Android fejlesztés

Kulcs-érték tárolás, fragmentek, hálózati kommunikáció alapok

peter.ekler@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

Miről volt szó az előző alkalommal? ☺

- RecyclerView további képességek
  - >LayoutManager
  - >KenburnsView
  - >Touch gesztusok
- Perzisztens adattárolás
- SQLite
- ORM
- Room

Tartalom

- Kulcs érték tárolás
  - >SharedPreferences
- Fragmentek
- Hálózati kommunikáció
- HTTP kapcsolatok kezelése

Adattárolási megoldások

- Androidon minden igényre van beépített megoldás:
  - **SQLite adatbázis**: strukturált adatok tárolására
  - ***SharedPreferences***: alaptípusok tárolása kulcs-érték párokban
  - **Privát lemezterület**: nem publikus adatok tárolása a fájlrendszerben
  - **SD kártya**: nagy méretű adatok tárolása, nyilvánosan hozzáférhető
  - **Hálózat**: saját webszerveren vagy felhőben tárolt adatok

SharedPreferences

Beállítások mentése hosszú távra

SharedPreferences

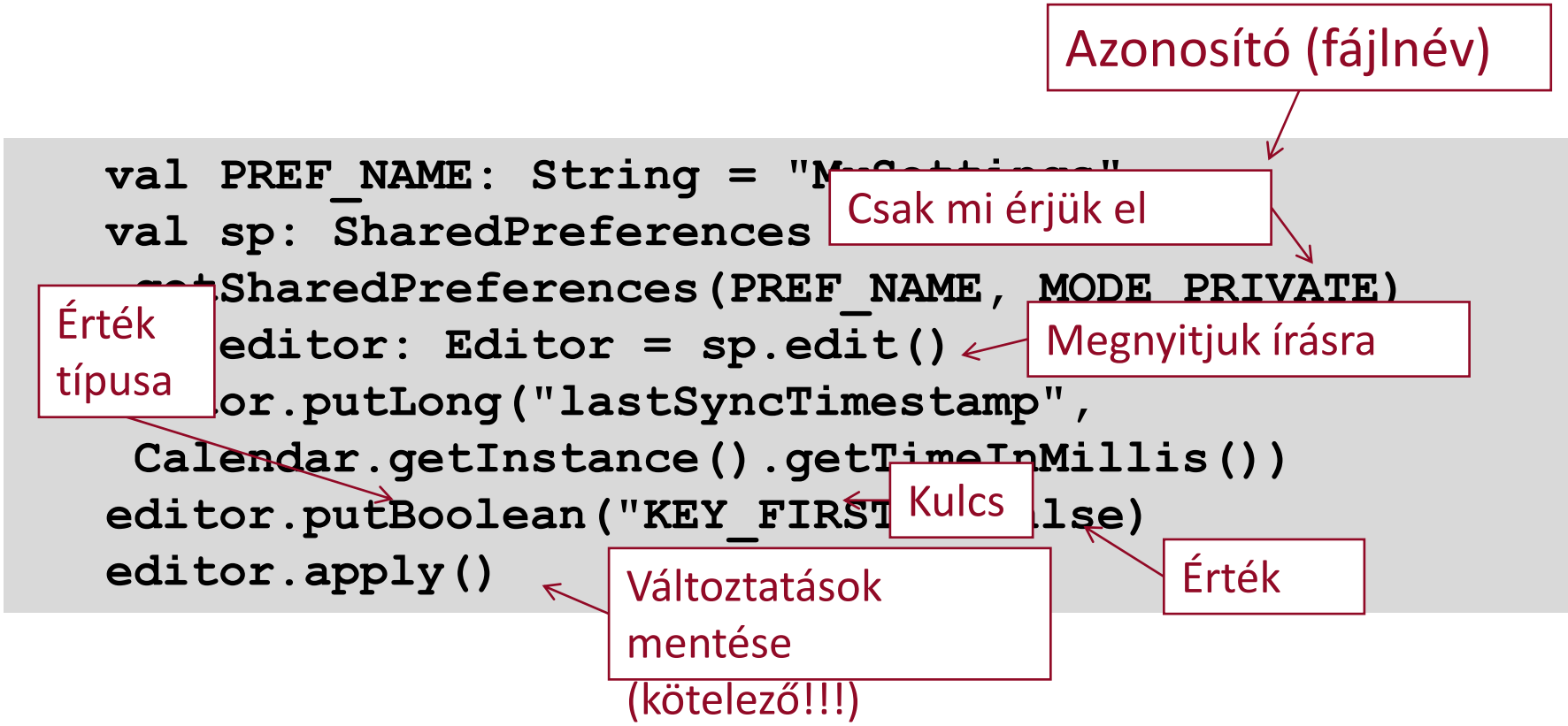
- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
  - Típusok: *int, long, float, String, boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
  - ***MODE_PRIVATE***: csak a saját alkalmazásunk érheti el
  - ***MODE_WORLD_READABLE***: csak a saját alkalmazásunk írhatja, bárki olvashatja
  - ***MODE_WORLD_WRITABLE***: bárki írhatja és olvashatja
- Megőrzi tartalmát az alkalmazás és a telefon újraindítása esetén is
  - Miért?

SharedPreferences

- Ideális olyan adatok tárolására, melyek primitív típussal könnyen reprezentálhatók, pl:
  - Default beállítások értékei
  - UI állapot
  - Settings-ben megjelenő adatok (innen kapta a nevét)
- Több ilyen *SharedPreferences* fájl tartozhat egy alkalmazáshoz, a nevük különbözteti meg őket
  - **`getSharedPreferences(name: String, mode: Int)`**
  - Ha még nem létezik ilyen nevű, akkor az Android létrehozza
- Ha elég egy SP egy Activity-hez, akkor nem kötelező elnevezni
  - **`getPreferences(mode: Int)`**

SharedPreferences írás

- Közvetlenül nem írható, csak egy ***Editor*** objektumon keresztül




```
val PREF_NAME: String = "M
val sp: SharedPreferences
 etSharedPreferences(PREF_NAME, MODE_PRIVATE)
    editor: Editor = sp.edit()
   or.putLong("lastSyncTimestamp",
    Calendar.getInstance().getTimeInMillis())
editor.putBoolean("KEY_FIRST  lse)
editor.apply()
```

SharedPreferences olvasás

- Az *Editor* osztály nélkül olvasható, közvetlenül a SharedPreferences objektumból
- Ismernünk kell a kulcsok neveit és az értékek típusát
  - > Emiatt sem alkalmas nagy mennyiségű adat tárolására

```
PREF_NAME = "MySettings"
val sp =
 getSharedPreferences(PREF_NAME, MODE_PRIVATE)
val lastSaved: Long = sp.getLong("lastSaved",0)
val isFirstRun: Boolean =
  sp.getBoolean("KEY_FIRST",true)
```

Tudni kell a kulcsot

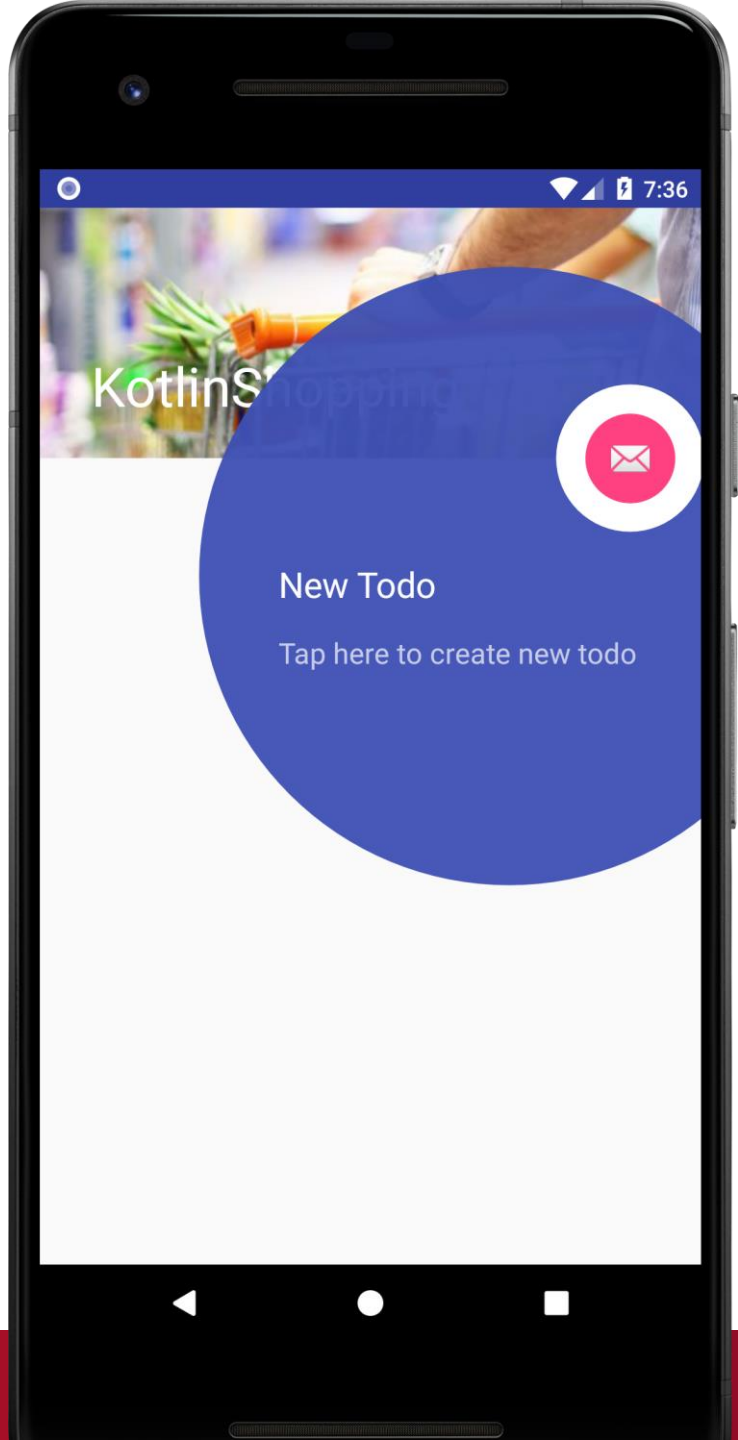
És a típust is!

Alapértelmezett érték

- Egy hasznos metódus:
  - > **`sp.getAll()`** –minden kulcs-érték pár egy Map objektumban
  - > Tutorial lib: https://github.com/sjwall/MaterialTapTargetPrompt

Gyakoroljunk!

Egészítsük ki a bevásárló lista alkalmazást, hogy csak legelső induláskor mutasson használati tippeket.

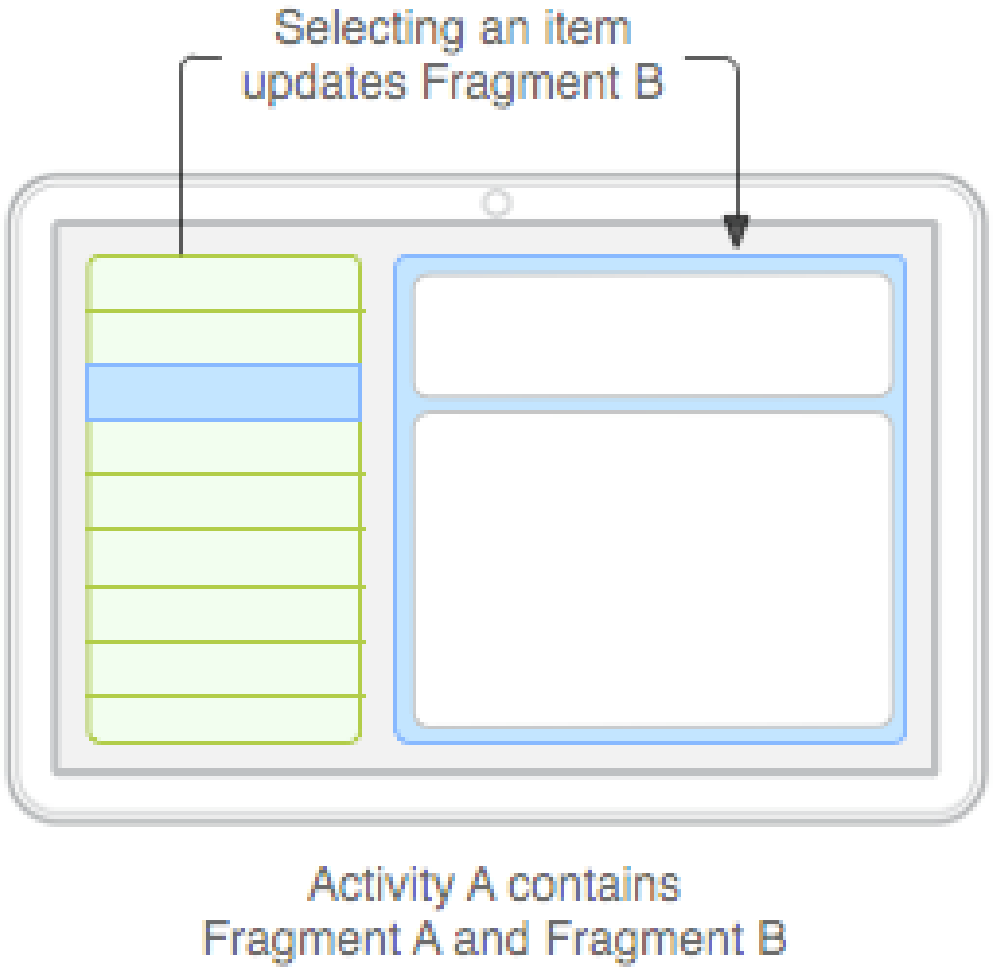


FRAGMENT-EK

Fragmentek

- Mik azok a Fragmentek?
  - Elsősorban: A képernyő egy nagyobb részéért felelős objektumok
  - Továbbá: A háttérben munkát végző objektumok is lehetnek
- Miért kellenek nekünk?
  - Nagy képernyőméret = több funkció egy képernyőn = bonyolultabb Activity-k
  - Fragment-ekkel modulárisabb, rugalmasabb architektúra építhető

Fragmentek


Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B

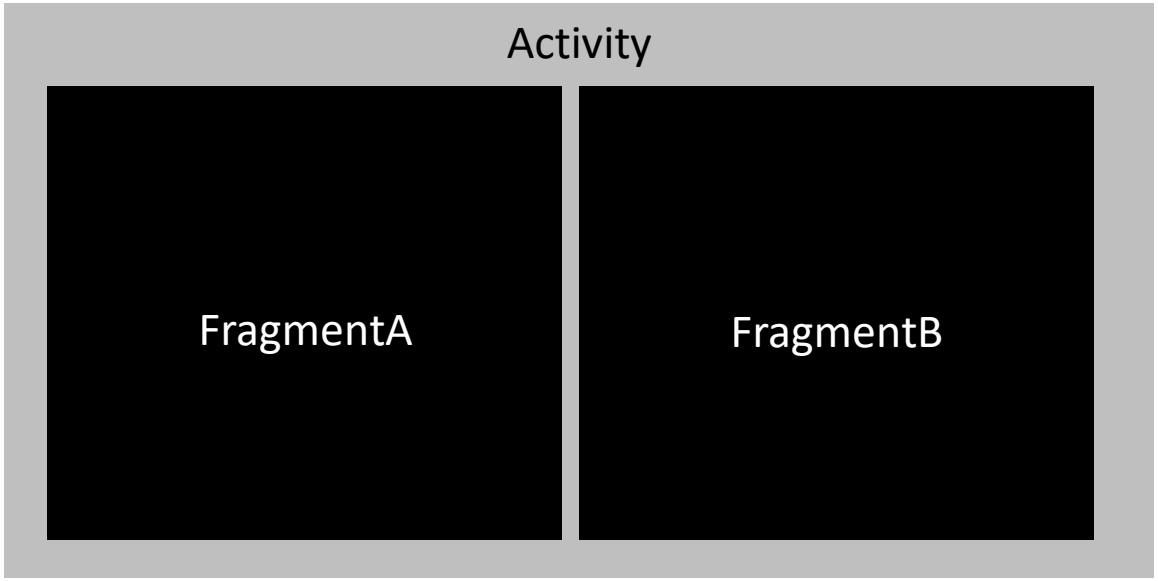
Fragmentek

- Miben másabb mint az Activity?
  - Kisebb granualitás, nem mindig teljes képernyő egy fragment
  - Az életciklusa nem mindig egyezik, pl. le lehet csatolni egy fragmentet úgy, hogy az activity előtérben marad.

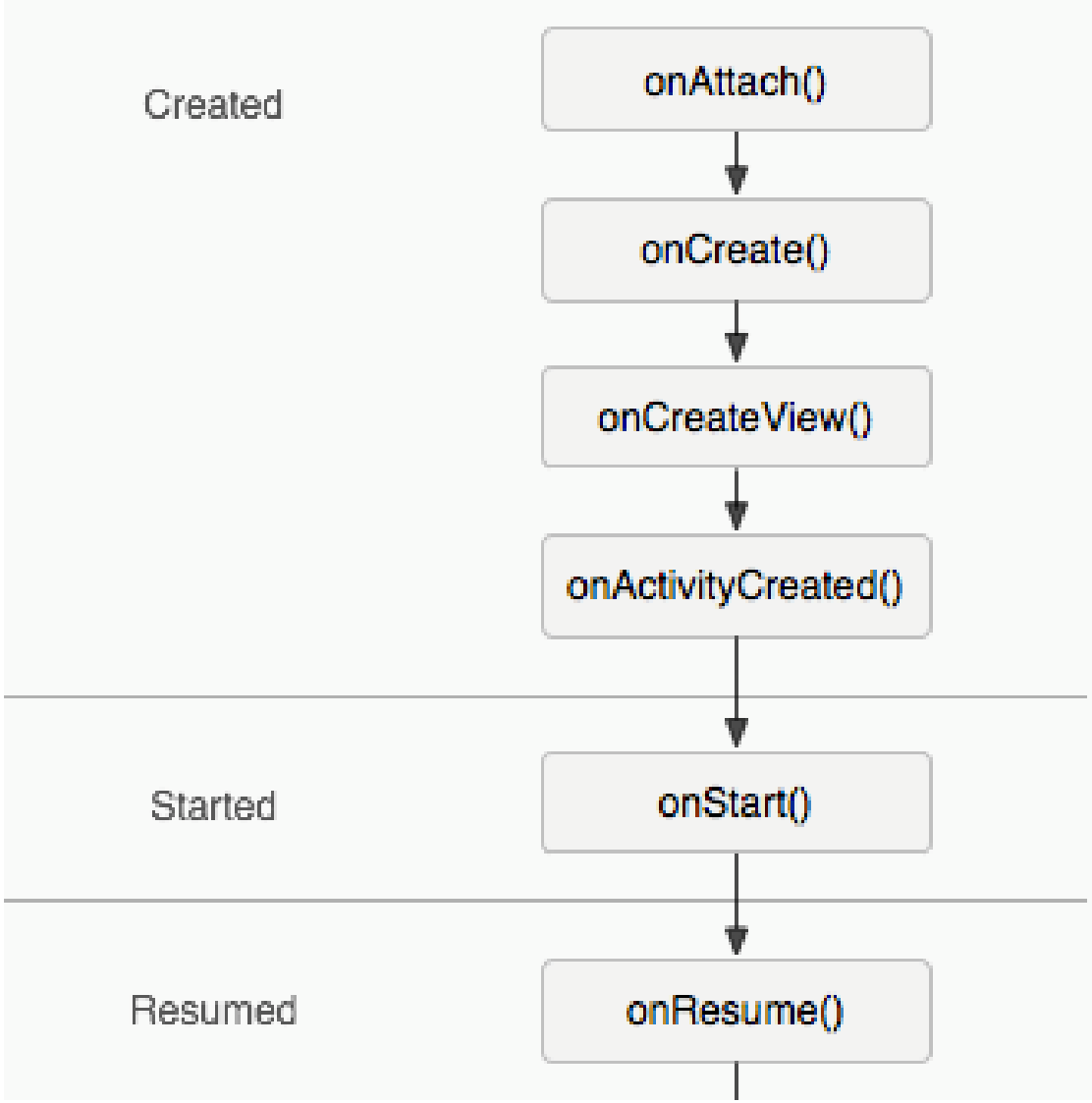
- Miben másabb mint egy Custom View
  - Összetett életciklus, mely az activity-t is figyelembe veszi
  - Előny, de hátrány is lehet!

Fragment és Activity

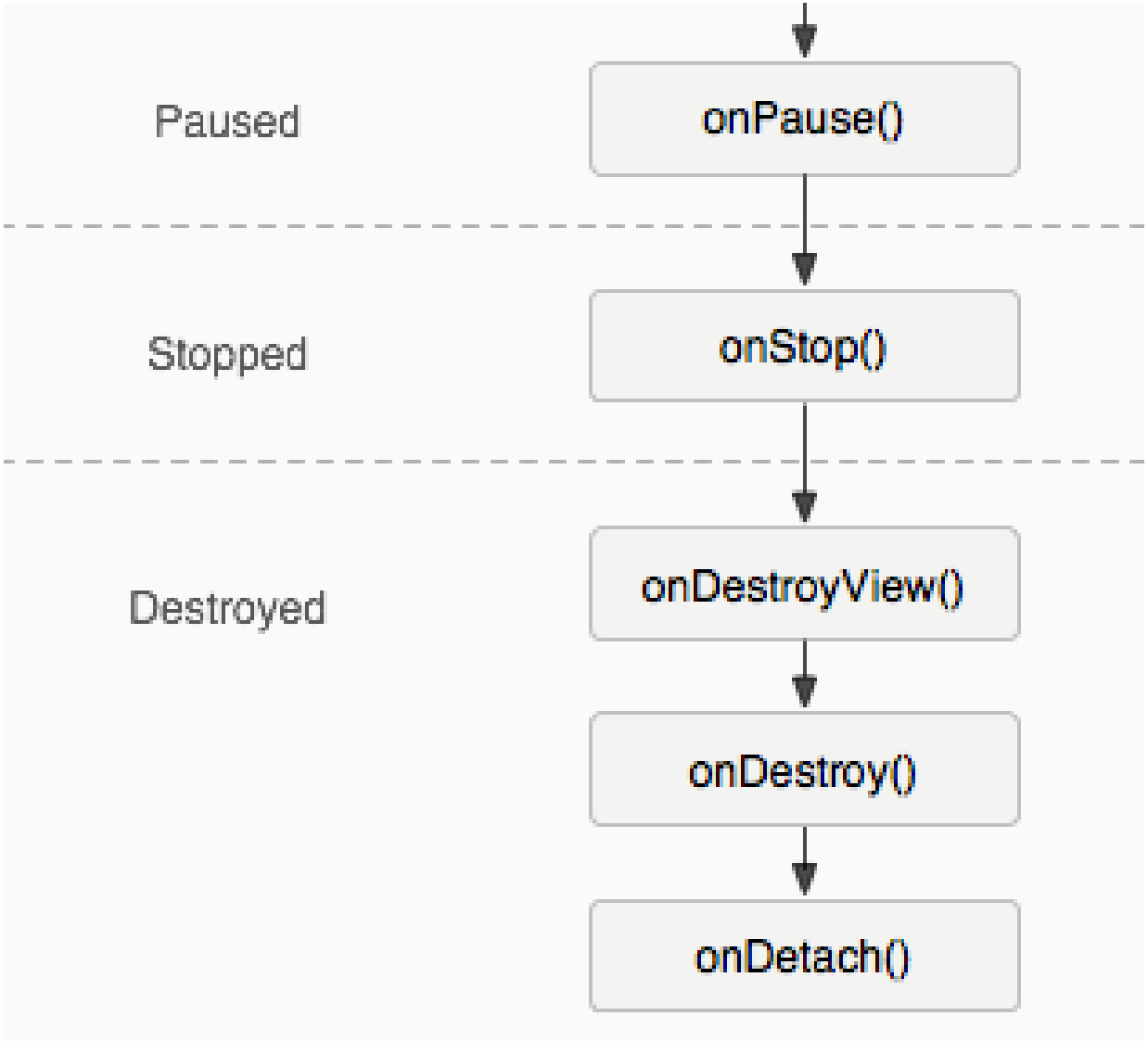
- Egy Fragment mindig egy Activity-hez csatoltan jelenik meg
- Az Activity életciklusa ráhatással van a Fragmentére

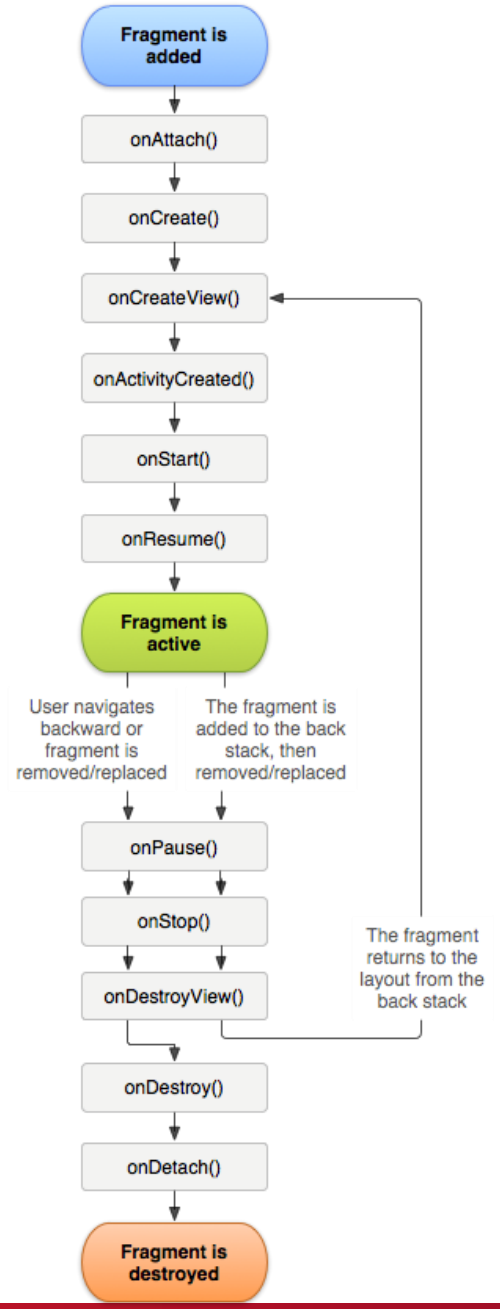


Fragment életciklus I.



Fragment életciklus II.




Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

Nem alkalmazás komponens

- Mi hozzuk létre, nem a rendszer
- Nem kell feltüntetni a manifestben
- Nem intentekkel kommunikálunk
  - Mezei függvényhívás az objektumon
  - pl. Activity hívja a Fragment objetumon
  - getActivity()/activity property – szülő activity

UI Fragment készítése…

- A megjelenítendő View-hierarchiát az .onCreateView() metódusban kell visszaadni

```
class FragmentProfile : Fragment() {

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        val rootView = inflater.inflate(R.layout.fragment_profile, container, false)
        return rootView
    }

}
```

… és csatolása

- Statikusan
  - Az Activity-hez tartozó layout-ban beégetjük a Fragment-et, nem módosítható később
  - <fragment …/> tag
- Dinamikusan
  - Az Activity futás közben tölti be a megfelelő Fragment-eket, adott ViewGroup-okba
  - Fragment-Tranzakciókkal módosítható

Statikus csatolás példa

```
<fragment class="hu.bme.aut.fragment.MenuListFragment"
    android:tag="MenuListFragment"
    android:layout_width="0dip"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

A FragmentManager

- A FragmentManager-el menedzselhetők a Fragment-ek
  - Activity: supportFragmentManager property
  - FragmentTransaction indítása
  - Aktív Fragment-ek közt keres
    - Tag alapján
    - ID alapján
  - **Fragment-stack-et menedzseli**

A FragmentManager

- Az activitynek a FragmentActivityből kell származnia – Neki van FragmentManagerje
- Activity.supportFragmentManager== fragment.fragmentManager
- Kezeli a fragmenteket, backstacek, állapotmentést ... stb.
- Fragmenten belül fragmentek, lehet: Fragment.childFragmentManager

FragmentTransaction osztály I.

- Ezen keresztül módosíthatók az aktív Fragment-ek
- A FragmentManager .beginTransaction() metódusával indítható
- Fontosabb műveletek:
  - .add(...) / .remove(...) / .replace(...)
    - Fragment példányok le- és felcsatolása az adott Activity-re
  - .commit()
    - Tranzakció végrehajtása

FragmentTransaction osztály II.

- .show(...) / .hide(...)
  - Fragment példány elrejtése / újra megjelenítése
- .setTransition(...) / .setCustomAnimations(...)
  - A tranzakció végrehajtásakor lejátszandó animáció beállítása
- .addToBackStack(...)
  - Rákerüljön-e a FragmentTransaction backstack-re a tranzakció?
- .commit()
  - Tranzakció végrehajtása

FragmentTransaction példa I.

- Fragment kicserélése:

```
val fragment=DetailsFragment.newInstance()

val ft = supportFragmentManager.beginTransaction()
ft.replace(R.id.fragmentContainer, fragment,
DetailsFragment.TAG)
ft.commit()
```

FragmentTransaction példa II.

- Fragment hozzáadása, a tranzakciót a backstack-re téve:

```
val fragment=DetailsFragment.newInstance()

val ft = supportFragmentManager.beginTransaction()
ft.add(R.id.fragmentContainer, fragment, TAG)
ft.setCustomAnimations(R.anim.slide_in_top,R.anim.slide_out_bot
tom)
ft.addToBackStack(null)
ft.commit()
```

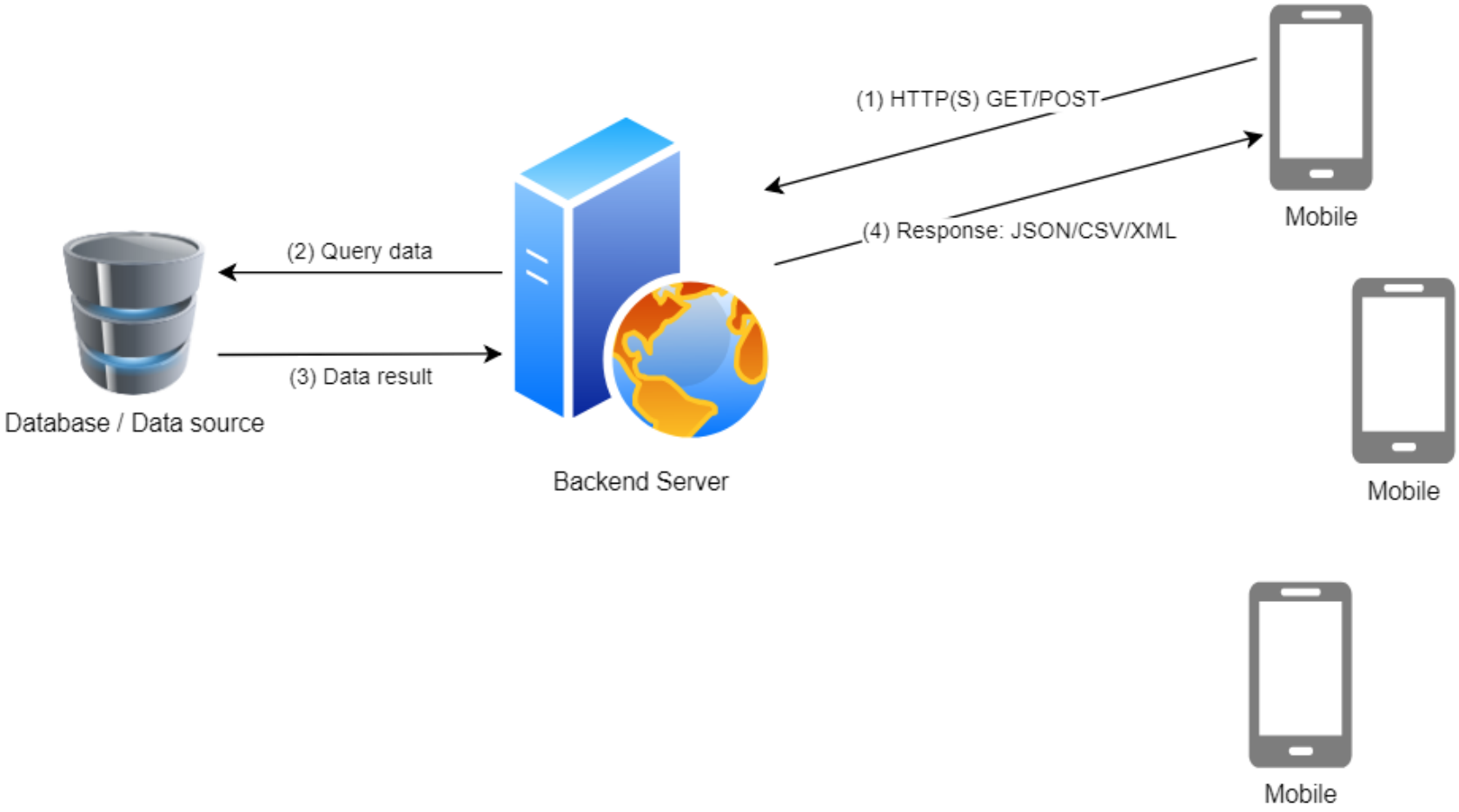
HÁLÓZATI KOMMUNIKÁCIÓ

Hálózati kapcsolatok

- Short-range
  - NFC
  - Bluetooth
  - Nearby API
  - Wifi-Direct
- Long-range/Internet
  - HTTP
  - TCP/IP-UDP

HTTP Kapcsolatok kezelése

Tipikus Architektúra



HTTP kommunikáció Android platformon

- Egyik leggyakrabban használt kommunikációs technológia
- HTTP metódusok
  - GET, POST, PUT, DELETE
- Teljes körű HTTPS támogatás és certificate import lehetőség
- REST kommunikáció támogatása (Representational State Transfer)

HTTP kapcsolatok kezelése

- Szükséges engedély:
  <uses-permission android:name=
  "android.permission.INTERNET"/>
- **Új szálban kell megvalósítani a hálózati kommunikáció hívást!**
- Ellenőrizzük a HTTP válasz kódot:
  - https://restfulapi.net/http-status-codes/
  - http://www.w3.org/Protocols/HTTP/HTRESP.html
- HTTP REST
  - http://en.wikipedia.org/wiki/Representational_state_transfer
- Ügyeljünk az alapos hibakezelésre
- HTTP GET példa:
  - http://numbersapi.com/10/math

HTTP könyvtárak Android-on

- A rendszer két megvalósítás is tartalmaz:
  - Standard Java HTTP implementáció (***HttpURLConnection***)
  - **Apache** HTTP implementáció (*HttpClient*)
- Apache Depricated – Ne használjuk, ki is vették
- Igazán egyik sem jó
  - 3rd party megoldás – Square OkHttp
  - http://square.github.io/okhttp/

HTTP GET - HttpURLConnection

```
fun httpGet(urlAddr: String) {
    var reader: BufferedReader? = null
    try {
        val url = URL("http://mysrver.com/api/getitems")
        val conn = url.openConnection() as HttpURLConnection
        reader = BufferedReader(InputStreamReader(conn.inputStream))
        var line: String?
        do {
            line = reader.readLine()
            System.out.println(line)
        } while (line != null)
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        if (reader != null) {
            try {
                reader.close()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }
}
```

HTTP POST- HttpURLConnection

```
fun httpPost(urlAddr: String, content: ByteArray) {
    // ...
    var os: OutputStream? = null
    try {
        val url = URL("http://mysrver.com/api/refreshitems")
        val conn = url.openConnection() as HttpURLConnection
        conn.requestMethod = "POST"
        conn.doOutput = true
        conn.useCaches = false
        os = conn.outputStream
        os.write(content)
        os.flush()
        // …
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        // …
        if (os != null) {
            try {
                os.close()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }
}
```

Timeout értékek beállítása

- Fontos, hogy minden hálózati kommunikáció megfelelő módon kezelje a timeout-ot
- Timeout a kapcsolat megnyitásra
- Timeout az eredmény kiolvasására
- Példa:

```
…
val conn = url.openConnection() as HttpURLConnection
…
conn.setConnectTimeout(10000)
conn.setReadTimeout(10000)
…
```

Header paraméterek beállítása

- Egyszerű Header beállítása:

```
val conn = url.openConnection() as HttpURLConnection
conn.setRequestProperty(”[KEY]”,”[VALUE]”)
```

- Cookie beállítása:

```
val conn = url.openConnection() as HttpURLConnection
conn.setRequestProperty(”Cookie”,”sessionid=abc;age=15”)
```

- Összetett példa:

```
val conn = url.openConnection() as HttpURLConnection
conn.setRequestProperty(„Content-Type”,”application/json”)
conn.setRequestProperty(”Cookie”,”sessionid=abc;age=15”)
```

Aszinkron kommunikáció

UI módosítása más szálból

- Az alkalmazás indításakor a rendszer létrehoz egy úgynevezett *main* szálat (UI szál)
- Sokáig tartó műveletek blokkolhatják a felhasználói felületet, ezért új szálba kell indítani őket
- Az ilyen műveletek a végén az eredményt a UI-on jelenítik meg, **azonban** az Android a UI-t csak a fő szálból engedi módosítani!
- Több megoldás is szóba jöhet:
  - *Activity.runOnUiThread(Runnable)*
  - *View.post(Runnable)*
  - *View.postDelayed(Runnable, long)*
  - *Handler*
  - *AsyncTask és LocalBroadcast*
  - *Külső libek, pl. EventBus, Otto*
  - *REST külső lib: RetroFit*

Tipikus adatformátumok

Adatok küldése, válaszok feldolgozása

- Sokszor egy előre definiált formátumban/protokollon történik a kommunikáció kliens és szerver között
- Legtöbb esetben egy harmadik fél szerverétől kapott válasz is valamilyen jól strukturált formátumban érkezik
- Tipikus formátumok:
  - CSV (Comma Separated Value(s))
  - JSON (JavaScript Object Notation)
  - XML (Extensible Markup Language)
- Természetesen lehet saját protokoll is

JSON formátum

- Szintaktikai elemek: ’{’, ’]’ ,’[’, ’]’ ,’:’ ,’;’
- Példa:

```
{
   "keresztnev" : "Elek",
   "vezeteknev" : "Teszt",
   "kor" : 23,
   "cim" :
   {
      "utca" : "Baross tér",
      "varos" : "Budapest",
      "iranyitoszam" : "1087"
   },
   "telefon":
   [
      {
        "tipus"  : "otthoni",
        "szam": "123 322 1234"
      },
      {
        "tipus"  : "mobil",
        "szam": "626 515 1567"
      }
   ]
 }
```

JSON feldolgozás

- *JSONObject*:
  - JSON objektumok parse-olása
  - Elemek elérhetősége a kulcs megadásával:
    - `getString(String name)`
    - `getJSONObject(String name)`
    - `getJSONArray(String name)`
  - JSON objektum létrehozása *String*-ből vagy *Map*-ból
- *JSONArray*:
  - *JSONObject*-hez hasonló működés JSON tömbökkel
  - Parse-olás, elemek lekérdezése index alapján, hossz
  - Létrehozás például *Collection*-ból

JSON API minták

- *Currency Exchange:*
  - *https://api.exchangeratesapi.io/latest?base=HUF*
- OpenWeather
  - http://api.openweathermap.org/data/2.5/weather?q=Budapest,hu&units=metric&appid=f3d694bc3e1d44c1ed5a97bd1120e8fe
- TV Show Data API:
  - http://api.tvmaze.com/search/shows?q=stargate

REST API gyűjtemények

- https://github.com/toddmotto/public-apis
- https://github.com/abhishekbanthia/Public-APIs
- https://github.com/Kikobeats/awesome-api

Külső osztálykönyvtárak XML és JSON feldolgozásra

- XML:
  - SimpleXML
- JSON:
  - GSON
- REST API tesztelésére:
  - Postman Chrome Client

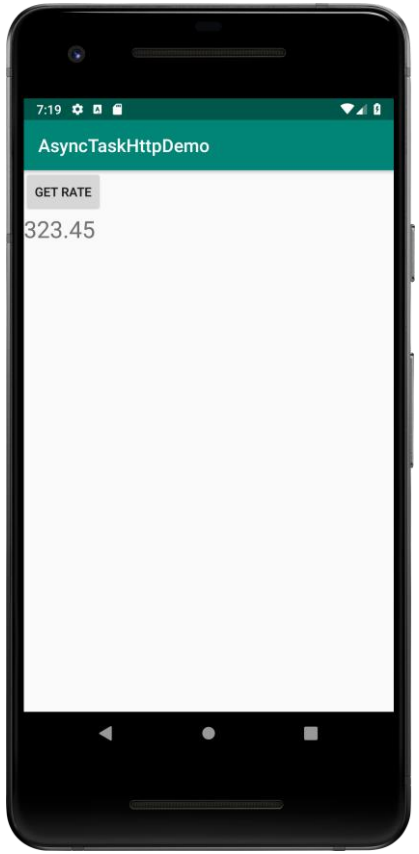
GSON POJO példa (Kotlin)

```
class PhoneInfo(
        @SerializedName("DeviceID")
        val deviceId: String,

        @SerializedName("OperatingSystem")
        val operatingSystem: String
)
```

REST API-k kezelése

Példa - Retrofit

- https://api.exchangeratesapi.io/latest?base=EUR
- Retrofit 2 + GSON



- HOST:
  - https://api.exchangeratesapi.io
- Path:
  - /latest
- Query paraméterek:
  - ? **base**=EUR&*key=value*

Retrofit

- HTTP API megjelenítése Java interface formában

```
interface ItemsService {
    @GET("/items/{item}/details")
    fun listItems(@Path("item") item: String): Call<List<Item>>
}
```

- Retrofit osztály a konkrét implementáció generálására

```
val retrofit = Retrofit.Builder()
        .baseUrl("https://api.myshop.com")
        .build()
val service = retrofit.create(ItemsService::class.java)
```

- Mindenhívás az *ItemsService* mehet szinkron és aszinkron módon:

```
val items: Call<List<Item>> = service.listItems("myItem")
```

Retrofit

- HTTP kérések leírása annotációkkal:
  - URL és query paraméterek
  - Body – objektum konverzió (JSON, protocol buffers)
  - Multipart request és file feltöltés
- Gradle:
  - implementation 'com.squareup.retrofit2:retrofit:2.4.0'
- További információk:
  - http://square.github.io/retrofit/

Retrofit – GSON támogatás

- Automatikus konverzió a háttérben
  - > Be kell állítani a Retrofitnak hogy mit használjon a konverzióhoz.

  - > val retrofit=Retrofit.Builder()
    .baseUrl("http://api.myserver.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

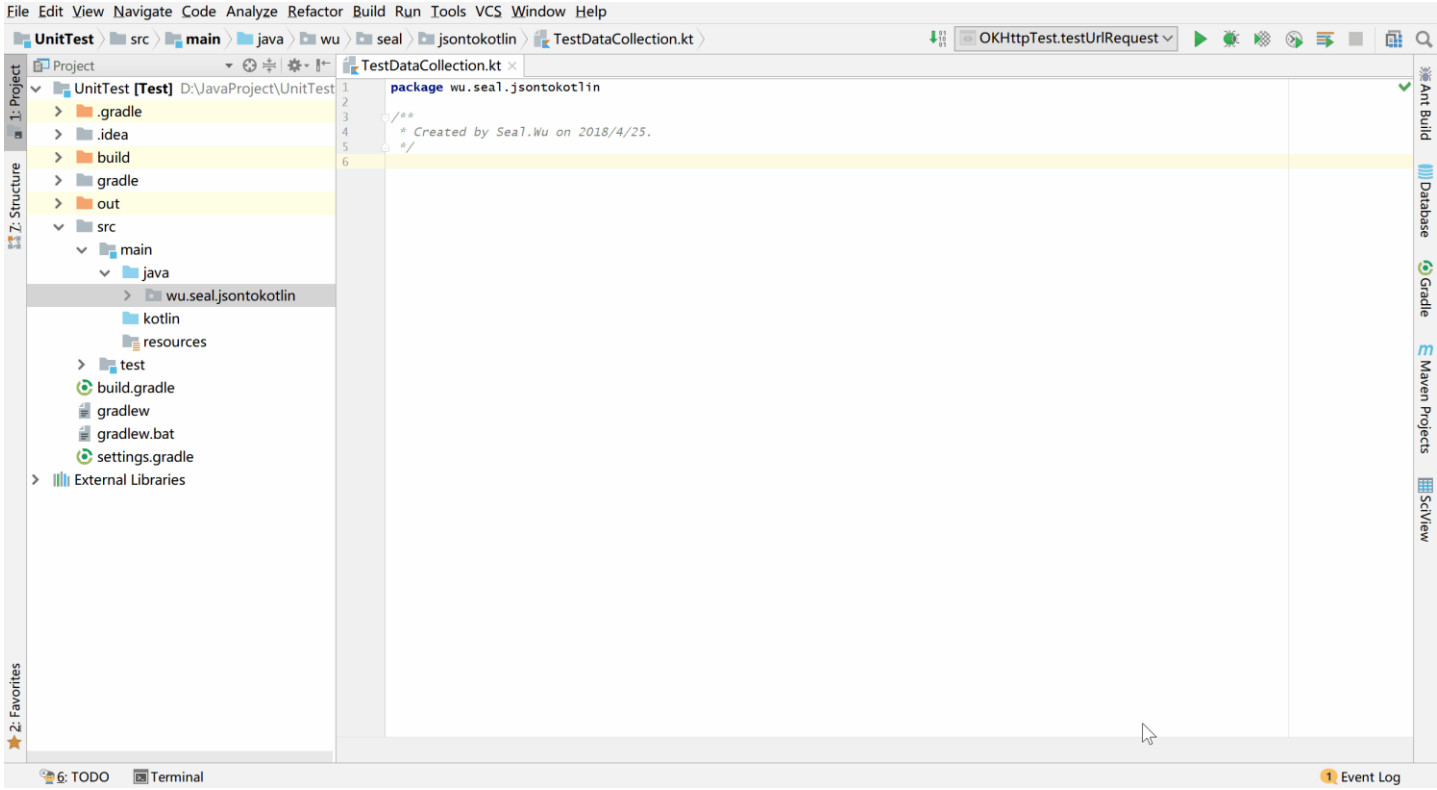
- Gradle:
  - > implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'com.squareup.retrofit2:converter-gson:2.4.0’

- További információk:
  - > http://square.github.io/retrofit/

Retrofit használatának lépései

Entitás vagy *data* class generálás JSON-ból

- *data* class, csak ha kellenek a *componentN* és egyéb függvények
- https://http4k-data-class-gen.herokuapp.com/
- https://github.com/wuseal/JsonToKotlinClass



Retrofit - Entitások / data class

```
data class MoneyResult(
    var date: String,
    var rates: Rates,
    var base: String
)

data class Rates(
    var BGN: Double,
    var CAD: Double,
…
}
```

Retrofit – API interface

```
import retrofit2.Call
import retrofit2.Callback
import retrofit2.http.GET
import retrofit2.http.Query


interface CurrencyExchangeAPI {
    @GET("/latest")
    fun getRates(@Query("base") base: String): Call<MoneyResult>
}
```

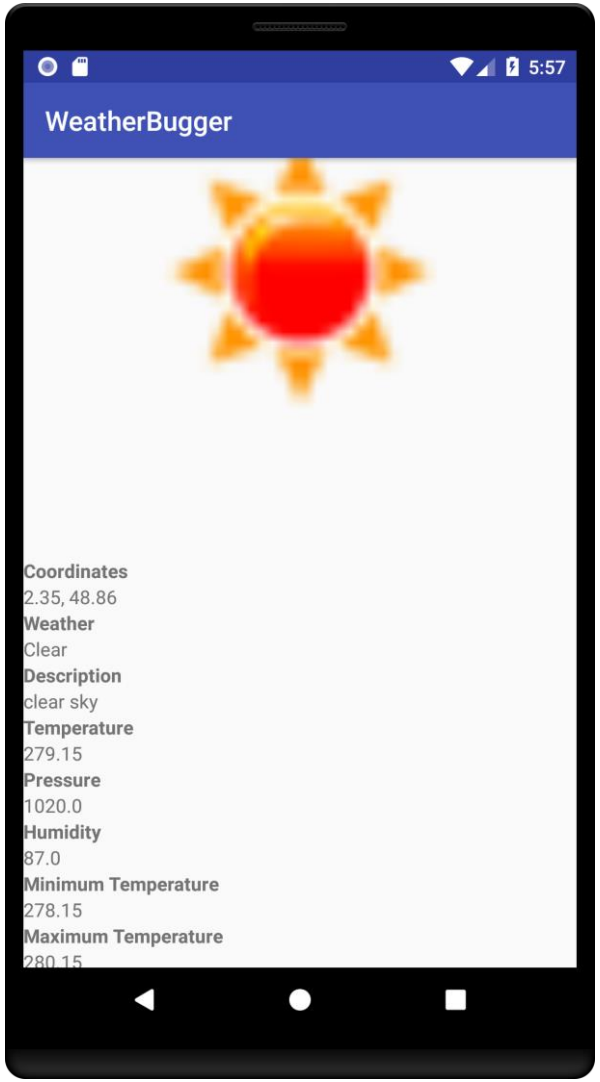
Retrofit - használat

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val retrofit = Retrofit.Builder()
        .baseUrl("https://api.exchangeratesapi.io/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val currencyAPI = retrofit.create(CurrencyExchangeAPI::class.java)

    btnGetRate.setOnClickListener {
        val ratesCall = currencyAPI.getRates("EUR")
        ratesCall.enqueue(object: Callback<MoneyResult> {
            override fun onFailure(call: Call<MoneyResult>, t: Throwable) {
                tvResult.text = t.message
            }

            override fun onResponse(call: Call<MoneyResult>,
              response: Response<MoneyResult>) {
                tvResult.text = response.body()?.rates?.HUF.toString()
            }
        })
    }
}
```

Házi feladat

- Időjárás alkalmazás
- *OpenWeatherMap:*
  - *http://api.openweathermap.org/data/2.5/weather?q=Budapest&units=metric&appid=f3d694bc3e1d44c1ed5a97bd1120e8fe*
  - ***Registration needed!***
- *Icon:*
  - *http://openweathermap.org/img/w/10d.png*
- *Icon pack:*
  - *http://openweathermap.org/weather-conditions*

Weather Report



OpenWeatherMap API elemek

- baseUrl:
  - https://api.openweathermap.org
- path:
  - data/2.5/weather
- Query paraméterek:
  - q[=Budapest]
  - units[=metric]
  - appid[=f3d694bc3e1d44c1ed5a97bd1120e8fe]

Retrofit - Weather

- Retrofit interfész – weather API

```
interface WeatherAPI {
    @GET("data/2.5/weather")
    fun getWeatherDetails(@Query("q") city: String,
                          @Query("units") units: String,
                          @Query("appid") appid: String): Call<Base>
}
```

- Retrofit objektum BaseURL megadással:

```
var retrofit = Retrofit.Builder()
        .baseUrl("https://api.openweathermap.org/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
var weatherAPI = retrofit.create(WeatherAPI::class.java)

val call = weatherAPI.getWeatherDetails(etCity.text.toString(),
    "metric",
    "[YOUR_KEY_HERE]")

call.enqueue(object : Callback<Base> {
    override fun onResponse(call: Call<Base>, response: Response<Base>) {
        ...
    }
    override fun onFailure(call: Call<Base>, t: Throwable) {
        ...
    }
})
```

Köszönöm a figyelmet! ☺

peter.ekler@aut.bme.hu