**What new in C Sharp 8.0**

**Notes:-**

**1-C# is language and so the compiler have to updated to support C# 8.0**

**2-Asp.Net Core is framework that support C# language compilation**

**Section01:- Readonly in methods and properities**

**Notes:-**

**1-we can type readonly on methods and properities as below**

**public struct Rectangle{**

**private double \_height;**

**public readonly double length { get; }**

**public double height {**

**readonly get {return \_height;}**

**set {\_height = value;} }**

**//it will initialize one time when the constructor initialized**

**//it will make definsive copy so it will make protection level and**

**//the system make copy instance of properties inside it that refer to properities of the struct**

**public readonly double Area(){**

**//you cannot change as below**

**//height += 1;**

**return length \* height;}}**

**C# default implementation of interface**

**Notes:-**

**1-C# 8.0 offer default implementation of the interface , so for the classes that implement from the interfaces , no need to implement the default implementation if they don’t do this**

**2-the better way is using only one interface with default implmenation methods as below**

**//with the default implementation of interface , you dont need to implement all the method in the interface , only the needed one**

**public interface IShoppingCart{**

**public string fullName { get; set; }**

**private static string defaultName = "Default ";**

**public static void setDefaultName(string name){defaultName = name;}**

**void CalcoulateSubTotal(){Console.WriteLine($"This is the {defaultName} Calcoulate Sub Total");}**

**void CalcoulateTotal();}**

**//we can implement multiple interfaces with defautl implementation with the same method name as below**

**public interface ISubTotal{**

**void CalcoulateSubTotal(){Console.WriteLine($"This is the Sub Total Calcoulate Sub Total");}}**

**//shoopingCart only see the CalcoulateTotal method**

**class ShoppingCart : IShoppingCart{**

**public string fullName { get; set; }**

**public void CalcoulateTotal(){**

**Console.WriteLine("This is the default Calcoulate Total for Shooping Cart");}}**

**//BetterShoppingCart see the CalcoulateSubTotal and CalcoulateTotal methods**

**public class BetterShoppingCart : IShoppingCart , ISubTotal{**

**public string fullName { get; set; }**

**public BetterShoppingCart(string name){IShoppingCart.setDefaultName(name);}**

**public void CalcoulateTotal(){**

**Console.WriteLine("This is the default Calcoulate Total for Better Shopping Cart");}}**

**class Program{**

**static void Main(string[] args){**

**//Shopping Cart only see the method it implement**

**var shoopingCart = new ShoppingCart();**

**shoopingCart.CalcoulateTotal();**

**//in order to access all methods we can make interface variable**

**IShoppingCart shoopingCart2 = new ShoppingCart();**

**shoopingCart2.CalcoulateTotal();**

**shoopingCart2.CalcoulateSubTotal();**

**//in order to access all methods we can make interface variable**

**IShoppingCart betterShoppingCart = new BetterShoppingCart("New One");**

**betterShoppingCart.CalcoulateTotal();**

**betterShoppingCart.CalcoulateSubTotal();**

**//we can impelment multiple interfaces with the default action name and access to the method we want by sepcify the interface name as below**

**ISubTotal subTotal = new BetterShoppingCart("New One");**

**subTotal.CalcoulateSubTotal();**

**Console.ReadLine();**

**Console.WriteLine("Hello World!");}}**

**Switch Expression**

**Notes:-**

**1-with using switch expression, we can simply our switch statement into the following code as below**

**public static double DoMath(double x, double y, MathType mathType){**

**//Swith Expression**

**var output = mathType switch{**

**MathType.Add => x + y,**

**MathType.Subtract => x - y,**

**MathType.Divide => x / y,**

**MathType.Multiply => x \* y,**

**//something else**

**\_ => throw new Exception("Bad Operation")};**

**Switch Statment**

**double output = 0;**

**switch (mathType){**

**case MathType.Add:**

**output = x + y;**

**break;**

**case MathType.Subtract:**

**output = x - y;**

**break;**

**case MathType.Multiply:**

**output = x \* y;**

**break;**

**case MathType.Divide:**

**output = x / y;**

**break;**

**default:**

**throw new Exception("Bad Operation");}**

**return output;}**

**Using Declaration**

**Notes:-**

**1-with using declaration you can simply the code by using command in the begin of each stream reader or stream writer variable as below**

**public static int ConvertFiles(){**

**//Declaration expression**

**int output = 0;**

**using var inputFile = new StreamReader(@"D:\InputFile.txt");**

**using var outputFile = new StreamWriter(@"D:\OutputFile.txt");**

**string line;**

**while ((line = inputFile.ReadLine()) != null){**

**outputFile.WriteLine(line);**

**output += 1;}**

**//normal stream reader / writer**

**int output = 0;**

**using (var inputFile = new StreamReader(@"D:\InputFile.txt")){**

**using (var outputFile = new StreamWriter(@"D:\OutputFile.txt")){**

**string line;**

**while ((line = inputFile.ReadLine()) != null){**

**outputFile.WriteLine(line);**

**output += 1;}}}**

**return output;}**

**Using Indexes Range in C sharp 8**

**Notes:-**

**1-we can using Index and apply index range as below**

**public static void Demo(){**

**var places = new string[] { "First", "Second", "Third", "Fourth", "Fifth" };**

**Console.WriteLine(places[^1]); // it will return the last item of the array**

**Console.WriteLine(places[^2]); // it will return the Fourth item**

**//it will iterate from item index 1 to index n-2 (from second to fourth)**

**foreach (var item in places[1..^1]){Console.Write($"{item} , ");}**

**Console.WriteLine();**

**//it will iterate from item index 0 to index n-2 (from first to fourth)**

**foreach (var item in places[..^1]){Console.Write($"{item} , ");}**

**Console.WriteLine();**

**//it will iterate from index 1 to index n-1 (from second to fifth)**

**foreach (var item in places[1..]){Console.Write($"{item} , ");}**

**Console.WriteLine();}**

**//it will iterate from index 0 to index n-1 (from second to fifth)**

**foreach (var item in places[..]){Console.Write($"{item} , ");}**

**Console.WriteLine();}**

**Null Coalescing Assignments**

**Notes:-**

**1-we using null coalescing assignment will check that if the previous property is null make something with below expression ??=**

**2-we using also null check ?. that indicate if the previous property is null ignore the rest**

**public class NullCoalescingAssignments{**

**public static void Demo(){**

**var example = new ListDemo();**

**//we have null check that we check if example is null ignore the rest**

**example?.LuckyNumbers?.Add(15);**

**//we use the null colescing by check if example.LuckyNumber is null create new list<int>**

**example.LuckyNumbers ??= new List<int>();**

**example.LuckyNumbers.Add(15);**

**foreach (var item in example.LuckyNumbers){**

**Console.WriteLine($"Luck Number : {item}");}}}**

**public class ListDemo{**

**public List<int> LuckyNumbers { get; set; }**

**//public ListDemo()//{// LuckyNumbers = new List<int> { 21 };//}}**