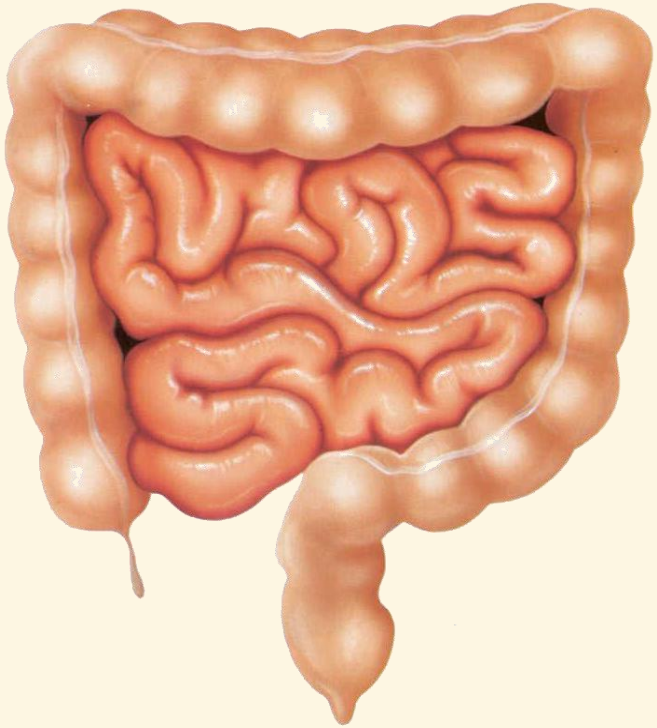
Kotlin:
два года в проде и ни единого разрыва

Паша Финкельштейн
Разработчик, тимлид

 asm0di0
 asm0dey
 asm0dey

О докладе

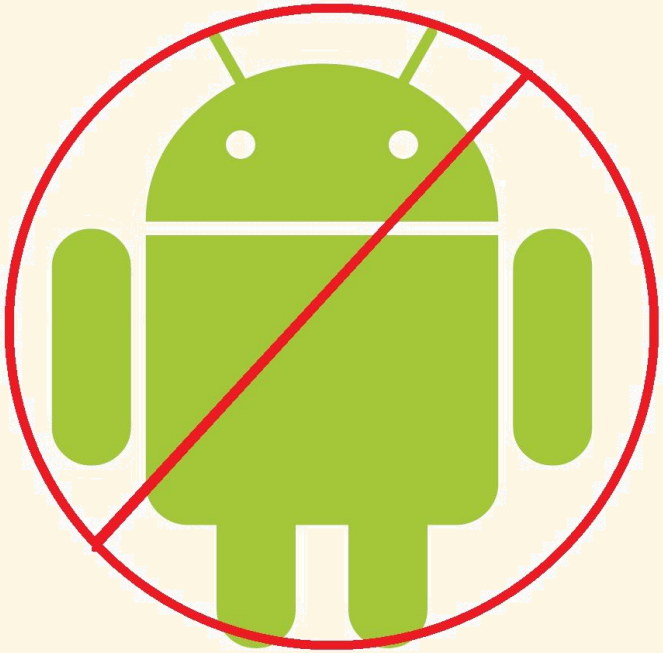


Кишки нормального доклада

О докладе

Кишки моего доклада

Не Android



Не для тех, кто уже

Отказ от гарантий

- Я никак не аффилирован с JetBrains (пока)
- Я не отвечаю за то что вам захочется попробовать
- Так же как и за то, что вам не захочется
- Я знаю не всё, а значит могу ошибаться или неправильно понимать то, что другие люди понимают правильно или просто там не ошибаются.
- Но я стараюсь

Несколько слов о себе?

- 12 лет в IT
- 10 лет в разработке
- Почти всё время — на JVM
- Учусь по референсам и на ошибках
- Экспериментирую
- Попробовал всякое от Java до Ceylon и Frege

```
module Hello where
greeting friend = "Hello, " ++ friend ++ "!"
main args = do
    println (greeting "World")
```

Intro: каким был мой первый раз

- Котлин был ещё чайником
- Дока была в конфлюэнсе
- Аннотаций были без собак
- Перегруженные конструкторы? Не, не слышал.
- И я даже что-то предлагал :)

А реальный опыт?

- Маленький стартап
- Маленькие сервисы
- Много сущностей, которые меняются
- Надоел бойлерплейт и костыли

```
@Getter
@Setter
@EqualsAndHashCode
@AllArgsConstructor
@RequiredArgsConstructor
public class Person{ //…
```

VS

Да, мы знали про @Data

Код здорового разработчика тут → `data class Person( //…`

А ещё?

- null-safety
- Автоматический вывод типов
- Очень хорошую совместимость с Java
- Идеальное делегирование
- Нормальный вызов функций без всякого `apply()`
- extension-методы

```
fun Iterable<Int>.sum() = reduce{a, b → a + b}
```

Это уже есть
в стандартной библиотеке

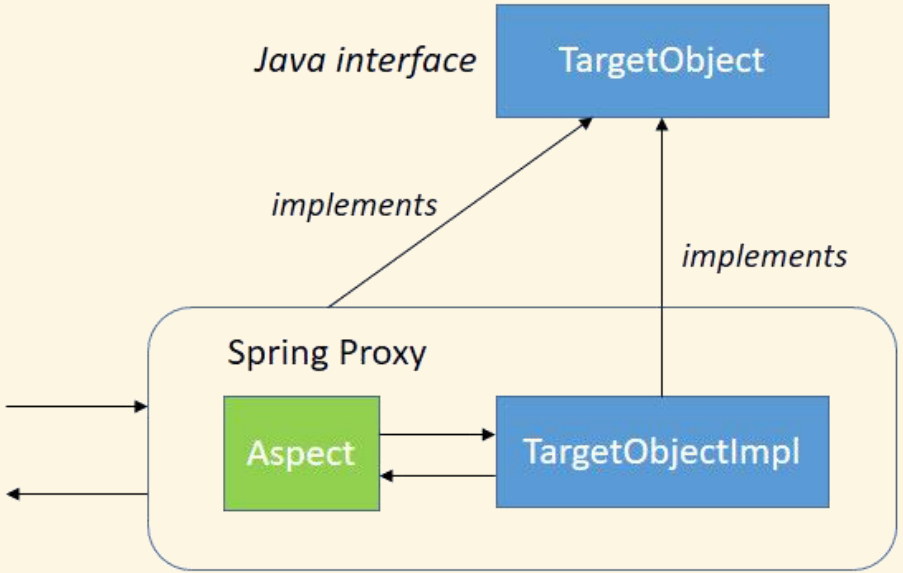
Spring + Kotlin. Part 1
@Transactional

```
@Transactional
class MySmartService(repo1: Repository1, repo2: Repository2)
{
  fun complex(datum: String): Result {
    val interim = repo1.save(datum)
    return repo2.destroy(someOp(interim))
  }
  private fun someOp(input: String) =
    if (notLucky(input)) throw Exception("Sorry, bro") else
input
```

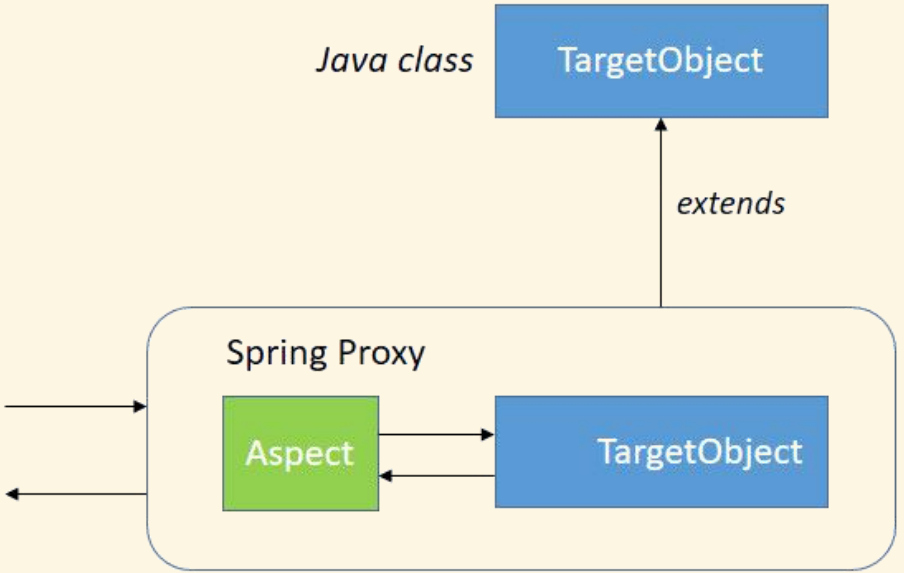
Что может пойти
не так?

Spring AOP Process

JDK Proxy *(interface based)*



CGLib Proxy *(class based)*



https://www.baeldung.com/spring-aop-vs-aspectj

Spring + Kotlin. Part 1
@Transactional

```
@Transactional
open class MySmartService(repo1: Repository1, repo2:
Repository2) {
  open fun complex(datum: String): Result {
    val interim = repo1.save(datum)
    return repo2.destroy(someOp(interim))
  }
  private fun someOp(input: String) =
    if (notLucky(input)) throw Exception("Sorry, bro")
else input
```

final by default

Spring + Kotlin. Part 1
final by default

Проблема

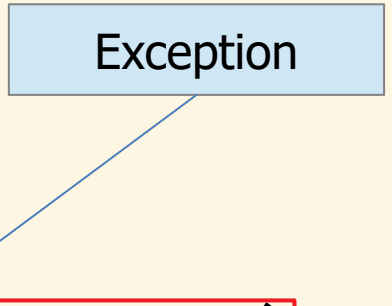
- Всё `final`
- Прокси не создаются
- Конфигурации не работают

Решения

- `kotlin-allopen + kotlin-spring`
  - Код, который ты видишь не соответствует реальности ⇒ кодревью усложняется
  - Конструкция хрупкая и иногда ломается
- Всюду явно писать `open`
  - Опять бойлерплейт...

Spring + Kotlin. Part 2
@Transactional

```
@Transactional
open class MySmartService(repo1: Repository1, repo2:
Repository2) {
  open fun complex(datum: String): Result {
    val interim = repo1.save(datum)
    return repo2.destroy(someOp(interim))
  }
  private fun someOp(input: String) =
    if (notLucky(input)) throw Exception("Sorry, bro") else
input
```



Исключения в Kotlin?

Понятный подход: `throw RuntimeException(Exception())`

НО НЕТ

```
public static RuntimeException sneakyThrow(Throwable t) {
  if (t == null) throw new NullPointerException("t");
  return Lombok.<RuntimeException>sneakyThrow0(t);
}

@SuppressWarnings("unchecked")
private static <T extends Throwable> T sneakyThrow0(Throwable t) throws T {
  throw (T)t;
}
```

Код из Lombok

Spring + Kotlin. Part 2
@Transactional

Проблемы

- final by default
- Нет checked exceptions

Решения

- см. раньше
- Внимательно следить за всеми исключениями
  - особенно в библиотеках
    - особенно там где есть IO

Spring + Kotlin. Part 3
#Security

```
public interface UserDetails extends Serializable {
    String getPassword();
    String getUsername();
    // …
```

- В интерфейсе объявлен геттер
- В котлине нет геттеров

Spring + Kotlin. Part 3
#Security

```
data class User(
    private val username: String,
    private val pass: String
): UserDetails {
    override fun getUsername() = username
    override fun getPassword() = pass
}
```

Не только спринг

Работа с БД. JOOQ

Всё работает!

- Получение работает даже красивее чем в Java
- Алиасы в стандартном синтаксисе не работают
- Даже маппинг в data-классы
- Лямбды красивее

Что нашли:

```
Book b = BOOK.as("b")
```

as — функция приведения в Kotlin

```
val b = Book("b")
```

Мы ынтырпрайз, у нас JPA, а не ваши хипстерские штучки

- У меня нет опыта
- Проблема — no-arg constructor в data-классах

Решение 1: не использовать data-классы
Абыдно

Решение 2: плагины

```
<artifactId>kotlin-maven-plugin</artifactId>
<groupId>org.jetbrains.kotlin</groupId>
<configuration>
<compilerPlugins><plugin>jpa</plugin></compilerPlugins>
   <jvmTarget>1.8</jvmTarget>
</configuration>
<dependencies>
  <dependency>
    <groupId>org.jetbrains.kotlin</groupId>
    <artifactId>kotlin-maven-noarg</artifactId>
  </dependency>
</dependencies>
```

Kotlin + JUnit. Тестирование. Part 1.

Всё работает!

Что нужно знать:

- `@BeforeAll` и `@AfterAll` нужно объявлять в компаньоне
- Для `Parametrized` тестов
  - `ClassRule`, `MethodRule` и `@Parameter` должен быть `@JvmField`

Kotlin + JUnit. Тестирование. Part 1.

```
companion object {
  @JvmStatic @BeforeAll
  fun setup() {}
  @ClassRule @JvmField
  val SPRING_CLASS_RULE = SpringClassRule()
}
@Rule @JvmField
var springMethodRule = SpringMethodRule()
```

Kotlin + JUnit. Тестирование. Part 2. Mockito.

- `anyObject()` возвращает `null`
  - это плохо потому что на обращении к объекту Kotlin проверяет что объект — не `null`
- `anyString()` тоже!
- `when` — ключевое слово

Kotlin + JUnit. Тестирование. Part 2. Mockito.

```
import org.mockito.Mockito.`when` as on
inline fun <reified T> kAnyObject(): T =
        kAnyObject(T::class.java)
inline fun <reified T> kAnyObject(t: Class<T>): T =
        Mockito.any(t)
```

Kotlin. Тестирование.
Part 3. Плюшки. Красивый mockito

```
@Test
fun doAction_doesSomething(){
  val mock = mock<MyClass> {
    on { getText() } doReturn "text"
  }
  val classUnderTest = ClassUnderTest(mock)
  classUnderTest.doAction()
  verify(mock).doSomething(any())
}
```

mockito-kotlin

Kotlin. Тестирование.
Part 3. Плюшки. Понятный нейминг

```
fun `I am readable unit test name which describes what is
tested`(){
  assertTrue(true)
}
```

Kotlin. Тестирование. Part 3. Плюшки. BDD

```
object SimpleSpec: Spek({
  describe("a calculator") {
    val calculator = SampleCalculator()
    on("addition") {
      val sum = calculator.sum(2, 4)
      it("should return the sum") {
          assertEquals(6, sum)
```

Spek v.1

Kotlin. Тестирование.
Part 3. Плюшки. Matchers

```
23.should.equal(23)
"Kotlin".should.not.contain("Scala")
listOf(1,2,3).should.have.size.above(1)
```

Expekt

To run{} or not to run{}

```
class Controller {
  fun apiCall(arg: String): ComplexBusinessDTO {
    val interim = myService.call(arg)
    return postProcess(interim)
  }
}
```

To run{} or not to run{}

Но у нас же есть автоматический вывод типов!

```
class Controller {
  fun apiCall(arg: String) = run {
    val interim = myService.call(arg)
    postProcess(interim)
  }
}
```

Не делайте так!

Пару слов об интеропе

- Всё очень хорошо
- Java 2 Kotlin работает достаточно плохо
  - Вообще, наверное, не используйте
- Котлин не умеет raw-types

```
public class BrowserWebDriverContainer<SELF extends
             BrowserWebDriverContainer<SELF>>  {...}
```

```
BrowserWebDriverContainer<Nothing>()
```

Не работают красивые билдеры

```
class KBrowserWebDriverContainer() :
  BrowserWebDriverContainer<KBrowserWebDriverContainer>()
```

Лапшекод
и мусор

Работа с коллекциями

- Всё работает
- Стримы тоже — нужна библиотека `kotlinx-support-jdk8`
  - Работают даже лучше за счёт методов `toList()` и тд
- Стримы не нужны — есть `asSequence()`
  - Который работает даже на массивах
- Больше функциональных методов
  - Например, `zip`

Bonus-track: Graal VM EE

- Sequence может догонять по производительности обычные for-циклы и далеко обгонять Stream
- Есть бенчмарк
  - Но греется до этого состояния очень долго

Mein kampf с аннотациями

```
data class Person (
  @NotNull
  val name: String,
  @Min(18)
  val age: Int,
  @CustomAnno
  val parents: List<Person>?
)
```

Что может пойти не так?

Mein kampf с аннотациями

```
data class Person (
  @NotNull
  val name: String,
  @Min(18)
  val age: BigDecimal,
  @CustomAnno
  val parents: List<Person>?
)
```

Что может пойти не так?

Mein kampf с аннотациями

```
data class Person (
  @field:NotNull
  val name: String,
  @field:Min(18)
  val age: BigDecimal,
  @field:CustomAnno
  val parents: List<Person>?
)
```

- Сразу непонятно, но аннотации садятся на аргументы конструктора
  - О которых `hibernate-validator` не знает
- Нас спасает ключевое слово `field`
- Иногда из-за логики сигнатура типа и аннотация конфликтуют

JSON + Jackson

- Всё работает!
- Чтобы работали data-классы и плюшки — нужен `jackson-module-kotlin`
- Работает с конструктором!
- и с полями тоже
- Поддерживает `Pair`, `Triple`, `Range`

```
import
com.fasterxml.jackson.module.kotlin.*

val mapper = jacksonObjectMapper()
```

Архитектура

- Архитектура приницпиально не меняется
- Появляются новые возможности
  - Можно прокидывать мапперы в репозитории
- В цепочках `CompletableFuture` прокидывать `Either` вместо отдельной обработки исключений

Как мы свои монады писали

- На самом деле только одну: `Either`
- Удобно когда у тебя в JSON может быть один из двух объектов
- Нужно для того чтобы пробрасывать результат выполнения, когда часть кода написана в функциональном стиле
- Отлично подходит для `Future`

Как мы свои монады писали

```
sealed class Either<out LEFT, out RIGHT> {
  class Left<LEFT>(private val left: LEFT) : Either<LEFT, Nothing>() {
    override fun <R> mapLeft(trans: (LEFT) -> R) = Left(trans(left))
    override fun <R> mapRight(trans: (Nothing) -> R) = this
  }
  class Right<RIGHT>(private val right: RIGHT) : Either<Nothing, RIGHT>() {
    override fun <R> mapLeft(trans: (Nothing) -> R) = this
    override fun <R> mapRight(trans: (RIGHT) -> R) = Right(trans(right))
  }
  abstract fun <R> mapLeft(trans: (LEFT) -> R): Either<R, RIGHT>
  abstract fun <R> mapRight(trans: (RIGHT) -> R): Either<LEFT, R>
}
```

Either

Раньше:

```
CompletableFuture
    .supplyAsync { someCall() }
    .exceptionally { /* What am I supposed to do here? Blow up? */ }
    .thenApply { input → process(input) }
    .thenAccept { result → println(result) }
```

Either

А теперь:

```
CompletableFuture
    .supplyAsync { eitherTry(someSyncCall()) }
    .thenApply { input → input.mapRight { process(it) } }
    .thenAccept { when(it) {
        is Left → handleError()
        is Right → handleResult()
    } }
```

> *В программировании есть две проблемы: инвалидация кешей и именование переменных*

Неизвестный мне философ

Именование переменных

- Функции без классов, обычно получают класс `FilenameKt`
  - Например если функции лежат в `Utils.kt` – из джавы они будут выглядеть как `UtilsKt.functionName()`
- Иногда нам надо чтобы функция именовалась в джаве иначе

- `@JvmName` to the rescue!

Как мы раскатали практику на полкомпании

- Количество кода уменьшилось
- Покрывать его тестами стало гораздо проще
- Если весь код написан на Kotlin — то как правило можно не думать о `null`
- Множество библиотек с очень приятным синтаксисом делает код красивее

- Этого оказалось достаточно чтобы распространить практику

Outro

- Фатальных проблем нет
- Те которые есть — решаются, в основном без костылей
- Экосистема огромна
- Писать приятно, кода становится меньше
- Рекомендую :)

Спасибо за внимание!

Остались вопросы?

 asm0di0

 asm0dey

 asm0dey