MONDAY, MARCH 11, 2013

## Difference between Singleton Pattern vs Static Class in Java

Singleton pattern vs Static Class (a class, having all static methods) is another interesting questions, which I missed while blogging about Interview questions on Singleton pattern in Java. Since both Singleton pattern and static class provides good accessibility, and they share some similarities e.g. both can be used without creating object and both provide only one instance, at very high level it looks that they both are intended for same task. Because of high level similarities, interviewer normally ask questions like, *Why you use Singleton instead of Static Methods,* or Can you replace Singleton with static class, and what are differences between Singleton pattern and static in Java. In order to answer these question, it's important to remember fundamental difference between Singleton pattern and static class, former gives you an Object, while later just provide static methods. Since an object is always much more capable than a method, it can guide you when to use Singleton pattern vs static methods.

In this Java article we will learn, where to use Singleton pattern in Java, and when static class is better alternative. By the way, JDK has examples of both singleton and static, and that too very intelligently e.g. `java.lang.Math` is a final class with full of static methods, on the other hand `java.lang.Runtime` is a Singleton class in Java. For those who are not familiar with Singleton design pattern or static class, static class is a Java class, which only contains static methods, good examples of static class is `java.lang.Math`, which contains lots of utility methods for various maths function e.g. `sqrt()`. While Singleton classes are those, which has only one instance during application life cycle like `java.lang.Runtime`.

## When to use Static Class in place of Singleton in Java

Indeed there are some situations, where static classes makes sense than Singleton. Prime example of this is `java.lang.Math` which is not Singleton, instead a class with all static methods. Here are few situation where I think using static class over Singleton pattern make sense:

1) If your Singleton is not maintaining any state, and just providing global access to methods, than consider using static class, as static methods are much faster than Singleton, because of static binding during compile time. But remember its not advised to maintain state inside static class, especially in concurrent environment, where it could lead subtle race conditions when modified parallel by multiple threads without adequate synchronization.

You can also choose to use static method, if you need to combine bunch of utility method together. Anything else, which requires singles access to some resource, should use Singleton design pattern.

## Difference between Singleton vs Static in Java

This is answer of our second interview question about Singleton over static. As I said earlier, fundamental difference between them is, one represent object while other represent a method. Here are few more differences between static and singleton in Java.

1) Static class provides better performance than Singleton pattern, because static methods are bonded on compile time.

2) One more difference between Singleton and static is, ability to override. Since static methods in Java cannot be overridden, they leads to inflexibility. On the other hand, you can override methods defined in Singleton class by extending it.

3) Static classes are hard to mock and consequently hard to test than Singletons, which are pretty easy to mock and thus easy to test. It's easier to write JUnit test for Singleton than static classes, because you can pass mock object whenever Singleton is expected, e.g. into constructor or as method arguments.

4) If your requirements needs to maintain state than Singleton pattern is better choice than static class, because maintaining state in later case is nightmare and leads to subtle bugs.

5) Singleton classes can be lazy loaded if its an heavy object, but static class doesn't have such advantages and always eagerly loaded.

6) Many Dependency Injection framework manages Singleton quite well e.g. Spring, which makes using them very easy.

These are some differences between static class and singleton pattern, this will help to decide between two, which situation arises. In next section we will when to choose Singleton pattern over static class in Java.

## Advantage of Singleton Pattern over Static Class in Java

Main advantage of Singleton over static is that former is more object oriented than later. With Singleton, you can use Inheritance and Polymorphism to extend a base class, implement an interface and capable of providing different implementations. If we talk about `java.lang.Runtime`, which is a Singleton in Java, call to `getRuntime()` method return different implementations based on different JVM, but guarantees only one instance per JVM, had `java.lang.Runtime` an static class, it's not possible to return different implementation for different JVM.

That's all on difference between Singleton and static class in Java. When you need a class with full OO capability , chose Singleton, while if you just need to store bunch of static methods together, than use static class.

Other **Java Design Pattern Tutorials** from Javarevisited Blog

When to use Builder design pattern in Java

A Real life example of Observer Pattern in Java

How to use Decorator pattern in Java

Difference between Factory and Abstract Factory pattern in Java

10 SOLID and Object Oriented design principles Java Programmer should know



Posted by Javin Paul at 5:46 AM +184 Recommend this on Google

Labels: core java , core java interview question , design patterns
Location: United States

### 15 comments :

**Anand Pritam said...**

I Disagree with following :
2) One more difference between Singleton and static is, ability to override. Since static methods in Java cannot be overridden, they leads to inflexibility. On the other hand, you can override methods defined in Singleton class by extending it.

Here the author say you can override methods defined in Singleton.Isn't it breaking Singleton.

March 11, 2013 at 10:48 PM

**SARAL SAXENA said...**

Hi Javin Gr8 article , I want to add two things...

1) Singleton class can be extended. Polymorphism can save a lot of repetition.
2) A Singleton class can implement an interface, which can come in handy when you want to separate implementation from API.
3)Singleton can be extended. Static not.
4)Singleton creation may not be threadsafe if it isn't implemented properly. Static not.
5)Singleton can be passed around as an object. Static not.
6)Singleton can be garbage collected. Static not.
7)Singleton object stores in Heap but, static object stores in stack
8)We can clone the object of Singleton but, we can not clone the static class object
9)Singleton class follow the OOP(object oriented principles) but not static class

Jav...
Top 15 Data Structures and Algorithm Interview Que...
Difference between Struts 1 and Struts 2 framework...
Bitwise and BitShift Operators in Java - AND, OR, ...
Difference between Singleton Pattern vs Static Cla...
How to Increase Console Buffer Size in Eclipse IDE...
ReentrantLock Example in Java, Difference between ...
5 books to learn Spring framework and Spring MVC f...
How to create Immutable Class and Object in Java -...
Steps to Create JUnit Test in Eclipse and Netbeans...

► February ( 18 )
► January ( 18 )

► 2012 ( 217 )
► 2011 ( 145 )
► 2010 ( 33 )

**References**

Java API documentation JDK 6
Spring framework doc
Struts
JDK 7 API
MySQL
Linux
Eclipse
jQuery

10)Another advantage of a singleton is that it can easily be serialized, which may be necessary if you need to save its state to disc, or send it somewhere remotely.

The big difference between a singleton and a bunch of static methods is that singletons can implement interfaces (or derive from useful base classes, although that's less common IME), so you can pass around the singleton as if it were "just another" implementation.

March 12, 2013 at 8:50 AM

**Javin @ Must Override Eclipse Error said...**

@Saral, Good points. On same note, Static class are good for utility classes, which doesn't maintain state and not required to be extended.

March 13, 2013 at 4:09 AM

**Octavian Nita said...**

I would agree with Anand in that most definitions consider extending a Singleton as breaking the definition... Likewise for cloning and serialization...

Also, since one can use/pass around between threads the Singleton instance, any mutable state it might have makes the object susceptible to race conditions as well...

The fact that "a Singleton follows the OOP principles" is utterly irrelevant since in order to really implement the pattern we have to renounce most of the OOP principles and flexibility anyway...

Moreover, having static classes/methods means less parameters to a method that needs this (static or singleton) functionality, eventually less injection, etc. so basically reduced "interfaces" which is normally good. However, this "lack of dependencies" might tend to hide things, introduce magic behavior and inflexible design. Hence, we get to the only situation and strong point (that Saral made and) that could favor a Singleton: implementing an interface and being passed around as just one of the possible implementations...

March 16, 2013 at 7:15 AM

**José Cruz said...**

I cant understand the discussion. Singleton it is a design pattern with a specific purpose, having just on single instance of a class. You can achieve that with different forms in Java: Enumerator and static classes. Within these forms you can use: Lazy Initialization, Eager Initialization, static block Initialization and the Enum way.
So, its not a question of difference, but when to use one or another or both.

March 16, 2013 at 3:25 PM

**Javin @ how classloader works said...**

@Jose Cruz, Thanks for your comment, yes ultimately it comes down to, when to use Singleton and Static class. There are situations like implementing Interface, being part of a type hierarchy where Singleton fits better than static classes, but if you just have some utility methods, wrapping them in static class is better.

March 18, 2013 at 6:03 AM

**Anonymous said...**

I can think of a situation when many classloaders are around and we wish to make a class singleton, in that case having a static class will not solve the problem as the class may have been loaded by number of classloaders, this happen all the time in web application servers. To protect this behaviour, classloader safe implementation of Singleton class is a must. Just think about a STATIC ConnectionPool object in a web server, if each of the application using its own classloader may potentially create its own connection pool object!!

April 4, 2013 at 1:04 AM

**Avtar said...**

Hello Javin, What is difference between Singleton and Factory pattern? Doesn't Singleton is a kind of factory e.g. getInstance() method returns an instance of Singleton rather than client creating instance using new() operator? doesn't this is similar to Factory method pattern? I would say Singleton is a combination of Factory method + job to keep Singleton as Singleton

April 29, 2013 at 11:29 PM

**Vaishak said...**

How can you extend a singleton class having a private constructor? There is no question of overriding the methods in a Singleton class because INHERITED instance methods can only be overridden. Nevertheless, a singleton class may override methods inherited from some other class.

June 16, 2013 at 12:58 AM

**Aditya Shanbhag said...**

Nice articles.. But In Difference between Singleton vs Static in Java
2nd Point..
How can a class extend Singleton class...if its constructor is private...

Please correct it..

July 12, 2013 at 5:05 AM

**Raistlic said...**

Hi Javin, I disagree with a few points here:

1 - If something maintains states, then making it a singleton is just as bad as maintaining states in static environment. You will face the same complexity if it is accessed in a multi-thread environment.

and when being stateless (or immutable) :

2 - Singleton can be lazy-instantiated: most of the time, this is a 'fake pro', because for most of the singleton classes, the first time you refer to its class name is to call its (static) getInstance() method, which means, most of the time it is instantiated when its class is loaded, which makes whatever 'lazy-instantiation' technic you use inside the getInstance() method pointless.

3 - 2 is true unless the class provides other services via other static methods(in which case you got other reasons to refer to its class name than calling getInstance() method) : but would you do that? most likely no, because why provide half the services via the singleton instance, while the other half via static methods?

4 - Singleton can be overridden: By definition, singleton is one type of 'instantiation control' (it is the one that maintains only one instance), otherwise you cannot enforce it be a singleton, and (other than using enum), the way to achieve instantiation control is to hide its constructor, which means, making its constructor private, ---- which means, you cannot extend the class outside the class itself (you can extend it in its own static member class). Which means, if you want to alter its internal implementation, -- you will need to modify the class itself. If you can modify the class, -- you can just modify the behaviour of a static method as well, the only promise you published when using a static method is its signature.

There is one thing I agree with: singleton is an object, an object is better/more flexible than a static method.

So what exactly makes an object more powerful than static methods? -- It can be passed around.

If you hard code the logic of getting a singleton instance EVERYWHERE in your business logic: Singleton.getInstance().somethod(...), then it is JUST AS EVIL AS XXXUtility.somemethod(...).

The right way of using singleton, I believe, is that you define the services a singleton provides as an interface, and use the interface in your business logic classes; then you define your singleton as to implement the interface, call Singleton.getInstance() in one place, and provide the instance to your business logic objects where it's needed.

In short, you gain the power of an object by DECOUPLE ITS INSTANTIATION LOGIC AND YOUR BUSINESS LOGIC, and this is one main reason frameworks such as Spring exist for. Otherwise it makes no difference whether you use a singleton pattern or use static methods, -- they are just equally bad.

November 6, 2013 at 3:26 PM

**Vivek Vermani said...**

More differences -

Serialization - Static members belong to the class and hence can't be serialized.
b. Though we have made the constructor private, static member variables still will be carried to subclass.
c. We can't do lazy initialization as everything will be loaded upon class loading only.

http://www.buggybread.com/2013/11/java-design-pattern-singleton.html

November 25, 2013 at 11:42 AM

**Saurabh Gupta said...**

I believe that this is not about the static class but class with static method , sometime it's make me confused.
please let me know if i am incorrect.

July 5, 2014 at 12:10 AM

**Anonymous said...**

The "pros" mentioned here I have never encountered in my 10 years of programming. Contrived

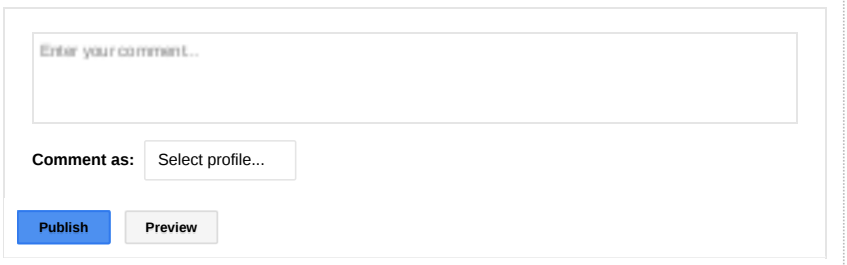
August 3, 2014 at 7:30 AM

**Igor Ganapolsky said...**

An important point to note is about garbage collection. If you want to keep memory low and efficient, a singleton can be garbage collected - as it is an object. But a static class or static method can lead to unnecessary memory usage and even memory leaks.

September 18, 2014 at 12:19 PM

**Post a Comment**




Newer Post Home Older Post

Subscribe to: Post Comments ( Atom )

About Me Privacy Policy