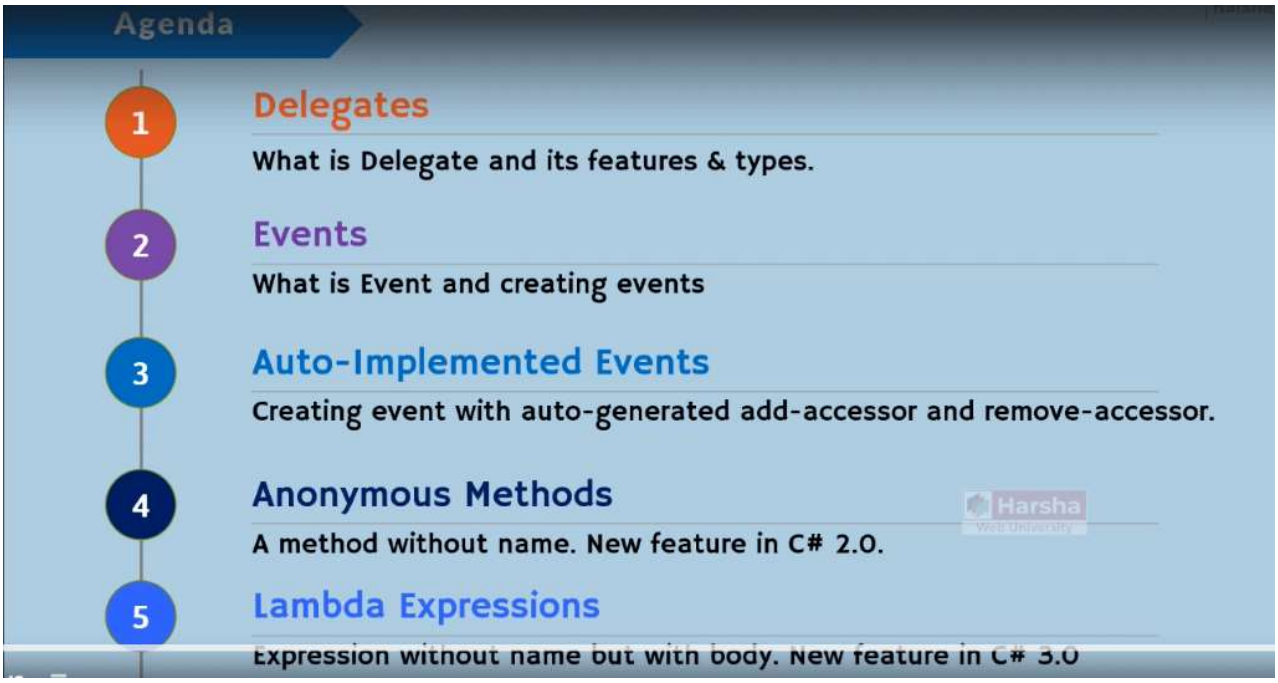
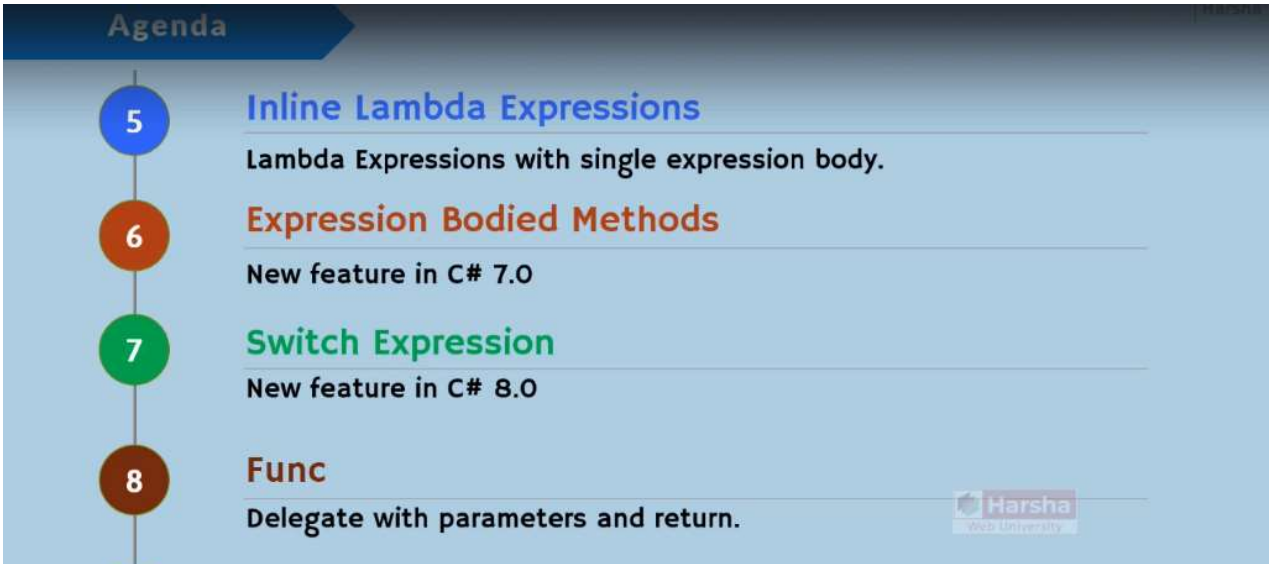
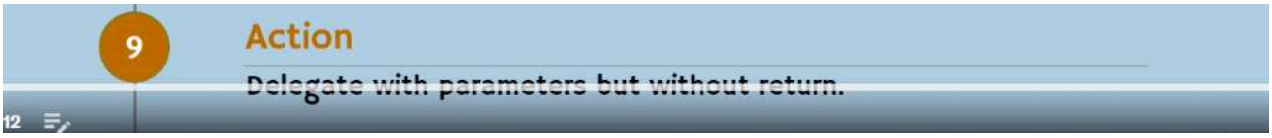
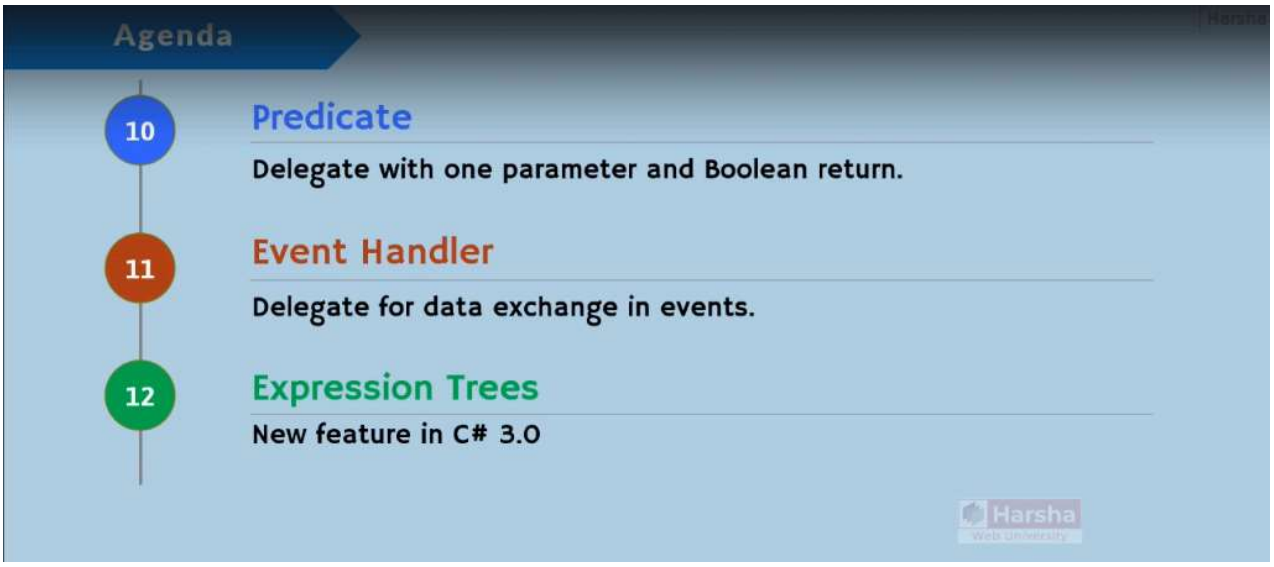
Section 20: Delegates & Events

Monday, July 15, 2024 6:02 PM



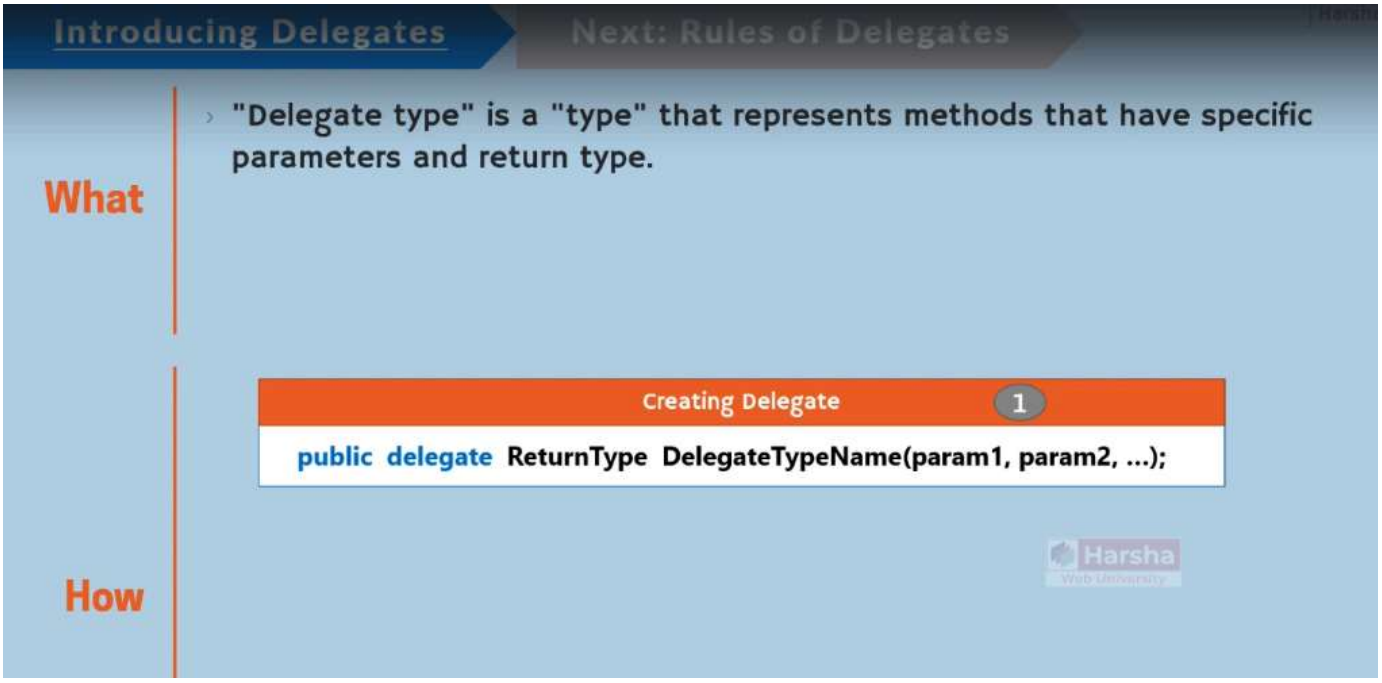






A delegate is a special object which stores a reference of a method. It can store the reference of one or more methods. Using delegate, you can call one or more methods indirectly.

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.
- 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 (1)

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

Creating Delegate Object (2)

```
DelegateTypeName ReferenceVariable = MethodName;
```

Invoke Method using Delegate Object (3)

```
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	Multi-Cast Delegates
Contains reference of only one method.	Contains references of multiple methods.
When called, it directly invokes the	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.

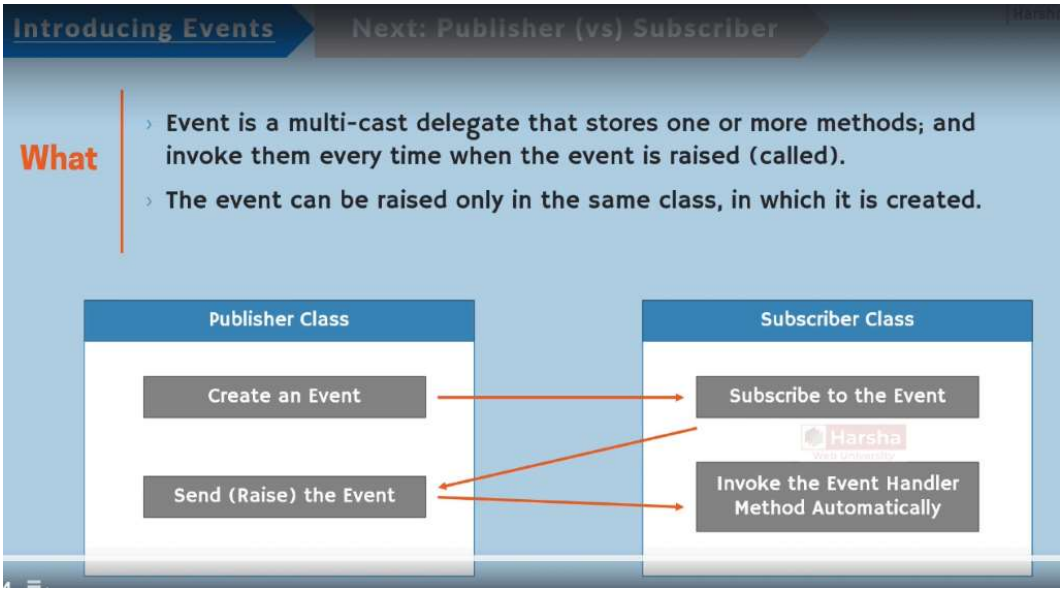
Event is a multi-case delegate that created and invoked in Publisher class and subscribed in Subscriber class. So Event is basically a delegate.

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 IntersetCalculator class, we have a method to calculate the interest based on the interest rate.

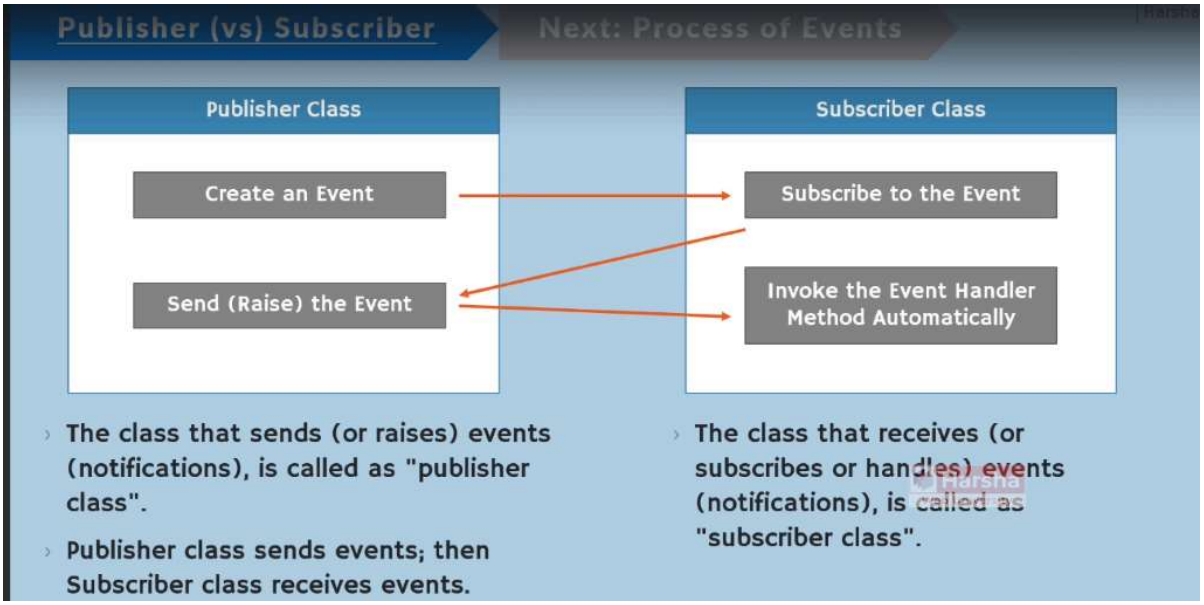
Whenever the interestRate has been changed in the BankAccount class, the BankAccount class wants to pass a notification to the InterestCalculator class so that InterestCalculator class can calculate the interest based on the modifed interestRate as per the BankAcoount class.

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

So that is the reason we can say that BankAccount class is the Publisher class and the InterstCalculator 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.



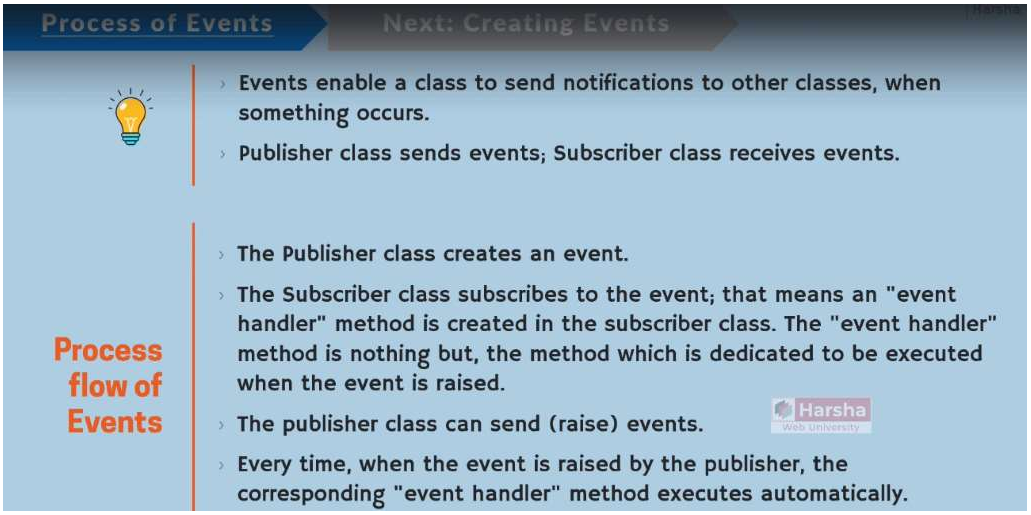
It's all about action and reaction. Action happens in Publisher class and Reaction happens in Subscriber class. That means, whenever the event is raised in Publisher class; the corresponding subscribed method will be executed in the Subscriber class. This is also called as Notification System in C#.

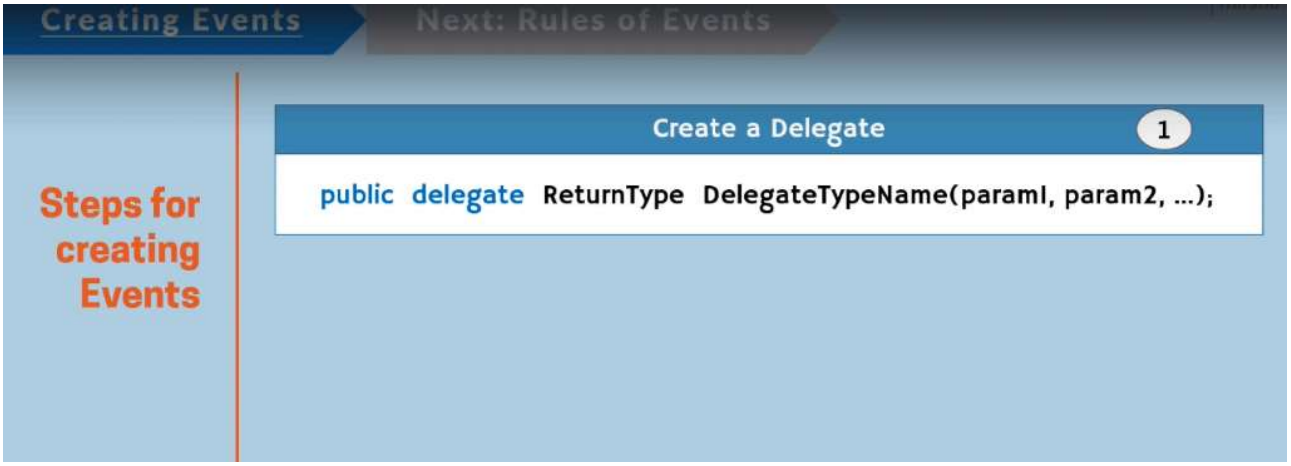
It is as simple as door bell. Whenever somebody press the door bell, the corresponding sound should be played in the door bell. So if 'Pressing the switch' is the event; 'Playing the sound ' is the corresponding subsribed method that executes automatically as soon as the event is raised.

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 calculcation will take place in the other class.
This is also called as 'Publisher' vs 'Subscriber' Pattern.

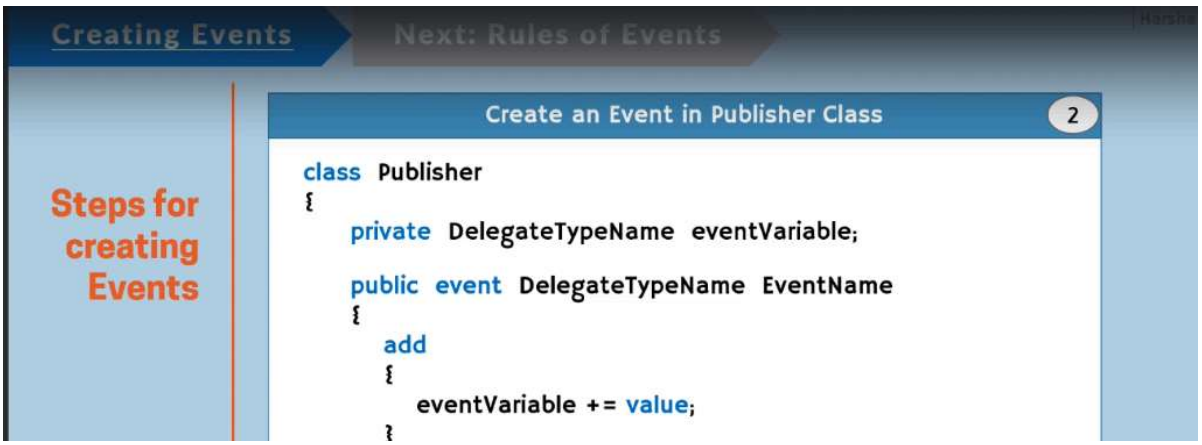
Another scenario of is, in GUI, whenever somebody clicks on the button, the corresponding method should be executed. Here Button is a class and the 'Form' is

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 methd will be executed.
Here click is the 'event'; Button is the Publisher class and the Form is the Subscriber class.





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



```
        remove
        {
            eventVariable -= value;
        }
    }
}
```

Harsha Web University

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
    }
}
```

Harsha Web University

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.

Harsha Web University

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

Understanding Auto-Impl Events

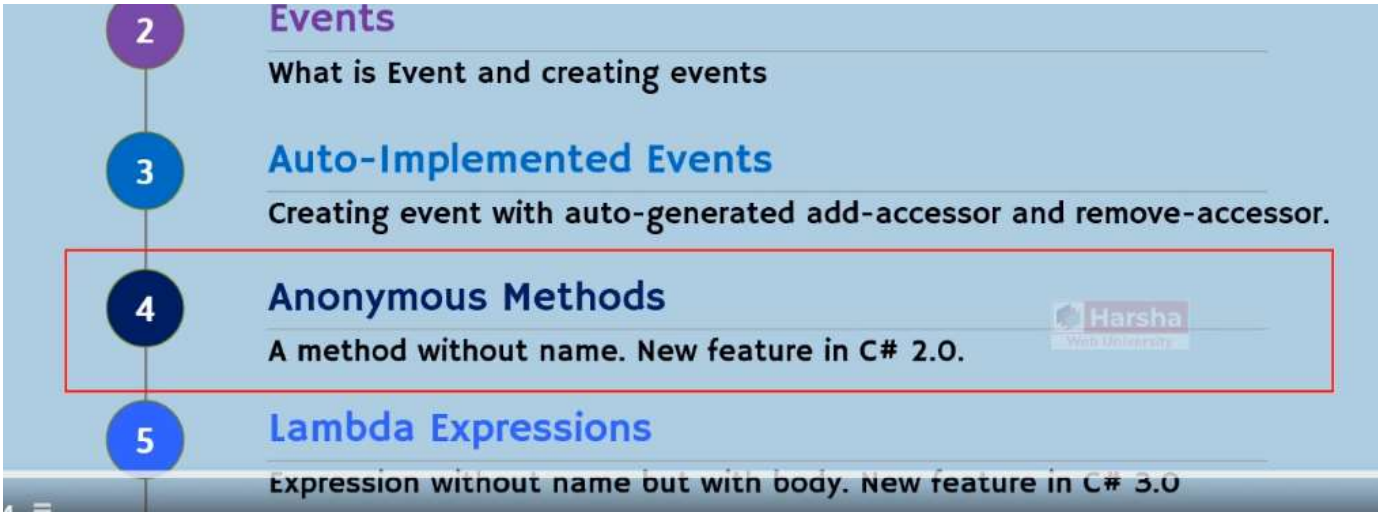
Next: Anonymous Methods

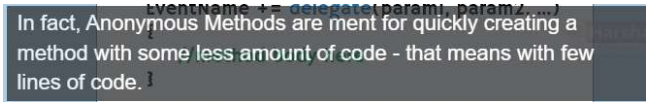
- You also not required to create a private multi-cast delegate; the compiler does the same automatically.
- Disadvantage: We can't define custom logic for "add accessor" and "remove accessor".

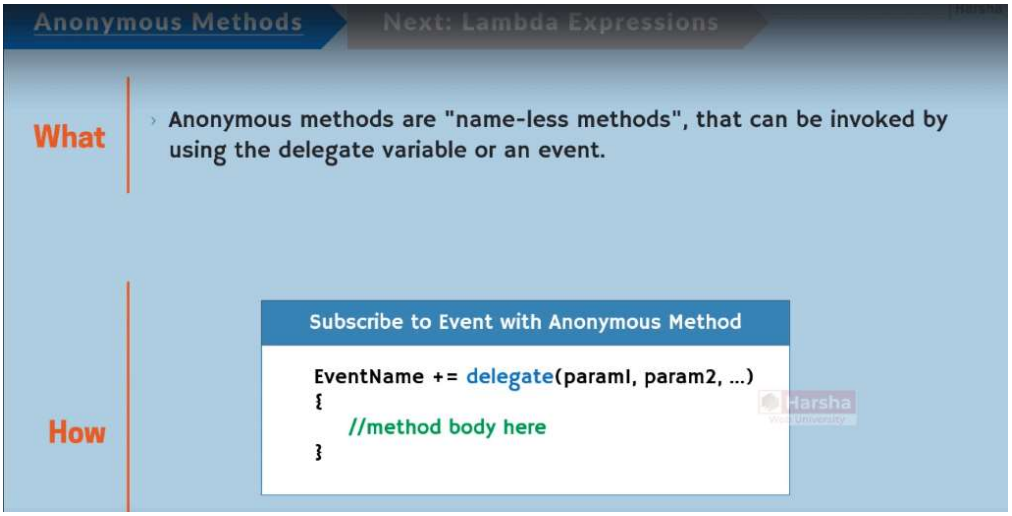
So in real time applications - whenever you want to quickly create the event without any additional logic in the add and remove accessor;

Agenda

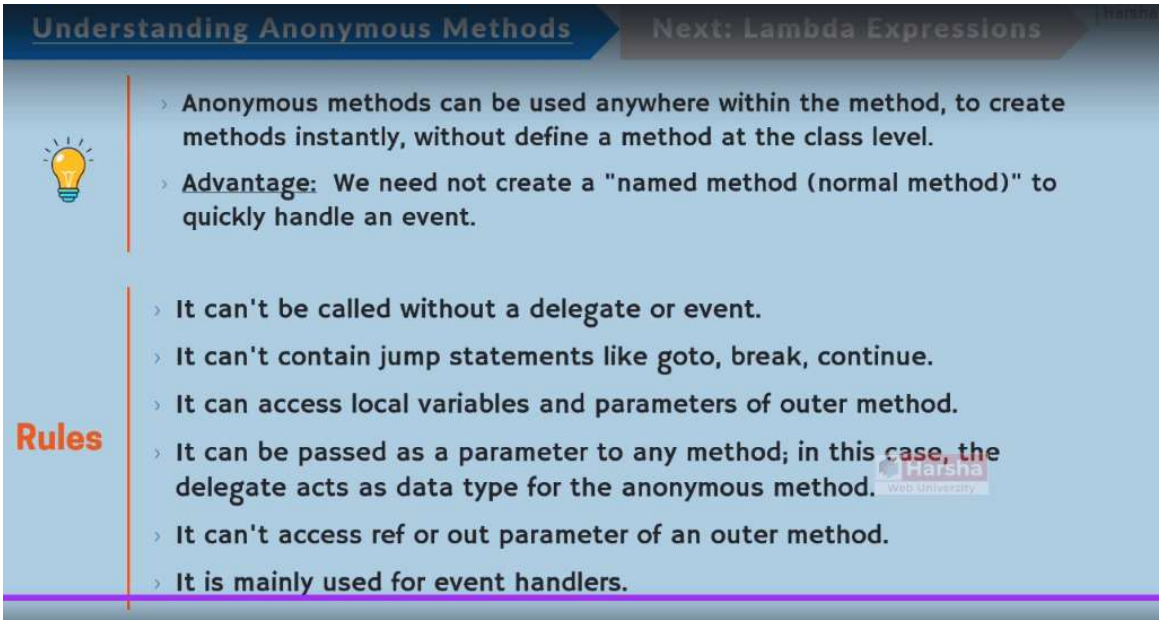
1. **Delegates**
   What is Delegate and its features & types.







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



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

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
}
```


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 tha 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.

Inline Lambda Expressions

Next: Expression Bodied Members

What

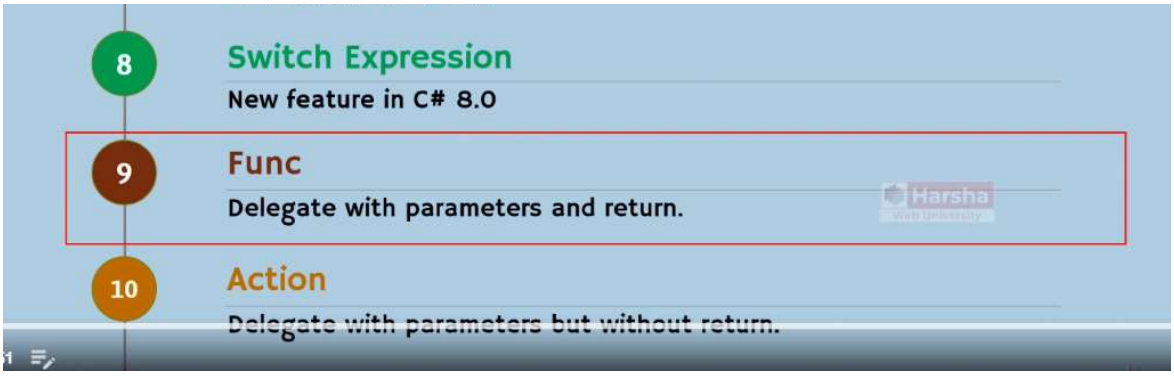
- "Inline Lambda Expressions" (a.k.a. Expression Lambda) are the lambda expressions, which performs a small calculation or condition check and returns a value.
- Inline lambdas can receive one or more arguments and must return a value.
- Advantage: It provides more easier and convenient syntax to create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

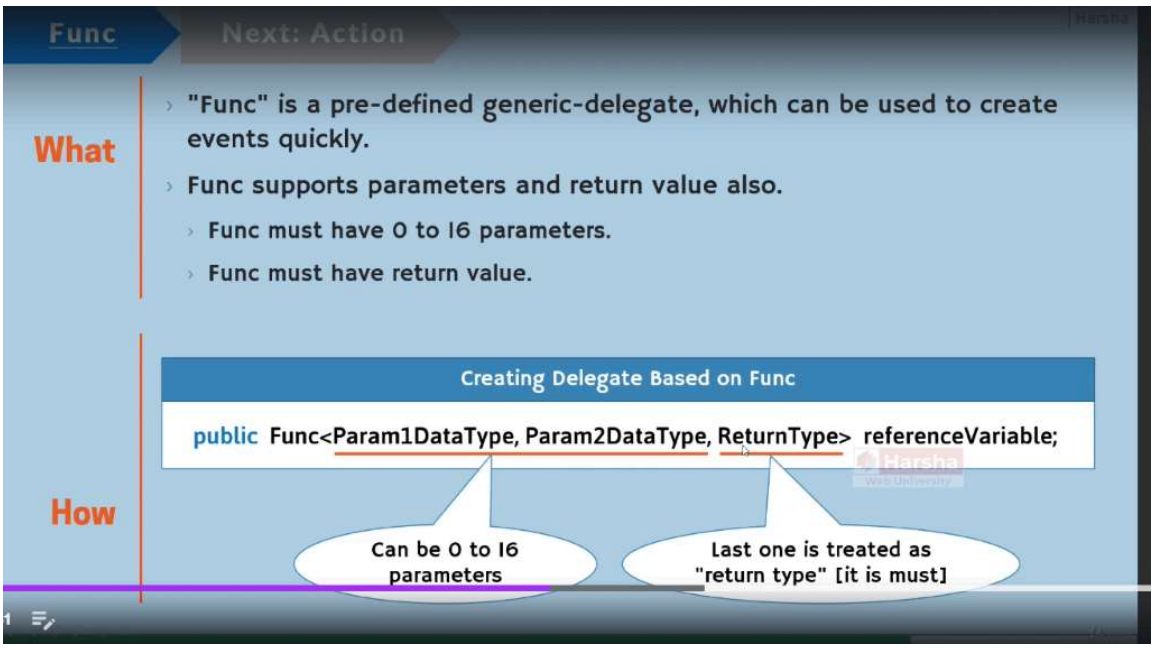
How

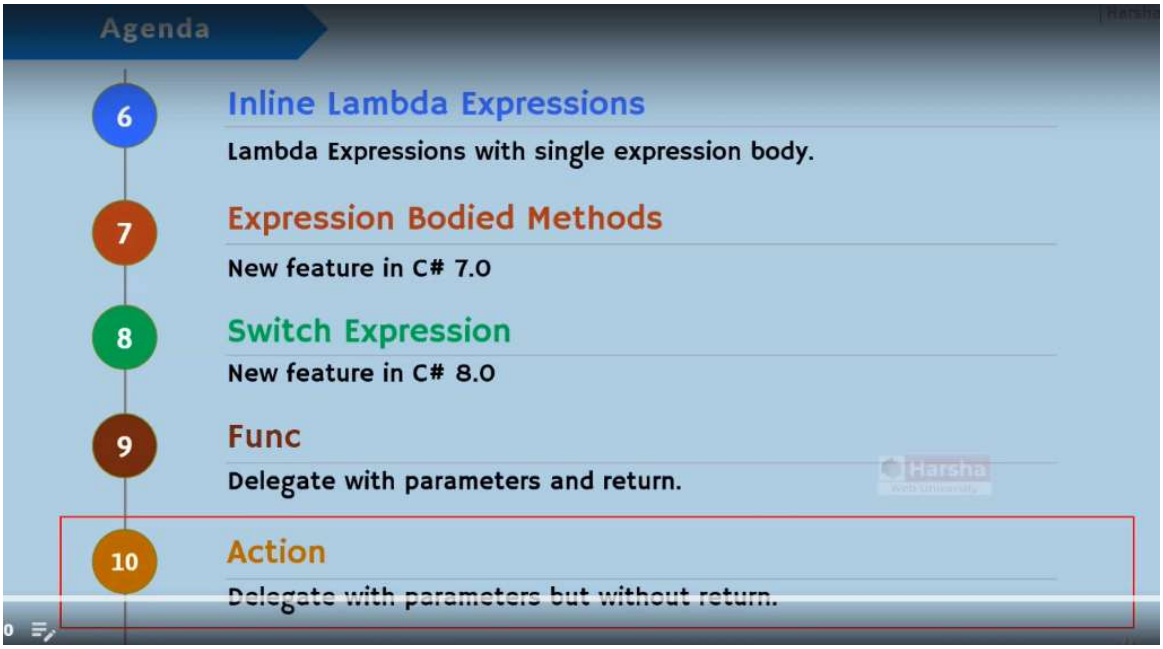
Handle Event with Inline Lambda Expressions
EventName += (param1, param2, ...) => condition or calculation

Agenda

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

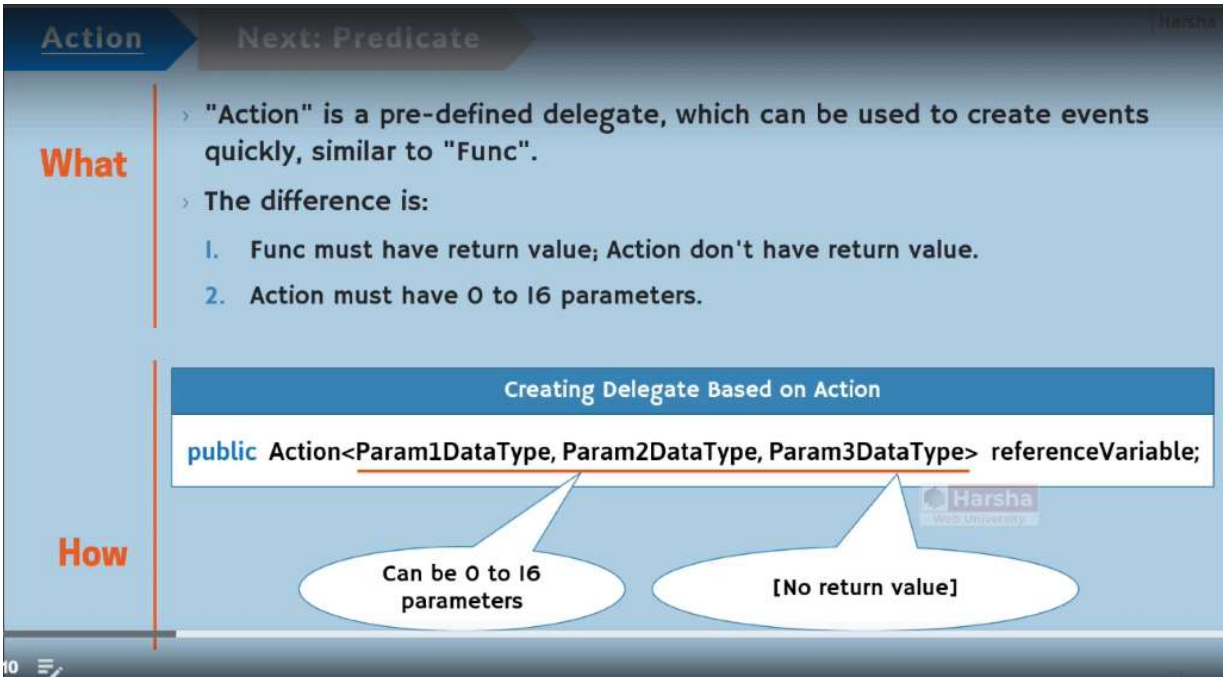


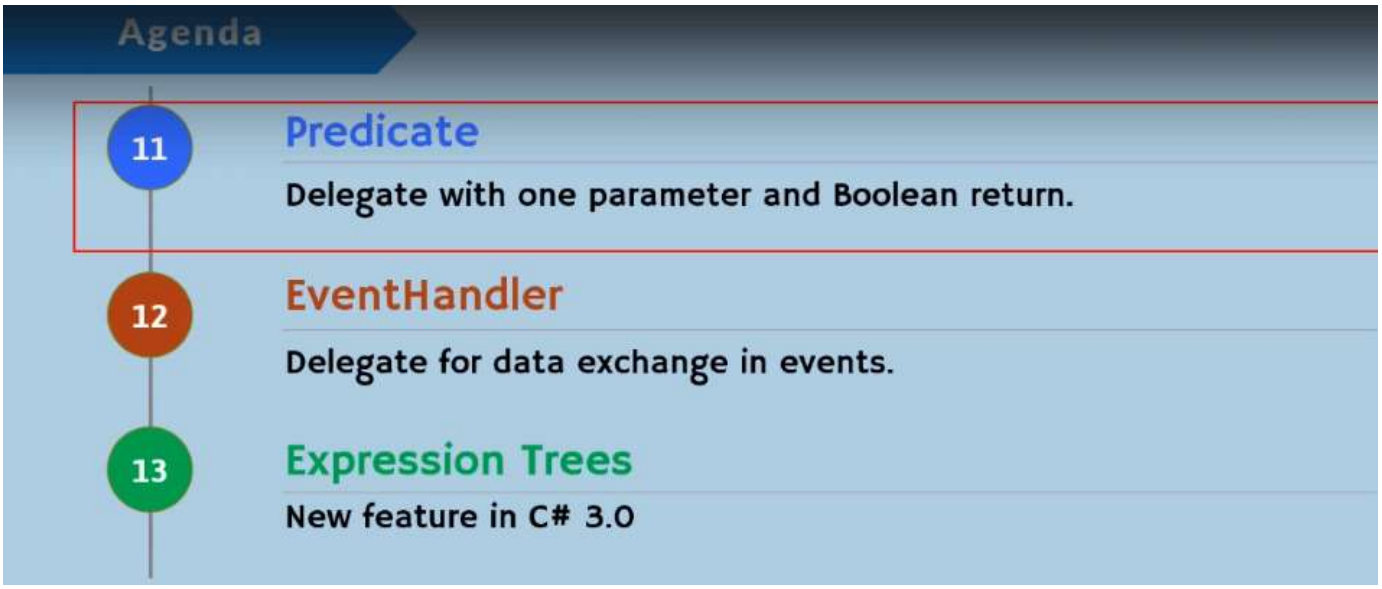




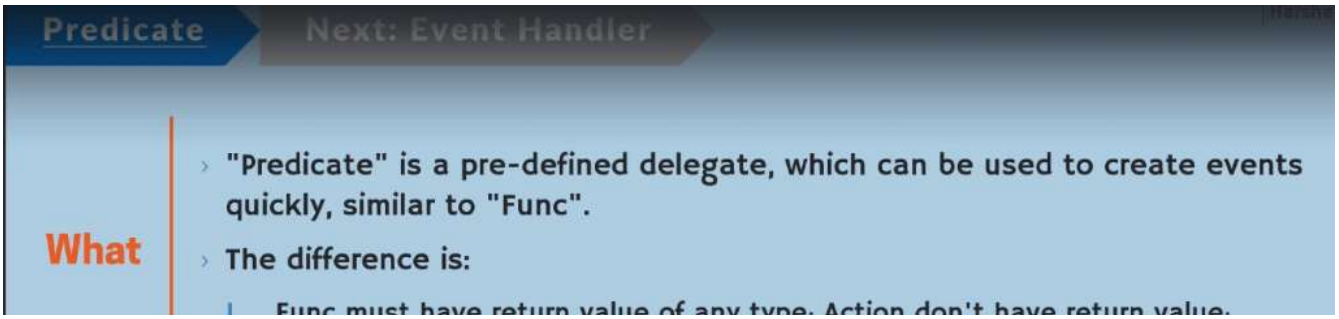
Action => pre-defined delegate in C#. Similar of Func.

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



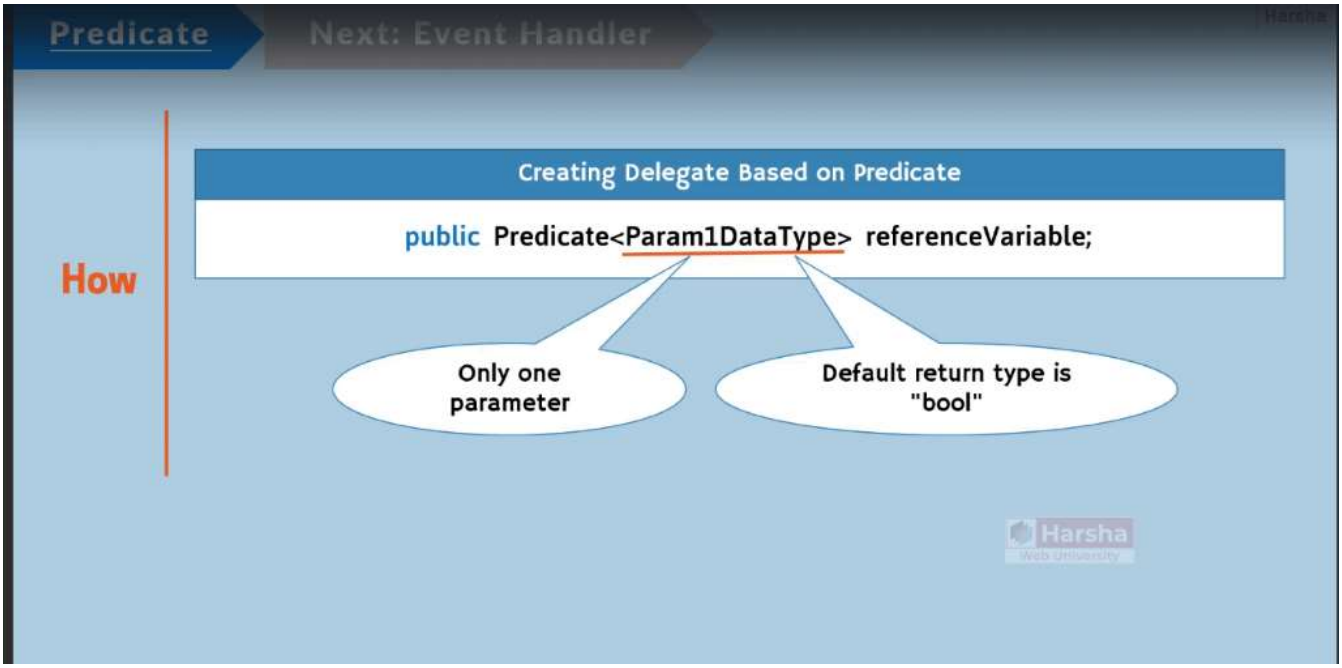


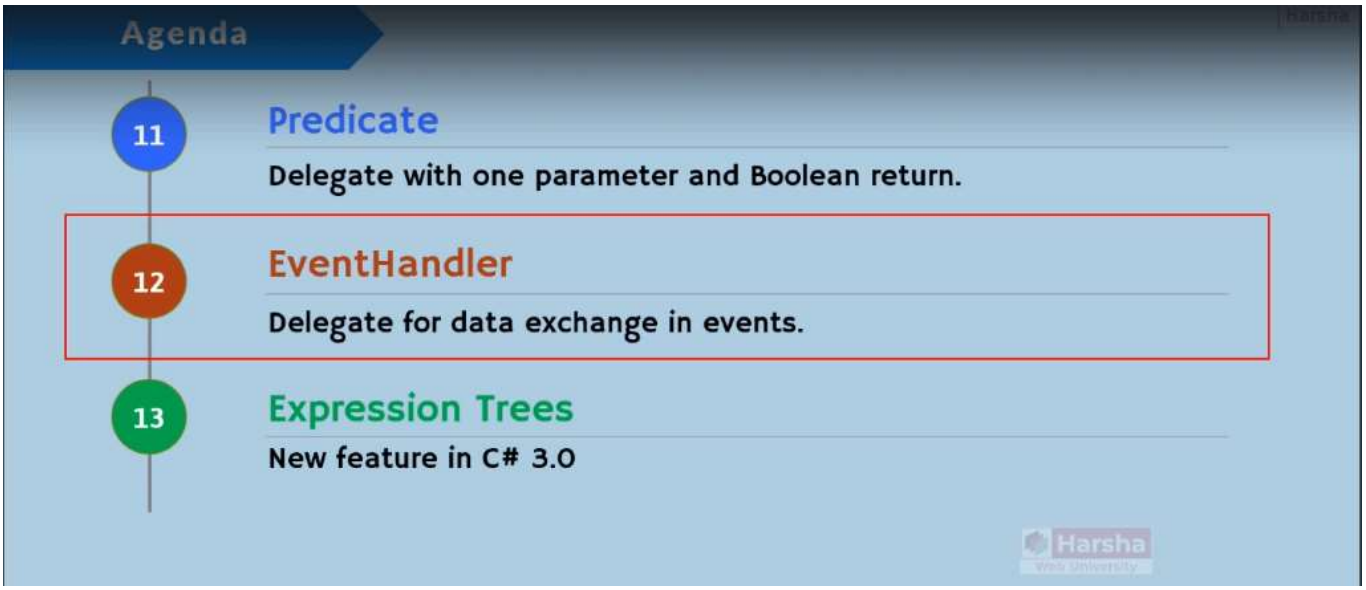
Whenever you want to have a return value – that is of bool type, then only use Predicate.



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.





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

What

parameters called "object sender" and "EventArgs e"; and no return.

Parameters

- object sender: Represents the source object, where the from 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

Creating Event Based on EventHandler

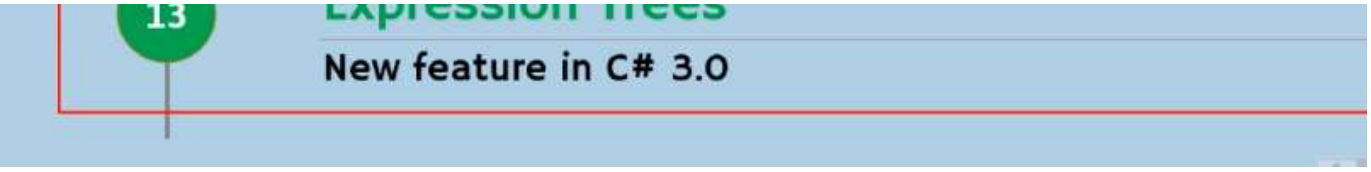
```
public event EventHandler EventName;
```

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

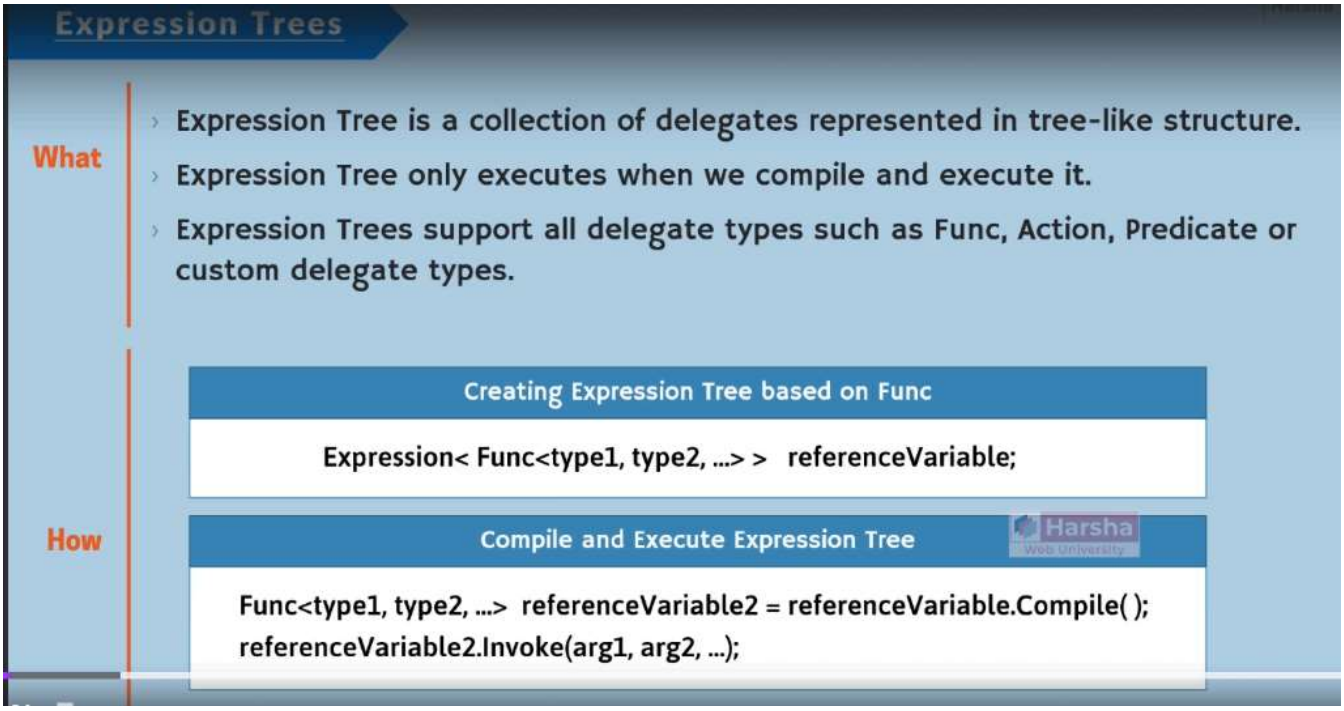
in order to subscribe to the predefined events such as Click, DoubleClick event etc., you can use the predefined delegate type called EventHandler.

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

Expression Trees

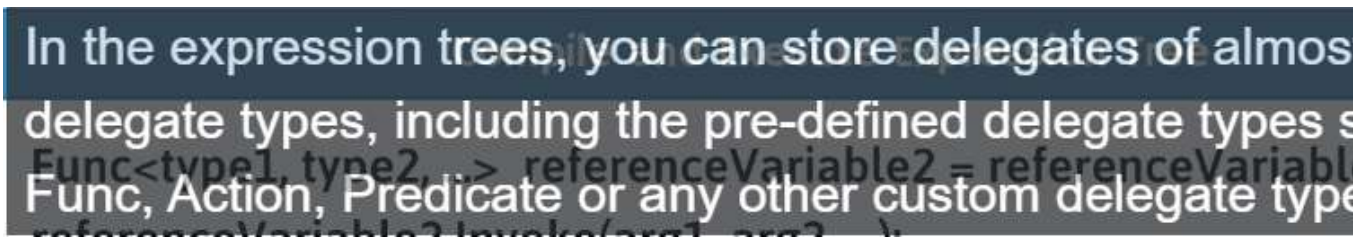


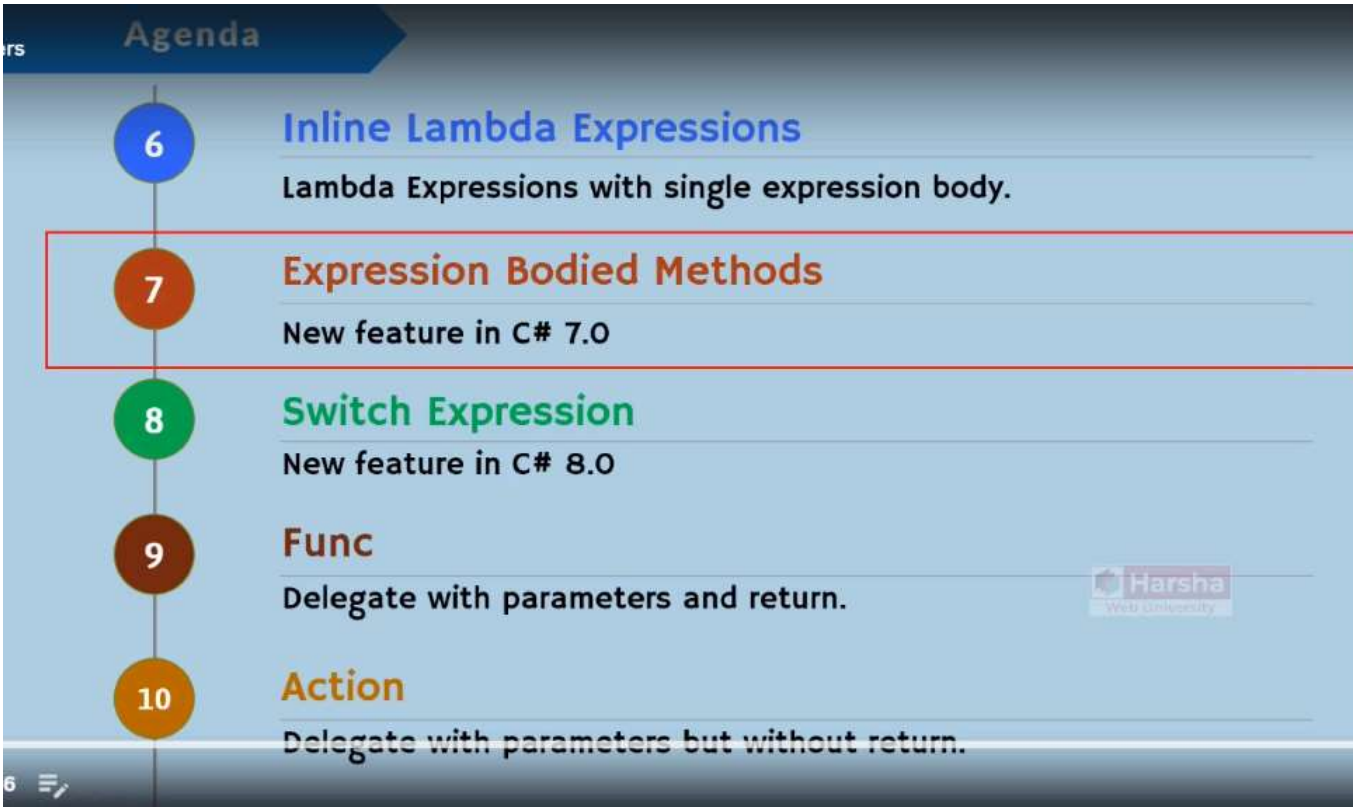
In real time applications, mostly we do not use Expression Trees Manually. But we use it i
LINQ. That means the concept of LINQ internally uses Expression Trees. As a developer w
expression trees in the real time applications.



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

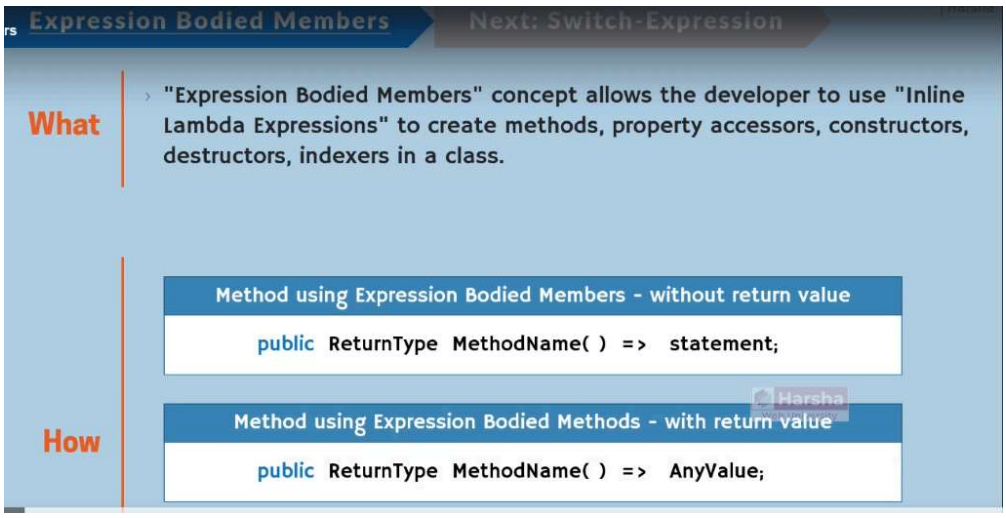
Whenever we invoke the expression tree, all the corresponding delegates will be execute
up approach. That means the child delegates will be first executed and then based on the
the parent delegate will be executed. For example, there is a child delegate that calculate
numbers and another child delegate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





As per expression bodied members, wherever you have single statement or single value a
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 { }.



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

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
}
```