LINQ - Language Integrated Query Introduction

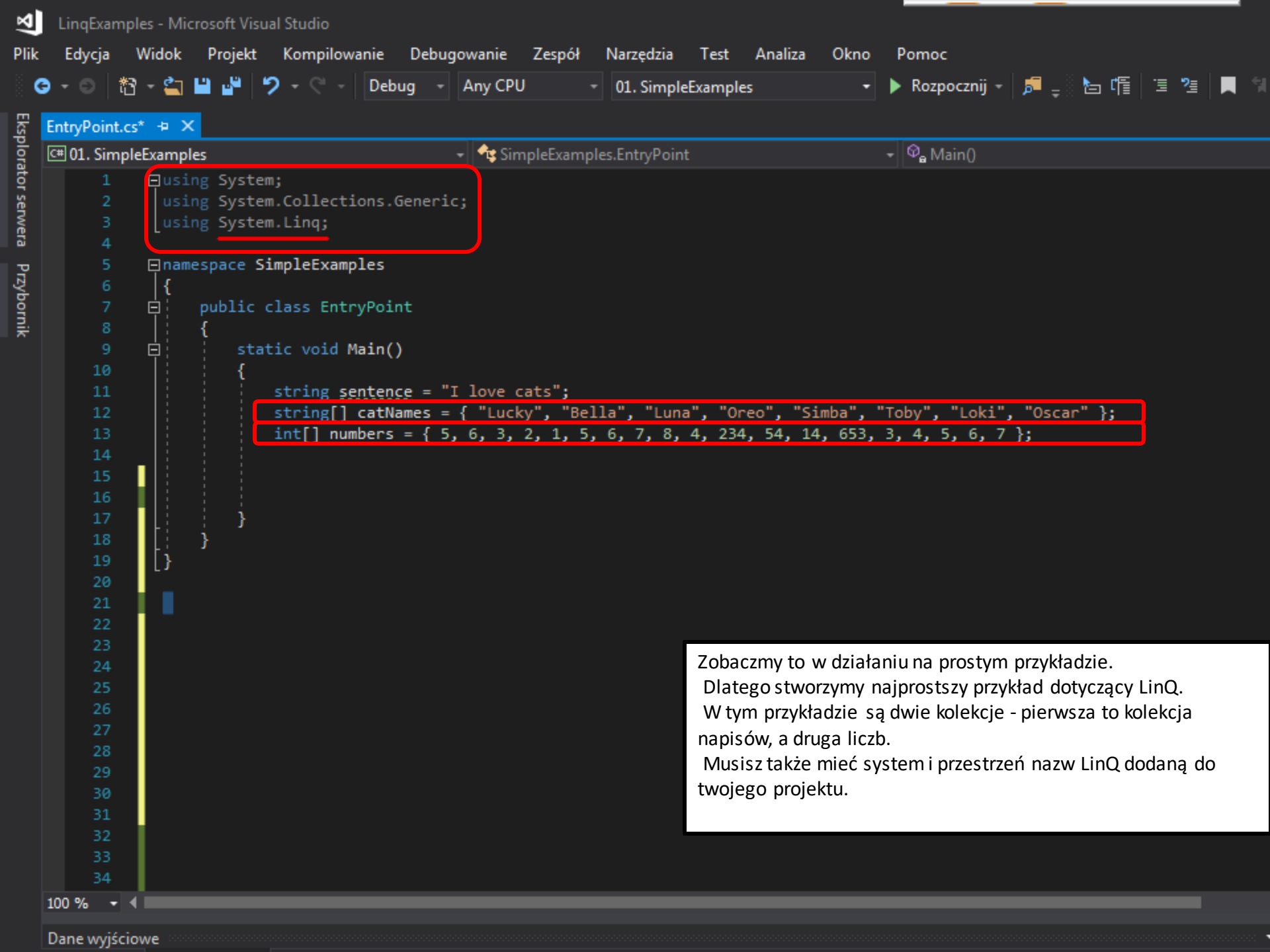
- Similar to SQL - works on Collections
- Three basic operations:
  - Get the source
  - Create the Query
  - Execute the Query

W porządku, zaczniemy LinQ.
LinQ oznacza Language Integrated Query czyli zapytanie zintegrowane z językiem.
Jest on bardzo podobny do SQL z tą różnicą, że LinQ działa w kolekcjach, a SQL w bazach danych.
Obydwa wykonują to samo.
Oba mają na celu wyodrębnienie niektórych informacji z kolekcji lub bazy danych na podstawie określonych warunków.
Wszystkie operacje LinQ można wyrazić w trzech krokach.
Najpierw pobieramy źródło, następnie tworzymy zapytanie, a następnie wykonujemy to zapytanie.

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

Zacznijmy analizować strukturę zapytań LinQ.
Pierwszą rzeczą, którą musimy zrobić, jest zdefiniowanie naszego źródła.
Zaczynamy od słowa kluczowego FROM.
Tutaj definiujemy zmienną, która będzie iterować po źródle.
FROM - zmienna iteracyjna - IN - źródło.
Następnie definiujemy niektóre warunki za pomocą słowa kluczowego WHERE
i WYBIERAMY wszystkie nowe elementy, które pasują do naszego warunku.


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 };
}
}
}
Zobaczmy to w działaniu na prostym przykładzie.
Dlatego stworzymy najprostszy przykład dotyczący LinQ.
W tym przykładzie są dwie kolekcje - pierwsza to kolekcja napisów, a druga liczb.
Musisz także mieć system i przestrzeń nazw LinQ dodaną do twojego projektu.
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
17
18
19          }
20      }
```

Użyjemy typu danych var, ponieważ zawsze będziemy używać innego typu kolekcji.
Nasz ostateczny typ może różnić się od tego z kolekcji.
Dlatego zawsze musisz wiedzieć, jakiego typu potrzebujesz dla swojego zapytania.
Więc po prostu łatwiej jest używać var.
W naszym zadaniu zamierzamy otrzymać dane kolekcji numbers.
Zaczynamy od słowa kluczowego FROM.
Więc po słowie kluczowym FROM pierwszą rzeczą, którą zamierzamy zrobić, jest użycie zmiennej iteracyjnej. Tak więc nazwą naszej zmiennej iteracyjnej będzie „num”.
Nazwa może być jakakolwiek - możesz po prostu użyć litery.

```
34
```

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

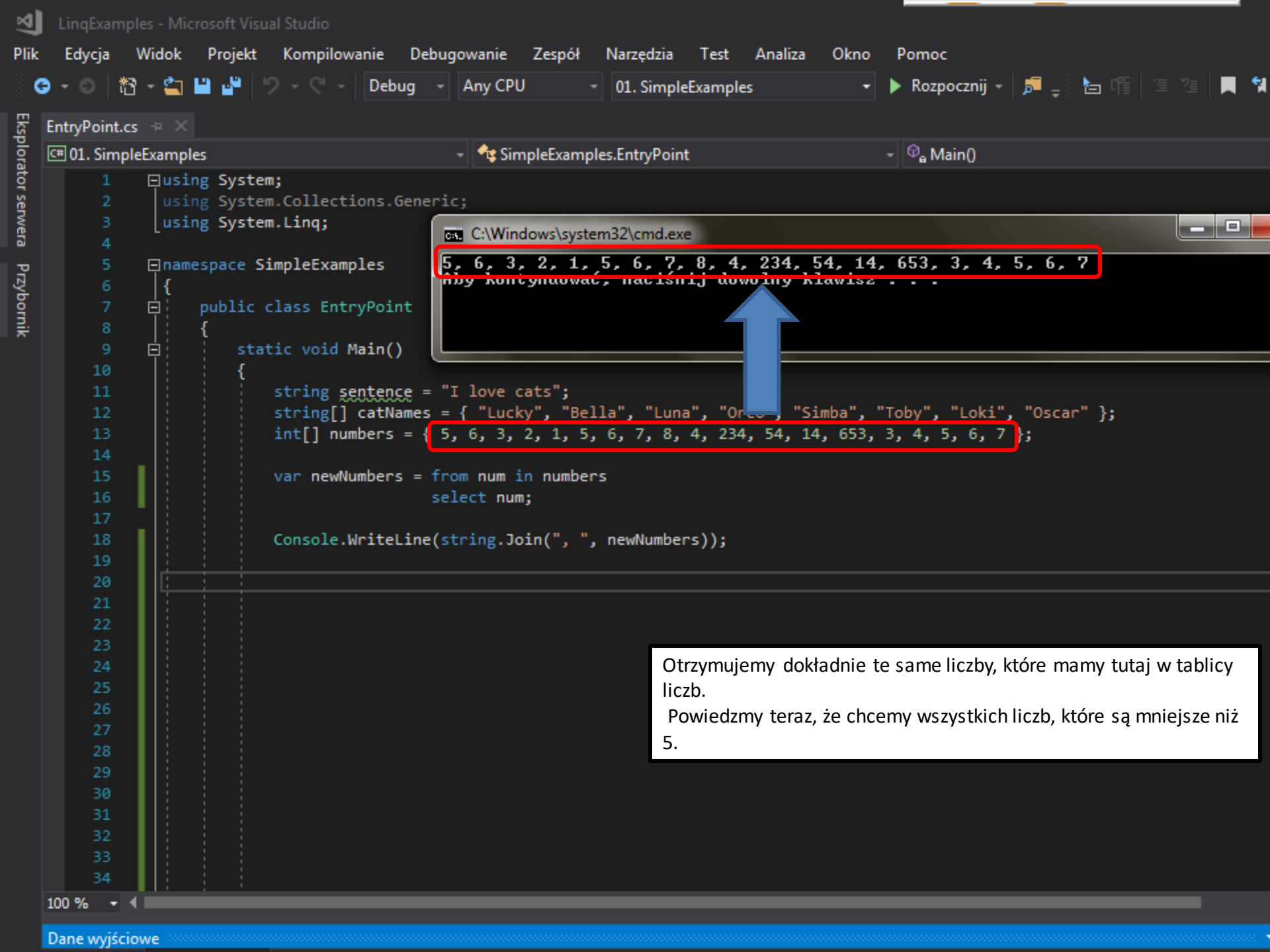
01. SimpleExamples | SimpleExamples.EntryPoint | Main()

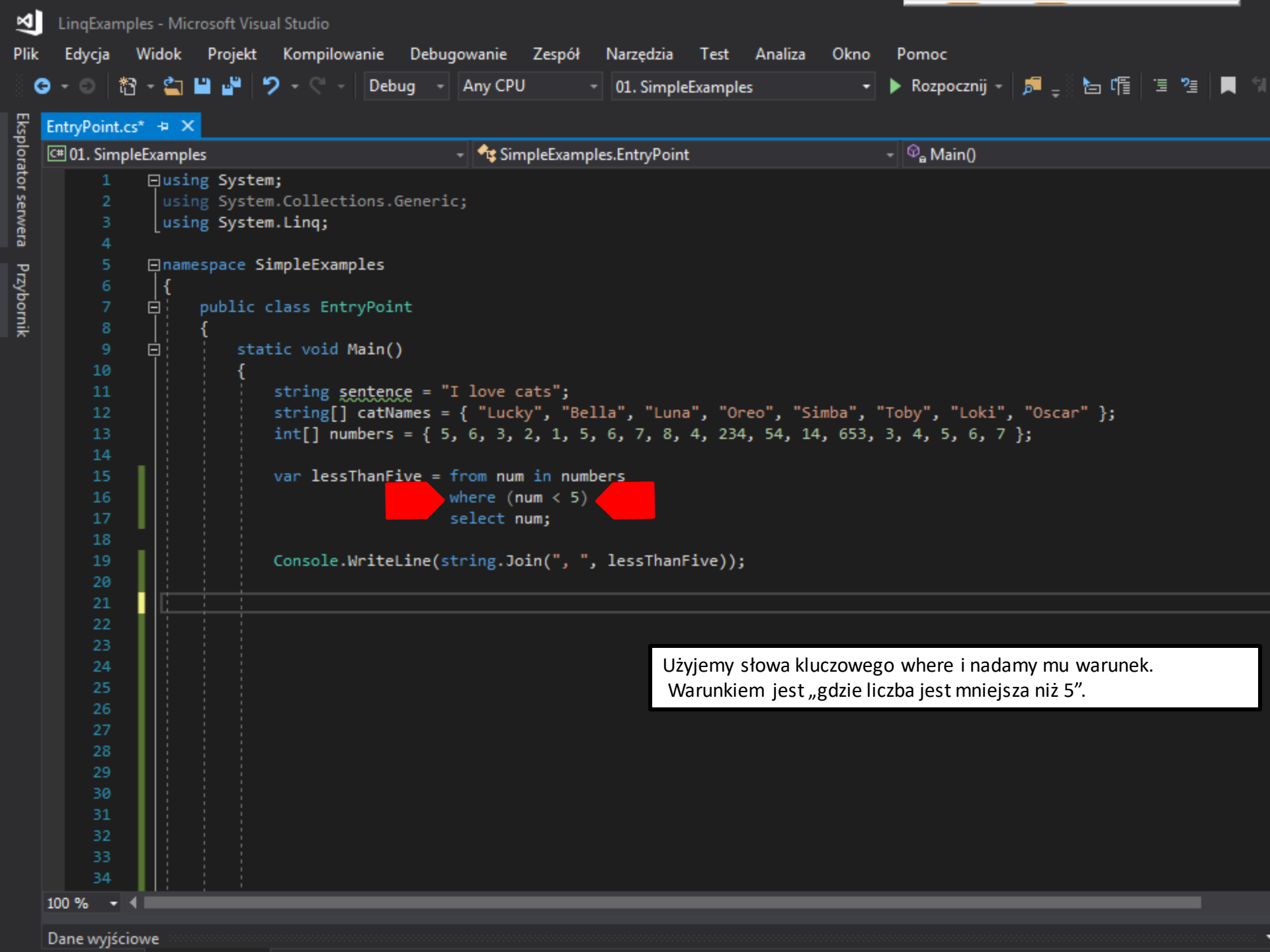
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace SimpleExamples
6  {
7      public class EntryPoint
8      {
9          static void Main()
10         {
11             string sentence = "I love cats";
12             string[] catNames = { "Lucky", "Bella", "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
```

A potem wybieramy “num”. Obecnie nie mamy żadnych warunków. Po prostu wybieramy wszystkie liczby z tablicy liczb. Wszystkie te nowe liczby wydrukujemy na konsoli, aby zobaczyć nasze wyniki. Istnieje szybki i łatwy sposób drukowania tablic w jednym wierszu kodu. Możemy użyć metody napisów - “Join”, aby wydrukować kolekcję jako jeden ciąg.

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", "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));
C:\Windows\system32\cmd.exe
5, 6, 3, 2, 1, 5, 6, 7, 8, 4, 234, 54, 14, 653, 3, 4, 5, 6, 7
Otrzymujemy dokładnie te same liczby, które mamy tutaj w tablicy liczb.
Powiedzmy teraz, że chcemy wszystkich liczb, które są mniejsze niż 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));
Użyjemy słowa kluczowego where i nadamy mu warunek.
Warunkiem jest „gdzie liczba jest mniejsza niż 5”.
100 %
Dane wyjściowe
Eksplorator serwera
Przybornik

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 lessThanFive = from num in numbers
16                                 where (num < 5)
17                                 select num;
18
19              Console.WriteLine(string.Join(", ", lessThanFive));
20
21              List<int> lessThanFiveList = new List<int>();
22
23              foreach (var number in numbers)
24              {
25                  if (number < 5)
26                  {
27                      lessThanFiveList.Add(number);
28                  }
29              }
30
31              Console.WriteLine(string.Join(", ", lessThanFiveList));
32
33
34
```

Jeśli teraz uruchomimy ten projekt, powinniśmy być w stanie uzyskać wszystkie liczby mniejsze niż pięć.

Są to 3, 2, 1, 4, 3 i 4 i żadne inne liczby nie są mniejsze niż 5.

Na dole tego przykładu wykonamy tę samą operację, ale użyjemy prostej pętli.

Po pierwsze, stworzymy listę liczb całkowitych, ponieważ nie wiemy, ile liczb będziemy mieć.

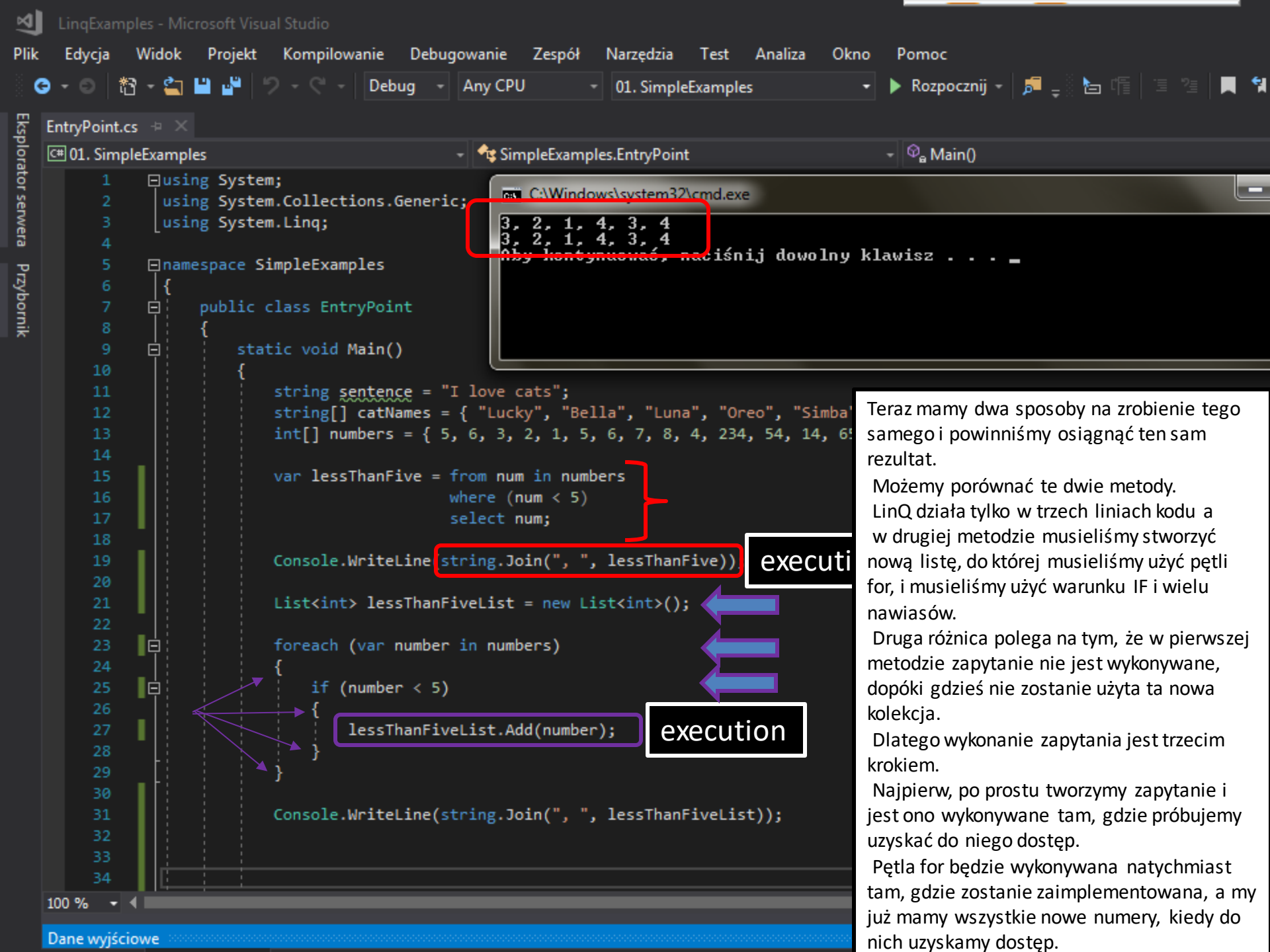
Dlatego nie możemy użyć tablicy. Tablica musi mieć zdefiniowany rozmiar.

A następnie stworzymy pętlę ForEach dla liczb w tablicy numbers.

Jeśli liczba jest mniejsza niż 5, dodamy ją do nowej listy numerów.

100 %

Dane wyjściowe



Teraz mamy dwa sposoby na zrobienie tego samego i powinniśmy osiągnąć ten sam rezultat.

Możemy porównać te dwie metody.

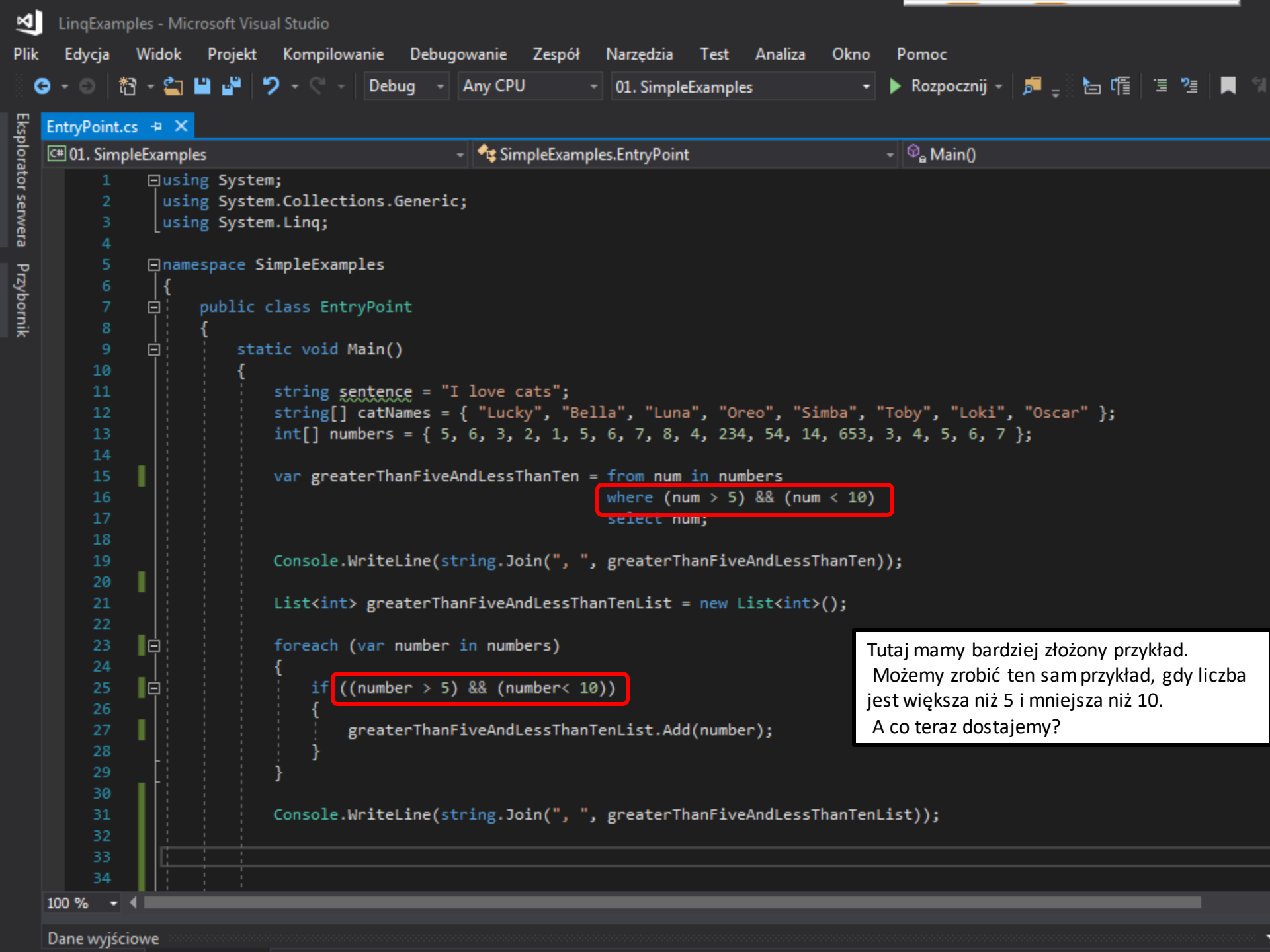
LinQ działa tylko w trzech liniach kodu a w drugiej metodzie musieliśmy stworzyć nową listę, do której musieliśmy użyć pętli for, i musieliśmy użyć warunku IF i wielu nawiasów.

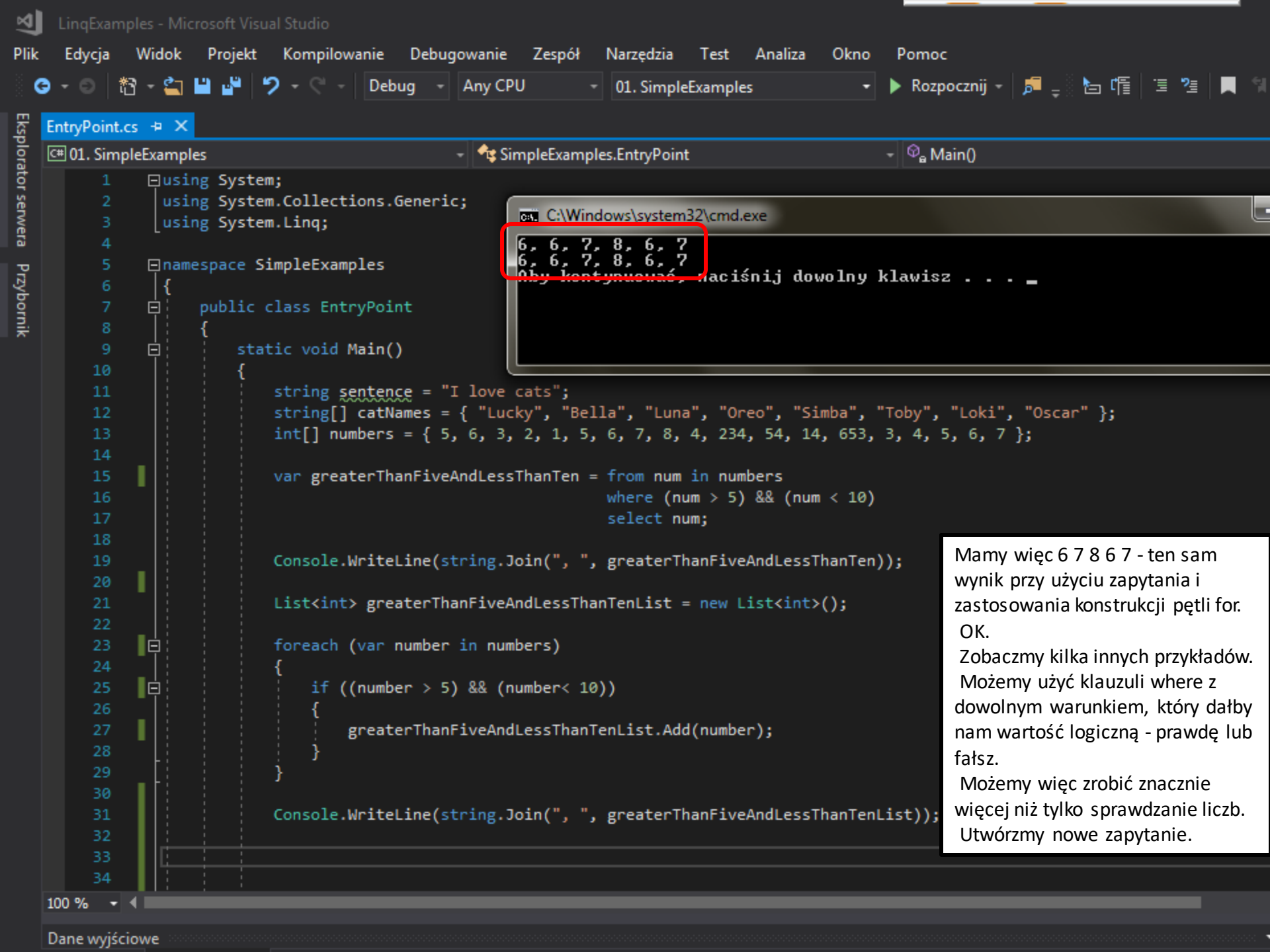
Druga różnica polega na tym, że w pierwszej metodzie zapytanie nie jest wykonywane, dopóki gdzieś nie zostanie użyta ta nowa kolekcja.

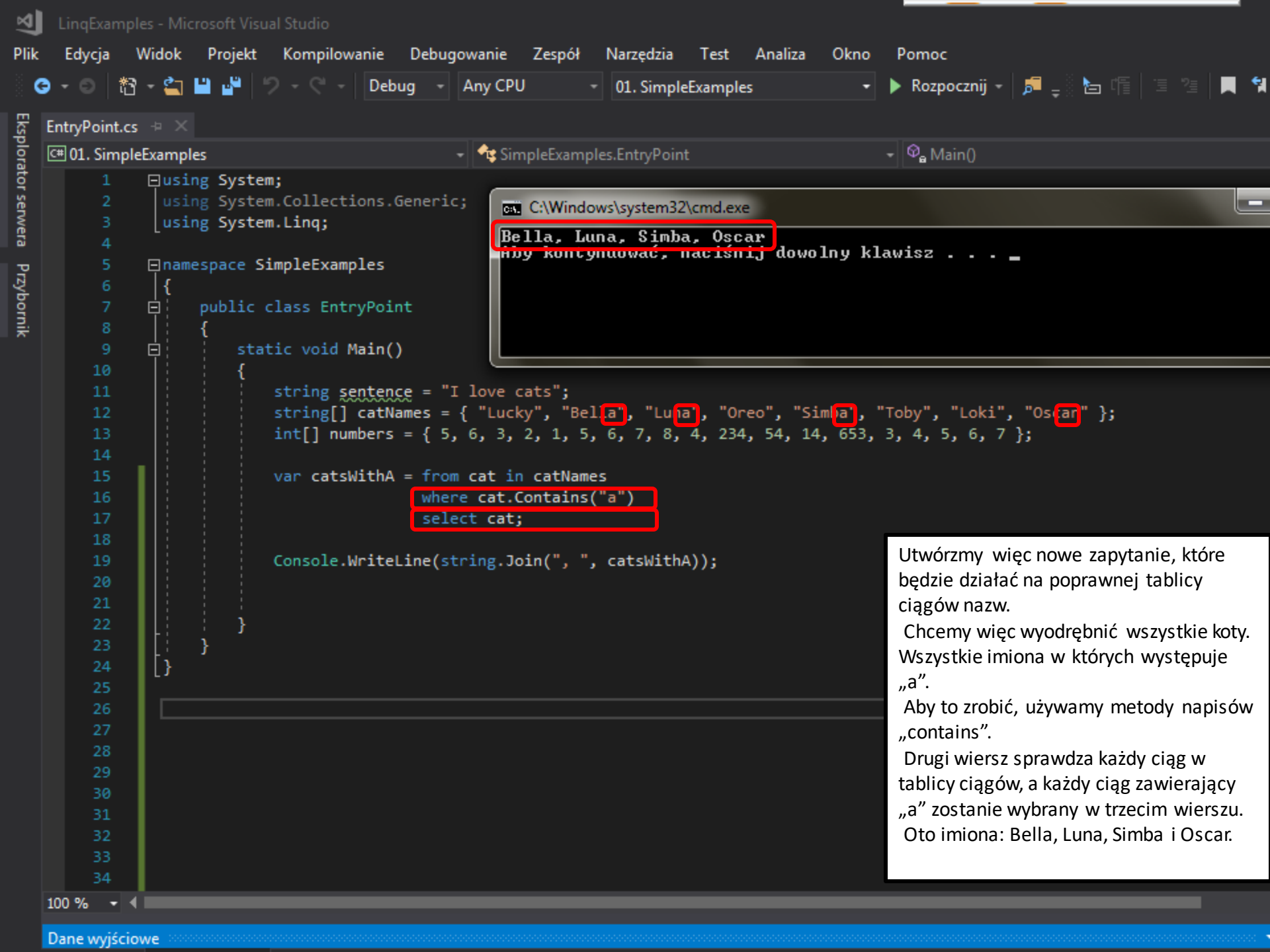
Dlatego wykonanie zapytania jest trzecim krokiem.

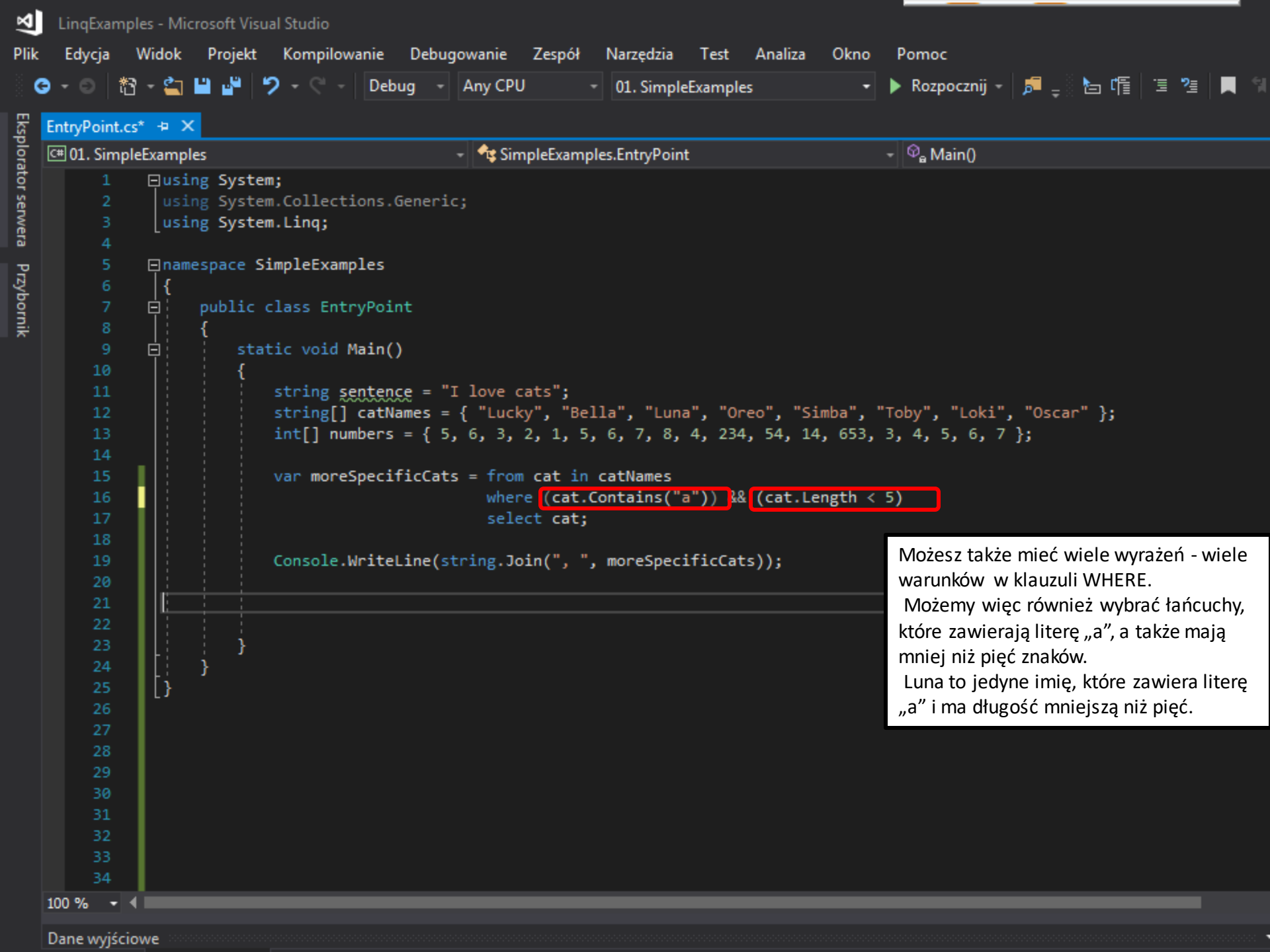
Najpierw, po prostu tworzymy zapytanie i jest ono wykonywane tam, gdzie próbujemy uzyskać do niego dostęp.

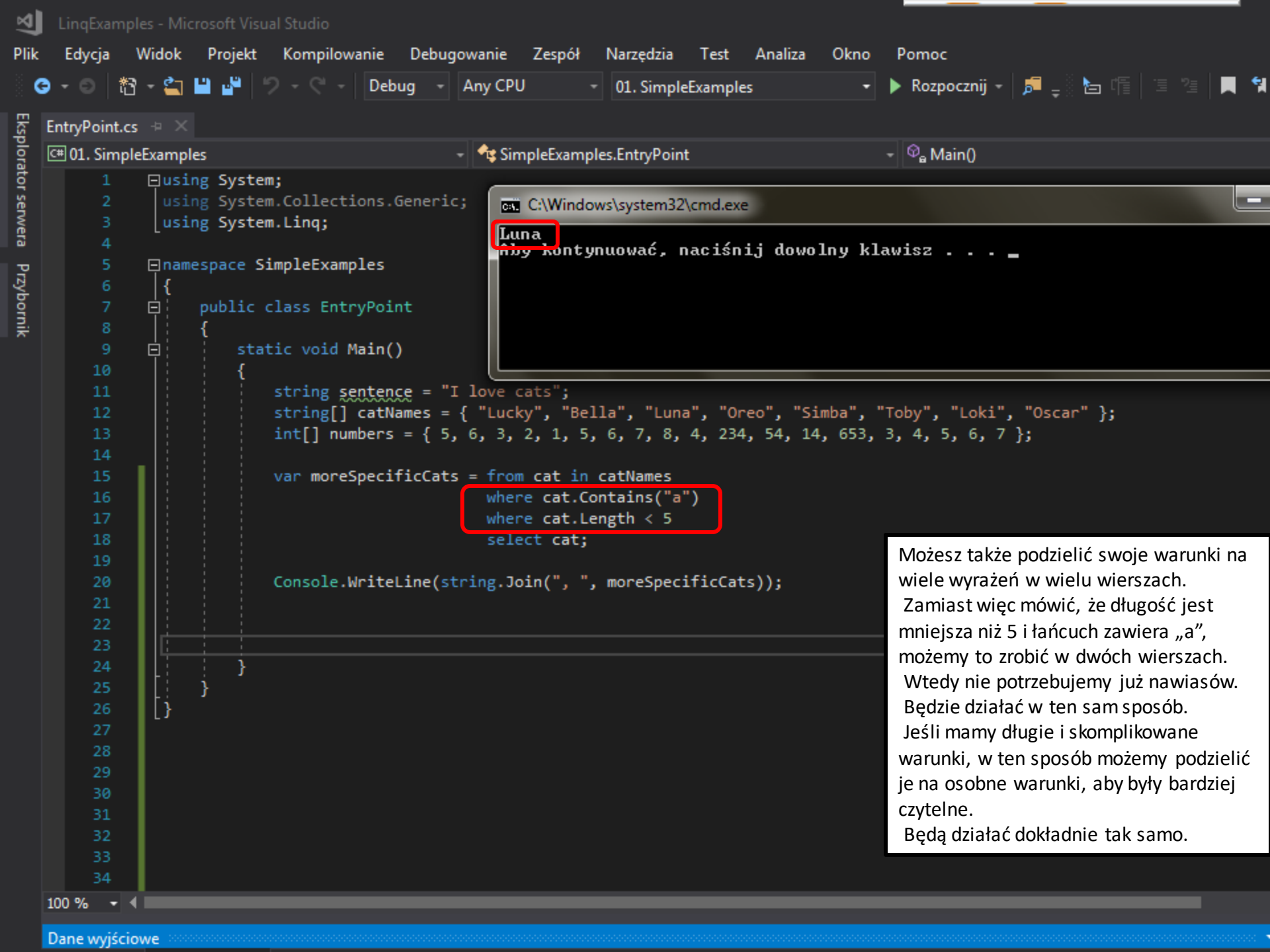
Pętla for będzie wykonywana natychmiast tam, gdzie zostanie zaimplementowana, a my już mamy wszystkie nowe numery, kiedy do nich uzyskamy dostęp.

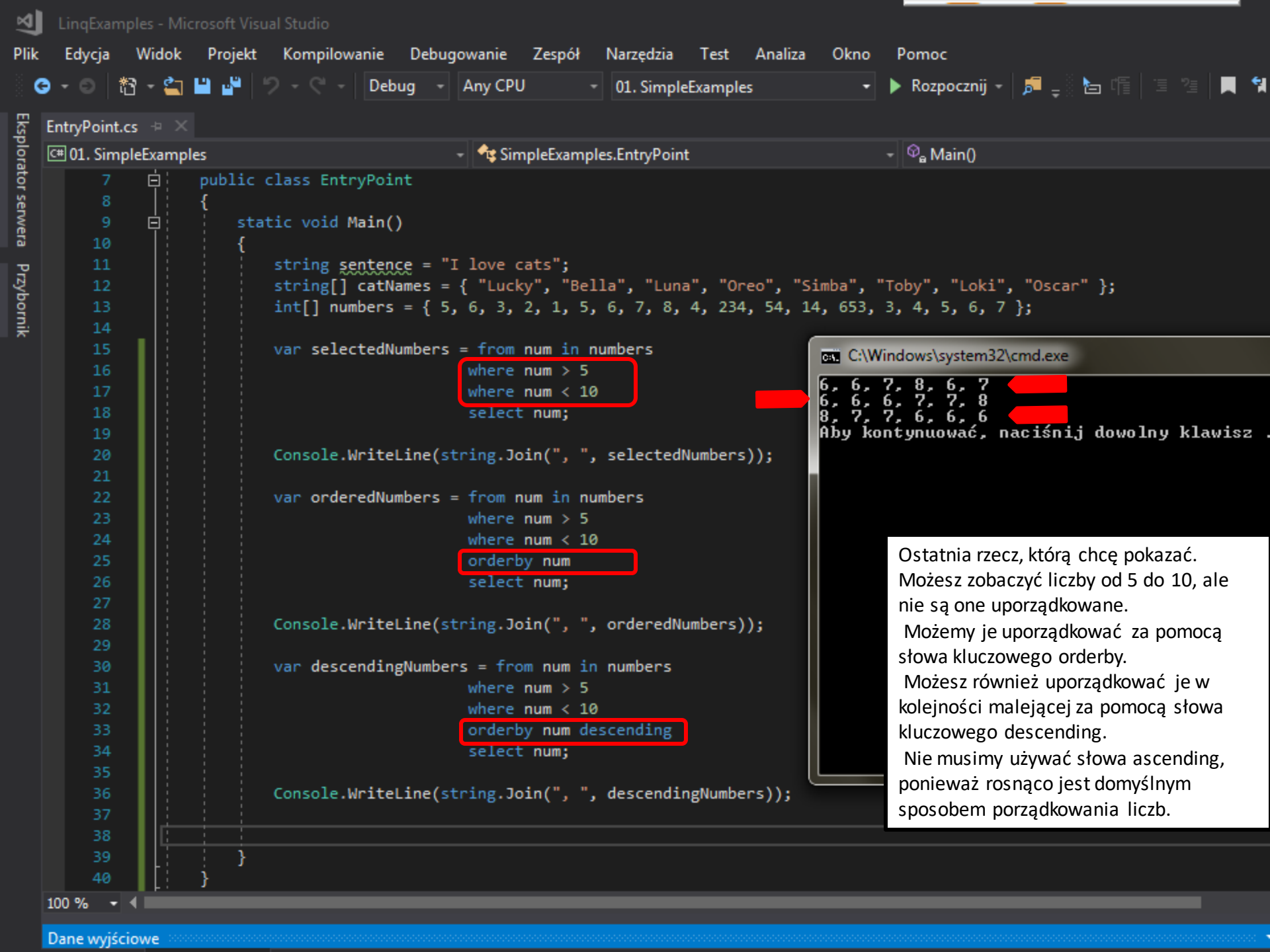

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System;
using System.Collections.Generic;
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));
Tutaj mamy bardziej złożony przykład.
Możemy zrobić ten sam przykład, gdy liczba jest większa niż 5 i mniejsza niż 10.
A co teraz dostajemy?
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
Aby kontynuować, naciśnij dowolny klawisz . . .
Mamy więc 6 7 8 6 7 - ten sam wynik przy użyciu zapytania i zastosowania konstrukcji pętli for.
OK.
Zobaczmy kilka innych przykładów.
Możemy użyć klauzuli where z dowolnym warunkiem, który dałby nam wartość logiczną - prawdę lub fałsz.
Możemy więc zrobić znacznie więcej niż tylko sprawdzanie liczb.
Utwórzmy nowe zapytanie.
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 . . .
Utwórzmy więc nowe zapytanie, które będzie działać na poprawnej tablicy ciągów nazw.
Chcemy więc wyodrębnić wszystkie koty. Wszystkie imiona w których występuje „a”.
Aby to zrobić, używamy metody napisów „contains”.
Drugi wiersz sprawdza każdy ciąg w tablicy ciągów, a każdy ciąg zawierający „a” zostanie wybrany w trzecim wierszu.
Oto imiona: Bella, Luna, Simba i Oscar.
100 %
Dane wyjściowe

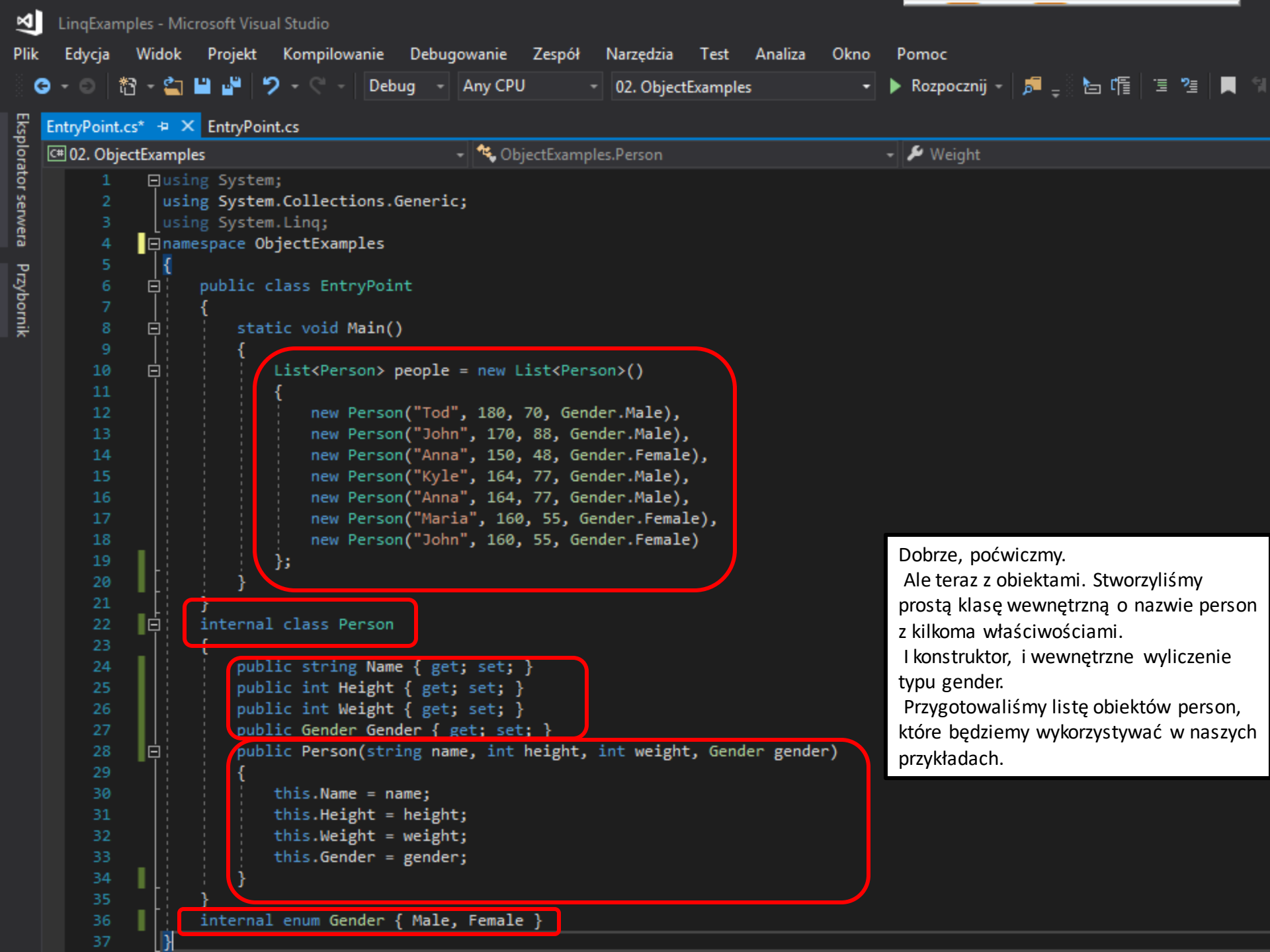

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs*
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
using System;
using System.Collections.Generic;
using System.Linq;
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 moreSpecificCats = from cat in catNames
where (cat.Contains("a")) && (cat.Length < 5)
select cat;
Console.WriteLine(string.Join(", ", moreSpecificCats));
}
}
}
Możesz także mieć wiele wyrażeń - wiele warunków w klauzuli WHERE.
Możemy więc również wybrać łańcuchy, które zawierają literę „a”, a także mają mniej niż pięć znaków.
Luna to jedyne imię, które zawiera literę „a” i ma długość mniejszą niż pięć.
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 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 . . .
Możesz także podzielić swoje warunki na wiele wyrażeń w wielu wierszach.
Zamiast więc mówić, że długość jest mniejsza niż 5 i łańcuch zawiera „a”, możemy to zrobić w dwóch wierszach.
Wtedy nie potrzebujemy już nawiasów.
Będzie działać w ten sam sposób.
Jeśli mamy długie i skomplikowane warunki, w ten sposób możemy podzielić je na osobne warunki, aby były bardziej czytelne.
Będą działać dokładnie tak samo.
100 %
Dane wyjściowe

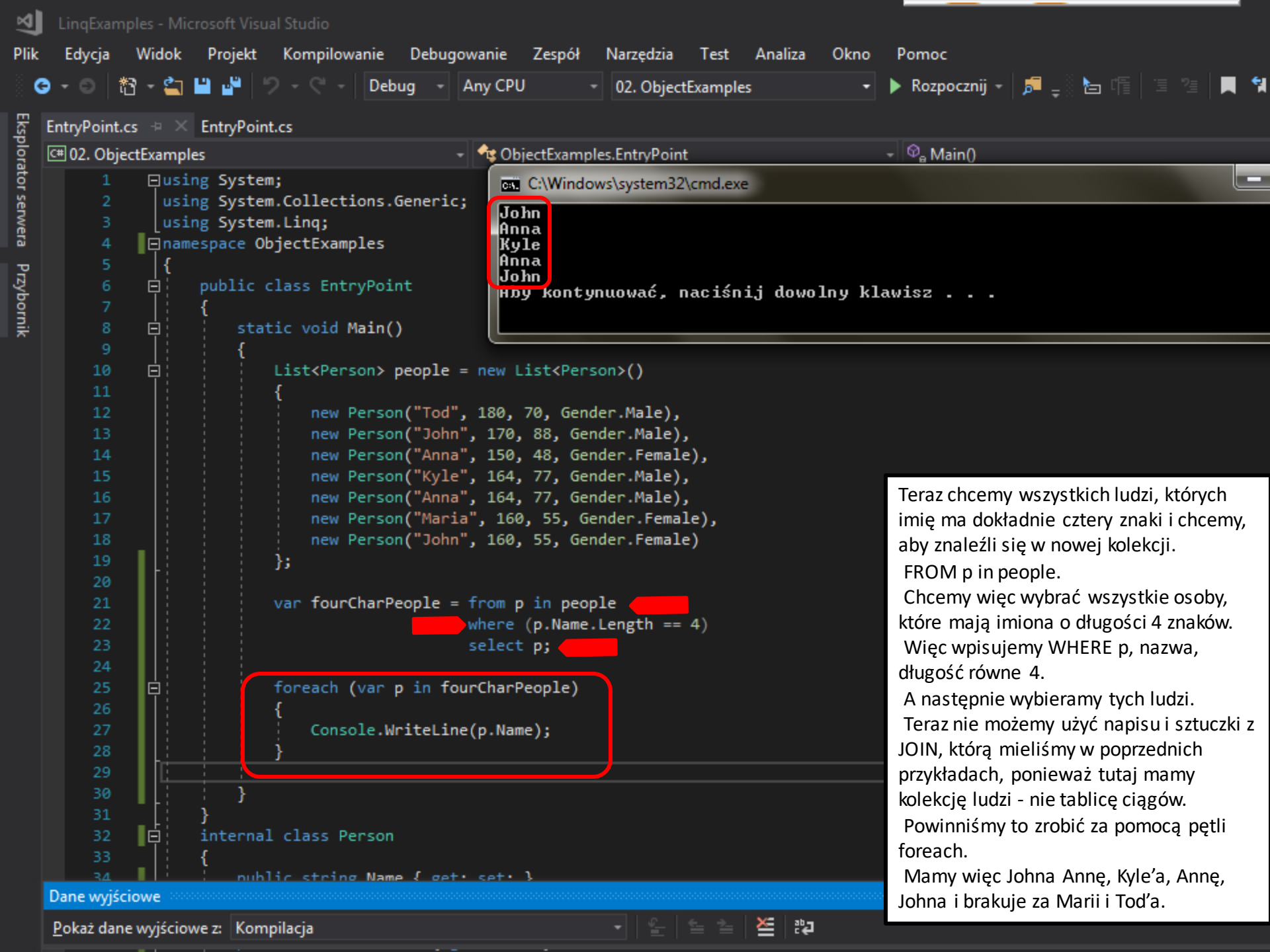

LinqExamples - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. SimpleExamples
Rozpocznij
EntryPoint.cs
01. SimpleExamples
SimpleExamples.EntryPoint
Main()
public class EntryPoint
{
static void Main()
{
string sentence = "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 .
Ostatnia rzecz, którą chcę pokazać. Możesz zobaczyć liczby od 5 do 10, ale nie są one uporządkowane.
Możemy je uporządkować za pomocą słowa kluczowego orderby.
Możesz również uporządkować je w kolejności malejącej za pomocą słowa kluczowego descending.
Nie musimy używać słowa ascending, ponieważ rosnąco jest domyślnym sposobem porządkowania liczb.
100 %
Dane wyjściowe

LINQ Queries on Objects

Zasadniczo to, co można robić z LinQ, będzie bardziej interesujące w następnym wykładzie. Zapytania LinQ dotyczące obiektów.



Dobrze, poćwiczmy.
Ale teraz z obiektami. Stworzyliśmy prostą klasę wewnętrzną o nazwie person z kilkoma właściwościami.
I konstruktor, i wewnętrzne wyliczenie typu gender.
Przygotowaliśmy listę obiektów person, które będziemy wykorzystywać w naszych przykładach.



Teraz chcemy wszystkich ludzi, których imię ma dokładnie cztery znaki i chcemy, aby znaleźli się w nowej kolekcji.

FROM p in people.

Chcemy więc wybrać wszystkie osoby, które mają imiona o długości 4 znaków.

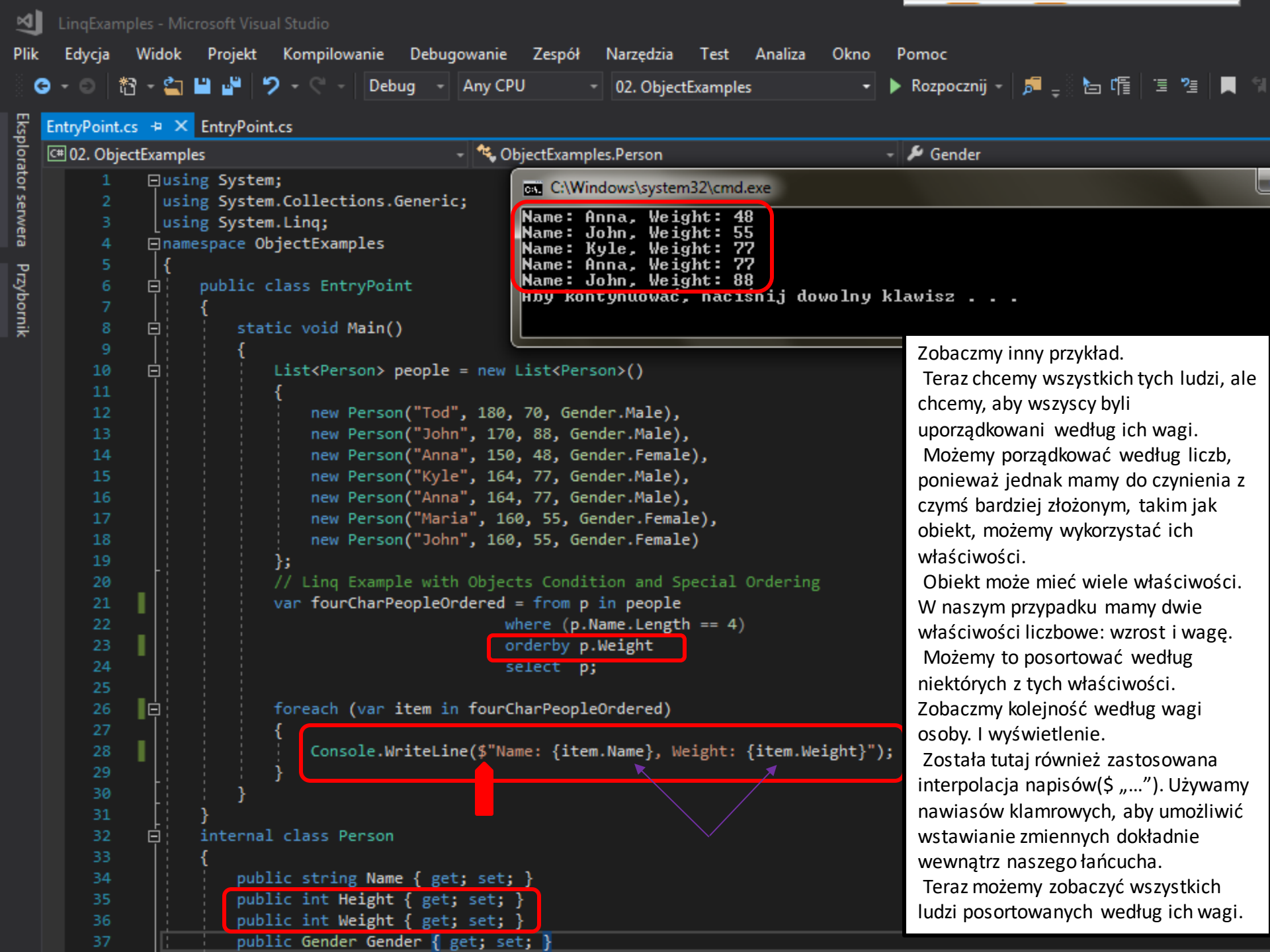
Więc wpisujemy WHERE p, nazwa, długość równe 4.

A następnie wybieramy tych ludzi.

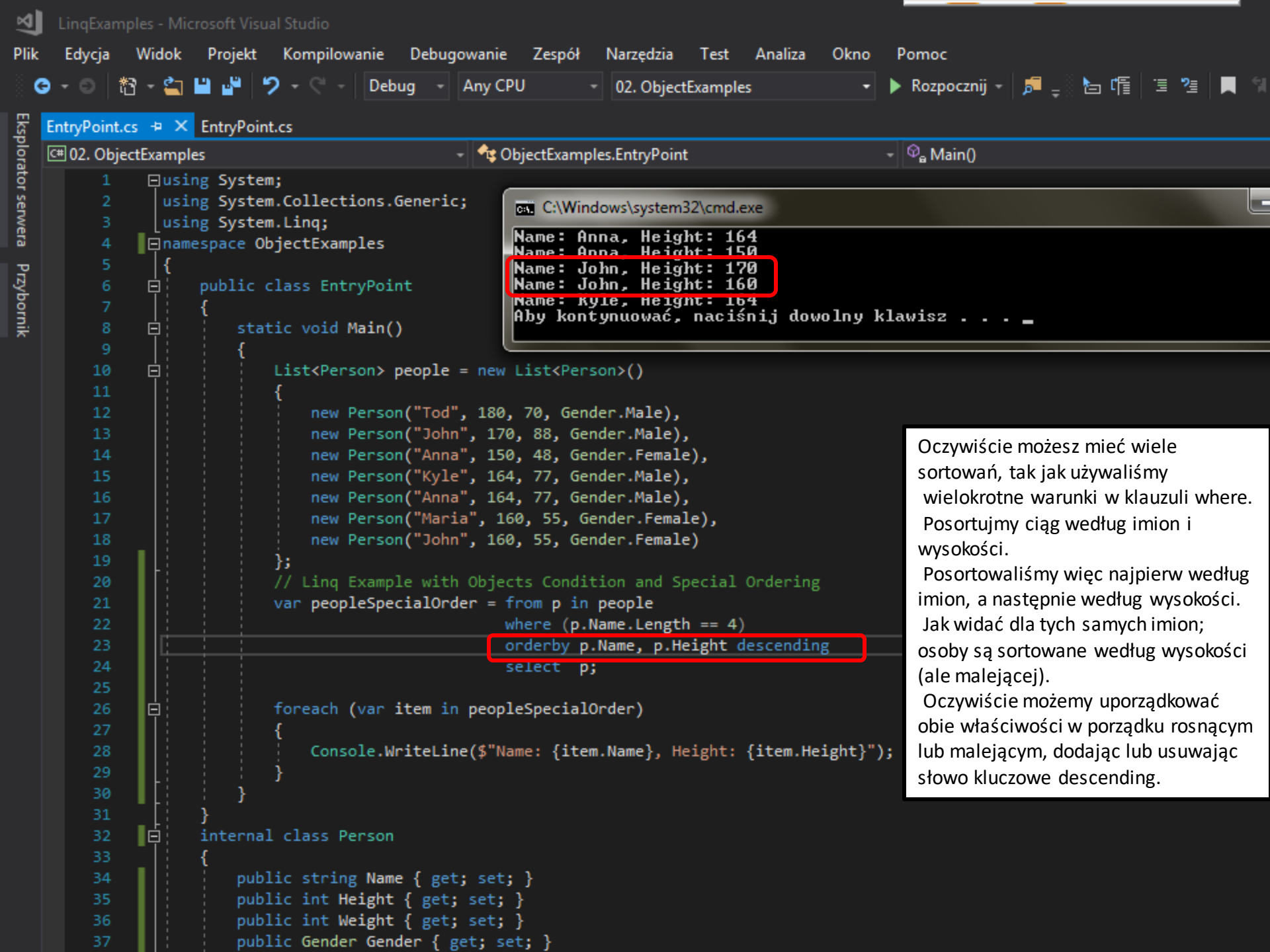
Teraz nie możemy użyć napisu i sztuczki z JOIN, którą mieliśmy w poprzednich przykładach, ponieważ tutaj mamy kolekcję ludzi - nie tablicę ciągów.

Powinniśmy to zrobić za pomocą pętli foreach.

Mamy więc Johna Annę, Kyle'a, Annę, Johna i brakuje za Marii i Tod'a.



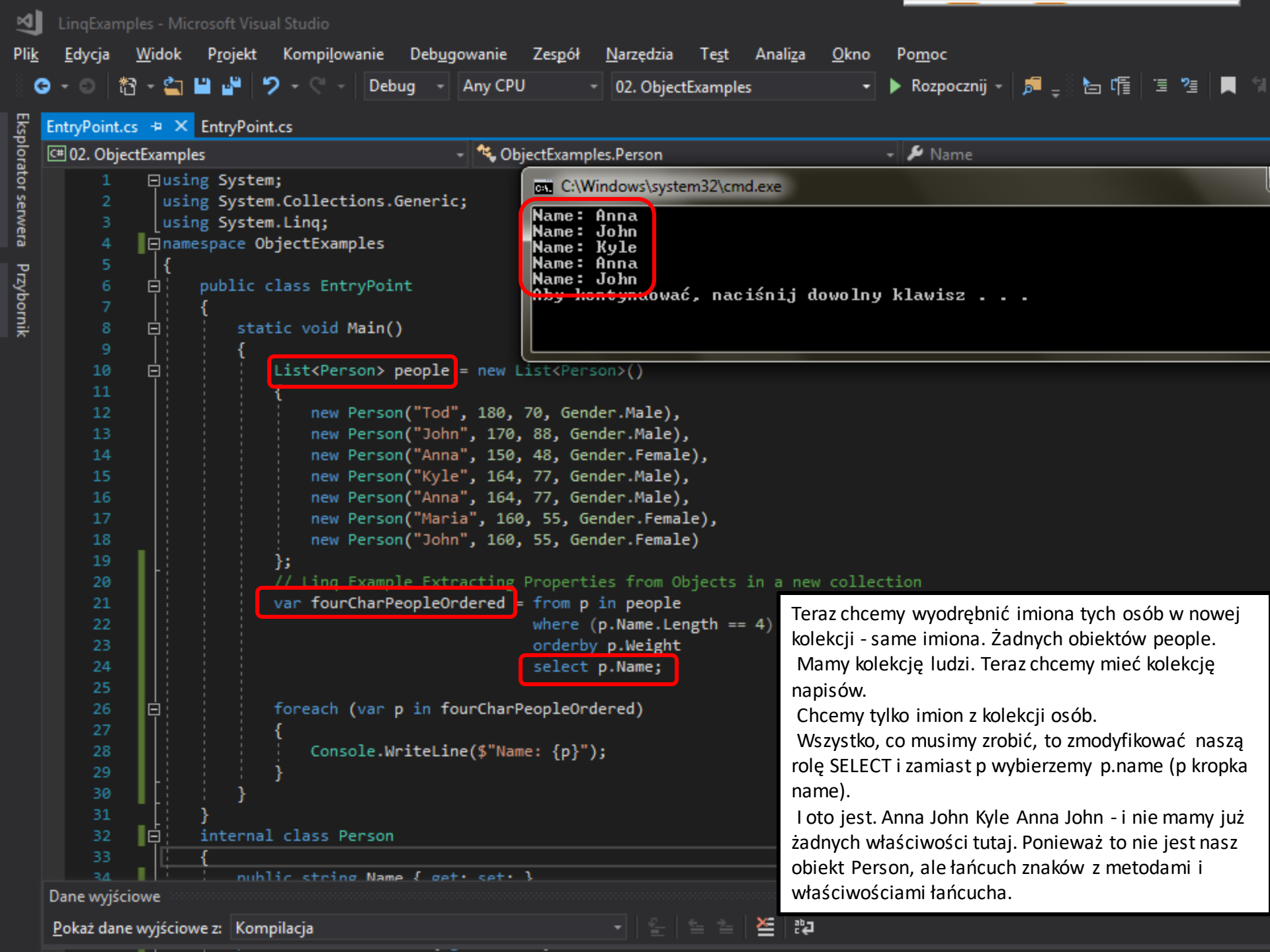
Zobaczmy inny przykład.
Teraz chcemy wszystkich tych ludzi, ale chcemy, aby wszyscy byli uporządkowani według ich wagi.
Możemy porządkować według liczb, ponieważ jednak mamy do czynienia z czymś bardziej złożonym, takim jak obiekt, możemy wykorzystać ich właściwości.
Obiekt może mieć wiele właściwości. W naszym przypadku mamy dwie właściwości liczbowe: wzrost i wagę.
Możemy to posortować według niektórych z tych właściwości.
Zobaczmy kolejność według wagi osoby. I wyświetlenie.
Została tutaj również zastosowana interpolacja napisów($ „…”). Używamy nawiasów klamrowych, aby umożliwić wstawianie zmiennych dokładnie wewnątrz naszego łańcucha.
Teraz możemy zobaczyć wszystkich ludzi posortowanych według ich wagi.



Oczywiście możesz mieć wiele sortowań, tak jak używaliśmy wielokrotne warunki w klauzuli where. Posortujmy ciąg według imion i wysokości.

Posortowaliśmy więc najpierw według imion, a następnie według wysokości. Jak widać dla tych samych imion; osoby są sortowane według wysokości (ale malejącej).

Oczywiście możemy uporządkować obie właściwości w porządku rosnącym lub malejącym, dodając lub usuwając słowo kluczowe descending.



Teraz chcemy wyodrębnić imiona tych osób w nowej kolekcji - same imiona. Żadnych obiektów people.

Mamy kolekcję ludzi. Teraz chcemy mieć kolekcję napisów.

Chcemy tylko imion z kolekcji osób.

Wszystko, co musimy zrobić, to zmodyfikować naszą rolę SELECT i zamiast p wybierzemy p.name (p kropka name).

I oto jest. Anna John Kyle Anna John - i nie mamy już żadnych właściwości tutaj. Ponieważ to nie jest nasz obiekt Person, ale łańcuch znaków z metodami i właściwościami łańcucha.

The End

Teraz jest koniec prezentacji, ale nie koniec LinQ.
Będziemy kontynuować LinQ, ale w inny sposób - używając wyrażenia Lambda.