

Section 20: Delegates & Events

Monday, July 15, 2024 6:02 PM



Harsha Vardhan

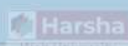
- > UI Expert
- > .NET Expert
- > Module Lead
- > Corporate Instructor

Delegates, Events and Lambda Expressions




Agenda

- 1 Delegates**
What is Delegate and its features & types.
- 2 Events**
What is Event and creating events
- 3 Auto-Implemented Events**
Creating event with auto-generated add-accessor and remove-accessor.
- 4 Anonymous Methods**
A method without name. New feature in C# 2.0.
- 5 Lambda Expressions**
Expression without name but with body. New feature in C# 3.0



Agenda

- 5 Inline Lambda Expressions**
Lambda Expressions with single expression body.
- 6 Expression Bodied Methods**
New feature in C# 7.0
- 7 Switch Expression**
New feature in C# 8.0
- 8 Func**
Delegate with parameters and return.



9

Action

Delegate with parameters but without return.

Agenda

10

Predicate

Delegate with one parameter and Boolean return.

11

Event Handler

Delegate for data exchange in events.

12

Expression Trees

New feature in C# 3.0

A delegate is a special object which stores a reference of a method. It can store the reference of one or more methods. Using delegate, you can call one or more methods indirectly.

So what is the benefit of calling the method indirectly? If you just use the delegates directly, there is no special or particular benefit. That is the reason, it is very rare to use delegates directly in real world applications. But the delegates are really helpful for building events.

Introducing Delegates
Next: Rules of Delegates

What

› "Delegate type" is a "type" that represents methods that have specific parameters and return type.

How

Creating Delegate 1

```
public delegate ReturnType DelegateTypeName(param1, param2, ...);
```

Introducing Delegates

Next: Rules of Delegates

What

- › "Delegate type" is a "type" that represents methods that have specific parameters and return type.
- › The "delegate" (a.k.a. delegate object), is an object that stores reference (address) of a specific method of a specific class, with compatible parameters and return type, which is already defined in the delegate type.

How

- Creating Delegate**

```
public delegate ReturnType DelegateTypeName(param1, param2, ...);
```
- Creating Delegate Object**

```
DelegateTypeName ReferenceVariable = MethodName;
```
- Invoke Method using Delegate Object**

```
ReferenceVariable.Invoke( arg1, arg2, ... );
```

Rules of Delegates

Next: Types of Delegates

Rules

- › You can invoke the methods using 'delegate objects' (or) 'delegates'.
- › Delegates are used to pass methods as arguments to other methods.
- › The method signature (parameters and return type) must match between the "method" and "delegate".
- › Delegates can be used as "parameter type" or "return type" of a method.
- › You can store references of non-static method or static method in the delegate object.
- › The methods, which reference is stored in the "single-cast delegate object", can have return value.
- › The methods, which reference is stored in the "multi-cast delegate object", can't have return value; in case, if they have return value, the return value of lastly-executed method only can be received; others will be ignored.

› All delegate types are derived from `System.Delegate` class.

Types of Delegates

Next: Events

Single-Cast Delegates

- › Contains reference of only one method.
- › When called, it directly invokes the

Multi-Cast Delegates

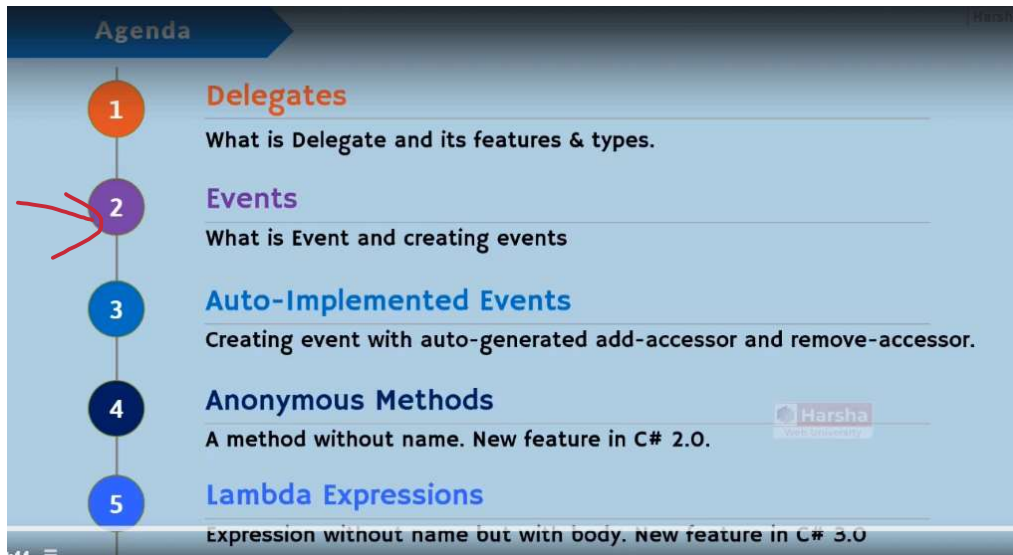
- › Contains references of multiple methods.
- › When called, it invokes all the

referenced method.

referenced methods, one-by-one in a sequence.

- › All methods' parameters and return type should be same.

It is recommended not to have the return value in case of multi-case delegates.



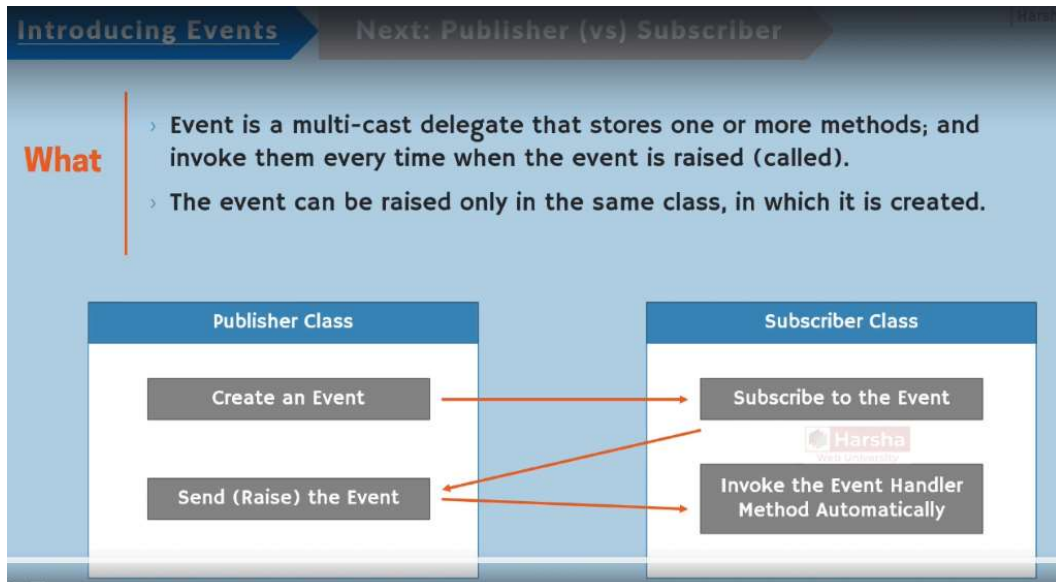
Event is a multi-case delegate that created and invoked in Publisher class and subscribed in Subscriber class. So Event is basically a delegate.

Event implements the architecture of 'Publisher vs Subscriber'. Whenever the publisher raises event, the corresponding method in the Subscriber class will be executed.

That means, whenever some data has been changed in the Publisher class; the Publisher class wants to provide a notification to the Subscriber class to inform the Subscriber class that some data has been changed.

So that, the Subscriber class can perform appropriate operation that should be executed as soon as the data has been changed in the Publisher class.

In order to implement this pattern, first we will create an event in the Publisher Class.



Assume in a banking application there are two classes called *BankAccount* and *InterestCalculator*. In the *BankAccount* we have a property called *InterestRate*. And in the *InterestCalculator* class, we have a method to calculate the interest based on the interest rate.

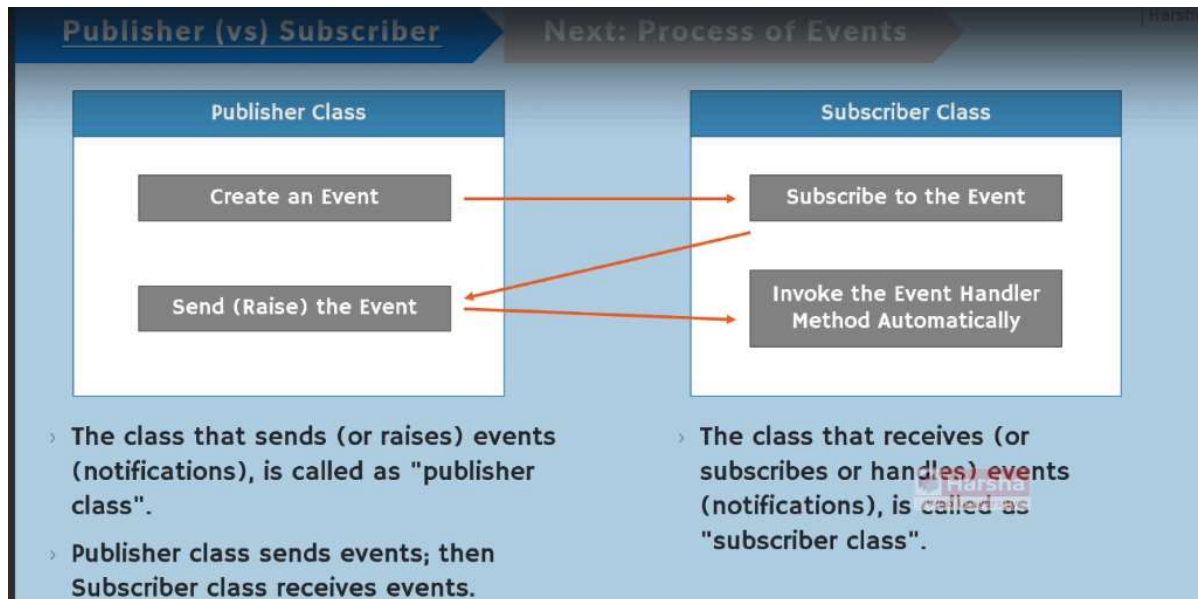
Whenever the *interestRate* has been changed in the *BankAccount* class, the *BankAccount* class wants to pass a notification to the *InterestCalculator* class so that *InterestCalculator* class can calculate the interest based on the modified *interestRate* as per the *BankAccount* class.

So here, the *BankAccount* is raising the event and *InterestCalculator* class receives the notification.

So that is the reason we can say that *BankAccount* class is the *Publisher* class and the *InterestCalculator* is the *Subscriber* class. The intention of the publisher class is that I'll pass essential notification as soon as some data has been changed in the class. So that, the opposite subscriber class can do their own work as per their requirement.

And here the *Subscriber* class's perspective is that, I'll subscribe to the *Publisher* class; that means I'll subscribe to the event and as soon as event is raised, the corresponding method will be executed in me. So that I'll perform essential operation that should be

executed when then notification is passed from the publisher. For example, interestRate has been changed, we need to recalculate the interest based on the newly modified interest rate.



It's all about action and reaction. Action happens in Publisher class and Reaction happens in Subscriber class. That means, whenever the event is raised in Publisher class; the corresponding subscribed method will be executed in the Subscriber class. This is also called as Notification System in C#.

It is as simple as door bell. Whenever somebody press the door bell, the corresponding sound should be played in the door bell. So if 'Pressing the switch' is the event; 'Playing the sound ' is the corresponding subscribed method that executes automatically as soon as the event is raised.

In the real world application, some data can be changed in one class, the same class will notify the other class that the data has been changed and the corresponding calculation will take place in the other class.

This is also called as 'Publisher' vs 'Subscriber' Pattern.


Another scenario of is, in GUI, whenever somebody clicks on the button, the corresponding method should be executed. Here Button is a class and the 'Form' is

another class that contains the method that shows relevant output to the user.

Whenever somebody clicks on the button, the button raises the event and corresponding form class catches the event and the corresponding subscribed method will be executed.

Here click is the 'event'; Button is the Publisher class and the Form is the Subscriber class.

Process of Events



Process flow of Events

Next: Creating Events

- Events enable a class to send notifications to other classes, when something occurs.
- Publisher class sends events; Subscriber class receives events.
- The Publisher class creates an event.
- The Subscriber class subscribes to the event; that means an "event handler" method is created in the subscriber class. The "event handler" method is nothing but, the method which is dedicated to be executed when the event is raised.
- The publisher class can send (raise) events.
- Every time, when the event is raised by the publisher, the corresponding "event handler" method executes automatically.

Creating Events

Steps for creating Events

Next: Rules of Events

Create a Delegate 1

```
public delegate ReturnType DelegateTypeName(param1, param2, ...);
```

Based on the delegate type, you will create the event.

Creating Events

Steps for creating Events

Next: Rules of Events

Create an Event in Publisher Class 2

```
class Publisher
{
    private DelegateTypeName eventVariable;
    public event DelegateTypeName EventName
    {
        add
        {
            eventVariable += value;
        }
    }
}
```



```

        remove
        {
            eventVariable -= value;
        }
    }
}

```

Creating Events
Next: Rules of Events

Steps for creating Events

Raise the event in Publisher Class
3

```
if (EventName != null) EventName(arg1, arg2, ...);
```

Create Event Handler Method in Subscriber Class
4

```
class Subscriber
{
    public ReturnType EventHandlerMethodName(param1, param2, ...)
    {
        Method body here
    }
}
```


Creating Events
Next: Rules of Events

Steps for creating Events

Subscribe to the Event [Inside or Outside the Subscriber class]
5

```
EventName += EventHandlerMethodName;
```

Rules of Events
Next: Auto-Implemented Events



- › The event should be created based on the delegate. That means, the event accepts the methods that are having specific parameters and return type, defined in the delegate.
- › An event can have multiple subscribers.
- › A subscriber can subscribe multiple events from multiple publishers.
- › Events are basically signals to inform to other classes, that some important thing happened in the publisher class.
- › Events are special kind of "multi-cast delegates", which can be raised only within the same class, in which they are created.
- › Events can be static, virtual, sealed and abstract.
- › Events will not be raised (throws exception), if there is no at least one subscriber.
- › Events can be defined in interfaces.
- › It's not a good idea to return value in events.

Agenda


- 1 Delegates**
What is Delegate and its features & types.
- 2 Events**
What is Event and creating events
- 3 Auto-Implemented Events**
Creating event with auto-generated add-accessor and remove-accessor.
- 4 Anonymous Methods**
A method without name. New feature in C# 2.0.
- 5 Lambda Expressions**
Expression without name but with body. New feature in C# 3.0

Harsha
Web University

Understanding Auto-Impl Events

Next: Anonymous Methods

- > You also not required to create a private multi-cast delegate; the compiler does the same automatically.
- > Disadvantage: We can't define custom logic for "add accessor" and "remove accessor".



So in real time applications - whenever you want to quickly create the event without any additional logic in the add and remove accessor;

Agenda

- 1 Delegates**
What is Delegate and its features & types.

2

Events

What is Event and creating events

3

Auto-Implemented Events

Creating event with auto-generated add-accessor and remove-accessor.

4

Anonymous Methods

A method without name. New feature in C# 2.0.

5

Lambda Expressions

Expression without name but with body. New feature in C# 3.0

In fact, Anonymous Methods are ment for quickly creating a method with some less amount of code - that means with few lines of code.

Anonymous Methods
Next: Lambda Expressions

What

Anonymous methods are "name-less methods", that can be invoked by using the delegate variable or an event.

How

Subscribe to Event with Anonymous Method

```

EventName += delegate(param1, param2, ...)
{
    //method body here
}
```

For Anonymous method, modifier or access modifier is not applicable.

Understanding Anonymous Methods
Next: Lambda Expressions

Anonymous methods can be used anywhere within the method, to create methods instantly, without define a method at the class level.

Advantage: We need not create a "named method (normal method)" to quickly handle an event.

Rules

- › It can't be called without a delegate or event.
- › It can't contain jump statements like goto, break, continue.
- › It can access local variables and parameters of outer method.
- › It can be passed as a parameter to any method; in this case, the delegate acts as data type for the anonymous method.
- › It can't access ref or out parameter of an outer method.
- › It is mainly used for event handlers.

Agenda

- 1 **Delegates**
What is Delegate and its features & types.
- 2 **Events**
What is Event and creating events
- 3 **Auto-Implemented Events**
Creating event with auto-generated add-accessor and remove-accessor.
- 4 **Anonymous Methods**
A method without name. New feature in C# 2.0.
- 5 **Lambda Expressions**
Expression without name but with body. New feature in C# 3.0

Harsha
Web University

Lambda Expressions

Next: Inline Lambda Expressions

What

› "Lambda Expressions" (a.k.a. Statement Lambda) are "name-less methods", that can be invoked by using the delegate variable or an event, much like anonymous methods.

How

Handle Event with Lambda Expressions


```

EventName += (param1, param2, ...) =>
{
    //method body here
}
                    
```

Harsha
Web University

Understanding Lambda Expressions

Next: Inline Lambda Exp



- › Lambda Expressions can be used anywhere within the method, methods instantly, without define a method at the class level.
- › **Advantage:** It provides more easier and convenient syntax than methods".

- › => operator is called as "goes to" or "goes into" operator.

Agenda

- 6** **Inline Lambda Expressions**
Lambda Expressions with single expression body.
- 7** **Expression Bodied Methods**
New feature in C# 7.0
- 8** **Switch Expression**
New feature in C# 8.0
- 9** **Func**
Delegate with parameters and return.
- 10** **Action**
Delegate with parameters but without return.

Harsha Web University

Inline Lambda Expressions

Next: Expression Bodied Members

What

- › "Inline Lambda Expressions" (a.k.a. Expression Lambda) are the lambda expressions, which performs a small calculation or condition check and returns a value.
- › Inline lambdas can receive one or more arguments and must return a value.
- › **Advantage:** It provides more easier and convenient syntax to create smaller methods that performs a single calculation or condition check.

How

Handle Event with Inline Lambda Expressions

```
EventName += (param1, param2, ...) => condition or calculation
```

Harsha Web University

Agenda

- 6** **Inline Lambda Expressions**
Lambda Expressions with single expression body.
- 7** **Expression Bodied Methods**
New feature in C# 7.0

Harsha

8

9

10

Switch Expression

New feature in C# 8.0

Func

Delegate with parameters and return.

Action

Delegate with parameters but without return.

Func
Next: Action

What

- > "Func" is a pre-defined generic-delegate, which can be used to create events quickly.
- > Func supports parameters and return value also.
 - > Func must have 0 to 16 parameters.
 - > Func must have return value.

How

Creating Delegate Based on Func

```
public Func<Param1DataType, Param2DataType, ReturnType> referenceVariable;
```

Can be 0 to 16 parameters

Last one is treated as "return type" [it is must]

Agenda
Harsha

6

7

8

9

10

Inline Lambda Expressions

Lambda Expressions with single expression body.

Expression Bodied Methods

New feature in C# 7.0

Switch Expression

New feature in C# 8.0

Func

Delegate with parameters and return.

Action

Delegate with parameters but without return.

Action => pre-defined delegate in C#. Similar of Func.

[https://onedrive.live.com/redir?resid=288727FEA60D266F%21294790&page=Edit&wd=target%28New Section 1.one%7Cb280e81f-0baa-415a...](https://onedrive.live.com/redir?resid=288727FEA60D266F%21294790&page=Edit&wd=target%28New%20Section%201.one%7Cb280e81f-0baa-415a...)

13/22

Difference is Func must contain a return type, Action is always void.

Action
Next: Predicate

What

- > "Action" is a pre-defined delegate, which can be used to create events quickly, similar to "Func".
- > The difference is:
 1. Func must have return value; Action don't have return value.
 2. Action must have 0 to 16 parameters.

How

Creating Delegate Based on Action

```
public Action<Param1DataType, Param2DataType, Param3DataType> referenceVariable;
```

Can be 0 to 16 parameters

[No return value]

Agenda

11

Predicate

Delegate with one parameter and Boolean return.

12

EventHandler

Delegate for data exchange in events.

13

Expression Trees

New feature in C# 3.0

Whenever you want to have a return value – that is of bool type, then only use Predicate.

Predicate
Next: Event Handler

What

- > "Predicate" is a pre-defined delegate, which can be used to create events quickly, similar to "Func".
- > The difference is:
 1. Func must have return value of any type; Action don't have return value.

<https://onedrive.live.com/redir?resid=288727FEA60D266F%21294790&page=Edit&wd=target%28New+Section+1.one%7Cb280e81f-0baa-415a...>

14/22

1. Func must have return value of any type; Action don't have return value; Predicate must have return value of "bool" type.
2. Func can have 0 to 16 parameters of any type; Action can have 0 to 16 parameters of any type; Predicate must have only one parameter of any type.



Predicate

Next: Event Handler

How

Creating Delegate Based on Predicate

```
public Predicate<Param1DataType> referenceVariable;
```

Only one
parameter

Default return type is
"bool"



Agenda

11

Predicate

Delegate with one parameter and Boolean return.

12

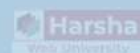
EventHandler

Delegate for data exchange in events.

13

Expression Trees

New feature in C# 3.0



This lecture is about a **predefined delegate type called Event Handler**.

EventHandler

Next: Expression Trees

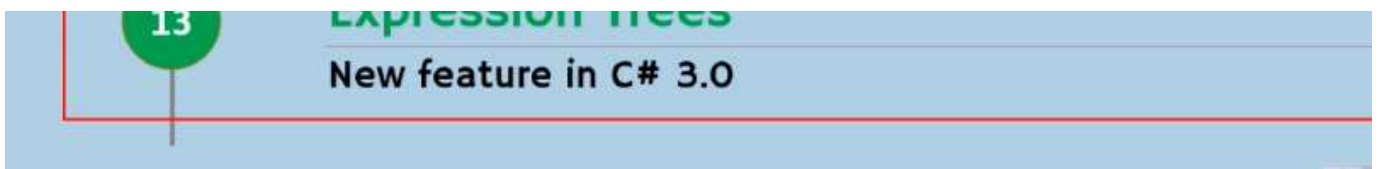
> 'EventHandler' is a pre-defined delegate type, which has two

What	parameters called "object sender" and "EventArgs e"; and no return.
Parameters	<ul style="list-style-type: none"> > object sender: Represents the source object, where the event is originally raised. > EventArgs e: Represents additional parameters to pass to 'event handler method'. It is recommended to create a child class for 'EventArgs' class.
How	<div> <div>Creating Event Based on EventHandler</div> <pre>public event EventHandler EventName;</pre> </div>

While working working WindowsForms, WPF or Asp.Net Web Forms;

in order to subscribe to the predefined events such as Click, DoubleClick event etc., you can use the predefined delegate type called EventHandler.

Because, in case of WinForms, WPF or ASP.NET Web Forms; most of the predefined events are of EventHandler type.



In real time applications, mostly we do not use Expression Trees Manually. But we use it in LINQ. That means the concept of LINQ internally uses Expression Trees. As a developer we use expression trees in the real time applications.

Expression Trees

What	<ul style="list-style-type: none"> › Expression Tree is a collection of delegates represented in tree-like structure. › Expression Tree only executes when we compile and execute it. › Expression Trees support all delegate types such as Func, Action, Predicate or custom delegate types.
How	<div style="background-color: #0056b3; color: white; text-align: center; padding: 5px; margin-bottom: 5px;"> Creating Expression Tree based on Func </div> <div style="background-color: white; color: black; text-align: center; padding: 10px; margin-bottom: 5px;"> <code>Expression< Func<type1, type2, ...> > referenceVariable;</code> </div> <div style="background-color: #0056b3; color: white; text-align: center; padding: 5px; margin-bottom: 5px;"> Compile and Execute Expression Tree </div> <div style="background-color: white; color: black; text-align: center; padding: 10px;"> <code>Func<type1, type2, ...> referenceVariable2 = referenceVariable.Compile(); referenceVariable2.Invoke(arg1, arg2, ...);</code> </div>

There will be a parent delegate which contains one or more delegates. And each child delegate contains other child delegates. So like this, whenever the delegates are arranged in tree-like structure, it is called an Expression Tree.

Whenever we invoke the expression tree, all the corresponding delegates will be executed in an up approach. That means the child delegates will be first executed and then based on the result of the child delegates, the parent delegate will be executed. For example, there is a child delegate that calculates numbers and another child delegate calculates multiplication of two numbers. Based on the results of these two delegates, the parent delegate will be executed.

In the expression trees, you can store delegates of almost all delegate types, including the pre-defined delegate types such as Func, Action, Predicate or any other custom delegate type.

Agenda

- 6 **Inline Lambda Expressions**
Lambda Expressions with single expression body.
- 7 **Expression Bodied Methods**
New feature in C# 7.0
- 8 **Switch Expression**
New feature in C# 8.0
- 9 **Func**
Delegate with parameters and return.
- 10 **Action**
Delegate with parameters but without return.

Harsha Web University

As per expression bodied members, wherever you have single statement or single value as your method or property, we need not define method body or property body separately.

Whenever you can calculate age of the person in a single statement; that means, in a single expression we need not create the method body by writing the braces { }.

Expression Bodied Members Next: Switch-Expression

What

- "Expression Bodied Members" concept allows the developer to use "Inline Lambda Expressions" to create methods, property accessors, constructors, destructors, indexers in a class.

How

Method using Expression Bodied Members - without return value

```
public ReturnType MethodName( ) => statement;
```

Method using Expression Bodied Methods - with return value

```
public ReturnType MethodName( ) => AnyValue;
```

The same syntax can also be used for methods, properties, constructors, destructors or indexers in the class.

Understanding Expression Bodied Members

Next: Switch-Expression



- › Expression Bodied Members may have or parameters; may / may not have return value.
- › Expression Bodied Members can have only one statement.
- › **Advantage:** It provides more easier and convenient syntax to create smaller methods that performs a single calculation or condition check.

Expression Bodied Members - Usage

Next: Switch-Expression

Constructor

Constructor with Expression Bodied Members

```
public ClassName(param1) => field = param1;
```

Property

Property with Expression Bodied Members

```
public type Property
{
    set => field = value;
    get => field;
}
```



Agenda

6

Inline Lambda Expressions

Lambda Expressions with single expression body.

7

Expression Bodied Methods

New feature in C# 7.0

8

Switch Expression

New feature in C# 8.0

9

Func

Delegate with parameters and return.

10

Action

Delegate with parameters but without return.



Switch Expression

Next: Func

What

- › Switch Expression is a short-form of "switch-case", which is used to check the value of source variable; assign value into result value based on the value of source variable.

How**Switch Expression**

```
sourceVariable switch  
{  
    value1 => result1,  
    value2 => result2,  
    ...  
    _ => defaultResult  
}
```



