

Re: Criteria to decide what is private? (was: Encapsulation vs Extensibility)

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2007-01/msg00160.html>

- *From:* "Mark Nicholls" <Nicholls.Mark@xxxxxxxxxx>
 - *Date:* 17 Jan 2007 03:56:54 -0800
-

Mark Nicholls wrote:

Daniel T. wrote:

Responding to Mark Nicholls:

This conversation seems to be going all over the place, so I'm guessing I have not explained myself well. Let me start again.

I think my brains being going all over the place a bit.

When you are implementing a class, you know the invariant, and you know which fields, if made public/protected, would allow outsiders to break that invariant. You know which methods have contracts that have post-conditions which conflict with the invariant. If you don't know these things, you have no business implementing the class.

yes OK..

The things that would allow outsiders to break your invariant should be made private, for the rest, it doesn't matter what access level you give it, so you might as well make it public.

hmmm....still not really convinced by that as an engineering rule of thumb.....I have had a minor penny drop, by my instinct is, your design a class to implement an interface, you do not expose stuff that is incidental to that interface because it may be useful, from a versioning/subtyping perspective you are making your life more difficult than it needs to be as all subtypes/version of this code must

Re: Criteria to decide what is private? (was: Encapsulation vs Extensibility)

also implement these things....yet they seem to be incidental to the purpose of the class....so make them private.

basically it's one of Meyers rules of OO design....small interfaces....OOSC2..

I don't see any difference between the principle applying to a protected interface as to a public one (as protected to equivalent to a public delegate interface).

Whether the various implementations you provide for these features are error free is irrelevant to the theoretical decision to make the method/field public or private.

Maybe we'll clarify with an example.

```
interface Vector
{
VectorAdd(Vector);
}
```

```
interface Vector
{
void SetX(int X);
void SetY(int Y);
int GetX();
int SetY();
VectorAdd(Vector);
}
```

I agree the 'science' tells me it 'does not matter' if I expose SetX,SetY etc

my original specification tells me that I need to have a vector class that supports adding of vectors.....I may as well expose the underlying implementation as it does not matter?

now I want to optimise my system to work on a system that requires polar coords....so I want to redevelop my implementation of Vector.....now *I do not know* if any clients have been written that access SetX,SetY, GetX,GetY.....I know they did not need to do this, because they could create any vector they liked through construction.....but now I have to implement polar to cartesian conversion just in case.....more complexity => more bugs & more cost.

Re: Criteria to decide what is private? (was: Encapsulation vs Extensibility)

We all write bugs on occasion, but we can't assume a priori that every method we write, no matter how thoroughly tested, is going to fail.

correct...but unnecessary coupling and complexity give rise to unnecessary code and unnecessary bugs and unnecessary cost.....a laissez faire approach allows unnecessary coupling.

Even given the above, I feel it is important to satisfy the UAP so in languages like C++ and Java, I make all fields private and implement getters and setters.

UAP?

why? surely the science tells you it doesn't matter?

(sometimes I make them public to minimize complexity!)