class SimpleClass

{

**static readonly long baseline;**

static SimpleClass()

{

**baseline = DateTime.Now.Ticks;**

}

}

1. Static variable that must be initialized at run time.
2. Static constructor is called at most one time, before any instance constructor is invoked or member is accessed.
3. A static constructor doesn't take access modifiers or have parameters.
4. A class or struct can only have one static constructor.
5. Static constructors cannot be inherited or overloaded.
6. A static constructor cannot be called directly and is only meant to be called by the common language runtime (CLR). It is invoked automatically.
7. The user has no control on when the static constructor is executed in the program.
8. A static constructor is called automatically. It initializes the [class](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) before the first instance is created or any static members declared in that class (not its base classes) are referenced. A static constructor runs before an instance constructor. A type's static constructor is called when a static method assigned to an event or a delegate is invoked and not when it is assigned. If static field variable initializers are present in the class of the static constructor, they're executed in the textual order in which they appear in the class declaration. The initializers run immediately prior to the execution of the static constructor.
9. If you don't provide a static constructor to initialize static fields, all static fields are initialized to their default value as listed in [Default values of C# types](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/default-values).
10. If a static constructor throws an exception, the runtime doesn't invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain. Most commonly, a [TypeInitializationException](https://learn.microsoft.com/en-us/dotnet/api/system.typeinitializationexception) exception is thrown when a static constructor is unable to instantiate a type or for an unhandled exception occurring within a static constructor. For static constructors that aren't explicitly defined in source code, troubleshooting may require inspection of the intermediate language (IL) code.
11. The presence of a static constructor prevents the addition of the [BeforeFieldInit](https://learn.microsoft.com/en-us/dotnet/api/system.reflection.typeattributes" \l "system-reflection-typeattributes-beforefieldinit) type attribute. This limits runtime optimization.
12. A field declared as static readonly may only be assigned as part of its declaration or in a static constructor. When an explicit static constructor isn't required, initialize static fields at declaration rather than through a static constructor for better runtime optimization.
13. The runtime calls a static constructor no more than once in a single application domain. That call is made in a locked region based on the specific type of the class. No additional locking mechanisms are needed in the body of a static constructor. To avoid the risk of deadlocks, don't block the current thread in static constructors and initializers. For example, don't wait on tasks, threads, wait handles or events, don't acquire locks, and don't execute blocking parallel operations such as parallel loops, Parallel.Invoke and Parallel LINQ queries.

**Usage**

* A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
* Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method.
* Static constructors are also a convenient place to enforce run-time checks on the type parameter that cannot be checked at compile time via type-parameter constraints.

public class Bus

{

// Static variable used by all Bus instances.

// Represents the time the first bus of the day starts its route.

protected static readonly DateTime globalStartTime;

// Property for the number of each bus.

protected int RouteNumber { get; set; }

// Static constructor to initialize the static variable.

// It is invoked before the first instance constructor is run.

static Bus()

{

globalStartTime = DateTime.Now;

// The following statement produces the first line of output,

// and the line occurs only once.

Console.WriteLine("Static constructor sets global start time to {0}",

globalStartTime.ToLongTimeString());

}

// Instance constructor.

public Bus(int routeNum)

{

RouteNumber = routeNum;

Console.WriteLine("Bus #{0} is created.", RouteNumber);

}

// Instance method.

public void Drive()

{

TimeSpan elapsedTime = DateTime.Now - globalStartTime;

// For demonstration purposes we treat milliseconds as minutes to simulate

// actual bus times. Do not do this in your actual bus schedule program!

Console.WriteLine("{0} is starting its route {1:N2} minutes after global start time {2}.",

this.RouteNumber,

elapsedTime.Milliseconds,

globalStartTime.ToShortTimeString());

}

}

class TestBus

{

static void Main()

{

// The creation of this instance activates the static constructor.

Bus bus1 = new Bus(71);

// Create a second bus.

Bus bus2 = new Bus(72);

// Send bus1 on its way.

bus1.Drive();

// Wait for bus2 to warm up.

System.Threading.Thread.Sleep(25);

// Send bus2 on its way.

bus2.Drive();

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}