Hidden Gems In Kotlin Stdlib

Adam McNeilly - @AdamMc331

Two Types Of Hidden Gems

Two Types Of Hidden Gems

- Included Methods

Two Types Of Hidden Gems

- Included Methods
- Language Features

Strings

Strings

Java developers use Apache Commons[1].

```
StringUtils.IsBlank("  "); // True
StringUtils.SubstringBefore("Adam.McNeilly", "."); // "Adam
StringUtils.SubstringAfter("Adam.McNeilly", "."); // "McNeilly"
StringUtils.LeftPad("1", 2); // " 1"
StringUtils.Chop("abc"); // "ab"
```

[1] https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html

Strings

Already built into Kotlin.

```
val blank = "   ".isBlank() // Also: CharSequence?.isNullOrBlank
val first = "Adam.McNeilly".substringBefore('.') // "Adam"
val last = "Adam.McNeilly".substringAfter('.') // "McNeilly"
val withSpaces = "1".padStart(2) // " 1"
val endSpaces = "1".padEnd(3, '0') // "100"
val dropStart = "Adam".drop(2) // "am"
val dropEnd = "Adam".dropLast(2) // "Ad"
```

Strings

Even more options[2].

```
"A\nB\nC".lines() // [A, B, C]

"One.Two.Three".substringAfterLast('.') // "Three"
"One.Two.Three".substringBeforeLast('.') // "One.Two"

"ABCD".zipWithNext() // [(A, B), (B, C), (C, D)]

val nullableString: String? = null
nullableString.orEmpty() // Returns ""
```

[2] https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/index.html

First & Last

String methods often handle opposite concerns.

```
substringBefore()
substringAfter()

isEmpty()
isNotEmpty()

padStart()
padEnd()

drop()
dropLast()

trimStart()
trimEnd()
```

@AdamMc331
@Brooklyn_Kotlin

Collections

Collections

Java `Collections` class for some work.

```
Collections.sort(myList);
Collections.max(myList);
Collections.min(myList);
Collections.shuffle(myList);
Collections.reverse(myList);
Collections.swap(myList, 1, 2);
```

Collections

Built in to Kotlin.

```
myList.sort()
myList.max()
myList.min()
myList.shuffle()
myList.reverse()
myList.swap(1, 2)
```

Iterating Collections

Iterating Collections

```
public String getAdam(List<Person> people) {
    Person result = null;

    for (Person person : people) {
        if (person.getName().equals("Adam")) {
            result = person;
            break;
        }
    }

    return result;
}
```

Iterating Collections

```
public boolean hasAdam(List<Person> people) {
    boolean result = false;

    for (Person person : people) {
        if (person.getName().equals("Adam")) {
            result = true;
            break;
        }
    }

    return result;
}
```

Iterating Collections

```
fun getAdam(people: List<Person>): Person? {
    return people.firstOrNull { it.name == "Adam" }
}

fun hasAdam(people: List<Person>): Boolean {
    return people.any { it.name == "Adam" }
}
```

Iterating Collections

```
myList.filter { }
myList.filterNot { }
myList.filterIsInstance()
myList.filterNotNull { }

myList.first { } // Also: indexOfFirst { }
myList.firstOrNull { }
myList.last { } // Also: indexOfLast { }
myList.lastOrNull { }
myList.single { }
myList.singleOrNull { }

myList.any { }
myList.none { }
myList.all { }
myList.single { }

myList.partition { } // Pair<List<T>, List<T>>
```

@AdamMc331
@Brooklyn_Kotlin

Language Features

Higher Order Functions and Lambdas

Lambda Syntax

"Lambda expressions and anonymous functions are 'function literals', i.e. functions that are not declared, but passed immediately as an expression."[3]

```
val evenNumbers = myNumberList.filter {
    it % 2 == 0
}
```

[3] https://kotlinlang.org/docs/reference/lambdas.html#lambda-expressions-and-anonymous-functions

Lambda Syntax

```
public inline fun <T> Array<out T>.filter(predicate: (T) -> Boolean): List<T> {
    return filterTo(ArrayList<T>(), predicate)
}
```

If the last parameter of a function accepts a function, a lambda expression can be placed outside the parentheses[4]:

```
val evenNumbers = myNumberList.filter {
    it % 2 == 0
}
```

---

[4] https://kotlinlang.org/docs/reference/lambdas.html#passing-a-lambda-to-the-last-parameter

Lambda Syntax

```
// Have our permissions check accept a function to run if it's granted:
private fun withWritePermission(callback: () -> Unit) {
    activity?.let { activity ->
        RxPermissions(activity)
                .request(Manifest.permission.WRITE_EXTERNAL_STORAGE)
                .subscribe { granted ->
                    if (granted) {
                        callback()
                    }
                }
    }
}
```

Lambda Syntax

```
// Call it with a cool lambda
withWritePermission {
    launchGallery()
}

// Could do API version checks too
supportsLollipop {
    doSomethingForAndroidL()
}
```

Lambda Syntax

Note: There's limitations to this - no chance for an else block.

```
// What if I didn't have permission?
withWritePermission {
    launchGallery()
}

// What if something changes again in P?
supportsLollipop {
    doSomethingForAndroidL()
}
```

Destructuring Declarations

Destructuring Declarations

```
// Java-esque approach
val coordinates = arrayOf(5, 10, 15)
val x = coordinates[0]
val y = coordinates[1]
val z = coordinates[2]
println("X coordinate: $x")
println("Y coordinate: $y")
println("Z coordinate: $z")
```

Destructuring Declarations

Destructure an object into individual variables[5].

```
val coordinates = arrayOf(5, 10, 15)
val (x, y, z) = coordinates
println("X coordinate: $x")
println("Y coordinate: $y")
println("Z coordinate: $z")
```

[5] https://kotlinlang.org/docs/reference/multi-declarations.html

Destructuring Declarations

```
// Use this to better traverse maps
val actionsMap: Map<String, Action> = hashMapOf(...)
for ((key, action) in actionsMap) {
    // ...
}
```

Destructuring Declarations

```
// Can be used on a data class
// Return two things from a function by making use of a data class
data class Result(val result: Int, val status: Status)
fun function(...): Result {
    return Result(result, status)
}

// Now, to use this function:
val (result, status) = function(...)
```

Destructuring Declarations

```
class Person(val name: String, val age: Int) {
    operator fun component1(): String {
        return name
    }

    operator fun component2(): Int {
        return age
    }
}

val person = Person("Adam", 25)
val (name, age) = person
```

Recap

Recap

- Strings class contains all common use cases, and then some.

Recap

- Strings class contains all common use cases, and then some.
- Collections package helps for filtering/modifying/iterating them.

Recap

- Strings class contains all common use cases, and then some.
- Collections package helps for filtering/modifying/iterating them.
- Utilize function types as the last parameter for lambda syntax.

Recap

- Strings class contains all common use cases, and then some.
- Collections package helps for filtering/modifying/iterating them.
- Utilize function types as the last parameter for lambda syntax.
  - BUT beaware of the limitation of not having an else block.

Recap

- Strings class contains all common use cases, and then some.
- Collections package helps for filtering/modifying/iterating them.
- Utilize function types as the last parameter for lambda syntax.
  - BUT beaware of the limitation of not having an else block.
- Destructuring is great for breaking down objects into variables.