“Open for extension and closed for modification”

“Don't code when we are thinking. Don't think when we are coding”

By Hitesh Bhalala

What is this.... Kotlin?

- New Programming language for modern multi platform applications
- For JVM (Kotlin build on JVM), Android, Browser, Native (Kernel level)
- Development start before 6 years (on 2011) by JetBrains (IntelliJ developers) team
- It is a Russian Island close to St. Petersburg
- Ads small overhead to the Android dev env and dex method count increase by ~6k methods
- Finally Android team announced first-class support for Kotlin on May 17, 2017

What have we been missing in Java?

- We can define our own higher order functions and lambda's, no need functional interface. (function as parameter or return function)
- Embedded lists iterators and mapping functions
- Class extensions (As like Collections.sort version)
- No elegant ways of avoiding NPE's

Why Kotlin?

- Drastically reduce amount of code
- Avoid some errors (null pointer exception)

Kotlin Benefit?

- **Compatibility**

  -JDK 6, older Android devices, Android Studio

- **Performance**

  -As fast as an equivalent Java one (Sometime faster than Java)

- **Interoperability**

  -Java -databinding -Dagger

- **Very compact runtime library**

  - Add less than 100K size at runtime

- **Supports efficient incremental compilation**

  -Support from kotlin version 1.1.1

Case studies

- Pinterest

  -150M Users

- Basecamp's Android app
- Keepsafe's App Lock app

  -30% decrease source code line

- Uber
- EverNote
- Atlassian

Tools for Kotlin in Android

- **Android Studio 3.0 and up**

  -Inbuilt Kotlin plugin

- **Kotlin Extensions**

  -Latest version: 1.1.60 (24th Nov 2017)

  -findViewById without databinding

  -// Using R.layout.activity_main from the 'main' source set

  ```
  Import kotlinx.android.synthetic.main.activity_main.*
  ```

- **Anko**

  -Android API, helper classes

How to setup Kotlin project for Android?

- classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.1.2"

  -External project gradle

- apply plugin: 'kotlin-android'

  apply plugin: 'kotlin-android-extensions

  //dependency

  compile "org.jetbrains.kotlin:kotlin-stdlib:1.1.60"

  //optional but required for other extensions

  ```
  androidExtensions{
          Experimental = true
  }
  ```

  -Add into app gradle

Java vs Kotlin

Hello World

```java
System.out.println("Hello, World!");
```

```kotlin
println("Hello, World!")
```

Variables and Constants

```java
String name;
int count = 50;
final String TAG = "SimpleJavaTag";
```

```kotlin
var name: String? = null
var count = 50
val TAG = "SimpleJavaTag"
```

Explicit Type

```java
final Double per = 99.99;
```

```kotlin
val per : Double = 99.99
```

String Interpolation

```java
final int op1 = 30;
final int op2 = 20;
String sum = "Sum of "+op1 + " and "+op2 +" is "+(op1+op2);
```

```kotlin
val op1 = 30
val op2 = 20
var sum = "Sum of ${op1} and ${op2} is ${op1 + op2}"
```

Java vs Kotlin

Collections

```
List<Integer> numbers = Arrays.asList(10, 20, 30);
```

```
var numbers = mutableListOf(10, 20, 30)
```

```
Map<String, Integer> map = new LinkedHashMap<>();
map.put("Android", 1);
map.put("IOS", 2);
```

```
var map = linkedMapOf("Android" to 1, "IOS" to 2)
```

```
Iterator<Map.Entry<String, Integer>> it = map.entrySet().iterator();
while (it.hasNext()){
   Map.Entry<String, Integer> entry = it.next();
   System.out.println(entry.getKey()+" and "+entry.getValue());
}
```

```
map.entries.forEach {
   println("${it.key} and ${it.value}")
}
```

```
for (Integer number : numbers) {
   System.out.println("value = "+number);
}
```

```
numbers.forEach {
   println("value = ${it}")
}
```

Say hello to kotlin

```
fun sayHello(name: String): Unit {
        print("Hello ${name}!" + " Welcome to Kotlin");
}
```

OR

```
fun sayHello(name: String): Unit = print("Hello ${name}!" + " Welcome to Kotlin");
```

OR

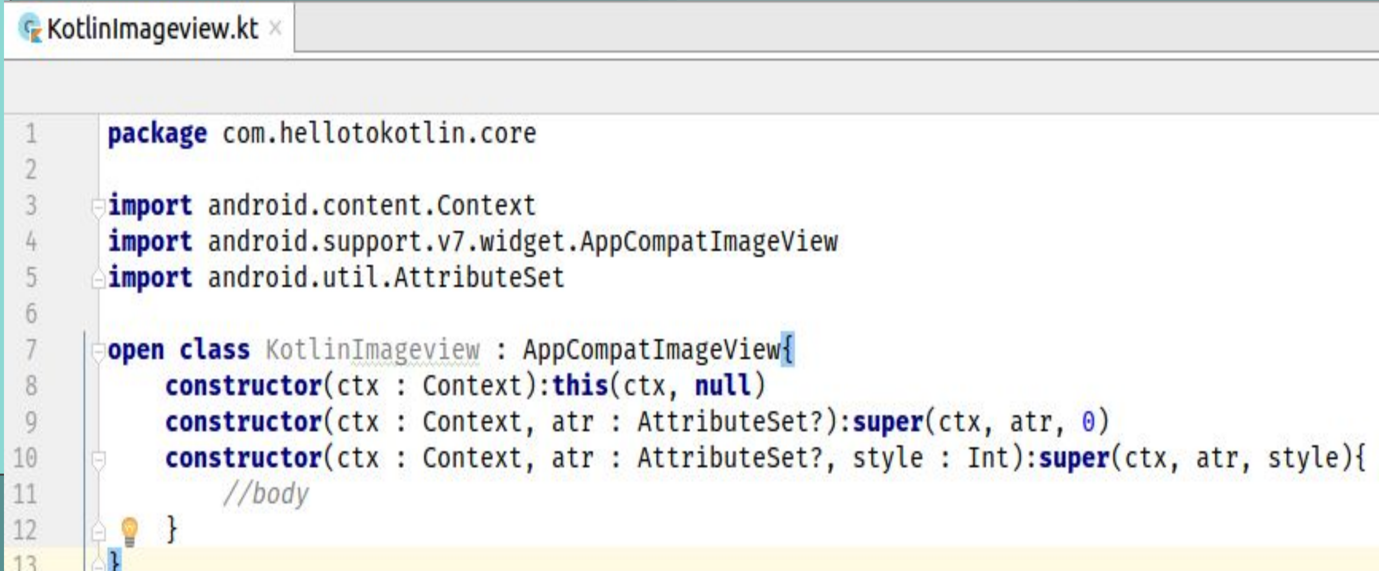
```
fun sayHello(name: String) = print("Hello ${name}!" + " Welcome to Kotlin");
```

Kotlin classes :

OOP Concept

- Inherit from Java's equivalent of Object to Any
- By default are final (Define open for public access)
- Primary constructor define with class name and if it exist then all other constructor should delegate that constructor
- If super class have not any constructor then syntax is

```
class  Sub : Super(){
}
```



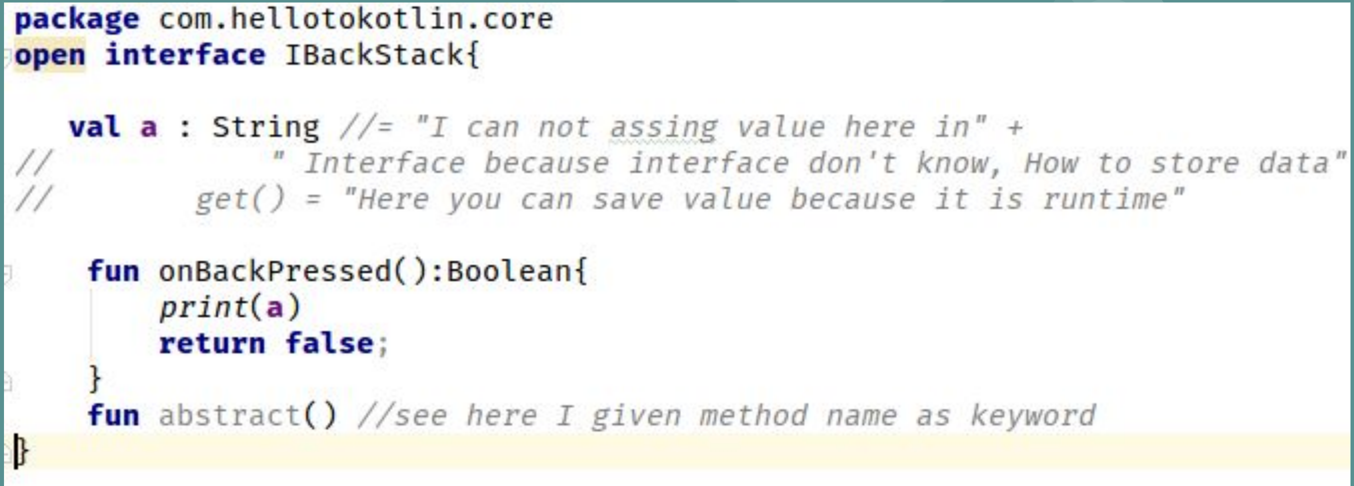
KotlinImageview.kt

```kotlin
package com.hellotokotlin.core

import android.content.Context
import android.support.v7.widget.AppCompatImageView
import android.util.AttributeSet

open class KotlinImageview : AppCompatImageView{
    constructor(ctx : Context):this(ctx, null)
    constructor(ctx : Context, atr : AttributeSet?):super(ctx, atr, 0)
    constructor(ctx : Context, atr : AttributeSet?, style : Int):super(ctx, atr, style){
        //body
    }
}
```

Interface :

- It is Stateless and similar to Java 8
- It can declare abstract methods and/or method with implementation
- Difference between Abstract class and interface in Kotlin

OOP Concept

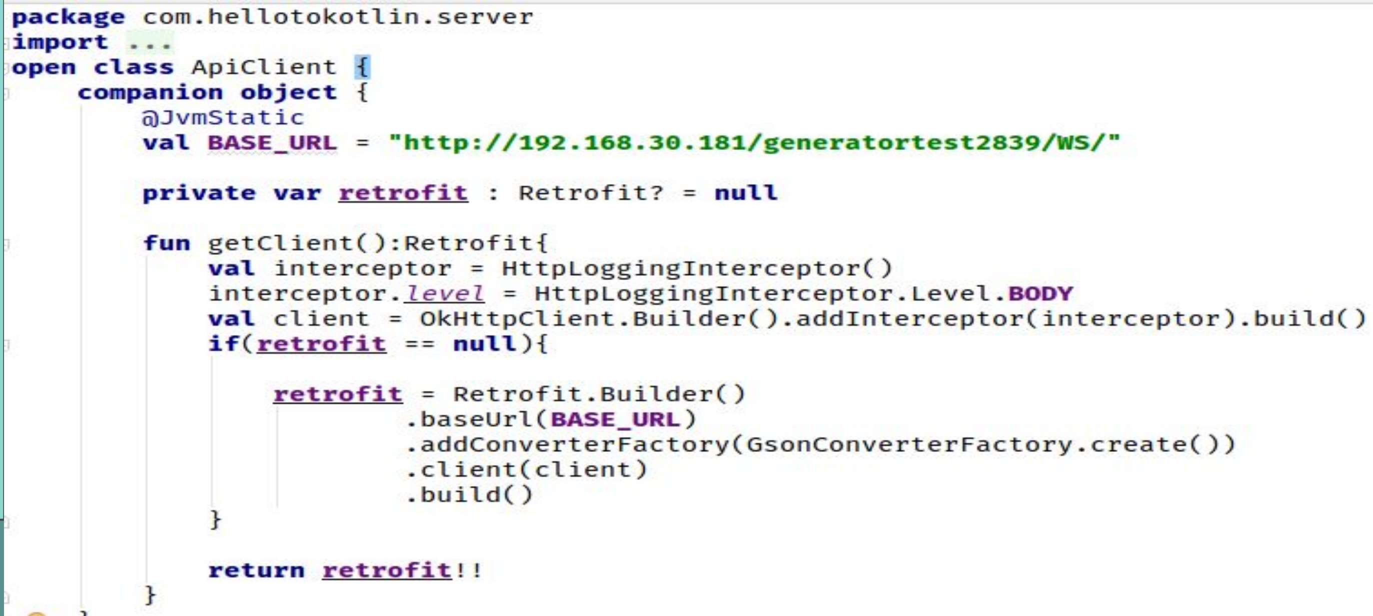


```
package com.hellotokotlin.core
open interface IBackStack{

   val a : String //= "I can not assing value here in" +
//              " Interface because interface don't know, How to store data"
//          get() = "Here you can save value because it is runtime"

    fun onBackPressed():Boolean{
        print(a)
        return false;
    }
    fun abstract() //see here I given method name as keyword
}
```

Companion Object:(Spend lot of time)

Static in Kotlin

- Companion object is initialized when class is loaded



```kotlin
package com.hellotokotlin.server
import ...
open class ApiClient {
    companion object {
        @JvmStatic
        val BASE_URL = "http://192.168.30.181/generatortest2839/WS/"

        private var retrofit : Retrofit? = null

        fun getClient():Retrofit{
            val interceptor = HttpLoggingInterceptor()
            interceptor.level = HttpLoggingInterceptor.Level.BODY
            val client = OkHttpClient.Builder().addInterceptor(interceptor).build()
            if(retrofit == null){

                retrofit = Retrofit.Builder()
                        .baseUrl(BASE_URL)
                        .addConverterFactory(GsonConverterFactory.create())
                        .client(client)
                        .build()
            }

            return retrofit!!
        }
```

Data class

Data Classes: Save a good bunch of lines of code

- We frequently create classes whose main purpose is to hold data.In Kotlin, this is called a data class and is marked as data

- It's a POJO complete with toString(), equals(), hashCode(), and copy(), and unlike in Java it won't take up 100 lines of code :)

These generally contain the same concepts every time:

-A constructor
-Fields to store data
-Getter and setter functions
-Copy(), ComponentX() methods
-hashCode(), equals() and toString() functions

```
data class Person(var name: String, var surname: String)
```

Smart cast

Smart cast object:

```
fun smartCast(v: View?) {
   if (v is ImageView) {
       v.setImageResource(R.drawable.ic_launcher_background)
   }
   if (v is TextView) {
       v.setText("Very Intelligent Kotlin that convert v to TextView")
   }
   when (v) {
       is ImageView -> v.setImageResource(R.drawable.ic_launcher_background)
       is TextView -> v.setText("Very Intelligent Kotlin that convert v to
TextView")
   }
}
```

Intuitive Equals

Intuitive Equals:

```
var p1 = PersonData("Hitesh")
var p2 = PersonData("Hitesh")

if (p1 == p2) //p1 and p2 are data class objects
   Log.i("PRINT_EQUAL", "p1 and p2 are equal")
else
   Log.e("PRINT_EQUAL", "p1 and p2 are not equal")

if (p1 === p2)
   Log.i("PRINT_EQUAL", "p1 and p2 are one object")
else
   Log.e("PRINT_EQUAL", "p1 and p2 are different object")


var map = linkedMapOf(p1 to 1, p2 to 2)
println("\nSize is ${map.size}")
```

Default

Arguments

Default Arguments:

```
fun defaultArgument(x: Int = 50, y: Int = 100): Int {
   return x + y
}

//call above function by below ways
defaultArgument()
defaultArgument(10)
defaultArgument(20, 30)
defaultArgument(y=30) //Named Arguments
```

When:

Very important "switch" case

```
fun whenExpression(x: Int) {
   when (x) {
       1 -> println("x is 1")
       2 -> println("x is 2")
       3, 4 -> println("x is 3 or 4")
       in 5..10 -> println("x is 5, 6, 7, 8, 9, or 10")
       else -> print("x is out of range")
   }
   val language: String = when {
       x == 0 -> "Java"
       x == 1 -> "Kotlin"
       else -> "PHP"
   }
   print("language = ${language}")
}
```

Operator

Overloading

Operator function:

```
open class Person(_name : String) {
   var name : String? = _name
   operator fun plus(v : Person?) = this.name?.length ?: 0 + (v?.name?.length
 ?: 0)
 }

 var p3 = Person("Hitesh Bhalala")
 var p4 = Person("Nirav Chauhan")

 println("Length of two person is ${p3 + p4}")
```

Extension functions

Extension function:

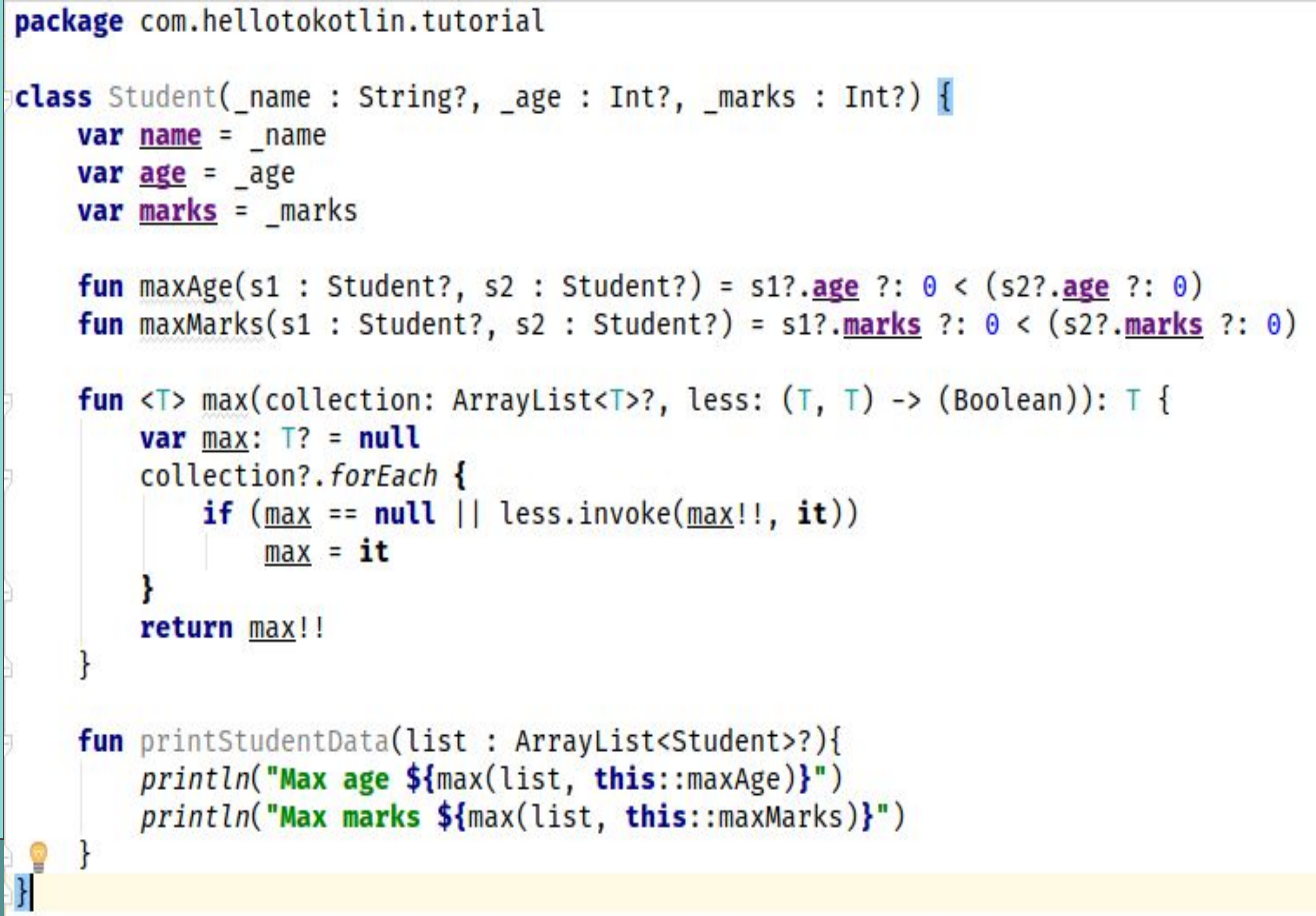
Extension functions are functions that, as the name implies, help us to extend the functionality of classes without having to touch their code Example : In java there is Collection.sort() method

```
//picasso library to load image
Picasso.with(imageView.context).load(url).into(imageView)
//Creating extension function inside class
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}

//Now our image loading line is very simple
imageView.loadUrl(url)
```

```
val Double.km : Double get() = this * 1000
val Double.mile : Double get() = this * 0.000621371192

fun conversion(){
  val meter = 25
  println("${meter} meter to KM is ${meter.toDouble().km}")
  println("${meter} meter to Mile is ${meter.toDouble().mile}")
}
```

Extension functions

Extension function:



```
package com.hellotokotlin.tutorial

class Student(_name : String?, _age : Int?, _marks : Int?) {
    var name = _name
    var age = _age
    var marks = _marks

    fun maxAge(s1 : Student?, s2 : Student?) = s1?.age ?: 0 < (s2?.age ?: 0)
    fun maxMarks(s1 : Student?, s2 : Student?) = s1?.marks ?: 0 < (s2?.marks ?: 0)

    fun <T> max(collection: ArrayList<T>?, less: (T, T) -> (Boolean)): T {
        var max: T? = null
        collection?.forEach {
            if (max == null || less.invoke(max!!, it))
                max = it
        }
        return max!!
    }

    fun printStudentData(list : ArrayList<Student>?){
        println("Max age ${max(list, this::maxAge)}")
        println("Max marks ${max(list, this::maxMarks)}")
    }
}
```

Finally we reach at NPE's

Null safety:

Kotlin's type system is aimed at eliminating the danger of null references from code

system distinguishes between references that can hold null (nullable references) and those that can not (non-null references)

```
var a: String = "abc"
a = null // compilation
error
a.length //no error
```

```
var b: String? = "abc"
b = null // ok
b?.length
```

1) **? Operator**

Non-null assertion (!!)

Non-null assertion (!!):

-This operator should use very rarely because it is sign of potential *NullPointerException*
-When you use this sign Kotlin code gives you **BIG** yellow warning that indicate that use only if you are 100% sure

```
var state = checkNotNull(name){
   Log.e("Error","Exception thrown here")
}
println("state.isEmpty() = ${state.isEmpty()}")
println("state.isBlank() = ${state.isBlank()}")
```

Thank you

By Hitesh Bhalala