DesignedInheritance

Martin Fowler *6 October 2006* *tags: encapsulation · API design*

One of the longest running arguments on object-oriented circles is the debate between OpenInheritance and Designed Inheritance. The principle of Designed Inheritance is probably best summed up by Josh Bloch: "Design and document for inheritance or else prohibit it". With this approach you take care to decide which methods can be inherited and Seal the others to stop them being overridden.

The most common argument for designed inheritance is that since inheritance is a very intimate (and encapsulation breaking) relationship, it's easy for subclasses to effectively break their superclasses by, for instance, omitting necessary behavior when they the methods are called.

Many developers, particularly those with a EnablingAttitude find this style of argument unconvincing. Another argument that I've found more appealing is that of Elliote Rusty Harold while discussing the design principles of XOM. The point here is that "APIs are written by experts for non experts". The library writer should be well versed in the technology that the library works with. She should work to simplify this technology for library users. Encapsulation is all about hiding secrets, so a good library should hide all sorts of complications and danger points from library users, whether they use that library though calling or inheritance. So with XOM I can safely override the library classes to do what I want yet the library guarantees that it will still produce well-formed XML, without me having to worry my ugly little head about all the things that might otherwise go wrong.

Such an argument is much more convincing to me that the usual one which carries the subtext that library writers are smart and users stupid. This isn't about ability, but about detailed knowledge. I want to be as ignorant as I can be the sordid details of XML. By relieving me of needing to understand these kinds of details, I can use my brain power the actual task I want to accomplish.

Despite this persuasive argument my instinct, and the fact that I have an enabling attitude, tends to prefer open inheritance. Perhaps the crux of the problem is the mechanism we use to signal safe areas of inheritance. Usually all we have is the ability to seal classes and methods. An alternative to placate both camps would be to have an ability to overcome a seal. That way you have to go out of your way to override something that wasn't designed. If you don't explicitly open the seal, then the compiler only allows normal inheritance, but if you use the seal-opening mechanism then the compiler will hand over the trust to you - and you are responsible for the consequences. So I'd prefer to replace Josh Bloch's 'prohibit' with a 'discourage'.

Taking a broader view my guess is there's still a lot of room for improvement in how we think about interfaces, both for calling and inheritance. Ideas like Consumer Driven Contracts are needed to help us rethink what it means to have an interface definition. I don't know what the answer would be, but I think a good answer would surprise all of us.

tags

API design · academia · agile · agile adoption · agile history · analysis patterns · application architecture · application integration · bad things · big data · build scripting · certification · clean code · collaboration · conferences · continuous integration · data analytics · database · delivery · dictionary · diversions · diversity · documentation · domain driven design · domain specific language · domestic · encapsulation · evolutionary design · extreme programming · gadgets · internet culture · language feature · lean · legacy rehab · metrics · microsoft · mobile · noSQL · object collaboration design · parser generators · photography · presentations · process theory · productivity · programming platforms · project planning · projects · recruiting · refactoring · refactoring boundary · requirements analysis · retrospective · ruby · scrum · software craftsmanship · team environment · team organization · technical debt · technical leadership · testing · thoughtworks · tools · travel · uml · version control · web · website · writing

All Content

translations

Japanese · Spanish · Korean · Chinese · Thai

blogroll

ThoughtBlogs

TW Alumni · Tim Anderson · Kent Beck · Tim Bray · Paul Duvall · Elizabeth Hendrickson · Jeremy Miller · Ralph Johnson · Dan Pritchett · xkcd

Guides

- Intro
- Design
- Agile
- NoSQL
- DSL
- Delivery
- About
- Me

Popular Articles

- New Methodology
- Dependency Injection
- Mocks aren't Stubs
- Is Design Dead?
- Continuous Integration

Books

- NoSQL Distilled
- Domain-Specific Languages
- Refactoring
- Patterns of Enterprise Application Architecture
- UML Distilled
- Analysis Patterns
- Planning Extreme Programming
- Signature Series

Site Sections

- FAQ
- Content Index
- Bliki
- Books
- DSL Catalog
- EAA Catalog
- EAA Dev
- Photos

ThoughtWorks

- Blogs
- Careers
- Mingle
- Twist
- Go