How can Kotlin change Android development

Mate Herber

Software Engineer @ Skyscanner

Native Development Meetup

November 25, 2015

Why would you want to replace Java?

- Android is stuck on Java 6-7
  - Several possible workarounds: JSR-310 backport, Retrolambda, RxJava

- Problems with Java
  - Not expressive
  - Verbose
  - Null checking hell (still NPE issues)
  - Util hell

About Kotlin

- JVM language
- Developed by JetBrains
- Named after Kotlin island near St. Petersburg
- Development started in 2010 going public in July, 2011
- Current version: 1.0.0-beta-2422

  (1.0 should be out by the end of this year)

Why Kotlin? There are many options...

Kotlin

Scala

Xtext

Groovy

Ceylon
Say more, more clearly

Why Kotlin? There are many options…

Some measurements (© Jake Wharton)

	Jar Size	Dex Size	Method Count	Field Count
kotlin-runtime-0.10.195	354 KB	282 KB	1071	391
kotlin-stdlib-0.10.195	541 KB	835 KB	5508	458
scala-library-2.11.5	5.3 MB	4.9 MB	50801	5820
groovy-2.4.0-grooid	4.5 MB	4.5 MB	29636	8069
guava-18.0	2.2 MB	1.8 MB	14833	3343

	Groovy	Xtext	Scala	Ceylon	Kotlin
concise	+	+	+	+	+
stronger type system	−	−	+	+	+
can be easily used from Java	+	+	−	−	+

Data Classes

- Let’s add a simple (immutable) data class in Java

Data Classes

```java
package com.meetup.ndev.kotlinsampleapplication.model;

public class Person {
    private final String mFirstName;
    private final String mLastName;
    private final int mAge = 25;

    public Person(String mFirstName, String mLastName, int mAge)
{
        this.mFirstName = mFirstName;
        this.mLastName = mLastName;
        this.mAge = mAge;
    }

    public String getFirstName() {
        return mFirstName;
    }

    public String getLastName() {
        return mLastName;
    }

    public int getmAge() {
        return mAge;
    }
```

```java
@Override
    public String toString() {
        return "Person{" +
                "mFirstName='" + mFirstName + '\'' +
                ", mLastName='" + mLastName + '\'' +
                ", mAge=" + mAge + '}';
    }

@Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
          return false;
        Person person = (Person) o;
        if (mAge != person.mAge) return false;
        if (mFirstName != null
          ? !mFirstName.equals(person.mFirstName)
          : person.mFirstName != null) false;
        return !(mLastName != null
          ? !mLastName.equals(person.mLastName)
          : person.mLastName != null);
    }

    @Override
    public int hashCode() {
        int result = mFirstName != null
          ? mFirstName.hashCode() : 0;
        result = 31 * result + (mLastName != null
          ? mLastName.hashCode() : 0);
        result = 31 * result + mAge;
        return result;
    }
}
```

Data Classes

```
package com.meetup.ndev.kotlinsampleapplication.model;

public class Person {
    private final String mFirstName;
    private final String mLastName;
    private final int mAge = 25;

    public Person(String mFirstName,
{
        this.mFirstName = mFirstName
        this.mLastName = mLastName;
        this.mAge = mAge;
    }

    public String getFirstName() {
        return mFirstName;
    }

    public String getLastName() {
        return mLastName;
    }

    public int getmAge() {
        return mAge;
    }
```

```
@Override
    public String toString() {
        return "Person{" +
                "mFirstName='" + mFirstName + '\'' +
                ", mLastName='" + mLastName + '\'' +
                ", mAge=" + mAge + '}';
    }

@Override
                 equals(Object o) {
                = o) return true;
                ull || getClass() != o.getClass())
                alse;
                son = (Person) o;
                = person.mAge) return false;
                Name != null
                Name.equals(person.mFirstName)
                .mFirstName != null) false;
                LastName != null
                Name.equals(person.mLastName)
                .mLastName != null);



                hCode() {
        int result = mFirstName != null
          ? mFirstName.hashCode() : 0;
        result = 31 * result + (mLastName != null
          ? mLastName.hashCode() : 0);
        result = 31 * result + mAge;
        return result;
    }
}
```

Data Classes

```kotlin
package com.meetup.ndev.kotlinsampleapplication.model

data class Person(val firstName: String, val lastName: String, val age: Int = 25) {
}
```

Data Classes

```
package com.meetup.ndev.kotlinsampleapplication.model

data class Person(val firstName: String, val lastName: String, val age: Int = 25) {
}
```

- equals/hashCode
- toString
- componentN() – multi-declarations support

```
val (firstName, lastName, age) = john
Log.d("Sample", "$firstName $lastName is $age old.")
```

- copy()

```
val john = Person("John", "Doe", 25)
val jane = john.copy(firstName = "Jane")
```

Lambda Expressions

- Java had Anonymous classes for lambda expressions (from Java 8 true lambda)
- Can use named functions by function references (::functionName) but is not ideal (reflection)

```
fun <T> filter(list: List<T>, filterFunc: (T) -> Boolean): List<T> {
    val result = arrayListOf<T>()
    for (item in list) {
        if (filterFunc(item)) {
            result.add(item)
        }
    }
    return result
}
```

```
val list = listOf("apple", "pear", "peach")
val filtered = filter(list, { item -> item.startsWith("p") }) // list = {"pear", "peach"}
```

- Inline functions -> inline both function and used lambda to avoid runtime penalties

Extension Functions

- C# like extension methods
  - You have to explicitly import the extension before usage
- Extend a class without inheritance
- Just a syntactic sugar but really powerful (Java Util classes)
- Can handle calling extensions on null values

Extension Functions

```
public inline fun <T> Iterable<T>.filter(predicate: (T) -> Boolean): List<T> {
    val arrayList = ArrayList<T>()
    for (element in this) {
        if (predicate(element)) {
            arrayList.add(element)
        }
    }
    return arrayList
}
```

```
public inline fun <T> Iterable<T>.forEach(operation: (T) -> Unit): Unit {
    for (element in this) operation(element)
}
```

```
listOf("apple", "pear", "peach").filter { it.startsWith("p") }.forEach { Log.d("Sample", "$it") } // pear, peach
```

Null Safety

- The old Java way:

```
public class Group {
    private List<Person> mPersons;

    public Group(List<Person> mPersons) {
        this.mPersons = mPersons;
    }

    public String getCapitalStreetOfFirstPerson() {
        if (mPersons != null && mPersons.size() > 0) {
            Person person = mPersons.get(0);
            if (person != null) {
                Address address = person.getAddress();
                if (address != null) {
                    String street = address.getStreet();
                    if (street != null) {
                        return street.toUpperCase();
                    }
                }
            }
        }
        return null;
    }
}
```

Null Safety

```
data class Group(val persons: List<Person>?) {
    fun getCapitalStreetOfFirstPerson(): String? = persons?.firstOrNull()?.address?.street?.toUpperCase()
}
```

- Null Safe operators:
  - ?. (safe call)
  - ?: (elvis operator)
  - !!
  - as? (safe cast)
- This is all nice…

Null Safety

- …But the real deal is:
  - By default **none** of the types are nullable
  - Compile time check for null handling

```
var a: String = null; // compile time error
var a: String? = null; // valid
```

```
fun printLog(nullableString : String?) {
    Log.d(TAG, nullableString.toUpperCase()); //compile time error
}
fun printLogSafe(nullableString : String) {
    Log.d(TAG, nullableString.toUpperCase()); //valid
}
```

Extension Function Expressions

- Really advanced language feature
- Plain old lambdas : (T) -> R (gets a T and gives back an R)
- Extension lambdas: K.(T) -> R (it’s an extension function of K which gets a T and returns an R)
- Basically they are anonymous extension methods on a defined type

```
public inline fun <T> T.apply(f: T.() -> Unit): T {
    f();
    return this
}
```

```
Person(Address("Washington Street")).apply {
    name = "John Doe"
}
```

And more…

- Smart casting
- Delegation
- Property Delegation
- Ranges
- Generics
    - Declaration-site variance
    - Type projections
- And more…

Mixing Kotlin and Java

Call Java from Kotlin

- Getters/Setters mapped to properties
- Void<->Unit & Object<->Any
- Platform types are not so null safe ☹

Call Kotlin from Java

- Package level functions go to a Java package
- Properties are mapped to getters and setters
- Behaviour annotations:
  - @JvmName
  - @JvmField
  - @JvmStatic
  - @JvmOverloads
- Can set null for the notnull parameters -> instant NPE (thanks compiler)

Kotlin Android Extensions

- Reach views in convenient way instead of findViewById
- R.layout.main -> import kotlinx.android.synthetic.main.*
- Generates a caching method to each Fragment / Activity
- Creates a property delegate for all views with id
- Best practice to name the views as kotlin properties

```
import kotlinx.android.synthetic.activity_main.*
import java.util.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //(TextView)findViewById(R.id.helloTextView).setText("Hello World");
        helloTextView.text = "Hello World"
    }
```

Anko

- Great collection of extensions for Android API
- Type Safe builder for android UI
- https://github.com/JetBrains/anko

Anko – UI builder

- Type safe builder for UI
- Easy to build instead of XML
- Inflating is just a method call
- Assigning listeners at the same place
- Easy to extend

```
verticalLayout {
    padding = dip(30)
    editText {
        hint = "Name"
        textSize = 24f
    }
    editText {
        hint = "Password"
        textSize = 24f
    }
    button("Login") {
        textSize = 26f
    }
}
```

And more…

- Intent builder
- Asynchronous task helper
- Dialog and Toast builder
- Logging helper
- And more…

Why don’t we use kotlin (yet)

- Increased method count and app size (not really relevant)
- Still in beta (soon™ to be released)
- Needs some workaround while using specific libraries (e.g: Mockito)
  - https://devnet.jetbrains.com/thread/443551
- Documentation sometimes lacks information about features
- There is not many community resource about kotlin specific libs / know-hows
- All the developers must learn the new language

References

- https://kotlinlang.org/docs/reference/
- https://kotlinlang.org/docs/kotlin-docs.pdf
- https://plus.google.com/+JakeWharton/posts/WSCoqkJ5MBj
- https://www.youtube.com/watch?v=-BvN0X5tqjw
- https://www.youtube.com/watch?v=6OUfz3OekMI
- https://realm.io/news/droidcon-michael-pardo-kotlin/
- https://skillsmatter.com/skillscasts/6651-advancing-development-with-the-kotlin-language
- https://leanpub.com/kotlin-for-android-developers
- http://blog.jetbrains.com/kotlin/

Questions?

Thanks for listening

CODE VOYAGERS

#codevoyagers
http://codevoyagers.com/
mate.herber@skyscanner.net