GDGDevFest
2017 Ahmedabad

Introduction to Kotlin for Android App Development

Hardik Trivedi

About Me

Hardik Trivedi

- I am a computer program writer
- Works at Globant, Pune, IN
- An active community speaker
- Co-author of an upcoming book "Kotlin Blueprints"
- I love writing tech blogs
- I am mentor for college graduates and professionals and consultant to companies

What is Kotlin?

- Statically typed object oriented language
- Targets JVM, Android and even JavaScript
- Can do anything which Java can.
- Combines Object Oriented and Functional Programming features.
- But it’s not Functional language.
- Developed and maintained by JetBrains
- Open source
- Released under Apache 2 OSS License

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

Safe

Avoid entire classes of errors such as Null pointer exception

Interoperable

100% interoperable with Java. Support for Android, Browser and Native

Tool-friendly

Excellent tooling support from JetBrains

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

Safe

Avoid entire classes of errors such as Null pointer exception

Interoperable

100% interoperable with Java. Support for Android, Browser and Native

Tool-friendly

Excellent tooling support from JetBrains

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

Safe

Avoid entire classes of errors such as Null pointer exception

Interoperable

100% interoperable with Java. Support for Android, Browser and Native

Tool-friendly

Excellent tooling support from JetBrains

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

Safe

Avoid entire classes of errors such as Null pointer exception

Interoperable

100% interoperable with Java. Support for Android, Browser and Native

Tool-friendly

Excellent tooling support from JetBrains

Why Kotlin?

- Stuck between Java 6 and 7
- No Lambdas, No Streams
- Inability to add methods in platform APIs, have to use Utils for that
- Loads of NullPointerException
- For Android, it’s Nullability everywhere
- For Android, it loves ceremonies of API
- Need of modern and smart coding language

Kotlin used by industry giants

Getting Started

- Kotlin is shipped with IntelliJ IDEA 15
- Plugin for Eclipse Luna or greater version. Setup Link
- Try online IDE http://try.kotlinlang.org/
- **Android Studio 3.0 is released and Kotlin is by default supported. No other setup required.** Download Android Studio 3.0
- To use Kotlin with the older versions or below Android Studio 3.0, we need to manually install the latest Kotlin Plugin. Setup Link

Syntax Crash Course

Compile some Kotlin in mind

Object Declaration and Initialization

```
// Immediate assignment
val num: Int = 10

// Implicitly inferred String type
val pName = "Hardik Trivedi"

// Explicit type declaration
var personList:List<String> = ArrayList()

// use underscores to make number constants more readable
val creditCardNumber = 1234_5678_9012_3456L
```

val vs var

```
// Immutable
"Anything which will not change is val"


// Mutable
"If value will be changed with time use var"
```

If as expression

```
/* if branches can be blocks, and the last
expression is the value of a block */
val max = if (a > b) {
    println("a is greater than b")
    a
}
else {
    println("a is not greater than b")
    b
}

// As expression
val max = if (a > b) a else  b
```

When function

```
// when replaces the switch operator of C, Java-like languages.
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}

when (x) {
    in 1,2 -> print("X is either 1 or 2")
    in validNumbers -> print("x is valid")
    in 10..20 -> print("x is inside the range")
    else -> print("none of the above")
}
```

Loops

```
// for loop iterates through anything that provides an iterator.
val list = listOf("Optimus Prime", "Bumblebee", "Ironhide")

// Simple for loop
for (item in list) {
    println(item)
}

// De structured
for ((index, value) in list.withIndex()) {
    println("the element at $index is $value")
}
```

Functions

```
//Functions in Kotlin are declared using the fun keyword
fun sum(a: Int, b: Int): Int {
   return a + b
}

/*When a function returns a single expression, the curly braces can be omitted and
the body is specified after a = symbol*/
fun sum(a: Int, b: Int, c: Int) = a + b + c


// Function with default argument
fun sum(a: Int, b: Int, c: Int, d: Int = 0) = a + b + c + d
```

Data classes

```
data class Country(val name: String, val capital: String)

//Code
val india=Country("India","New Delhi")
val norway=Country(capital = "Oslo",name = "Norway")
val (name,capital)=india
println(india.toString())
println(norway.toString())
println("$name ,$capital")

//Output
Country(name=India, capital=New Delhi)
Country(name=Norway, capital=Oslo)
India ,New Delhi
```

Null Safety

No more NullPointerException

Kotlin's type system is aimed to eliminate NullPointerException from our code.

Null Safety

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok


/* Now, if you call a method or access a property on a, it's guaranteed not to
cause an NPE, so you can safely say */
val l = a.length


val l = b.length//error: 'b' can be null NOW WHAT TO DO?
```

Null Safety

```
/* You can explicitly check if b is null, and handle the two options separately */
val l = if (b != null) b.length else -1

//Your second option is the safe call operator, written ?.
val length=b?.length

/* To perform a certain operation only for non-null values, you can use the safe call
operator together with let */

val listWithNulls: List<String?> = listOf("A", null)
for (item in listWithNulls) {
   item?.let { println(it) } // prints A and ignores null
}
```

Extensions

```
// Extension function
fun Date.isTuesday(): Boolean {
    return getDay() == 2
}
// Somewhere in code
val date=Date()
println(date.isTuesday())

/* Similarly to functions, Kotlin supports extension properties */
val <T> List<T>.lastIndex: Int
    get() = size - 1
```

Lambdas

```
// Lambdas provides implementation of functional interface (which has single
abstract method-SAM)

button.setOnClickListener(v -> { doSomething() })

// Lambdas other usage
forecastResult.list.forEachIndexed { index, forecast ->
    with(forecast) {
        println("For index $index value is ${forecast.toString()}")
    }
}
```

Higher order functions

- Higher order functions are those which accepts functions as a parameter or returns the function.

```
fun Int.perform(other: Int, func: (Int, Int) -> Int): Int {
    return func(this, other)
}

// Output
println(10.perform(20, { a, b -> a + b }))
println(10.perform(20) { a, b -> a - b })
println(10.perform(20, { a, b -> a * b }))
println(10.perform(20) { a, b -> a / b })
```

Lazy or Late

- Lazy object gets initialised only while it's first usage. And will remain always immutable. It's a Lambda function.

```
override val toolbar by lazy { find<Toolbar>(R.id.toolbar) }
```

- It's just a late initialisation with non null type. It remains mutable. Used when object is initialised by Dependency Injection or setUp method of unit test case

```
lateinit var  mockLoanCalculator: LoanCalculator
@Before
public void setUp() throws Exception {
    mockLoanCalculator = mock(LoanCalculator::class.java)
}
```

Let and apply

```
/**
 * Calls the specified function [block] with `this` value as its argument and returns
its result.
 */
val result = person.let { it.age * 2 }

/**
 * Calls the specified function [block] with `this` value as its receiver and returns
`this` value.
 */
supportActionBar?.apply {
    setDisplayHomeAsUpEnabled(true)
    setDisplayShowHomeEnabled(true)
}
```

Wow moments

See Kotlin in Action !!!

Wow moments

Java

```
// Possible only if it's Java 8
public interface InterfaceA {
    default void defaultMethod(){
        System.out.println("Interface A
default method");
    }
}
```

Kotlin

```
interface ToolbarManager {
    fun initToolbar() {
        toolbar.inflateMenu(R.menu.main_menu)
        toolbar.setOnMenuItemClickListener {
            // Your code
            true
        }
    }
}
```

Wow moments

Java

```
private String readInputStream(InputStream is) throws
Exception {
    String line = null;
    StringBuilder sb = new StringBuilder();

    BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(is));

    while ((line = bufferedReader.readLine()) != null)
{
        sb.append(line);
    }

    bufferedReader.close();
    return sb.toString();
}
```

Kotlin

```
val inputAsString =
is.bufferedReader().use
{ it.readText() }
// defaults to UTF-8
```

Wow moments

Java

```
Button clickButton = (Button)
findViewById(R.id.clickButton);
clickButton.setOnClickListener( new
OnClickListener() {

    @Override
    public void onClick(View v) {
        ***Do what you want with the
click here***
    }
});
```

Kotlin

```
import
kotlinx.android.synthetic.main.activity_
detail.*

clickButton.setOnClickListener {
    ***Do what you want with the click
here***
}
```

Wow moments

Java

```
button.setVisibility(View.VISIBLE)

button.setVisibility(View.GONE)
```

Kotlin

```
fun View.visible() {
    this.visibility = View.VISIBLE
}

fun View.gone() {
    this.visibility = View.GONE
}

button.visible()
textView.gone()
```

Wow moments

Java

```java
public class MySingleton {
    private static MySingleton myObj;

    private MySingleton(){
    }

    public static MySingleton getInstance(){
        if(myObj == null){
            myObj = new MySingleton();
        }
        return myObj;
    }
}
```

Kotlin

```kotlin
object MySingleton {
    var num: Int = 0

    fun domeSomeThing() {
        println("Kotlin is awesome!!!")
    }
}
```

Wow moments

Java

```
SharedPreferences sharedpreferences =
getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);

email.setText(sharedpreferences.getStrin
g(Email, ""));

SharedPreferences.Editor editor =
sharedpreferences.edit();
editor.putString("email",
"trivedi.hardik.11@gmail.com");
editor.apply();
```

Kotlin

```
private var email: String by
DelegatedPreference(this, "email", "")

email="trivedi.hardik.11@gmail.com"

txtEmail.text=email
```

Wow moments

Java

```
Movie movie = moviesList.get(position);
holder.title.setText(movie.getTitle());
holder.genre.setText(movie.getGenre());
holder.year.setText(movie.getYear());
```

Kotlin

```
Movie movie = moviesList[position]
with(movie) {
    holder.title.text=title
    holder.genre.text=genre
    holder.year.text=year
}
```

Wow moments

Java

```
private class GetWeatherTask extends
AsyncTask<String, Void, Forecast> {
     protected Forecast doInBackground(String
zipCode) {
         return WeatherAPI().execute(zipCode);
     }

     protected void onPostExecute(Forecast
result) {
          showData(result);
     }
 }

 new GetWeatherTask().execute(380015);
```

Kotlin

```
fun loadWeatherData() = async(UI) {
     val waiter = bg {
WeatherApi(zipCode).execute() }
     showData(waiter.await())
}
```

Wow moments

Java

```
Intent intent = Intent(this,BActivity.class)
intent.putExtra("id", 5)
intent.setFlag(Intent.FLAG_ACTIVITY_SINGLE_TOP)
startActivity(intent)
```

Kotlin

```
startActivity(intentFor<SomeOtherActivity>("id" to 5).singleTop())
```

Smart Cast

```
// Java style
if (obj instanceof TextArea) {
   ((TextArea)obj).append("some data");
} else {
   // Do something
}


// Kotlin does it smartly
if (view is TextArea) {
   view.append("Kotlin is awesome!")
} else {
   // Do something
}
```

Anko - Android's buddy for Kotlin

```
textView {
  id = R.id.errorView
  textColor = ContextCompat.getColor(ctx, R.color.red_error)
  text = string(R.string.error_view_login_text)
  textSize = 14f
  visibility = View.GONE
}

textView {
  lparams(width = matchParent, height = wrapContent) {
    gravity = Gravity.CENTER
    leftMargin = dip(16)
    rightMargin = dip(16)
  }
}
```

Anko - Android's buddy for Kotlin

```
// Showing Alerts
alert("Hi, are you enjoying the Kotlin talk?") {
    yesButton { toast("Yes :)") }
    noButton {}
}.show()

// Showing progress dialog
val dialog = progressDialog(message = "Please wait a bit…", title = "Fetching data")

// Showing SnackBar
snackbar(view, "Hi there!")
snackbar(view, R.string.message)
longSnackbar(view, "Wow, such duration")
snackbar(view, "Action, reaction", "Click me!") { doStuff() }
```

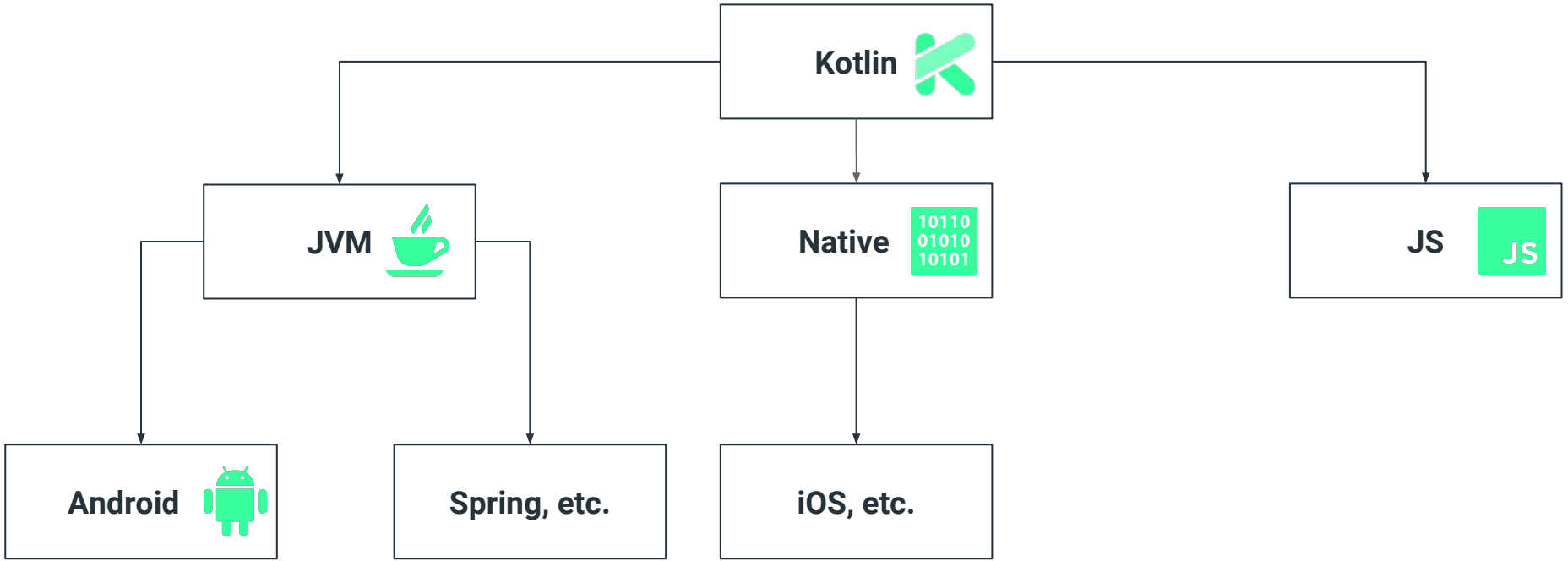
There is more to come than meets the eye

- Visibility Modifiers
- Companion Objects
- Nested, Sealed classes
- Generics
- Coroutines
- Operator Overloading
- Exceptions
- Annotations
- Reflection
- And Many more...

Improvement areas for Kotlin

- Increased build time by few seconds
- Adds approx 6K more methods in final package
- No static analysers, but same can be achieved using companion objects
- Mocking kotlin classes with Mockito is painful
- Operator overloading may lead to bad result if not used properly
- Smaller community and less available help
- Might be difficult to code in functional paradigm

Future of Kotlin



References

- Official Website
- Git repo from Developer Advocate at JetBrains
- Kotlin vs Java build comparison
- Android Development with Kotlin - Jake Wharton
- Antonio Leiva's Blog
- Anko Github page
- Sample Android app using Kotlin

End note

```
when (Talk.STATUS) {
        TALK_ENDS -> println("Let's discuss Queries")
        NO_QUERIES -> println("Thank you :)")
}
```

Thank You

trivedi.hardik.11@gmail.com

https://trivedihardik.wordpress.com

@11Trivedi