THURSDAY, JULY 13, 2017

Why Enum Singleton are better in Java

Enum Singletons are new way to implement Singleton pattern in Java by using Enum with just one instance. Though Singleton pattern in Java exists from long time Enum Singletons are relatively new concept and in practice from Java 5 onwards after introduction of Enum as keyword and feature. This article is somewhat related to my earlier post on Singleton, 10 interview questions on Singleton pattern in Java where we have discussed common questions asked on interviews about Singleton pattern and 10 Java enum examples, where we have seen how versatile enum can be. This post is about **why should we use Enum as Singleton in Java**, What benefit it offers compared to conventional singleton methods etc.

Java Enum and Singleton Pattern

Following are some reasons which make sense to me for using Enum to implement Singleton pattern in Java. By the way If you like articles on design pattern than you can also check my post on Builder design pattern and Decorator design pattern .

1) Enum Singletons are easy to write

This is by far biggest advantage, if you have been writing Singletons prior ot Java 5 than you know that even with double checked locking you can have more than one instances. though that issue is fixed with Java memory model improvement and gurantee provided by volatile variables from Java 5 onwards but it still tricky to write for many beginners. compared to double checked locking with synchronization Enum singletons are cake walk. If you don't believe than just compare below code for conventional singleton with double checked locking and Enum Singletons:

Singleton using Enum in Java

This is the way we generally declare Enum Singleton , it may contain instace variable and instance method but for sake of simplicity I haven't used any, just beware that if you are using any instance method than you need to ensure thread-safety of that method if at all it affect the state of object. By default creation of Enum instance is thread safe but any other method on Enum is programmers responsibility.

```
/**
* Singleton pattern example using Java Enumj
*/
public enum EasySingleton{
    INSTANCE;
}
```

You can acess it by `EasySingleton.INSTANCE`, much easier than calling `getInstance()` method on Singleton.

Singleton example with double checked locking

Below code is an example of double checked locking in Singleton pattern, here `getInstance()` method checks two times to see whether INSTANCE is null or not and that's why it's called double checked locking pattern, remember that double checked locking is broker before Java 5 but with the guranteed of volatile variable in Java 5 memory model, it should work perfectly.

```
/**
* Singleton pattern example with Double checked Locking
*/
public class DoubleCheckedLockingSingleton{
     private volatile DoubleCheckedLockingSingleton INSTANCE;

     private DoubleCheckedLockingSingleton(){}
```

Follow by Email
Email address... Submit

Interview Questions
core java interview question (169)
Coding Interview Question (72)
data structure and algorithm (70)
interview questions (48)
object oriented programming (31)
SQL Interview Questions (30)
design patterns (30)
thread interview questions (30)
collections interview questions (25)
spring interview questions (19)
database interview questions (16)
servlet interview questions (15)
Programming interview question (6)
hibernate interview questions (6)

Best of Javarevisited
How Spring MVC works internally?
How to design a vending machine in Java?
How HashMap works in Java?
Why String is Immutable in Java?
10 Articles Every Programmer Must Read
How to convert lambda expression to method reference in Java 8?
10 Tips to improve Programming Skill
10 OOP design principles programmer should know
How Synchronization works in Java?
10 tips to work fast in Linux
5 Books to improve Coding Skills

Java Tutorials
date and time tutorial (21)
FIX protocol tutorial (15)
Java Certification OCPJP SCJP (24)
java collection tutorial (73)
java IO tutorial (28)
Java JSON tutorial (12)
Java multithreading Tutorials (56)
Java Programming Tutorials (18)
Java xml tutorial (16)
JDBC (29)
jsp-servlet (37)
online resources (123)

Followers

```
    public DoubleCheckedLockingSingleton getInstance(){
        if(INSTANCE == null){
            synchronized(DoubleCheckedLockingSingleton.class){
                //double checking Singleton instance
                if(INSTANCE == null){
                    INSTANCE = new DoubleCheckedLockingSingleton();
                }
            }
        }
        return INSTANCE;
    }
}
```

You can call `DoubleCheckedLockingSingleton.getInstance()` to get access of this Singleton class.

Now Just look at amount of code needed to create a **lazy loaded thread-safe Singleton**. With Enum Singleton pattern you can have that in one line because creation of Enum instance is thread-safe and guranteed by JVM.

People may argue that there are better way to write Singleton instead of Double checked locking approach but every approach has there own advantages and disadvantages like I mostly prefer static field Singleton intialized during classloading as shwon in below example, but keep in mind that is not a **lazy loaded Singleton**:

Singleton pattern with static factory method

This is one of my favorite method to impelemnt Singleton pattern in Java, Since Singleton instance is static and final variable it initialized when class is first loaded into memeory so creation of instance is inherently thread-safe.

```
/**
* Singleton pattern example with static factory method
*/

public class Singleton{
    //initailzed during class loading
    private static final Singleton INSTANCE = new Singleton();

    //to prevent creating another instance of Singleton
    private Singleton(){}

    public static Singleton getSingleton(){
        return INSTANCE;
    }
}
```

You can call `Singleton.getSingleton()` to get access of this class.

2) Enum Singletons handled Serialization by themselves

Another problem with conventional Singletons are that once you implement serializable interface they are no longer remain Singleton because `readObject()` method always return a new instance just like constructor in Java. you can avoid that by using `readResolve()` method and discarding newly created instance by replacing with Singeton as shwon in below example :

```
    //readResolve to prevent another instance of Singleton
    private Object readResolve(){
        return INSTANCE;
    }
```

This can become even more complex if your Singleton Class maintain state, as you need to make them transient, but witn **Enum Singleton**, Serialization is guarnateed by JVM.

3) Creation of Enum instance is thread-safe

As stated in point 1 since creatino of Enum instance is thread-safe by default you don't need to worry about double checked locking.

In summary, given the **Serialzation and thraead-safety guaranteed** and with couple of line of code enum Singleton pattern is best way to create Singleton in Java 5 world. you can still use other popular methods if you feel so but I still have to find a *convincing reason not to use Enum as Singleto*n, let me know if you got any.

Further Learning

Complete Java Masterclass
Java Fundamentals: The Java Language
Java In-Depth: Become a Complete Java Engineer!

Other Java articles from Javarevisited blog:

Why multiple inheritance is not supported in Java
Why operator overloading is not supported in Java
How Generics works in Java
10 tips to write better comments while coding
10 Object oriented design principles every Java programmer should know
10 JVM options you should know.

Java 1.5 Generics Tutorial: How Generics in Java works with Example of Collections, Best practices, Gotchas

Java Generics Tutorial Generics in Java is one of important feature added in Java 5 along with Enum, autoboxing and varargs, to provide ...

Java Revisited

By javin paul at July 13, 2017

Labels: core java , java enum
Location: United States

28 comments :

Anonymous said...

U have forget the "static" in the "Singleton example with double checked locking" it should be :
public static DoubleCheckedLockingSingleton getInstance(){

also in the first example the enum one to make it like the rest it is more 1:1 to work like the rest the example to become:

```
public enum Singleton {
INSTANCE;

public static Singleton getInstance(){
return INSTANCE;
}
}
```

July 17, 2012 at 7:57 AM

Vineet said...

Thanks for your tutorial dude. I wanted to implement Singleton design pattern using Java enum and looking for an example, didn't know that it would be this easy. you Singleton pattern example with Java Enum is just a piece of cake.

July 22, 2012 at 11:41 PM

Unknown said...

EnumSingleton or SingletonClassHolder are fine when your singleton's scope == ClassLoader (static). For other cases (lazy singleton per conversation/request/process), the DoubleCheckedSingleton is better.

September 10, 2012 at 9:35 AM

Anonymous said...

Enums can't extend other enums or classes. And they are final.
Sometimes that's enough not to use such singletons...

September 11, 2012 at 5:11 AM

bracco23 said...

Enum singleton are perfect as soon as you don't need a singleton that inherit from something. In that case, you cannot use enum anymore and you must use other way...

January 2, 2013 at 9:11 PM

Michał Niwiński said...

In "Singleton pattern with static factory method" you didn't use factory method pattern but simple factory, which in fact is not "official" pattern.

February 3, 2013 at 11:10 AM

Anonymous said...

In DoubleCheckedLockingSingleton example, instance variable and getInstance() method must be static. Otherwise, the user has to call object.getInstance() where object is an instantiation of DoubleCheckedLockingSingleton . But the constructor is private; therefore, no one can call getInstance()

June 3, 2013 at 11:10 PM

Anonymous said...

This is the practical way:

```
public enum ConnectionManager {
INSTANCE;

private EntityManagerFactory emf;

private ConnectionManagerEnum() {
emf = Persistence.createEntityManagerFactory("cassandra_pu");
}

public EntityManager getEntityManager() {
return emf.createEntityManager();
}
}
```

Usage: ConnectionManager.INSTANCE.getEntityManager()

July 10, 2013 at 2:42 AM

Anonymous said...

in last comment, is it ConnectionManagerEnum or ConnectionManager?

August 9, 2013 at 2:59 PM

KK Sharma said...

we can't have private constructor
private Singleton(){}

August 21, 2013 at 4:51 AM

Saugat Lama said...

Fibonacci Series in Java Without Recursion - Progr...
java.lang.ClassNotFoundException: com.microsoft.sq...
Top 5 Books to learn UML (Unified Modeling Languag...
How SSL, HTTPS and Certificates Works in Java web ...
How to Convert Fahrenheit to Celsius in Java with ...
Role based Access control using Spring Security an...
How to solve java.lang.UnsatisfiedLinkError: no oc...
Top 30 Linked List Algorithm Questions from Progra...
How to resolve java.lang.UnsupportedClassVersionEr...
How to use wait, notify and notifyAll in Java - Pr...

► June (56)
► May (48)
► April (79)
► March (99)
► February (87)
► January (104)

► 2016 (117)

References

1. Oracle's Java Tech Network
2. jQuery Documentation
3. Microsoft SQL Server Documentation
4. Java SE 8 API Documentation
5. Spring Documentation
6. Oracle's JAva Certification
7. Spring Security 5 Documentation

Pages

Privacy Policy

You have to have "static" modifier in both the getInstance() method and volatile variable if you want double checked locking. If method is not static then, you cannot access getInstance by just Singleton.getInstance(), and if the volatile instance variable is not static, then the static getinstance method cannot access the variable.

```
/**
* Singleton pattern example with Double checked Locking
*/
public class DoubleCheckedLockingSingleton{
private volatile DoubleCheckedLockingSingleton INSTANCE;

private DoubleCheckedLockingSingleton(){}

public DoubleCheckedLockingSingleton getInstance(){
if(INSTANCE == null){
synchronized(DoubleCheckedLockingSingleton.class){
//double checking Singleton instance
if(INSTANCE == null){
INSTANCE = new DoubleCheckedLockingSingleton();
}
}
}
return INSTANCE;
}
}
```

December 10, 2013 at 6:02 PM

Anonymous said...

It should be ConnectionManager I guess. Because this method is like constructor, which will be called by Enum while accessed/called.

December 10, 2013 at 7:47 PM

Aniket Thakur said...

In case of Singleton pattern with static factory method what is the need to declare the Singleton Object as final? If you see java.lang.Runtime class which implements Singleton pattern with early initialization declaration is

private static Runtime currentRuntime = new Runtime();

Yes as in this case object will be created when class is loaded synchronization is not needed. But I do not understand why do we need to declare it as final as you have shown in your example code.

Also as others have mentioned in their comment in your double checked locking you need to declare your getInstance() method as well as the singleton object as static.

March 9, 2014 at 11:25 AM

Tschoah said...

Hi there,

This is regarding your example under "Singleton example with double checked locking". The method getInstance() first checks if INSTANCE hasn't been initialised yet. If not then it steps into the synchronized block and initialises INSTANCE. I see an issue here.

If two threads access the method getInstance on the same object and thread A stops right after the INSTANCE==null check and thread B now gets the CPU, that thread B will run the same check and then step into the synchronized block where it then initialises INSTANCE. After it will have left the synchronized block thread A gets the CPU back which then steps into the synchronized block where it initialises INSTANCE again. It basically overwrites the value of INSTANCE. Therefore this implementation isn't thread safe.

April 7, 2014 at 8:23 AM

János Kranczler said...

@Tschoah: When A steps into the synchronized block it will check the INSTANCE for null again.

Anyway the example is bad, because there is no static modifier on the INSTANCE variable and the getInstance() method.

May 9, 2014 at 2:44 AM

valjok said...

You cannot declare them locally, within a method.

June 14, 2014 at 3:30 AM

Anonymous said...

Please anyone explain INSTANCE object in enum singleton example

April 17, 2015 at 4:09 AM

Anonymous said...

why the author does not editing double check locking code in the above article, it has some serious mistakes!!!

July 29, 2015 at 10:17 AM

Javin Paul said...

@Anonymous, what is the mistake you see on doubled checked locking code? care to explain?

July 30, 2015 at 5:30 AM

Javin Paul said...

@Anonymous, INSTANCE object in enum is just one enum constant of enum type Singleton.

July 30, 2015 at 5:31 AM

Anonymous said...

"Static" in missing how can you access the getInstance() method. Double Checking

```
public class DoubleCheckedLockingSingleton{
private static volatile DoubleCheckedLockingSingleton INSTANCE;

private DoubleCheckedLockingSingleton(){}

public static DoubleCheckedLockingSingleton getInstance(){
if(INSTANCE == null){
synchronized(DoubleCheckedLockingSingleton.class){
//double checking Singleton instance
if(INSTANCE == null){
INSTANCE = new DoubleCheckedLockingSingleton();
}
}
}
return INSTANCE;
}
}
```

September 9, 2015 at 12:41 AM

Double check lock said...

A small correction : static is missing from getInstance method in double check example

August 29, 2016 at 1:06 PM

Anonymous said...

Looks like , Its taken from Effective Java by Joshua Bloch.
Due credits should be given :)

October 16, 2016 at 2:39 AM

Anonymous said...

We do get different Object of logger in different class , then how come logger class is considered as singleton.
For example , we get logger instance for SomeClass as

Logger someClassLogger = Logger.getLogger(SomeClass.class);

In AnotherClass logger instance would be:

Logger anotherClassLogger = Logger.getLogger(AnotherClass.class);

Now, if it returns the same object then how come during printing message, different class names are printed

Example :

(SomeClass:22) Hello, in SomeClass
(AnotherClass:451) I am not same with SameClass

February 18, 2017 at 4:00 PM

Anonymous said...

Enum singelton is lazy loaded or not?

April 24, 2017 at 3:41 AM

Javin Paul said...

@Anonymous, Enum Singleton is not lazy loaded. If you want a lazy loaded singleton then Singleton holder pattern is best.

April 24, 2017 at 5:26 AM

Gourav said...

enum singleton pattern is Lazy Loaded. please correct the post. see this for proof: https://stackoverflow.com/questions/16771373/singleton-via-enum-way-is-lazy-initialized

September 2, 2017 at 6:01 AM

Anonymous said...

http://www.oracle.com/technetwork/articles/java/singleton-1577166.html

February 12, 2018 at 8:10 AM

Post a Comment

Enter your comment...

Comment as: Google Accour

Publish Preview

Difference Between Factory and Abstract Factory Design Pattern in Java

Both Abstract Factory and Factory design pattern are creational design pattern and use to decouple clients from creating object they need,...

Java Revisited

Top 5 Data Structures and Algorithm Online Courses for Java, JavaScript, and Python Programmers - Best of Lot

Data Structure and Algorithm is one of the most important topics for programmers. The best thing about them is that they never get out-of-d...

Java Revisited

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Search This Blog

Search