LINQ - Language Integrated Query Introduction

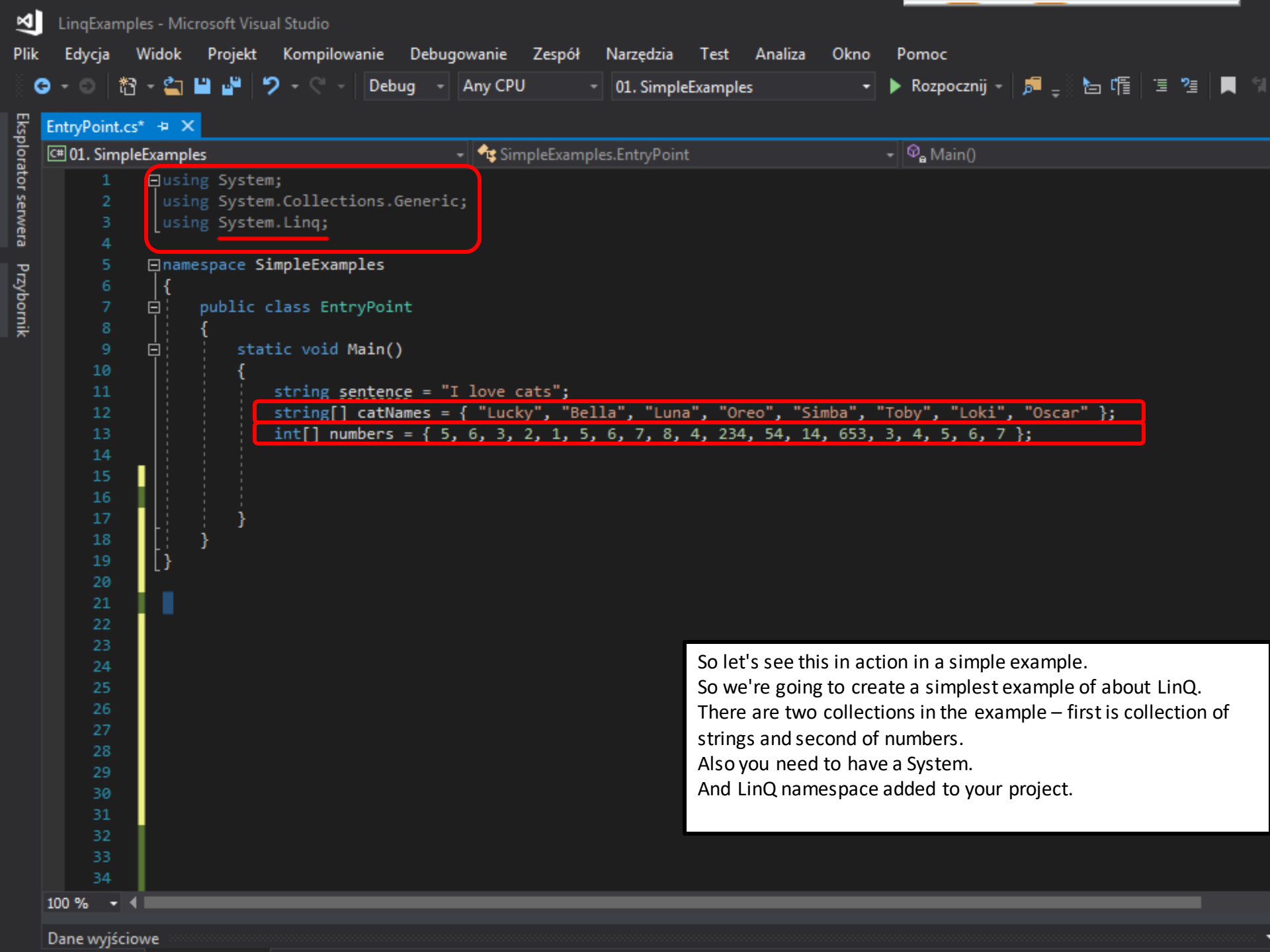
- Similar to SQL - works on Collections
- Three basic operations:
  - Get the source
  - Create the Query
  - Execute the Query

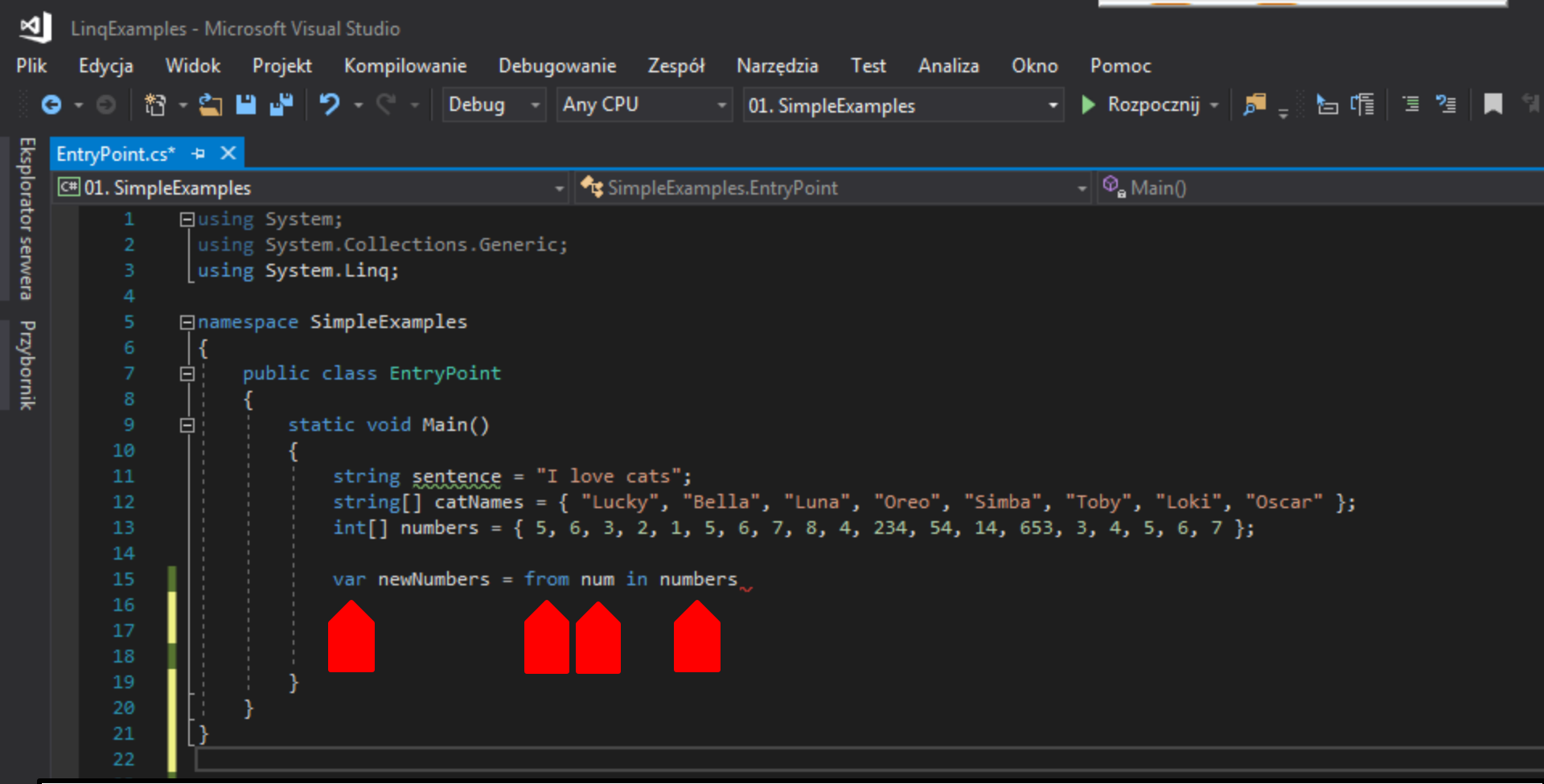
All right we're going to start linq.
LinQ means Language Integrated Query.
It's very similar to SQL with the difference that LinQ works in collections and SQL works in databases.
They both perform the same thing.
Both are designed to extract some information from a collection or database based on some conditions.
All LinQ operations can be expressed in three steps.
First we get the source then we create the query and then we execute the query.

LINQ - Language Integrated Query
Structure of a Query

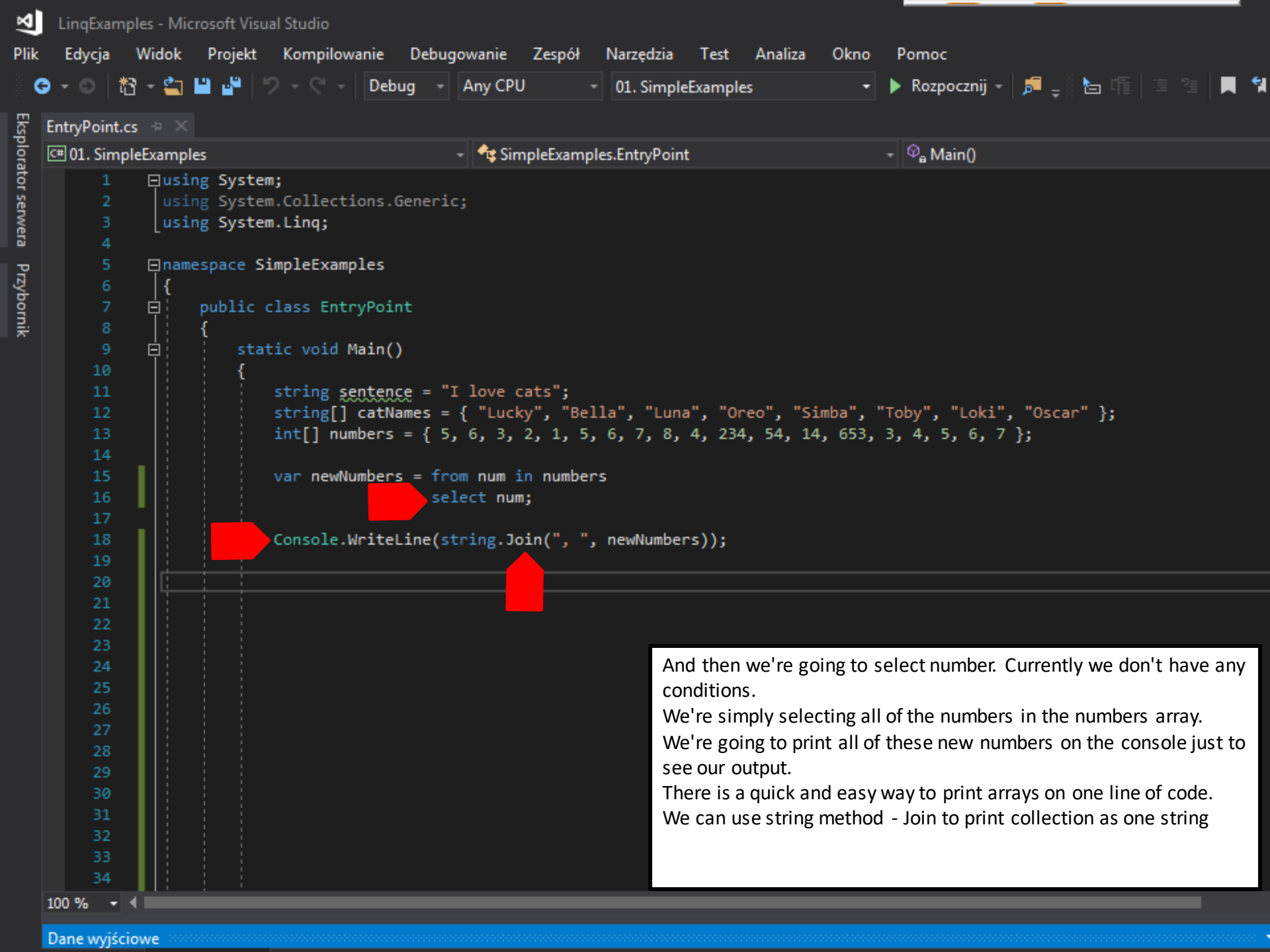
- Structure of a Query:
  - Define the source - from ... in ...
  - Define some conditions - where
  - Take the filtered output - select

Let's start analyzing the structure of LinQ queries.
The first thing that we have to do is that we have to define our source.
We start with the keyword FROM.
Here we define the variable that is going to iterate over the source.
FROM - iterating variable - IN - source.
After that we define some conditions by using the WHERE keyword
and we SELECT all of the new items that match our condition.


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs*
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
}
}
}
So let's see this in action in a simple example.
So we're going to create a simplest example of about LinQ.
There are two collections in the example – first is collection of strings and second of numbers.
Also you need to have a System.
And LinQ namespace added to your project.
100 %
Dane wyjściowe



We're going to use var data type because we're always going to use different type of collection.
Our final type may be different from that of the collection.
So you would always need to know what is a type that you need to have for your query.
So it's just easier to use var.
Get theuumbers is what I'm going to call my query.
We start with the keyword from.
So after keyword from the first thing that you're going to do is use the iterating variable. So the name of our iterating variable will be "num".
It can be anything you want you can just simply use the letter.
And then we choose what is our collection.


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var newNumbers = from num in numbers
select num;
Console.WriteLine(string.Join(", ", newNumbers));
And then we're going to select number. Currently we don't have any conditions.
We're simply selecting all of the numbers in the numbers array.
We're going to print all of these new numbers on the console just to see our output.
There is a quick and easy way to print arrays on one line of code.
We can use string method - Join to print collection as one string
100 %
Dane wyjściowe

LinqExamples - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug | Any CPU | 01. SimpleExamples | Rozpocznij

Eksplorator serwera | Przybornik

EntryPoint.cs

01. SimpleExamples | SimpleExamples.EntryPoint | Main()

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace SimpleExamples
6  {
7      public class EntryPoint
8      {
9          static void Main()
10         {
11             string sentence = "I love cats";
12             string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
13             int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
14
15             var newNumbers = from num in numbers
16                              select num;
17
18             Console.WriteLine(string.Join(", ", newNumbers));
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

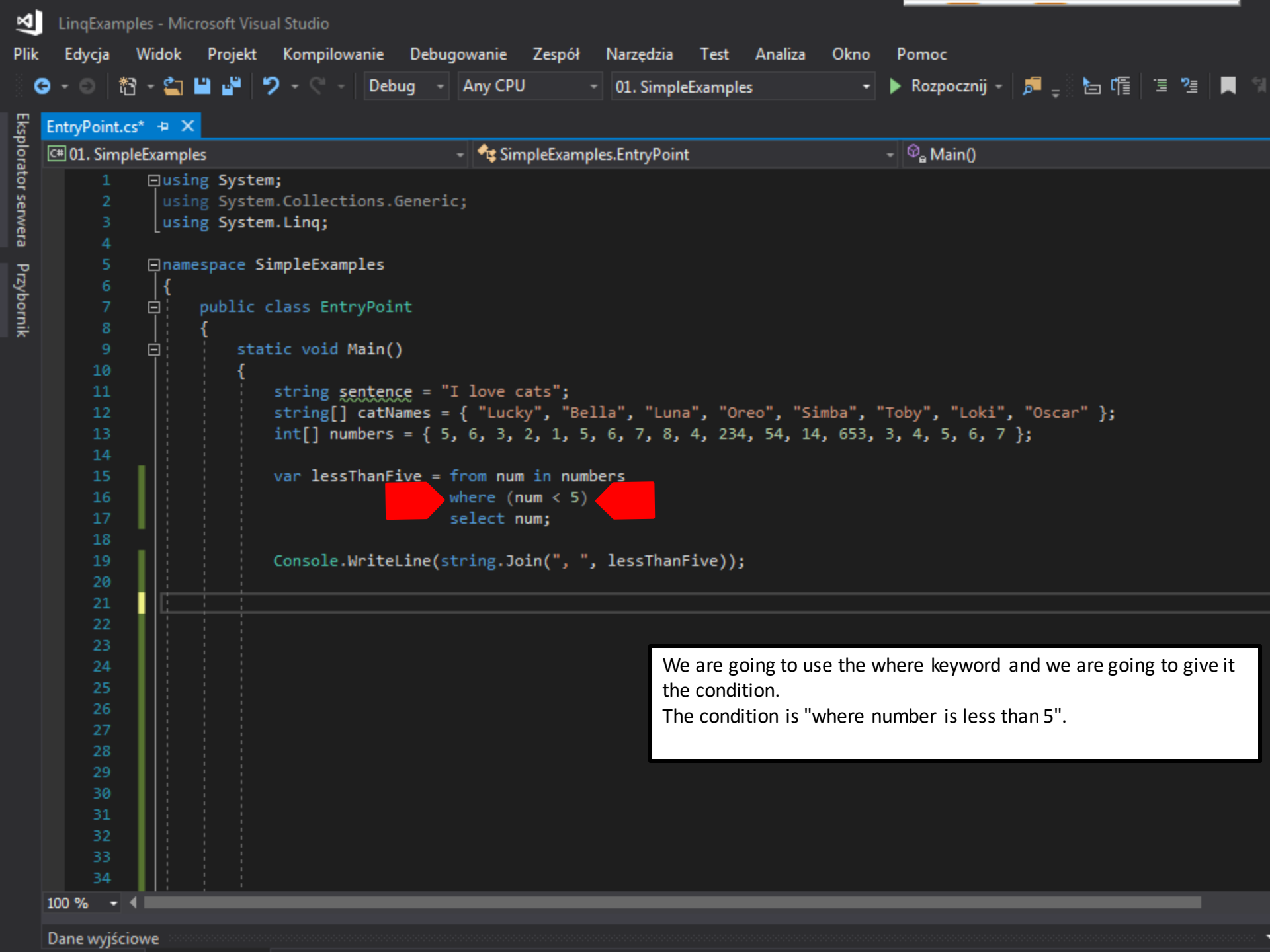
C:\Windows\system32\cmd.exe

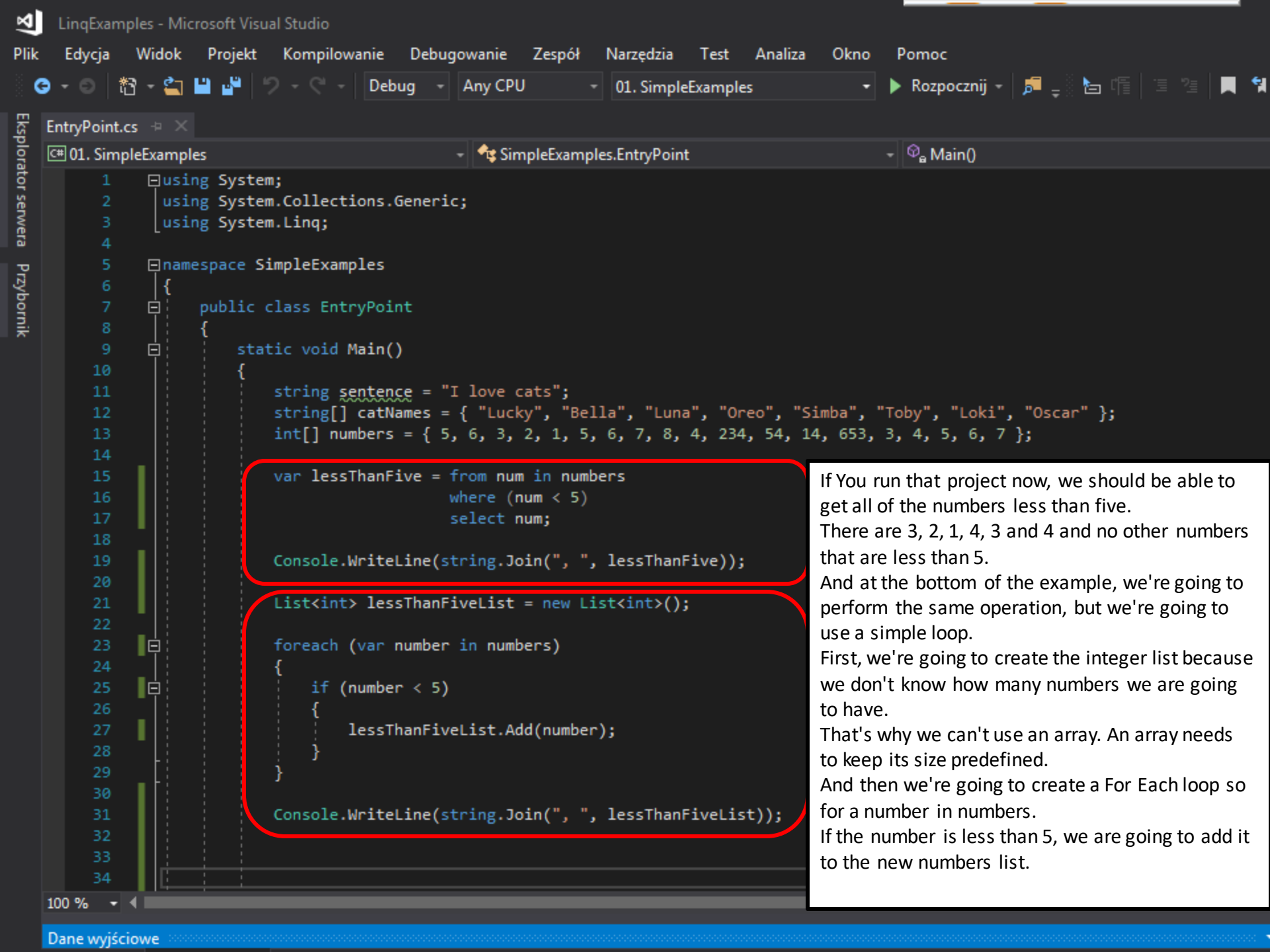
```
5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7
Aby kontynuować, naciśnij dowolny klawisz . . .
```

We get the exact same numbers that we have here in the numbers array.
Now let's say that we want all of the numbers that are less than 5.

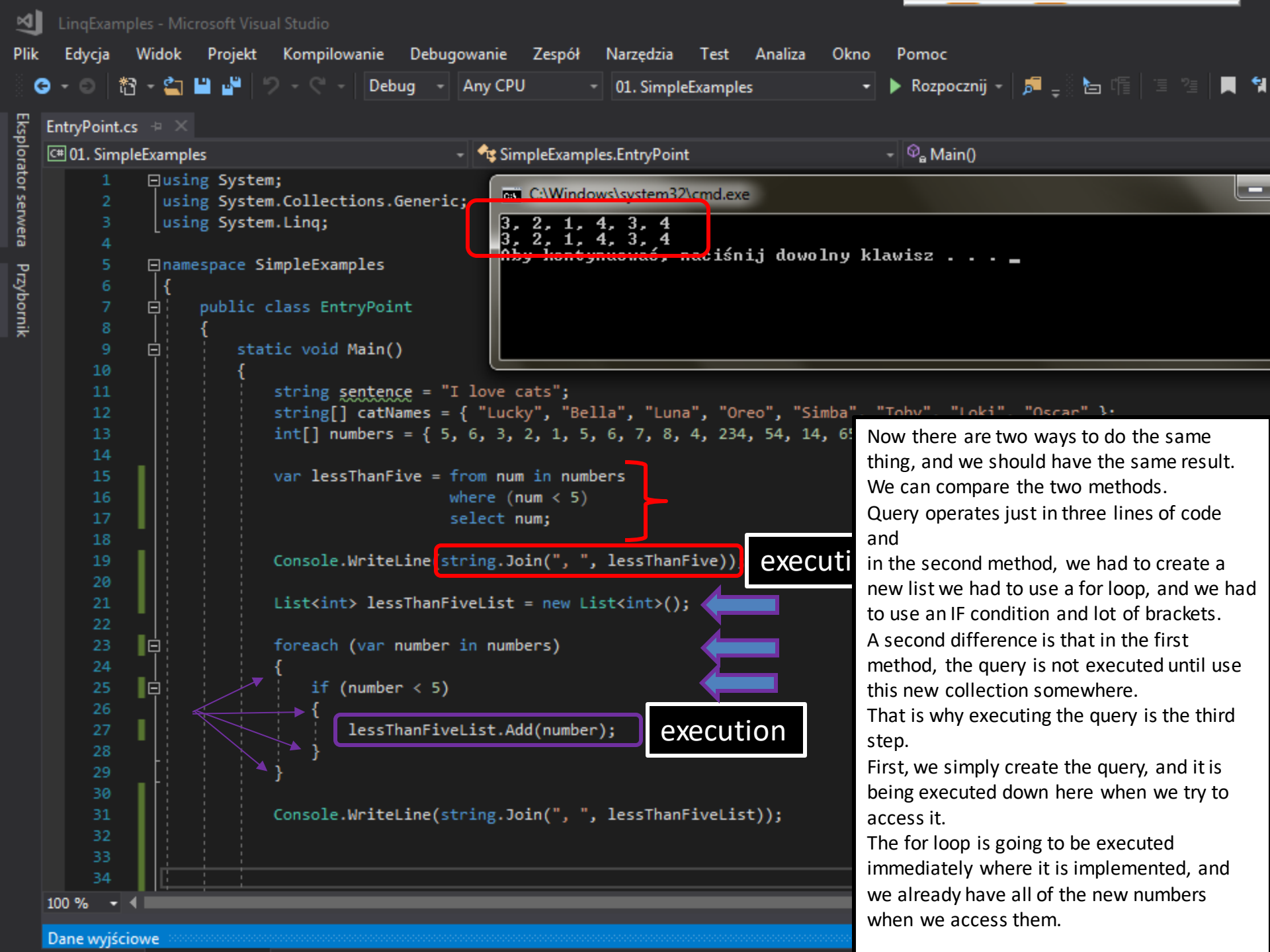
100 %

Dane wyjściowe

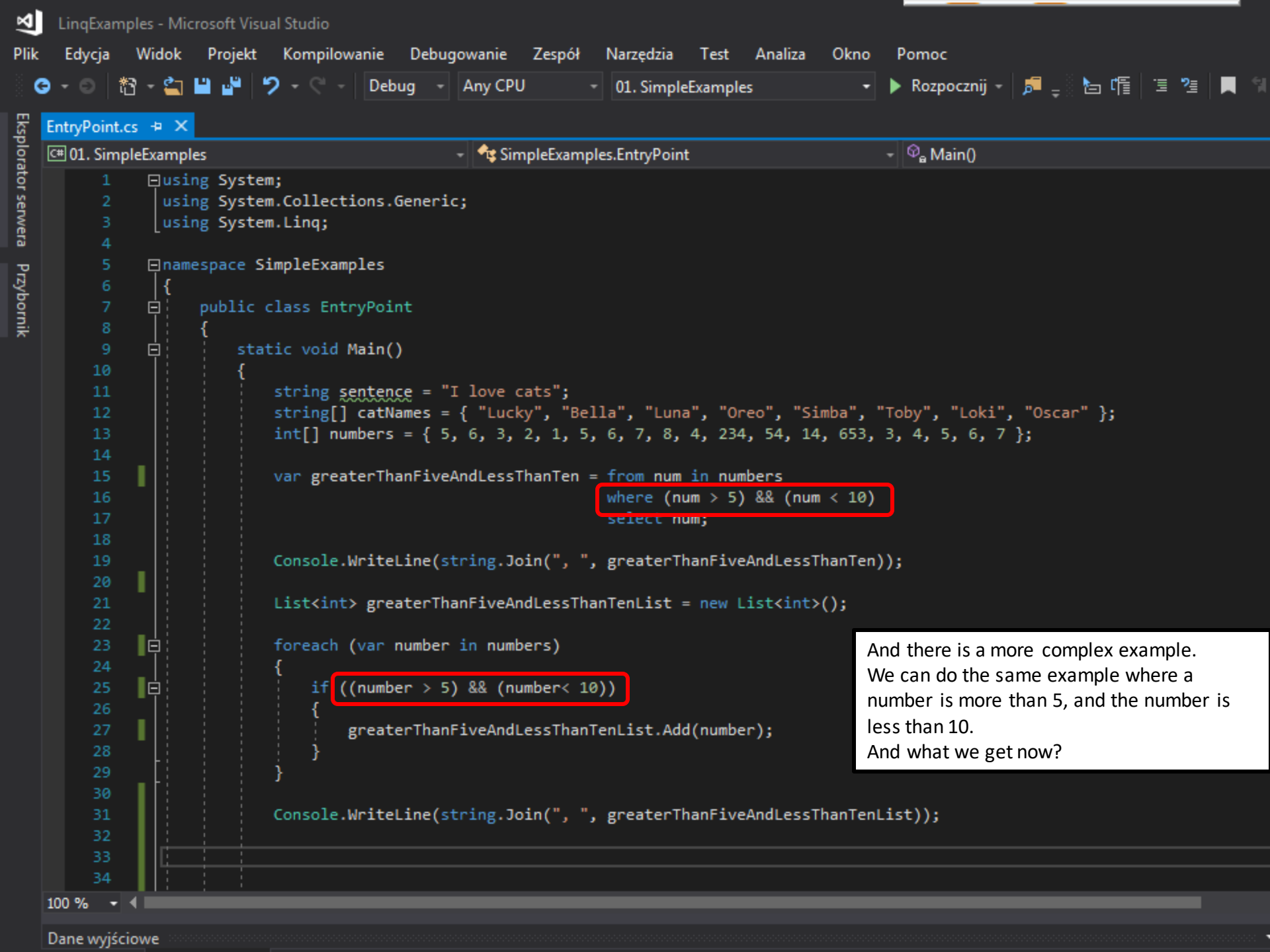

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs*
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var lessThanFive = from num in numbers
where (num < 5)
select num;
Console.WriteLine(string.Join(", ", lessThanFive));
We are going to use the where keyword and we are going to give it the condition.
The condition is "where number is less than 5".
100 %
Dane wyjściowe

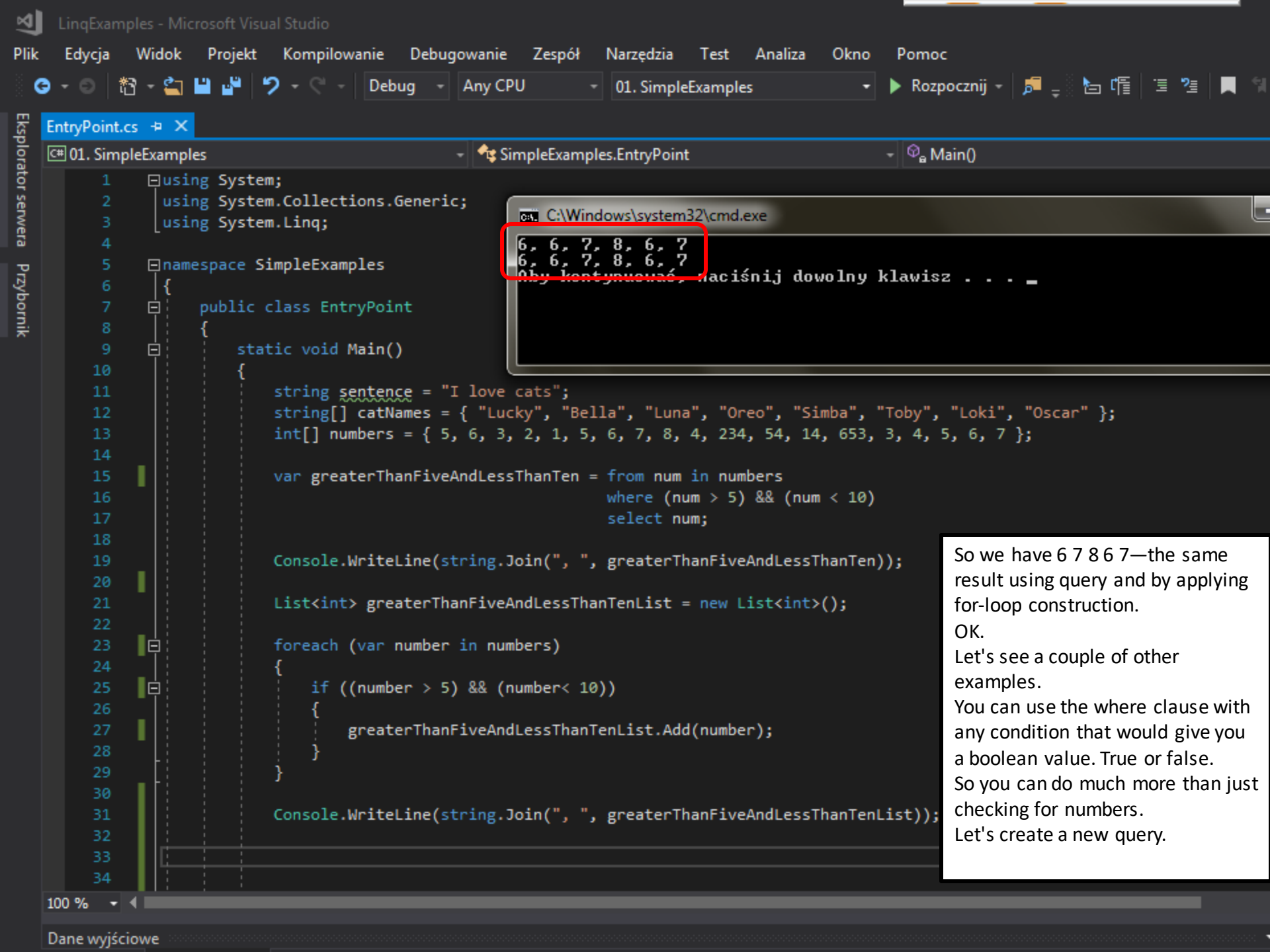


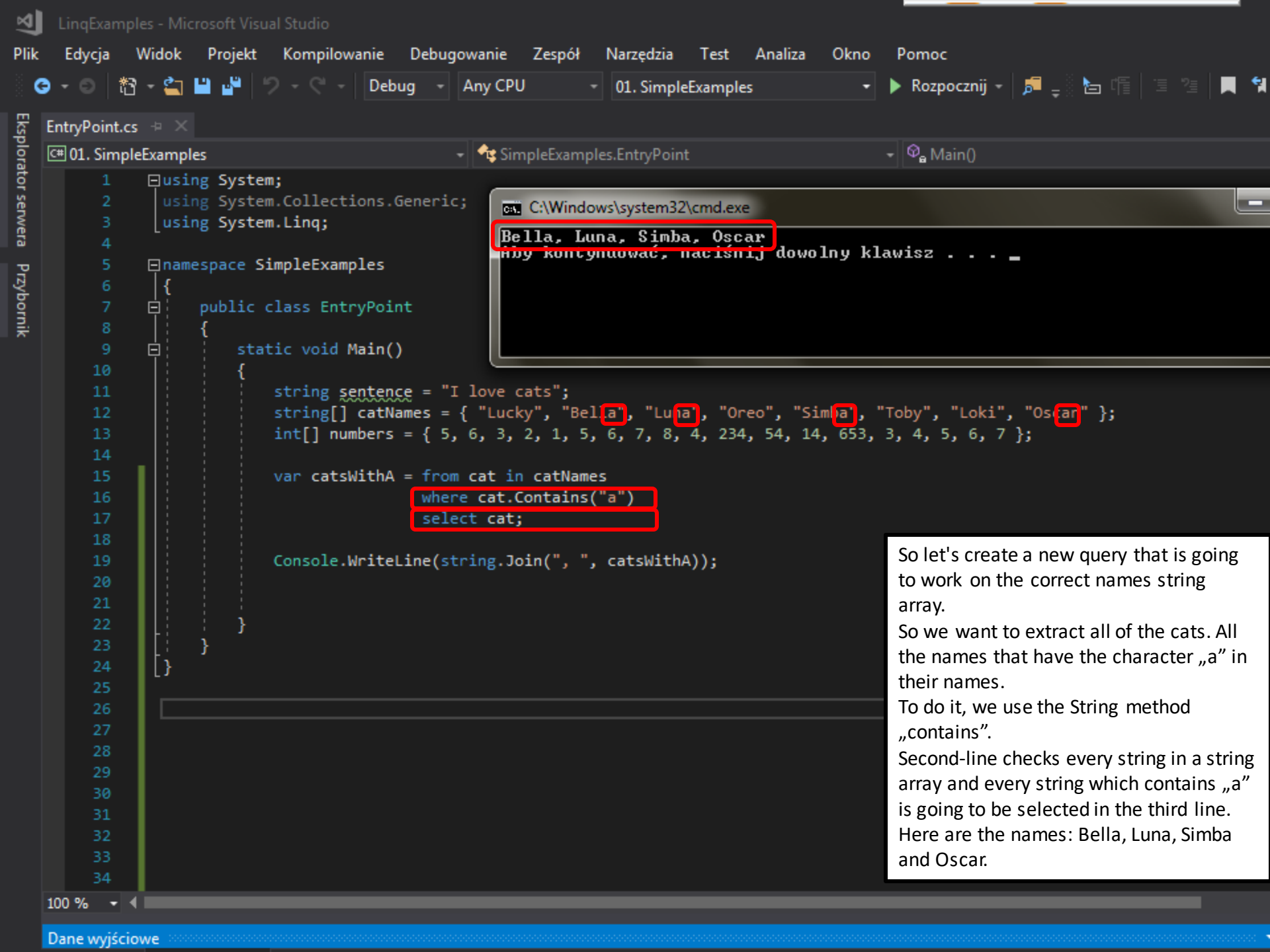
If You run that project now, we should be able to get all of the numbers less than five.
There are 3, 2, 1, 4, 3 and 4 and no other numbers that are less than 5.
And at the bottom of the example, we're going to perform the same operation, but we're going to use a simple loop.
First, we're going to create the integer list because we don't know how many numbers we are going to have.
That's why we can't use an array. An array needs to keep its size predefined.
And then we're going to create a For Each loop so for a number in numbers.
If the number is less than 5, we are going to add it to the new numbers list.

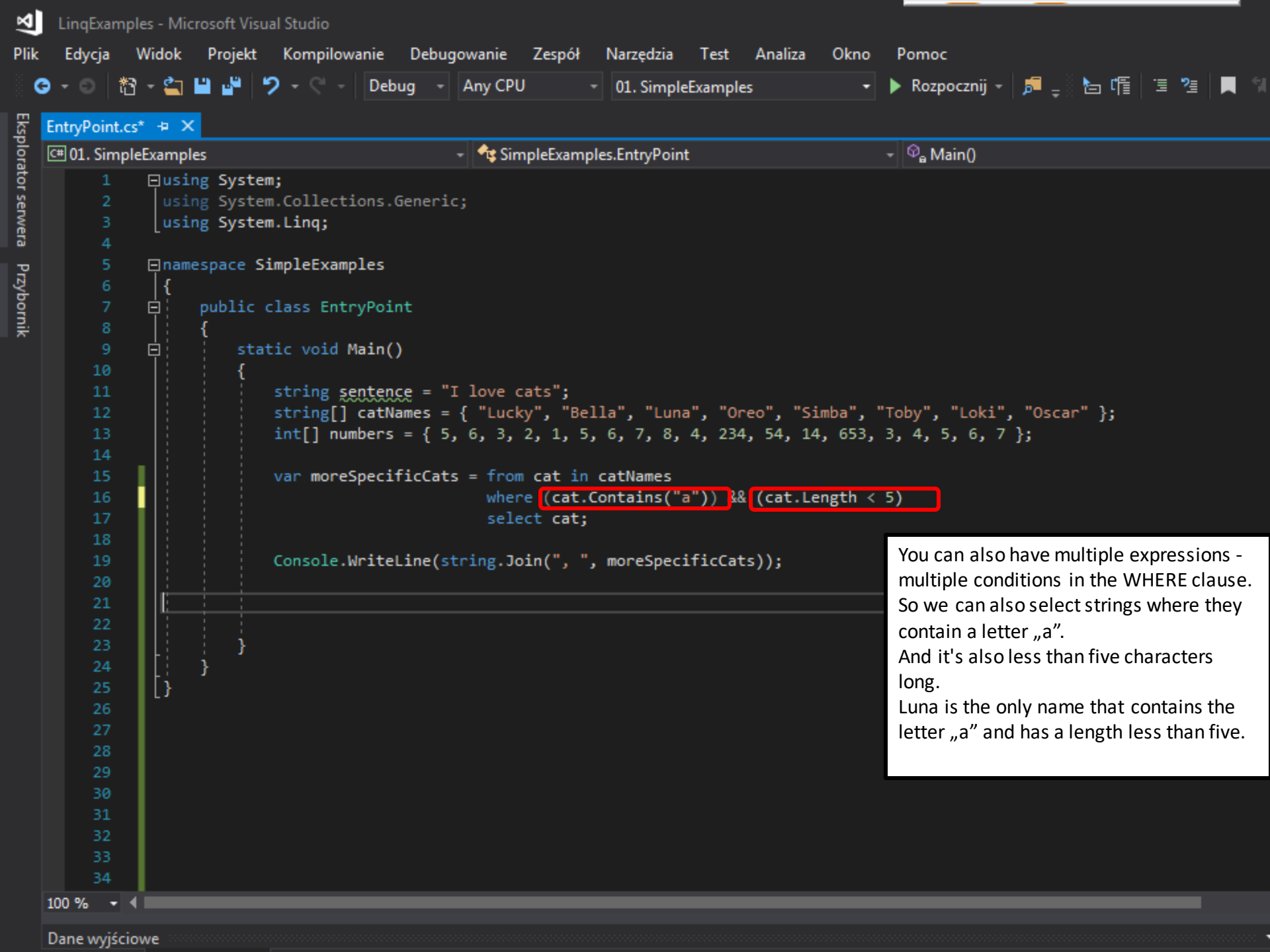


Now there are two ways to do the same thing, and we should have the same result.
We can compare the two methods.
Query operates just in three lines of code and
in the second method, we had to create a new list we had to use a for loop, and we had to use an IF condition and lot of brackets.
A second difference is that in the first method, the query is not executed until use this new collection somewhere.
That is why executing the query is the third step.
First, we simply create the query, and it is being executed down here when we try to access it.
The for loop is going to be executed immediately where it is implemented, and we already have all of the new numbers when we access them.

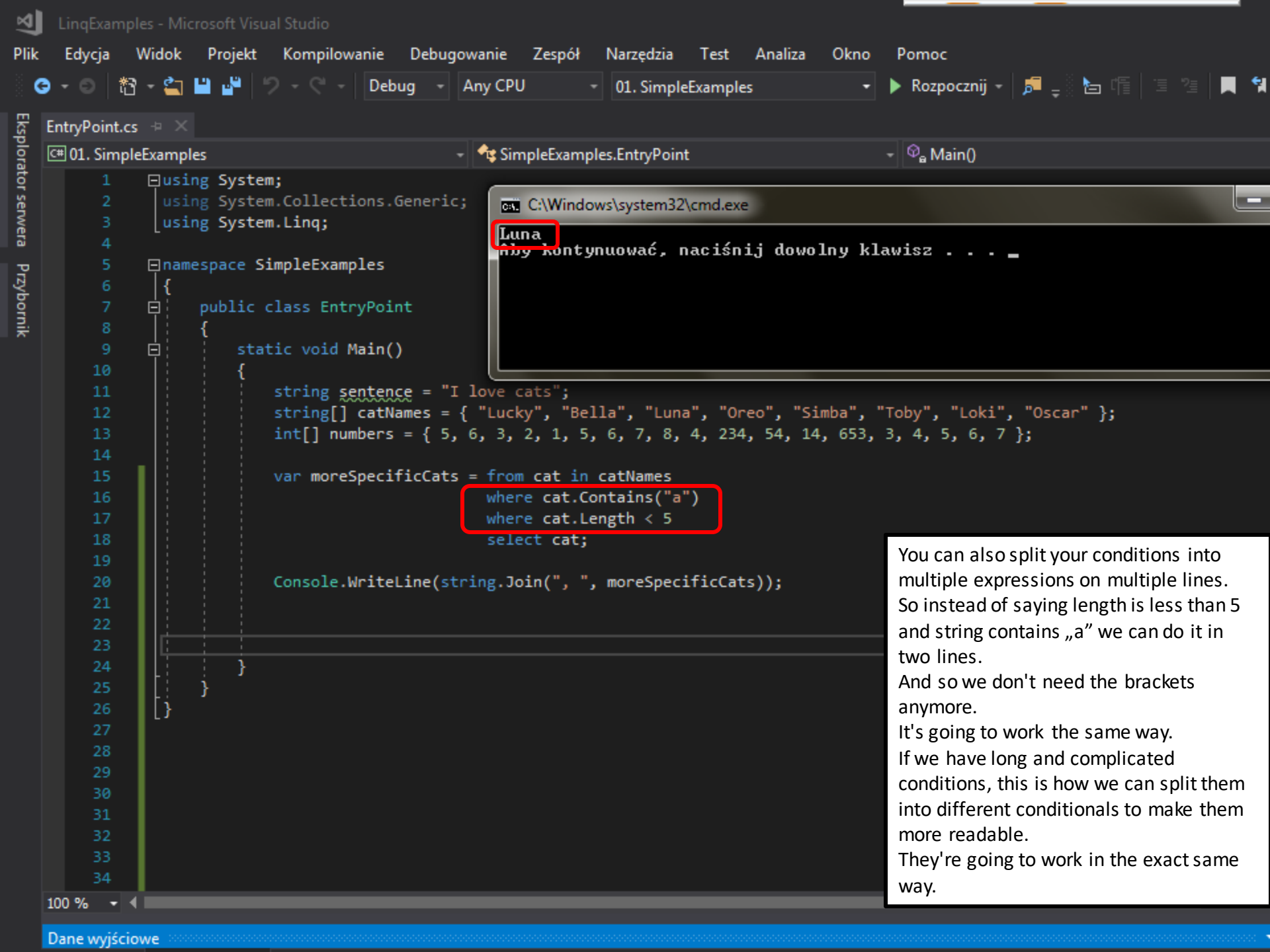

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var greaterThanFiveAndLessThanTen = from num in numbers
where (num > 5) && (num < 10)
select num;
Console.WriteLine(string.Join(", ", greaterThanFiveAndLessThanTen));
List<int> greaterThanFiveAndLessThanTenList = new List<int>();
foreach (var number in numbers)
{
if ((number > 5) && (number< 10))
{
greaterThanFiveAndLessThanTenList.Add(number);
}
}
Console.WriteLine(string.Join(", ", greaterThanFiveAndLessThanTenList));
And there is a more complex example.
We can do the same example where a number is more than 5, and the number is less than 10.
And what we get now?
100 %
Dane wyjściowe

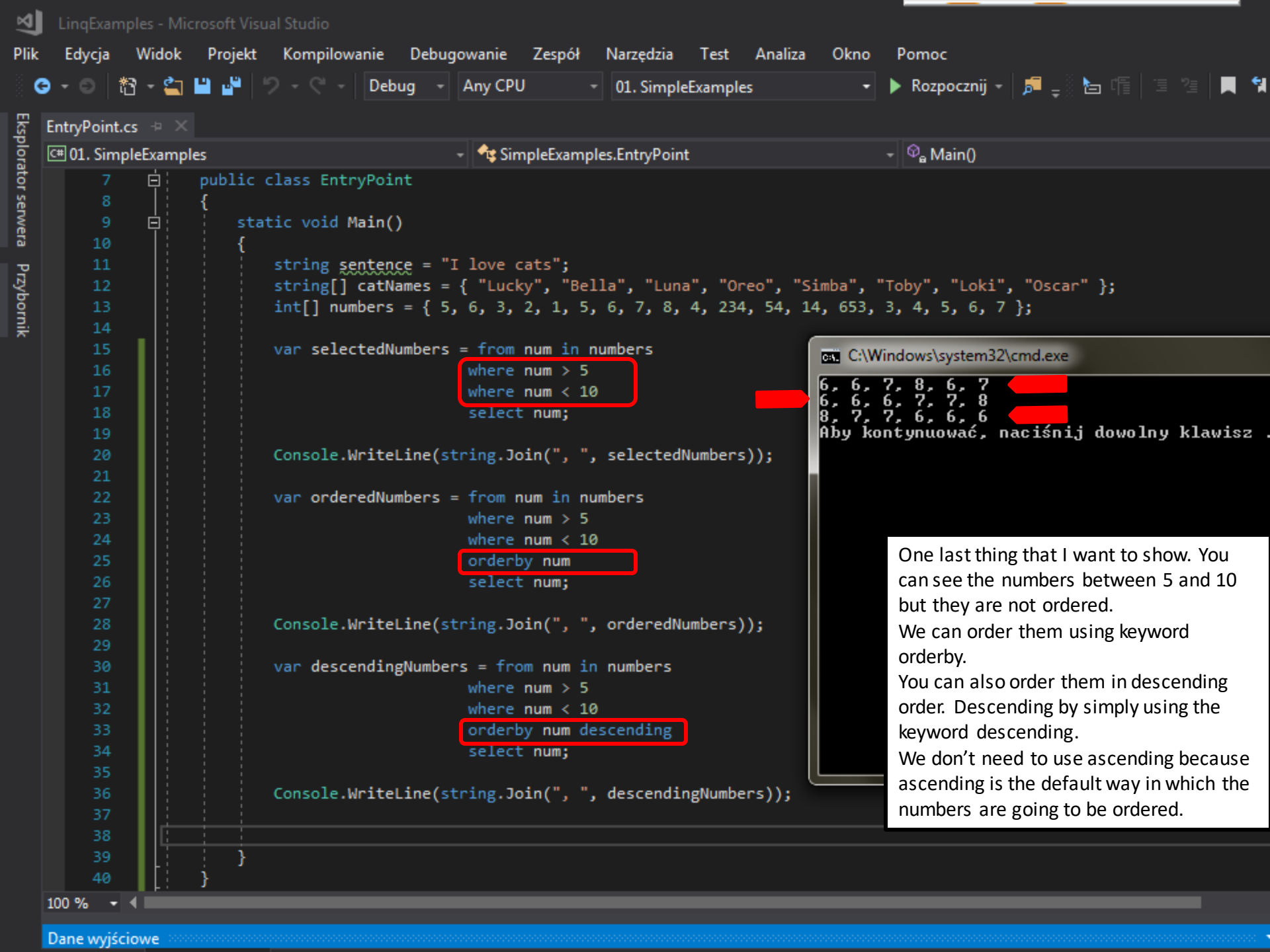

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var greaterThanFiveAndLessThanTen = from num in numbers
where (num > 5) && (num < 10)
select num;
Console.WriteLine(string.Join(", ", greaterThanFiveAndLessThanTen));
List<int> greaterThanFiveAndLessThanTenList = new List<int>();
foreach (var number in numbers)
{
if ((number > 5) && (number< 10))
{
greaterThanFiveAndLessThanTenList.Add(number);
}
}
Console.WriteLine(string.Join(", ", greaterThanFiveAndLessThanTenList));
C:\Windows\system32\cmd.exe
6, 6, 7, 8, 6, 7
6, 6, 7, 8, 6, 7
Naciśnij dowolny klawisz . . .
So we have 6 7 8 6 7—the same result using query and by applying for-loop construction.
OK.
Let's see a couple of other examples.
You can use the where clause with any condition that would give you a boolean value. True or false.
So you can do much more than just checking for numbers.
Let's create a new query.
100 %
Dane wyjściowe


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var catsWithA = from cat in catNames
where cat.Contains("a")
select cat;
Console.WriteLine(string.Join(", ", catsWithA));
}
}
}
C:\Windows\system32\cmd.exe
Bella, Luna, Simba, Oscar
Aby kontynuować, naciśnij dowolny klawisz . . .
So let's create a new query that is going to work on the correct names string array.
So we want to extract all of the cats. All the names that have the character „a” in their names.
To do it, we use the String method „contains”.
Second-line checks every string in a string array and every string which contains „a” is going to be selected in the third line.
Here are the names: Bella, Luna, Simba and Oscar.
100 %
Dane wyjściowe



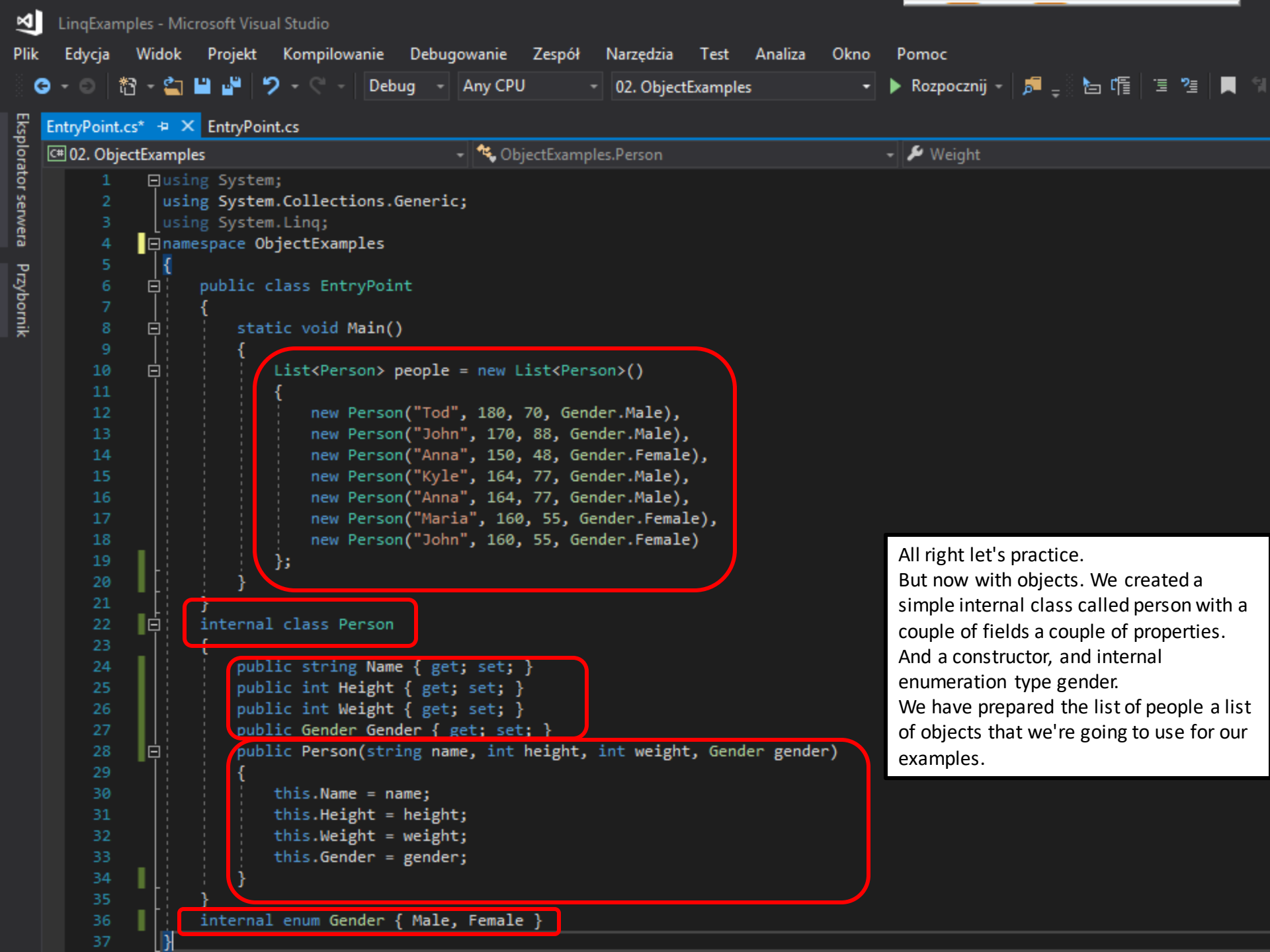
You can also have multiple expressions - multiple conditions in the WHERE clause. So we can also select strings where they contain a letter „a".
And it's also less than five characters long.
Luna is the only name that contains the letter „a" and has a length less than five.

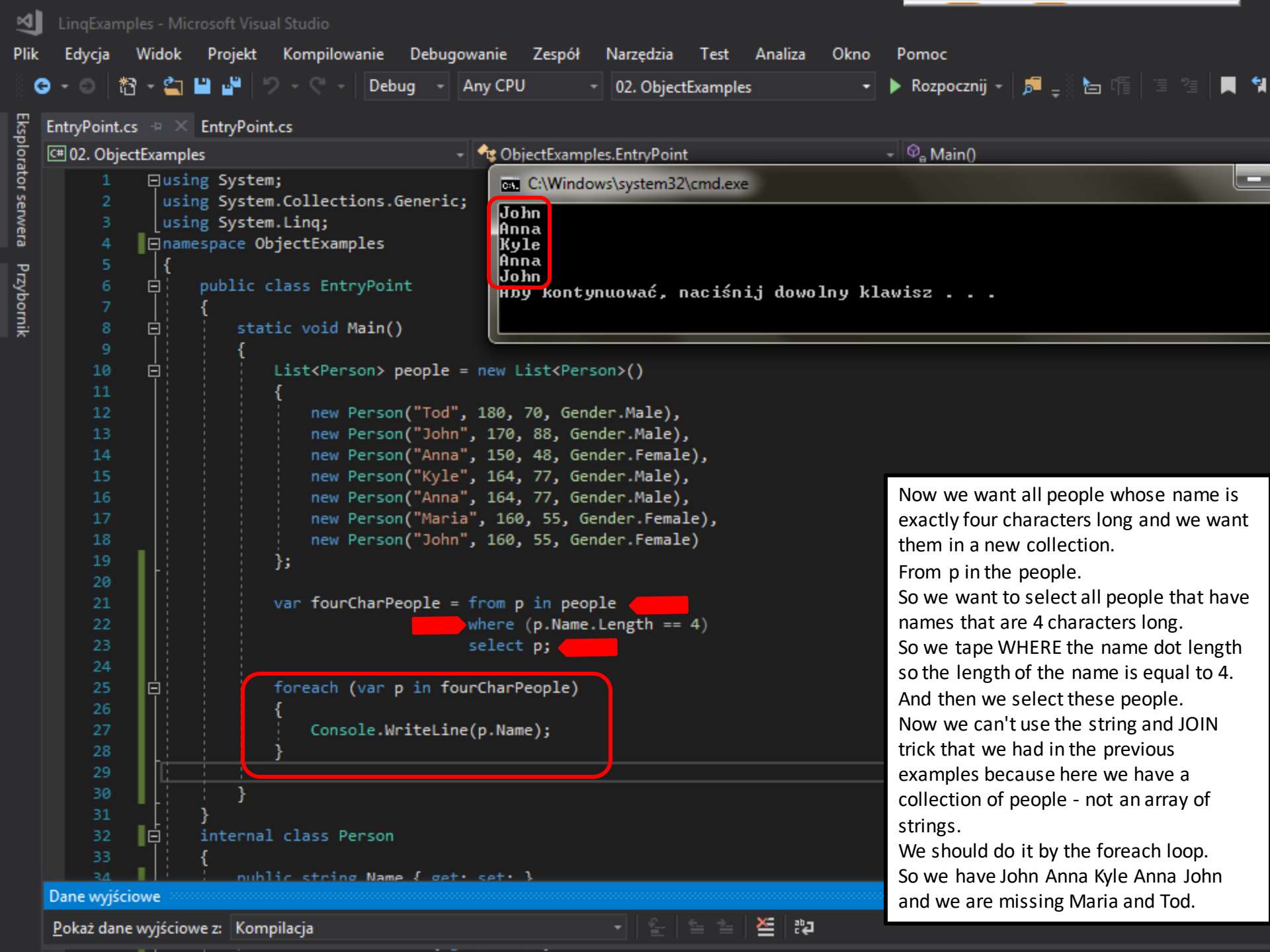

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace SimpleExamples
{
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var moreSpecificCats = from cat in catNames
where cat.Contains("a")
where cat.Length < 5
select cat;
Console.WriteLine(string.Join(", ", moreSpecificCats));
}
}
}
C:\Windows\system32\cmd.exe
Luna
Aby kontynuować, naciśnij dowolny klawisz . . .
You can also split your conditions into multiple expressions on multiple lines.
So instead of saying length is less than 5 and string contains „a” we can do it in two lines.
And so we don't need the brackets anymore.
It's going to work the same way.
If we have long and complicated conditions, this is how we can split them into different conditionals to make them more readable.
They're going to work in the exact same way.
100 %
Dane wyjściowe


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
public class EntryPoint
{
static void Main()
{
string sentence = "I love cats";
string[] catNames = { "Lucky", "Bella", "Luna", "Oreo", "Simba", "Toby", "Loki", "Oscar" };
int[] numbers = { 5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7 };
var selectedNumbers = from num in numbers
where num > 5
where num < 10
select num;
Console.WriteLine(string.Join(", ", selectedNumbers));
var orderedNumbers = from num in numbers
where num > 5
where num < 10
orderby num
select num;
Console.WriteLine(string.Join(", ", orderedNumbers));
var descendingNumbers = from num in numbers
where num > 5
where num < 10
orderby num descending
select num;
Console.WriteLine(string.Join(", ", descendingNumbers));
}
}
C:\Windows\system32\cmd.exe
6, 6, 7, 8, 6, 7
6, 6, 6, 7, 7, 8
8, 7, 7, 6, 6, 6
Aby kontynuować, naciśnij dowolny klawisz .
One last thing that I want to show. You can see the numbers between 5 and 10 but they are not ordered.
We can order them using keyword orderby.
You can also order them in descending order. Descending by simply using the keyword descending.
We don't need to use ascending because ascending is the default way in which the numbers are going to be ordered.
100 %
Dane wyjściowe

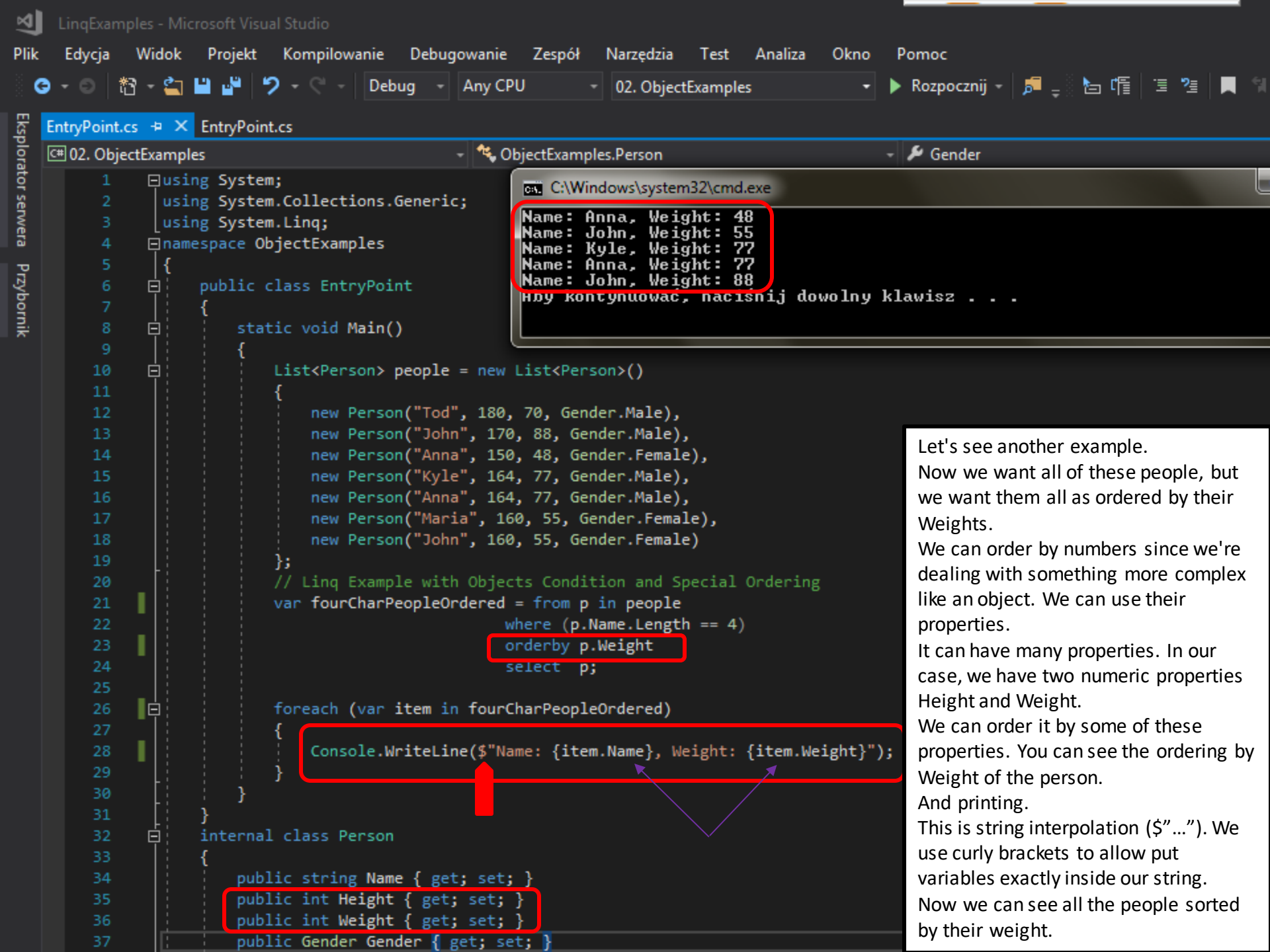
LINQ Queries on Objects

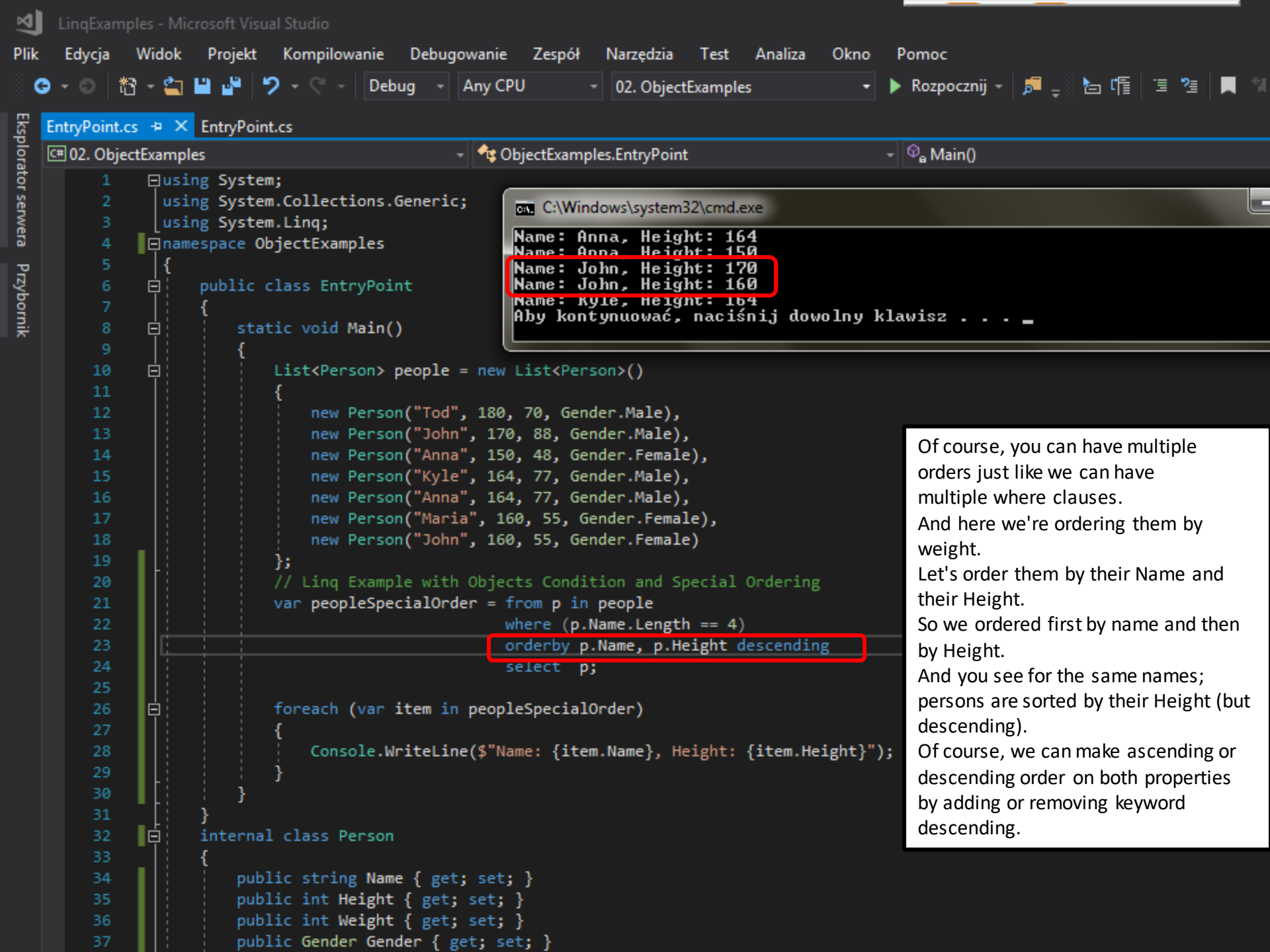
Basically what you do with LinQ things will be more interesting in the next lecture. LinQ Queries on Objects

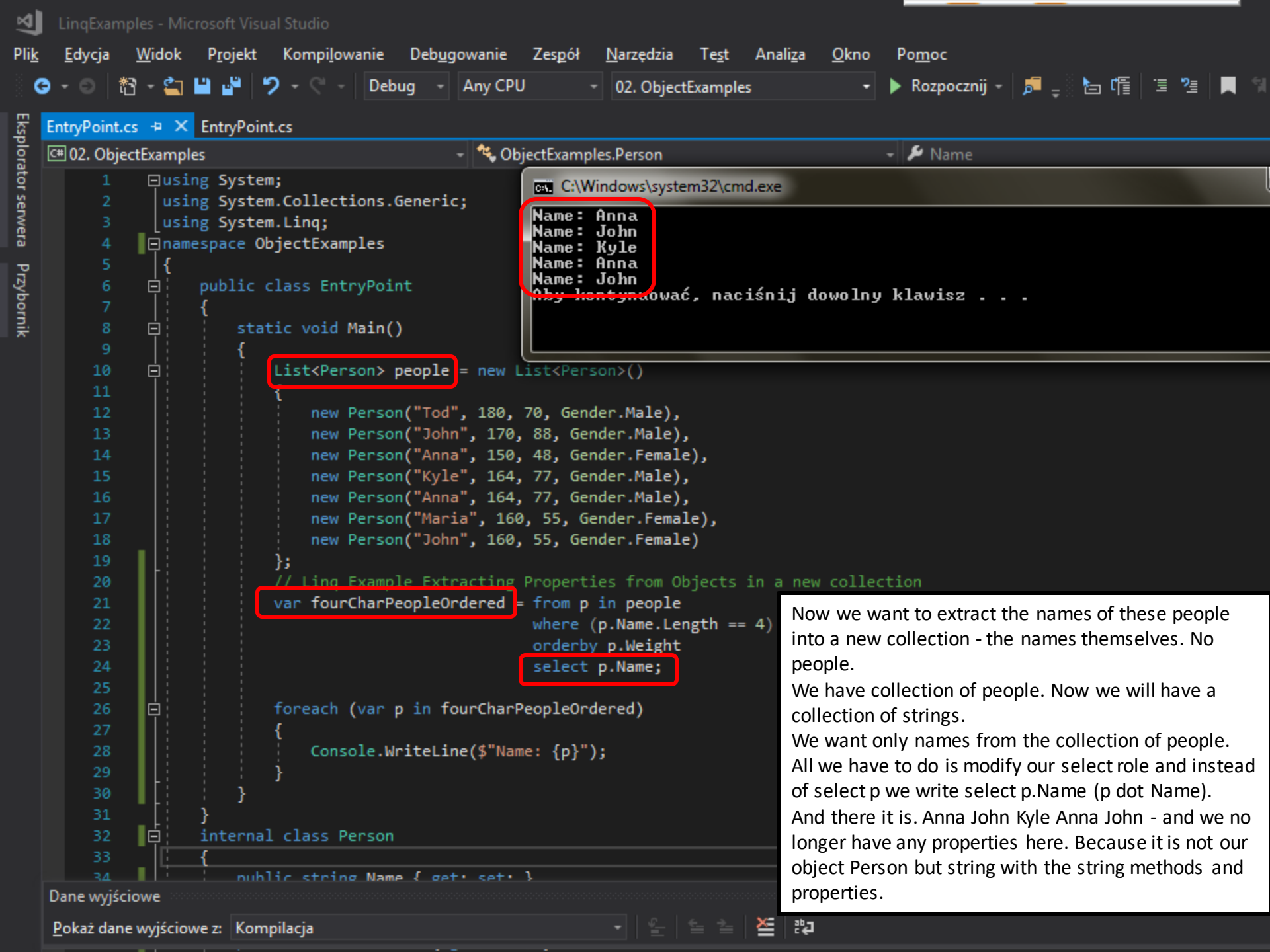

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. ObjectExamples
Rozpocznij
EntryPoint.cs*
EntryPoint.cs
02. ObjectExamples
ObjectExamples.Person
Weight
Eksplorator serwera
Przybornik
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 namespace ObjectExamples
5 {
6 public class EntryPoint
7 {
8 static void Main()
9 {
10 List<Person> people = new List<Person>()
11 {
12 new Person("Tod", 180, 70, Gender.Male),
13 new Person("John", 170, 88, Gender.Male),
14 new Person("Anna", 150, 48, Gender.Female),
15 new Person("Kyle", 164, 77, Gender.Male),
16 new Person("Anna", 164, 77, Gender.Male),
17 new Person("Maria", 160, 55, Gender.Female),
18 new Person("John", 160, 55, Gender.Female)
19 };
20 }
21 }
22 internal class Person
23 {
24 public string Name { get; set; }
25 public int Height { get; set; }
26 public int Weight { get; set; }
27 public Gender Gender { get; set; }
28 public Person(string name, int height, int weight, Gender gender)
29 {
30 this.Name = name;
31 this.Height = height;
32 this.Weight = weight;
33 this.Gender = gender;
34 }
35 }
36 internal enum Gender { Male, Female }
37 }
All right let's practice.
But now with objects. We created a simple internal class called person with a couple of fields a couple of properties.
And a constructor, and internal enumeration type gender.
We have prepared the list of people a list of objects that we're going to use for our examples.



Now we want all people whose name is exactly four characters long and we want them in a new collection.
From p in the people.
So we want to select all people that have names that are 4 characters long.
So we tape WHERE the name dot length so the length of the name is equal to 4.
And then we select these people.
Now we can't use the string and JOIN trick that we had in the previous examples because here we have a collection of people - not an array of strings.
We should do it by the foreach loop.
So we have John Anna Kyle Anna John and we are missing Maria and Tod.


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. ObjectExamples
Rozpocznij
EntryPoint.cs
EntryPoint.cs
02. ObjectExamples
ObjectExamples.Person
Gender
Eksplorator serwera
Przybornik
C:\Windows\system32\cmd.exe
Name: Anna, Weight: 48
Name: John, Weight: 55
Name: Kyle, Weight: 77
Name: Anna, Weight: 77
Name: John, Weight: 88
Aby kontynuować, naciśnij dowolny klawisz . . .
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 namespace ObjectExamples
5 {
6 public class EntryPoint
7 {
8 static void Main()
9 {
10 List<Person> people = new List<Person>()
11 {
12 new Person("Tod", 180, 70, Gender.Male),
13 new Person("John", 170, 88, Gender.Male),
14 new Person("Anna", 150, 48, Gender.Female),
15 new Person("Kyle", 164, 77, Gender.Male),
16 new Person("Anna", 164, 77, Gender.Male),
17 new Person("Maria", 160, 55, Gender.Female),
18 new Person("John", 160, 55, Gender.Female)
19 };
20 // Linq Example with Objects Condition and Special Ordering
21 var fourCharPeopleOrdered = from p in people
22 where (p.Name.Length == 4)
23 orderby p.Weight
24 select p;
25
26 foreach (var item in fourCharPeopleOrdered)
27 {
28 Console.WriteLine($"Name: {item.Name}, Weight: {item.Weight}");
29 }
30 }
31 }
32 internal class Person
33 {
34 public string Name { get; set; }
35 public int Height { get; set; }
36 public int Weight { get; set; }
37 public Gender Gender { get; set; }
Let's see another example.
Now we want all of these people, but we want them all as ordered by their Weights.
We can order by numbers since we're dealing with something more complex like an object. We can use their properties.
It can have many properties. In our case, we have two numeric properties Height and Weight.
We can order it by some of these properties. You can see the ordering by Weight of the person.
And printing.
This is string interpolation ($"..."). We use curly brackets to allow put variables exactly inside our string.
Now we can see all the people sorted by their weight.


LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. ObjectExamples
Rozpocznij
EntryPoint.cs
EntryPoint.cs
02. ObjectExamples
ObjectExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
namespace ObjectExamples
{
public class EntryPoint
{
static void Main()
{
List<Person> people = new List<Person>()
{
new Person("Tod", 180, 70, Gender.Male),
new Person("John", 170, 88, Gender.Male),
new Person("Anna", 150, 48, Gender.Female),
new Person("Kyle", 164, 77, Gender.Male),
new Person("Anna", 164, 77, Gender.Male),
new Person("Maria", 160, 55, Gender.Female),
new Person("John", 160, 55, Gender.Female)
};
// Linq Example with Objects Condition and Special Ordering
var peopleSpecialOrder = from p in people
where (p.Name.Length == 4)
orderby p.Name, p.Height descending
select p;
foreach (var item in peopleSpecialOrder)
{
Console.WriteLine($"Name: {item.Name}, Height: {item.Height}");
}
}
}
internal class Person
{
public string Name { get; set; }
public int Height { get; set; }
public int Weight { get; set; }
public Gender Gender { get; set; }
C:\Windows\system32\cmd.exe
Name: Anna, Height: 164
Name: Anna, Height: 150
Name: John, Height: 170
Name: John, Height: 160
Name: Kyle, Height: 164
Aby kontynuować, naciśnij dowolny klawisz . . .
Of course, you can have multiple orders just like we can have multiple where clauses.
And here we're ordering them by weight.
Let's order them by their Name and their Height.
So we ordered first by name and then by Height.
And you see for the same names; persons are sorted by their Height (but descending).
Of course, we can make ascending or descending order on both properties by adding or removing keyword descending.



Now we want to extract the names of these people into a new collection - the names themselves. No people.
We have collection of people. Now we will have a collection of strings.
We want only names from the collection of people.
All we have to do is modify our select role and instead of select p we write select p.Name (p dot Name).
And there it is. Anna John Kyle Anna John - and we no longer have any properties here. Because it is not our object Person but string with the string methods and properties.

The End

Now it is end of the presentation but not end of LinQ. We will continue with LinQ but in a different way – using the Lambda expression.