Why we love Kotlin?

@DoctorBox

Vivek Singh
Abhishek Bansal
(all things Android @DoctorBox)

What People Say?

1. Kotlin is less Verbose
    - Less boilerplate code to write
    - Do more with less
    - Readable
2. Interoperable with Java
    - Easy to get started
3. At par performance
    - Similar byte code structure
4. Modern Syntax
    - Lambdas/Streams/Functional/Type Inference
5. Null Safe
6. Android is now Kotlin first
7. Etc. etc.

Our Agenda

1. Do a direct comparison with Java and provide a practical guide to what Kotlin has to offer.
2. In terms of language features
    - Null safety in type system
    - Immutability
    - Data Classes
    - Extension Functions
    - Named Parameters, Default Values
    - Inbuilt Higher Order functions and chaining
    - Lambdas
    - Inline Functions
    - Delegated Properties
    - Sealed Classes
    - CoRoutines
    - Android Extensions and KTX

Null Safety

- Null safety is inbuilt in Kotlin’s Type System.
- Kotlin Type System differentiates between the references that can hold null or can not hold null.
- Nullable Types can be differentiated from non nullable types by using '**?**'.

```
val a = "Kotlin" // NON NULLABLE TYPE

val b: String? = null // NULLABLE TYPE
```

- Try using Not Null Type wherever possible.
- For Nullable Types use Safe Call Operator ‘**?.**’

Safe Casts

- Regular Cast results sometime in Class Cast Exception while explicitly casting them.
- E.g. y as String // This cast will never succeed
- If y = null, Code above will throw exception.
- To avoid throwing exception we can use safe cast operator 'as?'
- It will cast the variable to nullable type, which can be easily used with safe call operator
- E.g. y as? String.

Immutability

- Immutability - Once created, its value can not be changed.
- Can’t change the property value rather create a new value.
- Immutability In Kotlin System - val, const
- Val denotes Read Only types. Can’t change until custom getter is provided.
- Const denotes immutable types can’t change in any condition once initialized.

Data Class

- Its Main purpose is to hold Data.
- Contains only field and methods to access these fields
- Don't contain any additional Functionality
- Can't operate on their own.
- Compiler automatically create - equal(), hashCode(), toString(), componenN() functions and copy().
- data class User(val name: String, val age: Int)

Extension Functions/Properties

Ability to extend a class with new functionality without having to inherit from the class or use any type of design pattern .

Java

```
if(show) {
 saveButton.setVisibility(View.VISIBLE);
} else {
 saveButton.setVisibility(View.GONE);
}
```

OR

```
ViewUtils.toggleVisibility(saveButton, show);
```

Kotlin

```
saveButton.setVisible(show)
```

Extension Function

```
fun View.setVisible(isVisible: Boolean) {
 visibility = if (isVisible) View.VISIBLE
else View.GONE
}
```

More applications include animations, applying configuration, using system services and many more..

Named and Default Parameters

Animation Demo

Higher Order Functions

- Functions are First class Citizens in a class.
- Can be stored in variables, passed as arguments and returned from a function.
- HOF - Function that accept another function as parameter or return a function.
- Language provide many inbuilt higher order function like map, filter, groupBy, forEach, take, firstOrNull and many more.
- Can chain these function for more readability and comprehensibility.

Lambdas

- Are essentially anonymous functions.
- Has First class citizen support - anonymous function that can be assigned to variables, passed as an argument and can be returned from an argument.
- All lambdas are functions but not vice versa.
- val sum = {x: Int, y: Int -> x+y} // Lambda as property

Inline Function

- Using Higher Order Functions forces some penalties.
- Inline to rescue
- Instead of creating function object for parameters and generating a call, Compiler emits code.
- The inline modifier affects both the function itself and the lambdas passed to it, all of those will be inlined into the call site.
- In case you want only some of the lambdas passed to an inline function to be inlined, you can mark some of your function parameters with the **noinline** modifier.

Delegated Properties

- Delegation is an act of letting someone else do your job
- Kotlin lets you delegate property initialization to custom classes(delegates)
- Inbuilt Delegates
    - Lazy
    - Observable
    - Vetoable
    - notNull
- Demo

Sealed Classes

- Are used for representing restricted class hierarchies
- A value can have one of the types from a limited set, but cannot have any other type.
- Super powered Enums
- Sealed class is abstract by itself and cannot have non private constructor
- Become a powerful concept combined with when statement
- MVVM Demo

CoRoutines

1. New way of doing asynchronous programming
2. Behave and act like light weight threads but are **NOT** threads

Power of CoRoutines

https://github.com/abhishekBansal/coroutine-mvvm-architecture-demo/commit/b447eba82592fd77fdbe1300be416fd4e069c101?diff=split#diff-280be68b4daf32455c5f56e8ba0b3448L22

Extensions

1. Synthetics
    a. Part of KTX by jetbrains
    b. Easy to use
    c. caution: do not expose nullability
2. @Parcelize
    a. Just one annotation for generating parcelable implementation
    b. Handles enums, collections by default
    c. Experimental
3. KTX- https://developer.android.com/kotlin/ktx

When (the “switch” operator)

Java

```
Intent intent = null
switch (type) {
   TYPE1 -> {
       intent = Intent(context, Activity1::class.java);
       break;
   }
   TYPE2 -> {
       intent = Intent(context, Activity2::class.java);
       break;
   }
   TYPE3 -> {
       intent = Intent(context, Activity3::class.java);
       intent.putExtra(BUNDLE_KEY, "arg");
       break;
   }
}

if(intent != null) {
   startActivity(intent)
}
```

When (the “switch” operator)

Kotlin

```
val intent = when (type) {
     TYPE1 -> Intent(context, Activity1::class.java)
     TYPE2 -> Intent(context, Activity2::class.java)
     TYPE3 -> Intent(context, Activity3::class.java)
               .apply {putExtra(BUNDLE_KEY,”arg”)}
}
startActivity(intent)
```

Further Readings/References

1. https://proandroiddev.com/the-argument-over-kotlin-synthetics-735305dd4ed0
2. https://kotlinlang.org/docs/reference/sealed-classes.html
3. https://proandroiddev.com/delegation-in-kotlin-e1efb849641
4. https://proandroiddev.com/the-magic-in-kotlin-delegates-377d27a7b531
5. https://proandroiddev.com/kotlin-android-synthetics-performance-analysis-with-butterknife-90a54ca4325d

Thanks