Practical Reasons to use C# "named arguments"

Asked 9 years, 6 months ago Modified 9 years, 6 months ago Viewed 630 times

0

I find this feature "named arguments" from C# to be really weird because I see two flaws from it. The book says "named arguments give you the "ability to pass arguments in any order".

Two flaws I think is a problem for this C# feature:

1. It violates "information hiding" in computer science. (i.e.: the end user using the method needs to know the parameter name and data type to make use of the feature.) Coming from a Java background, this is weird. Why expose parameter names to the end of the user?
2. It is prone to ambiguity which can lead to bugs. (The programmer needs to do extra thinking and problems can creep up when the programmer wind up writing methods that use the same method name(aka overloading methods"). You will always get a "call is ambiguous when you have two methods with same name with the same parameters for each other even if the other method has a extra parameters with different data types. The only fix I can think of is to make the data type a "mandatory parameter" aka a parameter without a default value so the compiler does not get confused. But then this fix is only a bandage solution to leads to another worst case scenario (see below)

Do people in the industry even use this concept to this day? If so, why break the two rules to be given the "ability to pass arguments in any order when calling the method"?

TLDR: An Example to bring clarity to what I am talking about by introducing a possible worst case scenario(the compiler chose the wrong method call... despite resemblance of the two methods):

```
namespace ConsoleApplication1
{
    class Venusaur
    {
        static void Main(string[] args)
        {

            new Venusaur().optMethod(fourth: "s");
        }

        public void optMethod(  string third , string fourth =  "hello", int
fifth = 23, string two = "w")
        {
            // what if I wanted this method to run instead of the method below
```

```
me
            Console.WriteLine("did not execute");
        }


        public void optMethod(string third = "Byte", string fourth = "hello",
int fifth = 4)
        {
            // But this method ran instead
            Console.WriteLine("run");

        }


    }
}
```

c# methods parameters arguments optional-parameters

Share

Improve this question

Follow

edited Jun 20, 2015 at 15:50

Sumner Evans

9,147 ● 5 ● 31 ● 48

asked Jun 20, 2015 at 15:48

Nicholas

679 ● 2 ● 11 ● 30

If you wanted to run the first declaration of `optMethod` you would need to explicitly specify the `third` variable: `new Venusaur().optMethod("thirdStr", fourth: "s");` – Sumner Evans Jun 20, 2015 at 15:52

I'm not sure if this question will be considered on topic. However, to answer your question: using named arguments is quite unusual, even though it's allowed in C#. The only real world example I can recall is XNA and MonoGame (and it looked ugly and weird to be honest, I think they could have done a lot better). – Theodoros Chatzigiannakis Jun 20, 2015 at 15:59

@jsve explicitly specifying the third variable won't work because both method the same parameter names so the compiler gets confused. it will produce the "ambiguity method call error". I already tried this code: new Venusaur().optMethod(third: "thirdStr", fourth: "s"); Not quite sure how you "highlighted your code like that" for responses and replies. – Nicholas Jun 20, 2015 at 16:01

@Nicholas I see your point here. C#'s method resolution is complex, but it's well specified and named parameters don't change those rules either: stackoverflow.com/questions/5173339/… – ATL_DEV Jun 25, 2021 at 3:33

4 Answers

Sorted by: Highest score (default)

8

Named arguments is yet another piece of technology that was added to make interoperability with COM (and similar technologies) easier.

Take the method to open a Word document using COM:

```
Document Open(
    [In] ref object FileName,
    [In, Optional] ref object ConfirmConversions,
    [In, Optional] ref object ReadOnly,
    [In, Optional] ref object AddToRecentFiles,
    [In, Optional] ref object PasswordDocument,
    [In, Optional] ref object PasswordTemplate,
    [In, Optional] ref object Revert,
    [In, Optional] ref object WritePasswordDocument,
    [In, Optional] ref object WritePasswordTemplate,
    [In, Optional] ref object Format,
    [In, Optional] ref object Encoding,
    [In, Optional] ref object Visible,
    [In, Optional] ref object OpenAndRepair,
    [In, Optional] ref object DocumentDirection,
    [In, Optional] ref object NoEncodingDialog,
    [In, Optional] ref object XMLTransform
);
```

It has no less than 16 parameters, 15 of which are optional.

What if you want to specify the `XMLTransform` parameter but don't care about the rest?

You would have to specify the rest with positional arguments. With named arguments the call simply becomes:

```
doc.Open("somefilename.doc", XMLTransform: xxx);
```

Along with `dynamic` and a few other things, it wasn't meant to be a good solution to a lot of problems but when you have an API that relies heavily on methods with lots and lots of parameters, most of which are optional, named arguments makes sense.

Yes, you *can* specify arguments out of order. It doesn't mean it's a good idea or that it solves a problem someone really had.

Share Improve this answer Follow

answered Jun 20, 2015 at 16:21

Lasse V. Karlsen

391k ● 106 ● 646 ● 844

3

I find your first argument completely specious. Like it or not the names of parameters **are** part of each method's semantics, particularly for abstract classes and methods that will be overridden in subclasses. The names of the parameters are a key signalling to users of a method on the semantics of that method, and of each parameters role in that semantics. Further it is essential for understanding of the

semantics throughout the class hierarchy that parameter names be consistent, and not subject to the whims of individual programmers at each level.

Your second argument I find completely unintelligible. If you actually have a valid point here I suggest you rewrite this to achieve greater clarity.

That said, I rarely use named parameters in my method calls; the exception being for methods that accept more than two consecutive parameters of the same type. In this case I like to name them so that I, as the future reader of the code, can in fact decipher the method's use without having to constantly hover to see the intelli-sense.

Share
Improve this answer
Follow

edited Jun 20, 2015 at 16:26

answered Jun 20, 2015 at 16:06

Pieter Geerkens
11.9k ● 2 ● 33 ● 53

> Parameter names **are not** part of the method's signature. According to the C# specification (3.6): *The signature of a method consists of the name of the method, the number of type parameters and the type and kind (value, reference, or output) of each of its formal parameters, considered in the order left to right.* – Theodoros Chatzigiannakis Jun 20, 2015 at 16:17

> Though overload resolution will use the parameter names to find eligible methods. – Lasse V. Karlsen Jun 20, 2015 at 16:18

> @TheodorosChatzigiannakis: I have changed *signature* to *semantics*, as that meets the technical point you make and more clearly states the intent of my argument. Yet as Lasse Karlsen notes, neither are the parameter names *completely excluded* from the compiler's interpretation of the code. – Pieter Geerkens Jun 20, 2015 at 16:27

> @PieterGeerkens fair enough. I will either rethink the second part I said or just accept the quirks that is C#. It's just that Java does not have this feature called "named arguments" and using it is still foreign. – Nicholas Jun 20, 2015 at 17:51

2

1) I don't understand where "information hiding" lies there ; a method signature is what you ask the end user to give you. He obviously needs to know the data type and name to disambiguate why argument refers to what. Also he still doesn't know what you gonna do with those parameters you asked for (maybe nothing) so what needs to be hidden from him is still hidden.

2) the problem here is in a bad choice for two overloads which differ only by number of arguments, that combined with the ability to omit certain arguments (provided by optional arguments) makes you lose the "benefit" to distinguish them (unless you give all the arguments)

Named parameters were (partly) introduced with dynamic for interop scenario ; for example with Office Automation where in C#3 you needed to add a bunch of

`Type.Missing` arguments and then in C#4 you can just provide those you want/need.

Share Improve this answer Follow

answered Jun 20, 2015 at 16:24

Sehnsucht

5,049 ● 18 ● 27

0

The point about COM inter-op is most important.

Personally I tend to use it if I have a boolean parameter in the method signature that I want to use but there is no obvious variable that I can use. Example:

I find more readable

```
var isSuccessful = myObject.Method(isRecursive: true)
```

then

```
var isSuccessful = myObject.Method(true);
```

eventually you could more verbose syntax

```
var isRecursive = true;
var isSuccessful = myObject.Method(isRecursive);
```

to avoid named parameter.

Obvious disadvantage of calling methods like that is if for some reason signature of the method changes, you code will no longer build.

Share Improve this answer Follow

answered Jun 20, 2015 at 17:07

interjaz

193 ● 1 ● 1 ● 8