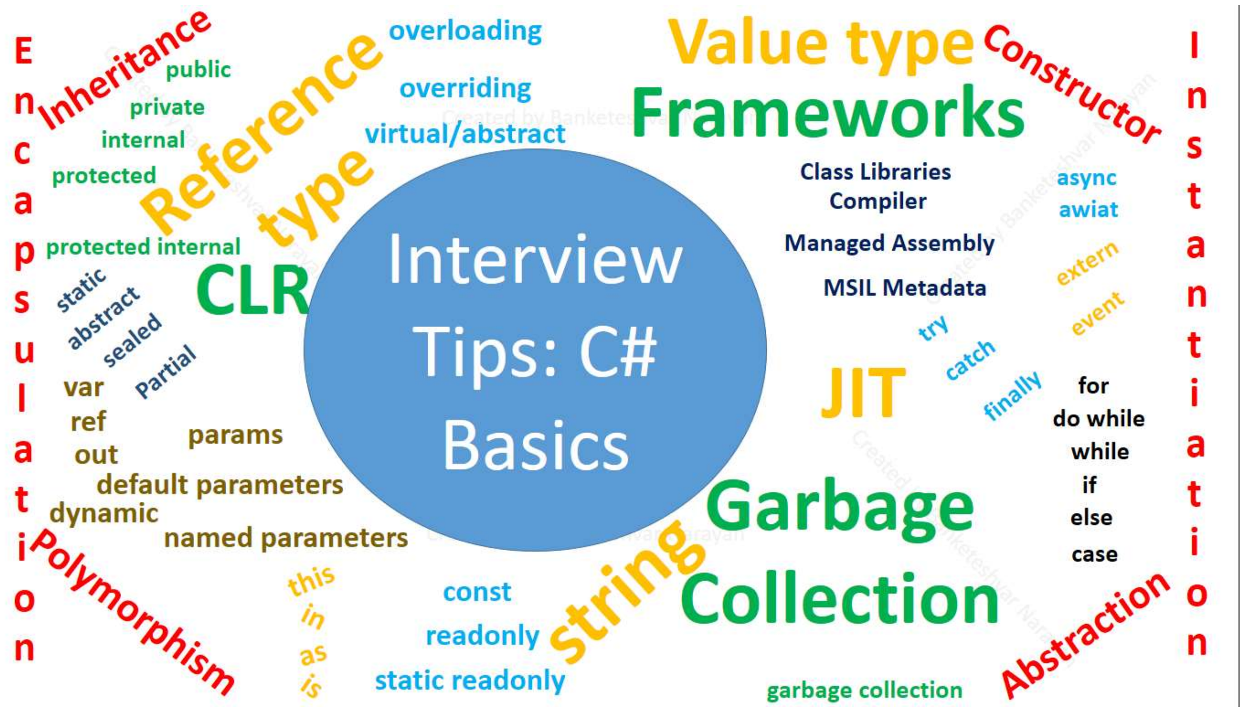
Basic Interview Tips in C#

By Banketeshvar Narayan

https://www.facebook.com/banketeshvar.narayan
https://twitter.com/BNarayanSharma

Basic Interview Tips in C#



In this article, I am going to explain about following C# basics concepts.

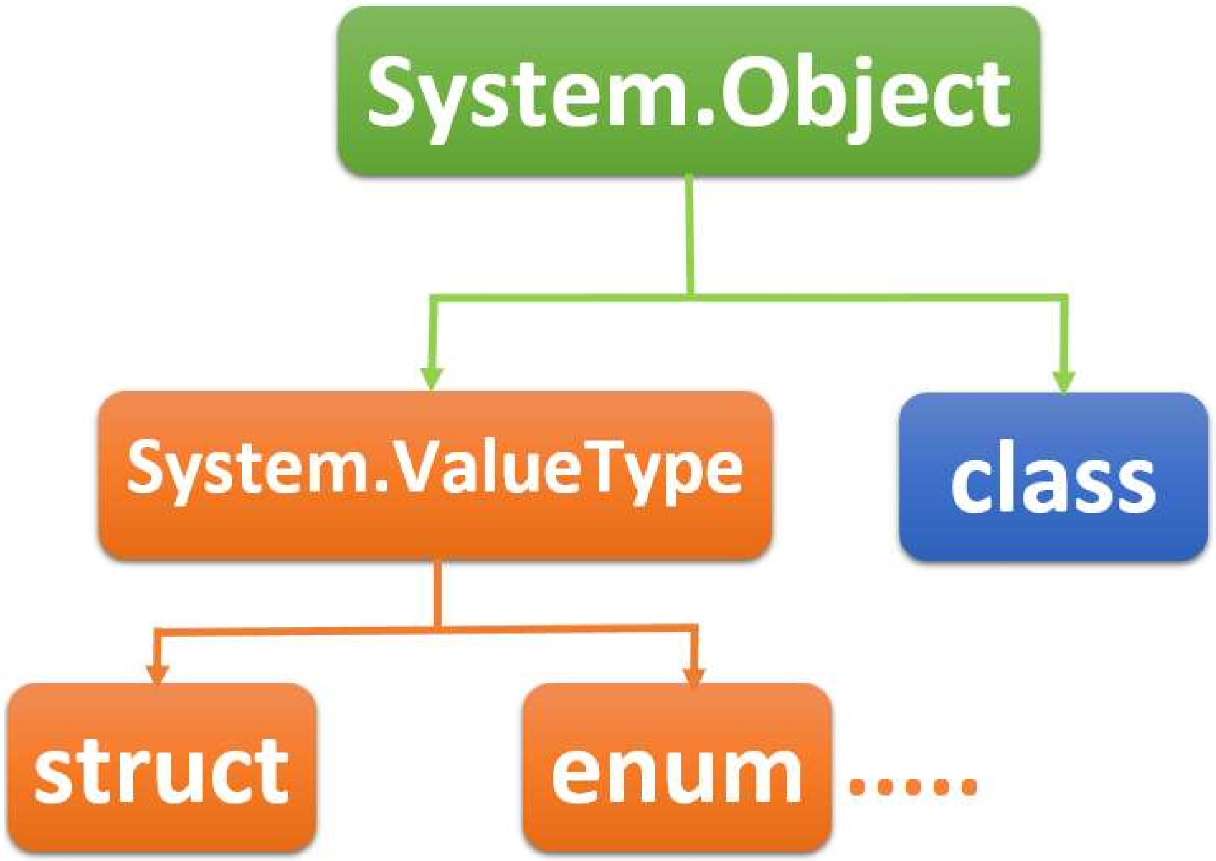
1. Value types & reference types
2. Access Modifiers in C#
3. Understanding Classes and other types
   a. class (Abstract, partial, sealed, static etc.)
   b. enum
   c. struct
   d. interface
4. Understanding Partial Types and Partial Methods.
5. use of ‘typeof’ keyword in C#.
6. Calling multiple constructor of a class with single object creation.
7. Understanding the output and behaviour of C# basic code snippet.
8. Implement and call methods of interfaces & base class with same name.
9. Usage of ‘ref’ & ‘out’ keyword in C#
10. Using ‘params’ & ‘dynamic’ in method parameter.

I will be explaining all the above topics in depth but explanation will be more focused on interview preparations i.e. I will explain all those things in the context of an interview. Actually, in recent past, I have gone through dozens of interviews and I would like to share those experiences with other developers.

I am going to start with very basic concepts and the purpose of this article is that readers of this article can make themselves ready for C# interviews.

Value types & reference types

In C#, types have been divided into 2 parts value types and reference types. Value types directly stores the data in the stack portion of the RAM whereas reference type store reference of data in the stack and saves actual data in heap. But this is a half-truth statement. I will explain about it later to prove that why it's half-truth.



Value types

A value type contains its data directly and it is stored in heap portion of the RAM. Example of some built-in value types

Numeric types (sbyte, byte, short, int, long, float, double, decimal), char, bool, IntPtr, date…

Struct and Enum is value type.

If I need to create a new value type, then I can use structs to create a new value type. I can also use enum keyword to create value type of enum type.

reference types

Reference types store address of their data i.e. pointer on the stack and actual data is stored in heap area of the memory. The heap memory is managed by garbage collector.

As I said that reference types do not store their data directly so assigning a reference type value to another reference type variable create a copy of reference address (pointer) and assign it to 2nd reference variable. So now both reference variables having the same address or pointer for which actual data is stored in heap. If I make any change in one variable, then that value also will be affected in another variable.

But in some cases, it is not valid like when reallocating memory, it may update only one variable for which new memory allocation has been done.

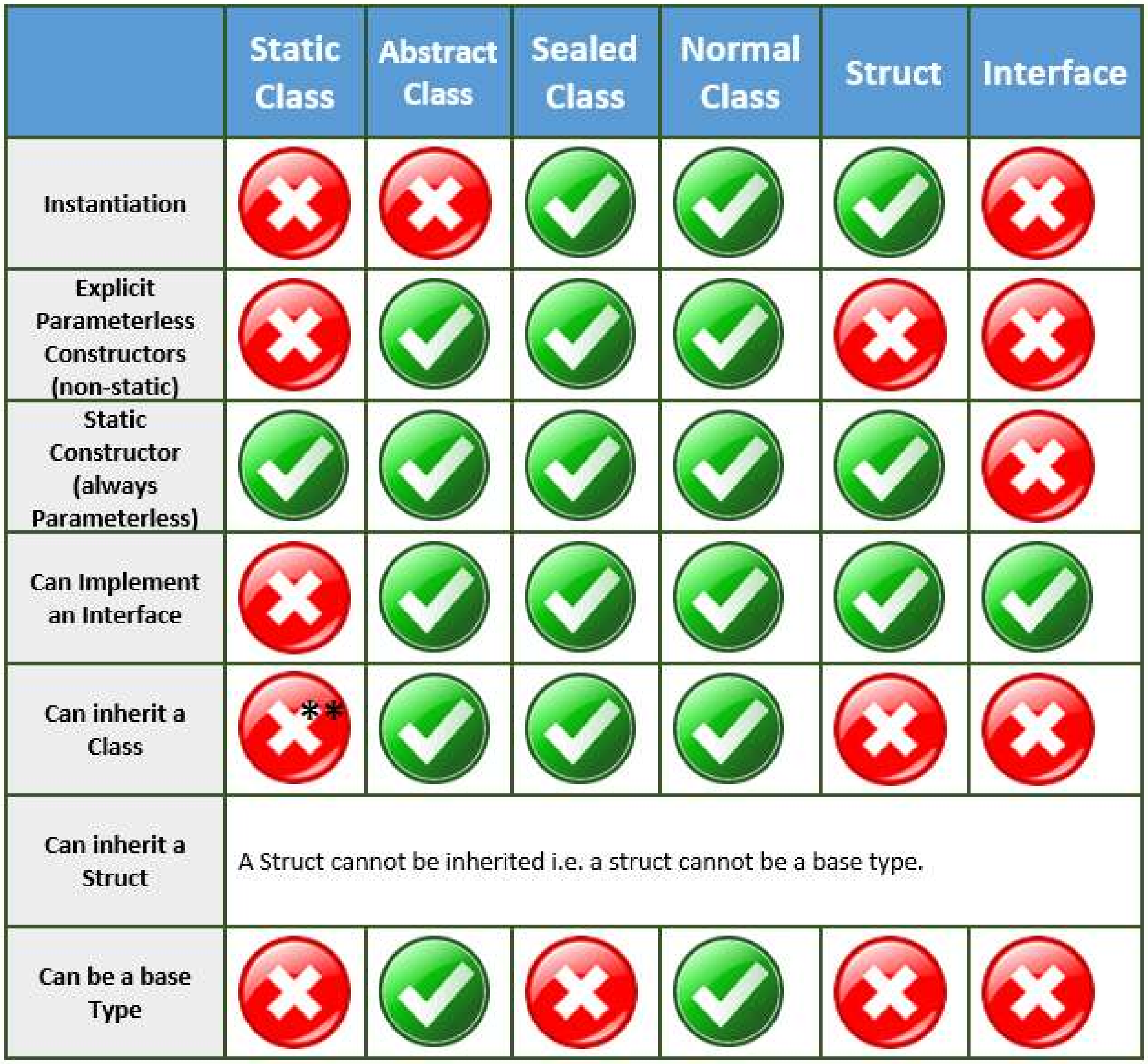
Suppose you are passing an object of customer class let say “objCustomer” inside a method “Modify(…)” and then re-allocating the memory inside the “Modify(…)” method & providing its properties value in that case customer object “objCustomer” inside the “Modify(…)” will have different value and those change will not be available to the variable “objCustomer” accessed from outside of “Modify(…)”.

You can also take the example of String class. ‘string’ is the best example for this scenario because in case of string each time new memory is allocated.

I will explain about all those scenarios in depth in “Usage of ‘ref’ & ‘out’ keyword in C#” & “Understanding the behaviour of ‘string’ in C#” section of this article.

Understanding Classes and other types

Below is a table for “Understanding Classes and other types”. If you are aware about all the behaviours of classes and other types, then you can skip this section and move to next section. I will explain all those things one by one for the developers who are not aware about it.

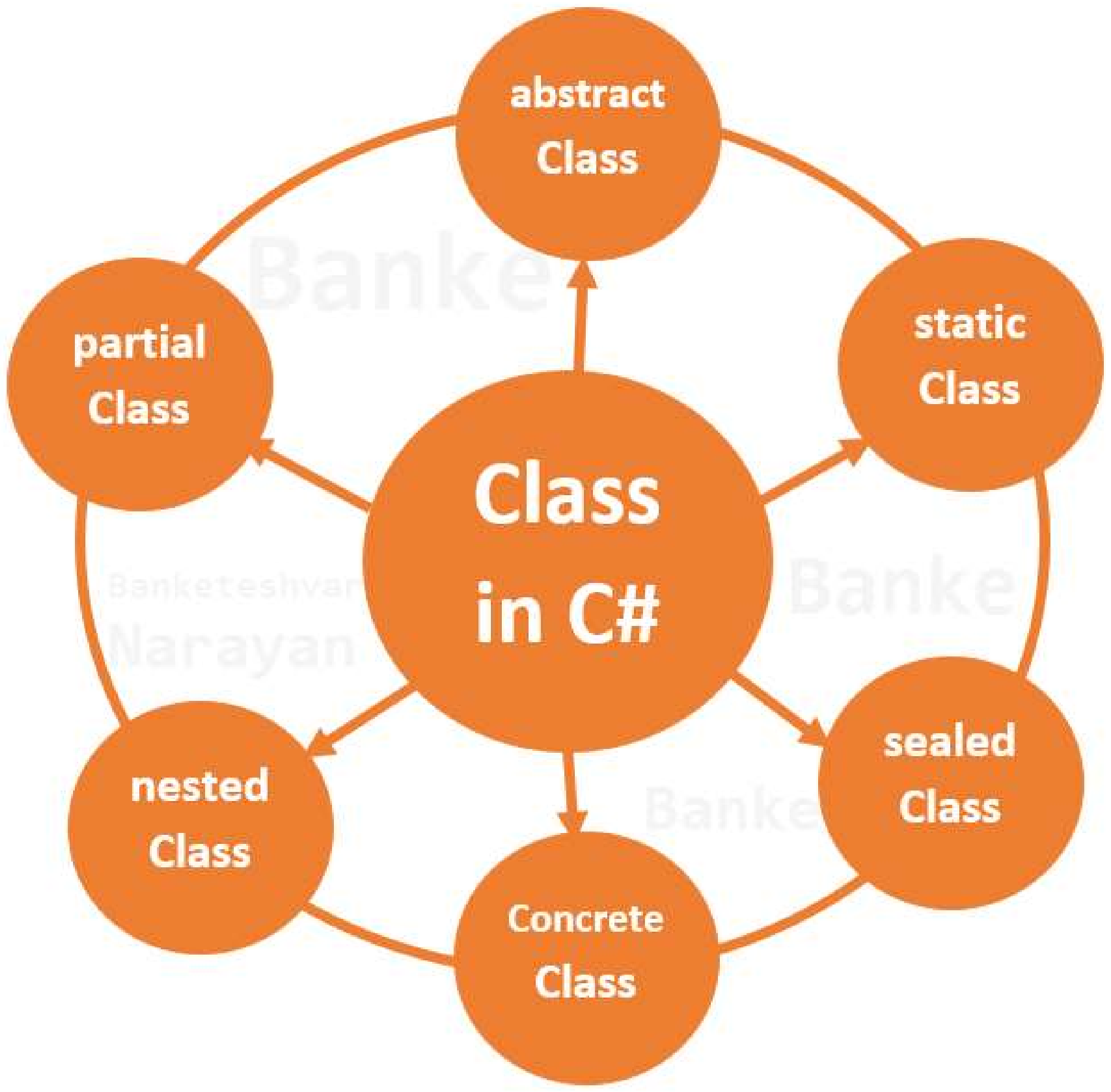


	Static Class	Abstract Class	Sealed Class	Normal Class	Struct	Interface
Instantiation	✗	✗	✓	✓	✓	✗
Explicit Parameterless Constructors (non-static)	✗	✓	✓	✓	✗	✗
Static Constructor (always Parameterless)	✓	✓	✓	✓	✓	✗
Can Implement an Interface	✗	✓	✓	✓	✓	✓
Can inherit a Class	✗**	✓	✓	✓	✗	✗
Can inherit a Struct	A Struct cannot be inherited i.e. a struct cannot be a base type.					
Can be a base Type	✗	✓	✗	✓	✗	✗

Note: you can see that for "static class can inherit a class" I have written no but used the * notation with that. I t means a static class cannot inherit any other class but must inherit System.Object. so, we can also write the System.Object Class name in the inheritance list but there is no need to write the name System.Object explicitly because all the classes are being inherited from System.Object implicitly and we do not need to write it explicitly.

What are the different types of classes we can have?

We can have abstract class, partial class, static class, sealed class, nested class & concrete class (simple class).



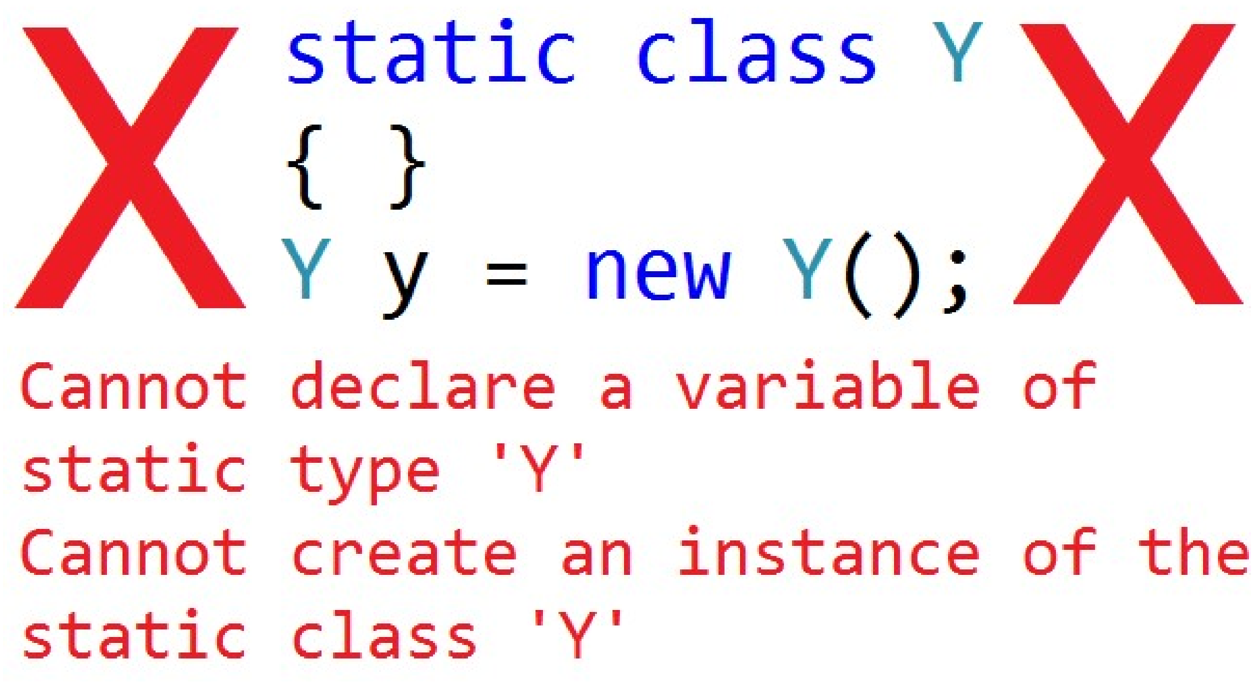
Static Class

```
static class Student
    {
        public static int StudentId { get; set; }
        public static string StudentName { get; set; }
    }
```

Static Partial Class

```
    public  static partial class Student
    {
        public static int StudentId { get; set; } = 1;

        public new static string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }
    }
    public static partial class Student
    {
        public static string StudentName { get; set; } = "Banketeshvar Narayan";
    }
```

A. Cannot create an instance of the static class



```
static class Y
{ }
Y y = new Y();
```

Cannot declare a variable of static type 'Y'
Cannot create an instance of the static class 'Y'

B. Static classes cannot have instance constructors but it can have static constructor.

The below code snippet will give compile time error

```
static class MyStaticClass
{
    MyStaticClass()
    {

    }
}
```

Error: CS0710 Static classes cannot have instance constructors

Whereas the below code snippet will compile successfully

```
static class MyStaticClass
   {
       static MyStaticClass()
       {
       }
   }
```

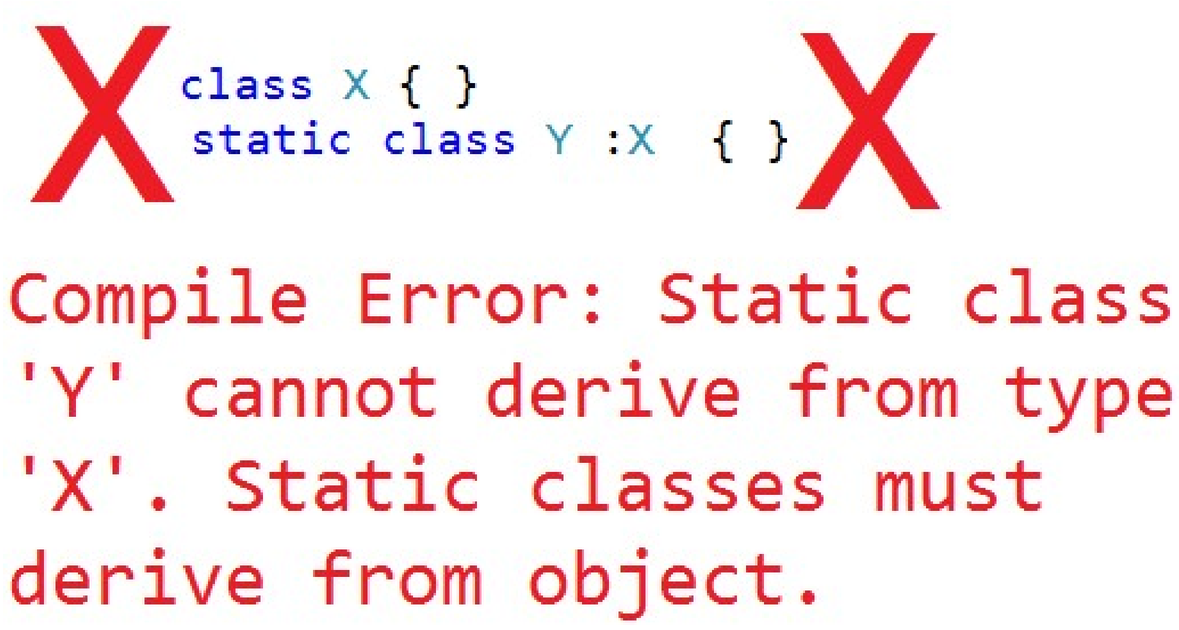
C. **static classes cannot implement interfaces**

The below code snippet will give compile time error

```
static class MyStaticClass: IInterface
    {
        static MyStaticClass()
        {
        }
    }
```

Error: CS0714 'MyStaticClass': static classes cannot implement interfaces

D. **Static class cannot derive from type. Static classes must derive from object.**



```
class X { }
 static class Y :X  { }
```

Compile Error: Static class 'Y' cannot derive from type 'X'. Static classes must derive from object.

E. **Static class cannot be a base class.**

The below code snippet will give the compile time error

```
static class MyStaticClass1
    {
      //your code
    }

    class MyClass2: MyStaticClass1
    {
        //your code
    }
```

Error CS0709 'MyClass2': cannot derive from static class 'MyStaticClass1'

Abstract Class

```
abstract class Student
    {
        public int StudentId { get; set; }
        public string StudentName { get; set; }

        public abstract void Add();

    }
```

Abstract Partial Class

```
  public abstract partial class Student
    {
        public static int StudentId { get; set; } = 1;

        public override string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }
    }
    public abstract partial class Student
    {
        public static string StudentName { get; set; } = "Banketeshvar Narayan";
    }
```

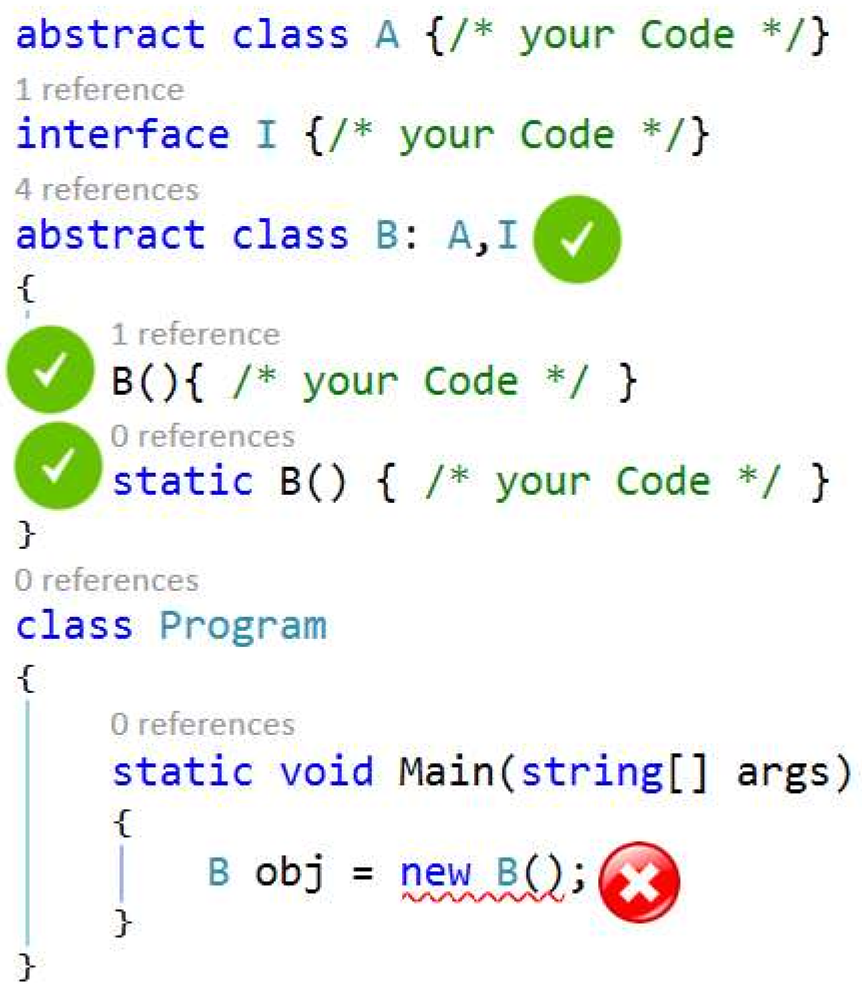
A. **Cannot create an instance of abstract class i.e. abstract class cannot be instantiated.**

The below code snippet will give the compile time error

```
abstract class MyClass
    {
        //Your code
    }
    class Program
    {
        static void Main(string[] args)
        {
            MyClass obj = new MyClass();
        }
    }
```

Error   CS0144 Cannot create an instance of the abstract class or interface 'MyClass'

B. Abstract class can have non-static constructor and static constructor as well. It can implement an interface and can derive from another class. Abstract class can be a base class also. Below is sample code image of code snippet for the same.



```
abstract class A {/* your Code */}
1 reference
interface I {/* your Code */}
4 references
abstract class B: A,I ✓
{
 ✓  1 reference
    B(){ /* your Code */ }
 ✓  0 references
    static B() { /* your Code */ }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        B obj = new B(); ✗
    }
}
```

Sealed Class

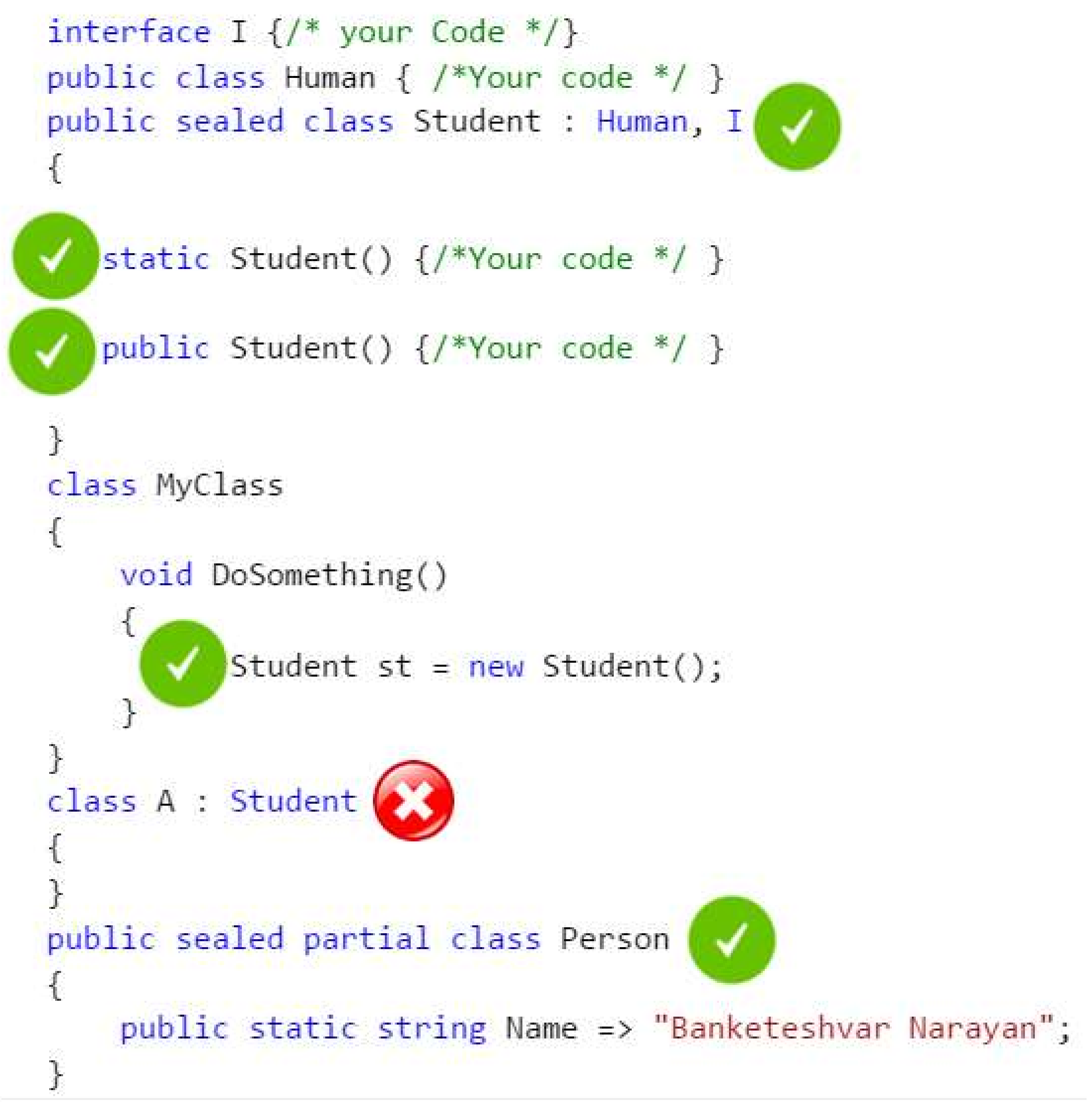
```
sealed class Student
    {
        public int StudentId { get; set; } = 1;
        public string StudentName { get; set; } = "Banketeshvar Narayan";

        public override string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }

           }
```

Sealed Partial Class

```
public sealed partial class Student
    {
        public static int StudentId { get; set; } = 1;

        public override string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }
    }
    public sealed partial class Student
    {
        public static string StudentName { get; set; } = "Banketeshvar Narayan";
    }
```

Sealed class cannot be further inherited i.e. it cannot be a base class for any type but it can inherit a class and can be a child class. It can be instantiated, can have static constructor, instance constructor and it can implement an interface also. It can be a partial class.

Below is the pictorial representation of the above statement.



```
interface I {/* your Code */}
public class Human { /*Your code */ }
public sealed class Student : Human, I ✓
{

✓ static Student() {/*Your code */ }

✓ public Student() {/*Your code */ }

}
class MyClass
{
    void DoSomething()
    {
     ✓ Student st = new Student();
    }
}
class A : Student ✗
{
}
public sealed partial class Person ✓
{
    public static string Name => "Banketeshvar Narayan";
}
```

If you would like to test the statement written in the pictorial representation, then you can use below code snippet

```
namespace SealedClassExamples
{
    interface I {/* your Code */}
    public class Human { /*Your code */ }
    public sealed class Student : Human, I
    {
       static Student() {/*Your code */ }
       public Student() {/*Your code */ }

    }
    class MyClass
    {
        void DoSomething()
        {
            Student st = new Student();
        }
    }
    class A : Student
    {

    }
    public sealed partial class Person
    {
        public static string Name => "Banketeshvar Narayan";
    }

}
```

Concrete Class (Simple Class)

```
class Student
    {
        public int StudentId { get; set; } = 1;
        public string StudentName { get; set; } = "Banketeshvar Narayan";

        public override string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }

    }
```

Partial Concrete Class

```
 public partial class Student

  {

    public int StudentId { get; set; } = 1;


        public override string ToString()
        {
            return $"Student Id is {StudentId:0000} and name is {StudentName}";
        }

    }
    public partial class Student
    {
        public string StudentName { get; set; } = "Banketeshvar Narayan";
    }
```

Concrete class or simple class is the class which do not use any restriction like static, sealed or abstract class.

Concrete classes can be a base class, child class, can be instantiated, can have static constructor, can instance constructor & can implement an interface.

Partial Method in C#

```
partial class Class1
    {
        partial void DoSomething();

    }
    partial class Class1
    {
        partial void DoSomething()
        {
            WriteLine("Hello");
        }
    }
```

We can have partial method in C# but there are some restrictions

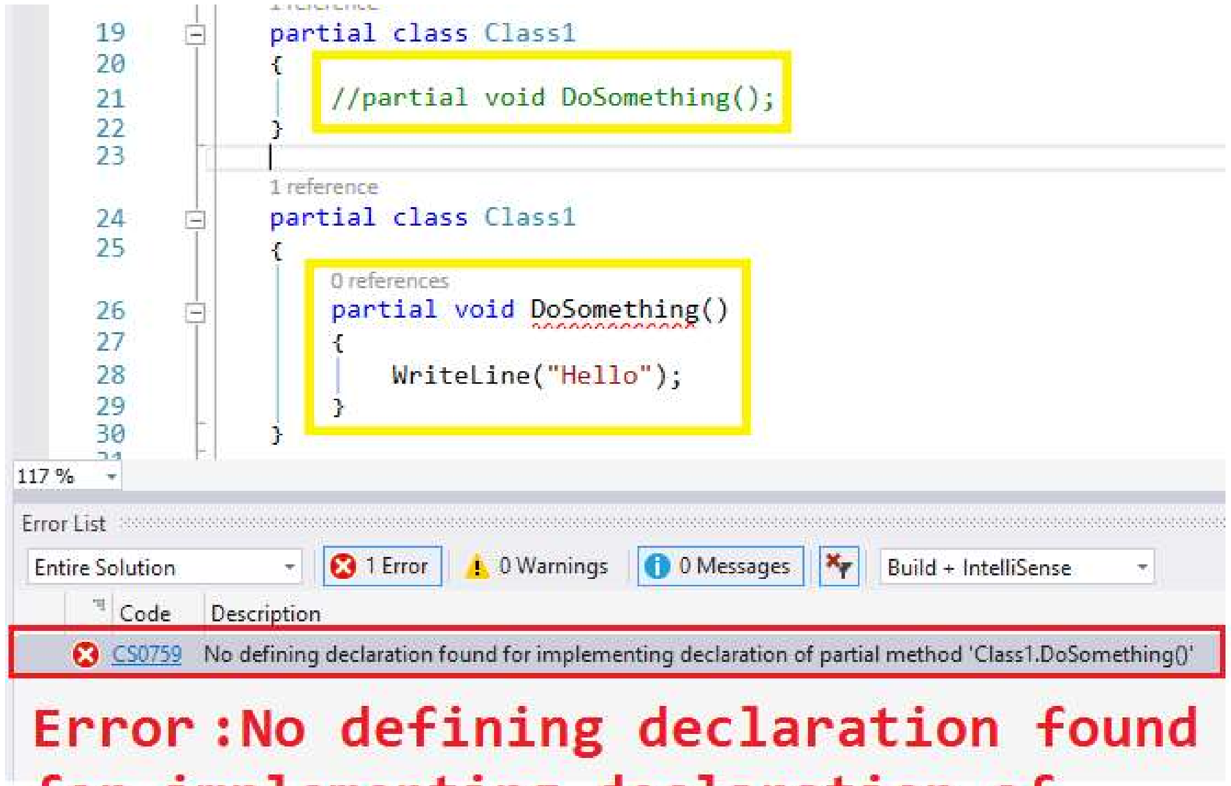
1. partial method must be declared already before implementing it

If you are going to implement the partial method then defining declaration must be available for the same i.e. partial method must be declared already before implementing it.

i.e. the below code will give compile time error

```
partial class Class1
   {
       //partial void DoSomething();
   }

   partial class Class1
   {
       partial void DoSomething()
       {
           WriteLine("Hello");
       }
   }
```

Error CS0759 : No defining declaration found for implementing declaration of partial method 'Class1.DoSomething()'



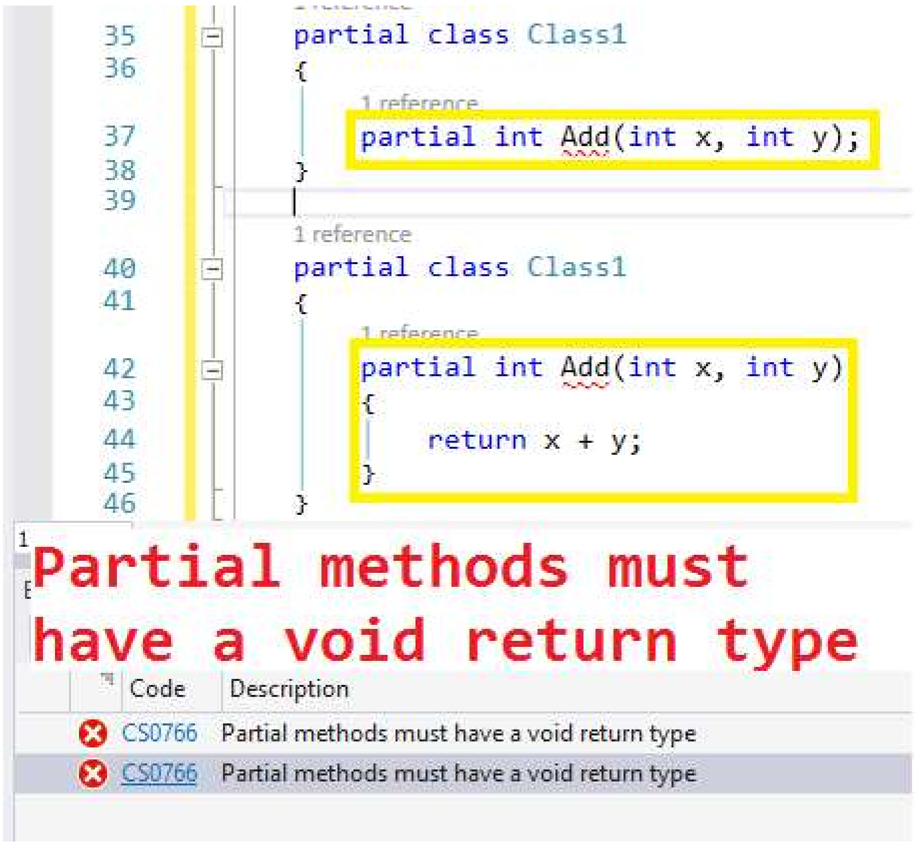
```
partial class Class1
{
    //partial void DoSomething();
}

1 reference
partial class Class1
{
    0 references
    partial void DoSomething()
    {
        WriteLine("Hello");
    }
}
```

```
CS0759   No defining declaration found for implementing declaration of partial method 'Class1.DoSomething()'
```

Error :No defining declaration found for implementing declaration of partial method 'Class1.DoSomething()'

2. Partial methods must have a void return type. The below code will give compile time error

```
partial class Class1
    {
        partial int Add(int x, int y);
    }

    partial class Class1
    {
        partial int Add(int x, int y)
        {
            return x + y;
        }
    }
```



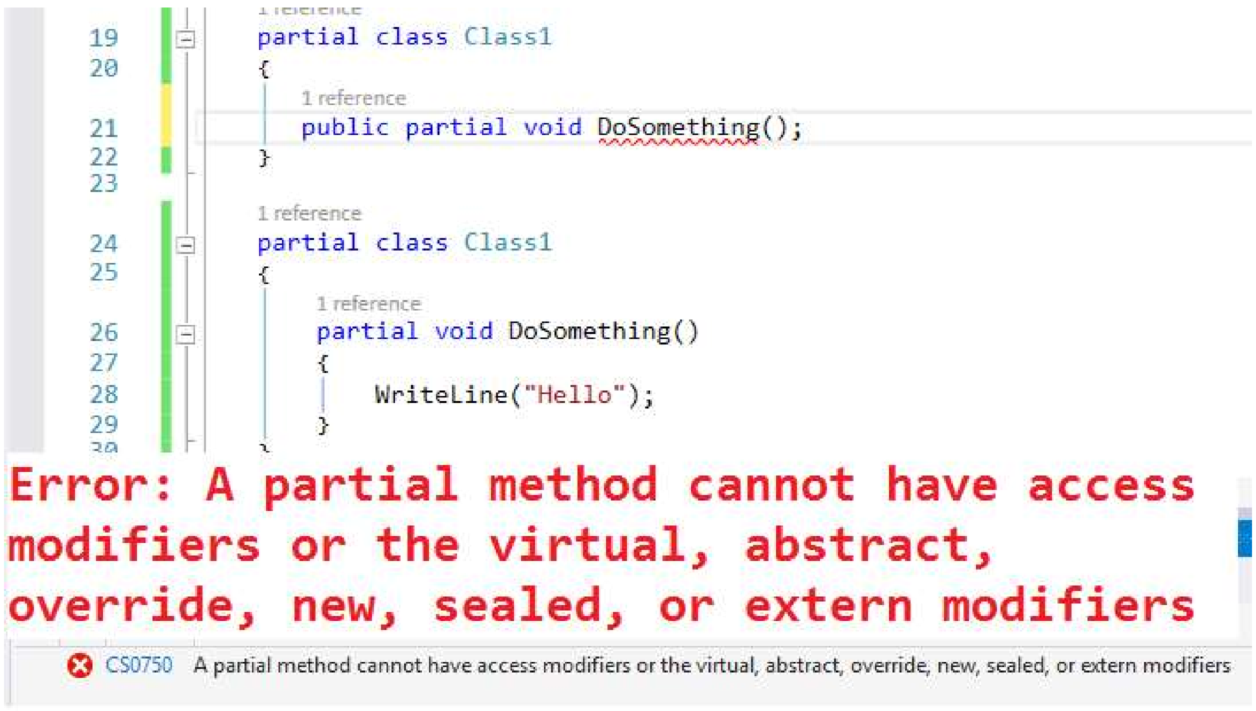
```
partial class Class1
{
    partial int Add(int x, int y);
}

partial class Class1
{
    partial int Add(int x, int y)
    {
        return x + y;
    }
}
```

Partial methods must have a void return type

Code	Description
CS0766	Partial methods must have a void return type
CS0766	Partial methods must have a void return type

3. Accessibility of partial method is private and no access modifier is allowed.

```
partial class Class1
{
    public partial void DoSomething();
    /* Compile Time Error: Error
   * CS0750   A partial method cannot have access modifiers
   * or the virtual, abstract, override, new, sealed,
   * or extern modifiers
   */
}
partial class Class1
{
    partial void DoSomething()
    {
        WriteLine("Hello");
    }
}
```



```
1 reference
partial class Class1
{
    1 reference
    public partial void DoSomething();
}

1 reference
partial class Class1
{
    1 reference
    partial void DoSomething()
    {
        WriteLine("Hello");
    }
```

Error: A partial method cannot have access modifiers or the virtual, abstract, override, new, sealed, or extern modifiers

CS0750 A partial method cannot have access modifiers or the virtual, abstract, override, new, sealed, or extern modifiers

4. Signature of the both partial methods should be the same

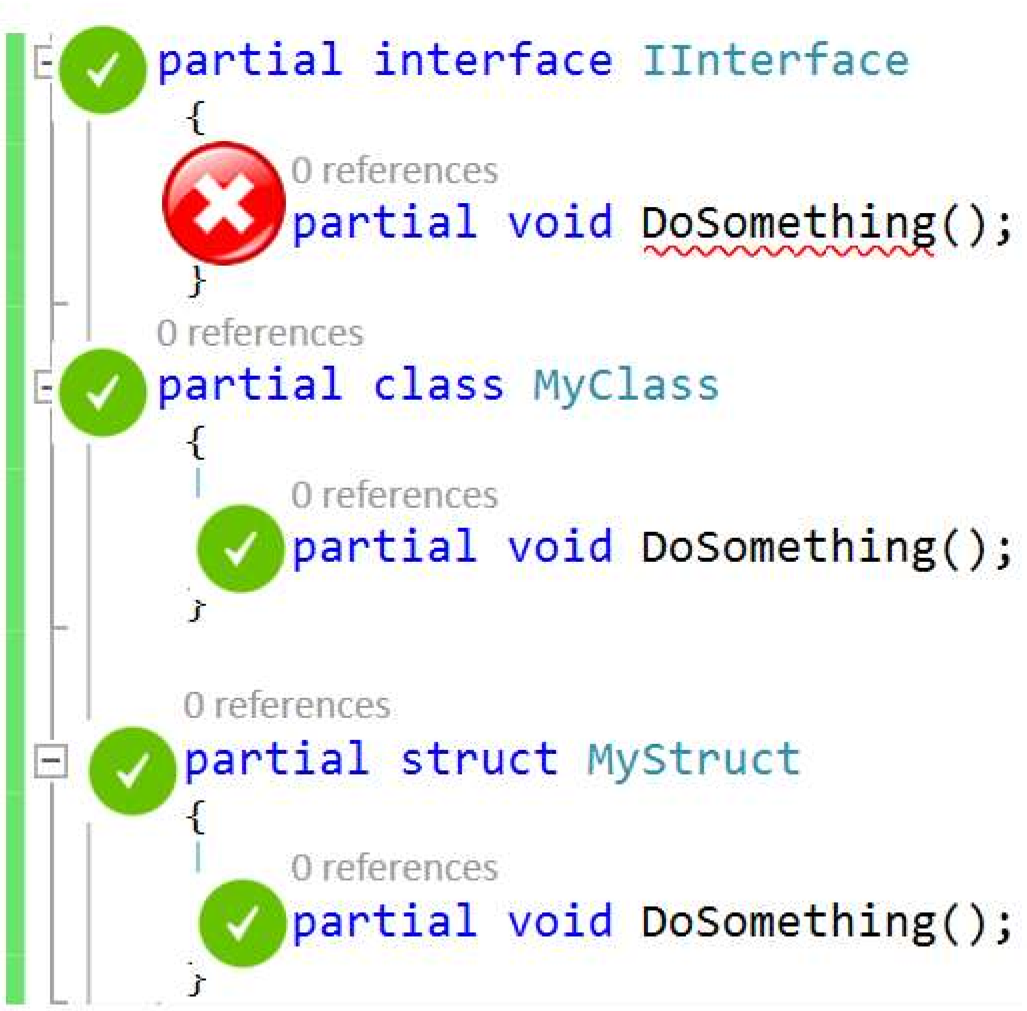
The below code will give the compile time error because signature does not match.

```
partial class Class1
   {
       partial void Add(int x, int y);
   }

   partial class Class1
   {
       partial void Add(int x, int y, int z)
       {
           WriteLine($"{x + y}");
       }

       /* Compilee Error: Error   CS0759   No defining declaration found for
        * implementing declaration of partial method 'Class1.Add(int, int, int)'
        */
   }
```

5. We can declare a partial method inside the partial class or inside the partial struct.
That is the type in which I am going to declare a partial method must be partial. Following type can be declared as partial

   a. Partial Interface
   b. Partial Class
   c. Partial Structure

So, you can see that we can have partial interface, partial class & partial struct but partial method can only be written inside partial struct and partial class.



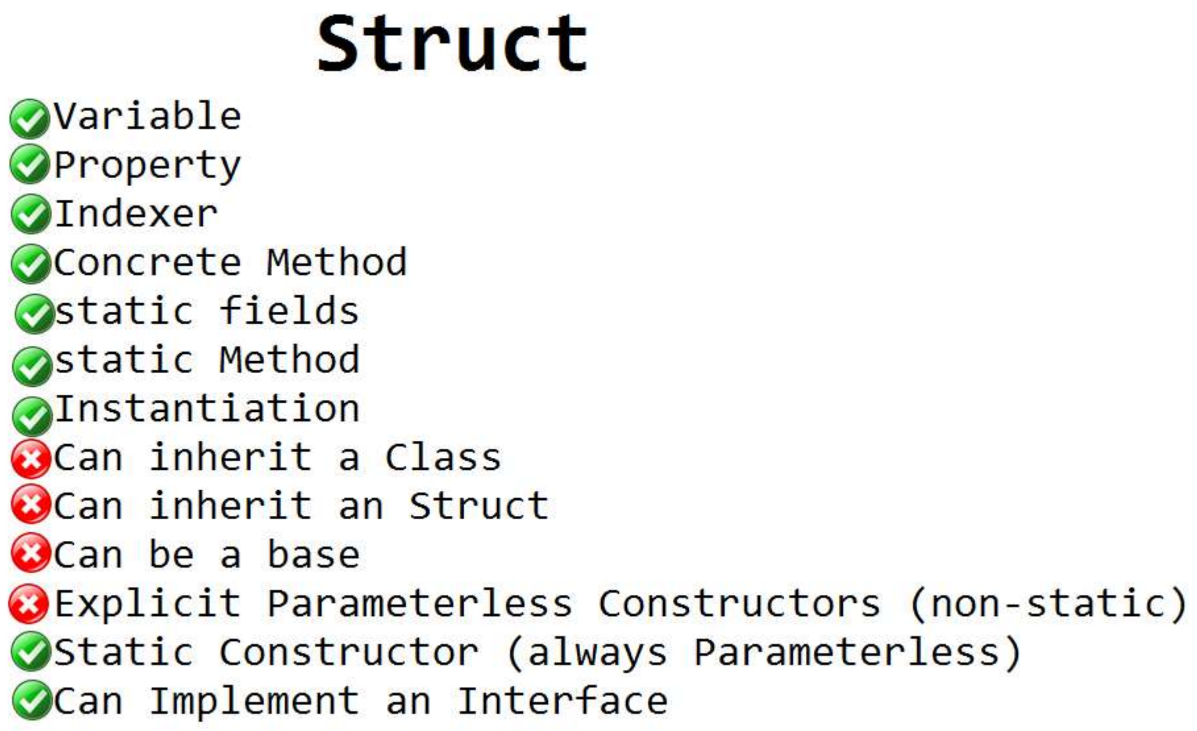
```
partial interface IInterface
{
    0 references
    partial void DoSomething();
}
0 references
partial class MyClass
{
    0 references
    partial void DoSomething();
}

0 references
partial struct MyStruct
{
    0 references
    partial void DoSomething();
}
```

Structure (struct) in C#

In C#, we can create a structure or struct using the keyword struct. As explained earlier struct is a value type and it is directly stored in the stack of the memory.

Create a struct

```
using static System.Console;
namespace StructExample
{
    class Program
    {
        static void Main(string[] args)
        {
            MyCoordinates obj;
            obj.xCoordinate = 30;
            obj.yCoordinate = 40;
            WriteLine($"{obj.xCoordinate} {obj.yCoordinate}");
        }
    }

    struct MyCoordinates
    {
        public int xCoordinate;
        public int yCoordinate;
    }
}
```



Struct

- ✓ Variable
- ✓ Property
- ✓ Indexer
- ✓ Concrete Method
- ✓ static fields
- ✓ static Method
- ✓ Instantiation
- ✗ Can inherit a Class
- ✗ Can inherit an Struct
- ✗ Can be a base
- ✗ Explicit Parameterless Constructors (non-static)
- ✓ Static Constructor (always Parameterless)
- ✓ Can Implement an Interface

Enum (enum) in C#

Enum is value type in C# and below is the code snippet to explain how to create an enum.

```
enum RainbowColors
    {
        RED=1,
        ORANGE=2,
        YELLOW=3,
        GREEN=4,
        BLUE=5,
        INDIGO=6,
        VIOLET=7
    }
```

Important Questions:

A. We can use foreach loop over the collection which implements IEnumerable interface but enum do not implement any interface so can we use foreach loop with enum?

Answer: Yes, below is the complete code to demonstrate that how we can use foreach loop over enum

```
using System;
namespace EnumExample
{
    class Program
    {
        static void Main(string[] args)
        {
            foreach(var color in Enum.GetValues(typeof(RainbowColors)))
            {
                Console.WriteLine(color);
            }
        }
    }

    enum RainbowColors
    {
        RED=1,
        ORANGE=2,
        YELLOW=3,
        GREEN=4,
        BLUE=5,
        INDIGO=6,
        VIOLET=7
    }
}
```

B. Can I write a partial enum?

Answer: No, if you will try to use partial keyword with enum then you will get the following compiler error
Error: CS0267 “The 'partial' modifier can only appear immediately before 'class', 'struct', 'interface', or 'void'”

Interface

Interface can be create using the keyword ‘interface’ in C#. Interface is a contract and that’s why all the Methods, properties, Indexers, Events which are part of interface must be implemented by the class or struct which implement the interface. An interface can contain only signature of Methods, Properties, Indexers and Events.

Example of an Interface

```
interface IInterface
   {
       int MyProperty { get; set; }
       void Display();
   }
```

Partial Interface

```
partial interface I1
    {
       void Method1();
        void Method2();

    }
```

A. Can an interface implement another interface?

```
Answer: Yes, an Interface can implement another interface.

Example:
```

```
interface I1
   {
       void Method1();
       void Method2();
   }

   interface I2:I1
   {
       void Method3();
       void Method4();
   }
```

B. Can an interface implement multiple interface?

Answer: Yes, an Interface can implement multiple interface.
Example:

```
interface I1
    {
        void Method1();
        void Method2();
    }

    interface I2:I1
    {
        void Method3();
        void Method4();
    }

    interface I3 : I1, I2
    {
        void Method5();
        void Method6();
    }
```

C. Can an interface have events?

Answer: Yes, an Interface can have methods, properties, indexers & events.

D. Can I use public keyword with interface members?

Answer: No, all the members of the interface are public by-default but we can cannot use any access modifiers explicitly with interface members.

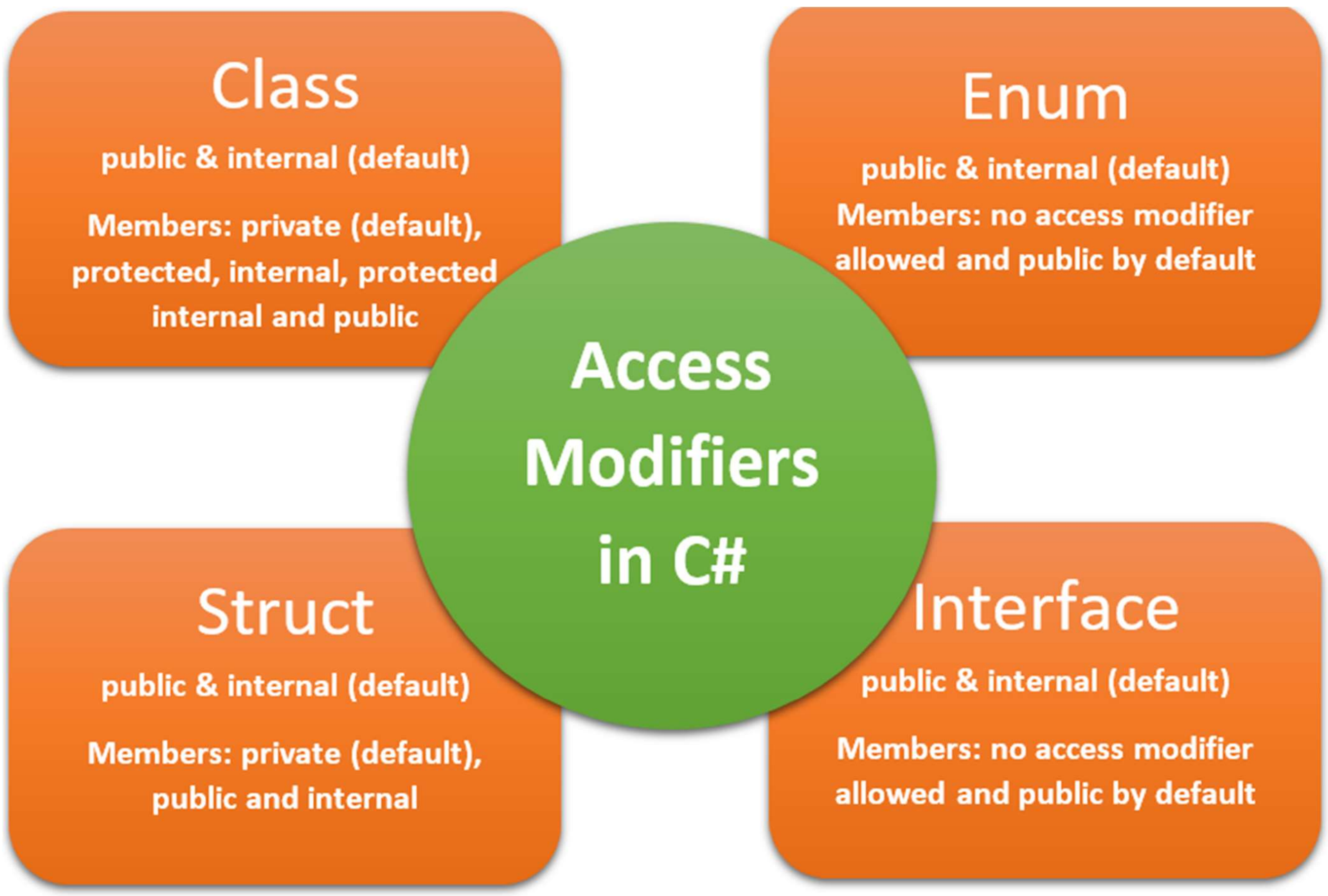
E. Can an interface have constructors?

Answer: No, Interfaces cannot contain constructors.

F. Can I use static keyword with the interface method?

Answer: No, we cannot use static and public modifiers with the interface method.

Access Modifiers in C#


Class
public & internal (default)
Members: private (default), protected, internal, protected internal and public
Enum
public & internal (default)
Members: no access modifier allowed and public by default
Access Modifiers in C#
Struct
public & internal (default)
Members: private (default), public and internal
Interface
public & internal (default)
Members: no access modifier allowed and public by default

What are the access Modifiers allowed with Class?

Class: public & internal (default) only.
Class Members: We can use all the 5 access modifier i.e. private(default), protected, internal, protected internal and public.

```
public class User { }
Or
class User1 { }

Or

internal class User2 { }
```

Class Members with all the 5 access modifiers

```
public class User
    {
        //private access modifiers
        private int UserId { get; set; } = 1;

        //protected access modifiers
        protected string UserName { get; set; } = "Banketeshvar Narayan";

        ////internal access modifiers
        internal string EmailId { get; set; } = "bnarayan.sharma@outlook.com";

        //protected internal access modifiers
        protected internal string City { get; set; } = "Delhi";

        //public access modifiers
        public string Country { get; set; } = "India";
    }
```

Note: The elements defined in a namespace cannot be explicitly declared as private, protected, or protected internal.

What are the access modifiers allowed with enum?

Enum: default access modifier is internal and we can use public & internal explicitly.

Enum Fields: No access modifier is allowed with enum fields and default member accessibility is public.

```
enum IndianTricolor
    {
        Saffron=1,
        White=2,
        Green=3
    }

Or

internal enum IndianTricolor
    {
        Saffron=1,
        White=2,
        Green=3
    }

Or

public enum IndianTricolor
    {
        Saffron=1,
        White=2,
        Green=3
    }
```

What are the access modifiers allowed with interface?

Interface: public & internal (default) only.

```
interface IUser1 { }

Or

internal interface IUser1 { }

Or

public interface IUser1 { }
```

Interface Members: No access modifier is allowed with interface members and default member accessibility is public.

What are the access modifiers allowed with struct?

Struct: public & internal (default) only.

Struct Members: Private, public and internal access modifiers are allowed with struct members and default member accessibility is private.

```
public/internal(default access modifier) struct MyStruct
    {
        private(default access modifier)/internal/protected int MyProperty { get; set; }
    }
```

Tips:

Generally, what happens that most of the developers start remembering thing rather than understanding actual structure and behaviour and it is not good for developers.

If you look at the access modifier for class, then you will notice that the default access modifier for class is internal whereas default access modifier for property is private so I am going to give you 4 tips which help you to understand allowed access modifiers and their behaviour.

1. The elements defined in a namespace cannot be explicitly declared as private, protected, or protected internal. So, we can only use internal and public access modifiers with class, interface, enum & struct.
2. Class members can use all the 5 access modifiers where struct members cannot use access modifiers “protected” and “protected internal” because struct cannot be inherited.
3. The default access modifier is "the most restricted access modifier you could declare for that member". So, default access for class, interface, enum & struct is “internal” for enum & interface member default is public and for class and struct member it’s private.
4. If members of a type can have only public accessibility, then we cannot use any access modifiers explicitly and it will have always public accessibility implicitly. So, interface and enum members cannot have access modifiers explicitly.

What is the use of ‘typeof’ keyword in C#?

It is used to get the types and using this we can do a lot of things of reflection without using complex logic few examples are:

```
Type t = typeof(User);
   WriteLine(t.Assembly);
   WriteLine(t.AssemblyQualifiedName);
   WriteLine(t.IsAbstract);
   WriteLine(t.IsSealed);
   WriteLine(t.IsArray);
   WriteLine(t.IsClass);
   WriteLine(t.IsVisible);
   WriteLine(t.IsValueType);
   WriteLine(t.IsInterface);
```

How to Implement and call same methods of interfaces and base class?

Condition 1: I have 2 interfaces and both is having same method and I would like to implement it the same class how it will be implemented

i.e. I have below Code

```
interface IInterface1
    {
        void Display();
    }

interface IInterface2
    {
        void Display();
    }
```

Now I am going to implement it in the same class. Code snippet is given below.

```
class A : IInterface1, IInterface2
    {
        public void Display()
        {
            WriteLine("Display method of class A");
        }
    }
```

Then calling it using the below code

```
 A objA = new A();
 IInterface1 objB = new A();
 IInterface2 objC = new A();
 objA.Display();
 objB.Display();
 objC.Display();
```

Output

```
Display method of class A

Display method of class A

Display method of class A
```

In the above code you can see that I have implemented only 1 Display() method. So if we are going to use more than one interfaces and they are having the same method name then it is not necessary to implement all those methods explicitly. But in this case all the instances will point to the same method as you can see in the above example.

Condition 2: Now I am going to Add 1 parameter to differentiate Interface methods

```
    interface IInterface1
    {
        void Display();
    }

    interface IInterface2
    {
        void Display(int x);
    }

    class A : IInterface1, IInterface2
    {
        public void Display()
        {
            WriteLine("Display method of class A");
        }

        public void Display(int x)
        {
            WriteLine($"Display method {x:0000}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            A objA = new A();
            IInterface1 objB = new A();
            IInterface2 objC = new A();
            objA.Display();
            objB.Display();
            objC.Display(50);

        }
    }
```

Output

```
Display method of class A

Display method of class A

Display method 0050
```

So, you can see that still there is no issue everything is working fine.

Condition 3: Now let's say Display() method in both of the interface is having 0 parameters and I want to implement is separately then I will implement it like below code snippet:

```
class A : IInterface1, IInterface2
    {
       void IInterface1.Display()
        {
            WriteLine("Display method of IInterface1");
        }

        void IInterface2.Display()
        {
            WriteLine("Display method of IInterface2");
        }

    }
```

If you look at the above code, you will notice 2 things

1. I have used Interface name with method name to implement it explicitly. This is known as "Implement Interface explicitly".

2. I have removed public access modifier because it will not allow any access modifiers. Do you know why it will not allow access modifier because now it is explicit implementation and behaving as interface member.

Condition 4: What is difference between these 2 codes snippets

Code Snippet1:

```
interface IInterface1
    {
        void Display();
    }

class A : IInterface1
    {
        void IInterface1.Display()
        {
            WriteLine("Display method of IInterface1");
        }
    }
```

Code Snippet2:

```
interface IInterface1
    {
        void Display();
    }

class A : IInterface1
    {

      public void Display()
        {
            WriteLine("Display method of IInterface1");
        }
    }
```

You can see that in “Code Snippet 1” I have implemented the interface method explicitly where as in “Code Snippet 2” interface method has been implemented implicitly.

You can call the implicitly implemented method of interface by creating an object of class which implements it without the interface reference.

i.e. Below code snippet will be compiled and will give the same output while being called by objA or objB.

```
interface IInterface1
    {
        void Display();
    }

class A : IInterface1
    {
        public void Display()
        {
            WriteLine("Display method of IInterface1");
        }
    }

A objA = new A();
IInterface1 objB = new A();
objA.Display();
objB.Display();
```

But the below code Snippet will not be compiled

```
interface IInterface1
    {
        void Display();
    }

class A : IInterface1
    {
        void IInterface1.Display()
        {
            WriteLine("Display method of IInterface1");
        }
    }

A objA = new A();
IInterface1 objB = new A();
objA.Display();
objB.Display();
```

Error: 'A' does not contain a definition for 'Display' and no extension method 'Display' accepting a first argument of type 'A' could be found (are you missing a using directive or an assembly reference?)

You may be thinking that why it is giving the compile time error. The reason for compile time error is the Display() method is having implementation inside class A but it is explicitly mentioned that it is only member of "IInterface1".

Now I am going put one more condition in front of you and let's try to understand the below code snippet.

Condition 5: A class is having a base class and also implementing 2 other interfaces and the Types having the same method name.

```
interface IInterface1
    {
        void Display();
    }

    interface IInterface2
    {
        void Display();
    }

    class B
    {
       public void Display()
        {
            WriteLine("Display method of class B");
        }
    }


A objA = new A();
IInterface1 objB = new A();
IInterface2 objC = new A();
objA.Display();
objB.Display();
objC.Display();


output:
```

Display method of class B

Display method of IInterface1

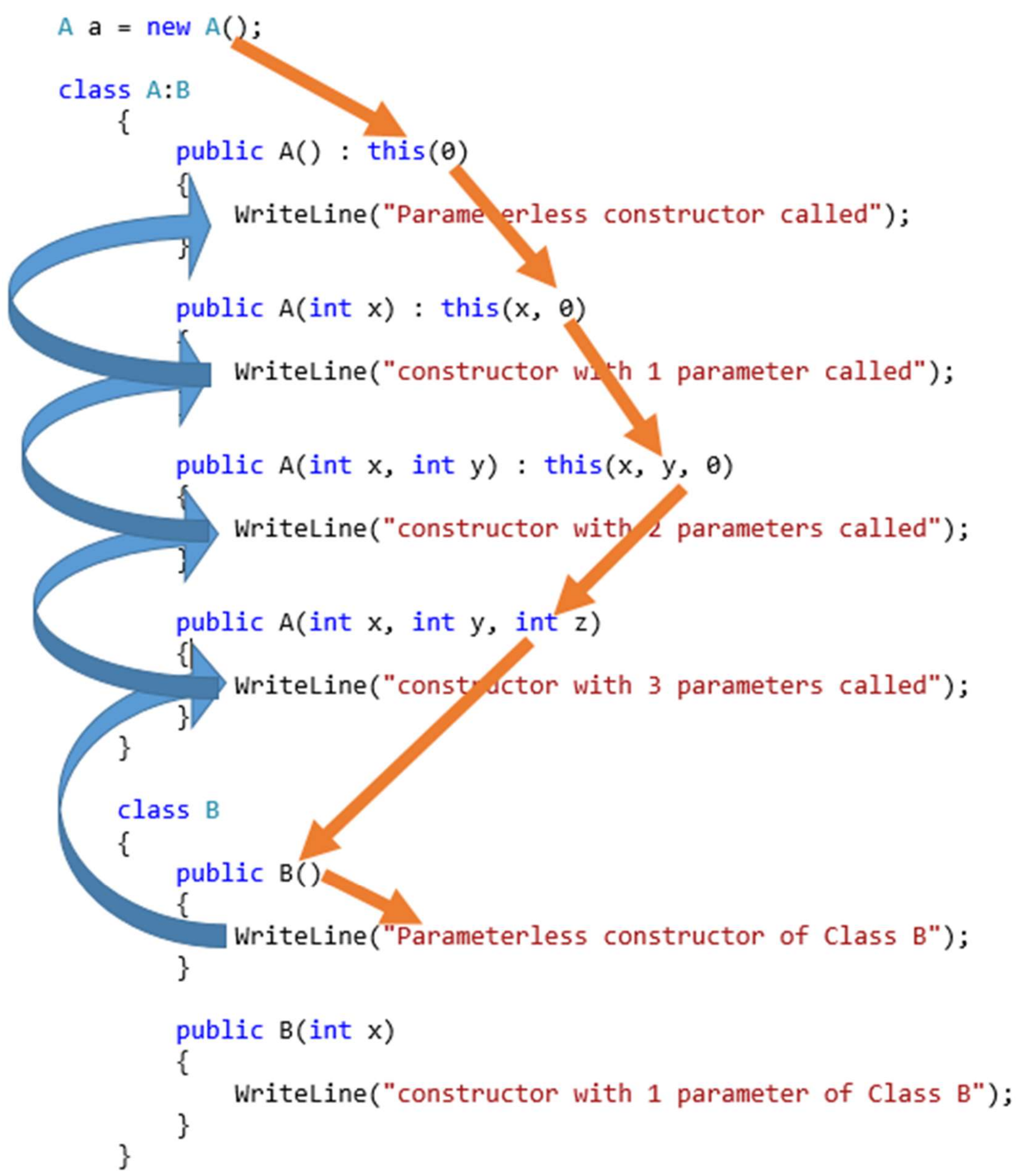
Display method of IInterface2

Condition 6:

```
class A :B, IInterface1, IInterface2
    {
       public void Display()
        {
            WriteLine("Display method written inside A");
        }
    }

A objA = new A();
objA.Display();

it will call method of Class A and will not call its parent class method.


If you want to call parent class method then you have to give reference of parent class method.
i.e.
B objD = new B(); objD.Display();
```

So After going through all these 6 different conditions I hope that if interviewer is trying to complicate the situation of implicit and explicit method implementation to any level you will not be confused and you will be able to answer it confidently.

How to call multiple constructor of a class with single object creation?



```
A a = new A();

class A:B
    {
        public A() : this(0)
        {
            WriteLine("Parameterless constructor called");
        }

        public A(int x) : this(x, 0)
        {
            WriteLine("constructor with 1 parameter called");

        public A(int x, int y) : this(x, y, 0)
        {
            WriteLine("constructor with 2 parameters called");
        }

        public A(int x, int y, int z)
        {
            WriteLine("constructor with 3 parameters called");
        }
    }

    class B
    {
        public B()
        {
            WriteLine("Parameterless constructor of Class B");
        }

        public B(int x)
        {
            WriteLine("constructor with 1 parameter of Class B");
        }
    }
```

In the below example I have a class "A" which is having 4 instance constructors. 1st constructor is Parameterless constructor, 2nd constructor having one parameter, 3rd constructor having two parameters and 4th constructor having three parameters.

```
class A
    {
        public A()
        {
            WriteLine("Parameterless constructor called");
        }

        public A(int x)
        {
            WriteLine("constructor with 1 parameter called");
        }

        public A(int x, int y)
        {
            WriteLine("constructor with 2 parameters called");
        }

        public A(int x, int y, int z)
        {
            WriteLine("constructor with 3 parameters called");
        }
    }
```

Now I would like to call all those 4 constructors by just creating only one object. Let's see how it can be done. So, the new code will be

```
class A
    {
        public A() : this(0)
        {
            WriteLine("Parameterless constructor called");
        }

        public A(int x) : this(x, 0)
        {
            WriteLine("constructor with 1 parameter called");
        }

        public A(int x, int y) : this(x, y, 0)
        {
            WriteLine("constructor with 2 parameters called");
        }

        public A(int x, int y, int z)
        {
            WriteLine("constructor with 3 parameters called");
        }
    }
A a = new A(); //or // A a = new A { };
```

And output will be

```
constructor with 3 parameters called

constructor with 2 parameters called

constructor with 1 parameter called

Parameterless constructor called
```

I am going to ask you few questions of C# code snippet output which is very popular and can be asked to anyone. So, let's continue with below code snippets.

What will be the output of below code Snippet?

```
using static System.Console;
namespace Example1
{
    class A
    {
        public void Display()
        {
            WriteLine($"Class A");
        }
    }
    class B : A
    {
        public void Display()
        {
            WriteLine($"Class B");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            A obj1 = new A();
            A obj2 = new B();
            B obj3 = new B();

            obj1.Display();
            obj2.Display();
            obj3.Display();
        }
    }
}
```

Output:

Class A
Class A
Class B

What will be the output of below code snippet If create different types of objects in different ways?

```
class A
    {
        public void Display()
        {
            WriteLine($"Class A");
        }
    }

    class B:A
    {
        public void Display()
        {
            WriteLine($"Class B");
        }
    }

    class C : A
    {
        public static void Display()
        {
            WriteLine($"Class C");
        }
    }
    class D : A
    {
        public virtual void Display()
        {
            WriteLine($"Class D");
        }
    }

    class E : A
    {
        public virtual new void Display()
        {
            WriteLine($"Class E");
        }
    }
    abstract class F : A
    {
        public abstract void Display();
    }

    abstract class G : A
    {
        public abstract new void  Display();
    }

    abstract class H : A
    {
        public abstract void Display(int x);
    }
```

I am not going to write output of the above code snippet here rather than I am leaving this as an exercise for the developers whose oops is not strong so they can execute all the above scenario with different combinations and can understand basics of oops. Still anybody think that he need output of the below code snippet then I will put it on comment section of this article later on. I am also going to give some tips specifically compiler error messages for the above code snippet so that you can understand it in better way.

Tips (compiler error messages):

1. A static member cannot be marked as override, virtual, or abstract.
2. Cannot declare instance members in a static class.
3. An object reference is required for the non-static field, method, or property.
4. An abstract class cannot be sealed or static. A class cannot be both static and sealed.
5. The 'partial' modifier can only appear immediately before 'class', 'struct', 'interface', or 'void'.
6. Elements defined in a namespace cannot be explicitly declared as private, protected, or protected internal.

What is the use of extern keyword in C#?

We use extern keyword with a method to indicate that the method has been implemented externally i.e. out of you current C# code. Below is a code sample from MSDN which describes the use of extern

```
[DllImport("User32.dll")]
```

```
public static extern int MessageBox(int h, string m, string c, int type);
```

note: while using [DllImport("User32.dll")] you need to include the namespace System.Runtime.InteropServices;

for complete details about extern visit here.

Sequence of Modifiers and other attribute with class and method:

In C# sometimes sequence of modifiers matters and you should be aware about that if you are going for an interview.

e.g. `public abstract partial class Student or public partial abstract  class Student`

you would be thinking that why I am explaining microlevel questions. I am explaining it because in some cases you have to face such questions and you will fill little bit irritating or upset if you are not aware about all those things.

Sometimes sequence matter and sometimes it doesn't matter but we should keep some best practices in mind so that we write error free code.

Class

[modifiers] [partial] className [:] [inheritance list]

Method

[modifiers] [partial]   returnType methodName([parameters])

Method Modifiers List:

new

internal

private

protected

public

abstract

async

extern

override

sealed

static

virtual

there are some more modifiers which I have not mentioned here because they are used rarely.

Usage of ref keyword in C#

When we pass a value type variable as parameter then it passes its value as parameter i.e. making a copy of the data. Making any changes inside the method in which it has been passed as parameter will not affect the variable accessed outside the method. Because it is passed as value and whatever changes will be made inside the local method will be done on a copy of that variable.

If we need to change the value type variable from inside the method in which it has been passed as parameter, then you can pass this value type variable as reference using the 'ref' keyword in that case reference of value type will be passed as parameter rather than value of the variable.

Let's Consider about reference type

A variable of refence type contains reference of its data and does not contain data directly. So, whenever we pass a reference type variable as parameter then data is not passed but the reference of the variable is passed inside the method i.e. passes the value of memory address not the value of data and that's why if I make any change to this variable inside the method then it will be available outside the method but it is only limited for the modification and if I am going to allocate a new memory and modifying the variable then these changes will not be reflected outside the method.

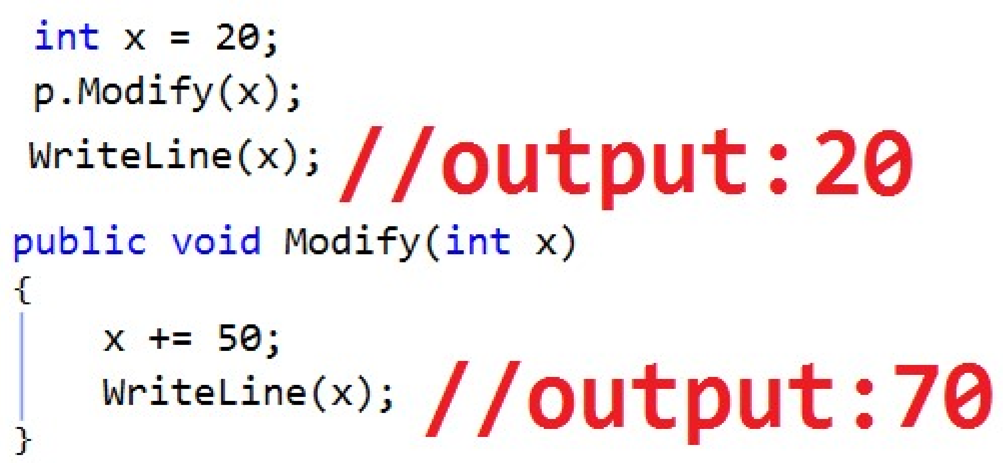
If I need that the changes done inside the method for the variable by allocating new memory should be reflected outside the method in that case, we need pass the parameter of reference type with 'ref' keyword. If I pass a reference type variable parameter using 'ref' keyword, then the reference of memory address is itself passed as reference and so allocating a new memory will also be available outside the method.

Below is the pictorial representation of the same thing stated above.

So, there are 4 combinations of passing parameter inside a method. There are a lot of other options to pass parameter as value but I am not going to discuss it here otherwise it will create confusion. I will discuss about other parameter passing options in a separate section.

1. Passing Value type variable **without 'ref'** keyword
2. Passing Value type variable **with 'ref'** keyword
3. Passing reference type variable **without 'ref'** keyword
4. Passing reference type variable **with 'ref'** keyword

1. Passing Value type variable without ‘ref’ keyword



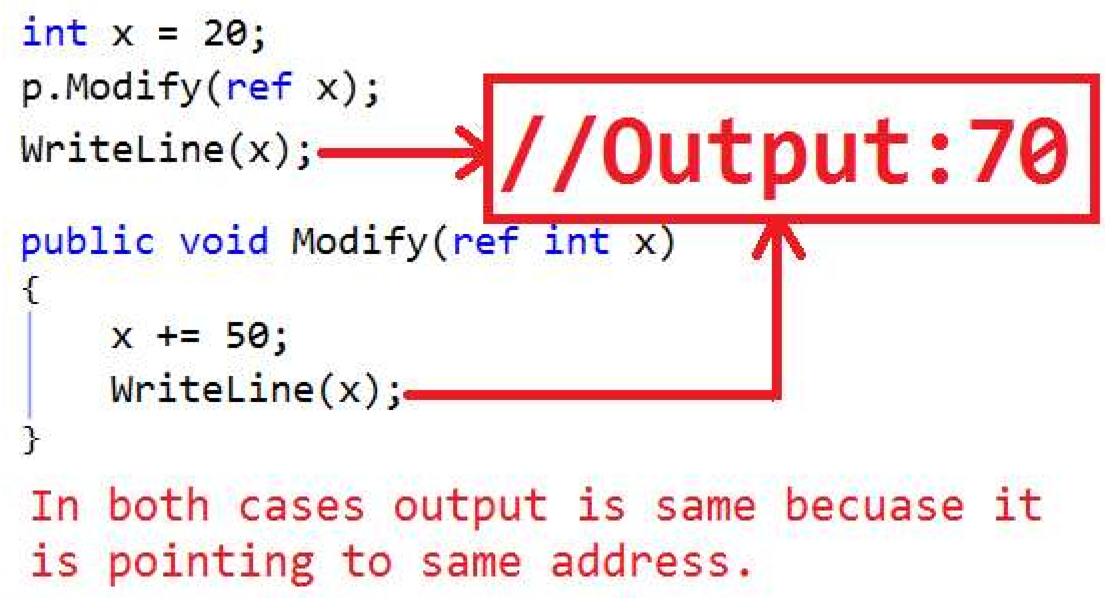
```
int x = 20;
p.Modify(x);
WriteLine(x); //output:20
public void Modify(int x)
{
    x += 50;
    WriteLine(x); //output:70
}
```

In the above case the variable x having initial value 20 and its value i.e. 20 is passed as parameter for the method Modify. Inside the method modify it has been modified and added 50 to its previous value so it become 70 but value of the variable ‘x’ outside the modify method is still 20 because it has modified its value not the reference.

Below is the complete code

```
using static System.Console;
namespace PassingValueTypeWithoutRef
{
    class Program
    {
        static void Main(string[] args)
        {
            Program p = new Program();
            int x = 20;
            p.Modify(x);
            WriteLine(x);
        }
        public void Modify(int x)
        {
            x += 50;
            WriteLine(x);
        }
    }
}
```

2. Passing Value type variable with ‘ref’ keyword



```
int x = 20;
p.Modify(ref x);
WriteLine(x);

public void Modify(ref int x)
{
    x += 50;
    WriteLine(x);
}
```

//Output:70
In both cases output is same becuase it is pointing to same address.

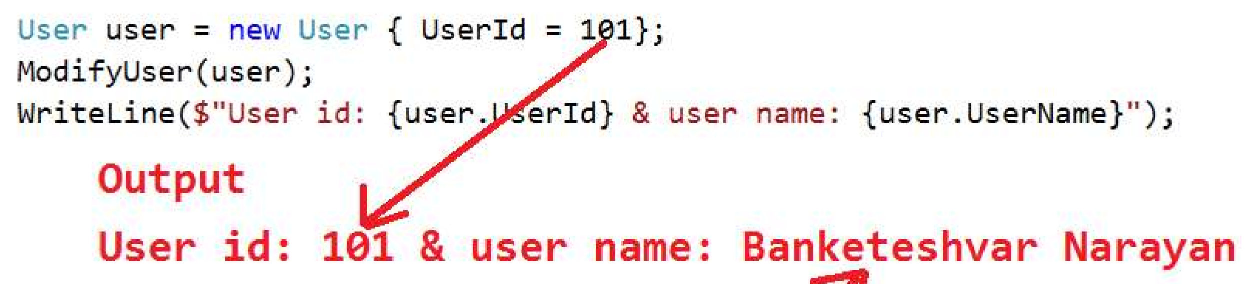
In the above case the variable x having initial value 20 and its reference passed as parameter for the method Modify. Inside the method modify it has been modified and added 50 to its previous value so it become 70. As this variable, has been passed as reference i.e. value of its address has been passed instead of data so the value of the variable 'x' outside the Modify method is also updated and you will get 70 everywhere.

Below is the complete code

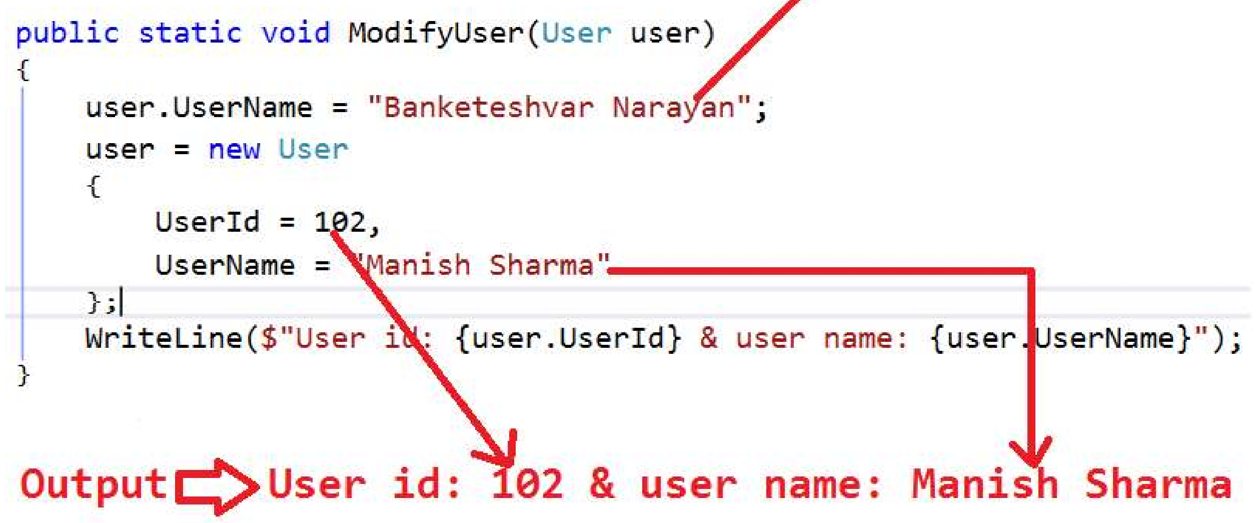
```
using static System.Console;
namespace PassingValueTypeWithRef
{
    class Program
    {
        static void Main(string[] args)
        {
            Program p = new Program();
            int x = 20;
            p.Modify(x);
            WriteLine(x);
        }
        public void Modify(int x)
        {
            x += 50;
            WriteLine(x);
        }
    }
}
```

3. Passing reference type variable without 'ref' keyword

In C#, whatever we passed inside the method as parameter is always passed as value. So, what happens that when we pass value type then copy of a value typed is passed where as in case of reference type variable its reference address is passes as method parameter.


```
User user = new User { UserId = 101};
ModifyUser(user);
WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
```

```
Output
User id: 101 & user name: Banketeshvar Narayan
```


```
public static void ModifyUser(User user)
{
    user.UserName = "Banketeshvar Narayan";
    user = new User
    {
        UserId = 102,
        UserName = "Manish Sharma"
    };
    WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
}
```

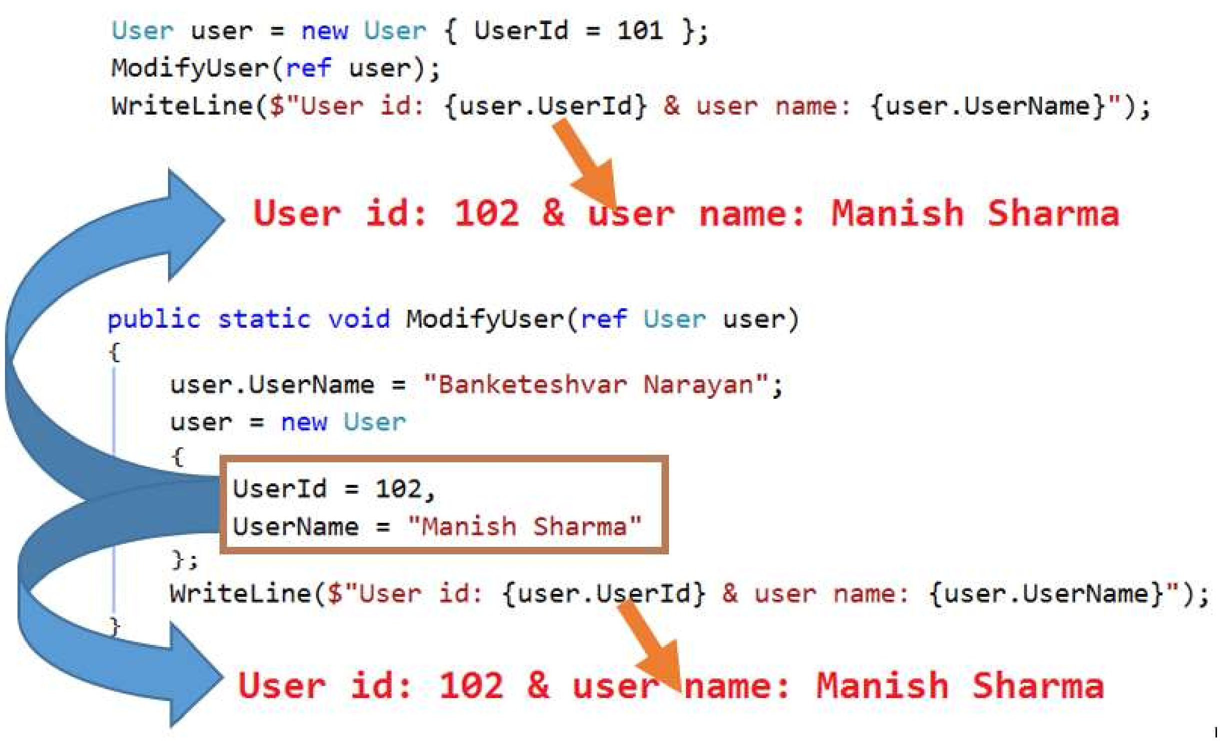
```
Output ⇨ User id: 102 & user name: Manish Sharma
```

In the above example, you can see that when I am passing a reference type as parameter then its address is being passed as data not the actual data. That's why if I change user object from the ModifyUser() method then it is being reflected outside but If I allocate a new memory and then make any change it will not be reflected outside the method.

Complete Code

```
using static System.Console;
namespace PassingReferenceTypeWithoutRef
{
    class Program
    {
        static void Main(string[] args)
        {
            User user = new User { UserId = 101};
            ModifyUser(user);
            WriteLine($"User id: {user.UserId} & user name: {user.UserName}");

        }

        public static void ModifyUser(User user)
        {
            user.UserName = "Banketeshvar Narayan";
            user = new User
            {
                UserId = 102,
                UserName = "Manish Sharma"
            };
            WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
        }
    }
    class User
    {
        public int UserId { get; set; }
        public string UserName { get; set; }
    }
}
```

4. Passing reference type variable with ‘ref’ keyword



```
User user = new User { UserId = 101 };
ModifyUser(ref user);
WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
```

```
User id: 102 & user name: Manish Sharma
```

```
public static void ModifyUser(ref User user)
{
    user.UserName = "Banketeshvar Narayan";
    user = new User
    {
        UserId = 102,
        UserName = "Manish Sharma"
    };
    WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
}
```

```
User id: 102 & user name: Manish Sharma
```

In the above example, you can see that when I am passing a reference type as parameter then its address is being passed as data not the actual data. That’s why if I change user object from the ModifyUser() method then it is being reflected outside even If I allocate a new memory and then make any change it will be reflected outside the method.

Complete Code

```
using static System.Console;
namespace PassingReferenceTypeWithRef
{
    class Program
    {
        static void Main(string[] args)
        {
            User user = new User { UserId = 101 };
            ModifyUser(ref user);
            WriteLine($"User id: {user.UserId} & user name: {user.UserName}");

        }

        public static void ModifyUser(ref User user)
        {
            user.UserName = "Banketeshvar Narayan";
            user = new User
            {
                UserId = 102,
                UserName = "Manish Sharma"
            };
            WriteLine($"User id: {user.UserId} & user name: {user.UserName}");
        }
    }
    class User
    {
        public int UserId { get; set; }
        public string UserName { get; set; }
    }
}
```

5. Passing string variable without 'ref' keyword

String is reference type but sometimes it behaves like value type. Even though in the current scenario if I pass string inside a method and modify it then it will not be affected to the outside variable from where method has been called. For the below code snippet output will not be same.

```
class Program
    {
        static void Main(string[] args)
        {
            string name = "B Narayan";
            ModifyName(name);
            WriteLine($"name is : {name}");
            //output: name is : B Narayan
        }

        public static void ModifyName(string name)
        {
            name = "Manish Sharma";
            WriteLine($"name is : {name}");
            //output: name is : Manish Sharma
        }

    }
```

As you can see the string modified inside the method is not affecting the outside variable so if we need to modify outer variable from `ModifyName() method then we have to pass the string parameter with ref keyword.`

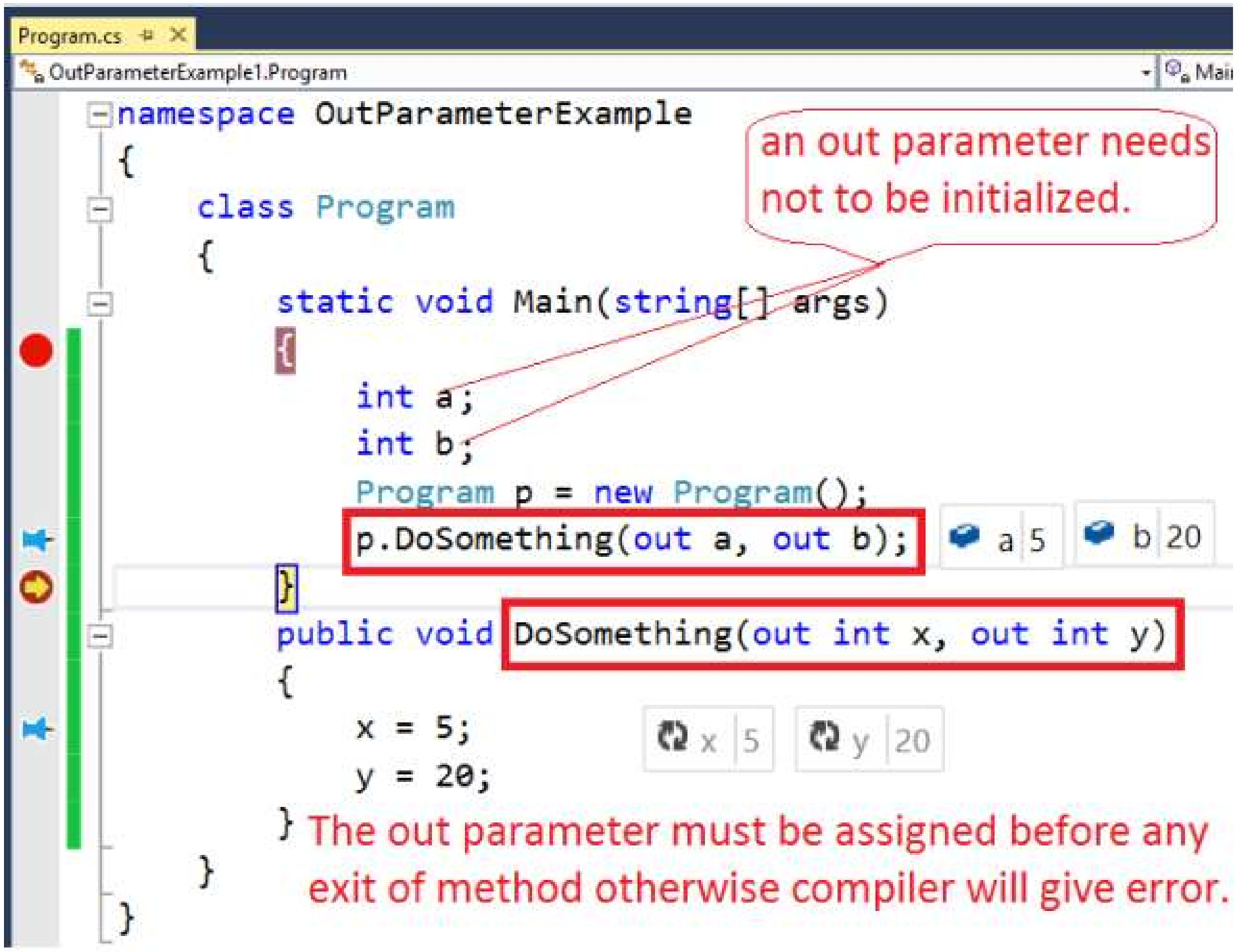
6. Passing string variable with 'ref' keyword

```
class Program
    {
        static void Main(string[] args)
        {
            string name = "B Narayan";
            ModifyName(ref name);
            WriteLine($"name is : {name}");
            //output: name is : Manish Sharma
        }
        public static void ModifyName(ref string name)
        {
            name = "Manish Sharma";
            WriteLine($"name is : {name}");
            //output: name is : Manish Sharma
        }

    }
```

Now you can see that I have passed the string variable name with ref keyword so it has been modified on both places.

Usage of ‘out’ keyword in C#

We can use ‘out’ keyword with parameter and it behaves very similar to the parameter passed with ‘ref’ keyword. But any argument passed as out parameter need not to be initialized whereas in case of ref it must be initialized. Another difference is that ref parameter may be assigned inside the method in which it is called (not necessary), but in case of “out” it must be assigned in the calling method before any exit.


Program.cs
OutParameterExample1.Program
Mai
an out parameter needs not to be initialized.
a 5
b 20
x 5
y 20
The out parameter must be assigned before any exit of method otherwise compiler will give error.

```
namespace OutParameterExample
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            int b;
            Program p = new Program();
            p.DoSomething(out a, out b);
        }
        public void DoSomething(out int x, out int y)
        {
            x = 5;
            y = 20;
        }
    }
}
```

Using ‘params’ in method parameter

The ‘params’ keyword can be used to specify a method parameter that takes a variable number of arguments. When we use ‘params’ keyword to pass variable number of arguments we can send any number of arguments or no arguments. But all the arguments passed should be of same type.

A method can have only parameter with ‘params’ keyword or with additional parameters and ‘params’ as last parameters but no other parameters can be passed after ‘params’. Let’s try to understand with some examples.

Example 1:

```
static void Main(string[] args)
        {
            string my1stLine = "This is my first Line";
            string my2ndLine = "This is my second Line";
            string my3rdline = "This is my Third Line";
            string my4thline = "This is my fourth Line";
            Print(my1stLine, my2ndLine, my3rdline, my4thline);
        }
        public static void Print(params string[] listToPrint)
        {
            foreach (var item in listToPrint)
            {
                WriteLine(item);
            }
        }
```

Example 2:

```
static void Main(string[] args)
        {
            int userId = 1001;
            string email1 = "abc@gmail.com";
            string email2 = "abc@yahoo.com";
            string email3 = "abc@hotmail.com";
            string email4 = "abc@outlook.com";
            UserDetails(userId, email1, email2, email3, email4);
        }

        public static void UserDetails(int UseId, params string[] emailIds)
        {
            foreach (var email in emailIds)
            {
                WriteLine($"userId is {UseId} and emailId is {email}");
            }
        }
```

Example 3:

```
public static void UserDetails(params string[] emailIds, int UseId)
        {
            foreach (var email in emailIds)
            {
                WriteLine($"userId is {UseId} and emailId is {email}");
            }
        }
```

Out of the above 3 examples Example1 & Example 2 will compile & run successfully but the third example will not be compiled. It will give following errors:

Error: A params parameter must be the last parameter in a formal parameter list

So far, I have discussed a lot about method and I would like to mention one last thing that we can also pass dynamic parameters in method.

```
class Program
        {
            static void MyDynamicMethod(dynamic dynamicparam)
            {
                dynamicparam.DoSomethingDynamic(); // Called dynamically
            }
            static void Main(string[] args)
            {
                MyDynamicMethod("dynamicValue");
            }
        }
```

This article has become very long already so I am not going to tell more about dynamic types but hopefully in next article of this series I will discuss on it in more depth.

For C# basics concept, you can refer some of the article written by me. Below is the list for the same.

Different Types of Method Parameters in C#

Break Vs Continue in C#

Understanding Delegates in C#

Understanding Delegates in C# - Part 2

Parallel.ForEach() Vs foreach() Loop in C#