0:00:00.809,0:00:03.110

Hello everyone,

0:00:03.110,0:00:05.170

and welcome

to my channel

0:00:05.170,0:00:06.209

dflessons,

0:00:06.209,0:00:07.689

dedicated to

0:00:07.689,0:00:09.949

C# programming.

0:00:09.949,0:00:12.129

As you know,

our last lesson

0:00:12.129,0:00:15.089

was dedicated to all

members of a type, except

0:00:15.089,0:00:15.909

Events.

0:00:15.909,0:00:17.809

That's why

our new lesson

0:00:17.809,0:00:20.019

is dedicated to

them.

0:00:20.019,0:00:22.160

But first of all,

we need to say that

0:00:22.160,0:00:23.910

events are built

0:00:23.910,0:00:26.300

on the C# mechanism,

called

0:00:26.300,0:00:27.589

delegates.

0:00:27.589,0:00:29.969

That's why we are

going to learn

0:00:29.969,0:00:32.210

what delegates are and

how to work with them.

0:00:32.210,0:00:34.140

Then we are going to learn

how to create events

0:00:34.140,0:00:36.590

based on delegates.

0:00:36.590,0:00:38.890

In the beginning, let's

talk about such word

0:00:38.890,0:00:41.170

as a signature

of a method.

0:00:41.170,0:00:42.580

Let's learn a definition.

0:00:42.580,0:00:45.080

A signature of a method

is a type and

0:00:45.080,0:00:45.750

an order

0:00:45.750,0:00:48.450

of its input and

output parameters.

0:00:48.450,0:00:50.589

Please, take a look

at all methods

0:00:50.589,0:00:52.800

that the Student class

consists of.

0:00:52.800,0:00:54.490

Don't pay any attention

to the realization

0:00:54.490,0:00:56.370

of these methods, as

it doesn't matter

0:00:56.370,0:00:57.110

for now.

0:00:57.110,0:00:59.310

All we need is to

declare a method.

0:00:59.310,0:01:00.740

In the following methods

0:01:00.740,0:01:03.980

Move and Grow have

the same methods,

0:01:03.980,0:01:05.160

because they have

0:01:05.160,0:01:08.040

the same return type - string

0:01:08.040,0:01:10.320

and the same

input type - int.

0:01:10.320,0:01:12.659

Methods SetInfo

and SetSchool,

0:01:12.659,0:01:13.869

on the contrary,

0:01:13.869,0:01:15.560

have different

signatures,

0:01:15.560,0:01:17.750

because even though

they have the same output

0:01:17.750,0:01:19.050

parameter - void,

0:01:19.050,0:01:21.220

and the same input types,

0:01:21.220,0:01:22.800

the order of

input parameters

0:01:22.800,0:01:24.020

is different.

0:01:24.020,0:01:26.310

SetInfo has - string first

then goes - int.

0:01:26.310,0:01:27.930

SetSchool has the opposite.

0:01:27.930,0:01:30.010

That's why we can say

that only 2 methods

0:01:30.010,0:01:32.540

in the Student class

have the same signature:

0:01:32.540,0:01:34.730

Move and Grow.

0:01:34.730,0:01:36.760

In other words,

a signature

0:01:36.760,0:01:38.670

of a method can be

seen as some layout,

0:01:38.670,0:01:40.460

that enumerates types

of all parameters.

0:01:40.460,0:01:42.460

You can see

some examples of

0:01:42.460,0:01:44.840

the signatures

on the screen.

0:01:44.840,0:01:47.210

As I've already said before,

neither method name

0:01:47.210,0:01:49.439

nor parameter name

matters

0:01:49.439,0:01:50.260

in the signatures.

0:01:50.260,0:01:52.210

Only a type

and an order

0:01:52.210,0:01:55.970

of parameters

are important.

0:01:55.970,0:01:58.030

Now let's find out

0:01:58.030,0:01:59.520

what delegate is.

0:01:59.520,0:02:02.140

First of all, a delegate, like

everything else in C# language,

0:02:02.140,0:02:03.890

is a data type.

0:02:03.890,0:02:06.280

Secondly, a delegate has

a reference type of the data

0:02:06.280,0:02:07.949

i.e. a class.

0:02:07.949,0:02:09.949

Soon, I will

show you this

0:02:09.949,0:02:11.389

using MSIL.

0:02:11.389,0:02:13.260

And, finally, a delegate

0:02:13.260,0:02:15.139

is a data type,

0:02:15.139,0:02:18.609

that can store a

link to a method,

0:02:18.609,0:02:20.739

that has the same

signature

0:02:20.739,0:02:22.509

as the signature

declared in a

0:02:22.509,0:02:23.729

delegate.

0:02:23.729,0:02:25.059

Please, note that

0:02:25.059,0:02:26.820

in our delegate named

0:02:26.820,0:02:28.550

CountDelegate

has been declared

0:02:28.550,0:02:31.310

a signature of the

following method:

0:02:31.310,0:02:34.039

return type - int,

0:02:34.039,0:02:36.759

parameter type (single parameter) - string,

0:02:36.759,0:02:39.379

Both method name

and parameter name

0:02:39.379,0:02:42.909

are not important.

0:02:42.909,0:02:45.419

2 methods in

the StringHelper class

0:02:45.419,0:02:47.749

corresponds to this signature:

0:02:47.749,0:02:48.810

GetCount

0:02:48.810,0:02:51.059

and GetCountSymbolA.

0:02:51.059,0:02:52.959

The GetCountSymbol method

0:02:52.959,0:02:55.579

doesn't fit to this

signature, as

0:02:55.579,0:02:57.109

its input parameters

uses

0:02:57.109,0:03:00.249

two parameters:

- string and - char.

0:03:00.249,0:03:02.619

Let's make sure that

a delegate is

0:03:02.619,0:03:04.939

just a simple

0:03:04.939,0:03:05.979

reference type.

0:03:05.979,0:03:08.289

Let's run disassembler

0:03:08.289,0:03:10.059

and open

0:03:10.059,0:03:12.139

our assembly.

0:03:12.139,0:03:15.079

Now choose CountDelegate.

0:03:15.079,0:03:16.869

You can see that

0:03:16.869,0:03:18.829

the keyword "class"

figures in the declaration

0:03:18.829,0:03:21.350

of this type.

0:03:21.350,0:03:23.770

This means, that

our CountDelegate data type

0:03:23.770,0:03:28.009

appears to be a class.

0:03:28.009,0:03:30.610

And due to CountDelegate

being a class,

0:03:30.610,0:03:32.930

we can declare

any variable

0:03:32.930,0:03:34.299

of this type.

0:03:34.299,0:03:35.530

You can see

0:03:35.530,0:03:37.630

how we declare delegates

d1 and d2

0:03:37.630,0:03:40.430

of the CountDelegate type

0:03:40.430,0:03:42.159

in the following example

on the screen.

0:03:42.159,0:03:43.850

And the values of these

0:03:43.850,0:03:44.669

variables are

0:03:44.669,0:03:46.620

the links on the

methods

0:03:46.620,0:03:47.830

which are corresponding to

0:03:47.830,0:03:50.779

the signature, that has

been declared on a delegate.

0:03:50.779,0:03:52.290

As you can see,

0:03:52.290,0:03:54.939

methods GetCount and GetCountSymbolA

0:03:54.939,0:03:56.520

have the same signatures,

0:03:56.520,0:03:58.440

that's why we can

assign a link

0:03:58.440,0:04:00.049

to the variables.

0:04:00.049,0:04:02.949

But the method GetCountSymbol

0:04:02.949,0:04:05.329

has a different signature

and that's why compiler

0:04:05.329,0:04:08.649

shows error.

0:04:08.649,0:04:11.139

Now let's learn

how to use our

0:04:11.139,0:04:14.019

variables. First,

let's declare

0:04:14.019,0:04:15.109

the TestDelegate method

0:04:15.109,0:04:16.680

with its first parameter

0:04:16.680,0:04:18.319

getting a link

0:04:18.319,0:04:19.620

to a method under test,

0:04:19.620,0:04:22.650

and the second parameter is linked

with the string under test.

0:04:22.650,0:04:24.900

After that. let's declare

a string under test: После чего объявим тестируемую строку:

0:04:24.900,0:04:26.189

= lamp.

0:04:26.189,0:04:28.930

Then let's set up

2 calls

0:04:28.930,0:04:31.820

In the first call, we invoke

0:04:31.820,0:04:32.530

the TestDelegate method

0:04:32.530,0:04:34.610

and send the reference to it

0:04:34.610,0:04:36.169

on the GetCount method

0:04:36.169,0:04:37.430

In the second invoke,

0:04:37.430,0:04:39.389

we invoke TestDelegate, sending

0:04:39.389,0:04:42.309

a reference to it on the GetSymbolA method.

0:04:42.309,0:04:44.559

What happens in our

0:04:44.559,0:04:46.340

TestDelegate method?

0:04:46.340,0:04:48.030

Compiler

0:04:48.030,0:04:50.249

gets a reference to a

method through

0:04:50.249,0:04:53.110

the Method variable

0:04:53.110,0:04:55.840

and invokes it by

the signature.

0:04:55.840,0:04:57.729

Which means, that there is

0:04:57.729,0:04:59.650

just one input parameter

a type - string

0:04:59.650,0:05:02.169

and the return type value is - int.

0:05:02.169,0:05:04.219

Let's run our

example and see

0:05:04.219,0:05:06.949

what's gonna happen.

0:05:06.949,0:05:09.419

We'll see exactly what

we expect.

0:05:09.419,0:05:11.460

The total number of the

characters: 4, number

0:05:11.460,0:05:14.469

of the characters "a": 1.

0:05:14.469,0:05:16.389

Students usually

0:05:16.389,0:05:18.310

have a question

by now:

0:05:18.310,0:05:20.630

why do we need

these delegates?

0:05:20.630,0:05:23.250

How and where can we use

we use them

0:05:23.250,0:05:24.490

Here's a simple example.

0:05:24.490,0:05:27.620

Let's declare the Move method

in the Student class.

0:05:27.620,0:05:29.729

This method is

going to "translocate"

0:05:29.729,0:05:30.830

our student.

0:05:30.830,0:05:32.319

And its input value

is the amount of

0:05:32.319,0:05:33.169

kilometres.

0:05:33.169,0:05:35.059

Let's organize a cycle

inside a class

0:05:35.059,0:05:36.929

showing a message

about traversed

0:05:36.929,0:05:38.349

kilometres

0:05:38.349,0:05:39.839

within this cycle.

0:05:39.839,0:05:41.789

That's how in looks

0:05:41.789,0:05:46.270

in action.

0:05:46.270,0:05:47.240

You'd say:

0:05:47.240,0:05:49.479

that's OK, everything

works just fine.

0:05:49.479,0:05:50.639

But let's think.

0:05:50.639,0:05:51.639

Let's assume, that

0:05:51.639,0:05:53.919

our Student class is

located on some

0:05:53.919,0:05:56.129

server. In this case,

0:05:56.129,0:05:57.590

we'd like to

have an ability to возможность

0:05:57.590,0:05:59.469

access this class

and get this

0:05:59.469,0:06:00.840

message by the internet

0:06:00.840,0:06:03.150

instead of console.

0:06:03.150,0:06:05.240

Or let's imagine, we are building

a non-console application -

0:06:05.240,0:06:07.280

a Windows application

0:06:07.280,0:06:09.710

with a graphic interface.

In this case, we'd like to

0:06:09.710,0:06:10.689

see this message

0:06:10.689,0:06:12.529

in some pretty

0:06:12.529,0:06:14.080

graphic window.

0:06:14.080,0:06:16.830

The whole point is that:

we need to organize

0:06:16.830,0:06:19.389

the Move method

in order to

0:06:19.389,0:06:21.639

get messages

from it independently

0:06:21.639,0:06:24.169

from the way of

representing

0:06:24.169,0:06:26.110

this message in

the interface.

0:06:26.110,0:06:28.180

Let's try this

on practise, using

0:06:28.180,0:06:31.080

the delegates.

0:06:31.080,0:06:33.300

First of all, let's

declare a delegate.

0:06:33.300,0:06:34.930

We'll need a method, Нам требуется метод,

0:06:34.930,0:06:36.360

that will receive one который будет

string parameter - принимать один

0:06:36.360,0:06:39.199

our message,

and the algorithm

0:06:39.199,0:06:40.660

of what to do

0:06:40.660,0:06:42.250

with it, without

returning anything.

0:06:42.250,0:06:43.289

So,

0:06:43.289,0:06:45.599

a signature of our

delegate: returning type - void?

0:06:45.599,0:06:47.839

input parameter

0:06:47.839,0:06:49.309

type - string.

0:06:49.309,0:06:50.270

Then

0:06:50.270,0:06:52.279

let's pass to our

Move method

0:06:52.279,0:06:54.229

the variable of this

0:06:54.229,0:06:54.979

delegate.

0:06:54.979,0:06:56.949

In other words,

a reference to

0:06:56.949,0:06:57.639

a method

0:06:57.639,0:07:01.349

with a congruous

signature.

0:07:01.349,0:07:07.020

0:07:07.020,0:07:08.389

And now,

0:07:08.389,0:07:10.870

let's replace an output invoke

to console of our

0:07:10.870,0:07:11.659

message

0:07:11.659,0:07:18.659

by a translation of this

message into our method.

0:07:26.300,0:07:28.869

What can we see now?

0:07:28.869,0:07:31.590

Instead of outputting

it into console,

0:07:31.590,0:07:33.380

the Move method

0:07:33.380,0:07:34.490

translates

its messages to

0:07:34.490,0:07:36.110

any other methods.

0:07:36.110,0:07:38.879

It doesn't even know

which one precisely.

0:07:38.879,0:07:39.340

A reference to that method

0:07:39.340,0:07:41.030

he receives from the

source code

0:07:41.030,0:07:42.009

from the outside.

0:07:42.009,0:07:43.569

Therefore, we've

0:07:43.569,0:07:45.249

completely left behind

0:07:45.249,0:07:48.240

the way of outputting

our information to the

0:07:48.240,0:07:49.569

user interface.

0:07:49.569,0:07:51.629

So how can we

use this method

0:07:51.629,0:07:52.710

now?

0:07:52.710,0:07:55.710

That's how the invoke of our

method from the previous

0:07:55.710,0:07:57.330

example looked like.

0:07:57.330,0:07:59.229

Now

0:07:59.229,0:08:00.639

let's do this.

0:08:00.639,0:08:01.419

First of all,

0:08:01.419,0:08:04.069

we've declared

the Show method,

0:08:04.069,0:08:05.999

that was suitable

to our delegate

0:08:05.999,0:08:08.740

signature.

Now we need to

0:08:08.740,0:08:15.740

assign a reference.

Let's do this.

0:08:18.110,0:08:21.139

Great, we've assigned

a reference to our method

0:08:21.139,0:08:23.659

variable. Then

let's pass

0:08:23.659,0:08:30.659

this variable as a second

parameter tp the Move method.

0:08:32.450,0:08:35.590

Great. Run. Look.

0:08:35.590,0:08:38.510

We see pretty much

the same.

0:08:38.510,0:08:40.469

But this time,

look here.

0:08:40.469,0:08:42.969

The Show method

allows you to change

0:08:42.969,0:08:45.240

output type to

0:08:45.240,0:08:47.700

any other way now:

0:08:47.700,0:08:49.759

- by web-interface;

0:08:49.759,0:08:51.609

- by Windows application;

0:08:51.609,0:08:54.240

and so on. That's why

you need the delegates -

0:08:54.240,0:08:56.250

to built up a higher

levels of

0:08:56.250,0:08:58.030

the abstraction.

0:08:58.030,0:08:59.640

Now let's briefly

talk about such

0:08:59.640,0:09:02.490

thing as a generic

delegate. Have a look:

0:09:02.490,0:09:04.560

we have a delegate

that has a reference to

0:09:04.560,0:09:06.690

a method with a signature - void (- string).

0:09:06.690,0:09:09.030

It may happen,

that in some

0:09:09.030,0:09:11.550

other place inside

our code we will have

0:09:11.550,0:09:13.019

exactly the same

situation,

0:09:13.019,0:09:15.209

and we'd need one

more delegate, that

0:09:15.209,0:09:17.850

has a reference to another

method - void (- string).

0:09:17.850,0:09:20.540

We might need a third delegate,

a forth and so on.

0:09:20.540,0:09:22.820

In theory, we can

create as many delegates,

0:09:22.820,0:09:25.130

as we want

with the same

0:09:25.130,0:09:26.490

signature.

0:09:26.490,0:09:28.620

The only difference

between them

0:09:28.620,0:09:31.700

will be their names and

the input parameters.

0:09:31.700,0:09:33.390

You might have

a question:

0:09:33.390,0:09:36.620

why do we need so many

equal delegates?

0:09:36.620,0:09:38.850

Microsoft developers

had thought the same

0:09:38.850,0:09:40.560

and created

a structure of

0:09:40.560,0:09:42.960

a generic delegates.

0:09:42.960,0:09:45.740

Let's look at two

types of these delegates.

0:09:45.740,0:09:47.959

First of all, it's the Action delegate.

0:09:47.959,0:09:50.059

Input parameters types

are listed in

0:09:50.059,0:09:51.260

chevrons.

0:09:51.260,0:09:53.910

This delegate can have

a reference to any

0:09:53.910,0:09:56.800

method without

output parameter, and

0:09:56.800,0:09:58.720

his input parameter types

0:09:58.720,0:10:01.630

are T1, T2, ..., Tn

and so on.

0:10:01.630,0:10:03.930

Letters T1 and T2 represent

0:10:03.930,0:10:05.090

a specific type.

0:10:05.090,0:10:07.570

Here's another example:

0:10:07.570,0:10:10.810

Action <string> - it's a delegate,

that fits into the structure

0:10:10.810,0:10:12.480

we saw in the previous

0:10:12.480,0:10:13.200

example.

0:10:13.200,0:10:15.510

It has a reference to a method

that has no output

0:10:15.510,0:10:18.060

parameter and the only one

input parameter - string.

0:10:18.060,0:10:19.620

Action<int, bool>

0:10:19.620,0:10:21.290

has a reference to a method

with zero output

0:10:21.290,0:10:22.590

parameters and with

two input

0:10:22.590,0:10:24.629

parameters with types - int and - bool.

0:10:24.629,0:10:26.590

This is the first and

the most popular

0:10:26.590,0:10:28.280

generic delegate.

0:10:28.280,0:10:29.449

The second delegate is

0:10:29.449,0:10:32.330

Func<T1, T2... T out>.

0:10:32.330,0:10:34.060

Unlike the previous

delegate

0:10:34.060,0:10:36.330

it has a reference to

a method with

0:10:36.330,0:10:37.160

an output parameter.

0:10:37.160,0:10:38.500

A type of an output

parameter

0:10:38.500,0:10:40.360

is defined by

the last element

0:10:40.360,0:10:41.470

of a sequence.

Which means, that

0:10:41.470,0:10:43.470

- T out is

the last type in

0:10:43.470,0:10:45.970

chevrons. It

defines what type

0:10:45.970,0:10:48.180

is going to have an output

0:10:48.180,0:10:49.930

parameter of our method.

0:10:49.930,0:10:51.670

One more example:

0:10:51.670,0:10:53.130

Func<string, string>

0:10:53.130,0:10:55.100

has a reference to a method

with an output

0:10:55.100,0:10:57.110

parameter type - string

and one input

0:10:57.110,0:10:58.580

parameter with a type - string too.

0:10:58.580,0:11:01.450

Func<ing, string, bool>

has a reference to a method

0:11:01.450,0:11:03.210

with an output

parameter type - bool

0:11:03.210,0:11:06.760

and two input parameters of

a type - int and - string.

0:11:06.760,0:11:09.100

Let's try to replace

our delegate with

0:11:09.100,0:11:14.260

a generic delegate.

It won't take much time.

0:11:14.260,0:11:20.220

We've declared it here,

0:11:20.220,0:11:24.460

and now we are changing

its type here.

0:11:24.460,0:11:25.930

That's all,

our application

0:11:25.930,0:11:28.580

has been compiled, and, as

you can see, it's working!

0:11:28.580,0:11:29.860

What are the pros of

0:11:29.860,0:11:31.190

the generic delegates?

0:11:31.190,0:11:33.980

- We don't need to

declare any extra

0:11:33.980,0:11:35.430

delegates anymore;

0:11:35.430,0:11:37.190

- Our code has become shorter now.

0:11:37.190,0:11:38.840

Of course, there is

no pros

0:11:38.840,0:11:39.630

without cons.

0:11:39.630,0:11:40.570

What are the cons?

0:11:40.570,0:11:42.870

Here, let's imagine,

that we might want

0:11:42.870,0:11:45.280

to use our delegate.

0:11:45.280,0:11:48.540

But we've lost

parameter name.

0:11:48.540,0:11:50.970

Now it's marked as - obj.

0:11:50.970,0:11:53.300

Parameter names

0:11:53.300,0:11:55.570

don't influence any functions.

0:11:55.570,0:11:56.910

Compiler doesn't care

0:11:56.910,0:11:58.949

about a parameter name.

0:11:58.949,0:12:01.930

It can be -obj or whatever.

0:12:01.930,0:12:03.750

But a name influences

a lot when we talk

0:12:03.750,0:12:05.920

about code's readability.

0:12:05.920,0:12:07.980

For comparison, let's look

at our previous example.

0:12:07.980,0:12:08.989

Have a look

0:12:08.989,0:12:10.889

here: when we

try to call

0:12:10.889,0:12:11.480

a method

0:12:11.480,0:12:13.069

with some parameters,

0:12:13.069,0:12:15.629

we can see a clue

about a parameter name.

0:12:15.629,0:12:17.079

Since now we address

not to any anonymous

0:12:17.079,0:12:19.270

parameter, but to a specified

parameter - message.

0:12:19.270,0:12:21.150

Of course, it's much

more informative than

0:12:21.150,0:12:22.590

a plain - obj.

0:12:22.590,0:12:24.490

Now let's see

how our delegate

0:12:24.490,0:12:26.010

will be transformed

to an event,

0:12:26.010,0:12:28.490

step-by step.

Have a look, please.

0:12:28.490,0:12:30.610

Our Move method

has a second parameter

0:12:30.610,0:12:32.640

as a reference to

another method.

0:12:32.640,0:12:35.290

Sometimes, it's not

comfortable. Imagine, we

0:12:35.290,0:12:37.510

want to invoke the Move

method without getting any

0:12:37.510,0:12:38.830

messages from it.

0:12:38.830,0:12:41.210

In this case, the second

parameter will be

0:12:41.210,0:12:41.940

useless.

0:12:41.940,0:12:44.230

We want to create

an optional subscription

0:12:44.230,0:12:45.860

to these messages.

We can subscribe,

0:12:45.860,0:12:48.010

if we want. Or we can

not subscribe either.

0:12:48.010,0:12:50.350

We could do it

this way:

0:12:50.350,0:12:52.460

- to create a Moving property;

0:12:52.460,0:12:55.840

- to invoke this property here;

0:12:55.840,0:13:02.060

- to delete this parameter here.

0:13:02.060,0:13:04.290

In this case, an invoke

of our method

0:13:04.290,0:13:05.050

looks like this:

0:13:05.050,0:13:07.339

after creating

an instance of a class,

0:13:07.339,0:13:09.509

we assign a reference to

our method for

0:13:09.509,0:13:10.650

a Moving property,

0:13:10.650,0:13:12.050

then - we call

0:13:12.050,0:13:12.980

the Move method.

0:13:12.980,0:13:14.050

Look here, please,

0:13:14.050,0:13:16.080

it's all the same.

0:13:16.080,0:13:17.500

But at the same

time, we have

0:13:17.500,0:13:18.920

an extra option -

0:13:18.920,0:13:20.820

we can subscribe to the

event, if we want;

0:13:20.820,0:13:22.649

or we can not subscribe,

if we don't want.

0:13:22.649,0:13:24.570

Ok, let's imagine, we

don't want it.

0:13:24.570,0:13:25.380

Now

0:13:25.380,0:13:26.880

running our

0:13:26.880,0:13:28.780

application will cause

an error.

0:13:28.780,0:13:30.940

Why did that happen?

Because we didn't

0:13:30.940,0:13:33.600

assign any value to

the Move method. See?

0:13:33.600,0:13:34.670

It have - null.

0:13:34.670,0:13:36.569

Obviously, calling - null

0:13:36.569,0:13:38.000

will cause an error.

0:13:38.000,0:13:40.830

We need to make a call

to our property

0:13:40.830,0:13:42.720

an optional feature.

0:13:42.720,0:13:44.290

In order to do this, let's

0:13:44.290,0:13:46.220

add an additional

0:13:46.220,0:13:52.900

condition before the call.

0:13:52.900,0:13:54.069

Therefore,

0:13:54.069,0:13:55.459

we will call our property

0:13:55.459,0:13:58.189

only in case, when

it actually has

0:13:58.189,0:14:00.429

a reference to

any method.

0:14:00.429,0:14:02.270

What do we have now?

0:14:02.270,0:14:04.470

In case #1, we

don't subscribe to

0:14:04.470,0:14:05.480

a property.

0:14:05.480,0:14:06.950

We don't receive

any messages,

0:14:06.950,0:14:08.820

we only call the Move method.

0:14:08.820,0:14:10.600

In case #2, we

do subscribe to

0:14:10.600,0:14:13.760

a message, we do

0:14:13.760,0:14:16.380

receive messages from the Move method.

0:14:16.380,0:14:18.460

Now there's only one step

left for a fuller

0:14:18.460,0:14:20.820

understanding of what

an event is.

0:14:20.820,0:14:22.820

Let's replace our

properties to this

0:14:22.820,0:14:25.580

construct. This is what's

called an event.

0:14:25.580,0:14:27.750

- public - it's an

access modifier;

0:14:27.750,0:14:29.810

- event - it is

a keyword indicating

0:14:29.810,0:14:31.560

that we do use

an event.

0:14:31.560,0:14:33.769

EventHandler - it is our delegate,

0:14:33.769,0:14:36.129

i.e. a reference to

our method that is going to

0:14:36.129,0:14:37.649

handle our event.

0:14:37.649,0:14:39.480

I'll explain, why it's

called EventHandler

0:14:39.480,0:14:41.999

later. And finally,

EventName - it is

0:14:41.999,0:14:44.200

a name of our event.

0:14:44.200,0:14:47.330

The following form is

what's been called as

0:14:47.330,0:14:48.750

"an event without parameters",

0:14:48.750,0:14:51.010

i.e. when we call this

event, we are not такое событие, мы не

0:14:51.010,0:14:52.630

able to send any

parameters

0:14:52.630,0:14:53.590

to it.

0:14:53.590,0:14:56.200

The second form of an event,

this one,

0:14:56.200,0:14:58.690

declares an event with

a parameter. The difference

0:14:58.690,0:15:01.350

between them is that now

we have types in chevrons:

0:15:01.350,0:15:02.800

EventArgs.

0:15:02.800,0:15:05.220

This is a class,

where we are going to

0:15:05.220,0:15:07.260

describe our passing

0:15:07.260,0:15:08.140

parameters to.

0:15:08.140,0:15:09.370

Now, with some

help from

0:15:09.370,0:15:10.319

these forms,

0:15:10.319,0:15:13.970

let's change our

properties to the events.

0:15:13.970,0:15:18.330

Ok, we are changing the

properties,

0:15:18.330,0:15:20.840

making everything

correspond to

0:15:20.840,0:15:23.180

an event form.

0:15:23.180,0:15:28.720

0:15:28.720,0:15:32.060

0:15:32.060,0:15:33.950

That's how we can call

0:15:33.950,0:15:35.810

our event.

0:15:35.810,0:15:36.640

0:15:36.640,0:15:39.160

It's mostly alike

a property, that's been

0:15:39.160,0:15:40.390

declared earlier.

0:15:40.390,0:15:42.650

The main difference is that,

firstly, we are sending

0:15:42.650,0:15:44.620

an additional

parameter - this,

0:15:44.620,0:15:47.380

that's indicating to our

current instance of a class;

0:15:47.380,0:15:50.459

and we are passing

a string using

0:15:50.459,0:15:52.160

an additional MovingEventArgs class

0:15:52.160,0:15:54.170

instead of a direct passing.

0:15:54.170,0:15:56.680

We'll talk about it later.

And now one more example

0:15:56.680,0:15:58.370

of how to use

our events:

0:15:58.370,0:16:00.619

We create the Student class.

0:16:00.619,0:16:02.539

Then we subscribe to

0:16:02.539,0:16:05.410

an event, using

the following form -

0:16:05.410,0:16:07.540

here's our Moving event; it's marked

0:16:07.540,0:16:08.630

by an arrow.

0:16:08.630,0:16:10.590

Then we press +=

0:16:10.590,0:16:13.570

and VisualStudio offers

0:16:13.570,0:16:15.860

to create some method,

0:16:15.860,0:16:17.330

that is going

to respond to

0:16:17.330,0:16:18.359

a delegate,

0:16:18.359,0:16:20.090

that has been

0:16:20.090,0:16:22.070

declared in the event.

0:16:22.070,0:16:23.859

Now, let's see

this in details.

0:16:23.859,0:16:26.129

We can delete it

right away, not

0:16:26.129,0:16:28.190

going to use it.

0:16:28.190,0:16:30.420

So, our event has a

delegate, represented by

0:16:30.420,0:16:32.730

this construction:

EventHandler and MovingEventArgs

0:16:32.730,0:16:34.470

in angle brackets.

0:16:34.470,0:16:37.399

MovingEventArgs - it is a class

0:16:37.399,0:16:38.879

I've created before.

0:16:38.879,0:16:41.809

It describes the passing

of all the necessary

0:16:41.809,0:16:43.950

parameters in the event.

0:16:43.950,0:16:46.350

All classes we are

about to use inside

0:16:46.350,0:16:48.380

EventHandler must

be inherited

0:16:48.380,0:16:50.390

from an EventArgs base class

0:16:50.390,0:16:52.310

We will discuss the

inheritance later.

0:16:52.310,0:16:54.570

For now, let's

fill in a form

0:16:54.570,0:16:56.510

to declare an

additional class,

0:16:56.510,0:16:58.490

that is going to

have our

0:16:58.490,0:16:59.620

parameters.

0:16:59.620,0:17:00.489

After that,

0:17:00.489,0:17:02.549

we describe all

passing parameters

0:17:02.549,0:17:04.179

using a class

constructor.

0:17:04.179,0:17:06.319

We need to create

a property to each

0:17:06.319,0:17:07.169

parameter.

0:17:07.169,0:17:09.579

It will be great, if

an accessor - get of this

0:17:09.579,0:17:11.589

property has

a public modifier,

0:17:11.589,0:17:13.839

and an accessor - set

has a private modifier,

0:17:13.839,0:17:15.789

in order to give us an

ability to assign

0:17:15.789,0:17:17.309

this properties

only inside

0:17:17.309,0:17:19.589

our additional class

How does a passing of

0:17:19.589,0:17:21.409

the parameters look like?

0:17:21.409,0:17:23.259

We pass parameters

to a constructor

0:17:23.259,0:17:24.830

while calling an event;

0:17:24.830,0:17:26.539

we receive these

0:17:26.539,0:17:28.269

parameters through

0:17:28.269,0:17:29.999

the corresponding

properties in

0:17:29.999,0:17:31.649

a class handler, in

a method handler, to be precise.

0:17:31.649,0:17:34.299

Here, we've passed our string

0:17:34.299,0:17:35.800

as a first parameter

of our additional

0:17:35.800,0:17:36.649

0:17:36.649,0:17:38.850

MovingEventArgs class

0:17:38.850,0:17:41.090

in a Student class.

0:17:41.090,0:17:43.580

Let's clarify, that

we'll call a class

0:17:43.580,0:17:45.490

that generates the events

as a class -

0:17:45.490,0:17:46.370

- publisher.

0:17:46.370,0:17:48.800

In our case,

the Student class is

0:17:48.800,0:17:50.400

a class-publisher.

0:17:50.400,0:17:52.210

And we'll call the classes

that subscribe to

0:17:52.210,0:17:54.180

the events of

a class-publisher

0:17:54.180,0:17:56.740

as the classes -

subscribers.

0:17:56.740,0:17:58.700

So, we've defined

a method for

0:17:58.700,0:18:00.460

an event, each time

it occurs

0:18:00.460,0:18:03.040

in our class -

subscriber.

0:18:03.040,0:18:05.690

We receive all event

parameters through

0:18:05.690,0:18:07.210

- a variable - e;

0:18:07.210,0:18:08.920

- an argument - e,

0:18:08.920,0:18:11.340

that has the MovingEventArgs type

0:18:11.340,0:18:12.039

Have a look, please.

0:18:12.039,0:18:14.789

Here's our message.

That's how we

0:18:14.789,0:18:17.030

can reduce our code

0:18:17.030,0:18:18.810

to the structure we've

discussed at the

0:18:18.810,0:18:21.750

very beginning.

0:18:21.750,0:18:23.260

Let's run it

0:18:23.260,0:18:25.100

and make sure

our code's working

0:18:25.100,0:18:28.840

exactly how we want

it to work.

0:18:28.840,0:18:31.169

We've just discussed

a standard form

0:18:31.169,0:18:34.030

of an event with

a standard delegate.

0:18:34.030,0:18:36.540

A standard form of an event

looks like this:

0:18:36.540,0:18:40.010

The 1st parameter of a delegate

is always a type

0:18:40.010,0:18:40.610

- object,

0:18:40.610,0:18:42.820

that has a reference to

a class-publisher.

0:18:42.820,0:18:44.700

In our case, it's

the Student class.

0:18:44.700,0:18:45.889

The 2nd parameter is

0:18:45.889,0:18:48.639

a variable of a type

that has been inherited

0:18:48.639,0:18:51.179

from EventArgs,

where we pass all the

0:18:51.179,0:18:53.470

necessary parameters to.

0:18:53.470,0:18:55.809

You might think

that this form is

0:18:55.809,0:18:57.809

kind of hard for you,

but it is

0:18:57.809,0:18:58.990

a common form.

0:18:58.990,0:19:02.370

You can use

an abstract form of

0:19:02.370,0:19:04.900

a delegate

this way:

0:19:04.900,0:19:06.690

If you don't want

0:19:06.690,0:19:07.810

to send

0:19:07.810,0:19:09.750

a reference to

a class-publisher,

0:19:09.750,0:19:11.300

or don't want to

create an additional

0:19:11.300,0:19:12.769

class MovingEventArgs;

0:19:12.769,0:19:14.649

or you want just to

create an event,

0:19:14.649,0:19:16.799

then you have a right to

replace your delegate

0:19:16.799,0:19:18.769

to a generic delegate,

that we'd discussed

0:19:18.769,0:19:19.580

before,

0:19:19.580,0:19:22.070

i.e. Action <string>.

0:19:22.070,0:19:24.760

It doesn't require

any extra efforts,

0:19:24.760,0:19:27.580

and it receives only

0:19:27.580,0:19:29.930

our string,

0:19:29.930,0:19:31.950

just like we've seen

in our previous

0:19:31.950,0:19:32.590

examples.

0:19:32.590,0:19:34.659

Of course, in this case,

we will need to modify

0:19:34.659,0:19:37.409

a method-receiver

as well. See this?

0:19:37.409,0:19:39.410

A compiler shows that

0:19:39.410,0:19:41.210

a signature of a method

doesn't correspond to

0:19:41.210,0:19:42.890

the declared event.

0:19:42.890,0:19:44.940

It order to make

it fit, we need to

0:19:44.940,0:19:47.900

simply change it to

0:19:47.900,0:19:49.309

a parameter - string

0:19:49.309,0:19:52.280

and pass it to a console.

0:19:52.280,0:19:54.360

Once again,

we can make sure

0:19:54.360,0:19:57.000

that everything is

copying and working fine.

0:19:57.000,0:19:58.110

Again:

0:19:58.110,0:19:58.830

0:19:58.830,0:20:01.179

this form is easier,

0:20:01.179,0:20:01.920

no doubt,

0:20:01.920,0:20:06.160

but at the same time,

this form is less readable.

0:20:06.160,0:20:08.050

In here, when

we try to

0:20:08.050,0:20:10.340

pass a parameter

0:20:10.340,0:20:11.440

to an event,

0:20:11.440,0:20:13.330

instead of using a parameter

name, we get

0:20:13.330,0:20:14.680

- obj again.

0:20:14.680,0:20:16.660

If there is only

one parameter,

0:20:16.660,0:20:18.070

and the event is known -

0:20:18.070,0:20:19.560

it's not that bad.

0:20:19.560,0:20:21.600

But if the event gets

0:20:21.600,0:20:23.910

a lot of parameters, they

will be named as

0:20:23.910,0:20:25.740

-obj1, - obj2, - obj3

and so on.

0:20:25.740,0:20:27.390

As you can see,

the readability

0:20:27.390,0:20:28.280

suffers a lot!

0:20:28.280,0:20:30.040

That's why I

always recommend

0:20:30.040,0:20:31.460

using a full

0:20:31.460,0:20:32.830

form of an event.

0:20:32.830,0:20:35.090

Also, because one day

0:20:35.090,0:20:37.360

other developers

might need a link

0:20:37.360,0:20:38.410

to a class-publisher,

0:20:38.410,0:20:41.910

and there is no such thing

in a short form of an event.

0:20:41.910,0:20:43.700

Let's finish talking

0:20:43.700,0:20:46.720

about the events now.

As I've said, discussing

0:20:46.720,0:20:48.220

just one event type

0:20:48.220,0:20:49.630

took more time than

0:20:49.630,0:20:51.260

talking about all

other members

0:20:51.260,0:20:51.830

of a type.

0:20:51.830,0:20:54.019

It is because

all events are based on

0:20:54.019,0:20:56.570

a pretty hard part

of a C# language

0:20:56.570,0:20:58.850

called the delegates.

Hope, my lesson

0:20:58.850,0:21:01.279

dedicated to the usage

of the events and the delegates

0:21:01.279,0:21:03.860

was interesting and

understandable to you.

0:21:03.860,0:21:04.909

In case you have

0:21:04.909,0:21:07.590

any questions,

suggestions or

0:21:07.590,0:21:08.780

ideas, please,

0:21:08.780,0:21:11.020

feel free to write me a comment.

I will answer to all of them and

0:21:11.020,0:21:11.780

your opinion is important!

Thank you!