

Содержание

Глава 1. Начало работы с Kotlin	11
Примечания	11
Компиляция Kotlin.....	11
Версии.....	12
Примеры	12
Hello World.....	12
Hello World с использованием Object Declaration.....	13
Hello World с использованием Companion Object.....	14
Основные методы с использованием varargs.....	15
Компиляция и запуск кода Kotlin в командной строке	15
Чтение ввода из командной строки.....	15
Глава 2. Аннотации	17
Примеры	17
Объявление аннотации.....	17
Мета-аннотации.....	17
Глава 3. Массивы	19
Примеры	19
Обобщенные массивы (Generic Arrays)	19
Массивы примитивов (Arrays of Primitives)	19
Расширения (Extensions)	20
Итерация по массиву (Iterate Array)	20
Создание массива	21
Создание массива с использованием замыкания (closure)	21
Создание неинициализированного массива.....	21
Глава 4. Основы лямбда-выражений	22
Синтаксис	22
Примечания	22
Примеры	23
Лямбда как параметр функции filter	23
Лямбда, передаваемая как переменная	23
Лямбда для измерения времени выполнения функции	23
Глава 5. Основы Kotlin	24
Введение	24
Примечания	24
Примеры	24
Базовые примеры	24

Глава 6. Делегирование классов	26
Введение	26
Примеры	26
Делегирование метода другому классу	26
Глава 7. Наследование классов	27
Введение	27
Синтаксис	27
Параметры	27
Примеры	28
Основы: ключевое слово <code>open</code>	28
Наследование полей от класса	28
Определение базового класса	28
Определение производного класса	28
Использование подкласса	29
Наследование методов от класса	29
Определение базового класса	29
Определение производного класса	29
Ninja имеет доступ ко всем методам в <code>Person</code>	29
Переопределение свойств и методов	29
Переопределение свойств (как только для чтения, так и изменяемых)	29
Переопределение методов	30
Глава 8. Коллекции	31
Введение	31
Синтаксис	31
Примеры	31
Использование списка (<code>list</code>)	31
Использование <code>map</code>	32
Использование множества (<code>set</code>)	32
Глава 9. Условные инструкции	33
Примечания	33
Примеры	33
Стандартное выражение <code>if</code>	33
<code>if</code> как выражение	33
Инструкция <code>when</code> вместо цепочки <code>if-else-if</code>	34
Сопоставление аргументов в инструкции <code>when</code>	35
Инструкция <code>when</code> как выражение	35
Инструкция <code>when</code> с перечислениями (<code>enum</code>)	35
Глава 10. Настройка сборки Kotlin	37
Примеры	37
Конфигурация Gradle	37
Для JVM	37
Для Android	37
Для JS	38
Использование Android Studio	38

Установка плагина	38
Настройка проекта	39
Преобразование Java	39
Миграция с Gradle с использованием Groovy-скриптов на Kotlin-скрипты	39
Глава 11. Корутины.....	41
Введение	41
Примеры	41
Глава 12. Делегированные свойства	42
Введение	42
Примеры	42
«Ленивая» инициализация.....	42
Наблюдаемые свойства	42
Свойства, поддерживаемые map	42
Кастомное делегирование.....	43
Делегат может использоваться для уменьшения шаблонного кода.....	43
Глава 13. Создание DSL	45
Введение	45
Примеры	45
Инфиксный подход для создания DSL	45
Переопределение метода invoke для создания DSL.....	46
Использование операторов с лямбдами	46
Использование расширений с лямбдами	46
Глава 14. Перечисления (Enum)	47
Примечания	47
Примеры	47
Инициализация	47
Функции и свойства в перечислениях.....	48
Простое перечисление.....	48
Изменяемость.....	48
Глава 15. Исключения (Exceptions)	49
Примеры	49
Перехват исключений с помощью try-catch-finally	49
Глава 16. Методы расширения (Extension Methods).....	50
Синтаксис.....	50
Примечания	50
Примеры	50
Расширения верхнего уровня.....	50
Потенциальная ловушка: расширения разрешаются статически.....	51
Пример расширения для типа Long для вывода читаемой строки	51
Пример расширения класса Java 7+ Path	52
Использование функций расширения для улучшения читаемости.....	52
Пример расширения классов Java 8 Temporal для вывода строки в формате ISO.....	53

Функции расширения для Companion Objects (имитация статических функций).....	53
Обходной путь для «ленивых» свойств расширения	54
Расширения для упрощения ссылок на View из кода	54
Расширения.....	55
Использование.....	55
Глава 17. Функции	56
Синтаксис.....	56
Параметры.....	56
Примеры	57
Функции, принимающие другие функции	57
Лямбда-функции	57
Ссылки на функции.....	58
Базовые функции.....	60
Сокращенные функции.....	60
Встроенные функции (Inline Functions)	60
Функции-операторы.....	61
Глава 18. Обобщения (Generics)	62
Введение	62
Синтаксис.....	62
Параметры	62
Примечания	63
Подразумеваемая верхняя граница — Nullable	63
Примеры	63
Вариативность на месте объявления (Declaration-site variance).....	63
Вариативность на месте использования (Use-site variance).....	64
Глава 19. Идиомы.....	65
Примеры	65
Создание DTO (POJO/POCO)	65
Фильтрация списка	65
Делегирование классу без предоставления его в публичном конструкторе	66
Serializable и serialVersionUID в Kotlin	66
Fluent-методы в Kotlin	67
Использование let или also для упрощения работы с nullable-объектами	67
Использование apply для инициализации объектов или для цепочки вызовов.....	68
Глава 20. Интерфейсы.....	69
Замечания	69
Примеры	69
Базовый интерфейс.....	69
Интерфейс с реализациями по умолчанию	69
Свойства.....	70
Множественные реализации	70

Свойства в интерфейсах.....	71
Конфликты при реализации нескольких интерфейсов с реализациями по умолчанию	72
Ключевое слово <code>super</code>	72
Глава 21. Эквиваленты <code>Stream</code> из <code>Java 8</code>.....	73
Введение	73
Замечания.....	73
О «ленивости» (<code>laziness</code>)	73
Почему нет типов?!	73
Повторное использование <code>Stream</code>	74
Примеры.....	75
Накопление имен в списке	75
Преобразование элементов в строки и их объединение через запятую.....	75
Вычисление суммы зарплат сотрудников	76
Группировка сотрудников по отделам	76
Вычисление суммы зарплат по отделам	76
Разделение студентов на сдавших и не сдавших экзамен	76
Имена участников мужского пола.....	77
Группировка имен участников по полу	77
Фильтрация списка в другой список.....	77
Поиск самой короткой строки в списке	77
Различные виды <code>Stream</code> #2 — «ленивое» использование первого элемента, если он существует.....	78
Различные виды <code>Stream</code> #3 — итерация по диапазону целых чисел.....	78
Различные виды <code>Stream</code> #4 — итерация по массиву, преобразование значений, вычисление среднего	78
Различные виды <code>Stream</code> #5 — «ленивая» итерация по списку строк, преобразование значений, конвертация в <code>Int</code> , поиск максимума	78
Различные виды <code>Stream</code> #6 — «ленивая» итерация по потоку <code>Int</code> , преобразование значений, вывод результатов	79
Различные виды <code>Stream</code> #7 — «ленивая» итерация по <code>Doubles</code> , преобразование в <code>Int</code> , преобразование в <code>String</code> , вывод каждого	79
Подсчет элементов в списке после применения фильтра.....	79
Как работают потоки — фильтрация, преобразование в верхний регистр, затем сортировка списка	80
Различные виды <code>Stream</code> #1 — активное использование первого элемента, если он существует.....	80
Пример сбора данных #5 — поиск людей совершеннолетнего возраста, вывод форматированной строки.....	81
Пример сбора данных #6 — группировка людей по возрасту, вывод возраста и имен вместе	82
Пример сбора данных #7a — преобразование имен, объединение с разделителем.....	83
Пример сбора данных #7b — сбор с использованием <code>SummarizingInt</code>	84
Глава 22. <code>JUnit</code>	86
Примеры.....	86
Правила	86

Глава 23. Kotlin Android Extensions	87
Введение	87
Примеры	87
Настройка	87
Использование представлений.....	87
Варианты продукта (Product Flavors).....	88
Painful listener для получения уведомлений, когда вид (view) полностью отрисован, теперь стал простым и впечатляющим	89
Глава 24. Подводные камни Kotlin	90
Примеры	90
Вызов toString() на nullable-типе	90
Глава 25. Kotlin для Java-разработчиков	91
Введение	91
Примеры	91
Объявление переменных	91
Интересные факты	91
Равенство и идентичность.....	92
IF, TRY и другие являются выражениями, а не операторами.....	93
Глава 26. Логирование (logging) в Kotlin	94
Примечание	94
Примеры	94
Использование kotlin.logging.....	94
Глава 27. Циклы в Kotlin	95
Примечание	95
Примеры	95
Повторение действия x раз	95
Итерация по коллекциям	95
Циклы while	96
Break и continue	96
Итерация по Map в Kotlin.....	97
Рекурсия	97
Функциональные конструкции для итерации	97
Глава 28. Null Safety (Null безопасность)	98
Примеры	98
Nullable и Non-Nullable типы.....	98
Оператор безопасного вызова (Safe Call Operator)	98
Идиома: вызов нескольких методов на одном и том же объекте с проверкой на null.....	98
Умные приведения (Smart Casts).....	99
Удаление null из Iterable и массива	99
Оператор объединения null / оператор Элвис (Null Coalescing / Elvis Operator)	99
Утверждение (Assertion).....	100
Оператор Элвис (?):.....	100

Глава 29. Диапазоны (Ranges)	101
Введение	101
Примеры	101
Диапазоны целочисленных типов (Integral Type Ranges).....	101
Функция <code>downTo()</code>	101
Функция <code>step()</code>	101
Функция <code>until</code>	102
Глава 30. RecyclerView в Kotlin	103
Примеры	103
Основной класс и адаптер.....	103
Глава 31. Рефлексия (Reflection)	105
Введение	105
Примечания	105
Примеры	105
Ссылка на класс	105
Ссылка на функцию	105
Взаимодействие с Java-рефлексией	106
Получение значений всех свойств класса	106
Установка значений всех свойств класса.....	107
Глава 32. Регулярные выражения (Regex)	109
Примеры	109
Идиомы для сопоставления с регулярными выражениями в выражении <code>when</code>	109
Использование неизменяемых локальных переменных (<code>immutable locals</code>).....	109
Использование анонимных временных переменных (<code>anopymous temporaries</code>).....	109
Использование шаблона «Посетитель» (<code>Visitor Pattern</code>).....	110
Введение в регулярные выражения в Kotlin	110
Класс <code>Regex</code>	111
Работа с <code>null</code> -безопасностью в регулярных выражениях.....	111
«Сырые» строки (<code>raw strings</code>) в шаблонах регулярных выражений.....	111
Функция <code>find(input: CharSequence, startIndex: Int): MatchResult?</code>	112
Функция <code>findAll(input: CharSequence, startIndex: Int): Sequence<MatchResult></code>	112
Функция <code>matchEntire(input: CharSequence): MatchResult?</code>	112
Функция <code>matches(input: CharSequence): Boolean</code>	113
Функция <code>containsMatchIn(input: CharSequence): Boolean</code>	113
Функция <code>split(input: CharSequence, limit: Int): List<String></code>	113
Функция <code>replace(input: CharSequence, replacement: String): String</code>	113
Глава 33. Объекты-одиночки (Singleton objects)	114
Введение	114
Примеры	114
Использование в качестве замены статических методов / полей Java	114
Использование в качестве синглтона	114

Глава 34. Строки (Strings)	116
Примеры	116
Элементы строки (Elements of String)	116
Строковые литералы (String Literals)	116
Шаблоны строк (String Templates)	117
Равенство строк (String Equality)	118
Глава 35. Псевдонимы типов (Type Aliases)	119
Введение	119
Синтаксис	119
Примечание	119
Примеры	119
Функциональный тип (Function Type)	119
Обобщенный тип (Generic Type)	119
Глава 36. Типобезопасные строители (Type-Safe Builders)	120
Замечания	120
Типичная структура типобезопасного строителя	120
Типобезопасные строители в библиотеках Kotlin	120
Примеры	120
Строитель типобезопасной древовидной структуры	120
Глава 37. Параметры с переменным количеством аргументов (Vararg Parameters) в функциях	122
Синтаксис	122
Примеры	122
Основы: использование ключевого слова vararg	122
Оператор spread: передача массивов в функции с vararg	123
Глава 38. Модификаторы видимости (Visibility Modifiers)	124
Введение	124
Синтаксис	124
Примеры	124
Пример кода	124
Благодарности	125

Глава 1.

Начало работы с Kotlin

Примечания

Kotlin — это статически типизированный объектно-ориентированный язык программирования, разработанный JetBrains, в первую очередь ориентированный на JVM (Java Virtual Machine). Kotlin разработан с целями быстрой компиляции, обратной совместимости, высокой типобезопасности и стопроцентной совместимости с Java. Kotlin также создан с целью предоставления многих функций, востребованных у разработчиков на языке Java. Стандартный компилятор Kotlin позволяет компилировать код как в байт-код Java для JVM, так и в JavaScript.

Компиляция Kotlin

Kotlin имеет стандартный IDE-плагин для Eclipse и IntelliJ. Kotlin также можно компилировать с использованием Maven, Ant, Gradle или через командную строку.

Стоит отметить, что команда `$ kotlinc Main.kt` создаст Java-класс файл, в данном случае `MainKt.class` (обратите внимание на добавление `Kt` к имени класса). Однако если попытаться запустить этот класс с помощью команды `$ java MainKt`, Java выдаст следующее исключение:

```
Exception in thread "main" java.lang.NoClassDefFoundError: kotlin/
jvm/internal/Intrinsics
    at MainKt.main(Main.kt)
Caused by: java.lang.ClassNotFoundException: kotlin.jvm.internal.
Intrinsics
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.
java:335)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    ... 1 more
```

Для того чтобы запустить полученный класс с помощью Java, необходимо добавить JAR-файл среды выполнения Kotlin в текущий classpath:

```
java -cp ./path/to/kotlin/runtime/jar/kotlin-runtime.jar MainKt
```

Для JS

apply plugin: 'kotlin2js'

По умолчанию используются следующие пути:

- Исходники Kotlin: src/main/kotlin
- Исходники Java: src/main/java
- Тесты Kotlin: src/test/kotlin
- Тесты Java: src/test/java
- Ресурсы времени выполнения: src/main/resources
- Ресурсы тестов: src/test/resources

Если вы используете нестандартную структуру проекта, может потребоваться настройка SourceSets. Подробнее о SourceSets вы можете прочитать, перейдя по ссылке <https://docs.gradle.org/current/dsl/org.gradle.api.tasks.SourceSet.html> или отсканировав QR-код справа:



Наконец, нужно добавить зависимость от стандартной библиотеки Kotlin в ваш проект:

```
dependencies {  
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"  
}
```

Если вы хотите использовать рефлексию Kotlin, также добавьте:

```
compile "org.jetbrains.kotlin:kotlin-reflect:$kotlin_version"
```

Использование Android Studio

Android Studio может автоматически настроить Kotlin в Android-проекте.

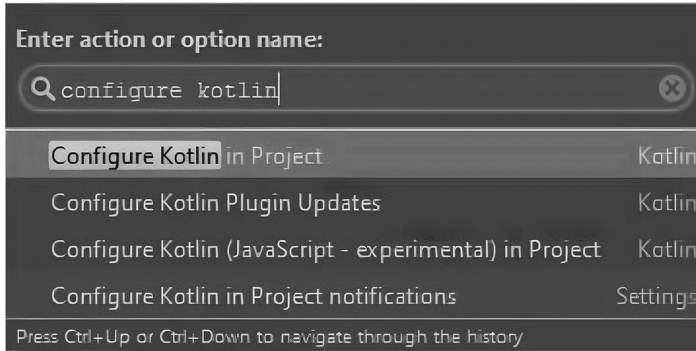
Установка плагина

Чтобы установить плагин Kotlin, перейдите в File > Settings > Editor > Plugins > Install JetBrains Plugin... > Kotlin > Install.

Затем перезапустите Android Studio, когда появится запрос.

Настройка проекта

Создайте проект в Android Studio как обычно, затем нажмите **Ctrl + Shift + A**. В поисковой строке введите «Configure Kotlin in Project» и нажмите **Enter**.



Android Studio изменит ваши файлы Gradle, чтобы добавить все необходимые зависимости.

Преобразование Java

Чтобы преобразовать Java-файлы в Kotlin-файлы, нажмите **Ctrl + Shift + A** и найдите «Convert Java File to Kotlin File». Это изменит расширение текущего файла на `.kt` и преобразует код в Kotlin.

```
package com.orangeflash81.myapplication;

public class Foo {
    private String name = "Joe Bloggs";

    String getName() { return name; }

    void setName(String value) { name = value; }
}
```

Миграция с Gradle с использованием Groovy-скриптов на Kotlin-скрипты

Шаги:

1. Клонировать проект `gradle-script-kotlin`.
2. Скопировать / вставить из клонированного проекта в ваш проект:

Как работают потоки — фильтрация, преобразование в верхний регистр, затем сортировка списка

```
// Java:
List<String> myList = Arrays.asList("a1", "a2", "b1", "c2", "c1");

myList.stream()
    .filter(s -> s.startsWith("c"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);

// C1
// C2

// Kotlin:
val list = listOf("a1", "a2", "b1", "c2", "c1")
list.filter { it.startsWith('c') }.map(String::toUpperCase).
sorted()
    .forEach(::println)
```

Различные виды Stream #1 — активное использование первого элемента, если он существует

```
// Java:
Arrays.asList("a1", "a2", "a3")
    .stream()
    .findFirst()
    .ifPresent(System.out::println);

// Kotlin:
listOf("a1", "a2", "a3").firstOrNull()?.apply(::println)

...или создайте функцию-расширение для String с именем ifPresent:

// Kotlin:
inline fun String?.ifPresent(thenDo: (String)->Unit) = this?.apply
{ thenDo(this) }

// теперь используйте новую функцию-расширение:
listOf("a1", "a2", "a3").firstOrNull().ifPresent(::println)
```

Смотрите также:

Более подробная информация про функции-расширения (Extension Functions) доступна по ссылке <https://kotlinlang.org/docs/extensions.html> или по QR-коду справа:



Более подробная информация про оператор безопасного вызова ?. доступна по ссылке <https://kotlinlang.org/docs/null-safety.html#safe-calls> или по QR-коду справа:



Более подробная информация о nullability доступна по ссылке <https://stackoverflow.com/questions/34498562/in-kotlin-what-is-the-idiomatic-way-to-deal-with-nullable-values-referencing-o/34498563#34498563> или по QR-коду справа:

**Пример сбора данных #5 — поиск людей совершеннолетнего возраста, вывод форматированной строки**

```
// Java:
String phrase = persons
    .stream()
    .filter(p -> p.age >= 18)
    .map(p -> p.name)
    .collect(Collectors.joining(" and ", "In Germany ", " are
of legal age."));

System.out.println(phrase);
// In Germany Max and Peter and Pamela are of legal age.

// Kotlin:
val phrase = persons
    .filter { it.age >= 18 }
    .map { it.name }
    .joinToString(" and ", "In Germany ", " are of legal age.")
```