

Базовый Kotlin

Матвей Попов

План

- Как и где можно писать на Kotlin
- Переменные, арифметические операции
- Интерполяция, операторы сравнения, условные операторы
- Циклы, массивы, коллекции
- Функции
- Классы, ООП, интерфейсы
- ENUM, лямбды, Extensions

Где можно писать Kotlin код?

- IntelliJ IDEA
- VS code
- KPlay: <https://play.kotlinlang.org/>
- Terminal + Kotlinc + JDK

Переменные

```
fun main() {  
  
    val number = 12  
  
    val text = "Some random text"  
}
```

Ключевое
слово

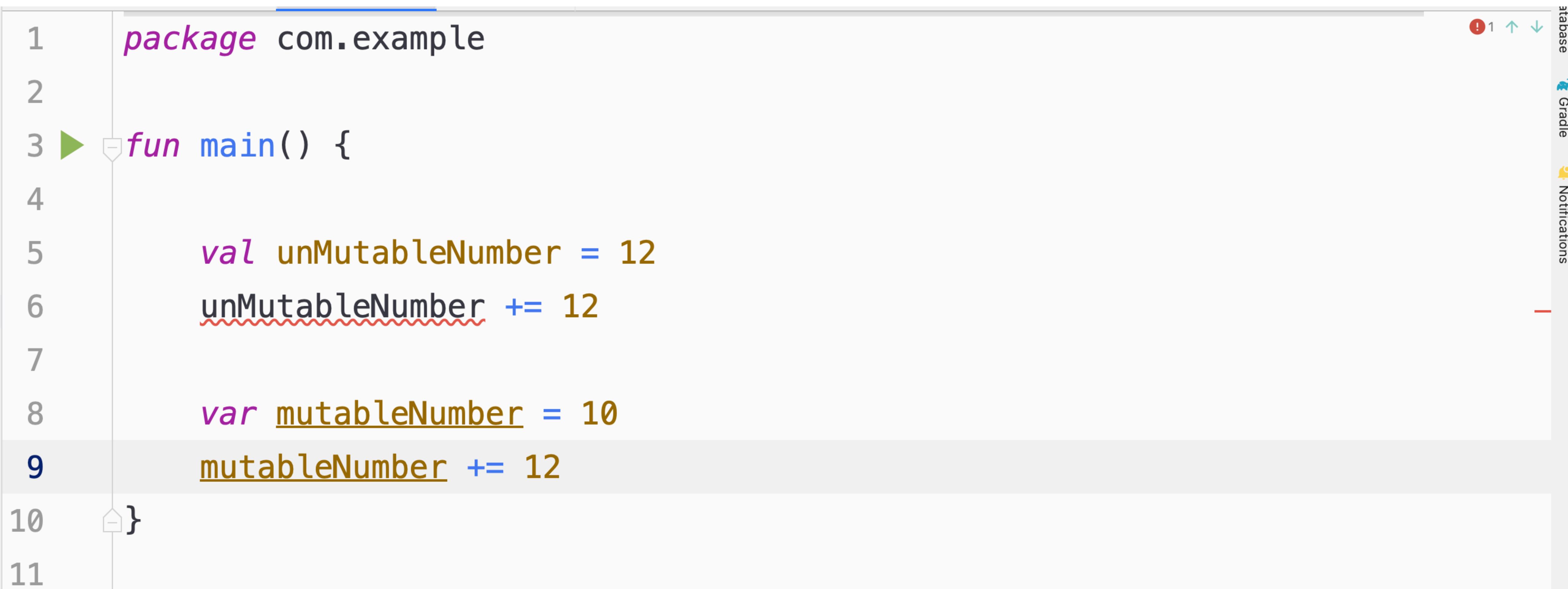
Название
переменной

Объект

Виды переменных

- Изменяемые
- Не изменяемые

```
1 package com.example  
2  
3 ► fun main() {  
4  
5     val unMutableNumber = 12  
6     unMutableNumber += 12  
7  
8     var mutableNumber = 10  
9     mutableNumber += 12  
10 }  
11
```



The screenshot shows a code editor with the following Java code:

```
package com.example

fun main() {
    val unMutableNumber = 12
    unMutableNumber += 12

    var mutableNumber = 10
    mutableNumber += 12
}
```

Annotations are present in the code:

- A red circle with the number "1" is located at the top right of the editor area.
- A green arrow pointing upwards is positioned above the opening brace of the main function at line 3.
- A red wavy underline is placed under the variable name "unMutableNumber" at line 6.

```
1 package com.example  
2  
3 ► fun main() {  
4  
5     val unMutableNumber = 12  
6     unMutableNumber += 12  
7     └─ Val cannot be reassigned  
8         Change to 'var' ↞ ↘ More actions... ↞ ↘  
9     var muta  
10    mutableN  
11 }
```

**А где указывается тип
данных?**

A screenshot of an IDE interface showing a Java code editor. The code is annotated with various icons and numbers:

- Line 1: **package** com.example
- Line 3: **fun** main() { (highlighted with a green arrow icon)
- Line 5: **val** myText = "my text"
- Line 7: **val** anotherMyText: String = "another my text" (with a red underline under String)
- Line 8: }

The right side of the screen features a vertical toolbar with the following items:

- ▲ 3 ↑ ↓ (with a yellow triangle icon)
- base (with a blue base icon)
- Gradle (with a teal Gradle icon)
- Notifications (with a yellow bell icon)

```
1 package com.example
2
3 ► fun main() {
4
5     val myText = "my text"
6
7     val ano
8 }
```

Variable 'myText' is never used

Remove variable 'myText' More actions...

val myText: String

kotlin.main

another my text"

Типы данных

- Целочисленные
- Вещественные
- Строковые
- Логические
- Объекты

```
1 package com.example
2
3 import com.example.example.MyClass
4
5 ► □ fun main() {
6
7     val myShort: Short = 1
8     val myInt: Int = 12
9     val myLong: Long = 89643
10    val myUnsignedByte: UByte = 1u
11
12    val myDouble: Double = 213.34
13    val myFloat: Float = -12.3f
14
15    val myString: String = "my string"
16    val myChar: Char = 'A'
17
18    val myBool: Boolean = false
19
20    val myClass: MyClass = MyClass()
21
22
```



Арифметические операции

```
1 package com.example
2
3 ► └ fun main() {
4
5     val firstNumber = 5
6     val secondNumber = 7
7
8     println(firstNumber + secondNumber) // 12
9 }
10 |
```

```
1 package com.example
2
3 ► └ fun main() {
4
5     val first = 10
6     val second = 3
7     println(first / second) // 3
8
9     val firstDouble: Double = 10.0
10    val secondDouble: Double = 3.0
11    println(firstDouble / secondDouble) // 3.333333333333335
12 }
13 |
```

▲² ↑ ↓

```
1 package com.example
2
3 ► □ fun main() {
4
5     val intNumber: Int = 5
6     val floatNumber: Float = 13.2f
7
8     println(intNumber + floatNumber) // 18.2
9     println(floatNumber / intNumber) // 2.6399999
10 }
11
```

▲ 2 ↑ ↓

```
1 package com.example  
2  
3 ➤ fun main() {  
4  
5     val intNumber: Int = 5  
6     val floatNumber: Float = 13.2f  
7  
8     println(intNumber > floatNumber) // false  
9     println(floatNumber == intNumber)  
10 }  
11 |
```

Operator '==' cannot be applied to 'Float' and 'Int'
val floatNumber: Float
ktor-sample.main

! 1 ▲ 4 ↑ ↓

Интерполяция

Интерполяция строк

Процесс **вычисления** строкового литерала, содержащего один или несколько **заполнителей**, в результате чего **заполнители заменяются соответствующими значениями**

```
1 package com.example
2
3 ► □ fun main() {
4
5     // Конкатинация строк
6
7     val greeting = "Добрый день"
8     val delimiter = ", "
9
10    println(greeting + delimiter + "уважаемые студенты.")
11
12 }
```

```
1 package com.example
2
3 ► fun main() {
4
5     // Интерполяция строк
6
7     val greeting = "Добрый день"
8     val delimiter = ","
9
10    println("$greeting$delimiter Звездный Лорд")
11 }
12
```

```
1 package com.example
2
3 ► fun main() {
4
5     // Интерполяция строк
6
7     val leftValue = 12
8     val rightValue = -6
9
10    println("Left value is bigger than right value: ${leftValue > rightValue}")
11 }
12
```

```
1 package com.example
2
3 ► □ fun main() {
4
5     // Многострочный текст
6
7     □ val multiLineText = """
8         first_line
9         second_line
10        third_line
11        fourth_line
12    """.trimIndent()
13
14    println(multiLineText)
15
16 }
```

Операторы сравнения

```
1 package com.example
2
3 ► fun main() {
4     // Логические операторы
5     // Операторы сравнения
6
7     val variable = 1 + 4
8     val boolVariable = (5 == (6 - 1))
9
10    // <, >, <=, >=, ==, != (операторы сравнения)
11    // ===, !== (операторы ссылочного сравнения)
12
13    // &&, ||, ! (логические операторы)
14    val result = (variable == 5) && boolVariable
15    val anotherResult = (variable != 2) and !boolVariable
16 }
17
```

▲ 7 ▲ 1 ↑ ↓

```
1 package com.example
2
3 import kotlin.random.Random
4
5 ► fun main() {
6
7     val leftSide = 15
8     val rightSize = 30
9
10    val inputVariable = Random(seed: 12).nextInt()
11
12    println(inputVariable in leftSide..rightSize)
13}
14
```

Условные операторы

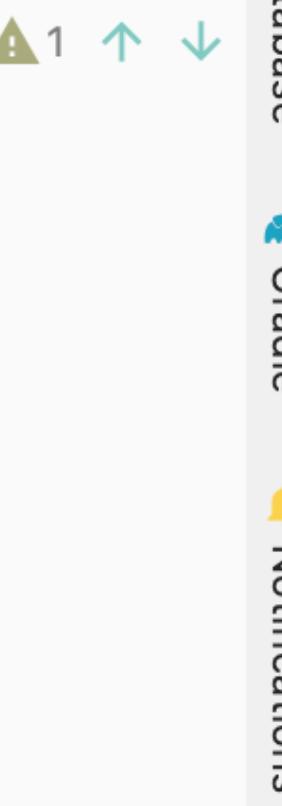
Или операторы ветвления

```
1 package com.example ✓
2
3 ► ┌ fun main() {
4
5     val includeParams = true
6
7     if (includeParams) {
8         println("Some text with params: $includeParams")
9     } else {
10        println("Something went wrong...")
11    }
12 }
13 |
```

**Конструкции if/else могут
возвращать значения**

```
1 package com.example
2
3 import kotlin.random.Random
4
5 ► └ fun main() {
6
7     val inputAge = Random(seed: 12).nextInt()
8
9     val resultText = if (inputAge >= 18) {
10         "You are adult!"
11     } else {
12         "You are child!"
13     }
14
15     println(resultText)
16 }
17
```

```
1 package com.example
2
3 import kotlin.random.Random
4
5 ► fun main() {
6
7     val randomInt = Random(seed: 12).nextInt()
8
9     when (randomInt) {
10         1 -> println("Your value is one")
11         2 -> println("Your value is two")
12         3 -> {
13             val randomLong = Random(seed: 11).nextLong()
14             println("Your long value is $randomLong")
15         }
16         else -> println("Your value is $randomInt")
17     }
18 }
19
```



Все о циклах

Цикл

Разновидность управляющей конструкции,
предназначенная для организации многократного
исполнения набора инструкций

```
1 package com.example
2
3 ► fun main() {
4
5     var counter = 5
6
7     if (counter > 0) {
8         println("Current step number: $counter")
9         counter--
10    }
11 }
12
```

```
1 package com.example
2
3 ► □ fun main() {
4
5     var counter = 5
6
7     □ do {
8         println("Current step number: $counter")
9         counter--
10    } while (counter > 0)
11 }
12 |
```

```
1 package com.example
2
3 ► └ fun main() {
4
5     val repeatCount = 10
6
7         repeat(repeatCount) { it: Int
8             println("Current step: $it")
9         }
10    }
11
```

Range и Progression

```
1 package com.example
2
3 ► └ fun main() {
4     // Диапазоны(интервалы)
5
6     val firstRange: IntRange = 1 .. ≤ 10
7     val secondRange: IntRange = 15 ≤ until < 40
8     val charRange: CharRange = 'a' .. ≤ 'z'
9
10    val doubleStepProgression: LongProgression = -5L .. ≤ 10000L step 2
11
12    val downProgression: IntProgression = -1 ≥ downTo ≥ -100
13}
14
```

▲ 5 ↑ ↓

А причем тут циклы?

```
1 package com.example ✓
2
3 ► └ fun main() {
4     // Цикл for
5
6     val reverseReport = 10 ≥ downTo ≥ 0
7
8     for (step in reverseReport) {
9         println("There are only $step")
10    }
11 }
12
```

```
1 package com.example
2
3 import kotlin.random.Random
4 import kotlin.random.nextInt
5
6 ➤ fun main() {
7     // Циклы for с операторами
8     val breakFlag = Random(seed: 10)
9         .nextInt(range: 10 ≤ ... ≤ 20)
10
11    for (step in 10 ≤ ... ≤ 20) {
12        if (step % 2 == 0) {
13            continue
14        }
15        if (step == breakFlag) {
16            break
17        }
18        if (step % 3 == 0) {
19            println("Received value: $step")
20        }
21
22        println("Current step: $step")
23    }
24 }
```

Массивы

```
1 package com.example
2
3 ► fun main() {
4
5     val stringArray: Array<String> = arrayOf("text_1", "text_2", "text_3")
6     val intArray: IntArray = intArrayOf(1, 2, 3, 4)
7     val charArray: CharArray = charArrayOf('a', 'b', 'c', 'd')
8 }
9
```

! 3 ↑ ↓

:base

Gradle

Notifications

**Что мы можем делать с
массивами?**

- Получение элемента по индексу
- Получение индекса элемента по его значению
- Получение размера массива
- Изменение значения по индексу

```
1 package com.example ✓  
2  
3 ► └ fun main() {  
4  
5     val intArray: IntArray = intArrayOf(1, 2, 3, 4, 5, 6, 7)  
6  
7     println(intArray.size)  
8     println(intArray[1])  
9  
10    intArray[2] = 100  
11  
12    println(intArray.indexOf(7))  
13 }  
14
```

```
1 package com.example
2
3 ► fun main() {
4
5     val intArray: IntArray = intArrayOf(1, 2, 3, 4, 5, 6, 7)
6
7     ▶ for (value in intArray) {
8         println("Received value: $value")
9     }
10
11    ▶ for ((index, value) in intArray.withIndex()) {
12        println("Received value: $value at position $index")
13    }
14
15 }
```

**Какие есть ограничения у
массивов?**

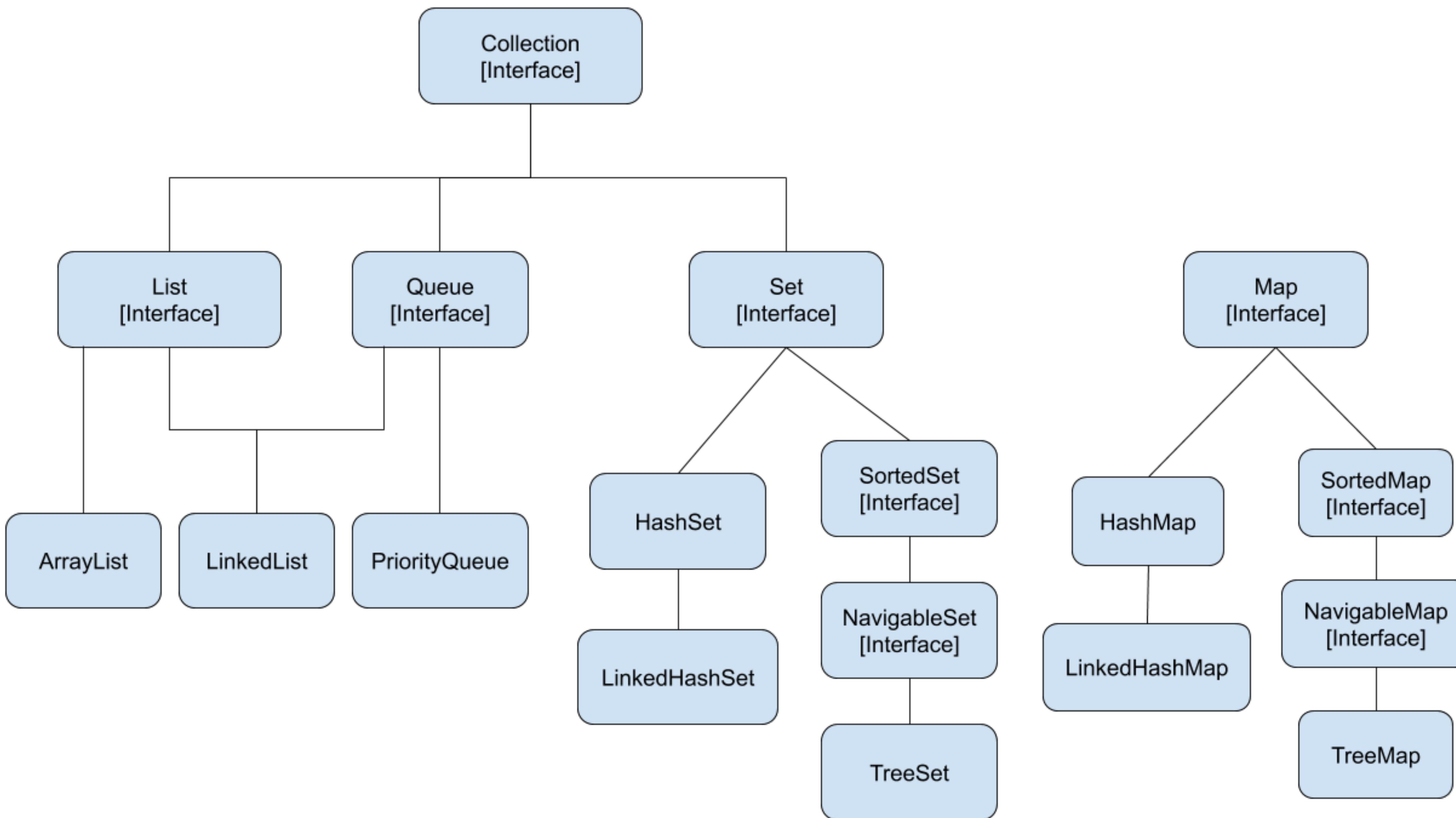
- Достаточно бедный API
- Нельзя добавлять/удалять элементы

Коллекции

Коллекция в Kotlin

Группы с **переменным** количеством элементов.

Как это выглядит?



```
1 package com.example
2
3 ► ┏ fun main() {
4     // List
5     val myList: List<Int> = listOf(1, 2, 3, 4)
6     // Set
7     val mySet: Set<Int> = setOf(1, 2, 3, 4)
8     // Map
9     val myMap: Map<String, Int> = mapOf(
10        "first" to 1,
11        "second" to 2,
12        "third" to 3
13    )
14 }
15 |
```

⚠³ ↑ ↓

```
1 package com.example  
2  
3 ► └ fun main() {  
4  
5     val commonList: Collection<Int> = listOf(1, 2, 3, 4)  
6     commonList.add(0)  
7         Unresolved reference: add :  
8             Rename reference ↕ More actions... ↕  
9     val mutableListOf: MutableCollection<Int> = mutableListOf(1, 2, 3, 4, 5)  
10    mutableListOf.add(0)  
11 }  
12
```

! 1 ↑ ↓

Arrays vs Collections

- У массива размер фиксирован - у коллекции нет
- Массив это класс Array - все коллекции это интерфейс
- Массивы оптимизированы для примитивов
- Отличается процесс сравнения элементов

ФУНКЦИИ

```
6 |  
7 | Ключевое слово  
8 |  
9 | fun functionName(  
10|   param1: Int,  
11|   param2: Int = 10  
12| ): Unit {  
13|   println(param1 + param2)  
14| }
```

Ключевое слово

Название функции

Аргументы

Возвращаемое значение

Тело функции

```
1 package com.example
2
3 ► └ fun main() {
4     println(calculate( leftValue: 1, rightValue: 1) + calculateShort( left: 10, right: 20))
5 }
6
7 fun calculate(
8     leftValue: Int,
9     rightValue: Int
10 ): Int {
11     return leftValue + rightValue
12 }
13
14 fun calculateShort(left: Int, right: Int) = left + right
15
```

Классы

```
1 package com.example.example
2
3 class MyClass
4
5
```

**Начнем с модификаторов
доступа**

Модификаторы доступа в Kotlin

- `public` - по умолчанию
- `private`
- `protected`
- `internal`

**Для создания класса нам
также необходим конструктор**

Виды конструкторов классов

- Primary
- Secondary

```
class MyClass constructor(  
    val field: Int  
)
```

```
class MyClass {  
  
    private val field: Int  
  
    constructor(  
        field: Int  
    ) {  
        this.field = field  
    }  
}
```

Виды конструкторов классов

- Primary
- Secondary

```
class MyClass constructor(  
    val field: Int  
)
```

```
class MyClass {  
  
    private val field: Int  
  
    constructor(  
        field: Int  
    ) {  
        this.field = field  
    }  
}
```

```
1 package com.example.example
2
3 class Dish(
4     val identifier: Long,
5     val name: String,
6     val ingredients: List<String>,
7     private var inFavorites: Boolean
8 ) {
9
10    fun show(): String {
11        return "[identifier]$name"
12    }
13
14    fun addToFavorite() {
15        inFavorites = true
16        logAction( payload: "Dish(identifier) added to favorites" )
17    }
18
19    fun removeFromFavorite() {
20        inFavorites = false
21        logAction( payload: "Dish(identifier) removed from favorites" )
22    }
23
24    private fun logAction(payload: String) {
25        println(payload)
26    }
27 }
```

Null safety

Что чаще всего **видит** Java
разработчик?



Как Kotlin решает* эту
проблему?

Nullable-типы и безопасный вызов

```
1 package com.example ▲1 ▲1
2
3 ► ┏ fun main() {
4
5     // NullPointerException(NPE)
6     // null
7
8     val nullableString: String? = null
9     val nonNullString: String = "some text"
10
11    println(nullableString?.length)
12    println(nonNullString.length)
13 ┎ }
```

```
1 package com.example
2
3 ► └ fun main() {
4
5     // NullPointerException(NPE)
6     // null
7
8     val nullableString: String? = null
9     val nonNullString: String = "some text"
10
11    if (nullableString != null) {
12        println(nullableString.length)
13    }
14    println(nonNullString.length)
15
16 }
```

⚠² ↑ ↓

```
1 package com.example
2
3 ► └ fun main() {
4
5     // NullPointerException(NPE)
6     // null
7
8     val nullableLine: String? = readLine()
9
10    val stringLength = nullableLine?.length ?: 0
11
12    println(stringLength)
13}
14
```

```
1 package com.example
2
3 ► └ fun main() {
4
5     val nullableLine: String? = readLine()
6
7     val nonNullLine: String = nullableLine!!
8
9     println(nonNullLine.length)
10
11 }
```

ENUM

```
1 package com.example.example
2
3 enum class MyEnum {
4
5     FIRST,
6     SECOND,
7     THIRD;
8 }
9
```

▲ 4 ↑

```
1 package com.example.example
2
3 enum class Status(
4     val id: Long,
5     val alias: String
6 ) {
7
8     NEW(id = 0, alias = "new"),
9     COOKING(id = 1, alias = "cooking"),
10    COMPLETED(id = 2, alias = "completed"),
11    ERROR(id = 3, alias = "error")
12 }
13
```

▲ 6 ↑ ↓

Почему ENUM в Kotlin это
курто

- Очень богатый программный API
- Компилятор проверяет что проверены все случаи

Лямбды и анонимные функции

```
1 package com.example
2
3 import java.util.Calendar
4
5 ► ┏ fun main() {
6
7     // Анонимные функции
8     // Лямды
9
10    val calendar = Calendar.getInstance()
11
12    val getDaysTillYearEnd: () -> Int
13        = fun () = 365 - calendar.get(Calendar.DAY_OF_YEAR)
14
15    println(getDaysTillYearEnd.invoke())
16    println(getDaysTillYearEnd())
17
18 }
```

```
2  
3 package com.example  
4  
5 ► └ fun main() {  
6  
7     // Анонимные функции  
8     // Лямды  
9  
10    val sayHello: (String) -> Unit  
11    sayHello = { input ->  
12        println("Hello, $input!")  
13    }  
14  
15    sayHello("World")  
16 }  
17
```

Extensions

Что это такое?

Extension функция

Это **функция**, не являющаяся участником какого-то **класса**, расширяет его **функционал**, имея доступ к публичным полям.

```
2  
3 package com.example  
4  
5 import java.io.File  
6  
7 ► ┏ fun main() {  
8  
9     // Extensions  
10  
11    val myFile = File(pathname: "myFile.txt")  
12  
13    myFile.writeToFileAndSend("Hello world!")  
14 }  
15  
16 ┏ fun File.writeToFileAndSend(message: String) {  
17     createNewFile()  
18     writeText(message)  
19     println("Message: \"\$message\" was added to file")  
20 }  
21
```

```
2  
3 package com.example  
4  
5 import java.io.File  
6  
7 val File.customName: String  
8     get() = "custom_$name"  
9  
10 ► □ fun main() {  
11  
12     // Extensions  
13  
14     val myFile = File(pathname: "myFile.txt")  
15     println(myFile.customName)  
16 }  
17
```

Kotlin Generics

2
3
4
5

```
class Box <T> (val instance: T)
```

```
3 | class Box <T> (val instance: T)
```

```
4 |
```

```
5 | class CollectionBox <T: Collection<*>> (val instance: T)
```

```
6 |
```

A little more about classes

```
3  sealed class Example( name: String, description: String, payload: T, public_name: String, public_description: String)
4     internal val name: String,
5     private val description: String,
6 )
7
8     class EmptyExample : Example( name: "", description: "" )
9
10    data class ExampleWithPayload <T: Any> (
11        val payload: T,
12        val public_name: String,
13        val public_description: String,
14    ) : Example(public_name, public_description)
15
16    object StaticExample : Example( name: "static_name", description: "static_description" )
17
18 }
```

**А теперь про домашнее
задание**

- Пройти простой курс на Stepik
- Там простые задачки на Kotlin
- Нужно сделать пункты
1-5(включительно)
- Время выполнения 20-30 минут
- Ссылка будет в материалах

Материалы

- Курс по Kotlin для начинающих
<https://clck.ru/35ZaAx>
- Официальные Kotlin koans
[https://play.kotlinlang.org/koans/
overview](https://play.kotlinlang.org/koans/overview)

Вопросы?

Спасибо за уделенное время!