



Coroutines

1

asynchrónnosť

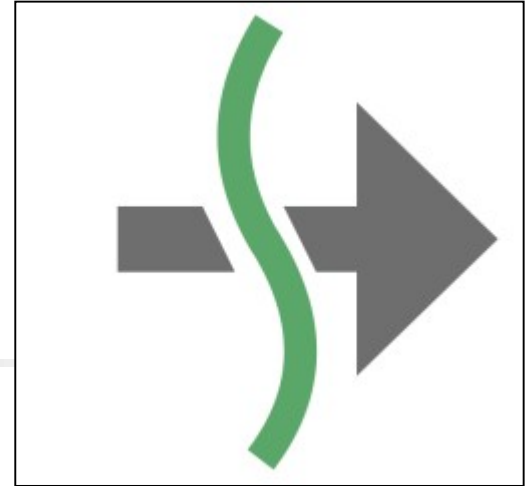
Peter Borovanský
KAI, I-18

MS-Teams: [2sf3ph4](#), [List](#), [github](#)

borovan 'at' ii.fmph.uniba.sk

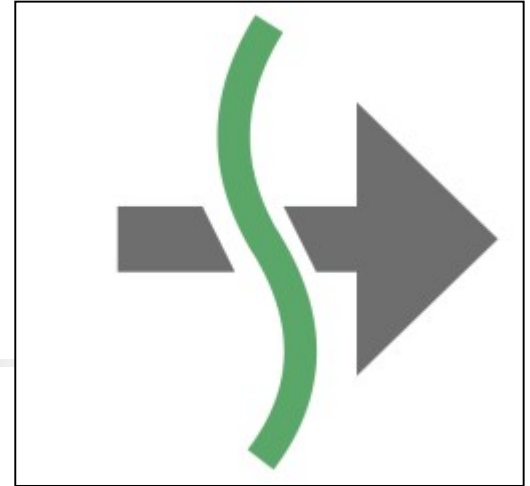
asi nabudúce	AsyncTask	asi na Cvičení
	Retrofit	
	RoomDB	
Coroutines		
■ channel		
■ flow		
■ Shared state		
■ Atomická premenná		
■ Prepínanie kontextov		
■ mutex		

Asynchronnosť



- je vážny problém
 - ako vykonávať niečo, čo môže dlho trvať, a event.
 - potrebujeme výsledok tohoto procesu pre ďalší svoj výpočet
- v rôznych jazykoch sa rieši rôzne
 - Javascript: callback vedie k tzv. callback hell
 - Java: Thread, FutureTask, RxJava
 - GO: go rutiny, kanály, ...
 - Android: AsyncTask (donedávna, ale dnes už je AsyncTask Deprecated)
- v Kotline od verzie 1.3 existuje koncept coroutiny
 - nie je to len knižnica
 - ale súčasť jazyka ([suspend](#))
- lepšie môžeme pochopiť koncept aj extra, bez Android environmentu
 - IntelliJ – projekt Coroutines1 obsahuje ~30 malých gradujúcich kotlin-coroutine príkladov

Asynchronnosť



- je vážny problém
 - ako vykonávať niečo, čo môže dlho trvať, a event.
 - potrebujeme výsledok tohoto procesu pre ďalší svoj výpočet
- súvisiace problémy - ako zdieľať dáta medzi konkurentne bežiacimi kódmi
 - posilať agresívne – zodpovedá koncept kanálu (**Channel<T>**)
 - do značnej miery zodpovedá jazyku GO, asi aj tam sa inšpirovali
 - je ale dravý/eager/**hot** – to čo do neho napíšete, to sa dá prečítať
 - posilať lenivo – zodpovedá koncept toku (**Flow<T>**)
 - do značnej miery zodpovedá generátorom z jazyka Python, resp. lazy listom z Haskellu
 - je lenivý/lazy/**cold** – začne sa do neho písať, len až sa niekto zaujíma o hodnoty

Callback je cesta do pekla ☺

čo je callback ?

```
fun getValueAsync(onCompletion: (R) -> Unit) {  
    val result = getValue() // toto trvá...  
    onCompletion(result)  
}
```

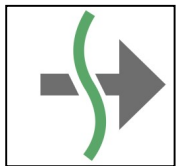
```
getValueAsync() {  
    result -> print(result)  
}
```

čo je callback hell ?

```
1 // Callback Hell  
2  
3  
4 a(function (resultsFromA) {  
5     b(resultsFromA, function (resultsFromB) {  
6         c(resultsFromB, function (resultsFromC) {  
7             d(resultsFromC, function (resultsFromD) {  
8                 e(resultsFromD, function (resultsFromE) {  
9                     f(resultsFromE, function (resultsFromF) {  
10                        console.log(resultsFromF);  
11                    })  
12                })  
13            })  
14        })  
15    })  
16 });  
17
```

Kotlin suspend function

podpora v jazyku



```
suspend fun getValue(): R {  
    return getValueAsync() // operácia trvá dlho  
}
```

```
launch { // coroutine scope  
    val res = getValue()  
    print(res)  
}
```

v coroutine scope (oblasť corutiny) môžeme volať iné **suspend** funkcie

```
launch {  
    val res = getValue() // trvá..  
    val nextRes = getNextValue(res)  
    print(nextRes)  
}
```

aj sekvenciu takých funkcií

Corutiny umožňujú písať

- elegantne,
- asynchrónny kód,
- bez vnárania
- bez callback hell



Corutina

`suspend fun` – je funkcia, ktorej výpočet môže dlho trvať z akýchkoľvek dôvodov. Taká funkcia NESMIE byť vykonávaná v hlavnom vlákne aplikácie, inak task manager zavrie aplikáciu, ktorá nereaguje na GUI eventy

Taká funkcia/výpočet sa púšťa v tzv. coroutine scope.

Výpočet corutiny v scope

- môže trvať (I/O, DB, NETWORK, scientific computation, simulácia čohosi)
- môže byť pozastavený bez toho, aby sa to dotklo hlavného vlákna
- môžu bežať konkurentne vo vláknach

```
launch {  
    val res = getValue()  
    val nextRes = getNextValue(res)  
    print(nextRes)  
}
```

```
launch {  
    val res = getValue()  
    val nextRes = getNextValue(res)  
    print(nextRes)  
}
```



Corutina

- je odľahčené vlákno <https://kotlinlang.org/docs/reference/coroutines/basics.html#coroutines-are-light-weight>
- non-preemptive multitasking
- 1958 zaviedli ich Donald Knuth a Melvin Conway, ale predbehli dobu
- vyskytujú sa v iných jazykoch, C#, javascript, continuation-passing style
- ale nemajú podporu jazyka

`suspend` je modifikátor funkcie, ktorá sa vykonávaná v corutine, a môže byť pozdržaná

`await()` je čaká na hodnotu výpočtu bez blokovania corutiny.

```
fun main() = runBlocking {  
    repeat(100_000) { // launch 10^5 coroutines  
        launch {  
            delay(5000L)  
            print(".")  
        }  
    }  
}
```

Hádanka 1

neobsahuje corutinu ale stream (Java StreamAPI)

```
fun main() {  
    println("Start")  
    val list = listOf<Int>(1,2,3,4,5,6,7,8,9,10)  
    val newList = list.stream().map {  
        Thread.sleep(1000)  
        it*it        // return it * it  
    }  
    println("End")  
    newList.forEach { // výpis kolekcie  
        println(it)  
    }  
}
```

Stream.of(1,2,3,4,5,6,7,8,9,10)

stream bez .collect() je *lenivá* kolekcia

```
08:59:32.832 Start  
08:59:32.834 End   Start+0sec.  
08:59:33.839 1      +1sec.  
08:59:34.841 4      +2sec.  
08:59:35.842 9  
08:59:36.844 16  
08:59:37.846 25  
08:59:38.849 36  
08:59:39.851 49  
08:59:40.854 64  
08:59:41.856 81  
08:59:42.858 100
```


Hádanka 2

neobsahuje corutinu ale stream (Java StreamAPI)

```
fun main() {  
    println("Start")  
    val list = listOf<Int>(1,2,3,4,5,6,7,8,9,10)  
    val newList = list.stream().map {  
        Thread.sleep(1000)  
        it*it  
    }.collect(Collectors.toList())  
    println("End")  
    newList.forEach { // výpis kolekcie  
        println(it)  
    }  
}
```

```
09:02:23.363 Start  
09:02:33.389 End   Start+10sec.  
09:02:33.389 1      +0sec.  
09:02:33.389 4      +0sec.  
09:02:33.389 9  
09:02:33.389 16  
09:02:33.389 25  
09:02:33.389 36  
09:02:33.390 49  
09:02:33.390 64  
09:02:33.390 81  
09:02:33.390 100
```

Hádanka 3

neobsahuje corutinu ale stream (Java StreamAPI)

```
fun main() {  
    println("Start")  
    val newList = listOf(1,2,3,4,5,6,7,8,9,10)  
        .parallelStream() ←  
        .map {  
            Thread.sleep(1000)  
            it*it  
        }.collect(Collectors.toList())  
    println("End")  
    newList.forEach { // výpis kolekcie  
        println(it)  
    }  
}  
  
parallelStream používá  
toľko paralelizmu, koľko je #cores  
Runtime.getRuntime()  
.availableProcessors() == 8
```

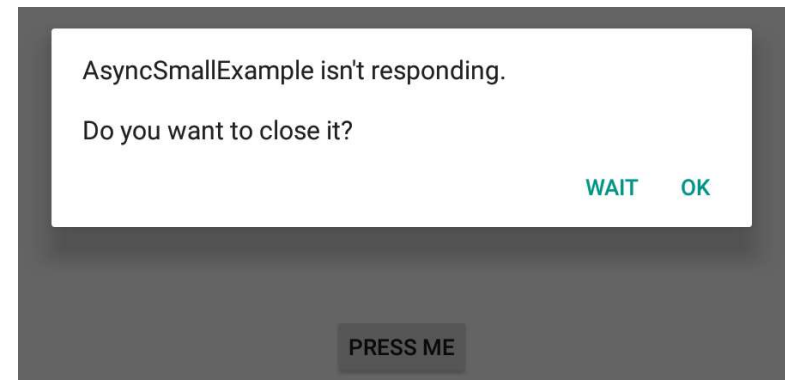
```
8  
09:04:06.410 Start  
09:04:09.420 End   Start+3sec.  
09:04:09.420 1      +0sec.  
09:04:09.420 4      +0sec.  
09:04:09.420 9  
09:04:09.420 16  
09:04:09.420 25  
09:04:09.420 36  
09:04:09.421 49  
09:04:09.421 64  
09:04:09.421 81  
09:04:09.422 100
```

Asynchrónne operácie

trochu androidu

- nie je možné robiť časovo náročné operácie v hlavnom vlákne aplikácie
 - extra komplikovaný (matematický) výpočet
 - simuláciu procesu spomaľovanú napr. `Thread.sleep(...)`
 - trvajúce požiadavky (napr. IO/http/sql-request), ktoré môžu trvať netriviálne dlho
- Takýto kód zablokuje hlavné vlákno, a ak vyvoláte GUI eventy (napr. pochabým klikaním v priebehu 20s), správca aplikácií usúdi, že aplikácia je mŕtva zavrie ju

```
fun buttonClick(view: View) {  
    for (i in 0..20) {  
        try {  
            Thread.sleep(1000) // zabije  
            i++                // hlavné vlákno  
        }  
        catch (e: Exception) {  
            e.printStackTrace()  
        }  
    }  
}
```





Async Task

(doInBackground)

Parametrizovaná trieda AsyncTask je thread-wrapper a rieši problém, existuje od API-3

```
        typ parametrov, type progresu, typ výsledku  
private inner class MyTask : AsyncTask<String, Int, String>() {
```

```
    {  
        override fun onPreExecute() {...} // vykoná sa pred doInBackground  
        // celé jadro toho, čo sa má vykonávať v extra vlákne
```

```
→ override fun doInBackground(vararg params: String): String {  
    while (i in 0..20) {  
        try {  
            Thread.sleep(1000)  
            publishProgress(i) // Counter = $i  
        } catch (e: Exception) { ... }  
        return "Button Pressed"  
    }  
}
```

```
    {  
        override fun onProgressUpdate(vararg values: Int?) { ... }  
        override fun onPostExecute(result:String) {...} // po doInBackground.  
    }
```



Async Task

(onPre/PostExecute)

```
private inner class MyTask : AsyncTask<String, Int, String>() {  
    var color : Int = Color.BLACK  
    → override fun onPreExecute() {  
        color = ... Random Color ...  
    }  
  
    {  
        override fun doInBackground(vararg params:String):String {  
            // varargs je variabilný počet argumentov, ako ... v Java  
  
            → override fun onProgressUpdate(vararg values: Int?) {  
                myTextView.setTextColor(color) // beží v main thread  
                val counter = values.get(0)  
                myTextView.text = "Counter = $counter"  
            }  
  
            → override fun onPostExecute(result:String) { "Button Pressed"  
                myTextView.setTextColor(color)  
                myTextView.text = result  
            }  
        }  
    }  
}
```



Async Task

(spustenie)

Štandardne sa rôzne inštancie AsyncTask spúšťajú sériovo, kým nedobehne jedna, ostatné čakajú vo fronte, blokujú sa...

```
val task1 = MyTask().execute() // serial run of AsyncTask
```

Ak ich chceme spustiť viacero a paralelne, tak cez POOL_EXECUTOR

```
task = MyTask().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR)
```

Ale počet paralelne bežiacich AsyncTaskov je limitovaný, v závislosti od počtu jadier CPU

```
val cpu_cores = Runtime.getRuntime().availableProcessors()
```

Reálne väčším problémom, že napriek popularite a jednoduchosti používania AsyncTask je od Android 11 AsyncTask zastaralý (*deprecated*)

https://www.xda-developers.com/asynctask-deprecate-android-11/amp/?__twitter_impression=true

Z toho zatiaľ nie je jasné, či ho Google odstráni, ale ...



Alternatívy

Kotlin - corutiny

Čo je alternatíva:

- RX-library
- Java's Concurrency framework
- Kotlin coroutines od verzie Kotlin 1.3

build.gradle:

- `implementation`
`"org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.2"`

import

- `import kotlinx.coroutines.*`

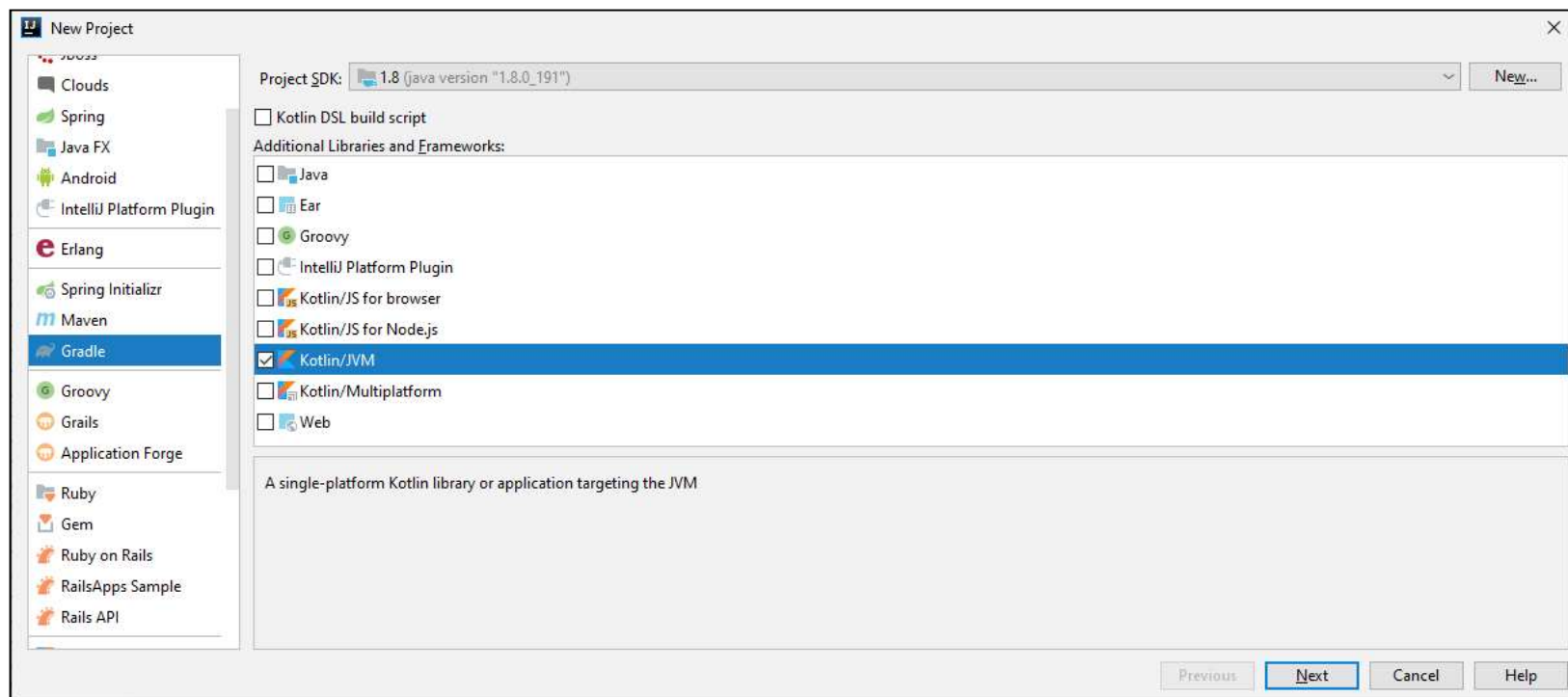
first-touch tutorial:

- <https://kotlinlang.org/docs/tutorials/coroutines/coroutines-basic-jvm.html>

IntelliJ/Gradle/KotlinJVM1.8

IntelliJ

- koncept corutiny je zložitý dosť na to ho študovať separátne, bez androidového okolia,
- ale ukážeme aj použitie korutín v reálnych android aplikáciach, časom...
- V IntelliJ si vytvorte Gradle project/KotlinJVM



Pridanie Coroutine dependencies do build.gradle

- na súbore build.gradle (app), right click/Generate/Add Maven Artifact dependencies/Search for artifacts:"coroutines", vyber

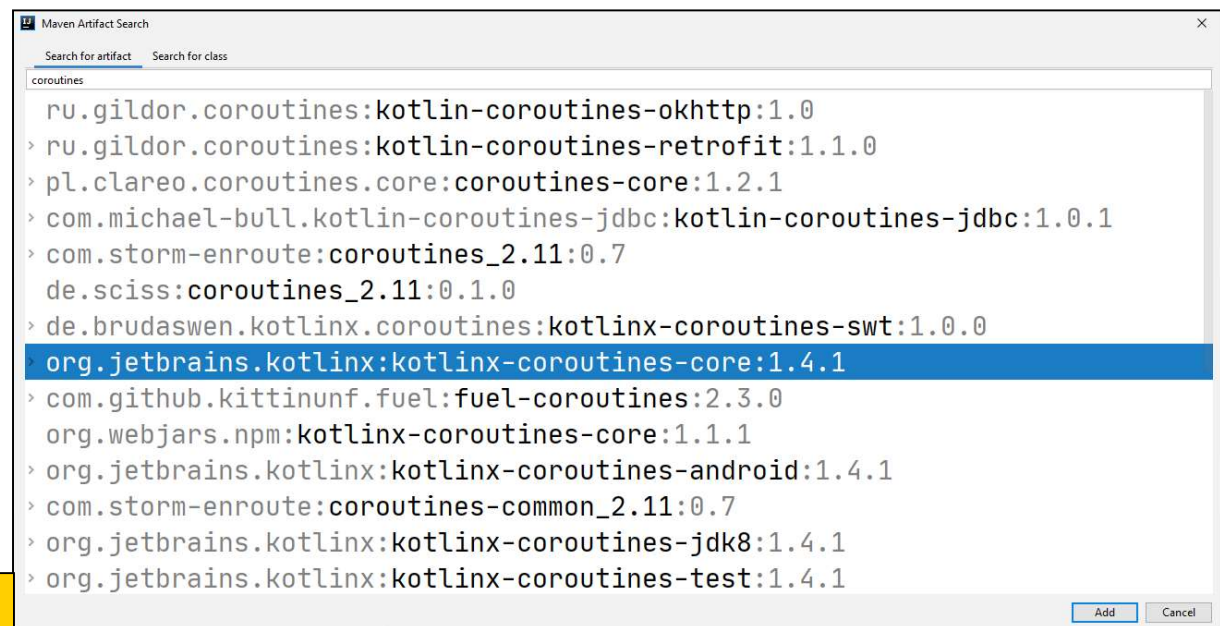
- org.jetbrains.kotlinx-coroutines-core:1.*.1

```
dependencies {  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8"  
    compile 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.1'  
}
```

nechajte syncovať
gradle, ~60sec.

Alebo si importujte
projekt Coroutines1

Coroutines1





Corutina

(Spustenie – blokujúce, neblokujúce)

- `.launch` spustí novú corutinu a neblokuje hlavné vlákno
- `.delay(ms)` pozastaví výpočet corutiny na ms..
- `.runBlocking` spustí novú corutinu a blokuje hlavné vlákno

```
Log.d(TAG, "Start")
→ GlobalScope.launch { // Start a coroutine, non-blocking
    delay(1000)          // wait 1s.
    Log.d(TAG, "Hello")
}
Thread.sleep(3000)      // wait for 3s.
Log.d(TAG, "Stop")
→ runBlocking {         // Start a coroutine, blocking
    delay(4000L)
}
Log.d(TAG, "Finish")
```

21:22:18.220	Start	
21:22:19.225	Hello	Start+1sec.
21:22:21.222	Stop	Start+3sec.
21:22:25.225	Finish	Start+7sec.



Corutina

(Spustenie – blokujúce, neblokujúce)

.join počká na dokončenie spustenej korutiny, ako Thread

```
println("Start")
→ val job = GlobalScope.launch {           // Start a non-blocking
    delay(1000)                             // wait 1s.
    println("Hello")
}
Thread.sleep(3000)                          // wait for 3s.
println("Stop")
runBlocking {                               // Start a blocking
→     job.join()                           // waiting until job finishes
}
println("Finish")
```

16:29:18.408	Start	
16:29:19.498	Hello	Start+1sec.
16:29:21.484	Stop	Start+3sec.
16:29:21.496	Finish	Start+3sec.



Corutina

(suspend)

suspend fun môže byť volaná len z coroutine scope
suspend fun môže volať inú suspend fun (napr. delay)

```
Log.d(TAG, "Start")
runBlocking {
    printHello()
}
Log.d(TAG, "Finish")
```

// Start a non-blocking

→

```
suspend fun printHello() {
    delay(1000L)
    Log.d(TAG, "Hello")
}
```

21:27:34.083	Start	
21:27:35.089	Hello	Start+1sec.
21:27:35.089	Finish	Start+1sec.



GlobalScope/launch/delay

```
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
```

```
fun main() {
    GlobalScope.launch {
        "world!".forEach {
            delay(200)
            print(it)
        }
    }
    print("Hello, ")
    Thread.sleep(2000)
}
```

// Start a non-blocking

```
12:46:15.811 Start
12:46:15.878 Hello,
12:46:16.106 w
12:46:16.318 o
12:46:16.519 r
12:46:16.721 l
12:46:16.924 d
12:46:17.130 !
12:46:17.882 Stop
```



Corutina

(suspend)

```
Log.d(TAG, "The main program is started")
```

```
→ GlobalScope.launch {  
    Log.d(TAG, "Background processing started")  
    delay(1000L)  
    Log.d(TAG, "Background processing finished")  
}  
Log.d(TAG, "The main program continues")
```

```
→ runBlocking {  
    delay(2000L)  
    Log.d(TAG, "The main program is finished")  
}
```

12:54:03.422 Start main
12:54:03.491 Continue main
12:54:03.495 Start background
12:54:04.501 Finish background
12:54:05.513 Stop main



Corutina

(async/await)

```
12:59:07.099 Start main
12:59:07.175 Awaiting computations...
12:59:08.192 Computation1 finished
12:59:09.188 Computation2 finished
12:59:09.188 The result is 3
12:59:09.189 Stop main
```

.async spustí novú corutinu, ktorá počíta nejaký výsledok
.await čaká na tento výsledok

```
runBlocking {
```

```
    val result1 = async { computation1() }
```

```
    val result2 = async { computation2() }
```

```
    Log.d(TAG, "Awaiting computations...")
```

```
    val result = result1.await() + result2.await()
```

```
    Log.d(TAG, "The result is $result")
```

```
} }
```

```
suspend fun computation1(): Int {
```

```
    delay(1000L) // simulated computation
```

```
    Log.d(TAG, "Computation1 finished")
```

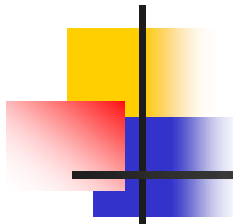
```
    return 1 }
```

```
suspend fun computation2(): Int {
```

```
    delay(2000L)
```

```
    Log.d(TAG, "Computation2 finished")
```

```
    return 2 }
```



Corutina

(cancel)

```
14:23:50.411 Start main
14:23:50.488 Processing 0 ...
14:23:51.499 Processing 1 ...
14:23:52.513 Processing 2 ...
14:23:53.520 Processing 3 ...
14:23:54.534 Processing 4 ...
14:23:55.546 Processing 5 ...
14:23:56.554 Processing 6 ...
14:23:57.568 Processing 7 ...
14:23:58.582 Processing 8 ...
14:23:59.597 Processing 9 ...
14:24:00.490 main: The user requests the cancellation
14:24:00.505 main: The batch is cancelled
```

```
runBlocking {
```

```
→    val job = launch { // Emulate some batch processing
```

```
        repeat(30) { i ->
```

```
            Log.d(TAG, "Processing $i ...")
```

```
            delay(1000L)
```

```
        }
```

```
    }
```

```
    delay(10000L)
```

```
    Log.d(TAG, "main: The user requests the cancellation")
```

```
→    job.cancelAndJoin()
```

```
        // cancel the job and wait for it's completion
```

```
    Log.d(TAG, "main: The batch is cancelled")
```

```
}
```




Corutina

(withTimeout)

```
14:28:58.109 Start main
14:28:58.192 Processing 0 ...
14:28:59.205 Processing 1 ...
14:29:00.214 Processing 2 ...
14:29:01.227 Processing 3 ...
14:29:02.239 Processing 4 ...
14:29:03.249 Processing 5 ...
14:29:04.262 Processing 6 ...
14:29:05.267 Processing 7 ...
14:29:06.280 Processing 8 ...
14:29:07.293 Processing 9 ...
14:29:08.194 The processing return status is: null
```

```
runBlocking {
    → val status = withTimeoutOrNull(10000L) {
        repeat(30) { i ->
            Log.d(TAG, "Processing $i ...")
            delay(1000L)
        }
        "Finished"
    }
    Log.d(TAG, "The processing return status is: $status")
}
```



Kotlin Coroutines

praktické použitie

- Coroutines:
 - už tušíme základy
- MVVM
 - download image
 - processing image, image filter, ...
- Retrofit
 - download json
 - upload json
- Coroutines ďalšie koncepty – asi na cviku, na pidi príkladičkoch...
 - Flow
 - Channel
- Room database – na budúce
 - lokálna SQL databáza
 - DAO – data access object



Dispatchers

vlákna v ktorých môžu bežať corutiny

- Dispatchers.Main - hlavné Android vlákno, interaguje s UI, pre ľahšie operácie
- Dispatchers.IO - vlákno optimalizované na sieťové IO mimo hlavného vlákna, použitie
 - databázové operácie
 - I/O, čítanie/písanie do súborov
 - sieťové veci
- Dispatchers.Default – vlákno optimalizované pre CPU intenzívne operácie mimo hlavného vlákna
 - ťažké výpočty, matematické výpočty, simulácie, triedenie zoznamov, ...
 - spracovanie väčších dát



Image download

from url

- download image from URL
- process image

```
CoroutineScope(Dispatchers.Main).launch { ← Main
→ val originalImage = async(Dispatchers.IO) { ← IO
    URL(IMAGE_URL).openStream().use { // download image from URL
        BitmapFactory.decodeStream(it)
    }
}
→ val originalBitmap = originalImage.await() // wait for download
→ val filteredImage = async(Dispatchers.Default) { ← Default
    toBlackAndWhite(originalBitmap)
}
→ val filteredBitmap = filteredImage.await()
    progressBar.visibility = View.GONE
    imageView.setImageBitmap(filteredBitmap)
    imageView.visibility = View.VISIBLE
}
```



Process Image

- image processing

```
fun toBlackAndWhite(source: Bitmap): Bitmap {  
    val w = source.width  
    val h = source.height  
    val bitmapArray = IntArray(w*h)  
    source.getPixels(bitmapArray, 0, w, 0, 0, w, h) // array from source  
    (0 until h).forEach { y->  
        (0 until w).forEach { x->  
            val index = x+y*w // index in 2D-matrix  
            val R = Color.red(bitmapArray[index])  
            val G = Color.green(bitmapArray[index])  
            val B = Color.blue(bitmapArray[index])  
            val grey = (R + G + B)/3  
            bitmapArray[index] = Color.rgb(grey, grey, grey)  
        }  
    }  
    val bitmapOut = Bitmap.createBitmap(w, h, Bitmap.Config.RGB_565)  
    bitmapOut.setPixels(bitmapArray, 0, w, 0, 0, w, h) // bitmap  
    bitmapOut // return bitmap  
}
```



Kotlin Coroutines

praktické použitie

- Coroutines:
 - už tušíme základy
- Download big image
 - download image
 - processing image, image filter, ...
- Retrofit + MVVM
 - download json
 - upload json
- Room database
 - lokálna SQL databáza
 - DAO – data access object

Retrofit



- Retrofit je REST klient pre Android
- zjednodušuje download & upload JSON (cez HTTP GET/POST)
- používa napr. Gson converter
- build.gradle treba doplniť o

```
implementation 'com.squareup.retrofit2:retrofit:2.6.2'  
implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
```

- data class zodpovedajúci JSONu (mapovanie na json tagy):

```
data class Stat (  
    @SerializedName("name")          /* -> */ val countryName: String?,  
    @SerializedName("capital")       /* -> */ val capital: String?,  
    @SerializedName("flagPNG")       /* -> */ val flag: String?,  
    @SerializedName("latlng")        /* -> */ val latlng: Array<Float>?,  
    @SerializedName("borders")       /* -> */ val borders: List<String>?,  
    @SerializedName("alpha3Code")    /* -> */ val code: String?  
)
```

- REST API pre Retrofit

```
interface StatInterface {  
    @GET("vlajky/staty.json")  
    suspend fun get(): Response<List<Stat>>  
}
```

Coroutines+MVVM+Retrofit

(model)

<https://dai.fmph.uniba.sk/courses/VMA/vlajky/staty.json>

```
data class Stat(  
    @SerializedName("name")          /* -> */ val countryName: String?,  
    @SerializedName("capital")       /* -> */ val capital: String?,  
    @SerializedName("flagPNG")       /* -> */ val flag: String?,  
    @SerializedName("latlng")        /* -> */ val latlng: Array<Float>?,  
    @SerializedName("borders")       /* -> */ val borders: List<String>?,  
    @SerializedName("alpha3Code")    /* -> */ val code: String?  
)
```

```
/*  
{  
    "alpha2Code": "SK",  
    "alpha3Code": "SVK",  
    "altSpellings": [  
        "SK",  
        "Slovak Republic",  
        "Slovensk\u00e1 republika"  
    ],  
    "area": 49037,  
    "borders": [  
        "AUT",  
        "CZE",  
        "HUN",  
        "POL",  
        "UKR"  
    ],  
    "callingCodes": [  
        "421"  
    ],  
    "capital": "Bratislava",  
    "currencies": [  
        {  
            "code": "EUR",  
            "name": "Euro",  
            "symbol": "\u20ac"  
        }  
    ],  
    "demonym": "Slovak",  
    "flagPNG":  
    "https://dai.fmph.uniba.sk/courses/VMA/vlajky/svk.png",  
    "gini": 26.0,  
    "languages": [  
        {  
            "iso639_1": "sk",  
            "iso639_2": "slk",  
            "name": "Slovak",  
            "nativeName": "sloven\u00fd jazyk"  
        }  
    ],  
    "latlng": [  
        48.66666666,  
        19.5  
    ],  
    "name": "Slovakia",  
    "nativeName": "Slovensko",  
    "numericCode": "703",  
    "population": 5426252,  
    "region": "Europe",  
    "regionalBlocs": [  
        {  
            "acronym": "EU",  
            "name": "European Union"  
        }  
    ],  
    "subregion": "Eastern Europe",  
    "timezones": [  
        "UTC+01:00"  
    ],  
}
```

```
    "gini": 26.0,  
    "languages": [  
        {  
            "iso639_1": "sk",  
            "iso639_2": "slk",  
            "name": "Slovak",  
            "nativeName": "sloven\u00fd jazyk"  
        }  
    ],  
    "latlng": [  
        48.66666666,  
        19.5  
    ],  
    "name": "Slovakia",  
    "nativeName": "Slovensko",  
    "numericCode": "703",  
    "population": 5426252,  
    "region": "Europe",  
    "regionalBlocs": [  
        {  
            "acronym": "EU",  
            "name": "European Union"  
        }  
    ],  
    "subregion": "Eastern Europe",  
    "timezones": [  
        "UTC+01:00"  
    ],  
}
```

CoroutineRetrofit

Coroutines+MVVM+Retrofit

(REST API - model)

```
interface RetroInterface {  
    @GET("vlajky/staty.json")  
    suspend fun get(): Response<List<Stat>>  
}  
  
object RetroService {  
    private val BASE_URL = "https://dai.fmph.uniba.sk/courses/VMA/"  
  
    fun get(): RetroInterface =  
        Retrofit.Builder()  
            .baseUrl(BASE_URL)  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
            .create(RetroInterface::class.java)  
}  
}
```

Coroutines+MVVM+Retrofit

(viewmodel)

```
class ListViewModel: ViewModel() {
    val service = RetroService.get()
    lateinit var job: Job
    val staty = MutableLiveData<List<Stat>>()

    fun fetch() {
        job = CoroutineScope(Dispatchers.IO)
            .launch {
                val response = service.get() // : Response<List<Stat>>
                withContext(Dispatchers.Main) {
                    if (response.isSuccessful)
                        staty.value = response.body()
                    else
                        Log.d("MODEL", "Error: ${response.message()}")
                }
            }
    }
    override fun onCleared() {
        super.onCleared()
        job.cancel()
    }
}
```

Coroutines+MVVM+Retrofit

(view)

```
class MainActivity : AppCompatActivity() {
    lateinit var viewModel: ListViewModel
    private val listAdapter = ListAdapter(arrayListOf())
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        viewModel = ViewModelProviders.of(this).get(ListViewModel::class.java)
        viewModel.fetch()
        listView.apply {
            layoutManager = LinearLayoutManager(context)
            adapter = listAdapter
        }
        observeViewModel()
    }
    fun observeViewModel() {
        viewModel.staty.observe(this, Observer { staty ->
            staty?.let { // if staty != null ...
                countriesList.visibility = View.VISIBLE
                listAdapter.updateCountries(it)
            }
        })
    }
}
```



- List adaptér používa Glide na čítanie obrázkov z URL
- <https://medium.com/@vlonjatgashi/using-glide-with-kotlin-5e345b557547>
- build.gradle:

```
apply plugin: 'kotlin-kapt'

dependencies {
    implementation 'com.github.bumptech.glide:glide:4.4.0'
    kapt 'com.github.bumptech.glide:compiler:4.4.0'
}
```

- kód:

```
import com.bumptech.glide.Glide

val options=RequestOptions().error(R.mipmap.ic_launcher_round)
Glide.with(imageView)
    .setDefaultRequestOptions(options)
    .load(country.flag)
    .into(imageView)
```



<https://eu1.unwiredlabs.com/v2/process.php>

```
{
  "token": "95b2941777892d",
  "mcc": 231,
  "mnc": 2,
  "cells": [{
    "lac": 1,
    "cid": 31441
  }],
  "address": 1
}
```

```
{
  "status": "ok",
  "balance": 97,
  "lat": 48.14875,
  "lon": 17.06679,
  "accuracy": 837,
  "address": "Botanická, Švédske  
domky, Bratislava, Karlova Ves,  
Bratislava, Region of Bratislava, 841  
04, Slovakia"
}
```

V prednáške o polohe sme narazili
na problém, že GSM súradnice prekladá do lat-long servis

- potrebujeme mu poslať a prečítať json-dáta, cez HTTP-POST
 - ak zavrhneme riešenie, že "lepíme reťazce" do JSON a vyhladávame v ňom podstringy, ...
 - riešenie založené na json knižnici `android.util.JsonReader/JsonWriter` (ukážeme)
 - riešenie založené na Gson knižnici (konvertuje json do objektu cez Java reflection model)
- nesmieme to robiť v hlavnom vlákne, lebo to môže trvať...
 - riešenie pomocou AsyncTask (old-school)
 - corutinovské riešenie (new-wave)

ako zo vzorky JSON to vyrobiť Kotlin Class ?

- build.gradle

Výmena dát so serverom

Výmena dát klient-server

- cez parametre GET/POST requestu,
- cez obsah POST requestu,
- cez cookies - nebude



uvidíme:

- cez JSON objekt
 - `pomocou org.json.*`
 - `pomocou com.google.gson.*`
- cez xml formát
 - `pomocou org.xml.sax.*;`
 - <http://dai.fmph.uniba.sk/courses/java2/sl/xml.pdf>





```
D/MyGSMLocation(19361): gsm cid: 396517
D/MyGSMLocation(19361): gsm lac: 1001
D/MyGSMLocation(19361): operator:23102
D/MyGSMLocation(19361): network: 23102
D/MyGSMLocation(19361): mcc: 231
D/MyGSMLocation(19361): mnc: 2
```

- zaregistrujete sa napr. na 7-dňový trial, max. 50 requests/day
- dostanete kľúč (token), 95b2941777892d (keď toto čítate, asi už neplatí ☹)
- skúste 95b2941777892d (7.dec 2017).

<http://locationapi.org/site/page?view=apiv2>

Request: 1 cell | 3 cells | 7 cells

```
1 {
2   "token": "1445573628",
3   "mcc": 231,
4   "mnc": 2,
5   "cells": [{
6     "cid": 396517,
7     "lac": 1001,
8     "signal": -60,
9     "tA": 13
10  }]
11 }
```

Response:

```
1 {
2   "status": "ok",
3   "balance": 45,
4   "lat": 48.16802,
5   "lon": 17.11049,
6   "accuracy": 1063,
7   "message": "Accuracy is in BETA!"
8 }
```

API v2 Documentation

1. [Usage](#)
2. [Test it out](#)
3. [Request body](#)
4. [Response body](#)
5. [Example Script - PHP](#)
6. [Example Script - Python](#)

Usage

Requests are sent using POST to the following url:

<http://locationapi.org/v2/process.php>

LocationAPI z aplikácie

- potrebujeme urobiť http-POST request na <http://locationapi.org/v2/process.php>
- keďže to niečo trvá, nesmieme to robiť v hlavnom vlákne – AsyncTask
- do tela dotazu (requestu) potrebujeme zakódovať (cellID, lac, mcc, mnc + môj token) hoc jednoduchý, ale predsa-len JSON objekt
- z tela odpovede (responzu) potrebujeme dekodovať hoc jednoduchý, ale JSON objekt, t.j. prečítať latitude-longitude

Request: 1 cell | 3 cells | 7 cells

```
1 {
2   "token": "1445573628",
3   "mcc": 231,
4   "mnc": 2,
5   "cells": [{
6     "cid": 396517,
7     "lac": 1001,
8     "signal": -60,
9     "tA": 13
10  }]
11 }
```

Response:

```
1 {
2   "status": "ok",
3   "balance": 45,
4   "lat": 48.16802,
5   "lon": 17.11049,
6   "accuracy": 1063,
7   "message": "Accuracy is in BETA!"
8 }
```


Vytvorenie (malého) JSON objektu

(pre GET LocationAPI)

```
val sw = StringWriter()
```

```
val jw = JsonWriter(sw)
```

```
try {
```

```
    jw.beginObject() -- {
```

```
        jw.name("token").value(token_locationAPIORG)
```

```
        jw.name("mcc").value(mcc)
```

```
        jw.name("mnc").value(mnc)
```

```
        jw.name("cells")
```

```
        jw.beginArray() -- [
```

```
            .beginObject() -- {
```

```
                jw.name("cid").value(cid)
```

```
                jw.name("lac").value(lac)
```

```
                jw.name("signal").value(-60)
```

```
                jw.name("tA").value(13)
```

```
            jw.endObject().endArray().endObject().close() -- } ] }
```

```
import android.util.JsonWriter
```

Request: 1 cell | 3 cells | 7 cells

```
1 {
2   "token": "1445573628",
3   "mcc": 231,
4   "mnc": 2,
5   "cells": [{
6     "cid": 396517,
7     "lac": 1001,
8     "signal": -60,
9     "tA": 13
10  }]
11 }
```

Project: MyGSMLocation.zip

Dekódovanie (malého) JSON

```
import android.util.JsonReader
```

```
val sr = StringReader(result)
```

```
val jr = JsonReader(sr)
```

```
jr.beginObject() -- {
```

```
    jr.nextName() -- skip: "status"
```

```
    jr.nextString() -- skip: "ok"
```

```
    jr.nextName() -- skip: "balance"
```

```
    jr.nextInt() -- skip: 45
```

```
    jr.nextName() -- skip: "lat"
```

```
    lat = jr.nextDouble()
```

```
    jr.nextName() -- skip: "lon"
```

```
    lng = jr.nextDouble()
```

```
    jr.nextName() -- skip: "accuracy"
```

```
    accur = jr.nextInt()
```

Response:

```
1 {
2   "status": "ok",
3   "balance": 45,
4   "lat": 48.16802,
5   "lon": 17.11049,
6   "accuracy": 1063,
7   "message": "Accuracy is in BETA!"
8 }
```



GSON

(fromJson)

```
{  
  "id": "1547257485",  
  "name": "Peter Borovansky",  
  "first_name": "Peter",  
  "last_name": "Borovansky",  
  "link": "http://www.facebook.com/  
         peter.borovansky",  
  "username": "peter.borovansky",  
  "gender": "male",  
  "locale": "cs_CZ"  
}
```

Idea: k JSON objektu definujeme zodpovedajúcu (1:1) java triedu
Obmedzenia (viac <https://github.com/google/gson/blob/master/UserGuide.md>):

- mená JSON tagov sa musia zhodovať s java menami polí v triede

```
class FBHeader {  
    public String id = "";  
    public String name = "";  
    public String first_name = "";  
    public String last_name = "";  
    public String link = "";  
    public String username = "";  
    public String gender = "";  
    public String locale = "";  
}
```

```
import com.google.gson
```

```
Gson gson = new GsonBuilder().create();
```

```
FBHeader header = gson.fromJson(jsonstring, FBHeader.class);
```



FB Friends

(fromJson)

```
{ "data":  
  [ { "name": "Zuzka B...", "id": "582749468" },  
    { "name": "Lubica K...", "id": "583024903" },  
    { "name": "Barbora F...", "id": "632007063" } ],  
  "paging": { "next": "https://graph.facebook.com/15..." }
```

```
class FBFriends { // dvojica  
    public FBPairs[] data = null;  
    public FB Paging paging = null; }  
class FB Pairs { // dvojica  
    public String name = "";  
    public String id = ""; }  
class FB Paging { // singleton  
    public String next = ""; }
```

```
import com.google.gson
```

```
Gson gson = new GsonBuilder().create();  
FBFriends friends = gson.fromJson(result, FBFriends.class);  
if (friends != null) {  
    if (friends.data != null)  
        for (int i = 0; i < friends.data.length; i++)  
            if (friends.data[i] != null)  
                tv.append(friends.data[i].name + ",");  
}
```

GSON – ako to funguje ?

Reflexivita

Ukázali sme

- `fromJson` (do Javy)
- ale analogicky funguje
- `toJson` (z Javy)

`org.json`

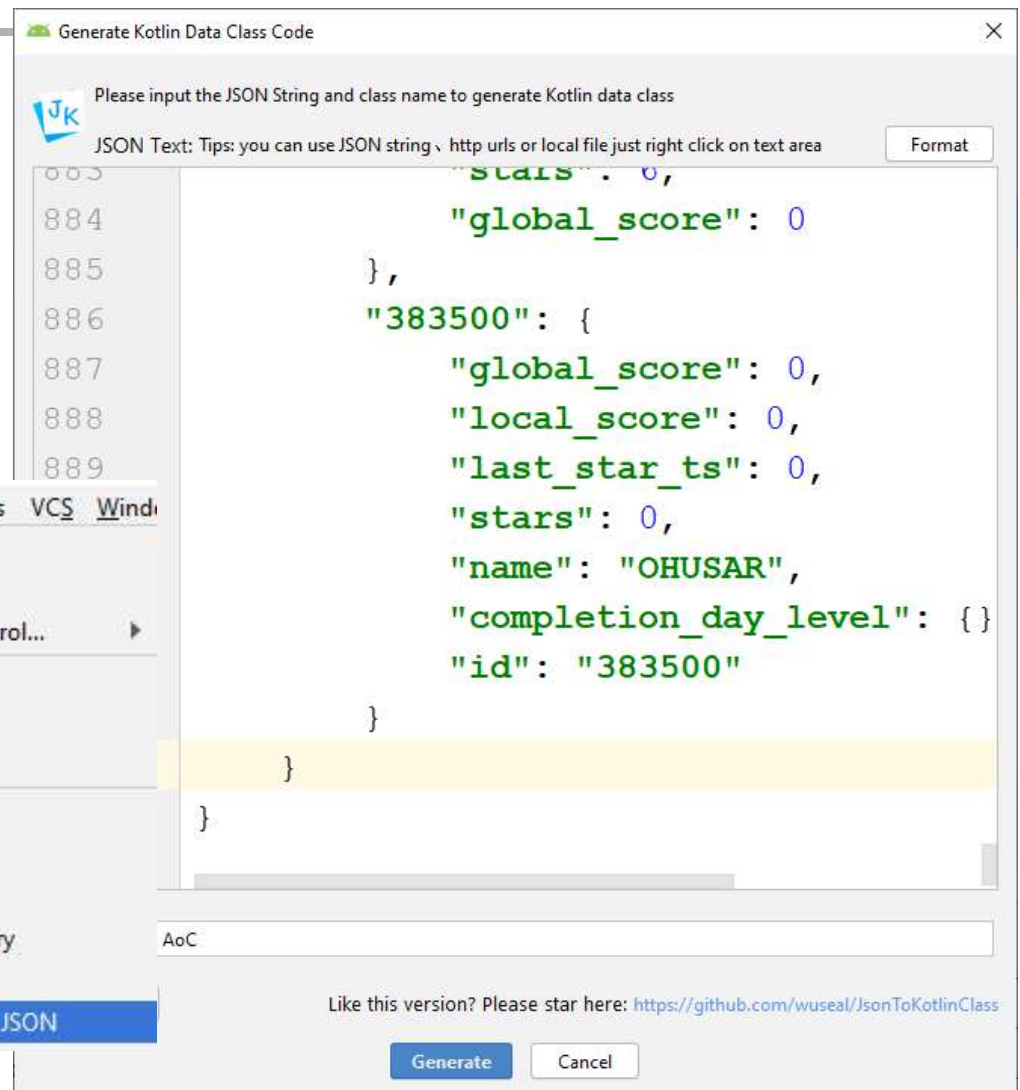
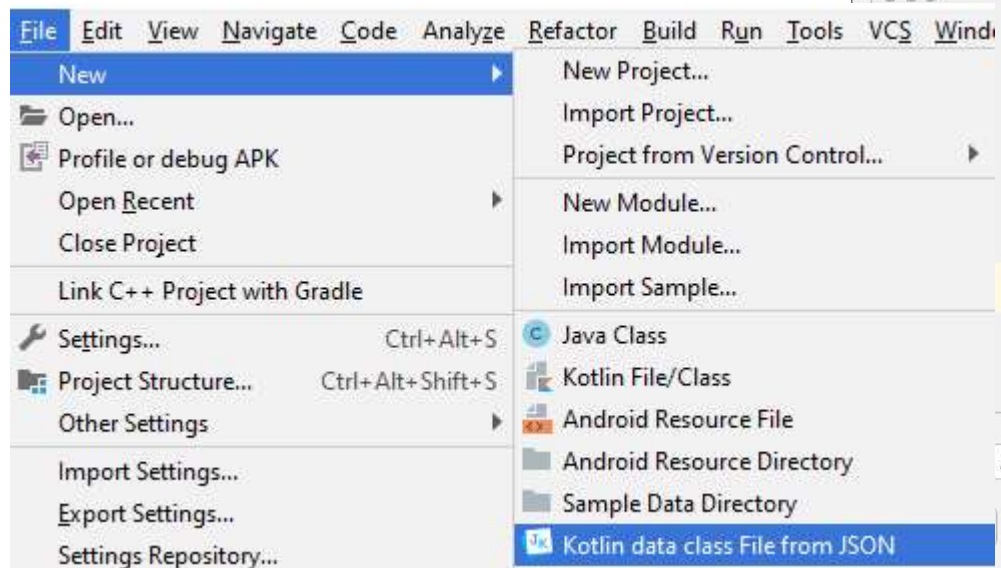
vs.

`com.google.gson`



Plugin JSON to Kotlin Class

- z daného JSON vytvorí definíciu Kotlin tried
- potom stačí zavolať fromJson prekonvertuje vám json-string do dátovej štruktúry



GSM-Retrofit

- JSON to Kotlin Class
- build.gradle

```
implementation 'com.google.code.gson:gson:2.8.5'  
implementation 'com.squareup.retrofit2:retrofit:2.6.2'  
implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
```

- toto si dáme vygenerovať pluginom JSON to Kotlin Class

```
data class Cell(  
    val cid: Int,  
    val lac: Int  
)
```

ak interné mená zodpovedajú JSON tagom,
tak neriešime `@SerializedName`

```
data class GSMRequest(  
    val address: Int,  
    val cells: List<Cell>,  
    val mcc: Int,  
    val mnc: Int,  
    val token: String  
)
```

```
data class GSMResponse(  
    val accuracy: Int,  
    val address: String,  
    val balance: Int,  
    val lat: Double,  
    val lon: Double,  
    val status: String  
)
```

GSMRetrofit

<https://eu1.unwiredlabs.com/v2/process.php>

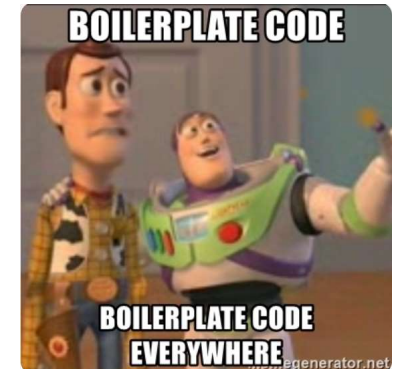
```
{  
  "token": "95b2941777892d",  
  "mcc": 231,  
  "mnc": 2,  
  "cells": [{  
    "lac": 1,  
    "cid": 31441  
  }],  
  "address": 1  
}
```

GSMRequest

```
{  
  "status": "ok",  
  "balance": 97,  
  "lat": 48.14875,  
  "lon": 17.06679,  
  "accuracy": 837,  
  "address": "Botanická,  
  Švédske domky, Bratislava,  
  Karlova Ves, Bratislava,  
  Region of Bratislava, 841 04,  
  Slovakia"  
}
```

GSMResponse

[process.php](#)



Rest API

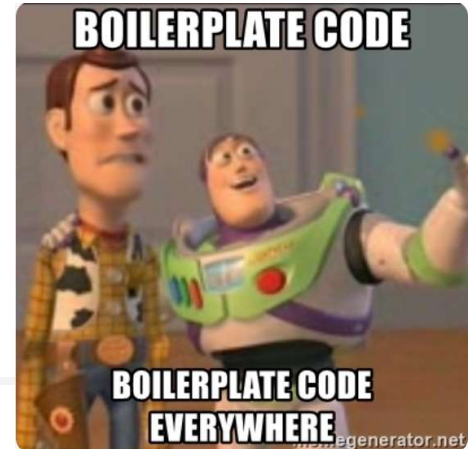
```
interface RestApiInterface {
    @Headers("Content-Type: application/json")
    @POST("process.php")
    fun gsm2latlong(@Body gsmRequest: GSMRequest): Call<GSMResponse>
}
```

```
class RestApiService {
    suspend
    fun gsm2latlong(gsmRequest: GSMRequest, onResult: (GSMResponse?) -> Unit){
        val retrofit = ServiceBuilder.get()
        retrofit.gsm2latlong(gsmRequest).enqueue(
            object : Callback<GSMResponse> {
                override fun onFailure(call: Call<GSMResponse>, t: Throwable) {
                    onResult(null) ← null
                }
                override fun onResponse(call: Call<GSMResponse>,
                    response: Response<GSMResponse>) {
                    val resp = response.body()
                    onResult(resp) ← !=null
                }
            }
        )
    }
}
```

GSMRetrofit

<https://eu1.unwiredlabs.com/v2/>

Service Builder



```
object ServiceBuilder {  
    private val client = OkHttpClient.Builder().build()  
  
    suspend  
    fun get(): RestApiInterface =  
        Retrofit.Builder()  
            .baseUrl("https://eu1.unwiredlabs.com/v2/")  
            .addConverterFactory(GsonConverterFactory.create())  
            .client(client)  
            .build()  
            .create(RestApiInterface::class.java)  
}
```

Volanie - bez corutiny

```
val request = GSMRequest(
    token = "95b2941777892d",
    mcc = mcc,
    mnc = mnc,
    cells = listOf(Cell(lac = lac, cid = cid)),
    address = 1
)

val apiService = RestApiService()
val response = apiService.gsm2latlong(request) {
    response -> // toto je onResult
    if (response != null) {
        Log.d(TAG, "${response.lat}, ${response.lon}")
        latTV.text = response.lat.toString()
        longTV.text = response.lon.toString()
    } else
        Log.d(TAG, "response is null")
}
```

```
class RestApiService {
    suspend
    fun gsm2latlong(gsmRequest: GSMRequest,
        onResult: (GSMResponse?) -> Unit)
```

Volanie – s corutinou

```
val request = GSMRequest(
    token = "95b2941777892d",
    mcc = mcc,
    mnc = mnc,
    cells = listOf(Cell(lac = lac, cid = cid)),
    address = 1
)
CoroutineScope(Dispatchers.IO).Launch {
    val apiService = RestApiService()
    val response = apiService.gsm2latlong(request) {
        response -> // toto je onResult
        if (response != null) {
            Log.d(TAG, "${response.lat}, ${response.lon}")
            latTV.text = response.lat.toString()
            longTV.text = response.lon.toString()
        } else
            Log.d(TAG, "response is null")
        }
    }
```

```
class RestApiService {
    suspend
    fun gsm2latlong(gsmRequest: GSMRequest,
        onResult: (GSMResponse?) -> Unit)
```



GUI len ako Dispatchers.Main

```
val request = GSMRequest(
    token = "95b2941777892d",
    mcc = mcc,
    mnc = mnc,
    cells = listOf(Cell(lac = lac, cid = cid)),
    address = 1
)
CoroutineScope(Dispatchers.IO).Launch {
    val apiService = RestApiService()
    val response = apiService.gsm2latlong(request) {
        response -> // toto je onResult
        if (response != null) {
            Log.d(TAG, "${response.lat}, ${response.lon}")
            CoroutineScope(Dispatchers.Main).Launch { ←
                latTV.text = response.lat.toString()
                longTV.text = response.lon.toString()
            }
        } else
            Log.d(TAG, "response is null")
    }
}
```