
GDG
DevFest 2018
Goma
Google

GDG Goma

DevFest

Good Morning KOTLIN DROID 4

Michel Isamuna (Ghost),
GDG Mentor Afrique Central
@misamuna

Statically typed programming language for the JVM, Android and the browser

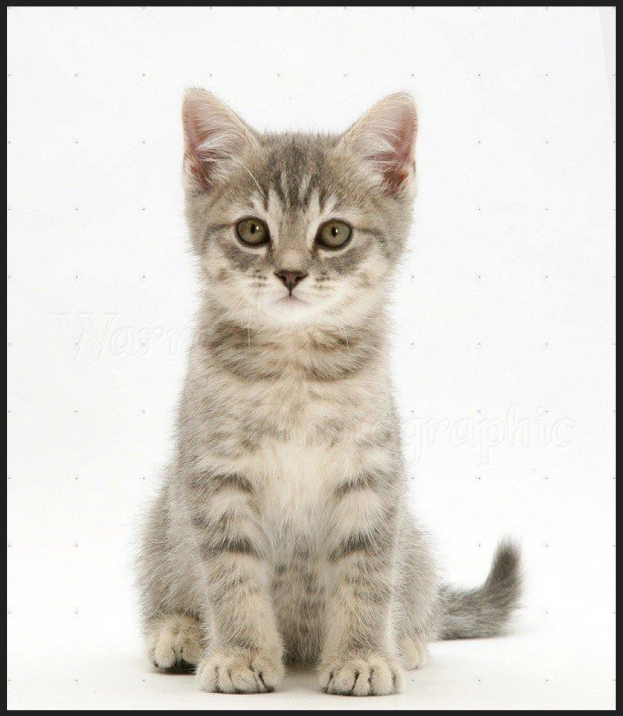
100% interoperable with Java™

@misamuna

What is the difference between Kotlin and Java?

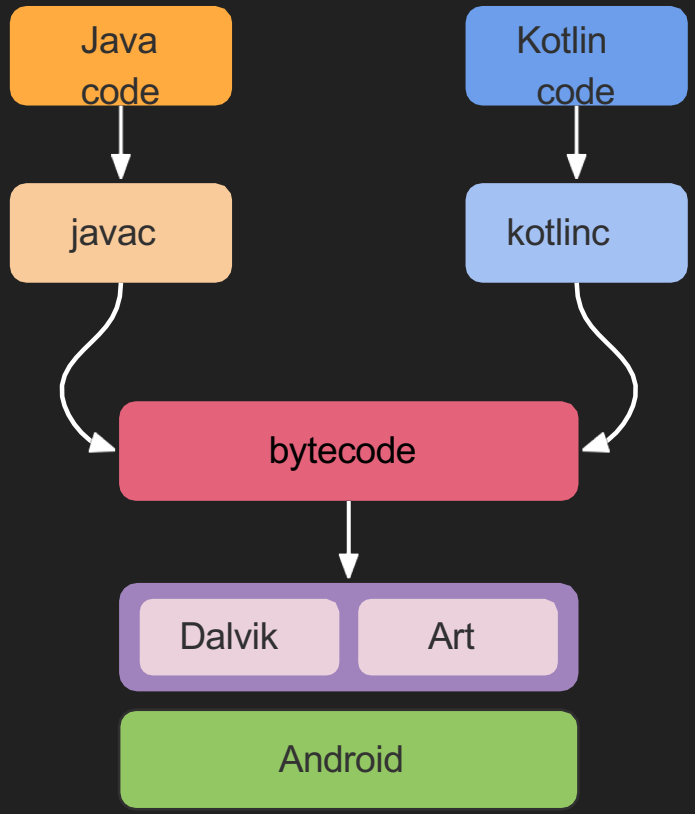
Non-technical explanation...

Kotlin Cat



Java Cat

@misamuna

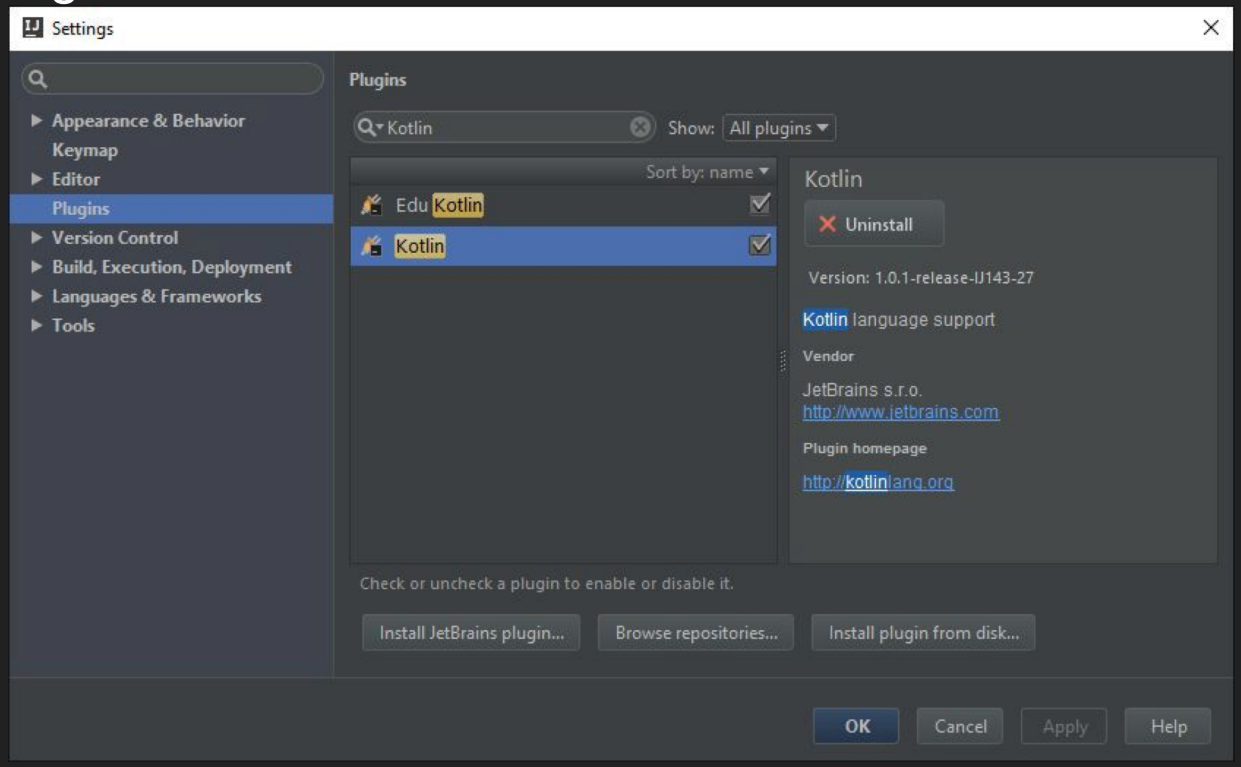

Java code
Kotlin code
javac
kotlinc
bytecode
Dalvik
Art
Android


GDG Goma
DevFest

@misamuna

How to add Kotlin to your project?

1. Add Kotlin Plugin



GDG Goma DevFest

How to add Kotlin to your project?

2. Declare dependencies

```
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

apply plugin: 'kotlin-android'

dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
}
```

How to add Kotlin to your project?

2. Declare dependencies

```
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

apply plugin: 'kotlin-android'

dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
}
```

What does it looks like?

Read only variable

```
val a: Int =  1
```

What does it look like?

```
val a: Int = 1  val
b = 1
```

Int type is inferred

What does it look like?

```
val a: Int = 1  val
b = 1
var c: Int
```

Mutable variable

What does it look like?

```
val a: Int = 1  val
b = 1
var c: Int  c
= 2
```

What does it look like?

```
val a: Int = 1  val
b = 1
var c: Int  c
= 2

val date = Date()
print(date.time)
```

Instances of class

What does it look like?

Class name

```
class Main {

    fun printSum(a: Int, b: Int) {
        print(a +  b)
    }

    fun returnSum(a: Int, b: Int): Int {  return a +
        b
    }
}
```

What does it look like?

```
class Main {

    fun printSum(a: Int, b: Int) {
        print(a +  b)
    }

    fun returnSum(a: Int, b: Int): Int {  return a +
        b
    }
}
```

What does it look like?

```
class Main {
        Function name
    fun printSum(a: Int, b: Int) {
            print(a +  b)
    }

    fun returnSum(a: Int, b: Int): Int {  return a +
            b
    }
}
```

What does it look like?

```
class Main {

    fun printSum(a: Int, b: Int) {
        print(a +  b)
    }

    fun returnSum(a: Int, b: Int): Int {  return a +
        b
    }
}
```

Function arguments

What does it look like?

```
class Main {

    fun printSum(a: Int, b: Int) {
        print(a + b)
    }

    fun returnSum(a: Int, b: Int): Int {  return a +
        b
    }
}
```

Return type

Kotlin Features

String templates
Properties
Lambdas
Data class
Smart cast
Null safety
Default values for function parameters
Lazy property
Extension Functions
Single-expression functions
When expression
let, apply, use, with
Collections
Kotlin Android Extensions Plugin
Anko

String templates

```
val query = "Kotlin"  val
language = "en"
val url = "https://www.google.com.ua/#q=$query&language=$language"
```

String templates

```
val query = "Kotlin"  val
language = "en"
val url = "https://www.google.com.ua/#q=$query&language=$language"
```

String templates

```
val query = "Kotlin"  val
language = "en"
val url = "https://www.google.com.ua/#q=$query&language=$language"
```

```
> https://www.google.com.ua/#q=Kotlin&language=en
```

Properties

```
class User {
    var name: String? = null  var
    age: Int? = null
}
```

Properties

```
class User {
    var name: String? = null  var
    age: Int? = null
}

val user = User()
user.name = "John"
user.age = 24

print("User name:${user.name}")
```

Properties

```
class User {
    var name: String? = null  var
    age: Int? = null
        set(value) {
            if (value >= 0)
                field = value
        }
}
```

Lambdas

Function that is not declared, but passed immediately as an expression.

Lambdas

```
button.setOnClickListener(object : View.OnClickListener {  override fun
      onClick(v: View?) {
            toast("Button clicked")
      }
})
```

Lambdas

An anonymous class

```
button.setOnClickListener(object : View.OnClickListener {  override fun
    onClick(v: View?) {
        toast("Button clicked")
    }
})
```

Lambdas

A lambda expression

```
button.setOnClickListener({ view -> toast("Button clicked") })
```

Lambdas

Parameter is optional

```
button.setOnClickListener({ view -> toast("Button clicked") })
```

Lambdas

```
button.setOnClickListener({ toast("Button clicked") })
```

Lambdas

Lambda is last parameter, can be moved out of the parentheses

```
button.setOnClickListener({ toast("Button clicked") })
```

Lambdas

```
button.setOnClickListener() { toast("Button clicked") }
```

Lambdas

Lambda is the only parameter, parentheses are optional

```
button.setOnClickListener() { toast("Button clicked") }
```

Lambdas

```
button.setOnClickListener { toast("Button clicked") }
```

Data class

What it does?

Nothing, just hold data

What it provides?

- getters, setters
- equals()/hashCode()
- toString() of the form "User(name=John, age=42)"
- copy() function

Data class

```
data class User(val name: String, val age: Int)
```

Data class

```
data class User(val name: String, val age: Int)

val user = User("Dmytro", 24)
print(user)
```

```
> User(name=Dmytro, age=24)
```

Smart cast

```
fun demo(x: Any) {
    if (x is String) {
        print(x.length)
    } else if (x is Int) {  print(x *
        x)
    }
}
```

Smart cast

```
fun demo(x: Any) {
    if (x is String) {
        print(x.length)
    } else if (x is Int) {  print(x *
        x)
    }
}
```

x is automatically cast to Int

Smart cast

```
fun demo(x: Any) {
    if (x is String) {
        print(x.length)
    } else if (x is Int) {  print(x *
        x)
    }
}
```

Null safety

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

Null safety

```
var name: String = null

override fun onCreate(savedInstanceState:                Bundle?) {
    super.onCreate(savedInstanceState)

    name = savedInstanceState.getString("name")
}
```

Null safety

Compilation error, non null variable

```
var name: String = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

        name = savedInstanceState.getString("name")
}
```

Null safety

```
var name: String? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    name = savedInstanceState.getString("name")
}
```

Null safety

```
var name: String? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    name = savedInstanceState.getString("name")
}
```

Compilation error, argument may be null

Null safety

```
var name: String? = null

override fun onCreate(savedInstanceState:                Bundle?) {
    super.onCreate(savedInstanceState)

    name = savedInstanceState?.getString("name")
}
```

Null safety

```
var name: String? = null

override fun onCreate(savedInstanceState:                Bundle?) {
    super.onCreate(savedInstanceState)

    name = savedInstanceState?.getString("name")
}
```

Default values for function parameters

Function parameters can have default values, which are used when a corresponding argument is omitted.

This allows for a reduced number of overloads compared to other languages.

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,  selection:
          String?,  selectionArgs :
          Array<String>?,  orderBy: String?):
          Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), null, null, "NAME")
```

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,
          selection: String? = null,  selectionArgs :
          Array<String>? = null,  orderBy: String? = null):
          Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), null, null, "NAME")
```

Default value

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,
          selection: String? = null,
          selectionArgs : Array<String>? = null,  orderBy:
          String? = null): Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), null, null, "NAME")
```

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,
          selection: String? = null,
          selectionArgs : Array<String>? = null,  orderBy:
          String? = null): Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), null, null, orderBy = "NAME")
```

Named argument

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,
          selection: String? = null,
          selectionArgs : Array<String>? = null,  orderBy:
          String? = null): Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), null, null, orderBy = "NAME")
```

Not need

Default values for function parameters

```
fun query(table: String,
          columns: Array<String>,
          selection: String? = null,
          selectionArgs : Array<String>? = null,  orderBy:
          String? = null): Cursor {
    ...
}

// function call
query("USERS", arrayOf("ID", "NAME"), orderBy = "NAME")
```

Lazy property

Value gets computed only upon first access.

Lazy property

```
val preference = getSharedPreferences("pref")

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val username = preference.getString("username")
}
```

Lazy property

Crash, require context

```
val preference = getSharedPreferences("pref")

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

        val username = preference.getString("username")
}
```

Lazy property

Won't be executed until the property is first used

```
val preference by lazy { getSharedPreferences("pref") }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val username = preference.getString("username")
}
```

Lazy property

```
val preference by lazy { getSharedPreferences("pref") }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val username = preference.getString("username")
}
```

Extension Functions

What it does?

Provides the ability to extend a class with new functionality without having to inherit from the class.

How?

Extensions do not actually modify classes they extend. By defining an extension, you do not insert new members into a class, but merely make new functions callable with the dot-notation on instances of this class.

Extension Functions

MainActivity.kt

```
imageView.loadUrl("http://....")
```

ViewExtensions.kt

```
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}
```

Extension Functions

MainActivity.kt

Extension function

```
imageView.loadUrl("http://....")
```

ViewExtensions.kt

```
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}
```

Extension Functions

MainActivity.kt

```
imageView.loadUrl("http://...")
```

ViewExtensions.kt

```
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}
```

Extension Functions

MainActivity.kt

imageView.loadUrl("http://...")

ViewExtensions.kt

Type binding extended

```
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}
```

Extension Functions

MainActivity.kt

```
imageView.loadUrl("http://....")
```

ViewExtensions.kt

```
fun ImageView.loadUrl(url: String) {
    Picasso.with(context).load(url).into(this)
}
```

Single-expression functions

When a function returns a single expression, the curly braces can be omitted and the body is specified after a = symbol.

Single-expression functions

```
fun createUrl(search: String): String {  return
    "www.google.com.ua/#q=$search"
}
```

Single-expression functions

```
fun createUrl(search: String): String {
    return "www.google.com.ua/#q=$search"
}
```

Function return single expression

Single-expression functions

Curly braces can be omitted

```
fun createUrl(search: String): String = "www.google.com.ua/#q=$search"
```

body is specified after a = symbol

Single-expression functions

Return type is optional

```
fun createUrl(search: String): String = "www.google.com.ua/#q=$search"
```

Single-expression functions

```
fun createUrl(search: String) = "www.google.com.ua/#q=$search"
```

When expression

When expression is similar to switch/case in Java, but far more powerful.

When expression

```
when (argument) {
    match1 -> fun1()
    match2 -> fun2()
    else -> fun3()
}
```

When expression

```
var result: Int = when (argument) {  match1
      -> fun1()
      match2 -> fun2()
      else -> fun3()
}
```

When expression

```
when (range) {
    in 1..10 -> print("x is in the range")
    !in 1..10 -> print("x is outside the range")  else ->
    print("none of the above")
}
```

When expression

```
fun onViewClicked(view: Any) {
    when (view) {
        is Button -> view.text = "Button"
        is CheckBox -> view.isChecked = true
    }
}
```

When expression

```
fun onViewClicked(view: Any) {
    when (view) {
        is Button -> view.text = "Button"  is
        CheckBox -> view.isChecked = true
    }
}
```

Smart cast

When expression

Can be placed after a = symbol

```
fun onViewClicked(view: Any) {
    when (view) {
        is Button -> view.text = "Button"
        is CheckBox -> view.isChecked = true
    }
}
```

When expression

```
fun onViewClicked(view: Any) = when (view) {  is
      Button -> view.text = "Button"
      is CheckBox -> view.isChecked = true  else
      -> print("none of the above")
}
```

let, apply, use, with

Higher-order functions - function that takes functions as parameters, or returns a function.

let (scope function)

```
Preferences.getUser().let {
    showUserName(it.name)
    showUserEmail(it.email)
}
```

let

Refers to user object

```
Preferences.getUser().let {
    showUserName(it.name)
    showUserEmail(it.email)
}
```

let

Variable visibility scope

```
Preferences.getUser().let {
    showUserName(it.name)
    showUserEmail(it.email)
}
```

let

Only execute if user is not null

```
Preferences.getUser()?.let {
    showUserName(it.name)
    showUserEmail(it.email)
}
```

let

```
Preferences.getUser()?.let {
    showUserName(it.name)
    showUserEmail(it.email)
}
```

use (try with resources function)

```
fun countUsers(): Long {
    val database = openDatabase()
    val result = database.count("users")
    database.close()
    return result
}
```

use

Automatically close database

```
fun countUsers() = openDatabase().use { it.count("users") }
```

use

Refers to database object

```
fun countUsers() = openDatabase().use { it.count("users") }
```

use

```
fun countUsers() = openDatabase().use { it.count("users") }
```

with

```
recyclerView.setHasFixedSize(true)
recyclerView.addItemDecoration(createDecorator())  recyclerView.layoutManager =
LinearLayoutManager(applicationContext)  recyclerView.adapter = myAdapter
```

with

```
with(recyclerView) {  recyclerView.setHasFixedSize(true)
    recyclerView.addItemDecoration(createDecorator())
    recyclerView.layoutManager = LinearLayoutManager(applicationContext)
    recyclerView.adapter = myAdapter
}
```

with

Not need

```
with(rec{  recyclerView.setHasFixedSize(true)
    recyclerView.addItemDecoration(createDecorator())
    recyclerView.layoutManager = LinearLayoutManager(applicationContext)
    recyclerView.adapter = myAdapter
}
```

with

```
with(recyclerView) {  setHasFixedSize(true)
    addItemDecoration(createDecorator())
    layoutManager = LinearLayoutManager(applicationContext)
    adapter = myAdapter
}
```

with

```
with(recyclerView) {  setHasFixedSize(true)
    addItemDecoration(createDecorator())
    layoutManager = LinearLayoutManager(applicationContext)
    adapter = myAdapter
}
```

apply

```
fun makeDir(path: String): File {  val
    result: File = File(path)
    result.mkdirs()
    return result
}
```

apply

```
fun makeDir(path: String) = File(path).apply { mkdirs() }
```

Collections

```
val numbers: MutableList<Int> = mutableListOf(1, 2, 3)
numbers.add(1)

val numbers2: List<Int> = listOf(1, 2, 3)
numbers2.add(1)
```

No such method

Collections

```
mutableListOf(1, null, 2, null, 3, 4, 5, 6, 7, 8, 9)
        .filterNotNull()
        .filter { it % 2 == 0 }
        .sortedDescending()
```

```
> [1, null, 2, null, 3, 4, 5, 6, 7, 8, 9]
```

Collections

```
mutableListOf(1, null, 2, null, 3, 4, 5, 6, 7, 8, 9)
        .filterNotNull()
        .filter { it % 2 == 0 }
        .sortedDescending()
```

```
> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Collections

```
mutableListOf(1, null, 2, null, 3, 4, 5, 6, 7, 8, 9)
        .filterNotNull()
        .filter { it % 2 == 0 }
        .sortedDescending()
```

```
> [2, 4, 6, 8]
```

Collections

```
mutableListOf(1, null, 2, null, 3, 4, 5, 6, 7, 8, 9)
        .filterNotNull()
        .filter { it % 2 == 0 }
        .sortedDescending()
```

```
> [8, 6, 4, 2]
```

Collections

getOrElse()
find() filter()
filterNot()
filterNotNull()
flatMap() take()
takeLast()
sortBy()
sortByDescending()

groupBy()
map()
mapNotNull()
all()
any()
maxBy()
minBy()
minWith()
sumBy()
zip()
...

Kotlin Android Extensions Plugin

What it does?

Provides reference to all layout views (which have id’s) with single line of code.

How?

Adds a hidden caching function and a field inside each Kotlin Activity.

Kotlin Android Extensions Plugin

```
buildscript {
        dependencies {
            classpath "org.jetbrains.kotlin:kotlin-android-extensions:$kotlin_version"
    }
}

apply plugin: 'kotlin-android-extensions'
```

Kotlin Android Extensions Plugin

```
// R.layout.activity_main
import kotlinx.android.synthetic.main.activity_main.*

public class MyActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        txtTitle.setText("Hello, Kotlin!")
        btnHello.setOnClickListener {...}
    }
}
```

Kotlin Android Extensions Plugin

```
// R.layout.activity_main
import kotlinx.android.synthetic.main.activity_main.*

public class MyActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        txtTitle.setText("Hello, Kotlin!")  btnHello.setOnClickListener
        {...}
    }
}
```

Kotlin Android Extensions Plugin

```
// R.layout.activity_main
import kotlinx.android.synthetic.main.activity_main.*

public class MyActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        txtTitle.setText("Hello, Kotlin!")
        btnHello.setOnClickListener {...}
    }
}
```

Instead of findView(R.id.txtTitle)

Kotlin Android Extensions Plugin

```
// R.layout.activity_main
import kotlinx.android.synthetic.main.activity_main.*

public class MyActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        txtTitle.setText("Hello, Kotlin!")  btnHello.setOnClickListener
        {...}
    }
}
```

Anko

What it does?

Kotlin library from JetBrains which provide API to make Android application development faster and easier.

Anko

```
dependencies {
    compile "org.jetbrains.anko:anko-common:$anko_version"  compile
    "org.jetbrains.anko:anko-sqlite:$anko_version"
}
```

Anko

```
toast("Hi there!")
toast(R.string.message)
longToast("Wow, such a duration")
```

Anko

```
alert("Santa", "You were a good boy?") {  positiveButton("Yes") { toast("You
    are now in GOOD list") }  negativeButton("No") { toast("You are now in
    BAD list") }
}.show()
```

Anko

```
val countries = listOf("Ukraine", "USA", "UK",)
selector("Where are you from?", countries) { i ->
    toast("So you're living in ${countries[i]}, right?")
}
```

Anko

```
async() {
    // long background task
    uiThread {
        // won't be executed if isFinishing() is true  toolbar.title =
        "Done"
    }
}
```

What's coming next?

Coroutines
Type aliases
Gradle Script Kotlin

- no more "callback hell"
- alternative names for existing type
- writing build scripts in Kotlin

Resources

www.kotlinlang.org/docs/reference www.try.kotlinlang.org
www.antonioleiva.com/kotlin

- documentation
- online compiler + exercises
- kotlin blog (by Antonio Leiva)

GDG Goma

DevFest

Thank you!

Michel Isamuna (Ghost),
GDG Mentor Afrique Central
@misamuna

GDG Goma

DevFest

Q&A