Events – Publisher and Subscribers

If you want a given action to be activated, if something happens in another object, then this is the place for "events".
They allow us to write both codes separately and avoid joins in the code between objects and yet the objects will work together.

What is event?

The event fires and executes different code. If you look at this example, this weapon shoots bullets and the soldiers are hiding. They react to the event.
The object that stores the event or the one that triggers it is called publisher in this example, while the methods added to this event are called subscribers.
Imagine, then, that all soldiers who are fired at with this weapon are different methods separate methods and all of them are subscribers to this event.

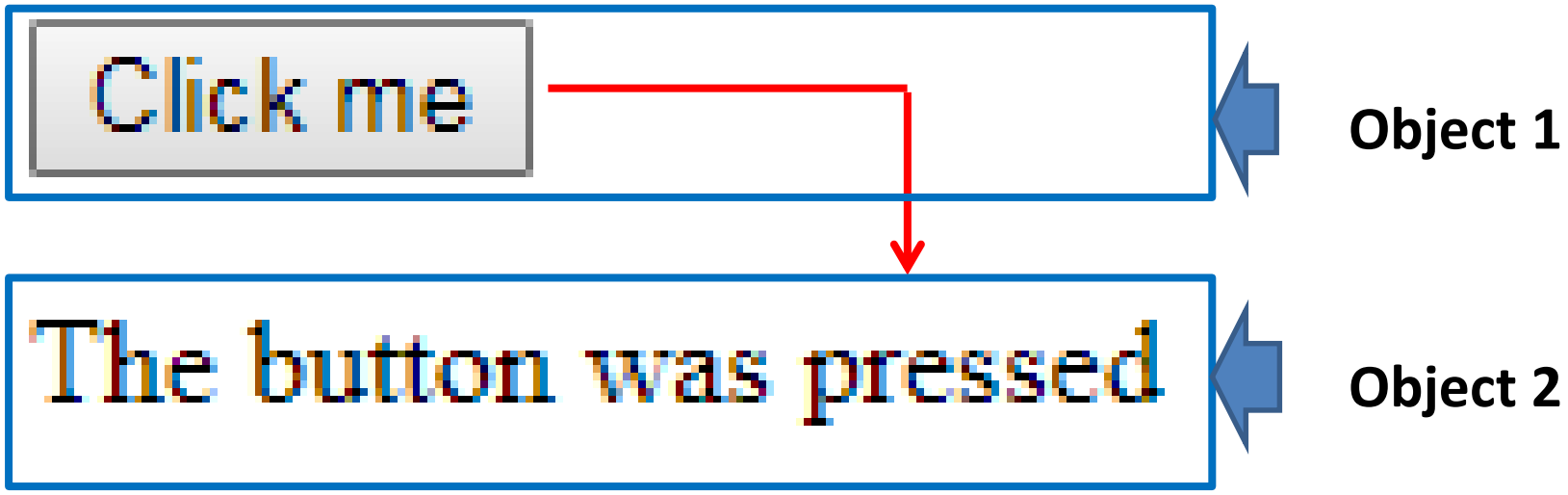
Publisher

Object which holds the event

Action

Subscribers

Methods that executes when event is fired



When you start working with Windows Forms, the simplest example would be clicking a button and something must happen. Click
activates the event, which in turn activated the subscribed method.

The anatomy of an Event

Anatomy of an Event

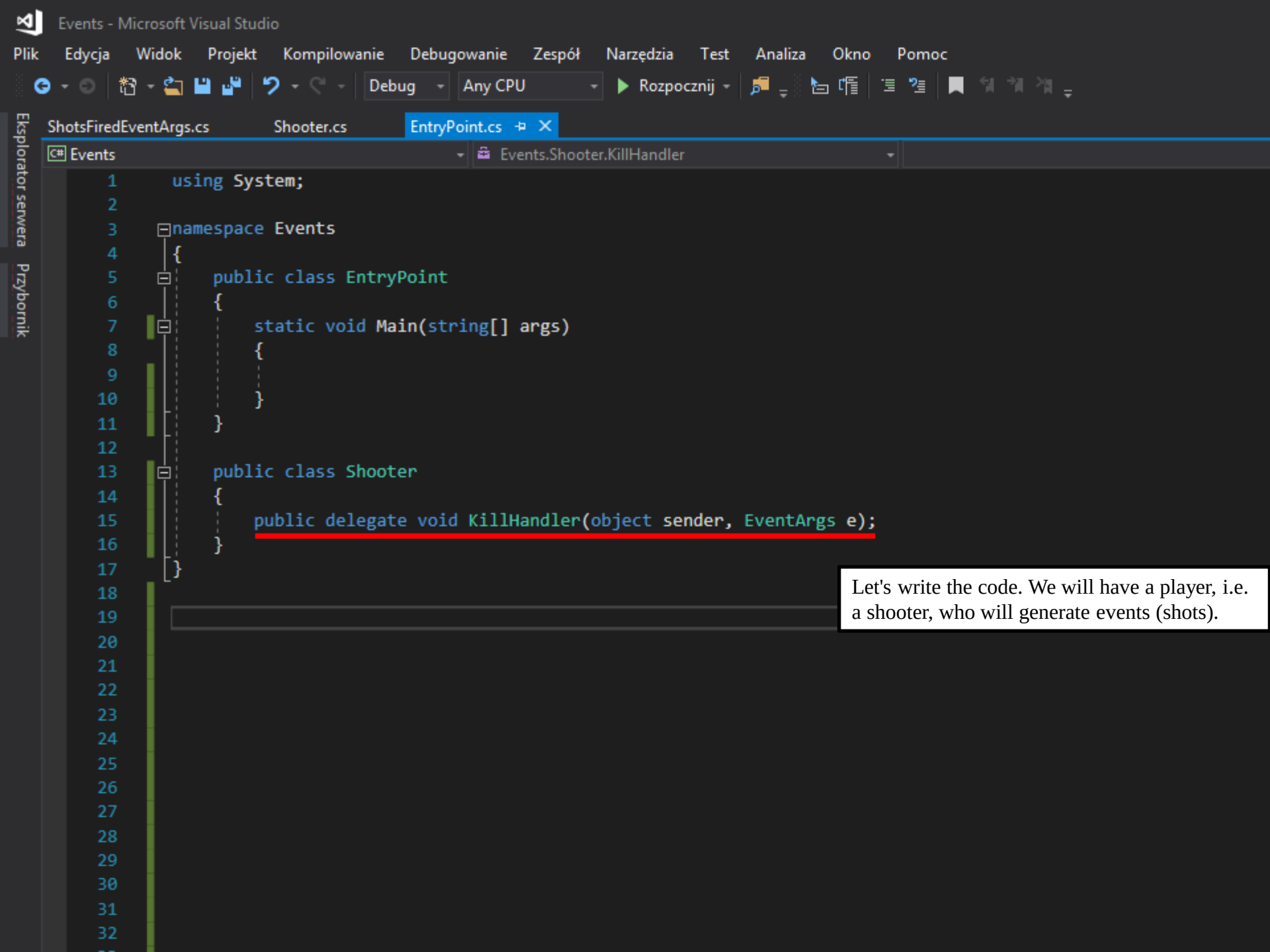
Publisher:

- **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**

You must have the same type of event as the delegate. And of course you have to hook them up. At some point you must trigger an event. These first three steps relate to part of the event for the publisher.
We need a class that will do some work and after its completion an event must be fired.


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
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
19
20
21
22
23
24
25
26
27
28
29
30
31
32
Let's write the code. We will have a player, i.e. a shooter, who will generate events (shots).

Events - Microsoft Visual Studio

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

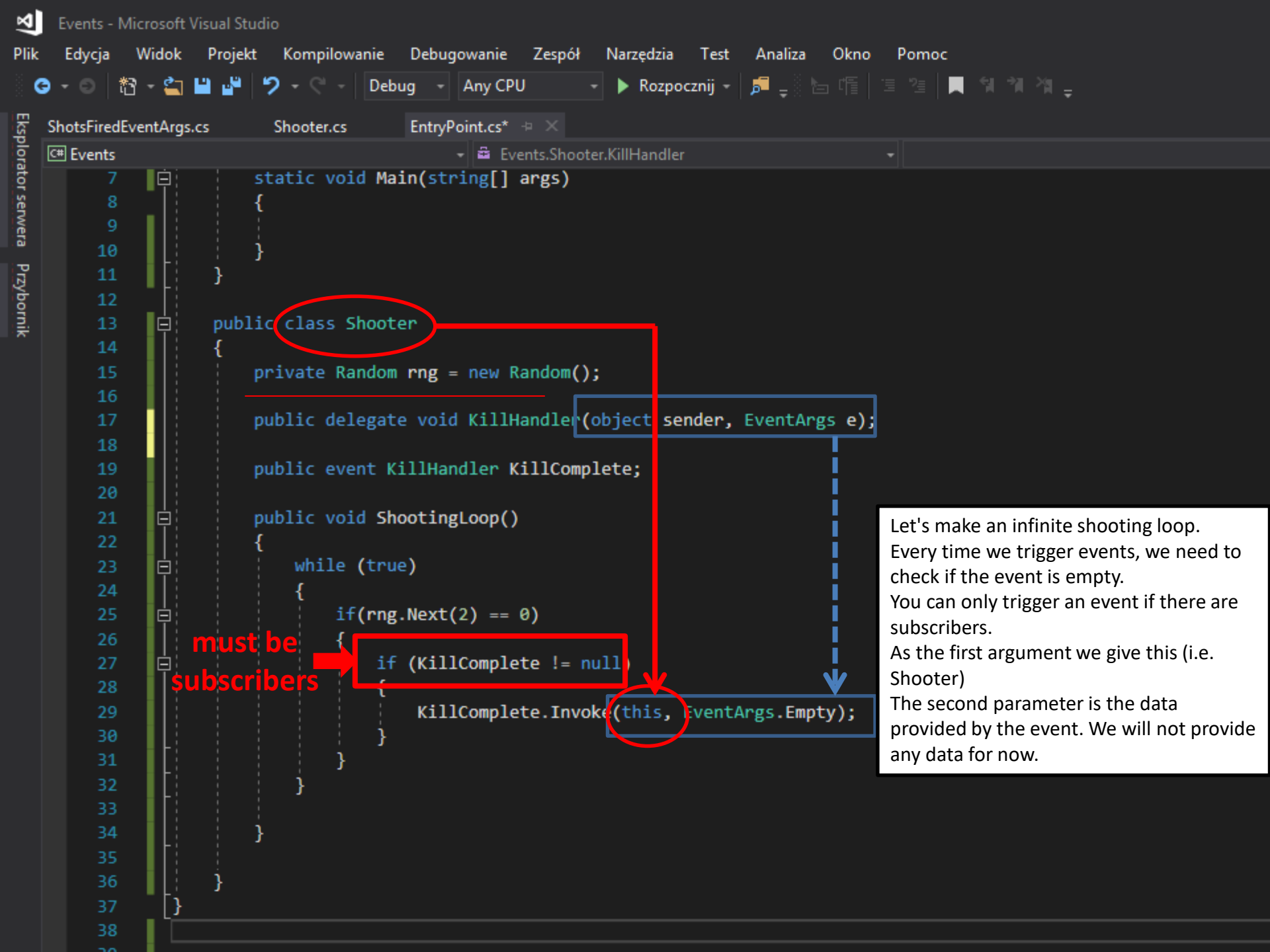
Debug | Any CPU | Rozpocznij

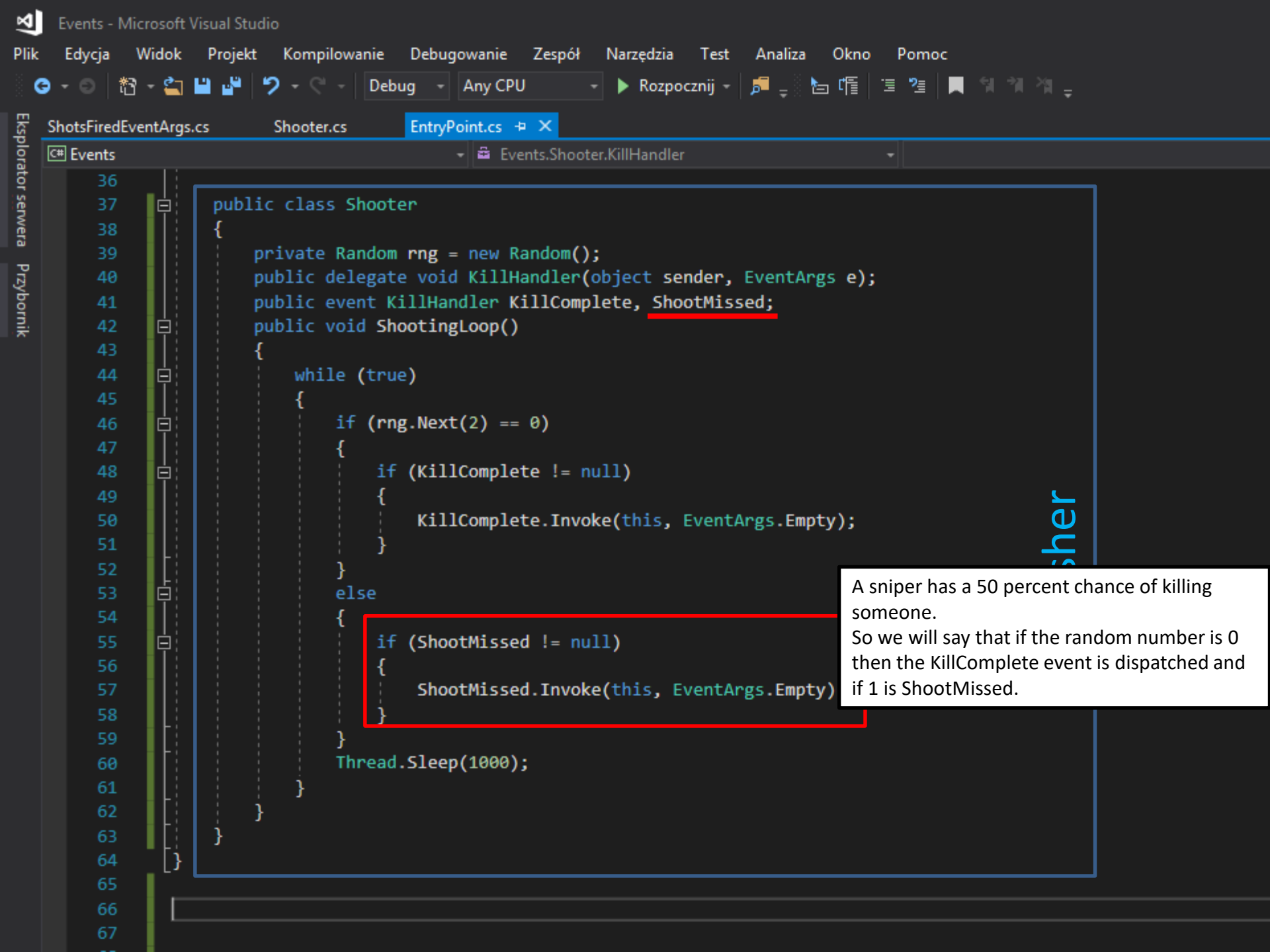
ShotsFiredEventArgs.cs | Shooter.cs | EntryPoint.cs*

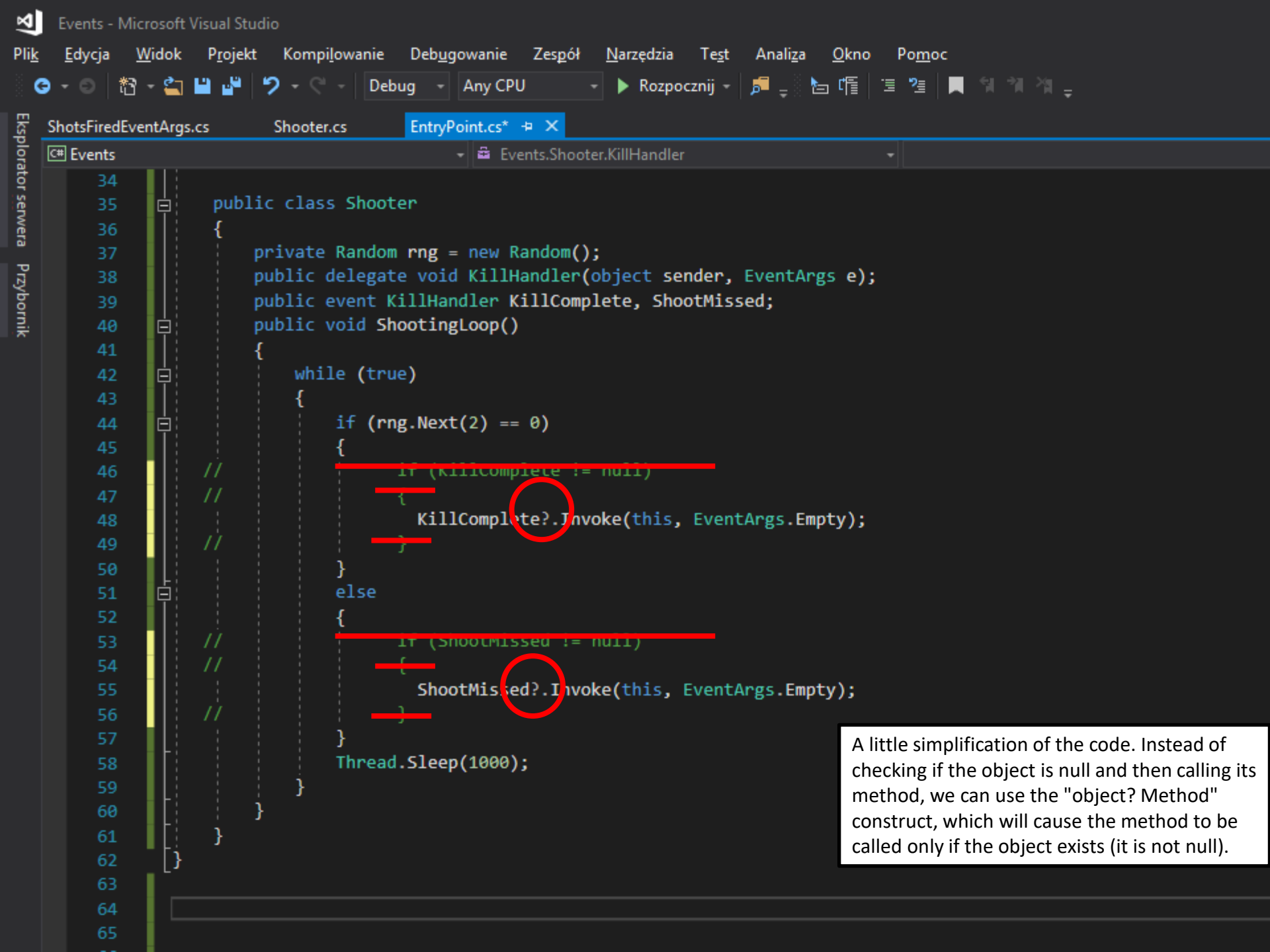
Events | Events.Shooter.KillHandler

```
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
```

Now we must create this event.
All events have the same signature - they have two arguments: the object that triggered it and the event arguments.
So let's create a delegate matching this signature - a public delegate, our return type will be void.
This is our delegate.
Now we need to create a delegate event and let's call it 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
Let's make an infinite shooting loop.
Every time we trigger events, we need to check if the event is empty.
You can only trigger an event if there are subscribers.
As the first argument we give this (i.e. Shooter)
The second parameter is the data provided by the event. We will not provide any data for now.


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);
}
}
}
A sniper has a 50 percent chance of killing someone.
So we will say that if the random number is 0 then the KillComplete event is dispatched and if 1 is 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
A little simplification of the code. Instead of checking if the object is null and then calling its method, we can use the "object? Method" construct, which will cause the method to be called only if the object exists (it is not 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)

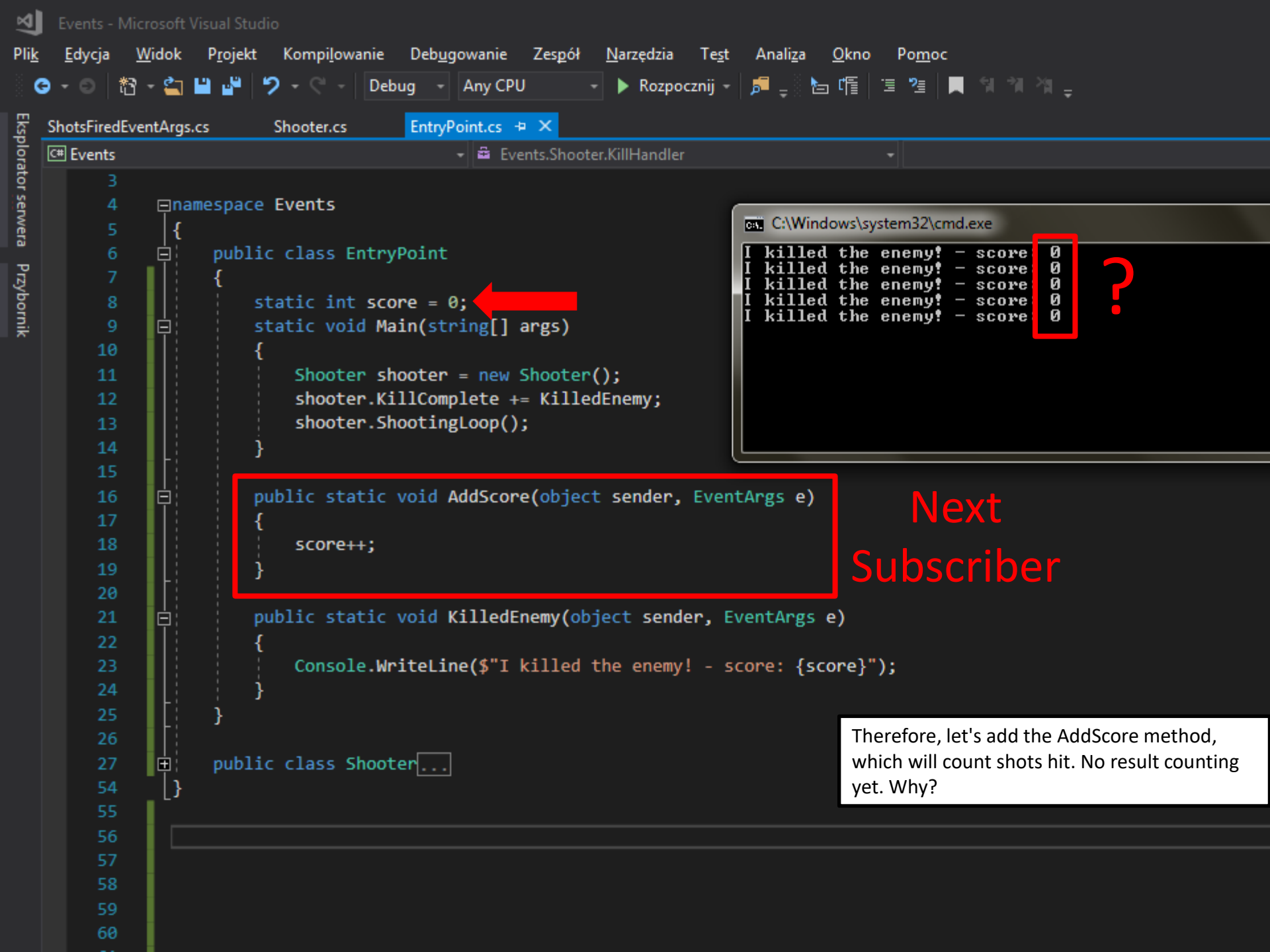
```
1  using System;
2  using System.Threading;
3
4  namespace Events
5  {
6      public class EntryPoint
7      {
8          static void Main(string[] args)
9          {
10             Shooter shooter = new Shooter();
11             shooter.KillComplete += KilledEnemy;
12             shooter.ShootingLoop();
13         }
14
15         public static void KilledEnemy(object sender, EventArgs e)
16         {
17             Console.WriteLine("I killed the enemy!");
18         }
19     }
20
21
22
23     public class Shooter
24     {
25         private Random rng = new Random();
26         public delegate void KillHandler(object sender, EventArgs e);
27         public event KillHandler KillComplete, ShootMissed;
28         public void ShootingLoop()
29         {
30             while (true)
31             {
32                 if(rng.Next(2) == 0)
```

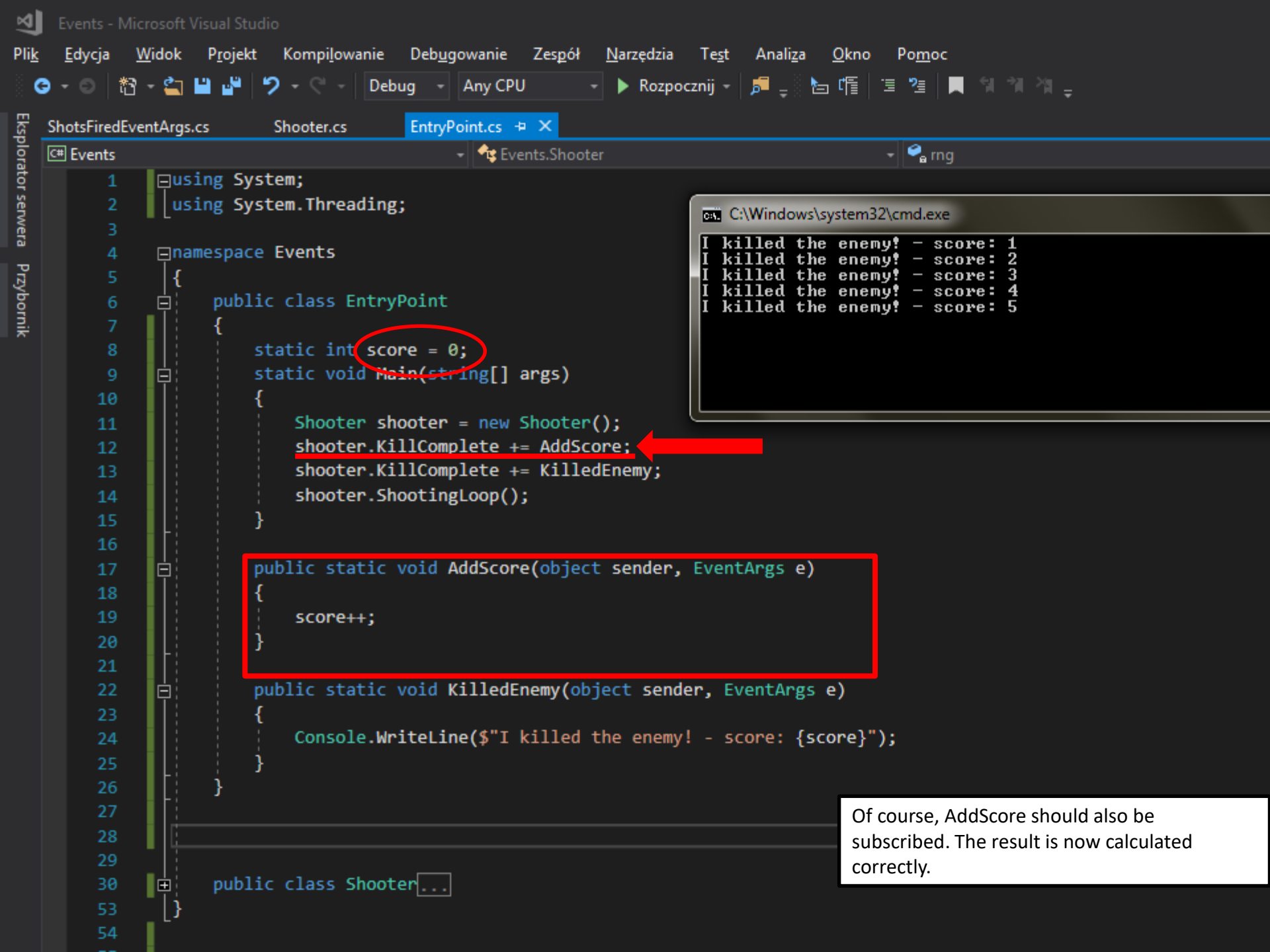
C:\Windows\system32\cmd.exe

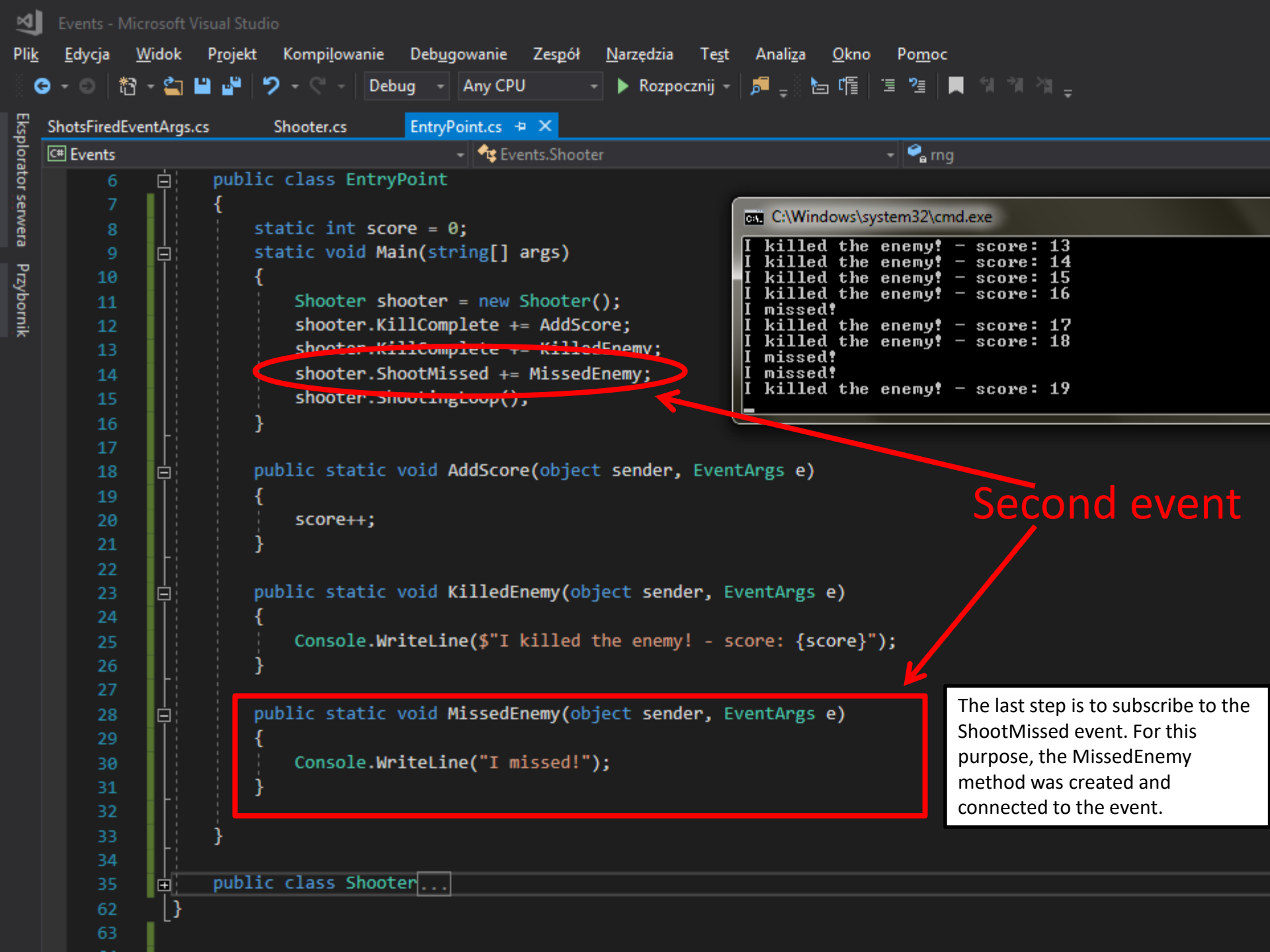
```
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
I killed the enemy!
```

Subscriber

Before we call this method, we need a method that will subscribe to our event. This is our Subscriber, a method belonging to another object that will be triggered when we fire the event. And now we have to subscribe to this method for our event. You should know this syntax. It's just a chain of delegates. And when this event is launched, several different things in different places can be activated.

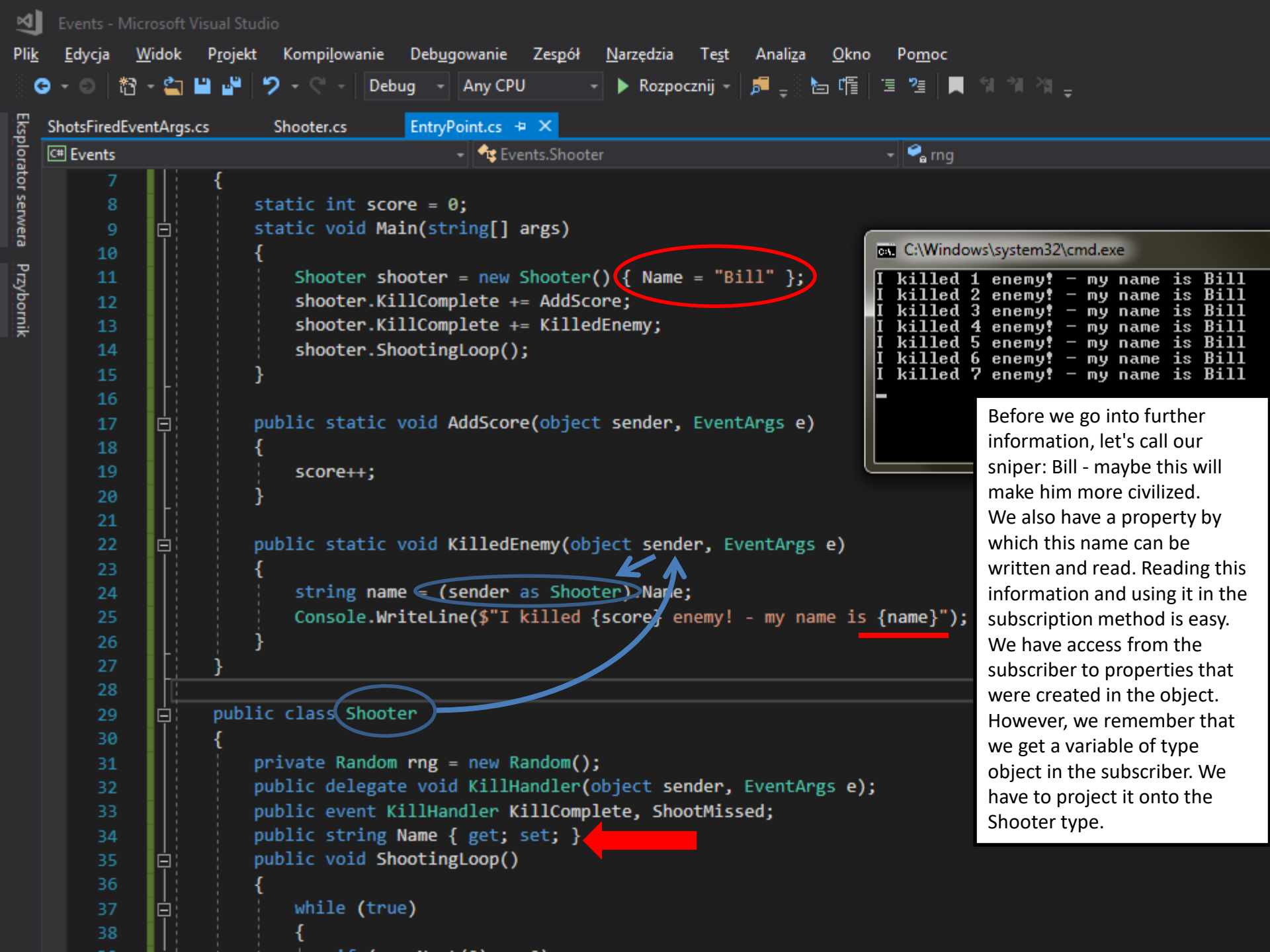

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter.KillHandler
Eksplorator serwera
Przybornik
namespace Events
{
public class EntryPoint
{
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
Therefore, let's add the AddScore method, which will count shots hit. No result counting yet. Why?

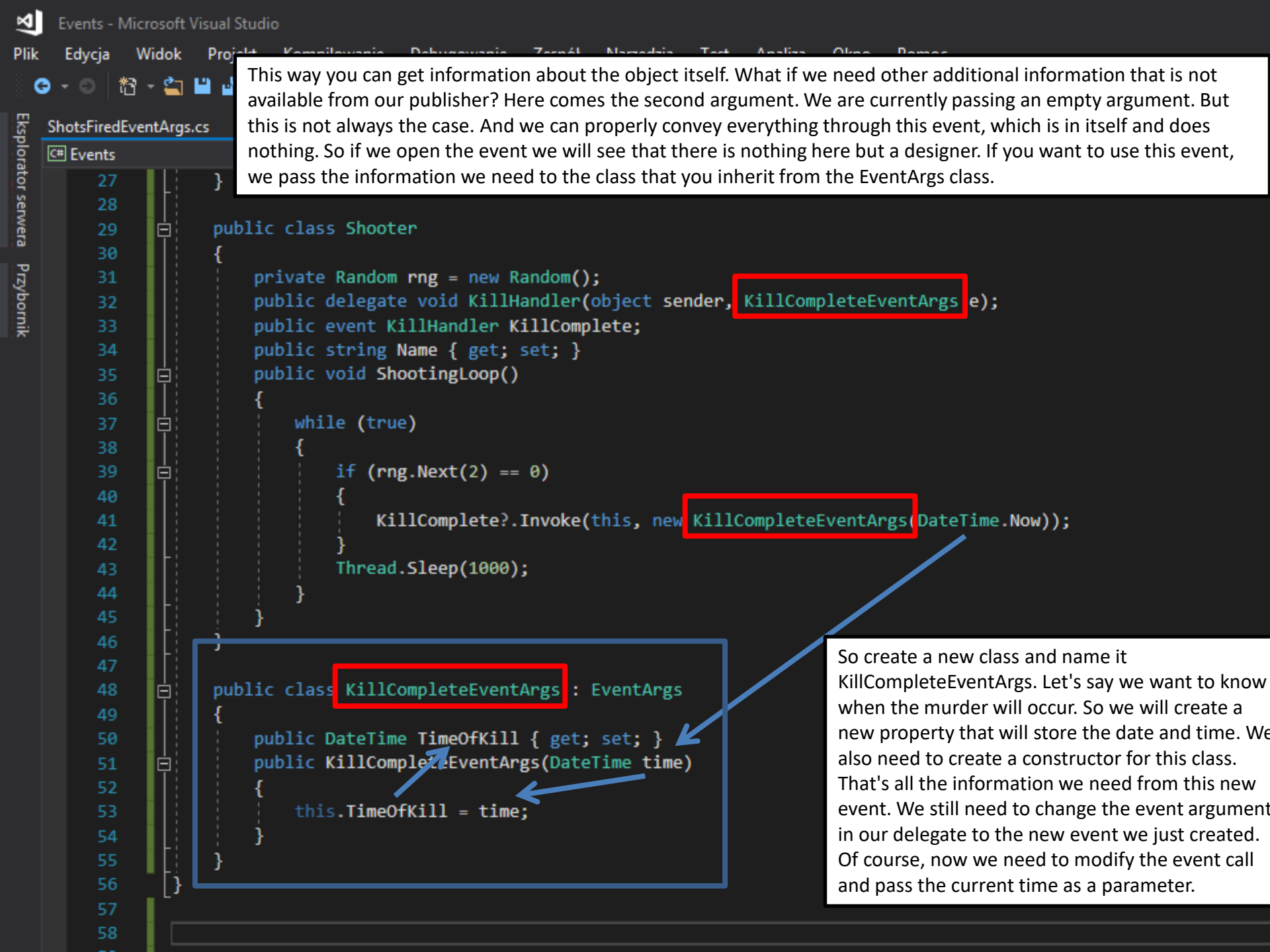

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
rng
Eksplorator serwera
Przybornik
using System;
using System.Threading;
namespace 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
Of course, AddScore should also be subscribed. The result is now calculated correctly.

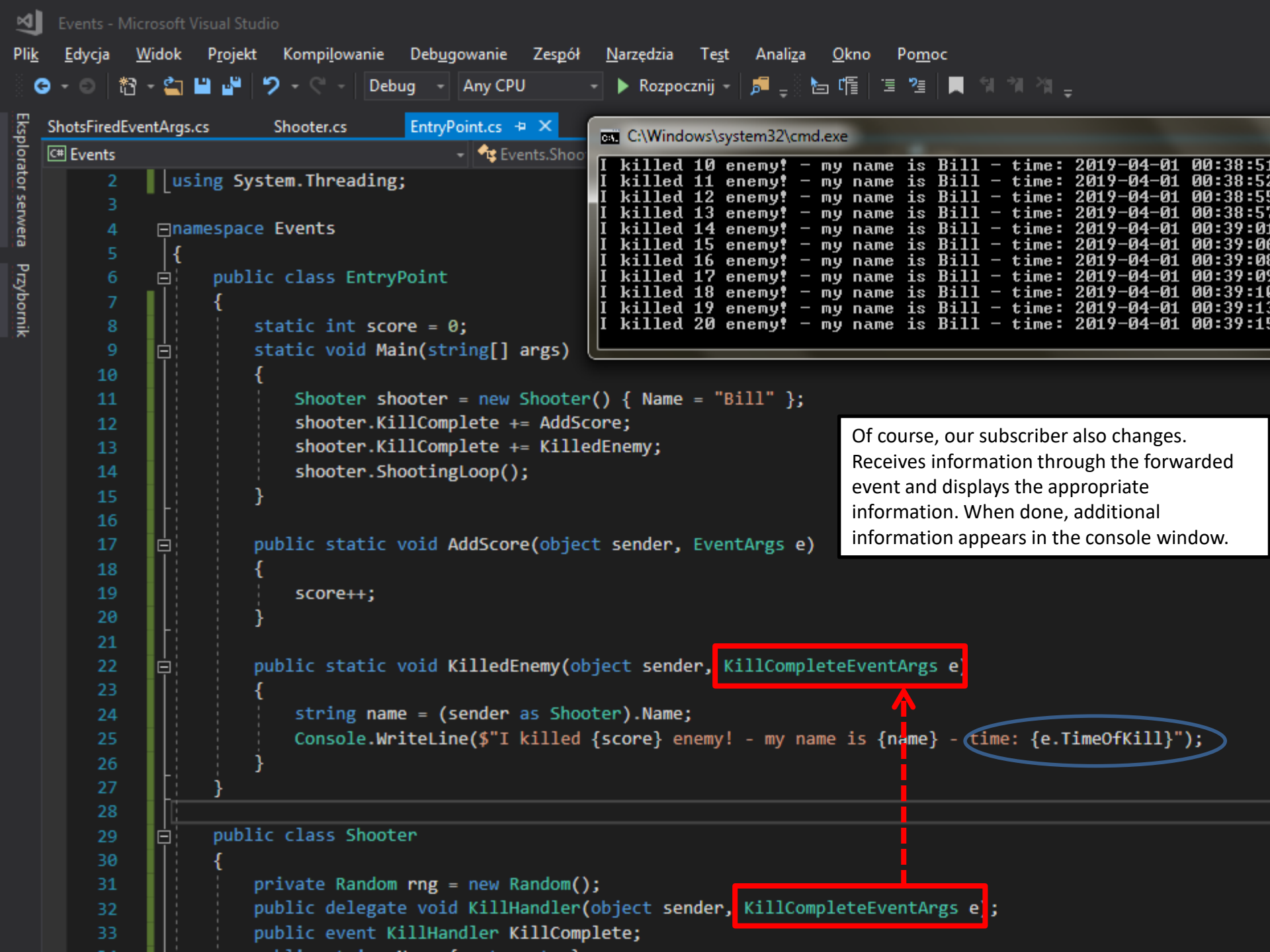

Events - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
ShotsFiredEventArgs.cs
Shooter.cs
EntryPoint.cs
Events
Events.Shooter
rng
Eksplorator serwera
Przybornik
public class EntryPoint
{
static int score = 0;
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
The last step is to subscribe to the ShootMissed event. For this purpose, the MissedEnemy method was created and connected to the event.

Passing additional information with events

Providing additional information by events.
In some cases, you may need to send information about the object that caused the event as well some other additional information about the caller. This of course will introduce some pairing of objects. Sometimes you just can't do without it.


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
Before we go into further information, let's call our sniper: Bill - maybe this will make him more civilized.
We also have a property by which this name can be written and read. Reading this information and using it in the subscription method is easy.
We have access from the subscriber to properties that were created in the object.
However, we remember that we get a variable of type object in the subscriber. We have to project it onto the Shooter type.

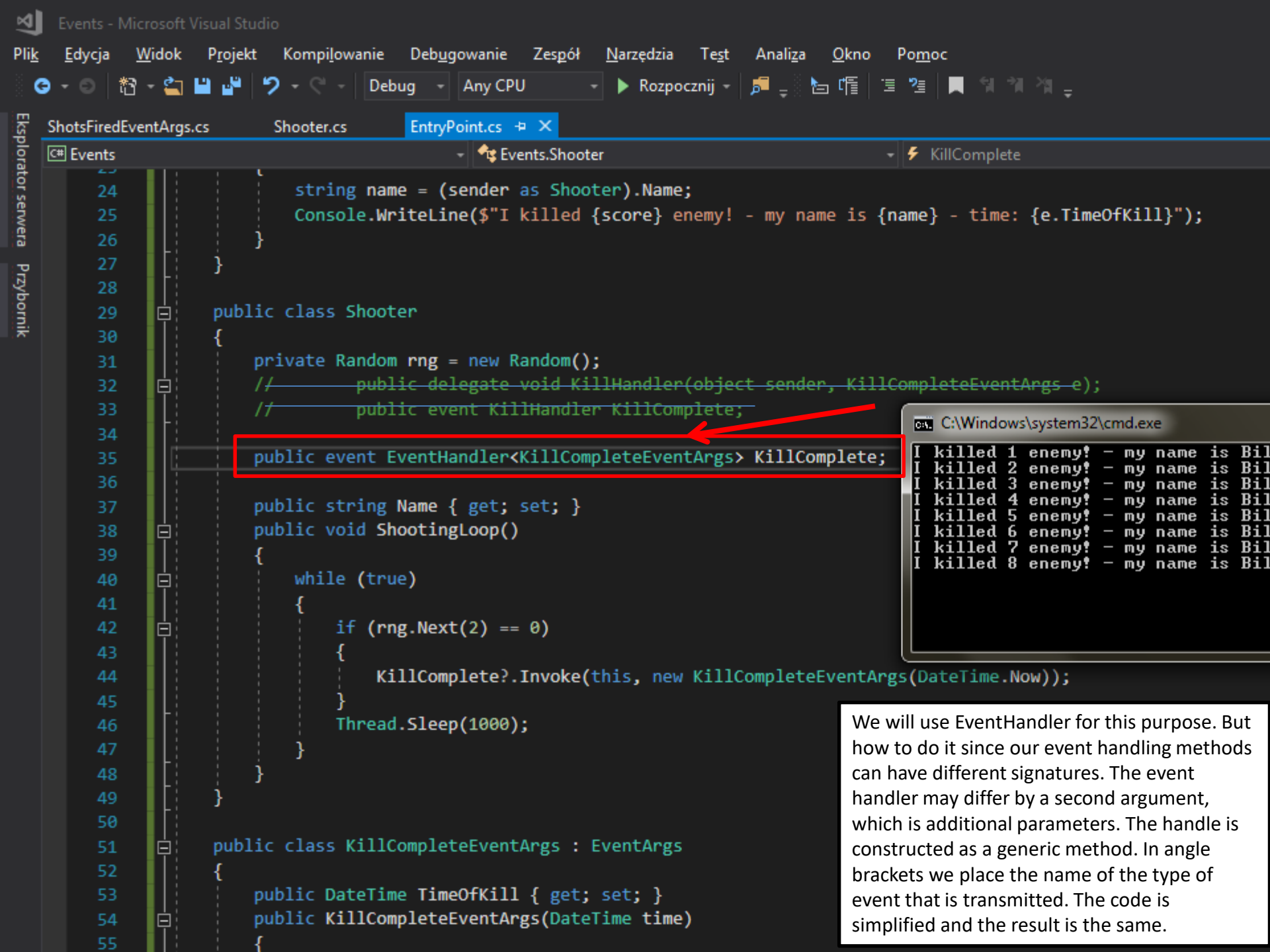

Events - Microsoft Visual Studio
Plik Edycja Widok
ShotsFiredEventArgs.cs
Events
Eksplorator serwera
Przybornik
This way you can get information about the object itself. What if we need other additional information that is not available from our publisher? Here comes the second argument. We are currently passing an empty argument. But this is not always the case. And we can properly convey everything through this event, which is in itself and does nothing. So if we open the event we will see that there is nothing here but a designer. If you want to use this event, we pass the information we need to the class that you inherit from the EventArgs class.
}
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;
}
}
So create a new class and name it KillCompleteEventArgs. Let's say we want to know when the murder will occur. So we will create a new property that will store the date and time. We also need to create a constructor for this class. That's all the information we need from this new event. We still need to change the event argument in our delegate to the new event we just created. Of course, now we need to modify the event call and pass the current time as a parameter.




```
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

The last thing I want to show you about events is that we have a shorter path in which we can achieve exactly the same.


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
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;
public event EventHandler<KillCompleteEventArgs> 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)
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
We will use EventHandler for this purpose. But how to do it since our event handling methods can have different signatures. The event handler may differ by a second argument, which is additional parameters. The handle is constructed as a generic method. In angle brackets we place the name of the type of event that is transmitted. The code is simplified and the result is the same.