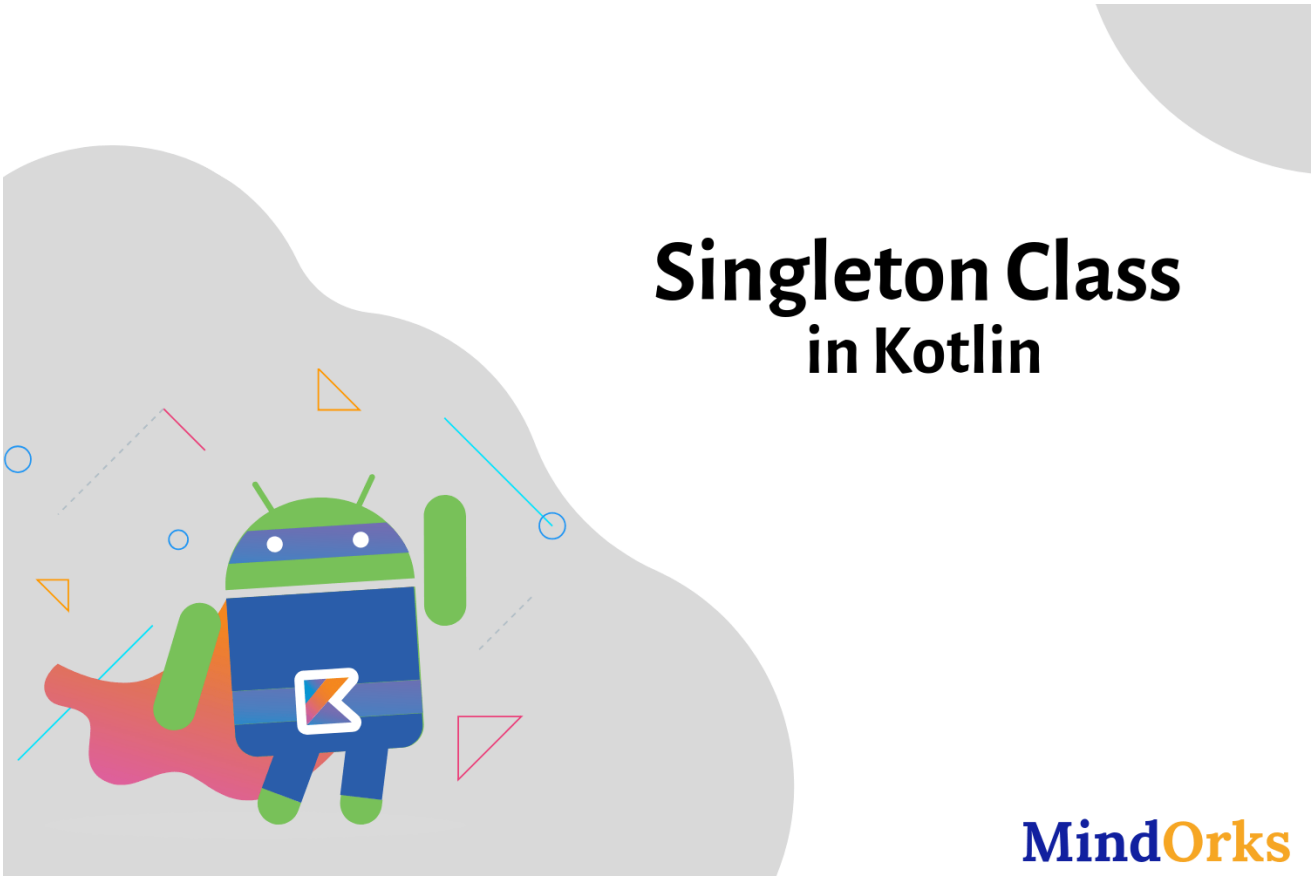
MindOrks

Sumit Mishra
4th September 2019

How to create a Singleton class in Kotlin?



In Android App, for an object which is required to be created only once and use everywhere, we use the **Singleton Pattern** . Singleton Pattern is a software design pattern that restricts the instantiation of the class to only “one” instance. So, to implement the Singleton pattern in our project or software, we make a **singleton class** . In this blog, we will learn how to make a singleton class in Kotlin? So, let’s get started.

Singleton Class

A singleton class is a class that is defined in such a way that only one instance of the class can be created and used everywhere.

Generally, it is done because it takes the resource of the system to create these objects again and again. So it is better to create only once and use again and again the same object.

Properties of Singleton Class

Following are the properties of a typical singleton class:

1. **Only one instance:** The singleton class has only one instance and this is done by providing an instance of the class, within the class. Also, outer classes and subclasses should be prevented to create the instance.
2. **Globally accessible:** The instance of the singleton class should be globally accessible so that each class can use it.

Rules for making a class Singleton

The following rules are followed to make a Singleton class:

1. A private constructor
2. A static reference of its class
3. One static method
4. Globally accessible object reference
5. Consistency across multiple threads

Singleton Example

Following is the example of Singleton class in java:

```
    private static Singleton instance = null;

    private Singleton() {

    }

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

When creating the instance to ensure that there is no thread interference, we use the **synchronized** keyword.

Let's look at the Kotlin code for the same. Below is the Kotlin code for Singleton class:

```
object Singleton
```

Ok, what to do after this?

Nothing! Are you kidding? No, that's the code for using Singleton class in Kotlin. Very simple? Don't worry, let's look at the explanation.

In Kotlin, we need to use the **object** keyword to use Singleton class. The **object** class can have functions, properties, and the **init** method. The

inside a class. The object gets instantiated when it is used for the first time.

Let's have an example of the Singleton class in Kotlin.

```
object Singleton{

    init {
        println("Singleton class invoked.")
    }
    var variableName = "I am Var"
    fun printVarName(){
        println(variableName)
    }

}

fun main(args: Array<String>) {
    Singleton.printVarName()
    Singleton.variableName = "New Name"

    var a = A()
}

class A {

    init {
        println("Class init method. Singleton variableName property
        Singleton.printVarName()
    }
}
```

Here, in the above example, we are having one function named **printVarName()** and one property named **"variableName".** When **A** class is instantiated, then changes can be reflected in the **object** class. So, the output of the above code will be:

```
I am Var
Class init method. Singleton variableName property : New Name
New Name
```

Object extending a class

You can use an object in Kotlin to extend some class or implement some interface just like a normal class. Let's have an example of the same:

```
fun main(args: Array<String>) {
    var a = A()
    Singleton.printVarName()
}

open class A {

    open fun printVarName() {
        print("I am in class printVarName")
    }

    init {
        println("I am in init of A")
    }
}

object Singleton : A() {

    init {
        println("Singleton class invoked.")
    }

    var variableName = "I am Var"
    override fun printVarName() {
        println(variableName)
    }
}
```

```
I am in init of A
I am in init of A
Singleton class invoked.
I am var
```

So, you can use object class just like a normal class in most of the cases.

Learn System Design for your next Interview from here.

Singleton class with Argument in Kotlin

In the earlier part of the blog, we learned that we can't have constructors in a singleton class. To initialize something, we can do so by using **init** in the singleton class. But what if you need to pass some argument for initialization just like in parameterized constructors? Since we can't use constructors here. So, we need to find some other way of doing the same.

If we particularly talk about Android, we know that in Android we generally need to pass a **context** instance to **init** block of a singleton. This can be done using **Early initialization** and **Lazy initialization** . In early initialization, all the components are initialized in the **Application.onCreate()** using the **init()** functions. But this results in slowing down the application startup by blocking the main thread. So, it is generally advised to use the **lazy initialization** way. In lazy initialization, we use the context as an argument to a function returning the instance of the singleton. We can achieve this by using a **SingletonHolder** class. Also, to make it thread-safe, we need to have a way of synchronization and double-checked locking.

```
open class SingletonHolder<out T: Any, in A>(creator: (A) -> T) {
    private var creator: ((A) -> T)? = creator
```

MindOrks

```
        fun getInstance(arg: A): T {
            val checkInstance = instance
            if (checkInstance != null) {
                return checkInstance
            }

            return synchronized(this) {
                val checkInstanceAgain = instance
                if (checkInstanceAgain != null) {
                    checkInstanceAgain
                } else {
                    val created = creator!!(arg)
                    instance = created
                    creator = null
                    created
                }
            }
        }
    }
```

The above code is the most efficient code for double-checked locking system and the code is somehow similar to the **lazy()** function in Kotlin and that's why it is called lazy initialization. So, whenever you want a singleton class with arguments then you can use the **SingletonHolder** class.

Here, in the above code, in place of the **creator** function which is passed as an argument to the **SingletonHolder** , a custom lambda can also be declared inline or we can pass a reference to the private constructor of the singleton class. So, the code will be:

```
class YourManager private constructor(context: Context) {
    init {
        // do something with context
    }
```

MindOrks

Now, the singleton can be easily invoked and initialized by writing the below code and this is lazy as well as thread-safe :)

```
YourManager.getInstance(context).doSomething()
```

That's it for the Singleton blog. Hope you like this blog. You can also refer to the Kotlin website . To learn more about some of the cool topics of Android, you can visit our blogging website and can join our journey of learning.

Keep Learning :)

Team MindOrks!

Recommended for You

How to create multiple apk files for android application

30th August 2017

Too many android devices and much more to come and among them, various different CPU architectures and different screen densities devices are there. Making single apk with all device compatible, increases your apk size, as it has resources for every device.