СПб АУ НОЦНТ РАН

Kotlin 03

13.11.2017

Foreach

Так делать не нужно

```
(0..10).forEach {
    println(it)
}
```

Foreach

Так делать можно

```
nullableIterable?.forEach {
    println(it)
}
```

Частый паттерн с let

```
val transformedValue =
    a.getNullableValue()?.let { transform(it) }
    ?: return null
```

Когда нужен object?

```
// Singleton pattern
object EmptyList : List<Nothing> {
    private const val serialVersionUID: Long = -7390468764508069838L
    override val size: Int get() = 0
    ...
}
```

Function type with receiver

```
fun builder(block: String.() -> Int) {
    block("someString")
    "someString".block() // the same
}
```

Function type with receiver

```
fun builder(block: String.() -> Int) {
    block("someString")
    "someString".block() // the same
}

builder { // lambda has receiver of type String
    this.length
    this@builder.length
    length // used as implicit receiver
}

builder(String::someStringExtension)

fun foo(x: String) = 1

builder(::foo)
```

Function literal with receiver

- Используется для построения DSL
- Нельзя вызывать private/protected

Примеры. with

```
fun <T, R> with(receiver: T, block: T.() -> R): R = receiver.block()

class A {
    fun String.spaceToCamelCase() {}
}

with(A()) {
    "someString".spaceToCamelCase()
    with("someString") {
        spaceToCamelCase
    }
}
```

Примеры. with

```
val shortDescription =
    with(database.findPerson(123)) {
        "$firstName $secondName, $email"
    }
```

Примеры. run

```
fun <R> run(block: () -> R): R = block()
```

Примеры. run

```
fun <R> run(block: () -> R): R = block()

val x: Int? = run l@{
    val result = findComponent() ?: return@l null
    val arg = findArgument() ?: return@l null

    result.callFunction(arg)
}
```

Примеры. also

```
public fun <T> T.also(block: (T) -> Unit): T {
    block(this)
    return this
}
```

Примеры. also

```
public fun <T> T.also(block: (T) -> Unit): T {
    block(this)
    return this
}

public fun createObjectOrGetCached(key: Name): Obj {
    if (key in cache) return cache[key]
    return Obj(key).also(::putToCache)
}
```

Примеры. apply

```
public fun <T> T.apply(block: T.() -> Unit): T {
    block();
    return this
}
```

Версия паттерна Builder

- Создаем изменяемый объект
- Инициализируем в лямбде
- (опционально) создаем immutable-версию

Примеры. apply

```
val person = Person().apply {
    name = someName // this.name = someName
    email = someEmail

    for (job in somePreviousJobs) {
        addPreviousJob(job) // this.addPreviousJob(job)
    }

}
```

Примеры. apply

```
val person = Person().apply {
    name = someName // this.name = someName
    email = someEmail

    for (job in somePreviousJobs) {
        addPreviousJob(job) // this.addPreviousJob(job)
    }

}
```

Примеры. apply

```
fun buildStringFromLines(lines: Iterable<String>): String {
    val sb = StringBuilder()

    for (line in lines) {
        sb.append(line + "\n")
    }

    return sb.toString()
}
```

Примеры. apply

```
fun buildStringFromLines(lines: Iterable<String>): String =
    StringBuilder().apply {
        for (line in lines) {
            append(line + "\n")
        }
    }.toString()
```

Примеры. apply

```
fun buildStringFromLines(lines: Iterable<String>): String =
    buildString {
        for (line in lines) {
            append(line + "\n")
        }
    }
```

Примеры. apply

```
fun buildAllSubStrings(s: String): List<String> =
    mutableListOf<String>().apply {
        for (i in s.indices) {
            for (j in i + 1..s.length) {
                add(s.substring(i, j))
            }
        }
    }
```

DSL builders. kotlinx.html

```
fun result() =
  createHTML().html {
      head {
          title {+"XML encoding with Kotlin"}
      }
      body {
          h1 {+"XML encoding with Kotlin"}
      }
  }
```

DSL builders. kotlinx.html

```
fun result(rows: Collection<Person>) =
  createHTML().html {
      head {
          title {+"XML encoding with Kotlin"}
      }
      body {
          h1 {+"XML encoding with Kotlin"}
          for (person in rows) {
            div { +person.toString() }
          }
      }
  }
```

DSL builders. kotlinx.html

```
fun LI.dropdownToggle(block : A.() -> Unit) {
    a("#", null, "dropdown-toggle") {
        attributes["data-toggle"] = "dropdown"
        role = "button"
        attributes["aria-expanded"] = "false"

        block()

        span {
            classes = setOf("caret")
        }
    }
}
```

DSL builders. kotlinx.html

```
fun html(init: HTML.() -> Unit): HTML {
    val html = HTML()
    html.init()
    return html
}

fun HTML.head(init: Head.() -> Unit) : Head {
    val head = Head()
    head.init()
    children.add(head)
    return head
}

fun HTML.body(...) {...}
```

DSL builders. kotlinx.html

```
protected fun <T : Element> initTag(tag: T, init: T.() -> Unit): T {
    tag.init()
    children.add(tag)
    return tag
}

fun head(init: Head.() -> Unit) = initTag(Head(), init)
```

DSL builders. kotlinx.html

```
interface TagWithText {
    val children: ...
    fun String.unaryPlus() {
        children.add(TextElement(this))
    }
}
```

@DslMarker

```
html {
    head {
        head {} // should be forbidden
    }
    // ...
}
```

@DslMarker

```
@DslMarker
annotation class HtmlTagMarker

@HtmlTagMarker
abstract class Tag(val name: String) { ... }

class HTML() : Tag("html") { ... }
class Head() : Tag("head") { ... }
```

@DslMarker

```
html {
    head {
        head { } // error: a member of outer receiver
        this@html.head { } // possible
    }
    // ...
}
```

Android

```
verticalLayout {
  val name = editText()
  button("Say Hello") {
      onClick { toast("Hello, \${name.text}!") }
  }
}
```

Стандартная библиотека. Коллекции

- map/filter/flatMap
- mapTo/filterTo/flatMapTo – куда сложить результат
- zip/fold/sum/sumBy/groupBy/sortBy/sortedBy/joinToString
- Множество всяких операций с коллекциями
- Почти все дублируются для массивов
- Большая часть – inline-функции

Стандартная библиотека. Коллекции

- listOf(1, 2), mutableListOf(..), arrayListOf(..)
- setOf(1, 2), mutableSetOf(..), hashSetOf(..), linkedHashSetOf(..)
- arrayOf(1, 2), intArrayOf(..), ..
- mapOf(1 to "1 ..), mutableMapOf(..), hashMapOf(..)
- emptyList(), emptyMap(), emptySet()

Collections

```
fun <T> List<T>?.orEmpty(): List<T> = this ?: emptyList()

val Collection<*>.indices: IntRange
    get() = 0..size - 1

val <T> List<T>.lastIndex: Int
    get() = this.size - 1

fun <T> List<T>.getOrNull(index: Int): T? {
    return if (index >= 0 && index <= lastIndex) get(index) else null
}
```

Varargs

```
fun <T> mutableListOf(vararg elements: T): MutableList<T> {
    elements[0] // elements : Array<out T>
    ...
}

fun main(args: Array<String>) {
    mutableListOf(args[0])
    mutableListOf(*args, "123") // *args - spread-operator
}
```

trimMargin

```
val text = """
|var a = 1
|var b = 2
|
|""".trimMargin()

println(text)
```

Regex

```
"\\d+".toRegex().containsMatchIn(args[0])
```

Preconditions

- assert/assertNotNull – AssertionError
- check/checkNotNull – IllegalStateException
- require/requireNotNull – IllegalArgumentException
- fun error(message: Any): Nothing = throw IllegalStateException(message.toString())

Домашнее задание

- Ветка “03-tex-builder”