goto; copenhagen

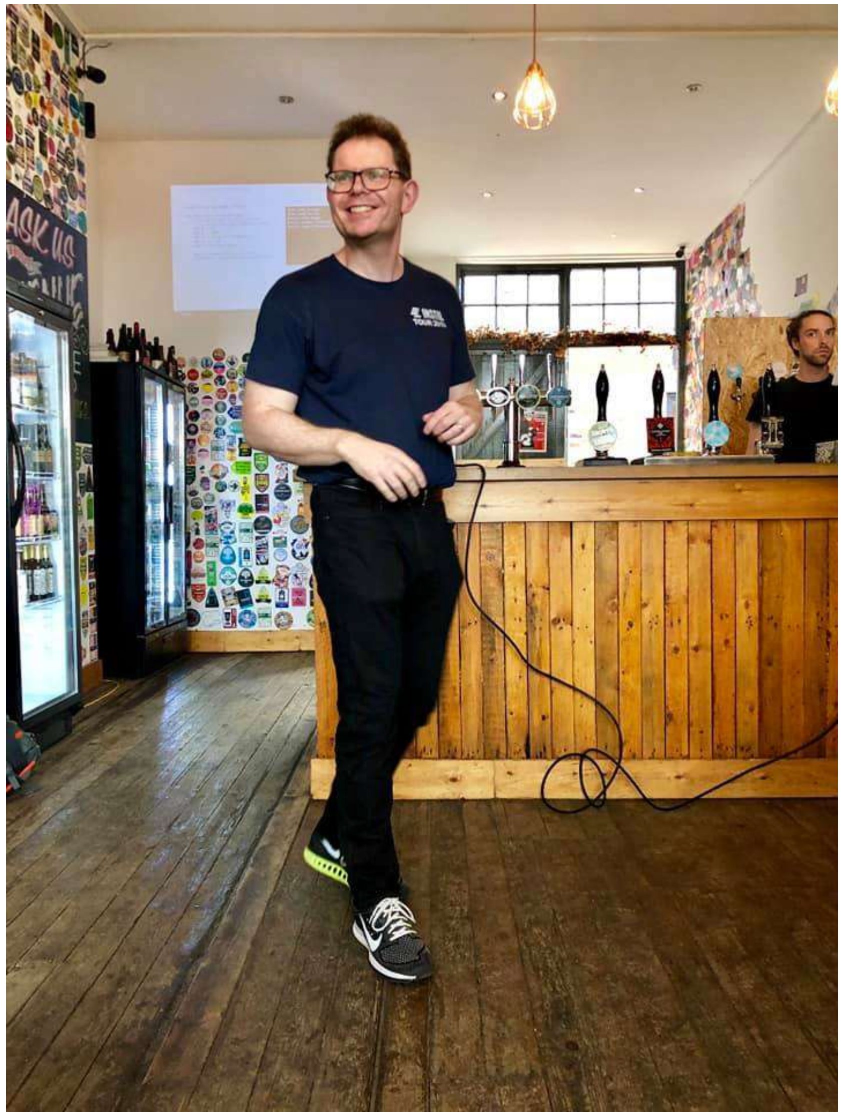
GOTO Copenhagen 2019
Conference Nov. 18 - 20

Transitioning Android Teams Into Kotlin

Garth Gilmour (@GarthGilmour)
Eamonn Boyle (@BoyleEamonn)

Follow us @gotocph

Me



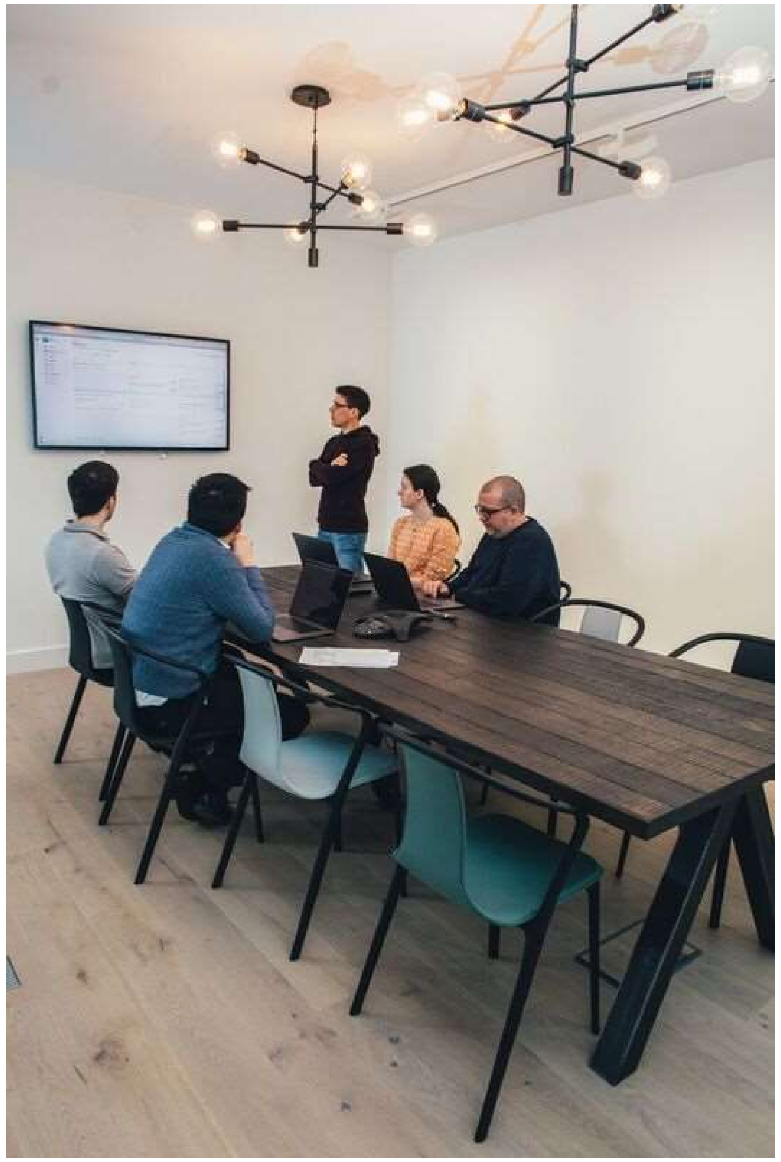
Clean Code
Extreme Programming Explained

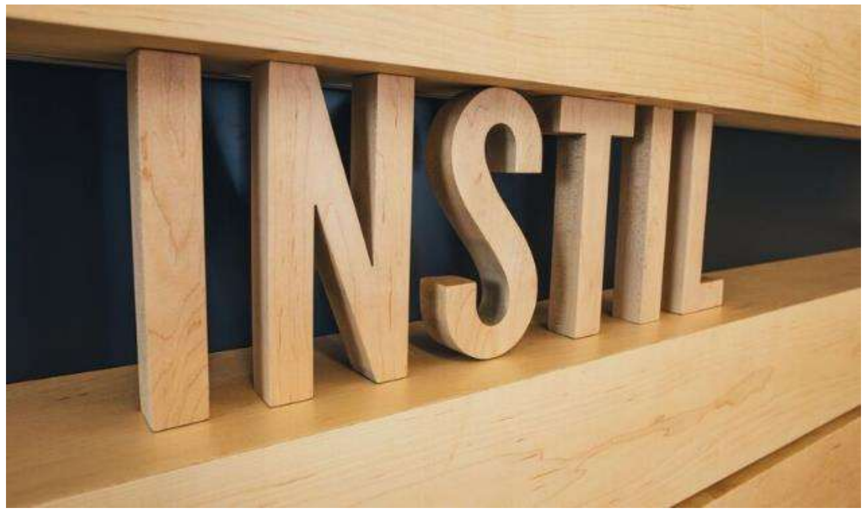
Garth Gilmour
@GarthGilmour
Richard Gibson
@rickityg
@Garth Gilmour
@rickityg

INSTIL

Him

Us





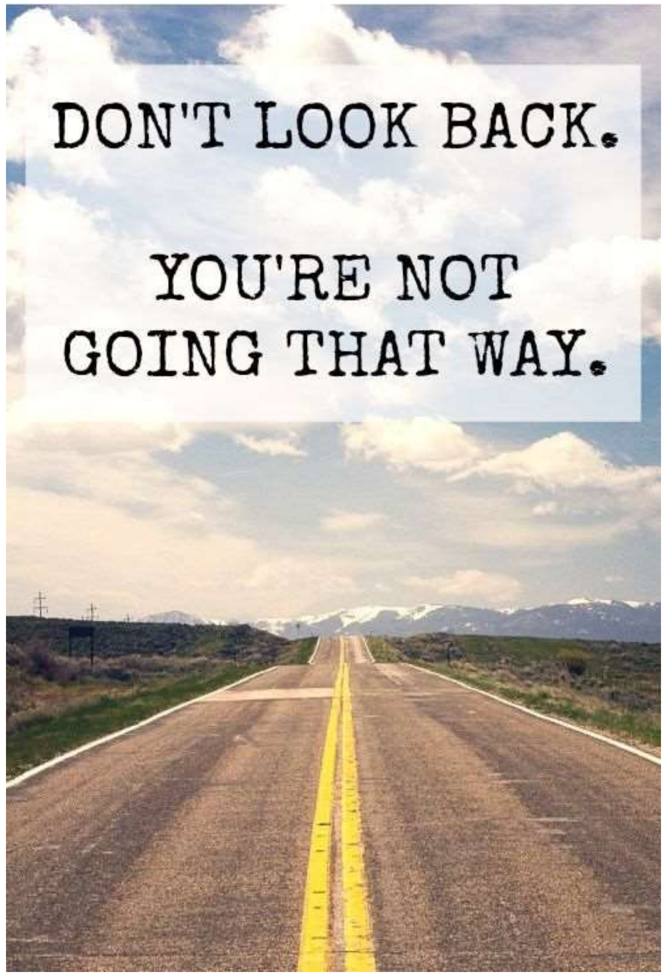
This Talk Is (Maybe) Slightly Out Of Date

2018 was a very significant year
- For developers on the JVM

We stopped asking:
- “Why aren’t you doing this in Java?”

We started asking:
- “Why **are** you doing this in Java?”

The revolution is actualised
- It’s a polyglot coding world now
- …and there’s no going back



“You thought Android devs still used Java”

They still are? Inconceivable!

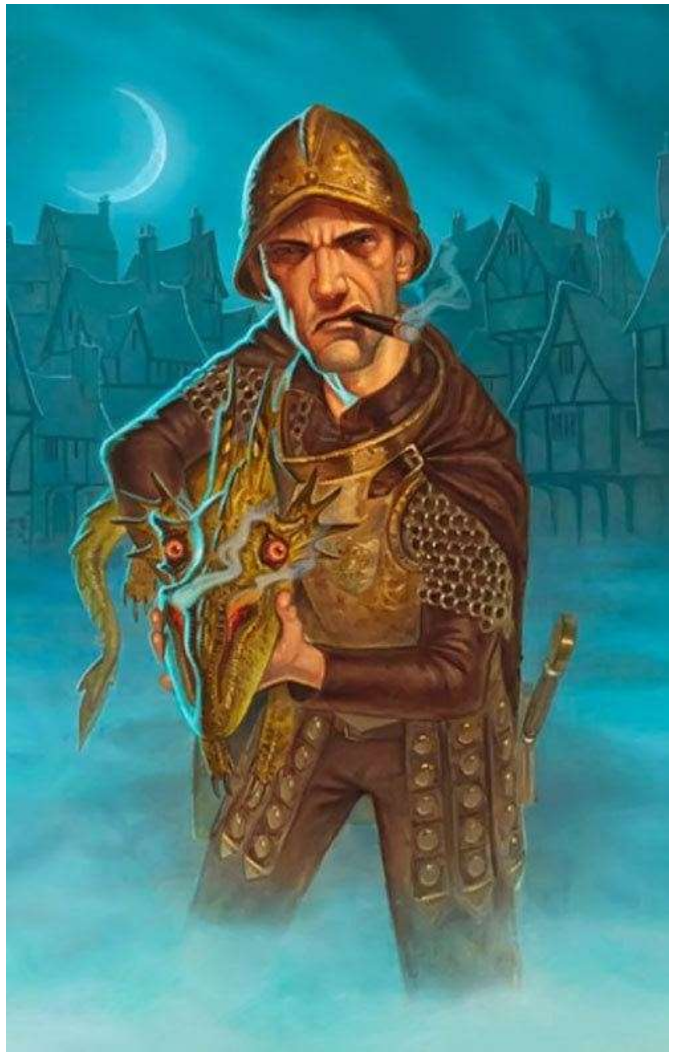
A Veteran's Guide To Moving Android Teams To Kotlin

This talk assumes:

- You know Java and Android
- You have the scars to prove it

We will convince you that:

- There is a better way
- Kotlin is that way



Why Code In Kotlin?

Excellent interoperability

- Call legacy Java code easily
- Make calls into Kotlin from Java

Really clear, yet concise syntax

- Reduces the codebase by over 40%

Beloved by frameworks

- Spring Boot (esp. with Kofu)
- Services (Ktor and Http4k)
- User Interfaces (TornadoFx)
- Build tools (Gradle from V5)
- Dependency Injection (Koin)

Null Safety
String Templates
Default parameters
Extensions
Free Functions
Coroutines
Single Expression Functions
Reified generics
Data classes and Properties
Type Inference
Smart Casts
Operator overloading

Let’s Print The Alphabet - In Java

```java
public class Program {
    public static void main(String[] args) {
        List<String> data = Arrays.asList(
                "ab","cde","fghi","jklmn",
                "op","qrs","tuvw","xyz");

        data.stream()
            .flatMapToInt(CharSequence::chars)
            .mapToObj(num -> (char)num)
            .forEach(c -> System.out.printf("%s ", c));
    }
}
```

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Let’s Print The Alphabet - In Kotlin

```kotlin
fun main() {
    val data = listOf(
        "ab","cde","fghi","jklmn",
        "op","qrs","tuvw","xyz")

    data.flatMap { it.toList() }
        .forEach { print("$it ") }
}
```

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Why Use Kotlin On Android?

“Today we’re announcing another big step: Android development will become increasingly Kotlin-first. Many new Jetpack APIs and features will be offered first in Kotlin. **If you’re starting a new project, you should write it in Kotlin**;”

Google I/O 2019

Why Use Kotlin On Android?

Codebase reduction by 40% (JetBrains figure)

Simplify the remaining code

Plaster over Androids cracks

Access FP features / patterns (Java 8+)

Future proof your skill set

Developer happiness ☺

Why Listen To Us?

We’ve written an awful lot of code…

- Especially mobile applications with complex networking
- Also embedded systems, web applications etc...

For an awful lot of clients…

- Atlassian, Johnson Controls, Shopkeep, Cybersource

On every platform you’ve ever heard of

- Plus (if you were lucky) a few you managed to avoid

We switched to Kotlin four years ago

- For all applications built on top of the JVM
- Plus we are experimenting with Native and JS

Why Listen To Us?

Instil is a JetBrains training partner

- Written 3 Kotlin courses so far

We have delivered training for

- Household names (Europe and US)
- Brand new start-ups

We believe Kotlin will help you write better solutions

- Engineering Expertise is part of our ethos and Kotlin helps us
- We believe in improving development practices within our community

We started the Kotlin Belfast User Group

- We promote it to help people, not to make money

Why Listen To Us?

We don’t get royalties from JetBrains or Google

We’re placing a bet on Kotlin being the future

INSTIL

Questions To Answer

- What motivated us to switch?
- How did we make the move?
- Which features do we love?
- What problems did we face?
- Did we require new frameworks?
- What would we do differently?
- Are coroutines of any use? *

* we want to talk about coroutines because they're awesome…

What Motivated Us?

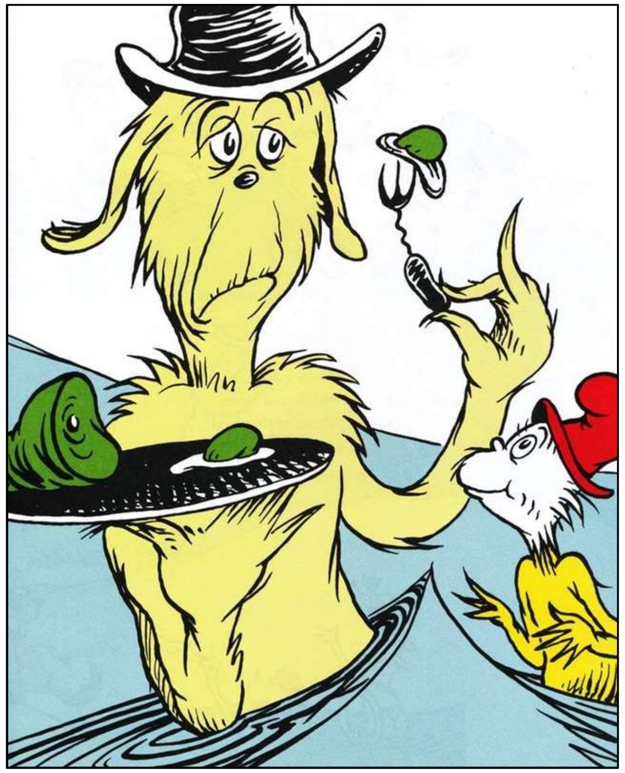
Prior to Kotlin we had written:

- iOS mobile apps in Objective-C and Swift
- Cross-platform mobile apps in Xamarin
- Standard desktop apps in C / C++ / C#
- Web applications in just about everything
- JEE in all its incarnations (including EJB)
- Spring from the earliest days to WebFlux

We were tendering for some major Android jobs

- The developers didn't want to use Java
- <u>They really didn't want to use Java</u>

Words From The Wise



"I do not like this Java lang
I do not like it, business man
It does not suit our software house
It makes me sad to click the mouse
I will not code it here or there
It sucks for Android anywhere"

David Robert Seuss, Software Engineer
(personal conversation with our CEO)

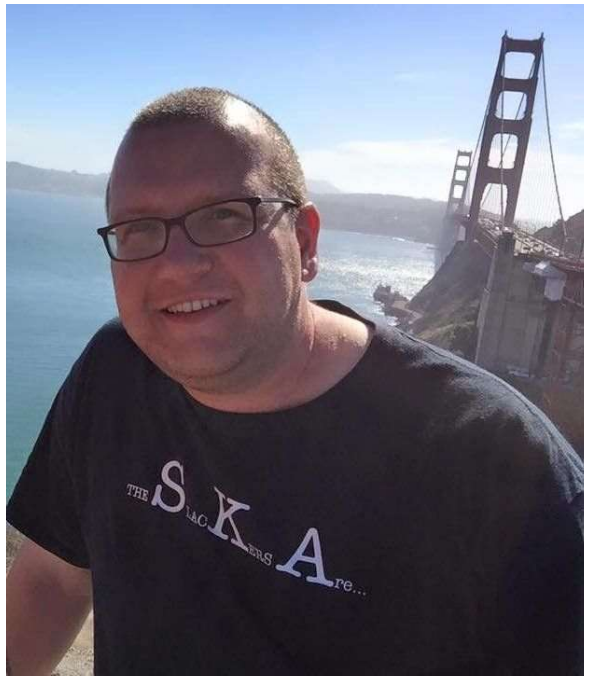
What Motivated Us?

We had a problem on the Java Virtual Machine
- Kotlin was a pragmatic solution

By todays standards Java is not great
- But lots of great software is written in it

We still leverage our historical knowledge
- By remaining within the JVM ecosystem

Words from the Wise



"Making the switch from Java to Kotlin was easy - being able to use the same tools and libraries that we knew and loved meant that we felt productive in Kotlin almost immediately.

Over time, our code then became simpler, more stable and more performant as we started to experiment with the Kotlin functional toolkit and coroutines."

Chris van Es, Head of Engineering, Instil

Words from the Wise

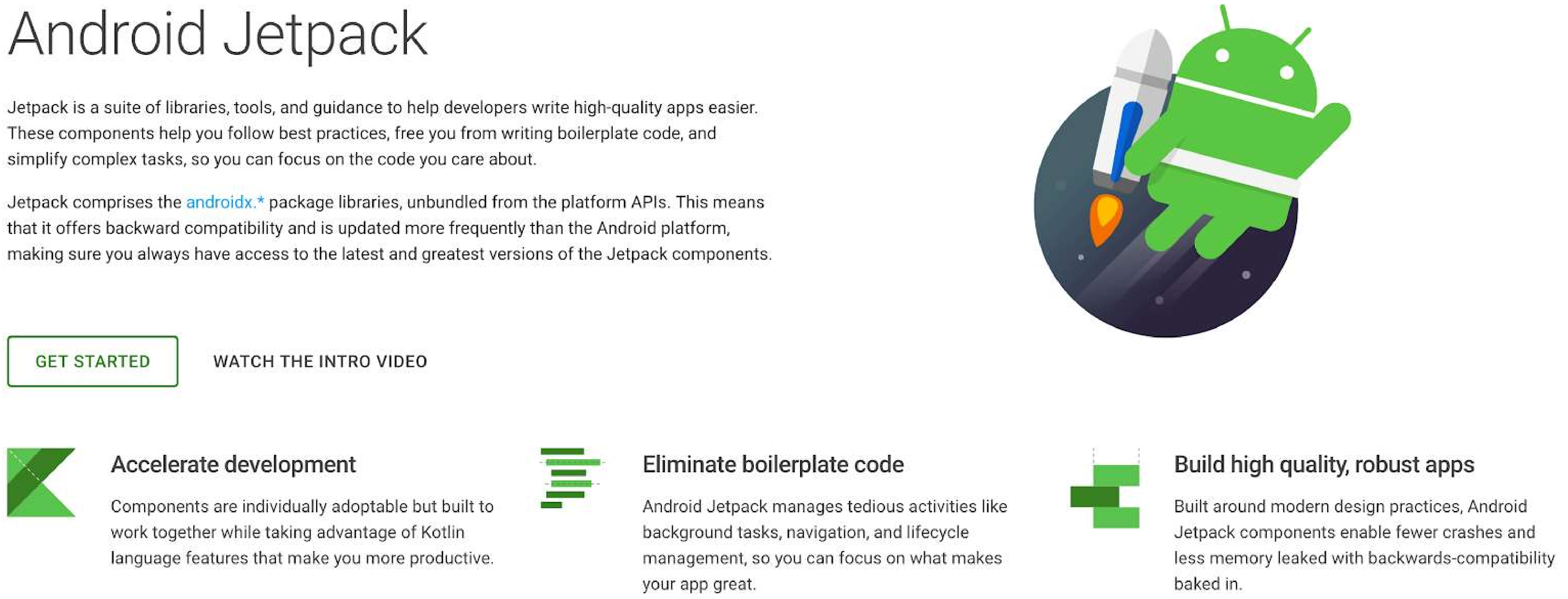
“Working in Java is a fantastic developer experience because you can use Kotlin.”

Kelvin Harron, Software Engineer, Instil

“Kotlin negates the biggest weakness of the Java Ecosystem, i.e. Java.”

Eoin Mullan, Principal Software Engineer, Instil

The Community Was Going That Way



Android Jetpack

Jetpack is a suite of libraries, tools, and guidance to help developers write high-quality apps easier. These components help you follow best practices, free you from writing boilerplate code, and simplify complex tasks, so you can focus on the code you care about.

Jetpack comprises the androidx.* package libraries, unbundled from the platform APIs. This means that it offers backward compatibility and is updated more frequently than the Android platform, making sure you always have access to the latest and greatest versions of the Jetpack components.

Accelerate development

Components are individually adoptable but built to work together while taking advantage of Kotlin language features that make you more productive.

Eliminate boilerplate code

Android Jetpack manages tedious activities like background tasks, navigation, and lifecycle management, so you can focus on what makes your app great.

Build high quality, robust apps

Built around modern design practices, Android Jetpack components enable fewer crashes and less memory leaked with backwards-compatibility baked in.

We wanted an epic, heroic tale of a warrior’s struggle to final victory

It was less Kill Bill, and more Kill Kenny – very easy

Features We

Love

In Case Of Interoperability Emergency…

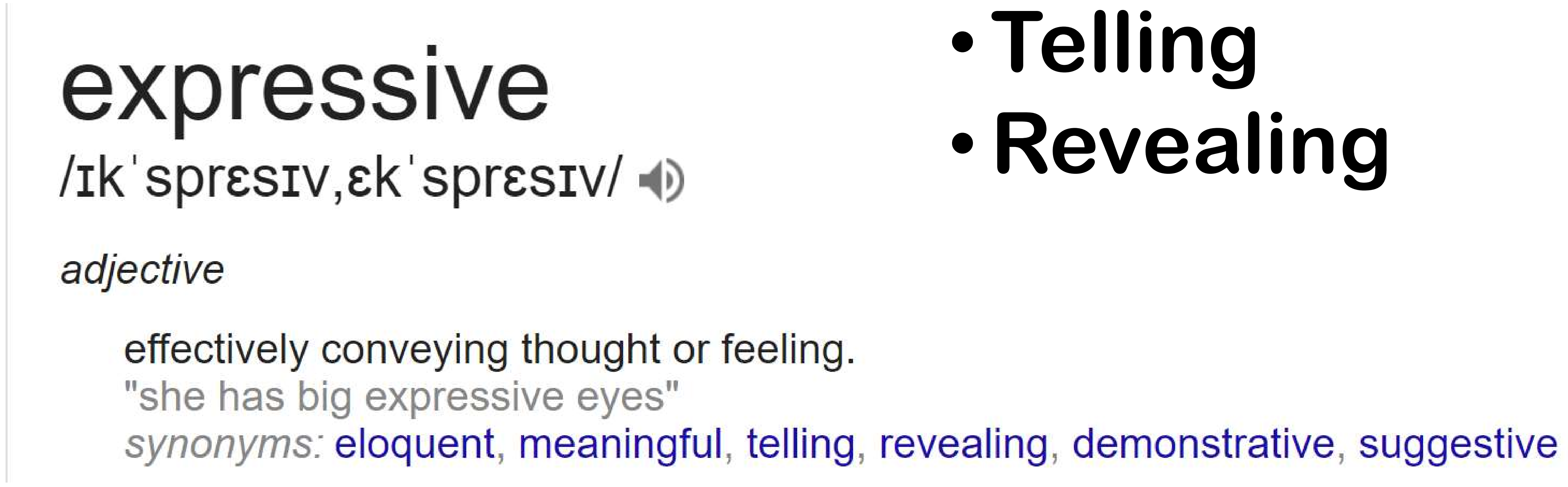
Kotlin's interop story is awesome

- @Throws (adds checked exception)
- @JvmOverloads (for default params)
- @JvmName (rename most things)
- @JvmWildcard (manage generics)

Interop never caused serious disruption

- But it was reassuring to know we had low level control should it be needed
- Emotional Support Annotations ☺

Expressive



expressive

/ɪkˈsprɛsɪv,ɛkˈsprɛsɪv/

adjective

effectively conveying thought or feeling.
"she has big expressive eyes"
synonyms: eloquent, meaningful, telling, revealing, demonstrative, suggestive

- **Meaningful**
- **Telling**
- **Revealing**

Basic Class Design

```java
public class Movie {
    private String title;
    private String description;
    private Rating rating;
    private Genre genre;
}
```

Basic Class Design

```java
public class Movie {
    private String title;
    private String description;
    private Rating rating;
    private Genre genre;

    public Movie(String title, String description, Rating rating, Genre genre) {
        this.title = title;
        this.description = description;
        this.rating = rating;
        this.genre = genre;
    }
}
```

Basic Class Design

```
public class Movie {
    private String title;
    private String description;
    private Rating rating;
    private Genre genre;

    public Movie(String title, String description, Rating rating, Genre genre) {
        this.title = title;
        this.description = description;
        this.rating = rating;
        this.genre = genre;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
```

Basic Class Design

```java
public class Movie {
    private String title;
    private String description;
    private Rating rating;
    private Genre genre;

    public Movie(String title, String description, Rating rating, Genre genre) {
        this.title = title;
        this.description = description;
        this.rating = rating;
        this.genre = genre;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Rating getRating() {
        return rating;
    }

    public void setRating(Rating rating) {
        this.rating = rating;
    }

    public Genre getGenre() {
        return genre;
    }

    public void setGenre(Genre genre) {
        this.genre = genre;
    }
}
```

```kotlin
class Movie(var title: String,
            var description: String,
            var rating: Rating,
            var genre: Genre) {
}
```

JOB DONE

Basic Class Design

```java
import java.util.Objects;

public class Movie {
    private String title;
    private String description;
    private Rating rating;
    private Genre genre;

    public Movie(String title, String description, Rating rating, Genre genre) {
        this.title = title;
        this.description = description;
        this.rating = rating;
        this.genre = genre;
    }

    public String getTitle() {
        return title;

    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Rating getRating() {
        return rating;
    }

    public void setRating(Rating rating) {
        this.rating = rating;
    }

    public Genre getGenre() {
        return genre;
    }

    public void setGenre(Genre genre) {
        this.genre = genre;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Movie movie = (Movie) o;
        return Objects.equals(title, movie.title) &&
                Objects.equals(description, movie.description) &&
                Objects.equals(rating, movie.rating) &&
                Objects.equals(genre, movie.genre);
    }

    @Override
    public int hashCode() {
        return Objects.hash(title, description, rating, genre);
    }

    @Override
    public String toString() {
        return "Movie{" +
                "title='" + title + '\'' +
                ", description='" + description + '\'' +
                ", rating=" + rating +
                ", genre=" + genre +
                '}';
    }
}
```

But what if we need:

- Equality
- Hashing
- Copying
- Decomposition

```kotlin
data class Movie(var title: String,
                 var description: String,
                 var rating: Rating,
                 var genre: Genre) {
```

JOB DONE

JOB DONE

INSTIL

Lambda’s With Receivers

```
menu {
    submenu(“Setup") {
        item(“Edit Details", ::editDetails)
        item(“Reset Password", ::resetPassword)
    }

    submenu(“Sessions") {
        item(“Create New Session", ::createNewSession)
        item(“View All Sessions", ::viewall)
        item(“Synchronize", ::synchronize)
    }
```

Lambda’s With Receivers

```kotlin
fun menu(builder: Menu.() -> Unit) = Menu(builder)

class Menu(builder: Menu.() -> Unit) {
    init {
        this.builder()
    }

    fun submenu(description: String, builder: Menu.() -> Unit) { ... }

    fun item(description: String, action: Action) { ... }

    . . .
}
```

Lambda’s With Receivers

```
fun welcome() = sendCommandsToDevice {
    appendBootstrap()
    appendSelfCheck()
    appendBeep(BeepCode.WELCOME)
}

fun resetDevice() = sendCommandsToDevice {
    appendReset()
    appendFlush()
}
```

Lambdas With Receivers

```kotlin
private fun sendCommandsToDevice(instructions: ICommandBuilder.() -> Unit) {
    val commands = ioPortDelegate
        .createDeviceCommandBuilder()
        .apply(instructions)
        .commands

    sendDeviceCommands(commands)
}
```

Lambdas With Receivers

```
sourceSets {
    create("integrationTest") {
        withConvention(KotlinSourceSet::class) {
            kotlin.srcDir("src/integrationTest/kotlin")
            resources.srcDir("src/integrationTest/resources")
            compileClasspath += sourceSets["main"].output + configurations["testRuntimeClasspath"]
            runtimeClasspath += output + compileClasspath + sourceSets["test"].runtimeClasspath
        }
    }
}
```

Lambdas With Receivers → Standard Utility Functions

```
fun <T> with(obj: T, f: T.() -> Unit) = obj.f()

class Tank {
    fun forward(distance: Int) = println("Going forward $distance")
    fun backward(distance: Int) = println("Going backward $distance")
    fun turn(degrees: Int) = println("Turning $degrees degrees")
    fun fire() = println("Firing cannon")
}

fun main() {
    val tank = Tank()
    with(tank) {
        forward(200)
        turn(45)
        backward(50)
        fire()
    }
}
```

```
Going forward 200
Turning 45 degrees
Going backward 50
Firing cannon
```

Standard Utility Functions

```
@kotlin.internal.InlineOnly
public inline fun <T, R> with(receiver: T, block: T.() -> R): R {
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }
    return receiver.block()
}
```

Standard Utility Functions

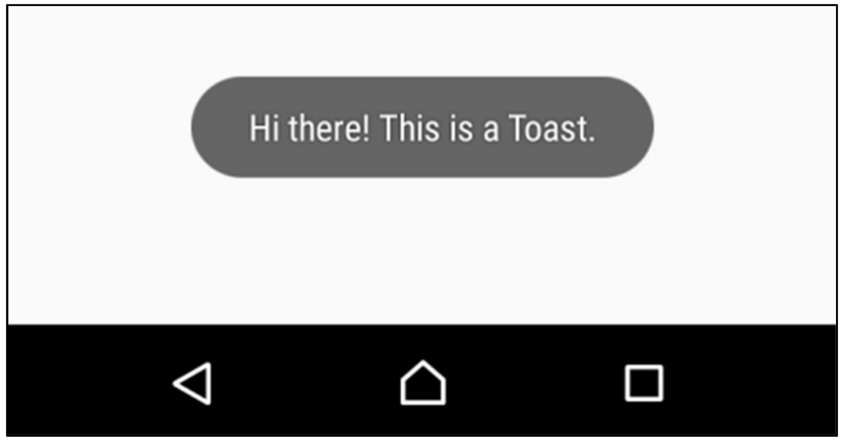
```
with(navigationBar) {
    navigationBarTitle.visibility = View.GONE
    searchBar.visibility = View.VISIBLE
    searchBar.inputType = inputType
    searchBar.transformationMethod = null
}
```

Standard Utility Functions

		Current Object Returned	Result Of Block Returned
Current Object passed as receiver	`with`	`apply`	`run`
Current Object passed via ‘it’		`also`	`let`

Standard Utility Functions

```
val toast = Toast.makeText(context, message, duration).apply {
    this.duration = duration
    this.view = buildToastView(view, message)
    this.setGravity(CENTER_VERTICAL, 0, Y_AXIS_OFFSET)
}
```


Hi there! This is a Toast.

Standard Utility Functions

```
leftSwipe.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}

rightSwipe.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}
```

Delegated Properties

```
class MyThing {
    var firstProp: Int = 123

    var secondProp: Int
        get() = firstProp * 2
        set(value) {
            firstProp = value / 2
        }

    var thirdProp by Delegate()
}
```

Delegated Properties

```
class Delegate {
    private var value: String = "Homer“

    operator fun getValue(thisRef: Any?,
                          property: KProperty<*>) : String {
        println("---> Getting $value")
        return value
    }
    operator fun setValue(thisRef: Any?,
                          property: KProperty<*>, value: String) {
        println("---> Setting $value")
        this.value = value
    }
}
```

Delegated Properties

Note the type
A lazy object manages the initialisation but it is consumed as a simple string

```
val saveButtonText: String by lazy {
    resourceService.getStringResource(R.string.save)
}

val addButtonText: String by lazy {
    resourceService.getStringResource(R.string.add)
}

val cancelButtonText: String by lazy {
    resourceService.getStringResource(R.string.cancel)
}
```

Delegated Properties

```
fun <T> weak(value: T? = null) = WeakReferenceDelegate(value)

class WeakReferenceDelegate<T>(value: T?) {
    private var weakReference = WeakReference(value)

    operator fun getValue(thisRef: Any,
                          prop: KProperty<*>): T? = weakReference.get()

    operator fun setValue(thisRef: Any, prop: KProperty<*>, value: T) {
        weakReference = WeakReference(value)
    }
}
```

Delegated Properties

```
private var dialog: InProgressDialog? by weak()
```

Note the type
A WeakReferenceDelegate object manages the WeakReference but it is read and written as the inner type

Implementation by Delegation

```
abstract class BaseActivity<ViewModelType : ActivityViewModel>
    : AppCompatActivity(),
      DisposableManager,
      CoroutineScope by MainScope() {

    . . .
}
```

Extension Functions

Extensions allow us write more expressive code

They also help us create Domain Specific Languages

```
deviceStatusStream()
    .filterForErrors()
    .collate().and()
    .send()
```

Extension Functions

```
private fun Observable<Long>.runningAverage(bufferSize: Int) =
    this.buffer(bufferSize, 1)
        .map { round(it.average()) }
```

Extension Functions

```
private val currentHardwareOffsets =
    liveSession.hwTimeStamps
        .timestamp(TimeUnit.MILLISECONDS, scheduler)
        .map { it.time() - it.value().timestampMilliseconds }
        .runningAverage(5)
```

INSTIL

Extension Functions

```
<TextView
    android:id="@+id/title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAlignment="center"/>
```

```
TextView titleView = activity.findViewById<TextView>(R.id.title)
titleView.text = "Hello World!"
```

```
title.text = "Hello World!"
```

Kotlin Android Extensions provides this
It most cases it will cache to avoid multiple calls to `findViewById`

Null Safety

Java

```java
private static void processUserPolicy(User user) {
    if (user != null && user.getPolicy() != null) {
        Policy policy = user.getPolicy();
        if (policy.isComplete()) {
            policyProcessor.process(policy);
        }
    }
}
```

Kotlin

```kotlin
private fun processUserPolicy(user: User) {
    val policy = user.policy
    if (policy.isComplete) {
        policyProcessor.process(policy)
    }
}
```

Words from the Wise

“What's a null pointer exception?

Aaron Finn, Associate Software Engineer, Instil

Null Safety (Plus Friends)

```kotlin
@BindingAdapter("onEnterKeyPressed")
fun onEnterKeyPressed(editText: EditText, runnable: Runnable?) {
    editText.setOnKeyListener { view, keyCode, _ ->
        if (keyCode == KeyEvent.KEYCODE_ENTER) {
            view.hideKeyboard()
            view.clearFocus()
            runnable?.run()
            true
        } else {
            false
        }
    }
}
```

Safe Call Operator

```xml
<androidx.appcompat.widget.AppCompatEditText
    app:onEnterKeyPressed="@{() -> vm.activate()}"
```

INSTIL

Null Safety (Plus Friends)

Top Level Function

```
@BindingAdapter("onEnterKeyPressed")
fun onEnterKeyPressed(editText: EditText, runnable: Runnable?) {
    editText.setOnKeyListener { view, keyCode, _ ->
        if (keyCode == KeyEvent.KEYCODE_ENTER) {
            view.hideKeyboard()
            view.clearFocus()
            runnable?.run()
            true
        } else {
            false
        }
    }
}
```

```
<androidx.appcompat.widget.AppCompatEditText
    app:onEnterKeyPressed="@{() -> vm.activate()}"
```

Null Safety (Plus Friends)

```kotlin
@BindingAdapter("onEnterKeyPressed")
fun onEnterKeyPressed(editText: EditText, runnable: Runnable?) {
    editText.setOnKeyListener { view, keyCode, _ ->
        if (keyCode == KeyEvent.KEYCODE_ENTER) {
            view.hideKeyboard()
            view.clearFocus()
            runnable?.run()
            true
        } else {
            false
        }
    }
}
```

Extension Method

```xml
<androidx.appcompat.widget.AppCompatEditText
    app:onEnterKeyPressed="@{() -> vm.activate()}"
```

INSTIL

Null Safety (Plus Friends)

```kotlin
@BindingAdapter("onEnterKeyPressed")
fun onEnterKeyPressed(editText: EditText, runnable: Runnable?) {
    editText.setOnKeyListener { view, keyCode, _ ->
        if (keyCode == KeyEvent.KEYCODE_ENTER) {
            view.hideKeyboard()
            view.clearFocus()
            runnable?.run()
            true
        } else {
            false
        }
    }
}
```

If as an expression

```xml
<androidx.appcompat.widget.AppCompatEditText
    app:onEnterKeyPressed="@{() -> vm.activate()}"
```

Null Safety (Plus Friends)

```
@BindingAdapter("onEnterKeyPressed")
fun onEnterKeyPressed(editText: EditText, runnable: Runnable?) {
    editText.setOnKeyListener { view, keyCode, _ ->
        if (keyCode == KeyEvent.KEYCODE_ENTER) {
            view.hideKeyboard()
            view.clearFocus()
            runnable?.run()
            true
        } else {
            false
        }
    }
}
```

Top Level Function

Extension Method

Safe Call Operator

If as an expression

```
<androidx.appcompat.widget.AppCompatEditText
    app:onEnterKeyPressed="@{() -> vm.activate()}"
```

Standard Utility Functions

```
leftSwipe.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}

rightSwipe.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}
```

Standard Utility Functions

```
leftSwipe?.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}

rightSwipe?.let {
    it.translationX += delta
    it.translationX = limitInDistance(it.translationX)
}
```

Null Safety in Reality

Null safety and immutability are great

Some scenarios are a little messier

- Reading in untyped JSON
- DI injected values

Kotlin has lots of ways of help

- Good tooling
- lateinit
- lazy

One is your locker. The other is a garbage dump in the Philippines
This one's the dump
They're both your locker

Null Safety in Reality

```
val saveButtonText: String by lazy {
    resourceService.getStringResource(R.string.save)
}
```

```
@Inject lateinit var loginViewModel: LoginViewModel
@Inject lateinit var approvalViewModel: ApprovalViewModel
@Inject lateinit var transferViewModel: TransferViewModel
@Inject lateinit var resourceService: ResourceService
```

Null Safety in Reality

```
data class Reward(
    @JsonProperty("type")      val type: RewardType,
    @JsonProperty("points")    val points: Int,
    @JsonProperty("name")      val name: String,
    @JsonProperty("eligible") val eligible: Boolean,
)
```

Non-nullable vals can be used when working with JSON
Retrofit uses Jackson Object Mapper (which has a Kotlin specific module)

Introducing Coroutines

```
suspend fun doWork() {
    disableUI()
    val exchangeRates = readExchangeRate()
    val report = buildReport(exchangeRates)
    saveReport(report)
    enableUI()
}

fun doWork() {
    disableUI()
    val exchangeRates = readExchangeRate()
    val report = buildReport(exchangeRates)
    saveReport(report)
    enableUI()
}
```

INSTIL

Introducing Coroutines

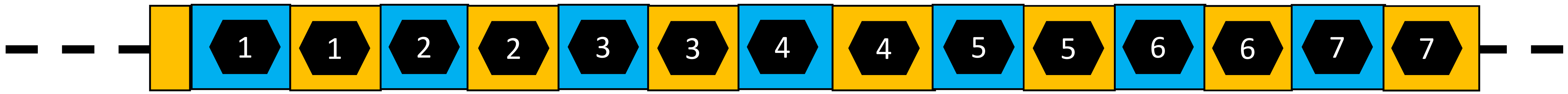
```
suspend fun doWork() {
        disableUI()
suspend val exchangeRates = readExchangeRate()
suspend val report = buildReport(exchangeRates)
suspend saveReport(report)
        enableUI()
return }
```

INSTIL

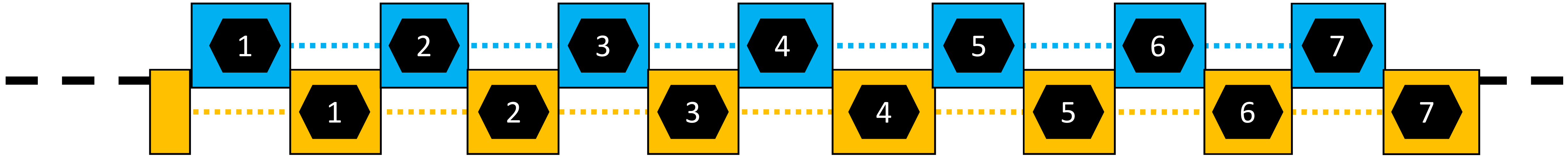
Introducing Coroutines

```kotlin
suspend fun doWorkWithThreadControl() {
    disableUI()
    withContext(Dispatchers.IO) {
        val exchangeRates = readExchangeRate()
        val report = buildReport(exchangeRates)
        saveReport(report)
    }
    enableUI()
}
```

INSTIL



INSTIL


1
2
3
4
5
6
7
1
2
3
4
5
6
7

Using Coroutines

Historically we had used RxJava extensively

- To bypass standard features like AsyncTask

With Kotlin, we’ve used RxKotlin heavily too

We started using coroutines when they were experimental

- We’ve found lots of code can be simplified by coroutines

We still use Rx when what we’re modelling is intuitively a stream

Using Coroutines

```
private suspend fun processUpdateRequest(update: Update,
                                         requestOptions: RequestOptions) {
    val customer = customerService.customerDetails(update.customerId)
    val transaction = transactionBuilder.build(requestOptions)
    transaction.customer = customer
    customerSaleService.updateTransactionWithCustomer(transaction)

    val updateCopy = update.createCopy()
    transactionDao.save(transaction)
    eventBus.requestComplete(transaction, update)
}
```

Using Coroutines

Realm threading restrictions were painful

Required specific threads for work

Coroutines made the threading easier

```
private fun fetchAndPersistSettings() = async(networkDispatcher) {
    val settings = fetchSettings()
    realmWrite { realm ->
        updateDao.overwrite(realm, settings)
    }
}
```

What Were The Pain Points?

We experimented with Kotlin Test but reverted to JUnit

- We were looking at Spek with Kotlin support but encountered issues

Some Java interop requires back ticking to avoid language clashes

```
@Test
fun shouldConvertBytesHeldAsPositiveToPositiveIntegers() {
    assertThat(1.toByte().toPositiveInt(), `is`(equalTo(1)))
    assertThat(12.toByte().toPositiveInt(), `is`(equalTo(12)))
    assertThat(123.toByte().toPositiveInt(), `is`(equalTo(123)))
    assertThat(127.toByte().toPositiveInt(), `is`(equalTo(127)))

}
```

```
doReturn("$20.00").`when`(monetaryFormatter).format(Monetary.TWENTY)
doReturn("$5.00").`when`(monetaryFormatter).format(Monetary.FIVE)
```

What Were The Pain Points?

Documentation in some areas are lacking

- When features are experimental or very new
- Requires trial and error
- Some best practices would be nice
- Only happy paths shown e.g. Kotlin Gradle DSL
    - Internally, some teams use and others have avoided

We saw some collisions in some Java class names

- But Kotlin type aliases came to rescue here

What Would We Do Differently?

Not too much “caught” us out but some things we would do differently

Some of this and the pain points are simply timing

- Better libraries and library support came later
- Language features stabilised e.g. coroutines

More and more libraries are writing Kotlin wrappers or Kotlin first interfaces

- Android, Realm, Apache Beam, Spring etc

Modern development requires so many dependencies – constant evolution

Frameworks We Used Or Would Use

Dagger	→	Koin
Retrofit	→	Rx style migrated to coroutines
Realm	→	Room
Glide	→	Coil (coroutine support)
Junit	→	Kotlin Test and AssertJ

Espresso Spoon, Awaitility

- These are just nicer with Kotlin

Kotlin Native

We write quite a few multi-platform apps using Xamarin

- This is a good, stable, cross platform solution for writing native apps

We are keeping an eye on Kotlin Native

- Still in beta but with a lot of potential

Could allow us to write more Kotlin (yeah!) making Android easier

- But leveraging the same code for iOS

One to watch

In Summary

Your existing skills remain relevant

The learning curve is gentle and straightforward

Interns and grads pick it up in no time

Developers with Java, C# & Swift experience adopt it very easily

The only pain is when go back to Java ☺

Words from the Wise

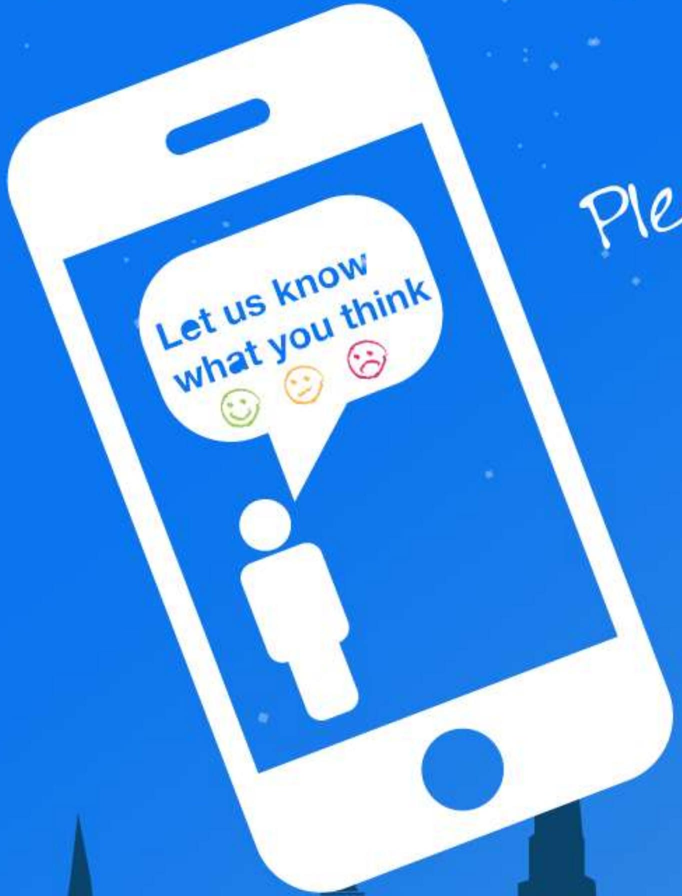
“As a services business we need to focus primarily on our customers needs. Thankfully, it's not just our developers that have grown to love the language. Our customers are in turn getting more maintainable software to market with lower lead times.”

Matt McComb, Director of Business Development, Instil

Questions?

goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20



Please
Remember to rate this session

Thank you!

Follow us @gotocph