C# 7.0 /C# 8.0 New Features

Monday, February 3, 2025 8:55 AM

1. Local variables
2. Tuples and deconstruction
3. Async methods
4. Pattern Matching
5. Binary literals
6. Lambda Expression

Extension Methods : It allows us to add new methods into a class without editing the source code of the class . For eg: if a class consists of a set of members in it and in future if we want to add new methods into that class , we can add those methods without making any changes to the source code .

So one of the use case if we have any of the sealed which cannot be inherited by the child class then we can extend the functionality using extension method.

```
Sealed class HRPayment   -- >  sealed can be a property or a class
{

}
Class employee : HRPayment
{

}



Sealed class  Service
{

Public Int x=100
Public void Test1()
{

}
Public void Test2()
{


}

}
```

If we want to add new functionality into the service class if we are not aware of the source code

```
 static class NewService
{
Public void Test3(this Service ser)
{

}
Public void Test4(this Service ser , int x)
{
 Console.Writeline(x)
  Console.Writeline(ser.x)
}





}

Class Program

{

Static void main(string[] ar)
{
Service sobj = new Service();

Sobj.Test1();
Sobj.Test2();
Sobj.Test3();
Sobj.Test4(100);


}



}
```

```
// Role of Out  Parameter
 string s = "03";
 DateTime date;

 if (DateTime.TryParse(s, out date))
 {
     Console.WriteLine(date);

 }

 Console.WriteLine("It's done");
```

//Pattern Matching
The main role of pattern matching is to handle multiple data types without using multiple if else condition

The enhanced pattern matching will be implemented via 2 ways:

1. Pattern Matching using "is" expression
2. Pattern Matching using "case" statements

```
Static void ObjectCheck(Object obj)
{

If(obj is int)
{


}
Else if (obj is string)
{

}


}
```

```
Static void ObjectCheck(Object obj)
{
Switch(obj)
{
Case int number
//statement
Break;
Case string text
//statements



}
```

Indexer :

Indexers in C# allow instances of a class to be indexed just like an array

```
Arr[0] = "hghg";
```

```
Employee(id int , string name , int age , double marks)
{

This.Name = name;


}

Employee e = new Employee(12,"Niti",30 ,90.00)
{
Console.writeline(e[1])


}
```

So , If we want to apply indexing directly to a class we require indexers for that

```
<modifier> <type> this [int index  or string name]
{
Get {} // to retreive
Set{}   // to assign

}
```

Local functions : Lambda Expression

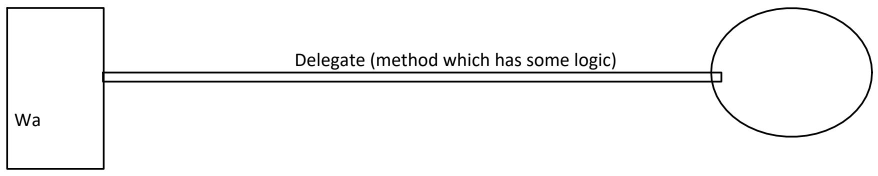
```
Class ABC
{

Void hello()
{


}
```

}

DELEGATE : Delegate is a class

A delegate is a type safe function or method pointer(a Delegate is pointing to a method which is used to invoke a method)
Multiple delegate
Single cast , Multicast

Event -- > Event listener

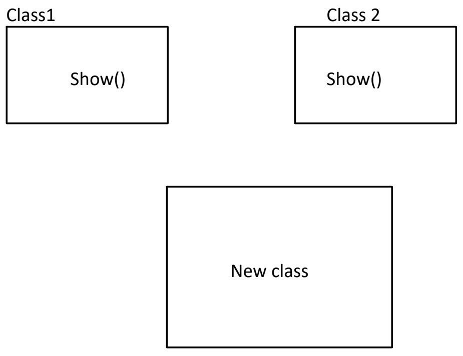


Syntax:

Accessmodifier delegate <datatype> delegate_name(parameter_list)

Public delegate void myshow(int x, int y); // this delegate will point to a method

```
Public static void print()
{


}


Public void print(int a , int b)
{


}c


Public int display()
{


}

Myshow my = new Myshow(print())
cla
```


Class1
Show()
Class 2
Show()
New class