СПб АУ НОЦНТ РАН

Kotlin 01

23.10.2017

Реквизиты

- denis.zharkov@jetbrains.com
- github.com/dzharkov

Kotlin

- <Здесь мог бы быть логотип >
- Платформы: JVM, JS, Native
- Год рождения: 2016 (2010)
- Появился как проект в JetBrains

Basic syntax

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

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

val a: Int = 1
val b = 1   // 'Int' type is inferred
```

Сравнивая со Scala

Точки зрения:

- + более простые концепции
- + Java interoperability
- + анализ в IDE работает также как в компиляторе
- - недостаточно гибкий (implicits/macro)
- - меньше возможностей для ФП

Начало

- https://github.com/JetBrains/kotlin-examples
  - Gradle/Maven projects
  - Webapp projects
- IntelliJ IDEA
- http://try.kotlinlang.org/
- Плагин для Eclipse
- Любой другой текстовый редактор

Операторы. If

```
if (a > 1) {
    print("more")
} else {
    print("less or equal")
}

val b = if (a > 1) "more" else "less or equal" // instead of ? :
```

Операторы. For

```
for (i in 1..10) {
    println(i)
}

// should have 'iterator()' method,
// but can be non-Iterable subclass
val x: IntRange = 1..10
```

Переменные/property

```
val name: Type
    get() { return value } // optional


var name: Type
    get() { return value } // optional
    set(value) { } // optional

instance.name = instance.name + "suffix"
// instance.setName(instance.getName() + "suffix")
```

- В большинстве случаев val достаточно

val with getter vs. function

- Отсутствие внешних эффектов при вычислении
- Вычисление за O(1)

Классы

```
interface SuperInterface {
    val value: Int
    fun computeName(): String
    fun myMethod() // Unit
}
class MyClass : SuperClass, SuperInterface {
    override val value: Int
    constructor(value: Int) : super(1) {
        this.value = value
    }

    override fun computeName() = "myName"

    override fun myMethod() {}
}
```

Классы. Первичный конструктор

```
class MyClass(
    override val value: Int
) : SuperClass(1), SuperInterface {
    override fun computeName() = "myName"
    override fun myMethod() {}
}
```

Модификаторы

- Visibility: public (default), private, protected, internal (внутри модуля)
- Modality: abstract, open, final (default)

data classes

```
data class Client(val name: String, val email: String)
```

```
public class Client {
    private final String name;
    private final String email;

    public Client(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getName() { return name; }

    public String getEmail() { return email; }

    // equals/hashCode/toString
}
```

Объекты

```
object MyObject : SuperClass(1), SuperInterface {
    override val value: Int
        get() = 123
    override fun computeName() = "myObject"
    override fun myMethod() {}
}

class MyClass {
    companion object {
        val CONSTANT = 1
        fun likeStatic() {}
    }
}

fun usage() {
    MyObject.value + MyClass.CONSTANT
}
```

Объекты vs. функции в файле

- В Kotlin можно объявлять функции и свойства в файле вне класса

Объекты vs. функции в файле

- В Kotlin можно объявлять функции и свойства в файле вне класса
- И чаще всего можно обойтись без объектов

Объекты vs. функции в файле

```
object SomeValuesFactory : Factory {
    override fun createNewValue(): Value {...}
}

class MyClass private constructor() {
    companion object : Factory {
        fun createNewValue() = MyClass()
    }
}

fun useFactory(factory: Factory) {}

useFactory(SomeValuesFactory)
useFactory(MyClass)
useFactory(MyClass.Companion)
```

Типы

- Int, Long, Char, Byte, Short, Double, Float

Типы

- Int, Long, Char, Byte, Short, Double, Float
- IntArray/.../Array<T>
  - IntArray != Array<Int>?
- Nothing <: Any (Bottom <: Top)

Типы

- Int, Long, Char, Byte, Short, Double, Float
- IntArray/.../Array<T>
  - IntArray != Array<Int>?
- Nothing <: Any (Bottom <: Top)
- Unit

Типы

- Int, Long, Char, Byte, Short, Double, Float
- IntArray/.../Array<T>
  - IntArray != Array<Int>?
- Nothing <: Any (Bottom <: Top)
- Unit
- Function2<Int, String, Double> == (Int, String) -> Double
  - Function2::invoke(x: T1, y: T2): R

Nullable types

- для любого простого типа T определен T?
- T <: T?
- $T? = \{T\} \cup null$
- ‘null‘ – значение типа ‘Nothing?‘
- x: T? ⇒ x может содержать null или значение типа T
- x.someMethod() – запрещены вызовы методов класса
- Запрещено использовать T? как T

Nullability

```
fun parseInt(str: String): Int? {
  // ...
}

val x = parseInt(args[0])
val y = parseInt(args[1])

// Using ‘x * y‘ yields error because
// they may hold nulls.
if (x != null && y != null) {
  // x and y are automatically cast to non-nullable
  // after null check
  print(x * y)
}
```

Операторы для nullable-значений

```
fun updateHeight() {
    this.height =
      maxOf(
          this.left?.height ?: 0,
          this.right?.height ?: 0
      ) + 1
}
```

Операторы для nullable-значений

```
myMap.get("KEY")!!.hashCode()
```

Use-site variance (aka Java wildcards)

```
fun copy(
    // Array<? extends CharSequence>
    x: Array<out CharSequence>,
    // Array<? super CharSequence>
    y: Array<in CharSequence>
) {
    for (i in x.indices) {
        y[i] = x[i]
    }
}
```

Declaration-site variance

```
interface List<out T> {
    fun get(index: Int): T
    // Error: T can't be used here
    fun set(index: Int, t: T)
}

fun bar(x: List<CharSequence>) {}
fun foo(x: List<String>) {
    bar(x)
}
```

Read-only/Mutable collections

```
interface MutableList<T> : List<T> {
    // ...
    fun set(index: Int, t: T)
    // ...
}

interface MutableCollection<T> : Collection<T>
interface MutableIterator<out T> : Iterator<T>
interface MutableSet<T> : Set<T>
interface MutableMap<K, V> : Map<K, V>
```

FunctionK interfaces

```
interface Function1<in T, out R> {
    fun invoke(x: T): R
}

fun bar(x: (String) -> Any?) {}
fun foo(y: (CharSequence) -> Boolean) {
    bar(y)
}
```

Smart casts

```
fun getStringLength(obj: Any): Int? {
  if (obj is String) {
    // ‘obj‘ is automatically
    // cast to ‘String‘ in this branch
    return obj.length
  }

  // ‘obj‘ is still of type
  // ‘Any‘ outside of the type-checked branch
  return null
}
```

When

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}
```

When

```
when (x) {
    0, 1 -> print("x == 0 or x == 1")
    else -> print("otherwise")
}
```

When

```
when (x) {
    parseInt(s) -> print("s encodes x")
    else -> print("s does not encode x")
}
```

When

```
when (x) {
    in 1..10 -> print("x is in the range")
    in validNumbers -> print("x is valid")
    !in 10..20 -> print("x is outside the range")
    else -> print("none of the above")
}
```

When

```
fun hasPrefix(x: Any) = when(x) {
    is String -> x.startsWith("prefix")
    else -> false
}
```

When

```
when {
    x.isOdd() -> print("x is odd")
    x.isEven() -> print("x is even")
    else -> print("x is funny")
}
```

Sealed classes

```
sealed class Expr
data class Const(val number: Double) : Expr()
data class Sum(val e1: Expr, val e2: Expr) : Expr()
object NotANumber : Expr()

fun eval(expr: Expr): Double = when(expr) {
    is Const -> expr.number
    is Sum -> eval(expr.e1) + eval(expr.e2)
    NotANumber -> Double.NaN
    // the ‘else‘ clause is not required because we’ve covered all the cas
}
```

Main

```
fun main(args: Array<String>) {
    println("Hello, world!")
}
```

Lambdas

```
val positives = list.filter { x -> x > 0 }
val positives = list.filter { it > 0 }
```

String interpolation

```
println("Name \$name \${someFunc()}")
```

Домашнее задание

- Создаем форк репозитория https://github.com/java-course-au/kotlin-course
- git checkout 01-just-for-fun
- Пишем решение, коммитим
- Если непонятно, читаем Readme на странице репозитория
- Все еще непонятно? Создаем issue в репозитории, упоминаем “@dzharkov”
- Создаем pull request в origin:01-just-for-fun с темой Kotlin. ДЗ 0_, <фамилия> <имя>
- В комментарии к PR упоминаем “@dzharkov”
- Нарушение формата сдачи — это очень важно!

На оценку влияет

- Соблюдение формата сдачи
- Соответствие кода стандартным Java coding conventions (где это имеет смысл) и Kotlin style guides
- Выполнение формальных требований задания
- Отсутствие предупреждений компилятора и инспекций в IDE
- Общая аккуратность решения
- Повторение предыдущих ошибок
- Количество итераций сдачи