Should enum objects be stateless?

Asked 15 years, 10 months ago Modified 9 years ago Viewed 4k times

9

As by design an enum constant in java is a singleton, and for sake of concurrent usage I normally create stateless enum instances and use method parameters to inject the data as needed.

Example:

Currently I am creating a REST service which has Operations (implemented as an enum using a variant of the strategy pattern).

```
public enum Operation {

  DO_THIS() {
    public Result doSomething(Object theData) {
    }
  } ,
  // Other Operations go here
  ;

  public abstract Result doSomething(Object theData);

}
```

Now I want to collect data about how often an operation has been called and how often it succeeded and the like.

I could save the state externally when using the enum instance but it rather seems that the state should be saved in the Operation as the operation should contain it's own state.

Now my general question is:

Is a stateful enum instance (besides from concurrency issues) a bad design?

java oop enums

Share Improve this question Follow

asked Feb 12, 2009 at 6:42

Daniel Hiller

3,475 ● 3 ● 25 ● 33

without question, yes. – KRK Owner May 22, 2017 at 23:37

7 Answers

Sorted by: Highest score (default)

19

I think it violates the Principle of Least Astonishment.

People expect the common usage of enums as they were originally designed - as constants or tokens, and not as general purpose classes with state.

Share Improve this answer Follow

answered Feb 12, 2009 at 6:47

mparaz
2,069 ● 3 ● 28 ● 48

In contrast to that enum members are indeed first class java object instances, as such they can have methods and variables. So maybe the whole enum design may be flawed? – Daniel Hiller Feb 12, 2009 at 8:25

I think the reason the Java designers used classes for Enums is because they allow reference uniqueness (in other words, == would work). The ideal scenario would be to use something like "symbols" but then they would need JVM support and are too big of a change. – mparaz Feb 12, 2009 at 8:30

1 @mparaz: no the Java designers made enums like classes to allow them to have behaviour, which is about the only language feature from Java I miss in C#. Enums with behaviour are awesome. – cletus Feb 12, 2009 at 12:40

7

Yes. And by 'yes' I mean 'Always'.

If you want to collate stats on the number of operations called, implement some observability.

Share Improve this answer Follow

answered Feb 12, 2009 at 6:56

cletus
625k ● 169 ● 917 ● 945

I will think about that. At the moment my idea would be to add a StateContainer instance to the method signature... – Daniel Hiller Feb 12, 2009 at 8:28

Hang on. If you add observability to the enum, you've just made it statefull! – Tom Hawtin - tackline Feb 12, 2009 at 13:52

Any form of mutable static is a sin. (Well, you might get away with non-leaky caches, some lazy initialisation and forms of logging.)

4

Share Improve this answer Follow

answered Feb 12, 2009 at 13:55

Tom Hawtin - tackline

147k ● 30 ● 221 ● 312

A stateful enumeration is an oxymoron, even an anti-pattern!

http://en.wikipedia.org/wiki/Enumeration

> An enumeration is a collection of items that is a complete, ordered listing of all of the items in that collection. The term is commonly used in mathematics and theoretical computer science to refer to a listing of all of the elements of a set. In statistics the term categorical variable is used rather than enumeration. The precise requirements for an enumeration (for example, whether the set must be finite, or whether the list is allowed to contain repetitions) depend on the branch of mathematics and the context in which one is working.

4

Enumerations have a finite number of values, which are supposed to be constant, which they are.

However, the fact that they are "first class" Java Objects totally goes against the grain of the intention or spirit of an enumeration.

If any kind of state is required, the enum (as mentioned earlier) should hold state in an Aspect or the offending enum, should at the very practical least, hold a reference to a delegate class holding state. Understanding "separation of concerns" will help.

Share Improve this answer Follow

answered Mar 29, 2014 at 22:56

KRK Owner

782 ● 7 ● 16

This seems like a bad use for enums - why not just go with a base abstract class with a new subclass for each operation?

1

Share Improve this answer Follow

answered Feb 12, 2009 at 6:54

Marc Novakowski
45.4k ● 11 ● 61 ● 63

because the operation as such is fix, the conversion to an operation type and use of a factory seems too much overhead for me... – Daniel Hiller Feb 12, 2009 at 8:27

I think you may be prematurely optimizing - there's nothing wrong with using a factory and in fact will make your code much easier to test. – Marc Novakowski Feb 12, 2009 at 16:44

1

I entirely agree with mparaz that it violates the Principle of Least Astonishment. People expect enums to be constants.

You can almost certainly work round the logging thing, by something like:

```
DO_THIS() {
  public Result doSomething(Object theData) {
    MyUtilClass.doSomething(Object theData);
  }
}
```

and put your logging in the other class.

HOWEVER if you can't work round this, the Principle of Least Astonishment is a guideline; you can violate it PROVIDED you give users of the class enough warnings about what is going on. Make sure the Enum declaration contains a BIG notice saying that it is mutable, and describing exactly what the mutability is. The Enum should still work; it's doing reference comparison against to single instance to test enum values.

Share Improve this answer Follow

answered Feb 12, 2009 at 15:19

DJClayworth
26.8k ● 9 ● 58 ● 80

0

There is a case which would probably justify it. An enum can implement an interface, usually with the particular use case in mind which lets you create on runtime/openly "some other types of the enum class" in a dynamic fashion, to name it someway.

That means that enum "singleton" instances can be forced to implement some mutable-intended method signatures (as setters), which of course, you still can hide with an empty code or a NotSupportedException.

Luckily, final methods in an interface don't allow any possibility to change state. That would have been the sole "understandable" case I could come up with.

Share

Improve this answer

Follow

edited Dec 2, 2015 at 1:50

answered Dec 2, 2015 at 1:44

Whimusical
6,619 ● 12 ● 67 ● 114