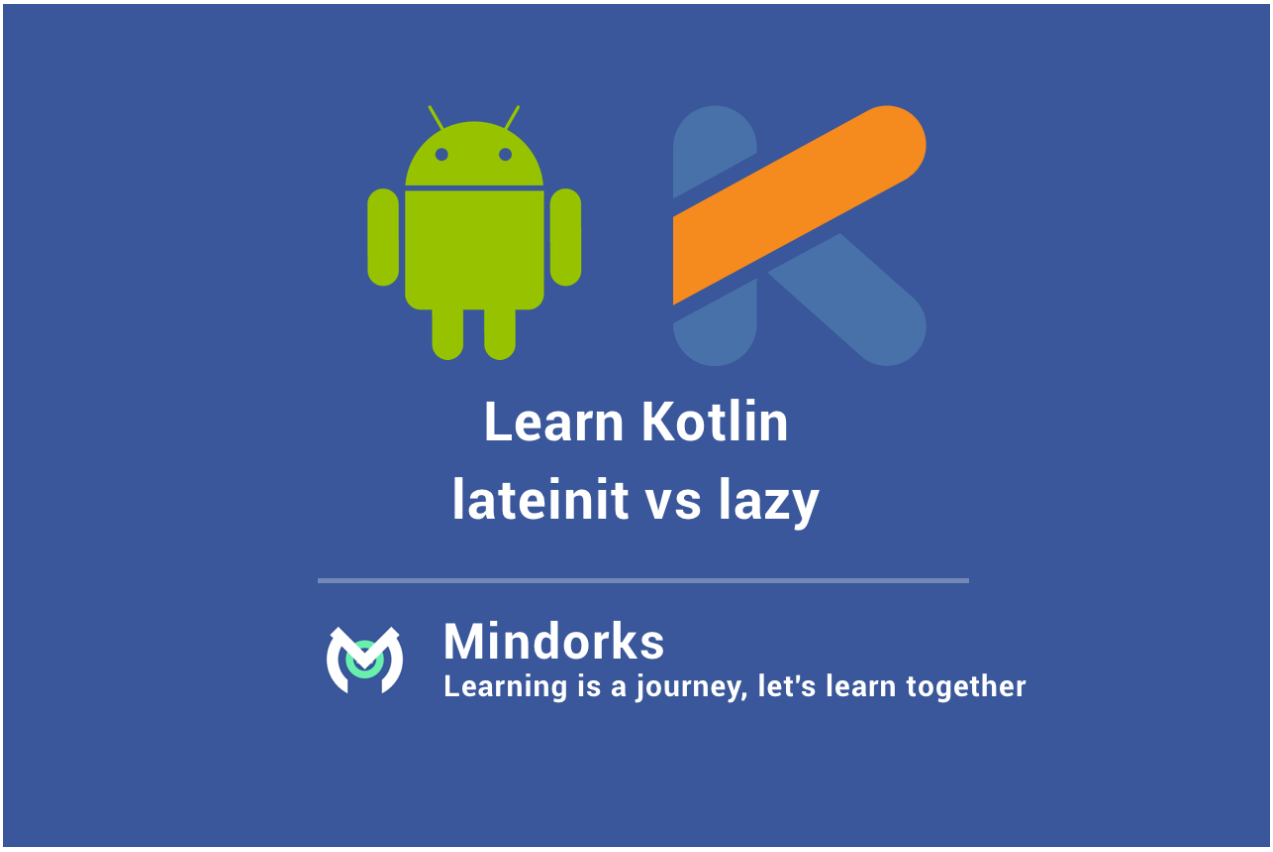
Amit Shekhar

22 May 2017

Learn Kotlin— lateinit vs lazy



lateinit vs lazy

There are many great features available in Kotlin, we can take advantage of all these features to write the better application in Kotlin. Among all those features, *lateinit* and *lazy* are important property initialization feature. We must know when to use which property initialization.

Kotlin Property Initialization

If you do not want to initialize a property in the constructor, then these are two important ways of property initialisation in Kotlin.

lateinit

lateinit is late initialization.

Normally, properties declared as having a non-null type must be initialized in the constructor. However, fairly often this is not convenient. For example, properties can be initialized through dependency injection, or in the setup method of a unit test. In this case, you cannot supply a non-null initializer in the constructor, but you still want to avoid null checks when referencing the property inside the body of a class.

To handle this case, you can mark the property with the *lateinit* modifier.

Example

```
public class Test {

  lateinit var mock: Mock

  @SetUp fun setup() {
     mock = Mock()
  }

  @Test fun test() {
     mock.do()
  }
}
```

The modifier can only be used on var properties declared inside the body of a class (not in the primary constructor), and only when the property does not have a custom getter or setter. The type of the property must be non-null, and it must not be a primitive type.

lazy() is a function that takes a lambda and returns an instance of Lazy which can serve as a delegate for implementing a lazy property: the first call to get() executes the lambda passed to lazy() and remembers the result, subsequent calls to get() simply return the remembered result.

Example

```
public class Example{
  val name: String by lazy { “Amit Shekhar” }
}
```

So the first call and the subsequent calls, **name** will return **“Amit Shekhar”**

Now, How to choose when to use which one?

- lazy can only be used for val properties, whereas lateinit can only be applied to vars because it can’t be compiled to a final field, thus no immutability can be guaranteed.
- lateinit var can be initialized from anywhere the object is seen from. If you want your property to be initialized from outside in a way probably unknown beforehand, use lateinit.

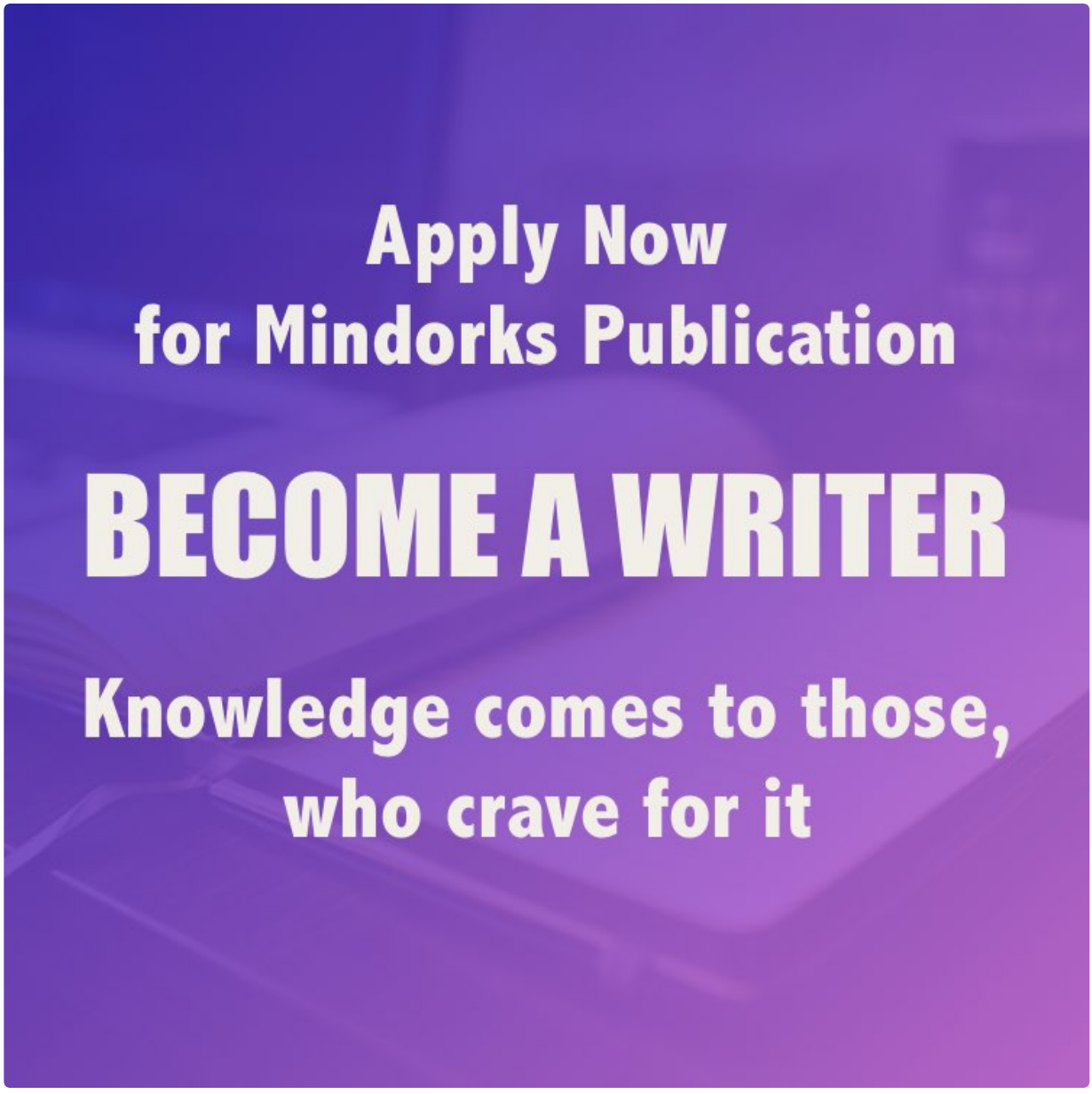






Facebook Comments Plugin

Mindorks Updates



Mindorks Android Training Center

Correctify App
Janishar Ali

English editing and learning app, a product of Mindorks.

Android Interview Questions
Amit Shekhar

Your cheat sheet for Android interview.

Better yourself
Dalai Lama

The goal is not to be better than the other man, but your previous self.

GitHub

MindorksOpenSource
Amit Shekhar
Janishar Ali

Connect With Mindorks

Facebook
Linkedin
Medium

Mindorks Community

Get Community Invite
Slack Community
Subscribe