C# - Static Class, Methods, Constructors, Fields

In C#, static means something which cannot be instantiated. You cannot create an object of a static class and cannot access static members using an object.

C# classes, variables, methods, properties, operators, events, and constructors can be defined as static using the `static` modifier keyword.

Static Class

Apply the `static` modifier before the class name and after the access modifier to make a class static. The following defines a static class with static fields and methods.

Example: C# Static Class

```
public static class Calculator
{
    private static int _resultStorage = 0;

    public static string Type = "Arithmetic";

    public static int Sum(int num1, int num2)
    {
        return num1 + num2;
    }

    public static void Store(int result)
    {
        _resultStorage = result;
    }
}
```

Above, the `Calculator` class is a static. All the members of it are also static.

You cannot create an object of the static class; therefore the members of the static class can be accessed directly using a class name like `ClassName.MemberName`, as shown below.

Example: Accessing Static Members

```
class Program
{
    static void Main(string[] args)
    {
        var result = Calculator.Sum(10, 25); // calling static method
        Calculator.Store(result);

        var calcType = Calculator.Type; // accessing static variable
        Calculator.Type = "Scientific"; // assign value to static variable
    }
}
```

Rules for Static Class

1. Static classes cannot be instantiated.
2. All the members of a static class must be static; otherwise the compiler will give an error.
3. A static class can contain static variables, static methods, static properties, static operators, static events, and static constructors.
4. A static class cannot contain instance members and constructors.
5. Indexers and destructors cannot be static
6. `var` cannot be used to define static members. You must specify a type of member explicitly after the `static` keyword.
7. Static classes are sealed class and therefore, cannot be inherited.
8. A static class cannot inherit from other classes.
9. Static class members can be accessed using `ClassName.MemberName`.
10. A static class remains in memory for the lifetime of the application domain in which your program resides.

Static Members in Non-static Class

The normal class (non-static class) can contain one or more static methods, fields, properties, events and other non-static members.

It is more practical to define a non-static class with some static members, than to declare an entire class as static.

Static Fields

Static fields in a non-static class can be defined using the `static` keyword.

Static fields of a non-static class is shared across all the instances. So, changes done by one instance would reflect in others.

Example: Shared Static Fields

```
public class StopWatch
{
    public static int InstanceCounter = 0;
    // instance constructor
    public StopWatch()
    {
    }
}

class Program
{
    static void Main(string[] args)
    {
        StopWatch sw1 = new StopWatch();
        StopWatch sw2 = new StopWatch();
        Console.WriteLine(StopWatch.NoOfInstances); //2

        StopWatch sw3 = new StopWatch();
        StopWatch sw4 = new StopWatch();
        Console.WriteLine(StopWatch.NoOfInstances);//4
    }
}
```

Static Methods

You can define one or more static methods in a non-static class. Static methods can be called without creating an object. You cannot call static methods using an object of the non-static class.

The static methods can only call other static methods and access static members. You cannot access non-static members of the class in the static methods.

Example: Static Method

```
class Program
{
    static int counter = 0;
    string name = "Demo Program";

    static void Main(string[] args)
    {
        counter++; // can access static fields
        Display("Hello World!"); // can call static methods

        name = "New Demo Program"; //Error: cannot access non-static members
        SetRootFolder("C:\MyProgram"); //Error: cannot call non-static method
    }

    static void Display(string text)
    {
        Console.WriteLine(text);
    }

    public void SetRootFolder(string path) {  }
}
```

Rules for Static Methods

1. Static methods can be defined using the `static` keyword before a return type and after an access modifier.
2. Static methods can be overloaded but cannot be overridden.
3. Static methods can contain local static variables.
4. Static methods cannot access or call non-static variables unless they are explicitly passed as parameters.

Static Constructors

A non-static class can contain a parameterless static constructor. It can be defined with the static keyword and without access modifiers like public, private, and protected.

The following example demonstrates the difference between static constructor and instance constructor.

Example: Static Constructor vs Instance Constructor

```
public class StopWatch
{
    // static constructor
    static StopWatch()
    {
        Console.WriteLine("Static constructor called");
    }

    // instance constructor
    public StopWatch()
    {
        Console.WriteLine("Instance constructor called");
    }

    // static method
    public static void DisplayInfo()
    {
        Console.WriteLine("DisplayInfo called");
    }

    // instance method
    public void Start() { }

    // instance method
    public void Stop() {  }
}
```

Above, the non-static class `StopWatch` contains a static constructor and also a non-static constructor.

The static constructor is called only once whenever the static method is used or creating an instance for the first time. The following example shows that the static constructor gets called when the static method called for the first time. Calling the static method second time onwards won't call a static constructor.

Example: Static Constructor Execution

```
StopWatch.DisplayInfo(); // static constructor called here
StopWatch.DisplayInfo(); // none of the constructors called here
```

Output:

```
Static constructor called.
DisplayInfo called
DisplayInfo called
```

The following example shows that the static constructor gets called when you create an instance for the first time.

Example: Static Constructor Execution

```
StopWatch sw1 = new StopWatch(); // First static constructor and then instance
constructor called
StopWatch sw2 = new StopWatch();// only instance constructor called
StopWatch.DisplayInfo();
```

Output:

```
Static constructor called
instance constructor called
instance constructor called
DisplayInfo called
```

Rules for Static Constructors

1. The static constructor is defined using the `static` keyword and without using access modifiers public, private, or protected.
2. A non-static class can contain one parameterless static constructor. Parameterized static constructors are not allowed.
3. Static constructor will be executed only once in the lifetime. So, you cannot determine when it will get called in an application if a class is being used at multiple places.
4. A static constructor can only access static members. It cannot contain or access instance members.

Note:
Static members are stored in a special area in the memory called High-Frequency Heap. Static members of non-static classes are shared across all the instances of the class. So, the changes done by one instance will be reflected in all the other instances.