Singleton Design Pattern in Java

Hi every one, Most of Java learners might feel some complexity to understand the single ton Design pattern in java. Even most of the Java professional will fail to debug the issues in singleton design pattern. So as Being a Java learner it is tough to understand the singleton design pattern, how it is internally implementing and its functionality. In recent times in most of interviewers are arising a minimum of 2 or 3 questions from STDP. So as being a part of my preparation, I would like to share my notes to all you Java Lovers, which I have written in my own simple words.

Coming to concept, Firstly what is Singleton design pattern??

Singleton Design is a part of GOF(Gang of Four) design pattern which ensures that **only one object** of a particular class is created. All further references to the objects of singleton class refer to the same instance, providing Global access to that class.

In simple words, singleton class is one which ensures to create **only one object,** and **remaining** all classes involved in it will be referred by same object (instance) reference.

Now, we will see How to create or implement singleton design pattern

1. Singleton class is that, which we need to define **Constructors as private,** which will restrict instantiation of that class from outside. This means the Private constructor which we defined will prevent anybody else to create object of that Singleton class. That means we can't create more than one object out here.

2. Need to declare **private static variables** which will hold the instance of that class.

Here question arises why we need to declare static variables only to hold instance of that class.

Answer:

As we all know that for static variables, only once memory will be created, so once we assign the reference of object to that, Further it won't allow us to assign one more object reference which is desirable one for us to create singleton class

3. Need to define **Public static methods** that return the single instance of the class.
   Mean as I said **remaining** all classes involved in it will be referred by same object (instance) reference, these public static methods will provide global access to client class(remaining class) to get referred by main class reference .

Example:

```
/**
 * @author Harish
 *
 */
public class singleton {
	public static void main(String[] args) {
		Hello h = Hello.getHello();
		System.out.println(h);                          // Hello@1fc4bec
		System.out.println(Hello.getHello());       //  Hello@1fc4bec
		System.out.println(Hello.getHello());       //  Hello@1fc4bec
	}
}
class Hello{
	private static Hello h= new Hello();
	private Hello(){}
	public static Hello getHello(){
		return h;
	}


}
```

Note: Observe every class is sharing the same reference.

==========

Next we will see forms of singleton design patterns .There are two forms of singleton design pattern

EagerInstantiation: creation of instance at load time.

Lazy Instantiation:creation of instance when required.

Firstly we will see Lazy Instantiation concept which has complexity relating to Threads compare to Eager instantiation, in which eager instantiation has no issues regarding Threads

Lazy Instantiation: As I said lazy instantiation is all about creation of instance when required means with lazy initialization you will create instance only when it is needed and not when the class is loaded. So you escape the unnecessary object creation.

For Example Program: 2

```
/**
 * @authorHarish
 *
 */
publicclass singleton {
	publicstaticvoid main(String[] args) {
		Hello obj = Hello.getHello();
		System.out.println(obj);
		System.out.println(Hello.getHello());
		System.out.println(Hello.getHello());
	}
}
class Hello{
	privatestatic Hello obj;
	private Hello(){}

	publicstatic Hello getHello(){
		if (obj == null){
			synchronized(Hello.class){
				if (obj == null){   ----------------------→ 2
			obj = new Hello();-→3//instance will be created at request time
				} }
		}	return obj;
	} }
```

See the arrow mark No.2 line

Here obj is null, so Hello class object has not yet been instantiated, even though main class is loaded.

But check arrow mark no.3 line

Here Lazy instantiation refers to the fact that a class is not instantiated until a request is made via the Hello.getHello() method. Here is a requirement to create to a object so it is instantiated with the new operator.

Now question arises, why you used **synchronized**keyword there .As I said in Lazy Instantiation there is some complexity regarding in Multithreading.

Consider from exam no: 2

We will try to call getHello() method

System.*out*.println(Hello.*getHello*()); //Hello@1fc4bec

System.*out*.println(Hello.*getHello*()); //Hello@1fc4bec

then at the first time only an instance will be created. During second time onwards for all subsequent calls we will be referring to the same object and the getHello() method returns the same instance of the Hello class which was created during the first time as shown in output .. check above output

The code no:2 works absolutely fine in a single threaded environment and processes the result faster because of lazy initialization.

When the same code works in multithreaded environment then the issue arises.In the multithreading environment to prevent each thread to create another instance of singleton object and thus creating concurrency issue we will need to use locking mechanism. This can be achieved by synchronized keyword which doesn't allow the multiple threads access concurrently means all threads at same time .

So this means that every time the getHello() is called it creates additional object which is undesirable for us . To prevent this expensive operation we will use double checked locking so that the synchronization happens only during the first call and we limit this expensive operation to happen only once

Double checked locking:

Imagine that multiple threads come concurrently and tries to create the new instance. In such situation the may be three or more threads are waiting on the synchronized block to get access. Since we have used synchronized only one thread will be given access. All the remaining threads which were waiting on the synchronized block will be given access when first thread exits this block. However when the remaining concurrent thread enters the synchronized block they are prevented to enter further due to the double check .Since the first thread has already created an instance no other thread will enter this loop.

All the remaining threads that were not lucky to enter the synchronized block along with the first thread will be blocked at the first double check. This mechanism is called double checked locking

EagerInstantiation:

As I said eager instantiation is all about creation of instance at load time, if there is a requirement to always need an instance,we can switch to eager initialization, which always creates an instance

For example: code no: 3

```
public class Singleton {

private static final Singleton INSTANCE = new Singleton();
```
always create objects whenever class loads

```
private Singleton() {}

public static Singleton getInstance() {

return INSTANCE;

   }

}
```

This method has a number of advantages:

The instance is not constructed until the class is used.

There is no need to synchronize the getInstance() method, meaning all threads will see the same instance and no (expensive) locking is required.

The final keyword means that the instance cannot be redefined, ensuring that one (and only one) instance ever exists.

Advantage of Singleton design pattern

Saves memory because object is not created at each request. Only single instance is reused again and again.

Coming soon I will make a note using real time examples for Singleton Design pattern

Hope this notes will useful to you all Java Lovers!!!!

My Humble don't copy this to anywhere. I am providing this a part of my small help to you all. Hope all you guys will consider my words

Regards

Harish Reddy