**Ref & Out**

* Ref and out parameters in C# allows us to pass the parameters by reference instead of Value.
* We can change this default behavior. If we want to change the original values inside our methods, we can do that by using ref and out keywords inside the method signature and inside the method call as well.
* We can use the ref keyword only if the variable which we use as an argument is initialized before calling a method.
* By using the out keyword, we don’t have to initialize a variable before calling a method but, we must initialize it inside a method.
* So, let’s simplify that. If we want to change an existing value of a variable inside a method, we are going to use the ref keyword.
* But, if we want to assign a completely new value to the variable inside a method, then we use the out keyword

class Program

{

    public static void ChangeRef(ref int numberRef)

    {

        numberRef = 25;

        Console.WriteLine($"Inside the ChangeRef method the numberRef is {numberRef}");

    }

    public static void ChangeOut( out int numberOut)

    {

        numberOut = 60;

        Console.WriteLine($"Inside the ChangeOut method the numberOut is {numberOut}");

    }

    static void Main(string[] args)

    {

        int numberRef = 15;

        Console.WriteLine($"Before calling the ChangeRef method the numberRef is {numberRef}");

        ChangeRef(ref numberRef);

        Console.WriteLine($"After calling the ChangeRef method the numberRef is {numberRef}");

        Console.WriteLine();

        int numberOut;

        Console.WriteLine("Before calling the ChangeOut method the numberOut is unassigned");

        ChangeOut(out numberOut);

        Console.WriteLine($"After calling the ChangeOut method the numberOut is {numberOut}");

        Console.ReadKey();

**Readony & constant keyword**

|  |  |
| --- | --- |
| **READONLY KEYWORD** | **CONST KEYWORD** |
| In C#, readonly fields can be created using readonly keyword | In C#, constant fields are created using const keyword. |
| ReadOnly is a runtime constant. | Const is a compile time constant. |
| The value of readonly field can be changed. | The value of the const field can not be changed. |
| It cannot be declared inside the method. | It can be declared inside the method. |
| In readonly fields, we can assign values in declaration and in the constructor part. | In const fields, we can only assign values in declaration part. |
| It can be used with static modifiers. | It cannot be used with static modifiers. |

**Extension method**

* Extension method concept allows you to add new methods in the existing class or in the structure without modifying the source code of the original type and you do not require any kind of special permission from the original type
* Suppose you have a class or a structure which contains three methods and you want to add two new methods in this class or structure, you did not have the source code of the class/structure, or do not have permissions from the class/structure, or the class is a sealed class, but you still want to add new methods in it, then you can use the concept extension method to add the new method in the existing class/structure.
* If the class is sealed than there in no concept of extending its functionality. For this a new concept is introduced, in other words extension methods

using System;

using System.Text;

namespace ClassLibExtMethod

{

    public class Class1

    {

        public string Display()

        {

            return ("I m in Display");

        }

        public string Print()

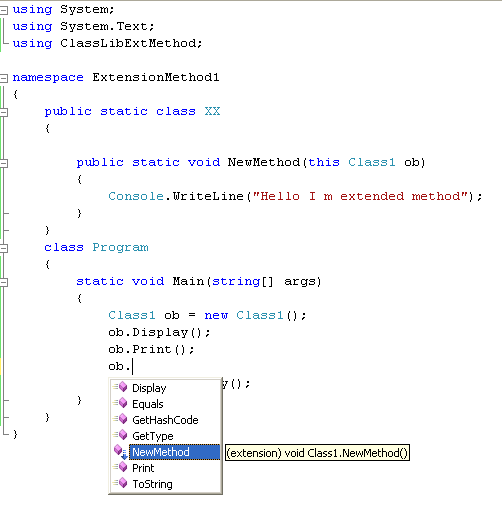
        {

            return ("I m in Print");

        }

    }

}



**POCO Class in EF**

POCOs(Plain old CLR objects) are simply entities of your Domain. Normally when we use entity framework the entities are generated automatically for you. This is great but unfortunately, these entities are interspersed with database access functionality, which is clearly against the SOC (Separation of concern). POCOs are simple entities without any data access functionality but still gives the capabilities all [EntityObject](http://msdn.microsoft.com/en-us/library/system.data.objects.dataclasses.entityobject.aspx) functionalities like

* Lazy loading
* Change tracking

Does not inherit from any class

**Include in EF**

* Eager loading is the process whereby a query for one type of entity also loads related entities as part of the query, so that we don't need to execute a separate query for related entities. Eager loading is achieved using the **Include()** method.
* In the following example, it gets all the students from the database along with its standards using the **Include()** method

**Clone in C#**

* In C#, Clone() is a String method. It is used to clone the string object, which returns another copy of that data
* The return value will be only another view of the same data.

|  |
| --- |
| class Geeks {          public static void Main(string[] args)      {          string s1 = "GeeksForgeeks";            // Cannot implicitly convert          // type object to the string.          // So explicit conversion          // using Clone() method          string s2 = (String)s1.Clone();            // Displaying both the string          Console.WriteLine("String : {0}", s1);          Console.WriteLine("Clone String : {0}", s2);      }  } |

Output:

String : GeeksForgeeks

Clone String : GeeksForgeeks