Содержание

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.

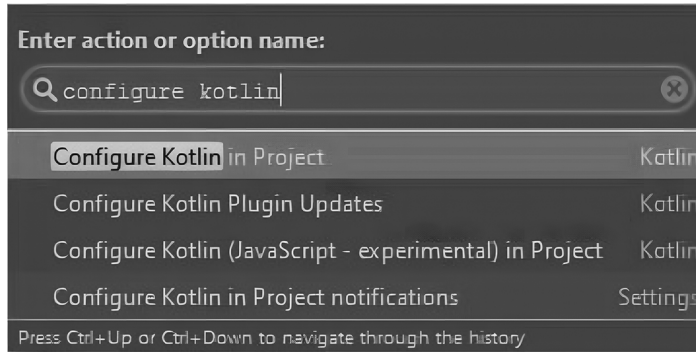
Установка плагина

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. Клонируйте проект `gradle-script-kotlin`.
2. Скопируйте / вставьте 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 инофрмация 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.")
```