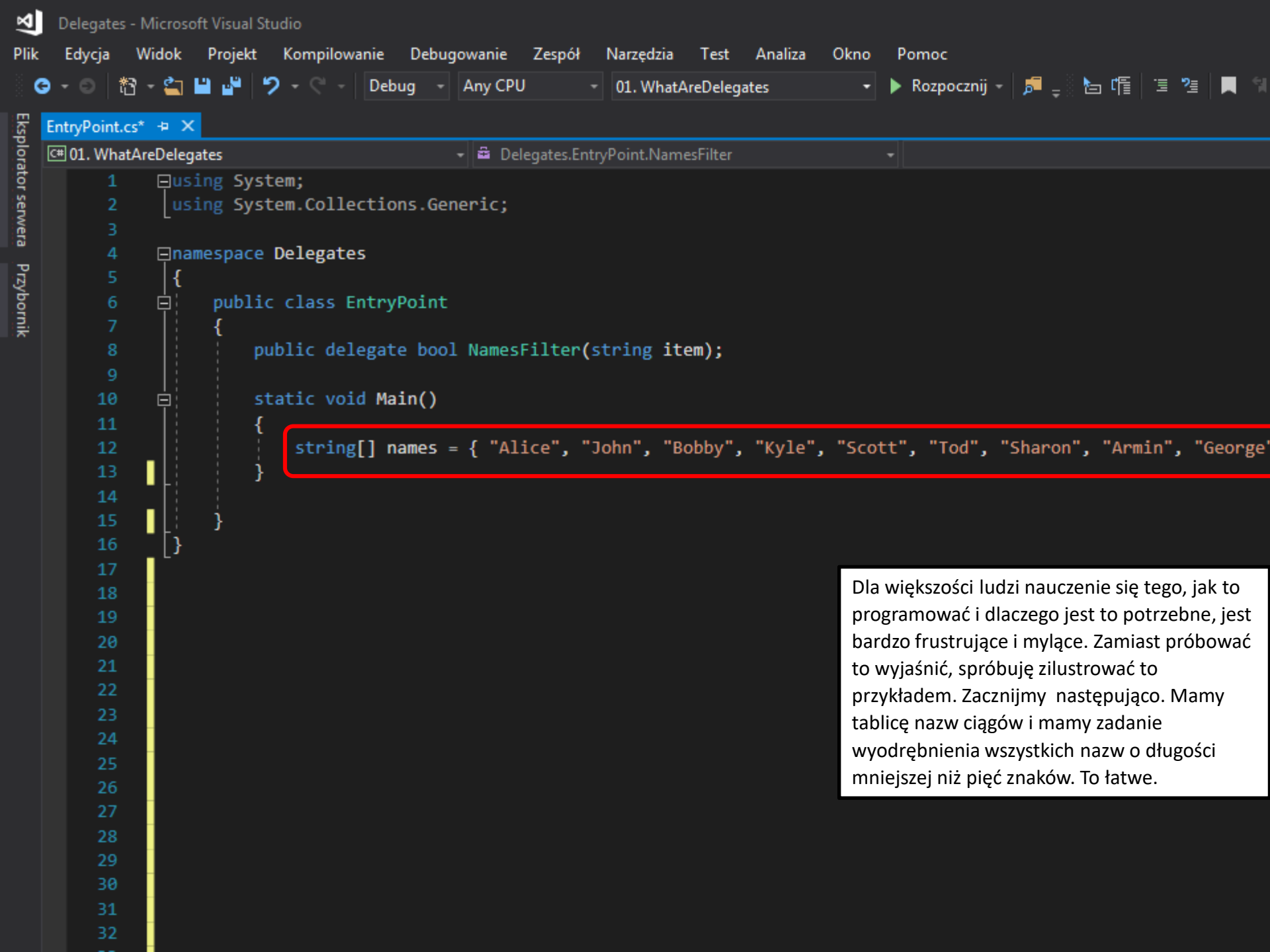
Delegates – methods as variables


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
Eksplorator serwera
Przybornik
EntryPoint.cs*
01. WhatAreDelegates
Delegates.EntryPoint.NamesFilter
1 using System;
2 using System.Collections.Generic;
3
4 namespace Delegates
5 {
6 public class EntryPoint
7 {
8 public delegate bool NamesFilter(string item);
9
10 static void Main()
11 {
12 string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
13 }
14
15 }
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
Dla większości ludzi nauczenie się tego, jak to programować i dlaczego jest to potrzebne, jest bardzo frustrujące i mylące. Zamiast próbować to wyjaśnić, spróbuję zilustrować to przykładem. Zacznijmy następująco. Mamy tablicę nazw ciągów i mamy zadanie wyodrębnienia wszystkich nazw o długości mniejszej niż pięć znaków. To łatwe.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 01. WhatAreDelegates | Rozpocznij

EntryPoint.cs

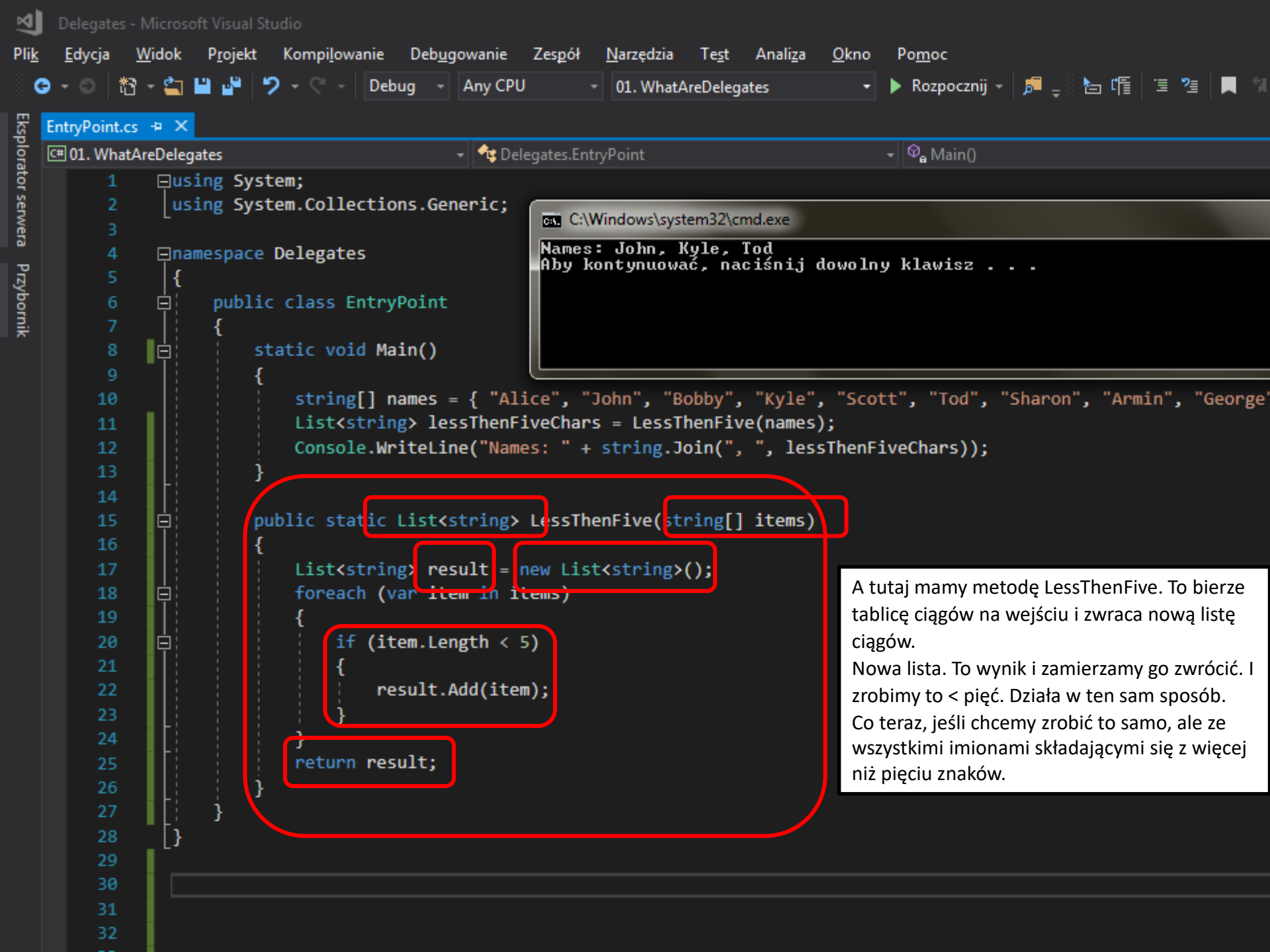
01. WhatAreDelegates | Delegates.EntryPoint | Main()

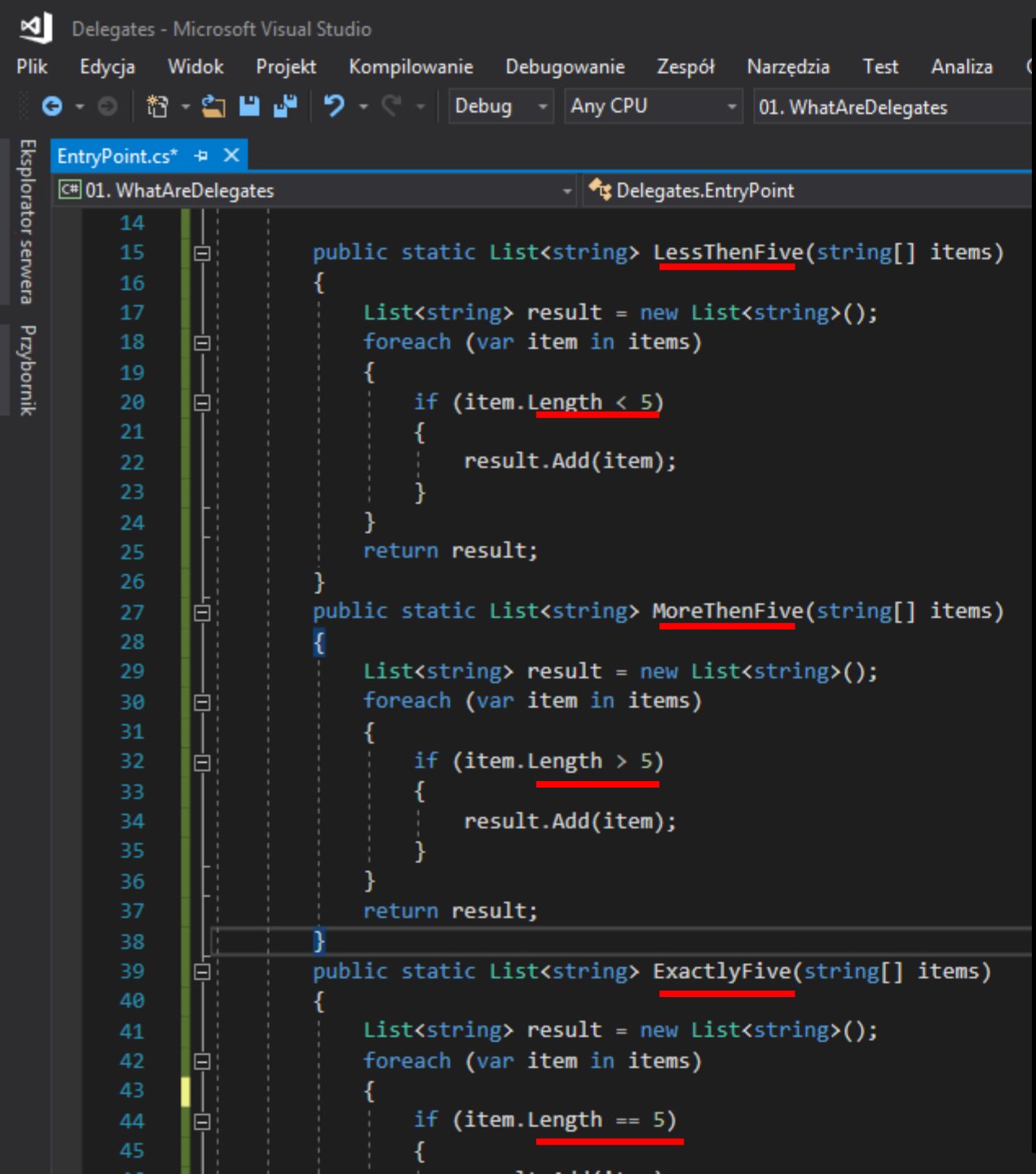
```
1  using System;
2  using System.Collections.Generic;
3
4  namespace Delegates
5  {
6      public class EntryPoint
7      {
8
9          static void Main()
10         {
11             string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
12
13             List<string> lessThenFiveChars = new List<string>();
14
15             foreach (var item in names)
16             {
17                 if (item.Length < 5)
18                 {
19                     lessThenFiveChars.Add(item);
20                 }
21             }
22
23             Console.WriteLine("Names: " + string.Join(", ", lessThenFiveChars));
24         }
25
26     }
27 }
28
29
30
31
32
```

```
C:\Windows\system32\cmd.exe
Names: John, Kyle, Tod
Aby kontynuować, naciśnij dowolny klawisz . . .
```

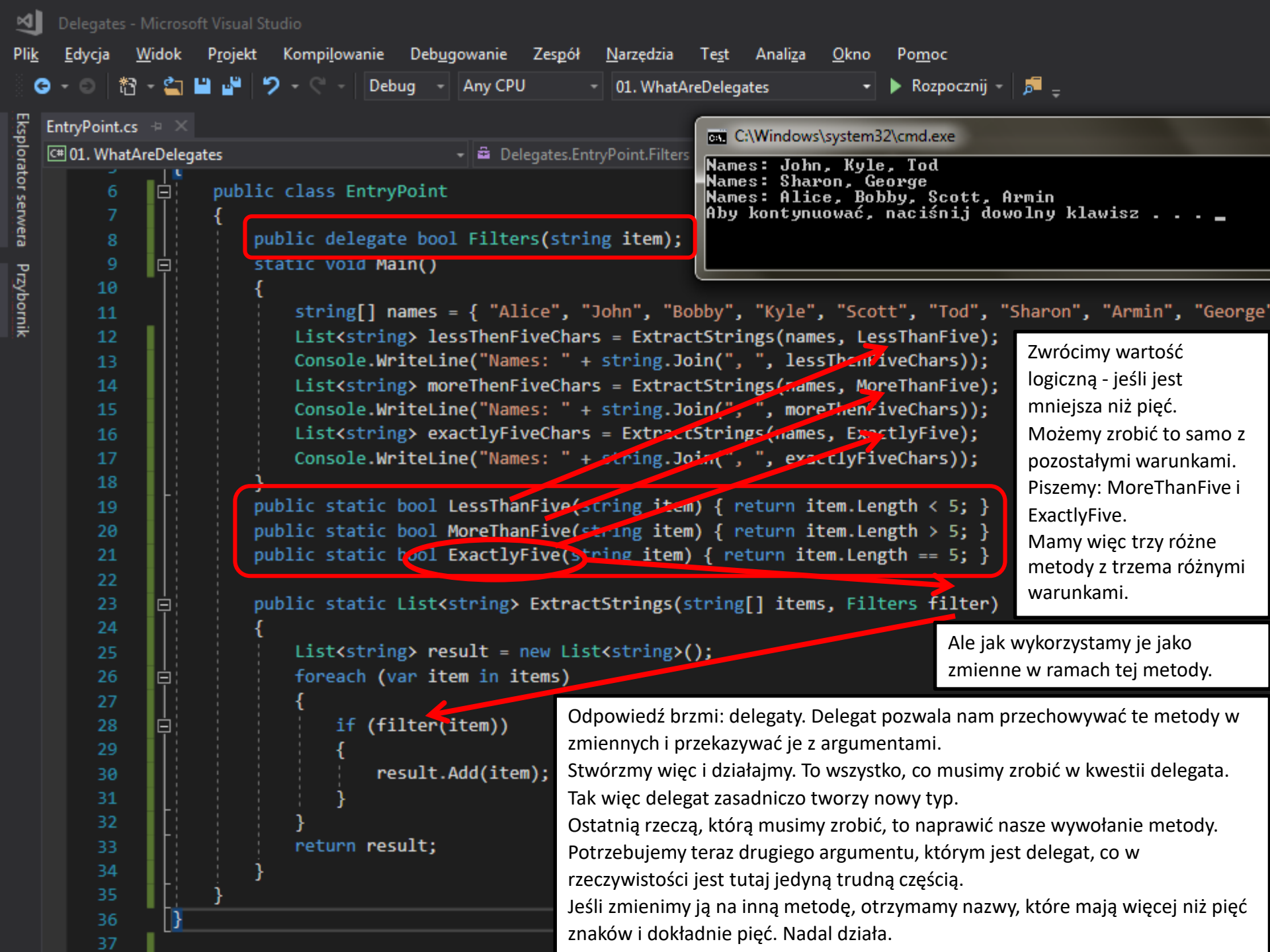
Wszystko, co musimy zrobić, to utworzyć pętlę z warunkiem IF i wyodrębnić je w nowej kolekcji. Więc zróbmy to. Napiszmy: lista stringów, których będziemy używać, lista stringów do przechowywania tych nowych mniej niż 5 znaków jest równa nowej liście stringów. I stworzymy dla nich pętlę for.

I sprawdzimy, czy ta długość jest mniejsza niż 5. A jeśli tak, to dodamy je do listy lessThenFiveChars. Wystarczająco proste. Spróbujmy. Tak więc otrzymujemy: John, Kyle i Tod.

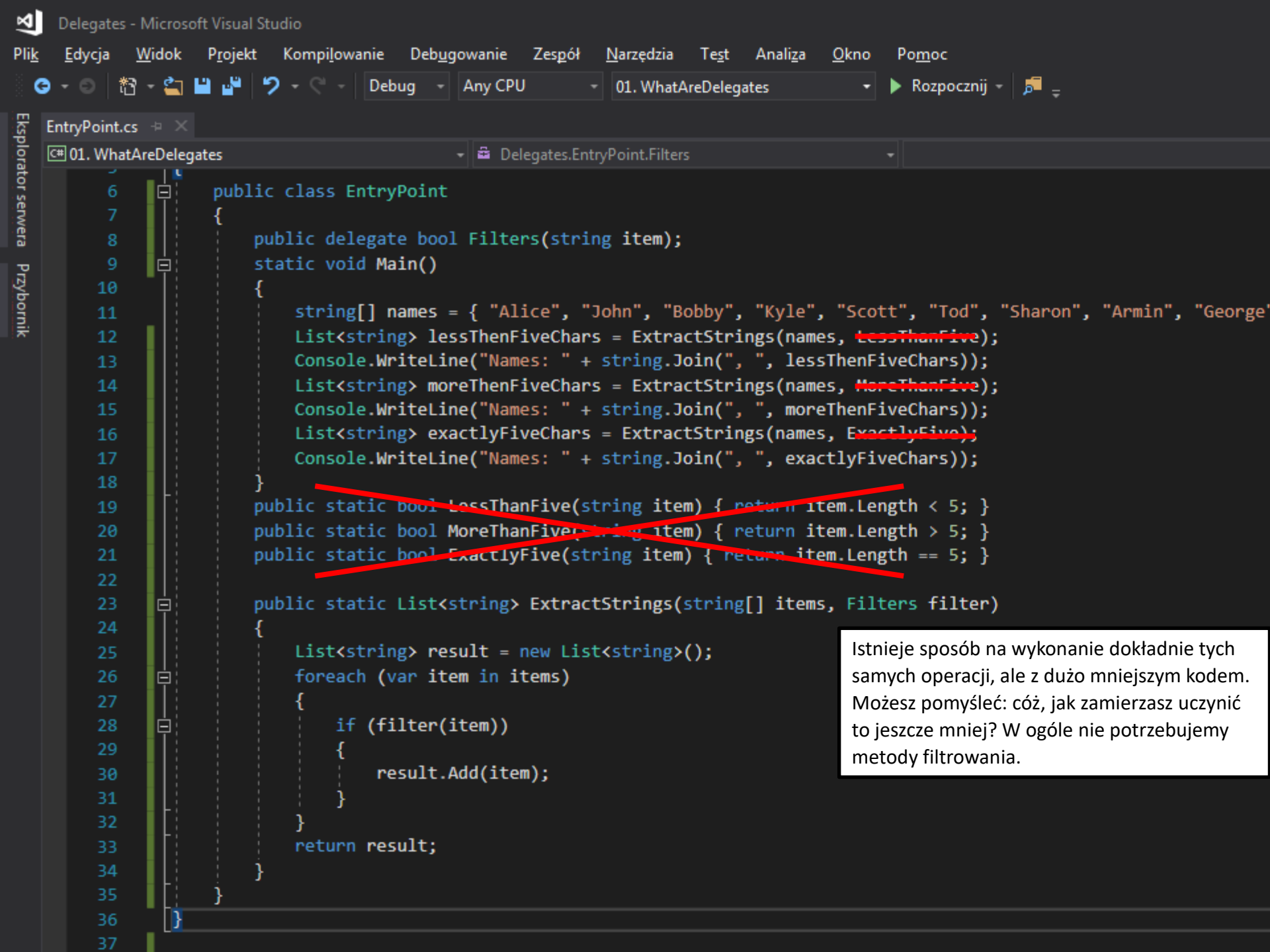

Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System;
using System.Collections.Generic;
namespace Delegates
{
public class EntryPoint
{
static void Main()
{
string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
List<string> lessThenFiveChars = LessThenFive(names);
Console.WriteLine("Names: " + string.Join(", ", lessThenFiveChars));
}
public static List<string> LessThenFive(string[] items)
{
List<string> result = new List<string>();
foreach (var item in items)
{
if (item.Length < 5)
{
result.Add(item);
}
}
return result;
}
}
}
C:\Windows\system32\cmd.exe
Names: John, Kyle, Tod
Aby kontynuować, naciśnij dowolny klawisz . . .
A tutaj mamy metodę LessThenFive. To bierze tablicę ciągów na wejściu i zwraca nową listę ciągów.
Nowa lista. To wynik i zamierzamy go zwrócić. I zrobimy to < pięć. Działa w ten sam sposób.
Co teraz, jeśli chcemy zrobić to samo, ale ze wszystkimi imionami składającymi się z więcej niż pięciu znaków.



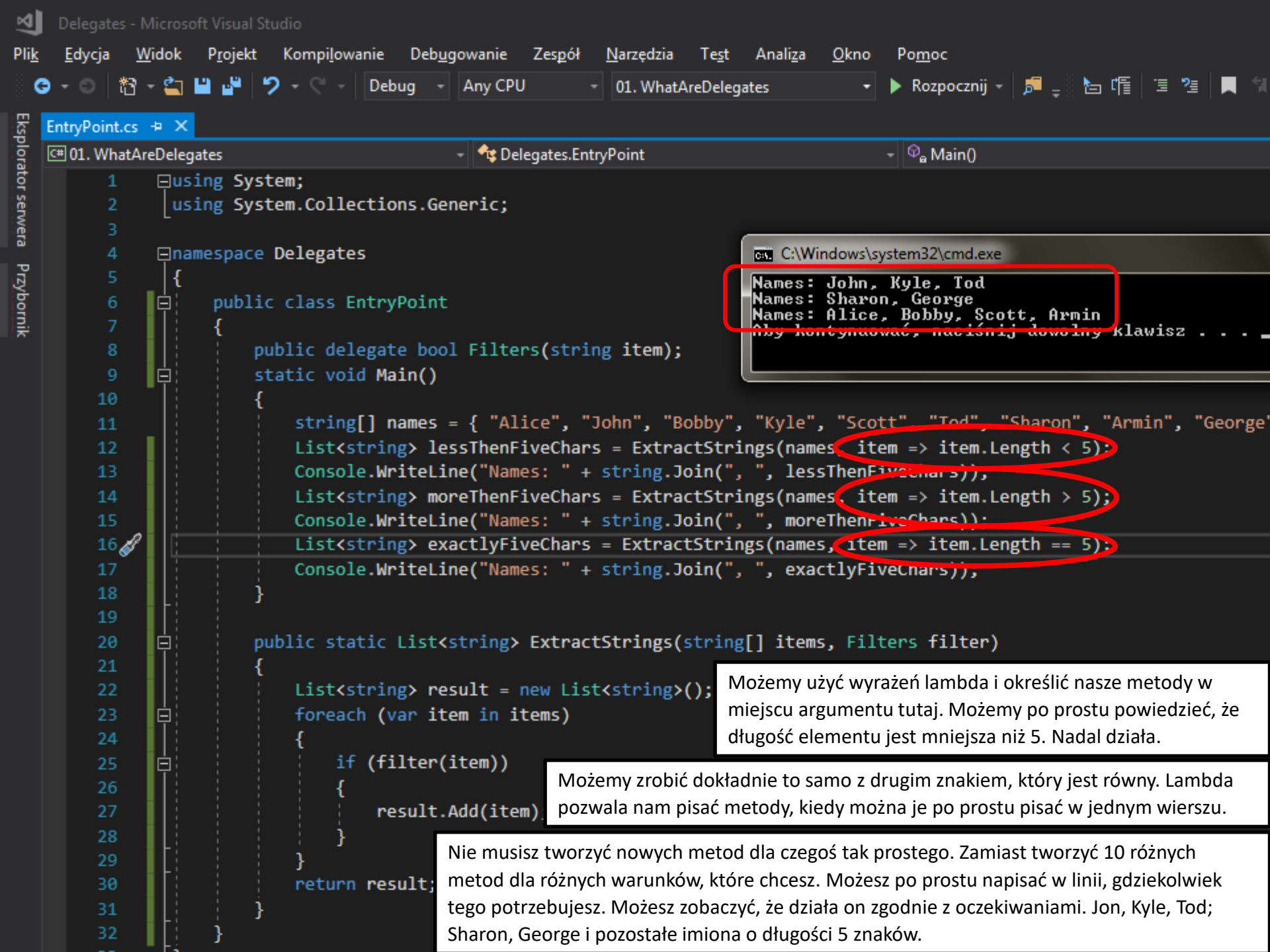
Ok, to wszystko. Możemy stworzyć inną metodę, aby to zrobić.
Ale co jeśli chcemy tylko te, które mają dokładnie pięć znaków. Otóż to. Co jeśli wszystkie mają mieć 10 znaków lub więcej.
Widzimy, jak szybko zaczynamy otrzymywać wiele różnych przypadków i nie jesteśmy w stanie stworzyć nieskończonej liczby metod ich rozwiązania. W tym miejscu pojawia się delegat.
Możesz wziąć kawałek kodu i zmieniać go.
Pokażmy więc, co rozumiemy przez to.
Rzućmy okiem na naszą metodę, którą stworzyliśmy.
Mamy tylko jedną część kodu, która jest zmienna. I to jest warunek, który filtruje nazwy.
Jeśli chcemy uzyskać nazwy zawierające więcej niż pięć znaków, musimy po prostu zmienić znak.
To miejsce, w którym zamierzamy zmienić ten fragment kodu. Aby wyodrębnić ten fragment kodu, musimy się trochę zastanowić. Czego potrzebuje jako danych wejściowych i jakie dane wyjściowe zwróci. (Coś jak podczas tworzenia metody).
Potrzebujemy łańcucha wejściowego i zwrócimy wartość logiczną: prawda lub fałsz, która wskaże, czy przedmiot ma więcej niż pięć znaków, czy nie.
Więc po prostu wyodrębnimy nasze warunki do nowych metod, które zawierają tylko warunki i nic więcej.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint.Filters
C:\Windows\system32\cmd.exe
Names: John, Kyle, Tod
Names: Sharon, George
Names: Alice, Bobby, Scott, Armin
Aby kontynuować, naciśnij dowolny klawisz . . .
public class EntryPoint
public delegate bool Filters(string item);
static void Main()
string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
List<string> lessThenFiveChars = ExtractStrings(names, LessThanFive);
Console.WriteLine("Names: " + string.Join(", ", lessThenFiveChars));
List<string> moreThenFiveChars = ExtractStrings(names, MoreThanFive);
List<string> exactlyFiveChars = ExtractStrings(names, ExactlyFive);
public static bool LessThanFive(string item) { return item.Length < 5; }
public static bool MoreThanFive(string item) { return item.Length > 5; }
public static bool ExactlyFive(string item) { return item.Length == 5; }
public static List<string> ExtractStrings(string[] items, Filters filter)
List<string> result = new List<string>();
foreach (var item in items)
if (filter(item))
result.Add(item);
return result;
Zwrócimy wartość logiczną - jeśli jest mniejsza niż pięć.
Możemy zrobić to samo z pozostałymi warunkami.
Piszemy: MoreThanFive i ExactlyFive.
Mamy więc trzy różne metody z trzema różnymi warunkami.
Ale jak wykorzystamy je jako zmienne w ramach tej metody.
Odpowiedź brzmi: delegaty. Delegat pozwala nam przechowywać te metody w zmiennych i przekazywać je z argumentami.
Stwórzmy więc i działajmy. To wszystko, co musimy zrobić w kwestii delegata.
Tak więc delegat zasadniczo tworzy nowy typ.
Ostatnią rzeczą, którą musimy zrobić, to naprawić nasze wywołanie metody.
Potrzebujemy teraz drugiego argumentu, którym jest delegat, co w rzeczywistości jest tutaj jedyną trudną częścią.
Jeśli zmienimy ją na inną metodę, otrzymamy nazwy, które mają więcej niż pięć znaków i dokładnie pięć. Nadal działa.

Delegates and lambda expressions

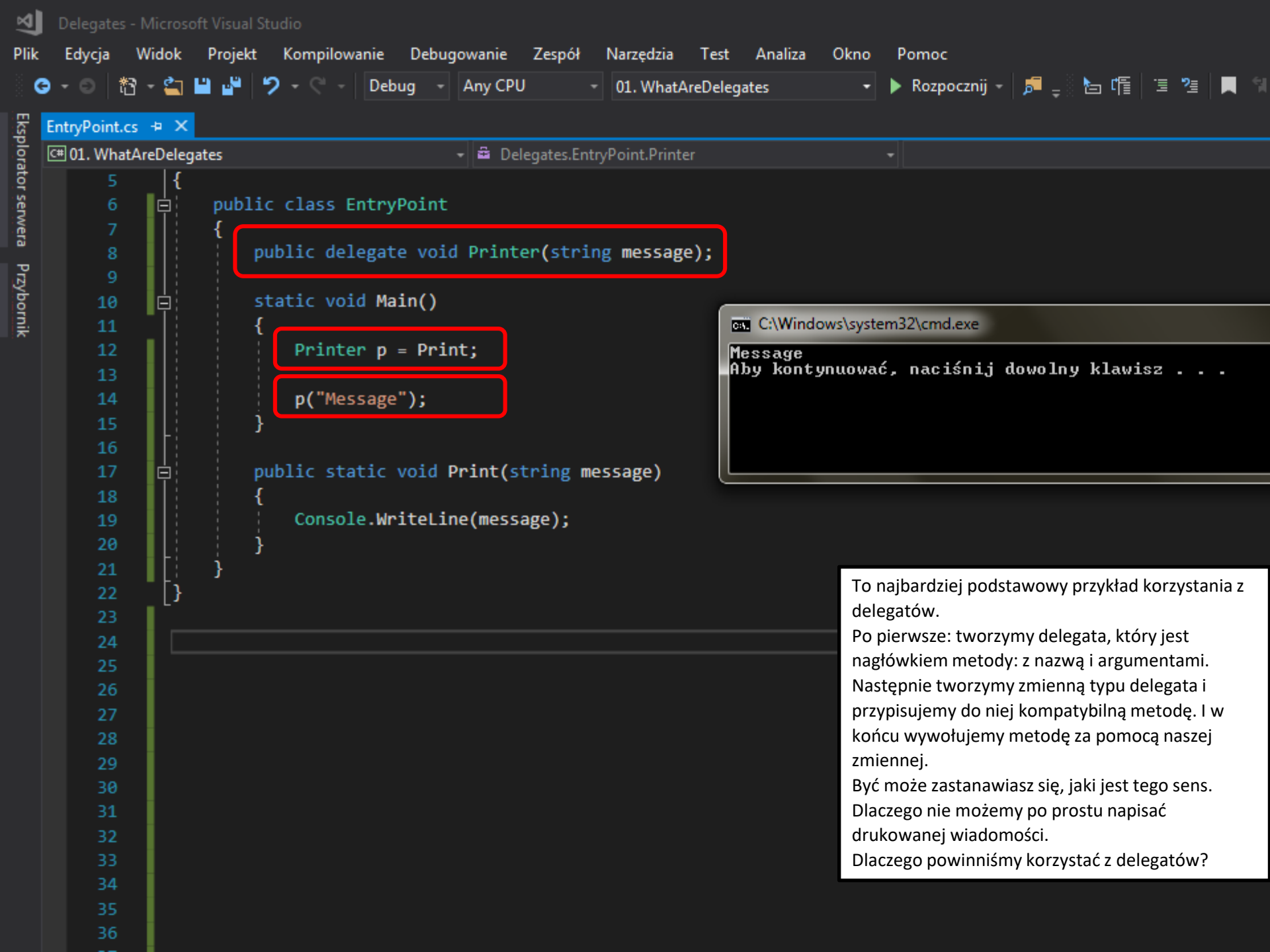


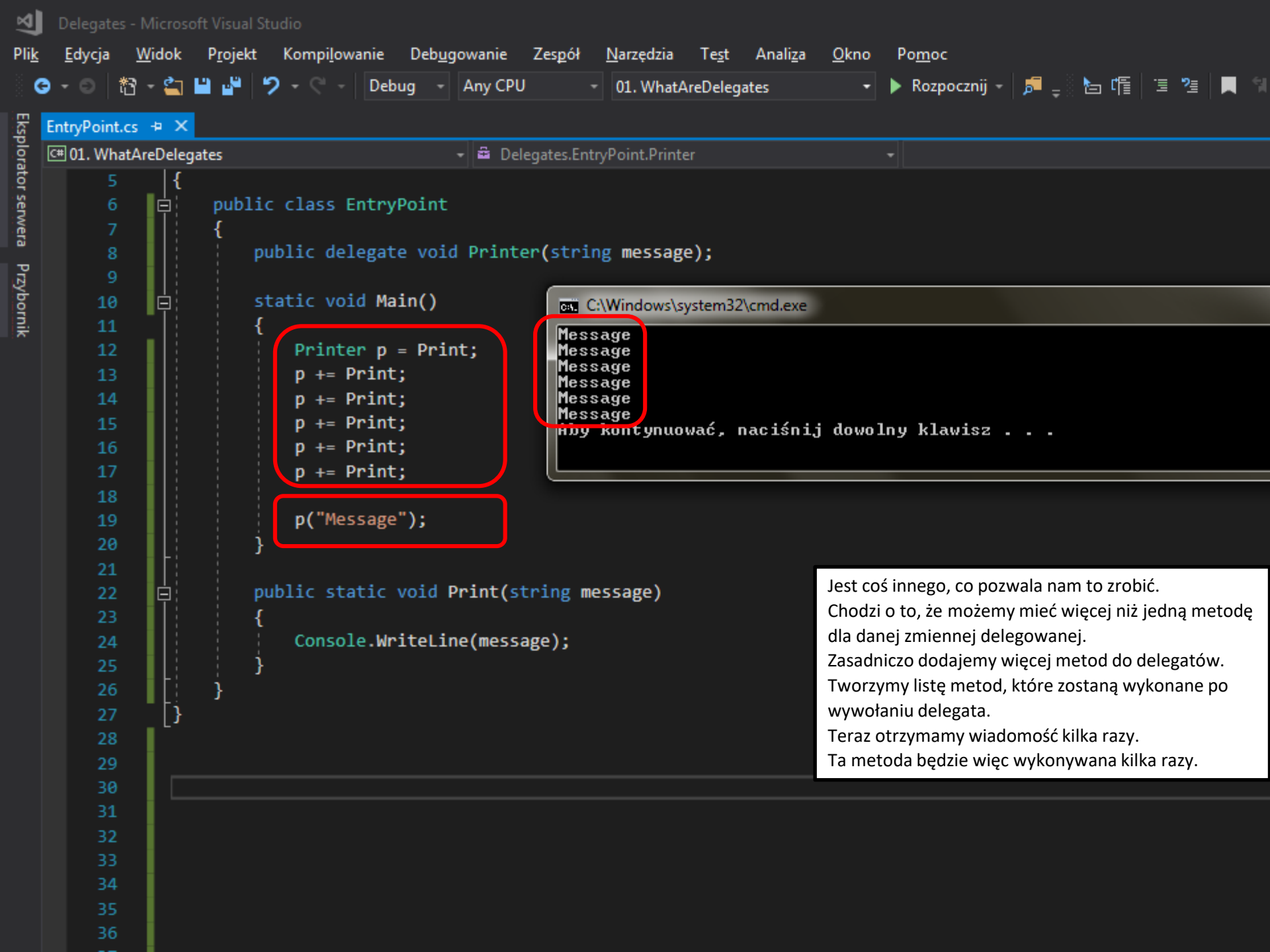
Istnieje sposób na wykonanie dokładnie tych samych operacji, ale z dużo mniejszym kodem. Możesz pomyśleć: cóż, jak zamierzasz uczynić to jeszcze mniej? W ogóle nie potrzebujemy metody filtrowania.

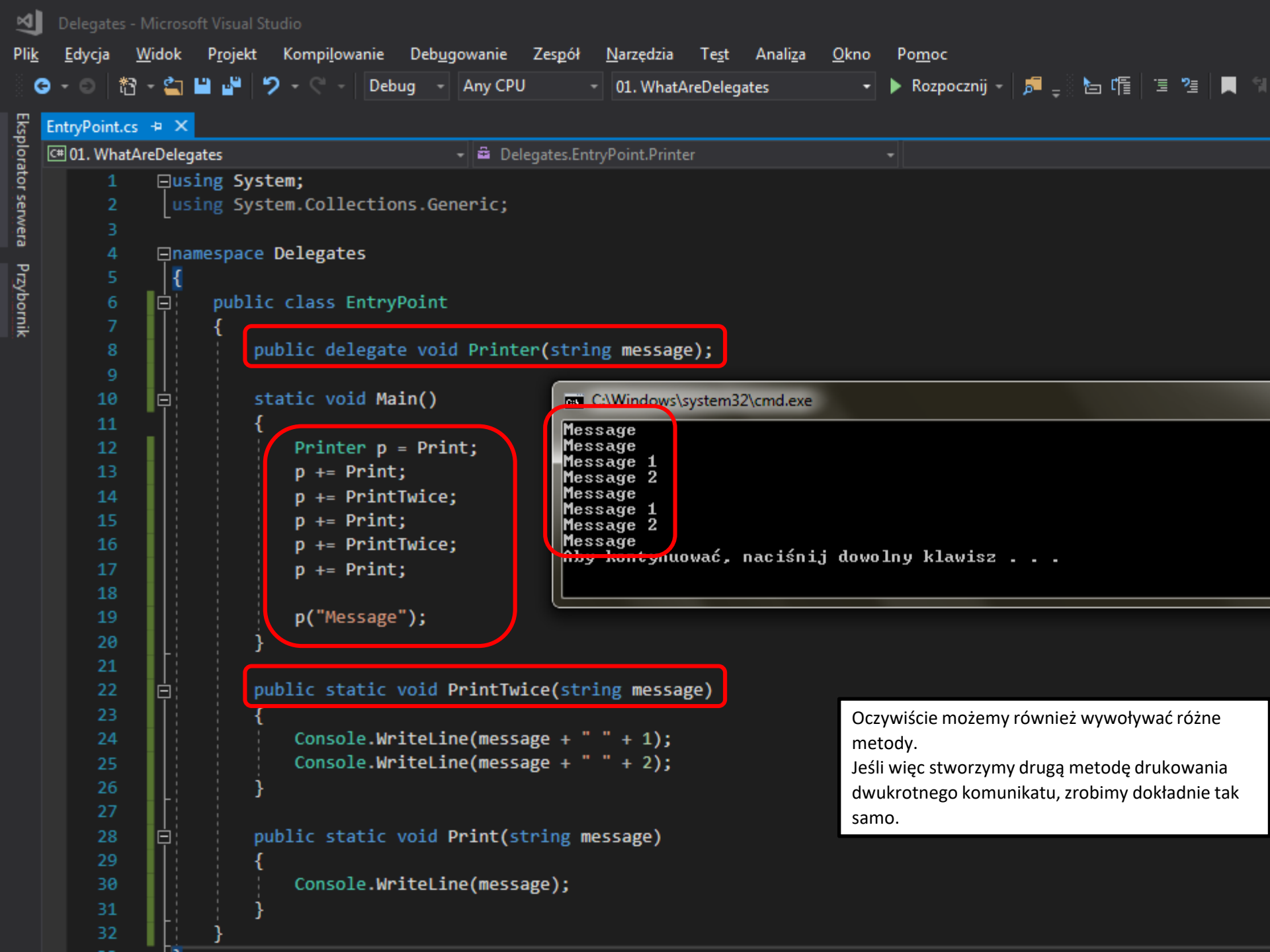

Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System;
using System.Collections.Generic;
namespace Delegates
{
public class EntryPoint
{
public delegate bool Filters(string item);
static void Main()
{
string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
List<string> lessThenFiveChars = ExtractStrings(names, item => item.Length < 5);
Console.WriteLine("Names: " + string.Join(", ", lessThenFiveChars));
List<string> moreThenFiveChars = ExtractStrings(names, item => item.Length > 5);
Console.WriteLine("Names: " + string.Join(", ", moreThenFiveChars));
List<string> exactlyFiveChars = ExtractStrings(names, item => item.Length == 5);
Console.WriteLine("Names: " + string.Join(", ", exactlyFiveChars));
}
public static List<string> ExtractStrings(string[] items, Filters filter)
{
List<string> result = new List<string>();
foreach (var item in items)
{
if (filter(item))
{
result.Add(item);
}
}
return result;
}
}
C:\Windows\system32\cmd.exe
Names: John, Kyle, Tod
Names: Sharon, George
Names: Alice, Bobby, Scott, Armin
Możemy użyć wyrażeń lambda i określić nasze metody w miejscu argumentu tutaj. Możemy po prostu powiedzieć, że długość elementu jest mniejsza niż 5. Nadal działa.
Możemy zrobić dokładnie to samo z drugim znakiem, który jest równy. Lambda pozwala nam pisać metody, kiedy można je po prostu pisać w jednym wierszu.
Nie musisz tworzyć nowych metod dla czegoś tak prostego. Zamiast tworzyć 10 różnych metod dla różnych warunków, które chcesz. Możesz po prostu napisać w linii, gdziekolwiek tego potrzebujesz. Możesz zobaczyć, że działa on zgodnie z oczekiwaniami. Jon, Kyle, Tod; Sharon, George i pozostałe imiona o długości 5 znaków.

Delegates chaining with many methods

Stwórzmy jedną bardzo prostą metodę, która po prostu wydrukuje coś na konsoli.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint.Printer
{
public class EntryPoint
{
public delegate void Printer(string message);
static void Main()
{
Printer p = Print;
p("Message");
}
public static void Print(string message)
{
Console.WriteLine(message);
}
}
}
C:\Windows\system32\cmd.exe
Message
Aby kontynuować, naciśnij dowolny klawisz . . .
To najbardziej podstawowy przykład korzystania z delegatów.
Po pierwsze: tworzymy delegata, który jest nagłówkiem metody: z nazwą i argumentami.
Następnie tworzymy zmienną typu delegata i przypisujemy do niej kompatybilną metodę. I w końcu wywołujemy metodę za pomocą naszej zmiennej.
Być może zastanawiasz się, jaki jest tego sens.
Dlaczego nie możemy po prostu napisać drukowanej wiadomości.
Dlaczego powinniśmy korzystać z delegatów?


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint.Printer
Eksplorator serwera
Przybornik
5 {
6 public class EntryPoint
7 {
8 public delegate void Printer(string message);
9
10 static void Main()
11 {
12 Printer p = Print;
13 p += Print;
14 p += Print;
15 p += Print;
16 p += Print;
17 p += Print;
18
19 p("Message");
20 }
21
22 public static void Print(string message)
23 {
24 Console.WriteLine(message);
25 }
26 }
27 }
28
29
30
31
32
33
34
35
36
C:\Windows\system32\cmd.exe
Message
Message
Message
Message
Message
Message
Aby kontynuować, naciśnij dowolny klawisz . . .
Jest coś innego, co pozwala nam to zrobić.
Chodzi o to, że możemy mieć więcej niż jedną metodę dla danej zmiennej delegowanej.
Zasadniczo dodajemy więcej metod do delegatów.
Tworzymy listę metod, które zostaną wykonane po wywołaniu delegata.
Teraz otrzymamy wiadomość kilka razy.
Ta metoda będzie więc wykonywana kilka razy.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint.Printer
Eksplorator serwera
Przybornik
1 using System;
2 using System.Collections.Generic;
3
4 namespace Delegates
5 {
6 public class EntryPoint
7 {
8 public delegate void Printer(string message);
9
10 static void Main()
11 {
12 Printer p = Print;
13 p += Print;
14 p += PrintTwice;
15 p += Print;
16 p += PrintTwice;
17 p += Print;
18
19 p("Message");
20 }
21
22 public static void PrintTwice(string message)
23 {
24 Console.WriteLine(message + " " + 1);
25 Console.WriteLine(message + " " + 2);
26 }
27
28 public static void Print(string message)
29 {
30 Console.WriteLine(message);
31 }
32 }
C:\Windows\system32\cmd.exe
Message
Message
Message 1
Message 2
Message
Message 1
Message 2
Message
Aby kontynuować, naciśnij dowolny klawisz . . .
Oczywiście możemy również wywoływać różne metody.
Jeśli więc stworzymy drugą metodę drukowania dwukrotnego komunikatu, zrobimy dokładnie tak samo.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 01. WhatAreDelegates | Rozpocznij

EntryPoint.cs

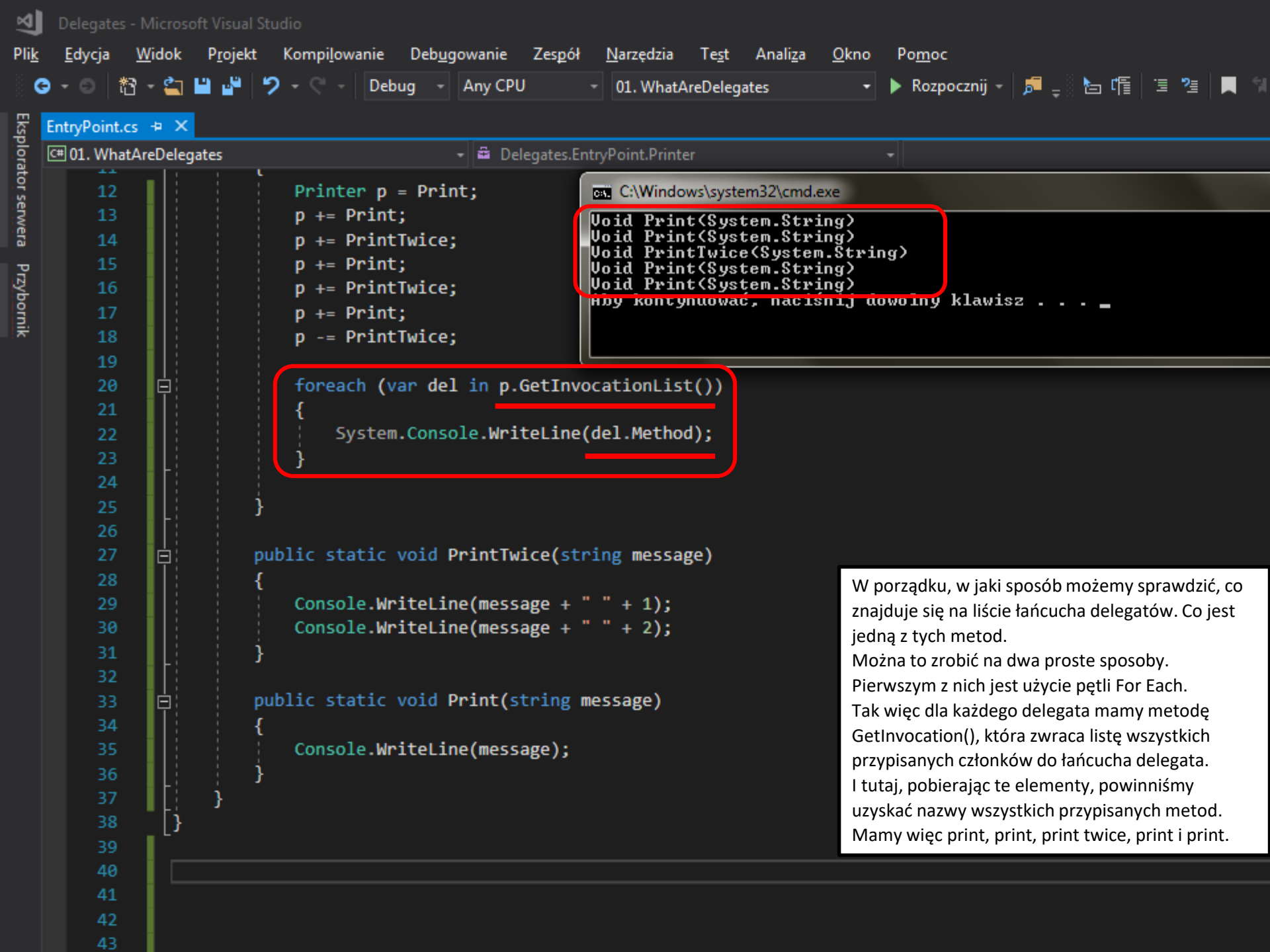
01. WhatAreDelegates | Delegates.EntryPoint | Main()

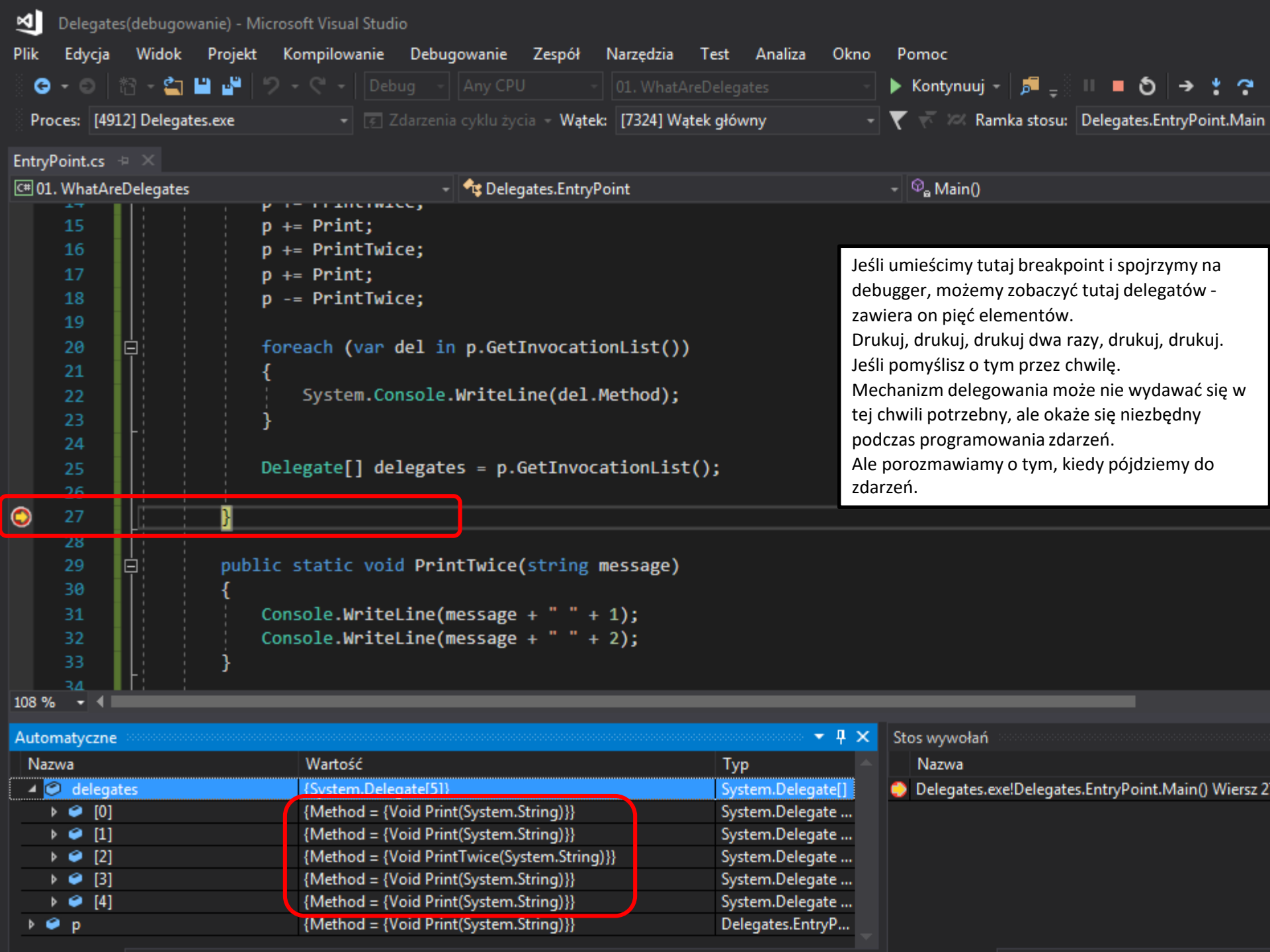
```
1   using System;
2   using System.Collections.Generic;
3
4   namespace Delegates
5   {
6       public class EntryPoint
7       {
8           public delegate void Printer(string message);
9
10          static void Main()
11          {
12              Printer p = Print;
13              p += Print;
14              p += PrintTwice;
15              p += Print;
16              p += PrintTwice;
17              p += Print;
18              p -= PrintTwice;
19
20              p("Message");
21          }
22
23          public static void PrintTwice(string message)
24          {
25              Console.WriteLine(message + " " + 1);
26              Console.WriteLine(message + " " + 2);
27          }
28
29          public static void Print(string message)
30          {
31              Console.WriteLine(message);
32          }
```

C:\Windows\system32\cmd.exe

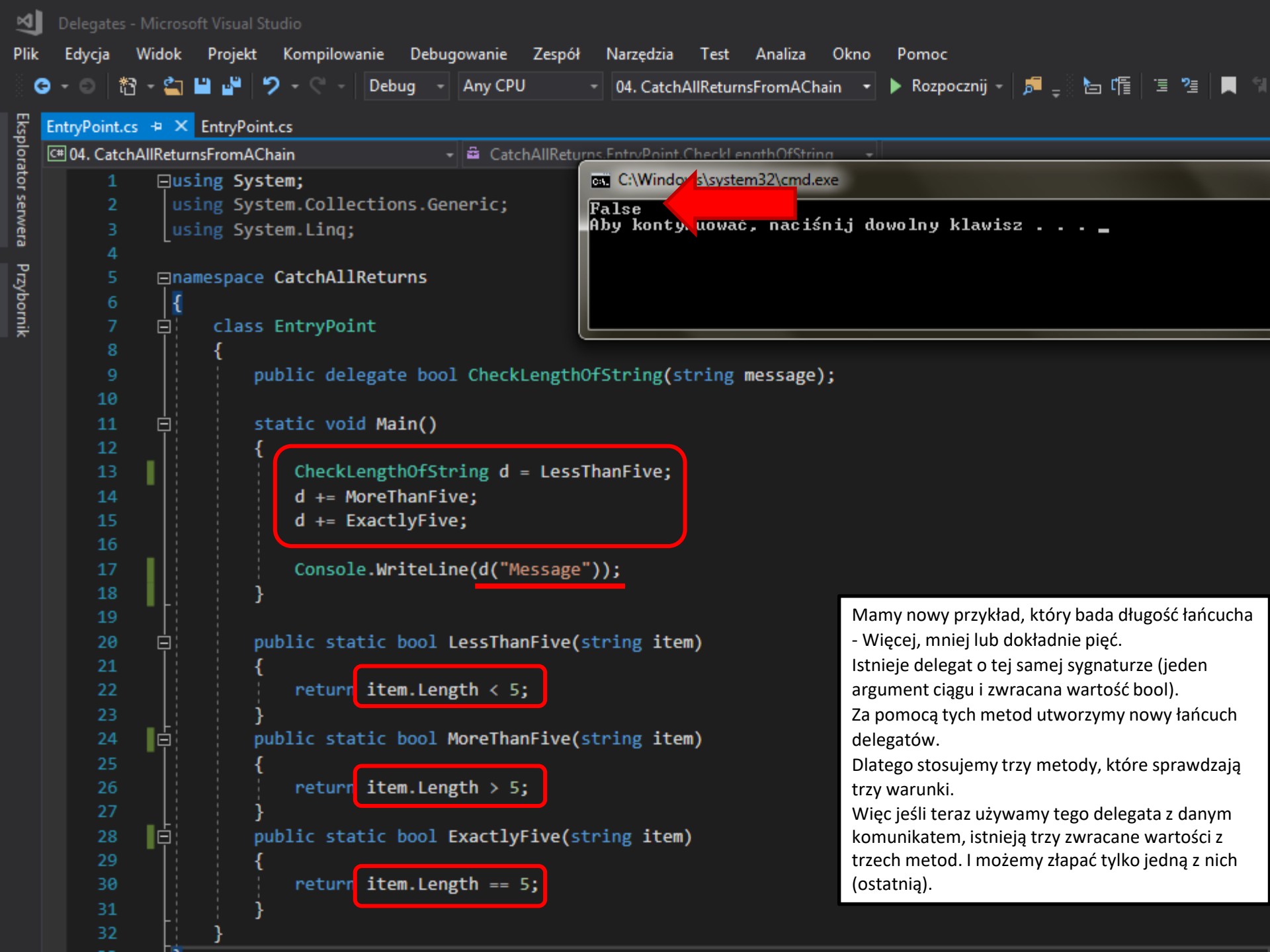
```
Message
Message
Message 1
Message 2
Message
Message
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Możemy również usunąć metody z łańcucha delegata. Piszemy: P minus przypisz PrintTwice. A jeśli ponownie wywołamy p, to PrintTwice dostaniesz tylko raz.
Gdy odejmujemy metody z łańcucha delegatów, usuwamy ostatnie wystąpienie delegata.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Rozpocznij
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint.Printer
Eksplorator serwera
Przybornik
Printer p = Print;
p += Print;
p += PrintTwice;
p += Print;
p += PrintTwice;
p += Print;
p -= PrintTwice;
foreach (var del in p.GetInvocationList())
{
System.Console.WriteLine(del.Method);
}
}
public static void PrintTwice(string message)
{
Console.WriteLine(message + " " + 1);
Console.WriteLine(message + " " + 2);
}
public static void Print(string message)
{
Console.WriteLine(message);
}
}
}
C:\Windows\system32\cmd.exe
Void Print(System.String)
Void Print(System.String)
Void PrintTwice(System.String)
Void Print(System.String)
Void Print(System.String)
Aby kontynuować, naciśnij dowolny klawisz . . .
W porządku, w jaki sposób możemy sprawdzić, co znajduje się na liście łańcucha delegatów. Co jest jedną z tych metod.
Można to zrobić na dwa proste sposoby.
Pierwszym z nich jest użycie pętli For Each.
Tak więc dla każdego delegata mamy metodę GetInvocation(), która zwraca listę wszystkich przypisanych członków do łańcucha delegata.
I tutaj, pobierając te elementy, powinniśmy uzyskać nazwy wszystkich przypisanych metod.
Mamy więc print, print, print twice, print i print.


Delegates(debugowanie) - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. WhatAreDelegates
Kontynuuj
Proces: [4912] Delegates.exe
Zdarzenia cyklu życia
Wątek: [7324] Wątek główny
Ramka stosu: Delegates.EntryPoint.Main
EntryPoint.cs
01. WhatAreDelegates
Delegates.EntryPoint
Main()
15 p += Print;
16 p += PrintTwice;
17 p += Print;
18 p -= PrintTwice;
20 foreach (var del in p.GetInvocationList())
21 {
22 System.Console.WriteLine(del.Method);
23 }
25 Delegate[] delegates = p.GetInvocationList();
27 }
29 public static void PrintTwice(string message)
30 {
31 Console.WriteLine(message + " " + 1);
32 Console.WriteLine(message + " " + 2);
33 }
Jeśli umieścimy tutaj breakpoint i spojrzymy na debugger, możemy zobaczyć tutaj delegatów - zawiera on pięć elementów.
Drukuj, drukuj, drukuj dwa razy, drukuj, drukuj.
Jeśli pomyślisz o tym przez chwilę.
Mechanizm delegowania może nie wydawać się w tej chwili potrzebny, ale okaże się niezbędny podczas programowania zdarzeń.
Ale porozmawiamy o tym, kiedy pójdziemy do zdarzeń.
108 %
Automatyczne
Nazwa
Wartość
Typ
delegates
{System.Delegate[5]}
System.Delegate[]
[0]
{Method = {Void Print(System.String)}}
System.Delegate ...
[1]
{Method = {Void Print(System.String)}}
System.Delegate ...
[2]
{Method = {Void PrintTwice(System.String)}}
System.Delegate ...
[3]
{Method = {Void Print(System.String)}}
System.Delegate ...
[4]
{Method = {Void Print(System.String)}}
System.Delegate ...
p
{Method = {Void Print(System.String)}}
Delegates.EntryP...
Stos wywołań
Nazwa
Delegates.exe!Delegates.EntryPoint.Main() Wiersz 2

Delegates chains with returning methods – catching all returns



Mamy nowy przykład, który bada długość łańcucha - Więcej, mniej lub dokładnie pięć.
Istnieje delegat o tej samej sygnaturze (jeden argument ciągu i zwracana wartość bool).
Za pomocą tych metod utworzymy nowy łańcuch delegatów.
Dlatego stosujemy trzy metody, które sprawdzają trzy warunki.
Więc jeśli teraz używamy tego delegata z danym komunikatem, istnieją trzy zwracane wartości z trzech metod. I możemy złapać tylko jedną z nich (ostatnią).

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 04. CatchAllReturnsFromAChain | Rozpocznij

Eksplorator serwera | Przybornik

EntryPoint.cs | EntryPoint.cs

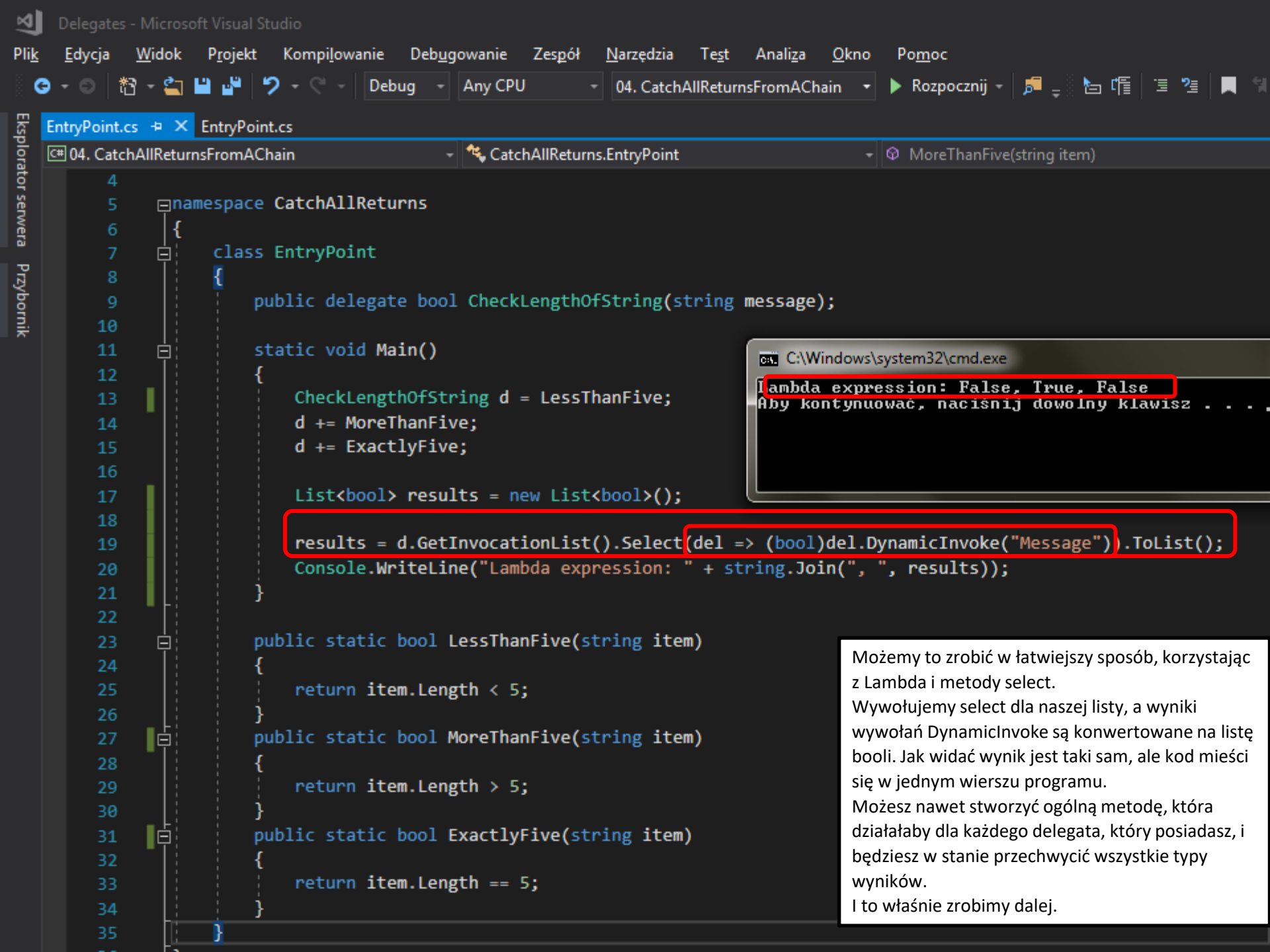
04. CatchAllReturnsFromAChain | CatchAllReturns.EntryPoint.CheckLengthOfString

```
4
5   namespace CatchAllReturns
6   {
7       class EntryPoint
8       {
9           public delegate bool CheckLengthOfString(string message);
10
11          static void Main()
12          {
13              CheckLengthOfString d = LessThanFive;
14              d += MoreThanFive;
15              d += ExactlyFive;
16
17              List<bool> results = new List<bool>();
18              foreach (var del in d.GetInvocationList())
19              {
20                  results.Add((bool)del.DynamicInvoke("Message"));
21              }
22
23              Console.WriteLine("Foreach loop results: " + string.Join(", ", results));
24          }
25
26          public static bool LessThanFive(string item)
27          {
28              return item.Length < 5;
29          }
30          public static bool MoreThanFive(string item)
31          {
32              return item.Length > 5;
33          }
34          public static bool ExactlyFive(string item)
35          {
```

C:\Windows\system32\cmd.exe

```
Foreach loop results: False, True, False
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Jak więc uzyskać wszystkie trzy wartości. Sposobem na to jest indywidualne wykonanie każdego elementu w tym delegacie i utworzenie listy wyników.
Będziemy używać listy wywołań i DynamicInvoke. W tym przykładzie zwracamy booleany do listy wyników. DynamicInvoke zwraca wartość obiektu. Co w zasadzie oznacza, że musimy ręcznie wpisać zmienić typ na boolean.
Wydrukujmy więc również wyniki tej operacji: otrzymujemy wszystkie trzy wyniki z tych metod. Otrzymujemy fałsz, prawda i fałsz.
Możemy to też zrobić w inny sposób i łatwiej.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
04. CatchAllReturnsFromAChain
Rozpocznij
EntryPoint.cs
EntryPoint.cs
04. CatchAllReturnsFromAChain
CatchAllReturns.EntryPoint
MoreThanFive(string item)
Eksplorator serwera
Przybornik
namespace CatchAllReturns
{
class EntryPoint
{
public delegate bool CheckLengthOfString(string message);
static void Main()
{
CheckLengthOfString d = LessThanFive;
d += MoreThanFive;
d += ExactlyFive;
List<bool> results = new List<bool>();
results = d.GetInvocationList().Select(del => (bool)del.DynamicInvoke("Message")).ToList();
Console.WriteLine("Lambda expression: " + string.Join(", ", results));
}
public static bool LessThanFive(string item)
{
return item.Length < 5;
}
public static bool MoreThanFive(string item)
{
return item.Length > 5;
}
public static bool ExactlyFive(string item)
{
return item.Length == 5;
}
}
C:\Windows\system32\cmd.exe
Lambda expression: False, True, False
Aby kontynuować, naciśnij dowolny klawisz . . .
Możemy to zrobić w łatwiejszy sposób, korzystając z Lambda i metody select.
Wywołujemy select dla naszej listy, a wyniki wywołań DynamicInvoke są konwertowane na listę booli. Jak widać wynik jest taki sam, ale kod mieści się w jednym wierszu programu.
Możesz nawet stworzyć ogólną metodę, która działałaby dla każdego delegata, który posiadasz, i będziesz w stanie przechwycić wszystkie typy wyników.
I to właśnie zrobimy dalej.

Generic methods to catch all returns

Możemy przechwycić wszystkie wyniki, które są zwracane przez łańcuch delegatów.
Ale dlaczego musimy za każdym razem przepisywać kod; możemy go wyodrębnić ogólną metodą.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 05. GenericReturnCatcher | Rozpocznij

EntryPoint.cs | EntryPoint.cs | EntryPoint.cs

05. GenericReturnCatcher | GenericReturnCatcher.EntryPoint

```
4
5   namespace GenericReturnCatcher
6   {
7       class EntryPoint
8       {
9           public delegate bool CheckLengthOfString(string message);
10
11          static void Main()
12          {
13              // Chain methods in a Delegate
14              CheckLengthOfString d = LessThanFive;
15              d += MoreThanFive;
16              d += ExactlyFive;
17
18              List<bool> boolResults = CatchAllResults<bool>(d, "Message");
19              Console.WriteLine(string.Join(", ", boolResults));
20          }
21
22          public static List<T> CatchAllResults<T>(Delegate del, object parameter = null)
23          {
24              List<T> result = del.GetInvocationList()
25                                  .Select(d => (T)d.DynamicInvoke(parameter))
26                                  .ToList();
27              return result;
28          }
29
30          public static bool LessThanFive(string item)...
34          public static bool MoreThanFive(string item)...
38          public static bool ExactlyFive(string item)...
42      }
43  }
44
```

Stwórzmy więc nową metodę public static.
Zwrócimy listę T, ponieważ za każdym razem będziemy zwracać inny typ i wywołajmy je i złapmy wyniki.
Tak więc bierzemy delegata jako argument wejściowy, a także bierzemy parametr obiektowy i sprawimy, że będzie on równy null, ponieważ tworzymy go jako opcjonalny, ponieważ nie wszystkie metody będą miały jakieś wejście.

C:\Windows\system32\cmd.exe

```
False, True, False
Aby kontynuować, naciśnij dowolny klawisz . . .
```

To jest opcjonalne.
I tutaj musimy stworzyć nową listę, do której przypiszemy wyniki. I metoda select z argumentem lambda, aby uzyskać tylko wyniki z łańcucha metod.
Teraz mamy jeden drobny problem - musimy zmienić typ na T i musimy to zwrócić. Mamy wynik.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 05. GenericReturnCatcher | Rozpocznij

EntryPoint.cs* | EntryPoint.cs | EntryPoint.cs

05. GenericReturnCatcher | GenericReturnCatcher.EntryPoint | LessThanFive(string item)

```
9   public delegate bool CheckLengthOfString(string message);
10  public delegate int GetLength(string message);
11
12  static void Main()
13  {
14      // Chain methods in a Delegate
15      CheckLengthOfString d = LessThanFive;
16      d += MoreThanFive;
17      d += ExactlyFive;
18
19      List<bool> boolResults = CatchAllResults<bool>(d, "Message");
20      Console.WriteLine(string.Join(", ", boolResults));
21
22      // Second Medthod Chain
23      GetLength p = x => x.Length;
24      p += x => x.Length + 1;
25      p += x => x.Length + 2;
26
27      List<int> lengths = CatchAllResults<int>(p, "asd");
28      Console.WriteLine(string.Join(", ", lengths));
29  }
30
31  public static List<T> CatchAllResults<T>(Delegate del, object parameter = null)
32  {
33      List<T> result = del.GetInvocationList()
34                          .Select(d => (T)d.DynamicInvoke(parameter))
35                          .ToList();
36      return result;
37  }
38
39  public static bool LessThanFive(string item)
43  public static bool MoreThanFive(string item)
```

C:\Windows\system32\cmd.exe

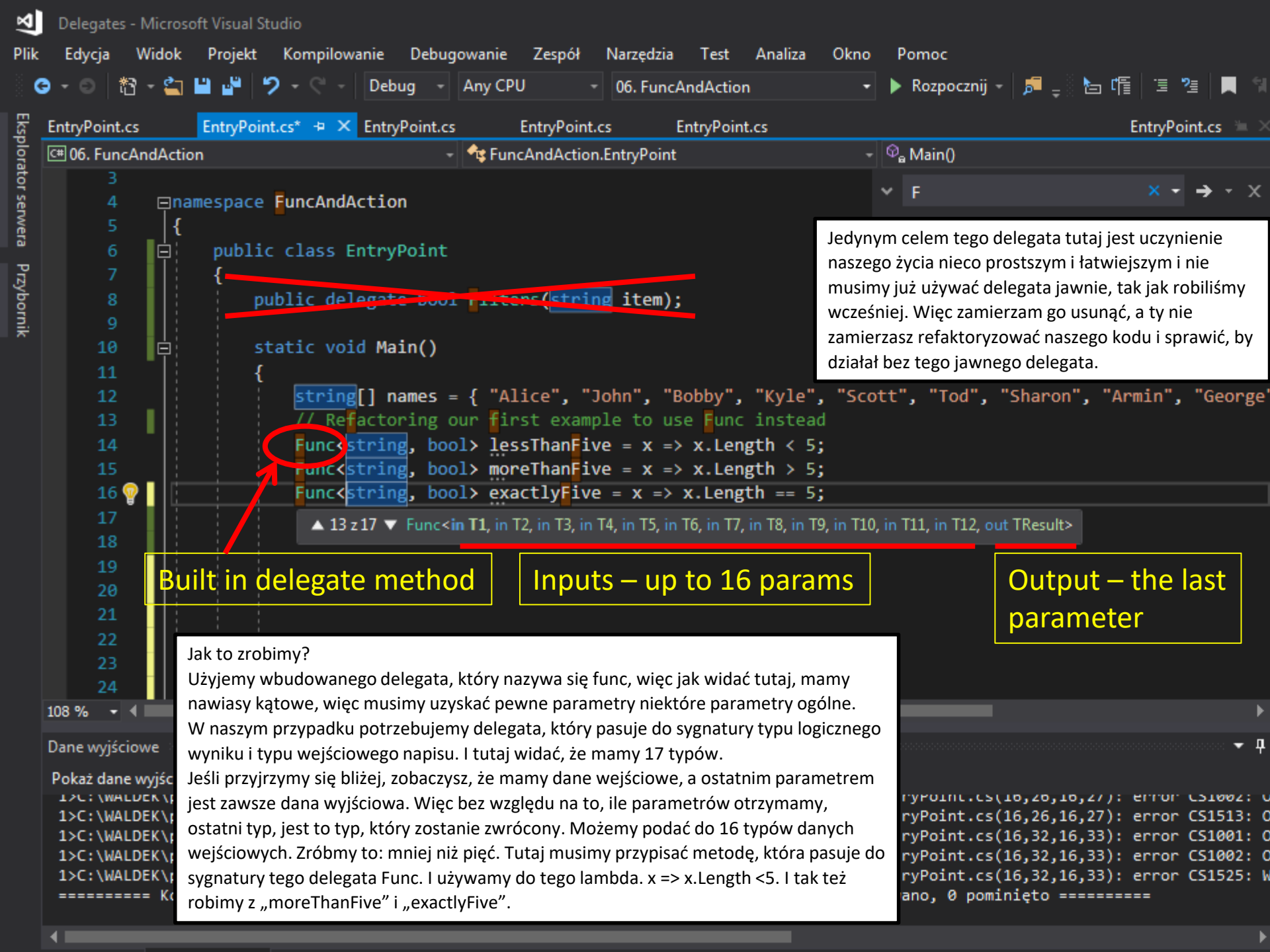
```
False, True, False
3, 4, 5
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Teraz, aby potwierdzić, że działa dla różnych typów, stwórzmy nowego delegata. Dostaje długości. Więc zmieniamy wyniki na liczby całkowite. Tym razem użyjemy wyrażeń lambda, ponieważ jest to łatwiejsze i szybsze. Przypisujemy trzy metody do łańcucha: długość, długość + 1, długość + 2.

Mamy więc dwie metody, które przyjmą długość argumentu wejściowego, którym jest komunikat. W pierwszym łańcuchu delegatów zwracamy wartości logiczne, a w drugich liczby całkowite. Tak więc w każdym przypadku łapiemy wszystkie wyniki i możemy wyświetlać wyniki na konsoli. Udało nam się również uogólnić tę metodę.

Func and *Action* delegates

Dobrze, pokażę ci jeden nowy typ.
Nie jest to dokładnie typ delegata, ale delegowanie..


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
06. FuncAndAction
Rozpocznij
EntryPoint.cs
EntryPoint.cs*
EntryPoint.cs
EntryPoint.cs
EntryPoint.cs
EntryPoint.cs
06. FuncAndAction
FuncAndAction.EntryPoint
Main()
namespace FuncAndAction
{
public class EntryPoint
{
public delegate bool Filters(string item);
static void Main()
{
string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
// Refactoring our first example to use Func instead
Func<string, bool> lessThanFive = x => x.Length < 5;
Func<string, bool> moreThanFive = x => x.Length > 5;
Func<string, bool> exactlyFive = x => x.Length == 5;
▲ 13 z 17 ▼ Func<in T1, in T2, in T3, in T4, in T5, in T6, in T7, in T8, in T9, in T10, in T11, in T12, out TResult>
Jedynym celem tego delegata tutaj jest uczynienie naszego życia nieco prostszym i łatwiejszym i nie musimy już używać delegata jawnie, tak jak robiliśmy wcześniej. Więc zamierzam go usunąć, a ty nie zamierzasz refaktoryzować naszego kodu i sprawić, by działał bez tego jawnego delegata.
Built in delegate method
Inputs – up to 16 params
Output – the last parameter
Jak to zrobimy?
Użyjemy wbudowanego delegata, który nazywa się func, więc jak widać tutaj, mamy nawiasy kątowe, więc musimy uzyskać pewne parametry niektóre parametry ogólne.
W naszym przypadku potrzebujemy delegata, który pasuje do sygnatury typu logicznego wyniku i typu wejściowego napisu. I tutaj widać, że mamy 17 typów.
Jeśli przyjrzymy się bliżej, zobaczysz, że mamy dane wejściowe, a ostatnim parametrem jest zawsze dana wyjściowa. Więc bez względu na to, ile parametrów otrzymamy, ostatni typ, jest to typ, który zostanie zwrócony. Możemy podać do 16 typów danych wejściowych. Zróbmy to: mniej niż pięć. Tutaj musimy przypisać metodę, która pasuje do sygnatury tego delegata Func. I używamy do tego lambda. x => x.Length <5. I tak też robimy z „moreThanFive” i „exactlyFive”.
Dane wyjściowe
Pokaż dane wyjśc
1>C:\WALDEK\p
1>C:\WALDEK\p
1>C:\WALDEK\p
1>C:\WALDEK\p
1>C:\WALDEK\p
ryPoint.cs(16,26,16,27): error CS1513:
ryPoint.cs(16,32,16,33): error CS1001:
ryPoint.cs(16,32,16,33): error CS1002:
ryPoint.cs(16,32,16,33): error CS1525:
ano, 0 pominięto ==========

Delegates - Microsoft Visual Studio

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

Debug Any CPU

EntryPoint.cs EntryPoint.cs* EntryPoint.cs EntryPoint.

06. FuncAndAction FuncAndAction

Nawet bez utworzenia przez nas nowego delegata, nadal będzie działać po prostu dlatego, że te metody są zgodne z sygaturą delegata func.
Użyjmy różnych podejść.
Zamiast podać mu metody, które już widzieliśmy, że działają, użyjemy wyrażenia.

```
8   static void Main()
9   {
10      string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
11      // Refactoring our first example to use Func instead
12      Func<string, bool> lessThanFive = x => x.Length < 5;
13      Func<string, bool> moreThanFive = x => x.Length > 5;
14      Func<string, bool> exactlyFive = x => x.Length == 5;
15
16
17
18
19
20      List<string> namesLessThanFiveChars = ExtractStrings(names, lessThanFive);
21      List<string> namesMoreThanFiveChars = ExtractStrings(names, moreThanFive);
22      List<string> namesExactlyFiveChars = ExtractStrings(names, exactlyFive);
23
24      Console.WriteLine("Names less than five chars: " + string.Join(", ", namesLessThanFiveChars));
25      Console.WriteLine("Names more than five chars: " + string.Join(", ", namesMoreThanFiveChars));
26      Console.WriteLine("Names exactly five chars: " + string.Join(", ", namesExactlyFiveChars));
27  }
28
29  public static List<string> ExtractStrings(string[] array, Func<string, bool> filter)
30  {
31      List<string> result = new List<string>();
32      for (int i = 0; i < array.Length; i++)
33      {
34          if (filter(array[i]))
35          {
36              result.Add(array[i]);
37          }
38      }
39      return result;
```

Type - delegate

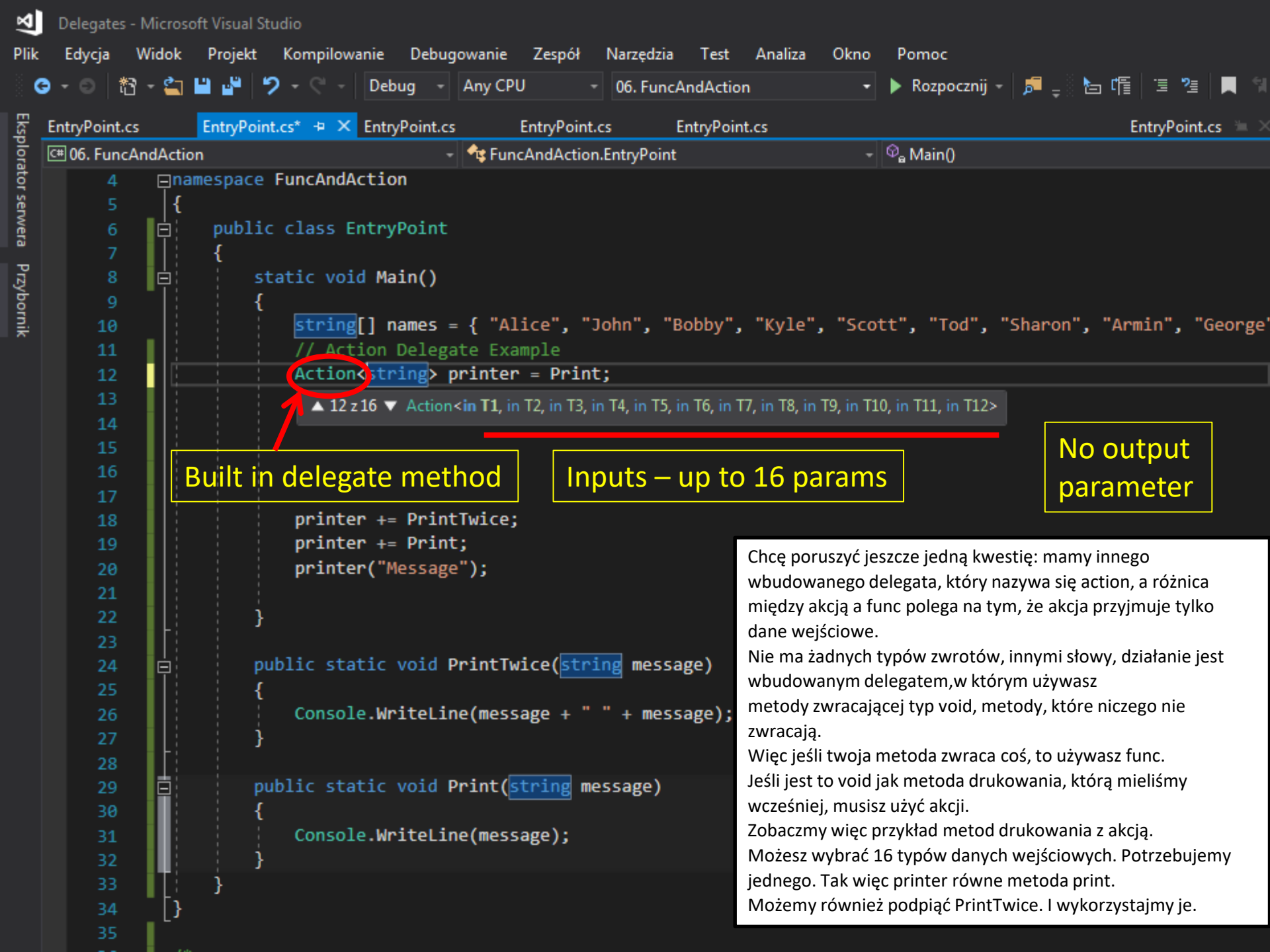
Variable – method name

Value – body of method

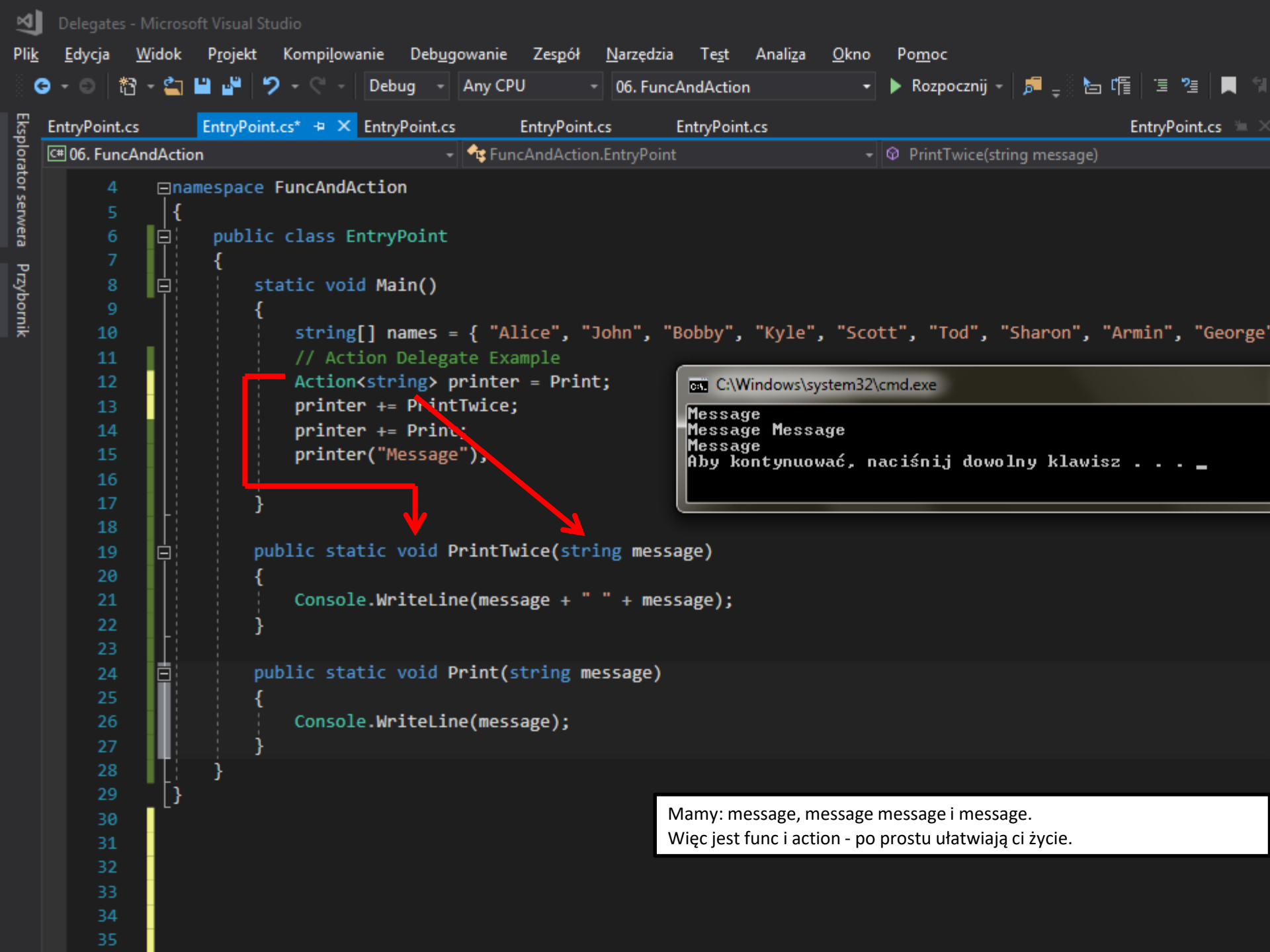
C:\Windows\system32\cmd.exe

```
Names less than five chars: John, Kyle, Tod
Names more than five chars: Sharon, George
Names exactly five chars: Alice, Bobby, Scott, Arm
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Zatem X jest wejściowym i zamiast używać tej metody, użyjemy tego delegata który tutaj stworzyliśmy. Nadal działa.
Możemy zrobić to samo dla więcej niż pięciu.
Dokładnie pięć, ale już widzieliśmy to wystarczająco dużo razy.

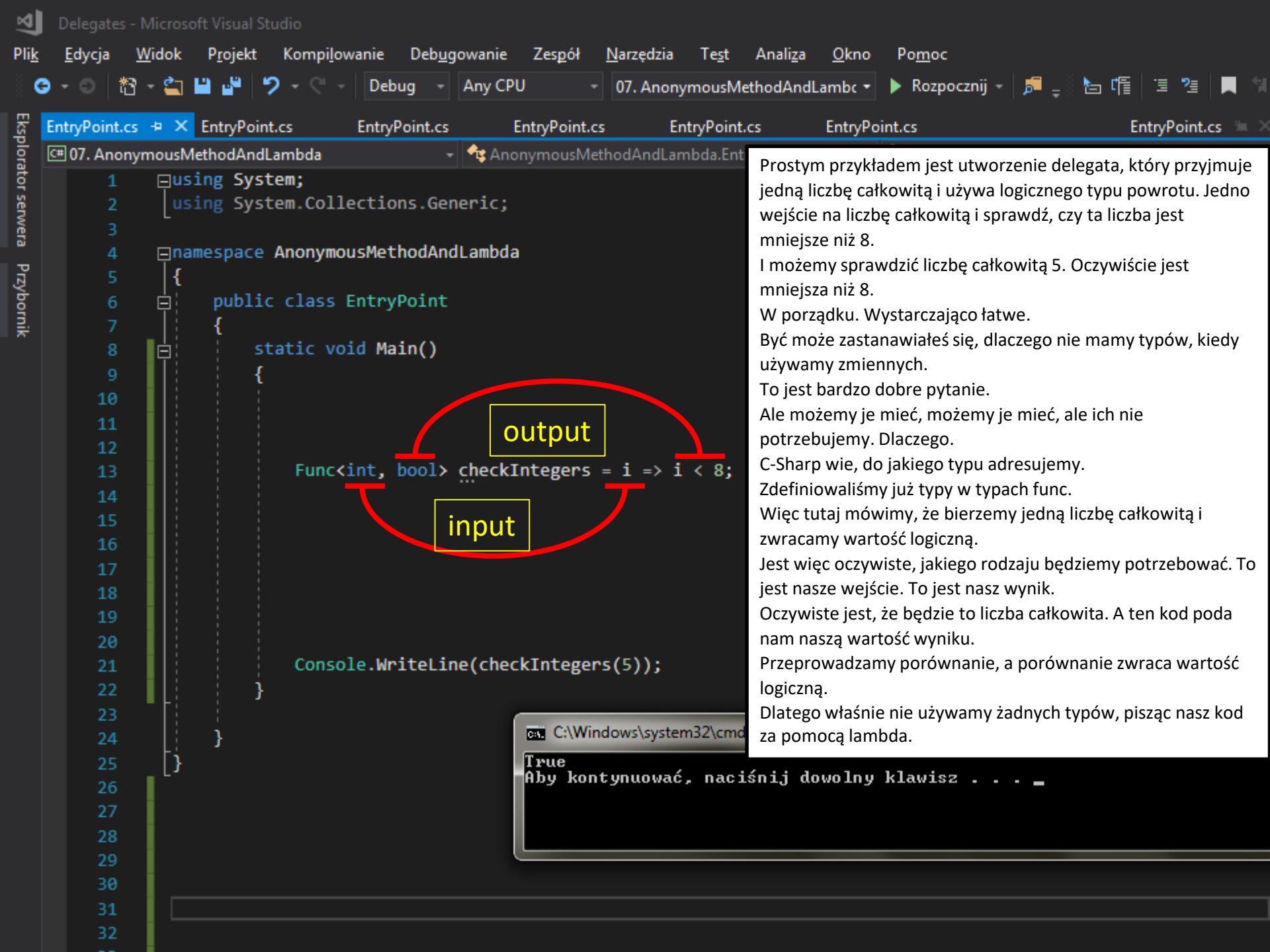


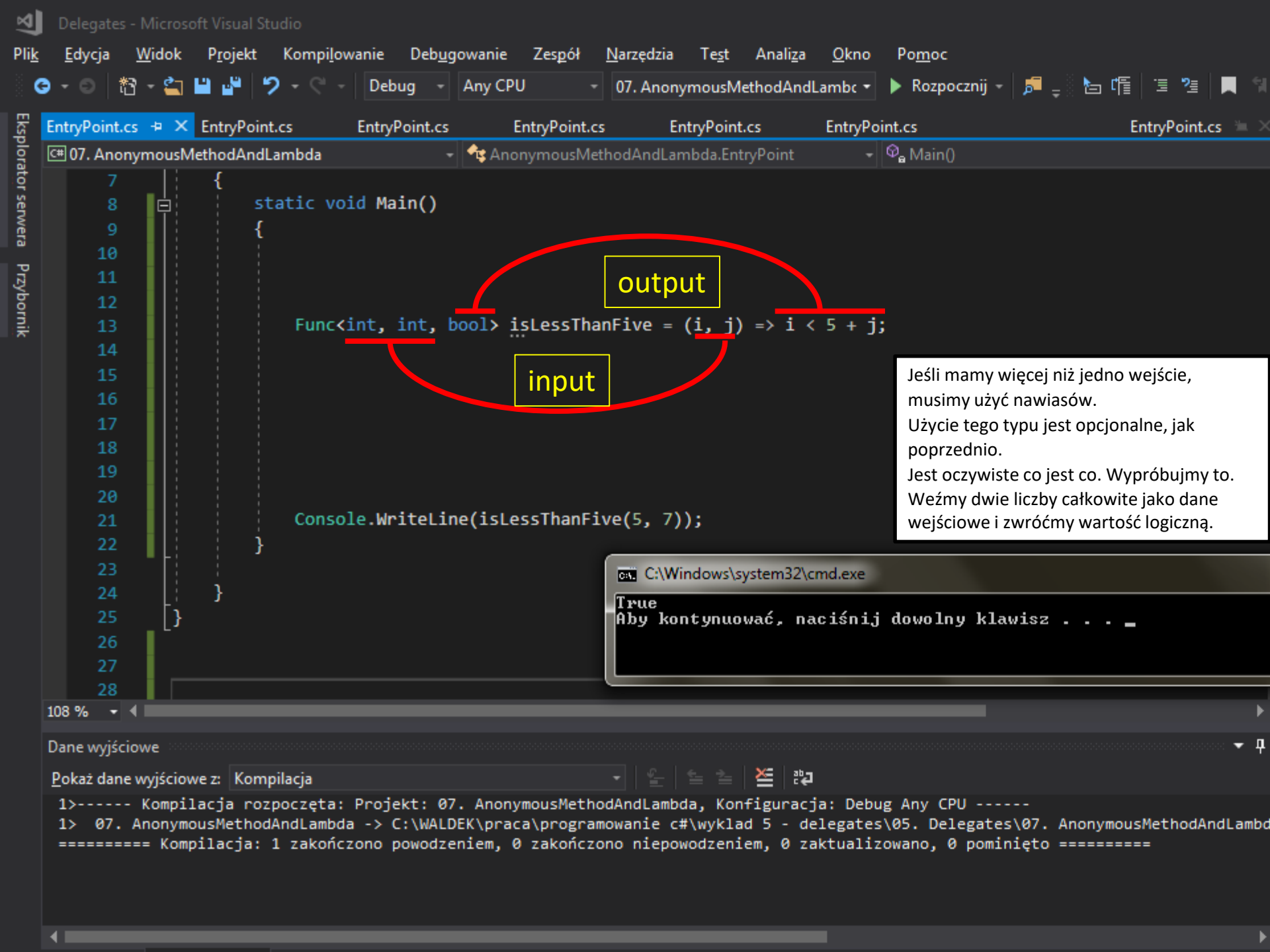
Chcę poruszyć jeszcze jedną kwestię: mamy innego wbudowanego delegata, który nazywa się action, a różnica między akcją a func polega na tym, że akcja przyjmuje tylko dane wejściowe.
Nie ma żadnych typów zwrotów, innymi słowy, działanie jest wbudowanym delegatem,w którym używasz metody zwracającej typ void, metody, które niczego nie zwracają.
Więc jeśli twoja metoda zwraca coś, to używasz func.
Jeśli jest to void jak metoda drukowania, którą mieliśmy wcześniej, musisz użyć akcji.
Zobaczmy więc przykład metod drukowania z akcją.
Możesz wybrać 16 typów danych wejściowych. Potrzebujemy jednego. Tak więc printer równe metoda print.
Możemy również podpiąć PrintTwice. I wykorzystajmy je.

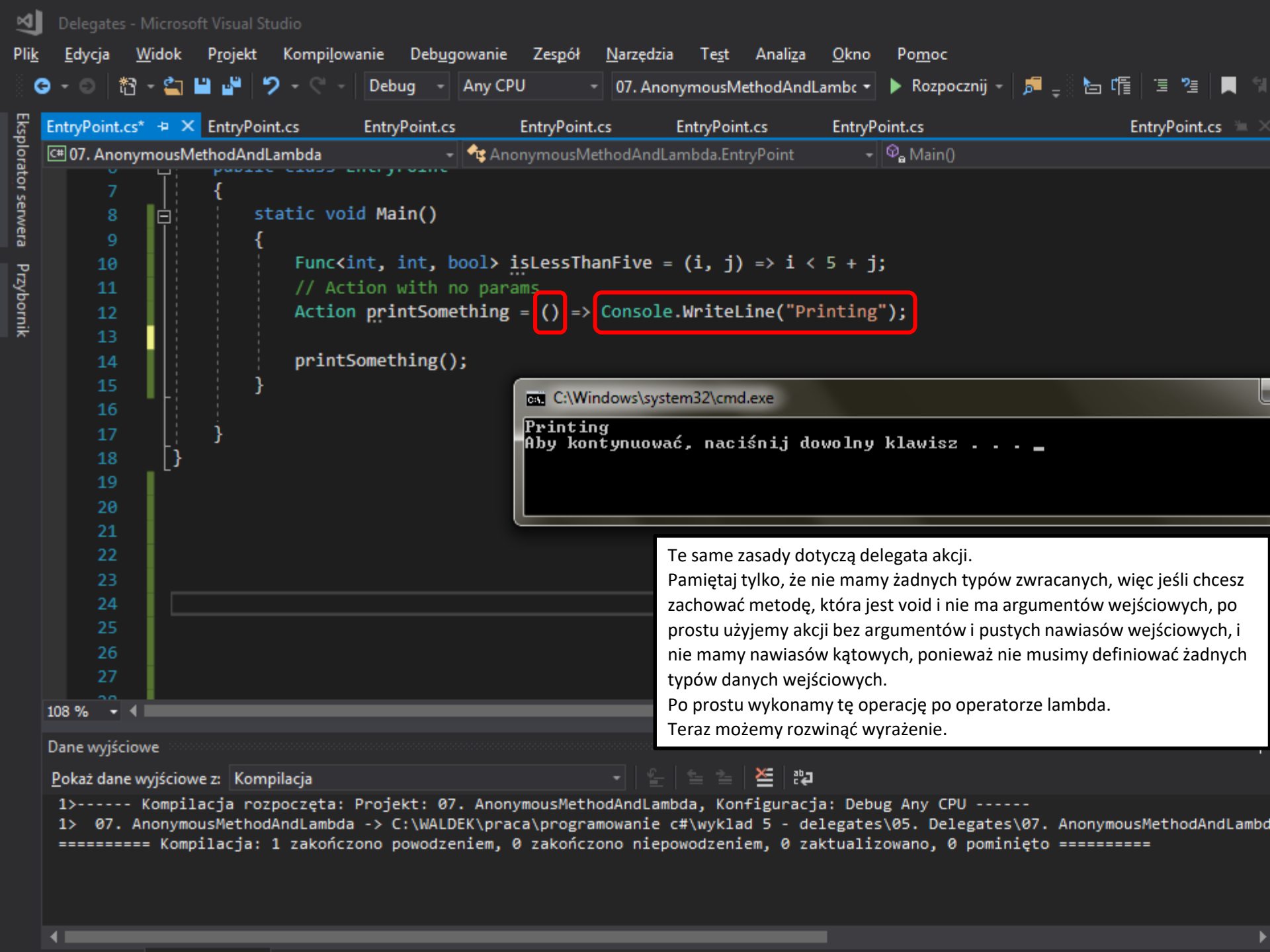

Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
06. FuncAndAction
Rozpocznij
EntryPoint.cs
EntryPoint.cs*
EntryPoint.cs
EntryPoint.cs
EntryPoint.cs
EntryPoint.cs
06. FuncAndAction
FuncAndAction.EntryPoint
PrintTwice(string message)
Eksplorator serwera
Przybornik
4 namespace FuncAndAction
5 {
6 public class EntryPoint
7 {
8 static void Main()
9 {
10 string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
11 // Action Delegate Example
12 Action<string> printer = Print;
13 printer += PrintTwice;
14 printer += Print;
15 printer("Message");
16
17 }
18
19 public static void PrintTwice(string message)
20 {
21 Console.WriteLine(message + " " + message);
22 }
23
24 public static void Print(string message)
25 {
26 Console.WriteLine(message);
27 }
28 }
29 }
30
31
32
33
34
35
C:\Windows\system32\cmd.exe
Message
Message Message
Message
Aby kontynuować, naciśnij dowolny klawisz . . .
Mamy: message, message message i message.
Więc jest func i action - po prostu ułatwiają ci życie.

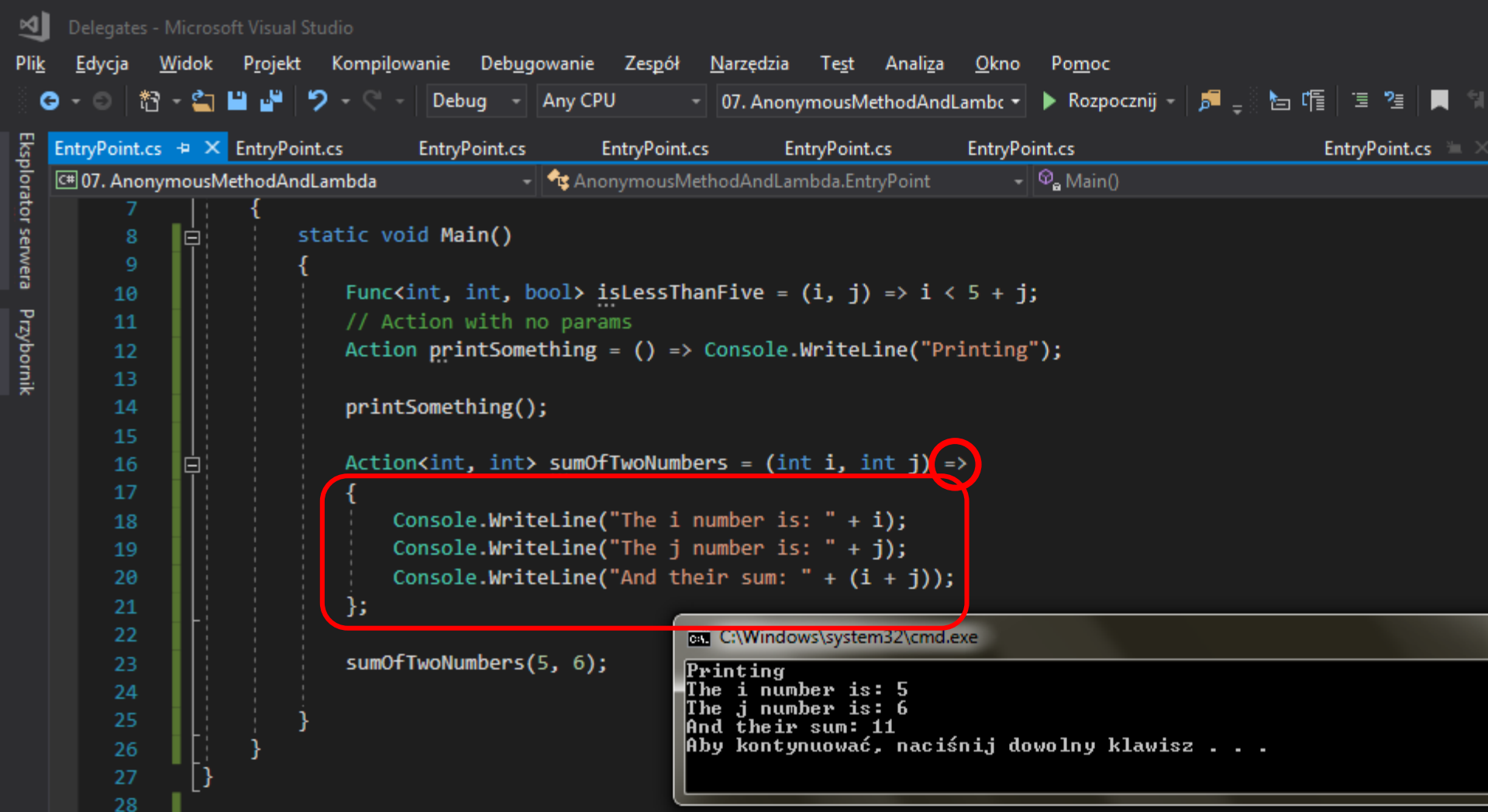
Anonymous methods and lambda expressions

Być może słyszałeś termin metody anonimowe. O tym mówimy. Można tego używać z delegatami func i action. Zobaczmy więc, jak to działa.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
07. AnonymousMethodAndLambc
Rozpocznij
EntryPoint.cs
07. AnonymousMethodAndLambda
AnonymousMethodAndLambda.Ent
Eksplorator serwera
Przybornik
using System;
using System.Collections.Generic;
namespace AnonymousMethodAndLambda
{
public class EntryPoint
{
static void Main()
{
Func<int, bool> checkIntegers = i => i < 8;
Console.WriteLine(checkIntegers(5));
}
}
}
output
input
C:\Windows\system32\cmd
True
Aby kontynuować, naciśnij dowolny klawisz . . .
Prostym przykładem jest utworzenie delegata, który przyjmuje jedną liczbę całkowitą i używa logicznego typu powrotu. Jedno wejście na liczbę całkowitą i sprawdź, czy ta liczba jest mniejsze niż 8.
I możemy sprawdzić liczbę całkowitą 5. Oczywiście jest mniejsza niż 8.
W porządku. Wystarczająco łatwe.
Być może zastanawiałeś się, dlaczego nie mamy typów, kiedy używamy zmiennych.
To jest bardzo dobre pytanie.
Ale możemy je mieć, możemy je mieć, ale ich nie potrzebujemy. Dlaczego.
C-Sharp wie, do jakiego typu adresujemy.
Zdefiniowaliśmy już typy w typach func.
Więc tutaj mówimy, że bierzemy jedną liczbę całkowitą i zwracamy wartość logiczną.
Jest więc oczywiste, jakiego rodzaju będziemy potrzebować. To jest nasze wejście. To jest nasz wynik.
Oczywiste jest, że będzie to liczba całkowita. A ten kod poda nam naszą wartość wyniku.
Przeprowadzamy porównanie, a porównanie zwraca wartość logiczną.
Dlatego właśnie nie używamy żadnych typów, pisząc nasz kod za pomocą lambda.


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
07. AnonymousMethodAndLambc
Rozpocznij
EntryPoint.cs
07. AnonymousMethodAndLambda
AnonymousMethodAndLambda.EntryPoint
Main()
{
static void Main()
{
Func<int, int, bool> isLessThanFive = (i, j) => i < 5 + j;
output
input
Console.WriteLine(isLessThanFive(5, 7));
}
}
}
Jeśli mamy więcej niż jedno wejście, musimy użyć nawiasów.
Użycie tego typu jest opcjonalne, jak poprzednio.
Jest oczywiste co jest co. Wypróbujmy to.
Weźmy dwie liczby całkowite jako dane wejściowe i zwróćmy wartość logiczną.
C:\Windows\system32\cmd.exe
True
Aby kontynuować, naciśnij dowolny klawisz . . .
108 %
Dane wyjściowe
Pokaż dane wyjściowe z: Kompilacja
1>------ Kompilacja rozpoczęta: Projekt: 07. AnonymousMethodAndLambda, Konfiguracja: Debug Any CPU ------
1> 07. AnonymousMethodAndLambda -> C:\WALDEK\praca\programowanie c#\wyklad 5 - delegates\05. Delegates\07. AnonymousMethodAndLambd
========== Kompilacja: 1 zakończono powodzeniem, 0 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========


Delegates - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
07. AnonymousMethodAndLambc
Rozpocznij
EntryPoint.cs*
EntryPoint.cs
07. AnonymousMethodAndLambda
AnonymousMethodAndLambda.EntryPoint
Main()
{
static void Main()
{
Func<int, int, bool> isLessThanFive = (i, j) => i < 5 + j;
// Action with no params
Action printSomething = () => Console.WriteLine("Printing");
printSomething();
}
}
C:\Windows\system32\cmd.exe
Printing
Aby kontynuować, naciśnij dowolny klawisz . . .
Te same zasady dotyczą delegata akcji.
Pamiętaj tylko, że nie mamy żadnych typów zwracanych, więc jeśli chcesz zachować metodę, która jest void i nie ma argumentów wejściowych, po prostu użyjemy akcji bez argumentów i pustych nawiasów wejściowych, i nie mamy nawiasów kątowych, ponieważ nie musimy definiować żadnych typów danych wejściowych.
Po prostu wykonamy tę operację po operatorze lambda.
Teraz możemy rozwinąć wyrażenie.
108 %
Dane wyjściowe
Pokaż dane wyjściowe z: Kompilacja
1>------ Kompilacja rozpoczęta: Projekt: 07. AnonymousMethodAndLambda, Konfiguracja: Debug Any CPU ------
1> 07. AnonymousMethodAndLambda -> C:\WALDEK\praca\programowanie c#\wyklad 5 - delegates\05. Delegates\07. AnonymousMethodAndLambd
========== Kompilacja: 1 zakończono powodzeniem, 0 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========



Do tej pory korzystaliśmy z metod jednowierszowych, ale możemy również zastosować nieco bardziej złożoną metodę.
Stwórzmy nową metodę akcji, która przyjmuje dwie liczby całkowite jako dane wejściowe i wyświetli trzy rzeczy: pierwszy argument, drugi argument i sumę.
Użycie typu jest opcjonalne.
Tutaj możesz napisać tyle kodu, ile chcesz.
Jest to treść anonimowej metody, którą zamierzamy stworzyć.
Wypróbujmy to. W porządku. W ten sposób możesz pisać wbudowane anonimowe metody.
Oczywiście, jeśli potrzebujesz więcej niż kilku wierszy kodu, lepiej byłoby stworzyć zupełnie nową metodę.

More complex anonymous methods

W porządku, więc będziemy pracować nad nieco bardziej złożoną anonimową metodą. Przykład, który zrobimy, to wykonać filtrowanie anonimową metodą.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 07. AnonymousMethodAndLambc | Rozpocznij

EntryPoint.cs* | EntryPoint.cs | EntryPoint.cs | EntryPoint.cs | EntryPoint.cs | EntryPoint.cs | EntryPoint.cs

07. AnonymousMethodAndLambda | AnonymousMethodAndLambda.EntryPoint | Main()

```
7       {
8           static void Main()
9           {
10              string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
11
12
13
14
15
16
17              Func<string[], Func<string, bool>, List<string>> extractStrings = (arr, filter) =>
18              {
19                  List<string> result = new List<string>();
20
21                  // condition
22
23                  return result;
24              };
25          }
26      }
27  }
28
29
30
31
32
33
34
35
36
37
38
```

Więc stworzymy delegata func.
Bierzemy tablicę ciągów jako pierwsze wejście, a delegata (który będzie naszym filtrem) jako drugi. Trzeci parametr (tablica ciągów) jest wyjściem. Drugi parametr, który jest delegatem, sprawdzi określony ciąg znaków i zwróci wartość logiczną.
Mamy więc typ wejściowy, który jest kolejną funkcją string i bool.

Delegates - Microsoft Visual Studio

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

Debug | Any CPU | 07. AnonymousMethodAndLambc | Rozpocznij

EntryPoint.cs | EntryPoint.cs | EntryPoint.cs

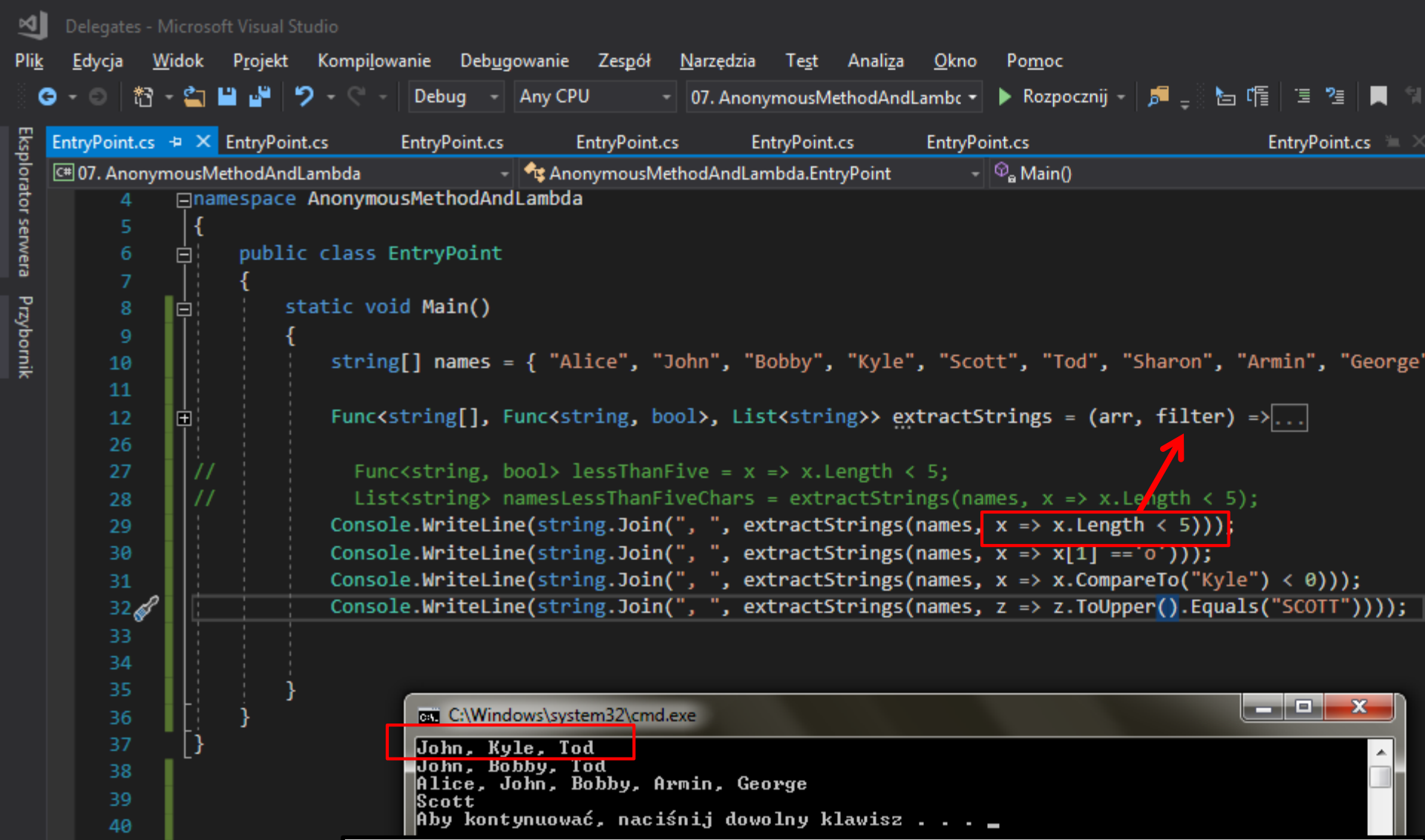
07. AnonymousMethodAndLambda

Delegat otrzymuje dane wejściowe, a my zwracamy listę napisów.
I tutaj zaczynamy tworzyć naszą anonimową metodę.
Będziemy również mieć listę napisów, które będą naszym wynikiem.
Wszystko, co musisz zrobić, to po prostu filtrować.
Potrzebujemy więc tylko pętli For Each dla tablicy.
Filtr z określonym warunkiem zostanie zastosowany do każdego elementu tablicy. Jeśli warunek zostanie spełniony, element zostanie przeniesiony do tablicy wyjściowej.

```
4   namespace AnonymousMethodAndLambda
5   {
6       public class EntryPoint
7       {
8           static void Main()
9           {
10              string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George"
11
12              Func<string[], Func<string, bool>, List<string>> extractStrings = (arr, filter) =>
13              {
14                  List<string> result = new List<string>();
15
16                  for (int i = 0; i < arr.Length; i++)
17                  {
18                      if (filter(arr[i]))
19                      {
20                          result.Add(arr[i]);
21                      }
22                  }
23
24                  return result;
25              };
26
27              Func<string, bool> lessThanFive = x => x.Length < 5;
28              List<string> namesLessThanFiveChars = extractStrings(names, lessThanFive);
29              Console.WriteLine(string.Join(", ", namesLessThanFiveChars));
30
31
32          }
33      }
34  }
35
```

C:\Windows\system32\cmd.exe

```
John, Kyle, Tod
Aby kontynuować, naciśnij dowolny klawisz . . .
```



Dobrze, skorzystajmy z tego. Stworzymy listę napisów, które będą naszym wyjściem. Potrzebujemy również filtra, który podamy jako drugi parametr wejściowy. Wybierzmy napisy, których długości są mniejsze niż 5. Zapisujemy filtr jako drugi parametr za pomocą wyrażenia lambda. X jest wejściem filtru, który jest ciągiem. Filtr zwraca wartość logiczną, która jest naszym wyjściem. W porządku. Jak widać działa. Ponieważ nasze wyrażenie filtru jest delegatem, możemy przez nią przepuszczać różne fragmenty kodu. W drugim przykładzie sprawdzamy, czy litera „o" znajduje się w określonej pozycji ciągu. W poniższych przykładach wywołujemy metody, które porównują i konwertują ciągi. Możemy również budować bardziej zaawansowane filtry. Jeśli jednak musimy zrobić coś naprawdę skomplikowanego, lepiej jest stworzyć osobną metodę.