ANDROID ALKALMAZÁSFEJLESZTÉS

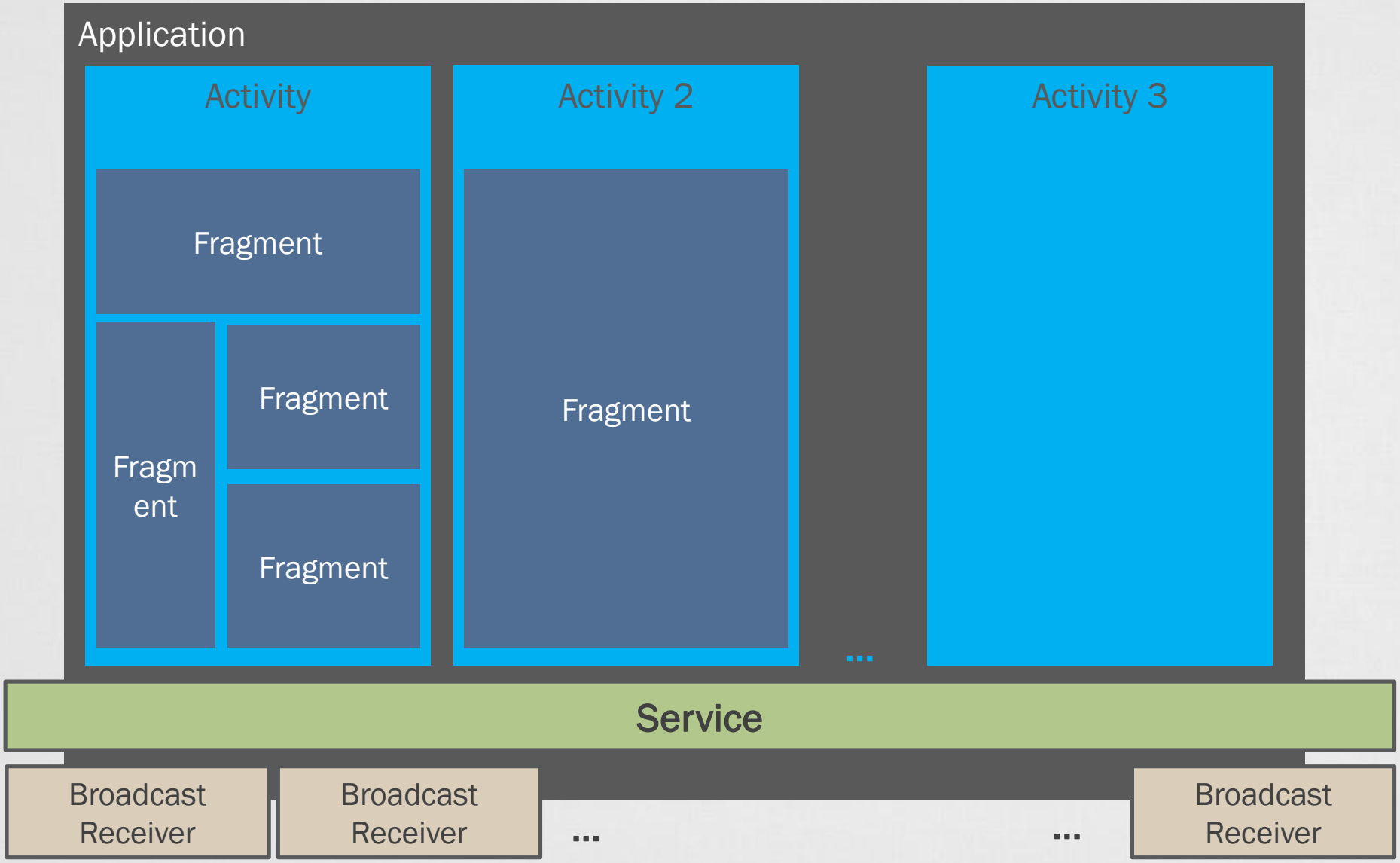
Activity
Fragment
Service
Broadcast Recevier
Architect Components
Runtime permissions

Sicz-Mesziár János
sicz-mesziar.janos@nik.uni-obuda.hu

Mezei József
mezei.jozsef@nik.uni-obuda.hu

2018. szeptember 29.

Overview


Application
Activity
Fragment
Fragm ent
Fragment
Fragment
Activity 2
Fragment
...
Activity 3
Service
Broadcast Receiver
Broadcast Receiver
...
...
Broadcast Receiver

AndroidManifest: Application

Az alábbi főbb komponenseket foglalja magába:

- Activity
- Service
- Broadcast Receiver

Fontosabb paraméterek:

- Icon – az alkalmazás ikonja, ez jelenik meg a Launcher-en is
- Label – az alkalmazás neve
- HardwareAccelerated – hardveres gyorsítás engedélyezése
- Theme – az alkalmazásra kiterjedő stílus
- Large heap – alkalmazás által használható maximális memóriaméret ideiglenes megemelése ha szükséges
- további paraméterek…

AndroidManifest: Activity

- Leírja, hogy hogyan működjön egy felhasználói felület, azaz ez tölti be a kontroller szerepét.
- Saját életciklusa van.
- Fontosabb működést befolyásoló paraméterek:
  - icon – az adott activity ikonja, például a „recent apps”-ban ez látszik
  - label – adott activity megnevezése
  - hardwareAccelerated – hardveres gyorsítás engedélyezése activity-n belül
  - screenOrientation – portrai, landscape, reverseLandscape, sensor, …
  - configChanges – fejlesztő kezel bizonyos változásokat: orientation, screensize
  - launchMode – activity példány működés
  - windowSoftInputMode – billentyűzet beállítások
  - theme – Activity-re alkalmazott stílus fájl, például noWindowTitle
  - exported – Activity elindítható third-party applikáció által is

Új Activity indítása

Új Activity hozzáadása:

`[package] > jobb klikk > New > Activity`

Új Activity hozzáadása kézzel:

- Layout XML → `res/layout/uj_activity.xml`
- Activity leszármaztatása → `src/[package]/UjActivity.kt`
- AndroidManifest.xml-ben felvenni

Új Activity indítása

```
var masikActivity = Intent(this, Masik::class.java)
startActivity(masikActivity)
```

Paraméterek átadása

Primitívek átadása Activity-nek

```
var masikActivity = Intent(this, Masik::class.java)
masikActivity.putExtra("szam", 100)
masikActivity.putExtra("key", "érték")
startActivity(masikActivity)
```

Primitívek

integer, double, float, string, boolean, byte, short, char

Átadott paraméterek elérése

```
protected fun onCreate(savedInstanceState: Bundle){
  super.onCreate(savedInstanceState);
  var szam = intent.extras.getInt("szam", 0)
  var kulcs = intent.extras.getString("key", null)
}
```

Paraméterek átadása (2) – Parcelable

```
class MyModel(var name: String, var value: Int) : Parcelable {

    constructor(parcel: Parcel) : this(
            parcel.readString(),
            parcel.readInt()
    )
    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(name)
        parcel.writeInt(value)
    }

    companion object CREATOR : Parcelable.Creator<MyModel> {
        override fun createFromParcel(parcel: Parcel): MyModel {
            return MyModel(parcel)
        }

    }

}
```

Összetett objektum átadása paraméterként.

Paraméterek átadása (2) – Visszatérés

Activity indítása visszatérési értékre várva

```
Context.startActivityForResult(i: Intent, requestCode: Int)
```

Visszatérési érték meghatározása

```
setResult(RESULT_OK, Intent())
```

Visszatérési érték levétele

```
class MainActivity : Activity {
      protected fun onActivityResult(requestCode: Int,
                                     resultCode: Int,
                                     data: Intent) {
        data.getIntExtra("kulcs", -1)
        super.onActivityResult(requestCode, resultCode, data)
    }
}
```

Service

Egy alkalmazás komponens:

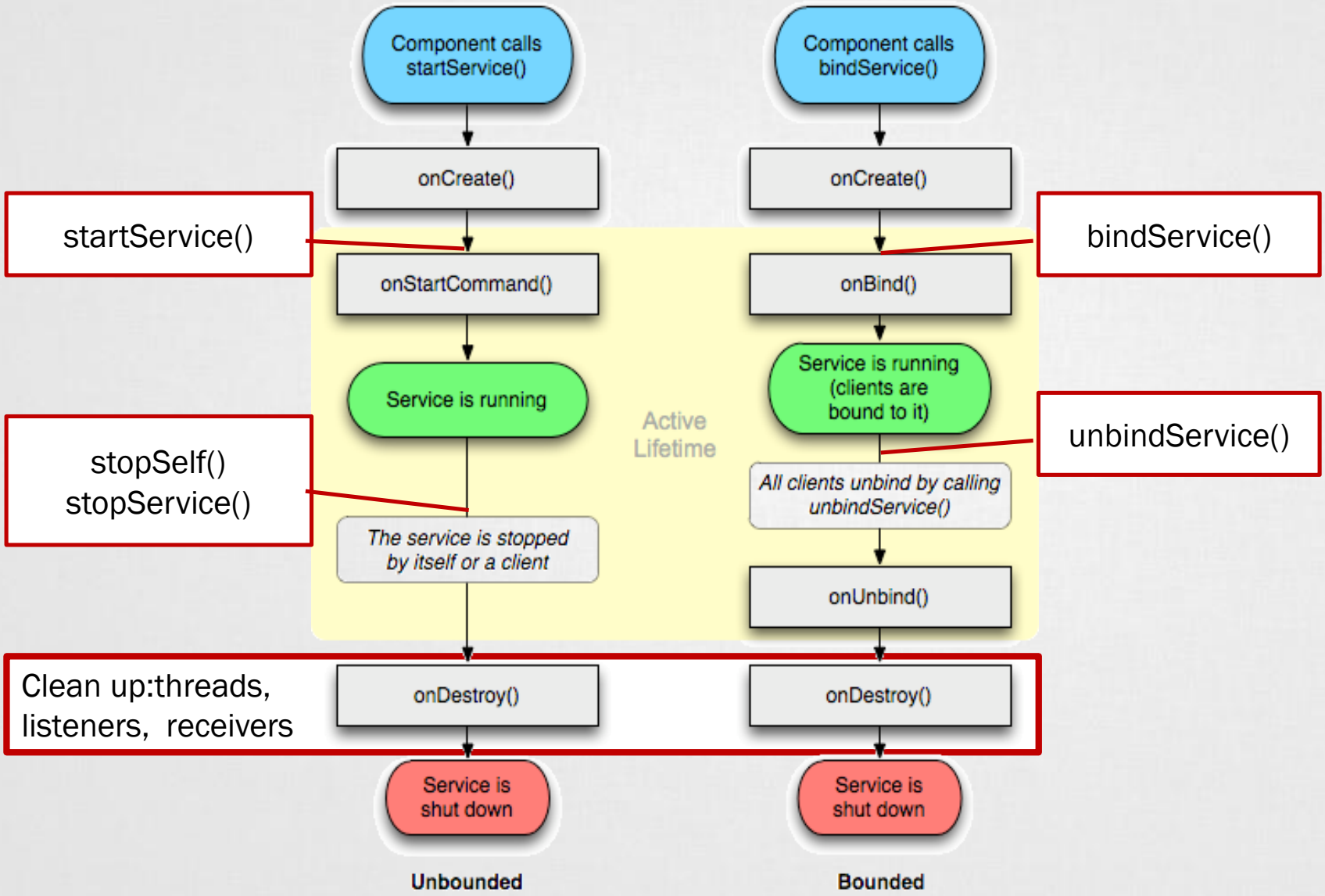
- hosszan futó műveletek végrehajtása
- háttérben fut, nincs UI

Nem külön szálon fut!
Nem egy külön folyamat!

Két formája van:

- Started
  - egy komponens (pl.: Activity) elindítja a startService() eljárással
  - Végtelenségig fut, akkor is ha az őt elindító komponens megsemmisül. (Nincs visszatérési érték) → **stopService()**
  - Általában egy műveletet hajt végre, majd megsemmisíti magát. Pl.: letöltés / feltöltés az internet irányába
- Bound
  - Egy komponens bindService()-al kötődik
  - Kliens-szerver felület a komponens és service között (request, results, …)
  - Addig fut amíg az összeköttetés él. (Pl.: zenelejátszás)

Service (2)


Component calls startService()
onCreate()
startService()
onStartCommand()
Service is running
stopSelf()
stopService()
The service is stopped by itself or a client
Clean up:threads, listeners, receivers
onDestroy()
Service is shut down
Unbounded
Active Lifetime
Component calls bindService()
onCreate()
bindService()
onBind()
Service is running (clients are bound to it)
unbindService()
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service is shut down
Bounded

Broadcast Receiver

Aszinkron, üzenetszórásos értesítés, azok akik felregisztráltak rá értesülnek róla

Két fő csoport:

- Normal broadcast
  - teljesen aszinkron,
  - „véletlen” sorrend
  - Hatékonyabb, de korlátok: nincs visszatérési érték, visszavonás
- Ordered broadcast
  - egyszerre csak egy „receiver” fut
  - visszavonható – abort
  - Prioritásokat adhatunk (android:priority)

onReceive() csak egyszer fut le - rövid életű

- Hosszan futó műveletekre alkalmatlan (timeout ~10sec)

Broadcast Receiver a gyakorlatban

Regisztrálás statikusan

- BroadcastReceiver osztály implementálása
- AndroidManifest.xml-ben <receiver> megadása

Regisztrálás dinamikusan, futási időben

- BroadcastReceiver osztály implementálása
- Context.registerReceiver()-el regisztálunk → onResume()
- Context.unregisterReceiver() leiratkozunk → onPause()

```
val receiver = object : BroadcastReceiver() {
  override fun onReceive(context: Context?, intent: Intent?) {
      val level = intent?.getIntExtra("level", 0)
      textview.text = "Level: ${level}%"
  }
}
```

Broadcast kibocsátás

- Context.sendBroadcast(Intent intent)

Architecture Components

2017 őszétől: Android Oreo újítása, visszafele kompatibilis verzió függetlenül

LifeCycle: "Életciklus-biztos" komponens – memory leaks és alkalmazás összeomlások elkerülése céljából

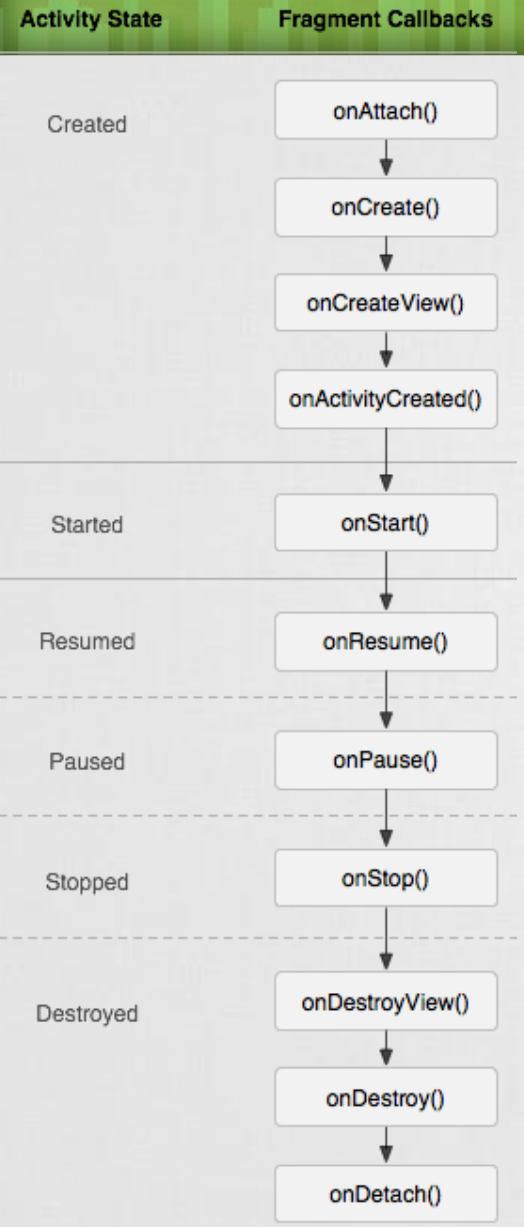
LiveData: "Életciklus-biztos" komponens – figyeli egy adat megváltozását, ha változik az adat a UI komponens is frissül

ViewModel: Felhasználói felülettel kapcsolatos adattárolás és menedzselés, konfiguráció változáskor adatörzés, pl rotation

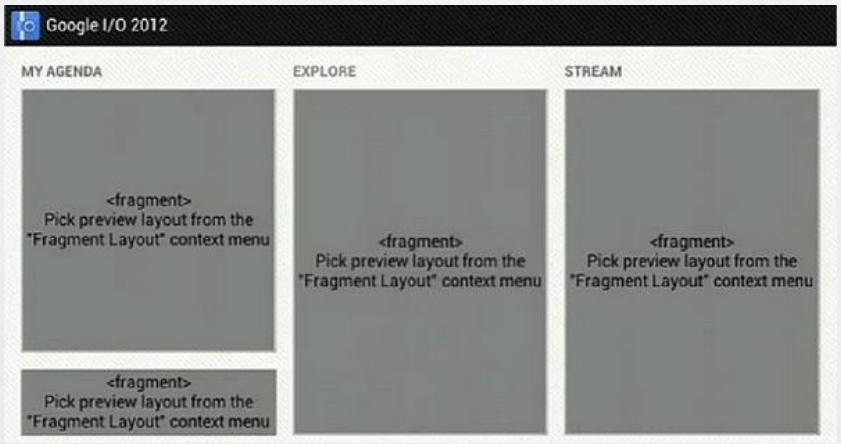
Room: absztrakciós réteg az SQLite felett, “boilerplate code” elkerülése, biztonságos adathozzáférés

Paging Library: adatok szakaszos betöltése, memóriát akkor pakoljuk tele ha igény is van az adatra, adatok lapozott betöltése, például RecycleView scrollozásakor

Fragments



- Activity moduláris része, saját életciklussal:
  - Ha egy Activity *„pause”* állapotba kerül akkor az összes benne lévő Fragment is!
  - Függetlenül manipulálható: futás időben hozzáadható, elvehető, „back stack”-be tehető.
  - Layout-on bárhol elhelyezhető.



Lásd
Google IO 2012
App demo

- Android 3.0 (API level 11) óta érhető el
- Android 1.6-tól is használható Support Library segítségével.

Fragment a gyakorlatban

```
class MyFragment : Fragment() {
    companion object {
        fun newInstance(): MyFragment {
            return MyFragment()
        }
    }
    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_my, container, false)
    }
    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        // TODO rest of code
    }
}
```

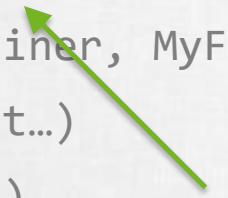
Fragment Design Pattern:
gyártó metódus – például ha paramétert adnánk át itt feltölthetnénk a setArguments() metódussal

Fragment Design Pattern:
a fragment controller szerepet tölt be, itt kell megadni, hogy milyen View kötődik hozzá

Fragment Design Pattern:
az Activity onCreate() metódusával egyezik meg

Fragment betöltése, futás időben

```
private fun loadFragment() {
    val fm = getSupportFragmentManager()
    val ft = fm.beginTransaction()
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
    ft.addToBackStack(MyFragment::class.java.name)
    ft.replace(R.id.container, MyFragment.newInstance())
    // ft.add(R.id.container, MyFragment.newInstance())
    // ft.remove(Fragment…)
    // ft.hide(Fragment…)
    ft.commit()
}
```



Azon container ID-val bíró ViewGroup, ahova a Fragment az onCreateView() metódusban visszaadott View-t betölti.

Backstack

- Hasonló az alkalmazás stack-hez
- Fragment váltások előzményeit tartalmazza
- Vissza gomb (back) esetén automatikusan betölti az előző fragmentet
- Ennek állapota lekérdezhető:
  - `getBackStackEntryAt(index: Int)`
  - `getBackStackEntryCount()`
  - `popBackStack()`
  - `addOnBackStackChangedListener(listener: OnBackStackChangedListener)`

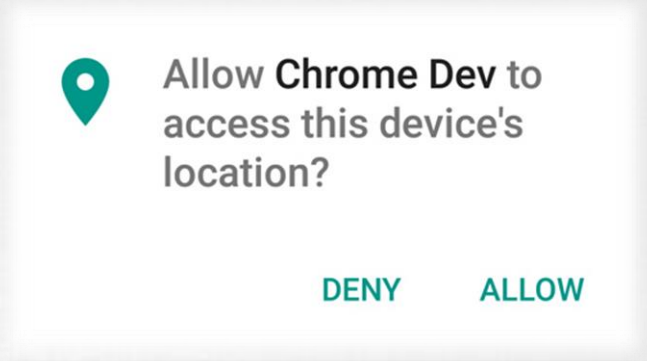
Runtime Permissions

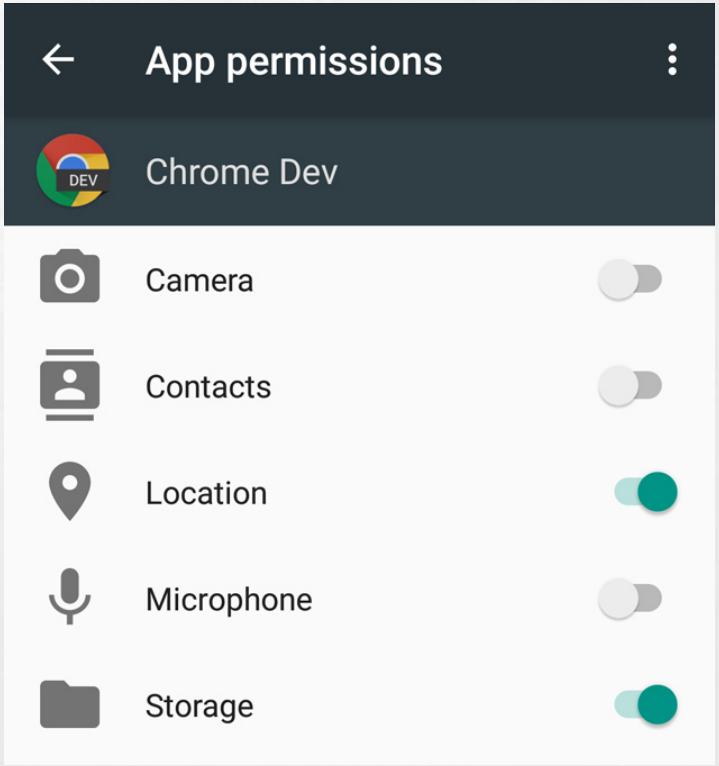
6.0+

Runtime permissions

Mi változott?

- Az Android 6.0-tól kezdve futás időben dönt a felhasználó a hozzáférésről
- Telepítésnél már nem kell beleegyeznie a jogosultság kérésekbe
- Visszavonható
- targetApiLevel: 23+
- Két kategória:
  - Normal
  - Dangerous
- Support library segít a kezelésben





Használata

```
// Hozzáférés meglétének lekérdezése
val permissionCheck = ActivityCompat.checkSelfPermission(
        requireContext(),
        Manifest.permission.WRITE_CALENDAR
)

// Hozzáférés ellenőrzése és igénylése
if (permissionCheck != PackageManager.PERMISSION_GRANTED) {

    // Opcionális: igényel-e magyarázatot --> dangerous
    if (ActivityCompat.shouldShowRequestPermissionRationale(
                    requireActivity(),
                    Manifest.permission.WRITE_CALENDAR
            )) {
        // Igényelt hozzáférés elmagyarázása a felhasználónak.
        // Aszinkron, válasz esetén újra kell futtatni ezt a kódot!
    } else {
        // Kérés benyújtása
        ActivityCompat.requestPermissions(
                requireActivity(),
                arrayOf(Manifest.permission.WRITE_CALENDAR),
                1027 // valamilyen szám
        )
    }

}
```