ТЕХНОТРЕК

Занятие №2

Современная разработка под Android

Юрий Береза
Кирилл Филимонов

Напоминание

А ты отметился о присутствии на занятии?

Kotlin - Часть 1

1. Введение
2. Hello, Kotlin!
3. Функции
4. Переменные и типы
5. Лямбды
6. Null safety типы и вспомогательные операторы
7. Функции with, let, apply, run
8. Операторы циклов и условий

Введение

Проблемы Java на Android

- Java 8 на Android 7.0
- А “большая Java” на данный момент OpenJDK 11
- Сложные споры с Oracle
- Большие размеры Java библиотек (RxJava, Guava)

Что дает Kotlin

- Легкий переход с Java
- Прозрачное взаимодействие с текущим кодом
- Более компактный код
- Легкое обновление
- Небольшой размер стандартной библиотеки
- Синтаксический сахар
- Корутины

Почему Kotlin

Лаконичный

Безопасный

Прагматичный

Почему Kotlin

Лаконичный

Лаконичный

```java
public final class Person {
    private final String name;

    public Person(String value) {
        this.name = value;
    }

    public String getName() { return name; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o instanceof Person) {
            Person person = (Person) o;
            return Objects.equals(name, person.name);
        }
        return false;
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
}
```

Лаконичный

```
data class Person(val name : String)
```

Почему Kotlin

Безопасный

Безопасный

```
fun safeFunction(person : Person) {
    println(person.name)
}

fun almostSafeFunction(person: Person?) {
    person.let {
        println(it)
    }
}
```

Почему Kotlin

Прагматичный*

** Реалистичный подход к делу, основанный на практическом опыте, а не на теории*

Прагматичный

- Совместимость с Java
- Статическая типизация
- Лямбды, свойства, функции высшего порядка
- Функции расширения
- Компактный код
- Функциональный стиль программирования

Hello, Kotlin

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val text = findViewById<TextView>(R.id.text)
        text.text = "Hello, Android!"
    }
}
```

Hello, Kotlin

Demo!

Отличия функций от Java

- Функции, а не методы
- Могут иметь параметры по умолчанию
- Именованные аргументы
- Вложенные функций (локальные функции)

Функции

```
fun ИмяФункции(ИмяПеременной : Тип, ИмяВторой : Тип = Значение по умолчанию) : Тип
```

Объявление функций

```kotlin
// Это самое простое объявление функции
fun nothing() { }

// Однострочная функция
fun singleLineFunction() = "Single Line!"

// Функция, принимающая один параметр. Возвращает тип стринг
fun oneParamFunction(who : String) : String {
    return "Hello $who! Your Kotlin"
}

// Однострочная функция, принимающая два параметра
fun sum(a : Int, b : Int) = a + b

// Функция, имеющая параметр по умолчанию и возвращающая значение типа String
fun getGreetings(greetings : String = "Hello Android! Your Kotlin. ") : String {
    return greetings
}
```

Именованные аргументы

```
fun joinGreeting(greeting : String = "Hello", who : String = "Android", from :
String = "Kotlin") : String {
    val builder = StringBuilder()
            .append(greeting)
            .append(" ")
            .append(who)
            .append("!")
            .append(" From ")
            .append(from)
    return builder.toString()
}

fun greetingFromTechnotrack() = joinGreeting(from = "Technotrack", who = "students")
```

Вложенные функции

```
fun localFunctions(name : String) {

    fun validateOrThrow() {
        if (name.isEmpty() || name.equals("Ivanov")) {
            throw IllegalStateException("Incorrect name")
        }
    }

    validateOrThrow()
    //
    //do next thing
    //
    validateOrThrow()
}
```

Переменные и типы

- Все типы - объекты
- Однако хранятся в виде примитивных типов
- Отсутствие прямого приведения типов
- Вместо символов для битовых операций используются функции
- Массивы - только в виде класса Array

Тип	Размер
Long	64
Int	32
Short	16
Byte	8
Double	64
Float	32

Объявление переменных

```
fun declaration() {
    val a : Int = 0

    val b = 0

    var c : String = "Hello"

    val d = a == b // OK, same type

    c = "World!"

    b = 10 // Error
}

fun conversion() {

    val a : Int = 0
    val b : Long = 0

    val compairError =  a == b // Error
    val compair = a.toLong() == b // OK
}
```

Битовые операции

Kotlin	Java
shl(bits)	<<
shr(bits)	>>
ushr(bits)	>>>
and(bits)	&
or(bits)	|
xor(bits)	^
inv()	~

```
val x = (1 shl 2) and 0x000FF000
```

Массивы

```
fun arrays() {
    val stringArrayOfFive = Array(5, { i -> i.toString() })

    val IntArrayOfThree = arrayOf(1, 2, 3)

    IntArrayOfThree[0] = IntArrayOfThree[1] + IntArrayOfThree[2]
}

fun main(args : Array<String>) {
    println(args.size)
}
```

Лямбды

Лямбда-выражения - это небольшие фрагменты кода, которые можно передавать другим функциям. Благодаря поддержке лямбда-выражений вы легко сможете выделить общий код в библиотечные функции, и стандартная библиотека Kotlin активно этим пользуется.

Kotlin in Action

Лямбды - блок не делающий ничего

```
val nothing = { }

val details : () -> Unit = { }
```

Лямбды

```
// Функция принимает два параметра и лямбду (фрагмент кода),
// осуществляющий операцию преобразования
fun doOperation(x : Int, y : Int, transform : (Int, Int) -> Int) {
    val result = transform(x, y)
}

fun callOfLambda() {
    //Лямбда, которая на входе получает два числа возвращает их сумму
    val sum = { x: Int, y : Int -> x + y }

    //Лямбда, которая на входе получает два числа и печатает их значаение
    val printer = { x : Int, y : Int ->
        println(x)
        println(y)
        0
    }

    doOperation(1, 2, sum)
    doOperation(1, 2, printer)
}
```

Лямбды - работа с коллекциями

```
fun findOldest() {
    val persons = arrayOf(Person("Bob", 29),
                          Person("Alice", 20))

    val oldest = persons.maxBy({ p : Person -> p.age })

    println(oldest)
}
```

Лямбды - упрощенная нотация

```
val oldest = persons.maxBy { it.age }
```

Лямбды - упрощенная нотация

```
// Функция принимает два параметра и лямбду (фрагмент кода),
// осуществляющий операцию преобразования
fun doOperation(x : Int, y : Int, transform : (Int, Int) -> Int) {
    val result = transform(x, y)
}

fun simpleCallOfLambda() {
    doOperation(1, 2) { x, y ->
        x + y
    }
}
```

Лямбды - функциональные интерфейсы

```
public class Button {
    public void setOnClickListener(OnClickListener I) { ... }
}

public interface OnClickListener {
    void onClick(View v);
}

button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {

    }
});
```

Лямбды - функциональные интерфейсы

```
button.setOnClickListener { view -> view.setVisiblity(View.GONE) }
```

Лямбды - функциональные интерфейсы

```
button.setOnClickListener { view -> view.setVisiblity(View.GONE) }
```

Лямбды - функциональные интерфейсы

```
public class ThreadCall {
    void exec() {
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Hello, from background thread");
            }
        });
    }
}
```

Лямбды - функциональные интерфейсы

```
fun threadCall() {
    val thread = Thread {
        println("Hello from background thread")
    }

    val runnable = Runnable {
        println("Hello from background thread")
    }

    val secondThread = Thread(runnable)

    thread.start()
    secondThread.start()
}
```

Null safety

Изобретение нулевой ссылки в 1965 году я называю своей ошибкой в миллиард долларов. В то время я разрабатывал первую систему ссылочных типов на объектно-ориентированном языке (ALGOL W). Моя цель состояла в том, чтобы гарантировать, что использование ссылок может быть абсолютно безопасным, причем проверка выполнялась автоматически компилятором. Но я не мог удержаться от соблазна добавить нулевую ссылку, просто потому, что ее было так легко реализовать. Это привело к бесчисленным ошибкам, уязвимостям и сбоям системы, которые, вероятно, стоили разрушений и боли на миллиарды долларов.

Тони Хор. Английский ученый в области computer science. Известен за разработку Quicksort.

Null safety

```
fun nullSafeError() : String {
    return null // Error
}

fun nullSafe() : String {
    return "" // OK
}

fun nullUnsafe() : String? {
    return null // OK
}

fun sample() {
    val name : String = nullSafe() // OK

    val surname : String = nullUnsafe() // Error

    val lastname : String? = nullUnsafe() // OK

    val len = lastname.length // Error

    val checkedLen = lastname?.length // OK

    val dl = if (lastname != null) lastname.length else 42 // OK

    val defaultLen = lastname?.length ?: 42 // OK
}
```

Функции run и with

```
fun withOrRun() {
    val lastVal : Int? = with(HashMap<String, Int>()) {
        put("Hello", 1)
        put("Android", 2)
    }

    val added : Boolean = ArrayList<Int>().run {
        add(42)
        add(100)
        add(200)
    }
}

fun withNullable(map : HashMap<String, Int>?) {
    with(map) {
        this?.put("Bye", 3)
        this?.put("Kotlin", 4)
    }

    map?.run {
        put("Key", 42)
        put("Value", 51)
    }
}
```

if - блок

```
public class JavaConditions {
    public static void conditionExpression(boolean hello) {
        String message = null;
        if (hello) {
            message = new StringBuilder("Hello").
                    append(", ").
                    append(" world!").
                    toString();
        }
        else {
            message = new StringBuilder("Bye-bye").
                    append(", ").
                    append(" world!").
                    toString();
        }
    }
}
```

if - выражение

```
fun conditionExpression(hello : Boolean) {
    val message = if (hello) {
        StringBuilder("Hello").
                append(", ").
                append(" world!").
                toString()
    }
    else {
        StringBuilder("Bye-bye").
                append(", ").
                append(" world!").
                toString()
    }
}
```

when

```
fun getMnemonicfcolor(color : Color) : String {
    val text : String = when (color) {
        Color.RED -> "Каждый"
        Color.ORANGE -> "Охотник"
        Color.YELLOW -> "Желает"
        Color.GREEN -> "Знать"
        Color.BLUE -> "Где"
        Color.INDIGO -> "Сидит"
        Color.VIOLET -> "Фазан"

    }
    return text
}
```

when

```
fun getMnemonicfcolor(color : Color) = when (color) {
    Color.RED -> "Каждый"
    Color.ORANGE -> "Охотник"
    Color.YELLOW -> "Желает"
    Color.GREEN -> "Знать"
    Color.BLUE -> "Где"
    Color.INDIGO -> "Сидит"
    Color.VIOLET -> "Фазан"
}
```

when как замена if

```
fun whenAsIfReplacement(c1 : Color, c2 : Color) = {
    when {
        (c1 == Color.RED && c2 == Color.YELLOW) ||
                (c1 == Color.YELLOW && c2 == Color.RED) -> Color.ORANGE
        (c1 == Color.YELLOW && c2== Color.BLUE) ||
                (c1 == Color.BLUE && c2 == Color.YELLOW) -> Color.GREEN
        (c1 == Color.BLUE && c2 == Color.VIOLET) ||
                (c1 == Color.VIOLET && c2 == Color.BLUE) -> Color.INDIGO
        else -> throw Exception("Dirty color")
    }
}
```

Цикл for

- Цикл for в Kotlin аналогичен for-each в Java
- Привычного for (int i = 0; i < 10; ++i) в Kotlin нет

```
fun simpleLoop() {
    val colors = arrayOf("red", "green", "blue")

    for (color in colors) {
        println(color)
    }
}
```

Цикл for с индексами

```
fun simpleLoopWithIndexes() {
    val colors = arrayOf("red", "green", "blue")

    for ((index, color) in colors.withIndex()) {
        println("$color at index $index")
        //red at index 0
        //green at index 1
        //blue at index 2
    }
}
```

Цикл for с диапазоном

```
fun simpleLoopWithRange() {
    val colors = arrayOf("red", "green", "blue")

    for (i in 0..colors.size - 1) {
        println("${colors[i]} at index $i")
        //red at index 0
        //green at index 1
        //blue at index 2
    }
}
```

Цикл for до границ диапазона

```
fun simpleLoopUtil() {
    val colors = arrayOf("red", "green", "blue")

    for (i in 0 until colors.size) {
        println("${colors[i]} at index $i")
        //red at index 0
        //green at index 1
        //blue at index 2
    }
}
```

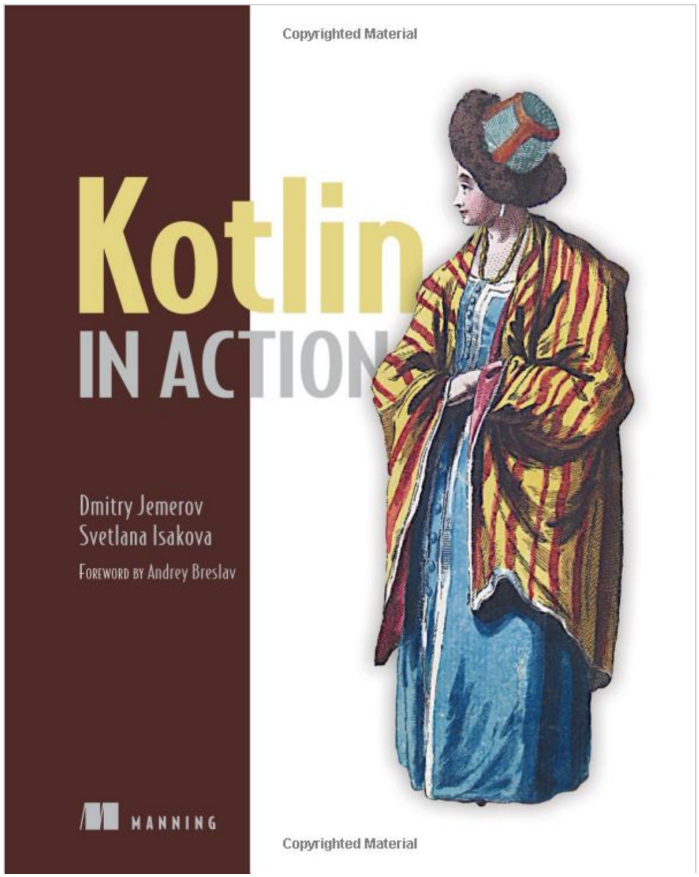
Цикл for в обратном порядке

```
fun simpleLoopDownto() {
    val colors = arrayOf("red", "green", "blue")

    for (i in colors.size - 1 downTo 0 step 1) {
        println("${colors[i]} at index $i")
        //blue at index 2
        //green at index 1
        //red at index 0
    }
}
```

Дополнительная информация

https://kotlinlang.org/docs/reference/



ТЕХНОТРЕК

Спасибо за внимание!

Юрий Береза
Кирилл Филимонов