When to use static classes in C# [duplicate]

Asked 16 years, 1 month ago Modified 4 years, 10 months ago Viewed 533k times

678

This question already has answers here:

Class with single method -- best approach? (15 answers)

Closed 9 years ago.

Here's what MSDN has to say under *When to Use Static Classes*:

```
static class CompanyInfo
{
    public static string GetCompanyName() { return "CompanyName"; }
    public static string GetCompanyAddress() { return "CompanyAddress"; }
    //...
}
```

> Use a static class as a unit of organization for methods not associated with particular objects. Also, a static class can make your implementation simpler and faster because you do not have to create an object in order to call its methods. It is useful to organize the methods inside the class in a meaningful way, such as the methods of the Math class in the System namespace.

To me, that example doesn't seem to cover very many possible usage scenarios for static classes. In the past I've used static classes for stateless suites of related functions, but that's about it. So, under what circumstances should (and shouldn't) a class be declared static?

c# class static

Share Improve this question

Follow

edited May 30, 2018 at 0:25

Ryan

3,215 ● 6 ● 33 ● 49

asked Oct 27, 2008 at 20:53

pbh101

10.4k ● 9 ● 33 ● 31

30 As a newbie in C#, it would be helpful to explain as to why this has been marked as a duplicate question of singleton vs static class and how does the two correlates with each other. – mr5 Jul 5, 2017 at 4:10

1 mr5, singleton and static class is basicly the exact same thing. Singleton is a design pattern used in other languages to simulate a Static class, since other languages (like Java) don't have built in Static classes, so you have to rely on the Singleton design patter to create such class. The Static class is a class that can't be instantiated and can be directly used (like the Console class for example). tutorialspoint.com/design_pattern/singleton_pattern.htm if you check this, you will see

that when you use the Singleton you are not creating a new instance... – Darkbound Jul 6, 2017 at 21:37

... you are using the one that was already created inside Singleton class, and you access it by the .getInstance() method. C# solves all of that by one simple keyword "static". – Darkbound Jul 6, 2017 at 21:39

8 Singleton and Static classes are fundamentally exactly opposite things. One can be instantiated, the other is forbidden from being instantiated. – Ian Boyd Nov 20, 2017 at 19:51

IMHO when designing attributes for the object think about instantiation for inside the box & static class for out of the box. – ashveli Oct 11, 2019 at 6:37

11 Answers

Sorted by: Highest score (default)

790

I wrote my thoughts of static classes in an earlier Stack Overflow answer: *Class with single method -- best approach?*

I used to love utility classes filled up with static methods. They made a great consolidation of helper methods that would otherwise lie around causing redundancy and maintenance hell. They're very easy to use, no instantiation, no disposal, just fire'n'forget. I guess this was my first unwitting attempt at creating a service-oriented architecture - lots of stateless services that just did their job and nothing else. As a system grows however, dragons be coming.

Polymorphism

Say we have the method UtilityClass.SomeMethod that happily buzzes along. Suddenly we need to change the functionality slightly. Most of the functionality is the same, but we have to change a couple of parts nonetheless. Had it not been a static method, we could make a derivate class and change the method contents as needed. As it's a static method, we can't. Sure, if we just need to add functionality either before or after the old method, we can create a new class and call the old one inside of it - but that's just gross.

Interface woes

Static methods cannot be defined through interfaces for logic reasons. And since we can't override static methods, static classes are useless when we need to pass them around by their interface. This renders us unable to use static classes as part of a strategy pattern. We might patch some issues up by passing delegates instead of interfaces.

Testing

This basically goes hand in hand with the interface woes mentioned above. As our ability of interchanging implementations is very limited, we'll also have trouble replacing production code with test code. Again, we can wrap them up, but it'll require us to change large parts of our code just to be able to accept wrappers instead of the actual objects.

Fosters blobs

As static methods are usually used as utility methods and utility methods usually will have different purposes, we'll quickly end up with a large class filled up with non-coherent functionality - ideally, each class should have a single purpose within the system. I'd much rather have a five times the classes as long as their purposes are well defined.

Parameter creep

To begin with, that little cute and innocent static method might take a single parameter. As functionality grows, a couple of new parameters are added. Soon further parameters are added that are optional, so we create overloads of the method (or just add default values, in languages that support them). Before long, we have a method that takes 10 parameters. Only the first three are really required, parameters 4-7 are optional. But if parameter 6 is specified, 7-9 are required to be filled in as well... Had we created a class with the single purpose of doing what this static method did, we could solve this by taking in the required parameters in the constructor, and allowing the user to set optional values through properties, or methods to set multiple interdependent values at the same time. Also, if a method has grown to this amount of complexity, it most likely needs to be in its own class anyway.

Demanding consumers to create an instance of classes for no reason

One of the most common arguments is: Why demand that consumers of our class create an instance for invoking this single method, while having no use for the instance afterwards? Creating an instance of a class is a very very cheap operation in most languages, so speed is not an issue. Adding an extra line of code to the consumer is a low cost for laying the foundation of a much more maintainable solution in the future. And finally, if you want to avoid creating instances, simply create a singleton wrapper of your class that allows for easy reuse - although this does make the requirement that your class is stateless. If it's not stateless, you can still create static wrapper methods that handle everything, while still giving you all the benefits in the long run. Finally, you could also make a class that hides the instantiation as if it was a singleton: MyWrapper.Instance is a property that just returns `new MyClass();`

Only a Sith deals in absolutes

Of course, there are exceptions to my dislike of static methods. True utility classes that do not pose any risk to bloat are excellent cases for static methods - System.Convert as an example. If your project is a one-off with no requirements for future maintenance, the overall architecture really isn't very important - static or non static, doesn't really matter - development speed does, however.

Standards, standards, standards!

Using instance methods does not inhibit you from also using static methods, and vice versa. As long as there's reasoning behind the differentiation and it's standardised. There's nothing worse than looking over a business layer sprawling with different implementation methods.

Share Improve this answer

Follow

edited Feb 21, 2020 at 20:05

Callum Watkins

2,981 ● 4 ● 32 ● 53

answered Oct 27, 2008 at 21:01

Mark S. Rasmussen

35.4k ● 4 ● 42 ● 58

13 Steve Yegge has written about it as well. But his post is wayyy longer than could be post here. If you're still not convinced, try steve.yegge.googlepages.com/singleton-considered-stupid – chakrit Oct 27, 2008 at 21:26

338 Has anyone ever noticed that the statement, "Only a Sith deals in absolutes," is an absolute? Sorry. Couldn't help it. – John Kraft Oct 27, 2008 at 21:44

5 Just as I go out on a limb to get my point across, so does Yegge. Some of his points comes down to normal programming consideration - if the singleton keeps valuable resources open, that may be a problem, of course. Given careful consideration, there are places for singletons and statics alike. – Mark S. Rasmussen Oct 27, 2008 at 22:16

70 @John Kraft: That was obviously written by a Sith. – ChrisLively Dec 9, 2010 at 19:53

10 His question was 'when to USE static classes' - but you appear to have answered the question 'when NOT to use static classes' - subtle. But I'm still none the wise as to when you SHOULD be using them. A few examples of appropriate use would be good + what if two threads call a static class or method very close together - when is that good / bad? – niico Nov 12, 2017 at 13:22

171

When deciding whether to make a class static or non-static you need to look at what information you are trying to represent. This entails a more '**bottom-up**' style of programming where you focus on the data you are representing first. Is the class you are writing a real-world object like a rock, or a chair? These things are physical and have physical attributes such as color, weight which tells you that you may want to instantiate multiple objects with different properties. I may want a black chair AND a red chair at the same time. If you ever need two configurations at the same time then you instantly know you will want to instantiate it as an object so each object can be unique and exist at the same time.

On the other end, static functions tend to lend more to actions which do not belong to a real-world object or an object that you can easily represent. Remember that C#'s predecessors are C++ and C where you can just define global functions that do not exist in a class. This lends more to '**top-down**' programming. Static methods can be used for these cases where it doesn't make sense that an 'object' performs the task. By forcing you to use classes this just makes it easier to group related functionality which helps you create more maintainable code.

Most classes can be represented by either static or non-static, but when you are in doubt just go back to your OOP roots and try to think about what you are representing. Is this an object that is performing an action (a car that can speed up, slow down, turn) or something more abstract (like displaying output).

Get in touch with your inner OOP and you can never go wrong!

Share Improve this answer

Follow

edited Jun 14, 2013 at 14:27

Timothy S. Van Haren

8,956 ● 2 ● 31 ● 34

answered Feb 1, 2012 at 2:50

Despertar

22.3k ● 11 ● 84 ● 81

18 “You must be shapeless, formless, like water. When you pour water in a cup, it becomes the cup. When you pour water in a bottle, it becomes the bottle. When you pour water in a teapot, it becomes the teapot. Water can drip and it can crash. Become like water my friend.” - Bruce Lee – Chef_Code Sep 19, 2016 at 20:21

7 @Chef_Code I'm sorry I couldn't understand - how is your quote relevant to given question if you don't mind me asking? – Jogi Sep 21, 2016 at 13:19

You can create in classes in C++, pretty much the same as C#, that is the main difference between C++ and C. – user2802557 Jan 16, 2017 at 13:53

2 GREAT explanation - best one I've seen for this in terms of simplicity and conciseness. I would recommend this one plus Mark Rasmussen's for more detail, to anyone on this topic! – user2075599 Jun 26, 2017 at 0:52

38

For C# 3.0, extension methods may only exist in top-level static classes.

Share Improve this answer

Follow

edited Oct 27, 2008 at 21:04

Jason Bunting

58.9k ● 15 ● 104 ● 94

answered Oct 27, 2008 at 20:57

Mark Cidade

99.8k ● 33 ● 229 ● 237

31

If you use code analysis tools (e.g. FxCop), it will recommend that you mark a method `static` if that method don't access instance data. The rationale is that there is a performance gain. MSDN: CA1822 - Mark members as static.

It is more of a guideline than a rule, really...

Share Improve this answer

Follow

edited Jun 3, 2018 at 8:27

Community Bot

1 ● 1

answered Oct 27, 2008 at 21:39

user25306

421 ● 2 ● 6 ● 7

4 In addition such a static declaration on a private method in an instance class conveys useful additional info to a developer. It informs that a such private static method does not change instance state .i.e. the method is a utility method within a class. – camelCase Aug 6, 2017 at 9:42

15

Static classes are very useful and have a place, for example libraries.

The best example I can provide is the .Net Math class, a System namespace static class that contains a library of maths functions.

It is like anything else, use the right tool for the job, and if not anything can be abused.

Blankly dismissing static classes as wrong, don't use them, or saying "there can be only one" or none, is as wrong as over using the them.

C#.Net contains a number of static classes that is uses just like the Math class.

So given the correct implementation they are tremendously useful.

We have a static TimeZone class that contains a number of business related timezone functions, there is no need to create multiple instances of the class so much like the Math class it contains a set of globally accesible TimeZone realated functions (methods) in a static class.

Share Improve this answer

Follow

edited Dec 15, 2013 at 13:13

BenMorel

36.4k • 51 • 202 • 334

answered Jan 4, 2013 at 23:47

Don

789 • 6 • 3

1 The answer saying "there can be only one" is not talking about the number of static classes, but rather cutely (though rather incorrectly) alluding to the fact that there can only be one instance (wrong, because actually there can only be zero instances) – Kirk Woll Jan 5, 2013 at 0:48

Ah thanks Kirk, hence my confusion; in any case simply dismissing static or blanketly stating that the singleton pattern is better (as I have seen) is in my opinion taking a very valuable tool out of the toolbox. – Don Jan 7, 2013 at 20:17

"There can be only one" is talking about concepts or things in the real world that can be represented by a static class. As with System.Math, you may need only one way of taking the particular input and going away and getting the correct answer. There can be only one way of doing basic math - anything else is too specialised to be associated with the standard library. Another example might be a simple video game with only one room or one playing field. – Bloopy Nov 27, 2017 at 16:11

13

I do tend to use static classes for factories. For example, this is the logging class in one of my projects:

```
public static class Log
{
   private static readonly ILoggerFactory _loggerFactory =
      IoC.Resolve<ILoggerFactory>();

   public static ILogger For<T>(T instance)
   {
      return For(typeof(T));
   }

   public static ILogger For(Type type)
   {
      return _loggerFactory.GetLoggerFor(type);
   }
}
```

You might have even noticed that IoC is called with a static accessor. *Most* of the time for me, if you can call static methods on a class, that's all you can do so I mark the class as static for extra clarity.

Share Improve this answer Follow

answered Oct 27, 2008 at 21:11

Rob

1,983 ● 2 ● 20 ● 29

Why do you define an unused parameter in the generic method? – John ClearZ Dec 28, 2013 at 19:29

4 @JohnClearZ maybe because then you can call `For` with an existing object and you get the logger for it without thinking about the actual type of the object. Just a guess. – Philipp M Sep 12, 2014 at 11:41

12

I've started using static classes when I wish to use functions, rather than classes, as my unit of reuse. Previously, I was all about the evil of static classes. However, learning F# has made me see them in a new light.

What do I mean by this? Well, say when working up some super DRY code, I end up with a bunch of one-method classes. I may just pull these methods into a static class and then inject them into dependencies using a delegate. This also plays nicely with my dependency injection (DI) container of choice Autofac.

Of course taking a direct dependency on a static method is still *usually* evil (there are some non-evil uses).

Share Improve this answer Follow

edited Aug 3, 2016 at 16:39

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jul 1, 2011 at 9:34

bentayloruk

4,101 ● 30 ● 31

9

I use static classes as a means to define "extra functionality" that an object of a given type could use under a specific context. Usually they turn out to be utility classes.

Other than that, I think that "Use a static class as a unit of organization for methods not associated with particular objects." describe quite well their intended usage.

Share Improve this answer Follow

answered Oct 27, 2008 at 21:05

Trap

12.4k ● 15 ● 57 ● 69

8

This is another old but very hot question since OOP kicked in. There are many reasons to use(or not) a static class, of course and most of them have been covered in the multitude of answers.

I will just add my 2 cents to this, saying that, I make a class static, when this class is something that would be unique in the system and that would really make no sense to have any instances of it in the program. However, I reserve this usage for big classes. I never declare such small classes as in the MSDN example as "static" and, certainly, not classes that are going to be members of other classes.

I also like to note that static *methods* and static *classes* are two different things to consider. The main disadvantages mentioned in the accepted answer are for static *methods*. static *classes* offer the same flexibility as normal classes(where properties and parameters are concerned), and all methods used in them should be relevant to the purpose of the existence of the class.

A good example, in my opinion, of a candidate for a static class is a "FileProcessing" class, that would contain all methods and properties relevant for the program's various objects to perform complex FileProcessing operations. It hardly has any meaning to have more than one instance of this class and being static will make it readily available to everything in your program.

answered Oct 17, 2014 at 7:46

ThunderGr

2,357 ● 29 ● 20

I only use static classes for helper methods, but with the advent of C# 3.0, I'd rather use extension methods for those.

I rarely use static classes methods for the same reasons why I rarely use the singleton "design pattern".

3

Share Improve this answer Follow

answered Oct 27, 2008 at 21:00

jonnii
28.3k ● 8 ● 82 ● 110

5 I'd rather not use extension methods for helper methods. Extension methods are messy, confusing for other developers and not generally intuitive across other languages. Extension methods have their purpose, but not as generic helper methods imho. – Mark S. Rasmussen Oct 27, 2008 at 21:02

2 Just to clarify, by helper method I mean something like: string.ToSentence(), or string.Camelize(). Essentially anything that would live in a class like StringHelpers. – jonnii Oct 27, 2008 at 21:15

Extension methods put you on a slippery slope...I think if you find yourself adding more than 2 extension methods for a given type, it might be time to examine why. – Jason Bunting Oct 27, 2008 at 21:45

1 Those two examples, I would especially not like. The more generic the type, the less suitable it is for extension methods. The worst example is beginning to add ToInt32() method to Object, ToDateTime() etc. I'd much rather have these in separate classes. – Mark S. Rasmussen Oct 27, 2008 at 21:49

Based on MSDN:

1. You cannot create the instance for static classes
2. If the class declared as static, member variable should be static for that class
3. Sealed [Cannot be Inherited]
4. Cannot contains Instance constructor
5. Memory Management

0

Example: Math calculations (math values) does not changes [STANDARD CALCULATION FOR DEFINED VALUES]

Share Improve this answer Follow

edited Dec 30, 2014 at 21:19

Amir
724 ● 1 ● 13 ● 38

answered Dec 12, 2014 at 6:57

Vicky
829 ● 2 ● 13 ● 30

16 You are just describing what a static class *is*, instead of describing when they would be practical, or what their *purpose* is. – Zack Apr 9, 2015 at 21:48

1 The question is about when to use static class and not what you can do with static class. – Mohit Dharmadhikari Aug 7, 2018 at 4:51