Answer on Question #42187, Programming, C# | for confirmation

When should we use static keyword in C# and in which scenarios we use non-static properties, functions and classes?

Static Demystified

Let's start with the memory first. Whenever a process is loaded in the RAM, we can say that the memory is roughly divided into three areas (within that process): **Stack**, **Heap**, and **Static** (which, in .NET, is actually a special area inside Heap only known as **High Frequency Heap**).

The static part holds the "`static`" member variables and methods. What exactly is static? Those methods and variables which don't need an instance of a class to be created are defined as being static. In C# (and Java too), we use the `static` keyword to label such members as static. For e.g.:

```
class MyClass
{
    public static int a;
    public static void DoSomething();
}
```

These member variables and methods can be called without creating an instance of the enclosing class. E.g., we can call the static method `DoSomething()` as:

```
MyClass.DoSomething();
```

We don't need to create an instance to use this static method.

m.DoSomething(); //wrong code. will result in compilation error.

An important point to note is that the static methods inside a class can only use static member variables of that class. Let me explain why:

Suppose you have a private variable in `MyClass` which is not static:

```
class MyClass
{
    // non-static instance member variable
    private int a;
    //static member variable
    private static int b;
    //static method
    public static void DoSomething()
    {
      //this will result in compilation error as "a" has no memory
      a = a + 1;
      //this works fine since "b" is static
      b = b + 1;
    }
}
```

Now, we will call the `DoSomething` method as:

```
MyClass.DoSomething();
```

Note that we have not created any instance of the class, so the private variable "`a`" has no memory as when we call a static method for a class, only the static variables are present in the memory (in the Static part). Instance variables, such as "`a`" in the above example, will only be created when we create an instance of the class using the "`new`" keyword, as:

```
MyClass m = new MyClass();  //now "a" will get some memory
```

But since we haven't created an instance yet, the variable "`a`" is not there in the process memory. Only "`b`" and "`DoSomething()`" are loaded. So when we call `DoSomething()`, it will try to increment the instance variable "`a`" by 1, but since the variable isn't created, it results in an error. The compiler flags an error if we try to use instance variables in static methods.

Now, what is a *static class*? When we use the `static` keyword before a class name, we specify that the class will only have static member variables and methods. Such classes cannot be instantiated as they don't need to: they cannot have instance variables. Also, an important point to note is that such static classes are `sealed` by default, which means they cannot be inherited further.

This is because static classes have no behavior at all. There is no need to derive another class from a static class (we can create another static class).

Why do we need static classes? As already written above, we need static classes when we know that our class will not have any behavior as such. Suppose we have a set of helper or utility methods which we would like to wrap together in a class. Since these methods are generic in nature, we can define them all inside a static class. Remember that helper or utility methods need to be called many times, and since they are generic in nature, there is no need to create instances. E.g., suppose that you need a method that parses an `int` to a `string`. This method would come in the category of a utility or helper method.

So using the `static` keyword will make your code a bit faster since no object creation is involved.

An important point to note is that a static class in C# is different from one in Java. In Java, the `static` modifier is used to make a member class a nested top level class inside a package. So using the `static` keyword with a class is different from using it with member variables or methods in Java (static member variables and methods are similar to the ones explained above in C#).

Also, the `static` keyword in C++ is used to specify that variables will be in memory till the time the program ends; and initialized only once. Just like C# and Java, these variables don't need an object to be declared to use them. Please see this link for the use of the `static` keyword in C++:

Writing about the `const` keyword brings me to a subtle but important distinction between `const` and `readonly` keywords in C#: `const` variables are implicitly static and they need to be defined when declared. `readonly` variables are not implicitly static and can only be initialized once.

E.g.: You are writing a car racing program in which the racing track has a fixed length of 100 Km. You can define a `const` variable to denote this as:

```
private const int trackLength = 100;
```

Now, you want the user to enter the number of cars to race with. Since this number would vary from user to user, but would be constant throughout a game, you need to make it `readonly`. You cannot make it a `const` as you need to initialize it at runtime. The code would be like:

```
public class CarRace
{
    //this is compile time constant
    private const int _trackLength = 100;
    //this value would be determined at runtime, but will
    //not change after that till the class's
    //instance is removed from memory
    private readonly int _noOfCars;

    public CarRace(int noOfCars)
    {}

    public CarRace(int noOfCars)
    {
        ///<REMARKS>
        ///Get the number of cars from the value
        ///use has entered passed in this constructor
        ///</REMARKS>
        _noOfCars = noOfCars;
    }
}
```

Summary

We examined the `static` keyword in C#, and saw how it helps in writing good code. It is best to think and foresee possible uses of the `static` keyword so that the code efficiency, in general, increases.