

Mobile Programming

Android 4: Nebenläufigkeit & Kommunikation



Nicola Keller



Inhalt

- Nebenläufigkeit (Concurrency)
 - Der main-Thread & Verhinderung von blockiertem UI
 - Nebenläufige Programmiermodelle
 - `ThreadWorker` (`WorkManager`)
- Kommunikation
 - Backend-Kommunikation über HTTP
 - HTTP-GET
 - Webservice mit JSON-Daten
 - Typisierter API-Konsum mit Retrofit



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Nebenläufigkeit: das Blockierungsproblem

Android und der Main-Thread

- Eine Applikation läuft per Default in genau einem Thread, dem **main-Thread**
 - In diesem main-Thread wird das ganze UI aufgebaut, d.h. **main-Thread = UI-Thread**
- Konsequenz: Falls main-Thread blockiert, friert das UI ein („UI freeze“)... ❄
- Wichtig: UI-Komponenten sind **NICHT** Thread-safe!
 - D.h. **UI-Zugriff nur aus main-Thread**, sonst Exception:
Caused by: android.view.ViewRootImpl\$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

UI & blockierende Methoden...

- Viele Netzwerk-Methoden (und auch andere!) können lange dauern und laufen synchron ab, d.h. blockieren bis sie fertig sind
 - z.B. `URLConnection.connect()`
 - oder `Bitmap.resize()` oder `Database.open()` ...
- Wird eine solche Methode auf dem main-Thread aufgerufen, so wird dieser blockiert und es werden keine UI-Events mehr verarbeitet
 - D.h. die App reagiert z.B. nicht auf Touch-Events, usw.

 **Das gilt es zu verhindern!**

Demo: Blockierendes UI (Ü4)

- Einfaches Blockieren mittels `Thread.sleep(long time)`
 - in Millisekunden
- Effekt: App reagiert nicht
 - Keine UI-Aktualisierungen
 - Keine Reaktion auf UI-Events
 - „App freeze“ !!

HSLU Mobile Programming

Nebenläufigkeit (Threads & Worker)

GUI 7 SEKUNDEN BLOCKIEREN

DEMO-THREAD STARTEN

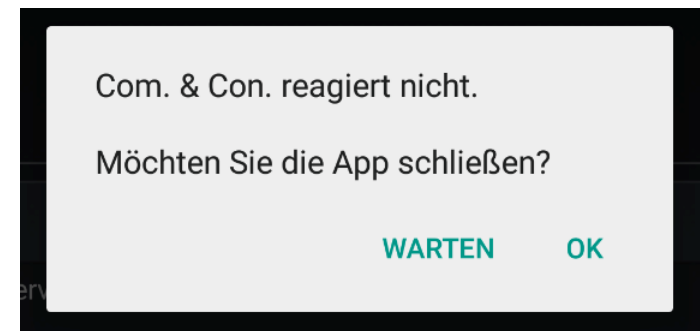
DEMO-WORKER STARTEN

```
fun freeze7Seconds(view: View?) {  
    try {  
        Thread.sleep(WAITING_TIME_MILLIS)  
    } catch (e: InterruptedException) { //  
    }  
}
```

Android-Überwachung: ANR

ANR = Application
Not Responding

- Android System überwacht Ansprechbarkeit (Responsiveness) von Apps
 - Kriterien (Siehe <https://developer.android.com/training/articles/perf-anr.html>)
 - Keine Reaktion auf Input-Event innert 5 Sek
 - Broadcast-Receiver nicht fertig innert 10 Sek
- Möglicher Effekt: ANR-Dialog
 - System-Mechanismus zum stoppen von „bösen“ Apps
- Was tun wir dagegen?



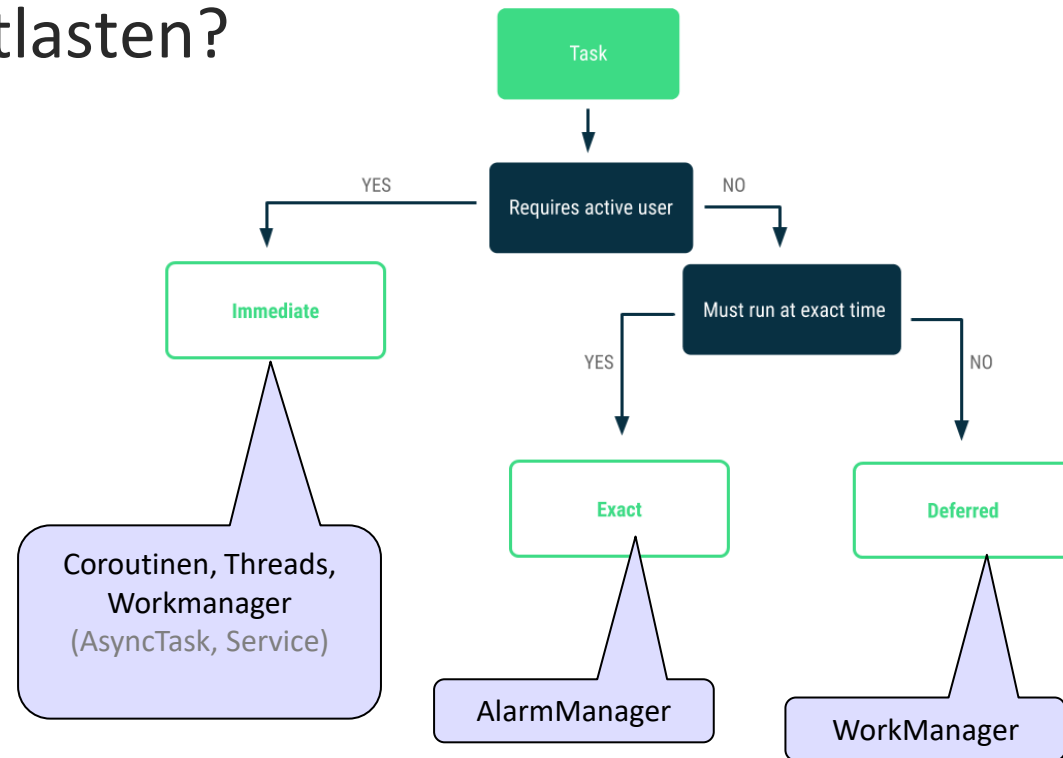
Wie den main-Thread entlasten?

Fragen zum Task:

- Braucht's Benutzer*in?
- Muss zu exakter Zeit laufen?

Herausforderungen:

- Resultat am Ende auf UI-Thread darstellen
- UI noch da? (Gelöst durch ViewModels 😊)



-> Selber lesen unter <https://developer.android.com/guide/background> JETZT! 😊

Wie zurück zum UI-Thread?

Eigener Thread, zwei Möglichkeiten:

- `Activity.runOnUiThread(Runnable action)`
- `View.post(Runnable action)`
- `View.postDelayed(Runnable, long)`
- Klasse `android.os.Handler`
 - Benutzt `MessageQueue` von `Thread`
 - Schauen wir nicht weiter an

Langandauernde Operationen, z.B. Netzwerk

- Android lässt gewisse Operationen (z.B. Network-API) per Default nicht auf main-Thread zu!

- z.B. Aufruf von `URLConnection.connect()` führt zu **NetworkOnMainThreadException**:

```
android.os.NetworkOnMainThreadException
    at android.os.StrictMode$AndroidBlockGuardPolicy.onNetwork(StrictMode.java:1147)
    at java.net.InetAddress.lookupHostByName(InetAddress.java:418)
    at java.net.InetAddress.getAllByNameImpl(InetAddress.java:252)
```

- Grund: Netzwerk-Kommunikation kann dauern!
 - Netzwerk-Calls nie auf UI-Thread ausführen
 - AsyncTask (oder eigenen Thread) verwenden
 - Und UI-Aktualisierungen in Methoden auf main-Thread!



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Nebenläufigkeit: Threads

Java-Threads: Kurze Repetition

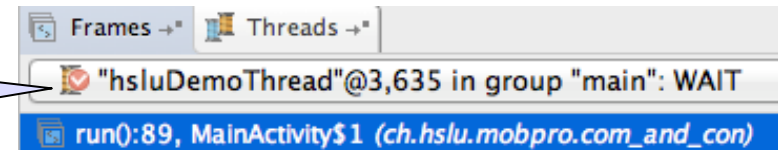
- `java.lang.Thread` implements `Runnable`
 - Muss ein `Runnable` gesetzt haben (übergeben im Konstruktor) oder selber `run()` implementieren
 - Wichtigste Methoden:
 - `run()`
 - `start()`
 - `sleep(long)`
 - `isAlive()`
- Functional Interface `java.lang.Runnable`
 - Eine Methode: `run()`
 - Implementierung des relevanten nebenläufigen Codes

Verwendung von Threads: Beispiel-Code

Button-Listener-Methode in Activity-Klasse

```
fun startDemoThread(view: View) {
    if (!demoThread.isAlive) { // only start a new task if there is no one running
        demoThread = createDemoThread()
        demoThread.start()
        main_thread_button.text = "[Demo-Thread läuft...]"
    } else {
        Toast.makeText(context: this, text: "DemoThread läuft schon!", Toast.LENGTH_SHORT)
    }
}
```

Thread-Name wird im
Debugger angezeigt



Instanziierung von
anonymer Sub-Klasse
von Thread mittels
object

Blockierung!

Thread-Name

```
private fun createDemoThread(): Thread {
    return object : Thread(name: "hsluDemoThread") {
        override fun run() {
            try {
                sleep(WAITING_TIME_MILLIS)
                runOnUiThread {
                    main_thread_button.text = resources.getString(R.string.demo_thread_running)
                    Toast.makeText(context: this@MainActivity, text: "DemoThread läuft", Toast.LENGTH_SHORT)
                }
            } catch (e: InterruptedException) {
                // Thread was interrupted
            }
        }
    }
}
```

Callback: Benachrichtigung des UI-Thread (= main-Thread) durch
Activity.runOnUiThread(Runnable)

GUI 7 SEKUNDEN BLOCKIEREN

DEMO-THREAD STARTEN

DEMO-WORKER STARTEN

Demo: DemoThread (Ü4)

- Auf Knopfdruck wird im Hintergrund 7 Sek gewartet
 - UI wird NICHT blockiert, da auf Hintergrund-Thread 😊
- Wichtig: UI-Anpassungen nur auf UI-Thread
 - Zurück auf main-Thread mit `Activity.runOnUiThread`
- Weiter implementiert
 - Änderung Button-Titel
 - Mechanismus, dass max. 1 DemoThread läuft

[DEMO-THREAD LÄUFT...]

DemoThread läuft schon!

Versionen & Änderungen...

(<https://developer.android.com/preview>)

Apropos API-Changes...

developers

Platform

Android Studio

Google Play

More ▼

Search

ENGLISH

About the platform

OVERVIEW

RELEASES

TECHNOLOGY

LIBRARIES

GAMES

MACHINE LEARNING

PRIVACY

Versions

Android 11 Developer Preview

Home

Overview & timeline

Privacy updates

Features and APIs

Behavior changes for apps

All apps

Apps targeting Android 11

Non-SDK restrictions

Get started with Android 11

Release notes

Feedback and issues

Android 10

Pie

Oreo

Nougat

Marshmallow

Lollipop

KitKat

Wichtiges
Kapitel 😊

Frühere
Versionen

The Developer Preview for Android 11 is now available; [test it out and share your feedback](#).

Android Developers > Platform > Releases

☆☆☆☆☆

Behavior changes: all apps

Contents ▼

Non-SDK interface restrictions

Gesture Navigation

NDK

Shared objects cannot contain text relocations

...

Android 10 includes behavior changes that may affect your app. The changes listed in this document apply to your app when running on Android 10, regardless of the app's `targetSdkVersion`. You should test your app and modify it as needed to support these changes properly.

If your app's `targetSdkVersion` is 29 or higher, you'll also need to support additional changes. Make sure to read [behavior changes for apps targeting 29](#) for details.

Framework security changes

Android 9 includes several behavior changes that improve your app's security. These changes take effect only if your app targets API level 28 or higher.

Network TLS enabled by default

If your app targets Android 9 or higher, the `isCleartextTrafficPermitted()` method returns `false` by default. If your app needs to enable cleartext for specific domains, you must explicitly set `cleartextTrafficPermitted` to `true` for those domains in your app's [Network Security Configuration](#).

Web-based data directories separated by process

In order to improve app stability and data integrity in Android 9, app data is now separated by process. Web-based data directory among [multiple processes](#). Typically, such data directory is used for web browser data, including other persistent and temporary storage related to web browser.

In most cases, your app should use `CookieManager`, in only one process. You should move all `WebView` into the same process and call `disableWebView()` in your app to prevent those other processes by mistake, even if it's being called from a dependent library.

If your app must use instances of `WebView` in more than one process, you must assign a unique data directory suffix for each process, using the `WebView.setDataDirectorySuffix()` method, before using a given instance of `WebView` in that process. This method places web data from each process in its own directory within your app's data directory.

★ **Note:** Even if you use `setDataDirectorySuffix()`, the system doesn't share cookies and other web data across your app's process boundaries. If multiple processes in your app need access to the same web data, you need to copy it between those processes yourself. For example, you can call `getCookie()` and `setCookie()` to manually transfer cookie data from one process to another.

Android P(ie), API-28

Beispiel für einen **Breaking Change**: Daher immer kurz überfliegen bei neuen Android-Versionen... 😊

Siehe Beispiel "Cleartext-HTTP" später in den Folien...



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Nebenläufigkeit: Worker

Worker / WorkManager



- `androidx.work.WorkManager`
 - Teil von den Architectural Components vom Android JetPack (Package `androidx.*`)
- Einfache Implementierung: `Worker`
 - Genau zwei public Methoden: `doWork()` und `startWork()`
 - Parameterübergabe im Konstruktor mittels `WorkerParameters`
- Empfehlenswertes Code-Lab zum kennenlernen:
<https://codelabs.developers.google.com/codelabs/android-workmanager/>

Verwendung Worker: Demo-Code

```
fun startDemoWorker(v: View?) {
    val workManager = WorkManager.getInstance(application)
    val demoWorkerRequest = OneTimeWorkRequestBuilder<DemoWorker>()
        .setInputData(Data.Builder()
            .putLong(DemoWorker.WAITING_TIME, 1000)
            .build())
        .build()
    workManager.enqueue(demoWorkerRequest)
```

Button-Listener-Methode in Activity-Klasse

Typen-Angabe vom gewünschten Worker

Parameter rein...

...mit enqueue (...) geht's los!

```
class DemoWorker(ctx: Context, private val params: WorkerParameters) : Worker(ctx, params)
```

```
    override fun doWork(): Result {
        return try {
            val waitingTime = params.inputData.getLong(WAITING_TIME_KEY, DEFAULT_WAITING_TIME)
            Thread.sleep(waitingTime)
            Result.success()
        } catch (throwable: Throwable) {
            Result.failure()
        }
    }
}
```

...und Parameter raus 😊

...gesehen: KEIN return!
(Trotz Rückgabe-Typ Result)

Bei Kotlin ist der Wert von letzten Ausdruck automatisch Rückgabewert 😊

Demo: WaitThread (Ü4)

- Auf Knopfdruck wird im Hintergrund 7 Sek gewartet
 - UI wird NICHT blockiert, da auf Hintergrund-Thread vom WorkManager 😊
- Das Warten in doWork von DemoWorker findet NICHT auf main-Thread statt
 - "Normaler" App-Code z.B. von der Activity läuft auf dem main-Thread

HSLU Mobile Programming

Nebenläufigkeit (Threads & Worker)

GUI 7 SEKUNDEN BLOCKIEREN

DEMO-THREAD STARTEN

DEMO-WORKER STARTEN

...hier leistet der Debugger gute Dienste! 😊

Frames Threads

▼ ✓ "pool-1-thread-1"@10,400 in group "main": RUNNING

doWork:12, DemoWorker {ch.hslu.mobpro.com_and_con}

run:85, Worker\$1 {androidx.work}

runWorker:1167, ThreadPoolExecutor {java.util.concurrent}

run:641, ThreadPoolExecutor\$Worker {java.util.concurrent}

run:919, Thread {java.lang}

Frames Threads

▼ ✓ "main"@9,108 in group "main": RUNNING

startDemoWorker:72, MainActivity {ch.hslu.mobpro.

invoke:-1, Method {java.lang.reflect}

onClick:397, AppCompatActivity\$DeclaredOnCli

performClick:7125, View {android.view}

performClickInternal:7102, View {android.view}

access\$2500:801, View {android.view}

Nebenläufigkeit: Abschlussbemerkungen

...spannende Lektüre /
gute Weiterbildung! 😊



- Es gibt verschiedene Optionen!
 - Gute Übersicht "asynchrone Programmiermodelle" in Kotlin (Threading, Callbacks, Futures & Promises, Reactive Extensions, Coroutines) unter <https://kotlinlang.org/docs/async-programming.html>
 - Zuviel (neuer bzw. nicht-trivialer) Stoff für dieses Modul, daher nicht weiter behandelt...
- Für Android/Kotlin interessant & relevant: Coroutinen
 - <https://developer.android.com/kotlin/coroutines>
 - <https://kotlinlang.org/docs/coroutines-overview.html>

Improve app performance with Kotlin coroutines

Kurze Demo: Coroutinen (nicht prüfungsrelevant)

- Quizfrage: Was produzieren die zwei angegebenen Methoden je für eine Ausgabe?

- Bsp. von <https://kotlinlang.org/docs/coroutines-basics.html#your-first-coroutine>

```
fun main() {  
    GlobalScope.launch { //  
        delay(1000L) // nc  
        println("World!")  
    }  
    println("Hello,") // m  
    Thread.sleep(2000L) //  
}
```

```
fun runLightWeigth() = runBlocking() {  
    println("1")  
    repeat(times: 10_000) {  
        launch { this: CoroutineScope  
            delay(timeMillis: 1000L)  
            print(".")  
        }  
    }  
    println("2")  
    delay(timeMillis: 1500L)  
    println("3")  
}
```




<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

(Backend-) Kommunikation über HTTP

Backend-Kommunikation

- Viele Apps kommunizieren mit einem Server im Hintergrund, welcher z.B. Daten hält oder Business-Logik bereitstellt oder User authentisiert 📖 **Backend**
- Kommunikation mit dem Backend findet i.d.R. über ein (REST-) HTTP-API statt
- Datenformat oft JSON, seltener XML

HTTP: Hyper Text Transfer Protocol

Was heisst „zustandslos“?

- Zustandsloses Kommunikationsprotokoll
 - Transport über TCP/IP
 - Request/Response Muster (Anfrage/Antwort)
 - Anfragemethoden: GET, PUT, POST, DELETE
 - Nachrichten bestehen aus zwei Teilen: Header + Body
 - 0..n Headers: Key-Value Paare
 - Body (Content): beliebig (typischerweise Text)
 - Mit jeder Antwort liefert Server einen Statuscode
 - z.B. 200 = OK, 404 = Not Found, 418 = I'm a teapot, ...
(http://developer.android.com/reference/java/net/HttpURLConnection.html#HTTP_200_OK)

...no kidding! ;-)

HTTP Requests absetzen

- Standard-Klassen `URL` und `URLConnection` erlauben das Absetzen von HTTP-Requests mit Java-Bordmitteln
 - Verwendung mühsam, uraltes API (letztes Jahrhundert)
- Besser: Eine HTTP-Client-Library verwenden
 - "Headless Browser"
 - Empfohlen: `OkHttpClient` (<http://square.github.io/okhttp/>)

HTTP: Cleartext Traffic erlauben

Erinnerung: API-Changes für Android 9 von früherer Folie!

- Seit Android 9 (API 28) ist Kommunikation über HTTP (unverschlüsselt) per default unterbunden

```
java.io.IOException: Cleartext HTTP traffic to wherever.ch not permitted
    at com.android.okhttp.HttpHandler$CleartextURLFilter.checkURLPermitted(HttpHandler.java:115)
    at com.android.okhttp.internal.huc.HttpURLConnectionImpl.execute(HttpURLConnectionImpl.java:458)
    at com.android.okhttp.internal.huc.HttpURLConnectionImpl.connect(HttpURLConnectionImpl.java:127)
    at ch.hslu.mobpro.com_and_con.MultiAsyncTask.openHttpConnection(MultiAsyncTask.java:51)
    at ch.hslu.mobpro.com_and_con.MultiAsyncTask.doInBackground(MultiAsyncTask.java:34)
    at ch.hslu.mobpro.com_and_con.MultiAsyncTask.doInBackground(MultiAsyncTask.java:19)
    at android.os.AsyncTask$2.call(AsyncTask.java:333)
```

- Manchmal aber nicht möglich oder nicht erwünscht! Kann "clear text traffic" in Manifest explizit erlauben

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="Kommunikation und Nebenläufigkeit"
    android:theme="@style/AppTheme"
    android:usesCleartextTraffic="true"
    tools:ignore="GoogleAppIndexingWarning">
```

(Lokales) Testen der Backend-Kommunikation

- Testen mit lokalem Server ist möglich!
 - Netzwerkkommunikation kann mit dem Emulator wie auch mit einem HW-Gerät getestet werden, ohne dass Server im Internet erreichbar sein müssen
- Vorgehen
 1. Webserver/Service auf Entwicklermaschine installieren
 2. Android-Applikation starten und verbinden
 - a) Im Emulator bezeichnet **10.0.2.2** die IP-Adresse des Hostrechners, auf dem der Emulator läuft
 - b) Auf HW-Gerät muss auf die IP-Adresse des Hostrechners verbunden werden (gleiches WLAN)

Manifest: Berechtigungen

- Für Internet-Zugriff (und Netzwerkstatus) muss Berechtigung vorliegen (resp. deklariert sein)
- Bekannter Mechanismus: Im Manifest eintragen

```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Hinweis: Nach Eintrag im Manifest muss ggf. App komplett neu installiert werden!



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

JSON-Webservices mit Retrofit konsumieren

"Definition" eines HTTP-Webservice

- Oft gesehen: Gesamte Information (d.h. Pfad und gewünschte Operation) in URI verpackt:
 - z.B.: GET http://foobar.io/?action=delete&itemId=23
 - Verwendet typischerweise nur eine bis wenige HTTP-Methoden (typisch: nur GET und ggf. POST)
 - Verwendet also auch GET um etwas zu löschen oder um Daten zu schicken (als Query-Param-Value)
- Besser: REST Semantik verwenden, d.h. HTTP-Methoden für gewünschte CRUD Operation verwenden
 - Siehe nächste Folie

REST-ful Webservices

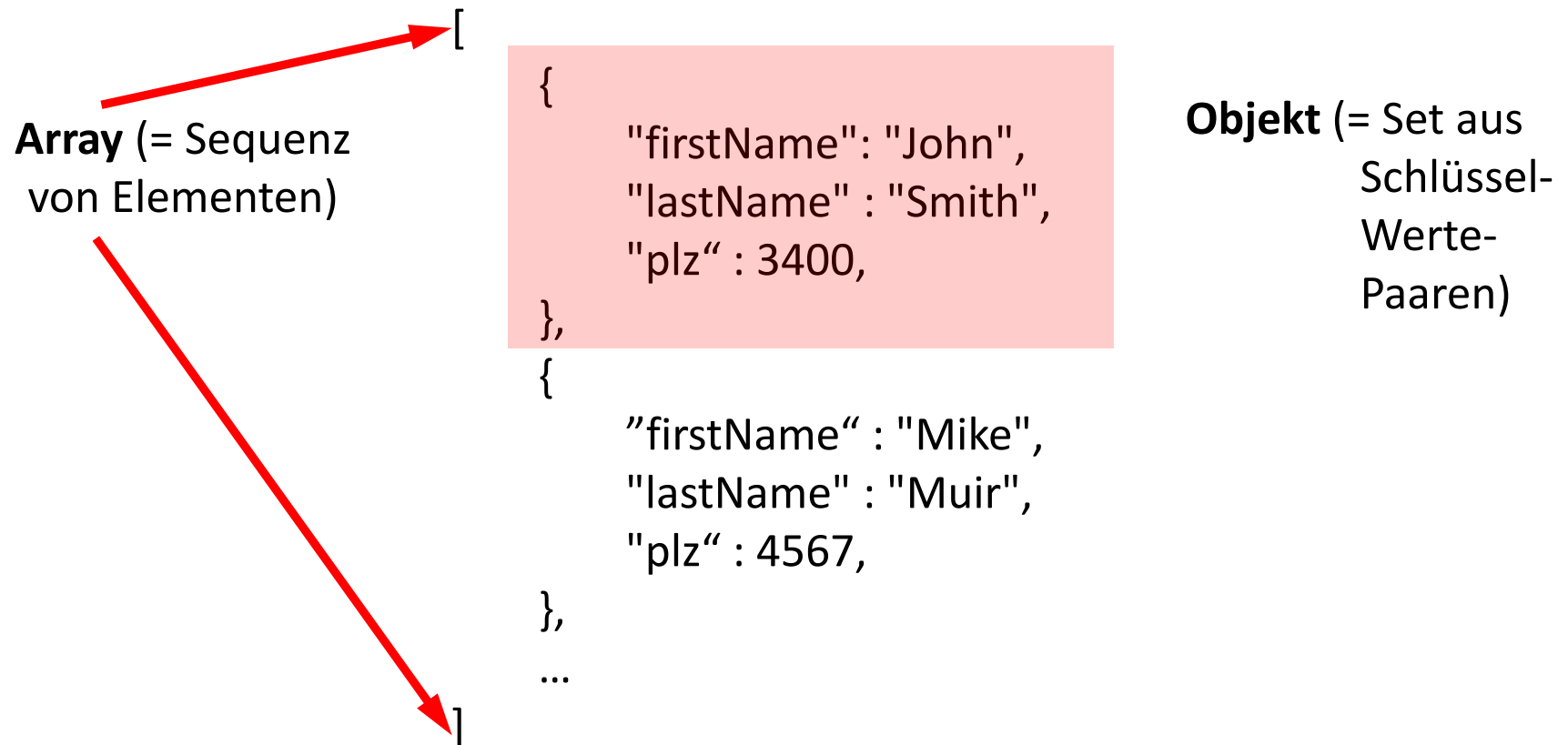
- Webservice auf der Basis von HTTP
- Grundidee (in purer Form)
 - Base-URL = Ressourcensammlung (<http://directory.com/contacts/>)
oder einer einzelnen Resource (<http://directory.com/contacts/17>)
 - HTTP-Methode = Operation auf Daten (GET, PUT, POST, DELETE)
 - Antwort-Datenformat = XML, JSON, ...

Wir erinnern uns:
Content-Provider!

Resource	GET	PUT	POST	DELETE
Collection URI, such as http://directory.com/contacts/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URL is usually returned by the operation.	Delete the entire collection.
Element URI, such as http://directory.com/contacts/17	Retrieve a representation of the addressed collection member, expressed in an appropriate media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Mini-Exkurs: Datenformat JSON

Mehr zum Standard und Format: <http://json.org>



Json vs. Kotlin (Siehe Übung 4)

■ Unsere Demo-Schnittstellen:

- <https://wherever.ch/hslu/rock-bands/all.json>
- [https://wherever.ch/hslu/rock-bands/info/\[CODE\].json](https://wherever.ch/hslu/rock-bands/info/[CODE].json)
 - z.B. für die Foo Fighters: <https://wherever.ch/hslu/rock-bands/info/ff.json> 📄

```
[  
  {  
    "name" : "AC/DC",  
    "code" : "acdc"  
  },  
  {  
    "name" : "Coldplay",  
    "code" : "cp"  
  },  
  {  
    "name" : "Foo Fighters",  
    "code" : "ff"  
  },  
]
```

data class BandCode(
 val name: String,
 val code: String
)

```
{  
  "name" : "Foo Fighters",  
  "foundingYear" : 1994,  
  "homeCountry" : "USA",  
  "bestOfCdCoverImageUrl" : "http  
}
```

data class BandInfo(
 val name: String,
 val foundingYear: Int,
 val homeCountry: String,
 val bestOfCdCoverImageUrl: String?
)

JSON-Parsing mit Moshi

- Moshi ist ein JSON-to-Java-Mapper
 - Bildet JSON-Strukturen auf äquivalente Java- bzw. Kotlin Klassen ab (ähnlich ORM, wie bei Room gesehen)
- Wir verwenden Moshi direkt in Kombination mit Retrofit um unsere JSON-Daten in Kotlin-Objekte zu konvertieren
 - Code-Beispiel siehe später
- <https://github.com/square/moshi>

HTTP-Anfragen mit Retrofit

- Aufrufe von Hand absetzen und JSON von Hand parsen & mappen ist rel. "low level"
- Analog zu den DAO-Interfaces in Room für DB-Zugriff möchten wir das Backend als Interface abstrahieren, um HTTP-Anfrage mit Java-Anfrage zu ersetzen: Das bietet Retrofit 😊

```
interface BandsApiService {
```

```
  @GET( value: "all.json")
```

```
  fun getBandNames(): Call<List<BandCode>>
```

Holt <https://wherever.ch/hslu/rock-bands/all.json> (Rest der URL kommt von Base-URL von Retrofit 😊)

(Siehe nächste Folie)

- <https://square.github.io/retrofit/>

Retrofit Konfiguration und Aufruf, inkl. Callback

Erzeugung von unserer Retrofit-Factory

```
private val retrofit = Retrofit.Builder()
    .client(OkHttpClient().newBuilder().build())
    .addConverterFactory(MoshiConverterFactory.create())
    .baseUrl(baseUrl: "https://wherever.ch/hslu/rock-bands/")
    .build()
```

Verwende Moshi
als Mapper

Base-URL (Präfix für alle
erzeugten Services)

Erzeugung von unserer Service-Instanz

```
private val bandsService = retrofit.create(BandsApiService::class.java)
```

Typischerweise ein Service pro
"Domäne" (vgl. DAO)

Wiederverwenden!

Verwendung vom Service

```
val call = bandsService.getBandNames()
call.enqueue(object : Callback<List<BandCode>> {
```

object: Erzeugung
von anonymer Klasse
aus Interface

```
    override fun onResponse(call: Call<List<BandCode>>, response:
        if (response.code() == HttpURLConnection.HTTP_OK) {
            bands.value = response.body().orEmpty()
        }
    })
```

Retrofit & Moshi: Dependencies

- Gradle

implementation 'com.squareup.retrofit2:retrofit:2.5.0'

implementation 'com.squareup.retrofit2:converter-moshi:2.4.0'

- Retrofit basiert auf OkHttp, diese Bibliothek ist somit automatisch vorhanden 😊



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Übung 4

Zur Übung 4: Demo

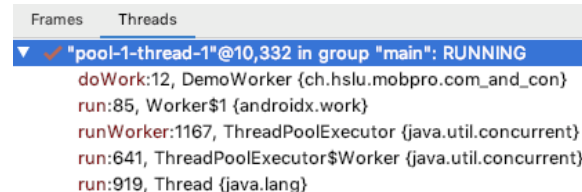
■ Nebenläufigkeit

- Blockier-Knopf 7"
- Warten in eigenem Thread



```
Frames  Threads
▼ ✓ "hsluDemoThread"@10,249 in group "main": RUNNING
    run:54, MainActivity$createDemoThread$1 {ch.hslu.mobpro.com_and_con}
```

- Warten in eigenem Worker



```
Frames  Threads
▼ ✓ "pool-1-thread-1"@10,332 in group "main": RUNNING
    doWork:12, DemoWorker {ch.hslu.mobpro.com_and_con}
    run:85, Worker$1 {androidx.work}
    runWorker:1167, ThreadPoolExecutor {java.util.concurrent}
    run:641, ThreadPoolExecutor$Worker {java.util.concurrent}
    run:919, Thread {java.lang}
```

■ HTTP-Kommunikation

- JSON-Daten mit Retrofit
- ViewModel mit LiveData & Observer
- Bilder anzeigen mit Picasso

HSLU Mobile Programming

Nebenläufigkeit (Threads & Worker)

GUI 7 SEKUNDEN BLOCKIEREN

DEMO-THREAD STARTEN

DEMO-WORKER STARTEN

Kommunikation (HTTP & JSON)

SERVER-ANFRAGE STARTEN

VIEW MODEL ZURÜCK SETZEN

#Bands = 9

ZEIGE BAND-AUSWAHL AN

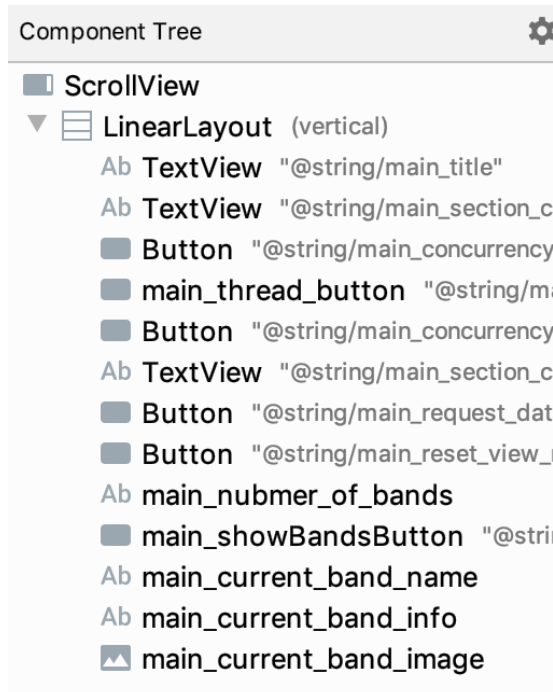
Foo Fighters

USA, Gründung: 1994



Eine Activity, ein Layout

- Übung 4
 - 1 Activity: MainActivity
 - 1 Layout-Datei: activity_main.xml
 - 0 Fragmente



HSLU Mobile Programming

Nebenläufigkeit (Threads & Worker)

GUI 7 SEKUNDEN BLOCKIEREN

DEMO-THREAD STARTEN

DEMO-WORKER STARTEN

Kommunikation (HTTP & JSON)

SERVER-ANFRAGE STARTEN

VIEW MODEL ZURÜCK SETZEN

#Bands = 9

ZEIGE BAND-AUSWAHL AN

Foo Fighters

USA, Gründung: 1994



BandsViewModel

```
class BandsViewModel : ViewModel() {
```

```
    val bands: MutableLiveData<List<BandCode>> = MutableLiveData()
    val currentBand: MutableLiveData<BandInfo?> = MutableLiveData()
```

Die aktuell
vorhandenen
Bands

Die aktuell Dargestellte Band ODER null (wenn
keine Band aktuell dargestellt werden soll),
daher ein Optional-Typ

- BandsViewModel setzen in MainActivity:

```
private val bandsViewModel: BandsViewModel by viewModels()
```

by: "Delegated Property", eine Kotlin-Spezialität, siehe
sehr gute Kotlin-Doku dazu inkl. Code-Bsp:
<https://kotlinlang.org/docs/delegated-properties.html>

"Kotlin-Magic" aus
den Kotlin Android
Extentions, siehe
nächste Folie 😊

Kotlin-Extension: viewModels()

private val bandsViewModel: BandsViewModel by viewModels()

- spart uns "Umweg" über ViewModelProvider...
 - Vereinfachung aus den "Kotlin Extensions for Android"
 - Ist eine "Extension Function" für die Klasse `ComponentActivity`

...erinnern Sie sich an das Beispiel mit `Int.myPrettyPrint()` aus dem Kotlin-Intro in SW01?

Siehe Folie 21 von "Kotlin (Intro)" 😊

- ginge alternativ bzw. traditionell so:

```
val bandsViewModel = ViewModelProvider(owner: this).get(BandsViewModel::class.java)
```

Kotlin: View Binding neu mit binding-Klasse

■ Beispiel-Code:

```
android {  
    ...  
    buildFeatures {  
        viewBinding true  
    }  
}
```

Automatisch generiert
aus activity_main.xml

Nameskonvention!
(_ vs. CamelCase)

```
import ch.hslu.mobpro.com_and_con.databinding.ActivityMainBinding
```

```
class MainActivity : AppCompatActivity() {
```

Deklaration vom
binding-Property

```
    private lateinit var binding: ActivityMainBinding
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        this.binding = ActivityMainBinding.inflate(layoutInflater)
```

```
        setContentView(this.binding.root)
```

Initialisierung vom binding-
Property mittels inflate

```
        this.binding.mainCurrentBandName.text = bandName
```

ContentView von
dieser Activity
setzen

Verwendung der binding-
Instanz

Name generiert aus der ID
main_current_band_name in
activity_main.xml

```
<TextView
```

```
    android:id="@+id/main_current_band_name"  
    android:layout_width="match_parent"
```

Binding: Gute Migrationsunterstützung

- Wie gewohnt gute Doku, direkt mit Bsp. Code

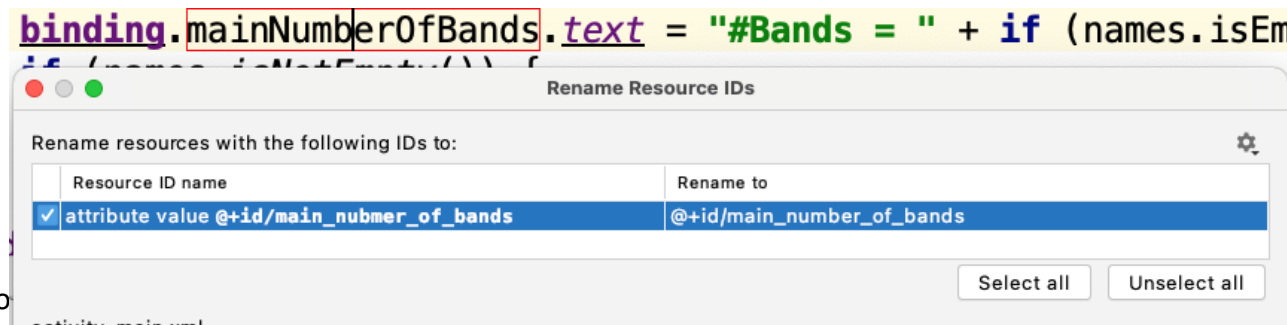
- <https://developer.android.com/topic/libraries/view-binding/migration>

```
// Reference to "name" TextView using synthetic properties.
name.text = viewModel.nameString

// Reference to "name" TextView using the binding class instance.
binding.name.text = viewModel.nameString
```

- Auch wie gewohnt gute IDE-Unterstützung

- AndroidStudio "kennt" die Namenskonventionen und passt bei Umbenennungen im Code automatisch auch ID in Layout.xml-Dateien gemäss Namensscheme an 😊



LiveData: Observer registrieren (in Activity)

Erinnerung: Der Typ von `currentBand` ist `MutableLiveData<BandInfo?>`

Direkte Instanziierung vom Interface `Observer<T>` mit Trailing-Lambda-Syntax... phua! 🤪

```
bandsViewModel.getCurrentBand().observe(owner: this, Observer { bandInfo ->
    binding.mainCurrentBandName.text = bandInfo?.name ?: ""
    if (bandInfo != null) {
        binding.mainCurrentBandInfo.text = bandInfo.homeCountry + ", Gründung:
```

Erinnerung: Zugriff auf `TextView` via `binding` und mit dieser ID aus `layout.xml` 😊

Bild anzeigen mit Picasso

- Bild von URL asynchron holen & in `ImageView` anzeigen:
geht kompakt mit Picasso

- <https://square.github.io/picasso/>

- Beispiel-Code (aus Activity):

...praktisch ein langer Ein-Zeiler! ;-)

```
Picasso.get()  
    .load(bandInfo.bestOfCdCoverImageUrl)  
    .into(binding.mainCurrentBandImage)
```

ID der `ImageView` ausm
layout.xml 😊

Übung vorzeigen heute...