Знакомство с объектно-ориентированными возможностями языка Kotlin

Романов Владимир Юрьевич
МГУ им. М.В.Ломоносова, ф-т ВМК
romanov.rvy@yandex.ru

Октябрь 10, 2020

1.1. ОБЗОР БАЗОВЫХ ВОЗМОЖНОСТЕЙ ЯЗЫКА KOTLIN.

Точки входа в программу

Точек входа в программу не ограниченно.

Пример точки входа без параметра:

```
fun main() {
    println("Hello Kotlin!")
}
```

Пример точки входа с параметром:

```
fun main(args: Array<String>) {
    println("Hello Kotlin!")
}
```

Массив - это обычный класс

Пакеты

- Описание пакета должно быть в начале файла.
- Не требуется соответствия структуры пакетов структуре папок в файловой системе.
- Файл программы может быть расположен в любом месте.

- Точки с запятой в конце строки не требуется

Пример объявления пакета:

```
package tool.geometry
```

Импорты

Импортироваться могут классы, переменные, функции, …

Импортируемые элементы могут получить псевдонимы.

Например, ***PI_From_Kotlin*** или ***E_From_Java***:

```
import java.lang.Math
import java.lang.Math.E as E_From_Java
import kotlin.math.PI as PI_From_Kotlin

fun main(args: Array<String>) {
    println("Pi = $PI_From_Kotlin")
    println("E = $E_From_Java")
    println("sin(Pi) = ${Math.sin(PI_From_Kotlin)}")
}
```

1.1.1. ПЕРЕМЕННЫЕ

Переменные только для чтения

Переменные только для чтения объявляются с использованием ключевого слова ***val***.
Тип переменной ***должен*** быть задан явно в объявлении переменной если НЕ выполняется ее инициализация в месте объявления.

```
val a: Int
```

Тип переменной ***может*** быть задан явно в объявлении переменной если в объявлении выполняется ее инициализация:

```
val b: Int = 1
```

Присваивание переменной для чтения ***необходимо*** выполнить только один раз.

```
fun main() {
    val x: Int = 1
    val y: Int;
    y = if (x > 0) 1 else -1
}
```

Выведение типа переменной

Тип переменной может быть выведен из присваемого ей значения:

```
val c = 2
val PI = 3.14
val language = "Kotlin"
val d = if (c > 0) 1 else -1
```

Изменяемые переменные

Изменяемые переменные объявляются с использованием ключевого слова ***var***.

```
var x: Int = 0
var y = 0

fun increment() {
    y = x
}
```

Тип переменной не изменяется:

```
var size = 10

fun increment() {
    size = "Kotlin" // Ошибка компиляции.
}
```

Переменные (свойства) верхнего уровня

Переменные могут быть объявлены вне методов или классов (на верхнем уровне).
В таком случае они называются ***свойства верхнего уровня***:

```
// Файл Colors.kt
package diagram

val red    = "FF0000"
val green  = "00FF00"
val blue   = "0000FF"
```

В результате по имени файла будет создан класс ***ColorsKt***, а свойства верхнего уровня будут *статическими переменными* этого класса.

```
import diagram.red

var lineColor = red
var textColor = diagram.green
```

1.1.1. ФУНКЦИИ

Функции

- Функция начинается с ключевого слова ***fun***
- Тип параметра указывается после имени параметра
- Тип результата вызова указывается после списка параметров. Этот тип отделен от списка параметров двоеточием.
- Тело функции может быть *блоком* (в фигурных скобках):

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

- Тело функции может быть *выражением*:

```
fun sum(a: Int, b: Int): Int = a + b
```

- Условный оператор в языке Kotlin - это ТАКЖЕ выражение (вырабатывает значение):

```
fun maxOf(a: Int, b: Int): Int =
    if (a > b) a else b
```

Вывод типа значения функции

- Если тело функции *выражение*, то тип возвращаемого значения может быть не указан. Этот тип выводится:

```
fun sum(a: Int, b: Int) = a + b

fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

Функции не возвращающие значения

Если функция значение не возвращает, то тип возвращаемого значения опускается или это тип ***Unit***

```
var size = 10

fun increment() {
    size++
}

fun decrement(): Unit {
    size++
}
```

Тип ***Unit*** имеет единственное значение - объект ***Unit***. Это аналог типа ***void*** в Java.

Умалчиваемые аргументы функции

Для аргументов функций могут быть заданы умалчиваемые значения.

```
fun outMessage(text: String, kind: String = "Info") {
    println("$kind: $text")
}

outMessage("Start execution")

outMessage("Unknown identifier", "Error")
```

Таким образом можно избежать большого числа одноименных функций с различным количеством параметров.
Например, большого числа конструкторов.

Именованные аргументы функций

При вызове функции можно указывать имена параметров для которых задается значение.

```
fun getColor(red: Int = 0, green: Int = 0, blue: Int = 0) =
    "rgb($red,$green,$blue)"

val darkBlue = getColor(blue = 100)
val lightRed = getColor(red = 255)
val violet = getColor(blue = 100, red = 200)
```

Таким образом можно избежать ошибок при неверном порядке значений однотипных аргументов функции.

При вызове из программы на языке Kotlin методов на языке ***Java*** именованные аргументы использовать нельзя.

В ***Java6*** имена параметров в байткоде не хранятся.

Функции с переменным числом параметров

Модификатор ***vararg*** позволяет передавать в функцию переменное число параметров указанного типа.

```
fun printAll(vararg messages: String) {
    for (m in messages) println(m)
}

fun main() {
    printAll("Hello", "Kotlin")
}
```

Внутри функции с параметром ***message*** можно работать как с массивом ***Array***.

Функции с переменным числом параметров (2)

Для дальнейшей передачи параметра ***message*** в другую функцию используется оператор *.

```
fun printAll(vararg messages: String) {
    for (m in messages) println(m)
}

fun log(vararg entries: String) {
    printAll(*entries)
}

fun main() {
    printAll("Hello", "Kotlin")
    log("Hello", "Kotlin")
}
```

В этом случае он будет использоваться как ***vararg messages: String*** а не как ***messages: Array***

Функции-расширения

При объявлении функции-расширения перед именем должен быть тип получателя. Этот тип расширяется.

```
// Файл StringsUtil.kt:
package ch1.strings

fun String.lastChar(): Char = this.get(this.length - 1)
```

Ключевое слово **this** в функции расширения соответствует объекту-получателю. Его можно опустить:

```
fun String.lastChar(): Char = get(length - 1)
```

Импорт функций-расширений

Возможен импорт функций-расширений:

```
// Файл StringUtil.kt:
package strings

import strings.lastChar

val c = "Kotlin".lastChar()
```

Можно переименовать импортируемую функцию-расширение:

```
import strings.lastChar as last

val c = "Kotlin".last()
```

Использование функций-расширений из языка Java

Использование в программе на ***Java***
статического метода **get*LastChar*** из класса ***StringsUtilKt***
расположенного в файле ***StringsUtil.kt***

```
/* Java */
import strings.StringsUtilKt;

public class JavaMainFun {
    public static void main(String[] args) {
        StringsUtilKt.lastChar("Kotlin");
    }
}
```

Свойства-расширения для чтения

```
// Файл Properties.kt
package strings

val String.lastChar: Char
    get() = get(length - 1)

fun main() {
    val kotlin = "kotlin"
    println(kotlin.lastChar)
}
```

Свойства-расширения для чтения и записи

```
// Файл Properties.kt
package string

var Array<Char>.firstChar: Char
    get() = get(0)
    set(value: Char) = this.set(0, value)

fun main() {
    var kotlinArray = arrayOf('k', 'o', 't', 'l', 'i', 'n' )
    kotlinArray.firstChar = 'K'
    kotlinArray.forEach { print(it) }
}
```