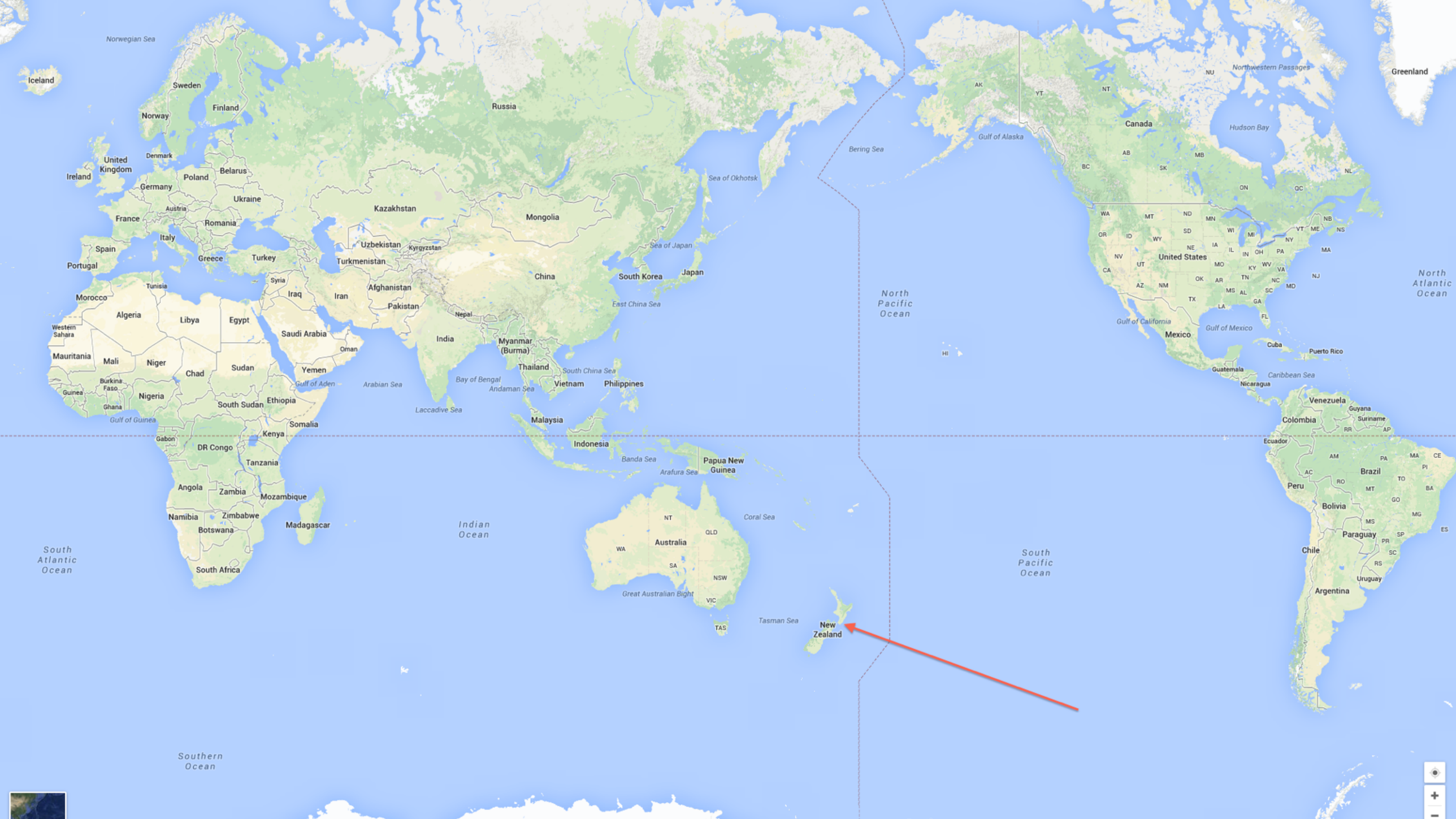
KAI KOENIG (@AGENTK)

LITTLE HELPERS FOR ANDROID DEVELOPMENT WITH KOTLIN


Norwegian Sea
Iceland
Sweden
Finland
Norway
Russia
United Kingdom
Ireland
Denmark
Germany
Poland
Belarus
Ukraine
France
Austria
Romania
Italy
Spain
Portugal
Greece
Turkey
Kazakhstan
Uzbekistan
Kyrgyzstan
Turkmenistan
Mongolia
China
South Korea
Japan
Sea of Okhotsk
Sea of Japan
East China Sea
Bering Sea
Gulf of Alaska
Canada
Hudson Bay
Northwestern Passages
Greenland
United States
Mexico
Gulf of California
Gulf of Mexico
Cuba
Puerto Rico
Caribbean Sea
Guatemala
Nicaragua
Venezuela
Guyana
Suriname
Colombia
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Uruguay
Argentina
North Pacific Ocean
North Atlantic Ocean
South Pacific Ocean
South Atlantic Ocean
Indian Ocean
Southern Ocean
Morocco
Western Sahara
Algeria
Tunisia
Libya
Egypt
Mauritania
Mali
Niger
Chad
Sudan
Burkina Faso
Guinea
Nigeria
Ghana
Gulf of Guinea
Gabon
DR Congo
Angola
Zambia
Mozambique
Namibia
Zimbabwe
Botswana
South Africa
Madagascar
Tanzania
Kenya
Somalia
Ethiopia
South Sudan
Yemen
Oman
Saudi Arabia
Gulf of Aden
Arabian Sea
Laccadive Sea
Iraq
Iran
Syria
Afghanistan
Pakistan
Nepal
India
Myanmar (Burma)
Thailand
Vietnam
Bay of Bengal
Andaman Sea
South China Sea
Philippines
Malaysia
Indonesia
Banda Sea
Arafura Sea
Papua New Guinea
Coral Sea
Australia
NT
QLD
WA
SA
NSW
VIC
TAS
Great Australian Bight
Tasman Sea
New Zealand

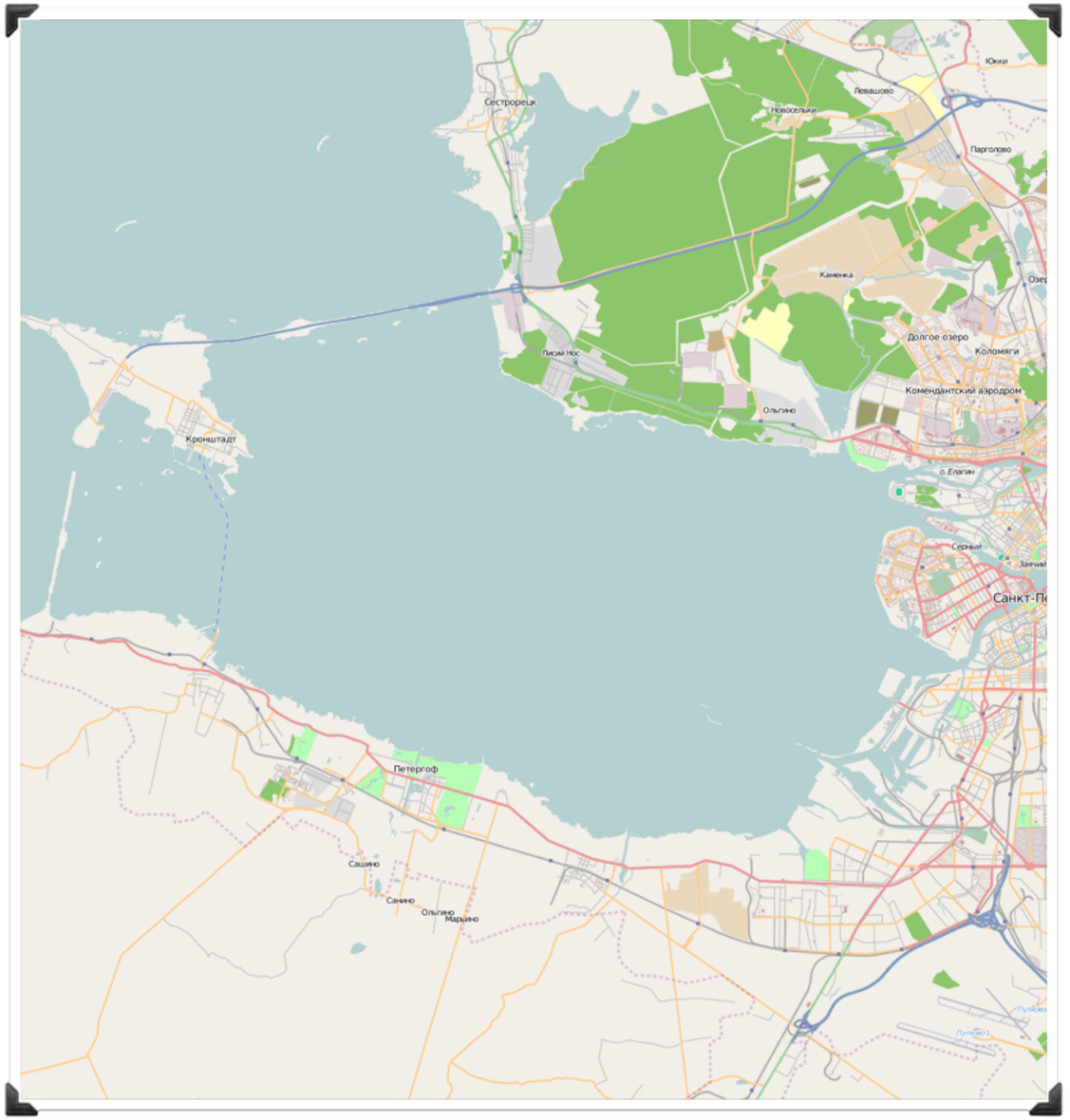
AGENDA

- What is Kotlin?
- Common idioms and language concepts
- Kotlin and Android
- Anko
- Other libraries and tools for Kotlin and Android
- Final thoughts

WHAT IS KOTLIN?

SOME FUNDAMENTALS

- Statically typed programming language for the JVM and Android as well as the browser
- Started as internal language “Project Kotlin” at Jetbrains in 2010
- Now: Open-Source, Apache License - Kotlin 1.0 released in Feb 2016
- Kotlin SDK plus tool support for IntelliJ, Android Studio, Eclipse
- Named after an island in the Gulf of Finland



MOTIVATION FOR KOTLIN

- The Java platform is awesome, but it has its issues:
  - Sometimes tied to backwards/legacy compatibility
  - Can be a very verbose language and produce bloated code
  - Type system has various flaws
- Kotlin aims to fix a lot of those issues, in particular when one has to use Java 6 or 7 (if we're lucky…) and can't use all the new, shiny features from Java 8 and soon Java 9 and 10.

HOW DOES A SIMPLE CONVERSION LOOK LIKE?

```java
public String listConvert(Collection<Integer> collection) {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    Iterator<Integer> iterator = collection.iterator();
    while (iterator.hasNext()) {
        Integer element = iterator.next();
        sb.append(element);
        if (iterator.hasNext()) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}
```

```kotlin
fun listConvert(collection: Collection<Int>): String {
    val sb = StringBuilder()
    sb.append("{")
    val iterator = collection.iterator()
    while (iterator.hasNext()) {
        val element = iterator.next()
        sb.append(element)
        if (iterator.hasNext()) {
            sb.append(", ")
        }
    }
    sb.append("}")
    return sb.toString()
}
```

```kotlin
fun listConvertKt(collection: Collection<Int>): String {
    return collection.joinToString(prefix = "{",postfix = "}")
}
```

https://www.flickr.com/photos/geraldford/6976818221/

COMMON IDIOMS & LANGUAGE PATTERNS

OVERVIEW

- **Immutability**
- **String templates & Enum classes**
- **Null safety**
- **Properties and Fields**
- **Type inference and casts**
- **Data classes**
- Syntactic sugar (loops, ranges etc)
- **Extension functions**
- Lambdas
- Collection API
- Type-safe builders
- **Java-Kotlin-Interop**

IMMUTABILITY

- Built-in support for mutable and immutable variables, properties and fields
- Keywords var and val
  - val - immutable (recommended)
  - var - mutable
- Similar concept applies for class properties, val creates getters, var creates getters and setters (more later)

```
val a: Int = 1
val b = 1

val c: Int
c = 1

var x = 23
x += 1
```

STRING TEMPLATES & ENUM CLASSES

- Kotlin Strings can contain template expressions
- Start with a $ character and
  - can contain simple references:
    $s
  - complex expressions in curly braces:
    ${s.length}
- Kotlin has a dedicated enum class, very similar to Java

```
val s = "abc"
val str = "$s.length is ${s.length}"

enum class Locale(val hello: String) {
  DE_DE("Hallo"), EN_NZ("Hello"), MI_NZ("Kia Ora")
}

class Customer(val firstName:String,
               val lastName:String,
               val locale: Locale = Locale.DE_DE) {
  fun sayHello() = println("${locale.hello},
                           $firstName $lastName")
}

fun main(args : Array<String>) {
  val myCustomer = Customer("Sandra",
                            "Musterfrau",
                            Locale.MI_NZ)
  myCustomer.sayHello()
}
```

NULL SAFETY

- Motivation: A better way to deal with NPEs
- Kotlin differentiates nullable types from non-nullable types by adding a ? to the type:
  - String: no nullable
  - String?: nullable
- Handle manually or use Safe Call operator ?. or use the !! operator to allow/trigger a NPE.

```
// Won't compile
var lastName: String = null

// Will compile
var lastNameNullable: String? = null

// Will also not compile
println(lastNameNullable.length)

// Option 1 (-1)
println(if (lastNameNullable != null)
        lastNameNullable.length else -1)

// Option 2 (null)
println(lastNameNullable?.length)

// Option 3 (NPE)
println(lastNameNullable!!.length)
```

PROPERTIES AND FIELDS

- Kotlin classes have mutable or immutable properties
- An automated backing field can be provided by the compiler (if deemed necessary)
- Default getter/setters for properties, can be customised
- lateinit modifier to deal with non-nullable properties that can’t be initialised in the constructor

```
var counter = 0
    set(value) {
        if (value >= 0)
            field = value
    }

public class MyTest {
    lateinit var subject: TestSubject

    @SetUp fun setup() {
        subject = TestSubject()
    }

    @Test fun test() {
        subject.method()
    }
}
```

TYPE INFERENCE AND CASTS (I)

- When possible, Kotlin will infer the type of variables
- Explicit conversions, type widening and inference
  - Smaller types are not subtypes of bigger types, no implicit conversion
  - Types are often inferred from the context

```
val b: Byte = 1
// This won't work
val i: Int = b
// This will
val i: Int = b.toInt()
```

```
val l = 1L + 3
```

TYPE INFERENCE AND CASTS (II)

- is or !is checks if an object adheres to a certain type
- Smart cast: Compiler tracks is-expressions for immutable values
  - works for val local variables and private, internal or in module performed casts
  - works for var local variables if the variable hasn't been modified between check and usage, never for var properties

```
fun whatIs(x: Any) {
    when (x) {
        is Int -> println(x + 42)
        is String -> println(x.length)
        is IntArray -> println(x.sum())
    }
}

whatIs(4) // 46
whatIs("4") // 1
whatIs(intArrayOf(1,2,3,4,5)) // 15
```

DATA CLASSES

- The POJOs or Beans of other languages...
- Data classes implicitly create:
  - getters/setters (the latter if a property is var)
  - equals(), hashCode(), toString(), copy() - can be overwritten by custom implementations
  - copy() has default parameters and can be used to alter a copy
  - parameterless constructors need default parameters specified

```
data class ChromeEncryptedPayload(
  val encryptedPayload: String,
  val encryptionHeader: String,
  val cryptoKeyHeader: String)
```

EXTENSION FUNCTIONS

- Allow adding new functionality to a class without inheritance or Decorators
- Kotlin allows extension functions and extension properties
  - Resolved statically, do not actually modify the class (excellent example why this has to be the case on https://kotlinlang.org/docs/reference/extensions.html)

```
fun Int.sum(otherInt: Int): Int = this +
otherInt

3.sum(7)


fun Activity.toast(message: CharSequence,
                   duration: Int =
                    TOAST.LENGTH_SHORT) {
    Toast.makeText(this, message,
                   duration).show()
}

// In onCreate of an Activity
override fun onCreate(...) {
    ...
    toast("Hi there")
    ...
}
```

JAVA-KOTLIN-INTEROP

- Java and Kotlin are fully interoperable from an integration point of view
  - Your Java code can call and use Kotlin code
  - Your Kotlin code can call and use Java code
- The latter is in particular useful because it means you can continue to use pretty much any existing Android/Java library
- Check out Hendrik Kokocinski's sample Kotlin app that uses all kinds of well known Android/Java libs: https://github.com/blob0815/kotlin-android-sample

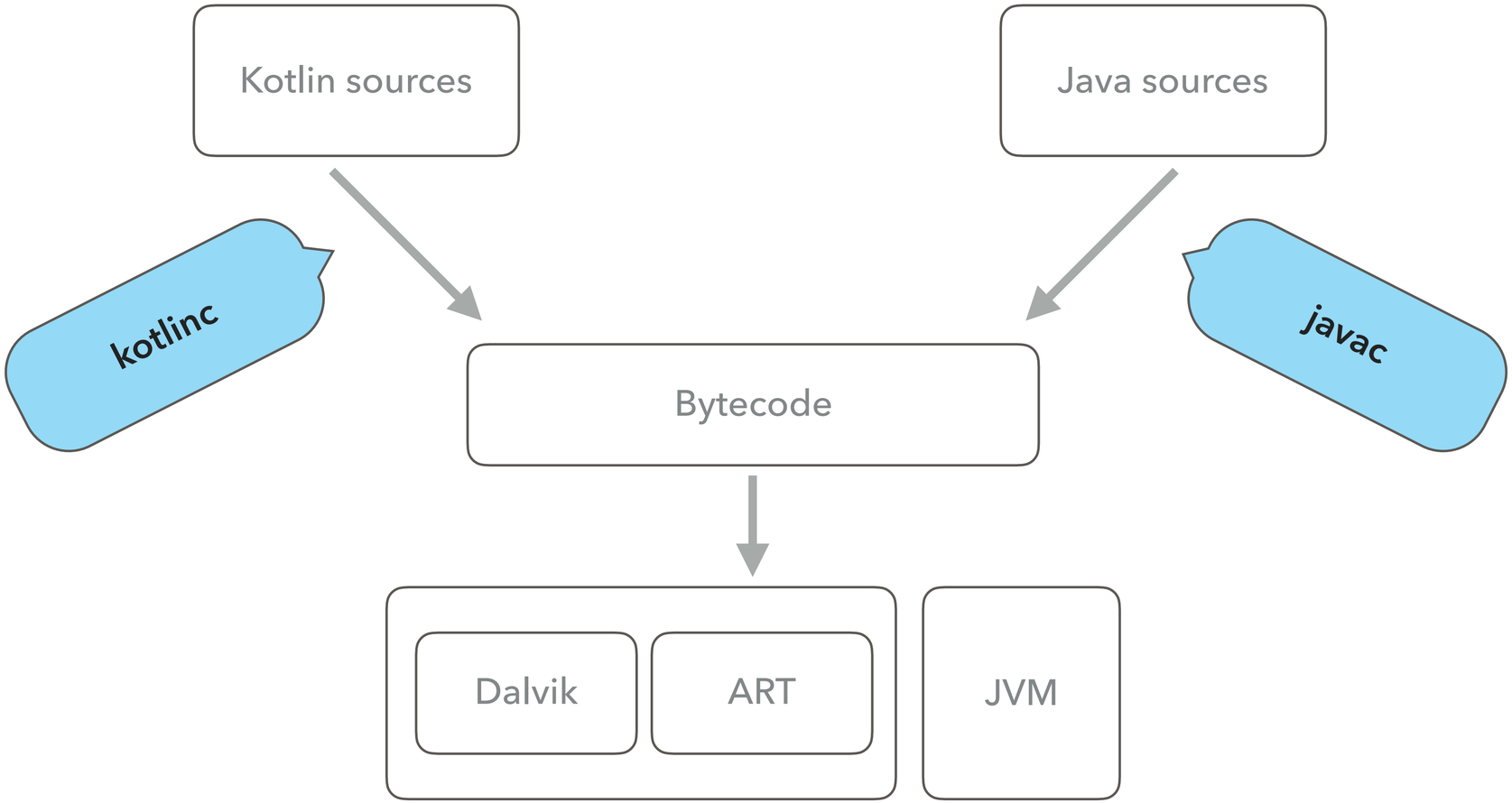
OVERVIEW

- **Immutability**
- **String templates & Enum classes**
- **Null safety**
- **Properties and Fields**
- **Type inference and casts**
- **Data classes**
- Syntactic sugar (loops, ranges etc)
- **Extension functions**
- Lambdas
- Collection API
- Type-safe builders
- **Java-Kotlin-Interop**

https://realm.io/assets/img/news/tutorial-kotlin-anko-cover.png

KOTLIN & ANDROID

TOOLCHAIN AND FLOW



PROJECT SETUP

- Use Android Studio 1.5.x/2.x or IntelliJ 15/2016
- Install Kotlin plugin (comes with “Jetbrains plugins” nowadays)
  - Gradle dependencies project-level:
    - classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.0.2"
  - Gradle dependencies module-level:
    - compile 'org.jetbrains.kotlin:kotlin-stdlib:1.0.2'
    - apply plugin: 'kotlin-android'
    - main.java.srcDirs += **'src/main/kotlin'**

KOTLIN EXTENSIONS FOR ANDROID (I)

- Provides of a set of synthetic properties that bind views to those properties
- Alternative to Butter Knife-style bindings, no need for additional runtime library (Kotlin Extensions for Android are a Kotlin compiler plugin)
  - **import** kotlinx.android.synthetic.main.<layout>.*
  - **import** kotlinx.android.synthetic.main.<layout>.view.*
  - usage: <componentid>.doSomething()
- Integrates nicely with build flavors, too

SYNTHETIC PROPERTIES

```
package ventegocreative.co.nz.kotlindemo

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        helloworld.text = "Hey, I'm dynamically set"
    }
}
```

KOTLIN EXTENSIONS FOR ANDROID (II)

- Used to be part of a separate plugin (https://plugins.jetbrains.com/plugin?pluginId=7717)
  - obsolete since December 2015
  - functionality rolled into the main Kotlin plugin
  - still needs a very particular setup (not properly documented but in this place: https://youtrack.jetbrains.com/issue/KT-10336)

    TLDR: move dependencies in app module(s), not top-level Gradle file.

KOTLIN EXTENSIONS FOR ANDROID (III)

- The current assumption is that Jetbrains plan to add more to the Kotlin Extensions for Android, but we don't really know for sure at this stage.
- Current feature of view binding:
  - Fresh take on view binding
  - Very simple to setup and use
  - Overall less powerful than Butter Knife, Kotter Knife

ANKO

A DSL FOR LAYOUTS

- Developed and maintained by Jetbrains, under Apache 2.0 license
- The most important element of Anko is the Layout DSL
  - Idea: Replace XML layout definitions by Kotlin code - without having to build the layout in a truly programmatic sense
  - Modular - as we're talking about UI/Layout, it's very important to select the right library for your minSDKVersion
  - Extensible - you can add your own DSL elements for custom UI controls

LAYOUT XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <EditText
        android:layout_width="match_parent"
        android:gravity="center"
        android:text="@string/empty_todos_message"
        android:layout_weight="7"
        android:layout_height="wrap_content" />
    <Button
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:text="Say Hello"
        android:layout_height="0dp" />
</LinearLayout>
```

PROGRAMMATIC LAYOUT IN KOTLIN

```
val act = this
val layout = LinearLayout(act)
layout.orientation = LinearLayout.VERTICAL
val name = EditText(act)
val button = Button(act)
button.text = "Say Hello"
button.setOnClickListener {
    Toast.makeText(act, "Hello, ${name.text}!", Toast.LENGTH_SHORT).show()
}
layout.addView(name)
layout.addView(button)
```

ANKO DSL

```
verticalLayout {
    val name = editText()
    button("Say Hello") {
        onClick { toast("Hello, ${name.text}!") }
    }
}
```

BUT THERE'S MORE (ANKO "ADVANCED TOPICS")

- Intent Wrappers for various purposes: e.g. sendSMS(number, [text])
- Service shortcuts
- Configuration qualifiers: configuration(screenSize = ScreenSize.LARGE, orientation = Orientation.LANDSCAPE) { … }
- Asynchronous tasks
- SQLLite
  - Removes all the tragic cursor handling and lot of the try/catch blocks necessary

https://www.flickr.com/photos/sillygwailo/5990089210/

LIBRARIES AND TOOLS

OVERVIEW

- Kotter Knife
- Butter Knife
- KAndroid
- Kovenant
- Quickies: Fuel, Injekt, Spek, Kotson

KOTTER KNIFE

- We don’t like findByViewId(…) - read: https://ragunathjawahar.wordpress.com/2015/03/23/kotlin-findviewbyid-dead-as-dinosaurs
- Kotter Knife provides view binding in a similar way to Butter Knife for Android/Java
- Why Kotter Knife (runtime library) over Kotlin Android Extensions (compiler plugin)?
  - Porting code from Java/Butter Knife to Kotlin
  - Features like listener binding and resources binding that don’t exist in KAE.

BUTTER KNIFE

- Not much to say about it, probably one of the most famous libraries in the Android world.
  - Butter Knife was difficult to use with Kotlin in the beginning
  - Since Kotlin 1.0 RC you can in fact just use Butter Knife in your Kotlin code
  - Sample code for Butter Knife/Kotlin in the official Jetbrains Kotlin Examples repo (https://github.com/JetBrains/kotlin-examples/tree/master/gradle/android-butterknife)

KOTTER KNIFE VS BUTTER KNIFE VS KAE (I)

```
findViewById(R.id.send).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d("MainActivity", "onClick: send")
    }
});
```

```
@OnClick(R.id.send)
void clickSend(View v) {
    Log.d("MainActivity", "onClick: send")
}
```

KOTTER KNIFE VS BUTTER KNIFE VS KAE (II)

```
findViewById(R.id.send).setOnClickListener { view -> Log.d("MainActivity",
"onClick: send") };
```

```
val btnSend: Button by bindView(R.id.send)
btnSend.setOnClickListener({ view -> Log.d("MainActivity", "onClick: send") })
```

```
import kotlinx.android.synthetic.activity_main.*
btn_send.setOnClickListener({ view -> Log.d("MainActivity", "onClick: send") })
```

KANDROID

- KAndroid is an extension library for Kotlin/Android
- Contains a variety of absolutely distinct and unconnected functionality
- Common theme: let’s get rid of boilerplate code
- Apache 2.0 license

KANDROID FEATURES

- View binding (again)
- TextWatcher
- SeekBar Extension
- SearchView Extension
- Shortcuts to system services
- Logging
- Dealing with Intents
- SDK Version API (from/to)
- Thread management

KANDROID EXAMPLES

```
runDelayed(1000) {
    // delayed execution
}

runDelayedOnUiThread(5000) {
    // delayed UI update
}

toApi(16, inclusive = true) {
    // handle devices running older APIs
}
```

KOVENANT

- In a nutshell: Promises for Kotlin
- Very modular built, you can essentially pick and choose the artifacts of Kovenant that you'd like to use - Kovenant is not an Android-specific library
  - Good staring set for Android: core, android, combine, jvm, functional
- MIT license

KOVENANT FEATURES

- Core, foundations of a Promise framework
  - Tasks & Callbacks
  - Chaining (Then, ThenApply)
  - Lazy Promises
  - Cancelling and Voiding
- Combine: combines 2-20 promises
- Functional: adds map, bind and apply to support more advanced HOF constructs in Kovenant
- JVM: Executors and Throttles (thread pools)
- Android: UI callbacks and interacting with UI Thread

KOVENANT EXAMPLES (I)

```
task {
    // some long-running thing
} success {
    println("result: $it")
} fail {
    println("problem: ${it.message}")
} always {
    // this will always happen
}
```

KOVENANT EXAMPLES (II)

```
promiseOnUi {
    // do some UI preparation
} then {
    // the actual work
} successUi {
    // update UI
}
```

OVERVIEW

- Kotter Knife
- Butter Knife
- KAndroid
- Kovenant
- **Quickies: Fuel, Injekt, Spek, Kotson**

https://www.flickr.com/photos/brickset/16099265973/

FINAL THOUGHTS

PERFORMANCE

- Runtime is pretty much on-par with Java
- Pre Kotlin 1.0.2: Build process is slower than a comparable app in Java - mainly due to how the Kotlin compiler works (no partial builds/compilation)
- Kotlin libraries do add to the size of the application as well as to the method count
  - Kotlin runtime + stdlib are similar in method count to support-v4 or play-services-base and add significantly less than Scala or Groovy

LANGUAGE AND MATURITY

- Kotlin 1.0 was a big step for the language
- Surprisingly mature for a 1.0 release (but 5+ years in the making)
- Full of great concepts and idioms
- Refreshing language that makes both Android and JVM development significantly more pleasant and fun
- Issue: Tooling around static analysis is non-existent at the moment (some basic listing for Android is available since 1.0.2)
- Ready for prime-time yet?

WHAT DID WE LEARN?

- What is Kotlin?
- Common idioms and language concepts
- Kotlin and Android
- Anko
- Other libraries and tools for Kotlin and Android

RESOURCES

- Kotlin: http://kotlinlang.org
- Anko: https://github.com/Kotlin/anko
- Kotter Knife: https://github.com/JakeWharton/kotterknife
- KAndroid: https://github.com/pawegio/KAndroid
- Kovenant: https://github.com/mplatvoet/kovenant
- Fuel: https://github.com/kittinunf/Fuel
- Injekt: https://github.com/kohesive/injekt
- Spek: http://jetbrains.github.io/spek/
- Kotson: https://github.com/SalomonBrys/Kotson

GET IN TOUCH

Kai Koenig

Email: kai@ventego-creative.co.nz

Twitter: @AgentK