2 years without Java or reload Android development with Kotlin

KirillRozov@EPAM

Who am I?

- Team Lead in EPAM
- More than 6 years in Android development
- Kotlin Evangelist
- Co-organizer GDG Minsk, Android Academy
- Active member of Belarus Kotlin User Group (BKUG)
- Core team of Mobile People BY


krl.rozov@gmail.com


krlrozov

In the begging was…

September, 2008 — **Android 1.0 Release**

Java SE 6*, Eclipse, ADT Plugin, Android Tools

July, 2011 — **Java SE 7 Release**

Diamond operator, Strings in Switch, Multi Catch, Try-With-Resource, NIO 2.0

April 11, 2012 — **My first workday as Android Developer**

October, 2013 — **Android 4.4 Release**

Java SE 7 Support*

March, 2014 — **Java SE 8 Release**

In the begging was…

March, 2014 — Java SE 8 Release

Functional programming, Lambda Expressions, Collections Stream API, Java Time API

August, 2016 — Android 7.0 Release

Native Java SE 8 Support*

October, 2016 — Android 7.1 Release

Added NIO 2.0 from Java 7

August, 2017 — Android 8.0 Release

Added java.time from Java 8 and Invoke Dynamic from Java 7

April, 2017 — Desugar Release

September, 2017 — Java SE 9 Release

September, 2018 — Java SE 11 Release

Java disappointments on Android

- “The Billion Dollar Mistake”
- *WhatEver*Utils classes
- Boilerplate code: data classes, class delegation and etc.
- Method overloading
- Named arguments name
- No immutable collections


TVland
Classic

Kotlin

First Release - February 2015

Basic Features

- Null Safety -“Billion Dollar Mistake” solved
- Extension Functions - *Utils must die
- Data Classes - no more POJO boilerplate code
- Objects - no more singleton boilerplate code
- Named arguments - *sample(true, false, false, true)* is not so complicated
- Collection functional operators: *map*, *filter*, *forEach* and etc.
- Java <-> Kotlin interoperability

High Order Functions

High Order Function

```
fun <T> Iterable<T>.doOnEach action: (T) -> Unit) {
    …
}
```

High Order Function

```
fun <T> Iterable<T>.doOnEach(action: (T) -> Unit) {
    …
}
```

```
items.doOnEach { println(it) }
```

High Order Function

```
fun <T> Iterable<T>.doOnEach(action: (T) -> Unit) {
    for(item in this) {
        action(item)
    }
}
```

```
inline fun <T> Iterable<T>.doOnEach(action: (T) -> Unit) {
    for(item in this) {
        action(item)
    }
}
```

High Order Function

```
database.beginTransaction()
try {
    // insert data
    database.setTransactionSuccessful()
} finally {
    database.endTransaction()
}
```

High Order Function

```
database.transaction {
    // insert data
}
```

```
fun SQLiteDatabase.transaction(action: SQLiteDatabase.() -> Unit) {
    beginTransaction()
    try {
        action()
        setTransactionSuccessful()
    } finally {
        endTransaction()
    }
}
```

Foundation

Architecture

Behaviour

UI

Android KTX

Module (artifact)	Version	Package
androidx.core:core-ktx	1.0.0	See all the core packages below.
androidx.fragment:fragment-ktx	1.0.0	androidx.fragment.app
androidx.palette:palette-ktx	1.0.0	androidx.palette.graphics
androidx.sqlite:sqlite-ktx	2.0.0	androidx.sqlite.db
androidx.collection:collection-ktx	1.0.0	androidx.collection
androidx.lifecycle:lifecycle-viewmodel-ktx	2.0.0	androidx.lifecycle
androidx.lifecycle:lifecycle-reactivestreams-ktx	2.0.0	androidx.lifecycle
android.arch.navigation:navigation-common-ktx	1.0.0-alpha06	androidx.navigation
android.arch.navigation:navigation-fragment-ktx	1.0.0-alpha06	androidx.navigation.fragment
android.arch.navigation:navigation-runtime-ktx	1.0.0-alpha06	androidx.navigation
android.arch.navigation:navigation-testing-ktx	1.0.0-alpha06	androidx.navigation.testing
android.arch.navigation:navigation-ui-ktx	1.0.0-alpha06	androidx.navigation.ui
android.arch.work:work-runtime-ktx	1.0.0-alpha10	androidx.work.ktx

Warning! High order functions usage

- Don’t use function as Listeners
- Don’t inline big functions
- Function param must be last

Fragment Creation

Fragment Creation

```
class UserFragment : Fragment() {

    companion object {

        private const val ARG_USER_ID = "userId"

        fun newInstance(userId: String): UserFragment {
            return UserFragment().apply {
                Bundle(1).apply { putString(ARG_USER_ID, userId) }
            }
        }
    }
}
```

Fragment Creation

```
UserFragment.newInstance("userId")
```

```
UserFragment("userId")
```

Fragment Creation

```
class UserFragment : Fragment() {

  companion object {

    private const val ARG_USER_ID = "userId"

    operator fun invoke(userId: String): UserFragment {
        return UserFragment().apply {
            arguments = Bundle(1).apply {
                putString(ARG_USER_ID, userId)
            }
        }
    }
  }
}
```

Fragment Creation

```
class UserFragment : Fragment()

private const val ARG_USER_ID = "userId"

fun UserFragment(userId: String): UserFragment {
    return UserFragment().apply {
        ARG_USER_ID, userId = Bundle(1).apply {
            putString(ARG_USER_ID, userId)
        }
    }
}
```

Fragment Creation

```
class UserFragment : Fragment()

private const val ARG_USER_ID = "userId"

fun UserFragment(userId: String): UserFragment {
    return UserFragment().apply {
        arguments = bundleOf(ARG_USER_ID to userId)
    }
}
```

Shared Preferences

Shared Preferences

```
class AppPreferences(context: Context) {

    private val prefs = context.getSharedPreferences(APP_PREFS, Context.MODE_PRIVATE)

    fun getLastAddress(): String? = prefs.getString(ADDRESS, null)

    fun setLastAddress(address: String?) {
        prefs.edit().putString(LAST_ADDRESS, address).apply()
    }
}
```

Shared Preferences

```
class AppPreferences(private val context: Context) {

    private val prefs = context.getSharedPreferences(APP_PREFS, Context.MODE_PRIVATE)

    var lastAddress: String?
        get() = prefs.getString(LAST_ADDRESS, null)
        set(value) = prefs.edit().putString(ADDRESS, value).apply()

    var useDefaultAddress: Boolean
        get() = prefs.getBoolean(USE_DEFAULT_ADDRESS)
        set(value) = prefs.edit().putBoolean(USE_DEFAULT_ADDRESS, value).apply()
}
```

Property Delegate

```
class StringPreferenceDelegate(
        private val sharedPreferences: SharedPreferences,
        private val prefName: String
) : ReadWriteProperty<Any, String?> {

    override fun getValue(thisRef: Any, property: KProperty<*>): String? {
        sharedPreferences.getString(prefName, null)
    }

    override fun setValue(thisRef: Any, property: KProperty<*>, value: String?) {
        sharedPreferences.edit().putString(prefName, value).apply()
    }
}
```

Property Delegate

```
class StringPreferenceDelegate(
        private val sharedPreferences: SharedPreferences,
        private val prefName: String
) : ReadWriteProperty<Any, String?> {

    override fun getValue(thisRef: Any, property: KProperty<*>): String? {
        sharedPreferences.getString(prefName, null)
    }

    override fun setValue(thisRef: Any, property: KProperty<*>, value: String?) {
        sharedPreferences.edit().putString(prefName, value).apply()
    }
}
```

Property Delegate

```
class StringPreferenceDelegate(
        private val sharedPreferences: SharedPreferences,
        private val prefName: String
) : ReadWriteProperty<Any, String?> {

    override fun getValue(thisRef: Any, property: KProperty<*>): String? {
        sharedPreferences.getString(prefName, null)
    }

    override fun setValue(thisRef: Any, property: KProperty<*>, value: String?) {
        sharedPreferences.edit().putString(prefName, value).apply()
    }
}
```

Property Delegate

```
class StringPreferenceDelegate(
        private val sharedPreferences: SharedPreferences,
        private val prefName: String
) : ReadWriteProperty<Any, String?> {

    override fun getValue(thisRef: Any, property: KProperty<*>): String? {
        sharedPreferences.getString(prefName, null)
    }

    override fun setValue(thisRef: Any, property: KProperty<*>, value: String?) {
        sharedPreferences.edit().putString(prefName, value).apply()
    }
}
```

```
fun SharedPreferences.stringPref(prefName: String): ReadWriteProperty<Any, String?> {
    return StringPreferenceDelegate(this, prefName)
}
```

Shared Preferences

```
class AppPreferences(context: Context) {

    private val prefs = context.getSharedPreferences(APP_PREFS, Context.MODE_PRIVATE)

    val lastAddress by prefs.stringPref(LAST_ADDRESS)
}
```

Android Parcel

Parcels

```java
public class Person {

    private final String mFirstName;
    private final String mSecondName;

    public Person(@NonNull String firstName, @NonNull String secondName) {
        mFirstName = firstName;
        mSecondName = secondName;
    }

    @NonNull
    public String getFirstName() {
        return mFirstName;
    }

    @NonNull
    public String getSecondName() {
        return mSecondName;
    }
}
```

Parcels

```
public class Person implements Parcelable {

    private final String mFirstName;
    private final String mSecondName;

    public Person(@NonNull String firstName, @NonNull String secondName) {
        …
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(@NonNull Parcel dest, int flags) {
        …
    }
}
```

Parcels

```
public class Person implements Parcelable {

    public Person(@NonNull Parcel source) {
        …
    }

     @Override
     public int describeContents() {
         return 0;
     }

     @Override
     public void writeToParcel(@NonNull Parcel dest, int flags) {
         dest.writeString(mFirstName);
         dest.writeString(mSecondName);
     }


    public static final Parcelable.Creator<Person> CREATOR = new Creator<Person>() {

        @Override
        public Person createFromParcel(@NonNull Parcel source) { return new Person(source); }

        @Override
        public Person[] newArray(int size) { return new Person[size]; }
    };

}
```

Parcels

```
class Person(val firstName: String, val secondName: String) : Parcelable {

    constructor(source: Parcel): this(…)

    override fun describeContents() = 0

    override fun writeToParcel(dest: Parcel, flags: Int) { … }

    companion object {

        @JvmField
        val CREATOR: Parcelable.Creator<Person> =
                object : Parcelable.Creator<Person> {

            override fun createFromParcel(source: Parcel)= Person(source)

            override fun newArray(size: Int): Array<Person?> = arrayOfNulls(size)
        }
    }
}
```

Parcels

```
inline fun <reified T : Any> creatorOf(
      crossinline creator: (Parcel) -> T
): Parcelable.Creator<T> {

    return object : Parcelable.Creator<T> {

        override fun createFromParcel(src: Parcel) = creator(src)

        override fun newArray(size: Int) = arrayOfNulls<T>(size)
    }
}
```

Parcels

```
class PersonKotlin(val firstName: String, val secondName: String) : Parcelable {

    constructor(source: Parcel): this(…)

    override fun describeContents() = 0

    override fun writeToParcel(dest: Parcel, flags: Int) { … }

    companion object {

        @JvmField
        val CREATOR = creatorOf { PersonKotlin(it) }
    }
}
```

Parcels

```
@Parcelize
class PersonKotlin(val firstName: String, val secondName: String) : Parcelable
```

Architecture Components

Arch Components - View Model

```
class UserViewModel(val userId: String, private val repo: Repo) {

    val user: LiveData<User> = repo.getUser(userId)
}
```

Arch Components - View Model

```
class Factory(
        private val userId: String,
        private val repo: Repo
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass == UserViewModel::class.java) {
            return UserViewModel(userId, repo) as T
        }

        error("Can't create ${modelClass.simpleName}")
    }
}
```

Arch Components - View Model

```
class UserFragment : Fragment() {

    private val userViewModelFactory: UserViewModel.Factory = …

    private lateinit var userViewModel: UserViewModel

    override fun onAttach(context: Context) {
        super.onAttach(context)
        …
        userViewModel =
            ViewModelProviders.of(this, userViewModelFactory).get(UserViewModel::class.java)
    }
}
```

Arch Components - View Model

```
class UserFragment : Fragment() {

    private val userViewModelFactory: UserViewModel.Factory = …

    private lateinit var userViewModel: UserViewModel

    override fun onAttach(context: Context) {
        super.onAttach(context)
        …
        userViewModel =
            ViewModelProviders.of(this, userViewModelFactory).get(UserViewModel::class.java)
    }
}
```

Arch Components - View Model

```
fun Fragment.viewModelProviderOf(factory: ViewModelProvider.Factory): ViewModelProvider {
    return ViewModelProviders.of(this, factory)
}
```

Arch Components - View Model

```
userViewModel =
            ViewModelProviders.of(this, userViewModelFactory).get(UserViewModel::class.java)
```

```
userViewModel =
            viewModelProvidersOf.of(userViewModelFactory).get(UserViewModel::class.java)
```

Arch Components - View Model

```
inline fun <reified VM : ViewModel> ViewModelProvider.get(): VM {
    return get(VM::class.java)
}
```

Arch Components - View Model

```
userViewModel =
            ViewModelProviders.of(this, userViewModelFactory).get(UserViewModel::class.java)
```

```
userViewModel = viewModelProvidersOf.of(userViewModelFactory).get(UserViewModel::class.java)
```

```
userViewModel = viewModelProvidersOf.of(userViewModelFactory).get()
```

Arch Components - View Model

```
inline fun <reified VM : ViewModel> Fragment.getViewModel(
        factory: ViewModelProvider.Factory
): VM {
    return viewModelProviderOf(factory).get()
}
```

Arch Components - View Model

```
class UserFragment : Fragment() {

    private val userViewModelFactory: UserViewModel.Factory = …

    private lateinit var userViewModel: UserViewModel

    override fun onAttach(context: Context) {
        super.onAttach(context)
        …
        userViewModel = getViewModel(userViewModelFactory)
    }
}
```

Async Requests

Async Requests

```
interface PostService {

    fun queryPostIndex(address: String, callback: Callback<String>)
}
```

```
interface Callback<T> {

    fun onSuccess(data: T)

    fun onError(exception: Throwable)
}
```

Async Requests

```
class AddressViewModel(private val service: PostService) : ViewModel() {

    fun queryIndex(address: String) {
        service.queryPostIndex(address, object : Callback<String> {

            override fun onSuccess(data: String) {
                // TODO Show results
            }

            override fun onError(exception: Throwable) {
                // TODO Show error
            }
        })
    }
}
```

Callback hell

```
// First, we must get user by id
CallEndpoint("api/getidbyusername/hotcakes", function(result) {
    CallEndpoint("api/getfollowersbyid/" + result.userID, function(result) {
        CallEndpoint("api/someothercall/" + result.followers, function(result) {
            CallEndpoint("api/someothercall/" + result, function(result) {
                CallEndpoint("api/someothercall/" + result, function(result) {
                    // Ahhhhhhh... but you didn't believe you'd end up here
                });
            });
        });
    });
}); // Always use semicolons; at work and at home.
```

Coroutines

```
class AddressViewModel(private val service: PostService) : ViewModel() {

    fun queryIndex(address: String) {
        launch {
            try {
                val postIndex = service.getPostIndex(address).await()
                // TODO Show result
            } catch (e: Exception) {
                // TODO Show error
            }
        }
    }
}
```

Coroutines

```
class AddressViewModel(private val service: PostService) : ViewModel() {

    fun queryIndex(address: String) {
        launch {
            try {
                val postIndex = service.getPostIndex(address).await()
                // TODO Show result
            } catch (e: Exception) {
                // TODO Show error
            }
        }
    }
}
```

Coroutines

```
class AddressViewModel(private val service: PostService) : ViewModel() {

    fun queryIndex(address: String) {
        launch {
            try {
                val postIndex = service.getPostIndex(address).await()
                // TODO Show result
            } catch (e: Exception) {
                // TODO Show error
            }
        }
    }
}
```


WORK IN PROGRESS
BREAKING THROUGH
VISITING THE DOC
Go see your reporter
(OPTIONAL) CORPORATE INVOLVEMENT
Go meet with the agent

Tooling

Code style and quality checker

- **ktlint**
- Android Lint
- Detekt
- SonarLint Kotlin

Frameworks

- Gradle Kotlin DSL
- Koin - Dependencies injection on pure Kotlin
- Ktor - building asynchronous servers and clients in connected systems

Testing

- Spek - Specification framework, more readable test
- mockk - Mocks on Kotlin
- JaCoCo Kotlin support

Kotlin History

February, 2015 — Kotlin 1.0 Release

Kotlin/JVM

March, 2017 — Kotlin 1.1 Release

Coroutines, Kotlin/JS

March, 2017 — Kotlin/Native Announcement

Kotlin without VM

May, 2017 — Kotlin is a first-class language for Android

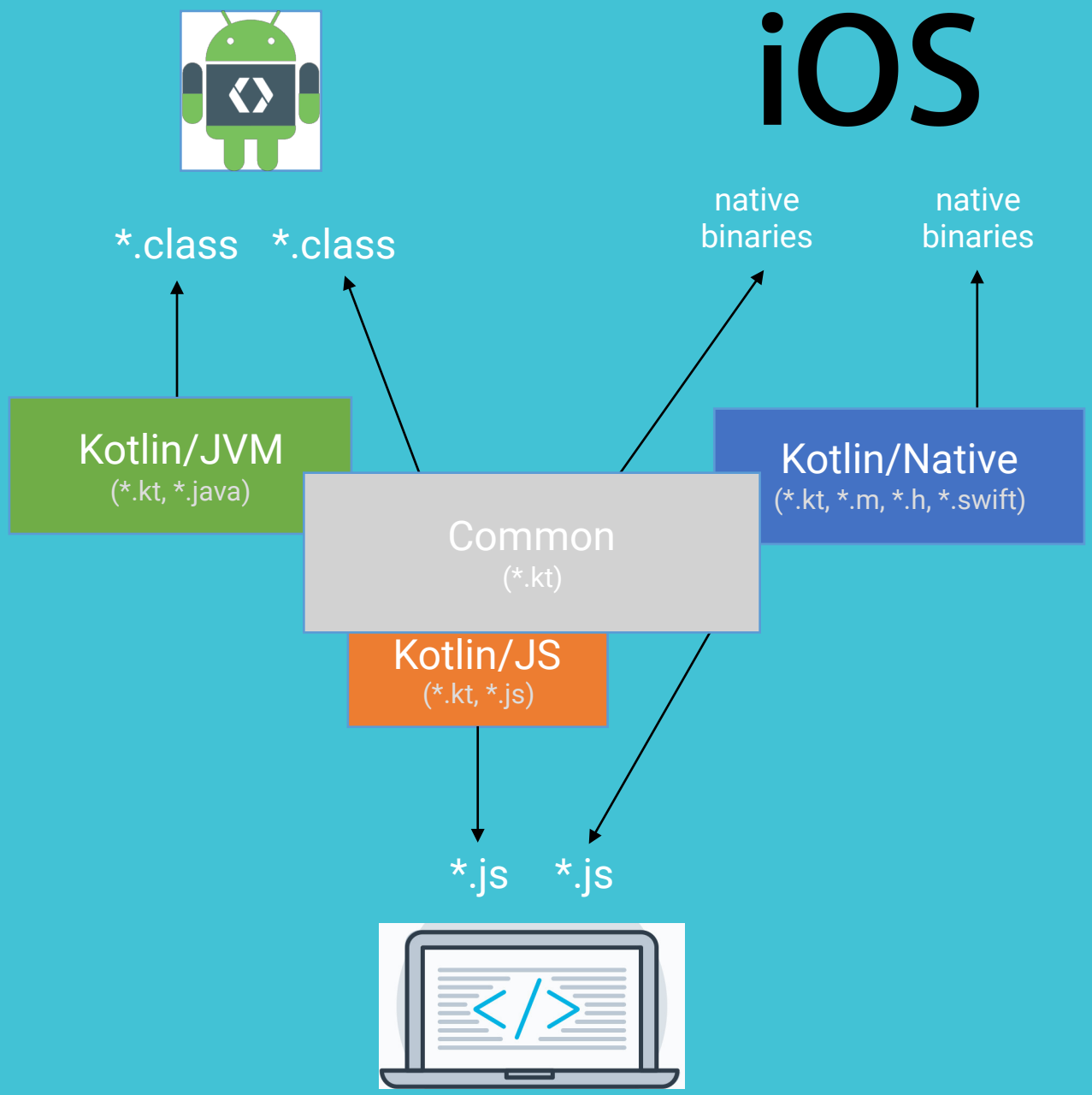
December, 2017 — Kotlin 1.2 Release

Multiplatform Projects (Experimental)

Kotlin Multiplatform Project

Cross-platform Frameworks

- ReactNative
- Xamarin
- Apache Cordova


iOS
*.class
*.class
native binaries
native binaries
Kotlin/JVM
(*.kt, *.java)
Common
(*.kt)
Kotlin/Native
(*.kt, *.m, *.h, *.swift)
Kotlin/JS
(*.kt, *.js)
*.js
*.js

Kotlin History

February, 2015 — **Kotlin 1.0 Release**

Kotlin/JVM

March, 2017 — **Kotlin 1.1 Release**

Coroutines, Kotlin/JS

March, 2017 — **Kotlin/Native Announcement**

Kotlin without VM

May, 2017 — **Kotlin is a first-class language for Android**

December, 2017 — **Kotlin 1.2 Release**

Multiplatform Projects (Experimental)

soon — **Kotlin 1.3 Release**

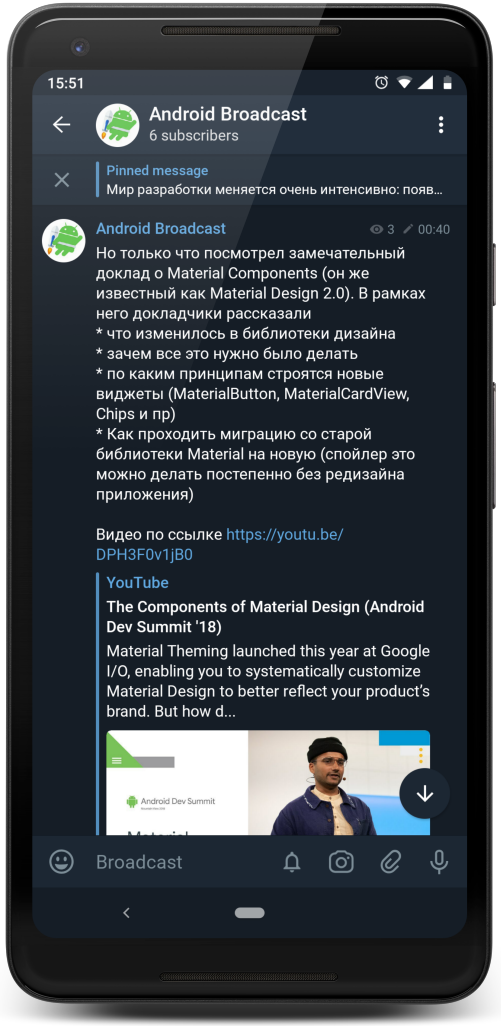
Stable Coroutines, Inline classes, Contracts, kotlinx.serialization, Unsigned types

Work in progress

- Improve compilation speed
- Kotlin Annotations Processing (kapt)
- Kapt incremental compilation support
- IDE possibilities

t.me/**android_broadcast**

High quality news for Android Devs




krl.rozov@gmail.com

krlrozov

Thank You
and Have fun!