**//---------------------------------------------------------- Extension Methods -----------------------------------------------------------//**

// Modifying an already existing class definition to include new features breaks backward compatability for any code base using the original defined class.  
// Inheritance may not be preferred to extend the functionality of the types , due to the fact that we will now have to replace all those instances of the base class with the instances of the derived one where we want to use our new functionality.

// **Extension methods**– help to **add new functionality/methods or properties** to already existing type(class, struct or interfaces), ***without*** *having to change its original code/modifying the original type* or *use inheritance/creating a new derived type or subclass.*  
// So, Extension Methods also work fine with types that cannot be inherited – like a sealed class or structures.   
// This new functionality is offered to a type only if the extension methods have been referenced for use in your current project. In other words, to refer to a specific extension method, we **should add “using” and the corresponding namespace**, where the static class, describing this method, is defined. Otherwise the compiler has no way of knowing about their existence.   
// By default, class accessibility is set to private. So **when creating the static class** for containing extension methods in your project, **make the class public**.

// For **Extension methods** first restriction is that they **must be defined within a static class** ; therefore, **each extension method must** be declared with the **static** keyword. Recall : A static class can contain only static data members and static methods.  
// So, **Extension methods** are **static methods**, but they **are called as if they were instance methods on the extended type**. In other words, extension methods are *applied directly to objects of the class/interface they extend*. They can also be *invoked statically through the static class they are defined in*, but it is not a good practice.  
  
// The second restriction is that for all extension methods, the **type of their first parameter is the class (or the interface) they extend**. This first argument **must** **be marked** by using the **this** *keyword* as a modifier. Note: Place the *keyword* **this** on the first (and only the first) parameter of the method in question. // The “**this** qualified” **parameter** represents the item being extended. That is what **makes them different from other static methods**, and indicates the compiler that this is an extension method.   
// The parameter with the keyword **this** in front of it can be used in the method body to create its functionality. Practically, it is the object that is used by the extension method.

namespace LearningRecursion  
{  
 public class Program  
 {  
 static void Main(string[] args)  
 {  
 Console.Write("Enter a integer for Factorial :");  
 int number = int.Parse(Console.ReadLine());  
 decimal factorial = FactorialRecursive(number);  
 Console.WriteLine("The factorial of {0} is {1}.", number, factorial);  
 Console.ReadLine();  
 }  
 static decimal FactorialRecursive(int n)  
 {  
 if (n == 0)  
 {  
 return 1;  
 }  
 else  
 {  
 return n \* FactorialRecursive(n - 1);  
 }  
 }  
 }  
}