

Objects

Objects



Objects, Singletons &
Companion objects

Object Expressions and Declarations |

Sometimes we need to create an object of a slight modification of some class, without explicitly declaring a new subclass for it. Java handles this case with *anonymous inner classes*. Kotlin slightly generalizes this concept with *object expressions* and *object declarations*.

Object expressions

To create an object of an anonymous class that inherits from some type (or types), we write:

```
window.addMouseListener(object : MouseAdapter() {  
    override fun mouseClicked(e: MouseEvent) { ... }  
  
    override fun mouseEntered(e: MouseEvent) { ... }  
})
```

If, by any chance, we need "just an object", with no nontrivial supertypes, we can simply say:

```
fun foo() {  
    val adHoc = object {  
        var x: Int = 0  
        var y: Int = 0  
    }  
    print(adHoc.x + adHoc.y)  
}
```

If a supertype has a constructor, appropriate constructor parameters must be passed to it. Many supertypes may be specified as a comma-separated list after the colon:

```
open class A(x: Int) {  
    public open val y: Int = x  
}  
  
interface B { ... }  
  
val ab: A = object : A(1), B {  
    override val y = 15  
}
```

Object declarations

[Singleton](#) may be useful in several cases, and Kotlin (after Scala) makes it easy to declare singletons:

```
object DataManager {  
    fun registerDataProvider(provider: DataProvider) {  
        // ...  
    }  
  
    val allDataProviders: Collection<DataProvider>  
        get() = // ...  
}
```

This is called an *object declaration*, and it always has a name following the **object** keyword. Just like a variable declaration, an object declaration is not an expression, and cannot be used on the right hand side of an assignment statement.

Object declaration's initialization is thread-safe.

To refer to the object, we use its name directly:

```
DataManager.registerDataProvider(...)
```

Such objects can have supertypes:

```
object DefaultListener : MouseAdapter() {  
    override fun mouseClicked(e: MouseEvent) { ... }  
  
    override fun mouseEntered(e: MouseEvent) { ... }  
}
```

Companion Objects

An object declaration inside a class can be marked with the `companion` keyword:

```
class MyClass {  
    companion object Factory {  
        fun create(): MyClass = MyClass()  
    }  
}
```

Members of the companion object can be called by using simply the class name as the qualifier:

```
val instance = MyClass.create()
```

The name of the companion object can be omitted, in which case the name `Companion` will be used:

```
class MyClass {  
    companion object { }  
}  
  
val x = MyClass.Companion
```