Initialize class fields in constructor or at declaration?

Asked 16 years, 4 months ago Modified 2 years ago Viewed 172k times

472

I've been programming in C# and Java recently and I am curious where the best place is to initialize my class fields.

Should I do it at declaration?:

```
public class Dice
{
    private int topFace = 1;
    private Random myRand = new Random();

    public void Roll()
    {
       // ......
    }
}
```

or in a constructor?:

```
public class Dice
{
    private int topFace;
    private Random myRand;

    public Dice()
    {
        topFace = 1;
        myRand = new Random();
    }

    public void Roll()
    {
        // .....
    }
}
```

I'm really curious what some of you veterans think is the best practice. I want to be consistent and stick to one approach.

java

Share
Improve this question
Follow

edited Sep 19, 2019 at 13:42
Mihir
95 ● 3 ● 11

asked Aug 23, 2008 at 19:59
mmcdole
92.7k ● 61 ● 188 ● 224

3 Note that for structs you cannot have instance field initializers, so you have no choice but to use the constructor. – yoyo Jul 10, 2014 at 23:45

Since you brought up "best practice": satisfice.com/blog/archives/5164, forbes.com/sites/mikemyatt/2012/08/15/best-practices-arent/… – Stephen C Feb 21, 2020 at 2:27

16 Answers

Sorted by: Highest score (default)

349

My rules:

1. Don't initialize with the default values in declaration (`null`, `false`, `0`, `0.0`…).
2. Prefer initialization in declaration if you don't have a constructor parameter that changes the value of the field.
3. If the value of the field changes because of a constructor parameter put the initialization in the constructors.
4. Be consistent in your practice (the most important rule).

Share
Improve this answer
Follow

edited Dec 16, 2015 at 9:28
poke
387k ● 80 ● 587 ● 627

answered Aug 23, 2008 at 20:04
kokos
43.6k ● 5 ● 37 ● 32

5 I expect that kokos means that you should not initialize members to their default values (0, false, null etc.), since the compiler will do that for you (1.). But if you want to initialize a field to anything other than its default value, you should do it in the declaration (2.). I think that it might be the usage of the word "default" that confuses you. – Ricky Helgesson Oct 11, 2012 at 17:12

106 I disagree with rule 1 - by not specifying a default value (regardless of whether it's initialised by the compiler or not) you are leaving the developer to *guess* or go looking for the documentation on what the default value for that particular language would be. For readability purposes, I would always specify the default value. – James Jan 9, 2013 at 12:50

35 The default value of a type `default(T)` is always the value that has an internal binary representation of `0`. – Olivier Jacot-Descombes Mar 13, 2013 at 19:26

15 Whether or not you like rule 1, it can't be used with readonly fields, which *must* be explicitly initialized by the time the constructor is finished. – yoyo Jul 10, 2014 at 23:39

50 I disagree with those who disagree with rule 1. It's OK to expect others to learn the C# language. Just as we don't comment each `foreach` loop with "this repeats the following for all items in the list", we don't need to constantly restate C#'s default values. We also don't need to pretend that C# has uninitialized semantics. Since the absence of a value has a clear meaning, it's fine to leave it out. If it's ideal to be explicit, you should always use `new` when creating new delegates (as was required in C# 1). But who does that? The language is designed for conscientious coders. – Edward Brey May 14, 2017 at 5:24

155

In C# it doesn't matter. The two code samples you give are utterly equivalent. In the first example the C# compiler (or is it the CLR?) will construct an empty constructor and initialise the variables as if they were in the constructor (there's a slight nuance to this that Jon Skeet explains in the comments below). If there is already a constructor then any initialisation "above" will be moved into the top of it.

In terms of best practice the former is less error prone than the latter as someone could easily add another constructor and forget to chain it.

Share

Improve this answer

Follow

edited Mar 7, 2019 at 14:59

answered Aug 24, 2008 at 16:04

Quibblesome
25.4k ● 10 ● 62 ● 104

7 That is not correct if you choose to initialize the class with GetUninitializedObject. Whatever is in the ctor will not be touched but the field declarations will be run. – Wolf5 Jan 3, 2013 at 9:18

37 Actually it does matter. If the base class constructor calls a virtual method (which is generally a bad idea, but can happen) which is overridden in the derived class, then using instance variable initializers the variable will be initialized when the method is called - whereas using initialization in the constructor, they won't be. (Instance variable initializers are executed *before* the base class constructor is called.) – Jon Skeet Aug 13, 2013 at 8:52

2 Really? I swear I grabbed this info from Richter's CLR via C# (2nd edition I think) and the gist of it was that this was syntactic sugar (I may have read it wrong?) and the CLR just jammed the variables into the constructor. But you're stating that this isn't the case, i.e. the member initialisation can fire before ctor initialisation in the crazed scenario of calling a virtual in base ctor and having an override in the class in question. Did I understand correctly? Did you just find this out? Puzzled as to this up-to-date comment on a 5 year old post (OMG has it been 5 years?). – Quibblesome Aug 13, 2013 at 19:18

@Quibblesome: A child class constructor will contain a chained call to the parent constructor. A language compiler is free to include as much or as little code before that as it likes, provided that the parent constructor is called exactly once on all code paths, and limited use is made of the object under construction prior to that call. One of my annoyances with C# is that while it can generate code which initializes fields, and code which uses constructor parameters, it offers no mechanism for initializing fields based upon constructor parameters. – supercat Feb 20, 2015 at 21:02

1 @Quibblesome, former one will be an issue, if the value is assigned and passed to a WCF method. Which will reset the data to default data type of the model. The best practice is to do the initialisation in constructor(later one). – Pranesh Janarthanan Dec 23, 2019 at 15:50

17

I think there is one caveat. I once committed such an error: Inside of a derived class, I tried to "initialize at declaration" the fields inherited from an abstract base class. The result was that there existed two sets of fields, one is "base" and another is the newly declared ones, and it cost me quite some time to debug.

The lesson: to initialize **inherited** fields, you'd do it inside of the constructor.

Share

Improve this answer

Follow

edited May 11, 2016 at 9:23

answered Nov 10, 2013 at 22:56

xji
8,177 ● 5 ● 46 ● 63

So if you refer the field by `derivedObject.InheritedField`, is it referring to your base one or the derived one? – RayLuo May 3, 2017 at 23:39

1 @RayLuo For this kind of situation, look at the compiler warnings. There would be a warning like: *CS0108 "MyProperty hides inherited member MyProperty. Use the new keyword if hiding was intended."* – JohnB Jul 23, 2022 at 17:44

16

The semantics of C# differs slightly from Java here. In C# assignment in declaration is performed before calling the superclass constructor. In Java it is done immediately after which allows 'this' to be used (particularly useful for anonymous inner classes), and means that the semantics of the two forms really do match.

If you can, make the fields final.

Share Improve this answer Follow

answered Sep 5, 2008 at 20:22
Tom Hawtin - tackline
147k ● 30 ● 221 ● 312

7

Assuming the type in your example, definitely prefer to initialize fields in the constructor. The exceptional cases are:

- Fields in static classes/methods
- Fields typed as static/final/et al

I always think of the field listing at the top of a class as the table of contents (what is contained herein, not how it is used), and the constructor as the introduction. Methods of course are chapters.

Share Improve this answer Follow

answered Aug 23, 2008 at 21:56
Noel
2,091 ● 5 ● 31 ● 47

2 Why is it "definitely" so? You provided only a style preference, without explaning its reasoning. Oh wait, never mind, according to @quibblesome 's answer, they are "utterly equivalent", so it is really just a personal style preference. – RayLuo May 3, 2017 at 23:45

7

In Java, an initializer with the declaration means the field is always initialized the same way, regardless of which constructor is used (if you have more than one) or the parameters of your constructors (if they have arguments), although a constructor might subsequently change the value (if it is not final). So using an initializer with a declaration suggests to a reader that the initialized value is the value that the field has *in all cases*, regardless of which constructor is used and regardless of the parameters passed to any constructor. Therefore use an initializer with the declaration only if, and always if, the value for all constructed objects is the same.

Share

Improve this answer

Follow

edited Jan 17, 2016 at 20:13

answered Jan 17, 2016 at 20:01

Raedwald

48.5k ● 47 ● 159 ● 245

6

There are many and various situations.

I just need an empty list

The situation is clear. I just need to prepare my list and prevent an exception from being thrown when someone adds an item to the list.

```
public class CsvFile
{
    private List<CsvRow> lines = new List<CsvRow>();

    public CsvFile()
    {
    }
}
```

I know the values

I exactly know what values I want to have by default or I need to use some other logic.

```
public class AdminTeam
{
    private List<string> usernames;

    public AdminTeam()
    {
         usernames = new List<string>() {"usernameA", "usernameB"};
    }
}
```

or

```
public class AdminTeam
{
    private List<string> usernames;

    public AdminTeam()
    {
         usernames = GetDefaultUsers(2);
    }
}
```

Empty list with possible values

Sometimes I expect an empty list by default with a possibility of adding values through another constructor.

```
public class AdminTeam
{
    private List<string> usernames = new List<string>();

    public AdminTeam()
    {
    }

    public AdminTeam(List<string> admins)
    {
         admins.ForEach(x => usernames.Add(x));
    }
}
```

Share

Improve this answer

Follow

edited Sep 18, 2016 at 18:42

Robert Columbia
6,390 ● 15 ● 33 ● 41

answered Aug 14, 2015 at 13:26

Miroslav Holec
3,217 ● 26 ● 24

4

What if I told you, it depends?

I in general initialize everything and do it in a consistent way. Yes it's overly explicit but it's also a little easier to maintain.

If we are worried about performance, well then I initialize only what has to be done and place it in the areas it gives the most bang for the buck.

In a real time system, I question if I even need the variable or constant at all.

And in C++ I often do next to no initialization in either place and move it into an Init() function. Why? Well, in C++ if you're initializing something that can throw an exception during object construction you open yourself to memory leaks.

Share

Improve this answer

Follow

edited Apr 10, 2012 at 19:46

almic
17 ● 6

answered Aug 23, 2008 at 22:09

Dan Blair
2,381 ● 3 ● 21 ● 32

3

The design of C# suggests that inline initialization is preferred, or it wouldn't be in the language. Any time you can avoid a cross-reference between different places in the code, you're generally better off.

There is also the matter of consistency with static field initialization, which needs to be inline for best performance. The Framework Design Guidelines for Constructor Design say this:

> ✓ CONSIDER initializing static fields inline rather than explicitly using static constructors, because the runtime is able to optimize the performance of types that don’t have an explicitly defined static constructor.

"Consider" in this context means to do so unless there's a good reason not to. In the case of static initializer fields, a good reason would be if initialization is too complex to be coded inline.

Share Improve this answer Follow

answered May 14, 2017 at 5:46

Edward Brey

41.6k ● 21 ● 209 ● 265

2

Being consistent is important, but this is the question to ask yourself: "Do I have a constructor for anything else?"

Typically, I am creating models for data transfers that the class itself does nothing except work as housing for variables.

In these scenarios, I usually don't have any methods or constructors. It would feel silly to me to create a constructor for the exclusive purpose of initializing my lists, especially since I can initialize them in-line with the declaration.

So as many others have said, it depends on your usage. Keep it simple, and don't make anything extra that you don't have to.

Share Improve this answer Follow

answered May 24, 2017 at 20:27

MeanJerry

79 ● 7

2

Consider the situation where you have more than one constructor. Will the initialization be different for the different constructors? If they will be the same, then why repeat for each constructor? This is in line with kokos statement, but may not be related to parameters. Let's say, for example, you want to keep a flag which shows how the object was created. Then that flag would be initialized differently for different constructors regardless of the constructor parameters. On the other hand, if you repeat the same initialization for each constructor you leave the possibility that you (unintentionally) change the initialization parameter in some of the constructors but not in others. So, the basic concept here is that common code should have a common location and not be potentially repeated in different locations. So I would say always put it in the declaration until you have a specific situation where that no longer works for you.

Share Improve this answer Follow

answered Oct 20, 2017 at 21:05
Joe Programmer
21 ● 2

1

There is a slight performance benefit to setting the value in the declaration. If you set it in the constructor it is actually being set twice (first to the default value, then reset in the ctor).

Share Improve this answer Follow

answered Aug 29, 2008 at 15:47
John Meagher
24.6k ● 14 ● 56 ● 57

2 In C#, fields are always set the default value first. The presence of an initializer makes no difference. – Jeffrey L Whitledge May 10, 2010 at 18:20

1

When you don't need some logic or error handling:

- Initialize class fields at declaration

When you need some logic or error handling:

- Initialize class fields in constructor

> This works well when the initialization value is available and the initialization can be put on one line. However, this form of initialization has limitations because of its simplicity. If initialization requires some logic (for example, error handling or a for loop to fill a complex array), simple assignment is inadequate. Instance variables can be initialized in constructors, where error handling or other logic can be used.

From https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html .

Share Improve this answer Follow

answered Apr 20, 2020 at 10:23
Yousha Aleayoub
5,604 ● 4 ● 58 ● 71

0

I normally try the constructor to do nothing but getting the dependencies and initializing the related instance members with them. This will make you life easier if you want to unit test your classes.

If the value you are going to assign to an instance variable does not get influenced by any of the parameters you are going to pass to you constructor then assign it at declaration time.

Share Improve this answer Follow

answered Aug 24, 2008 at 15:59
Iker Jimenez
7,235 ● 9 ● 50 ● 48

0

Not a direct answer to your question about **the best practice** but an important and related refresher point is that in the case of a generic class definition, either leave it on compiler to initialize with default values or we have to use a special method to initialize fields to their default values (if that is absolute necessary for code readability).

```
class MyGeneric<T>
{
    T data;
    //T data = ""; // <-- ERROR
    //T data = 0; // <-- ERROR
    //T data = null; // <-- ERROR

    public MyGeneric()
    {
        // All of the above errors would be errors here in constructor as well
    }
}
```

And the special method to initialize a generic field to its default value is the following:

```
class MyGeneric<T>
{
    T data = default(T);

    public MyGeneric()
    {
        // The same method can be used here in constructor
    }
}
```

Share Improve this answer Follow

answered Jul 8, 2018 at 12:32
user1451111
1,943 ● 4 ● 22 ● 33

0

"Prefer initialization in declaration", seems like a good general practice.

Here is an example which cannot be initialized in the declaration so it has to be done in the constructor. "Error CS0236 A field initializer cannot reference the non-static field, method, or property"

```
class UserViewModel
{
    // Cannot be set here
    public ICommand UpdateCommad { get; private set; }

    public UserViewModel()
    {
        UpdateCommad = new GenericCommand(Update_Method); // <== THIS WORKS
    }

    void Update_Method(object? parameter)
    {
    }
}
```

Share Improve this answer Follow

answered Dec 13, 2022 at 14:38

Indy Singh
37 ● 1 ● 6