Java Singleton vs static - is there a real performance benefit?

Asked 16 years, 4 months ago Modified 12 years, 9 months ago

Viewed 27k times

16

I am merging a CVS branch and one of the larger changes is the replacement wherever it occurs of a Singleton pattern with abstract classes that have a static initialisation block and all static methods.

Is this something that's worth keeping since it will require merging a lot of conflicts, what sort of situation would I be looking at for this refactoring to be worthwhile?

We are running this app under Weblogic 8.1 (so JDK 1.4.2)

---

sorry Thomas, let me clarify..

the HEAD version has the traditional singleton pattern (private constructor, getInstance() etc)

the branch version has no constructor, is a 'public abstract class' and modified all the methods on the object to be 'static'. The code that used to exist in the private constructor is moved into a static block.

Then all usages of the class are changed which causes multiple conflicts in the merge.

There are a few cases where this change was made.

java design-patterns singleton

Share

Improve this question

Follow

edited Mar 12, 2012 at 1:44

Bill the Lizard

405k ● 211 ● 572 ● 889

asked Aug 26, 2008 at 14:45

Matt

7 Answers

Sorted by: Highest score (default)

16

From a strict runtime performance point of view, the difference is really negligible. The main difference between the two lies down in the fact that the "static" lifecycle is linked to the classloader, whereas for the singleton it's a regular instance lifecycle. Usually it's better to stay away from the ClassLoader business, you avoid some tricky problems, especially when you try to reload the web application.

Share Improve this answer

Follow

answered Aug 26, 2008 at 14:52

Damien B

2,003 ● 15 ● 19

2 A singleton is tied to ClassLoader lifetime of the class loader that loaded it, as much as a static is. – Tom Hawtin - tackline Sep 11, 2008 at 21:30

5 And? Everything is tied to the lifetime of the ClassLoader, but by putting a singleton, you gain an extra lifecycle layer, with more chances (finalize) to handle things properly.
– Damien B Oct 3, 2008 at 21:13

16

I would use a singleton if it needed to store any state, and static classes otherwise. There's no point in instantiating something, even a single instance, unless it needs to store something.

Share Improve this answer

Follow

answered Aug 26, 2008 at 14:59

levand

8,480 ● 3 ● 42 ● 55

11

Static is bad for extensibility since static methods and fields cannot be extended or overridden by subclasses.

It's also bad for unit tests. Within a unit test you cannot keep the side effects of different tests from spilling over since you cannot control the classloader. Static fields initialized in one unit test will be visible in another, or worse, running tests concurrently will yield unpredictable results.

Singleton is generally an ok pattern when used sparingly. I prefer to use a DI framework and let that manage my

instances for me (possibly within different scopes, as in Guice).

Share Improve this answer

Follow

answered Aug 26, 2008 at 17:52

Mark Renouf

31k ● 19 ● 96 ● 125

3

If my original post was the correct understanding and the discussion from Sun that was linked to is accurate (which I think it might be), then I think you have to make a trade off between clarity and performance.

Ask yourself these questions:

1. Does the Singleton object make what I'm doing more clear?
2. Do I need an object to do this task or is it more suited to static methods?
3. Do I need the performance that I can gain by not using a Singleton?

Share Improve this answer

Follow

answered Aug 26, 2008 at 14:54

Thomas Owens

116k ● 99 ● 317 ● 436

The question is actually about the performance gain :-) Consider you have a controller that handles 100 requests per second. is this relevant for choosing between singleton or class as a stateless service ? – lisak Apr 1, 2011 at 21:15

3

From my experience, the only thing that matters is which one is easier to mock in unit tests. I always felt Singleton is easier and natural to mock out. If your organization lets you use JMockit, it doesn't matter since you can overcome these concerns.

Share Improve this answer

Follow

answered Aug 26, 2008 at 15:14

Cem Catikkas

7,174 ● 4 ● 30 ● 33

0

Does this discussion help? (I don't know if it's taboo to link to another programming forum, but I'd rather not just quote the whole discussion =))

Sun Discussion on this subject

The verdict seems to be that it doesn't make enough of a difference to matter in most cases, though technically the static methods are more efficient.

Share Improve this answer

Follow

answered Aug 26, 2008 at 14:50

EdgarVerona

1,588 ● 1 ● 15 ● 23

0

Write some code to measure the performance. The answer is going to be dependent on the JVM(Sun's JDK might perform differently than JRockit) and the VM flags your application uses.

Share Improve this answer

Follow

answered Sep 9, 2008 at 1:44

Rob Spieldenner

1,698 1 16 26