Oferiti un punct de vedere argumentat privind urmatoarea discutie:

Inheritance breaking encapsulation?

Question:

People say inheritance breaks encapsulation, which i agree with. They say delegation is better- although the modifiers in delegation can also be public/protected.

So is the real reason why inheritance breaks encapsulation because of the "knock-on" effect of the public/protected modifiers from the super class being exposed to any new classes which extend the current subclass?

Answer:

Yes. Since it gives the derived class access to members of the base class (depending on what language and which kind of inheritance) it is said that it breaks encapsulation. IMHO this is only if you are clinging to encapsulation in the strictest terms. IMHO it is reasonable to say that you are accepting the derived class as an extension of the base and therefore related in some way and not really breaking the encapsulation. Purists will disagree with this.

Take a look at http://www.ccs.neu.edu/research/demeter/papers/context-journal/node17.html and search for "breaks" for an academic explanation.

Comments:

- Hence why you should only use inheritance when the subclass IS A superclass for the WHOLE of it's lifetime? If this is satisfied then the subclass should have access to ALL of the super class's attributes/methods? – Jon Feb 18 '12 at 23:05
- Inheritance does not break encapsulation, even in a purist sense. If those members were to be encapsulated and not reachable from derived classes, they wouldn't have been public/protected. – Mooing Duck Feb 18 '12 at 23:06
- @MooingDuck Take a look at the link in my answer. It is widely acknowledged that inheritance breaks encapsulation in the purist sense. – Sid Feb 18 '12 at 23:10
- @Jon Well, in general, I think inheritance should be used when there is an "IS A" relationship. Sure you can use delegation but then you don't inherit the methods of the base. If you want the same interface you'd have to write the same methods in the contained class again. – Sid Feb 18 '12 at 23:12
- “If a derived class is allowed to access members inherited from a base class, changes in the base class may require maintenance of the derived class as well.” Isn’t that applicable to *any* consumer of any class if implementation changes are allowed to leak through the interface? I find the arguments in the cited source weak. – Douglas Feb 18 '12 at 23:14
- @Douglas The simple reason is that the derived class can manipulate the base class's data directly and that breaks encapsulation. – Sid Feb 18 '12 at 23:20
- Derived classes can manipulate the base class’s data directly only when they’re allowed to do so (through the `protected` access modifier). If you’re mislabelling private fields as protected, then it’s your design that’s broken. – Douglas Feb 18 '12 at 23:24
- @Douglas The academic argument for "Inheritance breaking encapsulation" is about the ability to have "protected" members AT ALL. The mere idea of being able to do so, even if intentional, is the reason for purists of OOP saying inheritance breaks encapsulation. – Sid Feb 18 '12 at 23:26
- When you declare a member as protected, you are effectively committing to expose it as part of the class’s interface – in this case, to a restricted set of consumers (namely, derived classes). The onus is still on the class designer to ensure that protected members do not leak the internal implementation of the class any more than public members do. Yes, misuse of the protected keyword often breaks encapsulation (as acknowledged by the Gang of Four). Proper use of it does not. – Douglas Feb 18 '12 at 23:33
- "restricted set of consumers"... which the designer of the superclass has no control over which consumers? – Jon Feb 18 '12 at 23:44