Kotlin: MAX PAYNE

Artur Dryomov, Juno Lab

@arturdryomov

KONG WHISKEY
ROSCOE ST
THE BRO

Juno ❤️ Kotlin

Kotlin 💔 Juno

Well, only sometimes, so no grudge here

Juno Rider 🚙

- Full-Kotlin + RxJava from the start
- 2 years
- Kotlin `1.1-EAP` in production for 6+ months 🔥
- Kotlin `1.2-EAP` right now 🔥
- 99.9 % crash-free
- 2 week sprints
- 5️⃣ developers, 2️⃣ QA

Juno Rider 🚙

- `cloc main/kotlin` — 40 K
- `cloc test/kotlin` — 38 K
- `cloc androidTest/kotlin` — 12 K

Style 💄

Checkstyle and Static Analysis

So yeah, forget about it ✨

Specification mumbo-jumbo 📐

- Java AST — Abstract Syntax Tree
    - Only Java
- PSI — Program Structure Interface
    - Only IJ
- UAST — Universal AST
    - Java and Kotlin
    - Android Lint is migrating to it!

Checkstyle

> How about not mapping it to AST and instead map it to UAST (Universal Abstract Syntax Tree)? That way also adding support for Kotlin, which is an official thing for Android now, can be supported.

> Checkstyle works only with AST as we depend on ANTLR and Java grammar.

Checkstyle and Static Analysis 🔦

Put your IJ config into repository

You can somewhat run IJ headless

Detekt works with PSI, so take it for a spin

∞ paths

Let’s build something!

```
val gameBuilder = StringBuilder()

gameBuilder.append("tic")
gameBuilder.append("tac")
gameBuilder.append("toe")

val game = gameBuilder.toString()
```

∞ paths

Also, `also`.

```kotlin
val game: String = StringBuilder().also {
    it.append("tic")
    it.append("tac")
    it.append("toe")
}
.toString()
```

∞ paths

Hmm, `let` it be?

```
val game: String = StringBuilder().let {
    it.append("tic")
    it.append("tac")
    it.append("toe")

    it.toString()
}
```

∞ paths

`apply` it!

```
val game: String = StringBuilder().apply {
    append("tic")
    append("tac")
    append("toe")
}
.toString()
```

∞ paths

`run` seems nice.

```
val game: String = StringBuilder().run {
    append("tic")
    append("tac")
    append("toe")

    toString()
}
```

∞ paths

How about `with`?

```
val game: String = with(StringBuilder()) {
    append("tic")
    append("tac")
    append("toe")

    toString()
}
```

∞ paths

- Better brew that coffee for long PRs ☕
- `?:` vs. `let` vs. `if (it == null)`

∞ paths 🔦

- `CODE_STYLE.md`
- It does not really matter, just go with it ✈️

Weird Parts 🎃

Destructuring

No joke, actually useful.

```
val (first, second) = pair
val (first, second, third) = triple

Observable
    .fromCallable { createPair() }
    .subscribe { (first, second) ->
        println("1st is $first, 2nd is $second")
    }
```

Destructuring

Supports `data class` and suggested by IJ.

```
data class Person(
    firstName: String,
    lastName: String
)

val (firstName, lastName) = person
```

Destructuring

Somebody swaps fields 🔫 Compiler is happy! ✨

```
data class Person(
    lastName: String,
    firstName: String
)

val (firstName, lastName) = person
```

Destructuring 🔦

We just ban it unless `Pair` or `Triple`

`null` safety?

This works just fine!

```
fun createObservable(): Observable<Int> {
    return Observable.just(null)
}
```

✨ OH HAI RxJava 2 ✨

Reproduced on RxJava 2.1.1.

`null` safety!!

- The language relies on `@NonNull` annotations.
- Your code is full of unicorns 🦄
- Java code is full of `null`.

`null` safety!! 🔦

- Read docs and sources carefully.
- Promote `@Nullability` to your favorite libraries.
- Hopefully Android SDK will catch up after announcements.

RxJava 2

- Streams panic on `null` values.
- Kotlin has to be friends with Java.
- Side road — Swift vs. Kotlin 🐦
    - Swift: `?` and `nil` just sugar for `enum Optional`.
    - Kotlin: `?` is a language helper, no guarantees though.

Koptional 🔦

https://github.com/gojuno/koptional

```
val some = Some(value)
val none = None

val optional = nullableValue.toOptional()
val nullable = o.toNullable()

val valueStream = optionalStream.filterSome()
val signalStream = optionalStream.filterNone()
```

Use `?:` and `let` — embrace the language.

SAM Conversion

Single Abstract Method

```
// RxJava 1
Observable.zip(a, b) { a, b -> a to b}

// RxJava 2
Observable.zip(a, b, BiFunction<Int, Int> { a, b ->
    a to b
})
```

SAM Conversion

```
// RxJava 1
public static <T1, T2, R> Observable<R> zip
(
    Observable<T1> o1, Observable<T2> o2,
    Func2<T1, T2, R> zipper
)

// RxJava 2
public static <T1, T2, R> Observable<R> zip
(
    ObservableSource<T1> o1, ObservableSource<T2> o2,
    BiFunction<T1, T2, R> zipper
)
```

SAM Conversion

```
// RxJava 1
public interface Func2<T1, T2, R> extends Function {
    R call(T1 t1, T2 t2);
}

// RxJava 2
public interface BiFunction<T1, T2, R> {
    R apply(T1 t1, T2 t2);
}
```

SAM Conversion

```
// RxJava 1
public class Observable<T> {
    // Redacted
}


// RxJava 2
public interface ObservableSource<T> {
    void subscribe(Observer<T> observer);
}
```

SAM Conversion

⭐ https://youtrack.jetbrains.com/issue/KT-14984

Impossible to pass not all SAM argument as function

SAM Conversion 🔦

We’ve made a bunch of extensions for now

RxKotlin has some

Gson and Reflection

```
data class Model {

    @SerializedName("model_field")
    val field: String

}

model.field.toString() // Crash!
```

Gson and Reflection

Gson is happy to write `null` to properties

Reflection is a thing in the Java world...

Gson and Reflection

Fail fast 💥

```
data class Model {

    @SerializedName("model_field")
    val field: String

} : Validatable {

    override fun validate() {
        if (field == null) throw ParseException()
    }
}
```

Gson and Reflection 🔦

Take a look at Moshi or Jackson

Uses Kotlin native reflection 😇

Optimizations 🚀

`inline` or `noinline`

- Neat, reduces methods count.
  - OH HAI Android ✨
- Feels like kernel development.
- Not emraced at all.
- Mostly forces `@Suppress("nothing_to_inline")`.

`inline` or `noinline`

⭐ https://youtrack.jetbrains.com/issue/KT-16769

Revisit compiler and IDE warning about inline functions

`inline` or `noinline` 🔦

Know about it

I’m looking at you, `private fun`

`companion object` and `const`

```kotlin
// Kotlin
companion object {
    private val NUMBER = 42
}
```

```java
// Java
public static final class Companion {
    private final int NUMBER = 42;

    private final int getNUMBER() {
        return NUMBER;
    }
}
```

`companion object` and `const`

- So yeah, +1️⃣ method without `const`.
- Methods will not count themselves!
- Invocation costs are real 🕘

`companion object` and `const`

```
// Kotlin
companion object {
    private const val NUMBER = 42
}
```

```
// Java
public static final class Companion {
    private static final int NUMBER = 42;

    // Will actually inline the value.
}
```

`companion object` and `const`

- Not advocated enough.
- Only primitives and `String`.
- Shouldn’t compiler do it itself?

`companion object` and `const` 🔦

Know about it

Hidden Costs 🔦

IJ → Tools → Show Kotlin Bytecode

Lots of fun, also — scary! 😱

Tooling 🔨

Bugs 🐛

```
fun multiply(double: Double) = double * double
```

Update Kotlin 1.1.2 ➡️ 1.1.3.

```
java.lang.IllegalStateException
org.jetbrains.kotlin.kapt3.diagnostic.KaptError
Error while annotation processing
```

Java keywords are suddenly banned.

Changelog? What changelog?

Bugs 🐛

⭐ https://youtrack.jetbrains.com/issue/KT-18377

Syntax error while generating kapt stubs

Bugs 🐛

Bugs in the language tooling are a thing

Assemble 🕘

- Coming from Java? +1️⃣ build step.
- Use Dagger? +1️⃣ build step.
- `kapt` incremental compilation didn’t work for a long time...
  - Have a change? `REBUILD` .
- Just remember you are doing it 💯 times per day.

Assemble 🕘

- Compilation is CPU intensive.
- Laptops have mobile CPUs.
- PCs are cheap, easy to upgrade...

Mainframer 💻

https://github.com/gojuno/mainframer

https://github.com/elpassion/mainframer-intellij-plugin

1. Laptop → *Source Code* → Build Machine.
2. Build Machine → *Build Results* → Laptop.

Mainframer 💻

```
$ ./gradlew clean assembleDebug --no-daemon
```

- Without MF: 4 minutes 10 seconds.
    - Laptop overheat 🔥
    - Basically cannot use your computer 💔
- With MF: 1 minute 10 seconds.

Assemble 🕘 🔦

Do all optimizations you can

- Minimum SDK to Lollipop.
- Gradle build cache.
- Gradle Android Plugin DEX build cache.
- Mainframer 💻

Spek 🎲

Spek

```
@RunWith(JUnitPlatform::class)
class Spec : Spek({

    val system by memoized { System() }

    context("boot") {

        beforeEachTest {
            system.boot()
        }

        it("reports boot status") {
            assertThat(system.status()).isEqualTo(BOOTED)
        }
    }
})
```

Spek: versions

- `1.0.89`
  - `DescribeBody` → `Dsl` 💣
- `1.1.0-beta3`
  - `Dsl` → `SpecBody` 💣
- Docs mention `1.0`, `1.1.19`.
  - Do not exist.
- Moved from JUnit 4 to JUnit 5 at some point 💣

Spek: versions 🔦

⭐ https://github.com/JetBrains/spek/issues/170

Please fix Spek versioning, compatibility and publishing

Spek: `memoized` caching

```kotlin
@RunWith(JUnitPlatform::class)
class OopsSpec : Spek({

    val system by memoized { System() }

    listOf(1, 2, 3).forEach { index ->

        context("boot") {

            it("boots") {}
        }
    }
}
```

Same `system` object for each run 🔫

Spek: `memoized` caching

```
@RunWith(JUnitPlatform::class)
class OkSpec : Spek({

    val system by memoized { System() }

    listOf(1, 2, 3).forEach { index ->

        context("boot $index") {

            it("boots") {}
        }
    }
}
```

Unique `system` object for each run 🎈

Spek: `memoized` caching 🔦

⭐ https://github.com/JetBrains/spek/issues/210

Memoized and same test names

Spek: `memoized` before and after

```
@RunWith(JUnitPlatform::class)
class JustSpec : Spek({

    val system by memoized { System() }

    beforeEachTest {
        println("before $system")
    }

    afterEachTest {
        println("after $system")
    }
})
```

Before and after `system` are different.

Spek: `memoized` before and after 🔦

⭐ https://github.com/JetBrains/spek/issues/213

Memoized value is not the same before and after test

Spek: JUnit

- Actually JUnit 5 from now on.
- Use `@RunWith(JUnitPlatform::class)` to work with JUnit 4.
- JUnit 5 is not released, so IJ may not work.
  - *OH HAI Android Studio 3 Beta*.
- `fon` and `fit` were removed, use Spek IJ plugin.
  - *Does not work with JUnit 4*.
- ✨ This is fun ✨

Spek

- Kind of advertized by JetBrains, but...
- Commits are two times a month on average 😴
- Breaks compatibility easily 💣
- Docs feel outdated all the time.
- There are bugs. Remember vanilla JUnit stability?
- Also, have 100 K tests? Have to maintain it 😡

BEE

Bonus round 🎮

Bonus round 🎮

You are on your own

Prepare that YouTrack account...

Bonus round 🎮

> …
> Kotlin is very reference semantics, it’s a thin layer on top of Java, and so it perpetuates through a lot of the Javaisms in its model.
> …
> If we had done an analog to that for Objective-C it would be like, everything is an NSObject and it’s objc_msgSend everywhere, just with parentheses instead of square brackets.
> …

Chris Lattner, Swift 🐦

Bonus round 🎮

- Java: Oracle, Red Hat, Intel, IBM, Eclipse, Twitter, JetBrains, etc.
- Kotlin: JetBrains.
- This is fine actually, but do not expect a miracle.
- For a single company a damn great language ❤️

Conclusion

- There will be dragons 🐉
- Kotlin will not make you a better developer.
  - It will give you tools though 🔨
- Remember about Java underneath.
  - Swift can break things, Kotlin cannot 🐦
- The language is top-notch and well-designed.
- Be careful!

KONG WHISKEY
ROSCOE ST
THE BRO