Kotlin

Основы языка

Содержание

1. Методы

Итак, давайте вернемся в самое начало. Мы написали нашу первую программу на джаве и она выглядела вот так

```
public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Как же теперь это все написать на Котлин? У нас есть магическая комбинация клавиш которая сделает все за нас – Ctrl+Alt+Shift+K. И вуаля! Котлин код:

```
object Main {

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

Во-первых заметьте что пропало слово class и вместо него пришло object. Позже я обьясню суть этого ключевого слова, но пока просто имейте ввиду – если в вашем классе статическая функция, то она должна лежать в object. Вообще самое интересное что в самом Котлин нет статических методов и т.д. самого слова статик нет. Потому и мы используем аннотацию @JvmStatic что значит – статика из джава. Да-да, все верно – котлин это не новый язык полностью – мы переиспользуем многие вещи из джавы. Именно поэтому я и настоятельно рекомендую ознакомиться с джавой перед тем как учить котлин. Итак, ладно. Разобрались. Что дальше? Вместо public static void main(String[] args) у нас возникло fun main(args:Array<String>).
Нет, в котлин есть модификаторы доступа по крайней мере private, а для паблик методов мы не пишем слово паблик. Это и так ясно. У-упрощение.
Идем дальше – слово fun нужно писать в обьявлении любой функции. Далее у нас идет аргумент. Если в джава мы писали сначала тип аргумента и после уже имя, то в котлин мы делаем с точностью наоборот. Сначала имя аргумента и потом тип, но перед этим ставим

знак “:”. И да, в котлин можно не писать void → если ваша функция не возвращает ничего, то ничего и не пишем. Чуть ниже посмотрим пример. И да, вместо джавовского String[] у нас котлиновский Array<T>. Фигурные скобки остались и у нас вместо System.out.println сразу println. Но обратите внимание, что в конце линии нет ;. Да, мы можем поставить ; и ничего страшного не произойдет, но оно не нужно больше. Минус одна ошибка при компиляции.

Ладно, давайте подробней остановимся на методах. Для этого напишем метод суммы 2 чисел.

```
object Main {

    @JvmStatic
    fun main(args: Array<String>) {
        println(sum( a: 1,  b: 2))
    }

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

Итак, самое интересное – мы вызвали не статик метод в статик мейне. Как так? А вот так. В котлин нет статик вообще. Аннотация @JvmStatic нужна чтобы запустить мейн метод.
И опять же, давайте посмотрим на метод суммы – private fun sum приватная функция суммы, принимает 2 инта – да, в Котлин пишем Int, и пишем возвращаемый тип : Int в конце функции. Внутри собственно return a+b. Но котлин не был бы таким обожаемым языком программирования если бы не дал упростить нам жизнь.

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

Так как у нас простая функция в 1 линию, то можно сделать так – вместо фигурных скобок сразу равно и не писать return. Но и это еще не все упрощения. Посмотрите-ка на это.

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

Компилятор котлина видит что у нас на входе 2 числа и может сам понять какой будет результат. Да, можно явно не указывать возвращаемый тип.
А давайте усложним себе жизнь и проверим что сумма чисел равна 10.

```
private fun sum(a: Int, b: Int) = a + b == 10
```

Конечно же надо было поменять еще и имя функции, но как видите никаких проблем с возвращаемым типом. Все автоматически.

Ладно, с функциями вроде разобрались, что насчет переменных?

2. Переменные

Переменные в котлин тоже пишутся в обратном порядке. И да, иногда можно не указывать их явный тип.

Ключевое слово var используется для обьявление переменной (variable). Это точно такая же переменная как и в джава. Но помните у нас в джава можно было обьявить переменную, которую нельзя переопределить? Точно так же есть и в Котлин.

```
@JvmStatic
fun main(args: Array<String>) {
    var str = "string"
    var line: String
    var text: String = "text"
}
```

```
@JvmStatic
fun main(args: Array<String>) {
    var str = "string"
    str = "other"

    val line = ""
    line = ""

}
```

Удобно, неправда ли? Вместо того чтобы писать final String line пишем сразу val line. И да, как видите компилятор подчеркнул переопределение.

Ну и последнее на сегодня – константы. Помните мы писали public static final String URL = “”

Теперь в котлин мы можем делать это короче.

```
*/
object Main {

    const val URL = "https://www.google.com"
```

Так что да, отвечая на вопрос – в чем первое и главное преимущество Котлина – краткость. В последующих лекциях мы увидим еще примеры которые доказывают это.

И на последок скажу пару слов про котлин – зачем же мы учили джава? Нельзя было сразу начать с котлина? Нет, дело в том, что мы в дальнейшем будем писать код под андроид и там огромное количество исходного кода написанного на джава. Его по крайней мере нужно читать и понимать. А писать вы можете сразу на джава. Но если вы не знаете основ джава, то как мне кажется писать на котлин без понимания сути нельзя. Да и многие классы в котлин это те же джава классы но в обертке.

Все ошибки из джава перекочевали в котлин например. Но про ошибки мы поговорим позже.

```
@SinceKotlin( version: "1.1") public actual typealias Error = java.lang.Error
@SinceKotlin( version: "1.1") public actual typealias Exception = java.lang.Exception
@SinceKotlin( version: "1.1") public actual typealias RuntimeException = java.lang.RuntimeException
@SinceKotlin( version: "1.1") public actual typealias IllegalArgumentException = java.lang.IllegalArgumentException
@SinceKotlin( version: "1.1") public actual typealias IllegalStateException = java.lang.IllegalStateException
@SinceKotlin( version: "1.1") public actual typealias IndexOutOfBoundsException = java.lang.IndexOutOfBoundsException
@SinceKotlin( version: "1.1") public actual typealias UnsupportedOperationException = java.lang.UnsupportedOperationException
```

А в качестве задания попробуйте переписать некий старый код на котлин.

3. Продвинутое использование методов

Давайте рассмотрим такую штуку в джава – есть метод с аргументами. И мы понимаем что вызываем его в коде много раз передавая одно и то же значение, лишь изредка передавая иное значение. Как бы мы могли решить эту задачу в джава?

```
public static void log(String heading, String body) {
    if (!heading.isEmpty()) {
        System.out.println(heading);
    }
    if (!body.isEmpty()) {
        System.out.println(body);
    }
}

public static void log(String heading) {
    log(heading,  body: "");
}
```

Пусть у нас метод принимает 2 строки и если они не пустые то выводит в консоль. Теперь, мы понимаем, что нам нужен метод с 1 аргументом, но у нас основная логика написана уже в методе с 2 аргументами. Значит мы можем вызвать главный метод из второго. Просто передав ему туда пустую строку. А теперь подумайте над тем, сколько перегруженных методов вам нужно если у вас 3 аргумента. Хотя да, мы говорим что чем меньше аргументов тем лучше, но бывает так, что код написан не нами и приходится иметь дело с тем что есть.

Как бы это сделали в котлин? Есть такая штука как дефолтные аргументы.

```
@JvmStatic
fun main(args: Array<String>) {
    log( header: "header1")
    log( header: "header2",  body: "body2")
}

fun log(header: String, body: String = "") {
    if (header.isNotEmpty()) {
        print(header)
    }
    if (body.isNotEmpty()) {
        print(body)
    }
}
```

Итого у нас 1 метод с дефолтными аргументами и можно не перегружать методы. Но и это не все. В котлин есть еще такая штука как выборочная передача аргументов.

Давайте посмотрим о чем речь.

```
@JvmStatic
fun main(args: Array<String>) {
    log( header: "header1",  body: "",  footer: "footer")
    log( header: "header2", footer = "footer2")
}

fun log(header: String, body: String = "", footer: String) {
    if (header.isNotEmpty()) {
        print(header)
    }
    if (body.isNotEmpty()) {
        print(body)
    }
    if (footer.isNotEmpty()) {
        print(footer)
    }
}
```

Итак, у нас 3 аргумента. Первый и последний обязательны, а второй имеет дефолтное значение. Теперь если мы будем вызывать этот метод, то надо передать второму аргументу пустую строку и это не круто. Для этого можно указать имя аргумента и сразу передать значение минуя второй аргумент. Сравните оба вызова в мейне.

И да, вы заметили метод проверки что строка не пустая? Он сразу есть, не нужно как в джава проверять на пустоту и потом отрицать условие через !. Удобно.

И давайте вспомним передачу любого количества аргументов. В джава это было так

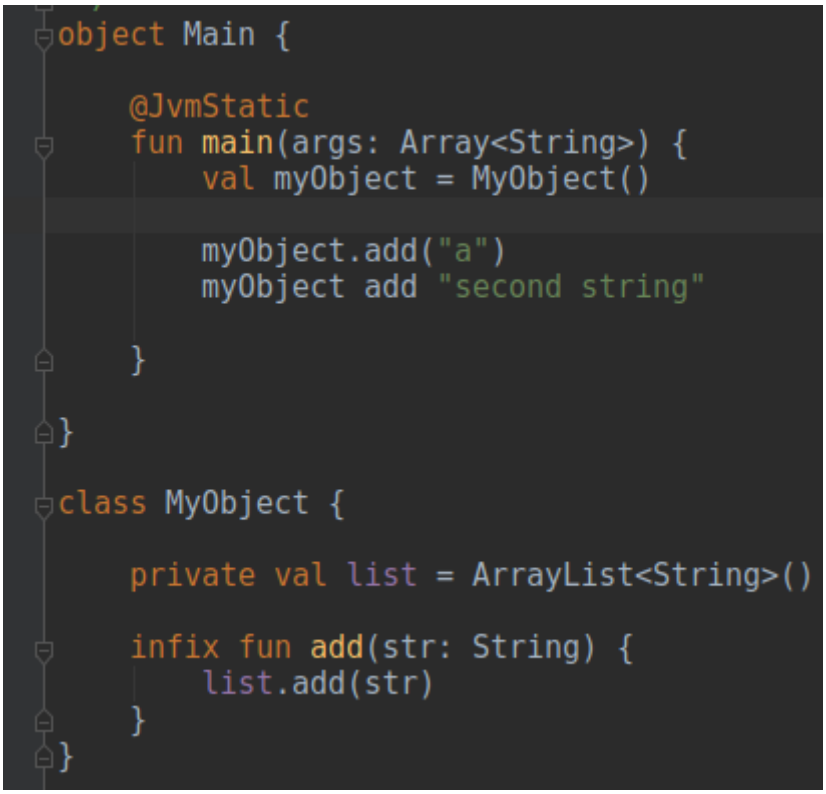
```
public static void print(String... args) {
    for (String string : args) {
        System.out.println(string);
    }
}
```

То же самое в котлин будет вот так

```
@JvmStatic
fun main(args: Array<String>) {
    print("")
    print("a", "b")
}

private fun print(vararg args: String) {
    for (string in args) {
        println(string)
    }
}
```

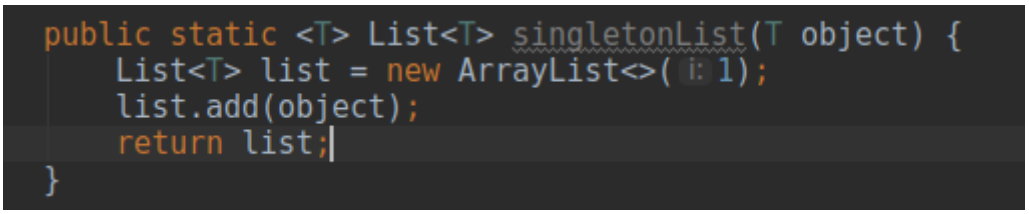
Еще одна забавная особеннось котлина (да, в котлине очень много сахара – удобных штук которые облегачают жизнь программисту) это инфикс функции.



Для начала мы написали класс MyObject в котором есть поле типа списка и инфикс функция добавления в этот список новой строки. Как видите в мейне вызвать эту инфикс функцию можно 2 способами – как обычную и диковинным способом – через пробелы. Это делает код красивым. Насколько вам удобно такое читать думайте сами. Вместо точки и скобок пробелы.

Вы возможно не будете писать код с инфикс функциями, но вам по крайней мере нужно знать о существовании таковых чтобы при чтении котлин кода другого программиста не теряться.

И еще одно – помните в джава были дженерики? Мы использовали их в списках, но так же может быть метод с использованием дженериков. Точно так и в котлине они есть.



Мы написали метод, который работает с любым классом и отдает список из 1 элемента. Т.е. он принимает любой обьект и отдает список содержащий этот обьект. Да, у вас может быть класс который не типизирован, но у него может быть 1 метод с дженериком. Теперь то же самое в котлин будет:

Заметьте, что имя аргумента object обрамлено с 2 сторон символами `object`. Это все потому что в котлине это слово ключевое и его нельзя использовать просто так. Эти 2 черточки

делают из ключевого слова в котлин просто имя аргумента. Кстати запомните этот пример, мы к нему вернемся позднее.

```
fun <T> singletonList(`object`: T): List<T> {
    val list = ArrayList<T>( p0: 1)
    list.add(`object`)
    return list
}
```

И на последок посмотрим на рекурсивные функции.

```
public class JavaClass {

    private static void print(String string, int times) {
        if (times == 0) {
            System.out.println("finished");
        } else {
            System.out.println(string);
            print(string, --times);
        }
    }

    public static void main(String[] args) {
        print( string: "str",  times: Integer.MAX_VALUE / 1000);
    }
}
```

Здесь мы печатаем строку в консоль столько раз пока аргумент не уменьшится до нуля. Но посмотрите на вызов, мы передаем самое большое число из Integer. Для простоты я уменьшил его в 1000 раз. Если вы запустите этот код, то он не завершится успешно. Так как будет ошибка вызовов.

А теперь посмотрим на тот же код в котлин

```
    @JvmStatic
    fun main(args: Array<String>) {
        printIt( str: "text",  times: Int.MAX_VALUE / 1000)
    }


    private tailrec fun printIt(str: String, times: Int) {
        if (times == 0) {
            println("done")
        } else {
            println(str + times)
            printIt(str,  times: times - 1)
        }
    }
}
```

Если запустите этот код, то все будет удачно завершено. В этом суть тейлрек функции. Она не

вызывает ошибки вызовов. Кстати, еще одна важная штука по поводу аргумента функции. В джава мы можем менять аргумент функции как нам угодно. Если же не написали final. А в котлин по умолчанию аргумент функции это val. И поэтому мы вызываем рекурсивню функцию передав туда times – 1, а не - - times как в джава.

И последнее на сегодня – экстеншн функции в котлин. В джава нет аналога, но я расскажу в чем суть проблемы. Помните мы написали наш класс обертку SafeList где все методы безопасно вызывались и не позволяли класть в список null и копии обьектов? То же самое можно сделать в котлин без создания класса обертки. Смотрите.

Мы написали функцию addItem которая принадлежит классу MutableList (да, в Котлин есть 2 вида списков – неизменяемый и изменяемый, мутабельный и обычный). И теперь можем вызывать наш метод так, как будто он был написан создателями котлина. Как видите теперь у нас не будет дубликатов и для этого не нужен класс обертка.

Вот вам задание – все методы из класса SafeList напишите экстеншн функциями в котлин и проверьте их. Так же найдите код в котором использовали рекурсию и напишите на котлин.

Так же опробуйте все фичи функций в котлине – инфиксы, варарги и т.д.