Singleton design pattern in Java

Singleton Pattern says that just**"define a class that has only one instance and provides a global point of access to it".**

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

There are two forms of singleton design pattern

- **Early Instantiation:** creation of instance at load time.
- **Lazy Instantiation:** creation of instance when required.

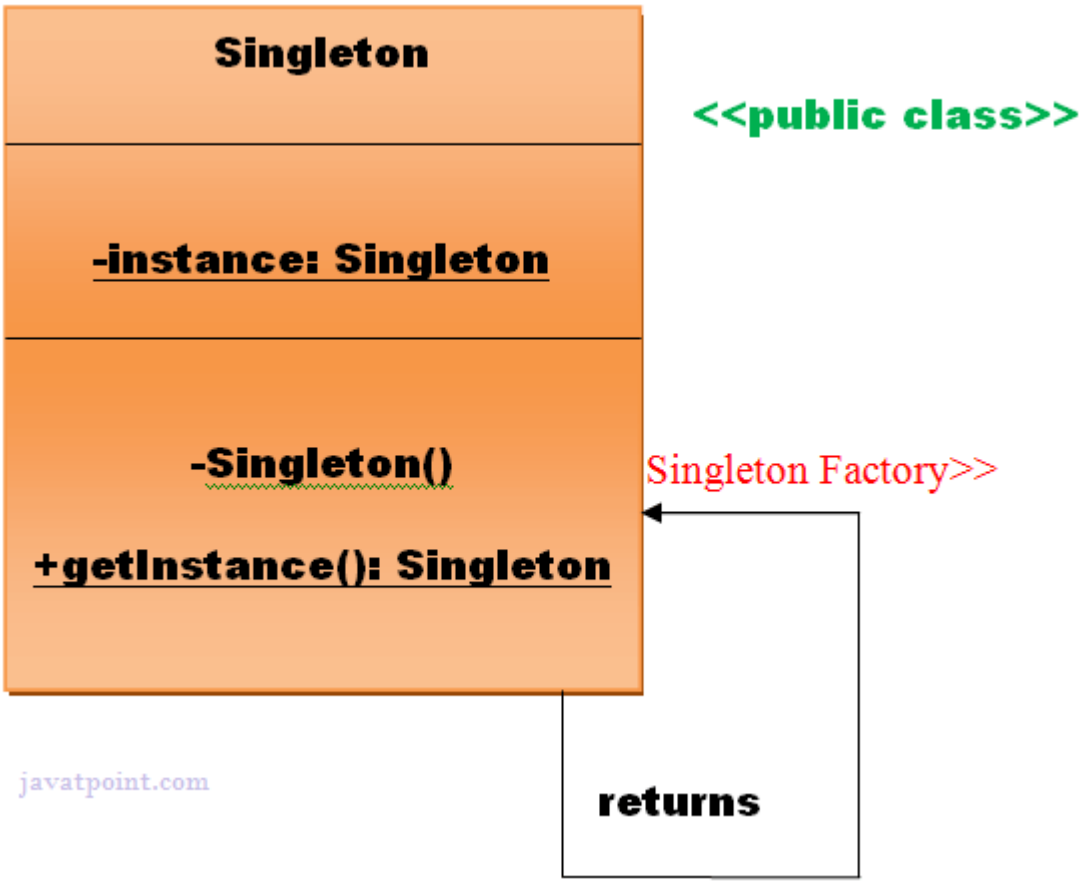
Advantage of Singleton design pattern

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

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

Uml of Singleton design pattern



How to create Singleton design pattern?

To create the singleton class, we need to have static member of class, private constructor and static factory method.

- **Static member:** It gets memory only once because of static, itcontains the instance of the Singleton class.
- **Private constructor:** It will prevent to instantiate the Singleton class from outside the class.
- **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

Understanding early Instantiation of Singleton Pattern

In such case, we create the instance of the class at the time of declaring the static data member, so instance of the class is created at the time of classloading.

Let's see the example of singleton design pattern using early instantiation.

File: A.java

```
class A{
 private static A obj=new A();//Early, instance will be created at load time
 private A(){}

 public static A getA(){
  return obj;
 }

 public void doSomething(){
 //write your code
 }
}
```

Understanding lazy Instantiation of Singleton Pattern

In such case, we create the instance of the class in synchronized method or synchronized block, so instance of the class is created when required.

Let's see the simple example of singleton design pattern using lazy instantiation.

File: A.java

```
class A{
 private static A obj;
 private A(){}

 public static A getA(){
   if (obj == null){
     synchronized(Singleton.class){
       if (obj == null){
           obj = new Singleton();//instance will be created at request time
       }
    }
    }
  return obj;
 }

 public void doSomething(){
 //write your code
 }
}
```

Significance of Classloader in Singleton Pattern

> If singleton class is loaded by two classloaders, two instance of singleton class will be created, one for each classloader.

Significance of Serialization in Singleton Pattern

If singleton class is Serializable, you can serialize the singleton instance. Once it is serialized, you can deserialize it but it will not return the singleton object.

To resolve this issue, you need to override the **readResolve() method** that enforces the singleton. It is called just after the object is deserialized. It returns the singleton object.