<u>Extension Methods</u>

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are a special kind of static methods, but they are called as if they were instance methods on the extended type. Extension methods are defined as static methods but are called by using instance method syntax. Their first parameter specifies which type the method operates on, and the parameter is preceded by the **this** modifier.

In your code you invoke the extension method with instance method syntax. However, the intermediate language (IL) generated by the compiler translates your code into a call on the static method. Therefore, the principle of encapsulation is not really being violated. In fact, extension methods cannot access private variables in the type they are extending. Rules for creating extension methods are as follows.

1. Extension Methods must be static.
2. Extension Methods must be in a static class.
3. First parameter in extension method must be preceded with the modifier **this.**
4. Type of first parameter must be the type you want to extend.
5. There is no need to pass an argument for the first parameter in extension methods.

<u>Syntax for creating Extension Method</u>

```
[Access Modifier] static returntype methodname(this type parameter[,parameters])
{

}
```

Example : The following example extends the string class of .net framework by adding a method print().

```
namespace ExtensionMethods
{
    static class Program
    {
        public static void Print(this string S)
        {
            Console.WriteLine(S);
        }
        static void Main(string[] args)
        {
            string S = "Naresh";
            S.Print();
        }
    }
}
```

Example : The following example extends int class of .net framework by adding two methods reverse() and fact().

```
namespace ExtensionMethods
{
    static class ExtendInt
    {
        public static int Reverse(this int N)
        {
            int R = 0;
            while (N > 0)
            {
                R = R * 10 + N % 10;
                N /= 10;
            }
            return R;
        }
        public static int Fact(this int N)
        {
            int F = 1;
            for (int i = 1; i <= N; i++)
            {
                F *= i;
            }
            return F;
        }
        static void Main()
        {
            int N1 = 234;
            int N2 = 5;
         Console.WriteLine("Reverse Of {0} Is {1}", N1, N1.Reverse());
         Console.WriteLine("Factorial Of {0} Is {1}", N2, N2.Fact());
        }
    }
}
```