Kotlin: классы

Марат Ахин

13 февраля 2018 г.

Санкт-Петербургский политехнический университет

Recap

То, о чем забыл рассеянный преподаватель

Kotlin IDE

IntellJ IDEA (от того же JetBrains)

- Есть альтернативы
    - NetBeans/Eclipse
    - Sublime/Atom
    - Vim
- По опыту, IDEA на голову выше всех альтернатив

Kotlin IDE

Если хочется попробовать очень-очень быстро?

- `https://try.kotlinlang.org/`
- IDE в вашем браузере
    - Может далеко не все (YMMV)
    - Не требует установки

Kotlin needs no semicolons

- Если вы не заметили, в Котлине не надо ставить точки с запятой
- Компилятор расставляет их за вас (`semicolon inference`)
    - В очень редких случаях у него может не получиться
    - Тогда можно ему подсказать и таки поставить `;`

В моем опыте такой ситуации не возникало ни разу ☺


CLASSES
CLASSES EVERYWHERE!

Объектно-ориентированное программирование

- Наследование
- Полиморфизм
- Инкапсуляция

Объектно-ориентированное программирование

- Все — это объект
- Объекты содержат данные (поля и/или свойства)
- Объекты могут обмениваться сообщениями (вызовы методов)

Несколько более естественный взгляд на ООП
По моему скромному мнению

Classes in Kotlin

Class

```
class SqlQuery
```

Constructors

```
class SqlQuery @ShouldNotBeOptimized constructor(
    sqlDialect: String)
```

Constructors

```
class SqlQuery(sqlDialect: String)
```

Properties

```
class SqlQuery(sqlDialect: String) {
    val _sqlDialect = sqlDialect
}
```

Properties

```
class SqlQuery(val sqlDialect: String)
```

Properties

```
class SqlDialect(val name: String) {
    val isDefault: Boolean
        get() {
            return "" == name
        }
}
```

Properties

```
class SqlDialect(val name: String) {
    val isDefault: Boolean
        get() = "" == name
}
```

Properties

```
class SqlDialect(val name: String) {
    val isDefault
        get() = "" == name
}
```

Properties

```
class SqlConfig {
    var hasCaching = false
        get() {
            println("Getting hasCaching: $field")
            return field
        }
        set(value) // ...
}
```

Properties

```
class SqlConfig {
    var hasCaching = false
        get() // ...
        set(value) {
            if (value == field) return
            println("Setting hasCaching: $field")
            if (value) {
                // Do some caching stuff
            } else {
                // Do some non-caching stuff
            }
            field = value
        }
}
```

Initialization

```
class SqlQuery(val sqlDialect: String) {
    val config: SqlConfig

    init {
        config = SqlConfig()
    }
}
```

Initialization

```
class SqlQuery(val sqlDialect: String) {
    val config = SqlConfig()
    val dialect = SqlDialect(sqlDialect)
}
```

Initialization

```
class SqlQuery(sqlDialect: String) {
    val config = SqlConfig()
    val dialect = SqlDialect(sqlDialect)
}
```

Secondary constructor

```
class SqlQuery(sqlDialect: String) {
    val config = SqlConfig()
    val dialect = SqlDialect(sqlDialect)

    constructor() : this("") {
        config.hasCaching = false
    }
}
```

Inheritance or aggregation?

```
class SqlConfig(sqlDialect: String) {
    val dialect = SqlDialect(sqlDialect)
    var hasCaching = false
        get() // ...
        set(value) // ...
}

class SqlQuery(sqlDialect: String) {
    val config = SqlConfig(sqlDialect)

    constructor() : this("") {
        config.hasCaching = false
    }
}
```

Inheritance

```
open class SqlQuery(...) { ... }

class SelectQuery(sqlDialect: String) : SqlQuery(sqlDialect)
```

Inheritance

```
class SelectQuery(
    sqlDialect: String,
    val table: String,
    val fields: Array<String>) : SqlQuery(sqlDialect)
```

Interfaces

```
interface Executable {
    fun execute(): Int

    fun beforeExecute() {}
    fun afterExecute() {}
}
```

Interfaces

```
interface Loggable {
    val log: Logger
        get() = Logger.getLogger(javaClass.name)
}
```

Interfaces

```
interface LoggingExecutable : Executable, Loggable {
    override fun beforeExecute() {
        log.info("Before executing: $this")
    }
    override fun afterExecute() {
        log.info("After executing: $this")
    }
}
```

Interfaces

```
class SelectQuery(
    sqlDialect: String,
    val table: String,
    val fields: Array<String>
) : SqlQuery(sqlDialect), LoggingExecutable {
    override fun execute(): Int { ... }
}
```

Interfaces

```
class SelectQuery(
    sqlDialect: String,
    val table: String,
    val fields: Array<String>
) : SqlQuery(sqlDialect), LoggingExecutable {
    override fun execute(): Int = TODO()
}
```

[Companion] objects

```
class SelectQuery(
    sqlConfig: SqlConfig,
    val table: String,
    val fields: Array<String>
) : SqlQuery(sqlConfig), LoggingExecutable {
    override fun execute(): Int = TODO()
}
```

[Companion] objects

```
class SqlQuery(val config: SqlConfig) {
    constructor() : this(SqlConfig("")) {
        config.hasCaching = false
    }
}
```

[Companion] objects

```
object DefaultSqlConfig : SqlConfig("") {
    init {
        hasCaching = false
    }
}

class SqlQuery(val config: SqlConfig) {
    constructor() : this(DefaultSqlConfig)
}
```

[Companion] objects

```
class SelectQuery(
    sqlConfig: SqlConfig,
    val table: String,
    val fields: Array<String>
) : SqlQuery(sqlConfig), LoggingExecutable {
    // ...
    companion object Whatever {
        fun create(
            table: String,
            fields: Array<String>
        ) = SelectQuery(DefaultSqlConfig, table, fields)
    }
}
```

Another kind of objects

```
class SqlRunner {
    fun run(exec: Executable): Int {
        exec.beforeExecute()
        val res = exec.execute()
        exec.afterExecute()
        return res
    }

    fun shutdown(): Unit = TODO()
}
```

Another kind of objects

```
class SqlTests {
    var runner: SqlRunner?

    @Test
    fun testRun() {
        runner?.run(TODO())
    }
}
```

Another kind of objects

```
class SqlTests {
    lateinit var runner: SqlRunner

    @BeforeEach
    fun before() {
        runner = SqlRunner()
    }

    @AfterEach
    fun after() {
        runner.shutdown()
    }
}
```

Another kind of objects

```
class SqlTests {
    @Test
    fun run() {
        assertEquals(42, runner.run(
            object : LoggingExecutable {
                override fun execute(): Int {
                    log.info("Look, ma!")
                    return 42
                }
            }
        ))
    }
}
```

Nested classes

```
class SqlRunner {
    fun run(exec: Executable): Int {
        return WorkItem(exec).call()
    }

    class WorkItem(val exec: Executable) : Callable<Int> {
        override fun call(): Int {
            exec.beforeExecute()
            val res = exec.execute()
            exec.afterExecute()
            return res
        }
    }
}
```

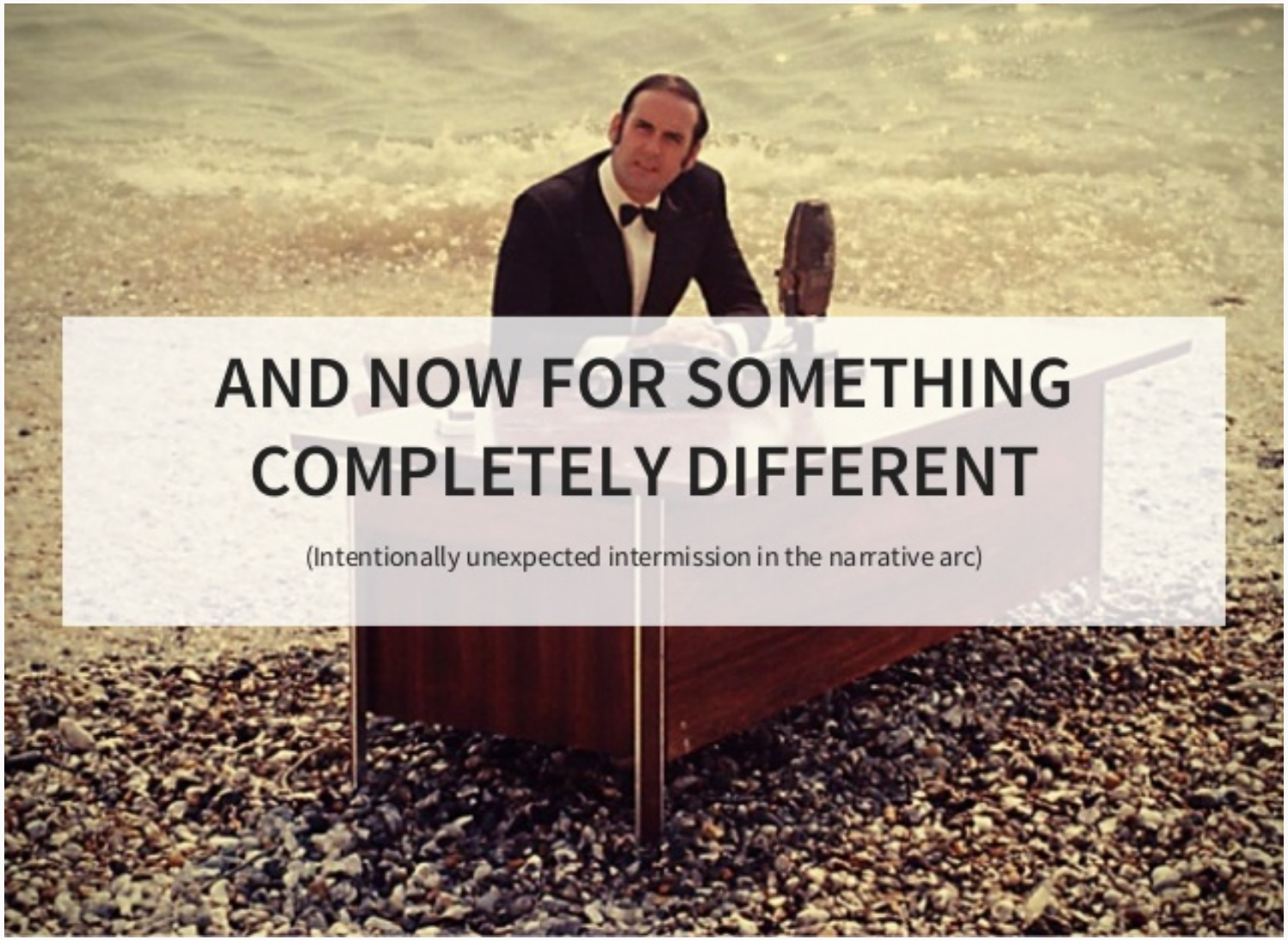
Nested classes

```
class SqlRunner {
    val pool: ExecutorService =
        Executors.newCachedThreadPool()

    fun run(exec: Executable): Int {
        return WorkItem(exec).call()
    }

    fun runAsync(exec: Executable): Future<Int> {
        return pool.submit(WorkItem(exec))
    }
}
```

Nested classes

```
class SqlRunner {
    open class WorkItem(...) { ... }

    inner class AsyncWorkItem(exec: Executable)
        : WorkItem(exec) {
        fun submit(): Future<Int> =
            this@SqlRunner.pool.submit(this)
    }
}
```

Intermission



Visibility

- `public` (по умолчанию)
- `protected`
- `internal` (`package-private++`)
- `private`

Правила использования — как и раньше — здравый смысл

Java interop quirks

- Типы со стороны Java рассматриваются как `platform types`
- `T` vs `T?` vs `T!`
- `Platform types` не проверяются на `null` при компиляции
    - `T!` может трактоваться и как `T`, и как `T?`
    - Очень простой способ выстрелить себе в ногу
- Можно уточнить работу с `platform types` при помощи JSR-305

Если есть сомнения, тип лучше уточнить явно

Type casts

- Смарт-кастов иногда недостаточно
- Например, когда непонятно, по какому критерию смарт-кастить

```
val userData: Map<String, Any> = getUserData(...);
val userName: String = userData["username"] as String
val userAddress: String? = userData["useraddress"] as? String
val userPhone: String? = userData["userphone"] as String?
```

Как приведения типов работают с дженериками, обсудим позже

Exceptions

```
class SelectQuery(...)
    : SqlQuery(sqlDialect), LoggingExecutable {
    override fun execute(): Int {
        return try {
            // ...
            recordCount
        } catch (ex: SqlException) {
            log.error("Oops, exception: ${ex.message}")
        } finally {
            doingSomeUsefulSideEffectingStuff()
        }
    }
}
```

Exceptions

```
class SelectQuery(...)
    : SqlQuery(sqlDialect), LoggingExecutable {
    override fun execute(): Int {
        // ...
        if (conn == null) {
            throw SqlException("SQL connection failed!")
        }
    }
}
```

Exceptions

```
class SelectQuery(...)
    : SqlQuery(sqlDialect), LoggingExecutable {
    override fun execute(): Int {
        // ...
        conn ?: throw SqlException("SQL connection failed!")
    }
}
```

UNchecked exceptions

- В Котлине все исключения являются `unchecked`
  - Потому что `checked` исключения лишь только раздувают код, но не делают его более безопасным
  - `IOException` anyone?

Examination of small programs leads to the conclusion that requiring exception specifications could both enhance developer productivity and enhance code quality, but experience with large software projects suggests a different result — decreased productivity and little or no increase in code quality. (c) Bruce Eckel

Auto getters/setters

Что за `ex.message`*?*

- Вместо свойств в Java используются геттеры/сеттеры
- Котлин автоматически оборачивает геттеры/сеттеры в соответствующее свойство
  - `getX()/setX(newX)` -> `var x`
  - `getX()` -> `val x`
  - `isX()/setX(newX) : Boolean` -> `var x`
  - `isX() : Boolean` -> `val x`

А если у меня и поле, и геттеры/сеттеры?

What I learned today?

What I learned today?

- В Котлине есть классы и интерфейсы
- Кроме полей, у классов и интерфейсов могут быть свойства
- Наследование и переопределение только через `open`
- Паттерн “синглетон” встроен в язык в виде объектов

What I learned today?

- Интероп с Java не всегда работает очевидным образом
- Нет checked исключений
- Когда не хватает смарт-кастов, можно приводить типы самому
- Котлин умеет оборачивать геттеры/сеттеры в свойства

Самые сообразительные уже увидели, откуда ноги растут...
Effective Java

Классы, которых в Java нет (из коробки)