Лекция #7.1. Операторы ?., ?: и !! языка Kotlin

Проблема **nullable**-типов в том, что многие методы их узких классов-аналогов (без поддержки **null**) становятся недоступными.

```
fun main() {
    val a: String = "Hello"
    val b: String? = "World"

    val aL = a.length
    val bL = b.length

    val aI = a[2]
    val bI = b[2]

    val aU = a.toUpperCase()
    val bU = b.toUpperCase()
}
```

На скрине показано, что несмотря на то, что переменная **b** не содержит **null**, использовать свойства и методы обычной строки нельзя.

Как быть в таких ситуациях? Очевидный ответ – проверить значение переменной на **null**. Если оно не является **null**, то с помощью оператора **as** привести его к узкому типу и только после этого вызывать методы.

```
fun main() {
    val b: String? = "World"
    var c: String

    if (b != null) {
```

```
        c = b as String
        println(c.length)
    }
}
```

Однако умный компилятор **Kotlin** позволяет не делать такое приведение вручную. Если сравнение в заголовке **if** возвращает истину (переменная не равна **null**), то в области действия сравнения компилятор будет считать это значение обычным. Как бы сам временно приведет переменную к ненуллабл-типу. Поэтому код выше упрощается до такого:

```
fun main() {
    val b: String? = "World"

    if (b != null)
        println(b.length)
}
```

Если требуется сохранить в переменную длину строки, а также какой-либо числовой сигнал, если был **null**, потребуется ветка **else**:

```
fun main() {
    val b: String? = "World"
    val bL: Int

    if (b != null)
        bL = b.length
    else
        bL = -1

    println(bL)
}
```

Таким образом проблема **nullable**-типов в **Kotlin** решается просто. Однако несколько громоздко и, даже если записать **if-else** в одну строку, длинно. Поэтому в языке предусмотрены специальные операторы и функции для работы с **nullable**-типами. В этом уроке будут рассмотрены три оператора **Kotlin** – оператор безопасного вызова, оператор "**элвис**" и утверждение "это не null".

?. – оператор безопасного вызова

Оператор безопасного вызова, обозначаемый вопросительным знаком с точкой, похож на проверку на **null** с **if** в варианте без **else**. Он проверят, что значение слева от него не равно **null**. Если же оно равно **null**, то ничего не происходит. Точнее, все выражение возвращает **null**.

Если же значение слева от оператора **?.** возвращает что-то отличное от **null**, то вызывается метод, стоящий справа от этого оператора. Можно представить, что слева от **?.** стоит проверка условия на неравенство **null**, а справа – это тело **if**, которое выполняется, если условие вернуло истину.

```
fun main() {
    val b: String? = "World"
    val c: String? = null

    val bL: Int? = b?.length
    val cL: Int? = c?.length

    println(bL) // 5
    println(cL) // null
}
```

Поскольку выражение с оператором безопасного вызова метода потенциально может вернуть **null**, то значение всего выражения с этим оператором всегда будет принадлежать какому-либо **nullable**-типу.

?: – оператор "элвис"

Оператор, обозначаемый вопросительным знаком с двоеточием, подобен проверке на **null** в варианте **if-else**. Он возвращает значение слева от себя, если оно не **null**. И возвращает значение справа от себя, если то, что слева, – **null**.

Оператор **?:** используется для замены **null**, каким-либо значением, принадлежащим обычно зауженному типу. В результате выражение с "**элвисом**" позволяет не увеличивать в программе количество **nullable**-переменных.

```
fun main() {
    val b: String? = readLine()
    val c: String = b ?: ""

    println(c.length)
}
```

Переменная **c** будет содержать либо строку, которую вернула функция **readLine()**, либо пустую строку, если **readLine()** вернет **null**. Избавимся от переменной **c**:

```
fun main() {
    val b: String = readLine() ?: ""

    println(b.length)
}
```

Оператор **?:** нередко используют вместе с оператором ?.. Если надо вызвать ненуллабл-метод на **nullable**-переменную, требуется оператор безопасного вызова. Поскольку в случае **null** метод вызываться не будет, а **null** надо заменить, требуется оператор "**элвис**".

```
fun main() {
    val b: String? = readLine()
```

```
    val c: Int = b?.length ?: -1

    println(c)
}
```

Последовательность действий в выражении **b?.length ?: -1** идет слева направо. Сначала выполняется подвыражение с оператором **?.**. Потом его результат подставляется как левый операнд оператора **?:**. Так если длина строки была удачно измерена, то она и запишется в переменную **c**. Если же оператор безопасного вызова вернул **null**, то в переменную запишется **-1**.

!! – утверждение "это не null"

Два восклицательных знака, стоящих после **nullable**-значения, преобразуют его к типу без поддержки **null**. При этом перед преобразованием никак не проверяется, что значение действительно не содержит **null**. Поэтому, если в процессе выполнения программы окажется, что значение, которое пытается преобразовать оператор !!, все-таки **null**, то останется только один выход – выбросить исключение **NullPointerException**. Если оно не обрабатывается кодом, программа **аварийно завершится**.

Поэтому, несмотря на удобство этого оператора, его следует использовать только там, где вы уверенны, что **null** быть не может.

```
fun main() {
    val b: String? = readLine()
    val c: String = b!!
    val d: String = readLine()!!
    val e: Int = b!!.length
}
```