(T9)討論 LinqToObject 的 ToList、ToArray、ToDictionary、ToLookup、Cast 和 OfType
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c
==========
(T9)討論 LinqToObject 的 ToList、ToArray、ToDictionary、ToLookup、Cast 和 OfType
==========

==========

0. Summary

1.
Deferred/Lazy Operators V.S. Immediate/Greedy Operators
Based on the behavior of query execution, Linq can be classified into 2 categories.
1.1. Deferred/Lazy Operators use deferred execution.
E.g. select, where, Take, Skip ...
1.2. Immediate/Greedy Operators use immediate execution.
E.g. count, average, min, max, ToList ...
1.3.
ToList, ToArray, ToDictionary, ToLookup, Cast, OfType, AsEnumerable, AsQueryable
are Linq Conversion Operators.

2.
Enumerable.ToList<TSource>
(this IEnumerable<TSource> source)
Reference:
https://msdn.microsoft.com/en-us/library/bb342261(v=vs.110).aspx
Creates a List<T> from an IEnumerable<T>.
This is a **Immediate/Greedy Operator**
and causes the query to be executed immediately.

3.
Enumerable.ToArray<TSource>
(this IEnumerable<TSource> source)
Reference:
https://msdn.microsoft.com/en-us/library/bb298736(v=vs.110).aspx
Creates an array from a IEnumerable<T>.
This is a Immediate/Greedy Operator
and causes the query to be executed immediately.
This is a **Immediate/Greedy Operator**
and causes the query to be executed immediately.

4.
ToDictionary
This is a **Immediate/Greedy Operator**
and causes the query to be executed immediately.

The Dictionary key must be unique, but the Lookup key can be identical.

4.1.

Enumerable.ToDictionary<TSource, TKey>
(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector)

Reference:
https://msdn.microsoft.com/en-us/library/bb549277(v=vs.110).aspx

Creates a Dictionary<TKey, TValue> from an IEnumerable<T>
according to a specified key selector function.

4.2.

Enumerable.ToDictionary<TSource, TKey, TElement>
(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement>
elementSelector)

Reference:
https://msdn.microsoft.com/en-us/library/bb548657(v=vs.110).aspx

Creates a Dictionary<TKey, TValue> from an IEnumerable<T>
according to specified key selector and element selector functions.

4.2.1.

keySelector
A function to extract a key from each element

4.2.2.

elementSelector
A function to produce a result element from each element in the sequence

5.

ToLookup

This is a **Immediate/Greedy Operator**
and causes the query to be executed immediately.

The Dictionary key must be unique, but the Lookup key can be identical.

5.1.

Enumerable.ToLookup<TSource, TKey>
(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector)

Reference:
https://msdn.microsoft.com/en-us/library/bb549073(v=vs.110).aspx

Creates a Lookup<TKey, TElement> from an IEnumerable<T>
according to a specified key selector function.

5.2.

Enumerable.ToLookup<TSource, TKey, TElement>
(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement>
elementSelector)

Reference:
https://msdn.microsoft.com/en-us/library/bb549211(v=vs.110).aspx

Creates a Lookup<TKey, TElement> from an IEnumerable<T>
according to specified key selector and element selector functions.

5.2.1.

keySelector
A function to extract a key from each element

5.2.2.

elementSelector
A function to produce a result element from each element in the sequence

6.

Enumerable.Cast<TResult>
(this IEnumerable source)
Reference:
https://msdn.microsoft.com/en-us/library/bb341406(v=vs.110).aspx
Casts the elements of an IEnumerable to the specified type
and return them in a new collection.
Throw exception if an item fails conversion
This is a **Deferred/Lazy Operators**
and causes the query use deferred execution

7.
Enumerable.OfType<TResult>
(this IEnumerable source)
Reference:
https://msdn.microsoft.com/en-us/library/bb360913(v=vs.110).aspx
Filters the elements of an IEnumerable based on a specified type
and return them in a new collection.
Ignore the element if an element fails conversion,
Include the elements that can be converted.

==========

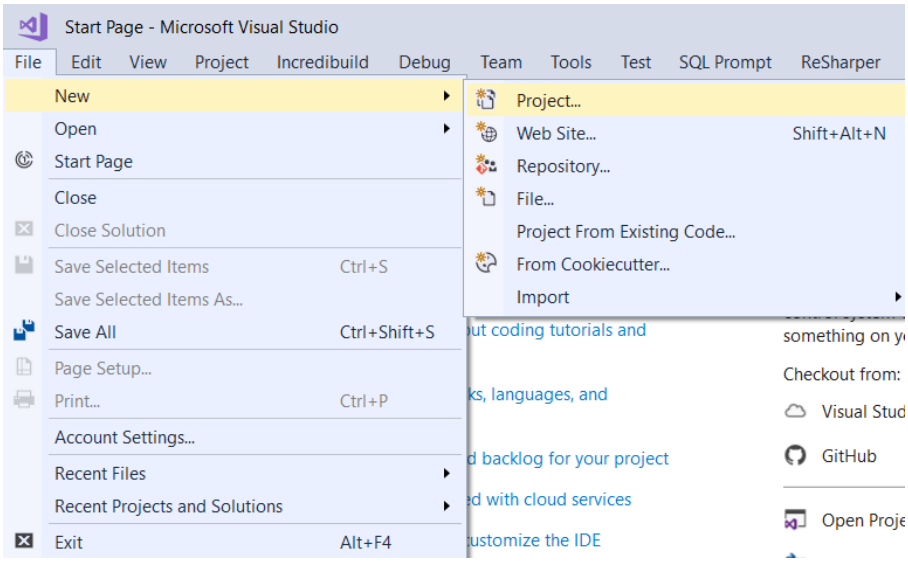
1. New Project

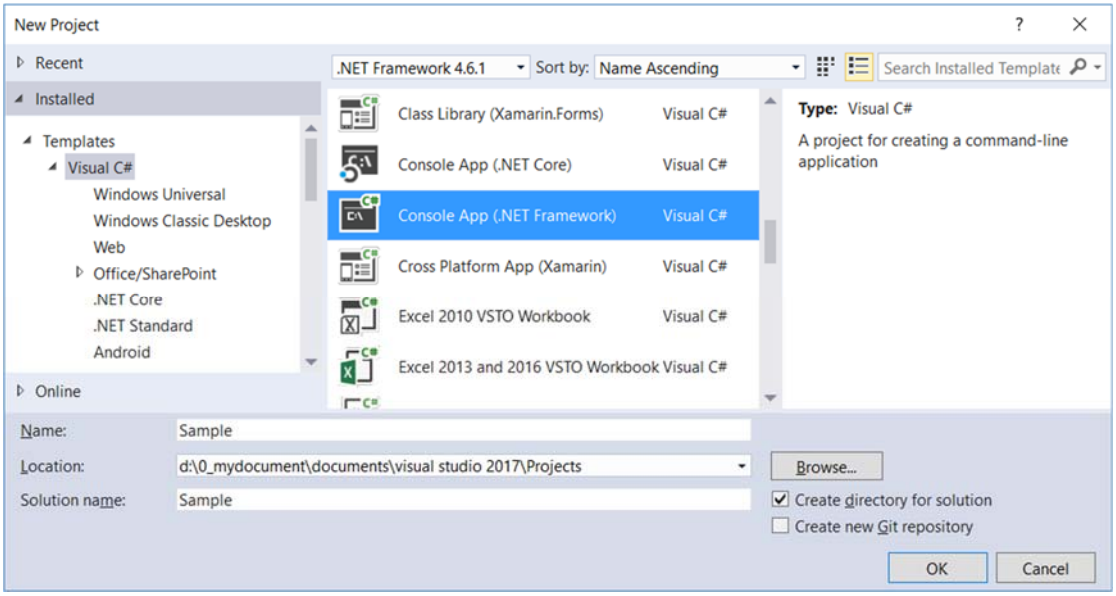
1.1. Create New Project : Sample

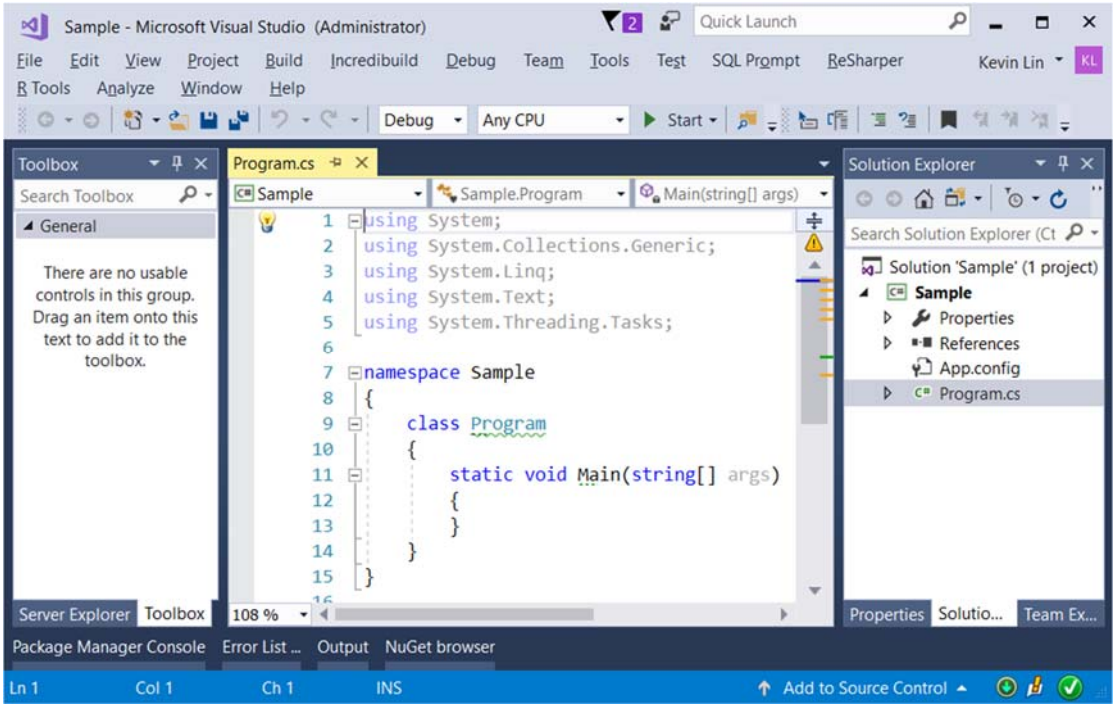
File --> New --> Project... -->

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

Name: **Sample**







==========

2. Sample : Program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using OnLineGame;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
        // 1. ====================================
        // ToListSample
        Console.WriteLine("1. ToListSample() ========================== ");
        ToListSample();
        // 2. ====================================
        // ToArraySample
        Console.WriteLine("2. ToArraySample() ========================== ");
        ToArraySample();
        // 3. ====================================
        // ToDictionarySample
        Console.WriteLine("3. ToDictionarySample() ========================== ");
        ToDictionarySample();
        // 4. ====================================
        // ToDictionarySample2
        Console.WriteLine("4. ToDictionarySample2() ========================== ");
        ToDictionarySample2();
        // 5. ====================================
        // ToLookupSample
        Console.WriteLine("5. ToLookupSample() ========================== ");
        ToLookupSample();
        // 6. ====================================
        // CastSample
        Console.WriteLine("6. CastSample() ========================== ");
        CastSample();
        // 7. ====================================
        // OfTypeSample
        Console.WriteLine("7. OfTypeSample() ========================== ");
        OfTypeSample();
        Console.ReadLine();
    }


// 1. ====================================
    // ToListSample
    static void ToListSample()
    {
        //2.
        //Enumerable.ToList<TSource>
        //(this IEnumerable < TSource > source)
        //Reference:
        //https://msdn.microsoft.com/en-us/library/bb342261(v=vs.110).aspx
        //Creates a List<T> from an IEnumerable<T>.
        //This is a Immediate / Greedy Operator
        //and causes the query to be executed immediately.
        int[] intArr = { 1, 2, 3, 4, 5 };
        List<int> intArrList = intArr.ToList();
        foreach (int i in intArrList)
        {
            Console.Write($" [ {i} ] ");
        }
        Console.WriteLine();
    }
    // [ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]
    // 2. ====================================
    // ToArraySample
```

```
        static void ToArraySample()
        {
            //3.
            //Enumerable.ToArray<TSource>
            //(this IEnumerable < TSource > source)
            //Reference:
            //https://msdn.microsoft.com/en-us/library/bb298736(v=vs.110).aspx
            //Creates an array from a IEnumerable<T>.
            //This is a Immediate / Greedy Operator
            //and causes the query to be executed immediately.
            //This is a Immediate / Greedy Operator
            //and causes the query to be executed immediately.
            List<string> magicTypeList = new List<string> { "Wood", "Fire", "Earth", "Gold", "Water" };
            // 2.1. lambda expression Linq query -------------------
            Console.WriteLine("2.1. lambda expression Linq query ------------- ");
            IOrderedEnumerable<string> magicEnumerable = magicTypeList.OrderBy(t => t);
            string[] magicTypeArr = magicEnumerable.ToArray();
            foreach (string magicType in magicTypeArr)
            {
                Console.Write($" [ {magicType} ] ");
            }
            Console.WriteLine();
            // 2.2. SQL like linq query -------------------
            Console.WriteLine("2.2. SQL Like Linq Query ------------- ");
            IOrderedEnumerable<string> magicEnumerable2 =
                from magicType in magicTypeList
                orderby magicType ascending
                select magicType;
            string[] magicTypeArr2 = magicEnumerable2.ToArray();
            foreach (string magicType in magicTypeArr2)
            {
                Console.Write($" [ {magicType} ] ");
            }
            Console.WriteLine();
        }
        // 2.1. lambda expression Linq query -------------
        // [ Earth ]  [ Fire ]  [ Gold ]  [ Water ]  [ Wood ]
        // 2.2. lambda expression Linq query -------------
        // [ Earth ]  [ Fire ]  [ Gold ]  [ Water ]  [ Wood ]


       // 3. ====================================
        // ToDictionarySample
        static void ToDictionarySample()
        {
            //4.2.
            //Enumerable.ToDictionary<TSource, TKey, TElement>
            //(this IEnumerable < TSource > source, Func < TSource, TKey > keySelector, Func < TSource,
TElement > elementSelector)
            //Reference:
            //https://msdn.microsoft.com/en-us/library/bb548657(v=vs.110).aspx
            //Creates a Dictionary<TKey, TValue> from an IEnumerable<T>
            //according to specified key selector and element selector functions.
            //4.2.1.
```

```
    //keySelector
    //A function to extract a key from each element
    //4.2.2.
    //elementSelector
    //A function to produce a result element from each element in the sequence
    List<Gamer> gamersList = GamerHelper.GetSampleGamers(5);
    Dictionary<int, string> gamersDictionary =
        gamersList.ToDictionary(g => g.Id, g => g.Name);
    foreach (KeyValuePair<int, string> gamersDictionaryItem in gamersDictionary)
    {
        Console.WriteLine($"gamersDictionaryItem.Key=={gamersDictionaryItem.Key}, " +
                          $"gamersDictionaryItem.Value=={gamersDictionaryItem.Value}");
    }
}
// gamersDictionaryItem.Key==1, gamersDictionaryItem.Value==Name1
// gamersDictionaryItem.Key==2, gamersDictionaryItem.Value==Name2
// gamersDictionaryItem.Key==3, gamersDictionaryItem.Value==Name3
// gamersDictionaryItem.Key==4, gamersDictionaryItem.Value==Name4
// gamersDictionaryItem.Key==5, gamersDictionaryItem.Value==Name5
// 4. ====================================
// ToDictionarySample2
static void ToDictionarySample2()
{
    //4.1.
    //Enumerable.ToDictionary<TSource, TKey>
    //(this IEnumerable < TSource > source, Func < TSource, TKey > keySelector)
    //Reference:
    //https://msdn.microsoft.com/en-us/library/bb549277(v=vs.110).aspx
    //Creates a Dictionary<TKey, TValue> from an IEnumerable<T>
    //according to a specified key selector function.
    List<Gamer> gamersList = GamerHelper.GetSampleGamers(5);
    Dictionary<int, Gamer> gamersDictionary =
        gamersList.ToDictionary(g => g.Id);
    foreach (KeyValuePair<int, Gamer> gamersDictionaryItem in gamersDictionary)
    {
        Console.WriteLine($"gamersDictionaryItem.Key=={gamersDictionaryItem.Key}, " +
                          $"gamersDictionaryItem.Value: {gamersDictionaryItem.Value}");
    }
}
// gamersDictionaryItem.Key==1, gamersDictionaryItem.Value: Id==1,Name==Name1,Gender==1184
// gamersDictionaryItem.Key==2, gamersDictionaryItem.Value: Id==2,Name==Name2,Gender==2373
// gamersDictionaryItem.Key==3, gamersDictionaryItem.Value: Id==3,Name==Name3,Gender==1869
// gamersDictionaryItem.Key==4, gamersDictionaryItem.Value: Id==4,Name==Name4,Gender==1149
// gamersDictionaryItem.Key==5, gamersDictionaryItem.Value: Id==5,Name==Name5,Gender==2548


// 5. ====================================
// ToLookupSample
static void ToLookupSample()
{
    //5.
    //ToLookup
    //This is a Immediate / Greedy Operator
    //and causes the query to be executed immediately.
    //The Dictionary key must be unique, but the Lookup key can be identical.
```

```
        List<GamerA> gamerAsList = new List<GamerA>
        {
            new GamerA { Id = 1, Name = "Name1", Gender = "Male", TeamName = "Team1"},
            new GamerA { Id = 2, Name = "Name2", Gender = "Female", TeamName = "Team2"},
            new GamerA { Id = 3, Name = "Name3", Gender = "Male", TeamName = "Team1"},
            new GamerA { Id = 4, Name = "Name4", Gender = "Male", TeamName = "Team1"},
            new GamerA { Id = 5, Name = "Name5", Gender = "Male", TeamName = "Team3"},
            new GamerA { Id = 6, Name = "Name6", Gender = "Female", TeamName = "Team3"},
            new GamerA { Id = 7, Name = "Name7", Gender = "Female", TeamName = "Team2"},
            new GamerA { Id = 8, Name = "Name8", Gender = "Female", TeamName = "Team3"},
            new GamerA { Id = 9, Name = "Name9", Gender = "Male", TeamName = "Team2"}
        };
        // 5.1. Lookup GamerA by Gender ----------------------------
        ILookup<string, GamerA> gamersByGenderLookup =
            gamerAsList.ToLookup(g => g.Gender);
        Console.WriteLine("5.1. Lookup GamerA by Gender ----------------------------");
        foreach (IGrouping<string, GamerA> gamersByGenderLookupItem in gamersByGenderLookup)
        {
            Console.WriteLine($"gamersByGenderLookupItem.Key=={gamersByGenderLookupItem.Key}");
            // Lookup GamerA by Gender
            foreach (GamerA gamer in gamersByGenderLookup[gamersByGenderLookupItem.Key])
            {
                Console.WriteLine(gamer);
            }
        }
        // 5.2. Lookup GamerA by TeamName ----------------------------
        ILookup<string, GamerA> gamersByTeamLookup =
            gamerAsList.ToLookup(g => g.TeamName);
        Console.WriteLine("5.2. Lookup GamerA by TeamName ----------------------------");
        foreach (IGrouping<string, GamerA> gamersByTeamLookupItem in gamersByTeamLookup)
        {
            Console.WriteLine($"gamersByTeamLookupItem.Key=={gamersByTeamLookupItem.Key}");
            // Lookup GamerA by TeamName
            foreach (GamerA gamer in gamersByTeamLookup[gamersByTeamLookupItem.Key])
            {
                Console.WriteLine(gamer);
            }
        }
    }
// 5.1. Lookup GamerA by Gender ----------------------------
// gamersByGenderLookupItem.Key==Male
// Id==1,Name==Name1,Gender==Male,TeamName==Team1
// Id==3,Name==Name3,Gender==Male,TeamName==Team1
// Id==4,Name==Name4,Gender==Male,TeamName==Team1
// Id==5,Name==Name5,Gender==Male,TeamName==Team3
// Id==9,Name==Name9,Gender==Male,TeamName==Team2
// gamersByGenderLookupItem.Key==Female
// Id==2,Name==Name2,Gender==Female,TeamName==Team2
// Id==6,Name==Name6,Gender==Female,TeamName==Team3
// Id==7,Name==Name7,Gender==Female,TeamName==Team2
// Id==8,Name==Name8,Gender==Female,TeamName==Team3
// 5.2. Lookup GamerA by TeamName ----------------------------
// gamersByTeamLookupItem.Key==Team1
// Id==1,Name==Name1,Gender==Male,TeamName==Team1
// Id==3,Name==Name3,Gender==Male,TeamName==Team1
// Id==4,Name==Name4,Gender==Male,TeamName==Team1
```

```
// gamersByTeamLookupItem.Key==Team2
// Id==2,Name==Name2,Gender==Female,TeamName==Team2
// Id==7,Name==Name7,Gender==Female,TeamName==Team2
// Id==9,Name==Name9,Gender==Male,TeamName==Team2
// gamersByTeamLookupItem.Key==Team3
// Id==5,Name==Name5,Gender==Male,TeamName==Team3
// Id==6,Name==Name6,Gender==Female,TeamName==Team3
// Id==8,Name==Name8,Gender==Female,TeamName==Team3


// 6. ====================================
// CastSample
static void CastSample()
{
    //6.
    //Enumerable.Cast<TResult>
    //(this IEnumerable source)
    //Reference:
    //https://msdn.microsoft.com/en-us/library/bb341406(v=vs.110).aspx
    //Casts the elements of an IEnumerable to the specified type
    //and return them in a new collection.
    //Throw exception if an item fails conversion
    //This is a Deferred / Lazy Operators
    //and causes the query use deferred execution
    // 6.1. arrayListCastInt ----------------
    Console.WriteLine("6.1. arrayListCastInt ---------------- ");
    ArrayList arrayList = new ArrayList();
    arrayList.Add(1);
    arrayList.Add(2);
    IEnumerable<int> arrayListCastInt = arrayList.Cast<int>();
    foreach (int i in arrayListCastInt)
    {
        Console.WriteLine(i);
    }
    // 6.2. arrayListCastInt2 ----------------
    Console.WriteLine("6.2. arrayListCastInt2 ---------------- ");
    try
    {
        ArrayList arrayList2 = new ArrayList();
        arrayList2.Add(1);
        arrayList2.Add(2);
        arrayList2.Add("ABC");  // cause an exception
        IEnumerable<int> arrayList2CastInt = arrayList2.Cast<int>();
        foreach (int i in arrayList2CastInt)
        {
            Console.WriteLine(i);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}
// 6.1. arrayListCastInt ----------------
// 1
// 2
```

```
        // 6.2. arrayListCastInt2 ---------------
        // 1
        // 2
        // System.InvalidCastException: Specified cast is not valid.
        // at System.Linq.Enumerable.<CastIterator>d__95`1.MoveNext()
        // at Sample.Program.CastSample() in D:\0_MyDocument\Documents\Visual Studio
2017\Projects\Sample\Sample\Program.cs:line 258
        // 7. ====================================
        // OfTypeSample
        static void OfTypeSample()
        {
            ArrayList arrayList = new ArrayList();
            arrayList.Add(1);
            arrayList.Add(2);
            arrayList.Add("3");
            arrayList.Add("ABC");  // cause an exception
            IEnumerable<int> arrayList2OfTypeInt = arrayList.OfType<int>();
            foreach (int i in arrayList2OfTypeInt)
            {
                Console.WriteLine(i);
            }
        }
        // 1
        // 2
    }
}


namespace OnLineGame
{
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Score { get; set; }
        public override string ToString()
        {
            return $"Id=={Id},Name=={Name},Score=={Score}";
        }
    }
    public class GamerHelper
    {
        // Create a List<Gamer> which contains numberOfGamers gamers.
        public static List<Gamer> GetSampleGamers(int numberOfGamers)
        {
            //int numberOfGamers = 10;
            List<Gamer> gamerList = new List<Gamer>();
            Random rnd = new Random();
            for (int i = 1; i <= numberOfGamers; i++)
            {
                int rndScore = rnd.Next(1000, 6000); // creates a number between 1000 and 6000
                gamerList.Add(new Gamer { Id = i, Name = $"Name{i}", Score = rndScore });
            }
            return gamerList;
        }
    }
    public class GamerA
```

```
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public string TeamName { get; set; }
        public override string ToString()
        {
            return $"Id=={Id},Name=={Name},Gender=={Gender},TeamName=={TeamName}";
        }
    }
}
```