Dealing with "global" data structures in an object-oriented world

Asked 16 years, 2 months ago Modified 16 years, 1 month ago

Viewed 5k times

15

This is a question with many answers - I am interested in knowing what others consider to be "best practice".

Consider the following situation: you have an object-oriented program that contains one or more data structures that are needed by many different classes. How do you make these data structures accessible?

1. You can explicitly pass references around, for example, in the constructors. This is the "proper" solution, but it means duplicating parameters and instance variables all over the program. This makes changes or additions to the global data difficult.
2. You can put all of the data structures inside of a single object, and pass around references to this object. This can either be an object created just for this purpose, or it could be the "main" object of your program. This simplifies the problems of (1), but the data structures may or may not have anything to do with one another, and collecting them together in a single object is pretty arbitrary.

3. You can make the data structures "static". This lets you reference them directly from other classes, without having to pass around references. This entirely avoids the disadvantages of (1), but is clearly not OO. This also means that there can only ever be a single instance of the program.

When there are a lot of data structures, all required by a lot of classes, I tend to use (2). This is a compromise between OO-purity and practicality. What do other folks do? (For what it's worth, I mostly come from the Java world, but this discussion is applicable to any OO language.)

oop

Share

Improve this question

Follow

edited Oct 1, 2008 at 8:59

asked Oct 1, 2008 at 8:29

Brad Richards

1,253 ● 3 ● 13 ● 23

9 Answers

Sorted by: Highest score (default)

9

Global data isn't as bad as many OO purists claim!

After all, when implementing OO classes you've usually using an API to your OS. What the heck is this if it isn't a huge pile of global data and services!

If you use some global stuff in your program, you're merely extending this huge environment your class implementation can already see of the OS with a bit of data that is domain specific to your app.

Passing pointers/references everywhere is often taught in OO courses and books, academically it sounds nice. Pragmatically, it is often the thing to do, but it is misguided to follow this rule blindly and absolutely. For a decent sized program, you can end up with a pile of references being passed all over the place and it can result in completely unnecessary drudgery work.

Globally accessible services/data providers (abstracted away behind a nice interface obviously) are pretty much a must in a decent sized app.

Share Improve this answer Follow

answered Oct 1, 2008 at 11:01

Scott Langham
60.3k ● 42 ● 136 ● 210

4

I must really really discourage you from using option 3 - making the data static. I've worked on several projects where the early developers made some core data static, only to later realise they did need to run two copies of the program - and incurred a huge amount of work making the data non-static and carefully putting in references into everything.

So in my experience, if you do 3), you will eventually end up doing 1) at twice the cost.

Go for 1, and be fine-grained about what data structures you reference from each object. Don't use "context objects", just pass in precisely the data needed. Yes, it makes the code more complicated, but on the plus side, it makes it clearer - the fact that a `FwurzleDigestionListener` is holding a reference to both a `Fwurzle` and a `DigestionTract` immediately gives the reader an idea about its purpose.

And by definition, if the data format changes, so will the classes that operate on it, so you have to change them anyway.

Share Improve this answer Follow

answered Oct 1, 2008 at 9:57

Zarkonnen
22.4k ● 14 ● 68 ● 81

1) is called *Dependency injection* and there are many tools that make it very easy (or at least less cumbersome)
– OGrandeDiEnne Mar 22, 2016 at 10:14

You might want to think about altering the requirement that lots of objects need to know about the same data

2

structures. One reason there does not seem to be a clean OO way of sharing data is that sharing data is not very object-oriented.

You will need to look at the specifics of your application but the general idea is to have one object responsible for the shared data which provides services to the other objects based on the data encapsulated in it. However these services should *not* involve giving other objects the data structures - merely giving other objects the pieces of information they need to meet *their* responsibilites and performing mutations on the data structures internally.

Share Improve this answer Follow

answered Oct 1, 2008 at 10:11

domgblackwell
5,090 ● 1 ● 21 ● 11

1

I tend to use 3) and be very careful about the synchronisation and locking across threads. I agree it is less OO, but then you confess to having global data, which is very un-OO in the first place.

Don't get too hung up on whether you are sticking purely to one programming methodology or another, find a solution which fits your problem. I think there are perfectly valid contexts for singletons (Logging for instance).

Share Improve this answer Follow

answered Oct 1, 2008 at 8:50

Simon
80.6k ● 26 ● 92 ● 119

1

I use a combination of having one global object and passing interfaces in via constructors.

From the one main global object (usually named after what your program is called or does) you can start up other globals (maybe that have their own threads). This lets you control the setting up of program objects in the main objects constructor and tearing them down again in the right order when the application stops in this main objects destructor. Using static classes directly makes it tricky to initialize/uninitialize any resources these classes use in a controlled manner. This main global object also has properties for getting at the interfaces of different sub-systems of your application that various objects may want to get hold of to do their work.

I also pass references to relevant data-structures into constructors of some objects where I feel it is useful to isolate those objects from the rest of the world within the program when they only need to be concerned with a small part of it.

Whether an object grabs the global object and navigates its properties to get the interfaces it wants or gets passed the interfaces it uses via its constructor is a matter of taste and intuition. Any object you're implementing that you think might be reused in some other project should definately be passed data structures it should use via its constructor. Objects that grab the global object should be more to do with the infrastructure of your application.

Objects that receive interfaces they use via the constructor are probably easier to unit-test because you can feed them a mock interface, and tickle their methods to make sure they return the right arguments or interact with mock interfaces correctly. To test objects that access the main global object, you have to mock up the main global object so that when they request interfaces (I often call these services) from it they get appropriate mock objects and can be tested against them.

Share Improve this answer Follow

edited Oct 1, 2008 at 10:03

answered Oct 1, 2008 at 9:29

Scott Langham

60.3k ● 42 ● 136 ● 210

1

I prefer using the singleton pattern as described in the GoF book for these situations. A singleton is not the same as either of the three options described in the question. The constructor is private (or protected) so that it cannot be used just anywhere. You use a get() function (or whatever you prefer to call it) to obtain an instance. However, the architecture of the singleton class guarantees that each call to get() returns the same instance.

Share Improve this answer Follow

answered Oct 2, 2008 at 18:21

mxg
1,357 ●1 ●12 ●15

1

We should take care not to confuse Object Oriented ***Design*** with Object Oriented ***Implementation***. Al too often, the term OO Design is used to judge an implementation, just as, imho, it is here.

Design

If in your design you see a lot of objects having a reference to exactly the same object, that means a lot of arrows. The designer should feel an itch here. He should verify whether this object is just commonly used, or if it is really a utility (e.g. a COM factory, a registry of some kind, ...).

From the project's requirements, he can see if it really needs to be a singleton (e.g. 'The Internet'), or if the object is shared because it's too general or too expensive or whatsoever.

Implementation

When you are asked to implement an OO Design in an OO language, you face a lot of decisions, like the one you mentioned: how should I implement *all the arrows* to the oft used object in the design?

That's the point where questions are addressed about 'static member', 'global variable' , 'god class' and 'a-lot-of-function-arguments'.

The Design phase should have clarified if the object needs to be a singleton or not. The implementation phase will decide on how this singleness will be represented in the program.

Share Improve this answer Follow

answered Nov 18, 2008 at 14:59

xtofl

41.5k ● 12 ● 109 ● 201

0

Option 3) while not purist OO, tends to be the most reasonable solution. But I would not make your class a singleton; and use some other object as a static 'dictionary' to manage those shared resources.

Share Improve this answer Follow

answered Oct 1, 2008 at 13:17

Patrick

-2

I don't like any of your proposed solutions:

1. You are passing around a bunch of "context" objects - the things that use them don't specify what fields or pieces of data they are really interested in
2. See here for a description of the God Object pattern. This is the worst of all worlds
3. Simply do not ever use Singleton objects for anything. You seem to have identified a few of the potential problems yourself

Share Improve this answer Follow

answered Oct 1, 2008 at 8:34

1800 INFORMATION
135k ● 30 ● 163 ● 242

Ah, but what solution would you then follow? You have 20 classes that all manipulate the same data structure or object. How do you give the 20 classes access? Properly, you have to pass it to them in a constructor or with a method (1). But this results in a lot of duplicate code. What do you do? – Brad Richards Oct 1, 2008 at 8:37

Your class informs the world by means of its constructor parameters what data it needs - the thing that creates it then passes that data to it – 1800 INFORMATION Oct 1, 2008 at 8:44

Is it worth passing the objects in its constructor for all the classes? for eg if you have 20 classes to manipulate the same data structure, do you prefer passing the same to each and every class? what will happen if you come across a change in the datastructure? – TG. Oct 1, 2008 at 8:49

Don't pass around context objects - your whole program becomes dependant on the details of those objects. Follow Demeter's law - this tells us that the constructors should not go looking in a context object for the data that they need – 1800 INFORMATION Oct 1, 2008 at 20:29