

Section 6 : Methods

09 July 2024 15:53

Agenda

- 1 **Understanding Methods**
What is method and syntax of creating methods.
Access modifiers of methods: private, protected, private protected etc.
- 2 **Encapsulation**
What is encapsulation and how to implement it.
- 3 **Abstraction**
What is abstraction and how to implement it.
- 4 **Local Variables & Parameters**
Differences between local variables & parameters.
Scope of local variables & parameters.

Agenda

- 5 **"this" keyword**
What is "this" keyword & When to use it.
- 6 **Static Methods**
Instance Methods (vs) Static Methods
- 7 **Reference Variables as Arguments**
Passing reference variables as arguments to methods & receiving them.
- 8 **Default Arguments**
Assigning default values to parameters. New features in C# 4.0
- 9 **Named Arguments**
Passing argument values based on their names. New feature in C# 4.0

Introducing Methods Next: Syntax of Method

What

- > Method is a function (group of statements), to do some process based on fields.
- > Methods are parts of the class.
- > Methods can receive one or more input values as "parameters" and return a value as "return".

```
class Car
{
    int calculateEmi( int carPrice, int noOfMonths, int interestRate )
    {
        //do calculation here
        return (emi);
    }
}
```

Syntax Next: Access Modifiers of Methods

SAME AS FIELDS

↓

- 1. private
- 2. protected
- 3. private protected
- 4. internal
- 5. protected internal
- 6. public

- 1. static
- 2. virtual
- 3. abstract
- 4. override
- 5. new
- 6. partial
- 7. sealed

accessModifier modifier returnType MethodName(parameter1, parameter2, ...)
{
 Method body here
}

Data type of return value (result) of the method.

Input values received by the method.

What

- Access Modifiers (a.k.a. "Access Specifier" or "Visibility Modifier) of methods, are same as access modifiers of fields.

Access Modifier	In the same class	In the child classes at the same assembly	In the other classes at the same assembly	Child classes at other assembly	Other classes at other assembly
private	Yes	No	No	No	No
protected	Yes	Yes	No	Yes	No
private protected	Yes	Yes	No	No	No
internal	Yes	Yes	Yes	No	No
protected internal	Yes	Yes	Yes	Yes	No
public	Yes	Yes	Yes	Yes	Yes

Harsha

Web University

Encapsulation**Next: Local Variables and Parameters**

Harsha

What

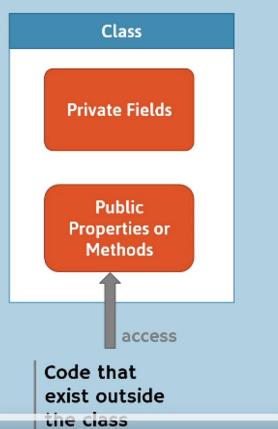
- Encapsulation is a concept of:
 - Bundling the data (fields) and operations (methods) that manipulate the data together.
 - Hides internal implementation details of an object and provide essential members to interact with them.

Benefits

- Modularity
- Hiding implementation details
- Data Integrity

Implemented using:

- Private fields &
- Public properties or public methods

**Agenda**

Harsha

1

Understanding Methods

What is method and syntax of creating methods.

Access modifiers of methods: private, protected, private protected etc.

2

Encapsulation

What is encapsulation and how to implement it.

3

Abstraction

What is abstraction and how to implement it.

4

Local Variables & Parameters

Differences between local variables & parameters.

Scope of local variables & parameters.

Local Variables and Parameters → Next: "this" keyword

```

class ClassName
{
    returnType MethodName(dataType parameterName)
    {
        dataType variableName;
    }
}

```

Parameter
Value will be received from the method caller.

Local Variable
Value will be initialized in the method itself.

Harsha
Web University

Local Variables and Parameters → Next: "this" keyword

Parameters <ul style="list-style-type: none"> > The variables that are being received from the "method caller" are called as "parameters". > The parameters are stored in the Stack of the method. > For every method call, a new stack will be created. 	Local Variables <ul style="list-style-type: none"> > The variables that are declared inside the method are called as "Local variables". Local variables can be used only within the same method. > Local variables are stored in the same stack, just like parameters. > The stack will be deleted at the end of method execution. So all local variables and parameters will be deleted.
---	--

Harsha
Web University

Agenda

- 5 **"this" keyword**
What is "this" keyword & When to use it.
- 6 **Static Methods**
Instance Methods (vs) Static Methods
- 7 **Reference Variables as Arguments**
Passing reference variables as arguments to methods & receiving them.
- 8 **Default Arguments**
Assigning default values to parameters. New features in C# 4.0
- 9 **Named Arguments**
Passing argument values based on their names. New feature in C# 4.0

"this" keyword

Next: Static Methods

Harsha

What

- > The "this" keyword refers to "current object", which method has invoked the method.
- > The "this" keyword is available only within the "instance methods".

```

Object 1          Object 2
                         ↓
Method
AccessModifier   Modifier   returnType   MethodName( parameter1, parameter2, ... )
{
    this → Object 2
}
  
```

Agenda

- 5 "this" keyword
What is "this" keyword & When to use it.
- 6 Static Methods
Instance Methods (vs) Static Methods
- 7 Reference Variables as Arguments
Passing reference variables as arguments to methods & receiving them.
- 8 Default Arguments
Assigning default values to parameters. New features in C# 4.0
- 9 Named Arguments
Passing argument values based on their names. New feature in C# 4.0

Instance Methods (vs) Static Methods

Next: Reference Variables as Arguments

	Instance Methods	Static Methods
Association	> Associated with Objects	> Associated with class.
Manipulates	> Manipulates instance fields.	> Manipulates static fields.
Declaration	> Declared without "static" keyword. > returnType methodName() { }	> Declared with "static" keyword. > static returnType methodName() { }
Accessible with	> Accessible with object (through reference variable).	> Accessible with class name only (not with object).

36

Instance Methods (vs) Static Methods		Next: Reference Variables as Arguments
	Instance Methods	Static Methods
Can access (fields)	> Can access both instance fields and static fields.	> Can't access instance fields; but can access static fields only.
Can access (methods)	> Can access both instance methods and static methods.	> Can't access instance methods; but can access static methods only.
"this" keyword	> Can use "this" keyword, as there must be "current object" to call instance method.	> Can't use "this" keyword, as there is NO "current object" while calling instance methods.

Agenda

- 5 "this" keyword
What is "this" keyword & When to use it.
- 6 Static Methods
Instance Methods (vs) Static Methods
- 7 Reference Variables as Arguments
Passing reference variables as arguments to methods & receiving them.
- 8 Default Arguments
Assigning default values to parameters. New features in C# 4.0
- 9 Named Arguments
Passing argument values based on their names. New feature in C# 4.0

0:09 / 7:45

Reference Variables as Arguments

Stack of Caller Method

Stack of Callee Method

If you pass "reference variable" as argument, the reference (address) of object will be passed to the method.
The parameter's data type will be the class name.
If you make any changes to object in the method, the same will be affected automatically in the caller method, as you are accessing the same object.

Agenda

Harsha

- 5 "this" keyword
What is "this" keyword & When to use it.
- 6 Static Methods
Instance Methods (vs) Static Methods
- 7 Reference Variables as Arguments
Passing reference variables as arguments to methods & receiving them.
- 8 Default Arguments
Assigning default values to parameters. New features in C# 4.0
- 9 Named Arguments
Passing argument values based on their names. New feature in C# 4.0

Default Arguments

Next: Named Arguments

What

- > Default value of the parameter of a method.
- > If you don't pass value to the parameter, the default value gets assigned to the parameter.

Why

- > To avoid bothering to pass value to the parameter; instead, take some default value into the parameter automatically, if the method caller has not supplied value to the parameter.

Syntax

```
accessModifier modifier returnType MethodName( parameter1 = defaultValue )
{
    Method body here
}
```

Agenda

56. Named Arguments

Harsha

- 5 "this" keyword
What is "this" keyword & When to use it.
- 6 Static Methods
Instance Methods (vs) Static Methods
- 7 Reference Variables as Arguments
Passing reference variables as arguments to methods & receiving them.
- 8 Default Arguments
Assigning default values to parameters. New features in C# 4.0
- 9 Named Arguments
Passing argument values based on their names. New feature in C# 4.0

What

- > Supply value to the parameter, based on parameter name.
- > Syntax: parametername: value

Why

- > You can change order of parameters, while passing arguments.
- > Parameter names are expressive (understandable) at method-calling time.

SyntaxCalling a method:**MethodName(ParameterName : value, ParameterName : value);**

Useful if method has so many no. of parameters. That means more than 3

Agenda**10****Method Overloading**

Multiple methods with same name with different signatures.

11**Parameter Modifiers**

default, ref, out, in

12**Ref Returns**

New feature in C# 7.3

13**Local Functions**

New feature in C# 7.0

14**Static Local Functions**

New feature in C# 8.0



0:05 / 7:43

What

- > Writing multiple methods with same name in the same class, with different parameters.

Why

- > Caller would have several options, while calling a method.

Rule

- > Difference between parameters of all the methods that have same name, is MUST.

Example

```
MethodName( )
MethodName( int )
MethodName( string )
MethodName( int, string )
MethodName( string , int)
MethodName( string , string, int)
```



Agenda

- 10 **Method Overloading**
Multiple methods with same name with different signatures.
- 11 **Parameter Modifiers**
default, ref, out, in
- 12 **Ref Returns**
New feature in C# 7.3
- 13 **Local Functions**
New feature in C# 7.0
- 14 **Static Local Functions**
New feature in C# 8.0

0:06 / 5:39

Harsha Web University

Parameter Modifiers Next: ref returns

What › Specifies how the parameter receives a value.

List › Default [No keyword]
› ref
› out
› in
› params

Harsha Web University

Parameter Modifiers (default) Next: ref returns

How › The "Argument" will be assigned into the "Parameter" but not reverse.

```

    Method Caller
    MethodName( Argument1, ...)

    Method Definition
    AccessModifier Modifier ReturnDataType MethodName( DataType Parameter1, ... )
    {
    }
  
```

Harsha Web University

Parameter Modifiers (ref) Next: ref returns

How › The "Argument" will be assigned into the "Parameter" and vice versa.
› The Argument must be a variable and must be pre-initialized.

```

    Method Calling
    MethodName( ref Argument1, ...)

    Method Definition
    AccessModifier Modifier ReturnDataType MethodName( ref DataType Parameter1, ... )
    {
      ....
      Parameter1 = value;
    }
  
```

Harsha Web University

Parameter Modifiers (out) Next: ref returns

How

- The "Argument" will not be assigned into the "Parameter" but only reverse.
- The Argument must be a variable; The Argument can be un-initialized.

Method Calling: MethodName(**out** Argument1, ...)

Method Definition:

```
AccessModifier Modifier ReturnDataType MethodName(out DataType Parameter1, ...)
{
    ....
    Parameter1 = value;
}
```

A red arrow points from the 'out' keyword in the method definition back to the 'out' keyword in the method call.

If you want to return multiple values from a function, use out parameters.

'out' variable declaration Next: ref returns

How

- You can declare out variable directly while calling the method with 'out' parameter.
- New feature in C# 7.0.

Method Calling: MethodName(**out type** Argument1, ...)

Method Definition:

```
AccessModifier Modifier ReturnDataType MethodName(out DataType Parameter1, ...)
{
    ....
    Parameter1 = value;
}
```

A red arrow points from the 'out type' declaration in the method definition back to the 'out type' declaration in the method call.

Parameter Modifiers (in) Next: ref returns

How

- The "Argument" will be assigned into the "Parameter", but the parameter becomes read-only.
- We can't modify the value of parameter in the method; if you try to change, compile-time error will be shown.
- New feature of C# 7.2.

Method Calling: MethodName(**in** Argument1, ...)

Method Definition:

```
AccessModifier Modifier ReturnDataType MethodName(in DataType Parameter1, ...)
{
    ....
    Parameter1 = value; //error, we can't change the value of parameter
}
```

A red arrow points from the 'in' keyword in the method definition back to the 'in' keyword in the method call.

ref returns → Next: parms

Harsha

How

- > The reference of return variable will be assigned to receiving variable.
- > New feature in C# 7.3.

Method Calling

```
ref variable = MethodName()
```

Method Definition

```
AccessModifier Modifier ReturnType MethodName(..)
{
    return ref variable;
}
```

0:44 / 6:02

Agenda

- 10 Method Overloading
Multiple methods with same name with different signatures.
- 11 Parameter Modifiers
default, ref, out, in, params
- 12 Ref Returns
New feature in C# 7.3
- 13 Local Functions
New feature in C# 7.0
- 14 Static Local Functions
New feature in C# 8.0

Parameter Modifiers (params) → Next: Local Functions

Harsha

How

- > All the set of arguments will be at-a-time received as an array into the parameter.
- > The "params" parameter modifier can be used only for the last parameter of the method; and can be used only once for one method.

Method Calling

```
MethodName( Argument1, Argument2, Argument3 ...)
```

Method Definition

```
AccessModifier Modifier ReturnType MethodName( params DataType[] Parameter1, ... )
{
    ....
    Parameter1[index] //To access value based on index
}
```

Agenda

- 10 Method Overloading
Multiple methods with same name with different signatures.
- 11 Parameter Modifiers
default, ref, out, in, params
- 12 Ref Returns
New feature in C# 7.3
- 13 Local Functions
New feature in C# 7.0
- 14 Static Local Functions
New feature in C# 8.0

Harsha

0:02 / 8:25

- > "Local functions" are functions, to do some small process, which is written inside a method.
- > Local functions are not part of the class; they can't be called directly through reference variable.
- > Local functions don't support "access modifiers" and "modifiers".
- > Local functions support parameters, return.

Local Function [Inside a method]

```
public void MethodName( param1, param2, ... )
{
    LocalFunctionName(); //Calling the Local Function

    ReturnDataType LocalFunctionName( param1, param2, ... )
    {
        //Local Function Body Here
    }
}
```



Agenda

10

Method Overloading

Multiple methods with same name with different signatures.

11

Parameter Modifiers

default, ref, out, in, params

12

Ref Returns

New feature in C# 7.3

13

Local Functions

New feature in C# 7.0

14

Static Local Functions

New feature in C# 8.0

0:07 / 7:40

Harsh



Static Local Functions

- > "Static Local functions" are functions, same as normal "Local Functions".
- > Only the difference is, static local functions can't access local variables or parameters of containing method.
- > This is to avoid accidental access of local variables or parameters of containing method, inside the local function.

Local Function [Inside a method]

```
public void MethodName( param1, param2, ... )
{
    LocalFunctionName(); //Calling the Local Function

    static ReturnDataType LocalFunctionName( param1, param2, ... )
    {
        //We can't access local variables or parameters of containing method.
    }
}
```



It's different than normal static method.



- › A method calls itself.
- › Useful in mathematic computations, such as finding factorial of a number.

Recursive Method

```
public void MethodName( )
{
    if (condition)
    {
        MethodName(); //Calling the same method
    }
}
```



- › A method calls itself.
- › Useful in mathematic computations, such as finding factorial of a number.

Recursive Method

```
public void MethodName( )
{
    if (condition)
    {
        MethodName(); //Calling the same method
    }
}
```

Key points to remember



- › Method is a part of class, that contains collection of statements to do some process.
- › Access modifiers of Methods: private, protected, private protected, internal, protected internal, public.
- › Modifiers of Methods: static, virtual, abstract, override, new, partial, sealed
- › For each method call, a new stack will be created; all local variables and parameters of the method will be stored in that stack; will be deleted automatically at the end of method execution.
- › In instance methods, the 'this' keyword refers to current object, based on which the method is called'.

Key points to remember



- › Instance methods can access & manipulate instance fields & static fields; Static methods can access only static fields.
- › But static method can create an object for the class; then access instance fields through that object.
- › Using named arguments , you can change order of parameters while calling the method.
- › Method Overloading is 'writing multiple methods with same name in the same class with different set of parameters'.
- › The 'ref' parameter is used to receive value into the method and also return some value back to the method caller; The 'out' parameter is only used to return value back to the method caller; but not for receiving value into the method.

```
public class MyClass
{
    private int nonStaticField = 42;
    private static int staticField = 100;

    private void NonStaticMethod()
    {
        Console.WriteLine("Non-static method called.");
    }

    public static void StaticMethod()
    {
        MyClass instance = new MyClass(); // Creating an instance of the class
        Console.WriteLine(instance.nonStaticField); // Accessing the non-static field
        instance.NonStaticMethod(); // Calling the non-static method

        // Accessing static fields directly
        Console.WriteLine(staticField);
    }
}

class Program
{
    static void Main()
    {
        MyClass.StaticMethod(); // Calling the static method
    }
}
```