Kotlin Cheatsheet: Scope Functions (let, run, apply, also, with)

sy is typing
Nov 6, 2019 · 2 min read

		returns	
		original object (Foo)	any object (eg, Bar)
original object (Foo) exposed as	this	**apply** *(this --> foo)* `val foo = Foo().apply {` `  this.field1 = 1` `}`	**run** *(this --> bar)* `val bar = Foo().run {` `  this.field1 = 1` `  this.toBar()` `}`
	it	**also** *(it --> foo)* `val foo = Foo().also {` `  doSomethingTo(it)` `}`	**let** *(it --> bar)* `val bar = getFoo()?.let {` `  it.toBar()` `}`

Note: `this` can be omitted.

There are recommended use-cases for each scope function but at the end of the day, choose the function that returns you the correct output and fits your code convention.

apply

Useful for:

- configuring an object:

```
val foo = Foo().apply {
    this.field1 = 1
}
```

run

Useful for:

- configuring an object and computing the result:

```
val bar = Foo().run {
    this.field1 = 1
    this.toBar()
}
```

- grouping several statements into an expression: `run` has a non-extension form too:

```
val bar = run {
    val foo = Foo()
    foo.field1 = 1
    foo.toBar()
}
```

also

Useful for:

- doing additional effects involving the object:

```
val foo = Foo().also {
     doSomethingTo(it)
}
```

let

Useful for

- executing a lambda on non-null objects:

```
val bar = getFoo()?.let {
     it.toBar()
}
```

- introducing an expression as a variable in local scope:

```
(...complicated expression...).let {
    doSomethingWith(it)
}
```

with

Useful for:

- grouping function calls on an object. Similar to `run` but not an extension:

```
val bar = with(Foo()) {
    this.field1 = 1
    this.toBar()
}
```

Other stories you may like

- Write fluent code in Kotlin
- Software Estimates are a Two-Sided Relationship
- Use Java Nullability Annotations to facilitate conversion to Kotlin

Hi, if you enjoyed this post, I thought that you might also enjoy these t-shirts with code-inspired designs.

Java Kotlin Programming Android App Development Software Engineering