Alternative to Singleton in Objective-C for better application design

Asked 13 years, 7 months ago Modified 10 years, 7 months ago Viewed 5k times

Part of Mobile Development Collective

17

It seems a lot of Objective-C code is using Singleton nowadays.

While a lot of people complaining about Singleton, e.g. Google (Where Have All the Singletons Gone?), their fellow engineers also use it anyway: http://code.google.com/mobile/analytics/docs/iphone/

I know we had some answers in Stack Overflow already but they are not totally specific to Objective-C as a dynamic language: Objective C has categories, while many other languages do not.

So what is your opinion? Do you still use Singleton? If so, how do you make your app more testable?

Updated: I think we need to use codes as example for more concrete discussion, so much discussions on SO are theory based without a single line of code

Let's use the Google Analytics iOS SDK as an example:

```
// Initialization
[[GANTracker sharedTracker] startTrackerWithAccountID:@"UA-0000000-1"
                                       dispatchPeriod:kGANDispatchPeriodSec
                                             delegate:nil];
// Track page view
[[GANTracker sharedTracker] trackPageview:@"/app_entry_point"
                                withError:&error];
```

The beauty of the above code is once you have initialized using the method "**startTrackerWithAccountID**", you can run method "**trackPageview**" throughout out your apps without passing through configurations.

If you think Singleton is bad, can you improve the above code?

Much thanked for your input, have a happy Friday.

MD iphone objective-c ios

Share

edited May 6, 2011 at 15:32

asked May 6, 2011 at 14:02

Improve this question

Follow

Howard

19.8k • 36 • 115 • 187

This should be moved to programmers.stackexchange.com – coneybeare May 6, 2011 at 14:31

you can improve on your example in several ways, depending on your goals. You could make it a plain global instead of a singleton, and you'd still be able to do the same things, but you would *also* get more flexible and more testable code. Or you could, at the cost of requiring more typing, pass the otherwise global data as parameters, which leads to conceptually cleaner and better structured code, but more typing and more overhead in writing the code. – Stack Overflow is garbage May 6, 2011 at 15:41

I don't see anything wrong with that Singleton code. You're not pulling data from it, so it's not getting used as global storage. I'd be worried if you were sticking view controllers or domain objects in there. – Terry Wilcox May 6, 2011 at 15:58

4 Answers

Sorted by: Highest score (default)

34

This post is likely to be downvote-bait, but I don't really understand why singletons get no love. They're perfectly valid, you just have to understand what they're useful for.

In iOS development, you have one and only one instance of the application you currently are. You're only one application, right? You're not two or zero applications, are you? So the framework provides you with a UIApplication singleton through which to get at application-level os and framework features. It models something appropriately to have that be a singleton.

If you've got data fields of which there can and should be only one, and you need to get to them from all over the place in your app, there's totally nothing wrong with modeling that as a singleton too. Creating a singleton as a globals bucket is probably a misuse of the pattern, and I think that's probably what most people object to about them. But if you're modeling something that has "singleness" to it, a singleton might well be the way to go.

Some developers seem to have a fundamental disgust for singletons, but when actually asked why, they mumble something about globals and namespaces and aesthetics. Which I guess I can understand, if you've really resolved once and for all that Singletons are an anti-pattern and to be abhorred in all cases. But you're not thinking anymore, at that point. And the framework design disagrees with you.

Share Improve this answer Follow

answered May 6, 2011 at 14:21

Dan Ray

21.9k • 7 • 64 • 88

1 +1 and I'm a guy who recommends against Singletons 90% of the time. – Terry Wilcox May 6, 2011 at 15:10

+1 - Your answer is very balanced and also quite detailed. However I think it is a good approach to tell new programmers not to rely on singletons. If they think they need one, it has to be justified. So your answer is perfectly valid, for me the short advice is still "try to avoid singletons, unless..." – GorillaPatch May 6, 2011 at 15:18

1 @Dan yes, you have only one application, but why do you need to model your application in the application? The process is also running on one, and only one CPU, in one, and only one country, on one, and only one planet. So f'ing what? Why does your application need an "application instance"? And what benefit do you gain by making that instance a singleton? The best you can say is that "there's no harm in it"... other than the additional complexity a singleton brings. But it doesn't give you anything positive to make up for it – Stack Overflow is garbage May 6, 2011 at 15:37

1 @Dan: and I say why? As I said in my blog post, a singleton ensures that 1) exactly one instance exists at any time, throughout the lifetime of the application. And yes, that makes logical sense when you run the app, but it's not necessary to *enforce*. And it makes no sense when testing, where you'd want to mock it out, replacing it with several short-lived mock objects. That's virtually impossible with a singleton. And 2) it means that *every* part of your application can access the application instance. Usually, you want to minimize the contact surface agaionst external APIs – Stack Overflow is garbage May 6, 2011 at 18:40

2 re. #1, people often get it wrong by thinking that "because I only need one instance, it would be beneficial to make it *impossible* to create two instances". That just doesn't logically follow. If you only need one instance, you *create* only one instance. That way, once you need two instances (for testing, for example), you haven't shot yourself in the foot – Stack Overflow is garbage May 6, 2011 at 18:41

33

I think most developers go through the Singleton phase, where you have everything you need at your fingertips, in a bunch of wonderful Singletons.

Then you discover that unit testing with Singletons can be difficult. You don't actually want to connect to the database, but your Singleton does. Add a layer of redirection and mock it.

Then you discover that unit testing isn't the only time you need different behaviour. You make your Singleton configurable to have different behaviour based on a parameter. You start to wonder if you need to split it into two Singletons. Then your code needs to know which Singleton to use, so you need a Singleton that knows which Singleton to use.

Then some other code starts messing with the values in your Singleton, while you're using it. How dare they! If you wanted just anybody to get at those values from anywhere, you'd make them global...

Once you get to this point, you start wondering if Singletons were the right solution. You start to see the dangers of global data, particularly within an OO design, where you just assume your data won't get poked at by other people.

So you go back and start passing the data along, rather than looking it up (this used to be called good OO design, but now it has a fancy name like "Dependency Injection").

Eventually you learn that Singletons are fine in moderation. You learn to recognize when your Singleton needs to stop being single.

So you get shared objects like UIApplication and NSUserDefaults. Those are good uses of Singletons.

I got burned enough in the Java Singleton craze a decade ago. I don't even consider writing my own Singletons. The only time I've needed anything similar in recent memory is wanting to cache the result of [NSCalendar currentCalendar] (which takes a long time). I created a category on NSCalendar and cached it as a static variable. I felt a bit dirty, but the alternative was painfully slow code.

To summarize and for those who tl;dr:

Singletons are a tool. They're not likely to be the right tool, but you have to discover that for yourself.

answered May 6, 2011 at 15:44

Terry Wilcox
9,040 ● 1 ● 36 ● 37

1 Good answer. The downfall of singletons in Objective-C is definitely unit testing. Each class needs to know how to instantiate another -- for example, calling `[[Singleton sharedSingleton] doSomething]`. That means that you can't readily replace your singleton with a mock object for unit tests. – asinesio May 6, 2011 at 15:50

Great answer. An off-topic note on the NSCalendar trick, make sure to spawn new calendars on background threads, as NSCalendar is not thread safe. – epologee Jun 8, 2012 at 7:23

It's so true. I remember when I was young and bulletproof with my several singletons per project. I can't exactly say what happened, but I can say I haven't used one in years... – Paul de Lange Jan 15, 2013 at 9:44

Very good answer, would like to add that the problem of singletons is not only unit testing(in ObjC we could use method swizzling replacing the sharedSingleton method for one returning a mock), but they make dependencies hidden and this is a big drawback comparing to use DI where the dependencies are visible and you can notice with ease when a class has too many. – e1985 Mar 12, 2014 at 12:28

3

Why do you need an answer that is "total Objective C specific"? Singletons aren't totally Obj-C specific either, and you're able to use those. Functions aren't Obj-C-specific, integers aren't Obj-C specific, and yet you're able to use all of those in your Obj-C code.

The obvious replacements for a singleton work in any language.

A singleton is a badly-designed global.

So the simplest replacement is to just make it a regular global, without the silly "one instance only" restriction.

A more thorough solution is, instead of having a globally accessible object at all, pass it as a parameter to the functions that need it.

And finally, you can go for a hybrid solution using a Dependency Injection framework.

Share Improve this answer Follow

answered May 6, 2011 at 14:10

Stack Overflow is garbage

247k ● 52 ● 353 ● 555

What are some dependency injection frameworks for Objective-C besides the Apple-provided ones? I've yet to see anything that compares to Spring (from Java) in the Objective-C world. – asinesio May 6, 2011 at 15:46

no clue. I'm not a huge fan of DI frameworks. But they're a popular option, so I thought I'd mention them. :) – Stack Overflow is garbage May 6, 2011 at 17:06

@asinesio Take a look at typhoonframework.org, a DI framework for Objc. (we also start by explaining how to do DI without a library). – Jasper Blues May 26, 2014 at 0:35

1

The problem with singletons is that they ***can*** lead to tight coupling. Let's say you're building an airline booking system: your booking controller might use an

```
id<FlightsClient>
```

A common way to obtain it within the controller would be as follows:

```
_flightsClient = [FlightsClient sharedInstance];
```

Drawbacks:

- It becomes difficult to test a class in isolation.

- If you want to change the flight client for another implementation, its necessary to search through the application and swap it out one by one.
- If there's a case where the application should use a different implementation (eg OnlineFlightClient, OfflineFlightClient), things get tricky.

A good workaround is to apply the dependency injection design pattern.

Think of dependency injectionas ***telling an architectural story***. When the key ***actors*** in your application are pulled up into an assembly, then the application's configuration is correctly modularized (removing duplication). Having created this script, its now easy to reconfigure or swap one actor for another.". In this way we need not understand all of a problem at once, its easy to evolve our app's design as the requirements evolve.

Here's a dependency injection library: https://github.com/typhoon-framework/Typhoon

edited May 26, 2014 at 0:44

answered Jan 15, 2013 at 8:10

Jasper Blues
28.7k ● 22 ● 109 ● 190