Android alkalmazás komponensek

Ekler Péter

BME VIK AUT, AutSoft

peter.ekler@aut.bme.hu

Tematika

1. Android platform bemutatása, Kotlin alapok
2. Alkalmazás komponensek, Kotlin konvenciók
3. Felhasználói felület
4. Fragmentek, haladó UI
5. Listák kezelése hatékonyan
6. Perzisztens adattárolás, adatbázisok, haladó Kotlin
7. Hálózati kommunikáció
8. Felhő szolgáltatások
9. Helymeghatározás, térkép kezelés
10. Architektúra komponensek, JetPack

Gyors Kotlin gyakorlat

- Autókölcsönző modell
- CarRental osztály
  - Autók tárolása listában
  - Autók értékének lekérdezhetősége
- Car osztály (típus, érték)
- ElectricCar leszármazott osztály (hatótáv)

Autókölcsönző modell

```
class CarRental(val name: String) {

    private val cars = mutableListOf<Car>()

    fun addCar(car: Car) {
        cars += car
    }

    fun getTotalPrice(): Int {
        return cars.sumBy { it.price }
    }
}

open class Car(val type: String, val price: Int)

class ElectricCar(type: String, price: Int, val range: Int) : Car(type, price)



val carRental = CarRental("EAutoBt")

carRental.addCar(ElectricCar("Tesla",1000, 300))
carRental.addCar(ElectricCar("Ampera",400, 80))
carRental.addCar(ElectricCar("Leaf",380, 140))
```

Tartalom

- Pong alklamazás befejezése
- Alkalmazás komponensek
- Activity életciklus
- Back Stack
- Több Activity kezelése
- Barkóba alkalmazás

Gyakoroljunk!

- Készítsünk egy Pong alkalmazást!
- Érintett témák
  - Egyedi nézetek
  - Rajzolás
  - Szálkezelés
  - Képek kezelése

Clean Code

Forrás: https://cleancoders.com/

Mit jelent a Clean Code? Miért van rá szükség?

- Mi a software igazi értéke?
  - Karbantarthatóság
  - Folyamatos szállítás biztosítása

- Napjainkban:
  - Folyamatosan változó követelmények
- Agilis fejlesztés
- Csapatmunka
- Kódminőség
- Software életciklus

Alapvető Clean Code elvek

- Elnevezés
  - Kis scopeon belül: hosszú, beszédes nevek
  - Nagy scopeban: rövid nevek
- Rövid osztályok
- Egy metódus csak egy dolgot csinál
- Rövid metódusok
  - Maximum ~4 sor!
- Kevés argumentum
  - Maximum 3
- Nincs boolean argumentum
- Nincs output argumentum

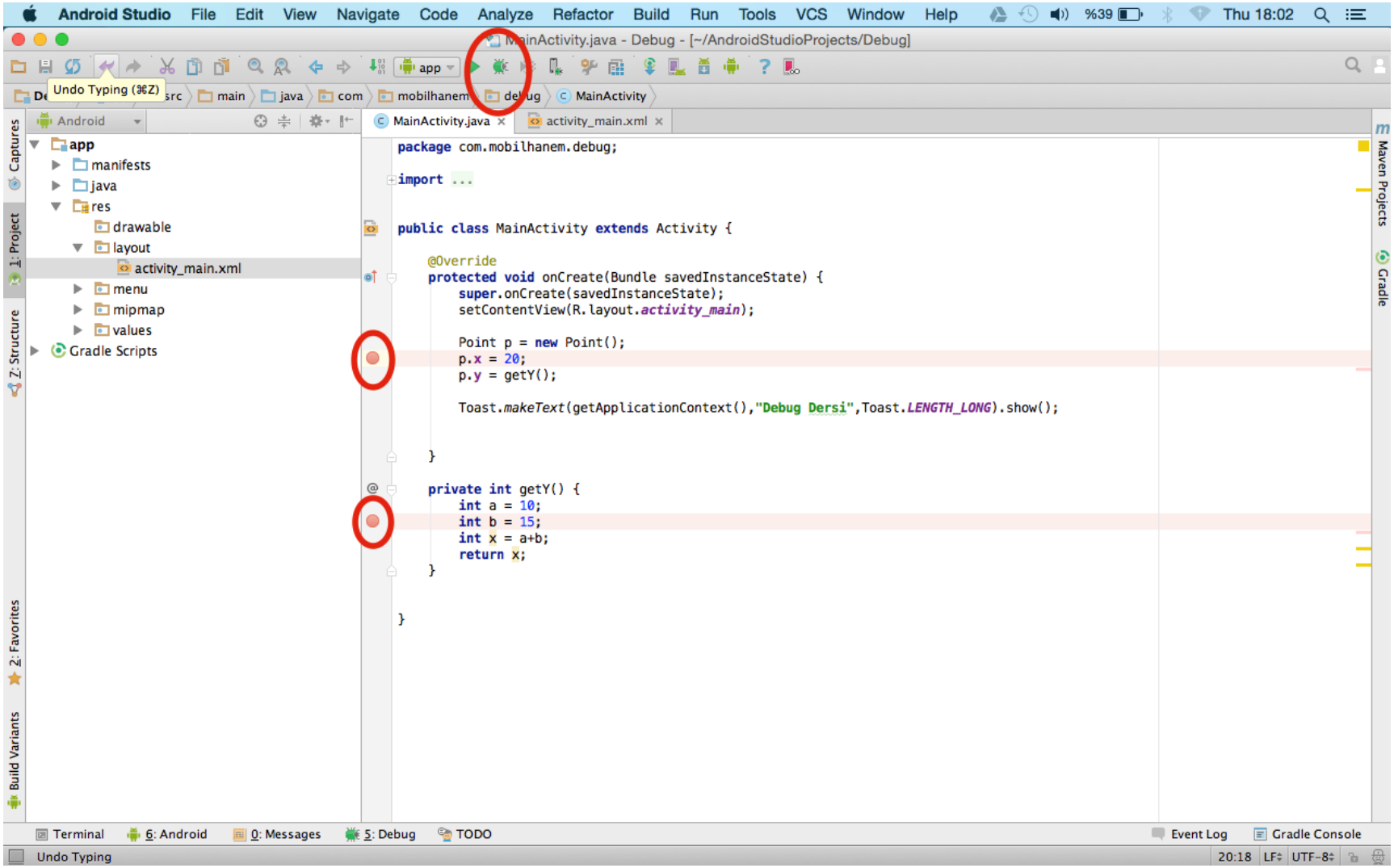
Android LogCat

- Log nézet
- Monitorozható
- Többféle log szint:
  - v(String, String) (verbose)
  - d(String, String) (debug)
  - i(String, String) (information)
  - w(String, String) (warning)
  - e(String, String) (error)
- Log.i("TAG_LOCATION", " Position: " + position);
- File-ba is átirányítható
  - logcat -f <filename>
  - http://developer.android.com/tools/help/logcat.html

Debug lehetőségek

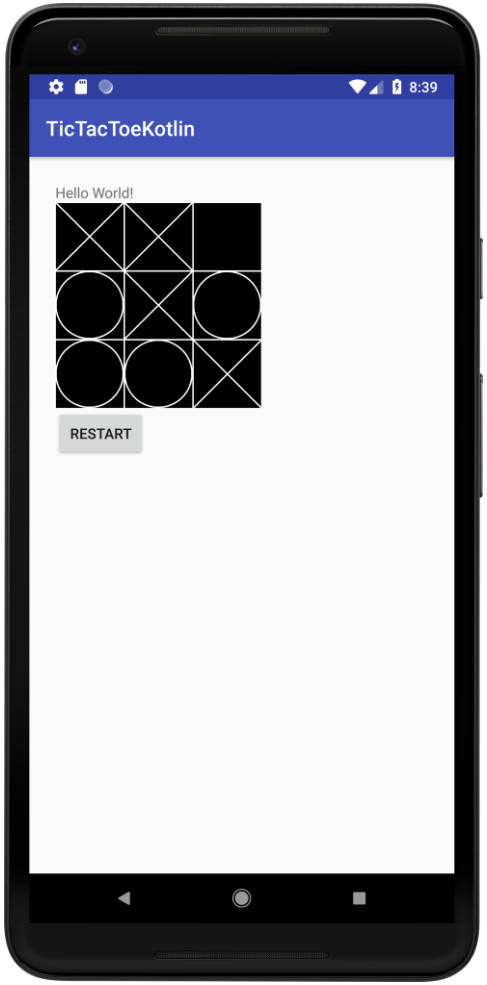
- SDK támogat többféle debug lehetőséget
  - Emulátor
  - On-device debug
- Debug folyamat:
  - Breakpointok elhelyzése
  - Alkalmazás indítása debug módban
  - Soronkénti végrehajtás

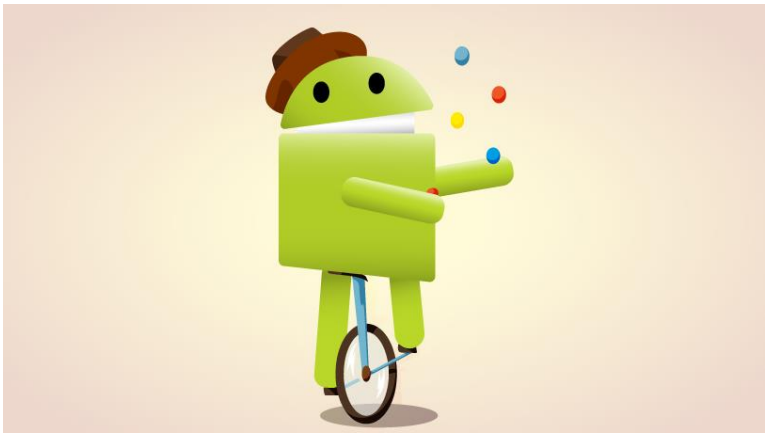
Debug indítása



Gyakorlás

- Készítsünk TicTacToe alkalmazást!
- Érintett témák
  - Singleton objektum
    - object TicTacToeModel {...}
  - Activity és egyedi nézet közti kapcsolat
  - Méret felüldefiniálása

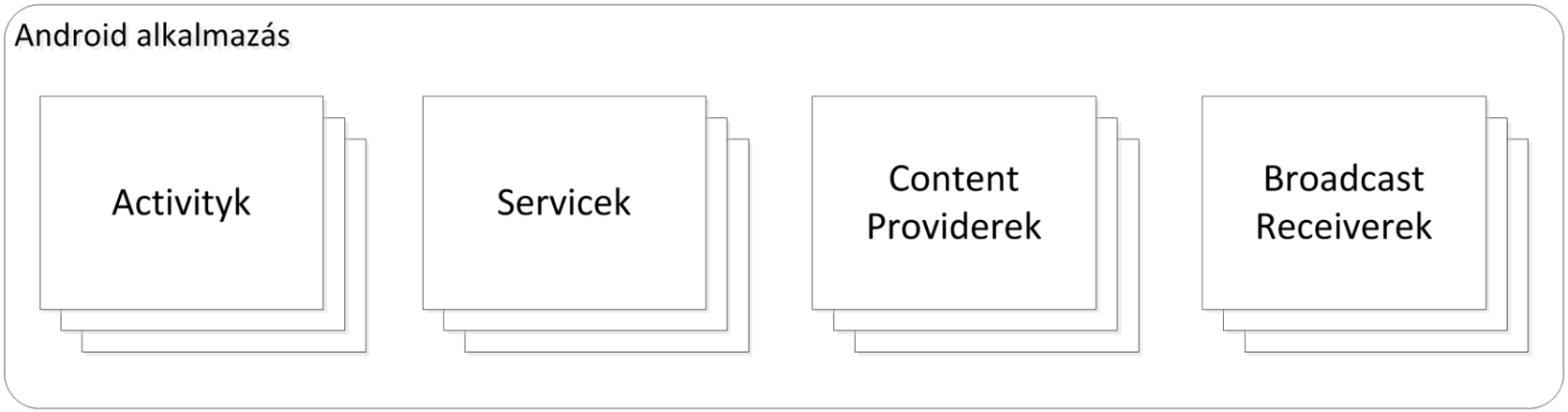




https://kahoot.it/

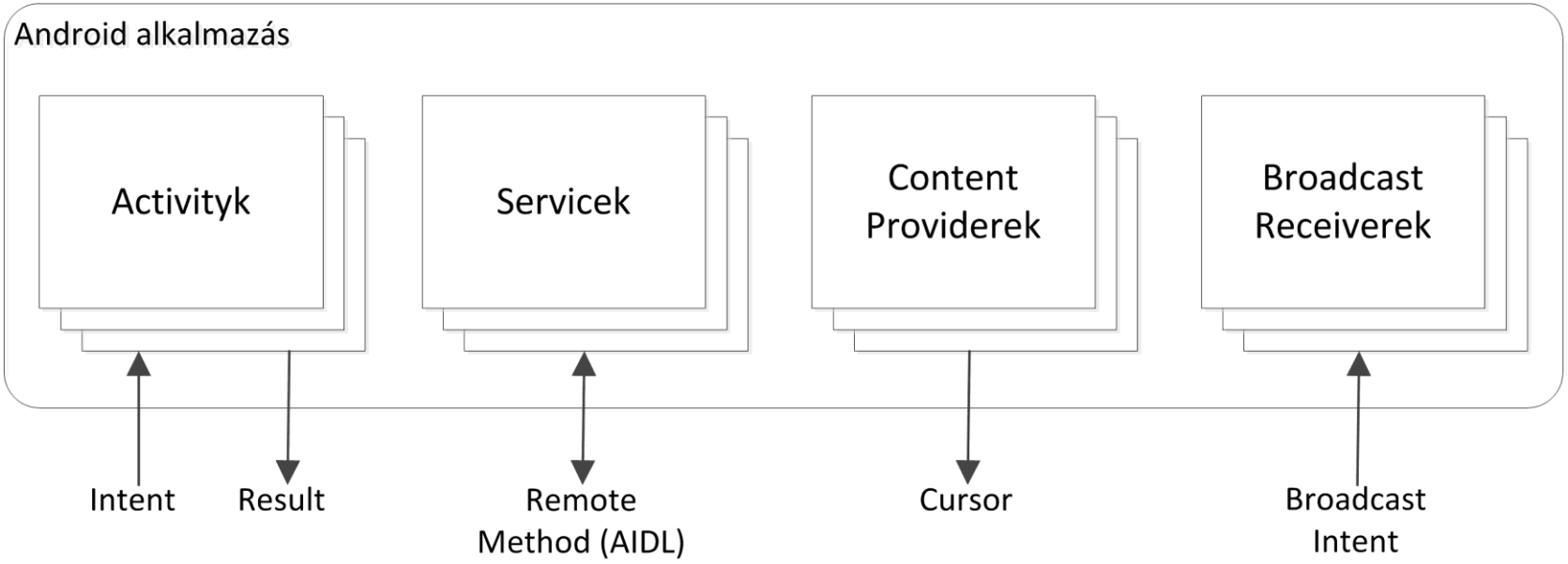
Android alkalmazás felépítése 1/2

- Egy Android alkalmazás egy vagy több alkalmazás komponensből épül fel:
  - Activity-k
  - Service-k
  - Content Provider-ek
  - Broadcast Receiver-ek



Android alkalmazás felépítése 2/2

- Minden komponensnek különböző szerepe van az alkalmazáson belül
- Bármelyik komponens önállóan aktiválódhat
- Akár egy másik alkalmazás is aktiválhatja az egyes komponenseket



Activity-k

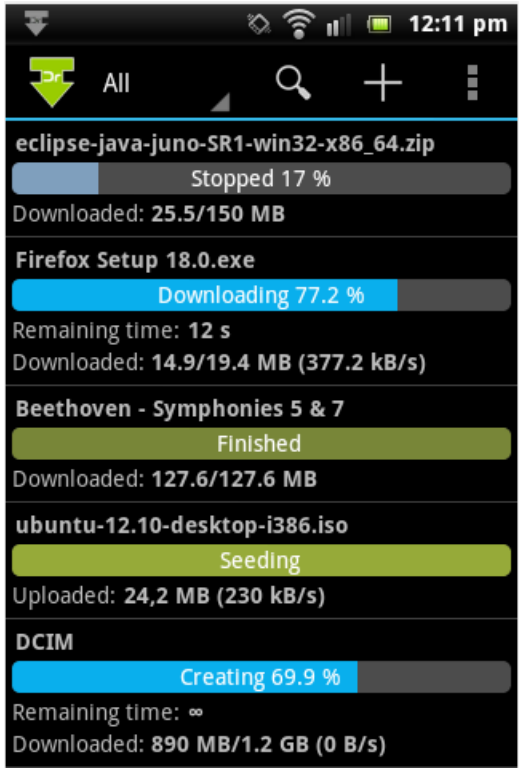
- Különálló nézet, saját UI-al
- Például:
  - Emlékeztető alkalmazás
  - 3 Activity: ToDo lista, új ToDo felvitele, ToDo részletek
- Független Activity-k, de együtt alkotják az alkalmazást
- Más alkalmazásból is indítható az Activity, például:
  - Kamera alkalmazás el tudja indítani az új ToDo felvitele Activity-t és a képet hozzá rendeli az emlékeztetőhöz
- Az `android.app.Activity` osztályból származik le

Service-k

- A Service komponens egy hosszabb ideig háttérben futó feladatot jelképez
- Nincs felhasználói felülete
- Például egy letöltő alkalmazás (torrent ☺) fut a háttérben, míg előtérben egy másik programmal játszunk
- Más komponens (pl. Activity) elindíthatja, vagy csatlakozhat (bind) hozzá vezérlés céljából
- Az `android.app.Service` osztályból kell öröklődnie

DrTorrent

- BitTorrent kliens Android platformra
- Megszokott funkciók és háttérben működés



Content provider-ek

- A Content provider (tartalom szolgáltató) komponens feladata egy megosztott adatforrás kezelése
- Az adat tárolódhat fájlrendszerben, SQLite adatbázisban, web-en, vagy egyéb perzisztens adattárban, amihez az alkalmazás hozzáfér
- A Content provider-en keresztül más alkalmazások hozzáférhetnek az adatokhoz, vagy akár módosíthatják is azokat
- Például: CallLog alkalmazás, ami egy Content provider-t biztosít, és így elérhető a tartalom
- A `android.content.ContentProvider` osztályból származik le és kötelezően felül kell definiálni a szükséges API hívásokat

Broadcast receiver-ek

- A Broadcast receiver komponens a rendszer szintű eseményekre (broadcast) reagál
- Például: kikapcsolt a képernyő, alacsony az akkumulátor töltöttsége, elkészült egy fotó, bejövő hívás, stb.
- Alkalmazás is indíthat saját „broadcast”-ot, például ha jelezni akarja, hogy valamilyen művelettel végzett (letöltődött a torrent ☺)
- Nem rendelkeznek saját felülettel, inkább valamilyen figyelmeztetést írnak ki például a status bar-ra, vagy elindítanak egy másik komponenst (jeleznek például egy service-nek)
- A `android.content.BroadcastReceiver` osztályból származik le; az esemény egy Intent (lásd. Később) formájában érhető el

Android alkalmazás struktúrája

- Az alkalmazás leíró (*Manifest*) állománynak deklarálnia kell a következőket:
  - Alkalmazás komponensek listája
  - Szükséges minimális Android verzió
  - Szükséges hardware konfiguráció
- A nem forráskód jellegű erőforrásoknak (képek, szövegek, nézetek, stb.) rendelkezésre kell állnia különböző nyelvű és képernyőméretű telefonokon

Activity bevezetés

- Egy Activity tehát tipikusan egy képernyő, amin a felhasználó valamilyen műveletet végezhet (login, beállítások, térkép nézet, stb.)
- Az Activity leginkább egy ablakként képzelhető el
- Az ablak vagy teljes képernyős, vagy pop-up jelleggel egy másik ablak fölött jelenik meg
- Egy alkalmazás tipikusan több Activity-ből áll, amik lazán csatoltak
- Legtöbb esetben létezik egy „fő” Activity, ahonnét a többi elérhető
- Bármelyik Activity indíthat újabbakat
- Tipikusan a „fő” Activity jelenik meg az alkalmazás indulása után elsőként

A „fő” Activity jelölése

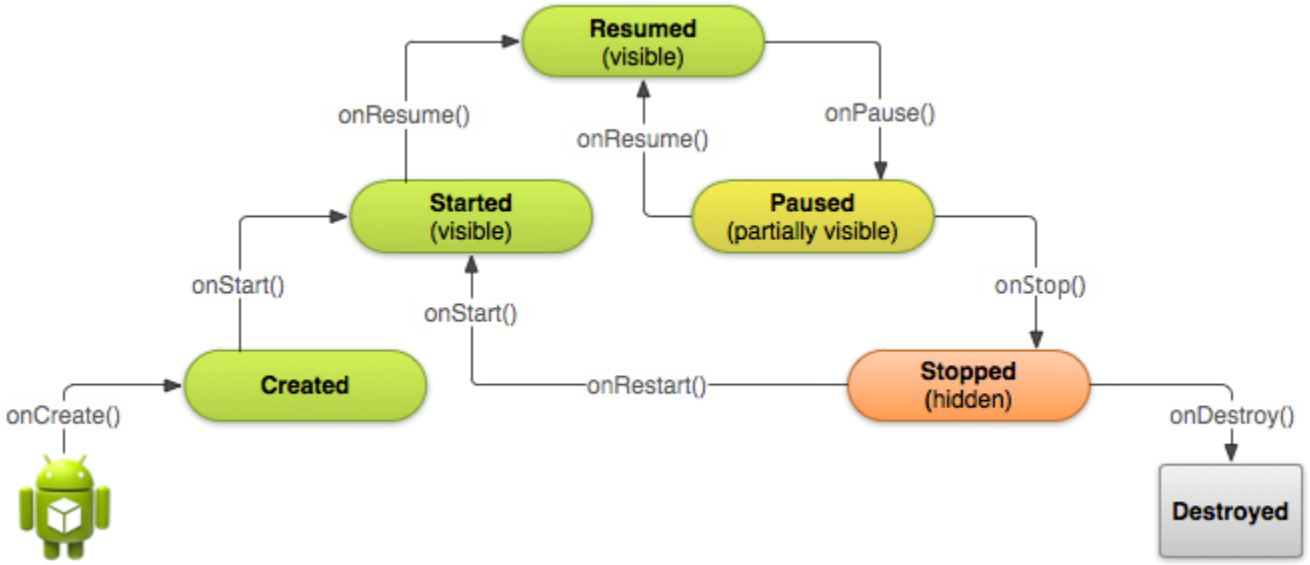
- Manifest állományban jelölnünk kell melyik Activity induljon el elsőként

```
<activity
    android:name=".ExampleActivity"
    android:icon="@drawable/app_icon" >
    <intent-filter>
        <action android:name=" android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- *action* tag: jelzi, hogy ez az alkalmazás fő belépési pontja
- c*ategory* tag: jelzi, hogy az Activity jelenjen meg az indítható programok listájában
- Intentekről (szándék) később…

Activity indítás

- Nincs *main()* metódus
- Az Android rendszer hívja a megfelelő Activity életciklus függvényeket
- Az Activity élete során az állapotától függő életciklus függvényeket hívja meg a rendszer



Activity állapotok

- Egy Activity 3 fő állapotban lehet:
  - **Resumed (running)**: az Activity előtérben van és a focus rá irányul
  - **Paused**: az Activity él, de egy másik Activity előrébb van, de ez még látszik (transparens a felső, vagy pop-up jellege miatt nem fedi el teljesen). A rendszer extrém alacsony memóriaállapot esetén felszabadíthatja.
  - **Stopped**: az Activity még él, de már egy másik Activity van teljesen előtérben és a Stopped állapotban lévőből semmi nem látszik. Alacsony memóriaállapot esetén a rendszer felszabadíthatja.

Activity skeleton 1/2

```
class ExampleActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Most jön létre az Activity
    }

    override fun onStart() {
        super.onStart()
        // Most válik láthatóvá az Activity
    }

    override fun onResume() {
        super.onResume()
        // Láthatóvá vált az Activity
    }
```

Activity skeleton 2/2

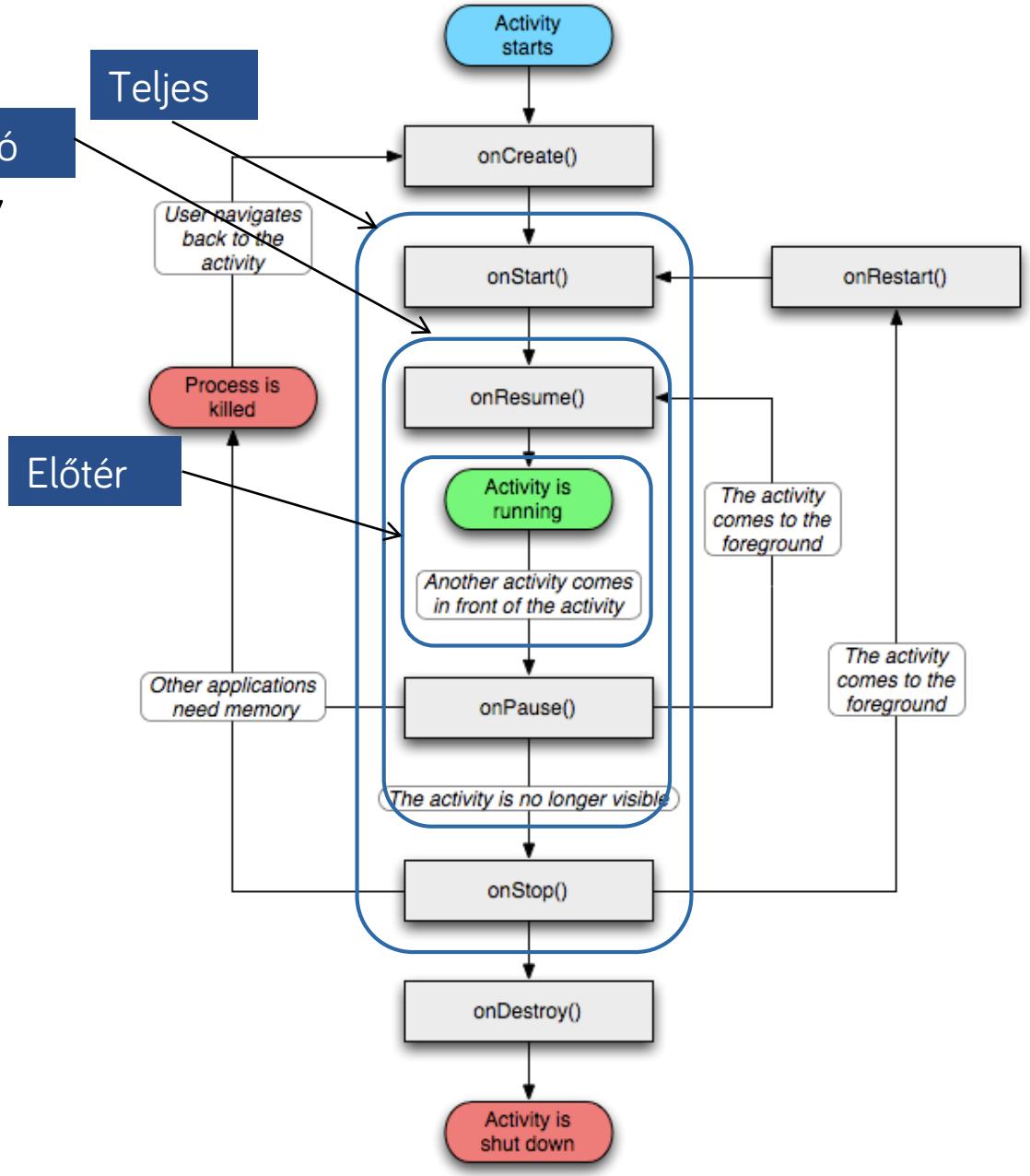
```
    override fun onPause() {
        super.onPause()
        // Másik Activity veszi át a focus-t
        // (ez az Activity most kerül „Paused” állapotba)
    }

    override fun onStop() {
        super.onStop()
        // Az Activity már nem látható
        // (most már „Stopped” állapotban van)
    }

    override fun onDestroy() {
        super.onDestroy()
        // Az Activity meg fog semmisülni
    }

}
```

Activity állapot típusok

- Teljes (**entire**) életciklus:
  - *onCreate()* és *onDestroy()* közti állapot
- Látható (**visible**) életciklus:
  - *onStart()* és *onStop()* közti állapot
- Előtér (**foreground**) életciklus:
  - *onResume()* és *onPause()* közti állapot

Activity életciklus

- A megfelelő életciklus függvény hívódik meg állapotváltáskor
- Az életciklus függvények felüldefiniálhatók
  - Az ősosztály függvényét kötelező meghívni (pl.: *super.onCreate()*)
- Fejlesztők felelőssége!



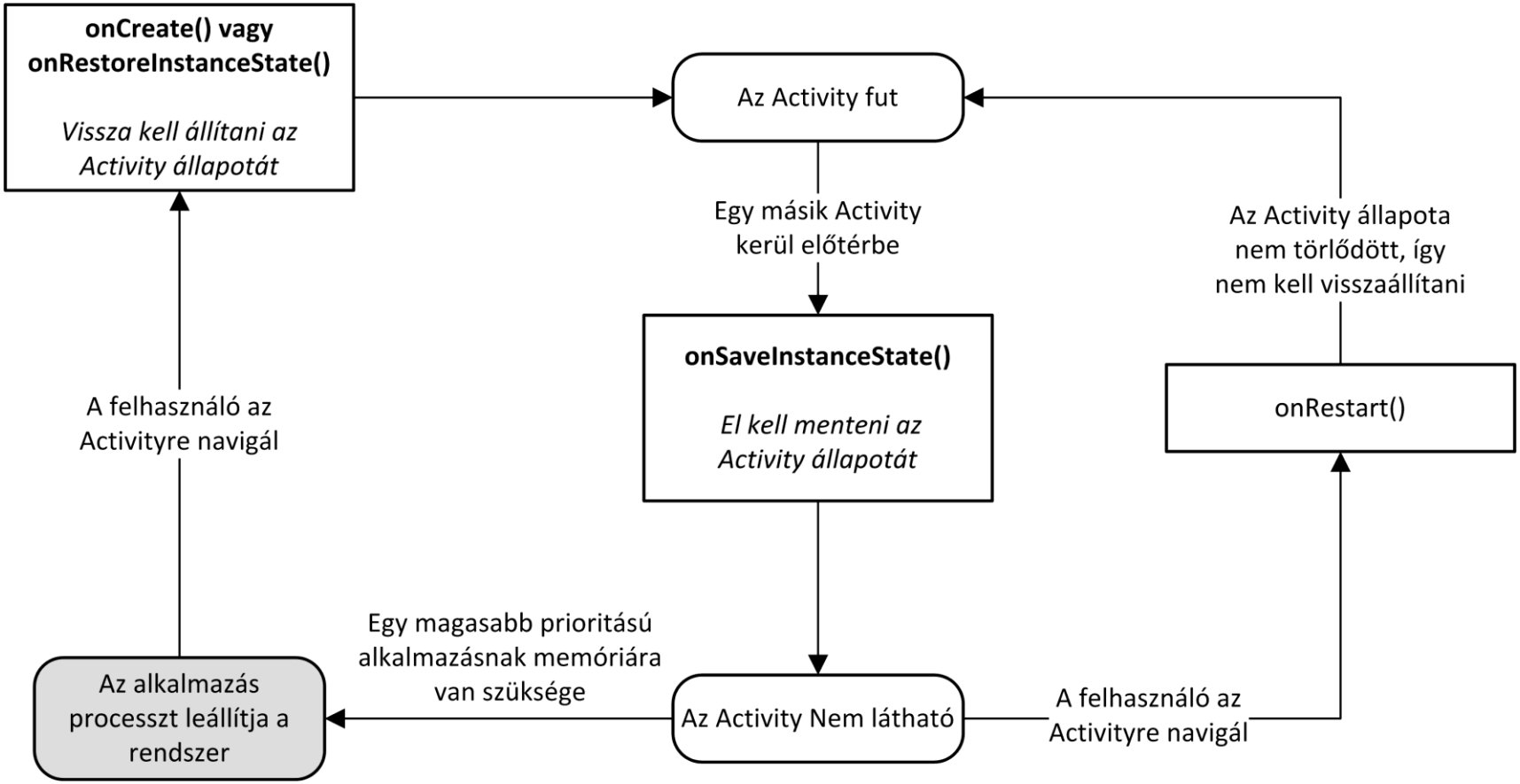
Launcher Activity

- Jelölése az AndroidManifest.xml-ben:

```
<activity android:name=".MainActivity"
  android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
          "android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Otherwise the application will not be visible

Activity állapotának elmentése 1/2



Activity állapotának elmentése 2/2

- Az *onSaveInstanceState()* tipikusan az *onPause()* és *onStop()* előtt hívódik meg
- Nincs rá garancia, hogy mindig meghívódik, például ha a felhasználó a „Vissza” gombbal lép ki (jelzi, hogy végzett ezzel az Activity-vel, nincs mit elmenteni)
- Belső változók és UI elemek értékét szokás ilyenkor elmenteni
- Semmiképp se használjuk perzisztens adatok mentésére!
- Az ős (super) implementációját mindig hívjuk meg
- A rendszer alapértelmezetten is menti az Activity és a rajta lévő UI elemek állapotát bizonyos szinten (lásd UI előadás)
- Tesztelés: képernyő elforgatásával

Konfiguráció változások kezelése az Activity-ben

- A készülék fontos paramétere néha változhat futás közben (képernyő orientáció, külső billentyűzet, nyelv, stb.)
- Ezen változások esetén a rendszer újraindítja az Activity-t (*onDestroy()* és egyből *onCreate()* hívás)
- Ok: a rendszer új erőforrásokat tölthet be az új konfigurációhoz (pl. háttér más lesz, ha változik az orientáció)
- Ilyenkor az állapot elmentésére az *onSaveInstanceState()* a legkézenfekvőbb
- Visszatöltéshez használható még az *onRestoreInstanceState()*, de az *onCreate()*-ben a jellemzőbb

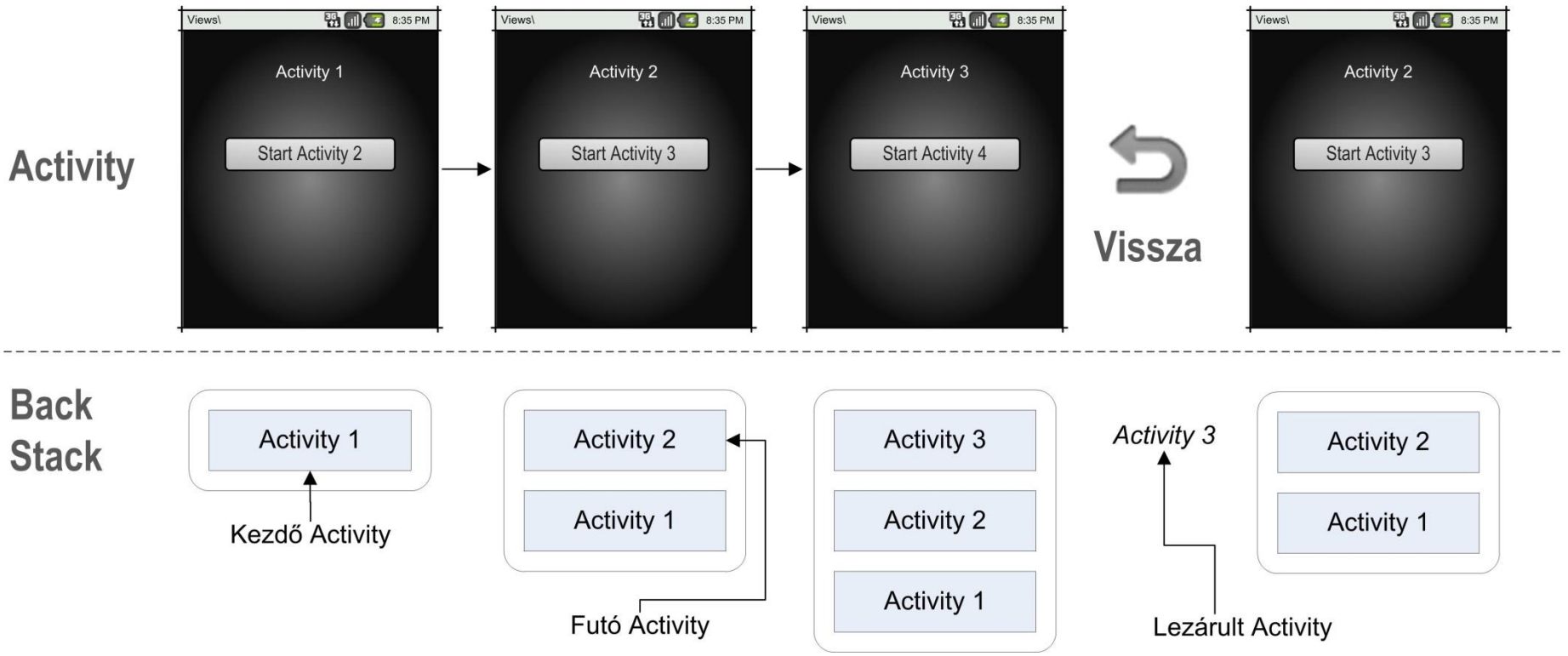
Activity váltás

- Életciklus callback függvények meghívási sorrendje:
  - A Activity *onPause()* függvénye
  - B Activity *onCreate()*, *onStart()* és *onResume()* függvénye (B Activity-n van már a focus)
  - A Activity *onStop()* függvénye, mivel már nem látható
- Ha a B Activity valamit adatbázisból olvas ki, amit az A ment el, akkor ez a mentés A-nak az *onPause()* függvényében kell megtörténjen, hogy a B aktuális legyen, mire a felhasználó előtt megjelenik

Activity Back Stack 1/2

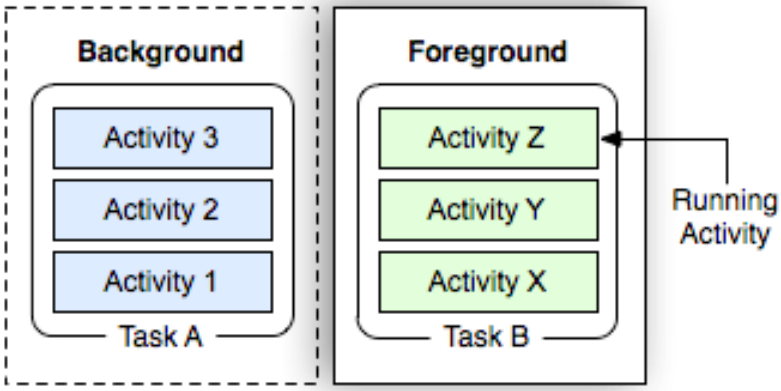
- Egy feladat végrehajtásához a felhasználó tipikusan több Activity-t használ
- A rendszer az Activity-ket egy ún. Back Stack-en tárolja
- Az előtérben levő Activity van a Back Stack tetején
- Ha a felhasználó átvált egy másik Activity-re, akkor eggyel lejjebb kerül a Stack-ben és a következő lesz legfelül
- Vissza gomb esetén legfelülről veszi ki a rendszer az megjelenítendő Activity-t
- Last in, first out

Activity Back Stack 2/2



Multitasking

- Task: Egy elvégzendő feladat, ami több Activity-t használ, de nem mindegyik Activity kell hogy ugyanabban az alkalmazásban szerepeljen
- A HOME gomb lenyomásával a rendszer visszatér a kezdő képernyőre és új taskot indíthatunk
- Új task indításakor a rendszer megőrzi az előző task Back Stack-jét, de memória gondok esetén bezárhat Activity-ket
- Az új task egy új Back Stack-et kap



Memória kezelés, processzek

- Activity külön processzben is futtatható (több memória)

```
<activity
        android:name="hu.bme.aut.JobActivity"
        android:process=":extrajob"
        android:label="@string/app_name"/>
```

Új Activity indítása

- SecondActivity indítása:

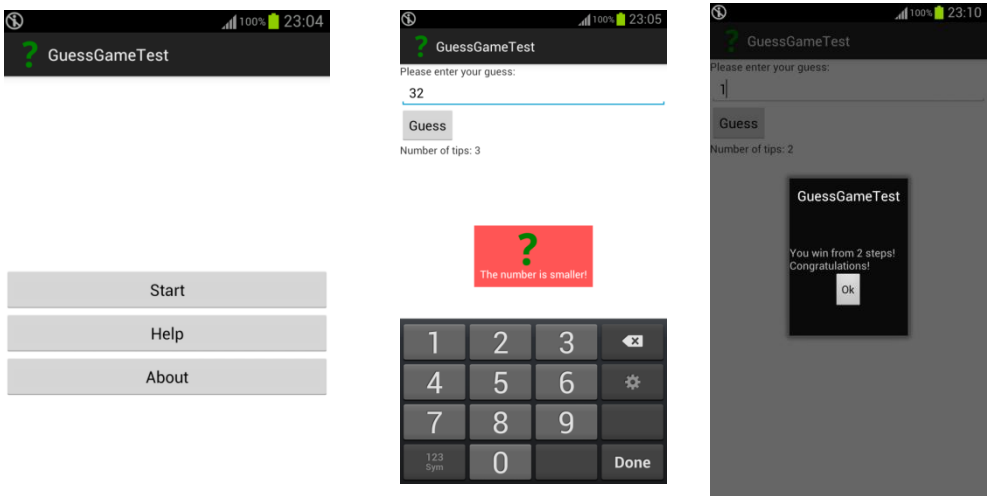
```
fun runSecondActivity() {
    val myIntent: Intent = Intent()
    myIntent.setClass(this@MainActivity,
                      SecondActivity::class.java)
    // Adat átadása
    myIntent.putExtra("KEY_DATA", "Hi there!")
    startActivity(myIntent)
}
```

Activity vezérlés

- Manifest property-k
  - taskAffinity: task leírás
  - launchMode: indulási mód
  - allowTaskReparenting: új taskot hozzon-e létre
  - clearTaskOnLaunch: többi Activity-t törli a taskról
  - alwaysRetainTaskState: a rendszer kezelje a task állapotát
  - finishOnTaskLaunch: állítsa le az Activity ha a felhasználó kilépett
  - screenOrientation: fix landscape/portrait
- *startActivity(...)* intent Tag-ek
  - FLAG_ACTIVITY_NEW_TASK
  - FLAG_ACTIVITY_CLEAR_TOP
  - FLAG_ACTIVITY_SINGLE_TOP

Gyakoroljunk!

- Késztsünk egy barkóba alkalmazást
- Képernyő forgatás kezelése
- *TextInputLayout* használata
- Eredmény dialógus Activity:
  android:theme="@style/Theme.AppCompat.Dialog"



BackStack kezelése

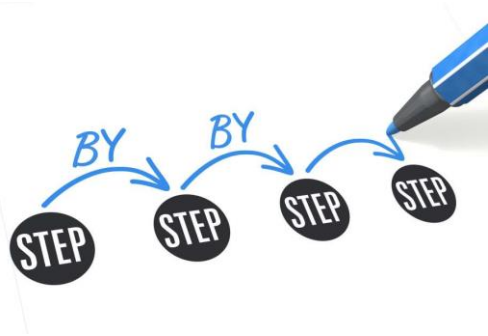
```
val showMain = Intent(this, MainActivity::class.java)
showMain.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
startActivity(showMain)
finish()
```

Kotlin nyelv bevezetése 1/2

- Előnyök bemutatása
  - > Produktivitás növelése
  - > Kevesebb hibalehetőség
  - > Motivált fejlesztők
  - > Vonzó pozíciók
- Tisztázni az alacsony kockázatot
  - > Java tudás teljes mértékben felhasználható
  - > Egyszerű és könnyen tanulható nyelv (~)

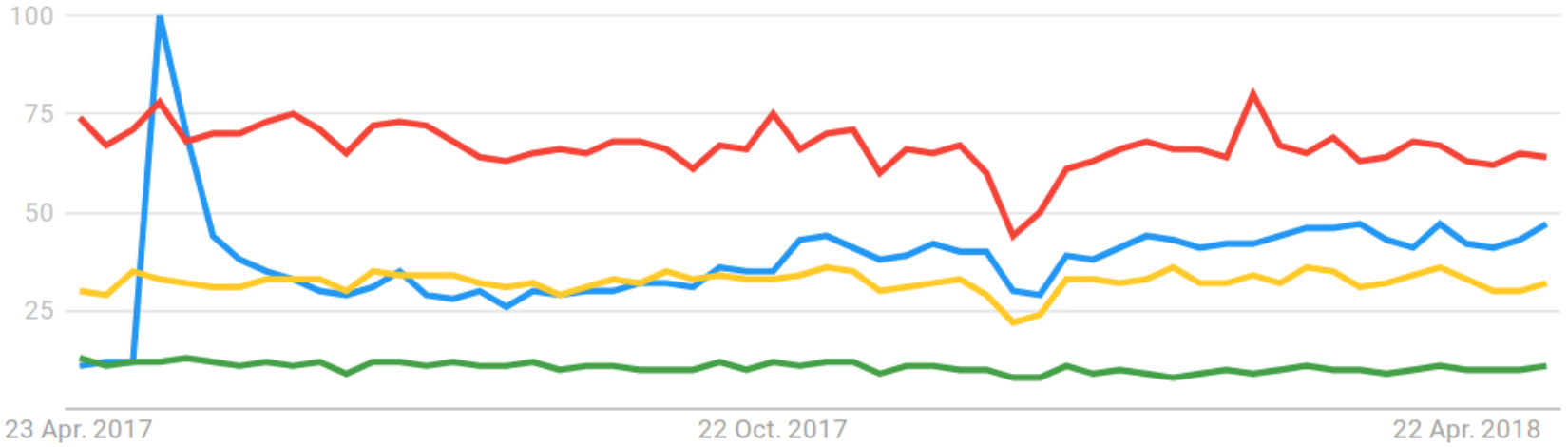
Kotlin nyelv bevezetése 2/2

- Esetleges hátrányok
  - JetBrains-től függ a továbbfejlesztés
  - Alacsony támogatás Eclipse és NetBeans környezetben
- Bevezetés lehetősége kis lépésekben
  - Kezdjünk kis kódbázison, alacsony üzleti érdekeltségű modulban
  - Prototípusok, belső eszközök
  - Kis kompromisszumokra egyszerűbb jóváhagyást szerezni☺
  - Referencia után már könnyebb a használat kiterjesztése



Kotlin terjedése

- Forrás: Google Trends



Összefoglalás

- Pong alkalmazás
- Alkalmazás komponensek
- Activity életciklus
- Back Stack
- Több Activity kezelése
- Barkóba alkalmazás

A következő alkalommal...

- Erőforrás típusok
- Erőforrásminősítők használata, jellemzőik
- Layout erőforrások (LinearLayout, RelativeLayout)
- Nézetek/View-k
- Képek kezelése egyszerűen
- Menükezelés
- Stílusok és témák használata
- Animációk

Köszönöm a figyelmet!

peter.ekler@aut.bme.hu

AutSoft