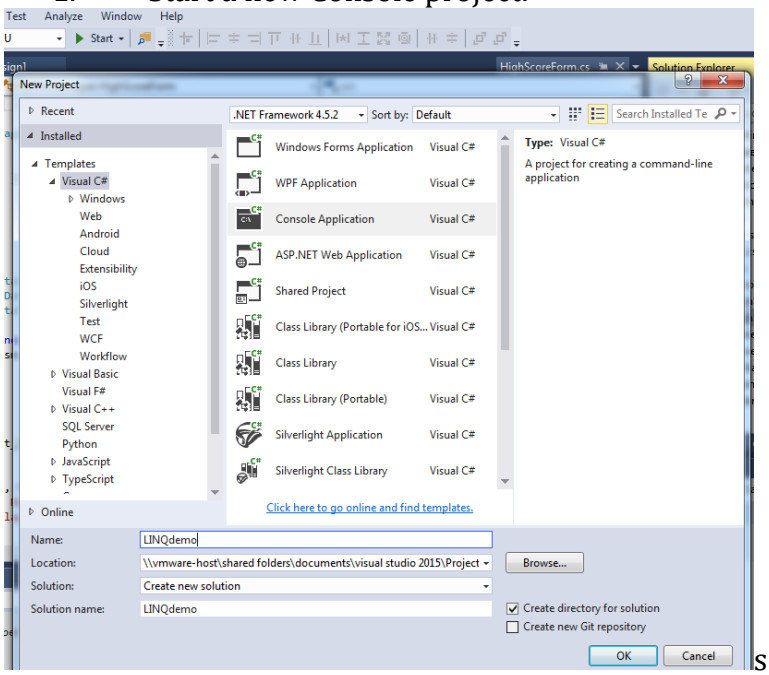
Activity 8: LINQ

Objective: Demonstrate the of filtering lists of data using for loops and LINQ expressions.

The example shown here is for a simple array. In the second part of this tutorial, we will use Objects and Lists.

1. Start a new Console project.



screenshot

2. Add an array of test scores.

```
rogram.cs*
LINQdemo                    LINQdemo.Program                    Main(string[] args)
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace LINQdemo
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13
14              // make some test scores
15              int[] scores = new[] { 50, 66, 90, 81, 77, 45, 0, 100, 99, 72, 87, 85, 81, 80, 77, 74, 95, 97 };
16
```

3. Print the scores with a for-each loop

Program.cs

LINQdemo | LINQdemo.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LINQdemo
{
    class Program
    {
        static void Main(string[] args)
        {

            // make some test scores
            var scores = new[] { 50, 66, 90, 81, 77, 45, 0, 100, 99, 72, 87, 85, 81, 80, 77, 74, 95, 97 };

            // print the scores
            foreach (var individualScore in scores ) {
                Console.WriteLine("One of the scores was {0}", individualScore);
            }

            // pause to see the output before closing
            Console.ReadLine();
        }
    }
}
```

4. Run the program. Take a screenshot of the application. Paste the image into a Microsoft Word document. Add a caption describing what is being demonstrated.

5. Filter out only the A students and print them to the screen.

```
INQdemo                    LINQdemo.Program                    Main(string[] args)
10      {
11          static void Main(string[] args)
12          {
13
14              // make some test scores
15              var scores = new[] { 50, 66, 90, 81, 77, 45, 0, 100, 99, 72, 87, 85, 81, 80, 77, 74, 95, 97 };
16
17              // print the scores
18              foreach (var individualScore in scores ) {
19                  Console.WriteLine("One of the scores was {0}", individualScore);
20              }
21
22              // pause to see the output before closing
23              Console.ReadLine();
24
25              // use a LINQ statement to filter the list.
26              var theBestStudents =
27                  from individualScore in scores
28                  where individualScore > 90
29                  select individualScore;
30
31              // print only the bestscores
32              foreach (var individualScore in theBestStudents)
33              {
34                  Console.WriteLine("One of the BEST scores was {0}", individualScore);
35              }
36              // pause to see the output before closing
37              Console.ReadLine();
38          }
39      }
```

6. Run the program. Take a screen shot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.
7. Now create a list of sorted scores.

```
Program.cs
LINQdemo                    LINQdemo.Program                    Main(string[] args)
            var theBestStudents =
                from individualScore in scores
                where individualScore > 90
                select individualScore;

            // print only the bestscores
            foreach (var individualScore in theBestStudents)
            {
                Console.WriteLine("One of the BEST scores was {0}", individualScore);
            }
            // pause to see the output before closing
            Console.ReadLine();


            // use LINQ to sort the results
            var sortedScores =
                from individualScore in scores
                orderby individualScore
                select individualScore;

            // print the sorted list
            foreach (var individualScore in sortedScores)
            {
                Console.WriteLine("One of the scores was {0}", individualScore);
            }
            // pause to see the output before closing
            Console.ReadLine();
```

8. Run the program. Take a screen shot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Challenge

1. Print a list of only the B students (80% to 89%) in ascending order.
2. Run the program. Take a screen shot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Two Ways to Write LINQ

You can write LINQ expressions in C# using two syntactical flavors:

- Method
- Query

In LINQ, operators are chained together to filter and condition sets of data. Method syntax uses the familiar dot operator to express this chain. Suppose you have a list of integers and you want to create a list only containing numbers greater than 5. In LINQ, using Method syntax:

```
var MethodSyntax = integerList.Where(obj => obj > 5).ToList();
```

Query syntax takes its inspiration from SQL SELECT statements. The major difference is that the select clause, which controls the shape of the resulting data, comes last in LINQ. That means the LINQ from clause is first. The from clause determines the source of the data. The same problem as above using Query syntax:

```
var QuerySyntax = from obj in integerList
where obj > 5
select obj;
```

In the following exercises, you will see examples and problems processing data sets using traditional loops and LINQ. In LINQ you will see examples in Method and Query syntax.

Starting Code

1. Create a new console project. Name it **LINQActors**.
2. Add the following class to the project.

```
namespace LINQActors
{
    class Actor
    {

        public string Name { get; set; }
        public decimal TotalGross { get; set; }
        public int MovieCount { get; set; }
        public decimal AvgPerMovie { get; set; }
        public string TopMovie { get; set; }
        public decimal TopMovieGross { get; set; }

    }
}
```

3. Add a **toString** method that will display each property only if it has a value. We will use the **ternary** operator to check if the property value is null or has a value.

```
6   {
        18 references
7       class Actor
8       {
9
            18 references
10          public string Name { get; set; }
            6 references
11          public decimal TotalGross { get; set; }
            12 references
12          public int MovieCount { get; set; }
            3 references
13          public decimal AvgPerMovie { get; set; }
            18 references
14          public string TopMovie { get; set; }
            3 references
15          public decimal TopMovieGross { get; set; }
16
            0 references
17          public override string ToString()
18          {
19              return "Name: " + Name +
20                  ( TotalGross != 0 ? " TotalGross: " + TotalGross : null ) +
21                   (MovieCount != 0 ? " MovieCount: " + MovieCount : null ) +
22                   (AvgPerMovie != 0 ? " AvgPerMovie: " + AvgPerMovie : null) +
23                   (TopMovie != null ? " TopMovie: " + TopMovie : null ) +
24                   (TopMovieGross != 0 ? " TopMovieGross: " + TopMovieGross : null) +
25                  "\n";
26          }
27      }
```

4. In the **Program.cs** file, add the following string of data that will be used to create a list of actors. You can copy and paste the code below the screen capture image.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        string data = @"[
        [""Harrison Ford"", 4871.7, 41, 118.8, ""Star Wars: The Force Awakens"", 936.7],
        [""Samuel L. Jackson"", 4772.8, 69, 69.2, ""The Avengers"", 623.4],
        [""Morgan Freeman"", 4468.3, 61, 73.3, ""The Dark Knight"", 534.9],
        [""Tom Hanks"", 4340.8, 44, 98.7, ""Toy Story 3"", 415],
        [""Robert Downey Jr."", 3947.3, 53, 74.5, ""The Avengers"", 623.4],
        [""Eddie Murphy"", 3810.4, 38, 100.3, ""Shrek 2"", 441.2],
        [""Tom Cruise"", 3587.2, 36, 99.6, ""War of the Worlds"", 234.3],
        [""Johnny Depp"", 3368.6, 45, 74.9, ""Dead Man's Chest"", 423.3],
        [""Michael Caine"", 3351.5, 58, 57.8, ""The Dark Knight"", 534.9],
        [""Scarlett Johansson"", 3341.2, 37, 90.3, ""The Avengers"", 623.4],
        [""Gary Oldman"", 3294, 38, 86.7, ""The Dark Knight"", 534.9],
        [""Robin Williams"", 3279.3, 49, 66.9, ""Night at the Museum"", 250.9],
        [""Bruce Willis"", 3189.4, 60, 53.2, ""Sixth Sense"", 293.5],
        [""Stellan Skarsgard"", 3175, 43, 73.8, ""The Avengers"", 623.4],
        [""Anthony Daniels"", 3162.9, 7, 451.8, ""Star Wars: The Force Awakens"", 936.7],
        [""Ian McKellen"", 3150.4, 31, 101.6, ""Return of the King"", 377.8],
        [""Will Smith"", 3149.1, 24, 131.2, ""Independence Day"", 306.2],
        [""Stanley Tucci"", 3123.9, 50, 62.5, ""Catching Fire"", 424.7],
        [""Matt Damon"", 3107.3, 39, 79.7, ""The Martian"", 228.4],
        [""Robert DeNiro"", 3081.3, 79, 39, ""Meet the Fockers"", 279.3],
        [""Cameron Diaz"", 3031.7, 34, 89.2, ""Shrek 2"", 441.2],
        [""Liam Neeson"", 2942.7, 63, 46.7, ""The Phantom Menace"", 474.5],
        [""Andy Serkis"", 2890.6, 23, 125.7, ""Star Wars: The Force Awakens"", 936.7],
        [""Don Cheadle"", 2885.4, 34, 84.9, ""Avengers: Age of Ultron"", 459],
        [""Ben Stiller"", 2827, 37, 76.4, ""Meet the Fockers"", 279.3],
        [""Helena Bonham Carter"", 2822, 36, 78.4, ""Harry Potter / Deathly Hallows(P2)"", 381],
        [""Orlando Bloom"", 2815.8, 17, 165.6, ""Dead Man's Chest"", 423.3],
        [""Woody Harrelson"", 2815.8, 50, 56.3, ""Catching Fire"", 424.7],
        [""Cate Blanchett"", 2802.6, 39, 71.9, ""Return of the King"", 377.8],
        [""Julia Roberts"", 2735.3, 42, 65.1, ""Ocean's Eleven"", 183.4],
        [""Elizabeth Banks"", 2726.3, 35, 77.9, ""Catching Fire"", 424.7],
        [""Ralph Fiennes"", 2715.3, 36, 75.4, ""Harry Potter / Deathly Hallows(P2)"", 381],
        [""Emma Watson"", 2681.9, 17, 157.8, ""Harry Potter / Deathly Hallows(P2)"", 381],
        [""Tommy Lee Jones"", 2681.3, 46, 58.3, ""Men in Black"", 250.7],
        [""Brad Pitt"", 2680.9, 40, 67, ""World War Z"", 202.4],
        [""Adam Sandler"", 2661, 32, 83.2, ""Hotel Transylvania 2"", 169.7],
        [""Daniel Radcliffe"", 2634.4, 17, 155, ""Harry Potter / Deathly Hallows(P2)"", 381],
        [""Jonah Hill"", 2605.1, 29, 89.8, ""The LEGO Movie"", 257.8],
        [""Owen Wilson"", 2602.3, 39, 66.7, ""Night at the Museum"", 250.9],
        [""Idris Elba"", 2580.6, 26, 99.3, ""Avengers: Age of Ultron"", 459],
        [""Bradley Cooper"", 2557.7, 25, 102.3, ""American Sniper"", 350.1],
        [""Mark Wahlberg"", 2549.8, 36, 70.8, ""Transformers 4"", 245.4],
        [""Jim Carrey"", 2545.2, 27, 94.3, ""The Grinch"", 260],
        [""Dustin Hoffman"", 2522.1, 43, 58.7, ""Meet the Fockers"", 279.3],
        [""Leonardo DiCaprio"", 2518.3, 25, 100.7, ""Titanic"", 658.7],
        [""Jeremy Renner"", 2500.3, 21, 119.1, ""The Avengers"", 623.4],
        [""Philip Seymour Hoffman"", 2463.7, 40, 61.6, ""Catching Fire"", 424.7],
        [""Sandra Bullock"", 2462.6, 35, 70.4, ""Minions"", 336],
        [""Chris Evans"", 2457.8, 23, 106.9, ""The Avengers"", 623.4],
        [""Anne Hathaway"", 2416.5, 25, 96.7, ""The Dark Knight Rises"", 448.1]
    ";
```

```
string data = @"[
                    [""Harrison Ford"", 4871.7, 41, 118.8, ""Star Wars: The Force Awakens"", 936.7],
                    [""Samuel L. Jackson"", 4772.8, 69, 69.2, ""The Avengers"", 623.4],
                    [""Morgan Freeman"", 4468.3, 61, 73.3, ""The Dark Knight"", 534.9],
                    [""Tom Hanks"", 4340.8, 44, 98.7, ""Toy Story 3"", 415],
                    [""Robert Downey Jr."", 3947.3, 53, 74.5, ""The Avengers"", 623.4],
                    [""Eddie Murphy"", 3810.4, 38, 100.3, ""Shrek 2"", 441.2],
                    [""Tom Cruise"", 3587.2, 36, 99.6, ""War of the Worlds"", 234.3],
                    [""Johnny Depp"", 3368.6, 45, 74.9, ""Dead Man's Chest"", 423.3],
                    [""Michael Caine"", 3351.5, 58, 57.8, ""The Dark Knight"", 534.9],
                    [""Scarlett Johansson"", 3341.2, 37, 90.3, ""The Avengers"", 623.4],
                    [""Gary Oldman"", 3294, 38, 86.7, ""The Dark Knight"", 534.9],
                    [""Robin Williams"", 3279.3, 49, 66.9, ""Night at the Museum"", 250.9],
                    [""Bruce Willis"", 3189.4, 60, 53.2, ""Sixth Sense"", 293.5],
                    [""Stellan Skarsgard"", 3175, 43, 73.8, ""The Avengers"", 623.4],
                    [""Anthony Daniels"", 3162.9, 7, 451.8, ""Star Wars: The Force Awakens"", 936.7],
                    [""Ian McKellen"", 3150.4, 31, 101.6, ""Return of the King"", 377.8],
                    [""Will Smith"", 3149.1, 24, 131.2, ""Independence Day"", 306.2],
                    [""Stanley Tucci"", 3123.9, 50, 62.5, ""Catching Fire"", 424.7],
                    [""Matt Damon"", 3107.3, 39, 79.7, ""The Martian"", 228.4],
                    [""Robert DeNiro"", 3081.3, 79, 39, ""Meet the Fockers"", 279.3],
                    [""Cameron Diaz"", 3031.7, 34, 89.2, ""Shrek 2"", 441.2],
                    [""Liam Neeson"", 2942.7, 63, 46.7, ""The Phantom Menace"", 474.5],
                    [""Andy Serkis"", 2890.6, 23, 125.7, ""Star Wars: The Force Awakens"", 936.7],
                    [""Don Cheadle"", 2885.4, 34, 84.9, ""Avengers: Age of Ultron"", 459],
                    [""Ben Stiller"", 2827, 37, 76.4, ""Meet the Fockers"", 279.3],
                    [""Helena Bonham Carter"", 2822, 36, 78.4, ""Harry Potter / Deathly Hallows(P2)"", 381],
                    [""Orlando Bloom"", 2815.8, 17, 165.6, ""Dead Man's Chest"", 423.3],
                    [""Woody Harrelson"", 2815.8, 50, 56.3, ""Catching Fire"", 424.7],
                    [""Cate Blanchett"", 2802.6, 39, 71.9, ""Return of the King"", 377.8],
                    [""Julia Roberts"", 2735.3, 42, 65.1, ""Ocean's Eleven"", 183.4],
                    [""Elizabeth Banks"", 2726.3, 35, 77.9, ""Catching Fire"", 424.7],
```

```
            [""Ralph Fiennes"", 2715.3, 36, 75.4, ""Harry Potter / Deathly Hallows(P2)"", 381],
            [""Emma Watson"", 2681.9, 17, 157.8, ""Harry Potter / Deathly Hallows(P2)"", 381],
            [""Tommy Lee Jones"", 2681.3, 46, 58.3, ""Men in Black"", 250.7],
            [""Brad Pitt"", 2680.9, 40, 67, ""World War Z"", 202.4],
            [""Adam Sandler"", 2661, 32, 83.2, ""Hotel Transylvania 2"", 169.7],
            [""Daniel Radcliffe"", 2634.4, 17, 155, ""Harry Potter / Deathly Hallows(P2)"", 381],
            [""Jonah Hill"", 2605.1, 29, 89.8, ""The LEGO Movie"", 257.8],
            [""Owen Wilson"", 2602.3, 39, 66.7, ""Night at the Museum"", 250.9],
            [""Idris Elba"", 2580.6, 26, 99.3, ""Avengers: Age of Ultron"", 459],
            [""Bradley Cooper"", 2557.7, 25, 102.3, ""American Sniper"", 350.1],
            [""Mark Wahlberg"", 2549.8, 36, 70.8, ""Transformers 4"", 245.4],
            [""Jim Carrey"", 2545.2, 27, 94.3, ""The Grinch"", 260],
            [""Dustin Hoffman"", 2522.1, 43, 58.7, ""Meet the Fockers"", 279.3],
            [""Leonardo DiCaprio"", 2518.3, 25, 100.7, ""Titanic"", 658.7],
            [""Jeremy Renner"", 2500.3, 21, 119.1, ""The Avengers"", 623.4],
            [""Philip Seymour Hoffman"", 2463.7, 40, 61.6, ""Catching Fire"", 424.7],
            [""Sandra Bullock"", 2462.6, 35, 70.4, ""Minions"", 336],
            [""Chris Evans"", 2457.8, 23, 106.9, ""The Avengers"", 623.4],
            [""Anne Hathaway"", 2416.5, 25, 96.7, ""The Dark Knight Rises"", 448.1]
        ]";
```

3. Parse the JSON-formatted data into a list of Actor objects.

```
        List<Actor> actorList = new List<Actor>();

        JArray a = JArray.Parse(data);

        foreach (var item in a)
        {
            Actor actor = new Actor
            {

                Name = (string)item[0],
                TotalGross = (decimal)item[1],
                MovieCount = (int)item[2],
                AvgPerMovie = (decimal)item[3],
                TopMovie = (string)item[4],
                TopMovieGross = (decimal)item[5]

            };
            actorList.Add(actor);
        }


        Console.WriteLine("Actors count = " + actorList.Count);
        Console.ReadLine();
```

4. Run the program to see if the data is parsed successfully.

```
C:\Users\shadsluiter\source\repos\

Actors count = 50
```

Three Ways to Filter a List

Now that we have built the actor list, we will filter the list using three techniques: for loop, LINQ Query, and LINQ Method expressions. In each case, we are supposed to get the same results. The point of this exercise is to be able to use all three types of selection methods.

Example 1 – Create a List of the Wealthiest Actors

1. Add the following code section to create a list of the richest actors using a for-each loop.

```
// standard for loop to select the richest actors from the list
var richActors1 = new List<Actor>();

foreach(Actor ac in actorList)
{
    if (ac.TotalGross > 4000)
    {
        richActors1.Add(ac);
    }
}
Console.WriteLine("Rich actors that made more than $400 million");
Console.WriteLine(printList(richActors1));
Console.ReadLine();
```

2. Run the program. You should see only the richest actors in the list.

```
C:\Users\shadsluiter\source\repos\LINQActors\bin\Debug\netcoreapp3.1\LINQActors.exe
Actors count = 50

Rich actors that made more than $400 million
Name: Harrison Ford TotalGross: 4871.7 MovieCount: 41 AvgPerMovie: 118.8 TopMovie: Star Wars: The Force Awakens TopMovie
Gross: 936.7
Name: Samuel L. Jackson TotalGross: 4772.8 MovieCount: 69 AvgPerMovie: 69.2 TopMovie: The Avengers TopMovieGross: 623.4
Name: Morgan Freeman TotalGross: 4468.3 MovieCount: 61 AvgPerMovie: 73.3 TopMovie: The Dark Knight TopMovieGross: 534.9
Name: Tom Hanks TotalGross: 4340.8 MovieCount: 44 AvgPerMovie: 98.7 TopMovie: Toy Story 3 TopMovieGross: 415

Total items printed:4
```

3. Run the program. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

4. Add the following code to select the same list of actors, but this time, use a LINQ Query statement to filter the list.

```
// using LINQ to select the richest actors
var richActors2 = from actor in actorList
                  where actor.TotalGross > 4000
                  select actor;

Console.WriteLine("Rich actors selected using LINQ: ");
Console.WriteLine(printList(richActors2));
Console.ReadLine();
```

5. Run the program. You should see the same list of people selected.

```
C:\Users\shadsluiter\source\repos\LINQActors\bin\Debug\netcoreapp3.1\LINQActors.exe
Actors count = 50

Rich actors that made more than $400 million
Name: Harrison Ford TotalGross: 4871.7 MovieCount: 41 AvgPerMovie: 118.8 TopMovie: Star Wars: The Force Awakens TopMovie
Gross: 936.7
Name: Samuel L. Jackson TotalGross: 4772.8 MovieCount: 69 AvgPerMovie: 69.2 TopMovie: The Avengers TopMovieGross: 623.4
Name: Morgan Freeman TotalGross: 4468.3 MovieCount: 61 AvgPerMovie: 73.3 TopMovie: The Dark Knight TopMovieGross: 534.9
Name: Tom Hanks TotalGross: 4340.8 MovieCount: 44 AvgPerMovie: 98.7 TopMovie: Toy Story 3 TopMovieGross: 415

Total items printed:4



Rich actors selected using LINQ:
Name: Harrison Ford TotalGross: 4871.7 MovieCount: 41 AvgPerMovie: 118.8 TopMovie: Star Wars: The Force Awakens TopMovie
Gross: 936.7
Name: Samuel L. Jackson TotalGross: 4772.8 MovieCount: 69 AvgPerMovie: 69.2 TopMovie: The Avengers TopMovieGross: 623.4
Name: Morgan Freeman TotalGross: 4468.3 MovieCount: 61 AvgPerMovie: 73.3 TopMovie: The Dark Knight TopMovieGross: 534.9
Name: Tom Hanks TotalGross: 4340.8 MovieCount: 44 AvgPerMovie: 98.7 TopMovie: Toy Story 3 TopMovieGross: 415

Total items printed:4
```

6. Run the program. Take a screen shot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

7. Add the following code to select the same group, but use LINQ Method statements to perform the selection.

```
// using Lambda with where and select
var richActors3 = actorList.Where(a => a.TotalGross > 4000);

Console.WriteLine("Rich actors selected using Lambda expressions. ");
Console.WriteLine(printList(richActors3));
Console.ReadLine();
```

The Lambda expression is a => a.TotalGross > 4000. Lambda expressions are a quick way to express functions. They are relatively new to programming languages but have been widely embraced by C#, Java, and JavaScript developers. A lambda tutorial: ([LINQ - Lambda Expressions (tutorialspoint.com)](). In this case, if the lambda function result is true, then the item is included in the output.

8. Run the program. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Example 2 – Create a List of Actors Who Have Starred in an Even Number of Films

A rather contrived example of another filter function is to determine whether or not an actor starred in an even or odd number of films.

1. Add the following code to create a list of even numbered film events.

```
122
123             // even number of films
124
125             // standard foreach loop
126             var actorswithEvenNumber1 = new List<Actor>();
127             foreach (Actor a2 in actorList)
128             {
129                 if (a2.MovieCount % 2 == 0)
130                 {
131                     actorswithEvenNumber1.Add(
132                         new Actor
133                         {
134                             Name = a2.Name,
135                             MovieCount = a2.MovieCount
136                         }
137                     );
138                 }
139             }
140             Console.WriteLine("Even number actors = ");
141             Console.WriteLine(printList(actorswithEvenNumber1));
142             Console.ReadLine();
```

Can't re-use the letter "a" so we will loop with a2.

Create a new instance of the actor with only two properties set to make it easier to read.

2. Run the program to verify that only even-numbered actors appear in the list. It looks like **20 actors had an even number** of films. Notice that the **toString** method in the **Actor** class allows us to display only the properties that have a value. The other properties are ignored.

```
C:\Users\shadsluiter\source\repos\LINQActors\bin\Debug\ne

Even number actors =
Name: Tom Hanks MovieCount: 44
Name: Eddie Murphy MovieCount: 38
Name: Tom Cruise MovieCount: 36
Name: Michael Caine MovieCount: 58
Name: Gary Oldman MovieCount: 38
Name: Bruce Willis MovieCount: 60
Name: Will Smith MovieCount: 24
Name: Stanley Tucci MovieCount: 50
Name: Cameron Diaz MovieCount: 34
Name: Don Cheadle MovieCount: 34
Name: Helena Bonham Carter MovieCount: 36
Name: Woody Harrelson MovieCount: 50
Name: Julia Roberts MovieCount: 42
Name: Ralph Fiennes MovieCount: 36
Name: Tommy Lee Jones MovieCount: 46
Name: Brad Pitt MovieCount: 40
Name: Adam Sandler MovieCount: 32
Name: Idris Elba MovieCount: 26
Name: Mark Wahlberg MovieCount: 36
Name: Philip Seymour Hoffman MovieCount: 40

Total items printed:20
```

3. Run the program. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

4. Now let's do the same list but using a LINQ Query:

```
143
144             // using LINQ
145             var actorsWithEvenNumberOfFilms2 = from actor in actorList
146                                                where actor.MovieCount % 2 == 0
147                                                select new Actor
148                                                {
149                                                    Name = actor.Name,
150                                                    MovieCount = actor.MovieCount
151                                                };
152             Console.WriteLine("Even number of films using LINQ : " );
153             Console.WriteLine(printList(actorsWithEvenNumberOfFilms2));
154             Console.ReadLine();
155
```

5. Run the program and verify.

```
Even number of films using LINQ :
Name: Tom Hanks MovieCount: 44
Name: Eddie Murphy MovieCount: 38
Name: Tom Cruise MovieCount: 36
Name: Michael Caine MovieCount: 58
Name: Gary Oldman MovieCount: 38
Name: Bruce Willis MovieCount: 60
Name: Will Smith MovieCount: 24
Name: Stanley Tucci MovieCount: 50
Name: Cameron Diaz MovieCount: 34
Name: Don Cheadle MovieCount: 34
Name: Helena Bonham Carter MovieCount: 36
Name: Woody Harrelson MovieCount: 50
Name: Julia Roberts MovieCount: 42
Name: Ralph Fiennes MovieCount: 36
Name: Tommy Lee Jones MovieCount: 46
Name: Brad Pitt MovieCount: 40
Name: Adam Sandler MovieCount: 32
Name: Idris Elba MovieCount: 26
Name: Mark Wahlberg MovieCount: 36
Name: Philip Seymour Hoffman MovieCount: 40

Total items printed:20
```

6. Run the program. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

7. Now let's do the same filter exercise but using a LINQ Method expression:

```
// using Lambda
var actorsWithEvenNumberOfFilms3 = actorList
    .Where(a => a.MovieCount % 2 == 0)
    .Select(x => new Actor
        {
            Name = x.Name,
            MovieCount = x.MovieCount
        }
    );

Console.WriteLine("Even number of films selected using Lambda");
Console.WriteLine(printList(actorsWithEvenNumberOfFilms3));

Console.ReadLine();
```

9. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Example 3 – In This Query, We're Going to Select Only the Actors Whose Top-Grossing Film was a Star Wars Movie

1. Add the following for loop to find the Star Wars stars.

```
// example 3 - find star wars actors.
var starWarsActors1 = new List<Actor>();

foreach (Actor a3 in actorList)
{
    if (a3.TopMovie.Contains("Star Wars"))
    {
        starWarsActors1.Add(new Actor { Name = a3.Name, TopMovie = a3.TopMovie });
    }
}

Console.WriteLine("Star wars actors made with for loop: ");
Console.WriteLine(printList(starWarsActors1));

Console.ReadLine();
```

2. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

```
Star wars actors made with for loop:
Name: Harrison Ford TopMovie: Star Wars: The Force Awakens
Name: Anthony Daniels TopMovie: Star Wars: The Force Awakens
Name: Andy Serkis TopMovie: Star Wars: The Force Awakens

Total items printed:3
```

3. Create a LINQ statement to select Star Wars actors.

```
// star wars using LINQ
var starWarsActors = from actor in actorList
                     where actor.TopMovie.Contains("Star Wars")
                     select new Actor
                     {
                         Name = actor.Name,
                         TopMovie = actor.TopMovie
                     };

Console.WriteLine("Star wars actors made with LINQ : ");
Console.WriteLine(  printList(starWarsActors));

Console.ReadLine();
```

4. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated

5. Do the same filter but with a LINQ Method expression:

```
// star wars with Lambda
var starWarsActors3 = actorList
    .Where(a => a.TopMovie.Contains("Star Wars"))
    .Select(x => new Actor
    {
        Name = x.Name,
        TopMovie = x.TopMovie
    });

Console.WriteLine("Star wars actors made with Lambda : ");
Console.WriteLine(printList(starWarsActors3));

Console.ReadLine();
```

3. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Example 3 – Grouping

LINQ expressions have the ability to aggregate, group, count, and sum a list of values.

1. Add the following code to demonstrate a group method as LINQ Query:

```
// group items using LINQ
var groupedByMovies = from actor in actorList
                      group actor by actor.TopMovie into newGroup
                      orderby newGroup.Key
                      from actorinGroup in newGroup
                      select new Actor
                      {
                          Name = actorinGroup.Name,
                          TopMovie = actorinGroup.TopMovie
                      };

Console.WriteLine("Grouped by movie : ");
Console.WriteLine(printList(groupedByMovies));
Console.ReadLine();
```

2. The output should show the actors grouped according to their top-grossing movie. The movie name is the "key" that they are grouped by.

```
Grouped by movie :
Name: Bradley Cooper TopMovie: American Sniper
Name: Don Cheadle TopMovie: Avengers: Age of Ultron
Name: Idris Elba TopMovie: Avengers: Age of Ultron
Name: Stanley Tucci TopMovie: Catching Fire
Name: Woody Harrelson TopMovie: Catching Fire
Name: Elizabeth Banks TopMovie: Catching Fire
Name: Philip Seymour Hoffman TopMovie: Catching Fire
Name: Johnny Depp TopMovie: Dead Man's Chest
Name: Orlando Bloom TopMovie: Dead Man's Chest
Name: Helena Bonham Carter TopMovie: Harry Potter / Deathly Hallows(P2)
Name: Ralph Fiennes TopMovie: Harry Potter / Deathly Hallows(P2)
Name: Emma Watson TopMovie: Harry Potter / Deathly Hallows(P2)
Name: Daniel Radcliffe TopMovie: Harry Potter / Deathly Hallows(P2)
Name: Adam Sandler TopMovie: Hotel Transylvania 2
Name: Will Smith TopMovie: Independence Day
Name: Robert DeNiro TopMovie: Meet the Fockers
Name: Ben Stiller TopMovie: Meet the Fockers
Name: Dustin Hoffman TopMovie: Meet the Fockers
Name: Tommy Lee Jones TopMovie: Men in Black
Name: Sandra Bullock TopMovie: Minions
Name: Robin Williams TopMovie: Night at the Museum
Name: Owen Wilson TopMovie: Night at the Museum
Name: Julia Roberts TopMovie: Ocean's Eleven
Name: Ian McKellen TopMovie: Return of the King
Name: Cate Blanchett TopMovie: Return of the King
Name: Eddie Murphy TopMovie: Shrek 2
Name: Cameron Diaz TopMovie: Shrek 2
Name: Bruce Willis TopMovie: Sixth Sense
```

3. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

4. Do the same grouping but with a LINQ Method expression:

```
// grouping with Lambda
var group2 = actorList.GroupBy(a => a.TopMovie)
    .OrderBy(b => b.Key)
    .SelectMany(c => c)
    .Select( d => new Actor {
        Name = d.Name,
        TopMovie = d.TopMovie
    });

Console.WriteLine("Grouped by movie with lambda : ");
Console.WriteLine(printList(group2));

Console.ReadLine();
```

5. Run the program to verify it is working. Take a screenshot of the application. Paste the image into a Word document. Add a caption describing what is being demonstrated.

Write It Yourself Challenges

Now that you have seen multiple examples of each type of filtering, write three expressions for each of these scenarios.

Challenge #1
Show the list of the actors who are "poor." You determine what level of "poverty" qualifies an actor to be included in the list. First, use a List and foreach loop. Second, use a LINQ Query. Third, create a LINQ Method expression. Run the program to verify they are working. Take screenshots of the application. Paste the images into a Word document. Add captions describing what is being demonstrated.

Challenge #2
Choose your own rule to use in selecting actors. Choose some characteristics and write a query using LINQ Query syntax. Then write the same query using LINQ Method syntax. Explain what your query is supposed to show. Run the program to verify they are working. Take screenshots of the application. Paste the images into a Word document. Add captions describing what is being demonstrated.

Deliverables:

1. Zip file containing all source code.
2. Word document containing screenshots of the application being run. Be sure to demonstrate all features that were created in the tutorial, as well as the challenges.

Screenshot Checklist

Student Scores

1. All scores
2. A students
3. Sorted Scores
4. B students

Actors

Rich Actors

5. forloop
6. LINQ Query Syntax
7. Query Syntax

Even Number of Movies

8. For-each loop
9. LINQ Query syntax
10. LINQ Method syntax

Star Wars Filter

11. LINQ Query syntax
12. LINQ Method syntax

Group by Top-Grossing Movie

13. LINQ Query syntax
14. LINQ Method syntax

Challenges

Poor Actors

15. For loop
16. LINQ Query Syntax
17. Query Syntax

Your Own Query

18. LINQ Query Syntax
19. Query Syntax