Events – Publisher and Subscribers

Jeśli chcesz, aby dana akcja została aktywowana, jeśli coś wydarzy się w innym obiekcie, to jest to miejsce na "zdarzenia". Pozwalają nam pisać oba kody osobno i unikać sprzężeń w kodzie pomiędzy obiektami a jednak obiekty będą ze sobą współpracować.

What is event?

Zdarzenie jest uruchamiane i wykonuje inny kod. Jeśli spojrzysz na ten przykład, ta broń strzela pociskami, a żołnierze się ukrywają. Reagują na wydarzenie. Obiekt, który przechowuje zdarzenie lub ten, który je uruchamia, nazywa się w tym przykładzie publisher, natomiast metody dodane do tego wydarzenia nazywane są subskrybentami. Wyobraźcie sobie więc, że wszyscy żołnierze, którzy są ostrzeliwani tą bronią, są różnymi metodami osobne metody i wszystkie z nich są subskrybentami tego wydarzenia.

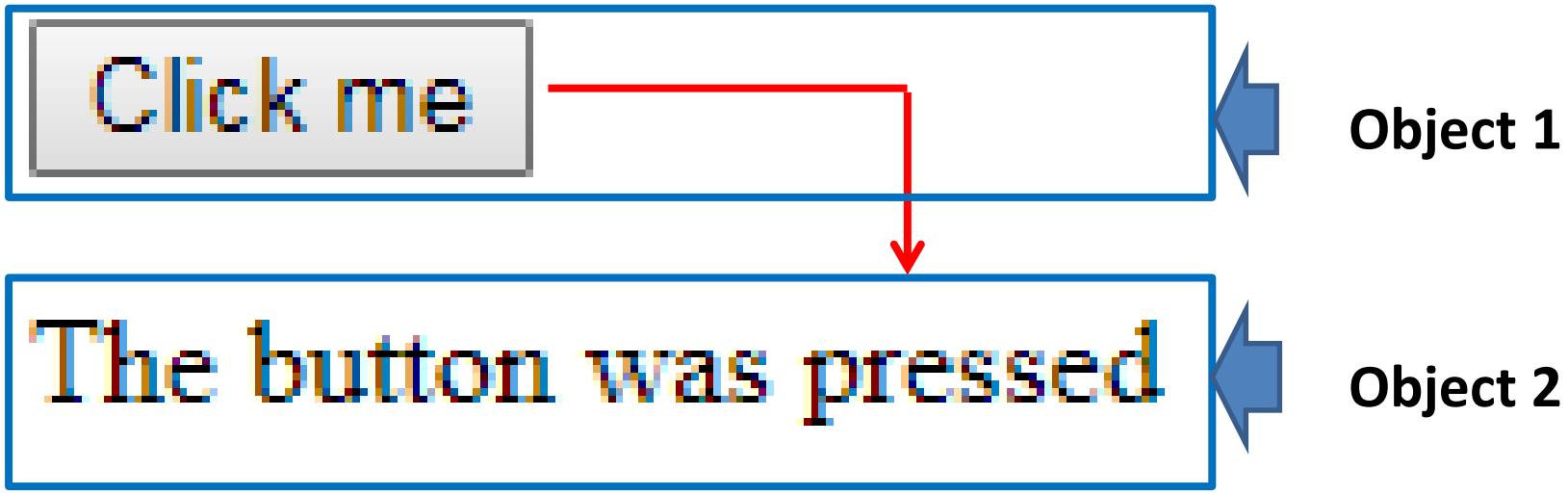
Publisher

Object which holds the event

Action

Subscribers

Methods that executes when event is fired



Kiedy zaczynasz pracę z Windows Forms, najprostszym przykładem będzie kliknięcie przycisku i coś musi się wydarzyć. Kliknięcie aktywuje zdarzenie, które z kolei aktywowuje subskrybowaną metodę.

The anatomy of an Event

Anatomia zdarzenia

Anatomy of an Event

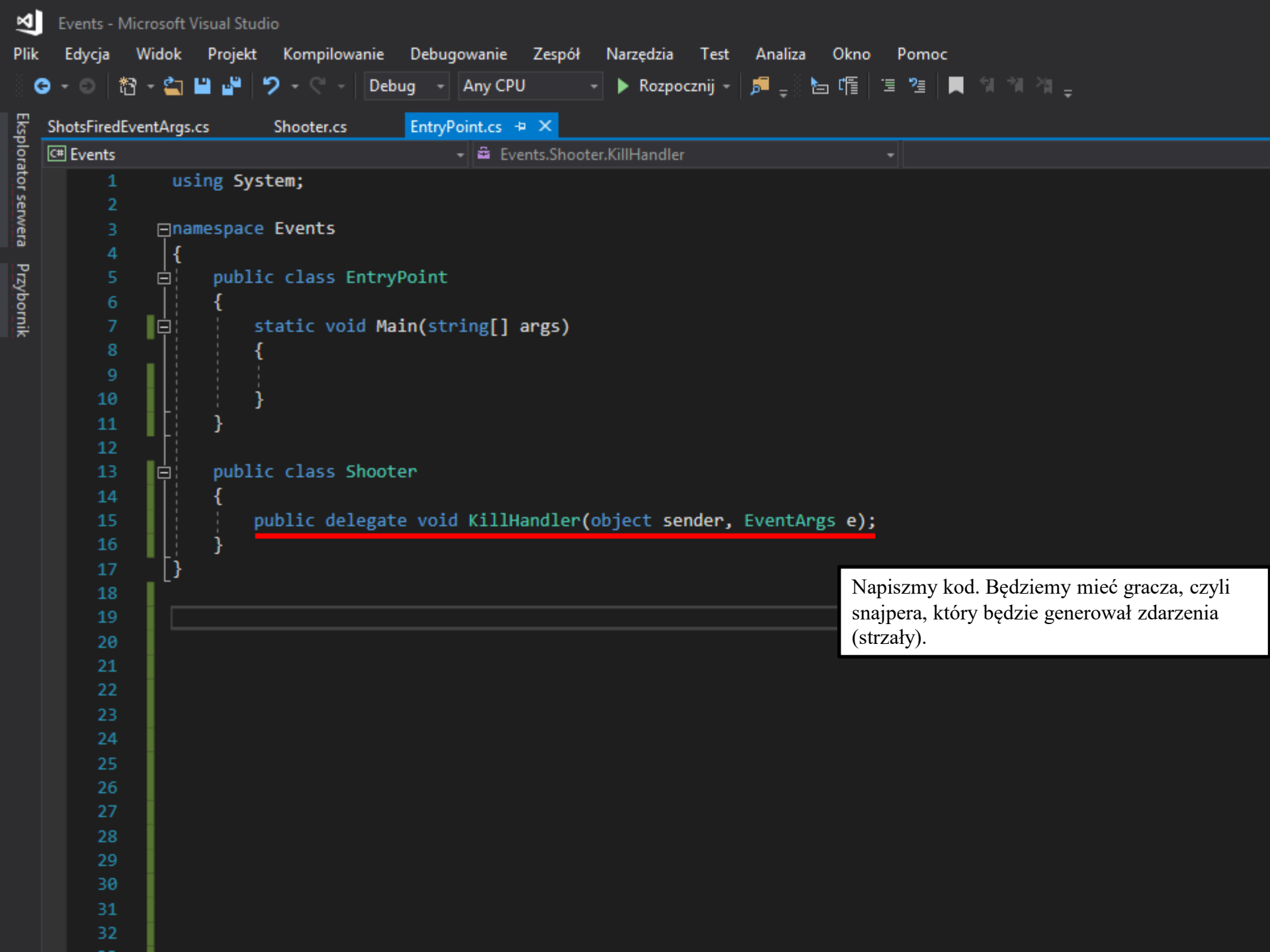
Publisher:

- **Delegate – matching the event signature**
- **Event – of the type of delegate**
- **Raise the event in some point**

Subscribers:

- **A method(s) with matching signature**
- **Subscribed to the event**

Musisz mieć zdarzenie tego samego typu co delegat. I oczywiście musisz je podpiąć. W pewnym momencie musisz wywołać zdarzenie. Te trzy pierwsze kroki dotyczą części zdarzenia dla obiektu publisher. Potrzebujemy klasy, która wykona jakąś pracę, a po jej ukończeniu musi zostać odpalone zdarzenie.


Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
using System;
namespace Events
{
public class EntryPoint
{
static void Main(string[] args)
{
}
}
public class Shooter
{
public delegate void KillHandler(object sender, EventArgs e);
}
}
Napiszmy kod. Będziemy mieć gracza, czyli snajpera, który będzie generował zdarzenia (strzały).

Events - Microsoft Visual Studio

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

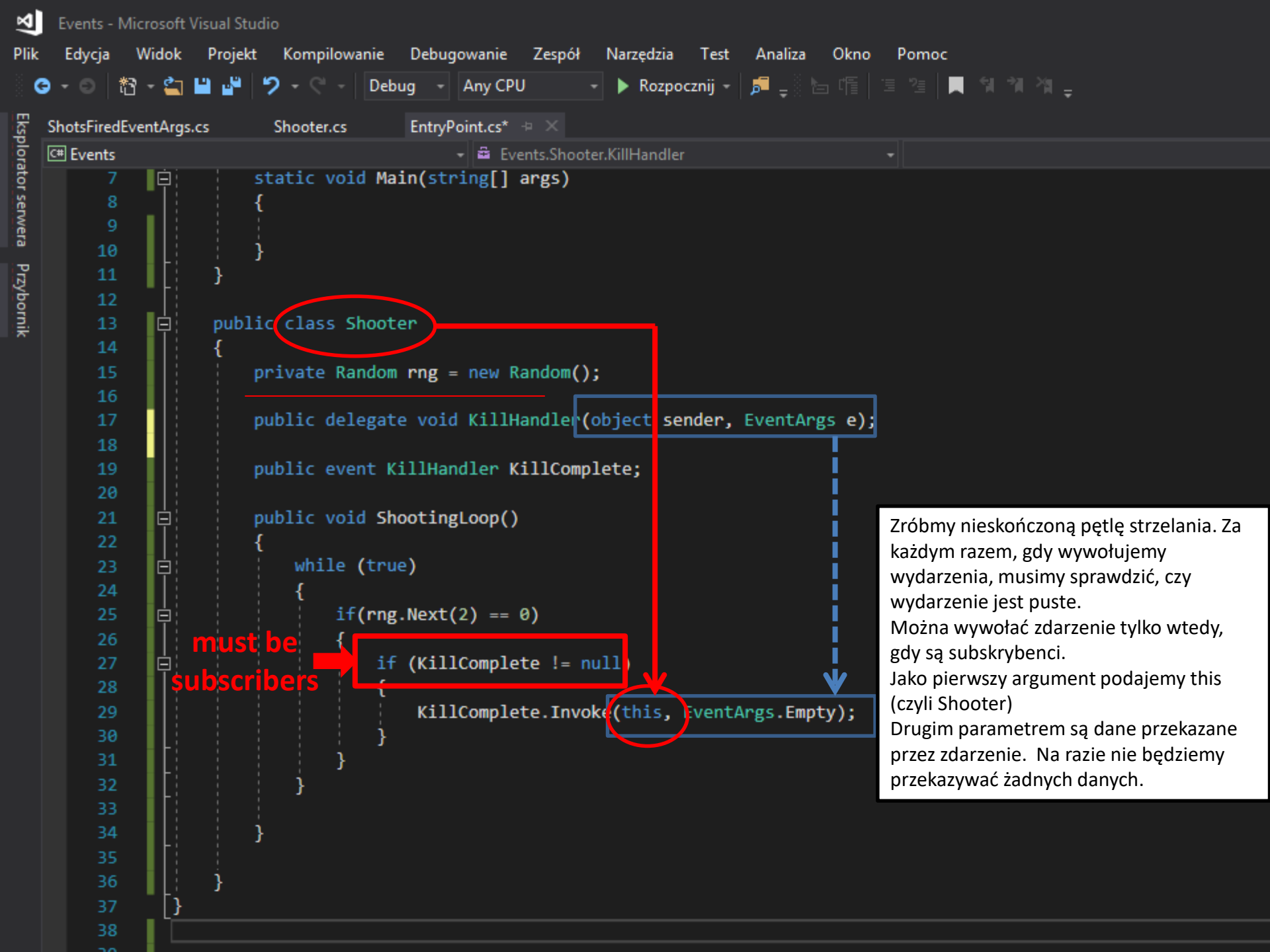
Debug | Any CPU | Rozpocznij

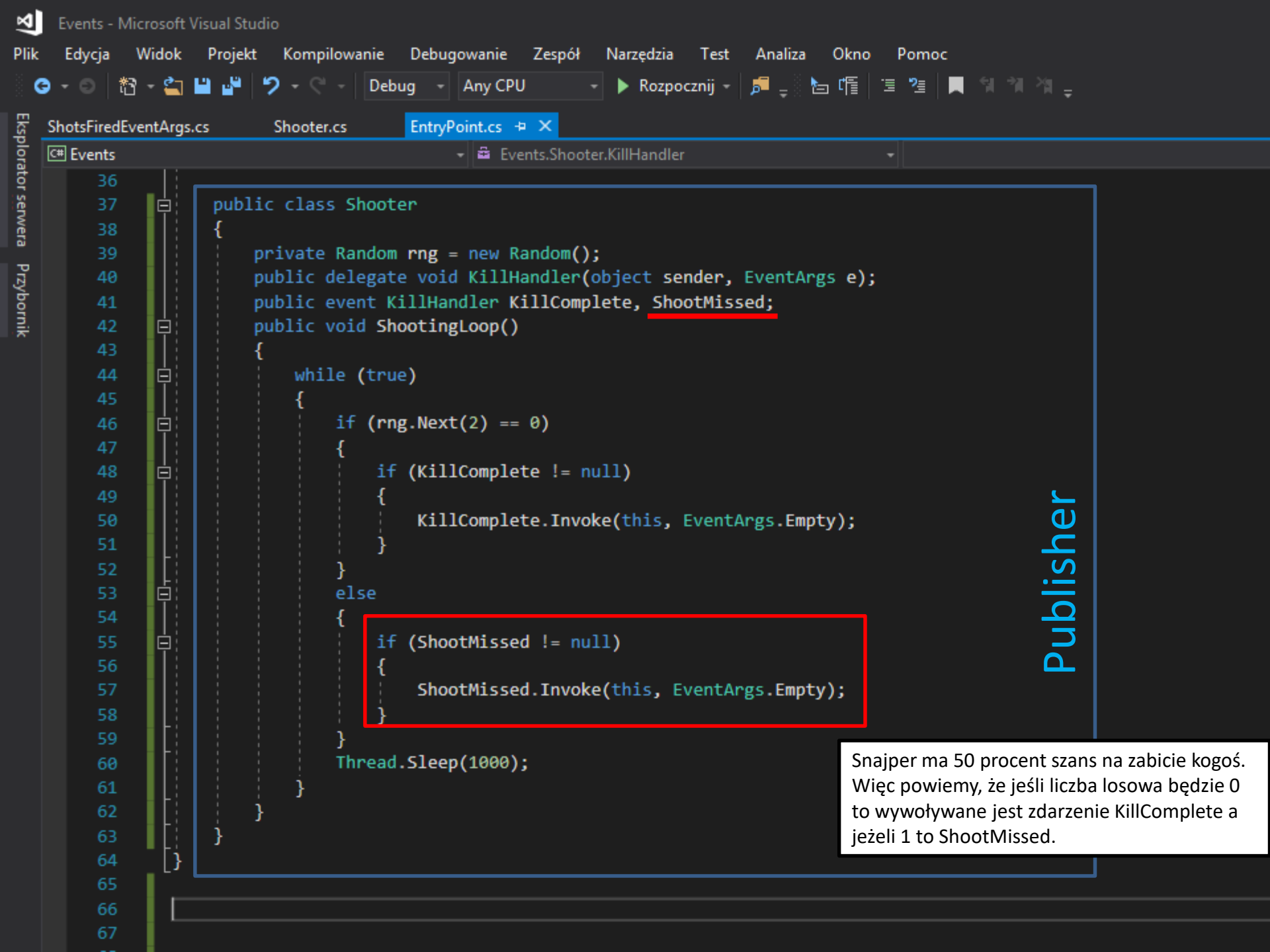
ShotsFiredEventArgs.cs | Shooter.cs | EntryPoint.cs*

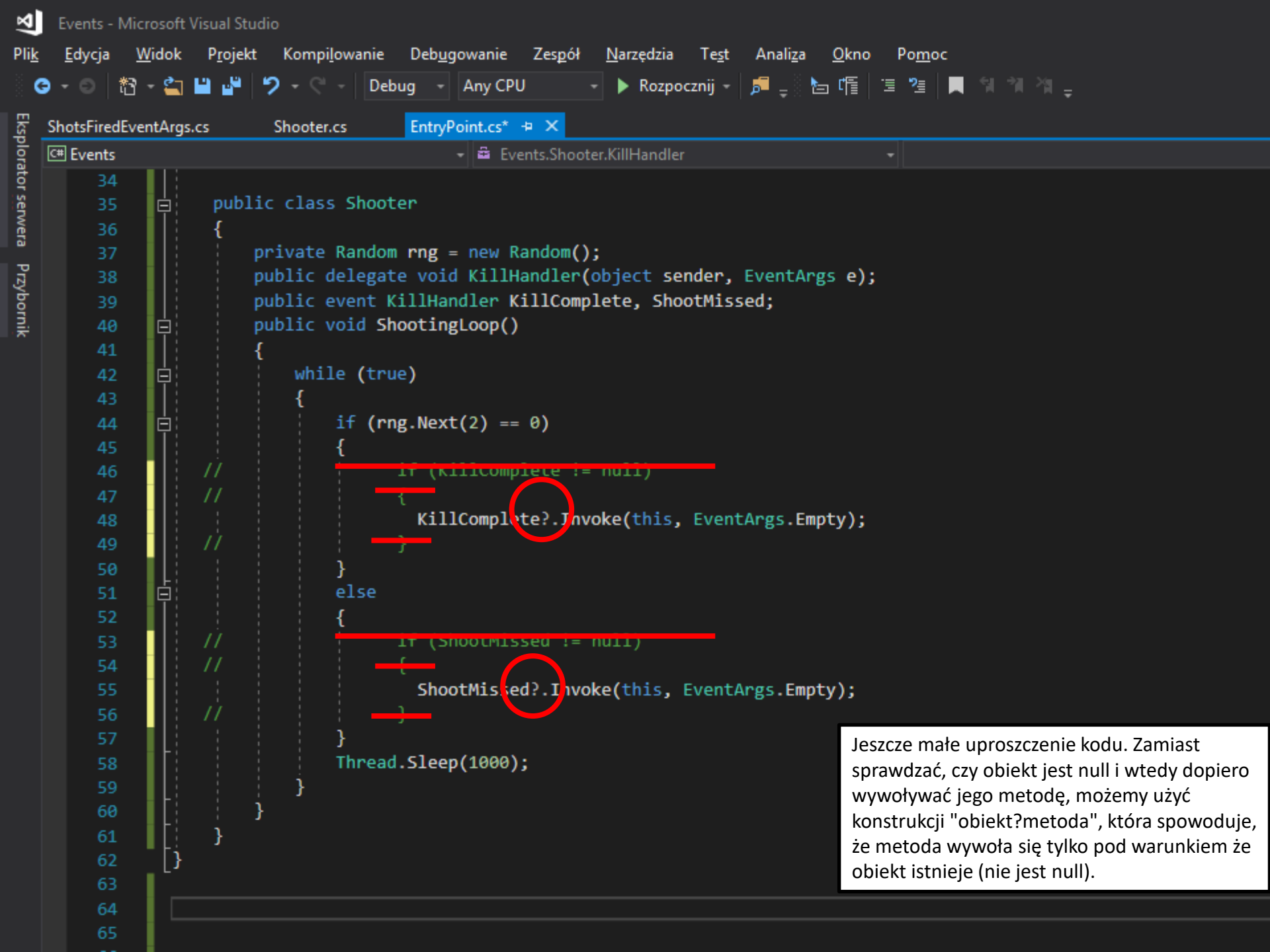
Events | Events.Shooter.KillHandler

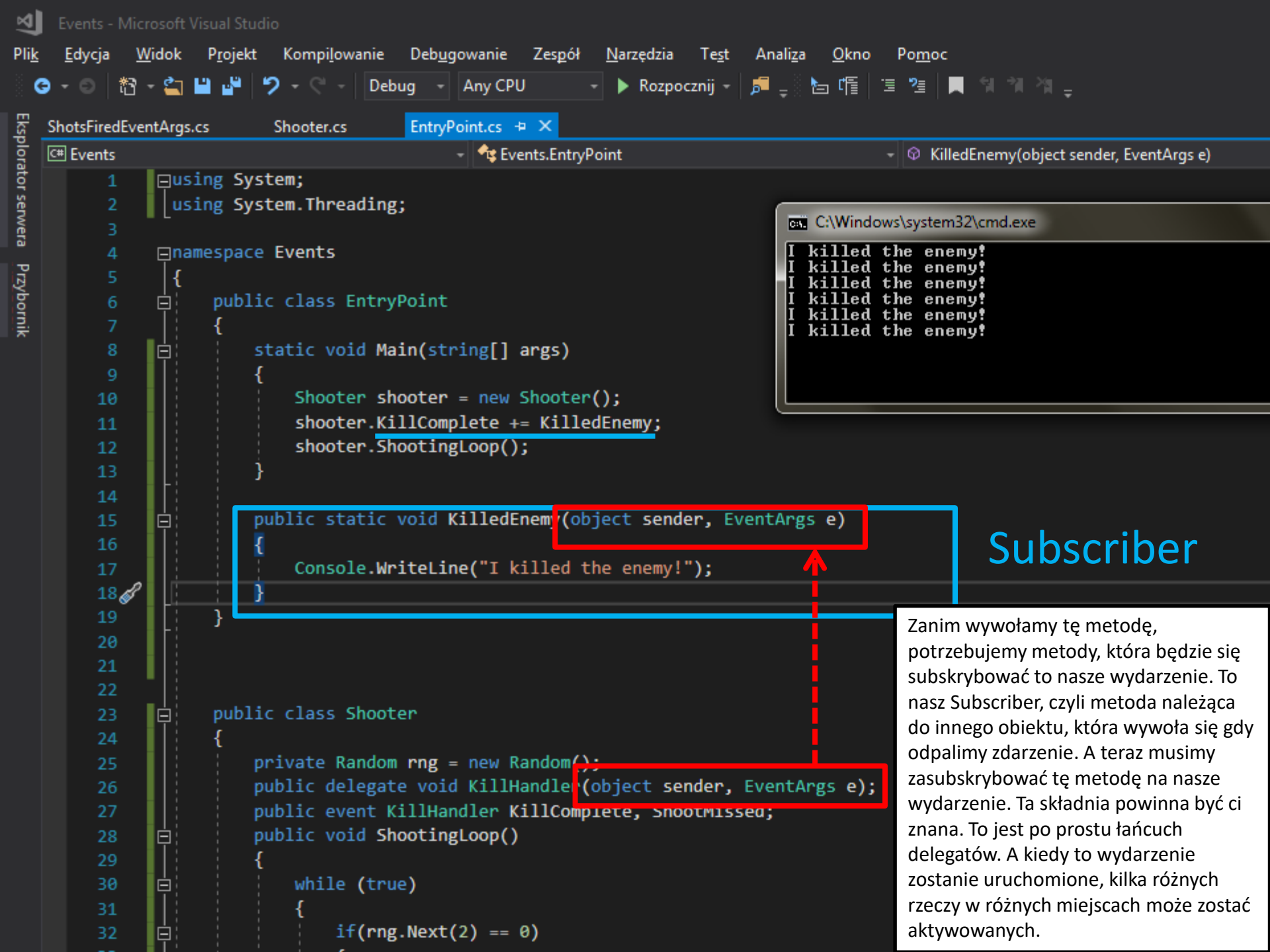
```
1   using System;
2
3   namespace Events
4   {
5       public class EntryPoint
6       {
7           static void Main(string[] args)
8           {
9
10          }
11      }
12
13      public class Shooter
14      {
15          public delegate void KillHandler(object sender, EventArgs e);
16
17
18          public event KillHandler KillComplete;
19      }
20  }
21
22
23
24
25
26
27
28
29
30
31
32
```

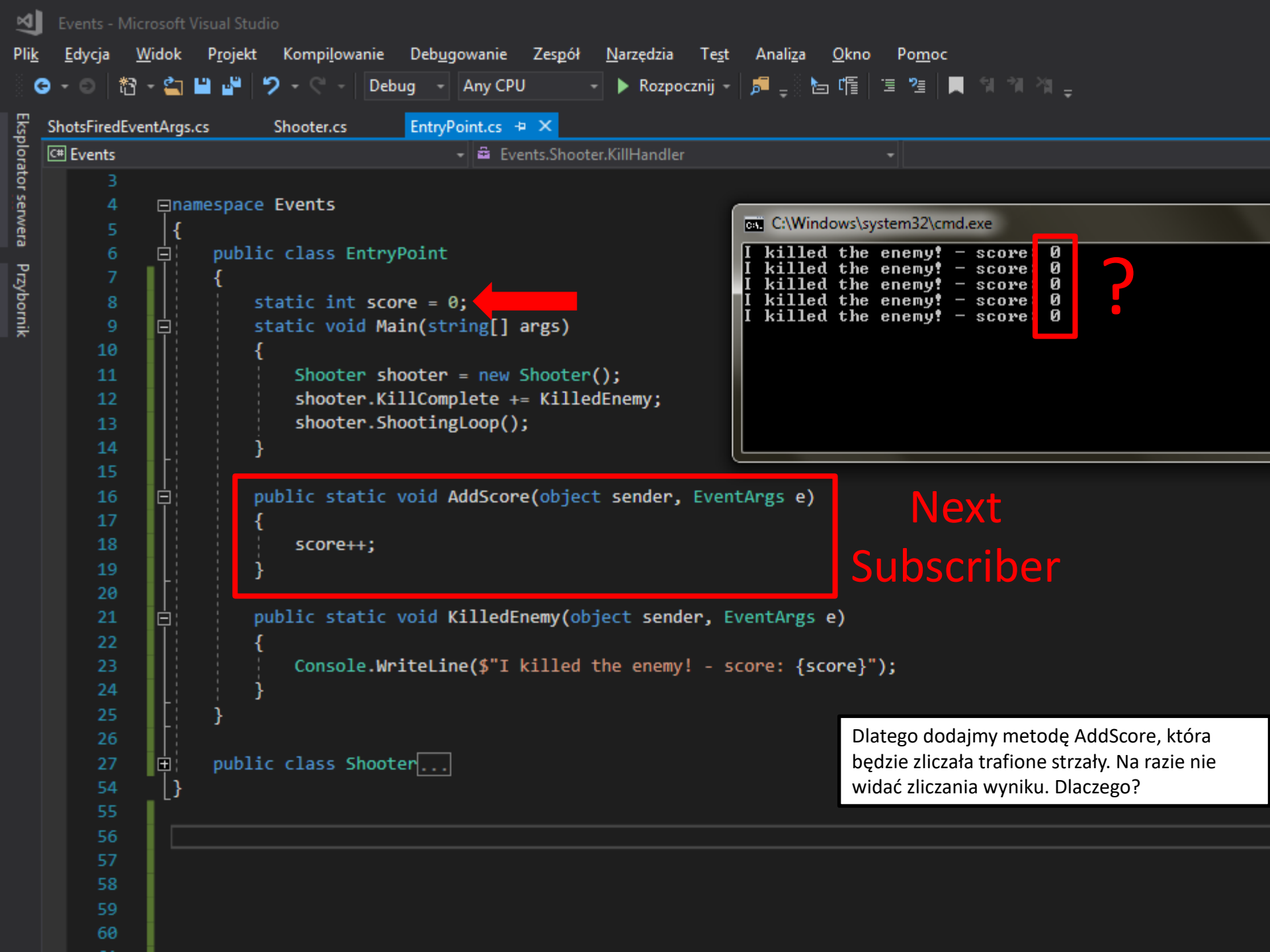
Teraz musimy stworzyć to wydarzenie.
Wszystkie wydarzenia mają tą samą sygnaturę - mają dwa argumenty: obiekt który go wywołał i argumenty zdarzenia.
Stwórzmy więc delegata pasującego do tej sygnatury - delegata publicznego, nasz typ zwrotu będzie void.
To jest nasz delegat.
Teraz musimy stworzyć wydarzenie typu delegata i nazwijmy je KillComplete.

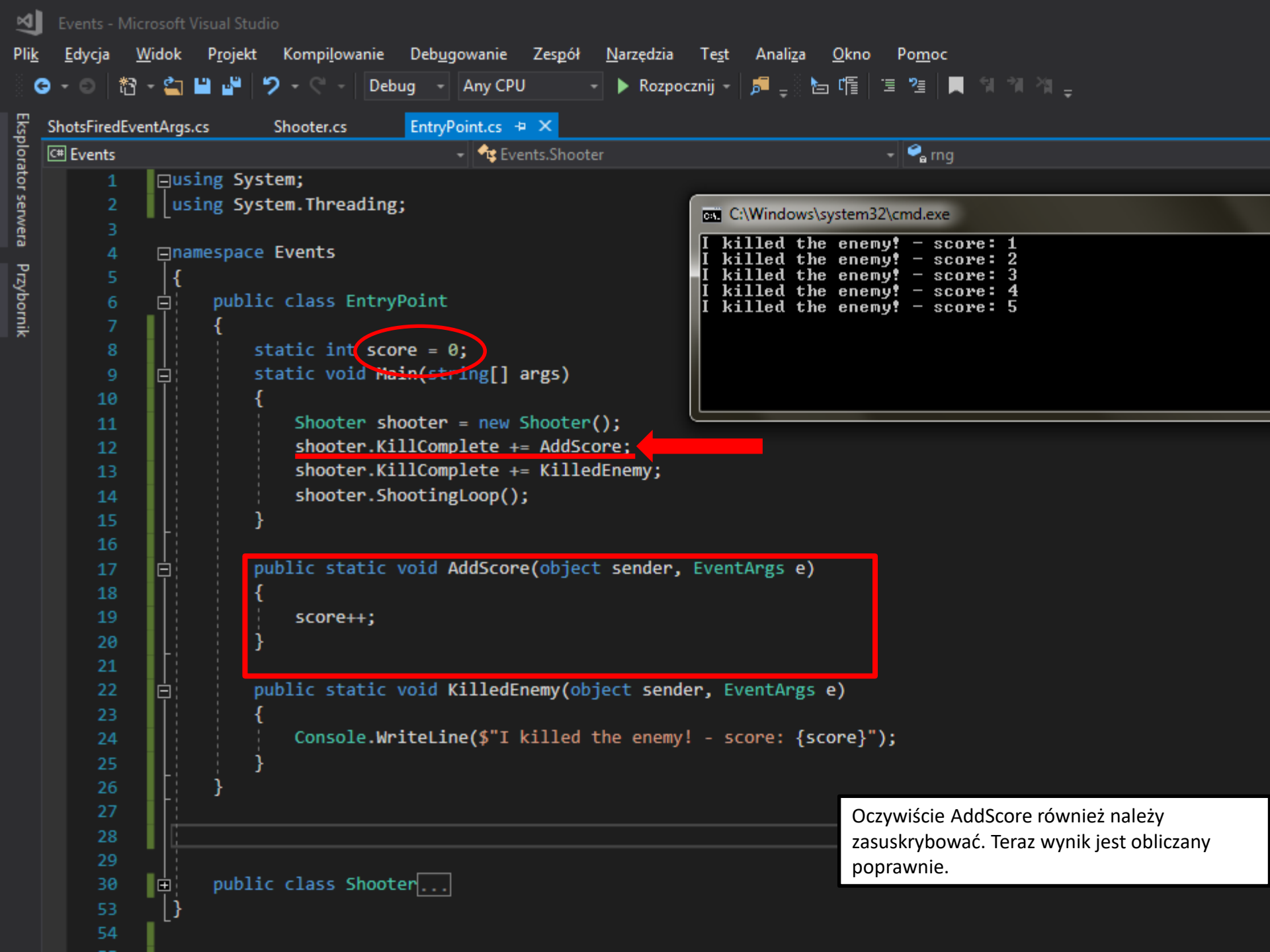

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs*
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
static void Main(string[] args)
{
}
}
public class Shooter
{
private Random rng = new Random();
public delegate void KillHandler(object sender, EventArgs e);
public event KillHandler KillComplete;
public void ShootingLoop()
{
while (true)
{
if(rng.Next(2) == 0)
{
if (KillComplete != null)
{
KillComplete.Invoke(this, EventArgs.Empty);
}
}
}
}
}
}
must be subscribers
Zróbmy nieskończoną pętlę strzelania. Za każdym razem, gdy wywołujemy wydarzenia, musimy sprawdzić, czy wydarzenie jest puste.
Można wywołać zdarzenie tylko wtedy, gdy są subskrybenci.
Jako pierwszy argument podajemy this (czyli Shooter)
Drugim parametrem są dane przekazane przez zdarzenie. Na razie nie będziemy przekazywać żadnych danych.

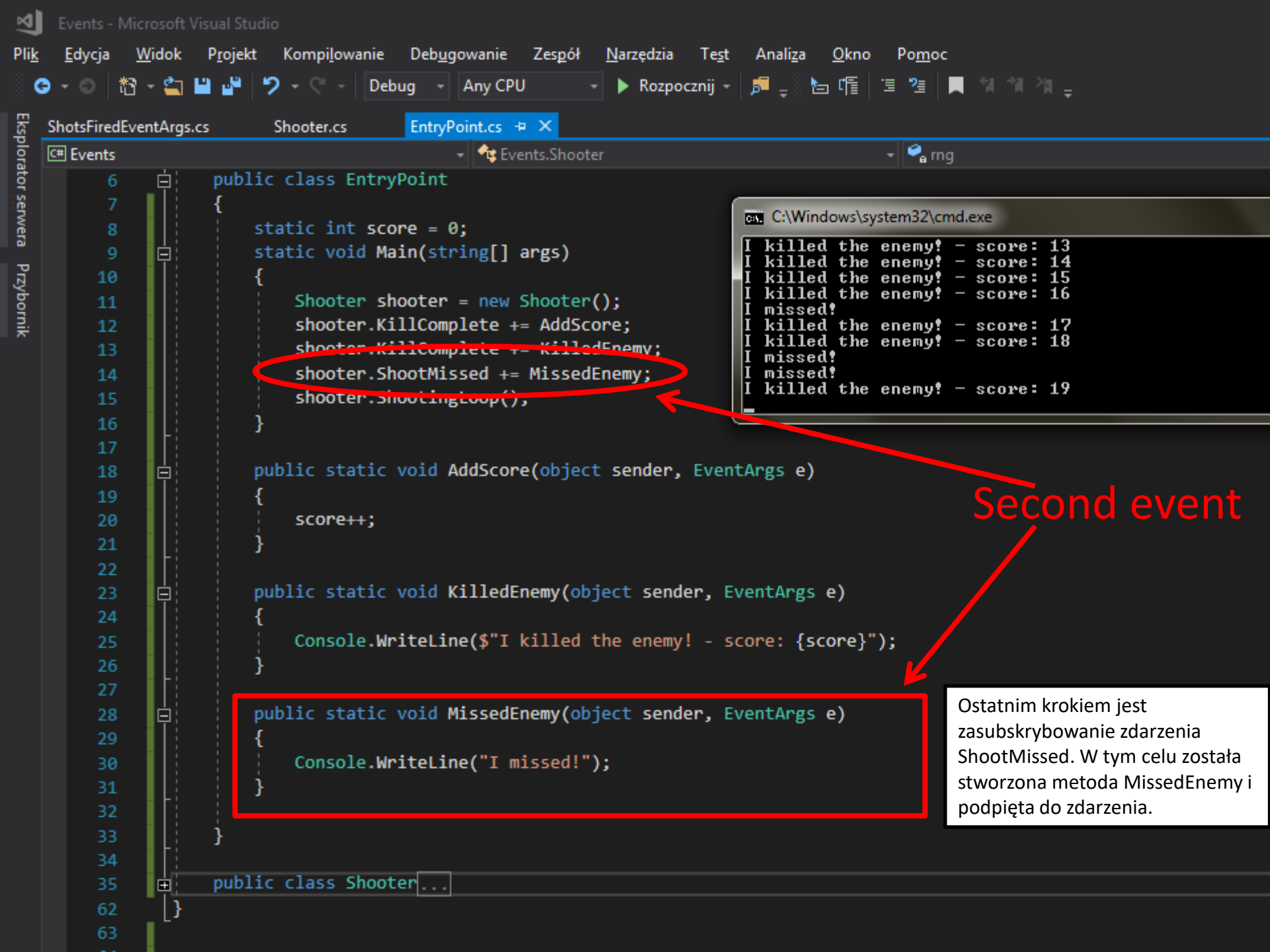

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
public class Shooter
{
private Random rng = new Random();
public delegate void KillHandler(object sender, EventArgs e);
public event KillHandler KillComplete, ShootMissed;
public void ShootingLoop()
{
while (true)
{
if (rng.Next(2) == 0)
{
if (KillComplete != null)
{
KillComplete.Invoke(this, EventArgs.Empty);
}
}
else
{
if (ShootMissed != null)
{
ShootMissed.Invoke(this, EventArgs.Empty);
}
}
Thread.Sleep(1000);
}
}
}
Publisher
Snajper ma 50 procent szans na zabicie kogoś. Więc powiemy, że jeśli liczba losowa będzie 0 to wywoływane jest zdarzenie KillComplete a jeżeli 1 to ShootMissed.


Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs*
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
34 public class Shooter
35 {
36 private Random rng = new Random();
37 public delegate void KillHandler(object sender, EventArgs e);
38 public event KillHandler KillComplete, ShootMissed;
39 public void ShootingLoop()
40 {
41 while (true)
42 {
43 if (rng.Next(2) == 0)
44 {
45 // if (KillComplete != null)
46 // {
47 KillComplete?.Invoke(this, EventArgs.Empty);
48 // }
49 }
50 else
51 {
52 // if (ShootMissed != null)
53 // {
54 ShootMissed?.Invoke(this, EventArgs.Empty);
55 // }
56 }
57 Thread.Sleep(1000);
58 }
59 }
60 }
61 }
62
63
64
65
Jeszcze małe uproszczenie kodu. Zamiast sprawdzać, czy obiekt jest null i wtedy dopiero wywoływać jego metodę, możemy użyć konstrukcji "obiekt?metoda", która spowoduje, że metoda wywoła się tylko pod warunkiem że obiekt istnieje (nie jest null).


Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.EntryPoint
KilledEnemy(object sender, EventArgs e)
Eksplorator serwera
Przybornik
using System;
using System.Threading;
namespace Events
{
public class EntryPoint
{
static void Main(string[] args)
{
Shooter shooter = new Shooter();
shooter.KillComplete += KilledEnemy;
shooter.ShootingLoop();
}
public static void KilledEnemy(object sender, EventArgs e)
{
Console.WriteLine("I killed the enemy!");
}
}
public class Shooter
{
private Random rng = new Random();
public delegate void KillHandler(object sender, EventArgs e);
public event KillHandler KillComplete, ShootMissed;
public void ShootingLoop()
{
while (true)
{
if(rng.Next(2) == 0)
C:\Windows\system32\cmd.exe
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
Subscriber
Zanim wywołamy tę metodę, potrzebujemy metody, która będzie się subskrybować to nasze wydarzenie. To nasz Subscriber, czyli metoda należąca do innego obiektu, która wywoła się gdy odpalimy zdarzenie. A teraz musimy zasubskrybować tę metodę na nasze wydarzenie. Ta składnia powinna być ci znana. To jest po prostu łańcuch delegatów. A kiedy to wydarzenie zostanie uruchomione, kilka różnych rzeczy w różnych miejscach może zostać aktywowanych.

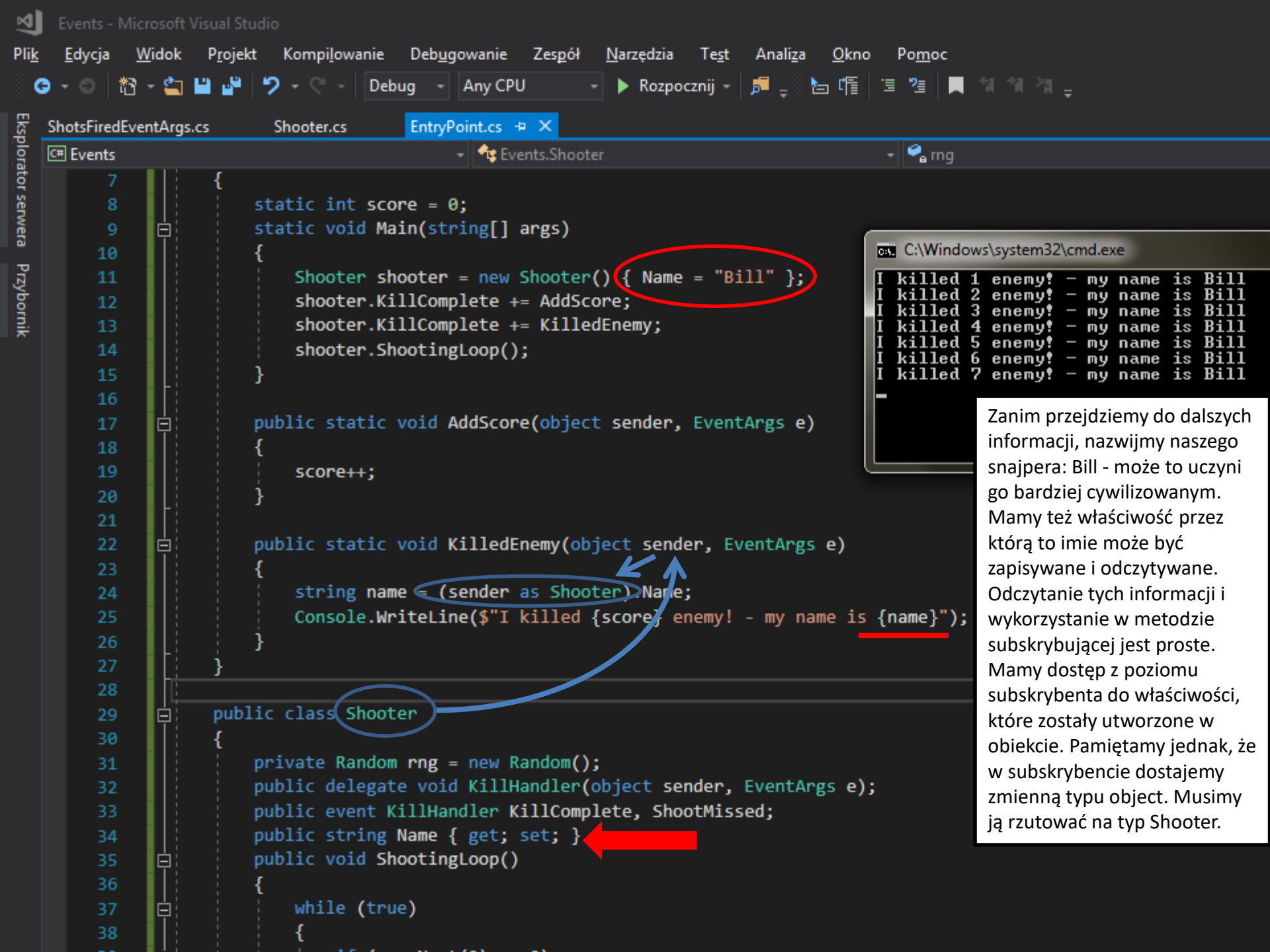

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
namespace Events
{
public class EntryPoint
{
static int score = 0;
static void Main(string[] args)
{
Shooter shooter = new Shooter();
shooter.KillComplete += KilledEnemy;
shooter.ShootingLoop();
}
public static void AddScore(object sender, EventArgs e)
{
score++;
}
public static void KilledEnemy(object sender, EventArgs e)
{
Console.WriteLine($"I killed the enemy! - score: {score}");
}
}
public class Shooter...
}
C:\Windows\system32\cmd.exe
I killed the enemy! - score: 0
I killed the enemy! - score: 0
I killed the enemy! - score: 0
I killed the enemy! - score: 0
I killed the enemy! - score: 0
?
Next Subscriber
Dlatego dodajmy metodę AddScore, która będzie zliczała trafione strzały. Na razie nie widać zliczania wyniku. Dlaczego?

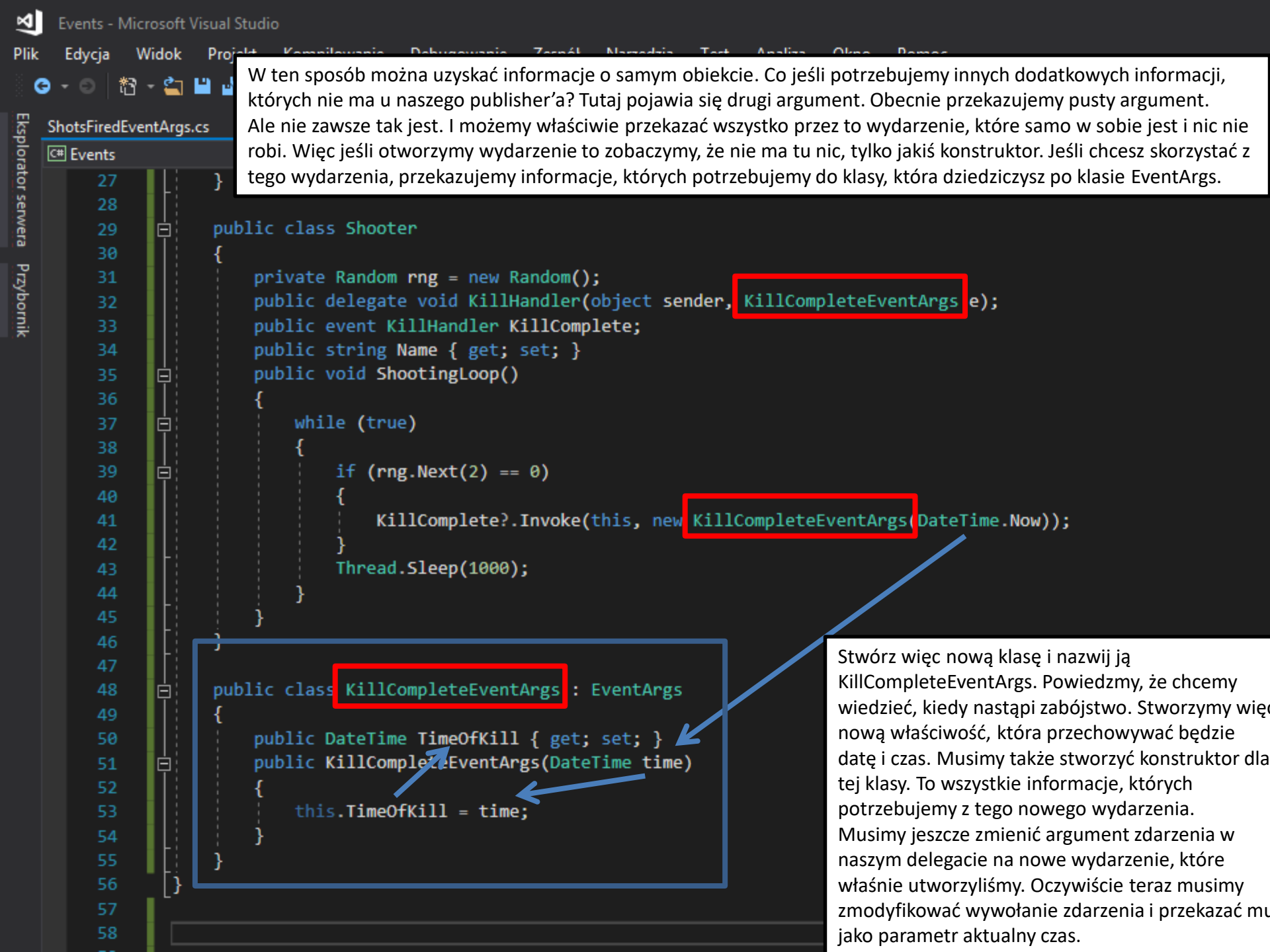

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
rng
Eksplorator serwera
Przybornik
using System;
using System.Threading;
namespace Events
{
public class EntryPoint
{
static int score = 0;
static void Main(string[] args)
{
Shooter shooter = new Shooter();
shooter.KillComplete += AddScore;
shooter.KillComplete += KilledEnemy;
shooter.ShootingLoop();
}
public static void AddScore(object sender, EventArgs e)
{
score++;
}
public static void KilledEnemy(object sender, EventArgs e)
{
Console.WriteLine($"I killed the enemy! - score: {score}");
}
}
public class Shooter...
}
C:\Windows\system32\cmd.exe
I killed the enemy! - score: 1
I killed the enemy! - score: 2
I killed the enemy! - score: 3
I killed the enemy! - score: 4
I killed the enemy! - score: 5
Oczywiście AddScore również należy zasuskrybować. Teraz wynik jest obliczany poprawnie.

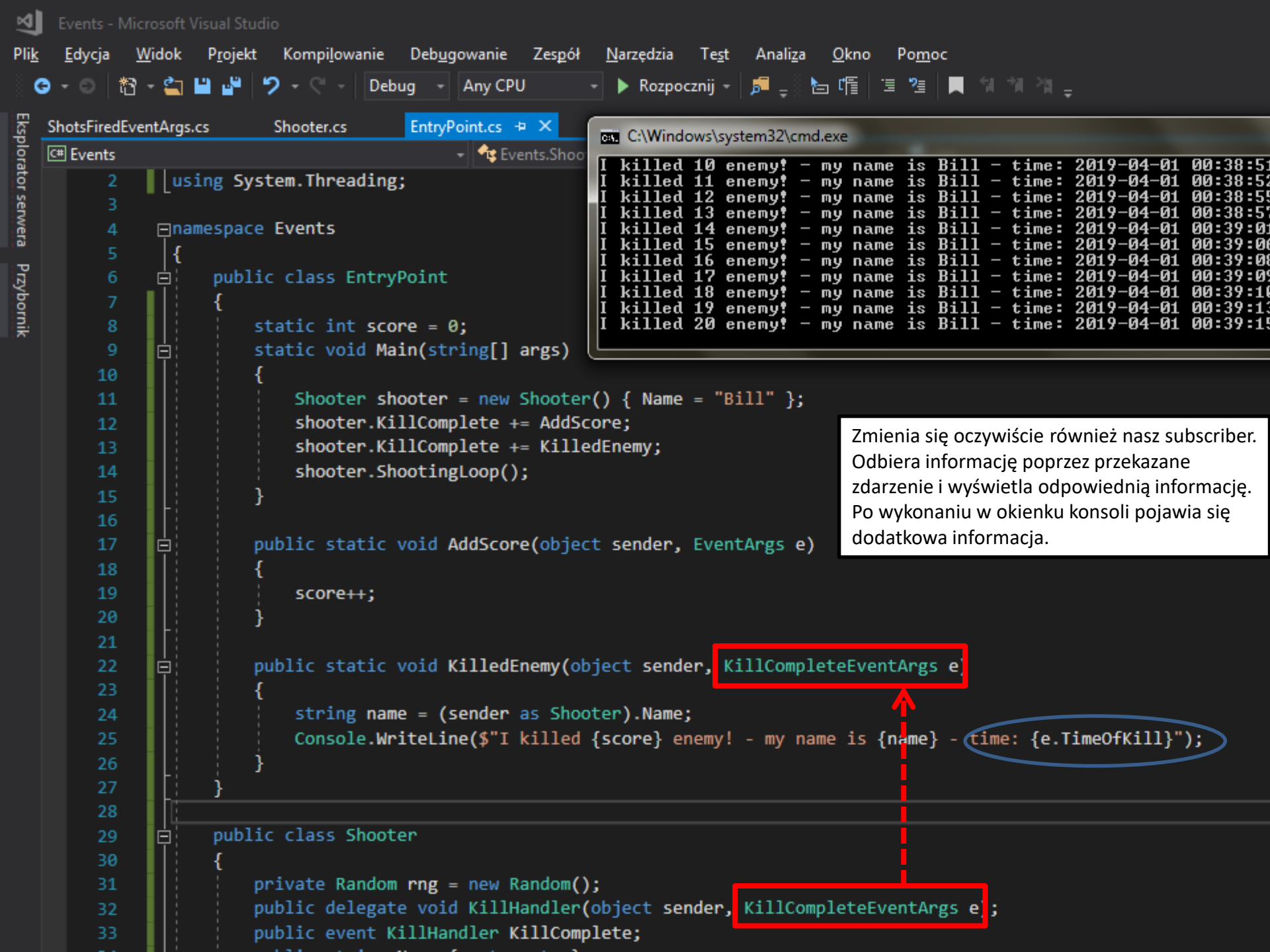

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
rng
Eksplorator serwera
Przybornik
public class EntryPoint
{
static int score = 0;
static void Main(string[] args)
{
Shooter shooter = new Shooter();
shooter.KillComplete += AddScore;
shooter.ShootMissed += MissedEnemy;
}
public static void AddScore(object sender, EventArgs e)
{
score++;
}
public static void KilledEnemy(object sender, EventArgs e)
{
Console.WriteLine($"I killed the enemy! - score: {score}");
}
public static void MissedEnemy(object sender, EventArgs e)
{
Console.WriteLine("I missed!");
}
}
public class Shooter...
}
C:\Windows\system32\cmd.exe
I killed the enemy! - score: 13
I killed the enemy! - score: 14
I killed the enemy! - score: 15
I killed the enemy! - score: 16
I missed!
I killed the enemy! - score: 17
I killed the enemy! - score: 18
I missed!
I missed!
I killed the enemy! - score: 19
Second event
Ostatnim krokiem jest zasubskrybowanie zdarzenia ShootMissed. W tym celu została stworzona metoda MissedEnemy i podpięta do zdarzenia.

Passing additional information with events

Przekazywanie dodatkowych informacji przez zdarzenia.
W niektórych przypadkach może być konieczne przesłanie informacji o obiekcie, który wywołał zdarzenie, a także
kilka innych dodatkowych informacji o obiekcie wywołującym.
To oczywiście wprowadzi pewne sparowanie obiektów.
Czasami po prostu nie można się bez tego obejść.


Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
rng
Eksplorator serwera
Przybornik
7 {
8 static int score = 0;
9 static void Main(string[] args)
10 {
11 Shooter shooter = new Shooter() { Name = "Bill" };
12 shooter.KillComplete += AddScore;
13 shooter.KillComplete += KilledEnemy;
14 shooter.ShootingLoop();
15 }
16
17 public static void AddScore(object sender, EventArgs e)
18 {
19 score++;
20 }
21
22 public static void KilledEnemy(object sender, EventArgs e)
23 {
24 string name = (sender as Shooter).Name;
25 Console.WriteLine($"I killed {score} enemy! - my name is {name}");
26 }
27 }
28
29 public class Shooter
30 {
31 private Random rng = new Random();
32 public delegate void KillHandler(object sender, EventArgs e);
33 public event KillHandler KillComplete, ShootMissed;
34 public string Name { get; set; }
35 public void ShootingLoop()
36 {
37 while (true)
38 {
C:\Windows\system32\cmd.exe
I killed 1 enemy! - my name is Bill
I killed 2 enemy! - my name is Bill
I killed 3 enemy! - my name is Bill
I killed 4 enemy! - my name is Bill
I killed 5 enemy! - my name is Bill
I killed 6 enemy! - my name is Bill
I killed 7 enemy! - my name is Bill
Zanim przejdziemy do dalszych informacji, nazwijmy naszego snajpera: Bill - może to uczyni go bardziej cywilizowanym. Mamy też właściwość przez którą to imie może być zapisywane i odczytywane. Odczytanie tych informacji i wykorzystanie w metodzie subskrybującej jest proste. Mamy dostęp z poziomu subskrybenta do właściwości, które zostały utworzone w obiekcie. Pamiętamy jednak, że w subskrybencie dostajemy zmienną typu object. Musimy ją rzutować na typ Shooter.

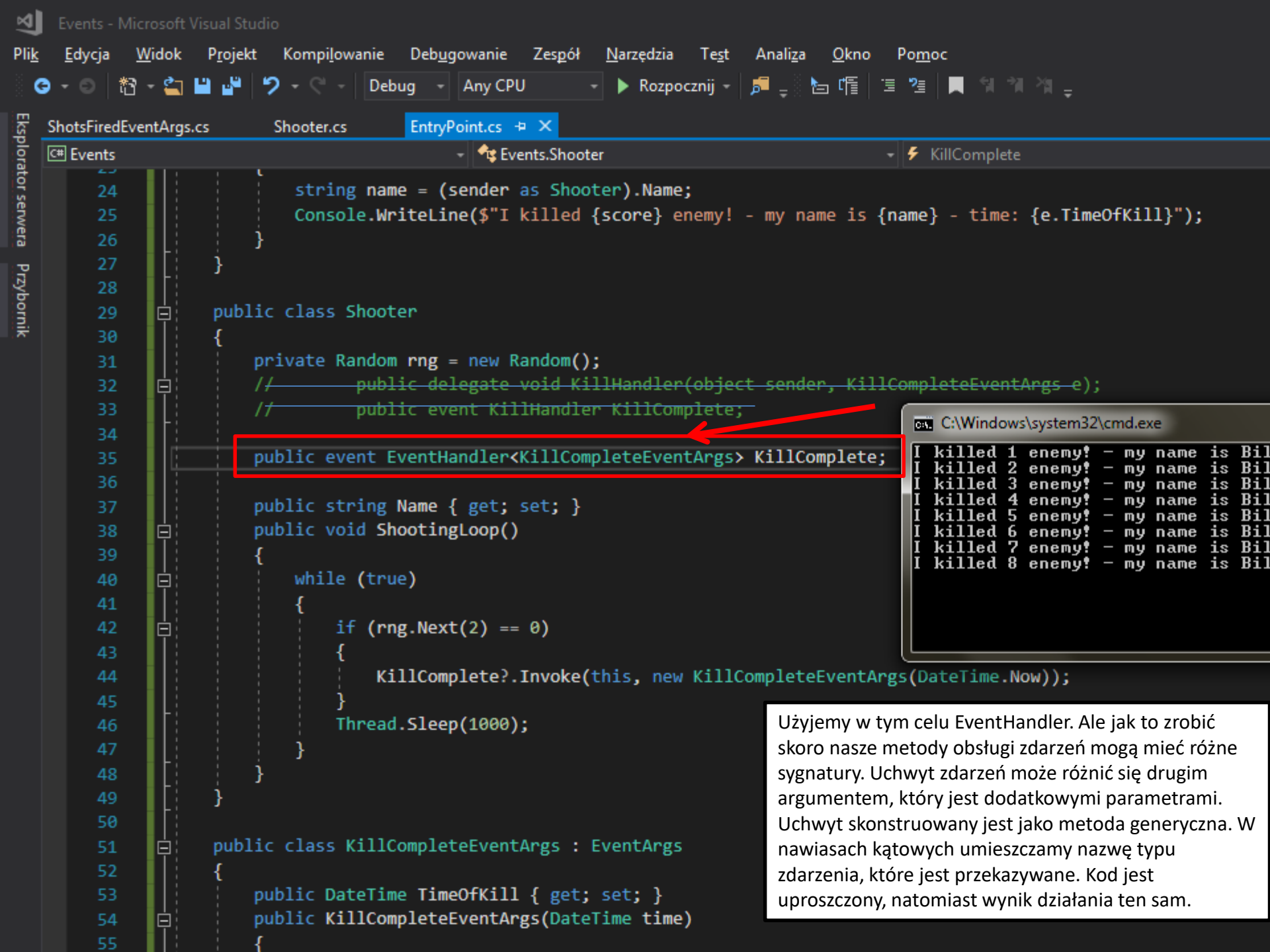

Events - Microsoft Visual Studio
Plik Edycja Widok
ShotsFiredEventArgs.cs
Events
W ten sposób można uzyskać informacje o samym obiekcie. Co jeśli potrzebujemy innych dodatkowych informacji, których nie ma u naszego publisher'a? Tutaj pojawia się drugi argument. Obecnie przekazujemy pusty argument. Ale nie zawsze tak jest. I możemy właściwie przekazać wszystko przez to wydarzenie, które samo w sobie jest i nic nie robi. Więc jeśli otworzymy wydarzenie to zobaczymy, że nie ma tu nic, tylko jakiś konstruktor. Jeśli chcesz skorzystać z tego wydarzenia, przekazujemy informacje, których potrzebujemy do klasy, która dziedziczysz po klasie EventArgs.
public class Shooter
{
private Random rng = new Random();
public delegate void KillHandler(object sender, KillCompleteEventArgs e);
public event KillHandler KillComplete;
public string Name { get; set; }
public void ShootingLoop()
{
while (true)
{
if (rng.Next(2) == 0)
{
KillComplete?.Invoke(this, new KillCompleteEventArgs(DateTime.Now));
}
Thread.Sleep(1000);
}
}
}
public class KillCompleteEventArgs : EventArgs
{
public DateTime TimeOfKill { get; set; }
public KillCompleteEventArgs(DateTime time)
{
this.TimeOfKill = time;
}
}
Stwórz więc nową klasę i nazwij ją KillCompleteEventArgs. Powiedzmy, że chcemy wiedzieć, kiedy nastąpi zabójstwo. Stworzymy więc nową właściwość, która przechowywać będzie datę i czas. Musimy także stworzyć konstruktor dla tej klasy. To wszystkie informacje, których potrzebujemy z tego nowego wydarzenia. Musimy jeszcze zmienić argument zdarzenia w naszym delegacie na nowe wydarzenie, które właśnie utworzyliśmy. Oczywiście teraz musimy zmodyfikować wywołanie zdarzenia i przekazać mu jako parametr aktualny czas.




```
using System.Threading;

namespace Events
{
    public class EntryPoint
    {
        static int score = 0;
        static void Main(string[] args)
        {
            Shooter shooter = new Shooter() { Name = "Bill" };
            shooter.KillComplete += AddScore;
            shooter.KillComplete += KilledEnemy;
            shooter.ShootingLoop();
        }

        public static void AddScore(object sender, EventArgs e)
        {
            score++;
        }

        public static void KilledEnemy(object sender, KillCompleteEventArgs e)
        {
            string name = (sender as Shooter).Name;
            Console.WriteLine($"I killed {score} enemy! - my name is {name} - time: {e.TimeOfKill}");
        }
    }

    public class Shooter
    {
        private Random rng = new Random();
        public delegate void KillHandler(object sender, KillCompleteEventArgs e);
        public event KillHandler KillComplete;
```

No Delegates - EventHandler

Ostatnią rzeczą, którą chcę wam pokazać o wydarzeniach, to że mamy krótszą drogę w której możemy osiągnąć dokładnie to samo.


Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
KillComplete
Eksplorator serwera
Przybornik
24 string name = (sender as Shooter).Name;
25 Console.WriteLine($"I killed {score} enemy! - my name is {name} - time: {e.TimeOfKill}");
26 }
27 }
28
29 public class Shooter
30 {
31 private Random rng = new Random();
32 // public delegate void KillHandler(object sender, KillCompleteEventArgs e);
33 // public event KillHandler KillComplete;
34
35 public event EventHandler<KillCompleteEventArgs> KillComplete;
36
37 public string Name { get; set; }
38 public void ShootingLoop()
39 {
40 while (true)
41 {
42 if (rng.Next(2) == 0)
43 {
44 KillComplete?.Invoke(this, new KillCompleteEventArgs(DateTime.Now));
45 }
46 Thread.Sleep(1000);
47 }
48 }
49 }
50
51 public class KillCompleteEventArgs : EventArgs
52 {
53 public DateTime TimeOfKill { get; set; }
54 public KillCompleteEventArgs(DateTime time)
55 {
C:\Windows\system32\cmd.exe
I killed 1 enemy! - my name is Bil
I killed 2 enemy! - my name is Bil
I killed 3 enemy! - my name is Bil
I killed 4 enemy! - my name is Bil
I killed 5 enemy! - my name is Bil
I killed 6 enemy! - my name is Bil
I killed 7 enemy! - my name is Bil
I killed 8 enemy! - my name is Bil
Użyjemy w tym celu EventHandler. Ale jak to zrobić skoro nasze metody obsługi zdarzeń mogą mieć różne sygnatury. Uchwyt zdarzeń może różnić się drugim argumentem, który jest dodatkowymi parametrami. Uchwyt skonstruowany jest jako metoda generyczna. W nawiasach kątowych umieszczamy nazwę typu zdarzenia, które jest przekazywane. Kod jest uproszczony, natomiast wynik działania ten sam.