Java Singleton Design Pattern Best Practices with Examples

APRIL 2, 2018 BY PANKAJ — 197 COMMENTS

Java Singleton Pattern is one of the **Gangs of Four Design patterns** and comes in the **Creational Design Pattern** category. From the definition, it seems to be a very simple design pattern but when it comes to implementation, it comes with a lot of implementation concerns. The implementation of Java Singleton pattern has always been a controversial topic among developers. Here we will learn about Singleton design pattern principles, different ways to implement Singleton design pattern and some of the best practices for it's usage.

Java Singleton

- Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
- The singleton class must provide a global access point to get the instance of the class.
- Singleton pattern is used for logging, drivers objects, caching and thread pool.
- Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.
- Singleton design pattern is used in core java classes also, for example `java.lang.Runtime`, `java.awt.Desktop`.

Java Singleton Pattern

To implement Singleton pattern, we have different approaches but all of them have following common concepts.

- Private constructor to restrict instantiation of the class from other classes.
- Private static variable of the same class that is the only instance of the class.
- Public static method that returns the instance of the class, this is the global access point for outer world to get the instance of the singleton class.

In further sections, we will learn different approaches of Singleton pattern implementation and design concerns with the implementation.

Eager initialization

In eager initialization, the instance of Singleton Class is created at the time of class loading, this is the easiest method to create a singleton class but it has a drawback that instance is created even though client application might not be using it.

Here is the implementation of static initialization singleton class.

```
package com.journaldev.singleton;

public class EagerInitializedSingleton {
    private static final EagerInitializedSingleton instance = new
EagerInitializedSingleton();

    //private constructor to avoid client applications to use constructor
    private EagerInitializedSingleton(){}

    public static EagerInitializedSingleton getInstance(){
        return instance;
    }
}
```

If your singleton class is not using a lot of resources, this is the approach to use. But in most of the scenarios, Singleton classes are created for resources such as File System, Database connections etc and we should avoid the instantiation until unless client calls the `getInstance` method. Also this method doesn't provide any options for exception handling.

Static block initialization

Static block initialization implementation is similar to eager initialization, except that instance of class is created in the static block that provides option for exception handling.

```
package com.journaldev.singleton;

public class StaticBlockSingleton {

    private static StaticBlockSingleton instance;

    private StaticBlockSingleton(){}

    //static block initialization for exception handling
    static{
        try{
            instance = new StaticBlockSingleton();
        }catch(Exception e){
            throw new RuntimeException("Exception occured in creating singleton
instance");
        }
    }

    public static StaticBlockSingleton getInstance(){
        return instance;
    }
}
```

Both eager initialization and static block initialization creates the instance even before it's being used and that is not the best practice to use. So in further sections, we will learn how to create Singleton class that supports lazy initialization.

Read: Java static

Lazy Initialization

Lazy initialization method to implement Singleton pattern creates the instance in the global access method. Here is the sample code for creating Singleton class with this approach.

```java
package com.journaldev.singleton;

public class LazyInitializedSingleton {

    private static LazyInitializedSingleton instance;

    private LazyInitializedSingleton(){}

    public static LazyInitializedSingleton getInstance(){
        if(instance == null){
            instance = new LazyInitializedSingleton();
        }
        return instance;
    }
}
```

The above implementation works fine incase of single threaded environment but when it comes to multithreaded systems, it can cause issues if multiple threads are inside the if loop at the same time. It will destroy the singleton pattern and both threads will get the different instances of singleton class. In next section, we will see different ways to create a thread-safe singleton class.

Thread Safe Singleton

The easier way to create a thread-safe singleton class is to make the global access method synchronized, so that only one thread can execute this method at a time. General implementation of this approach is like the below class.

```java
package com.journaldev.singleton;

public class ThreadSafeSingleton {
    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton(){}

    public static synchronized ThreadSafeSingleton getInstance(){
        if(instance == null){
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }

}
```

Above implementation works fine and provides thread-safety but it reduces the performance because of cost associated with the synchronized method, although we need it only for the first few threads who might create the separate instances (Read: Java Synchronization). To avoid this extra overhead every time, **double checked locking** principle is used. In this approach, the synchronized block is used inside the if condition with an additional check to ensure that only one instance of singleton class is created.

Below code snippet provides the double checked locking implementation.

```
public static ThreadSafeSingleton getInstanceUsingDoubleLocking(){
    if(instance == null){
        synchronized (ThreadSafeSingleton.class) {
            if(instance == null){
                instance = new ThreadSafeSingleton();
            }
        }
    }
    return instance;
}
```

Read: Thread Safe Singleton Class

Bill Pugh Singleton Implementation

Prior to Java 5, java memory model had a lot of issues and above approaches used to fail in certain scenarios where too many threads try to get the instance of the Singleton class simultaneously. So Bill Pugh came up with a different approach to create the Singleton class using a inner static helper class. The Bill Pugh Singleton implementation goes like this;

```
package com.journaldev.singleton;
```

```
public class BillPughSingleton {

    private BillPughSingleton(){}

    private static class SingletonHelper{
        private static final BillPughSingleton INSTANCE = new BillPughSingleton();
    }

    public static BillPughSingleton getInstance(){
        return SingletonHelper.INSTANCE;
    }
}
```

Notice the **private inner static class** that contains the instance of the singleton class. When the singleton class is loaded, `SingletonHelper` class is not loaded into memory and only when someone calls the *getInstance* method, this class gets loaded and creates the Singleton class instance.

This is the most widely used approach for Singleton class as it doesn't require synchronization. I am using this approach in many of my projects and it's easy to understand and implement also.

Read: Java Nested Classes

Using Reflection to destroy Singleton Pattern

Reflection can be used to destroy all the above singleton implementation approaches. Let's see this with an example class.

```
package com.journaldev.singleton;

import java.lang.reflect.Constructor;

public class ReflectionSingletonTest {

    public static void main(String[] args) {
        EagerInitializedSingleton instanceOne =
EagerInitializedSingleton.getInstance();
        EagerInitializedSingleton instanceTwo = null;
        try {
            Constructor[] constructors =
EagerInitializedSingleton.class.getDeclaredConstructors();
            for (Constructor constructor : constructors) {
                //Below code will destroy the singleton pattern
                constructor.setAccessible(true);
                instanceTwo = (EagerInitializedSingleton) constructor.newInstance();
                break;
            }
        } catch (Exception e) {
            e.printStackTrace();
```

When you run the above test class, you will notice that hashCode of both the instances are not same that destroys the singleton pattern. Reflection is very powerful and used in a lot of frameworks like Spring and Hibernate, do check out **Java Reflection Tutorial**.

Enum Singleton

To overcome this situation with Reflection, Joshua Bloch suggests the use of Enum to implement Singleton design pattern as Java ensures that any enum value is instantiated only once in a Java program. Since Java Enum values are globally accessible, so is the singleton. The drawback is that the enum type is somewhat inflexible; for example, it does not allow lazy initialization.

```
package com.journaldev.singleton;

public enum EnumSingleton {

    INSTANCE;

    public static void doSomething(){
        //do something
    }
}
```

Read: Java Enum

Serialization and Singleton

Sometimes in distributed systems, we need to implement Serializable interface in Singleton class so that we can store it's state in file system and retrieve it at later point of time. Here is a small singleton class that implements Serializable interface also.

```
package com.journaldev.singleton;

import java.io.Serializable;

public class SerializedSingleton implements Serializable{

    private static final long serialVersionUID = -7604766932017737115L;

    private SerializedSingleton(){}

    private static class SingletonHelper{
        private static final SerializedSingleton instance = new SerializedSingleton();
    }

    public static SerializedSingleton getInstance(){
        return SingletonHelper.instance;
    }

}
```

The problem with above serialized singleton class is that whenever we deserialize it, it will create a new instance of the class. Let's see it with a simple program.

```
package com.journaldev.singleton;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

public class SingletonSerializedTest {

    public static void main(String[] args) throws FileNotFoundException, IOException,
ClassNotFoundException {
        SerializedSingleton instanceOne = SerializedSingleton.getInstance();
        ObjectOutput out = new ObjectOutputStream(new FileOutputStream(
                "filename.ser"));
        out.writeObject(instanceOne);
        out.close();

        //deserailize from file to object
```

Output of the above program is;

```
instanceOne hashCode=2011117821
instanceTwo hashCode=109647522
```

So it destroys the singleton pattern, to overcome this scenario all we need to do it provide the implementation of `readResolve()` method.

```
    protected Object readResolve() {
        return getInstance();
    }
```

After this you will notice that hashCode of both the instances are same in test program.

Read: Java Serialization and Java Deserialization.

I hope this article helps you in grasping fine details of Singleton design pattern, do let me know through your thoughts and comments.

« **PREVIOUS**
Thread Safety in Java Singleton Classes with Example Code

NEXT »
Factory Design Pattern in Java

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

Akash says

JULY 12, 2018 AT 3:41 AM

Easy to understand Thanks nice explanation.

Reply

pankaj says

JUNE 29, 2018 AT 11:09 PM

```
package com.singleton;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.io.Serializable;
class Demo1 implements Serializable {
/**
*
*/
private static final long serialVersionUID = 1L;
private static Demo1 obj = null;
private Demo1() {
}
protected Object readResolve() {
System.out.println("read reslove called");
return getInstance();
}
// if making synchronized then one thread will execute it at one time…//if
// performance of get instance
public static Demo1 getInstance() {
if (obj == null) {
System.out.println("obj null");
return new Demo1();
} else
return obj;
```

}
}
public class Demo {
public static void main(String args[]) throws FileNotFoundException, IOException,
ClassNotFoundException {
Demo1 obj = Demo1.getInstance();
ObjectOutput out = new ObjectOutputStream(new FileOutputStream("abc.ser"));
out.writeObject(obj);
out.close();
ObjectInput in = new ObjectInputStream(new FileInputStream("abc.ser"));
Demo1 obj1 = (Demo1) in.readObject();
System.out.println(obj.hashCode());
System.out.println(obj1.hashCode());
}
}
output :obj null
read reslove called
obj null
26253138
8860464
why hashcode different as i did readReslove()called
Reply

Bathula says
JUNE 30, 2018 AT 1:15 PM
You implemented static factory method instead of a singleton pattern. Above getInstance method returns a new Singular object after every invocation, hence different hashcodes.
Replace with below snippet and thread unsafe singleton class is ready.
public static Singular getInstance() {
if (obj == null) {
System.out.println("obj null");
return obj=new Singular();
} else
return obj;
}
Reply

pankaj says
JULY 3, 2018 AT 12:08 AM
Thanks for finding the error!Now its working
Reply

Wasim says

JUNE 26, 2018 AT 9:05 PM

Very nicely explained.

Reply

Pooja says

JUNE 25, 2018 AT 9:07 PM

Good article

Reply

Sandeep says

JUNE 3, 2018 AT 6:56 AM

Thanks for this – good one.

Reply

Patel says

MAY 8, 2018 AT 12:22 PM

Thank you so much for giving this kind of article for helping us a lot. Its very useful and helpful to understand various type to create the singleton classes.

Reply

Nitin says

APRIL 20, 2018 AT 11:55 AM

Nice one!!
But you didn't cover about cloning the singleton.
To avoid it you must override the clone method as below:
public Object clone() throws CloneNotSupportedException{
throw new CloneNotSupportedException("Singleton, cannot be clonned");
}

Reply

uday says

FEBRUARY 27, 2018 AT 4:41 AM

give an example on Enum singleton

Reply

FStarred says
FEBRUARY 21, 2018 AT 3:31 AM
Nice examples, altough Enum singleton is awful in every way, it opposites to OOP and it looks like a bad trick.
Reply

Marko says
MARCH 30, 2018 AT 6:25 AM

```
import java.util.Scanner;
public enum EnumSingleton {
INSTANCE;
private EnumSingleton(){
scanner = new Scanner(System.in);
}
private Scanner scanner;
public Scanner getScanner(){
return scanner;
}
}
```

Reply

Ugur says
FEBRUARY 20, 2018 AT 5:58 AM
Thanks for the detailed explanation and good work.
Reply

MIsa says
FEBRUARY 13, 2018 AT 3:43 PM
You are the best !!!
Reply

Neetesh says
FEBRUARY 4, 2018 AT 3:55 AM

Very nice explanation ..
Reply

Dattatray says
DECEMBER 27, 2017 AT 1:41 AM
I have use BillPughSingleton implement ion and try to destroy using reflection
and got exception as
java.lang.IllegalArgumentException: wrong number of arguments …
when i check .class file a one arg found
…
instanceTwo = (BillPughSingleton)constructor.newInstance(new Object[0]);
and its accept BillPughSingleton instance
Reply

Gaurav Jain says
MARCH 7, 2018 AT 12:41 AM
Try this:
instanceTwo = (BillPughSingleton)constructor.newInstance();
Reply

Manish Kumar Prasad says
DECEMBER 24, 2017 AT 12:25 PM
does BillPughSingleton supports multi-threaded env…?
Reply

rohit saroha says
MAY 4, 2018 AT 9:58 PM
yes a final variable is initialized once and remain same throughout its life-cycle.
Reply

Manish Kumar Prasad says
DECEMBER 24, 2017 AT 12:15 PM
Can we create singleton design pattern which can work in
multithreaded env and which can not be broken by reflection API ?
Reply

Barba of the Pale Horse says

MARCH 27, 2018 AT 6:12 PM

Throw an exception from private constructor should defeat serialization attack…

Reply

Barba of the Pale Horse says

MARCH 27, 2018 AT 6:17 PM

Meant to say reflection not serialization…

Reply

GauravD says

MAY 8, 2018 AT 12:02 AM

Bu if you throw exception from private constructor it will always throw excpetion and wont allow to create object of that class in any case , even single instance

Reply

Sangchual Cha says

OCTOBER 27, 2017 AT 1:57 PM

Based on the number of bytecode instructions, synchronized method would cost less than synchronized block

———————————————————————————

Java code

———————————————————————————

```
package com.sangchual.java.exercise;
public class SynchronizedPerforamnce {
Integer counter = new Integer(0) ;
public synchronized void runWithSynchronizedMethod() {
counter ++ ;
}
public void runWithSynchronizedBlock() {
synchronized (this) {
counter ++ ;
}
}
}
```

———————————————————————————

Bytecodes

——

```
~/w/j/o/p/c/c/s/j/exercise javap -c SynchronizedPerforamnce.class
Compiled from "SynchronizedPerforamnce.java"
public class com.sangchual.java.exercise.SynchronizedPerforamnce {
java.lang.Integer counter;
public com.sangchual.java.exercise.SynchronizedPerforamnce();
Code:
0: aload_0
1: invokespecial #1 // Method java/lang/Object."":()V
4: aload_0
5: new #2 // class java/lang/Integer
8: dup
9: iconst_0
10: invokespecial #3 // Method java/lang/Integer."":(I)V
13: putfield #4 // Field counter:Ljava/lang/Integer;
16: return
public synchronized void runWithSynchronizedMethod();
Code:
0: aload_0
1: getfield #4 // Field counter:Ljava/lang/Integer;
4: astore_1
5: aload_0
6: aload_0
7: getfield #4 // Field counter:Ljava/lang/Integer;
10: invokevirtual #5 // Method java/lang/Integer.intValue:()I
13: iconst_1
14: iadd
15: invokestatic #6 // Method java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
18: dup_x1
19: putfield #4 // Field counter:Ljava/lang/Integer;
22: astore_2
23: aload_1
24: pop
25: return
public void runWithSynchronizedBlock();
Code:
0: aload_0
1: dup
2: astore_1
3: monitorenter
4: aload_0
5: getfield #4 // Field counter:Ljava/lang/Integer;
8: astore_2
9: aload_0
10: aload_0
11: getfield #4 // Field counter:Ljava/lang/Integer;
14: invokevirtual #5 // Method java/lang/Integer.intValue:()I
17: iconst_1
18: iadd
19: invokestatic #6 // Method java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
22: dup_x1
23: putfield #4 // Field counter:Ljava/lang/Integer;
26: astore_3
27: aload_2
28: pop
29: aload_1
30: monitorexit
31: goto 41
34: astore 4
36: aload_1
37: monitorexit
38: aload 4
40: athrow
41: return
Exception table:
from to target type
4 31 34 any
34 38 34 any
}
```

Reply

Pankaj says
OCTOBER 27, 2017 AT 5:16 PM
`synchronized (this)`: you are locking your object itself, just like synchronized method. Create a dummy variable in the class and use that for synchronized block.
Reply

Rakesh Yadav says
DECEMBER 3, 2017 AT 6:03 AM
If it would, it would do it for the few on only one thread in that case.
Reply

Lakhan says
SEPTEMBER 6, 2017 AT 10:23 AM
How are singleton object serialized even if object reference is static?
According to serialization, static variabal or object are not serialized

Reply

Kuldeep Singh says
AUGUST 24, 2017 AT 11:24 PM
Very good explanation but I think you missed the concept of cloning, cloning can also break your singleton patten if you class is cloneable. Correct me if I am wrong.
Reply

Pankaj says
AUGUST 25, 2017 AT 12:28 AM
Yes, so don't implement Cloneable interface if you want to create a Singleton class.
Reply

Tejas says
AUGUST 5, 2017 AT 11:07 AM
Hi Pankaj,
I have a doubt in your Bill paugh Singleton implementation. You have stated that static nested class is not loaded into memory during class loading . But i have studied as static stuffs are always loaded with class initialization in JAVA. and they share with every class instance. Since in Bill paugh implemetation it is loaded only when we call getInstance method why ?
Reply

Disha says
AUGUST 17, 2017 AT 9:31 PM
The classloader doesnot differentiate whether its a inner class or not.
Our concern here is when its initialized irrespective of when its loaded.
It is initialised when it is needed inside the getInstance().
Reply

pankaj says
JUNE 29, 2018 AT 10:45 PM
INSTANCE will be creating on class loading?? I m confused about this.Please clear ..as per my understanding instance should be instantiated on class loading
Reply

Asutosh Swain says

JUNE 19, 2017 AT 11:14 AM

Good Knowledge bro

Reply

m&m says

MAY 22, 2017 AT 7:27 PM

Excellent explanation .Thank you.

Reply

Devendra Singh says

MAY 19, 2017 AT 10:48 PM

Hi Pankaj,

This is very very helpful. I appreciate your knowledge.

Reply

Murali Krishna Varanasi says

APRIL 28, 2017 AT 3:26 AM

Hi Pankaj,

Your way of explanation with examples is awesome.

Can you provide the example for "Enum Singleton".

Thanks,

Murali

Reply

rama devi says

MAY 1, 2017 AT 5:07 AM

Very neat explanation and covered many ways to create singleton class.

Thank you.

Reply

Joseph Grigg says

MAY 30, 2017 AT 10:24 AM

Based on Pankaj's Enum singleton above, you would call "doSomething()" like this:

package your.package.name;
import com.journaldev.singleton.EnumSingleton;
public class Client {
public static void main(String[] args) {
EnumSingleton.INSTANCE.doSomething(); // OR use class variable like below…
}
}
package your.package.name;
import com.journaldev.singleton.EnumSingleton;
public class Client {
private final EnumSingleton INST = EnumSingleton.INSTANCE;
public static void main(String[] args) {
INST.doSomething();
}
}
For the Enum Singleton, as it is an Enum type – you can implement but NOT extend classes, so consider that when choosing between what design to use for your Singletons.
Reply

sourab wasnik says
APRIL 25, 2017 AT 3:28 AM
helpful indeed! I got a clear idea about how can we implement singleton in different ways.
Reply

nitin says
APRIL 11, 2017 AT 9:30 PM
very good..keep posting such things
Reply

Babanna S Duggani says
APRIL 6, 2017 AT 9:33 AM
Very detailed and well explained sir, Thanks.
Reply

Neetu says
APRIL 5, 2017 AT 9:20 PM

Well formatted, well organized, & Excellent tutorial on Singleton!!

Reply

amit ghorpade says

APRIL 3, 2017 AT 8:37 PM

Good explanation. Examples helped to understand each point

Reply

Janaksinh M says

MARCH 26, 2017 AT 1:24 AM

Thank you..!!

Reply

Anupam Tamrakar says

MARCH 23, 2017 AT 3:41 AM

Awesome tutorial to get started with design patterns

Thanks a lot !

Reply

sasi says

MARCH 6, 2017 AT 10:28 AM

Hi,

I have written the simple program like below. It is not throwing any error even though I am creating object again in main method. Please let me know if we are able to create object again in main method what is the use of singleton.

package com.java.designpatterns;

public class SingletonPattern {

private static final SingletonPattern singletonPattern = new SingletonPattern();

private SingletonPattern(){}

public static SingletonPattern getInstance(){

return singletonPattern;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

SingletonPattern singletonPattern = new SingletonPattern();

singletonPattern.getInstance();

System.out.println("Hi");
}
}
Reply

sai kumar says
MARCH 23, 2017 AT 12:52 AM
Dear Sasi,
You are creating the object of SingletonPattern in the same class hence you can access the private constructor which you have created in the same class. You should create a new test class to test whether it is allowing you to create a new object for SingletonPattern.
Try This,
class SingletonPattern {
private static final SingletonPattern singletonPattern = new SingletonPattern();
private SingletonPattern(){}
public static SingletonPattern getInstance(){
return singletonPattern;
}
}
Test class:
class Test
{
SingletonPattern singletonPattern = new SingletonPattern();// Error, since the constructor is private
singletonPattern.getInstance();
System.out.println("Hi");
}
Reply

Balaji says
FEBRUARY 23, 2017 AT 6:42 AM
Gotcha with method :
public Object readResolve()
{
return getInstance();
}
Only works in Single main thread. if you write serialized Singleton object in first program. In second program if you deserialize and try to readObject() then to get Singleton Object, But it will give you NULL object.
So this method won't work across program or jvm
Reply

Kanij Sheikh says

FEBRUARY 20, 2017 AT 11:51 PM

Best tutorial for design patterns!

Reply

Prashant Bagade says

FEBRUARY 18, 2017 AT 1:42 AM

It is best tutorial i have seen ever for design pattern……..

Reply

Anupkumar Mahamalla says

FEBRUARY 14, 2017 AT 4:05 AM

Initially i was surprise with the number of comments.

but could not stop myself to write this comment, It is really good article for singleton design pattern.

Reply

satya says

FEBRUARY 17, 2017 AT 2:42 AM

its Awesome explanation . it will be good if u add more lectures on

protected Object readResolve() {

return getInstance();

}

Reply

Eddy says

JANUARY 8, 2017 AT 2:04 AM

The following code doesn't work, please help, thanks!

————————————————————————————————————

public class SerializedSingleton implements Serializable{

private static final long serialVersionUID = -7604766932017737115L;

private SerializedSingleton(){}

private static class SingletonHelper{

private static final SerializedSingleton instance = new SerializedSingleton();

}

public static SerializedSingleton getInstance(){

return SingletonHelper.instance;

}
protected Object readResolve() {
return getInstance();
}
}
———————————————————————————————————-
Thanks!
Reply

Mehdi MAHMOUD says
JANUARY 12, 2017 AT 4:20 AM
Try to add throws ObjectStreamException to your readResolve method
Reply

Manish Kumar Prasad says
DECEMBER 24, 2017 AT 11:40 AM
even this giving 2 different objects.
Reply

AUR Dogar says
JANUARY 6, 2017 AT 11:08 PM
Good but not video lecture
Reply

umamaheswararao says
DECEMBER 30, 2016 AT 1:45 AM
Excellent explanation . Even beginner will understands your explanation.
Keep it up.
Reply

priyanka says
DECEMBER 14, 2016 AT 9:45 AM
very useful article .Easy to understand .
Reply

prakash agarwalla says
NOVEMBER 20, 2016 AT 10:58 PM
Bill Pugh Singleton Implementation, how it will resolve the double locking issue? if two threads accessing it first time ?
Reply

Keval Solanki says
OCTOBER 18, 2016 AT 11:11 PM
Very well explained! Hats off
Reply

kirubakaran says
OCTOBER 4, 2016 AT 10:41 PM
Hi Guys,
How to use singleton in serialization…? & Which method we can use for singleton…?
Thanks,
Kirubakaran
Reply

Anil kumar says
SEPTEMBER 20, 2016 AT 12:03 PM
It's really great and simple article to understand. Thank you very much.
Reply

Charlie says
SEPTEMBER 8, 2016 AT 10:47 PM
Thank you for you a series of greate article. May I ask why bill pugh singleton implementation is thread safe? Thnaks
Reply

GOPINATH M B says
SEPTEMBER 28, 2016 AT 10:18 AM
It is because SingletonHelper class will be loaded only when we refer SingletonHelper.instance;

and as class loading is thread-safe, static members initialization also happens only once and hence only one object is created though multiple threads call at the same time.
Since classsloader initializing the member, it is thread safe.
Reply

Vishwas Tyagi says
AUGUST 4, 2016 AT 10:04 AM
Write about clone method also.
Reply

Brijesh Baser says
AUGUST 26, 2016 AT 3:38 AM
Hi Vishwas,
If we are not implement singleton class with Cloneable interface then no one can create a clone object of the Singleton class.
If anyone trying to do clone() then it will throw CloneNotSupportedException.
Reply

Vishwas Tyagi says
JANUARY 7, 2017 AT 6:32 AM
Thanks, Brijesh.
Reply

Sujeet says
JULY 25, 2016 AT 12:23 AM
In that case we never have lazy initialization with thread safe singleton ?
Reply

Rakesh kumar Swain says
JULY 15, 2016 AT 4:48 AM
Hi Pankaj,
We can restrict the object creation by using Reflection by putting below piece of code in Constructor.
if(SingletonHelper.INSTANCE != null){
throw new RuntimeException("Singleton already initialized");

}
Or
if(!Thread.currentThread().getStackTrace()[2].getClassName().equals("BillPughSingleton")){
throw new RuntimeException("Singleton already initialized");
}
Can you please confirm Is this a correct approach to make a class singleton?
Reply

Krishna Chowdary Garapati says
DECEMBER 27, 2016 AT 10:00 PM
This is not a singleton. Because it will not return same object instead it throws exception. But that is not what singleton implementation.
Reply

Brijesh Baser says
JUNE 28, 2016 AT 3:49 AM
I have written the singleton with the use of AtomicReference, for first few thread create multiple instance but getInstance() always return single object. For some senario we can use this also, there is no synchronization used.
import java.util.concurrent.atomic.AtomicReference;
public class SingleTon {
private static AtomicReference ref = new AtomicReference();
private SingleTon(){
}
public static SingleTon getInstance(){
SingleTon singleTon = ref.get();
if(singleTon == null){
SingleTon singleTon2 = new SingleTon();
if(ref.compareAndSet(null, singleTon2)){
return singleTon2;
}else {
singleTon = ref.get();
}
}
return singleTon;
}
public void test(){
System.out.println("SingleTon.test()*******");
}
}
Reply

Pankaj says
JUNE 28, 2016 AT 6:32 AM
It's almost similar to double locking singleton method. There also synchronized is inside the if null check, so it will be used only once.
Reply

Sameer Patel says
JUNE 17, 2016 AT 2:45 AM
Reference variable of any singleton object would be Static type and as per my knowledge we can not serialize Static object. My question is how are we serializing an Static object??
Reply

Lalit says
JUNE 8, 2016 AT 6:55 PM
Hi Pankaj,
Very nice article, covers all of the scenarios and approaches for creating Singleton object.
Thanks!
Reply

panky031 says
JUNE 3, 2016 AT 7:35 AM
Hi Pankaj,
Can you please tell me Singleton example in jdk.
Reply

Rizwan says
JUNE 1, 2016 AT 4:33 AM
Pankaj,
Very informative article.
I think all the drawback of singleton can be eliminated by using ENUM ie , serialization, synchronization and reflection.
Reply

Sindhura says
MAY 25, 2016 AT 12:08 AM
Simple and perfect demonstration !!
Reply

Rajeev says
MAY 24, 2016 AT 4:36 AM
I think a pattern made of "Double checked Locking(DCL) with Volatile keyword" exists which has the features of Lazy loading, Multithreading and Singleton . It's considered better than Bill pugh Singleton Implementation as Bill pugh's Implementation is considered lazier than lazy Initialization! Please say a word on the difference between Bill pugh Implementation and DCL with volatile keyword.
Reply

shailesh singh says
APRIL 18, 2016 AT 7:11 PM
thanks very good series of examples at single place,
Reply

Dndnnz says
APRIL 18, 2016 AT 6:58 PM
When we read your website in mobile then gmail linkdin share option cover the page Content also please we resolve this problem because we due to this we are not able read website content properly.
Reply

Pankaj says
APRIL 19, 2016 AT 1:31 AM
can you take a snapshot and email me? the share icons in mobile are stick to the end of screen as horizontal bar and don't cause any issue while reading article. Can you tell me which mobile you are using, so I can check?
Reply

Surya says
MARCH 24, 2016 AT 4:38 AM
Excellent writing, easy to understand. Thank you so much.

Reply

Nagarjuna Aluru says

MARCH 16, 2016 AT 10:46 PM

It's awesome … very easy to understand

Reply

mehdi says

FEBRUARY 23, 2016 AT 7:55 AM

Thank you very much

which is better?

Reply

Harish Kumar Tharala says

NOVEMBER 20, 2015 AT 12:11 AM

Hi Pankaj,

Great and nice Explanation.Easy to understood for beginners and experienced people.

Reply

siva kumar says

NOVEMBER 17, 2015 AT 10:34 PM

i was not getting about enum singleton could you please give an example program

Reply

Anurag says

OCTOBER 19, 2015 AT 1:16 AM

Quite a informative article and nice writing style. Keep it up.

Reply

Raja says

OCTOBER 13, 2015 AT 8:37 AM

What about multiple classloaders

Reply

Amsidh Lokhande says
SEPTEMBER 17, 2015 AT 10:45 AM
This is very clear cut representation of Singleton Design Pattern.
Thank you very much Pankaj.
Reply

Mohit Garg says
SEPTEMBER 17, 2015 AT 3:36 AM
Very smooth and incremental explanation Pankaj. Great work
Reply

Vasudha Singal says
SEPTEMBER 16, 2015 AT 5:05 AM
It is an extremely informative article !!
Reply

Himansu says
SEPTEMBER 11, 2015 AT 4:24 AM
It is a very nice post for singleton design pattern with practical example for all the different kind of scenario.
Thanks Pankaj.
Reply

Prasann Mishra says
AUGUST 28, 2015 AT 7:54 AM
Provided explanation is excellent but the same time this code is not covering the clone scenario. It break the code.
Reply

Ravi says
AUGUST 10, 2015 AT 12:22 AM
Why Bill Pugh Singleton Implementation doesn't require synchronization ?
Reply

Arvind Ajimal says
SEPTEMBER 23, 2015 AT 10:14 AM
Because in any tyqe of environment at any instance of time only one thread will make the call to getInstance() method and if it is not the very first call (which will load the inner class and therefore create the INSTANCE first time) the method returns already loaded same INSTANCE. In bill's class getInstance() method has only one task – to return the INSTANCE.
Reply

John says
APRIL 18, 2016 AT 11:56 PM
Hi,
In Bill Pugh implementation, the approach is good, but what he has done is he created the instance as final to make it constant. samething we can do it in eagerly loaded or static initialization or even in synchronised one. what do u say.
Reply

Bharat says
JULY 29, 2015 AT 9:23 PM
This is one of the excellent explanation for the Singleton design pattern. Thanks Pankaj for details explanation.
Reply

Rasheed says
JULY 27, 2015 AT 10:18 AM
Hi Pankaj,
Awesome article. Minor correction, private constructor was missed in Enum singleton.
Thanks,
Rasheed
Reply

Arvind Ajimal says
SEPTEMBER 23, 2015 AT 10:19 AM
There is no need of that. Enum definition. Java 5 onwards– Only one instance will be there like in the above examle the instance named as INSTANCE. you can use any other name e.g.
public enum EnumSingleton {
ONLYTHIS;

```
public static void doSomething(){
//do something
}
}
```

Reply

Manish Kumar says

OCTOBER 28, 2015 AT 3:10 AM

By default Enum constructor are private in java.

Reply

Gayatri says

JULY 25, 2015 AT 3:31 PM

Very nice article with lot of scenarios covered. Thanks.

Reply

Suresh reddy says

JULY 21, 2015 AT 5:49 AM

your article gave me great knowledge comparing to other..thanks & keep it up

Reply

Rishi says

JUNE 25, 2015 AT 4:58 AM

All the approached greatly explained along with concern issues and benefits. Awesome article. Pankaj I personally appreciate your understanding. Best ever I had seen a Singleton article. Keep it up!!

Reply

hamid says

JUNE 15, 2015 AT 3:23 AM

Instance==null
Why you are doing in synchonize contex

Reply

Arvind Ajimal says
SEPTEMBER 27, 2015 AT 9:15 AM
Very Simple. Because if by this time the instance has not been created i.e ==null is true and lets say it is the very first thread executing the code, the thread has the lock (…is inside synchronized block) it will create the instance. And if this not the very first thread, lets say it is the 2nd thread that now has acquired the lock it won't create the instance instead it returns the already created one.
Reply

Rasmita says
JUNE 15, 2015 AT 12:18 AM
Very very nice explanation . I would have searched so many blogs but i find the best,this is yours. Thanks a lot.
Reply

Kiran Panda says
JUNE 12, 2015 AT 2:21 AM
Your the messiah for Java developers.
God bless you.
Reply

Sivamurugan Subramaniam (Siva) says
JUNE 10, 2015 AT 5:00 AM
Dear Pankaj,
I always read your technical blogs, this is awesome explanation with all type of implementation, which really useful in long scale project. thank you so much for writing. keep it up.
God bless you.
Thanks,
Siva
Reply

Anil says
JUNE 8, 2015 AT 10:59 PM
Best example i ever got in whole internet.
thanks
Reply

Harriesh says
MAY 8, 2015 AT 11:02 PM
Awesome article !!!!!!!!
Reply

kinglee says
MARCH 23, 2015 AT 5:18 AM
Cool examples, Nice one, thanks for keeping it clean and simple
Reply

neeraj goyal says
MARCH 17, 2015 AT 8:53 PM
Really helpful post. Learnt basic concepts for singleton.
Reply

Swapnil says
MARCH 3, 2015 AT 4:23 PM
Awesome and complete article about singleton..Thanks a lot !!
Reply

Priyanka says
FEBRUARY 15, 2015 AT 10:53 PM
public static ThreadSafeSingleton getInstanceUsingDoubleLocking(){
if(instance == null){
synchronized (ThreadSafeSingleton.class) {
if(instance == null){
instance = new ThreadSafeSingleton();
}
}
}
return instance;
}
This won't work in case if initializing ThreadSafeSingleton is a heavy process and takes time to initialize.. any other thread may get half initialized object as the first check itself will say that ThreadSafeSingleton

!= null.
In a thread safe environment you should get or set the object by acquiring the lock.
For the rest of the article, it is very good…
Reply

Paras says
JANUARY 29, 2015 AT 8:51 PM
Good Work Dude…
Reply

Anuj Pankaj says
JANUARY 28, 2015 AT 2:32 AM
I like your posts. This one is really very nice.
Reply

Daniel Lim says
JANUARY 25, 2015 AT 7:22 AM
Thanks for post.
 It's very helpful of me.
Reply

Surya says
JANUARY 10, 2015 AT 10:18 AM
Good tutorial about singleton.I always get lot of confusion about this.Thanks for clarifying it out.
Reply

indrjeet kumar says
DECEMBER 31, 2014 AT 6:18 AM
Hi Pankaj,
The above article was very good.
You have covered all the case for the implementation .
Are you planning to crate a vedio also.
Wish you a Happy new year and all the best
Regards,
Indrjeet Kumar

Reply

Anoop says

DECEMBER 23, 2014 AT 10:03 PM

if (instance == null) {

synchronized (ThreadSafeSingleton.class) {

if (instance == null) {

instance = new ThreadSafeSingleton();

}

}

}

This double checked locking code won't work. If you write this code in Netbeans/ Intellij IDEA, it will show warning. Try to avoid this code.

Reply

Neeraj says

DECEMBER 11, 2014 AT 6:21 PM

ur examples are too good as usual

Reply

subrahmanyam says

DECEMBER 9, 2014 AT 10:28 PM

Wonderful article. Thanq.

Reply

sudheer.maguluri says

NOVEMBER 28, 2014 AT 10:39 AM

here concept is good .but implementation is not that much good .pls try to give good stuff

Reply

Pankaj says

NOVEMBER 28, 2014 AT 8:51 PM

I am not sure what else you are looking in the implementation examples, as far I see the comments here, people are liking it.

Reply

Phool Chand Nishad says
JULY 2, 2016 AT 4:48 AM
excelent explanation of singletone desine patteron
Reply

Pawan Bhawsar says
NOVEMBER 21, 2014 AT 7:43 PM
Hello Pankaj,
Like serialization, cloning can also destroy Singleton Pattern? Do you have any example or scenario to prevent object cloning on singleton class ?
Reply

Arvind Ajimal says
SEPTEMBER 27, 2015 AT 9:32 AM
To be able to clone an object it must (or a super class of it) specifically implement Cloneable interface. So generally if you are designing you won't use Cloneable if you are creating a Singleton. But lets say your singleton is extending from a super class (that is the case some time) and if that super class has a public clone() method, then you should override it and make sure that it throws CloneNotSupportedException.
Reply

Anup Kumar Shrivastava says
NOVEMBER 21, 2014 AT 6:24 PM
How distributed systems creates only one object where multiple clusters were involved
Reply

suman says
DECEMBER 4, 2014 AT 3:05 AM
I am not sure my answer was good or not, but still I wanted to share my thoughts.
As multiple clusters involved I hope multiple JVMS will get involved one for each cluster. as there is no way the other JVM know about the object on the other one. so I think it is not possible to achieve what you are thinking.
Reply

Mohan says
JANUARY 15, 2015 AT 5:42 PM
Hi Anup,
you mean, how distributed systems uses only one object? if it is your question, i think we can serialize the object and share those object into distributed systems.
Reply

SivaPrasad says
NOVEMBER 14, 2014 AT 6:46 AM
Can you please add a scenario how to prevent breaking Singleton using Reflection with an example.?
Reply

Ramakrisha says
NOVEMBER 5, 2014 AT 1:30 AM
its good
Reply

Abhijeet Kedare says
OCTOBER 31, 2014 AT 1:13 AM
Joshua Bloch also suggested , in case of SerializedSingleton we need to make all instance variables transient along with implemention of readResolve() method
Reply

Preeti says
OCTOBER 21, 2014 AT 5:32 AM
Great article
Reply

sumit says
OCTOBER 20, 2014 AT 1:31 AM
AWESOME
Reply

Sridhar says
OCTOBER 5, 2014 AT 9:17 AM
Thanks for your blog, i have a doubt. actually not clear with, where and how to use below:
can you please explain with example?
protected Object readResolve() {
return getInstance();
}
Reply

Priyank says
JULY 27, 2015 AT 11:56 AM
To avoid the object creation while de-serialization process.
Reply

Manish Kumar says
OCTOBER 28, 2015 AT 3:29 AM
Can you please explain with an example?
How to avoid with this?
protected Object readResolve() {
return getInstance();
}
Reply

Manav says
SEPTEMBER 30, 2014 AT 2:21 PM
Great Article Buddy.. Thanks !!
Reply

lokesh says
SEPTEMBER 26, 2014 AT 12:21 AM
if i call clone() method on Singleton object was it create new object or return existing one
if i load a class, using classloaders, is Singleton class has same behavior?
Could please explain on above….
Reply

suraj says
OCTOBER 7, 2014 AT 5:15 AM
try to implement the method with cloneable interface and override clone method by throwing clonenotsupported exception, so that the object will not be cloneable as well.
Thanks.
Reply

Hari says
OCTOBER 13, 2014 AT 6:24 AM
Implement Cloneable interface and Override clone method then return same object like the following:

```
class MySingleton implements Serializable,Cloneable{
private static MySingleton mySing = new MySingleton();
private MySingleton(){ }
public static MySingleton getinst(){
return mySing;
}
//Solution for Serialization
protected Object readResolve() {
return getinst();
}
@Override
protected Object clone() throws CloneNotSupportedException {
return getinst();
}
}
```

Note: It also works for Serilization
Reply

Arvind Ajimal says
SEPTEMBER 27, 2015 AT 9:35 AM
Dear Lokesh, see this link
http://blog.yohanliyanage.com/2009/09/breaking-the-singleton/
Reply

Naga says

SEPTEMBER 23, 2014 AT 7:47 PM

The synchronized block of your code creating multiple objects in multi threaded programming

Reply

Naga says

SEPTEMBER 24, 2014 AT 11:37 PM

Sorry for post my previous info.it's working fine

Reply

MohanV says

SEPTEMBER 18, 2014 AT 2:51 AM

nice post..everything is discussed here…

Reply

Prashant Shahapur says

SEPTEMBER 6, 2014 AT 5:45 AM

in Singleton Class implements Serilizable, I added readResolve() method but still serialized object and deserialzed objects hashcodes are different…Please confirm me

Reply

kush singh says

AUGUST 24, 2014 AT 10:06 AM

Hi pankaj,
This is a nice article and almost everything is covered. it help me lot to understand singleton

Reply

tadveer Verma says

AUGUST 7, 2014 AT 6:13 AM

its nice tutorial ……………………

Reply

Adyasha Nayak says

AUGUST 6, 2014 AT 11:16 PM

@Anuj Agarwal
declaring INSTANCE as a final object restricts re – assignment of a new instance to it.
final Singleton INSTANCE = new Singleton();
INSTANCE = new Singleton(); // this won't work
Reply

Nishant Kshirsagar says
AUGUST 1, 2014 AT 5:45 AM
Hi Pankaj,
Thanks for the *elaborate* explanations.
I have a suggestion for you:
Can you please add a scenario whereby cloning of the Singleton object is explained with an example. It would definitely help to understand different situations where Singleton can be used.
Best Regards.
Thanks,
Nishant Kshirsagar
Reply

Zaid says
JULY 20, 2014 AT 11:55 PM
Great!!!!!
Reply

Kapil Agarwal says
JULY 31, 2014 AT 5:06 AM
I love reading your tutorials. Great explaination !!!!
Reply

Shashi says
JULY 19, 2014 AT 7:21 AM
Awesome explanation and very clear thanks a ton to pankaj
Reply

Vinodini Vikram says
JULY 16, 2014 AT 7:12 PM

This tutorial gave me good clarity on Singleton Pattern. Thank you.
Reply

Priyanka says
JULY 8, 2014 AT 5:33 PM
This is the best one stop post for singleton pattern i've come across.I had to give a training and this post gave me everything to discuss.
Thanks a ton Pankaj for your amazing work.
Reply

Akhil says
JULY 3, 2014 AT 12:35 PM
Hi Pankaj,
your articles are very help to build concepts.
Could you please explain more about this statement of Bill Pugh approach: "This is the most widely used approach for Singleton class as it doesn't require synchronization."
Why this approach doesn't require synchronization?
Reply

Pankaj says
JULY 9, 2014 AT 1:15 AM
Because when you will call getInstane() method, the class will be loaded first in memory and an instance will get created. So even if multiple threads call getInstance() simultaneously, the class gets loaded only once and you dont need synchronization.
Reply

Anuj Agarwal says
AUGUST 1, 2014 AT 10:49 PM
It means, in Bill Pugh Singleton Implementation approach, we dont need final keyword as used in below:
private static final BillPughSingleton INSTANCE = new BillPughSingleton();
It will work fine in all scenarios even if we use it like :
private static BillPughSingleton INSTANCE = new BillPughSingleton();
Reply

Jeegar Kumar says
JULY 27, 2015 AT 5:56 AM
You must declare INSTANCE as final for surety.

If you don't then it will allow to change the value of INSTANCE in getInstance () method.
One can easily change the value for it in your getInstance () method as per below:
SingletonHelper.INSTANCE = null; // or by any value
So it is much sure for one to declaring it as final.
Reply

micheal says
JULY 1, 2014 AT 10:29 PM
Thanks for your wonderful article
Reply

Pavan says
JUNE 24, 2014 AT 2:07 AM
Hi Pankaj,
it is very nice post….one of the best among all intenet post on singleton
Reply

RAJ9992 says
JUNE 6, 2014 AT 12:06 AM
Really Nice that Sharing your Knowledge to e-learners
Reply

Amasa says
MAY 27, 2014 AT 5:11 AM
Very nice post…It was vry helpfull
Reply

Sumedh says
MAY 10, 2014 AT 2:45 AM
Awesome post Pankaj…..This post is really very helpful…..Thank you
Reply

Neha says
APRIL 30, 2014 AT 4:53 AM
All the articles are very supporting. thanks
Reply

Jim Halvorsen says
APRIL 15, 2014 AT 1:56 PM
If you are using an enum as a singleton (only one enum constant "INSTANCE"), it is lazy loaded. If there are multiple enum constants, then they are all created at first access of any constant.
Reply

Likhit says
APRIL 14, 2014 AT 7:46 PM
Awesome Work buddy !!!
Reply

Velmurugan A says
APRIL 7, 2014 AT 7:38 PM
Is Reflection breaks Bill Pugh Singleton Implementation?
I tried it does not create new instance, it throws IllegalArgumentException
But using serialization it breaks Bill Pugh Singleton Implementation. To avoid this we must use readResolve() method.
Is it right? if i did mistake please mention it….
Reply

Pankaj says
APRIL 7, 2014 AT 10:12 PM
Yes Reflection will break Bill Pugh implementation by making constructor accessible.
Reply

Velmurugan A says
APRIL 8, 2014 AT 9:03 PM
thanks…
Reply

Velmurugan A says

APRIL 8, 2014 AT 9:09 PM

Please look at this below code…… and tell the reason why it's throwing IllegalArgumentException

```
import java.io.Serializable;
public class Singleton implements Serializable{
private static final long serialVersionUID = 2L;
private Singleton(){
System.out.println("Singleton");
}
public static Singleton getInstance(){
return Instance.s;
}
private static class Instance{
private static Singleton s = new Singleton();
}
protected Object readResolve(){
return Instance.s;
}
}
```

——————————————————————————————————————-

```
import java.lang.reflect.Constructor;
public class Test {
public static void main(String[] args) throws Exception {
Singleton s1 = Singleton.getInstance();
Singleton s2 = null;
try {
Constructor[] constructors = Singleton.class.getDeclaredConstructors();
for (Constructor constructor : constructors) {
constructor.setAccessible(true);
s2 = (Singleton) constructor.newInstance();
break;
}
} catch (Exception e) {
e.printStackTrace();
}
System.out.println(s1);
System.out.println(s2);
}
}
```

———-

output:

```
Singleton
java.lang.IllegalArgumentException: wrong number of arguments
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at
```

sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
at java.lang.reflect.Constructor.newInstance(Constructor.java:408)
at com.vel.Test.main(Test.java:14)
com.vel.Singleton@106d69c
null
———
Note : if i remove readResolve() in Singleton class it does not throw any Exception. but I put this method it is throwing IllegalArguementException
Reply

Vikram says
APRIL 10, 2014 AT 1:32 PM
Velmurugan, the method readResolve() works only when the object is read from Stream. This is not the case when you're using Reflection to destroy the Singleton pattern.
Reply

Velmurugan A says
APRIL 11, 2014 AT 8:10 PM
but I am using readResolve() method, without implements Serializable interface at the time Reflection throws same Exception.
Is Reflection internally create an Object using Stream? or Internally any concept is going on in readResolve() method?
Please help me….. to understand

Valerio P says
MAY 9, 2014 AT 2:30 PM
Hello, i copy/paste your code in my Eclipse. It works good and no IllegalArgumentException was launched
Reply

Prabhavathi says
APRIL 3, 2014 AT 5:25 AM

Can you please explain the Singleton enum prevents lazy loading ?I feel that the instance creation happens lazily when it is first accessed.
Reply

Pankaj says
APRIL 3, 2014 AT 3:50 PM
Enum variables are static by default, so as soon as you load an Enum all the variable instances will get loaded too. That's why Enum can't be used for lazy loading.
Reply

Durga says
APRIL 16, 2015 AT 10:48 PM

```
public enum SingletonEnum {
INSTANCE;
public static final String Id="0″;
private SingletonEnum() {
System.out.println("SingletonEnum initialized");
}
public void execute (String arg) {
// Perform operation here
}
}
class Snippet {
public static void main(String[] args) {
System.out.println(SingletonEnum.Id);
}
}
```

Example to say enum approach is lazy loaded.
Reply

Mohan says
MARCH 28, 2014 AT 4:29 AM
Hi Pankaj,
Really nice explanation.
Could you please let us know the other scinarios also where singleton is not a singleton.
like, cloning, how distributed systems creates only one object where multiple clusters were involved, etc…
Thanks in advance,….
Reply

Rajesh says
MARCH 22, 2014 AT 8:41 AM
Very nice, explained in simple terms.
Reply

DHARMENDRA KUMAR SAHU says
MARCH 13, 2014 AT 3:23 AM
Very nice tutorial.
It help me lot to understand the singleton.
Could you please help me how protected Object readResolve() method make i class a singleton.
Reply

Anshul Jain says
JANUARY 22, 2014 AT 10:15 AM
This is a nice article and almost everything is covered. Thanks a ton.
Reply

Prakash says
JANUARY 6, 2014 AT 12:41 PM
Very good site for design patterns. Explanation is easy and up to the mark. Thanks.
Reply

H Singh says
NOVEMBER 20, 2013 AT 11:10 AM
Hi Pankaj,
Can we create more than one object of a singleton class using serialization ?
Please provide your view in detail about in which condition Singleton classes can have more than one objects.
Thanks in advance.
Regards,
H Singh
Reply

Pankaj says

NOVEMBER 20, 2013 AT 7:34 PM

Yes we can and thats why we can have readResolve() method implementation to get the same instance.

Reply

H Singh says

NOVEMBER 21, 2013 AT 7:11 AM

What is the purpose of readResolve() method? how it helps in maintaining only one object of singleton class.

Reply

Taufique Alam says

DECEMBER 19, 2013 AT 5:47 PM

can you tell in which class we can find viewResolve().

Reply

Vishnu says

NOVEMBER 7, 2013 AT 1:13 PM

This web site useful for me, Thanks a ton.

Reply

Vishnu says

NOVEMBER 7, 2013 AT 1:12 PM

Sir,
As per u provide contains which one is the best practices for the Singleton Pattern.
Regards
Vishnu

Reply

Pankaj says

NOVEMBER 11, 2013 AT 4:24 PM

It depends on the situation, if your singleton is not heavyweight I would prefer pre-initialization but if you want lazy init, then I prefer helper inner class or Enum for singleton.

Reply

Vikki says

APRIL 24, 2017 AT 3:52 AM

But you'd written that enum Singleton is not lazy loaded. So can you say enum singleton is lazy or not?

Reply

Tarun Soni says

OCTOBER 14, 2013 AT 7:16 AM

nice tutorial thanks …..

i have one question fro you

How annotations works internally?

Reply

erick says

AUGUST 9, 2013 AT 7:21 PM

Very nice post:-) keep up the good work

Reply

poonam says

MAY 8, 2013 AT 7:22 PM

owesome post…evrything is discussed about singleton.keep it up….nd thanks

Reply

Pankaj says

JUNE 21, 2013 AT 8:54 PM

Thanks Poonam, glad you liked it.

Reply

rajendra says

JULY 18, 2014 AT 4:21 AM

what is difference between singleton and static method

Reply

Tejas says

AUGUST 5, 2017 AT 11:08 AM

singleton is a design pattern and static is a key word in JAVA

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

Java EE Tutorials