Static keyword

A C# class may define any number of static members, which are declared using the static keyword. When you do so, the member in question must be invoked directly from the class level, rather than from an object reference variable.

There is no need to create an instance of the class before invoking the member. While any class can define static members, they are quite commonly found within *utility classes*. By definition, a utility class is a class that does not maintain any object-level state and is not created with the new keyword. Rather, a utility class exposes all functionality as class-level members. For example,in the System namespace of mscorlib.dll, you can see that all the members of the Console, Math, Environment, and GC classes expose all their functionality via static members. These are but a few utility classes found within the .NET base class libraries.

So, static members can be part of any class definition at all. Just remember that static members promote a given item to the class level rather than the object level.

static keyword can be applied to the following:

- Data of a class
- Methods of a class
- Properties of a class
- A constructor
- The entire class definition
- In conjunction with the C# using keyword

Most of the time when designing a class, you define data as instance-level data or, said another way, as nonstatic data. When you define instance-level data, you know that every time you create a new object, the object maintains its own independent copy of the data. In contrast, when you define **static data** of a class, the memory is shared by all objects of that category. So, the static data is allocated once and shared among all objects of the same class category.

It is a compiler error for a static member to reference nonstatic members in its implementation. on a related note, it is an error to use the this keyword on a static member because this implies an object!

A typical constructor is used to set the value of an object's instance-level data at the time of creation. **Static constructor** is a special constructor that is an ideal place to initialize the values of static data when the value is not known at compile time (e.g., you need to read in the value from an external file, read in the value from a database, generate a random number, or whatnot). The CLR calls all static constructors before the first use (and never calls them again for that instance of the application).

*Here are a few points of interest regarding **static constructors**:*

- A given class may define only a single static constructor. In other words, the static constructor cannot be overloaded.
- A static constructor does not take an access modifier and cannot take any parameters.
- A static constructor executes exactly one time, regardless of how many objects of the type are created.
- The runtime invokes the static constructor when it creates an instance of the class or before accessing the first static member invoked by the caller.
- The static constructor executes before any instance-level constructors.

When a **class** has been defined as **static**, it is not creatable using the new keyword, and it can contain only members or data fields marked with the static keyword. If this is not the case, you receive compiler errors.

Recall that a class (or structure) that exposes only static functionality is often termed a utility class. When designing a utility class, it is good practice to apply the static keyword to the class definition.

The following list provides the main features of a static class:

- Contains only static members.
- Cannot be instantiated.
- Is sealed.
- Cannot contain Instance Constructors.

```
public class Car
{
        public static int Speed { get; set; }
        public static void Drive() {   // some logic   }
}
static void Main(string[] args)
{
        Car.Speed = 150;
        Car.Drive();
}
```

C# supports importing **static members with the using keyword.** To illustrate, consider the C# file currently defining the utility class. Because you are making calls to the WriteLine() method of the Console class, as well as the Now property of the DateTime class, you must have a using statement for the System namespace. Since the members of these classes are all static, you could alter your code file with the following static using directives:

```
// Import the static members of Console and DateTime.
using static System.Console;
using static System.DateTime;
```