Конспект по Kotlin

версия для PDF

Ruslan Zangizyarov

Оглавление

1. Добро пожаловать на сайт "Конспект по Kotlin"

Перейдите к категориям для доступа к главам.

2. Основы языка

2.1 Структура программы

Спецификация пакета находится вверху файла

```
package my.demo
```

Нет необходимости в том, чтобы директория проекта совпадала с директорией пакета. Исходные файлы могут быть размещены произвольным образом в системе.

Точка входа в программу

Выполнение любой программы начинается с функции `main`. Функция `main` - точка входа в программу.

```
fun main() {                    // Блок кода заключается в фигурные
скобки: {}.
    println("Hello, Kotlin!")  // Инструкция.
}
```

Определении функции начинается с `fun`.

Инструкции и блоки кода

Инструкция (англ. *statement*) - основной строительный блок программы. Блок кода заключается в фигурные скобки `{}` и содержит в себе инструкции, размещённые на отдельных строчках. Если требуется разместить несколько инструкций на одной строке, их нужно разделить знаком `;`.

Комментарии

При компиляции комментарии игнорируются

```
/*
    Многострочный
    комментарий.
*/

fun main() {                    // Однострочный комментарий.
    println("Hello, Kotlin!")
}
```

2.2 Переменные

Переменнная - контейнер, содержащий какое-либо значение примитивного типа данных, либо ссылку на объект класса.

При компиляции кода Kotlin в байт-код Java компилятор, если есть такая возможность, будет использовать примитивы Java, так как они обеспечивают лучшую производительность.

В Java переменные могут хранить только значения байтов (источник):

- примитивные типы переменных хранят непосредственно значение байтов данных;
- ссылочные типы переменных хранят байты адреса объекта в *heap*.

Вероятно, в Kotlin Native предусмотрены только ссылочные переменные.

Примитивные типы данных в Java: `byte`, `short`, `int`, `long`, `double`, `float`, `boolean`, `char`;
Ссылочные типы данных: те, что создаются с помощью конструкторов классов.

Примитивные переменные	**Ссылочные переменные**
Хранят **значение**.	Хранят **адрес объекта** в памяти
Создаются присваиванием значения	Создаются через конструктор класса (присваивание только создаёт **вторую ссылку** на существующий объект)
Имеют строго заданный диапазон допустимых значений	---
Значение по умолчанию зависит от типа (`int` - `0`, `float` - `0.0f`)	Значение по умолчанию - `null` (нет ссылки)
В аргументы методов попадают копии значения переменной (передача по значению)	В аргументы методов попадают значения ссылки. Операции выполняются над оригинальным объектом.
---	Могут исопользоваться для ссылки на объект объявленного или совместимого типа данных

Изменяемые и неизменяемые переменные

Kotlin - статически типизированный язык - типы переменных проверяются компилятором во время компиляции, а не во время выполнения.

Перед использованием переменной необходимо обязательно инициализировать её начальным значением, иначе будет выведена ошибка. Синтаксис объявления и инициализации переменной:

```
<тип_переменной_(val_или_var)> <имя_переменной>: <тип данных> =
<значение_или_объект>
```

```
val a: Int      // Объявление переменной.
a = 5           // Инициализация переменной.
val b: Int = 5  // Можно инициализировать переменную сразу после
объявления.
```

Также Kotlin поддерживает *type inference* - автоматическое выведение типов. Kotlin может сам определить тип для переменной на основе типа передаваемого объекта.

```
val a = 5       // 1. Компилятор видит, что значение `5` является
целым число.
                //    Создаётся объект `5` типа `Int`.
                // 2. На основе типа объекта будет выведен тип
переменной `a` - `Int`.
                //    Этот тип навсегда закрепляется за переменной.
                // 3. В переменной `a` сохраняется ссылка на объект
`5`.
```

Если переменная объявляется с ключевым словом `val`, то ссылка на объект сохраняется раз и навсегда и не может быть заменена (*immutable variable*).
Если переменная объявляется с ключевым словом `var`, то ссылка на объект может быть заменена ссылкой на другой объект того же или совместимого типа (*mutable variable*).

```
var a = 5       // Если не планируется менять значение переменной,
строго рекомендуется определять её с val.
a = 6
a = 7
```

Определение констант времени компиляции

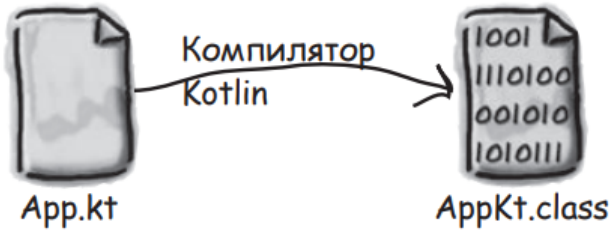
Свойства констант времени компиляции:

- значение констант времени компиляции устанавливается во время компиляции (значение `val` переменных устанавливается во время выполнения);
- тип данных константы должен соответствовать примитивному + либо `String`;
- константа должна объявляться на самом верхнем уровне (вне класса и функций);
- нельзя изменить значение константы.

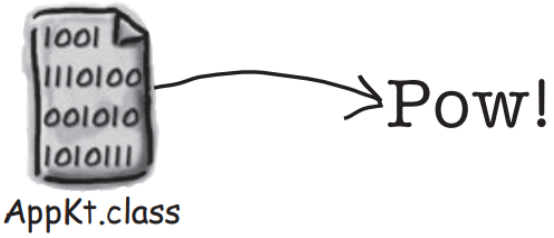
```
const val maxAge = 120  // Константа времени компиляции

fun main() {
    println(maxAge)
}
```

Что делает команда Run?

При выполнении команды Run интегрированная среда разработки (IDE - Integrated Development Environment) выполняет пару операций перед тем, как показать вывод вашего кода:
1. IDE компилирует исходный код Kotlin в байт-код JVM (Java Virtual Machine). Если код не содержит ошибок компиляции, создаётся один или несколько файлов классов, которые могут выполняться на JVM. Например, из файла `App.kt` будет создан файл класса с именем `AppKt.class`.



2. IDE запускает JVM и выполняет файл класса `AppKt.class`. JVM преобразует байт-код в формат, понятный для текущей платформы.



Компиляторы и интерпретаторы

Инструкции интерпретируемого языка выполняются непосредственно программой, называемой интерпретатором. Инструкции компилируемого языка преобразуются в представление, которое выполняется как собственная программа либо непосредственно на аппаратном процессоре, либо на виртуальной машине, эмулирующей процессор.

Языки *C*, *C++*, *Go*, *Rust* компилируются в машинный код, который работает непосредственно на аппаратном центральном процессоре. Такие языка, как Kotlin и Java компилируются в байт-код (формат промежуточного уровня), который выполняется на виртуальной машине JVM. Преимущество JVM - переносимость. Один и тот же байт-код может работать на любом устройстве с виртуальной машиной JVM. Виртуальные машины оптимизируются для конкретного оборудования. JVM содержит много лет таких оптимизаций на множестве платформ.

Исходный код Kotlin может быть скомпилирован для работы на различных платформах:
- JVM. Исходный код компилируется в байт-код JVM и может быть запущен на любой виртуальной машине JVM.
- Android. Исходный код проходит ряд компиляций: Kotlin -> JVM -> Dalvik. Android имеет свою среду выполнения ART, исполняющую файлы `*.dex` - JavaScript. Для запуска в веб-браузере. - Машинный код. Собственные двоичные файлы, скомпилированные для конкретных платформ и процессоров.

2.3 Типы данных

В Kotlin все компоненты программы, включая переменные, представляют собой объекты, которые имеют определённый тип данных. Тип данных определяет, какой размер памяти может занимать объект данного типа и какие операции можно с ним производить.

Целочисленные типы

Каждый целочисленный тип занимает в памяти фиксированное количество битов. Бит - единица измерения количества информации, может принимать значения `0` и `1`. Восемь бит образуют один байт.

- `Byte`: занимает 1 байт (2^8 комбинаций); хранит целое число от –128 до 127
- `Short`: занимает 2 байта (2^{16} комбинаций); хранит целое число от –32 768 до 32 767
- `Int`: занимает 4 байта (2^{32} комбинаций); хранит целое число от –2 147 483 648 ($–2^{31}$) до 2 147 483 647 ($2^{31} – 1$)
- `Long`: занимает 8 байт (2^{64} комбинаций); хранит целое число от –9 223 372 036 854 775 808 ($–2^{63}$) до 9 223 372 036 854 775 807 ($2^{63}–1$)

Также Kotlin поддерживает целочисленные типы без знаков
- `UByte`: занимает 1 байт; хранит целое число от 0 до 255;
- `UShort`: занимает 2 байта; хранит целое число от 0 до 65 535;
- `UInt`: занимает 4 байта; хранит целое число от 0 до 2^{32} - 1;
- `ULong`: занимает 8 байт; хранит целое число от 0 до 2^{64}-1.

Объекты целочисленных типов хранят целые числа:

```
fun main() {
    val a: Byte = -10
    val b: Short = 45
    val c: Int = -250
    val d: Long = 30000
    val e = 45000           // По умолчанию будет создан объект и
переменная типа Int.
    val f =
2_147_483_648  // Если присваиваемое число слишком велико, чтобы
поместиться в Int,
                           // то будет создан объект и переменная типа
Long.
    val g = 6L              // Явный литерал переменной типа Long.
}
```

Для передачи значений объектам, которые представляют беззнаковые целочисленные типы данных, после числа указывается суффикс `U`:

```
fun main() {
    val a: UByte = 10U
    val b: UShort = 45U
    val c: UInt = 250U
    val d: ULong = 30000U
}
```

Кроме использования *цифр* в десятичной системе мы можем определять целые *числа* в двоичной и шестнадцатеричной системах. Шестнадцатеричная запись числа начинается с `0x`, затем идет набор символов от `0` до `F`, которые представляют число.

Двоичная запись числа предваряется символами `0b`, после которых идет последовательность из нулей и единиц.

```
fun main() {
    val a: Int = 0x0A1   // Шестнадцетиричная запись числа 161.
    println(address)     // 161

    val b: Int = 0b0101  // Двоичная запись числа 5.
    println(address)     // 161
}
```

Числа с плавающей точкой

Кроме целочисленных типов в Kotlin есть два типа для чисел с плавающей точкой, которые позволяют хранить дробные числа: - Float: занимает 4 байта; хранит число с плавающей точкой от $-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$; - Double: занимает 8 байт; хранит число с плавающей точкой от $-1.8 \cdot 10^{308}$ до $1.8 \cdot 10^{308}$.

```
fun main() {
    val height: Double = 1.78
    val c: Double = 3e8       // Типы Double и Float поддерживают
экспоненциальную запись.
    val l: Float = 1e-3f
    val PI: Float = 3.14F     // В качестве литерала объекта типа
Float используется f или F.
}
```

Логический тип Boolean

Тип Boolean может хранить одно из двух значений: `true` (истина) или `false` (ложь).

```
fun main() {
    val a: Boolean = true
    val b: Boolean = false
}
```

Символы

Символьные данные представлены типом Char. Он представляет отдельный символ, который заключается в одинарные кавычки.

```
fun main() {
    val a: Char = 'A'
    val b: Char = 'B'
}
```

Также тип Char может представлять специальные символы (эскейп-последовательности), которые интерпретируются особым образом: - `\t`: табуляция; - `\n`: перевод строки; - `\r`: возврат каретки; - `\'`: одинарная кавычка; - `\"`: двойная кавычка; - `\\`: обратный слеш.

Строки

Строки представлены типом `String`. Строка представляет последовательность символов, заключенную в двойные кавычки, либо в тройные двойные кавычки.

```
fun main() {
    val name: String = "Ruslan"
    println(name)  // Ruslan
}
```

Строка может содержать эскейп-последовательности

```
val text: String = "Hello \nKotlin"
```

Литералы примитивных типов и строк для выведения типа (type inference)

Компилятор Kotlin позволяет выводить тип переменной на основании данных, которым переменная инициализируется. Поэтому при инициализации переменной тип можно опустить. Ниже приведены литералы примитивных типов и строк.

```
val age = 5         // Целое число по умолчанию указывает на тип Int.
val sum2 = 45U      // Целое число с U на конце по умолчанию указывает
на тип UInt.
                    // * Суффикс U также применяется при явном указании
типов:
                    //   UByte, UShort, UInt, ULong.
val sum1 = 45L      // Целое число с L на конце указывается на тип Long
val sum3 = 45UL
val height = 4.0    // Число с плавающей точкой по умолчанию указывает
на тип Double
val weight = 57.0f // Число с плавающей точкой и f или F на конце по
умолчанию
                    // указывает на тип Float.
val a = 'a'         // Содержимое одинарных кавычек указывает на Char.
                    // * Может быть только один символ.
val name = "Tom"    // Содержимое двойных, либо трёх двойных кавычек
указывает на String.
```