immutable class should be final?

Asked 16 years, 2 months ago Modified 1 year, 11 months ago

Viewed 9k times

23

It says in this article that:

> Making a class final because it is immutable is a good reason to do so.

I'm a bit puzzled by this... I understand that immutability is a good thing from the POV of thread-safety and simplicity, but it seems that these concerns are somewhat orthogonal to extensibility. So, why is immutability a good reason for making a class final?

java oop final

Share

Improve this question

Follow

edited Apr 14, 2016 at 16:19

mkobit

47.1k • 12 • 160 • 155

asked Sep 28, 2008 at 17:32

Dónal

187k • 176 • 581 • 843

6 Answers

Sorted by: Highest score (default)

13

The explanation for this is given in the book 'Effective Java'

Consider `BigDecimal` and `BigInteger` classes in Java .

It was not widely understood that immutable classes had to be effectively final when `BigInteger` and `BigDecimal` were written, so all of their methods may be overridden. Unfortunately, this could not be corrected after the fact while preserving backward compatibility.

If you write a class whose security depends on the immutability of a BigInteger or BigDecimal argument from an un-trusted client, you must check to see that the argument is a “real” BigInteger or BigDecimal, rather than an instance of an un trusted subclass. If it is the latter, you must defensively copy it under the assumption that it might be mutable.

```
public static BigInteger safeInstance(BigInteger val)
    if (val.getClass() != BigInteger.class)
        return new BigInteger(val.toByteArray());

    return val;
}
```

If you allow sub classing, it might break the "purity" of the immutable object.

Share Improve this answer

Follow

edited Jan 22, 2023 at 7:22

Ryan H.
2,593 ● 1 ● 16 ● 26

answered Sep 26, 2012 at 11:27

Vinoth Kumar C M
10.6k ● 28 ● 90 ● 131

10

Because if the class is final you can't extend it and make it mutable.

Even if you make the fields final, that only means you cannot reassign the reference, it does not mean you cannot change the object that is referred to.

I don't see a lot of use in a design for an immutable class that also should be extended, so final helps keep the immutability intact.

Share Improve this answer

Follow

answered Sep 28, 2008 at 17:36

cynicalman
5,881 ● 3 ● 31 ● 30

3 But you might want to extend an immutable class in order to add additional immutbale properties. – Dónal Sep 28, 2008 at 17:41

2 "Because if the class is final you can't extend it and make it mutable" But you can't extend a non-final immutable class and make it mutable, you can only make the subclass mutable – Dónal Sep 28, 2008 at 17:41

4 As a caller, you don't know whether you're using the immutable class or its mutable subclass - unless you happened to call the constructor directly. That's polymorphism. – slim Sep 28, 2008 at 17:43

10

Following the Liskov Substitution Principle a subclass can extend but never redefine the contract of its parent. If the base class is immutable then its hard to find examples of where its functionality could be usefully extended without breaking the contract.

Note that it is possible in principle to extend an immutable class and change the base fields e.g. if the base class contains a reference to an array the elements within the array cannot be declared final. Obviously the semantics of methods can also be changed via overriding.

I suppose you could declare all the fields as private and all the methods as final, but then what would be the use of inheriting?

Share Improve this answer Follow

edited Sep 28, 2008 at 17:54

answered Sep 28, 2008 at 17:46

Garth Gilmour

11.2k ● 5 ● 28 ● 36

1 Say you have an immutable Shape class, which has an area property. You might want to create a Circle subclass which has additional properties such as radius. These new properties might themselves be immutable, thus preserving the immutability property of the parent. What's wrong with that? – Dónal Sep 28, 2008 at 18:05

1 I don't think there's anything wrong with that, as long as you are the only person extending from Shape. But you would

have to create precise guidelines for other developers and be confident they would follow them. So I think the argument is that 99.9% of the time its not worth the effort...
– Garth Gilmour Sep 28, 2008 at 18:38

GG's on the money. If you make a guarantee you have to keep it. – CurtainDog Feb 15, 2010 at 7:33

@Don Wouldn't `Shape` be an abstract class in that scenario? – crush Aug 15, 2013 at 13:09

6

Mainly security I'd think. For the same reason String is final, anything that any security-related code wants to treat as immutable must be final.

Suppose you have a class defined to be immutable, call it MyUrlClass, but you don't mark it final.

Now, somebody might be tempted to write security manager code like this;

```
void checkUrl(MyUrlClass testurl) throws SecurityExcep
    if (illegalDomains.contains(testurl.getDomain()))
SecurityException();
}
```

And here's what they'd put in their DoRequest(MyUrlClass url) method:

```
securitymanager.checkUrl(urltoconnect);
Socket sckt = opensocket(urltoconnect);
sendrequest(sckt);
getresponse(sckt);
```

But they can't do this, because you didn't make MyUrlClass final. The reason they can't do it is that if they did, code could avoid the security manager restrictions simply by overriding getDomain() to return "www.google.com" the first time it's called, and "www.evilhackers.org" the second, and passing an object of their class into DoRequest().

I have nothing against evilhackers.org, by the way, if it even exists...

In the absence of security concerns it's all about avoiding programming errors, and it is of course up to you how you do that. Subclasses have to keep their parent's contract, and immutability is just a part of the contract. But if instances of a class are supposed to be immutable, then making it final is one good way of making sure they really are all immutable (i.e. that there aren't mutable instances of subclasses kicking around, which can be used anywhere that the parent class is called for).

I don't think the article you referenced should be taken as an instruction that "all immutable classes must be final", especially if you have a positive reason to design your immutable class for inheritance. What it was saying is that protecting immutability is a valid reason for final, where imaginary performance concerns (which is what it's really talking about at that point) are not valid. Note that it gave "a complex class not designed for inheritance" as an equally valid reason. It can fairly be argued that failing to account for inheritance in your complex classes is

something to avoid, just as failing to account for inheritance in your immutable classes is. But if you can't account for it, you can at least signal this fact by preventing it.

Share Improve this answer Follow

edited Sep 28, 2008 at 20:32

answered Sep 28, 2008 at 18:05

Steve Jessop
279k ● 40 ● 469 ● 709

> As I said, "It can fairly be argued that failing to account for inheritance in your complex classes is something to avoid ... But if you can't account for it, you can at least signal this fact by preventing it.". And you are arguing it. Fairly.
> – Steve Jessop Nov 19, 2011 at 18:26

0

Its a good idea to make a class immutable for performance reasons too. Take Integer.valueOf for example. When you call this static method it does not have to return a new Integer instance. It can return a previously created instance safe in the knowledge that when it passed you a reference to that instance last time you didn't modify it (I guess this is also good reasoning from a security reason perspective too).

I agree with the standpoint taken in Effective Java on these matters -that you should either design your classes for extensibility or make them non-extensible. If its your

intention to make something extensible perhaps consider an interface or abstract class.

Also, you don't have to make the class final. You can make the constructors private.

Share Improve this answer

Follow

answered Jan 27, 2010 at 10:54

ishmeister

609 ● 1 ● 4 ● 8

> Just a few notes. 1) Usually, a mutable class instance is better for the performance reasons comparing to an equivalent immutable instance. The immutable instances are usually thread-safe, but immutability costs if you want to get a newer object with even slightly different state (using a Prototype or a Builder pattern). 2) `Integer.valueOf` is an example of a Flightweight pattern example, and the pattern is usually not related to the matter of immutability. 3) Having a private constructor cannot prevent sub-classing using an inner class. – Lyubomyr Shaydariv Jan 13, 2013 at 20:30

0

So, why is immutability a good reason for making a class final?

As stated in oracle docs there are basically 4 steps to make a class immutable.

So one of the point states that

to make a class Immutable class should be marked as either final or have private constructor

Below are the 4 steps to make a class immutable (straight from the oracle docs)

1. Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
2. Make all fields final and private.
3. Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.
4. If the instance fields include references to mutable objects, don't allow those objects to be changed:
    - Don't provide methods that modify the mutable objects.
    - Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

answered May 27, 2018 at 12:21

Mateen

1,671 ● 1 ● 23 ● 28