDefault(x => x.Contains("Spell03"))  :  Spell03
//listMagicSpell.FirstOrDefault(x => x.Contains("Spell0"))  :  Spell01


//1.4.2. Exists() --------------------------------
Console.WriteLine("1.4.2. Exists() ---------------------- ");
bool existsEndWith03 = listMagicSpell.Exists(s => s.EndsWith("03"));
Console.WriteLine($"listMagicSpell.Exists(s => s.EndsWith(\"03\"))  ==  " +
                  $"{existsEndWith03}");
string findEndWith03 = listMagicSpell.Find(s => s.EndsWith("03"));
Console.WriteLine($"listMagicSpell.Find(s => s.EndsWith(\"03\"))  ==  " +
                  $"{findEndWith03}");
//1.3.2. Exists() ----------------------
//listMagicSpell.Exists(s => s.EndsWith("03")) == True
//listMagicSpell.Find(s => s.EndsWith("03")) == Spell03
//1.4.3. FindAll(), Find(), FindLast() --------------------------------
Console.WriteLine("1.4.3. FindAll(), Find(), FindLast() ---------------------- ");
//Convert last 2 characters from the string to integer.
//if that integer > 3,
        //then add to List<T> and return the List<T>.
        List<string> findAllGreaterThan3 = listMagicSpell.FindAll(
             s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3);
        Console.WriteLine($"listMagicSpell.FindAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >
3)  :  ");
        Console.Write("== {");
        foreach (string magicSpell in findAllGreaterThan3)
        {
            if (magicSpell == findAllGreaterThan3.Last())
            {
                Console.WriteLine($"\"{magicSpell}\"" + " }");
            }
            else
            {
                Console.Write($"\"{magicSpell}\", ");
            }
        }
        string findGreaterThan3 = listMagicSpell.Find(
             s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3);
        Console.WriteLine($"listMagicSpell.Find(s => Convert.ToInt32(s.Substring(s.Length - 2)) >
3)  :  " +
                          $"{findGreaterThan3}");
        string findLastGreaterThan3 = listMagicSpell.FindLast(
             s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3);
        Console.WriteLine($"listMagicSpell.FindLast(s => Convert.ToInt32(s.Substring(s.Length - 2)) >
3)  :  " +
                          $"{findLastGreaterThan3}");
        //1.4.3. FindAll(), Find(), FindLast() ----------------------
        //listMagicSpell.FindAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :
        //== {"Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
        //listMagicSpell.Find(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :  Spell04
        //listMagicSpell.FindLast(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :  Spell08


       // 1.5. -------------------------------
        //ListObject.FindIndex()
        Console.WriteLine("1.5. ListObject.FindIndex() ---------------------- ");
        //1.6.
        //1.6.1.
        ////List<T>.FindIndex(Predicate<T> match)
        //Use the conditions defined by the lambda expression
        //to search for the index of the element,
        //then return the first matching index.
        //Return - 1 if the index can not be found.
        //1.6.2.
        ////List<T>.FindIndex(int startIndex, Predicate<T> match)
        //From the startIndex position,
        //use the conditions defined by the lambda expression
        //to search for the index of the element,
        //then return the first matching index.
        //1.6.3.
        ////List<T>.FindIndex(int startIndex, int count, Predicate<T> match)
        //From the "startIndex" position until "count" numbers of elements,
        //Use the conditions defined by the lambda expression
```

```
            //to search for the index of the element,
            //then return the first matching index.
            Console.WriteLine("1.5.1. ListObject.FindIndex() ---------------------- ");
            int indexOfSpell03 = listMagicSpell.FindIndex(a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindIndex(a => a == \"Spell03\")  :  " +
                              $"{indexOfSpell03}");
            int indexOfSpell0 = listMagicSpell.FindIndex(a => a == "Spell0");
            Console.WriteLine($"listMagicSpell.FindIndex(a => a == \"Spell0\")  :  " +
                              $"{indexOfSpell0}");
            int indexOfSpell03V2 = listMagicSpell.FindIndex(2, a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindIndex(2, a => a == \"Spell03\")  :  " +
                              $"{indexOfSpell03V2}");
            int indexOfSpell03V3 = listMagicSpell.FindIndex(3, a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindIndex(3, a => a == \"Spell03\")  :  " +
                              $"{indexOfSpell03V3}");
            int indexOfSpell03V4 = listMagicSpell.FindIndex(0, 3, a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindIndex(0, 3, a => a == \"Spell03\")  :  " +
                              $"{indexOfSpell03V4}");
            int indexOfSpell03V5 = listMagicSpell.FindIndex(0, 2, a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindIndex(0, 2, a => a == \"Spell03\")  :  " +
                              $"{indexOfSpell03V5}");
            //1.5.1. ListObject.FindIndex() ----------------------
            //listMagicSpell.FindIndex(a => a == "Spell03")  :  2
            //listMagicSpell.FindIndex(a => a == "Spell0")  :  -1
            //listMagicSpell.FindIndex(2, a => a == "Spell03")  :  2
            //listMagicSpell.FindIndex(3, a => a == "Spell03")  :  -1
            //listMagicSpell.FindIndex(0, 3, a => a == "Spell03")  :  2
            //listMagicSpell.FindIndex(0, 2, a => a == "Spell03")  :  -1
            Console.WriteLine("1.5.2. ListObject.FindIndex() ---------------------- ");
            int indexOfSpell03Sub1 = listMagicSpell.FindIndex(s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindIndex(s => Convert.ToInt32(s.Substring(s.Length - 2))
>= 3)  :  " +
                              $"{indexOfSpell03Sub1}");
            int indexOfSpell03Sub2 = listMagicSpell.FindIndex(2, s => Convert.ToInt32(s.Substring(s.Length
- 2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindIndex(2, s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3)  :  " +
                              $"{indexOfSpell03Sub2}");
            int indexOfSpell03Sub3 = listMagicSpell.FindIndex(3, s => Convert.ToInt32(s.Substring(s.Length
- 2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindIndex(3, s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3)  :  " +
                              $"{indexOfSpell03Sub3}");
            int indexOfSpell03Sub4 = listMagicSpell.FindIndex(0, 3, s
=> Convert.ToInt32(s.Substring(s.Length - 2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindIndex(0, 3, s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3)  :  " +
                              $"{indexOfSpell03Sub4}");
            int indexOfSpell03Sub5 = listMagicSpell.FindIndex(0, 2, s
=> Convert.ToInt32(s.Substring(s.Length - 2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindIndex(0, 2, s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3)  :  " +
                  $"{indexOfSpell03Sub5}");
            //1.5.2.ListObject.FindIndex()----------------------
            //listMagicSpell.FindIndex(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
            //listMagicSpell.FindIndex(2, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
            //listMagicSpell.FindIndex(3, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  3
            //listMagicSpell.FindIndex(0, 3, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
            //listMagicSpell.FindIndex(0, 2, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  -1


           // 1.6. -------------------------------
            //ListObject.FindLastIndex()
            Console.WriteLine("1.6. ListObject.FindLastIndex() ---------------------- ");
            //1.7.
            //1.7.1.
            ////List<T>.FindLastIndex(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the index of the element,
            //then return the last matching index.
            //Return - 1 if the index can not be found.
            int findLastIndexOfSpell03 = listMagicSpell.FindLastIndex(a => a == "Spell03");
            Console.WriteLine($"listMagicSpell.FindLastIndex(a => a == \"Spell03\")  :  " +
                  $"{findLastIndexOfSpell03}");
            int findLastIndexOfSpell0 = listMagicSpell.FindLastIndex(a => a == "Spell0");
            Console.WriteLine($"listMagicSpell.FindLastIndex(a => a == \"Spell0\")  :  " +
                  $"{findLastIndexOfSpell0}");
            int findLastIndexOfSpell03Sub1 = listMagicSpell.FindLastIndex(s
=> Convert.ToInt32(s.Substring(s.Length - 2)) >= 3);
            Console.WriteLine($"listMagicSpell.FindLastIndex(s => Convert.ToInt32(s.Substring(s.Length -
2)) >= 3)  :  " +
                  $"{findLastIndexOfSpell03Sub1}");
            //1.6.ListObject.FindLastIndex()----------------------
            //listMagicSpell.FindLastIndex(a => a == "Spell03")  :  2
            //listMagicSpell.FindLastIndex(a => a == "Spell0")  :  -1
            //listMagicSpell.FindLastIndex(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  7
            // 1.7. -------------------------------
            //ListObject.IndexOf()
            Console.WriteLine("1.7. ListObject.IndexOf() ---------------------- ");
            int indexOfSpell03_2 = listMagicSpell.IndexOf("Spell03");
            Console.WriteLine($"listMagicSpell.IndexOf(\"Spell03\") :  " +
                  $"{indexOfSpell03_2}");
            int indexOfSpell0_2 = listMagicSpell.IndexOf("Spell0");
            Console.WriteLine($"listMagicSpell.IndexOf(\"Spell0\")  :  " +
                  $"{indexOfSpell0_2}");
            //1.7. ListObject.IndexOf() ----------------------
            //listMagicSpell.IndexOf("Spell03") :  2
            //listMagicSpell.IndexOf("Spell0")  :  -1
            // 1.8. -------------------------------
            //ListObject.FirstOrDefault(x => x.Equals(...))
            Console.WriteLine("1.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
");
            string firstEqualsSpell02 = listMagicSpell.FirstOrDefault(x => x.Equals("Spell02"));
            Console.WriteLine($"listMagicSpell.FirstOrDefault(x => x.Equals(\"Spell02\"))  :  " +
                   $"{firstEqualsSpell02}");
            string firstEqualsSpell0 =
                listMagicSpell.FirstOrDefault(x => x.Equals("Spell0")) == null ?
                "Not Found" :
                listMagicSpell.FirstOrDefault(x => x.Equals("Spell0"));
            Console.WriteLine($"listMagicSpell.FirstOrDefault(x => x.Equals(\"Spell0\"))  :  " +
                              $"{firstEqualsSpell0}");
            //1.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
            //listMagicSpell.FirstOrDefault(x => x.Equals("Spell02"))  :  Spell02
            //listMagicSpell.FirstOrDefault(x => x.Equals("Spell0"))  :  Not Found



            // 1.9. ------------------------------
            //ListObject[index]
            Console.WriteLine("1.9. ListObject[index] ---------------------- ");
            string magicSpell0 = listMagicSpell[0]; // get index 0 value string.
            Console.WriteLine($"listMagicSpell[0]  ==  {listMagicSpell[0]}");
            Console.WriteLine($"listMagicSpell[1]  ==  {listMagicSpell[1]}");
            Console.WriteLine($"listMagicSpell[2]  ==  {listMagicSpell[2]}");
            Console.WriteLine($"listMagicSpell[3]  ==  {listMagicSpell[3]}");
            //Console.WriteLine($"listMagicSpell[4]  ==  {listMagicSpell[4]}");   // Error! Throw Example.
            //1.9. ListObject[index] ----------------------
            //listMagicSpell[0] == Spell01
            //listMagicSpell[1] == Spell02
            //listMagicSpell[2] == Spell03
            //listMagicSpell[3] == Spell04
            // 1.10. ------------------------------
            //List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ; List<T>.RemoveRange(int index, int
count)
            Console.WriteLine("1.10. List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ;
List<T>.RemoveRange(int index, int count) ----------");
            //1.14.
            //1.14.1.
            ////List<T>.Remove(T item)
            //Removes the first occurrence of a specific object from the List<T>.
            //1.14.2.
            ////List<T>.RemoveAt(int index)
            //Removes the element at the specified index of the List<T>.
            //1.14.3.
            ////List<T>.RemoveRange Method (int index, int count)
            //From "index" postion and take "count" elements, then remove them from the list.
            Console.WriteLine("1.10.1. listMagicSpell.Remove(\"Spell01\") ------------");
             listMagicSpell.Remove("Spell01");   //Remove "Spell01"
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
             {
                if (magicSpell == listMagicSpell.Last())
                 {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                 }
                else
                 {
                    Console.Write($"\"{magicSpell}\", ");
                 }
             }
            Console.WriteLine("1.10.2. listMagicSpell.RemoveAt(0) ------------");
             listMagicSpell.RemoveAt(0);   //Remove index 0 postion item
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
             {
                if (magicSpell == listMagicSpell.Last())
                 {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                 }
                else
                 {
                    Console.Write($"\"{magicSpell}\", ");
                 }
             }
            Console.WriteLine("1.10.3. listMagicSpell.RemoveRange(0,2) ------------");
             listMagicSpell.RemoveRange(0, 2);   //Remove 2 items from index 0 postion
            Console.Write("listMagicSpell == {");
            foreach (string magicSpell in listMagicSpell)
             {
                if (magicSpell == listMagicSpell.Last())
                 {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                 }
                else
                 {
                    Console.Write($"\"{magicSpell}\", ");
                 }
             }
            //1.10.1.listMagicSpell.Remove("Spell01")------------
            //listMagicSpell == { "Spell02", "Spell03", "Spell04", "Spell05", "Spell06", "Spell07", 
"Spell08" }
            //1.10.2.listMagicSpell.RemoveAt(0)------------
            //listMagicSpell == { "Spell03", "Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
            //1.10.3.listMagicSpell.RemoveRange(0, 2)------------
            //listMagicSpell == { "Spell05", "Spell06", "Spell07", "Spell08" }

           // 1.11. -------------------------------
            //TrueForAll()
            Console.WriteLine("1.11. TrueForAll() ---------------------------------------");
            //1.17
            ////List<T>.TrueForAll(Predicate<T> match)
            //Determines whether every element in the List<T>
            //matches the conditions defined by the specified predicate.
            bool trueForAll1 = listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 
5);
            Console.WriteLine($"listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) 
>= 5)  ==  {trueForAll1}");
            bool trueForAll2 = listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 
6);
            Console.WriteLine($"listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) 
>= 6)  ==  {trueForAll2}");
            //1.11.TrueForAll()---------------------------------------
            //listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 5) == True
            //listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 6) == False
            // 1.12. -------------------------------
            Console.WriteLine("1.12. listMagicSpell.Clear() ---------------------------------------");
             listMagicSpell.Clear();   //Remove all elements.
            if (listMagicSpell.Count > 0)
            {
                Console.Write("listMagicSpell == {");
                foreach (string magicSpell in listMagicSpell)
                {
                    if (magicSpell == listMagicSpell.Last())
                    {
                        Console.WriteLine($"\"{magicSpell}\"" + " }");
                    }
                    else
                    {
                        Console.Write($"\"{magicSpell}\", ");
                    }
                }
            }
            else
            {
                Console.WriteLine("listMagicSpell has no element.");
            }
            //1.12.listMagicSpell.Clear()-------------------------------------- -
            //listMagicSpell has no element.
            // 1.13. -------------------------------
            //listMagicSpell2.Capacity  ;  listMagicSpell2.TrimExcess()
            Console.WriteLine("1.13. listMagicSpell2.Capacity  ;  listMagicSpell2.TrimExcess() -----------
----------");
            List<string> listMagicSpell2 = new List<string>(100);
            listMagicSpell2.AddRange(new List<string> { "Spell01", "Spell02" });
            Console.WriteLine("listMagicSpell2 capacity == " +
                listMagicSpell2.Capacity);
            Console.WriteLine("listMagicSpell2.TrimExcess();");
            // TrimExcess() set the capacity to the number of elements in the List
            listMagicSpell2.TrimExcess();
            Console.WriteLine("listMagicSpell2 capacity == " +
                listMagicSpell2.Capacity);
            //1.13.listMagicSpell2.Capacity; listMagicSpell2.TrimExcess()-------------------- -
            //listMagicSpell2 capacity == 100
            //listMagicSpell2.TrimExcess();
            //listMagicSpell2 capacity == 2
            // 1.14. -------------------------------
            //List<T>.AsReadOnly()
            Console.WriteLine("1.14. List<T>.AsReadOnly() -------------------------------");
            Console.WriteLine("ReadOnlyCollection<string> readOnlylistMagicSpell2 =
listMagicSpell2.AsReadOnly();");
            ReadOnlyCollection<string> readOnlylistMagicSpell2 = listMagicSpell2.AsReadOnly();
            // readOnlylistMagicSpell2 can not Add() or Remove any more.
            // readOnlylistMagicSpell2 can only be read now.
            Console.Write("readOnlylistMagicSpell2 == {");
            foreach (string magicSpell in readOnlylistMagicSpell2)
            {
                if (magicSpell == readOnlylistMagicSpell2.Last())
                {
                    Console.WriteLine($"\"{magicSpell}\"" + " }");
                }
                else
                {
                    Console.Write($"\"{magicSpell}\", ");
                }
            }
            //1.14.List<T>.AsReadOnly()------------------------------ -
            //ReadOnlyCollection < string > readOnlylistMagicSpell2 = listMagicSpell2.AsReadOnly();
            //readOnlylistMagicSpell2 == { "Spell01", "Spell02" }
        }
    }
}
```



```
1. ListValueTypeSample() ==============================
1.1. listMagicSpell, foreach ------------------
listMagicSpell == {"Spell01", "Spell02", "Spell07", "Spell08", "Spell06", "Spell03", "Spell04", "Spell05" }
1.2. Value Type Sort ------------------
1.2.1. listMagicSpell.Reverse() ------------------
listMagicSpell == {"Spell05", "Spell04", "Spell03", "Spell06", "Spell08", "Spell07", "Spell02", "Spell01" }
1.2.2. listMagicSpell.Sort() ------------------
listMagicSpell == {"Spell01", "Spell02", "Spell03", "Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
1.2.3. listMagicSpell.Reverse() ------------------
listMagicSpell == {"Spell08", "Spell07", "Spell06", "Spell05", "Spell04", "Spell03", "Spell02", "Spell01" }
1.2.4. listMagicSpell.Sort() ------------------
listMagicSpell == {"Spell01", "Spell02", "Spell03", "Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
1.3. getRangeList, for loop ------------------
getRangeList == {"Spell03", "Spell04", "Spell05", "Spell06" }
1.4. Contains() ; Exists() ; Find() ; FindLast() ; FindAll() ----------------------
1.4.1. Contains() ----------------------
listMagicSpell.Contains("Spell03")  :  True
listMagicSpell.Contains("Spell0")  :  False
listMagicSpell.FirstOrDefault(x => x.Contains("Spell03"))  :  Spell03
listMagicSpell.FirstOrDefault(x => x.Contains("Spell0"))  :  Spell01
1.4.2. Exists() ----------------------
listMagicSpell.Exists(s => s.EndsWith("03"))  ==  True
listMagicSpell.Find(s => s.EndsWith("03"))  ==  Spell03
1.4.3. FindAll(), Find(), FindLast() ----------------------
listMagicSpell.FindAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :
== {"Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
listMagicSpell.Find(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :  Spell04
listMagicSpell.FindLast(s => Convert.ToInt32(s.Substring(s.Length - 2)) > 3)  :  Spell08
1.5. ListObject.FindIndex() ----------------------
1.5.1. ListObject.FindIndex() ----------------------
listMagicSpell.FindIndex(a => a == "Spell03")  :  2
listMagicSpell.FindIndex(a => a == "Spell0")  :  -1
listMagicSpell.FindIndex(2, a => a == "Spell03")  :  2
listMagicSpell.FindIndex(3, a => a == "Spell03")  :  -1
listMagicSpell.FindIndex(0, 3, a => a == "Spell03")  :  2
listMagicSpell.FindIndex(0, 2, a => a == "Spell03")  :  -1
1.5.2. ListObject.FindIndex() ----------------------
listMagicSpell.FindIndex(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
listMagicSpell.FindIndex(2, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
listMagicSpell.FindIndex(3, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  3
listMagicSpell.FindIndex(0, 3, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  2
listMagicSpell.FindIndex(0, 2, s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  -1
```



```
1.6. ListObject.FindLastIndex() ----------------------
listMagicSpell.FindLastIndex(a => a == "Spell03")  :  2
listMagicSpell.FindLastIndex(a => a == "Spell0")  :  -1
listMagicSpell.FindLastIndex(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 3)  :  7
1.7. ListObject.IndexOf() ----------------------
listMagicSpell.IndexOf("Spell03") :  2
listMagicSpell.IndexOf("Spell0")  :  -1
1.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
listMagicSpell.FirstOrDefault(x => x.Equals("Spell02"))  :  Spell02
listMagicSpell.FirstOrDefault(x => x.Equals("Spell0"))  :  Not Found
1.9. ListObject[index] ----------------------
listMagicSpell[0]  ==  Spell01
listMagicSpell[1]  ==  Spell02
listMagicSpell[2]  ==  Spell03
listMagicSpell[3]  ==  Spell04
1.10. List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ; List<T>.RemoveRange(int index,?int count) ----------
1.10.1. listMagicSpell.Remove("Spell01") ------------
listMagicSpell == {"Spell02", "Spell03", "Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
1.10.2. listMagicSpell.RemoveAt(0) ------------
listMagicSpell == {"Spell03", "Spell04", "Spell05", "Spell06", "Spell07", "Spell08" }
1.10.3. listMagicSpell.RemoveRange(0,2) ------------
listMagicSpell == {"Spell05", "Spell06", "Spell07", "Spell08" }
1.11. TrueForAll() ----------------------------------------
listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 5)  ==  True
listMagicSpell.TrueForAll(s => Convert.ToInt32(s.Substring(s.Length - 2)) >= 6)  ==  False
1.12. listMagicSpell.Clear() ----------------------------------------
listMagicSpell has no element.
1.13. listMagicSpell2.Capacity  ;  listMagicSpell2.TrimExcess() ---------------------
listMagicSpell2 capacity == 100
listMagicSpell2.TrimExcess();
listMagicSpell2 capacity == 2
1.14. List<T>.AsReadOnly() ------------------------------
ReadOnlyCollection<string> readOnlylistMagicSpell2 = listMagicSpell2.AsReadOnly();
readOnlylistMagicSpell2 == {"Spell01", "Spell02" }
2. ListReferenceTypeSample() ============================
```

==

2. T021_ListReferenceType

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using OnLineGame;


namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 2. ----------------------------------------------------
            Console.WriteLine("2. ListReferenceTypeSample() =============================");
             ListReferenceTypeSample();
        }


       // 2. ==============================================================
        static void ListReferenceTypeSample()
        {
            // 2.0. -------------------------------------------------------
            //Add 2 values to List<T> when initialize.
            //1.11.
            //1.11.1.
            ////List<T>.Add(T item)
            //Adds an object to the end of the List<T>.
            //1.11.2.
            ////List<T>.AddRange(IEnumerable<T> collection)
            //Adds the elements of the specified collection to the end of the List<T>.
            //1.13.
            //1.13.1.
            ////List<T>.Insert(int index, T item)
            //Inserts an element into the List<T> at the specified index.
            //1.13.2.
            ////List<T>.InsertRange Method (int index, IEnumerable<T> collection)
            //Inserts the elements of a collection into the List<T> at the specified index.
            List<GamerA> listGamerAs = new List<GamerA>
            {
                new GamerA { Id = 1, Name = "Name01", Email = "1@1.com" },
                new GamerA { Id = 2, Name = "Name02", Email = "2@2.com" },
                new GamerA { Id = 3, Name = "Name03", Email = "3@3.com" },
                new GamerA { Id = 4, Name = "Name04", Email = "4@4.com" }
            };
            List<Gamer> listGamers = new List<Gamer>
            {
                new Gamer { Id = 1, Name = "Name01", Email = "1@1.com" },
                new Gamer { Id = 2, Name = "Name02", Email = "2@2.com" }
            };
    listGamers.Add(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" });
    listGamers.AddRange(new List<Gamer>
    {
        new Gamer { Id = 4, Name = "Name04", Email = "4@4.com" },
        new Gamer { Id = 5, Name = "Name05", Email = "5@5.com" }
    });
    listGamers.Insert(2, new Gamer { Id = 6, Name = "Name06", Email = "6@6.com" });
    listGamers.InsertRange(2, new List<Gamer>
    {
        new Gamer { Id = 7, Name = "Name07", Email = "7@7.com" },
        new Gamer { Id = 8, Name = "Name08", Email = "8@8.com" }
    });
   // 2.1. --------------------------------------------------------
   //List of Gamers, foreach
   Console.WriteLine("2.1. listGamers, foreach ------------------ ");
   Console.Write("listGamers == {");
   foreach (Gamer gamerItem in listGamers)
    {
        if (gamerItem == listGamers.Last())
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
   //2.1. listGamers, foreach ------------------
   //listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
   //"Id == 2 ; Name == Name02 ; Email : 2@2.com",
   //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
   //"Id == 8 ; Name == Name08 ; Email : 8@8.com",
   //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
   //"Id == 3 ; Name == Name03 ; Email : 3@3.com",
   //"Id == 4 ; Name == Name04 ; Email : 4@4.com",
   //"Id == 5 ; Name == Name05 ; Email : 5@5.com" }


// 2.2. --------------------------------------------------------
   //Value Type Sort
   Console.WriteLine("2.2. Reference Type Sort ------------------ ");
   //3.3.3.
   //listClassA.Sort((x, y) => x.ID.CompareTo(y.ID));
   Console.WriteLine("2.2.1. listGamers.Reverse() ------------------ ");
    listGamers.Reverse();
   Console.Write("listGamers == {");
   foreach (Gamer gamerItem in listGamers)
    {
        if (gamerItem == listGamers.Last())
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
```

```
 }
//2.2.1. listGamers.Reverse() ------------------
//listGamers == {"Id == 5 ; Name == Name05 ; Email : 5@5.com",
//"Id == 4 ; Name == Name04 ; Email : 4@4.com",
//"Id == 3 ; Name == Name03 ; Email : 3@3.com",
//"Id == 6 ; Name == Name06 ; Email : 6@6.com",
//"Id == 8 ; Name == Name08 ; Email : 8@8.com",
//"Id == 7 ; Name == Name07 ; Email : 7@7.com",
//"Id == 2 ; Name == Name02 ; Email : 2@2.com",
//"Id == 1 ; Name == Name01 ; Email : 1@1.com" }
Console.WriteLine("2.2.2. listGamers.Sort() ------------------ ");
//listGamers.Sort();  //Error, because it doesn't know how to sort Gamer.
 listGamers.Sort((g1, g2) => g1.Id.CompareTo(g2.Id));  // It will sort by Id.
Console.Write("listGamers == {");
foreach (Gamer gamerItem in listGamers)
 {
    if (gamerItem == listGamers.Last())
     {
        Console.WriteLine($"\"{gamerItem}\"" + " }");
     }
    else
     {
        Console.WriteLine($"\"{gamerItem}\", ");
     }
 }
//2.2.2. listGamers.Sort() ------------------
//listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
//"Id == 2 ; Name == Name02 ; Email : 2@2.com",
//"Id == 3 ; Name == Name03 ; Email : 3@3.com",
//"Id == 4 ; Name == Name04 ; Email : 4@4.com",
//"Id == 5 ; Name == Name05 ; Email : 5@5.com",
//"Id == 6 ; Name == Name06 ; Email : 6@6.com",
//"Id == 7 ; Name == Name07 ; Email : 7@7.com",
//"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
Console.WriteLine("2.2.3. listGamers.Reverse() ------------------ ");
 listGamers.Reverse();
Console.Write("listGamers == {");
foreach (Gamer gamerItem in listGamers)
 {
    if (gamerItem == listGamers.Last())
     {
        Console.WriteLine($"\"{gamerItem}\"" + " }");
     }
    else
     {
        Console.WriteLine($"\"{gamerItem}\", ");
     }
 }
//2.2.3. listGamers.Reverse() ------------------
//listGamers == {"Id == 8 ; Name == Name08 ; Email : 8@8.com",
//"Id == 7 ; Name == Name07 ; Email : 7@7.com",
//"Id == 6 ; Name == Name06 ; Email : 6@6.com",
//"Id == 5 ; Name == Name05 ; Email : 5@5.com",
//"Id == 4 ; Name == Name04 ; Email : 4@4.com",
//"Id == 3 ; Name == Name03 ; Email : 3@3.com",
//"Id == 2 ; Name == Name02 ; Email : 2@2.com",
            //"Id == 1 ; Name == Name01 ; Email : 1@1.com" }


          Console.WriteLine("2.2.4. listGamers.Sort() ------------------ ");
            //listGamers.Sort();  //Error, because it doesn't know how to sort Gamer.
             listGamers.Sort((g1, g2) => g1.Id.CompareTo(g2.Id));  // It will sort by Id.
            Console.Write("listGamers == {");
            foreach (Gamer gamerItem in listGamers)
            {
                if (gamerItem == listGamers.Last())
                {
                    Console.WriteLine($"\"{gamerItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerItem}\", ");
                }
            }
            //2.2.4. listGamers.Sort() ------------------
            //listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
            //"Id == 2 ; Name == Name02 ; Email : 2@2.com",
            //"Id == 3 ; Name == Name03 ; Email : 3@3.com",
            //"Id == 4 ; Name == Name04 ; Email : 4@4.com",
            //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
            //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
            //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
            //"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
            // 2.3. --------------------------------------------------------
            //List of Gamers, for loop
            //1.12.
            ////List<T>.GetRange(int index, int count)
            //From the "index", and take "count" elements, then return a List<T>
            List<Gamer> getRangeListGamers = listGamers.GetRange(2, 4);
            Console.WriteLine("2.3. getRangeListGamers, for loop ------------------ ");
            Console.Write("getRangeListGamers == {");
            for (int index = 0; index < getRangeListGamers.Count; index++)
            {
                Gamer gamerItem = getRangeListGamers[index];
                if (gamerItem == getRangeListGamers.Last())
                {
                    Console.WriteLine($"\"{gamerItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerItem}\", ");
                }
            }
            //2.3. getRangeListGamers, for loop ------------------
            //getRangeListGamers == {"Id == 3 ; Name == Name03 ; Email : 3@3.com",
            //"Id == 4 ; Name == Name04 ; Email : 4@4.com",
            //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
            //"Id == 6 ; Name == Name06 ; Email : 6@6.com" }
            // 2.4. --------------------------------------------------------
            Console.WriteLine("2.4. Contains() ; Exists() ; Find() ; FindLast() ; FindAll() --------------
-------- ");
            //2.1.
            ////List<T>.Contains(T item)
            //Returns true if the items exists in the list, else false.
            //2.2.
            ////List<T>.Exists(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the element, then return true
            //if first matching item is existed, otherwise return false.
            //2.3.
            ////List<T>.Find(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the element, then return the first matching item.
            //2.4.
            ////List<T>.FindLast(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the element, then return the last matching item.
            //2.5.
            ////List<T>.FindAll(Predicate<T>)
            //Use the conditions defined by the lambda expression
            //to search for the element, then return the last matching all of them.
            //2.4.1. Contains() -------------------------------
            //listGamers contains a collection of Gamer objects which does not override Equals() and
GetHashCode().
            //listGamerAs contains a collection of GamerA objects which override Equals() and
GetHashCode().
            Console.WriteLine("2.4.1. Contains() ---------------------- ");
            bool containsGamerId3 = listGamers.Contains(new Gamer { Id = 3, Name = "Name03", Email
= "3@3.com" });
            Console.WriteLine($"listGamers.Contains(new Gamer {{ Id = 3, Name = \"Name03\", Email
= \"3@3.com\" }})  :  " +
                              $"{containsGamerId3}"); //False
            bool containsGamerAId3 = listGamerAs.Contains(
                new GamerA { Id = 3, Name = "Name03", Email = "3@3.com" });
            Console.WriteLine($"listGamerAs.Contains(new Gamer {{ Id = 3, Name = \"Name03\", Email
= \"3@3.com\" }})  :  " +
                              $"{containsGamerAId3}");   //True
            bool containsGamerAId0 = listGamerAs.Contains(
                new GamerA { Id = 3, Name = "Name0", Email = "0@0.com" });
            Console.WriteLine($"listGamerAs.Contains(new GamerA {{ Id = 3, Name = \"Name0\", Email
= \"0@0.com\" }})  :  " +
                              $"{containsGamerAId0}");   //False


           //2.4.1. Contains() ----------------------
            //listGamers.Contains(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" })  :  False
            //listGamerAs.Contains(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" })  :  True
            //listGamerAs.Contains(new GamerA { Id = 3, Name = "Name0", Email = "0@0.com" })  :  False
            //2.4.2. Exists() -------------------------------
            Console.WriteLine("2.4.2. Exists() ---------------------- ");
            bool existsId03 = listGamers.Exists(s => s.Id == 3);
            Console.WriteLine($"listGamers.Exists(s => s.Id==3)  :  " +
                              $"{existsId03}");
            Gamer findId03 = listGamers.Find(s => s.Id == 3);
            Console.WriteLine($"listGamers.Find(s => s.Id == 3)  :  " +
                              $"{findId03}");
            //2.4.2. Exists() ----------------------
            //listGamers.Exists(s => s.Id == 3) : True
            //listGamers.Find(s => s.Id == 3) : Id == 3; Name == Name03; Email: 3@3.com
            //2.4.3. FindAll(), Find(), FindLast() --------------------------------
            Console.WriteLine("2.4.3. FindAll(), Find(), FindLast() ---------------------- ");
            //then add to List<T> and return the List<T>.
            List<Gamer> findAllIdGreaterThan3 = listGamers.FindAll(g => g.Id > 3);
            Console.WriteLine($"listGamers.FindAll(g => g.Id > 3)  :  ");
            Console.Write("{");
            foreach (Gamer gamerItem in findAllIdGreaterThan3)
            {
                if (gamerItem == findAllIdGreaterThan3.Last())
                {
                    Console.WriteLine($"\"{gamerItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerItem}\", ");
                }
            }
            Gamer findIdGreaterThan3 = listGamers.Find(g => g.Id > 3);
            Console.WriteLine($"listGamers.Find(g => g.Id > 3)  :  " +
                              $"{findIdGreaterThan3}");
            Gamer findLastIdGreaterThan3 = listGamers.FindLast(g => g.Id > 3);
            Console.WriteLine($"listGamers.FindLast(g => g.Id > 3)  :  " +
                              $"{findLastIdGreaterThan3}");
            //2.4.3. FindAll(), Find(), FindLast() ----------------------
            //listGamers.FindAll(g => g.Id > 3)  :
            //{"Id == 4 ; Name == Name04 ; Email : 4@4.com",
            //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
            //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
            //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
            //"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
            //listGamers.Find(g => g.Id > 3)  :  Id == 4 ; Name == Name04; Email: 4@4.com
            //listGamers.FindLast(g => g.Id > 3)  :  Id == 8; Name == Name08; Email: 8@8.com
            // 2.5. -------------------------------
            //ListObject.FindIndex()
            Console.WriteLine("2.5. ListObject.FindIndex() ---------------------- ");
            //1.6.
            //1.6.1.
            ////List<T>.FindIndex(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the index of the element,
            //then return the first matching index.
            //Return - 1 if the index can not be found.
            //1.6.2.
            ////List<T>.FindIndex(int startIndex, Predicate<T> match)
            //From the startIndex position,
            //use the conditions defined by the lambda expression
            //to search for the index of the element,
            //then return the first matching index.
            //1.6.3.
            ////List<T>.FindIndex(int startIndex, int count, Predicate<T> match)
```

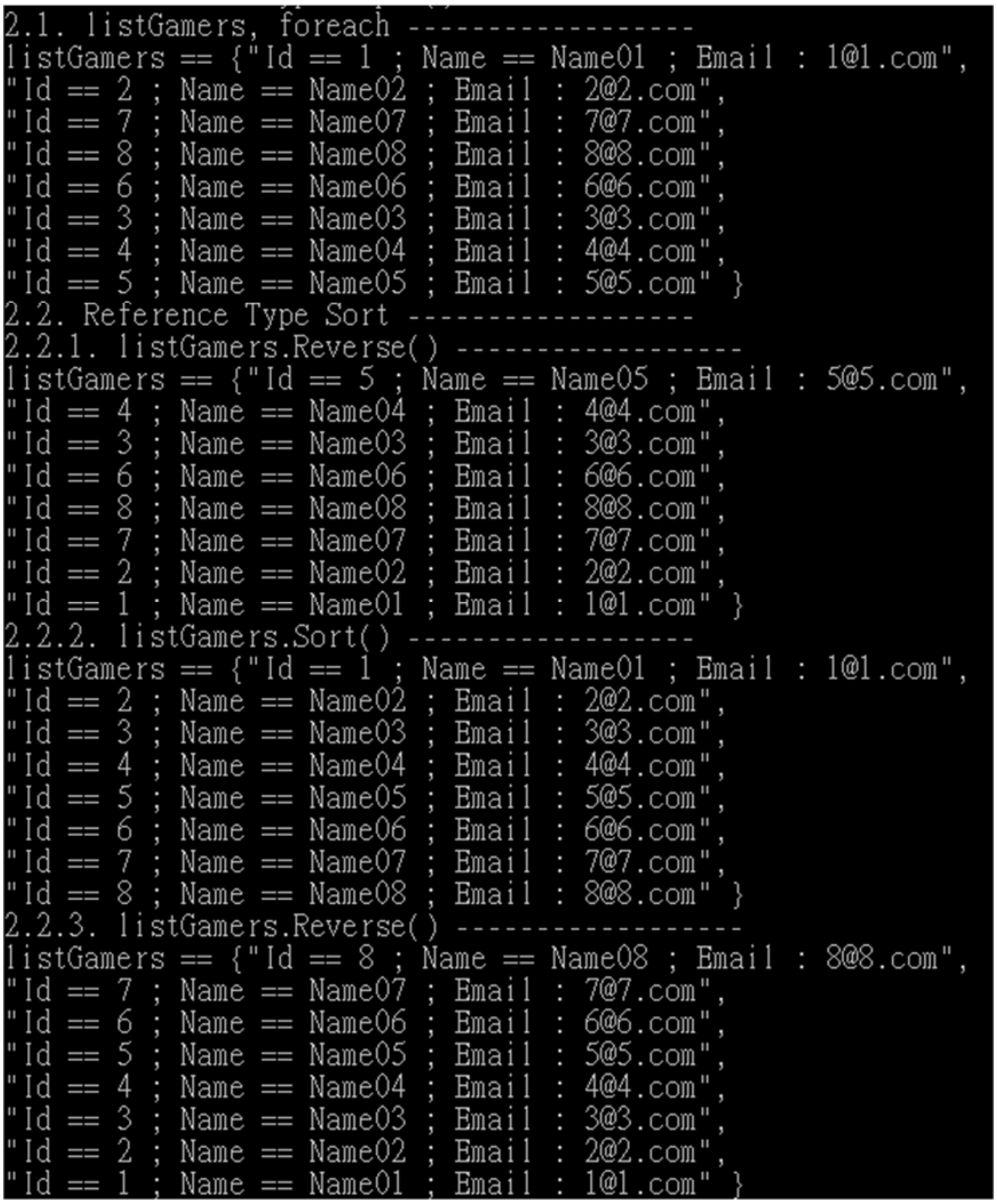
```
//From the "startIndex" position until "count" numbers of elements,
//Use the conditions defined by the lambda expression
//to search for the index of the element,
//then return the first matching index.
Console.WriteLine("2.5.1. ListObject.FindIndex() ---------------------- ");
int findIndexGamerId03 = listGamers.FindIndex(g => g.Id == 3);
Console.WriteLine($"listGamers.FindIndex(g => g.Id == 3)  :  " +
                 $"{findIndexGamerId03}");
int findIndexGamerId03V2 = listGamers.FindIndex(2, g => g.Id == 3);
Console.WriteLine($"listGamers.FindIndex(2, g => g.Id == 3)  :  " +
                 $"{findIndexGamerId03V2}");
int findIndexGamerId03V3 = listGamers.FindIndex(3, g => g.Id == 3);
Console.WriteLine($"listGamers.FindIndex(3, g => g.Id == 3)  :  " +
                 $"{findIndexGamerId03V3}");
int findIndexGamerId03V4 = listGamers.FindIndex(0, 3, g => g.Id == 3);
Console.WriteLine($"listGamers.FindIndex(0, 3, g => g.Id == 3)  :  " +
                 $"{findIndexGamerId03V4}");
int findIndexGamerId03V5 = listGamers.FindIndex(0, 2, g => g.Id == 3);
Console.WriteLine($"listGamers.FindIndex(0, 2, g => g.Id == 3)  :  " +
                 $"{findIndexGamerId03V5}");
//2.5.1. ListObject.FindIndex() ----------------------
//listGamers.FindIndex(g => g.Id == 3)  :  2
//listGamers.FindIndex(2, g => g.Id == 3)  :  2
//listGamers.FindIndex(3, g => g.Id == 3)  :  -1
//listGamers.FindIndex(0, 3, g => g.Id == 3)  :  2
//listGamers.FindIndex(0, 2, g => g.Id == 3)  :  -1
Console.WriteLine("2.5.2. ListObject.FindIndex() ---------------------- ");
int findIndexGamerIdGreaterThan03V1 = listGamers.FindIndex(g => g.Id >= 3);
Console.WriteLine($"listGamers.FindIndex(g => g.Id >= 3)  :  " +
                 $"{findIndexGamerIdGreaterThan03V1}");
int findIndexGamerIdGreaterThan03V2 = listGamers.FindIndex(2, g => g.Id >= 3);
Console.WriteLine($"listGamers.FindIndex(2, g => g.Id >= 3)  :  " +
                 $"{findIndexGamerIdGreaterThan03V2}");
int findIndexGamerIdGreaterThan03V3 = listGamers.FindIndex(3, g => g.Id >= 3);
Console.WriteLine($"listGamers.FindIndex(3, g => g.Id >= 3)  :  " +
                 $"{findIndexGamerIdGreaterThan03V3}");
int findIndexGamerIdGreaterThan03V4 = listGamers.FindIndex(0, 3, g => g.Id >= 3);
Console.WriteLine($"listGamers.FindIndex(0, 3, g => g.Id >= 3)  :  " +
                 $"{findIndexGamerIdGreaterThan03V4}");
int findIndexGamerIdGreaterThan03V5 = listGamers.FindIndex(0, 2, g => g.Id >= 3);
Console.WriteLine($"listGamers.FindIndex(0, 2, g => g.Id >= 3)  :  " +
                 $"{findIndexGamerIdGreaterThan03V5}");
//2.5.2. ListObject.FindIndex() ----------------------
//listGamers.FindIndex(g => g.Id >= 3)  :  2
//listGamers.FindIndex(2, g => g.Id >= 3)  :  2
//listGamers.FindIndex(3, g => g.Id >= 3)  :  3
//listGamers.FindIndex(0, 3, g => g.Id >= 3)  :  2
//listGamers.FindIndex(0, 2, g => g.Id >= 3)  :  -1



// 2.6. ------------------------------
//ListObject.FindLastIndex()
```

```
            Console.WriteLine("2.6. ListObject.FindLastIndex() ---------------------- ");
            //1.7.
            //1.7.1.
            ////List<T>.FindLastIndex(Predicate<T> match)
            //Use the conditions defined by the lambda expression
            //to search for the index of the element,
            //then return the last matching index.
            //Return - 1 if the index can not be found.
            int findLastIndexGamerId3 = listGamers.FindLastIndex(g => g.Id == 3);
            Console.WriteLine($"listGamers.FindLastIndex(g => g.Id == 3)  :  " +
                             $"{findLastIndexGamerId3}");
            int findLastIndexGamerId0 = listGamers.FindLastIndex(g => g.Id == 0);
            Console.WriteLine($"listGamers.FindLastIndex(g => g.Id == 3)  :  " +
                             $"{findLastIndexGamerId0}");
            int findLastIndexGamerIdGreaterThan3 = listGamers.FindLastIndex(g => g.Id >= 3);
            Console.WriteLine($"listGamers.FindLastIndex(g => g.Id >= 3)  :  " +
                             $"{findLastIndexGamerIdGreaterThan3}");
            //2.6. ListObject.FindLastIndex() ----------------------
            //listGamers.FindLastIndex(g => g.Id == 3)  :  2
            //listGamers.FindLastIndex(g => g.Id == 3)  :  -1
            //listGamers.FindLastIndex(g => g.Id >= 3)  :  7
            // 2.7. -------------------------------
            //ListObject.IndexOf()
            Console.WriteLine("2.7. ListObject.IndexOf() ---------------------- ");
            Console.WriteLine("2.7.1. listGamers.IndexOf() ---------------------- ");
            int indexOfGamerId3 = listGamers.IndexOf(
                new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" });
            Console.WriteLine($"listGamers.IndexOf(new Gamer {{ Id = 3, Name = \"Name03\", Email
= \"3@3.com\" }}) :  " +
                             $"{indexOfGamerId3}");
            Console.WriteLine("2.7.2. listGamerAs.IndexOf() ---------------------- ");
            //in order to use IndexOf(),
            //The Value Type must override Equals() and GetHashCode()
            int indexOfGamerAId3 = listGamerAs.IndexOf(
                new GamerA { Id = 3, Name = "Name03", Email = "3@3.com" });
            Console.WriteLine($"listGamerAs.IndexOf(new GamerA {{ Id = 3, Name = \"Name03\", Email
= \"3@3.com\" }}) :  " +
                             $"{indexOfGamerAId3}");
            //2.7. ListObject.IndexOf() ----------------------
            //2.7.1.listGamers.IndexOf()----------------------
            //listGamers.IndexOf(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" }) :  -1
            //2.7.2.listGamerAs.IndexOf()----------------------
            //listGamerAs.IndexOf(new GamerA { Id = 3, Name = "Name03", Email = "3@3.com" }) :  2
            // 2.8. -------------------------------
            //ListObject.FirstOrDefault(x => x.Equals(...))
            Console.WriteLine("2.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
");
            Console.WriteLine("2.8.1. listGamers.FirstOrDefault(x => x.Equals(new Gamer{...})) -----------
----------- ");
            Gamer firstEqualsGamerId2 = listGamers.FirstOrDefault(x => x.Equals(
                new Gamer { Id = 2, Name = "Name02", Email = "2@2.com" }));
            //string firstEqualsGamerId2Str = firstEqualsGamerId2==null?
            //    "Not Found":
```

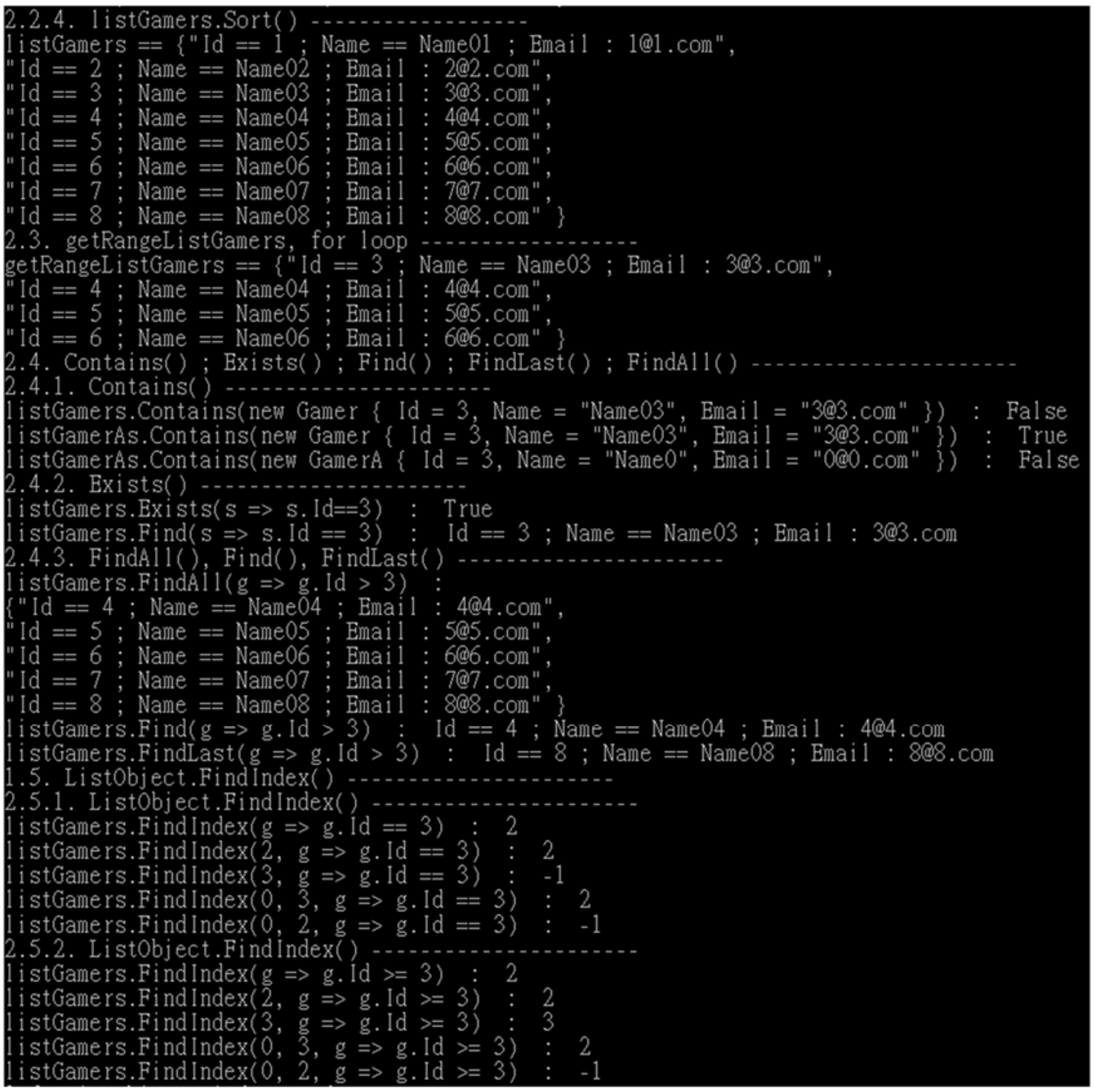
```
            //    firstEqualsGamerId2.ToString();
            string firstEqualsGamerId2Str = firstEqualsGamerId2?.ToString() ?? "Not Found";
            Console.WriteLine($"listGamers.FirstOrDefault(x => x.Equals(x => x.Equals(" +
                              $"new Gamer {{ Id = 2, Name = \"Name02\", Email = \"2@2.com\" }})) :  " +
                              $"{firstEqualsGamerId2Str}");
            Console.WriteLine("2.8.2. listGamerAs.FirstOrDefault(x => x.Equals(new GamerA{...})) ---------
------------- ");
            GamerA firstEqualsGamerAId2 = listGamerAs.FirstOrDefault(x => x.Equals(
                new GamerA { Id = 2, Name = "Name02", Email = "2@2.com" }));
            Console.WriteLine($"listGamerAs.FirstOrDefault(x => x.Equals(x => x.Equals(" +
                              $"new GamerA {{ Id = 2, Name = \"Name02\", Email = \"2@2.com\" }})) :  " +
                              $"{firstEqualsGamerAId2}");
            //string firstEqualsGamerAId0Str = listGamerAs.FirstOrDefault(g => g.Id==0)==null?
            //    "Not Found":
            //    firstEqualsGamerAId0.ToString();
            string firstEqualsGamerAId0Str = listGamerAs.FirstOrDefault(g => g.Id == 0)?.ToString() ?? "Not
Found";
            Console.WriteLine($"listGamerAs.FirstOrDefault(g => g.Id==0)  :  {firstEqualsGamerAId0Str}");
            //2.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
            //2.8.1.listGamers.FirstOrDefault(x => x.Equals(new Gamer {...}))----------------------
            //listGamers.FirstOrDefault(x => x.Equals(x => x.Equals(new Gamer { Id = 2, Name = "Name02",
Email = "2@2.com" }))  :  Not Found
            //2.8.2.listGamerAs.FirstOrDefault(x => x.Equals(new GamerA {...}))----------------------
            //listGamerAs.FirstOrDefault(x => x.Equals(x => x.Equals(new GamerA { Id = 2, Name = "Name02",
Email = "2@2.com" }))  :  Id == 2 ; Name == Name02; Email: 2@2.com
            //listGamerAs.FirstOrDefault(g => g.Id == 0)  :  Not Found
            // 2.9. ------------------------------
            //ListObject[index]
            Console.WriteLine("2.9. ListObject[index] ---------------------- ");
            Console.WriteLine("2.9.1. listGamers[index] ---------------------- ");
            Gamer listGamersGamer01 = listGamers[0]; // get index 0 value object.
            Console.WriteLine($"listGamers[0]  ==  {listGamers[0]}");
            Console.WriteLine($"listGamers[1]  ==  {listGamers[1]}");
            Console.WriteLine($"listGamers[2]  ==  {listGamers[2]}");
            Console.WriteLine($"listGamers[3]  ==  {listGamers[3]}");
            //Console.WriteLine($"listGamers[100]  ==  {listGamers[100]}");   // Error! Throw Example.
            //2.9. ListObject[index] ----------------------
            //2.9.1.listGamers[index]----------------------
            //listGamers[0] == Id == 1; Name == Name01; Email: 1@1.com
            //listGamers[1] == Id == 2; Name == Name02; Email: 2@2.com
            //listGamers[2] == Id == 3; Name == Name03; Email: 3@3.com
            //listGamers[3] == Id == 4; Name == Name04; Email: 4@4.com
            Console.WriteLine("2.9.2. listGamerAs[index] ---------------------- ");
            GamerA listGamerAsGamer01 = listGamerAs[0]; // get index 0 value object.
            Console.WriteLine($"listGamerAs[0]  ==  {listGamerAs[0]}");
            Console.WriteLine($"listGamerAs[1]  ==  {listGamerAs[1]}");
            Console.WriteLine($"listGamerAs[2]  ==  {listGamerAs[2]}");
            Console.WriteLine($"listGamerAs[3]  ==  {listGamerAs[3]}");
            //Console.WriteLine($"listGamerAs[100]  ==  {listGamerAs[100]}");   // Error! Throw Example.
            //2.9.2.listGamerAs[index]----------------------
            //listGamerAs[0]  ==  Id == 1 ; Name == Name01 ; Email : 1@1.com
            //listGamerAs[1] == Id == 2; Name == Name02; Email: 2@2.com
            //listGamerAs[2] == Id == 3; Name == Name03; Email: 3@3.com
            //listGamerAs[3] == Id == 4; Name == Name04; Email: 4@4.com
            // 2.10. -------------------------------
            //List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ; List<T>.RemoveRange(int index, int 
count)
            Console.WriteLine("2.10. List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ; 
List<T>.RemoveRange(int index, int count) ----------");
            //1.14.
            //1.14.1.
            ////List<T>.Remove(T item)
            //Removes the first occurrence of a specific object from the List<T>.
            //1.14.2.
            ////List<T>.RemoveAt(int index)
            //Removes the element at the specified index of the List<T>.
            //1.14.3.
            ////List<T>.RemoveRange Method (int index, int count)
            //From "index" postion and take "count" elements, then remove them from the list.
            Console.WriteLine("2.10.1. listGamers --------------------------------------");
            Console.WriteLine("2.10.1.1. listGamers.Remove(new Gamer " +
                              "{ Id = 1, Name = \"Name01\", Email = \"1@1.com\" }) ------------");
            listGamers.Remove(new Gamer
            {
                Id = 1,
                Name = "Name01",
                Email = "1@1.com"
            });
            //It will not remove "Id1", because Gamer does not override Equals() and GetHashCode()
            Console.Write("listGamers == {");
            foreach (Gamer gamerItem in listGamers)
            {
                if (gamerItem == listGamers.Last())
                {
                    Console.WriteLine($"\"{gamerItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerItem}\", ");
                }
            }
            //2.10.1.listGamers--------------------------------------
            //2.10.1.1.listGamers.Remove(new Gamer { Id = 1, Name = "Name01", Email = "1@1.com" })--------
----
            //listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
            //"Id == 2 ; Name == Name02 ; Email : 2@2.com",
            //"Id == 3 ; Name == Name03 ; Email : 3@3.com",
            //"Id == 4 ; Name == Name04 ; Email : 4@4.com",
            //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
            //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
            //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
            //"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
            Console.WriteLine("2.10.1.2. listGamers.RemoveAt(0) ------------");
            listGamers.RemoveAt(0);   //Remove index 0 postion item
            Console.Write("listGamers == {");
            foreach (Gamer gamerItem in listGamers)
            {
                if (gamerItem == listGamers.Last())
                {
```

```
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
    //2.10.1.2. listGamers.RemoveAt(0) ------------
    //listGamers == {"Id == 2 ; Name == Name02 ; Email : 2@2.com",
    //"Id == 3 ; Name == Name03 ; Email : 3@3.com",
    //"Id == 4 ; Name == Name04 ; Email : 4@4.com",
    //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
    //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
    //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
    //"Id == 8 ; Name == Name08 ; Email : 8@8.com" }


   Console.WriteLine("2.10.1.3. listGamers.RemoveRange(0,2) ------------");
    listGamers.RemoveRange(0, 2);   //Remove 2 items from index 0 postion
    Console.Write("listGamers == {");
    foreach (Gamer gamerItem in listGamers)
    {
        if (gamerItem == listGamers.Last())
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
    //2.10.1.3. listGamers.RemoveRange(0,2) ------------
    //listGamers == {"Id == 4 ; Name == Name04 ; Email : 4@4.com",
    //"Id == 5 ; Name == Name05 ; Email : 5@5.com",
    //"Id == 6 ; Name == Name06 ; Email : 6@6.com",
    //"Id == 7 ; Name == Name07 ; Email : 7@7.com",
    //"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
    Console.WriteLine("2.10.2. listGamerAs --------------------------------------");
    Console.WriteLine("2.10.2.1. listGamerAs.Remove(new GamerA " +
                      "{ Id = 1, Name = \"Name01\", Email = \"1@1.com\" }) ------------");
    listGamerAs.Remove(new GamerA
    {
        Id = 1,
        Name = "Name01",
        Email = "1@1.com"
    });
    //It will remove "Id1", because GamerA override Equals() and GetHashCode()
    Console.Write("listGamerAs == {");
    foreach (GamerA gamerItem in listGamerAs)
    {
        if (Equals(gamerItem, listGamerAs.Last()))
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
    //2.10.2. listGamerAs --------------------------------------
    //2.10.2.1.listGamerAs.Remove(new GamerA { Id = 1, Name = "Name01", Email = "1@1.com" })------
------
    //listGamerAs == {"Id == 2 ; Name == Name02 ; Email : 2@2.com",
    //"Id == 3 ; Name == Name03 ; Email : 3@3.com",
    //"Id == 4 ; Name == Name04 ; Email : 4@4.com" }
    Console.WriteLine("2.10.2.2. listGamerAs.RemoveAt(0) ------------");
    listGamerAs.RemoveAt(0);   //Remove index 0 postion item
    Console.Write("listGamerAs == {");
    foreach (GamerA gamerItem in listGamerAs)
    {
        if (Equals(gamerItem, listGamerAs.Last()))
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
    //2.10.2.2. listGamerAs.RemoveAt(0) ------------
    //listGamerAs == {"Id == 3 ; Name == Name03 ; Email : 3@3.com",
    //"Id == 4 ; Name == Name04 ; Email : 4@4.com" }
    Console.WriteLine("2.10.2.3. listGamerAs.RemoveRange(0,2) ------------");
    listGamerAs.RemoveRange(0, 1);   //Remove 1 items from index 0 postion
    Console.Write("listGamerAs == {");
    foreach (GamerA gamerItem in listGamerAs)
    {
        if (Equals(gamerItem, listGamerAs.Last()))
        {
            Console.WriteLine($"\"{gamerItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerItem}\", ");
        }
    }
    //2.10.2.3.listGamerAs.RemoveRange(0, 2)------------
    //listGamerAs == { "Id == 4 ; Name == Name04 ; Email : 4@4.com" }
    // 2.11. ------------------------------
    //TrueForAll()
    Console.WriteLine("2.11. TrueForAll() ---------------------------------------");
    //1.17
    ////List<T>.TrueForAll(Predicate<T> match)
    //Determines whether every element in the List<T>
    //matches the conditions defined by the specified predicate.
    bool trueForAll1 = listGamers.TrueForAll(g => g.Id >= 4);
    Console.WriteLine($"listGamers.TrueForAll(g => g.Id>=4)  ==  {trueForAll1}");
    bool trueForAll2 = listGamers.TrueForAll(g => g.Id >= 5);
    Console.WriteLine($"listGamers.TrueForAll(g => g.Id>=5)  ==  {trueForAll2}");
    //2.11. TrueForAll() --------------------------------------
    //listGamers.TrueForAll(g => g.Id >= 4) == True
    //listGamers.TrueForAll(g => g.Id >= 5) == False
```

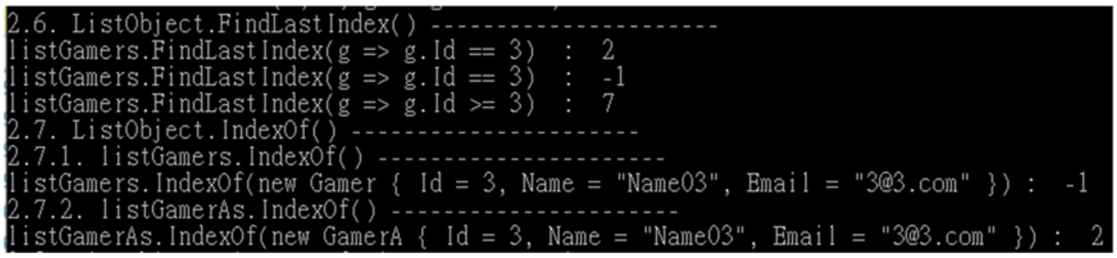
```
            // 2.12. -------------------------------
             Console.WriteLine("2.12. listGamers.Clear() ---------------------------------------");
             listGamers.Clear();   //Remove all elements.
             if (listGamers.Count > 0)
             {
                 Console.Write("listGamers == {");
                 foreach (Gamer gamerItem in listGamers)
                 {
                     if (gamerItem == listGamers.Last())
                     {
                         Console.WriteLine($"\"{gamerItem}\"" + " }");
                     }
                     else
                     {
                         Console.WriteLine($"\"{gamerItem}\", ");
                     }
                 }
             }
             else
             {
                 Console.WriteLine("listGamers has no element.");
             }
             //2.12. listGamers.Clear() ---------------------------------------
             //listGamers has no element.
         }
     }
}


namespace OnLineGame
{
    // 2. ==============================================================
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name} ; Email : {Email}";
        }
    }
    public class GamerA
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name} ; Email : {Email}";
        }
        public override bool Equals(object obj)
        {
            GamerA gamerA = (GamerA)obj;
            return gamerA != null &&
                 gamerA.Id == Id &&
                gamerA.Name == Name &&
                gamerA.Email == Email;
        }
        public override int GetHashCode()
        {
            return Id.GetHashCode() ^
                Name.GetHashCode() ^
                Email.GetHashCode();
        }
        //For more detail,
        //please see previous tutorial OverrideEquals_OverrideGetHashCode
        //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.
        //^ means xor
        //The popular way to override GetHashCode is
        //using XOR to connect all fields.
    }
}
```



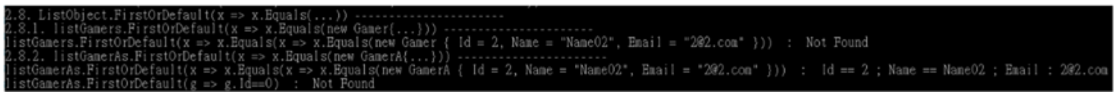
```
2.1. listGamers, foreach ------------------
listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com" }
2.2. Reference Type Sort ------------------
2.2.1. listGamers.Reverse() ------------------
listGamers == {"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 1 ; Name == Name01 ; Email : 1@1.com" }
2.2.2. listGamers.Sort() ------------------
listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
2.2.3. listGamers.Reverse() ------------------
listGamers == {"Id == 8 ; Name == Name08 ; Email : 8@8.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 1 ; Name == Name01 ; Email : 1@1.com" }
```



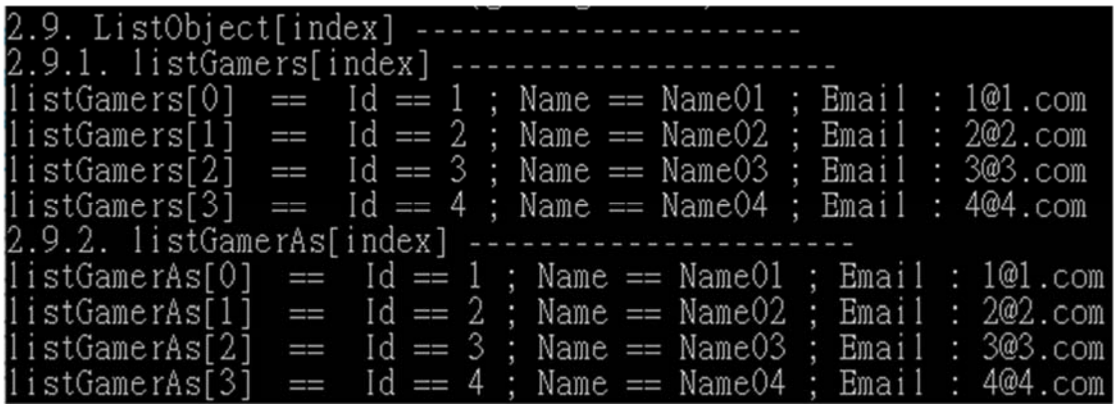
```
2.2.4. listGamers.Sort() ------------------
listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
2.3. getRangeListGamers, for loop ------------------
getRangeListGamers == {"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com" }
2.4. Contains() ; Exists() ; Find() ; FindLast() ; FindAll() ----------------------
2.4.1. Contains() ----------------------
listGamers.Contains(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" })  :  False
listGamerAs.Contains(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" })  :  True
listGamerAs.Contains(new GamerA { Id = 3, Name = "Name0", Email = "0@0.com" })  :  False
2.4.2. Exists() ----------------------
listGamers.Exists(s => s.Id==3)  :  True
listGamers.Find(s => s.Id == 3)  :  Id == 3 ; Name == Name03 ; Email : 3@3.com
2.4.3. FindAll(), Find(), FindLast() ----------------------
listGamers.FindAll(g => g.Id > 3)  :
{"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
listGamers.Find(g => g.Id > 3)  :  Id == 4 ; Name == Name04 ; Email : 4@4.com
listGamers.FindLast(g => g.Id > 3)  :  Id == 8 ; Name == Name08 ; Email : 8@8.com
1.5. ListObject.FindIndex() ----------------------
2.5.1. ListObject.FindIndex() ----------------------
listGamers.FindIndex(g => g.Id == 3)  :  2
listGamers.FindIndex(2, g => g.Id == 3)  :  2
listGamers.FindIndex(3, g => g.Id == 3)  :  -1
listGamers.FindIndex(0, 3, g => g.Id == 3)  :  2
listGamers.FindIndex(0, 2, g => g.Id == 3)  :  -1
2.5.2. ListObject.FindIndex() ----------------------
listGamers.FindIndex(g => g.Id >= 3)  :  2
listGamers.FindIndex(2, g => g.Id >= 3)  :  2
listGamers.FindIndex(3, g => g.Id >= 3)  :  3
listGamers.FindIndex(0, 3, g => g.Id >= 3)  :  2
listGamers.FindIndex(0, 2, g => g.Id >= 3)  :  -1
```



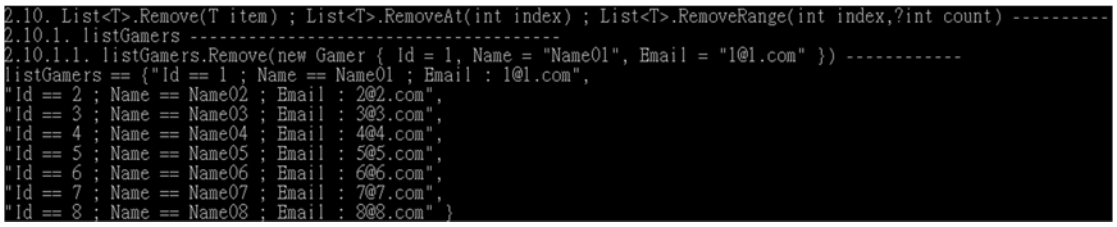
```
2.6. ListObject.FindLastIndex() ----------------------
listGamers.FindLastIndex(g => g.Id == 3)  :  2
listGamers.FindLastIndex(g => g.Id == 3)  :  -1
listGamers.FindLastIndex(g => g.Id >= 3)  :  7
2.7. ListObject.IndexOf() ----------------------
2.7.1. listGamers.IndexOf() ----------------------
listGamers.IndexOf(new Gamer { Id = 3, Name = "Name03", Email = "3@3.com" }) :  -1
2.7.2. listGamerAs.IndexOf() ----------------------
listGamerAs.IndexOf(new GamerA { Id = 3, Name = "Name03", Email = "3@3.com" }) :  2
```



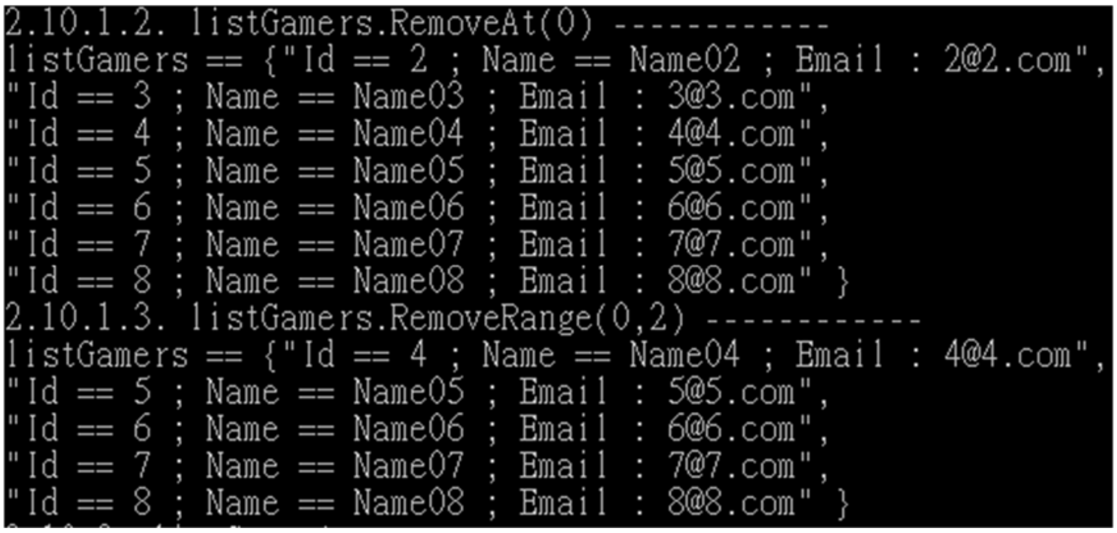
```
2.8. ListObject.FirstOrDefault(x => x.Equals(...)) ----------------------
2.8.1. listGamers.FirstOrDefault(x => x.Equals(new Gamer{...})) ----------------------
listGamers.FirstOrDefault(x => x.Equals(x => x.Equals(new Gamer { Id = 2, Name = "Name02", Email = "2@2.com" }))  :  Not Found
2.8.2. listGamerAs.FirstOrDefault(x => x.Equals(new GamerA{...})) ----------------------
listGamerAs.FirstOrDefault(x => x.Equals(x => x.Equals(new GamerA { Id = 2, Name = "Name02", Email = "2@2.com" }))  :  Id == 2 ; Name == Name02 ; Email : 2@2.com
listGamerAs.FirstOrDefault(g => g.Id==0)  :  Not Found
```



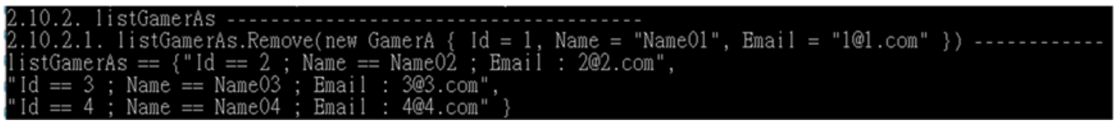
```
2.9. ListObject[index] ----------------------
2.9.1. listGamers[index] ----------------------
listGamers[0]  ==  Id == 1 ; Name == Name01 ; Email : 1@1.com
listGamers[1]  ==  Id == 2 ; Name == Name02 ; Email : 2@2.com
listGamers[2]  ==  Id == 3 ; Name == Name03 ; Email : 3@3.com
listGamers[3]  ==  Id == 4 ; Name == Name04 ; Email : 4@4.com
2.9.2. listGamerAs[index] ----------------------
listGamerAs[0]  ==  Id == 1 ; Name == Name01 ; Email : 1@1.com
listGamerAs[1]  ==  Id == 2 ; Name == Name02 ; Email : 2@2.com
listGamerAs[2]  ==  Id == 3 ; Name == Name03 ; Email : 3@3.com
listGamerAs[3]  ==  Id == 4 ; Name == Name04 ; Email : 4@4.com
```



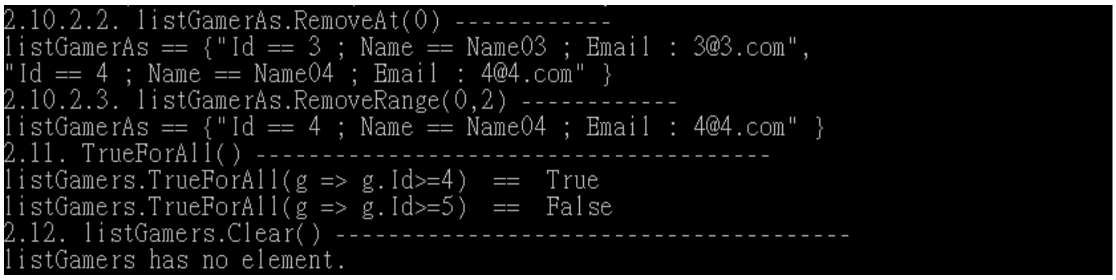
```
2.10. List<T>.Remove(T item) ; List<T>.RemoveAt(int index) ; List<T>.RemoveRange(int index,?int count) ----------
2.10.1. listGamers -----------------------------------
2.10.1.1. listGamers.Remove(new Gamer { Id = 1, Name = "Name01", Email = "1@1.com" }) ------------
listGamers == {"Id == 1 ; Name == Name01 ; Email : 1@1.com",
"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
```



```
2.10.1.2. listGamers.RemoveAt(0) ------------
listGamers == {"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
2.10.1.3. listGamers.RemoveRange(0,2) ------------
listGamers == {"Id == 4 ; Name == Name04 ; Email : 4@4.com",
"Id == 5 ; Name == Name05 ; Email : 5@5.com",
"Id == 6 ; Name == Name06 ; Email : 6@6.com",
"Id == 7 ; Name == Name07 ; Email : 7@7.com",
"Id == 8 ; Name == Name08 ; Email : 8@8.com" }
```



```
2.10.2. listGamerAs -----------------------------------
2.10.2.1. listGamerAs.Remove(new GamerA { Id = 1, Name = "Name01", Email = "1@1.com" }) ------------
listGamerAs == {"Id == 2 ; Name == Name02 ; Email : 2@2.com",
"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com" }
```



```
2.10.2.2. listGamerAs.RemoveAt(0) ------------
listGamerAs == {"Id == 3 ; Name == Name03 ; Email : 3@3.com",
"Id == 4 ; Name == Name04 ; Email : 4@4.com" }
2.10.2.3. listGamerAs.RemoveRange(0,2) ------------
listGamerAs == {"Id == 4 ; Name == Name04 ; Email : 4@4.com" }
2.11. TrueForAll() -------------------------------------
listGamers.TrueForAll(g => g.Id>=4)  ==  True
listGamers.TrueForAll(g => g.Id>=5)  ==  False
2.12. listGamers.Clear() -------------------------------------
listGamers has no element.
```

==

3. T022_ListReferenceTypeSort

==

學生提問

https://www.facebook.com/groups/934567793358849/posts/2052970214851929/

問題如下

老師及各位前輩好，小弟最近看了 C#深入解析中的單元 10，T022_ListReferenceTypeSort 影片 ，有問題兩個問題在標題 4.

再麻煩指教了，謝謝

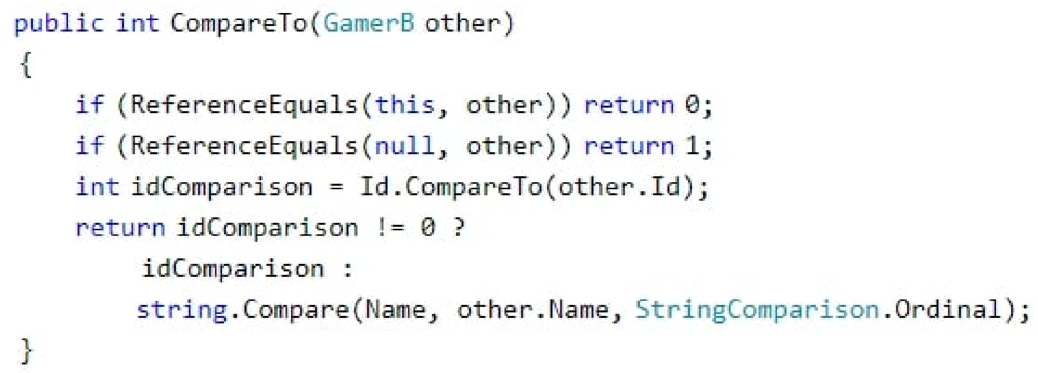
1.課程名稱

C#深入解析中的單元 10，T022_ListReferenceTypeSort

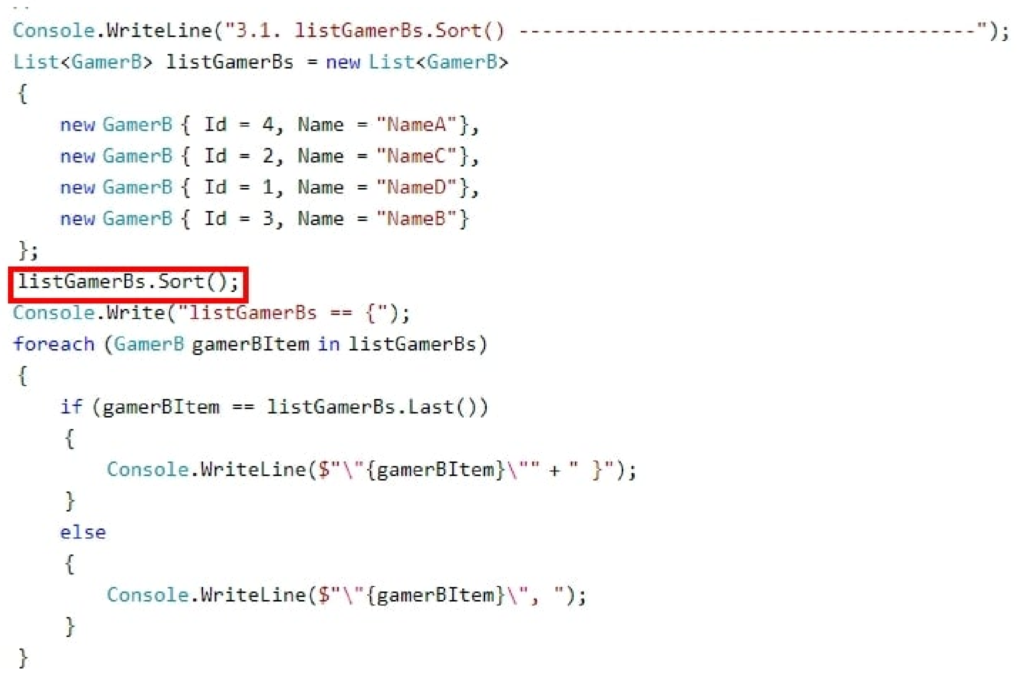
2.

影片 連結 , 影片的 分鐘數 和 秒數 (必填)

https://hiskio.com/courses/182/lectures/7138
04:24 開始
----
3. 講義連結 和 講義小節名稱 (必填)
https://ithandyguytutorial.blogspot.com/2017/12/t022listreferencetypesort.html
講義小節名稱 : 3.1 處的
public class GamerB : IComparable<GamerB>中的 Method
public int CompareTo(GamerB other)
----
4. 我講了什麼? 你不懂什麼? (必填)
對於課程影片中 04:24 實作 public int CompareTo(GamerB other) 的部分有以下兩個疑問:
一、影片 08:33 中的第 1462 行的 listGamerBs.soft()會呼叫影片 7:39 處 1595 行實作的 public int CompareTo(GamerB other)進行排序嗎(這部份想確定一下)
二、若真的是呼叫上述影片 7:39 處 1595 行實作的 public int CompareTo(GamerB other)進行排序，那這個會回傳 int 的方法是如何讓 listGamerBs 進行排序的?背後的運作原理是什麼?因為這會影響到設計邏輯，我想弄清楚運作原理，謝謝!



```
public int CompareTo(GamerB other)
{
    if (ReferenceEquals(this, other)) return 0;
    if (ReferenceEquals(null, other)) return 1;
    int idComparison = Id.CompareTo(other.Id);
    return idComparison != 0 ?
        idComparison :
        string.Compare(Name, other.Name, StringComparison.Ordinal);
}
```



```
Console.WriteLine("3.1. listGamerBs.Sort() ---------------------------------------");
List<GamerB> listGamerBs = new List<GamerB>
{
    new GamerB { Id = 4, Name = "NameA"},
    new GamerB { Id = 2, Name = "NameC"},
    new GamerB { Id = 1, Name = "NameD"},
    new GamerB { Id = 3, Name = "NameB"}
};
listGamerBs.Sort();
Console.Write("listGamerBs == {");
foreach (GamerB gamerBItem in listGamerBs)
{
    if (gamerBItem == listGamerBs.Last())
    {
        Console.WriteLine($"\"{gamerBItem}\"" + " }");
    }
    else
    {
        Console.WriteLine($"\"{gamerBItem}\", ");
    }
}
```

Q1.
影片 08:33 中的第 1462 行的 listGamerBs.sort()會呼叫影片 7:39 處 1595 行實作的 public int CompareTo(GamerB other)進行排序嗎(這部份想確定一下)

A:
Yes
GamerB 的 Object 如果想做 Sort()
那麼 GamerB 就必須實作 IComparable<GamerB>

實作 IComparable<GamerB>就必須要實作 public int CompareTo(GamerB other)
而 Sort()就會呼叫 CompareTo(GamerB other)來做排序的動作

Q2.
若真的是呼叫上述影片 7:39 處 1595 行實作的 public int CompareTo(GamerB other)進行排序，那這個會回傳 int 的方法是如何讓 listGamerBs 進行排序的?背後的運作原理是什麼?因為這會影響到設計邏輯，我想弄清楚運作原理。

A:
你這個問題超綱了
你應該去學 演算法
https://www.udemy.com/course/data-structures-and-algorithms-in-csharp/
https://www.youtube.com/watch?v=kPRA0W1kECg
https://www.interviewbit.com/tutorial/sorting-algorithms/
有很多種 Sort 的演算法
Quick Sort
Bubble Sort
Merge Sort
Insertion Sort
Selection Sort
Heap Sort
Radix Sort
Bucket Sort
我不知道 C#用哪一種
但是不管用哪一種
他們都只能 Sort 有實作 IComparable<GamerB>的 Object

我再補充一點
你不需要知道 C# Sort()到底是用哪種 Sort 運算法
你只需要知道
如果你的 Object 需要做 Sort()
就記得該 Class 必須實作 IComparable<T>或是 IComparer<T>
如果你的 Object 的 Compare 方式只有一種，請使用 IComparable<T>
如果你的 Object 的 Compare 方式有很多種，請使用 IComparer<T>
==

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using OnLineGame;


namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 3. ----------------------------------------------------
            Console.WriteLine("3. ListReferenceTypeSortSample() ==============================");
            ListReferenceTypeSortSample();
            Console.ReadLine();
        }


        // 3. ==============================================================
        static void ListReferenceTypeSortSample()
        {
            //3.
            //Reference Type Sort
            //-----------------------------------
            //3.1.
            ////List<T>.Sort()
            //Sorts the elements in the entire List<T> using the default comparer.
            //-------------------
            //Need to implement IComparable<T> and CompareTo() before using Sort().
            ////public class ClassA : IComparable<ClassA>
            //...
            ////public int CompareTo(ClassA other){...the compare logic...}
            //if CompareTo()==0,
            //it means current instance object is equal to "other" instance object.
            //if CompareTo()>0,
            //it means current instance object > "other" instance object.
            //if CompareTo()<0,
            //it means current instance object < "other" instance object.
            //...
            ////listClassA.Sort();
            //Sort the listClassA
            Console.WriteLine("3.1. listGamerBs.Sort() ---------------------------------------");
            List<GamerB> listGamerBs = new List<GamerB>
            {
                new GamerB { Id = 4, Name = "NameA"},
                new GamerB { Id = 2, Name = "NameC"},
                new GamerB { Id = 1, Name = "NameD"},
                new GamerB { Id = 3, Name = "NameB"}
            };
            listGamerBs.Sort();
            Console.Write("listGamerBs == {");
            foreach (GamerB gamerBItem in listGamerBs)
            {
                if (gamerBItem == listGamerBs.Last())
                {
                    Console.WriteLine($"\"{gamerBItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerBItem}\", ");
                }
            }
            //3.1.listGamerBs.Sort()-------------------------------------- -
            //listGamerBs == {"Id == 1 ; Name == NameD",
            //"Id == 2 ; Name == NameC",
            //"Id == 3 ; Name == NameB",
            //"Id == 4 ; Name == NameA" }
            //-----------------------------------
            //3.2.
```

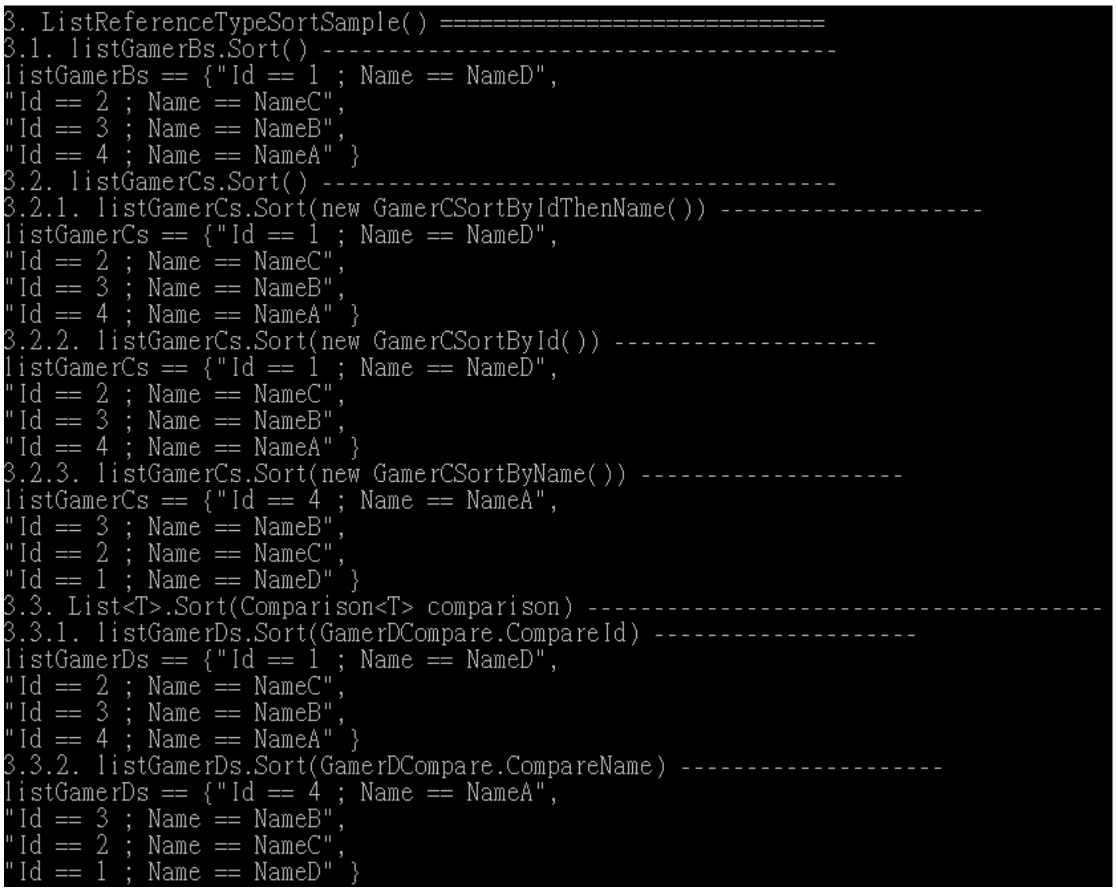
```
            ////List<T>.Sort(IComparer<T> comparer)
            //Sorts the elements in the entire List<T> using the specified comparer.
            //-------------------
            //If you are not allowed to change ClassA code.
            //Then you care create other class to implement IComparer
            ////public class SortByName : IComparer<T>
            ////{
            ////    public int Compare(T currentObj, T otherObj)
            ////    {
            ////        //...Compare Logic...
            ////    }
            ////}
            //...
            ////listCutomers.Sort(new SortByName());
            Console.WriteLine("3.2. listGamerCs.Sort() ---------------------------------------");
            List<GamerC> listGamerCs = new List<GamerC>
            {
                new GamerC { Id = 4, Name = "NameA"},
                new GamerC { Id = 2, Name = "NameC"},
                new GamerC { Id = 1, Name = "NameD"},
                new GamerC { Id = 3, Name = "NameB"}
            };
            Console.WriteLine("3.2.1. listGamerCs.Sort(new GamerCSortByIdThenName()) --------------------
");
            listGamerCs.Sort(new GamerCSortByIdThenName());
            Console.Write("listGamerCs == {");
            foreach (GamerC gamerCItem in listGamerCs)
            {
                if (gamerCItem == listGamerCs.Last())
                {
                    Console.WriteLine($"\"{gamerCItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerCItem}\", ");
                }
            }
            //3.2. listGamerCs.Sort() ---------------------------------------
            //3.2.1.listGamerCs.Sort(new GamerCSortByIdThenName())--------------------
            //listGamerCs == {"Id == 1 ; Name == NameD",
            //"Id == 2 ; Name == NameC",
            //"Id == 3 ; Name == NameB",
            //"Id == 4 ; Name == NameA" }
            Console.WriteLine("3.2.2. listGamerCs.Sort(new GamerCSortById()) --------------------");
            listGamerCs.Sort(new GamerCSortById());
            Console.Write("listGamerCs == {");
            foreach (GamerC gamerCItem in listGamerCs)
            {
                if (gamerCItem == listGamerCs.Last())
                {
                    Console.WriteLine($"\"{gamerCItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerCItem}\", ");
                }
```

```
    }
    //3.2.2. listGamerCs.Sort(new GamerCSortById()) --------------------
    //listGamerCs == {"Id == 1 ; Name == NameD",
    //"Id == 2 ; Name == NameC",
    //"Id == 3 ; Name == NameB",
    //"Id == 4 ; Name == NameA" }


  Console.WriteLine("3.2.3. listGamerCs.Sort(new GamerCSortByName()) --------------------");
    listGamerCs.Sort(new GamerCSortByName());
    Console.Write("listGamerCs == {");
    foreach (GamerC gamerCItem in listGamerCs)
    {
        if (gamerCItem == listGamerCs.Last())
        {
            Console.WriteLine($"\"{gamerCItem}\"" + " }");
        }
        else
        {
            Console.WriteLine($"\"{gamerCItem}\", ");
        }
    }
    //3.2.3. listGamerCs.Sort(new GamerCSortByName()) --------------------
    //listGamerCs == {"Id == 4 ; Name == NameA",
    //"Id == 3 ; Name == NameB",
    //"Id == 2 ; Name == NameC",
    //"Id == 1 ; Name == NameD" }
    //-----------------------------------
    //3.3.
    ////List<T>.Sort(Comparison<T> comparison)
    //Sorts the elements in the entire List<T> using the specified System.Comparison<T>.
    //-------------------
    //3.3.1.
    ////private static int CompareClassA(ClassA c1, ClassB c2)
    ////{
    ////    return c1.ID.CompareTo(c2.ID);
    ////}
    //Create a function whose signature matches the signature of System.Comparison delegate.
    //which is
    ////public void Sort(Comparison<T> comparison)
    //...
    ////Comparison<ClassA> classAComparer = new Comparison<ClassA>(CompareClassA);
    //Create an instance of System.Comparison delegate and point to CompareClassA method.
    //...
    ////listClassA.Sort(classAComparer);
    //Pass the delegate to sort parameter.
    //-------------------
    //3.3.2.
    ////listClassA.Sort(delegate(ClassA c1, ClassB c2)
    ////{
    ////     return (c1.ID.CompareTo(c2.ID));
    ////});
    //-------------------
    //3.3.3.
```

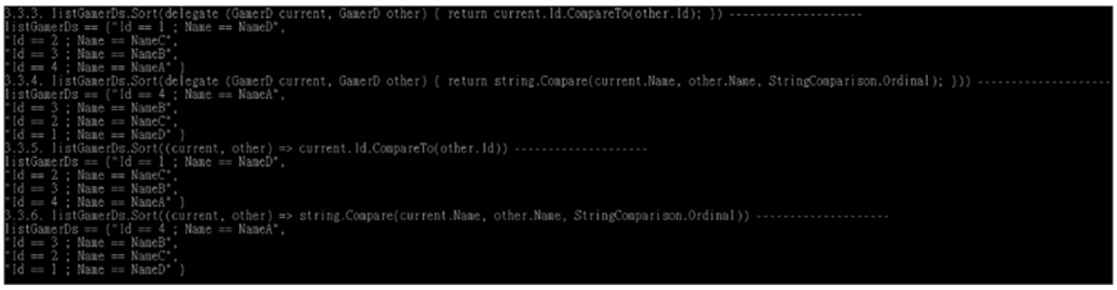
```
            ////listClassA.Sort((x, y) => x.ID.CompareTo(y.ID));
            Console.WriteLine("3.3. List<T>.Sort(Comparison<T> comparison) -------------------------------
--------");
            List<GamerD> listGamerDs = new List<GamerD>
            {
                new GamerD { Id = 4, Name = "NameA"},
                new GamerD { Id = 2, Name = "NameC"},
                new GamerD { Id = 1, Name = "NameD"},
                new GamerD { Id = 3, Name = "NameB"}
            };
            Console.WriteLine("3.3.1. listGamerDs.Sort(GamerDCompare.CompareId) --------------------");
            //Comparison<GamerD> gamerDCompareId = new Comparison<GamerD>(GamerDCompare.CompareId);
            listGamerDs.Sort(GamerDCompare.CompareId);
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3. List<T>.Sort(Comparison<T> comparison) ---------------------------------------
            //3.3.1.listGamerDs.Sort(GamerDCompare.CompareId)--------------------
            //listGamerDs == {"Id == 1 ; Name == NameD",
            //"Id == 2 ; Name == NameC",
            //"Id == 3 ; Name == NameB",
            //"Id == 4 ; Name == NameA" }


          Console.WriteLine("3.3.2. listGamerDs.Sort(GamerDCompare.CompareName) --------------------");
            //Comparison<GamerD> gamerDCompareName = new Comparison<GamerD>(GamerDCompare.CompareName);
            listGamerDs.Sort(GamerDCompare.CompareName);
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3.2. listGamerDs.Sort(GamerDCompare.CompareName) --------------------
            //listGamerDs == {"Id == 4 ; Name == NameA",
            //"Id == 3 ; Name == NameB",
            //"Id == 2 ; Name == NameC",
            //"Id == 1 ; Name == NameD" }
            Console.WriteLine("3.3.3. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return
current.Id.CompareTo(other.Id); }) --------------------");
            listGamerDs.Sort(delegate (GamerD current, GamerD other)
{ return current.Id.CompareTo(other.Id); });
```

```
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3.3. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return 
current.Id.CompareTo(other.Id); }) --------------------
            //listGamerDs == {"Id == 1 ; Name == NameD",
            //"Id == 2 ; Name == NameC",
            //"Id == 3 ; Name == NameB",
            //"Id == 4 ; Name == NameA" }
            Console.WriteLine("3.3.4. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return 
string.Compare(current.Name, other.Name, StringComparison.Ordinal); })) --------------------");
            listGamerDs.Sort(delegate (GamerD current, GamerD other) { return string.Compare(current.Name, 
other.Name, StringComparison.Ordinal); });
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3.4. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return 
string.Compare(current.Name, other.Name, StringComparison.Ordinal); })) --------------------
            //listGamerDs == {"Id == 4 ; Name == NameA",
            //"Id == 3 ; Name == NameB",
            //"Id == 2 ; Name == NameC",
            //"Id == 1 ; Name == NameD" }
            Console.WriteLine("3.3.5. listGamerDs.Sort((current, other) => current.Id.CompareTo(other.Id)) 
--------------------");
            listGamerDs.Sort((current, other) => current.Id.CompareTo(other.Id));
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3.5. listGamerDs.Sort((current, other) => current.Id.CompareTo(other.Id)) ----------------
----
            //listGamerDs == {"Id == 1 ; Name == NameD",
            //"Id == 2 ; Name == NameC",
            //"Id == 3 ; Name == NameB",
            //"Id == 4 ; Name == NameA" }
            Console.WriteLine("3.3.6. listGamerDs.Sort((current, other) => string.Compare(current.Name,
other.Name, StringComparison.Ordinal)) --------------------");
            listGamerDs.Sort((current, other) => string.Compare(current.Name,
other.Name, StringComparison.Ordinal));
            Console.Write("listGamerDs == {");
            foreach (GamerD gamerDItem in listGamerDs)
            {
                if (gamerDItem == listGamerDs.Last())
                {
                    Console.WriteLine($"\"{gamerDItem}\"" + " }");
                }
                else
                {
                    Console.WriteLine($"\"{gamerDItem}\", ");
                }
            }
            //3.3.6. listGamerDs.Sort((current, other) => string.Compare(current.Name, other.Name,
StringComparison.Ordinal)) --------------------
            //listGamerDs == {"Id == 4 ; Name == NameA",
            //"Id == 3 ; Name == NameB",
            //"Id == 2 ; Name == NameC",
            //"Id == 1 ; Name == NameD" }
        }
    }
}
```

```
namespace OnLineGame
{
    // 2. ==============================================================
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name} ; Email : {Email}";
        }
    }
    public class GamerA
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name} ; Email : {Email}";
        }
        public override bool Equals(object obj)
        {
            GamerA gamerA = (GamerA)obj;
            return gamerA != null &&
                 gamerA.Id == Id &&
                 gamerA.Name == Name &&
                 gamerA.Email == Email;
        }
        public override int GetHashCode()
        {
            return Id.GetHashCode() ^
                 Name.GetHashCode() ^
                 Email.GetHashCode();
        }
        //For more detail,
        //please see previous tutorial OverrideEquals_OverrideGetHashCode
        //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.
        //^ means xor
        //The popular way to override GetHashCode is
        //using XOR to connect all fields.
    }


    // 3.1. ==============================================================
    public class GamerB : IComparable<GamerB>
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name}";
        }
        //if CompareTo()==0,
        //it means current instance object is equal to "other" instance object.
        //if CompareTo()>0,
        //it means current instance object > "other" instance object.
        //if CompareTo()<0,
        //it means current instance object < "other" instance object.
        /// <summary>
        /// Compare Id first, then compare Name
        /// </summary>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public int CompareTo(GamerB other)
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            int idComparison = Id.CompareTo(other.Id);
            return idComparison != 0 ?
                 idComparison :
                string.Compare(Name, other.Name, StringComparison.Ordinal);
        }
    }
    // 3.2. ==============================================================
    public class GamerC
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name}";
        }
    }
    //If you are not allowed to change ClassA code.
    //Then you care create other class to implement IComparer
    public class GamerCSortByIdThenName : IComparer<GamerC>
    {
        //if Compare(GamerC current, GamerC other)==0,
        //it means current instance object is equal to "other" instance object.
        //if Compare(GamerC current, GamerC other) >0,
        //it means current instance object > "other" instance object.
        //if Compare(GamerC current, GamerC other) <0,
        //it means current instance object < "other" instance object.
        /// <summary>
        /// Compare Id first, then compare Name
        /// </summary>
        /// <param name="current">current object</param>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public int Compare(GamerC current, GamerC other)
        {
            if (ReferenceEquals(current, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            if (ReferenceEquals(null, current)) return -1;
            int idComparison = current.Id.CompareTo(other.Id);
            return idComparison != 0 ?
                 idComparison :
                string.Compare(current.Name, other.Name, StringComparison.Ordinal);
        }
    }
    public class GamerCSortById : IComparer<GamerC>
    {
        //if Compare(GamerC current, GamerC other)==0,
        //it means current instance object is equal to "other" instance object.
        //if Compare(GamerC current, GamerC other) >0,
        //it means current instance object > "other" instance object.
        //if Compare(GamerC current, GamerC other) <0,
        //it means current instance object < "other" instance object.
        /// <summary>
        /// Compare Id
        /// </summary>
        /// <param name="current">current object</param>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public int Compare(GamerC current, GamerC other)
        {
            if (ReferenceEquals(current, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            if (ReferenceEquals(null, current)) return -1;
            return current.Id.CompareTo(other.Id);
        }
    }
    public class GamerCSortByName : IComparer<GamerC>
    {
        //if Compare(GamerC current, GamerC other)==0,
        //it means current instance object is equal to "other" instance object.
        //if Compare(GamerC current, GamerC other) >0,
        //it means current instance object > "other" instance object.
        //if Compare(GamerC current, GamerC other) <0,
        //it means current instance object < "other" instance object.
        /// <summary>
        /// Compare Name
        /// </summary>
        /// <param name="current">current object</param>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public int Compare(GamerC current, GamerC other)
        {
            if (ReferenceEquals(current, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            if (ReferenceEquals(null, current)) return -1;
            return string.Compare(current.Name, other.Name, StringComparison.Ordinal);
        }
    }


   // 3.3. ==============================================================
    //3.3.1.
    ////private static int CompareClassA(ClassA c1, ClassB c2)
    ////{
    ////    return c1.ID.CompareTo(c2.ID);
    ////}
    //Create a function whose signature matches the signature of System.Comparison delegate.
    //which is
    ////public void Sort(Comparison<T> comparison)
    //...
    ////Comparison<ClassA> classAComparer = new Comparison<ClassA>(CompareClassA);
    //Create an instance of System.Comparison delegate and point to CompareClassA method.
    //...
    ////listClassA.Sort(classAComparer);
    //Pass the delegate to sort parameter.
    public class GamerD
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public override string ToString()
        {
            return $"Id == {Id} ; Name == {Name}";
        }
    }
    public class GamerDCompare
    {
        //if Compare(GamerC current, GamerC other)==0,
        //it means current instance object is equal to "other" instance object.
        //if Compare(GamerC current, GamerC other) >0,
        //it means current instance object > "other" instance object.
        //if Compare(GamerC current, GamerC other) <0,
        //it means current instance object < "other" instance object.
        /// <summary>
        /// Compare Id
        /// </summary>
        /// <param name="current">current object</param>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public static int CompareId(GamerD current, GamerD other)
        {
            if (ReferenceEquals(current, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            if (ReferenceEquals(null, current)) return -1;
            return current.Id.CompareTo(other.Id);
        }
        /// <summary>
        /// Compare Name
        /// </summary>
        /// <param name="current">current object</param>
        /// <param name="other">Other object</param>
        /// <returns>Return the comparison</returns>
        public static int CompareName(GamerD current, GamerD other)
        {
            if (ReferenceEquals(current, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            if (ReferenceEquals(null, current)) return -1;
            return string.Compare(current.Name, other.Name, StringComparison.Ordinal);
        }
    }
}
```



```
3. ListReferenceTypeSortSample() ==============================
3.1. listGamerBs.Sort() ----------------------------------------
listGamerBs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.2. listGamerCs.Sort() ----------------------------------------
3.2.1. listGamerCs.Sort(new GamerCSortByIdThenName()) --------------------
listGamerCs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.2.2. listGamerCs.Sort(new GamerCSortById()) --------------------
listGamerCs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.2.3. listGamerCs.Sort(new GamerCSortByName()) --------------------
listGamerCs == {"Id == 4 ; Name == NameA",
"Id == 3 ; Name == NameB",
"Id == 2 ; Name == NameC",
"Id == 1 ; Name == NameD" }
3.3. List<T>.Sort(Comparison<T> comparison) ----------------------------------------
3.3.1. listGamerDs.Sort(GamerDCompare.CompareId) --------------------
listGamerDs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.3.2. listGamerDs.Sort(GamerDCompare.CompareName) --------------------
listGamerDs == {"Id == 4 ; Name == NameA",
"Id == 3 ; Name == NameB",
"Id == 2 ; Name == NameC",
"Id == 1 ; Name == NameD" }
```



```
3.3.3. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return current.Id.CompareTo(other.Id); }) --------------------
listGamerDs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.3.4. listGamerDs.Sort(delegate (GamerD current, GamerD other) { return string.Compare(current.Name, other.Name, StringComparison.Ordinal); })) --------------------
listGamerDs == {"Id == 4 ; Name == NameA",
"Id == 3 ; Name == NameB",
"Id == 2 ; Name == NameC",
"Id == 1 ; Name == NameD" }
3.3.5. listGamerDs.Sort((current, other) => current.Id.CompareTo(other.Id)) --------------------
listGamerDs == {"Id == 1 ; Name == NameD",
"Id == 2 ; Name == NameC",
"Id == 3 ; Name == NameB",
"Id == 4 ; Name == NameA" }
3.3.6. listGamerDs.Sort((current, other) => string.Compare(current.Name, other.Name, StringComparison.Ordinal)) --------------------
listGamerDs == {"Id == 4 ; Name == NameA",
"Id == 3 ; Name == NameB",
"Id == 2 ; Name == NameC",
"Id == 1 ; Name == NameD" }
```