AMERICAS APPFORUM 2019 | ELEVATING ENTERPRISE INTELLIGENCE

ZEBRA CAPTURE YOUR EDGE

Developing Kotlin applications for Zebra Android devices

Darryn Campbell
SW Architect, Zebra Technologies
@darryncampbell
October 1st / 2nd 2019

Agenda

- What is Kotlin?
  - A brief overview of Kotlin as a programming language
- Kotlin Syntax Crash Course
  - A crash course in Kotlin Syntax and the differences from Java
- Why Kotlin?
  - Why you should be using Kotlin to develop your applications
- Why not Kotlin?
  - Does Kotlin have any drawbacks?
- Kotlin and Zebra
  - Some examples of how you can use Kotlin in your Zebra application

ZEBRA TECHNOLOGIES

What is Kotlin?

A brief overview of the Kotlin Language

What is Kotlin?

A brief overview of the Kotlin Language

- Statically typed programming language for the Java Virtual Machine (JVM)
- Used anywhere Java is used today (which is virtually anywhere)
- Developed by JetBrains (developers of IntelliJ IDEA)
- Officially supported by Google for Android
- Focused on interoperability, safety, clarity, and tooling support

Statically Typed

- Kotlin, like Java, is a statically typed language
  - Type Checking is performed at compile time, not run time

- Statically Typed languages:
  - Provide better code completion
  - Catches errors earlier
  - Reduce errors in production code

Built for JVM

- JVM (Java Virtual Machine) Runs java programs or other languages that compile to Java bytecode
- Kotlin compiles to Java bytecode and will run on any device that supports the JVM

Developed by JetBrains, Supported by Google

- Developed by JetBrains
  - IntelliJ
  - WebStorm & PhpStorm
- Officially supported by Google since 2017
  - Google says that Kotlin is “a brilliantly designed, mature language that we believe will make Android development faster and more fun.”
  - Joins Java & C++ as a supported language

Interoperability, Safety & Clarity

- Interoperability
  - Works anywhere Java does and can be used in conjunction with Java
- Safety
  - Reduces errors by introducing, among other features, null safety
- Clarity
  - Much less verbose than Java syntax

ZEBRA TECHNOLOGIES

Kotlin Syntax Crash Course

A crash course in Kotlin Syntax

Kotlin Syntax

Variables

- Variables are declared in Kotlin using var and val keywords
- Val variables are immutable (read-only)
- Var variables are mutable (can be changed)

```
// Immutable Val
private val valExample = "this is an immutable variable"

// Mutable Var
private var varExample = "this is a mutable variable"

fun test() {

    valExample = "This is an error"

    varExample = "This is fine"

}
```

```
// Immutable Val
private val valExample = "this is an immutable variable"

// Mutable Var
private var varExample = "this is a mutable variable"

fun test() {

    valExample = "This is an error"

    varExample = "This is fine"

}
```

Kotlin Syntax

Functions

- Functions are defined with the ‘fun’ key word
- Return type is after function parameters
- Parameter declarations are reversed

```java
// Java
public int sum(int a, int b) {
  return 0;
}
```

```kotlin
fun sum(a: Int, b: Int): Int {
    return 0
}
```

Kotlin Syntax

Funct

```java
// Java
public int sum(int a, int b) {
    return 0;
}
```

- Func
- Retu

- Parameter declarations are reversed

```kotlin
fun sum(a: Int, b: Int): Int {
    return 0
}
```

Kotlin Syntax

Type Inference

- Variables don’t specify a type
- Kotlin can infer the variables type from the initializer expression
- Type declaration is mandatory if not initializing variable at time of declaration

```
// Inferred as a String
private val inferredExample = "A String"

// Declared as a String
private val declaredString: String = "A String"
```

Kotlin Syntax

Type Inference

```
// Inferred as a String
private val inferredExample = "A String"

// Declared as a String
private val declaredString: String = "A String"
```

- V
- K
  i
- T
  variable at time of declaration

Kotlin Syntax

Data Types

- Kotlin, like other languages, has predefined types like Int, Double Boolean, Char etc…
- In Kotlin, everything (even the basic types like Int and Boolean) is an object. More specifically, everything behaves like an Object.
- This is contrary to languages like Java that has separate primitive types like int, double etc, and their corresponding wrapper types like Integer, Double etc.

Kotlin Syntax

Data Types - Numbers

- Numeric types in Kotlin are similar to Java. They can be categorized into integer and floating-point types.

```kotlin
// Kotlin Numeric Types Examples
val myShort: Short = 125
val myByte: Byte = 10
val myDouble = 325.49
val myInt = 1000

// The suffix 'L' is used to specify a long value
val myLong = 1000L

// The suffix 'f' or 'F' represents a Float
val myFloat = 126.78f
```

Ko

Da

- I
  t

```kotlin
// Kotlin Numeric Types Examples
val myShort: Short = 125
val myByte: Byte = 10
val myDouble = 325.49
val myInt = 1000

// The suffix 'L' is used to specify a long value
val myLong = 1000L

// The suffix 'f' or 'F' represents a Float
val myFloat = 126.78f
```

Kotlin Syntax

Data Types - Characters

- Characters are represented using the type Char
- Unlike Java, Char types cannot be treated as numbers.
- Char types are declared using single quotes

```
// Characters
val letterChar = 'A'
val digitChar = '9'
```

Kotlin Syntax

Data Types - Characters

- Characters are represented using the type Char
- Unlike Java, Char types cannot be treated as numbers
- Char types are

```
// Characters
val letterChar = 'A'
val digitChar = '9'
```

Kotlin Syntax

Data Types - Strings

- Strings are represented using the String class.

- Strings are immutable, that means you cannot modify a String by changing some of its elements.
  - Strings can be re-assigned but cannot be modified.

- You can access the character at an index in a String using str[index].

```
var name = "John"
var firstCharInName = name[0]  // 'J'
var lastCharInName = name[name.length - 1]  // 'n'
```

Kotlin Syntax

Data Types - Strings

- Strings are represented using the String class.

- Strings are immutable, that means you cannot modify a String by changing some of its elements.
  - Strings can be re-assigned but cannot be modified.

- You can access the character at an index in a String using str[index].

```
var name = "John"
var firstCharInName = name[0]   // 'J'
var lastCharInName = name[name.length - 1]   // 'n'
```

Kotlin Syntax

Data Types - Escaped / Raw Strings

- Strings declared in double quotes can have escaped characters like ‘\n’ (new line), ‘\t’ (tab) etc

- In Kotlin, you also have an option to declare raw strings. These Strings have no escaping and can span multiple lines

```
// Escaped String
var myEscapedString = "Hello Reader,\nWelcome to my Blog"
// Raw String
var myMultilineRawString =
        """
The Quick Brown Fox
Jumped Over a Lazy Dog.
        """
```

Kotlin Syntax

Data Types - Escaped / Raw Strings

- Strings declared in double quotes can have escaped characters like ‘\n’ (new line), ‘\t’ (tab) etc
- In Kotlin, you also have an option to declare raw strings. These Strings have no escaping and can

```
// Escaped String
var myEscapedString = "Hello Reader,\nWelcome to my Blog"
// Raw String
var myMultilineRawString =
        """
The Quick Brown Fox
Jumped Over a Lazy Dog.
        """
```

Kotlin Syntax

Data Types - Arrays

- Arrays in Kotlin are represented using the Array class
- You can create an array in Kotlin either using the library function arrayOf() or using the Array() constructor
- Note that you can also pass values of mixed types to the arrayOf() function, and it will still work

```
var numbers = arrayOf(1, 2, 3, 4, 5)
var animals = arrayOf("Cat", "Dog", "Lion", "Tiger")
var mixedArray = arrayOf(1, true, 3, "Hello", 'A')
```

Kotlin Syntax

Data Types - Arrays

- Arrays in Kotlin are represented using the Array class
- You can create an array in Kotlin either using the library function arrayOf() or using the Array() constructor
- Note that you can also pass values of mixed types to the arrayOf() function, and it will still work

```
var numbers = arrayOf(1, 2, 3, 4, 5)
var animals = arrayOf("Cat", "Dog", "Lion", "Tiger")
var mixedArray = arrayOf(1, true, 3, "Hello", 'A')
```

Kotlin Syntax

Data Types - Type Conversion

- Unlike Java, Kotlin doesn’t support implicit conversion from smaller types to larger types.

- Every number type contains helper functions that can be used to explicitly convert one type to another.

```
val myInt = 100
val myLong: Long = myInt // Compiler Error
val myLong2 = myInt.toLong() // Converted Successfully
```

Kotlin Syntax

Data Types - Type Conversion

- Unlike Java, Kotlin doesn’t support implicit conversion from smaller types to larger types.

- Every number type contains helper functions that can be used to explicitly convert one type to another.

```
val myInt = 100
val myLong: Long = myInt // Compiler Error
val myLong2 = myInt.toLong() // Converted Successfully
```

Why Kotlin?

Why should you develop applications in Kotlin?

Why Kotlin?

Overview

- Interoperability
- Clarity
- Exception Handling
- Null Safety
- Smart Casts

Interoperability

Completely interoperable with Java

- Kotlin is completely interoperable with Java
  - Compiles to Java bytecode to run on JVM
- Kotlin can be used side-by-side with Java
  - A project can contain both Kotlin & Java code
- Automatic Java to Kotlin Conversion in Android Studio
  - Ctrl + Alt + Shift + K

Clarity

Much more concise language than Java

- Much more succinct than Java
  - Reduces number of lines in code by 40% on average https://kotlinlang.org/docs/reference/faq.html
- No “New” keyword required:
- In-line String Templates
- Data Classes (POJO’s) can be created in a single line

Java POJO Kotlin

```java
class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // toString…

    // hashCode…

    // equals…

    // copy…
}
```

```kotlin
data class Person(val name: String)
```

Clarity

Single Line Data Class

Exception Handling

How Java handles Exceptions

- Two types of exceptions in Java
  - Checked
    - Explicit handling of exception using try / catch or throws
  - Unchecked
    - Exception produced by flaw in code (NullPointerException, ArithmeticException etc…)

- All exceptions are unchecked in Kotlin
  - We don’t have to catch exceptions if we don’t want to

- Exceptions can still be handled using try / catch just like Java, but it’s not enforced

Null Safety

Kotlin is inherently safer than Java

- Kotlin's type system is aimed at eliminating the danger of null references from code
- Distinguishes between references that can hold null (nullable references) and those that can’t (non-null references)
- Introduces new syntax for handling null values and avoiding NPE’s

Null Safety

Null vs Non-null refences

- Normal references cannot hold null value

- Adding ‘?’ after the data type allows a reference to be nullable

Null Safety

Null vs Non-null refences

- Null vs Non-Null Example:

```
// Non-nullable
private var nonNullExample : TestClass = TestClass()

// Nullable
private var nullExample : TestClass? = TestClass()


fun test() {
    // Compiler Error, Cannot be null
    nonNullExample = null

    // No error thrown
    nullExample = null
}
```

```
// Non-nullable
private var nonNullExample : TestClass = TestClass()

// Nullable
private var nullExample : TestClass? = TestClass()


fun test() {
    // Compiler Error, Cannot be null
    nonNullExample = null

    // No error thrown
    nullExample = null
}
```

Null Safety

Handling Null values using Safe Calls

- Nullable properties require a null check every time before utilization
- Using Safe Calls ?. Will perform a null check and prevents NPEs (Null Pointer Exception)
  - Calls the method if the property is not null, or returns null without throwing an NPE
- Safe Call can also be chained

Null Safety

Handling Null values using Safe Calls

- Safe Call Example:

```
// Nullable
private var nullExample : TestClass? = TestClass()


fun test() {
    // Set Value to Null
    nullExample = null

    // Will return null and not throw NPE
    nullExample?.someMethod()

    // Chained Safe Call will not throw NPE
    nullExample?.SubClass()?.BottomClass()
}
```

Nu

Ha

- S

```
// Nullable
private var nullExample : TestClass? = TestClass()


fun test() {
    // Set Value to Null
    nullExample = null

    // Will return null and not throw NPE
    nullExample?.someMethod()

    // Chained Safe Call will not throw NPE
    nullExample?.SubClass()?.BottomClass()
}
```

Null Safety

Handling Null values using the Elvis Operator ?:

- Like Safe Calls
- Allows returning of non-null values even if property is null
- Like Ternary (conditional) operator in Java

```
// Set Value to Null
nullExample = null

// result will be false
val result = nullExample?.someMethod() ?: false
```

Null Safety

Handling Null values using the Elvis Operator ?:

- Like Safe Calls

- Allows returning of non-null values even if property is null

```
// Set Value to Null
nullExample = null

// result will be false
val result = nullExample?.someMethod() ?: false
```

Null Safety

Handling Null values using the !! operator

- Used to avoid null values and throw NPE if value is null

```
// Set Value to Null
nullExample = null

// Will throw NPE
nullExample!!.someMethod()
```

Null Safety

Handling Null values using the !! operator

- Used to avoid null values and throw NPE if value is null

```
// Set Value to Null
nullExample = null

// Will throw NPE
nullExample!!.someMethod()
```

Smart Casts

Casting objects is simpler & more efficient in Kotlin than Java

- Java uses `instanceof` operator to check types
- Java requires explicit casting after `instanceof` returns true
- Kotlin uses `is` or `!is` operator to check types
- Kotlin compiler automatically casts variable after `is` returns true

Java

```java
private void javaTypeCast() {
    Object object = "Example String";

    if (object instanceof String) {
        // Explicit Casting to `String`
        String string = (String) object;
    }
}
```

Kotlin

```kotlin
private fun javaTypeCast() {
    val `object` = "Example String"
    if (`object` is String) {
        // No Cast Required, object will be String
    }
}
```

Smart Casts

Casting objects is simpler & m

- Java uses `instanceof` operat
- Java requires explicit casting after true
- Kotlin uses `is` or `!is` operator t
- Kotlin compiler aut true

```java
private void javaTypeCast() {
    Object object = "Example String";

    if (object instanceof String) {
        // Explicit Casting to `String`
        String string = (String) object;
    }
}
```

```kotlin
private fun javaTypeCast() {
    val `object` = "Example String"
    if (`object` is String) {
        // No Cast Required, object will be String
    }
}
```

ZEBRA TECHNOLOGIES

Kotlin – Are there any drawbacks?

Exploring some possible pitfalls of the Kotlin Platform

Method count

Reduced lines in code but increased number of methods

- Kotlin reduces number of lines in code but usually increases method count of compiled code
- This is largely due to how Kotlin implements properties - all val / var declarations create a property
- This removes the need for getters / setters but means every var will generate a getter & setter, whether you need it or not.

Namespace Declaration

Top-level functions can become confusing

- Kotlin allows top-level functions (functions outside of any class, object or interface)
- Top-level functions in Kotlin can be used as a replacement for the static utility methods inside helper classes we code in Java.
- If we have multiple top-level functions within the same application, it can be confusing to know which function is being called

ZEBRA TECHNOLOGIES

Kotlin & Zebra

Some examples of Kotlin Integration with a Zebra Enterprise Device

Simplifying Switch Statements

Using Kotlin ‘”when”’ to simplify Scanner Status Method

Java

```java
@Override
public void onStatus(StatusData statusData) {
    switch (statusData.getState()) {
        case IDLE:
            startScannerRead();
            break;
        case WAITING:
            Logger.i(TAG, logText: "Scanner waiting...");
            break;
        case SCANNING:
            Logger.i(TAG, logText: "Scanner scanning...");
            break;
        case DISABLED:
            Logger.i(TAG, logText: "Scanner Disabled...");
            break;
        case ERROR:
            Logger.i(TAG, logText: "Scanner Error!");
            break;
        default:
            break;
    }
}
```

Kotlin

```kotlin
override fun onStatus(statusData: StatusData?) {
    when(statusData?.state) {
        StatusData.ScannerStates.IDLE -> startScannerRead()
        StatusData.ScannerStates.WAITING -> Log.i(TAG, msg: "Waiting...")
        StatusData.ScannerStates.SCANNING -> Log.i(TAG, msg: "Scanning...")
        StatusData.ScannerStates.DISABLED -> Log.i(TAG, msg: "Disabled!")
        StatusData.ScannerStates.ERROR -> Log.i(TAG, msg: "Error!")
    }
}
```

ZEBRA TECHNOLOGIES

Java

```java
@Override
public void onStatus(StatusData statusData) {
    switch (statusData.getState()) {
        case IDLE:
            startScannerRead();
            break;
        case WAITING:
            Logger.i(TAG, logText: "Scanner waiting...");
            break;
        case SCANNING:
            Logger.i(TAG, logText: "Scanner scanning...");
            break;
        case DISABLED:
            Logger.i(TAG, logText: "Scanner Disabled...");
            break;
        case ERROR:
            Logger.i(TAG, logText: "Scanner Error!");
            break;
        default:
            break;
    }
}
```

```
ıta: StatusData?) {

:es.IDLE -> startScannerRead()
:es.WAITING -> Log.i(TAG, msg: "Waiting...")
:es.SCANNING -> Log.i(TAG, msg: "Scanning...")
:es.DISABLED -> Log.i(TAG, msg: "Disabled!")
:es.ERROR -> Log.i(TAG, msg: "Error!")
```

Simplifying Switch Statements

Using Kotlin “”when”” to simplify Scanner Status Method

```kotlin
override fun onStatus(statusData: StatusData?) {
    when(statusData?.state) {
        StatusData.ScannerStates.IDLE -> startScannerRead()
        StatusData.ScannerStates.WAITING -> Log.i(TAG, msg: "Waiting...")
        StatusData.ScannerStates.SCANNING -> Log.i(TAG, msg: "Scanning...")
        StatusData.ScannerStates.DISABLED -> Log.i(TAG, msg: "Disabled!")
        StatusData.ScannerStates.ERROR -> Log.i(TAG, msg: "Error!")
    }
}
```

```
        break;
    case ERROR:
        Logger.i(TAG, logText: "Scanner Error!");
        break;
    default:
        break;
}
}
```

Enabling Scanning with a Co-Routine

Using Kotlin Co-routines to simplify Scanner Enablement

Java

```java
private void startScannerRead() {
    try {
        try { Thread.sleep( millis: 100); }
        catch (InterruptedException e) { e.printStackTrace(); }
        mScanner.read();
    } catch (ScannerException e) {
        Logger.e(TAG, exceptionQualifier: "ScannerException: " + e.getMessage(), e);
    }
}
```

Kotlin

```kotlin
fun startScannerRead() {
    GlobalScope.launch { this: CoroutineScope
        // Pause
        delay( timeMillis: 100L)
        // Execute
        mScanner?.read()
    }
}
```

- No `try / catch` blocks
- No `Thread.sleep();`
- Null Safe

Java

```java
private void startScannerRead() {
    try {
        try { Thread.sleep( millis: 100); }
        catch (InterruptedException e) { e.printStackTrace(); }
        mScanner.read();
    } catch (ScannerException e) {
        Logger.e(TAG, exceptionQualifier: "ScannerException: " + e.getMessage(), e);
    }
}
```

Kotlin

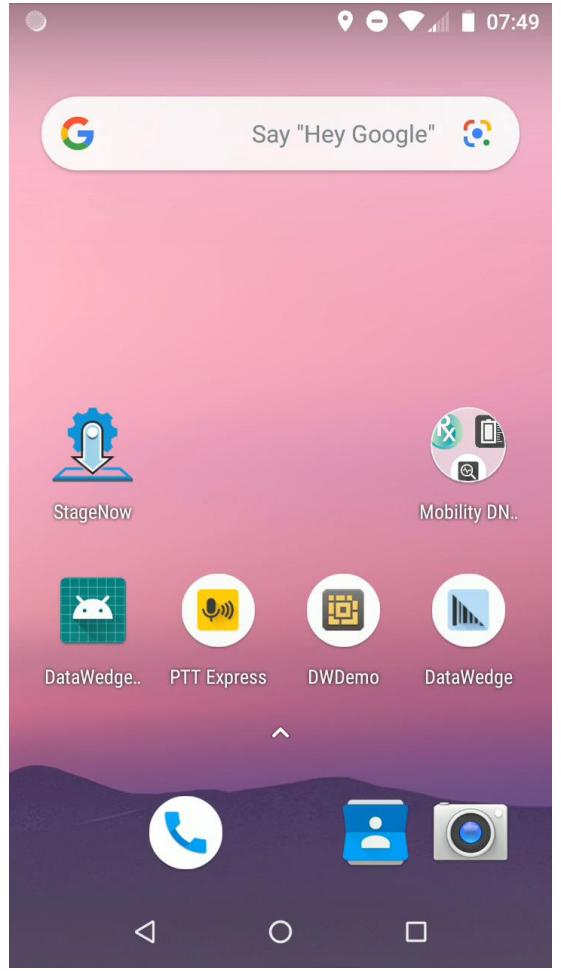
```kotlin
fun startScannerRead() {
    GlobalScope.launch { this: CoroutineScope
        // Pause
        delay( timeMillis: 100L)
        // Execute
        mScanner?.read()
    }
}
```

- No `try / catch` blocks
- No `Thread.sleep();`
- Null Safe

Developing Kotlin applications for Zebra Android devices

Demo / Example: Kotlin-based Barcode Sample 1

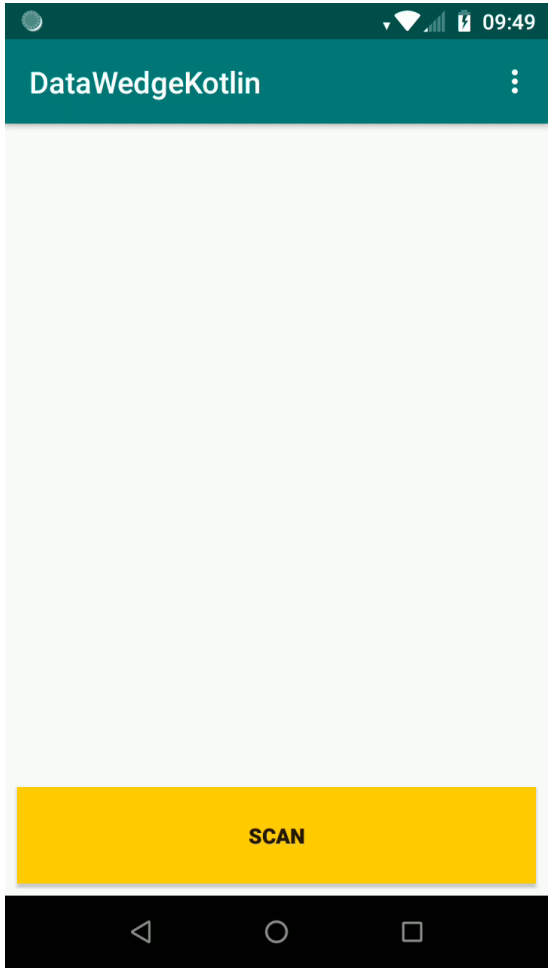
- Official Java-based sample for Barcode Scanning is:
  - https://github.com/Zebra/samples-emdkforandroid-7_3/tree/master/BarcodeSample1
- Converted to Kotlin this sample is:
  - https://github.com/darryncampbell/samples-emdkforandroid-7_3/tree/BarcodeSample1-Kotlin/BarcodeSample1
- Uses EMDK library from Gradle (Jar file)
  - **In future**, Jar will be annotated with @Nullable and @NonNull to enhance Kotlin integration



Developing Kotlin applications for Zebra Android devices

Demo / Example: Kotlin-based Scanning with DataWedge

- Using the DataWedge API we can fully integrate barcode scanning into a Kotlin application using Android Intents
  - Approach #2 as detailed at the following developer portal post:
    - https://developer.zebra.com/community/home/blog/2018/11/19/kotlin-and-developing-kotlin-applications-for-zebra-devices
  - Sample app: https://github.com/darryncampbell/DataWedgeKotlin
    - This is a proof of concept. See video (right)



Questions?

ZEBRA DEVELOPER PORTAL
http://developer.zebra.com

Sign up for news
Join the ISV program

Thank You

BACKUP