

# Mobil- és webes szoftverek

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)

01. Előadás

Mobil platformok, Android alapok, első alkalmazás

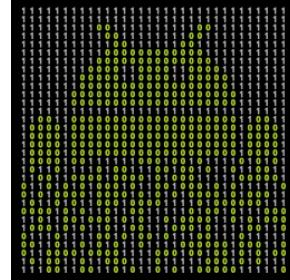


Department of  
Automation and  
Applied Informatics

# Mobil és web az alapképzésben?

- Igen! ☺
- A piaci igényekhez igazodva frissült az alapképzés
  - > Átalakuló követelmények és felhasználói igények
  - > Széles körű elterjedtség
- A tematikát folyamatosan frissen tartjuk!

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
  <head>
    <meta name="TITLE" content="BME / UT Alapképzés - Mobil és web alkalmazások" />
    <meta name="KEYWORD" content="BME, UT, Alapképzés, Mobil, web, alkalmazások" />
    <meta name="DESCRIPTION" content="BME / UT Alapképzés - Mobil és web alkalmazások" />
    <link rel="stylesheet" type="text/css" href="css/style.css" />
    <script language="JavaScript" type="text/javascript" src="js/jquery.js" />
  </head>
  <body bgcolor="#ffffcc" width="100%" height="100%>
```



# Előadók

- Ekler Péter
  - > Q.B.226
  - > [Ekler.Peter@aut.bme.hu](mailto:Ekler.Peter@aut.bme.hu)
- Gincsai Gábor
  - > Q.B.223
  - > [Gincsai.Gabor@aut.bme.hu](mailto:Gincsai.Gabor@aut.bme.hu)

# Hasznos linkek

- Honlap
  - > <https://edu.vik.bme.hu/course/view.php?id=4763>
- VIK oldal
  - > <https://portal.vik.bme.hu/kepzes/targyak/VIAUAC00>
- Demó kódok
  - > <https://github.com/VIAUAC00/EA>

# Oktatási forma

- Előadás
  - > Kedd 14:15
    - Teams Live
  - > Gyakorlat orientált
- Labor
  - > minden héten, a 2. héttől
  - > Kész megoldások feltöltése a Moodle portálra (File->Export to zip)

# Követelmények

- Előadás
- Laborok
  - > Beugró (igaz-hamis, a/b/c/d, mini kódrész)
    - Első labor esetén próba
  - > ~6 db kisZH
  - > Labor végén a megoldás feltöltése
- Házi feladat
  - > Mobil részből
  - > Specifikáció 6. hét végére (laborvezetővel egyeztetve)
  - > Véleges megoldás 13. hét végére
- ZH információk:
  - > ZH: 2020. 11. 17.
  - > PótZH: 2020. 12. 01.
- Vizsga

# Értékelés

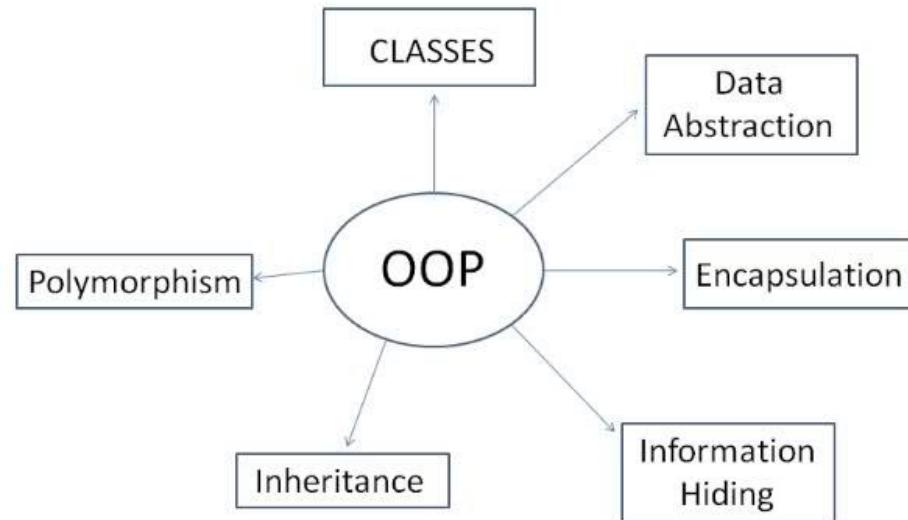
- ZH: 20%
  - Labor eredmények: 15%
  - KisZH: 10%
  - HF: 15%
  - Vizsga: 40%
- 
- A zárthelyin és a vizsgán az elégséges határa 40%

# iMSc pontok

- Max 30 pont
- ZH-n 10 pont:
  - > Extra feladatok
- NagyHF-n 10 pont:
  - > Extra funkciók, igényes felület, kódminőség
  - > Rövid dokumentáció a megoldásról és indoklása a fent felsorolt elemeknek
- Laboron 10 pont
  - > Extra feladatok
  - > Külön feltöltő felület (számonkérés)
- iMSc pont szerzésére bármely hallgató jogosult, aki az előtte lévő feladatokkal már végzett

# Javasolt előkészületek

- Objektum orientált programozás
  - > Készítettél már legalább egy projektet amiben több osztály volt átgondolt architektúrában
- Alábbi fogalmak ismerete:
  - > Osztály vs. példány
  - > Ősosztály, leszármazott
  - > Metódus felüldefiniálás
  - > Interface



# Tematika

- Android platform
  - > Mobil platform alapok
  - > Platform teljes áttekintése
  - > Android alkalmazás komponensek
  - > Felhasználói felület, kommunikáció
  - > Főként Java (kis Kotlin bevezetés)
- Webes rendszerek
  - > HTML, JavaScript, CSS
  - > Modern eszközök

# A kurzus célja

- Alkalmazás fejlesztés alapjainak bemutatása
- Gyakorlat orientált tapasztalatok
- Legjobb gyakorlatok és eszközök bemutatása
- Kódolási konvenciók és javaslatok
- Esettanulmányok



# Tartalom

- Mobil platformok áttekintése
- Android bevezetés
- Android platform szerkezete
- Fejlesztőkörnyezet bemutatása
- Android HelloWorld

# Mobilszoftver- platformok



Department of  
Automation and  
Applied Informatics

# A mobilpiac szereplői

- Hálózat operátor
  - > Kiépíti és karbantartja a hálózatot, lehetővé téve a készülékek közötti kommunikációt
  - > Pl. T-Mobile, Telenor, Vodafone
- Szolgáltatók
  - > Különféle szolgáltatásokat nyúlt a mobilkészülékek ill. a –hálózat felhasználóinak
  - > Hangátvitel: Jelenleg még sokszor azonos az operátorral, de egyre inkább szétválik: VMNO (Virtual Mobile Network Operator)
  - > Pl. Skype, Google, alkalmazásfejlesztők, stb... is
  - > 5G
- Készülékgyártók
  - > Pl. Apple, Samsung, Asus, Huawei, OnePlus, ZTE, Xaomi, stb.
- Felhasználók

# Mobilkészülék funkciói

- Mire lehet használni egy mai modern mobilkészüléket?
  - > Kommunikáció (hang, email, sms, stb.)
  - > Szervezői funkciók (naptár, jegyzet)
  - > Webböngészés
  - > Játék
  - > Zene- / videólejátszás, fénykép / film...
  - > Navigáció
  - > Fizetés, azonosítás, kártyák
  - > VR
  - > ...

# Platformok

- Bevezetés
- Mobil szoftverplatformok történelme
- Symbian OS
- Java ME
- Python
- Windows Mobile / Windows Phone
- Maemo/Meego/Tizen
- Android
- iPhone OS
- Egyéb (ami kimaradt)

# Symbian OS

A történelem kezdete ☺



# Symbian OS - Bevezetés



- Kifejezetten mobilkészülékekre kifejlesztett operációs rendszer
- Hardver erőforrásokban szegény készülékekre
  - > Gyenge processzor
  - > Kevés memória
  - > Korlátozott üzemiidő (akkumulátor)
- Magas rendelkezésre állásra tervezve
  - > Reboot csak ritkán megengedett
- Személyi adatkezelő funkciók OS szintű támogatása (kontaktok, naptár, stb.)
- Kommunikációs protokollok széles tárházának támogatása
- Okostelefonokra

# Symbian OS – Fejlesztés

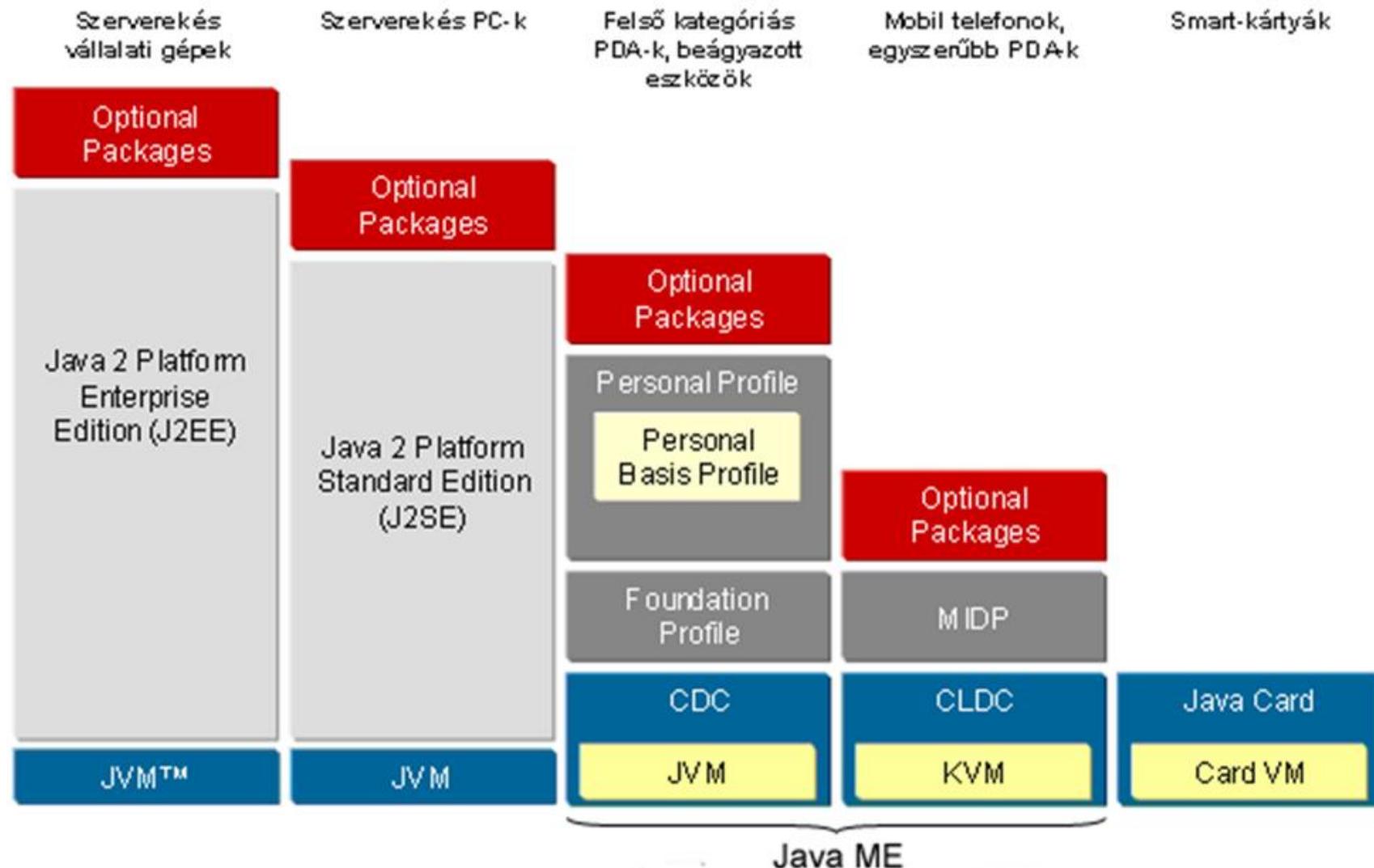
- Natív C++
  - > Magát az operációs rendszert is C++-ban írták
  - > Alacsony szintű API-k is elérhetők
  - > A C++ nyelv egy korlátozott, megkötésekkel teli verzióját használja
    - Hosszú tanulási idő
    - Új szemléletet igényel
- Open C/C++
- Qt, Qt quick (QML)
- Java ME
- Python

# Java ME



Department of  
Automation and  
Applied Informatics

# Java család



# Java ME - Csomagok

- A profilokon túl további un. opcionális kiegészítő csomagokkal bővíthetők az API-k
  - > Pl. adatbázis kapcsolat, grafikai függvények
- Példák
  - > Siemens C55
    - CLDC 1.0
    - MIDP 1.0
  - > Nokia 6600
    - CLDC 1.0
    - MIDP 2.0
    - JSR 205: Wireless Messaging API
      - Bluetooth



# Python



Department of  
Automation and  
Applied Informatics

# Python - Bevezetés

- Szkriptnyelv
- Magas szintű, objektumorientált
- Jól használható prototípus alkalmazások gyors fejlesztésére
- A kódot nem kell lefordítani, egy futtatókörnyezet (interpreter) hajtja végre
- A futtatókörnyezet több platformra is létezik (így elvileg platformfüggetlen a kód)
  - > Windows
  - > Linux
  - > Mobil platformok (S60, Meego, Android, stb.)



# Python Androidra

- Az SL4A (Scripting Layer for Android) plug-in-je
- Alaposan integrált fejlesztőeszközök
- Standard Python modulokon kívül API facade-ok különböző Androidos szolgáltatásokra

# Windows Mobile Windows Phone



Department of  
Automation and  
Applied Informatics

# Windows Mobile

- Windows mobil mutációja volt
  - > Okostelefonokra (Windows Mobile Standard)
  - > PDA-kra (Windows Mobile Professional, Classic)
- Windows CE-re épül
- Az operációs rendszerhez hozzá tartozik számos ismert alkalmazás mobil verziója (Excel Mobile, Word Mobile, stb.)
- Java ME, .NET CF



# Windows Phone 7, 8

- Könnyen testreszabható és mindenkor friss home screen
- Az úgynevezett „live tile”-ok mindenkor a legfontosabb információkat jelenítik meg (hívás, sms, naptár, időjárás, stb.)
- Fejlesztés Silverlight alapon (C#, VB)
  - > XAML
- vagy natív C++ (WP8)
- Népszerű szolgáltatásokat integrál:
  - > XBOX Live
  - > Office
- WP8 már nem CE alapú!



# Windows 10 mobile

- Asztali és mobil OS egyesülés
- Emiatt asztalival azonos elemek
  - > Kernel, UI elemek, menük, beállítások...
  - > Cortana...
- Universal Windows Platform alkalmazások:
  - > Egy API set és package az összes device támogatásához



# Maemo, MeeGo, Tizen



Department of  
Automation and  
Applied Informatics



# Meego- Bevezetés

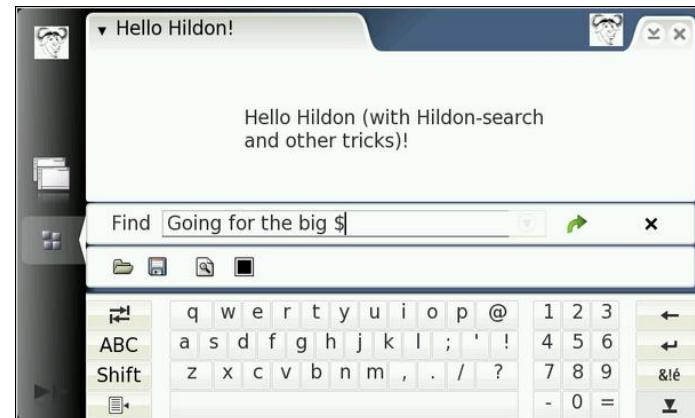
- Linux alapú operációs rendszer  
Internettáblákra, mobilokra, set-top boxokra
  - > Maemo + Intel MobLin
  - > Internetezésre kifejlesztve
  - > Nagyméretű érintőképernyő
  - > Debian GNU/Linux
- Fejlesztés
  - > Régen csak scratchbox, linux alól
  - > Python
  - > C/C++
- Qt + Mobility API

maemo™



# Meego

- Linux alkalmazások viszonylag könnyen portolhatók rá
- Elsősorban „Linux hackerek” fejlesztettek rá...
- Maemo 5: Nokia N900
- MeeGo: pl. Nokia N9



# Tizen

- MeeGo fejlesztők (Linux Foundation) és Inteleszek áttértek rá
- Samsung, Intel beálltak mögé
  - > Samsung: TV, óra, mobil... 50M eszköz
  - > Bada platformot is beolvastotta
  - > „-Galaxy”
- Fejlesztés:
  - > Natív appok: C, C++, Python, Lua
  - > HTML5 és JavaScript alapú appok
  - > .NET support
  - > Android alkalmazások
    - (módosított Dalvik VM)



# iOS



# iOS

- Eredetileg iPhone OS, az Apple-től
  - > Specialitás: Exkluzívan az ő hardvereirek
- 14 jelenleg
- Objective C, Swift
- Jól bevált eszközök
  - > XCode, Interface Builder, ...
- Apple újdonsága volt: Központilag kontrollált terjesztés, app modellek
- Rengeteg elérhető irodalom és dokumentáció

# iOS

- iPod Touch (2007), iPhone (2007), iPad (2010), Watch (2014)...
- Objective C, Swift
- Jól bevált eszközök
  - > XCode, Interface Builder, ...
- Speciális UX ötlet: multi-touch gesztúrák, közvetlen manipulálás
- OS X-el közös alapok (Core Foundation, Foundation Kit), de speciális UI (Cocoa Touch)

# További operációs rendszerek 1

- Palm OS, majd WebOS
  - > Operációs rendszer beágyazott rendszerekre
  - > Alapvetően PDA-kra
  - > Megvette a HP a Palm-ot, majd a HP kapitulált, most épp LG...
  - > Jelenleg nincs számottevően jelen a piacon
  - > smartTV felé mozdult (LG, 2015), de pl. hűtökön is...
- OpenMoko
  - > Linux alapú
  - > Teljesen nyílt platform (még a firmware-ek nagy része is GPL licenszen)
  - > Qtopia-ra épül (Trolltech mobil UI platformja)
  - > Jelenleg egyetlen kereskedelmi forgalomban kapható készülék: Neo (1973)



# További operációs rendszerek 2

- BlackBerry
  - > USA-ban igen elterjedt volt (2013-ban 85M készülék)
  - > Üzleti célú felhasználás
  - > Szerver oldali alkalmazások, vállalati szinkronizáció (push üzenetek) tették naggyá
  - > Majd: BlackBerry 10, illetve PlayBook OS (2019-ben kivezetik ezeket)
  - > (2015-től BB az Android vonaton ül)
- Firefox OS (2016 májusig)
- Ubuntu for phone (2017 június)
- Jolla, Sailfish...

# Android bevezetés

# Az Android keletkezése

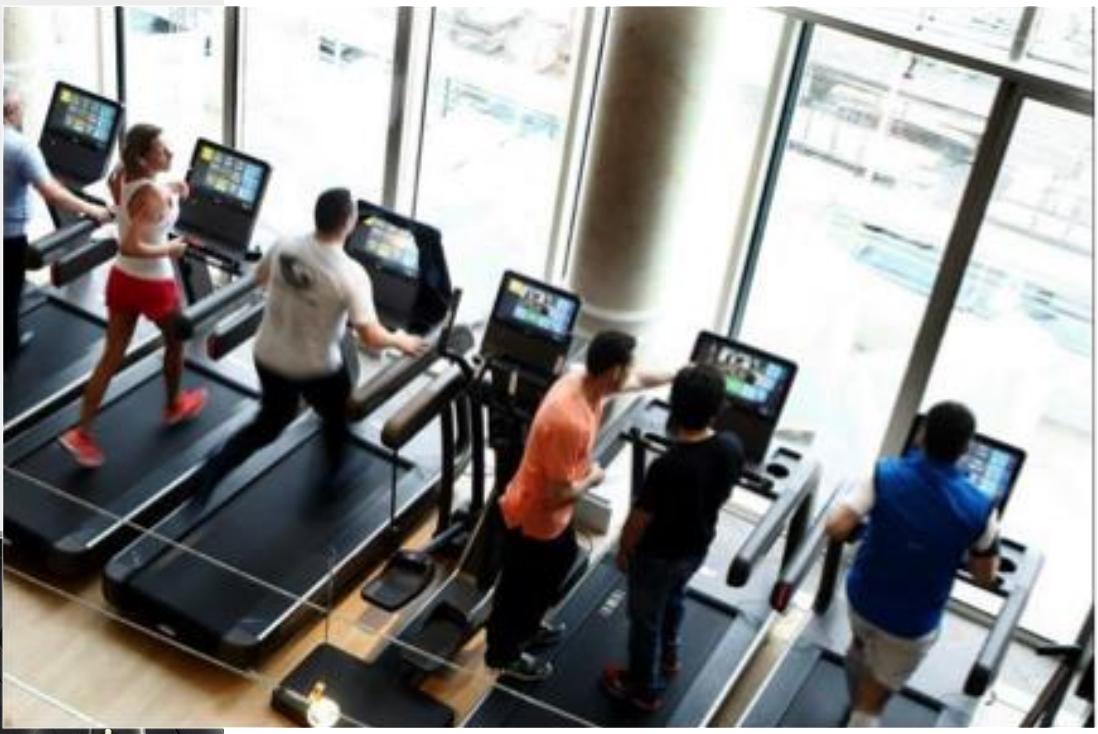
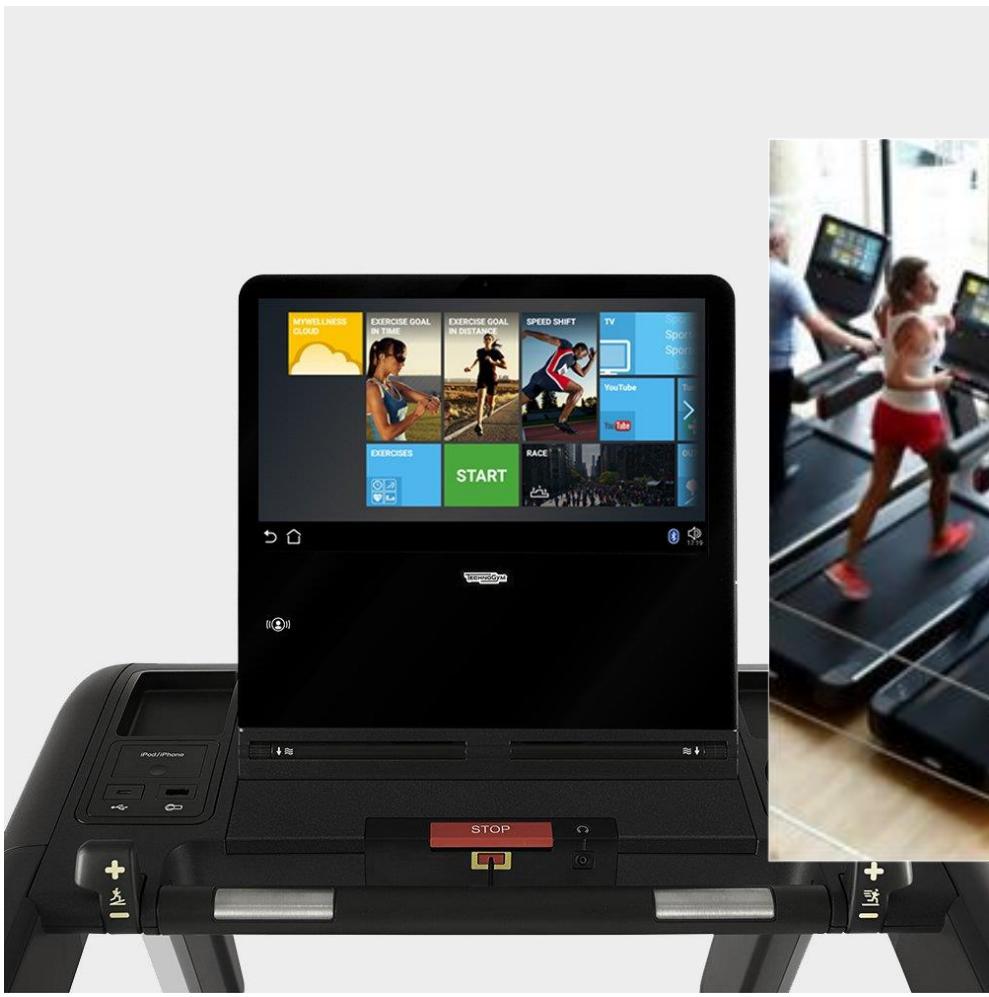
- Az Android napjaink egyik „sláger” mobil operációs rendszere
  - > Egy „brand” lett belőle (TV reklámok, hirdetések, stb.)
  - > Az „IT óriás” Google áll mögötte
- Egy egységes, kiválóan működő rendszer képét nyújtja
- Forradalmasította a mobil operációsrendszerekről alkotott képet
- De minden figyeljünk a részletekre is!

# Android eszközök 1/2

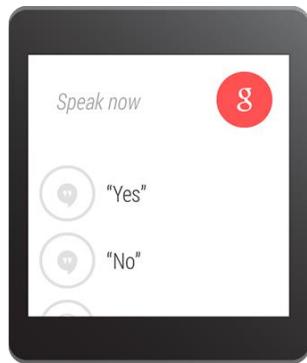
- Mobiltelefon és a Tablet gyártók
- Gépjárművek fedélzeti számítógépét és navigációját szállító cégek
- Android Wear
- Ipari automatizálás irányából is
- minden olyan helyen kényelmes az Android
  - > Alapvetően kicsi a kijelző (Google TV megoldás)
  - > Más jellegű erőforrások
  - > Az adatbevitel nem tipikusan egérrel és/vagy billentyűzettel történik
  - > Android@Home



# Android eszközök 2/2



# Android: nem csak mobil!



# Mi is az Android?

- Egyszerűen: operációs rendszer (nem csak mobiltelefonokra!)
- Másról: ezt a rendszert futtató eszközök (telefonok, táblagépek, stb.) összessége
- Részben vagy teljes egészében(?) a Google fejlesztése
- **2005**-ben felvásárlásra került az *Android Incorporated* nevű kaliforniai cég
- **2007** elején kezdtek kiszivárogni olyan hírek, hogy a Google belép a mobil piacra
- **2007 november 5-én** az Open Handset Alliance bejelentette az Android platformot
- **2008** végén piacra került a T-Mobile által forgalmazott, HTC G1-es készülék

# Mi volt a Google célja?

- Nyílt forráskódú, rugalmas, könnyen alakítható rendszer fejlesztése, melyre könnyű a külső alkalmazások fejlesztése
  - > Ok: külsős szoftverek is hozzáférnek a rendszer erőforrásainak jelentős részéhez
- Moduláris Linux kernel alapú
- A kód túlnyomó része Apache, nyílt forráskódú vagy szabad program licence alatt van (pl. org.apache csomagok)

# Android verziók

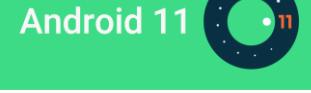
- Fontos a verziók nyomon követése
- Egyes verziók között komoly API-beli különbségek lehetnek
- Törekednek a visszafele kompatibilitásra, de lehetnek éles szakadékok (pl. 3.0)
- Fejlesztés előtt alaposan gondoljuk át a támogatott minimum verziót
- Verzió kódnev: valamilyen édesség ☺

# Android verziók



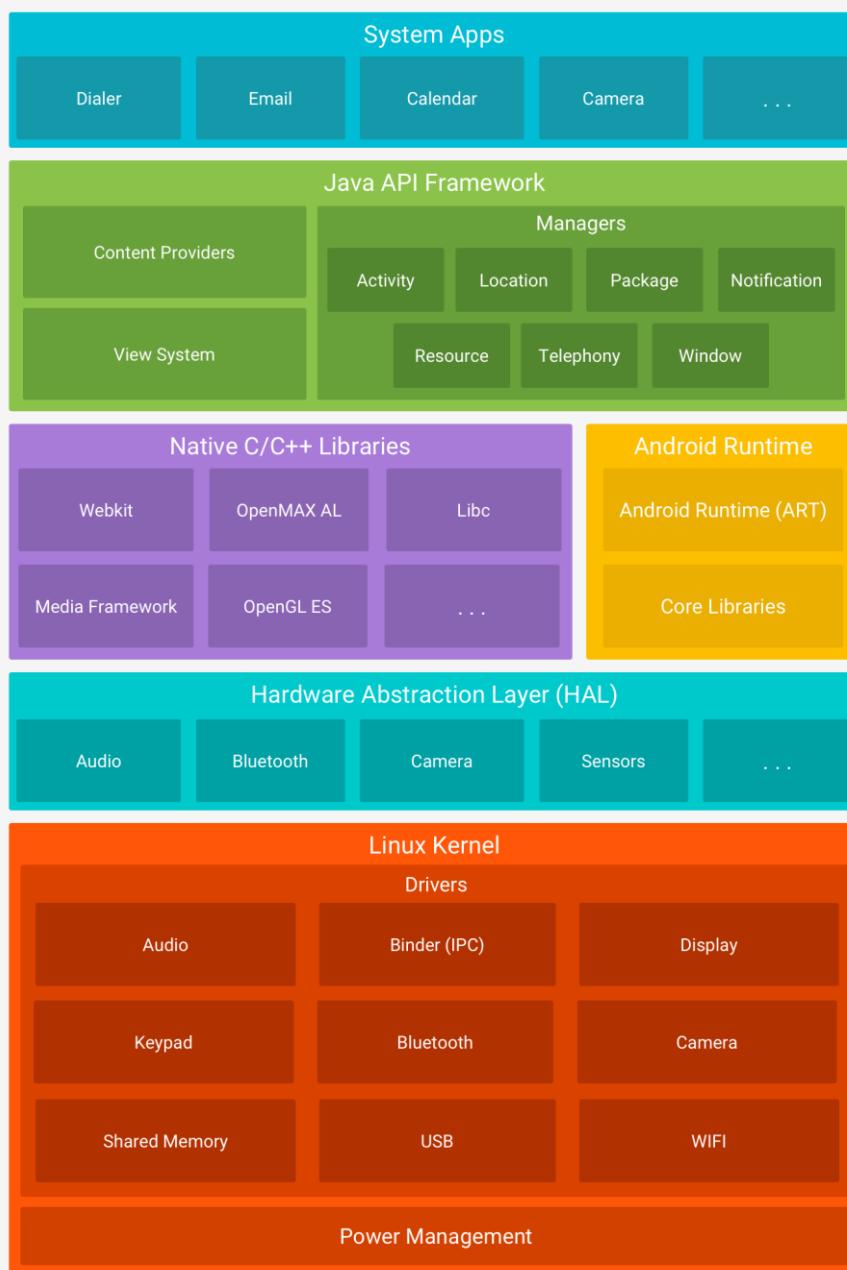
ANDROID

- Android 1.0 – 2008. October
- Android 1.1 – 2009. February
- Android 1.5 (Cupcake) – 2009. April
- Android 1.6 (Donut) – 2009. September
- Android 2.0 and 2.1 (Eclair) – 2009. October
- Android 2.2 (Froyo) – 2010. May
- Android 2.3 (Gingerbread) – 2010. December
- Android 3.0-3.2 (Honeycomb) – 2011 January-July
- Android 4.0 (Ice Cream Sandwich) – 2011. October
- Android 4.1 (Jelly Bean) – 2012. July
- Android 4.2 (Jelly Bean) – 2012. November
- Android 4.3 (Jelly Bean)
- Android 4.4 (KitKat)
- Android 5.0, 5.1 (Lollipop)
- Android 6.0 (Marshmallow)
- Android 7.0, 7.1 (Nougat)
- Android 8.0, 8.1 (Oreo)
- Android 9.0 (Pie)
- Android 10 (Q)
- Android 11



# Android platform szerkezete és fejlesztőkörnyezet

# Az Android platform szerkezete



ART: virtuális gép (Android RunTime)

Régebben: DVM  
Dalvik Virtual Machine

# A platform felépítése 1/3

- Távolról tekintve a platform felépítése egyszerű és világos
- A **vörös** színnel jelölt rész a Linux kernel, amely tartalmazza:
  - > A hardver által kezelendő eszközök meghajtó programjait
  - > Ezeket azon cégek készítik el, amelyek az Android platformot saját készülékükön használni kívánják, hiszen a gyártónál jobban más nem ismerheti a mobil eszközbe integrált perifériákat
  - > Memória kezelés, a folyamatok ütemezése és az alacsony fogyasztást elősegítő teljesítmény-kezelés

# A platform felépítése 2/3

- A Linux kernelen futnak a **lila** részben található programkönyvtárak/szolgáltatások, mint például: libc, SSL, SQLite, stb. ezek C/C++ nyelven vannak megvalósítva
- Részben ezekre épül a sárga egységben található virtuális gép
  - > Nem kompatibilis a Sun virtuális gépével
  - > Teljesen más az utasítás készlete
  - > Más bináris programot futtat
  - > A Java programok nem egy-egy .class állományba kerülnek fordítás után, hanem egy nagyobb Dalvik Executable formátumba, amelynek kiterjesztése .dex, és általában kisebb, mint a forrásul szolgáló .class állományok mérete, mivel a több Java fájlban megtalálható konstansokat csak egyszer fordítja bele a fordító
  - > A Java csak mint nyelv jelenik meg!

# A platform felépítése 3/3

- A zöld és kék színnel jelölt részekben már csak Java forrást találunk
- Java forrást a virtuális gép futtatja, ez adja az Android lényegét:
  - > Látható és tapintható operációs rendszert
  - > Futó programokat
- A virtuális gép akár teljesen elrejti a Linux által használt fájlrendszert, és csak az Android Runtime által biztosított fájlrendszert láthatjuk

# Szoftverfejlesztési eszközök Android platformra

- **Android SDK (Software Development Kit):**
  - > Fejlesztő eszközök
  - > Emulátor kezelő (AVD Manager)
  - > Frissítési lehetőség
  - > Java, Kotlin
- **Android NDK (Native Development Kit):**
  - > Lehetővé teszi natív kód futtatását
  - > C++
- **Android ADK (Accessory Development Kit):**
  - > Támogatás Android kiegészítő eszközök gyártásához (dokkoló, egészségügyi eszközök, időjárás kiegészítő eszközök stb.)
  - > Android Open Accessory protocol (USB és Bluetooth)

# Az SDK felépítése

- add-ons/:
  - > Kiegészítők külső könyvtárak használatához, pl. Google APIs
- docs/:
  - > Offline dokumentáció
- platform-tools/:
  - > Eszközök, mint például az adb az emulátorok, készülékek vezérléséhez
- platforms/:
  - > Az egyes platform verziók
- samples/:
  - > Példa kódok platform verziónként
- tools/:
  - > Platform verzió független eszközök, pl.: emulátor, ddms, stb.
- SDK Manager.exe, AVD Manager.exe :
  - > SDK és AVD (emulátor) kezelő eszköz

# Fejlesztő eszköz: Android Studio

- 2013 májusában került bemutatásra a Google I/O-n
- IntelliJ IDEA alapú fejlesztőkörnyezet
- Windows, OSX, Linux támogatás
- Plugin fejlesztési lehetőség
- Fejlett refaktor képességek
- Teljes körű támogatás
- 4.0.1



# Emulátor

- Teljes operációs rendszer emulálása (lassabb emiatt)
  - > Beépített alkalmazások elérhetők
  - > Ctrl+F11 (screen orientáció állítás)
- Alternatíva: Genymotion emulátor (<https://www.genymotion.com/>)



# Android Hello World

# Az első Android alkalmazás

Ősosztály

```
public class HelloAndroid extends Activity {
```

Ősosztály  
implementáció  
meghívása

    Called when the activity is first created. \*/

ide

```
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);
```

```
        TextView tv = new TextView(this);  
        tv.setText("Hello BME!");  
        setContentView(tv);
```

TextView  
megjelenítése



# Android HelloWorld XML alapú UI-al 1/2

Hello Android XML (*layout/main.xml*):

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android=  
    "http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical" >  
    <TextView  
        android:id="@+id/tvHello"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/hello" />  
</LinearLayout>
```

Egyedi ID

# Android HelloWorld XML alapú UI-al 2/2

```
package hu.bute.daai.amorg.examples;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView myTextView = (TextView) findViewById(R.id.tvHello);
        myTextView.append("\n--MODIFIED--");
    }
}
```

XML alapú layout

UI komponens kikeresése ID  
alapján

# Egyszerű esemény kezelés

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    final TextView myTextView =  
        (TextView)findViewById(R.id.tvHello);  
    myTextView.append("\n--MODIFIED--");  
    myTextView.setOnClickListener(new OnClickListener() {  
        public void onClick(View v) {  
            myTextView.append("\n--CLICKED--");  
        }  
    });  
}
```

Mivel anonim  
osztályból férünk  
hozzá

Egyszerű érintés  
esemény kezelés

Holnaptól Kotlinn  
kell tanítanunk

Bazzeg

# Az első Android alkalmazás Kotlin-ban ☺



# Egyszerű esemény kezelés

Kotlin extensions miatt  
használható

Lambda hívás

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        myTextView.append("#")  
  
        myTextView.setOnClickListener{  
            myTextView.append("\n--CLICKED--")  
        }  
    }  
}
```

# Függvény mint paraméter

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btnTime.setOnClickListener(::click)  
    }  
  
    private fun click(view: View) {  
        Toast.makeText(this,  
            Date(System.currentTimeMillis()).toString(),  
            Toast.LENGTH_LONG).show()  
    }  
}
```

# Eseménykezelő megadása layout-ban

# Összefoglalás

- Tárgykövetelmények
- A mobil piac helyzete napjainkban
- Mobil platformok
- Android bevezetés
- Android fejlesztőkörnyezet és eszközök
- *1. Labor:* Android fejlesztő környezet megismerése és a legfontosabb eszközök kipróbálása

# Mi várható a félévben?

# Szoftver minőség

Hány százalékát használják valójában az alkalmazásoknak?

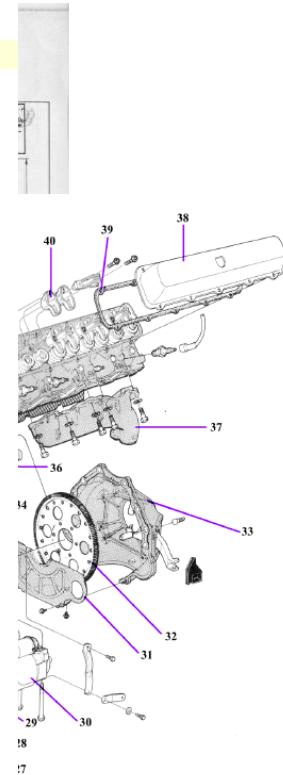
5%  
desktop/web



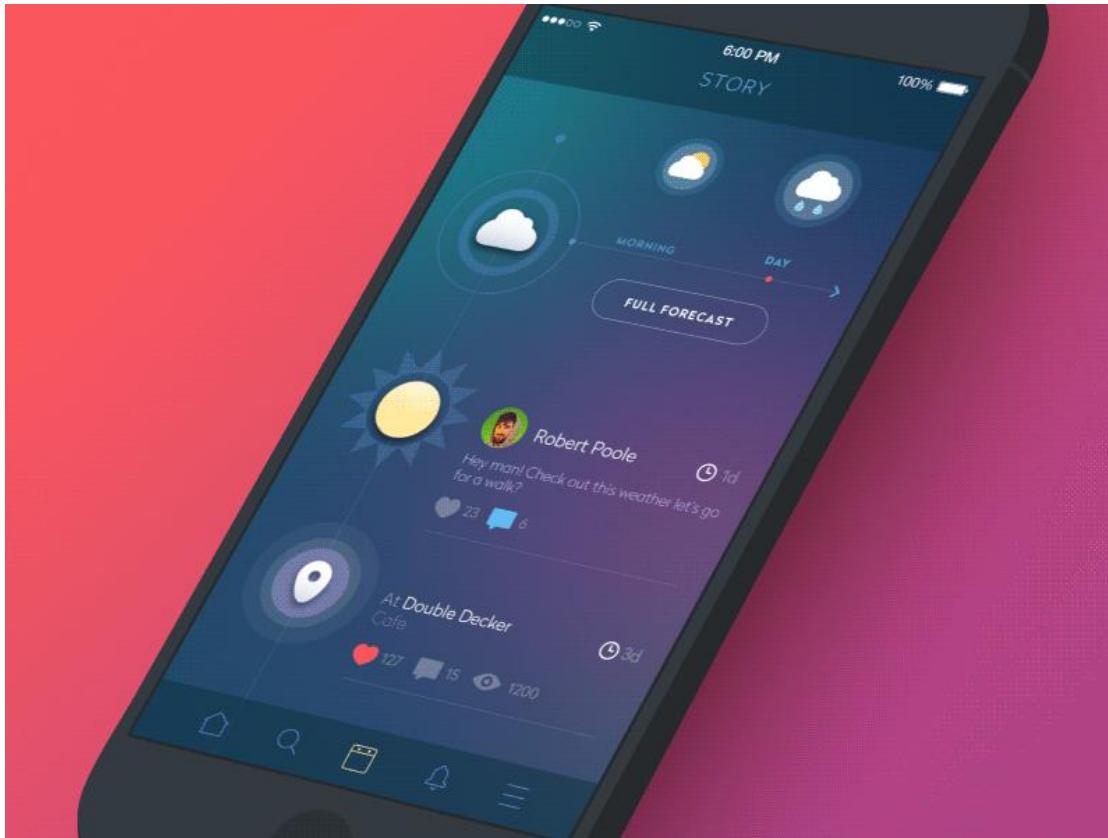
~1% mobil

# Szoftverfejlesztő feladata

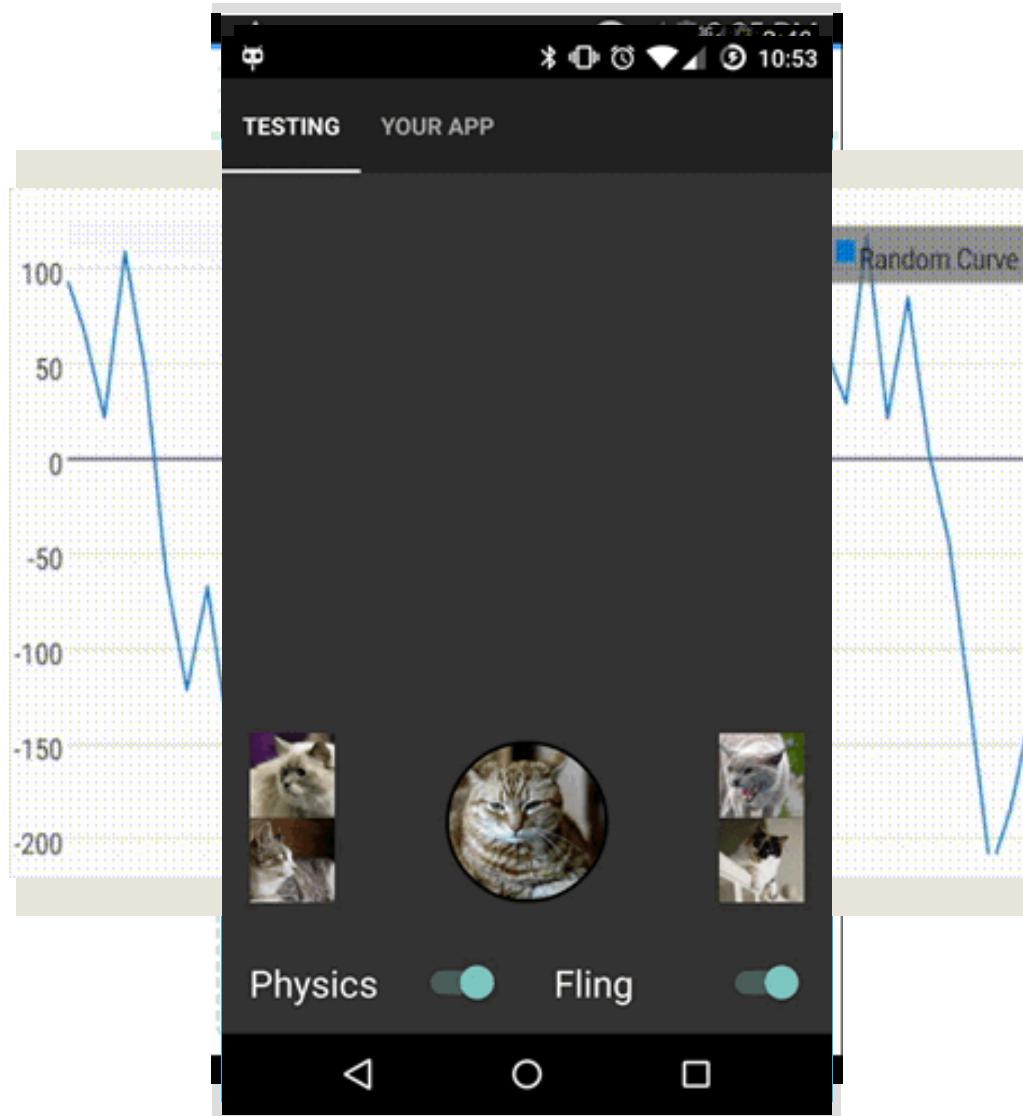
```
* upload, independent of whether the original activity is paused, stopped,  
💡 or finished.  
*/  
  
public class Activity extends ContextThemeWrapper  
    implements LayoutInflater.Factory2,  
    Window.Callback, KeyEvent.Callback,  
    OnCreateContextMenuListener, ComponentCallbacks2,  
    Window.OnWindowDismissedCallback {  
  
    private static final String TAG = "Activity";  
    private static final boolean DEBUG生命周期 = false;  
  
    /** Standard activity result: operation canceled. */  
    public static final int RESULT_CANCELED = 0;  
    /** Standard activity result: operation succeeded. */  
    public static final int RESULT_OK = -1;  
    /** Start of user-defined activity results. */  
    public static final int RESULT_FIRST_USER = 1;  
  
    static final String FRAGMENTS_TAG = "android:fragments";  
  
    private static final String WINDOW_HIERARCHY_TAG = "android:viewHierarchyState";  
    private static final String SAVED_DIALOG_IDS_KEY = "android:savedDialogIds";  
    private static final String SAVED_DIALOGS_TAG = "android:savedDialogs";  
    private static final String SAVED_DIALOG_KEY_PREFIX = "android:dialog_";  
    private static final String SAVED_DIALOG_ARGS_KEY_PREFIX = "android:dialog_args_";  
  
    private static class ManagedDialog {  
        Dialog mDialog;  
        Bundle mArgs;  
    }  
    private SparseArray<ManagedDialog> mManagedDialogs;  
  
    // set by the thread after the constructor and before onCreate(Bundle savedInstanceState) is called.  
    private Instrumentation mInstrumentation;  
    private IBinder mToken;  
    private int mIdent;  
    /*package*/ String mEmbeddedID;  
    private Application mApplication;  
    /*package*/ Intent mIntent;
```



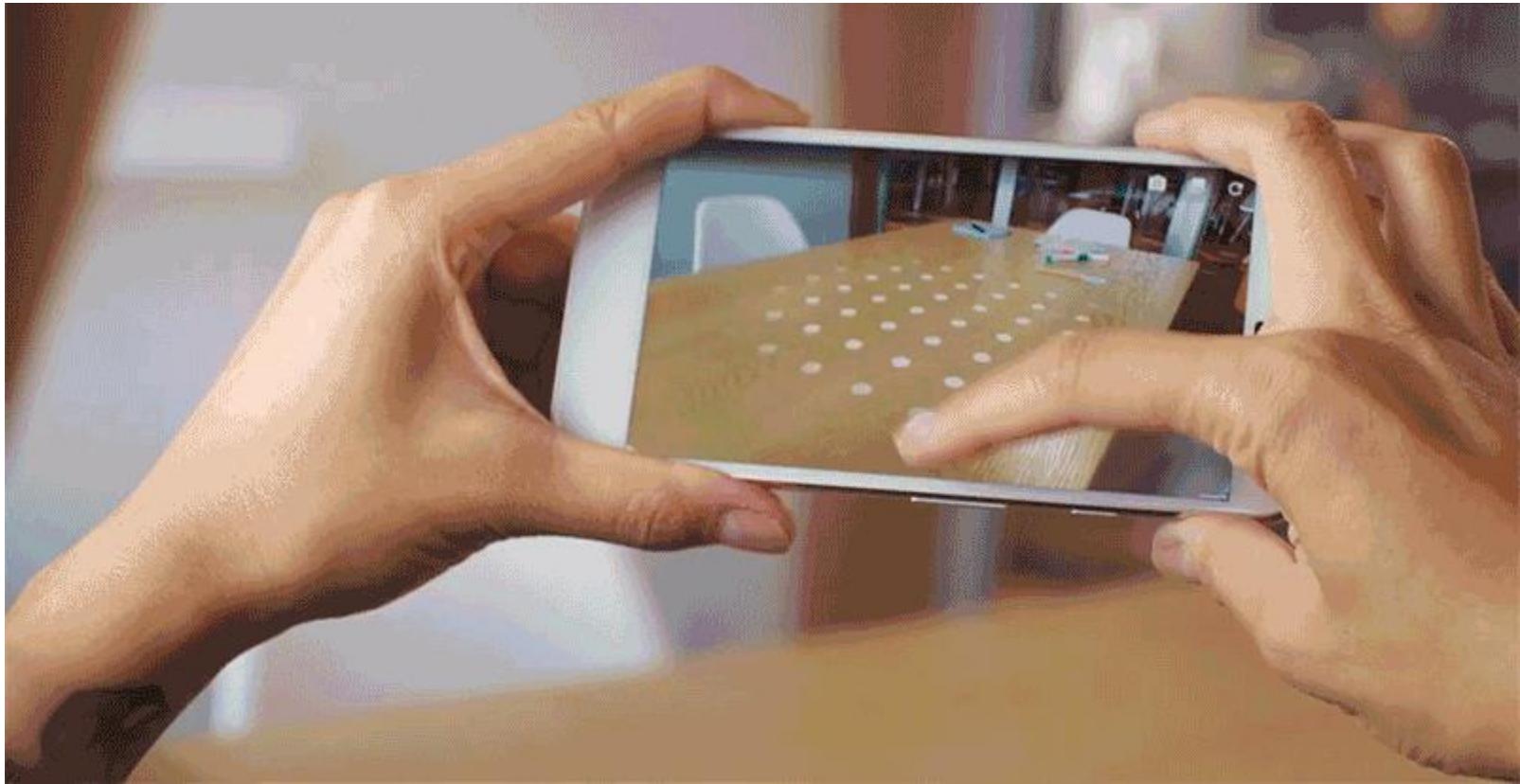
# Felhasználói felület



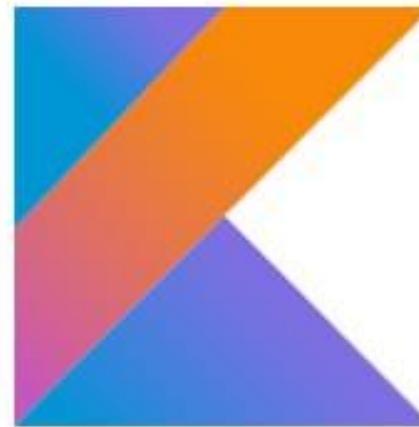
# Felületi elemek



# Kiterjesztett valóság



# Kotlin (Android) – Swift (iOS)



## Swift is like Kotlin

Swift

```
var movieCount = 0
var songCount = 0

for item in library {
    if item is Movie {
        movieCount += 1
    } else if item is Song {
        songCount += 1
    }
}
```

Kotlin

```
var movieCount = 0
var songCount = 0

for (item in library) {
    if (item is Movie) {
        ++movieCount
    } else if (item is Song) {
        ++songCount
    }
}
```

# Karrier út

- Junior Android fejlesztő
- Medior Android fejlesztő
- Senior Android fejlesztő
- Vezető fejlesztő
- Architekt



# Köszönöm a figyelmet!

The graphic consists of several text elements arranged around the central 'THANK YOU' message:

- Top Left:** 'GRACIAS' in large, bold, black letters.
- Top Right:** 'BIYAN SHUKRIA' in smaller, vertical black letters.
- Middle Left:** 'ARIGATO SHUKURIA' in bold black letters.
- Middle Center:** 'TASHAKKUR ATU' in bold black letters.
- Middle Right:** 'TINGKI' in small black letters above 'THANK'.
- Bottom Left:** 'JUSPAXAR' in bold black letters.
- Bottom Center:** 'BOLZİN MERCI' in bold black letters.
- Bottom Right:** 'THANK YOU' in large, bold, black letters.

Surrounding these main words are numerous smaller, semi-transparent text snippets representing various languages and cultures, such as 'SPASIBO DANKSCHEEN', 'HURUN SNACHALHUYA', 'YAQHANYELAY CHALTU', 'WABEEJA MAITEKA HUI', 'SUKSAMA EKHMET VYUPGARATAM', 'MAJKE PHANVYRAAD UNHACHESH', 'GRAZIE ATTO AHUA SPASIBO DENKAU-JA', 'MEHRBANI PALDIES SIKOMO', 'GOZAIMASHITA NEHACHALIYA', 'EFCHARISTO AGUJJE MAKETTA', 'FAKAAU MINMONCHAR', and 'TAVAPUCHI MEDAWAGGI'. These smaller texts are repeated in multiple colors (black, white, grey) across the background.

# Mobil- és webes szoftverek

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)

02. Előadás  
Activity életciklus, több képernyős alkalmazások



Department of  
Automation and  
Applied Informatics

# Információk

- ZH, PótZH és KisZH időpontok:
  - > <https://edu.vik.bme.hu/course/view.php?id=4763>
- IMSc pontok:
  - > Külön feltöltési űrlap a laboroknál, ugyanazt a projektet kell feltölteni, ha van benne IMSc megoldás is (így lesz külön oszlop az eredmények táblába nekünk)
- Beugró:
  - > 2 véletlen kérdés, 50% kell a sikerhez

# Tartalom

- Kotlin alapok
- Android projekt felépítése, fordítás mechanizmusa
- Android alkalmazás komponensek
- Manifest állomány, jogosultságok
- Activity Back Stack
- Multitasking
- Navigálás Activity-k között

# Kotlin alapok

Forrás: <https://kotlinlang.org/docs/reference/>



# A Kotlin főbb jellemzői

- JVM byte kódra (vagy akár JavaScriptre is) fordul
- Meglévő Java API-k, keretrendszerek és könyvtárak használhatók
- Automatikus konverzió Java-ról Kotlinra
- Null-safety
  - > Vége a NullPointerException korszaknak
- Kód review továbbra is egyszerű
  - > A nyelv alapos ismerete nélkül is olvasható a kód

# Konstansok, változók (val vs. var)

- Egyszeri értékkadás – „val”

```
val score: Int = 1 // azonnali értékkadás
val idx = 2 // típus elhagyható
val age: Int // típus szükséges ha nincs azonnali értékkadás
age = 3 // későbbi értékkadás
```

- Változók (megváltoztatható) – „var”

```
var score = 0 // típus elhagyható
score += 1
```

- String sablonok

```
var score = 1
val scoreText = "$score pont"
```

```
score = 2
// egyszerű kifejezések string-ek esetében:
val newScoreText = "${scoreText.replace("pont", "volt, most ")} $score"
```

# Változók null értéke

- Alapból a változók értéke nem lehet `null`

```
var a: Int = null  
error: null can not be a value of a non-null type Int
```

- A '?' operátorral engedélyezhetjük a `null` értéket

```
var a: Int? = null
```

> Lista, melyben lehetnek `null` elemek

> Lista, mely lehet `null`

> Lista, mely lehet `null` és az elemei is lehetnek `null`-ok

```
var x: List<String?> =  
    listOf(null, null,  
    null)
```

```
var x: List<String>? = null
```

```
var x: List<String?>?  
= null  
x = listOf(null,  
null, null)
```

# Null tesztelés és az Elvis operátor

```
var nullTest : Int? = null  
nullTest?.inc()
```

- > inc() nem hívódik meg, ha nullTest null

```
var x: Int? = 4  
var y = x?.toString() ?: ""
```

- > ha x null, akkor y "" értéket kap

# “Double bang” operator

- Kivételt dob, ha a változó értéke null

```
var x: Int? = null  
x!!.toString()  
kotlin.NullPointerException
```

# Függvények

- Függvény szintaxis

```
fun add(a: Int, b: Int): Int {  
    return a + b  
}
```

- Kifejezés törzs, visszatérési típus elhagyható

```
fun add(a: Int, b: Int) = a + b
```

- Érték nélküli visszatérés – Unit

```
fun printAddResult(a: Int, b: Int): Unit {  
    println("$a + $b értéke: ${a + b}")  
}
```

- Unit elhagyható

```
fun printAddResult(a: Int, b: Int) {  
    println("$a + $b értéke: ${a + b}")  
}
```

# Osztályok

```
class Car {  
    private String type;  
  
    public Car(String type) {  
        this.type = type;  
    }  
}  
  
class Car constructor(val type: String) {  
    val typeUpper = type.toUpperCase()  
  
    init {  
        Log.d("TAG_DEMO", "Car created: ${type}")  
    }  
  
    constructor(type: String, model: String) : this(type) {  
        Log.d("TAG_DEMO", "Car model: ${model}")  
    }  
}
```

# Osztályok

```
class Car constructor(val type: String) {  
    val typeUpper = type.toUpperCase()  
  
    init {  
        Log.d("TAG_DEMO", "Car created: ${type}")  
    }  
  
    constructor(type: String, model: String) : this(type) {  
        Log.d("TAG_DEMO", "Car model: ${model}")  
    }  
  
    // példányosítás  
    val car = Car("Toyota")
```

constructor elhagyható

primary constructor paraméterekkel

primary constructor tagváltozóira lehet hivatkozni

primary constructor inicializáló blokk

secondary constructor

# Leszármaztatás

alapesetben minden final

```
open class Item(price: Int) {  
    open fun calculatePrice() {}  
    fun load() {}  
}
```

öröklés

```
class SpecialItem(price : Int) : Item(price) {  
    final override fun calculatePrice() {}  
}
```

Később már nem  
lehet felülírni

# Osztály elemek

opcionális hozzáférés  
módosító

konstruktur  
opcionális  
hozzáférés  
módosítója

kulcsszó (kötelező, ha van  
hozzáférés módosítója a  
konstruktornak)

fejléc

```
public class Car internal constructor(aPlateNumber: String) {
```

```
    val plateNumber: String  
    var motorNumber: String? = null
```

read-only  
property

mutable  
property

```
    constructor(aPlateNumber: String, aMotorNumber: String):  
        this(aPlateNumber) {  
            motorNumber = aMotorNumber.toUpperCase()  
        }
```

az elsődleges  
konstruktornak  
nincs body-ja

inicializáló blokk

másodlagos  
konstruktur

```
    fun start(targetVelocity: Int) {  
        // some code  
    }
```

függvény

# Java field vs. Kotlin property

## Java

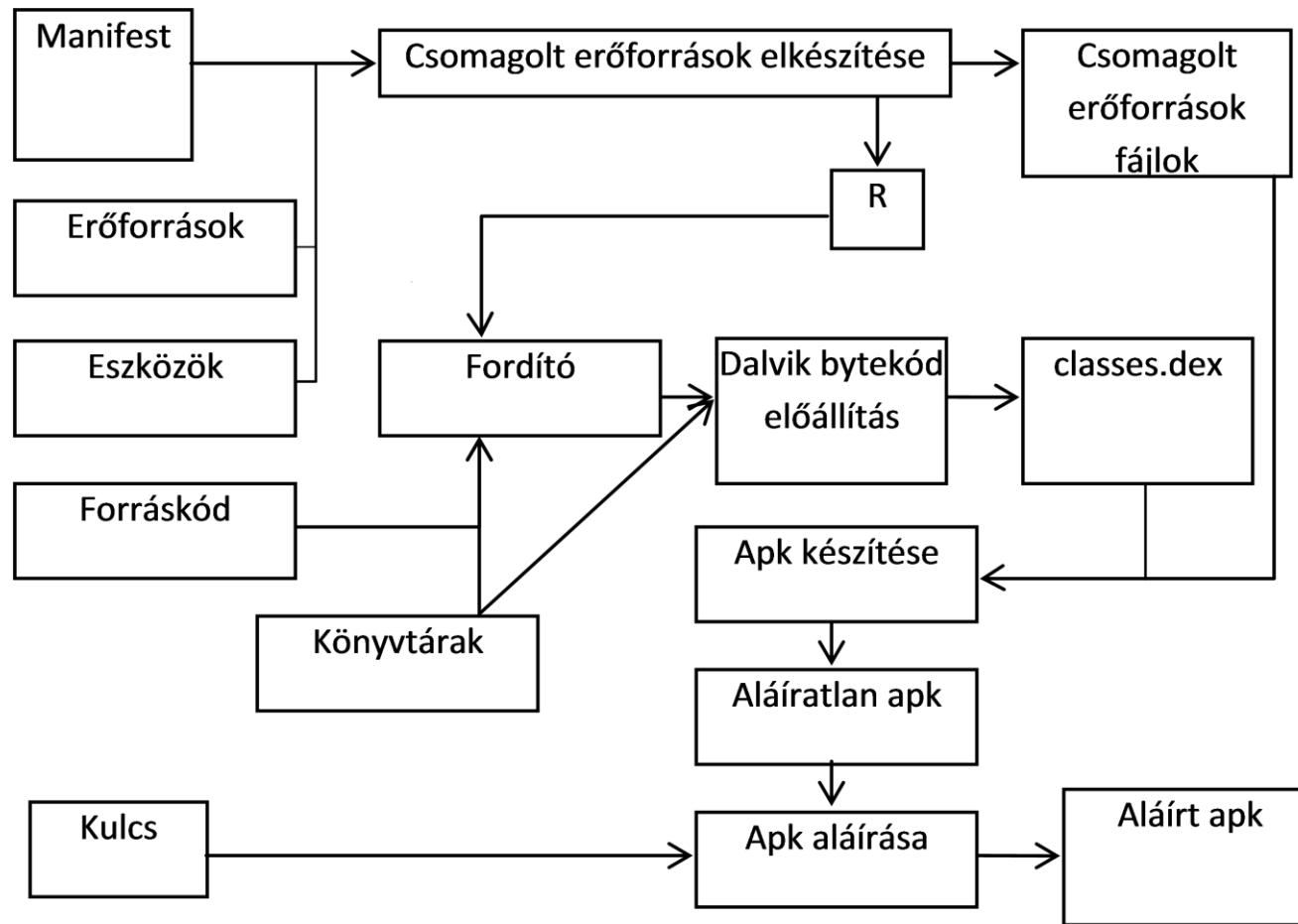
```
public class Car {  
    private String type;  
  
    public String getType() {  
        return type;  
    }  
  
    public void setType(String type) {  
        Log.d("TAG_CAR", "type SET");  
        this.type = type;  
    }  
}
```

## Kotlin

```
class Car {  
    var type: String? = null  
    set(type) {  
        Log.d("TAG_CAR", "type SET")  
        field = type  
    }  
}
```

# Android projekt felépítése, fordítás

# A fordítás menete (forrás->.apk)



# Az Android .apk állomány

- Leginkább a Java világban megszokott jar-hoz hasonlítható, de vannak jelentős eltérések
- Tömörített állomány, mely tipikusan a következő tartalommal rendelkezik:
  - > META-INF könyvtár
    - CERT.RSA: alkalmazás tanúsítvány
    - MANIFEST.MF: meta információk kulcs érték párokban
    - CERT.SF: erőforrások listája és SHA-1 hash értékük, pl:

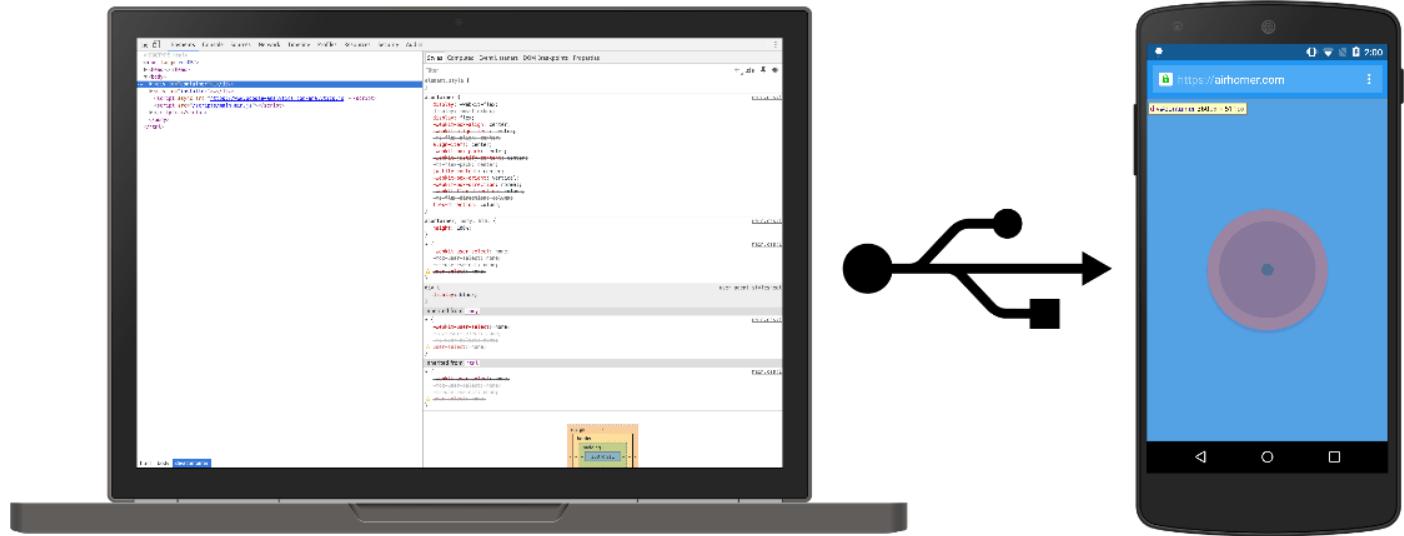
```
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: wxqnEAI0UA5nO5QJ8CGMwj kGGWE=
...
Name: res/layout/exchange_component_back_bottom.xml
SHA1-Digest: eACjMjESj7Zkf0cBFTZ0nqWrt7w=
...
Name: res/drawable-hdpi/icon.png
SHA1-Digest: DGEqylP8W0n0iV/ZzBx3MW0WGCA=
```
  - > Res könyvtár: erőforrásokat tartalmazza
  - > AndroidManifest.xml: név, verzió, jogosultság, könyvtárak
  - > classes.dex: lefordított osztályok a VM számára érhető formátumban
  - > resources.arsc

# Android alkalmazások telepítése

- A Play-ből az alkalmazások egy .apk állományban kerülnek letöltésre
  - > Vagy App Bundle-ban
- A telepítésért nem a Play alkalmazás, hanem egy úgynévezett *PackageManagerService* felelős
- A PackageManagerService akkor is látható, ha direktbe töltjük le az .apk-t
- Az alkalmazások telepíthetők a készülék memóriájára és bizonyos körülmények között külső SD kártyára is

# Tesztelés valós telefonon

- Fejlesztői mód bekapcsolása
- USB debugging engedélyezése
- Készülék összekötése és jóváhagyás



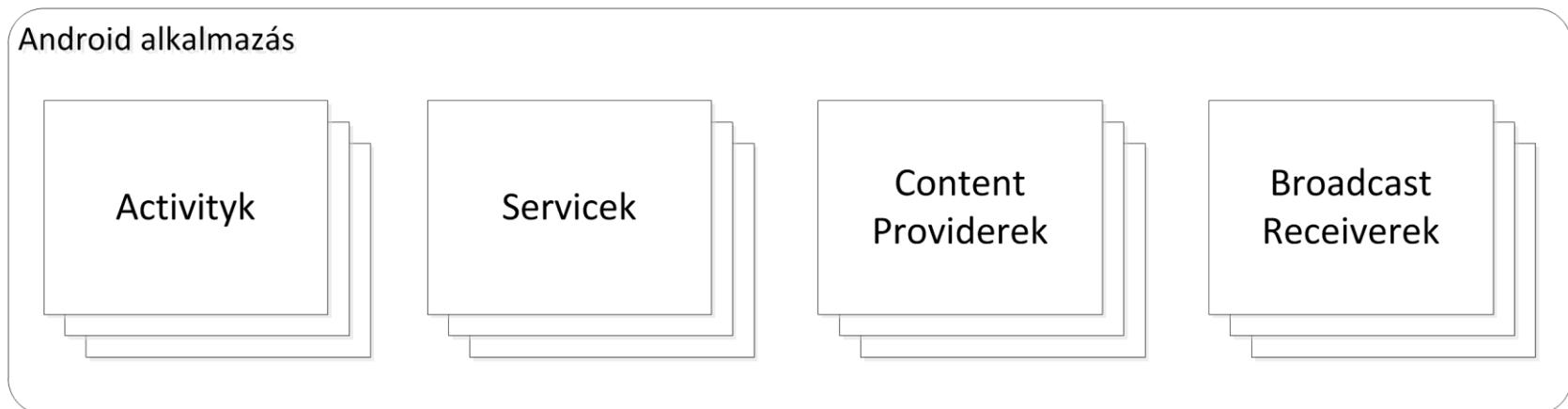
# Android LogCat

- Rendszer debug kimenet
- Beépített rendszer üzenetek is monitorozhatók
- Beépített Log osztály
  - > v(String, String) (verbose)
  - > d(String, String) (debug)
  - > i(String, String) (information)
  - > w(String, String) (warning)
  - > e(String, String) (error)
- Log.i("MyActivity", "Pozíció: " + position);
- Átirányítható file-ba is
  - > *adb logcat > myfile.txt*

# Android alkalmazás komponensek

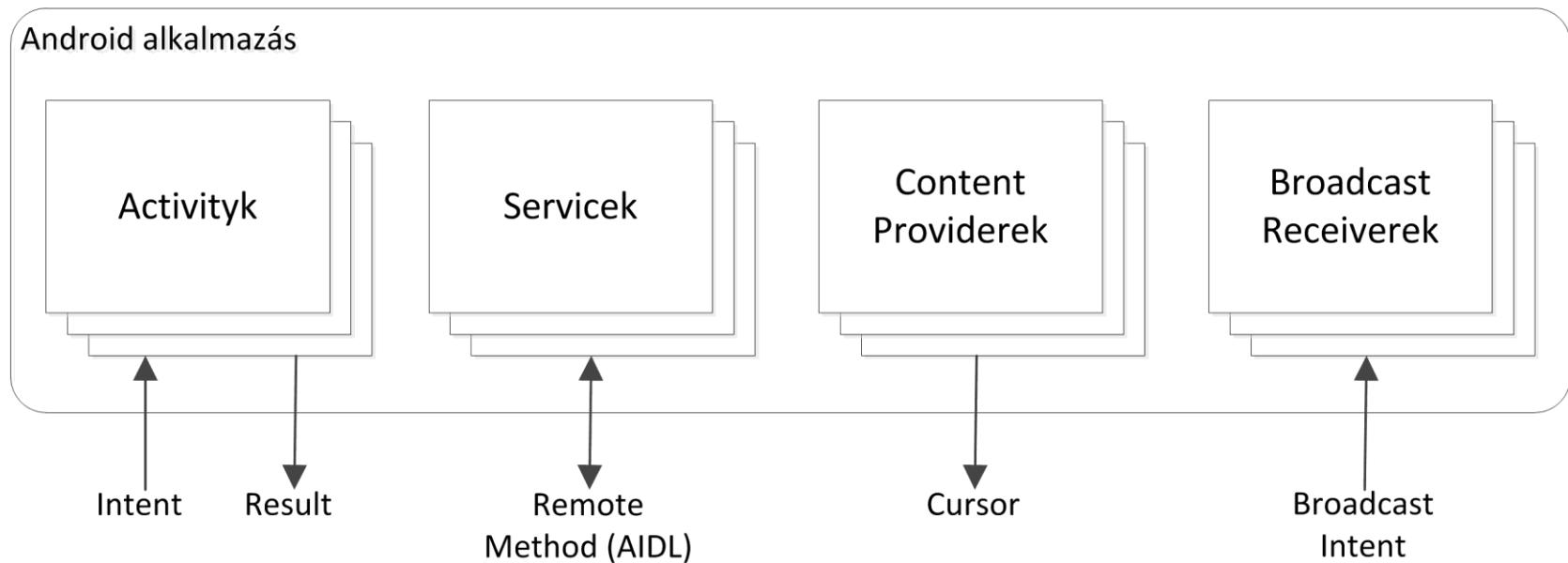
# Android alkalmazás felépítése 1/3

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



# Android alkalmazás felépítése 2/3

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



# Android alkalmazás felépítése 3/3

- 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-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

# 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

Mi nem igaz az alkalmazás komponensekkel kapcsolatban?

- A. 4 Android alkalmazás komponens van.
- B. Kötelező legalább egy Activity egy alkalmazáshoz.
- C. Készíthetünk UI nélküli alkalmazásokat is.
- D. A ContentProvider WebServeren tárolt adatokat is elérhetővé tud tenni.

# Manifest állomány

# Manifest állomány

- Alkalmazás leíró, definiálja az alkalmazás komponenseit
- XML állomány
- Komponens indítás előtt a rendszer a manifest állományt ellenőrzi, hogy definiálva van-e benne a kért komponens
- További feladatokat is ellát (pl. mik az alkalmazás futtatásának minimális követelményei)
- Alkalmazás telepítésekor ellenőrzi a rendszer



# Manifest állomány tartalma

- Alkalmazást tartalmazó java package – **egyedi azonosítóként szolgál**
- Engedélyek, amelyekre az alkalmazásnak szüksége van (pl. internet elérés, névjegyzék elérés, stb.)
- Futtatáshoz szükséges minimum API szint
- Hardware és software funkciók, amit az alkalmazás használ (pl. kamera, bluetooth, stb.)
- Külső API könyvtárak (pl. Google Maps API)

# Manifest példa 1/2

```
<?xml version="1.0" encoding="utf-8"?>  
  
<manifest xmlns:android=  
          "http://schemas.android.com/apk/res/android"  
          package="hu.bute.daai.amorg.examples"  
          android:versionCode="1"  
          android:versionName="1.0" >  
    <uses-sdk android:minSdkVersion="7" />  
  
    <application  
        android:icon="@drawable/ic_launcher"  
        android:label="@string/app_name" >  
      <activity ...>...</activity>  
    </application>  
  
</manifest>
```

Egyedi package név  
(azonosító)

Legkisebb támogatott  
verzió

Alkalmazás ikon és  
címke

# Manifest példa 2/2

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest .../>  
...  
<application ...>  
    <activity  
        android:name=".AndHelloWorldActivity"  
        android:label="@string/app_name">  
        <intent-filter>  
            <action android:name=  
                    "android.intent.action.MAIN"/>  
            <category android:name=  
                    "android.intent.category.LAUNCHER"/>  
        </intent-filter>  
    </activity>  
</application>  
</manifest>
```

The diagram illustrates the structure of an Android manifest file with annotations:

- Activity osztály és cím**: Points to the line `android:name=".AndHelloWorldActivity"`.
- Alkalmazás belépési pont jelölő**: Points to the line `"android.intent.action.MAIN"` within the `<action>` tag.
- Megjelenik a futtatható alkalmazások listájában (Launcher)**: Points to the line `"android.intent.category.LAUNCHER"` within the `<category>` tag.

# Mi igaz a Manifest állományra?

- A. Csak az Activity komponenseket kell felsorolni benne.
- B. Csak egy Service komponenst tartalmazhat.
- C. Az összes alkalmazás komponenst fel kell sorolni benne kivéve a dinamikusan regisztrálható BR komponenseket.
- D. XML és Java kód keveredhet benne.

# Erőforrások kezelése

# Alkalmazás erőforrások

- Egy Android alkalmazás nem csak forráskódból áll, hanem erőforrásokból is, úgy mint: képek, hanganyagok, stb.
- Emellett erőforrások az XML-ben definiált felületek is: elrendezés, animáció, menü, stílus, szín.
- Erőforrások használatával sokkal rugalmasabban változtatható az alkalmazás
- minden erőforráshoz a rendszer automatikusan egy egyedi azonosítót generál, amin keresztül elérhető a forráskódból

# Erőforrás hivatkozás példa

- Tegyük fel, hogy készítettünk egy `logo.png`-t és elmentettük a `res/drawable/` könyvtárba
- Az SDK eszköz előállít egy egyedi erőforrást hozzá mentés után automatikusan
- Az azonosító: `R.drawable.logo`
- Ezzel az azonosítóval lehet hivatkozni bárhol az erőforrásra
- Az azonosítók az `R.java` állományban tárolódnak (soha ne módosítsuk ezt az állományt!)

# Erőforrás használat előnyei

- Az egyik legnagyobb előny, hogy a készülék képességeihez lehet igazítani az erőforrásokat
- A könyvtárak után „minősítő”-ket írhatunk, amellyel megadjuk hogy mely tulajdonságok teljesülése esetén vegye a rendszer ebből a könyvtárból az erőforrásokat
- Többnyelvűség támogatása:
  - > *strings.xml*
  - > *res/values/*
  - > *res/values-fr/*
  - > *res/values-hu-land/*

# Activity életciklus

# 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

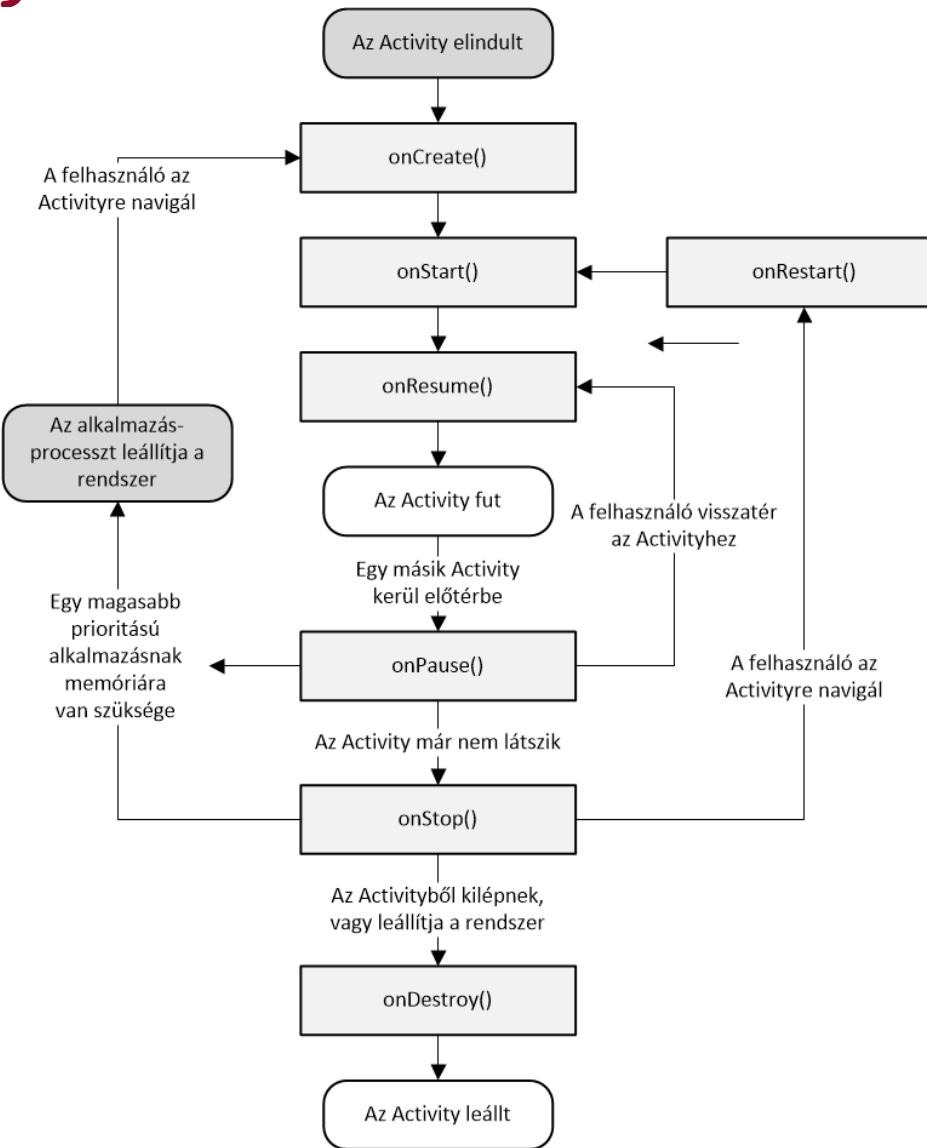
# Activity életciklus-callback

- Amikor egy Activity leáll egy másik indulása miatt, az Activity az eseményről értesítést kap az úgynevezett életciklus-callback metódusokon keresztül
- Számos callback metódus támogatott (create, stop, resume, destroy, stb.), amikre megfelelően reagálhat az Activity
- Például stop esemény hatására tipikusan a nagyobb objektumokat érdemes elengedni (DB/hálózati kapcsolat)
- Amikor az Activity visszatér (resume), újra kell kérni az erőforrásokat
- Ezek az átmenetek tipikus részei az Activity életciklusának

# Activity életciklus

- Egy megbízható és flexibilis alkalmazás esetén kritikus fontosságú az Activity életciklus-callback függvények megfelelő felüldefiniálása
- Az Activity életciklusát a vele együttműködő többi Activity határozza meg
- Elengedhetetlen az Activity működésének tesztelése a különböző életciklus állapotokban

# Activity életciklus modell



# Activity bezárása a rendszer által

- Paused, vagy Stopped állapotban a rendszer bármikor leállíthatja memória-felszabadítás céljából
- A leállítás történhet a *finish()* hívással, vagy kritikusabb esetben a Process leállításával
- Ha az Activity-t újra megnyitják (miután be lett zárva), a rendszer újra létrehozza

# Életciklus callback függvények

- Amikor az Activity állapotot vált, megfelelő callback függvények hívódnak meg
- Ezek a callback függvények „hook” jellegű függvények, melyeket a rendszer hív
- Fontos a metódusok felül definiálása és a megfelelő részek implementálása
  - > Mindig meg kell hívni az ōs osztály implementációját is (pl. `super.onCreate()` ;)!
- A rendszer felelőssége meghívni ezeket a függvényeket, de a fejlesztő felelőssége a helyes implementáció

# 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 életciklus callback függvények 1/2

- *onCreate()*: Activity létrejön és beállítja a megfelelő állapotokat (layout, munka szálak létrehozása, stb.)
- *onDestroy()*: minden még lefoglalt erőforrás felszabadítása
- *onStart()*: Az Activity látható, a vezérlők is. Például BroadcastReceiverek-re feliratkozás, amik módosítják a UI-t
- *onStop()*: Az Activity nem látható. Például BroadcastReceiverek-ről leiratkozás
  - > Az Activity élete során többször válthat látható és nem látható állapotok között.

# Activity életciklus callback függvények 2/2

- *onRestart()*: Az Activity leállítása (*onStop()*) majd újraindítása után hívódik meg, még az indítás (*onStart()*) előtt
- *onResume()*: Az Activity láthatóvá válik és előtérben van, a felhasználó eléri a vezérlőket és tudja kezelní azokat
- *onPause()*: Az Activity háttérbe kerül, de valamennyire látszik a háttérben, például egy másik Activity pop-up jelleggel előjön, vagy sleep állapotba kerül a készülék

# Mi igaz az Activity életciklus függvényekre?

- A. Kötelező minden életciklus függvényt felüldefiniálni, különben nem fordul az alkalmazás kódja.
- B. Kötelező az ősosztály implementációjának meghívása.
- C. Az Activity élete során minden függvény csak egyszer hívódhat meg.
- D. Szükség esetén manuálisan is meg kell hívni.

# Activity váltás

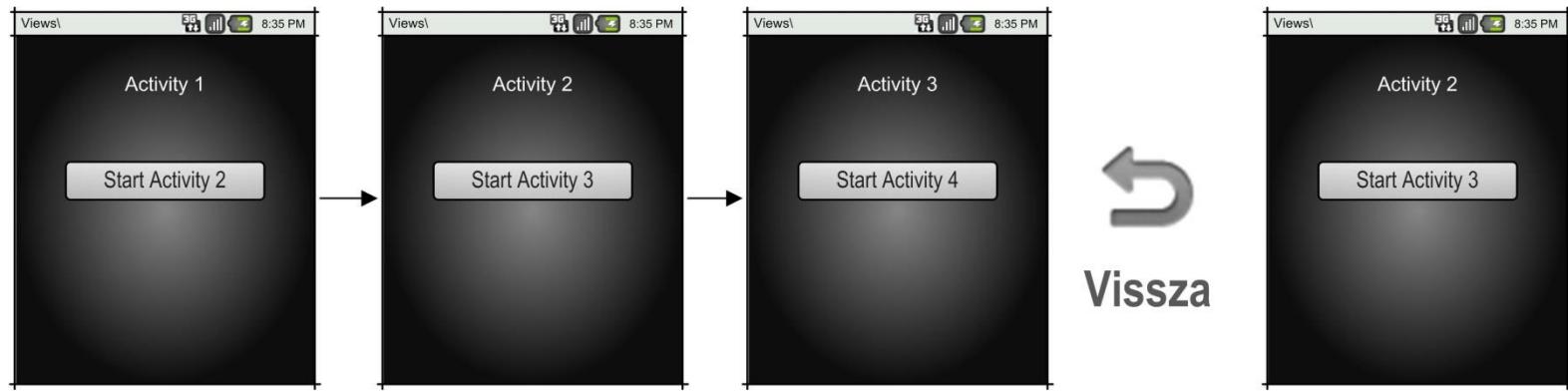
- Életciklus callback függvények meghívási sorrendje:
  - > A Activity *onPause()* függvénye
  - > B Activity *onCreate()*, *onStart()* és *onResume()* függvénye (B Activity-n van már a focus)
  - > A Activity *onStop()* függvénye, mivel már nem látható
- Ha a B Activity valamit adatbázisból olvas ki, 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

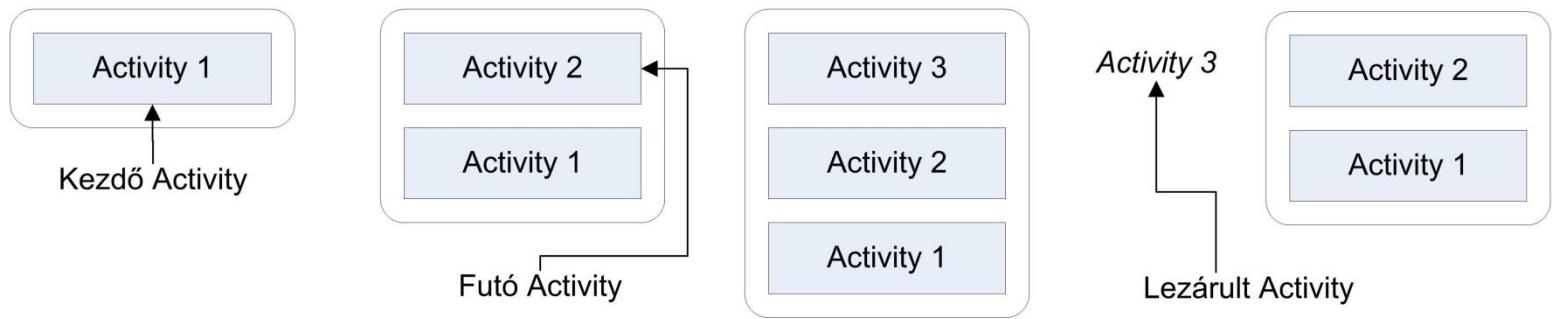
- Egy feladat végrehajtásához a felhasználó tipikusan több Activity-t használ
- A rendszer az Activity-ket egy ún. Back Stack-en tárolja
- Az előtérben levő Activity van a Back Stack tetején
- Ha a felhasználó átvált egy másik Activity-re, akkor eggyel lejjebb kerül a Stack-ben és a következő lesz legfelül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

# Activity Back Stack 2/2

Activity



Back  
Stack



# Activity vezérlés 1/2

- Legtöbb esetben az alapértelmezett Back Stack viselkedés kielégíti az igényeket
- Néha azonban szükség lehet ezen alapértelmezett viselkedés felül definiálására
- Back Stack törlése, ha a Vissza hatására minden kezdő Activity-re kell visszalépni
- Az alapértelmezett viselkedés felülírása:
  - > Manifest állományban az <activity>-be
  - > *startActivity(...)* fv. Paramétereként
- Amennyiben az alapértelmezett viselkedést módosítjuk, mindenkor teszteljük az alkalmazást navigálás és felhasználói élmény szempontjából, mert sokszor a programozó szempontjából jó megoldás nem ideális felhasználói szempontból

# Activity vezérlés 2/2

- Az `<activity>` tag attribútumai (új Activity hogyan viselkedjen a többihez képest):
  - > `taskAffinity`: melyik taskhoz tartozik
  - > `launchMode`: indítási mód (mindig új példány, stb.)
  - > `allowTaskReparenting`: új taskhoz kerül át
  - > `clearTaskOnLaunch`: minden Activity-t töröl a task-ból
  - > `alwaysRetainTaskState`: a rendszer kezelje-e a task állapotát
  - > `finishOnTaskLaunch`: le kell-e állítani az Activity-t ha a felhasználó kilép a task-ból (pl. HOME gomb)
- `startActivity(...)` függvény paraméter értékei (az új Activity hogyan viselkedjen a most futóhoz képest):
  - > `FLAG_ACTIVITY_NEW_TASK`
  - > `FLAG_ACTIVITY_CLEAR_TOP`
  - > `FLAG_ACTIVITY_SINGLE_TOP`
- További részletek az Intent előadásban

# 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)  
}
```

# Hogy is volt?

- Magyarázza el a fordítás mechanizmusát!
- Egy Android alkalmazás milyen komponensekből épülhet fel?
- Mi a Service komponens?
- Miket kell tartalmaznia a manifest állománynak?
- Az Activity callback életciklus-függvények felüldefiniálásakor meg kell-e hívni kötelezően az ōs osztály implementációját?
- Ha A Activity-ből átváltunk B Activity-re, milyen sorrendben hívódnak meg az életciklus függvények?
- Magyarázza el az Activity Back Stack működési elvét!

# Összefoglalás

- Kotlin alapok
- Android projekt felépítése, fordítás mechanizmusa
- Android alkalmazás komponensek
- Manifest állomány, jogosultságok
- Activity Back Stack
- Multitasking
- Navigálás Activity-k között

# Köszönöm a figyelmet!

The graphic features the words "THANK" and "YOU" in large, bold, black letters. Interspersed within these letters are numerous smaller, black text snippets representing "thank you" in various languages. These include:

- GRACIAS (Spanish)
- ARIGATO (Japanese)
- SHUKURIA (Arabic)
- JUSPAXAR (Burmese)
- TASHAKKUR ATU (Kyrgyz)
- YAQHANYELAY (Uzbek)
- WABEEJA MATTEKA HUI (Malay)
- YUPGARATAM (Khmer)
- TINGKI (Filipino)
- BIYAN SHUKRIA (Persian)
- THANK (repeated in large letters)
- YOU (repeated in large letters)
- BOLZİN MERCI (Turkish)
- MERASTANHIV (Lao)
- GAEUTHIO (Lao)
- GOZAIMASHITA (Japanese)
- EFCHARISTO (Greek)
- AGUYJE (Tagalog)
- FAKAUAU (Fijian)
- MAJAKE (Swahili)
- LAH (Lao)
- ATTO (Lao)
- AHUA (Lao)
- SPASIBO DENKAUA-JO NEHAACHALIYA (Mongolian)
- UNHACHESH (Lao)
- HAIUT GUE (Lao)
- EBOU SIKOMO (Lao)
- MAKETTA (Lao)
- MINMONCHAR (Lao)

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)

# Mobil- és webes szoftverek

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)

03. Előadás

Erőforrások, felhasználói felület, layout-ok, nézetek



Department of  
Automation and  
Applied Informatics

# Miről volt szó az előző órán?

- Kotlin alapok
- Fordítás mechanizmusa
- R.java
- Alkalmazás komponensek
- Activity életciklus
- Több Activity kezelése
- Manifest állomány

# Gyakorló kérdések

- Magyarázza el a fordítás mechanizmusát!
- Egy Android alkalmazás milyen komponensekből épülhet fel?
- Mi a Service komponens?
- Miket kell tartalmaznia a manifest állománynak?
- Az Activity callback életciklus-függvények felüldefiniálásakor meg kell-e hívni kötelezően az ōs osztály implementációját?
- Ha A Activity-ből átváltunk B Activity-re, milyen sorrendben hívódnak meg az életciklus függvények?
- Magyarázza el az Activity Back Stack működési elvét!

# Tartalom

- Felhasználói felület alapok
- Erőforrás típusok
- View/ViewGroup-ok
- Menü kezelés, Toolbar
- Felugró ablakok
- Stílusok&Témák
- Grafikai erőforrások

# Felhasználói felület alapfogalmak

# Legfontosabb fogalmak 1/2

- Képernyő méret (*screen size*):
  - > Fizikai képátló
  - > Az egyszerűség kedvéért az Android 4 kategóriát különböztet meg: small, normal, large, és extra large
- Képernyő sűrűség (*screen density – dpi*): A pixelek száma egy adott fizikai területen belül, tipikusan inchenkénti képpont (*dpi – dots per inch*)
  - > Az Android 6 kategóriát különböztet meg: low, medium, high és extra high, xxhigh, xxxhigh
- Orientáció (*orientation*): A képernyő orientációja a felhasználó nézőpontjából:
  - > Álló (*portrait*)
  - > Fekvő (*landscape*)
  - > Az orientáció futási időben is változhat, például a készülék eldöntésével
  - > Lehetőség van rögzíteni az orientációt

# Legfontosabb fogalmak 2/2

- Felbontás (*resolution – px*): Képernyő pixelek száma
  - > A UI tervezéskor nem felbontással dolgozunk, hanem mérettel és pixel sűrűsséggel
- Sűrűség független pixel (*density-independent pixel – dp*)
  - > Virtuális pixel egység, amit UI tervezéskor célszerű használni
  - > Egy dp egy fizikai pixelnek felel meg egy 160 dpi-s képernyőn (160 az egységes középperték)
  - > A rendszer futási időben kezel minden szükséges skálázást a definiált dp-nek megfelelően
  - >  $px = dp * (dpi / 160)$
  - > Például egy 240 dpi-s képernyőn, 1 dp 1.5 fizikai pixelnek felel meg
- Sűrűség független méretezés szövegekhez - sp

# SP vs. DP

- „A dp is a density-independent pixel that corresponds to the physical size of a pixel at 160 dpi. An sp is the same base unit, but is scaled by the user's preferred text size (it's a scale-independent pixel), so you should use this measurement unit when defining text size (but never for layout sizes).”
- Forrás:
  - <http://developer.android.com/training/multiscreen/screensanddensities.html>

# Erőforrás típusok

# Gyakran használt erőforrások

- Drawable
  - > Kép és dinamikus XML drawable
- Hangok, videok
- Felhasználói felület leíró
- Animáció
- Stílusok, téma
- Szöveges erőforrások
- Bármilyen „nyers” (raw) állomány

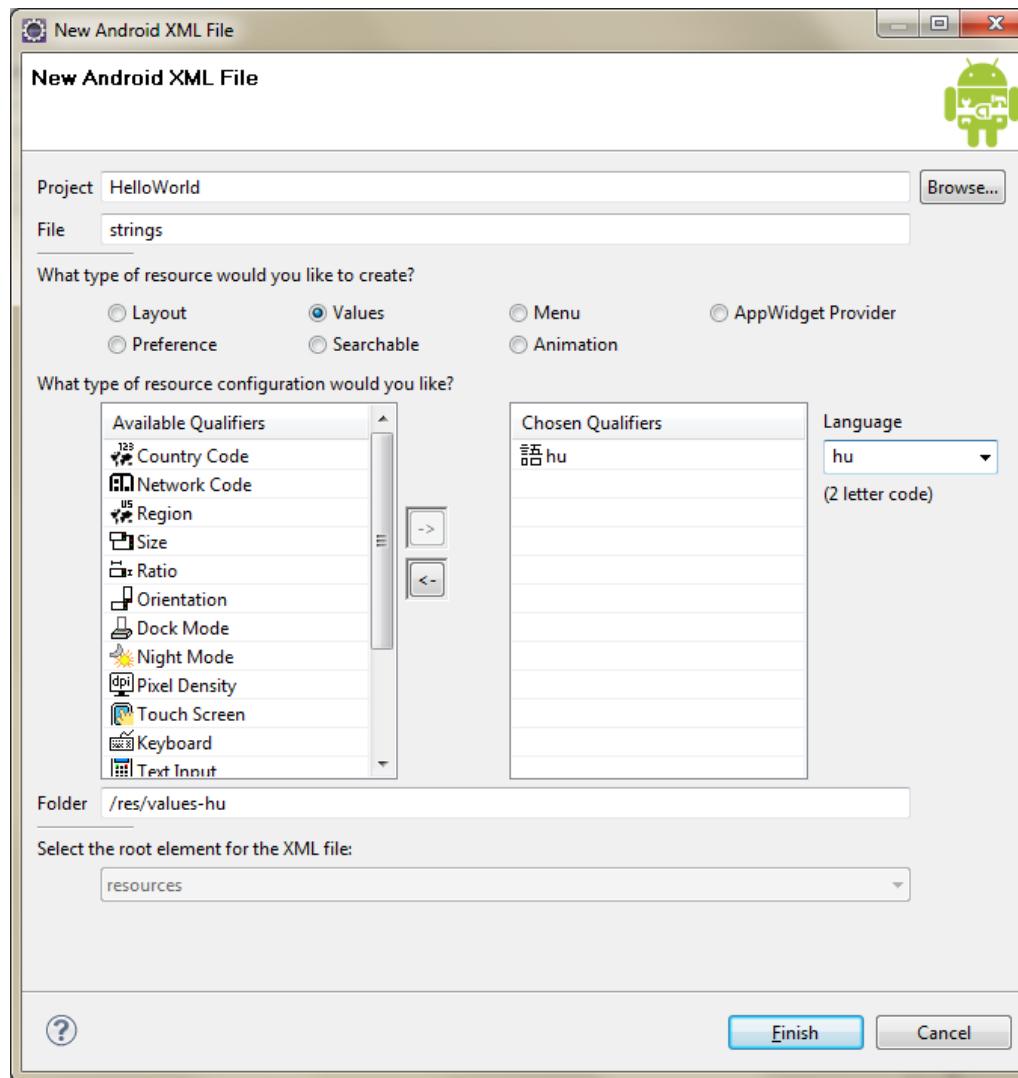
# Szöveges erőforrások

- res/values/strings.xml
- Többnyelvűség
- Paraméterezhetőség:
  - > `<string name="timeFormat">%1$d minutes ago</string>`
  - > Használat:
    - `Context.getString(R.strings.timeFormat, 14)`

# Internalizáció/Lokalizáció

- Többnyelvűség támogatása
- Nyelvfüggő felület és erőforrások
- Lokalizációt támogató erőforrás típusok:
  - > Képek
  - > Elrendezések
  - > Szöveges erőforrások
- Alapértelmezett könyvtárak: itt keres a rendszer, ha nincs a kiválasztott lokalizációnak megfelelő erőforrás
  - > *res/drawable*
  - > *res/layout*
  - > *res/value*

# Lokalizált erőforrás hozzáadása



# Futás idejű működés

- A megjelenítés optimalizálása érdekében lehetőség van alternatív erőforrások megadására a különböző méretek és sűrűségek támogatásához
- Tipikusan különböző layout-ok és eltérő felbontású képek definiálása szükséges
- A rendszer futási időben kiválasztja a megfelelő erőforrást
- Általában nincs szükség minden méret és sűrűség kombináció megadására

# Erőforrás választó algoritmus

- Futás közben egy meghatározott logika alapján választ a rendszer
- Megkeresi a passzoló erőforrást az erőforrás minősítő alapján (könyvtár utoni postfix jelölő, pl **values-hu** vagy **layout-large**)
- Ha nincs az aktuálishez passzoló, akkor egy kisebb/älacsonyabb sűrűségűt választ (pl. *large* mérethez *normal* méretet választ)
- Amennyiben az elérhető erőforrások csak nagyobb képernyőkhöz vannak, mint a készülék képernyője, akkor hibát jelez az alkalmazás
- Például ha az összes egy típusú erőforrás *xlarge*-al van megjelölve, akkor *normal* képernyős eszközökön hiba keletkezik

# Melyik állítás nem igaz?

- A. Az Android automatikusan átméretezi a képet, ha nincs megfelelően illeszkedő.
- B. Az Android támogatja a sűrűségfüggetlen megjelenítést.
- C.  $\text{px} = \text{dp} * (\text{dpi} / 160)$
- D. Közvetlenül pixelben nem adhatók meg a méretek.

# Sűrűség függetlenség

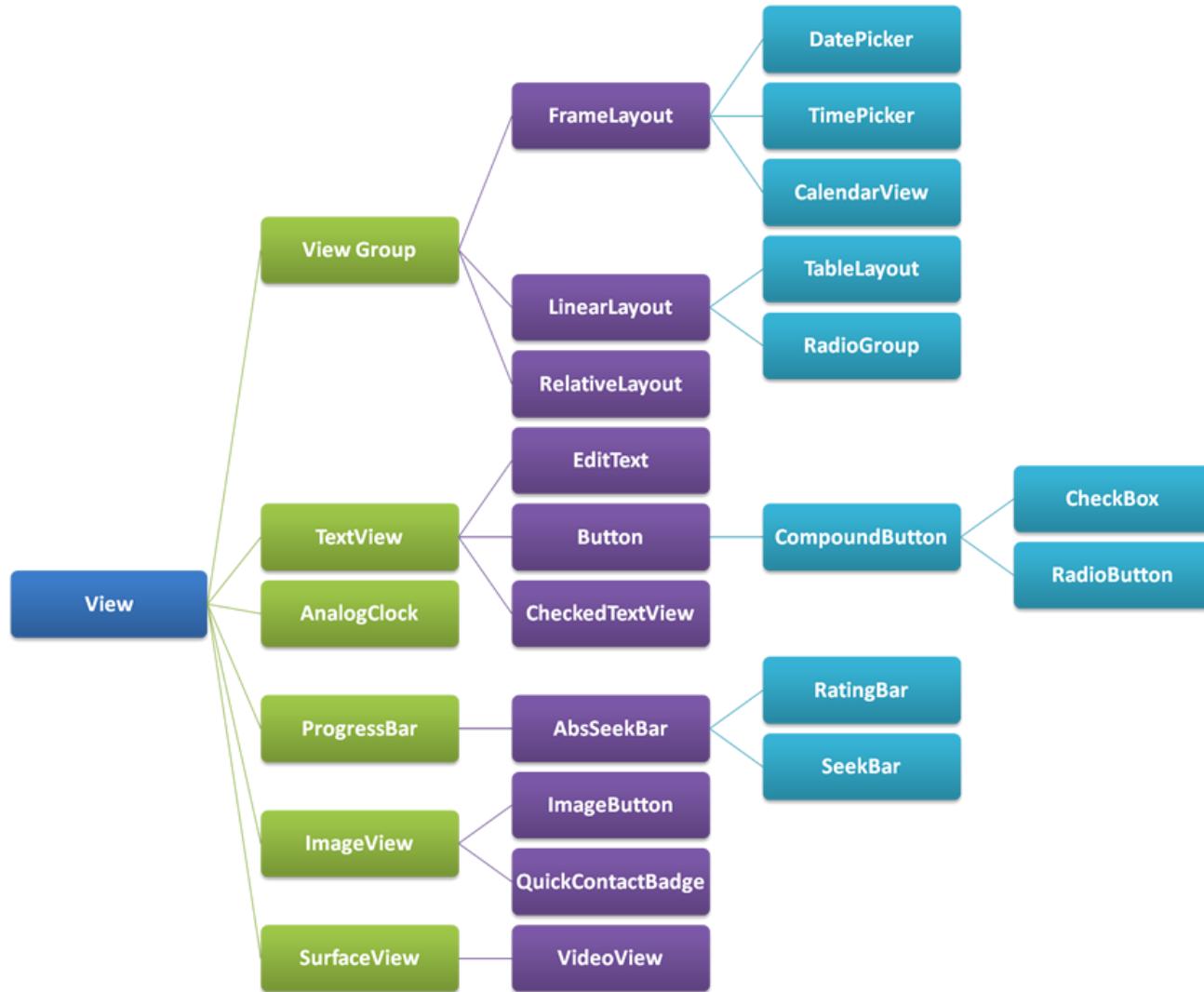
- Az alkalmazás akkor lehet „sűrűség független”, ha a felhasználói felületi elemek a felhasználó szemszögéből megőrzik a fizikai méretüket különböző sűrűségeken
- A „sűrűség függetlenség” fenntartása nagyon fontos, hiszen például egy gomb fizikailag nagyobbnak tűnhet egy alacsonyabb sűrűségű képernyőn
- A képernyő sűrűséghez kapcsolódó problémák jelentősen befolyásolhatják az alkalmazás felhasználhatóságát.
- Az Android kétféle módon is segít elérni a sűrűség függetlenséget:
  - > A rendszer a `dp` kiszámítása alapján skálázza a felhasználói felületet az aktuális képernyő sűrűségnek megfelelően
  - > A rendszer a képernyő sűrűség alapján automatikusan átskálázza a kép erőforrásokat

# Legfontosabb tényezők

- Használjuk a `wrap_content`, `match_parent`, vagy `dp` egységeket, amikor egy felületet készítünk!
- Súlyozás
- Ne használunk beégetett pixel értékeket!
- Ne használjuk az `AbsoluteLayout`-ot (elavult)!
- Mindenképp készítsünk különböző kép erőforrásokat az eltérő képernyősűrűségekhez
- Szövegek méretezéséhez érdemes használni az `sp` (scale-independent pixel) mértéket, `dp`-hez hasonlóan működik

# Felhasználói felület felépítése

# View hierarchia



# Android felhasználói felület felépítése

- minden elem a View-ból származik le
- Layout-ok (elrendezések):
  - > ViewGroup leszármazottak
  - > ViewGroup is a View-ból származik le!
- ViewGroup-ok egymásba ágyazhatók
- Saját View és ViewGroup is készíthető, illetve a meglevők is kiterjeszhetők

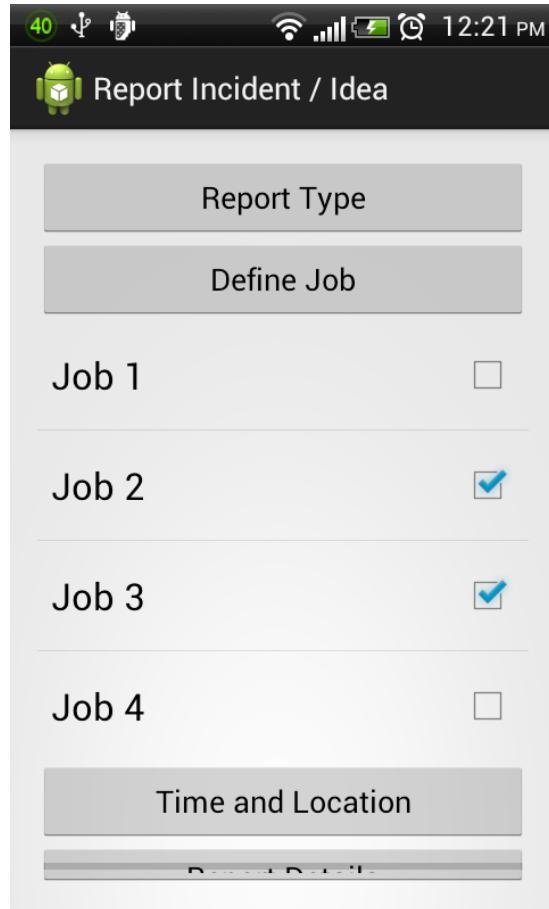
# Layout-ok (ViewGroup)

- LinearLayout
- RelativeLayout
- ConstraintLayout (~iOS AutoLayout)
- AbsoluteLayout (NEM használjuk!)
- GridLayout
- ...
- Teljes lista:
  - > <http://developer.android.com/reference/android/view/ViewGroup.html>

```
public class  
    RelativeLayout  
        extends ViewGroup  
  
java.lang.Object  
↳ android.view.View  
↳ android.view.ViewGroup  
↳ android.widget.RelativeLayout
```

# LinearLayout

- LinearLayout != Lista



# Súlyozás Layout tervezéskor

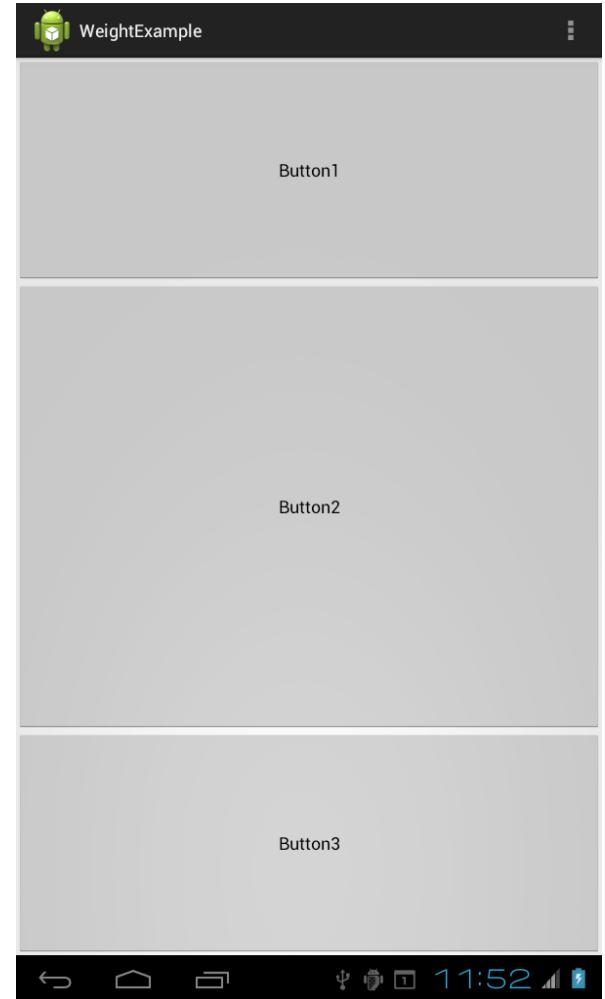
- Megadható egy layout teljes súly értéke (weightSum)
- Elemek súly értéke megadható és az alapján töltődik ki a layout
  - > layout\_weight érték
  - > A megfelelő width/height ilyenkor Odp legyen!
- Hasonló, mint HTML-ben a %-os méret megadás

# Layout súlyozás példa

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="4"
    android:orientation="vertical">

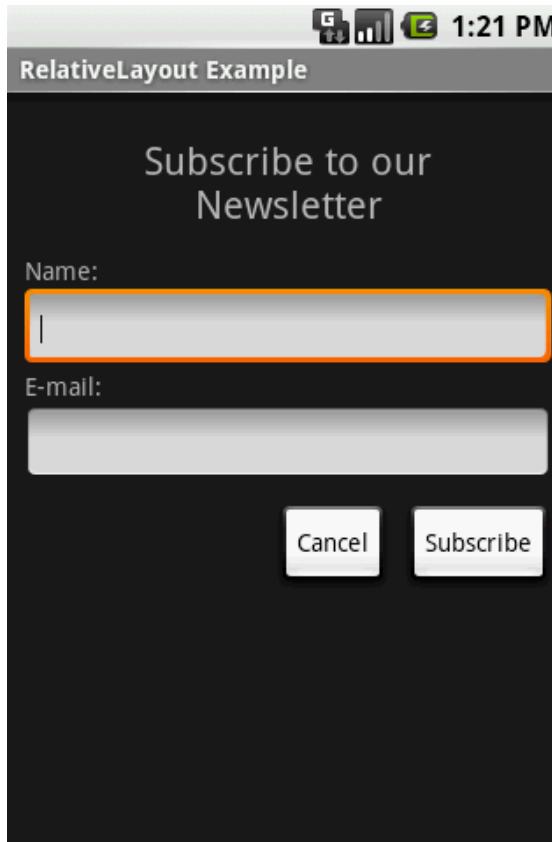
    <Button
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Button1" />
    <Button
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:text="Button2" />
    <Button
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Button3" />

</LinearLayout>
```



# RelativeLayout

- Elemek egymáshoz való viszonya definiálható
- Demo



# CONSTRAINTLAYOUT

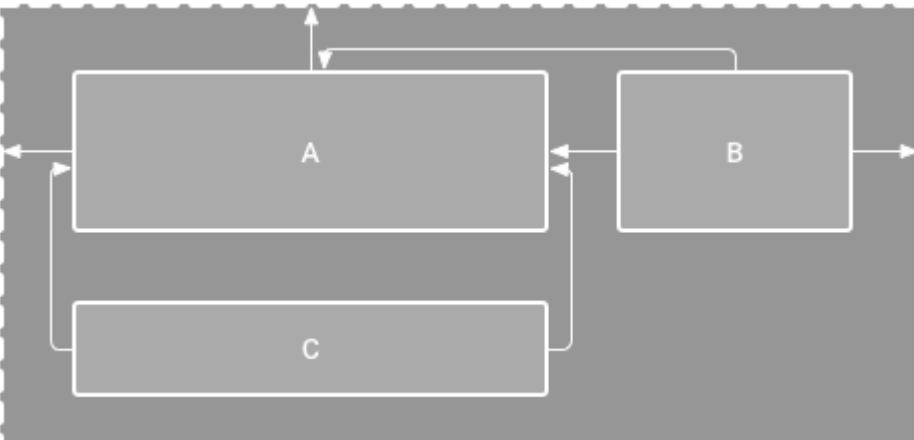
# Reszponzív felületek ConstraintLayout-al

- Összetett, komplex layout-ok flat view hierachiával
  - > Nincs szükség egymásba ágyazott layout-okra
- RelativeLayout-hoz hasonló
- Layout Editor támogatás
- Támogatás Android 2.3-tól (API Level 9)
- Komplex példák:
  - > <https://github.com/googlesamples/android-ConstraintLayoutExamples>

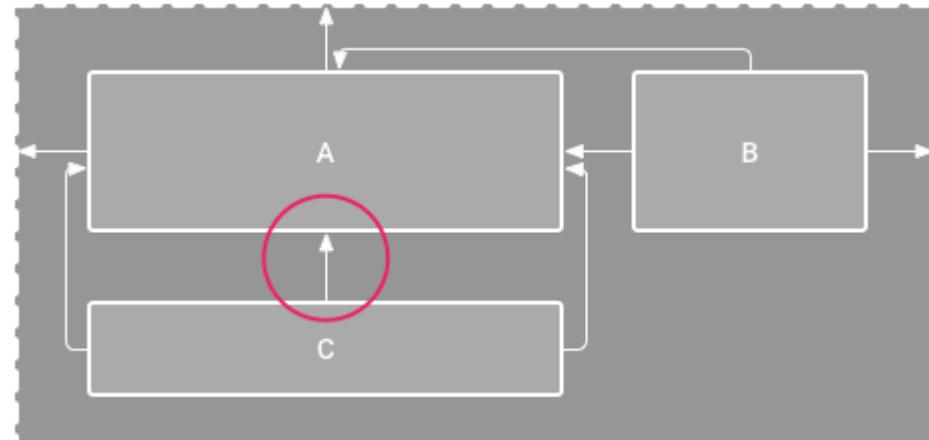
# Áttekintés

- Pozíció megadáshoz szükséges:
  - > Horizontális és vertikális „szabály” (constraint)
- minden szabály egy kapcsolat (connection)/igazítás (alignment):
  - > Egy másik view-hez képest
  - > Szülőhöz képest
  - > Egy láthatatlan sorvezetőhöz (guideline) képest
- Attól még, hogy a *LayoutEditor*-ban jól néz ki, nem biztos, hogy eszközön is jó lesz
- Android Studio jelzi a hiányzó szabályokat

Hibás:

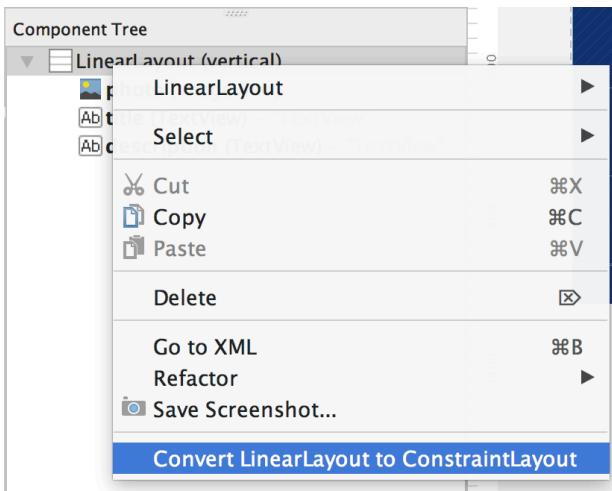


Helyes, mert C tudja, hogy A alatt van:



# ConstraintLayout eszközök

- Gradle import:
  - > compile 'com.android.support.constraint:constraint-layout:1.0.2'
- Automatikus átalakítás
  - > Nem tökéletes...



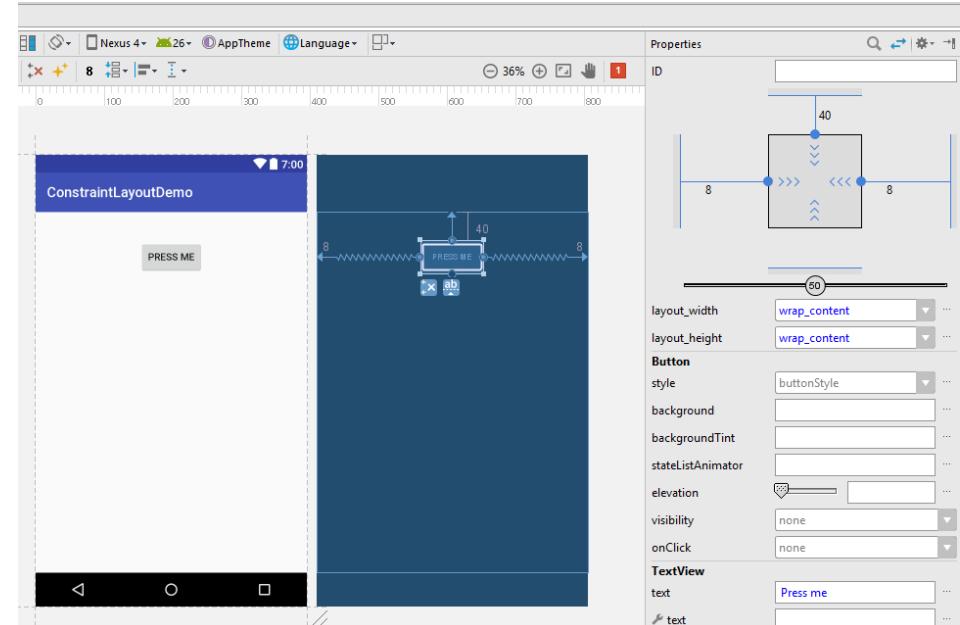
# ConstraintLayout használat

- Kötelező legalább egy horizontális és vertikális „szabály”

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Press me"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginLeft="8dp"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="8dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="40dp"
    />

</android.support.constraint.ConstraintLayout>
```



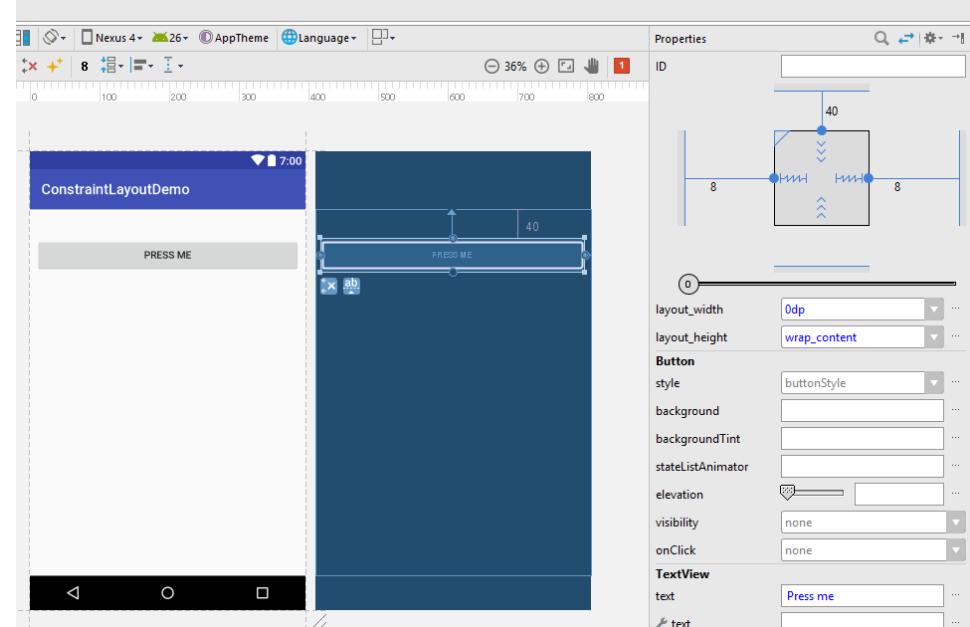
# Mekkora lesz a gomb mérete?

- Nem egyértelmű a szélesség, ellentétes szabályok, jele: 
  - > Kettő közé helyezi
- Helyette automata méretezés:
  - > `android:layout_width="0dp"`

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

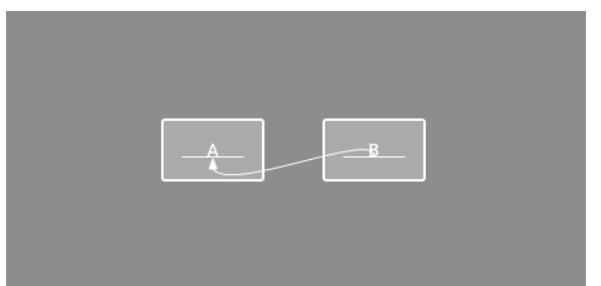
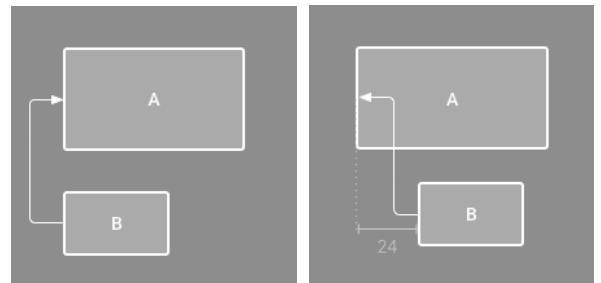
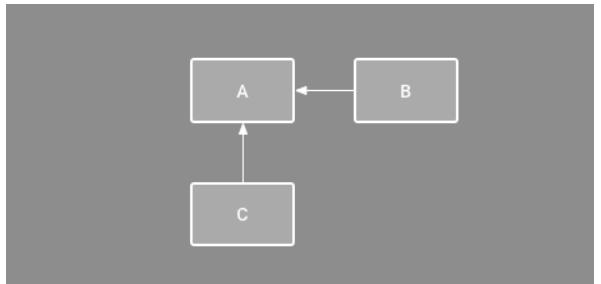
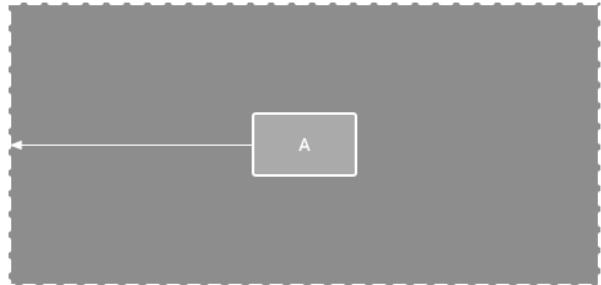
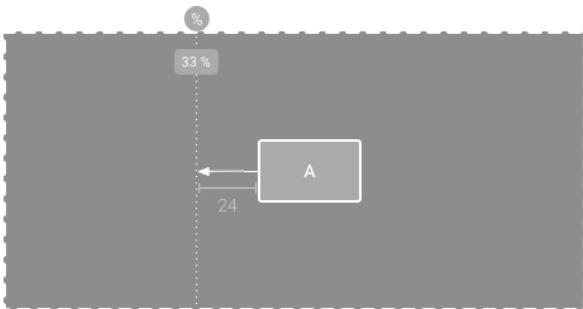
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Press me"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginLeft="8dp"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="8dp",
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="40dp"
    />

</android.support.constraint.ConstraintLayout>
```



# Constraint lehetőségek

- Szülőhöz képest
- Másik View széleihez képest
- Másik View alapvonalához képest
- Guidelinehez (láthatatlan vezetővonalhoz)



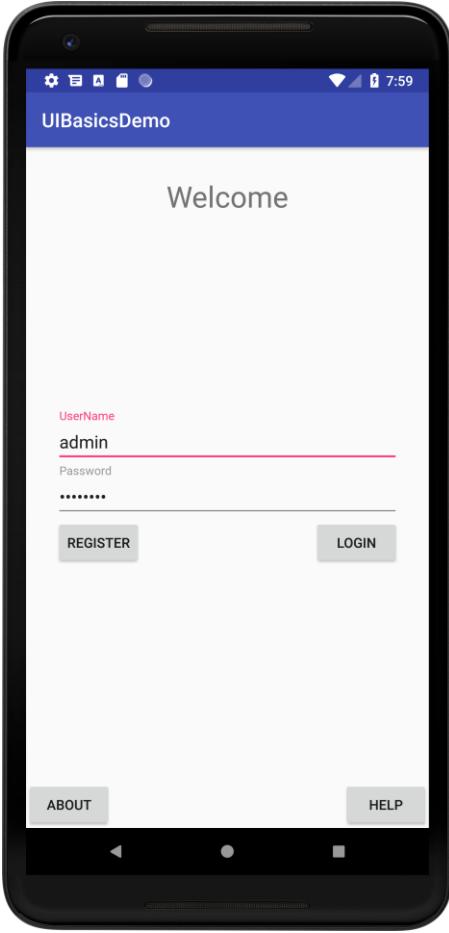
# ConstraintLayout teljesítmény

- <https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html>

# Házi feladat



- Készítsünk egy Login képernyőt



# View-k 1/2



- *Button, EditText, CheckBox, RadioButton, ToggleButton*
- *ImageButton*
- *ListView*
- *GridView*
- *Spinner*
- *AutoCompleteTextView*
- *Gallery*
- *ImageSwitcher*
- *DatePicker, TimePicker*

# View-k 2/2

Views/Controls/1. Light Theme

Save

Checkbox 1

Checkbox 2

RadioButton 1

RadioButton 2

Star

OFF

OFF

Views/ImageButton

Views/Lists/1. Array

- Abbaye de Belloc
- Abbaye du Mont des Cats
- Abertam
- Abondance
- Ackawi
- Acorn
- Adelost

Views/Grid/2. Photo Grid

Views/Spinner

Color: green

Planet: Mars

Views/Auto Complete/1. Screen Top

Type in the text field for auto-completion.

Country: CO

Give me suggestions

- Cote d'Ivoire
- Cocos (Keeling) Islands
- Colombia
- Comoros
- Congo

Views/Gallery/1. Photos



Views/Date Widgets/1. Dialog

3-12-2008 20:49

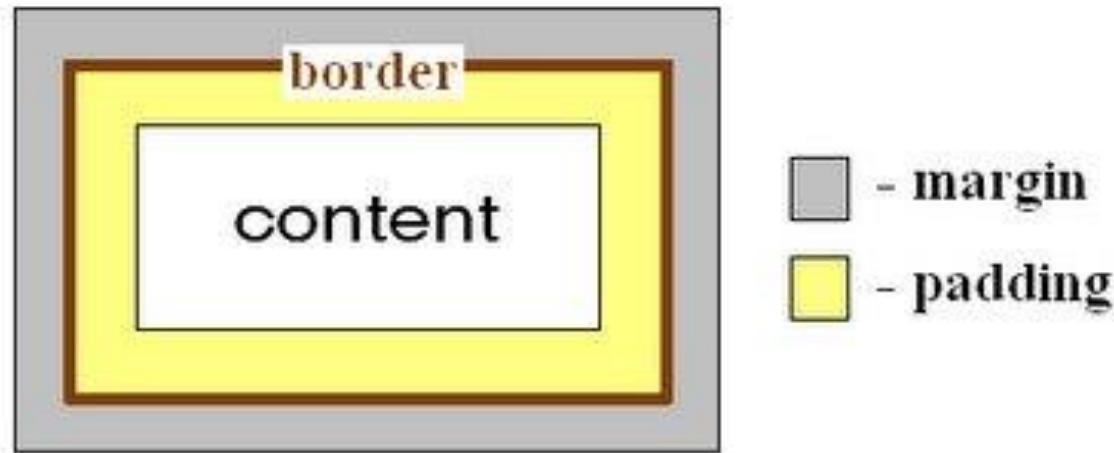
change the date

Thu, August 14, 2008

+	+	+
Aug	14	2008
-	-	-

Set      Cancel

# Padding és Margin



# DINAMIKUS UI KEZELÉS

# Dinamikus UI kezelés - LayoutInflater

- LayoutInflater feladata:
  - > XML-ben összeállított felületi elemek példányosítása
- Használati mód:

```
val myView = getLayoutInflater().inflate(  
    R.layout.activity_main, null)
```

# Validáció támogatása

- `setError(errorText)` (pl. EditText-nél)



# Menük

- *ActionBar->Toolbar* része
- Menü definiálása kódból
- Menü definiálása erőforrásból
- Dinamikus menük
  - > Láthatóság beállítása
  - > Manipuláció Java kódból
- Almenük támogatása

# Menü erőforrás

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/item1"
          android:title="@string/item1"/>

    <item android:id="@+id/item2"
          android:title="@string/item2"/>

    <item android:id="@+id_submenu"
          android:title="@string_submenu_title">

        <menu>

            <item android:id="@+id_submenu_item1"
                  android:title="@string_submenu_item1" />

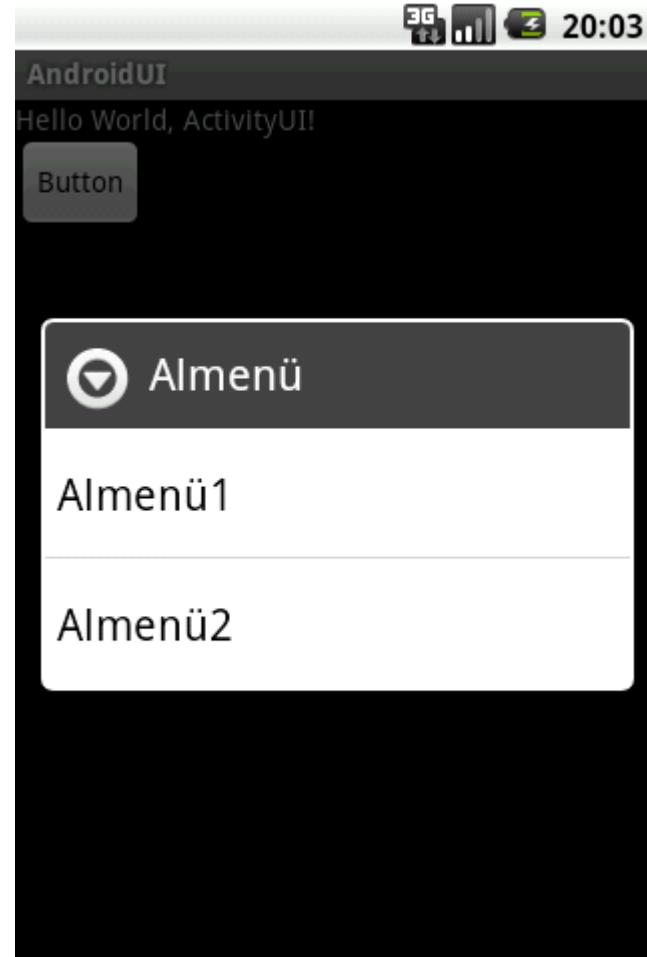
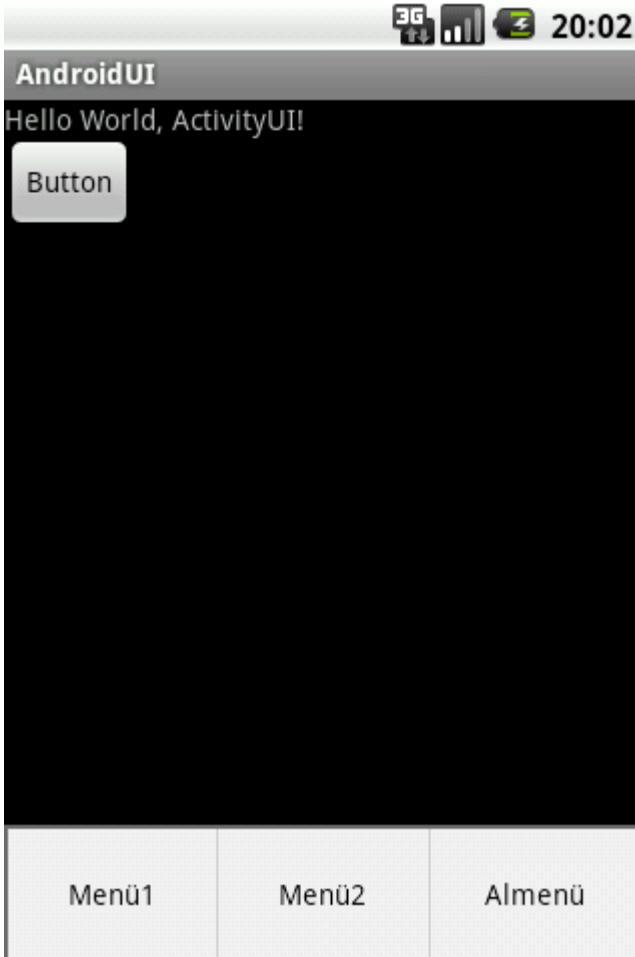
            <item android:id="@+id_submenu_item2"
                  android:title="@string_submenu_item2" />

        </menu>
    </item>
</menu>
```

# Menü kezelése

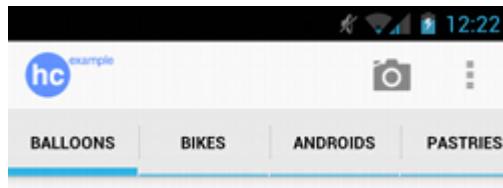
```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {  
    val inflater = menuInflater  
    inflater.inflate(R.menu.mymenu, menu)  
    return true  
}  
  
fun onOptionsItemSelected(item: MenuItem): Boolean {  
    Toast.makeText(  
        this@ActivityUI,  
        item.getTitle(), Toast.LENGTH_LONG  
    ).show()  
    return super.onOptionsItemSelected(item)  
}
```

# Menü és almenü



# ActionBar és Menük

- Dedikált alkalmazás menü, logo és cím
- Tipikus felhasználás:
  - > Menü
  - > Branding (logo/background)  
és alkalmazás ikon
  - > Konzisztens alkalmazás navigáció
  - > Fő funkciók bemutatása



## ActionBar specifikus XML menü paraméterek

```
<item android:id="@+id/action_time"  
      android:title="@string/action_show_time"  
      android:orderInCategory="5"  
      android:icon="@drawable/clock_icon"  
      android:showAsAction="always|withText" />
```

# ActionBar -> Toolbar

- ActionBar helyett ToolBar
- Sokkal dinamikusabb viselkedés
- Menü erőforrások támogatása
- Custom elemek támogatása
- Manuális pozicionálás
- Toolbar tutorialok:
  - > <http://javatechig.com/android/android-lollipop-toolbar-example>
  - > <http://www.101apps.co.za/index.php/articles/using-toolbars-in-your-apps.html>

# Toolbar használat

- ActionBar nélküli téma(styles.xml):

- > Theme.AppCompat.NoActionBar

- Layout erőforrás:

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:minHeight="?attr/actionBarSize"  
    android:background="#2196F3"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
</android.support.v7.widget.Toolbar>
```

- Activity onCreate(...):

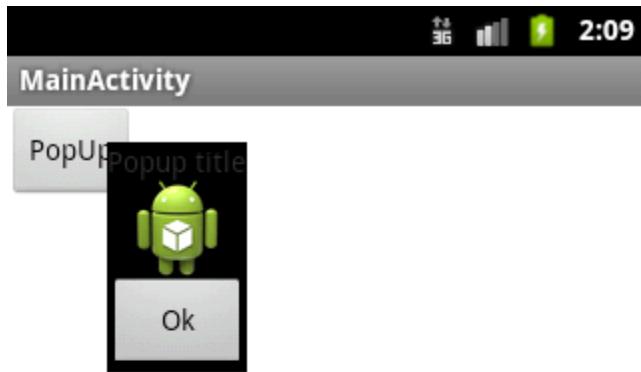
```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    // Set a toolbar to replace the action bar.  
    val toolbar: Toolbar = findViewById<View>(R.id.toolbar) as Toolbar  
    setSupportActionBar(toolbar)  
}
```

# FELUGRÓ ABLAKOK

# Felugró ablakok

- Activity felugró ablakban
  - > Manifest állományba:  
`android:theme="@android:style/Theme.Dialog"`
- *PopupWindow*
  - > PopupWindow objektum létrehozása és layout beállítása
  - > Eseménykezelő a gombokra
- *AlertDialog*
  - > *AlertDialog.Builder*
- *Toast*
  - > Szöveg megjelenítése egy kis ablakban
- *SnackBar*

# PopUpWindow példa



- Tetszőleges XML layout-al inicializálható

# PopUp megjelenítése

```
fun showPopUp() {  
    val layoutInflater = baseContext.getSystemService(LAYOUT_INFLATER_SERVICE) as LayoutInflater  
    val popupView: View = layoutInflater.inflate(  
        R.layout.popuptest, null  
    )  
    val popupWindow = PopupWindow(  
        popupView,  
        LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT  
    )  
  
    val btnDismiss: Button = popupView.findViewById<View>(R.id.btnPopUpOk) as Button  
    btnDismiss.setOnClickListener(object : OnClickListener() {  
        fun onClick(v: View?) {  
            popupWindow.dismiss()  
        }  
    })  
  
    popupWindow.showAsDropDown(btnShow, 50, -30)  
}
```

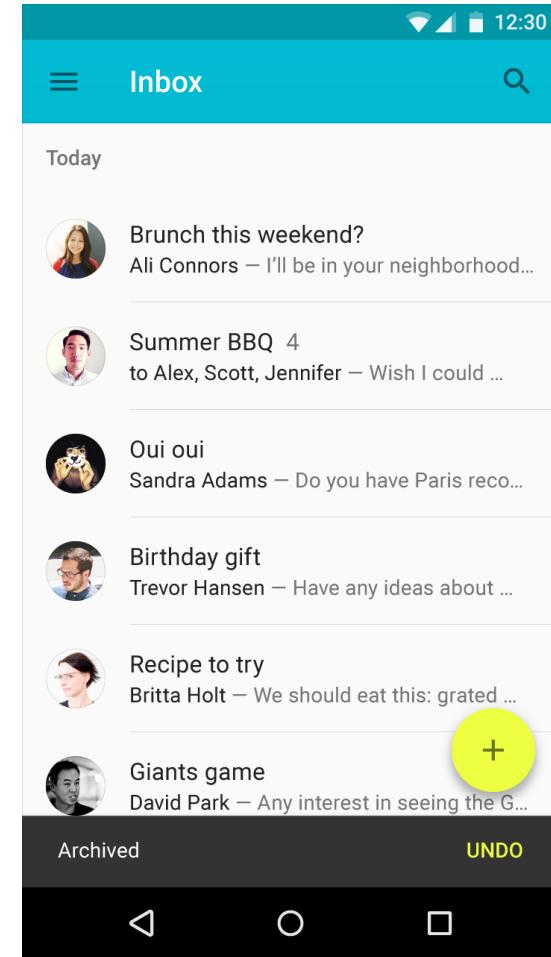
# AlertDialog példa

```
private fun showAlertMessage(aMessage: String) {  
    val alertbox: AlertDialog.Builder = Builder(this)  
    alertbox.setMessage(aMessage)  
    alertbox.setNeutralButton("Ok",  
        DialogInterface.OnClickListener { arg0, arg1 ->  
            // Ok kiválasztva  
        })  
    alertbox.show()  
}
```

# SnackBar

- Képernyő alján megjelenő információs sáv
  - > Toast helyett gyakran hasznos
  - > Egyedi akció

```
Snackbar.make(android.R.id.content,  
    "Demo message", Snackbar.LENGTH_LONG)  
    .setAction("Undo", mOnClickListener)  
    .setActionTextColor(Color.RED).show()
```



# STÍLUSOK, TÉMÁK

# Stílusok készítése

- Stílus file: res/values/styles.xml

```
<style name="ExampleStyle">  
    <item name="android:textSize">22sp</item>  
    <item name="android:textColor">#0000EE</item>  
</style>
```

- Stílus használata:

```
<TextView  
    android:id="@+id/tvHello"  
    android:text="@string/hello_world"  
    style="@style/ExampleStyle" />
```

# Saját témák használata

- Stílusokhoz hasonlóan definiálhatók:

```
<style name="CustomTheme" parent="android:Theme">  
    <item name="android:windowTitleSize">50dip</item>  
    <item name="android:textColor">#000000</item>  
    <item name=  
        "android:windowBackground">@color/white_color</item>  
</style>
```

- A témák öröklődhetnek egymásból
- Manifestben állítható be:

```
<activity android:label="" android:name=".MainActivity"  
        android:screenOrientation="portrait"  
        android:theme="@style/CustomTheme"/>
```

# GRAFIKAI ERŐFORRÁSOK XML-BEN

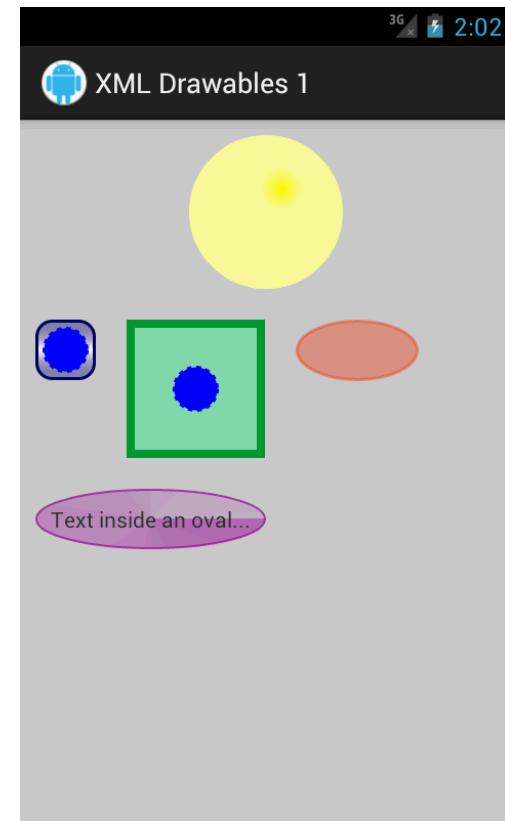
# Grafikai erőforrások 1/2

- Grafikai elemek, hátterek is leírhatók XML-ben

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <gradient
        android:type="radial"
        android:gradientRadius="20"
        android:centerX=".6"
        android:centerY=".35"
        android:startColor="#FFFF00"
        android:endColor="#FFFF99" />

    <size
        android:width="100dp"
        android:height="100dp"/>
</shape>
```



# Grafikai erőforrások 2/2

- Zöld szaggatott vonal:

```
<?xml version="1.0" encoding="utf-8"?>

<shape xmlns:android=
"http://schemas.android.com/apk/res/android"

    android:shape="line">

    <stroke

        android:width="2dp"

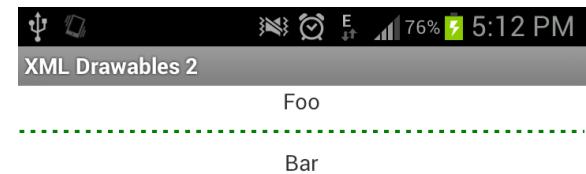
        android:color="#008000"

        android:dashWidth="3dp"

        android:dashGap="4dp"/>

    <size android:height="20dp" />

</shape>
```



# Állapotok leírása grafikai erőforrásokban

- Például gombfelirat színe:

- > res/color/button\_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>  
<selector xmlns:android="http://schemas.android.com/apk/res/android">  
    <item android:state_pressed="true"  
        android:color="#ffff0000"/> <!-- pressed -->  
    <item android:state_focused="true"  
        android:color="#ff0000ff"/> <!-- focused -->  
    <item android:color="#ff000000"/> <!-- default -->  
</selector>
```

- Használat:

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    android:textColor="@color/button_text" />
```

# Összefoglalás

- Felhasználói felület alapok
- Erőforrás típusok
- View/ViewGroup-ok
- Menü kezelés, Toolbar
- Felugró ablakok
- Stílusok&Témák
- Grafikai erőforrások

# Köszönöm a figyelmet!



*peter.ekler@aut.bme.hu*

# Mobil- és webes szoftverek

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)



Department of  
Automation and  
Applied Informatics

# Miről volt szó az előző órán?

- Felhasználói felület alapok
- Erőforrás típusok
- View/ViewGroup-ok
- Menü kezelés, Toolbar
- Felugró ablakok
- Stílusok&Témák
- Grafikai erőforrások

# Rövid áttekintés

- Mobil piac helyzete
- Android platform szerkezete
- Fejlesztőkörnyezet, projekt felépítése
- Alkalmazás komponensek
- Activity életciklus
- Activity BackStack
- Erőforrások
- Felhasználói felülete alapok
- Menük
- Grafikai erőforrások

# Mi jön még? ☺

- Animációk
- Listák kezelése
- Adatbázis kezelés
- Hálózati kommunikáció
- Alkalmazások közti kommunikáció
- ...



# Kvíz!

QUIZ  
TIME!

<https://kahoot.it/>



# Tartalom

- Animációk
- Fragmentek
- Listakezelés
  - > RecyclerView
  - > Új elem hozzáadása
  - > Törlés
  - > Módosítás

# ANIMÁCIÓK

# Animációk

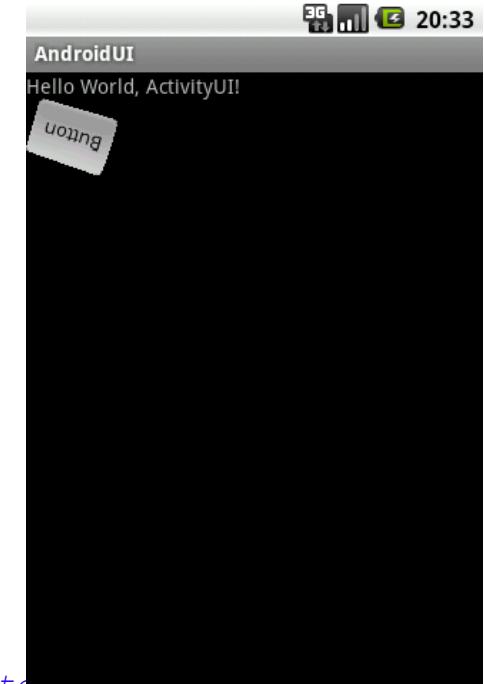
- Animációk támogatása
  - > XML erőforrás (*res/anim*)
  - > Programkód
- Layout animáció
  - > *Scale*
  - > *Rotate*
  - > *Translate*
  - > *Alpha*
- Hárrom fő típus:
  - > Tween animáció
  - > Frame animáció
  - > Property animator

# Tween animáció erőforrás

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
        android:interpolator=
            "@android:anim/accelerate_interpolator"
        android:fromXScale="0.0"
        android:toXScale="1.0"
        android:fromYScale="0.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="1000" />

    <alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:duration="5000"/>

    <rotate
        android:interpolator="@android:anim/accelerate_interpolator"
        android:fromDegrees="0.0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
</set>
```



# Animáció lejátszása

```
class MainActivity : AppCompatActivity() {

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

        var demoAnim = AnimationUtils.loadAnimation(this,
            R.anim.demo_anim)

        btnAnim.setOnClickListener {
            btnAnim.startAnimation(demoAnim)
        }
    }
}
```

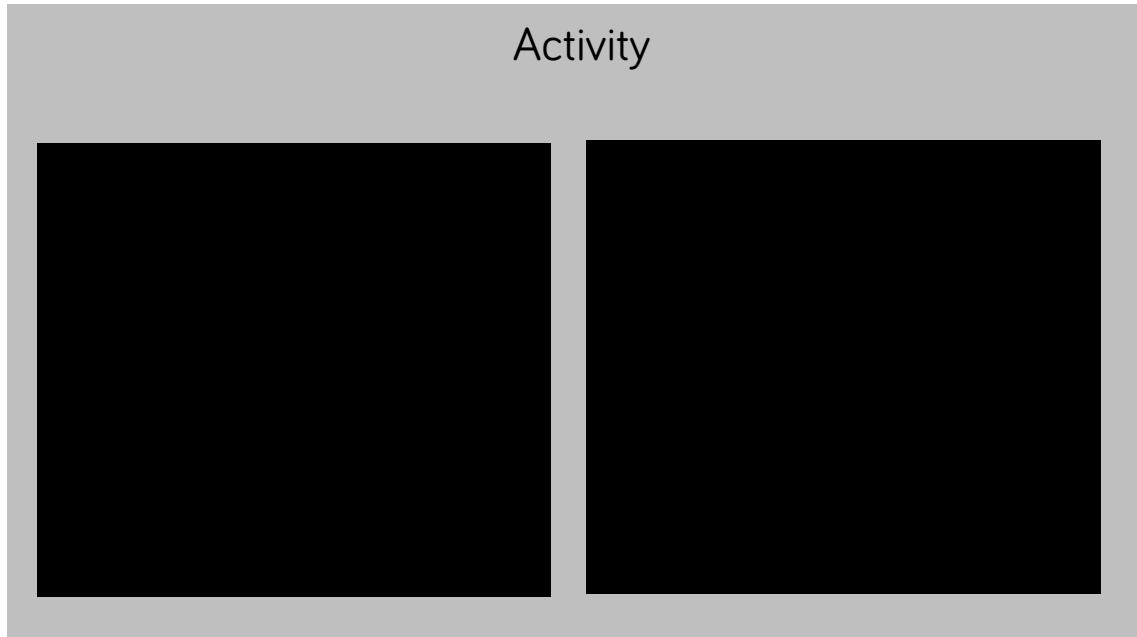
# FRAGMENT-EK

# Fragment API

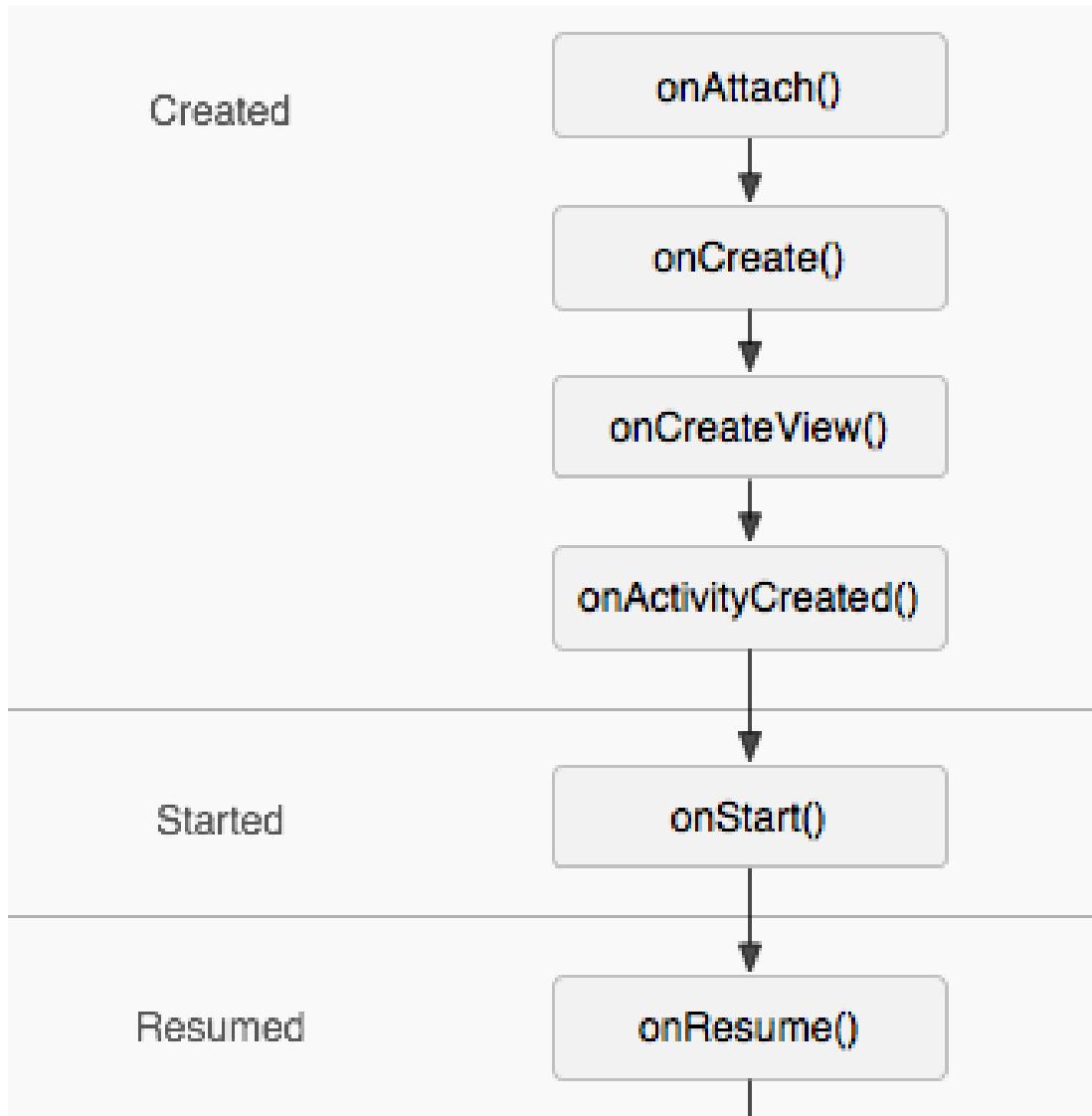
- Mik azok a Fragment-ek?
  - > Elsősorban: A képernyő egy nagyobb részéért felelős objektumok
  - > Továbbá: A háttérben munkát végző objektumok
- Miért kellenek nekünk?
  - > Nagy képernyőményet = több funkció egy képernyőn = bonyolultabb Activity-k
  - > Fragment-ekkel modulárisabb, rugalmasabb architektúra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

# Fragment és Activity

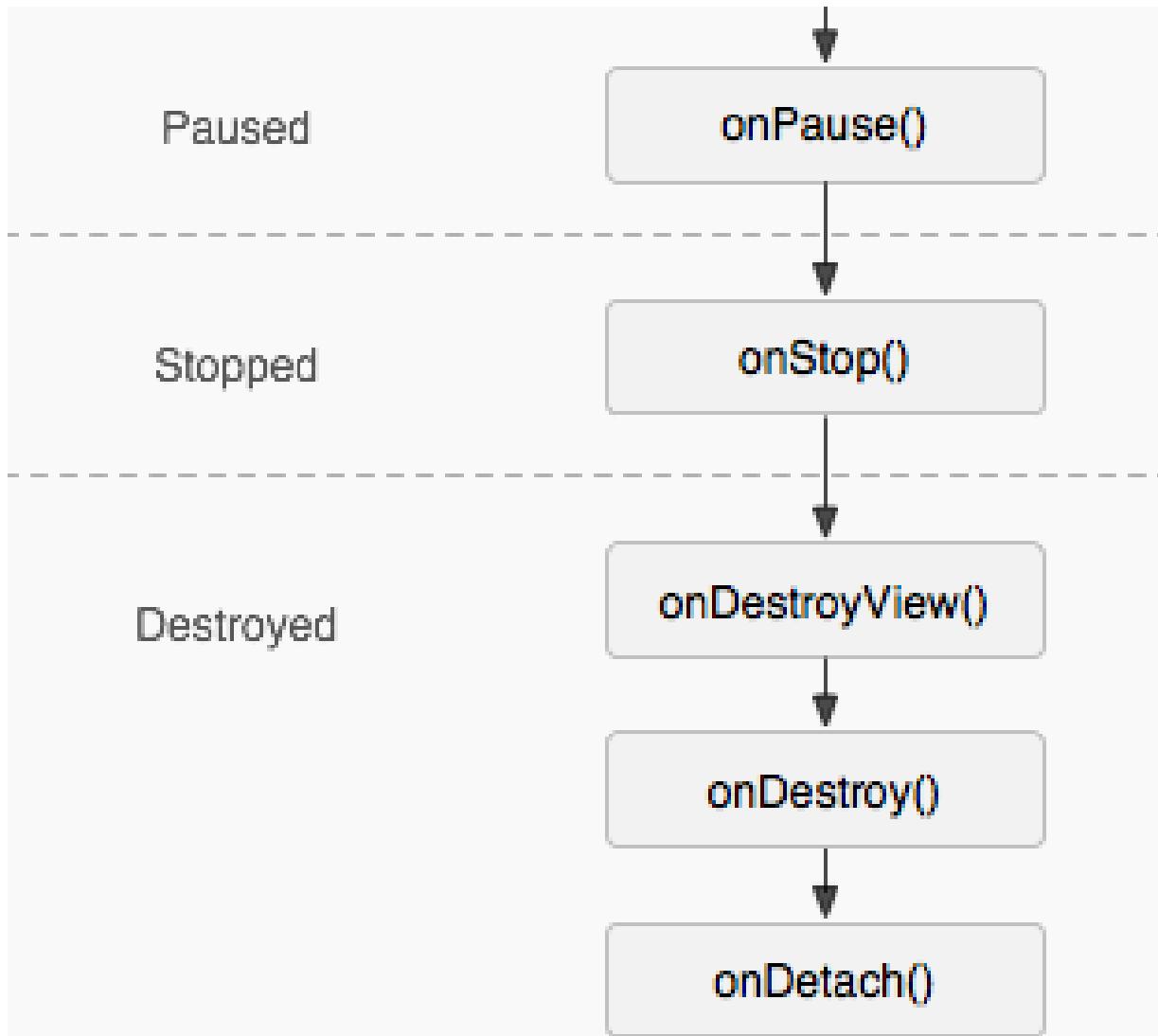
- Egy Fragment mindig egy Activity-hez csatoltan jelenik meg
- Az életciklusaiak nagyrészt megegyeznek

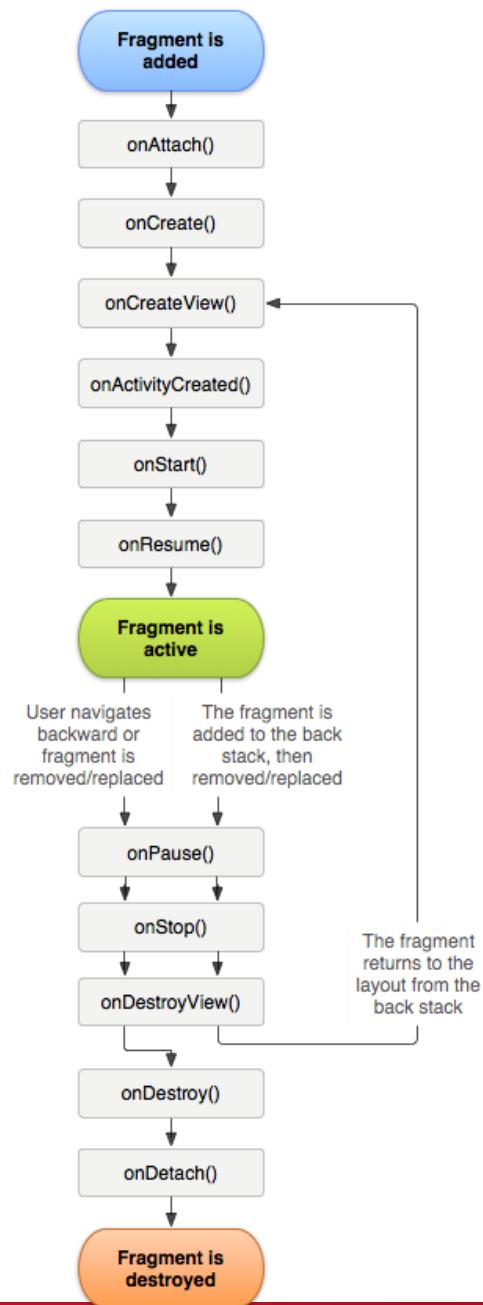


# Fragment életciklus I.

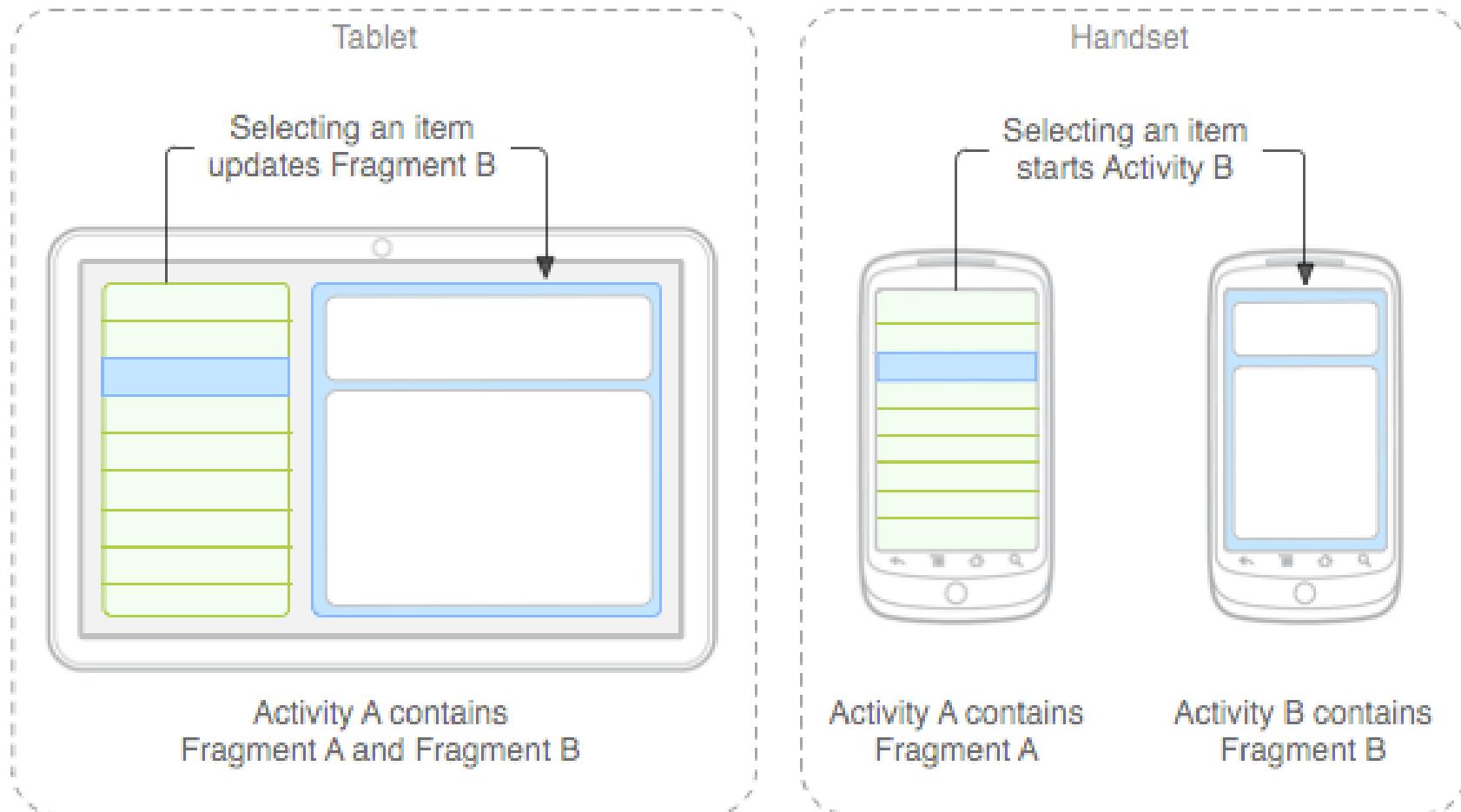


# Fragment életciklus II.





# Eltérő képernyőméretek



# 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: .getFragmentManager()
  - > FragmentTransaction indítása
  - > Aktív Fragment-ek között keres
    - Tag alapján
    - ID alapján
  - > Fragment-stack-et menedzseli

# FragmentTransaction osztály I.

- Ezen keresztül módosíthatók az aktív Fragmentek
- 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.

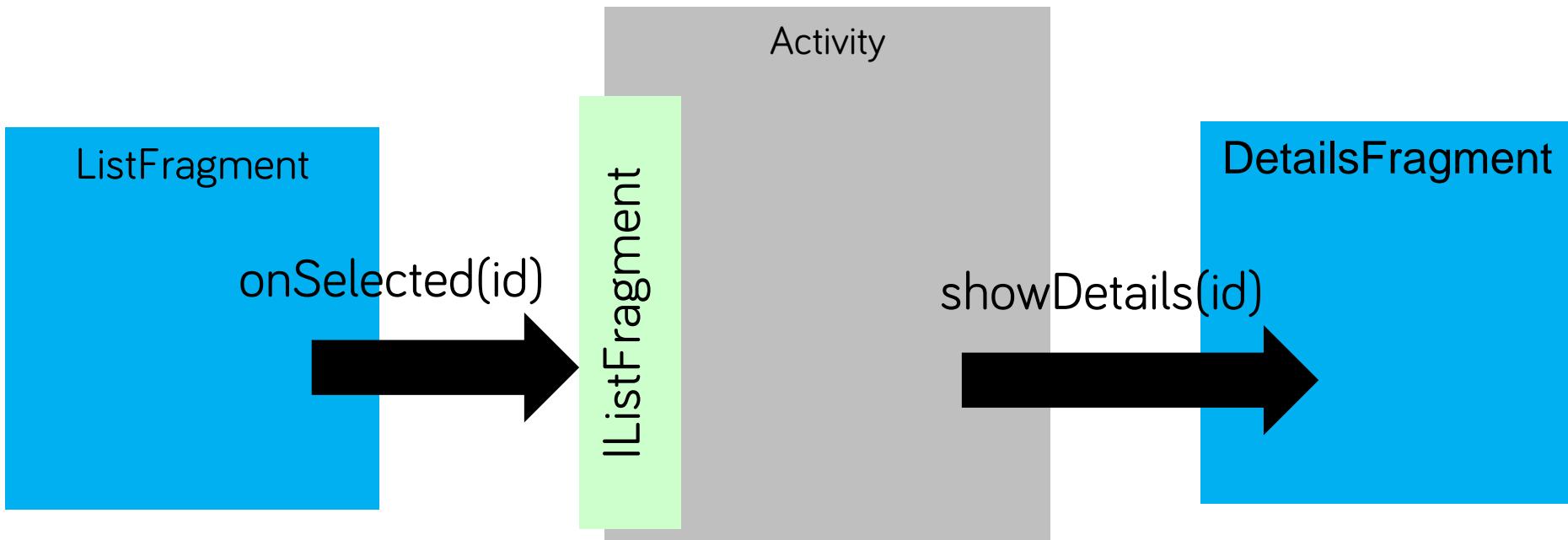
- Fragment hozzáadása, a tranzakciót a backstack-re téve:

```
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_bottom)
ft.addToBackStack(null)
ft.commit()
```

# Fragment kommunikáció

- Egy Fragment-nek egységbezártnak kell lennie  
=> közvetett kommunikáció
  - > Az Activity közvetít



# DialogFragment I.

- Egy Fragment dialógusként is megjelenhet
  - > Dialógus egyedi layout-al
  - > Az AlertDialog.Builder továbbra is használható
- Így egy dialógus is ugyanolyan életciklussal rendelkezik, mint egy Fragment
- A FragmentDialog-ok is rákerülhetnek a BackStack-re

# DialogFragment II.

- `.onCreateDialog()`
  - > Ez a metódus is visszatérhet a megjelenítendő Dialog-al
- `.onCreateView()`
  - > Ha nem használjuk az `.onCreateDialog()`-ot
  - > Tetszőleges megjeleníthető tartalom
- Egy DialogFragment egyben Fragment is!
  - > Ha kell, akár Activity-be ágyazottan is megjeleníthető

# Mi nem igaz a Fragment-ekre?

- A. Saját életciklus függvényekkel rendelkeznek.
- B. Dialógusként nem jeleníthetők meg.
- C. Dinamikusan és statikusan is csatolhatók.
- D. A tableték felhasználói felületének kialakításakor különösen hasznosak.

# LISTÁK KEZELÉSE

# Listák kezelése – régi mód

- `ListActivity` / `ListFragment`
- Tartalmaz egy `ListView`-t, melyet a `getListView()` függvényel kérhetünk el
- Beállítható egy `ListAdapter`, mely a lista elemeinek tárolásáért és megjelenítéséért felelős
  - > Tartalmazza az elemeket (objektumok listája)
  - > Felelős új elem hozzáadásáért, eléréséért és törléséért
  - > A `getView(int position, View convertView, ViewGroup parent)` függvény felüldefiniálásával megadhatjuk, hogy a lista egy sorában egy elem hogyan renderelődjön
  - > A `convertView` egy korábbi lista sor (nem tudjuk biztosan melyik), ami felhasználható az új sor létrehozására

# ListAdapter példa – régi mód

```
public class MyContactsAdapter extends ArrayAdapter {
    private List myContacts;
    private Context context;

    public ContactsArrayAdapter(Context context, int textViewResourceId,
        List contacts) {
        super(context, textViewResourceId, data);
        this.context = context;
        this.myContacts = contacts;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;
        if(v == null) {
            LayoutInflator inflater = LayoutInflater.from(mContext);
            v = inflater.inflate(R.layout.layout_list_item, parent);
        }
        TextView txtName = (TextView)v.findViewById(R.id.txtName);
        TextView txtMail = (TextView)v.findViewById(R.id.txtMail);
        MyContact entry = myContacts.get(position);
        txtName.setText(entry.getName());
        txtMail.setText(entry.getMail());
        return v;
    }
}
```

# getView(...) használata – egyszerű eset

```
@Override  
  
public View getView(int position, View convertView, ViewGroup parent) {  
  
    View v = convertView;  
  
    if(v == null) {  
  
        LayoutInflator inflater = LayoutInflator.from(mContext);  
  
        v = inflater.inflate(R.layout.layout_list_item, parent);  
  
    }  
  
    TextView txtName = (TextView)v.findViewById(R.id.txtName);  
    TextView txtMail = (TextView)v.findViewById(R.id.txtMail);  
  
    Contact entry = mList.get(position);  
  
    txtName.setText(entry.getName());  
    txtMail.setText(entry.getMail());  
  
    return v;  
}
```

# getView(...) használata – ViewHolder-el (gyors!)

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View v = convertView;
    if(v == null) {
        LayoutInflator inflater = LayoutInflator.from(mContext);
        v = inflater.inflate(R.layout.layout_list_item, parent);
        ViewHolder holder = new ViewHolder();
        holder.txtName = (TextView)v.findViewById(R.id.txtName);
        holder.txtMail = (TextView)v.findViewById(R.id.txtMail);
        v.setTag(holder);
    }

    Contact entry = mList.get(position);
    if(entry != null) {
        ViewHolder holder = (ViewHolder)v.getTag();
        holder.txtName.setText(entry.getName());
        holder.txtMail.setText(entry.getMail());
    }
    return v;
}

static class ViewHolder {
    TextView txtName;
    TextView txtMail;
}
```

# ViewHolder pattern előnyei

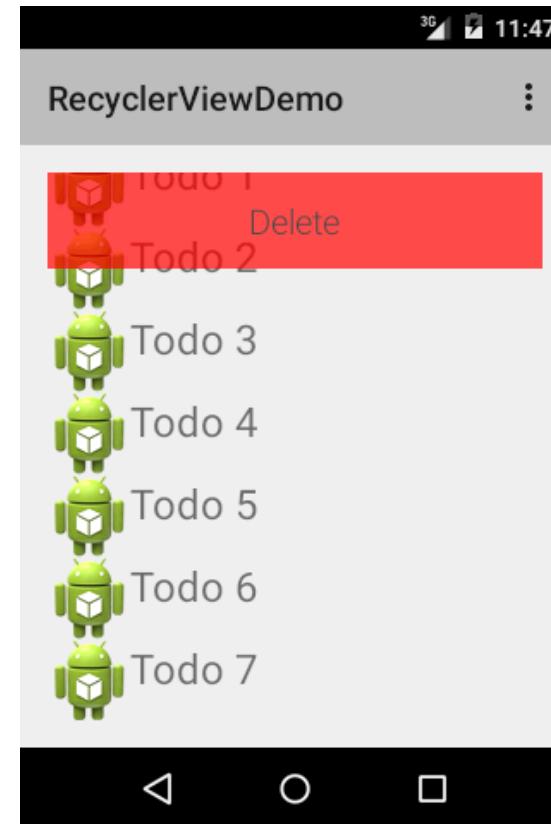
- Statikus *ViewHolder* objektum, cache támogatás
- Nincs folyamatos *findViewById(...)* hívás
- Gyors működés

# RecyclerView

- A *ListView* fejlett és flexibilis változata
- *ViewHolder* minta kikényszerítése
- Hatékony elem újrafelhasználás
- Fejlesztés:
  - > *ListView*, és *Adapter* mellett egy *LayoutManager*-t is kell készíteni
- A *LayoutManager* feladata a *findViewById(...)* felesleges sokszori hívása

# RecyclerView

- Fejlett ListView
- ListView helyett javasolt
- Flexibilis
- *ViewHolder* pattern használata alapértelmezetten
- *Gradle dependency:*
  - > compile 'com.android.support:recyclerview-v7:23.0.1'



# RecyclerView.Adapter<ViewHolder> 1/3

- Inicializálás, konstruktor

```
private val context: Context
private val items: MutableList<ShoppingItem> = mutableListOf<ShoppingItem>(
    ShoppingItem("milk", 200, false),
    ShoppingItem("car", 4000, false),
    ShoppingItem("beer", 1, false)
)

constructor(context: Context) : super() {
    this.context = context
}
```

- Egy sor nézetének beállítása: onCreateViewHolder

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val view = LayoutInflater.from(parent.context).inflate(
        R.layout.row_item, parent, false
    )
    return ViewHolder(view)
}
```

# ViewHolder implementáció

```
class ViewHolder(itemView: View?) : RecyclerView.ViewHolder(itemView) {  
    val tvName = itemView.tvName  
    val tvPrice = itemView.tvPrice  
    val cbBought = itemView.cbBought  
    val btnEdit = itemView.btnEdit  
}
```

# RecyclerView.Adapter<ViewHolder> 2/3

- Sorban levő elemek értékeinek beállítása
- Eseménykezelők beállítása
- ViewHolder binding

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
    val (name, price, bought) = items[holder.adapterPosition]  
    holder.tvName.text = name  
    holder.tvPrice.text = price.toString()  
    holder.cbBought.isChecked = bought  
  
    holder.btnEdit.setOnClickListener{  
        (context as MainActivity).showEditTodoDialog(items[holder.adapterPosition])  
    }  
}
```

# RecyclerView.Adapter<ViewHolder> 3/3

- Elemek száma, hozzáadás, törlés

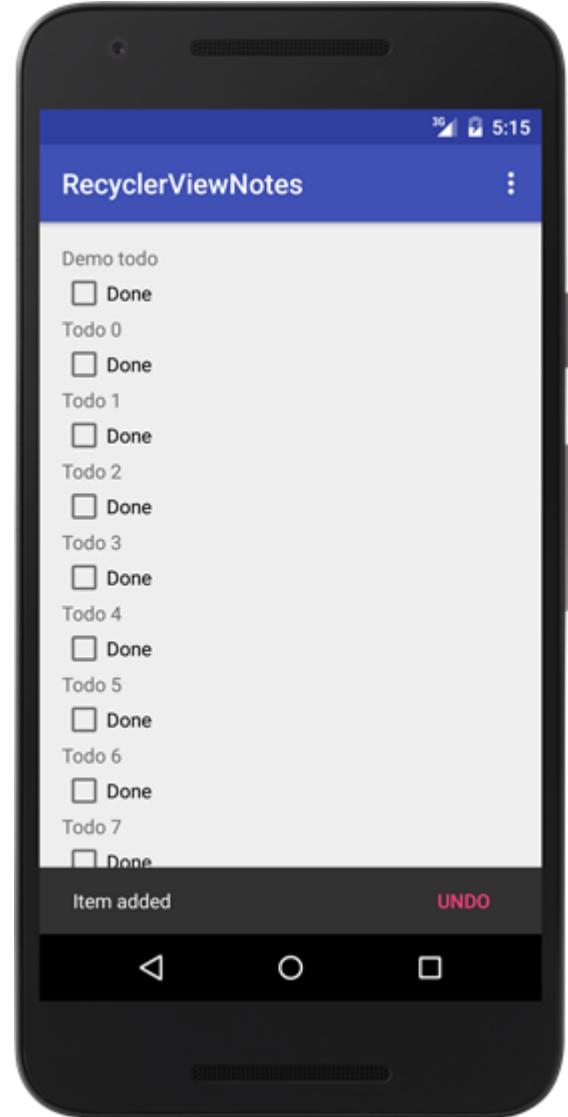
```
override fun getItemCount() = items.size

fun addItem(item: ShoppingItem) {
    items += item
    notifyItemInserted(items.lastIndex)
}

private fun deleteItemBasedOnPosition(position: Int) {
    items.removeAt(position)
    notifyItemRemoved(position)
}
```

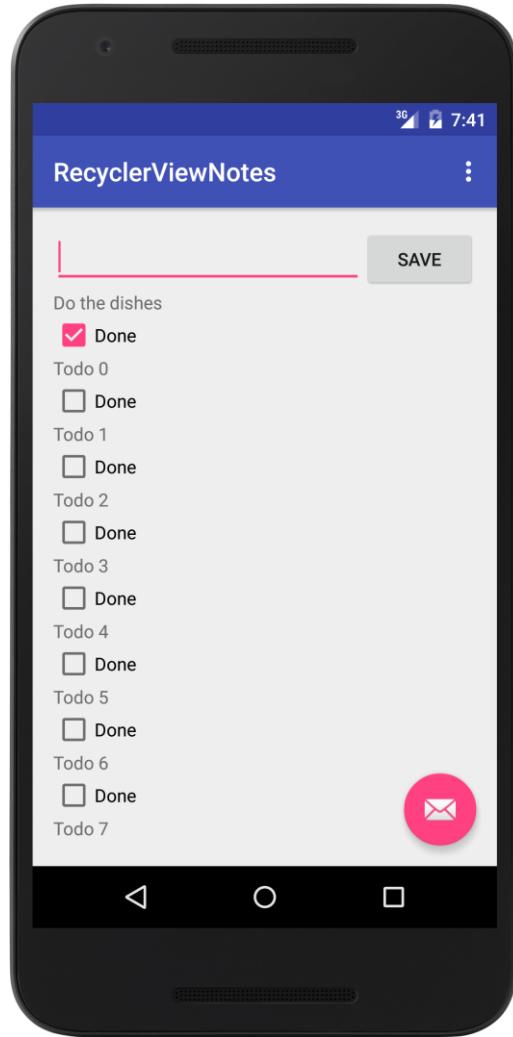
# Gyakoroljunk!

- Készítsünk egy Todo alkalmazást RecyclerView-val
- Jelenítsünk meg egy checkbox-ot minden elem előtt
- Valósítsunk meg hozzáadás után egy undo lehetőséget



# Gyakoroljunk!

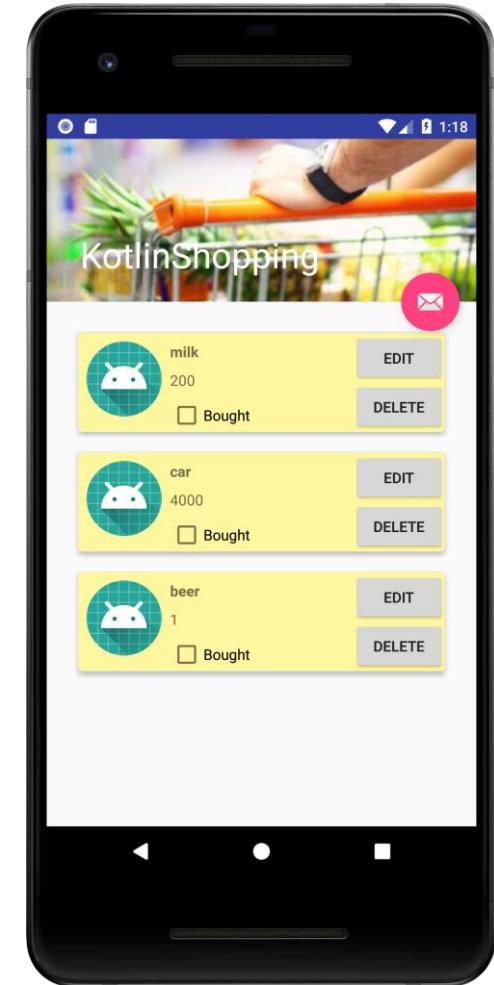
- Egészítsük ki az alkalmazást Todo hozzáadása funkcióval
- Alternatívaként a *DialogFragment* is kipróbálható



1. Data class
2. Egy sor layout-ja
3. RecyclerView elhelyezése
4. Adapter elkészítése
  - > RecyclerView és adapter összekötése

# Bevásárló lista példa - RecyclerView

- Termékek listázása
  - > Név, ár, vásárlás állapota
- Új termék felvitelle
  - > DialogFragment
- Törlés
- Szerkesztés
- Touch gesztusok



# Összefoglalás

- Animációk
- Fragmentek
- Listakezelés
  - > RecyclerView
  - > Új elem hozzáadása
  - > Törlés
  - > Módosítás

# Köszönöm a figyelmet!



*peter.ekler@aut.bme.hu*

# Mobil- és webes szoftverek

Dr. Ekler Péter

[ekler.peter@aut.bme.hu](mailto:ekler.peter@aut.bme.hu)



Department of  
Automation and  
Applied Informatics

# NagyHF emlékeztető

- <https://edu.vik.bme.hu/mod/resource/view.php?id=24069>
- Specifikáció – Október 23.

# Miről volt szó az előző órán?

- Animáció
- Fragment
- Listakezel
  - > Recycle
  - > Új elem
  - > Törlés
  - > Módosít



# Miről volt szó az előző órán?

- Animációk
- Fragmentek
- Listakezelés
  - > RecyclerView
  - > Új elem hozzáadása
  - > Törlés
  - > Módosítás

# Tartalom

- Perzisztens adattárolási lehetőségek
  - > SQLite, ORM
  - > SharedPreferences
  - > File
- Futási idejű engedélyek
- ContentProvider
- Az Intent fogalma

# Perzisztens adattárolás

# Bevezetés

- Gyakorlatilag minden Android alkalmazásnak kell perzisztensen tárolnia bizonyos adatokat
  - > Beállítások szinte minden alkalmazásban vannak
  - > Kamera alkalmazások: új fénykép fájl mentése
  - > Online erőforrásokat használó appok: lokális cache
  - > Email alkalmazások: levelek indexelt adatbázisa
  - > Bejelentkezést tartalmazó appok: be van-e jelentkezve a felhasználó
  - > Első indításkor tutorial megjelenítése: első vagy későbbi indítás?
  - > Picasa, Dropbox: elsődleges tárhely a felhőben

# Bevezetés

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

# SQLite

# SQLite

- Az Android alapból tartalmaz egy teljes értékű relációs adatbáziskezelőt
  - > SQLite – majdnem MySQL
- Strukturált adatok tárolására ez a legjobb választás
- Alapból nincs objektum-relációs réteg (ORM) fölötté, nekünk kell a sémát meghatározni és megírni a query-eket
- Külső ORM osztálykönyvtárak
- Mivel SQL, érdemes minden táblában elsődleges kulcsot definiálni
  - > autoincrement támogatás
  - > Ahhoz, hogy *ContentProvider*-el ki tudjuk ajánlani (később), illetve UI elemeket Adapterrel feltölteni (pl. list, grid), kötelező egy ilyen oszlop, melynek neve: „\_id”

# Android SQLite jellemzői 1/2

- Standard relációs adatbázis szolgáltatások:
  - > SQL szintaxis
  - > Tranzakciók
  - > Prepared statement
- Támogatott oszlop típusok (a többet ilyenekre kell konvertálni):
  - > TEXT (Java String)
  - > INTEGER (Java long)
  - > REAL (Java double)
- Az SQLite nem ellenőrzi a típust adatbeíráskor, tehát pl Integer érték automatikusan bekerül Text oszlopba szövegként

# Android SQLite jellemzői 2/2

- Az SQLite adatbázis elérés file rendszer elérést jelent, ami miatt lassú lehet!
- Adatbázis műveleteket érdemes asszinkron módon végrehajtani (pl *AsyncTask* használata v. *Loader*)

# SQLite debug

- Az Android SDK „tools” mappájában található egy konzolos adatbázis kezelő: sqlite3
- Ennek segítségével futás közben láthatjuk az adatbázist, akár emulátoron, akár telefonon
- Hasznos eszköz, de sajnos nincs grafikus felülete
- Használata (emulátoron, vagy root-olt eszközön):
  - > Konzolban megnyitjuk a **platform-tools** könyvtárat
  - > „adb shell” futtatása, egy eszköz legyen csatlakoztatva
  - > „**sqlite3 data/data/[Package név]/databases/[DB neve]**” futtatása
  - > Megkapjuk az SQLite konzolt, itt már az adatbázison futathatók közvetlen parancsokat (Pl. „.dump orak;”)

# Alap SQLite API (Java)

# SQLite használata

- Az adatbázis megnyitásához és a séma létrehozásához egy segédosztály használható: SQLiteOpenHelper

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
        KEY_WORD + " TEXT, " +
        KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

# SQLite használata

- Példányosítjuk a saját *SQLiteOpenHelper* osztályunkat
- A példányon meghívjuk a **getWritableDatabase ()** vagy **getReadableDatabase ()** metódust, amik egy *SQLiteDatabase* objektumot adnak vissza. Ez a teljes adatbázisunkat reprezentálja.
- Ezek az objektumon futtathatunk query-eket
  - > Ha nem akarjuk kézzel megírni, használhatjuk az *SQLQueryBuilder* segédosztályt
- minden query egy *Cursor* objektummal tér vissza, ami az eredmény adathalmazra mutat
  - > Rengeteg metódus a cursor használatára
  - > Érdemes do-while ciklussal végigmenni az adathalmazon

# SQLite használata

- INSERT és UPDATE: ContentValues objektumon keresztül

```
/* INSERT new record */
public long createOra(Ora ora) {
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, ora.getName());
    values.put(KEY_START, ora.getStart());
    values.put(KEY_END, ora.getEnd());
    values.put(KEY_LOCATION, ora.getLocation());
    long rowId = mDb.insert(DATABASE_TABLE, null, values);
    return rowId;
}
```

# SQLite használata

- Összes rekord lekérése

```
/* minden rekord lekérése */
public Cursor fetchAll() {
    return mDb.query(
        DATABASE_TABLE, // tábla
        new String[]{ KEY_NAME, KEY_START, KEY_END, KEY_LOCATION
        }, // oszlopok
        null, null, null, null, // WHERE, GROUP, HAVING
        KEY_START); // ORDER BY
}
```

# SQLite használata

- Cursor használata

```
OraDBAdapter oraDBAdapter = new OraDBAdapter(getApplicationContext());
Cursor cursor = oraDBAdapter.fetchAll();
Ora ora;
if(cursor.moveToFirst()) {
    do{
        String oraStart = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_START));
        String oraEnd = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_END));
        String oraName = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_NAME));
        String oraLocation = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_LOCATION));
        ora = new Ora(oraStart, oraEnd, oraName, oraLocation);
        // rendelkezésre áll az óra objektum
        //...
    } while(cursor.moveToNext());
}
```

- Adapter is képes Cursor-ból feltölteni a UI-t
  - > *BaseAdapter* helyett *CursorAdapter*-ból származtatunk
  - > Lásd jövő órán

# SQLite debug

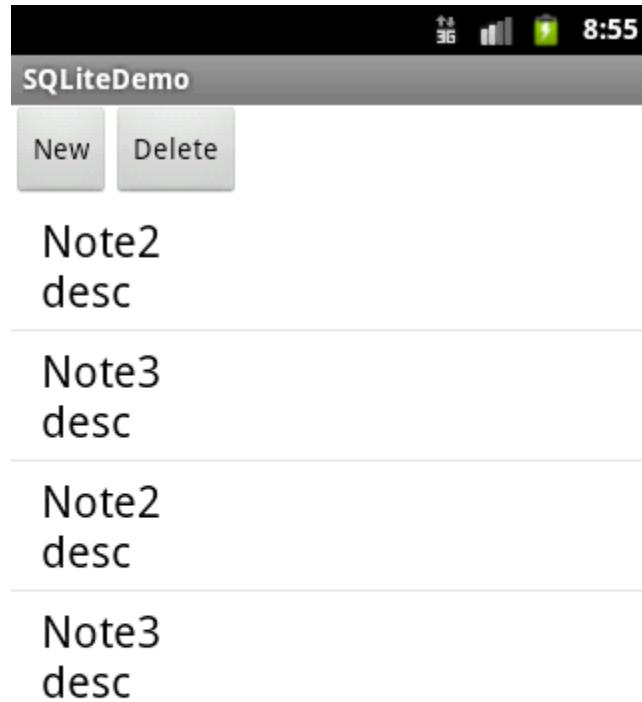
- Az Android SDK „tools” mappájában található egy konzolos adatbázis kezelő: sqlite3
- Ennek segítségével futás közben láthatjuk az adatbázist, akár emulátoron, akár telefonon
- Hasznos eszköz, de sajnos nincs grafikus felülete
- Használata (emulátoron, vagy root-olt eszközön):
  - > Konzolban megnyitjuk a **platform-tools** könyvtárat
  - > „adb shell” futtatása, egy eszköz legyen csatlakoztatva
  - > „**sqlite3 data/data/[Package név]/databases/[DB neve]**” futtatása
  - > Megkapjuk az SQLite konzolt, itt már az adatbázison futathatók közvetlen parancsokat (Pl. „.dump orak;”)

# SQLite Példa

# Példa leírása

- Készítsünk egy alkalmazást, melyben jegyzeteket (Note) tudunk tárolni.
- Note mezői:
  - > \_id: long
  - > title: String
  - > desc: String
- Tegyük lehetővé új Note létrehozását, törlését és listázását

# SQLite példa – Note készítő



# Note POJO (Plain Old Java Object)

```
public class Note {  
  
    private long id;  
  
    private String title;  
  
    private String desc;  
  
    // Getterek és Setterek  
    // ...  
  
    @Override  
    public String toString() {  
        return title+"\n"+desc;  
    }  
}
```

# Note tábla külön osztályba

```
public class NoteTable {  
    public static final String TABLE_NOTE = "note";  
    public static final String COLUMN_ID = "_id";  
    public static final String COLUMN_TITLE = "title";  
    public static final String COLUMN_DESC = "desc";  
  
    private static final String DATABASE_CREATE = "create table " +  
TABLE_NOTE  
        + "(" + COLUMN_ID + " integer primary key autoincrement, "  
        + COLUMN_TITLE + " text not null, " + COLUMN_DESC  
        + " text not null" + ");";  
  
    public static void onCreate(SQLiteDatabase database)  
    {  
        database.execSQL(DATABASE_CREATE);  
    }  
  
    public static void onUpgrade(SQLiteDatabase database, int oldVersion,  
    int newVersion) {  
        Log.w(NoteTable.class.getName(), "Eredeti verzió: " + oldVersion  
            + " új verzió" + newVersion);  
        database.execSQL("DROP TABLE IF EXISTS " + TABLE_NOTE);  
        onCreate(database);  
    }  
}
```

SQLite támogatja az  
AutoIncrement-et

Jelenleg csak újra  
létrehozzuk a táblát

# Saját SQLiteOpenHelper

```
public class NoteDBHelper extends SQLiteOpenHelper {  
    private static final String DATABASE_NAME = "notetable.db";  
    private static final int DATABASE_VERSION = 1;  
  
    public NoteDBHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase database) {  
        NoteTable.onCreate(database);  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase database, int oldVersion,  
    int newVersion) {  
        NoteTable.onUpgrade(database, oldVersion, newVersion);  
    }  
}
```

Adatbázis  
állomány neve és  
verziója

Tábla létrehozása

Tábla frissítése, ha  
változik a verzió

# DAO osztály az adatok eléréséhez 1/3

```
public class NoteDataSource {  
    // Database fields  
  
    private SQLiteDatabase database;  
  
    private NoteDBHelper dbHelper;  
  
    private String[] allColumns = { NoteTable.COLUMN_ID,  
        NoteTable.COLUMN_TITLE, NoteTable.COLUMN_DESC };
```

```
public NoteDataSource(Context context) {  
    dbHelper = new NoteDBHelper(context);  
}  
}
```

SQLiteDatabase  
példány

```
public void open() throws SQLException {  
    database = dbHelper.getWritableDatabase();  
}
```

Adatbázis  
megnyitása és  
lezárása

```
public void close() {  
    dbHelper.close();  
}
```

# DAO osztály az adatok eléréséhez 2/3

```
public Note createNote(String aTitle, String aDesc) {  
    ContentValues values = new ContentValues();  
    values.put(NoteTable.COLUMN_TITLE, aTitle);  
    values.put(NoteTable.COLUMN_DESC, aDesc);  
    long insertId = database.insert(NoteTable.TABLE_NOTE,  
        null, values);  
    Cursor cursor = database.query(NoteTable.TABLE_NOTE,  
        allColumns, NoteTable.COLUMN_ID + " = " + insertId, null,  
        null, null, null);  
    cursor.moveToFirst();  
    Note newNote = cursorToNote(cursor);  
    cursor.close();  
    return newNote;  
}  
  
private Note cursorToNote(Cursor cursor) {  
    Note note = new Note();  
    note.setId(cursor.getLong(0));  
    note.setTitle(cursor.getString(1));  
    note.setDesc(cursor.getString(2));  
    return note;  
}
```

ContentValues  
használata  
beillesztéshez

# DAO osztály az adatok eléréséhez 3/3

```
public void deleteNote(Note note) {  
    long id = note.getId();  
    database.delete(NoteTable.TABLE_NOTE, NoteTable.COLUMN_ID  
        + " = " + id, null);  
}  
  
public List<Note> getAllNotes() {  
    List<Note> notes = new ArrayList<Note>();  
    Cursor cursor = database.query(NoteTable.TABLE_NOTE,  
        allColumns, null, null, null,  
        cursor.moveToFirst(); ←  
    while (!cursor.isAfterLast()) {  
        Note note = cursorToNote(cursor);  
        notes.add(note);  
        cursor.moveToNext();  
    }  
    // Ne felejtsük bezárni!  
    cursor.close();  
    return notes;  
}
```

Iteráció Cursor  
segítségével

# ListActivity 1/2

```
public class MainActivity extends ListActivity {  
    private NoteDataSource datasource;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        datasource = new NoteDataSource(this);  
        datasource.open(); ←  
        List<Note> values = datasource.getAllNotes();  
        // SimpleCursorAdapter használata az elemek egyszerű megjelenítéséhez  
        ArrayAdapter<Note> adapter = new ArrayAdapter<Note>(this,  
            android.R.layout.simple_selectable_list_item, values);  
        setListAdapter(adapter);  
    }  
    @Override  
    protected void onResume() {  
        datasource.open();  
        super.onResume();  
    }  
    @Override  
    protected void onPause() {  
        datasource.close(); ←  
        super.onPause();  
    }  
}
```

Adatbázis  
megnyitása

Lista feltöltése

Adatbázis  
kapcsolat  
bezárása

# ListActivity 2/2

```
// XML-ból állítjuk be az eseménykezelőt
public void onClick(View view) {
    @SuppressWarnings("unchecked")
    ArrayAdapter<Note> adapter = (ArrayAdapter<Note>) getListAdapter();
    Note note = null;
    switch (view.getId()) {
        case R.id.newnote:
            String[] notes = new String[] { "Note1", "Note2", "Note3" };
            int nextInt = new Random().nextInt(3);
            note = datasource.createNote(notes[nextInt], "desc");
            adapter.add(note);
            break;
        case R.id.delnote:
            if (getListAdapter().getCount() > 0) {
                note = (Note) getListAdapter().getItem(0);
                datasource.deleteNote(note);
                adapter.remove(note);
            }
            break;
    }
    adapter.notifyDataSetChanged();
}
```

Lista értesítése  
a változásról

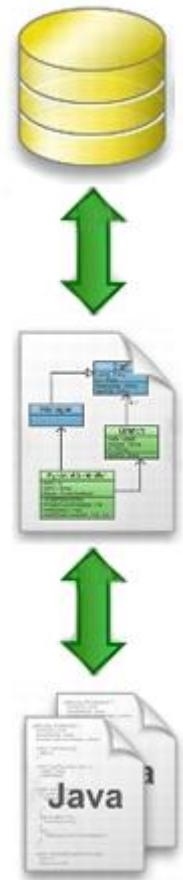
# Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <LinearLayout
        android:id="@+id/group"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/newnote"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="New"
            android:onClick="onClick"/>
        <Button
            android:id="@+id/delnote"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Delete one"
            android:onClick="onClick"/>
    </LinearLayout>
    <ListView
        android:id="@+id/list" <-->
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="empty" />
</LinearLayout>
```

ListActivity ezen azonosító  
alapján azonosítja

# Mi az ORM?

- Objektumok tárolása relációs adatbázisban
- Alapelvek:
  - > Osztálynév -> Tábla név
  - > Objektum -> Tábla egy sora
  - > Mező -> Tábla oszlopa
  - > Stb.



# ORM példa – objektumok tárolása

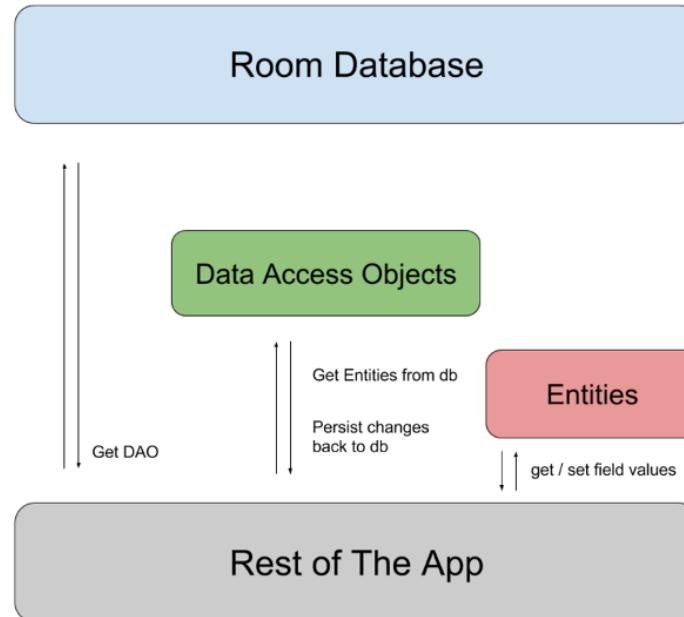
```
Class Person {  
    Private String name;  
    Private String address;  
    Private String tel;  
}
```

# ORM könyvtárak Androidon

- Room – Official Google
  - > <https://developer.android.com/topic/libraries/architecture/room>
- Realm.io
  - > <http://realm.io/>
- Sugar-ORM
  - > <http://satyan.github.io/sugar/index.html>
- ORMLite
  - > <http://ormlite.com/>
- GreenDAO
  - > <http://greendao-orm.com/>
- MapDB:
  - > <http://www.mapdb.org/>

# Room Persistence Library

- Absztraktiós réteg az SQLite felett
- SQLite teljes képességeinek használata
- Room architektúra:



# Szálkezelés

- *Thread*
  - > <https://developer.android.com/guide/components/processes-and-threads.html>
- A felhasználói felület csak a fő szálról módosítható:
  - > *runOnUiThread(runnable: Runnable)*
- Szálakat le kell állítani
  - > Biztosítani kell, hogy a run() függvény befejeződjön, ne maradjon végtelen ciklusban

# Szál példa

```
private inner class MyThread : Thread() {
    override fun run() {
        while (threadEnabled) {
            runOnUiThread {
                Toast.makeText(this@MainActivity,
                    "Message", Toast.LENGTH_LONG).show()
            }
            Thread.sleep(6000)
        }
    }
}

MyThread().start()
```

# Room példa - Entity

```
@Entity(tableName = "grade")
data class Grade(
    @PrimaryKey(autoGenerate = true) var gradeId: Long?,
    @ColumnInfo(name = "studentid") var studentId: String,
    @ColumnInfo(name = "grade") var grade: String
)
```

# Room példa - DAO

```
@Dao
interface GradeDAO {
    @Query("""SELECT * FROM grade WHERE grade="B""""")
    fun getBGrades(): List<Grade>

    @Query("SELECT * FROM grade")
    fun getAllGrades(): List<Grade>

    @Query("SELECT * FROM grade WHERE grade = :grade")
    fun getSpecificGrades(grade: String): List<Grade>

    @Insert
    fun insertGrades(vararg grades: Grade)

    @Delete
    fun deleteGrade(grade: Grade)
}
```

# RoomDatabase

```
@Database(entities = [Grade::class], version = 1)
abstract class AppDatabase : RoomDatabase() {

    abstract fun gradeDao(): GradeDAO

    companion object {
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            if (INSTANCE == null) {
                INSTANCE = Room.databaseBuilder(context.applicationContext,
                    AppDatabase::class.java, "grade.db").build()
            }
            return INSTANCE!!
        }

        fun destroyInstance() {
            INSTANCE = null
        }
    }
}
```

# Room használat

- Insert

```
val grade = Grade(null, etStudentId.text.toString(),
    etGrade.text.toString())

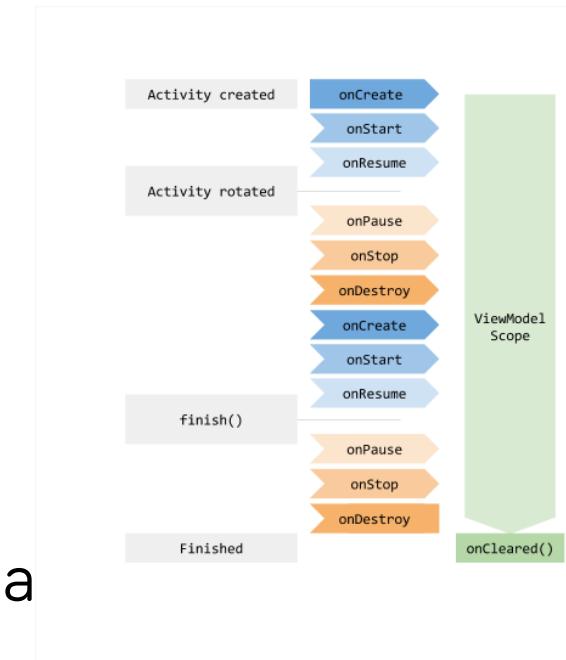
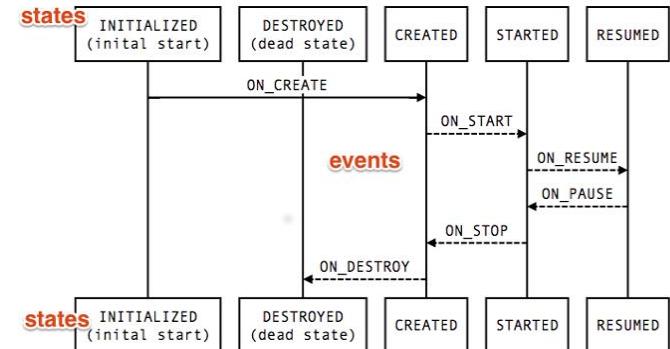
val dbThread = Thread {
    AppDatabase.getInstance(this@MainActivity).gradeDao().insertGrades(grade)
}
dbThread.start()
```

- Query

```
val dbThread = Thread {
    val grades = AppDatabase.getInstance(this@MainActivity).gradeDao()
        .getSpecificGrades("A+")
    runOnUiThread {
        tvResult.text = ""
        grades.forEach {
            tvResult.append("${it.studentId} ${it.grade}\n")
        }
    }
}
dbThread.start()
```

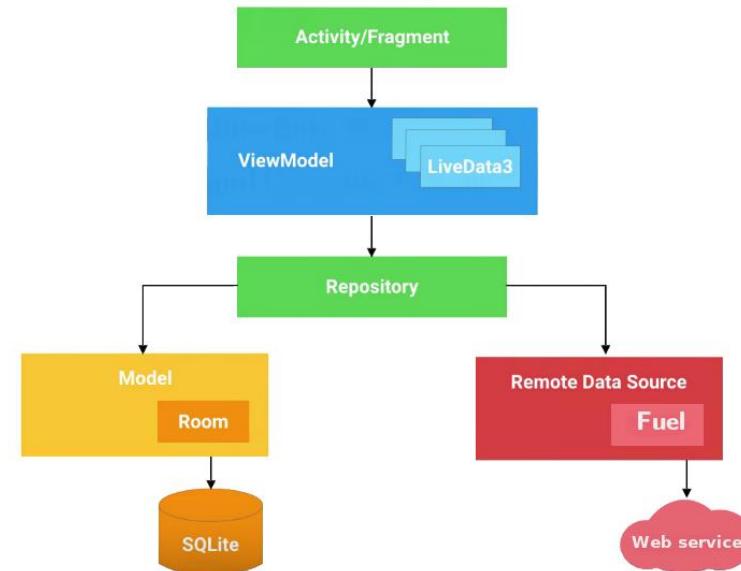
# Architecture Components elemek

- Lifecycle / LifecycleObserver
  - > Lifecycle-függő komponensek, objektumok hozhatók létre
- LiveData
  - > Adat kezelő, megfigyelhetővé teszi az adatot
- ViewModel
  - > UI-on megjelenő adatok egyszerű kezelése, mely független a konfiguráció változástól
- Room Persistence Library
  - > Google saját ORM megoldása, a teljes SQLite képességeket kihasználja



# Javasolt architektúra

- Az *Activity*-k, *Fragmentek*, egyedi nézetek *ViewModel*-eket használnak
- A *ViewModel*-ek *LiveData*-kon keresztül teszik megfigyelhetővé az adatokat
- A *ViewModel*-ek *Repository*-kat használnak az adatforrások elrejtéséhez
- Perzisztencia használata offline működés támogatására



# További javaslatok

- Room-LiveData kapcsolat
  - > Room lekérdezés LiveData<T>-t is képes visszaadni
- Könnyű ListAdapterrel összekötni
- Egyszerűbb és egységes RecyclerView kezelés

# Shared Preferences

# SharedPreferences

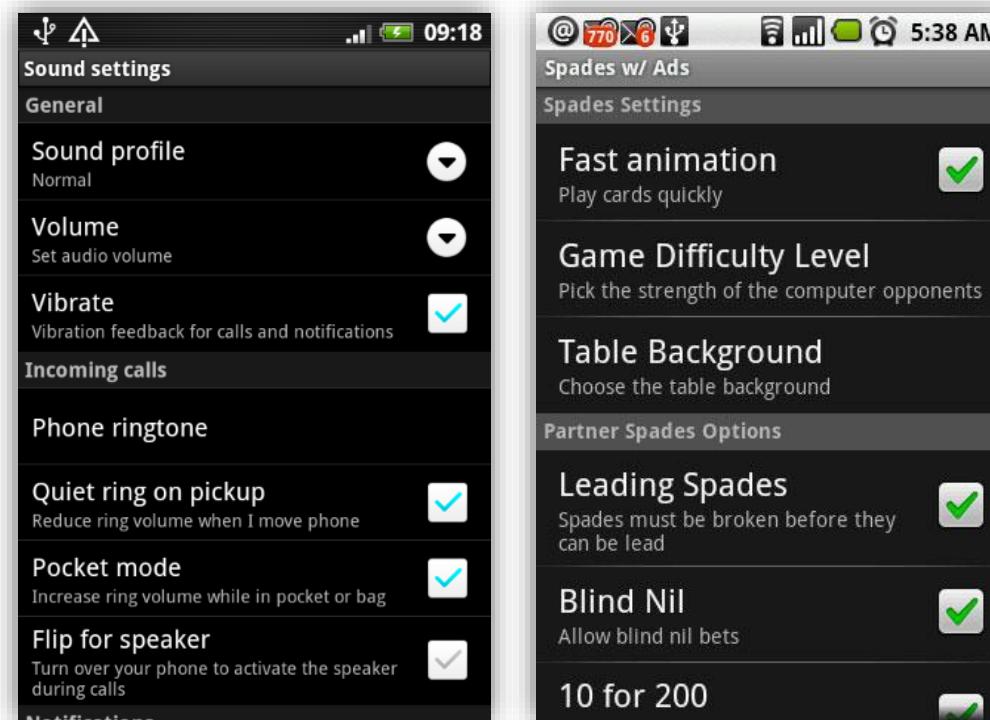
- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
  - > Típusok: *int*, *long*, *float*, *String*, *boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
  - > **MODE\_PRIVATE**: csak a saját alkalmazásunk érheti el
  - > **MODE\_WORLD\_READABLE**: 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 (String name, int mode)**
  - > Ha még nem létezik ilyen nevű, akkor az Android létrehozza
- Ha elég egy SP egy Activity-hez, akkor nem kötelező elnevezni
  - > **getPreferences ()**

# Preferences Framework

- Az Android biztosít egy XML alapú keretrendszeret saját Beállítások képernyő létrehozására
  - > Ugyanúgy fog kinézni mint az alap Beállítások alkalmazás
  - > Más alkalmazásokból, akár az op.rendszerből is átemelhető részek



# Példa Preferences nézet - XML

```
<?xml version="1.0" encoding="utf-8"?> <PreferenceScreen xmlns:android= "http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="Választás" >
        <ListPreference
            android:title="Horoszkóp"
            android:summary="Kérem válasszon"
            android:key="listPref"
            android:entries="@array/listDisplayMarks"
            android:entryValues="@array/listReturnMarks"
        </PreferenceCategory>
    </PreferenceScreen>
    <PreferenceCategory android:title="Beléptetés" >
        <EditTextPreference
            android:defaultValue="empty"
            android:key="name"
            android:title="Username" />
        <CheckBoxPreference
            android:defaultValue="false"
            android:key="autologin"
            android:title="Automatikus belépés" />
        <SwitchPreference
            android:title="Adatok megjegyzése"
            android:key="remember"
            android:summaryOff="Kikapcsolva"
            android:summaryOn="Bekapcsolva"/>
    </PreferenceCategory>
```

# Android Engedélyek (Permission modell)

# Mire szolgálnak az engedélyek?

- Veszélyes/kritikus/személyes adatokat érintő műveletekhez engedélyre van szükség a felhasználótól.
- Korábban:
  - > Manifest állományban egyszerű bejegyzés:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```
  - > Majd az összes engedély egyszeri elkérése telepítéskor
- Új permission modell 6-os Androidtól felfele:
  - > Engedély elkérése futási időben Java kódból

# Futási idejű permission kezelés

- Permission meglétének ellenőrzése:

```
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity,  
    Manifest.permission.WRITE_CALENDAR);
```

- Permission kérés Activity-ben:

- Megvizsgálni, hogy már nem kaptuk-e meg az engedélyt
- Ha szükséges elmagyarázni a felhasználónak, hogy miért kérjük, majd újra kérni
- Permission kérés eredményének kezelése egyesével
  - Engedélyezés
  - Tiltás

# Permission típusok

- Manifest állományban továbbra is érdemes felsorolni a permission-öket
  - > Normál permission-öket automatikusan megkap az alkalmazás
  - > Veszélyes permission-öket kérni kell Java kódból
- Normál és veszélyes permission-ök listája:
  - > <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>
- Korábban megadott engedélyeket a felhasználó visszavonhatja!
- További részletek:
  - > <https://developer.android.com/training/permissions/requesting.html>

# Permission kérés 1/2

```
private fun requestNeededPermission() {
    if (ContextCompat.checkSelfPermission(this,
            android.Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                android.Manifest.permission.CAMERA)) {
            Toast.makeText(this,
                    "I need it for camera", Toast.LENGTH_SHORT).show()
        }

        ActivityCompat.requestPermissions(this,
                arrayOf(android.Manifest.permission.CAMERA),
                PERMISSION_REQUEST_CODE)
    } else {
        // már van engedély
    }
}
```

# Permission kérés 2/2

```
override fun onRequestPermissionsResult(requestCode: Int,
permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        PERMISSION_REQUEST_CODE -> {
            if (grantResults.isNotEmpty() && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "CAMERA perm granted",
                    Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "CAMERA perm NOT granted",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

# Külső könyvtár permission kérésre

- <https://github.com/hotchemi/PermissionsDispatcher>

# Legjobb gyakorlatok permission kezelésre

- Használjuk lehetőleg a rendszer beépített szolgáltatásait (pl. kamera)!
- Csak a szükséges engedélyeket kérjük el!
- Ne terheljük túl a felhasználót kérésekkel!
  - > Csak akkor kérjük el az engedélyt, amikor szükség van rá!
- Magyarázzuk el a felhasználónak, hogy miért van szükség az adott engedélyre!
- Teszteljük mindenkét permission kezelést (manifest és Java kód) korábbi és új Android verziókon!

# File kezelés

# Fájlkezelés Androidon

- Ugyanaz mint sima Java esetén
- Néhány specialitás Android környezetben
- Két lemezterületet különböztet meg
  - > **Internal storage**: az a védett tárhely, amit kizárolag az alkalmazás érhet el, se a user, se más appok nem fér hozzá
    - (az „Android biztonság” előadáson meglátjuk hogy mennyire igaz ☺)
  - > **External storage**: felhasználó által is írható-olvasható terület (~SD kártya)
- *External storage* is lehet belső memóriában, ha nincs SD kártya a készülékben (mondjuk mert tablet)

# Internal storage

- *openFileOutput(String filename, int mode)*
  - > filename-ben nem lehet „\”, egyébként kivételt dob (Miért?)
  - > Támogatott módok:
    - Context.MODE\_PRIVATE: alapértelmezett megnyitási mód, felülírja a fájlt ha már van benne valami
    - Context.MODE\_APPEND: hozzáfűzi a fájlhoz amit beleírunk
    - Lehet WORLD\_READABLE vagy WORLD\_WRITEABLE is, ha szükséges, de nem ez a javasolt módja az adatok kialakításának, hanem a ContentProvider (később)
  - > Privát vagy Append mód esetén nincs értelme kiterjesztést megadni, mert máshonnan úgysem fogják megnyitni
  - > Ha nem létezik a fájl akkor létrehozza, a **WORLD\_\*** módok csak ekkor értelmezettek

# Internal storage

- Fájl olvasása ugyanígy:
  - > **openFileInput (String filename)** hívása (FileNotFoundException-t dobhat)
  - > Byte-ok kiolvasása a visszakapott *FileInputStream*-ből a **read ()** metódussal
  - > Stream bezárása **close()** metódussal!
- Cache használata
  - > Beépített mechanizmus arra az esetre, ha cache-ként akarunk fájlokat használni
  - > **getCacheDir ()** metódus visszaad egy File objektumot, ami a cache könyvtárra mutat (miért File?)
  - > Ezen belül létrehozhatunk cache fájlokat
  - > Kevés lemezterület esetén először ezeket törli az Android
    - Nem számíthatunk rá, hogy mindenkor ott lesznek!
  - > Google ajánlás: maximum 1MB-os fájlokat rakjunk ide (Miért?)

# Statikus fájlok egy alkalmazáshoz

- Szükséges lehet a fejlesztett alkalmazáshoz statikusan fájlokat linkelni
  - > Kezdeti, nagy méretű, feltöltött bináris adatbázis fájl
  - > Egyedi formátumú állomány
  - > Bármi ami fájl, de nem illik a res könyvtár mappáiba (drawable, xml, stb)
- Fejlesztéskor a **res/raw** mappába kell rakkunk őket
- Ezek telepítéskor szintén az internal storage-be kerülnek
- Read-only lesz telepítés után, nem tudjuk utólag módosítani
- Olvasásuk futásidőben:

```
val instream: InputStream =  
    resources.openRawResource(R.raw myfile)
```

# Nyilvános lemezterület

- Lehet akár SD kártyán, akár belső (nem kivehető) memóriában
- Bárki által írható, olvasható a teljes fájlrendszer
- Amikor a felhasználó összeköti a telefont a számítógépével, és „*USB storage*” módra vált (mount), a fájlok hirtelen csak olvashatóvá válnak az alkalmazások számára
- Semmilyen korlátozás/tiltás nincs arra, hogy a nyilvános területen lévő fájljainkat a felhasználó letörölje, lemásolja vagy módosítsa!
  - > Amit ide írunk, az bármikor elveszhet

# Nyilvános lemezterület

- Legfontosabb tudnivalók
  - > Használat előtt ellenőrizni kell a tárhely elérhetőségét
  - > Fel kell készülni arra, hogy bármikor elérhetetlenné válik

```
val state = Environment.getExternalStorageState()
// sokféle állapotban lehet, nekünk kettő fontos:
if (state.equals(Environment.MEDIA_MOUNTED)) {
    // Olvashatjuk és írhatjuk a külső tárat
} else if (state.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    // Csak olvasni tudjuk
} else {
    // Valami más állapotban van, se olvasni,
    // se írni nem tudjuk
}
```

# Nyilvános lemezterület

- Fájlok elérése a nyilvános tárhelyen 2.2 verziótól felfelé:

```
val filesDir = getExternalFilesDir(int type)
```

- type: megadhatjuk milyen típusú fájlok könyvtárát akarjuk használni, például:
  - > null: nyilvános tárhely gyökere
  - > DIRECTORY\_MUSIC: zenék, ahol az zenelejátszó keres
  - > DIRECTORY\_PICTURES: képek, ahol a galéria keres
  - > DIRECTORY\_RINGTONES: csengőhangok, ez is hang fájl, de nem zenelejátszóban akarjuk hallgatni
  - > DIRECTORY\_DOWNLOADS: letöltések default könyvtára
  - > DIRECTORY\_DCIM: a kamera ide rakja a fényképeket
  - > DIRECTORY\_MOVIES: filmek default könyvtára

# Nyilvános lemezterület

- Média típusonként külön alapértelmezett könyvtárak
- Így az azokat lejátszó/kezelő alkalmazásoknak nem kell az egész lemezt végigkeresni, csak a megfelelő könyvtárakat
- Indexelésüket a MediaScanner osztály végzi
  - > Ez mindenhol keres, és ha a talált média fájlok nem default könyvtárban vannak, akkor megpróbálja kategorizálni őket kiterjesztésük és MIME típusuk szerint
  - > Ha nem szeretnénk beengedni egy könyvtárba, akkor egy üres fájlt kell elhelyezni, melynek neve: ".nomedia"
    - Így például egy alkalmazás által készített fotók nem fognak látszódni a galériában
  - > A megfelelő default könyvtárba rakjuk az alkalmazásunk által létrehozott fájlokat, ha meg akarjuk osztani a userrel
- *android.permission.WRITE\_EXTERNAL\_STORAGE*

# CONTENT PROVIDER

# Motiváció 1/2

Eddigi lehetőségeink adatok megosztására komponensek/alkalmazások között:

- **Intent Data**

- > Nem erre való
  - > Intent kell hozzá, ami néha felesleges

- **SharedPreferences**

- > Nem kényelmes sok adat esetén
  - > Ismerni kell a kulcsok nevét
  - > Komplex adatstruktúrához használhatatlan

- **Fájlok a nyilvános lemezterületen**

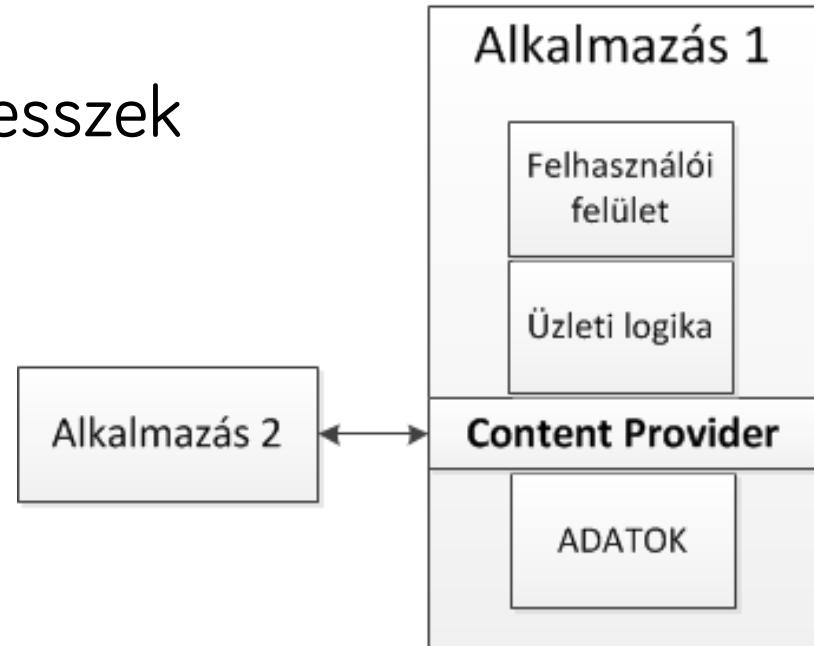
- > Bármikor elérhetetlenné válhat
  - > Látható és módosítható, akár törölhető a felhasználó által

# Motiváció 2/2

- Egyik sem igazán jó megoldás
- A funkcióra azonban gyakran szükség van
  - > Komplex alkalmazás fejlesztése esetén érdemes elválasztani az adat és az üzleti logika rétegeket (Miért?)
  - > „Natív” adatok elérése - névjegyzék, naptár, SMS, felhasználói fiókok, stb...
  - > Saját alkalmazásunk által létrehozott adatok elérhetővé tétele mások számára

# Content Provider

- Megoldás: olyan mechanizmus, ami
  - > Elérési réteget biztosít strukturált adatokhoz
  - > Elfedi az adat tényleges tárolási módját
  - > Adatvédelem biztosítható
  - > Megvalósítható akár a processzek közti adatmegosztás is
- Neve: Content Provider



# Adattárolás

- Konkrét adattárolási struktúra/módszer az alkalmazásra bízva
- Ami szükséges: a megfelelő interfész megvalósítása
  - > Leszármaztatás az absztrakt *ContentProvider* osztályból és a kötelező metódusok implementálása
- A legegyszerűbb SQLite adatbázissal csinálni, de nem kötelező ezzel

# Content Provider elérése

- Content Provider-től kérdezhetjük le az adatokat
  - > „content://contacts”
- A komponens ami képes a lekérdezések futtatására és a válasz feldolgozására: **ContentResolver**
  - > Csak ez tudja lekérdezni a Content Providert
  - > Lehet akár ugyanabban, akár másik alkalmazásban (processzben)
  - > A kommunikációhoz szükséges IPC-t az Android elintézi a fejlesztő helyett, teljesen átlátszó
  - > Egy Content Providerből egyszerre egy példány futhat (singleton), ezt éri el az összes Resolver

# ContentProvider műveletek

- Nem csak adatlekérés lehet, hanem teljes CRUD funkcionalitás:
  - > **SELECT: `getContentResolver().query(...)`**
    - Visszatérés: Cursor az eredményhalmazra
  - > **INSERT: `getContentResolver().insert(...)`**
    - Visszatérés: a beszúrt adatra mutató URI
  - > **UPDATE: `getContentResolver().update(...)`**
    - Visszatérés: az update által érintett sorok száma
  - > **DELETE: `getContentResolver().delete(...)`**
    - Visszatérés: a törölt sorok száma

# CONTENT\_URI

- Azonosítja a Content Provider-t, és azon belül a táblát
- Pl. **UserDictionary.Words.CONTENT\_URI = content://user\_dictionary/words**
- Felépítése:
  - > **content://** - séma, ez mindenkor van, ebből tudja a rendszer hogy ez egy Content URI
  - > **user\_dictionary** - „authority”, azonosítja a Providert, globálisan egyedinek kell lennie
  - > **words** - „path”, az adattábla (NEM adatbázis tábla!) neve amelyre a lekérés vonatkozik, egy Provider több táblát is kezelhet

# Telefonkönyv listázás példa

```
val cursorContacts = contentResolver.query(
    ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
    arrayOf(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME,
        ContactsContract.CommonDataKinds.Phone.NUMBER),
    ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME + " LIKE '%Tamás%'",
    //null,
    null,
    ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME + " DESC")

//Toast.makeText(MainActivity.this, ""+c.getCount(), Toast.LENGTH_LONG).show();

while (cursorContacts.moveToNext()) {
    val name = cursorContacts.getString(cursorContacts.getColumnIndex(
        ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME))
    Log.d(KEY_LOG, name)
    Toast.makeText(this@MainActivity, name, Toast.LENGTH_LONG).show()
}
```

# Naptár beszúrás példa (API 14-től)

```
val values = ContentValues()
values.put(CalendarContract.Events.DTSTART, System.currentTimeMillis())
values.put(CalendarContract.Events.DTEND, System.currentTimeMillis() + 60000)

values.put(CalendarContract.Events.TITLE, "Vége")
values.put(CalendarContract.Events.DESCRIPTION, "Legyen már vége az órának")

values.put(CalendarContract.Events.CALENDAR_ID, 1)
values.put(CalendarContract.Events.EVENT_TIMEZONE, TimeZone.getDefault().getID())

val uri = contentResolver.insert(CalendarContract.Events.CONTENT_URI, values)
```

# Komponens közi kommunikáció, Intent

# Lazán csatolt komponensek és köztük a kommunikáció

## Mi a helyzet Androidon?

- Alkalmazások külön ART VM példányokban (ez is sandbox)
- Kritikus műveletekhez engedély szükséges
- Alkalmazás = komponensek halmaza

A komponensek akár alkalmazások között is kommunikálhatnak egymással (!)

- Két komponens között: *Intent*
- Egy komponensből mindenki másnak: *Broadcast Intent*
- Csak adat megosztása (ContentProvider)

# Kommunikáció formái 1/3

- Egyik komponensből a másikba: *Intent*

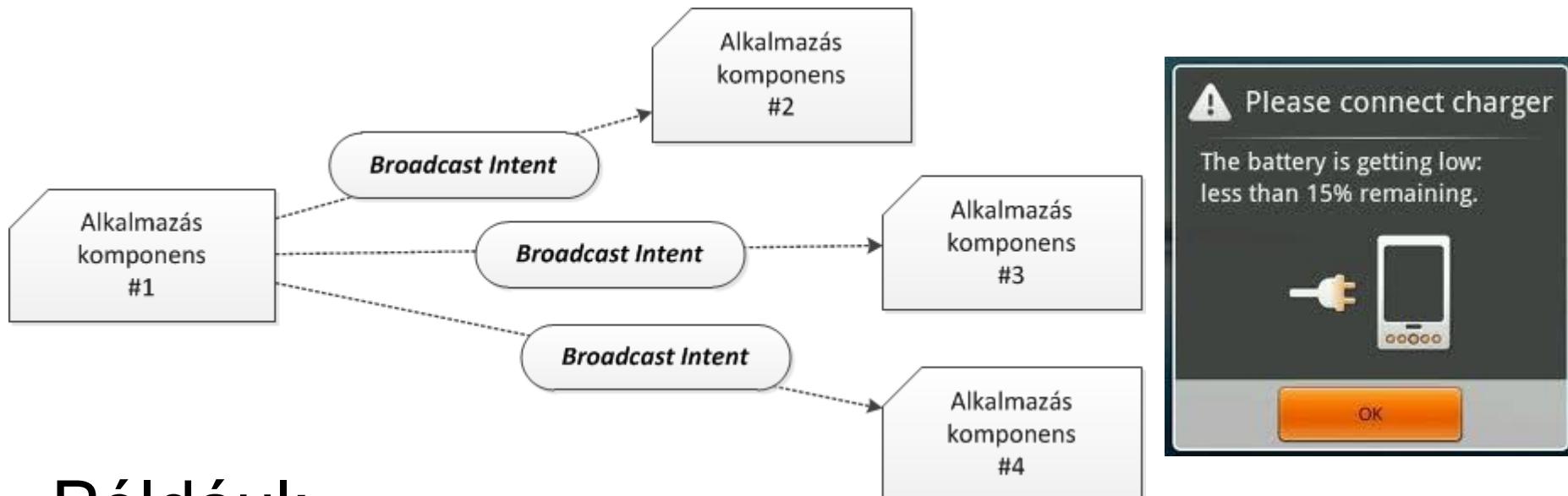


Például:

- Következő képernyőre lépés (új Activity indítása)
- Zenelejátszó service indítása

# Kommunikáció formái 2/3

- Egy komponensből mindenki másnak: *Broadcast Intent*

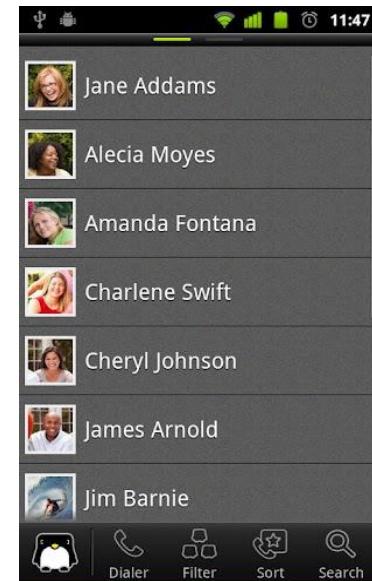
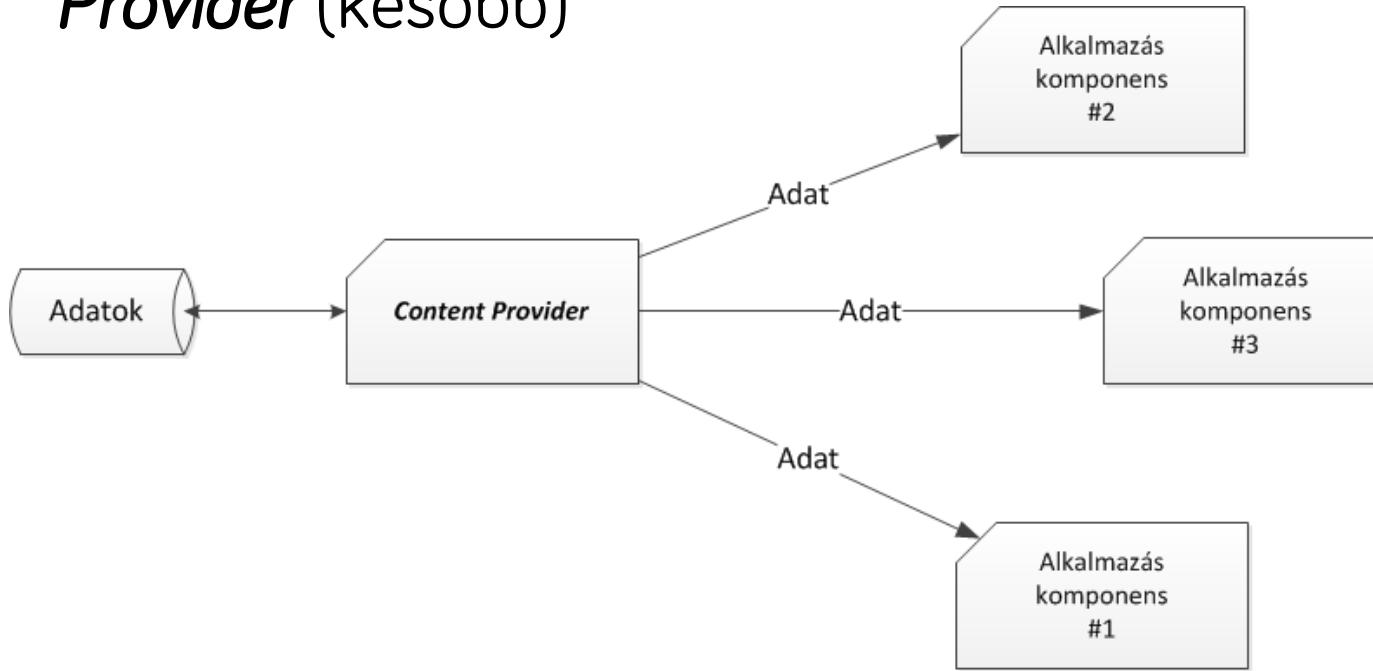


Például:

- „Akkufeszültség alacsony” rendszerüzenet

# Kommunikáció formái 3/3

Adatok szolgáltatása komponensek közt: *Content Provider* (később)

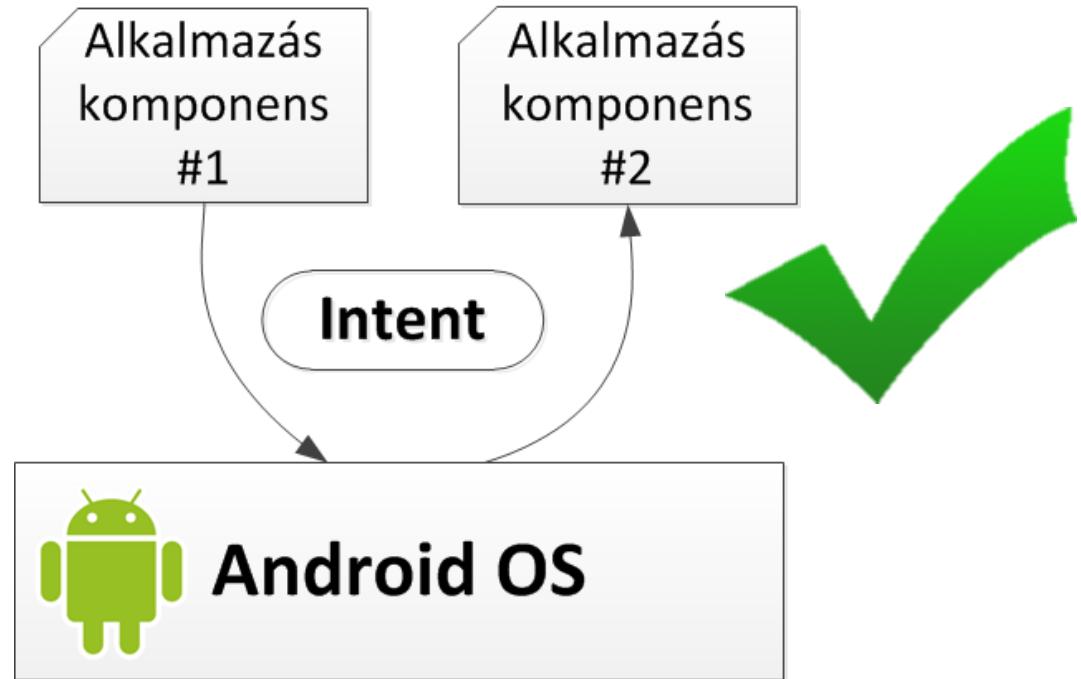
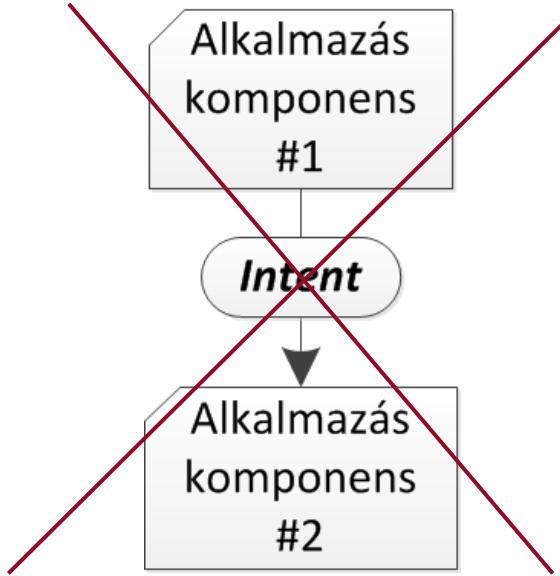


Például:

- Névjegyzék elérése saját alkalmazásból

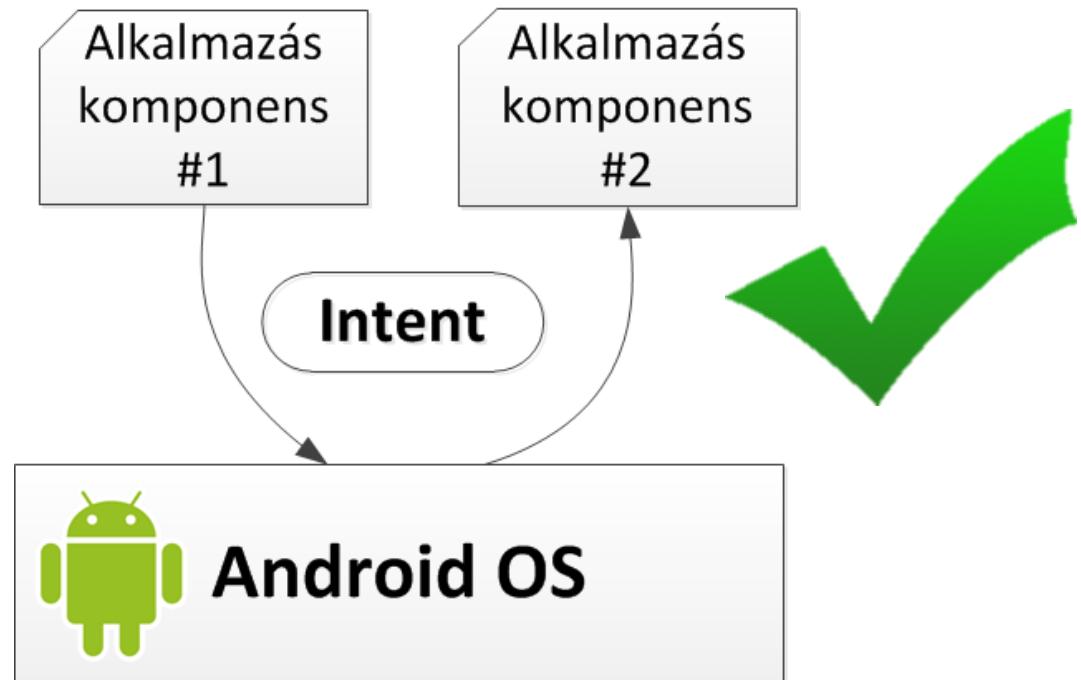
# Intent átadása

- Mindig az Android runtime-on keresztül!



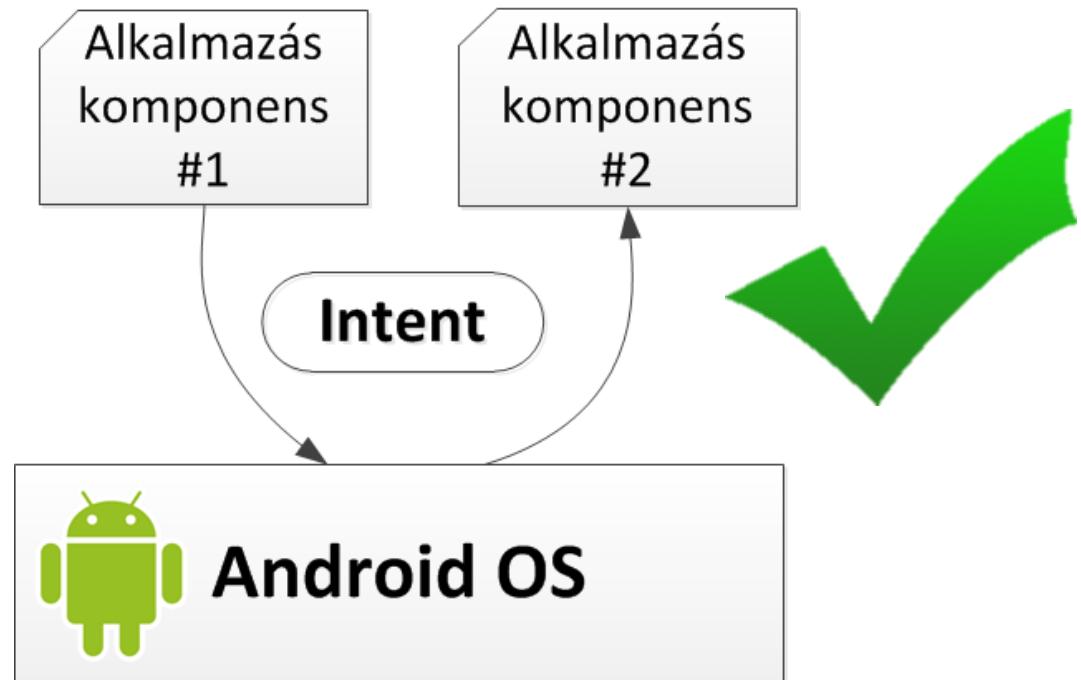
# Intent átadása

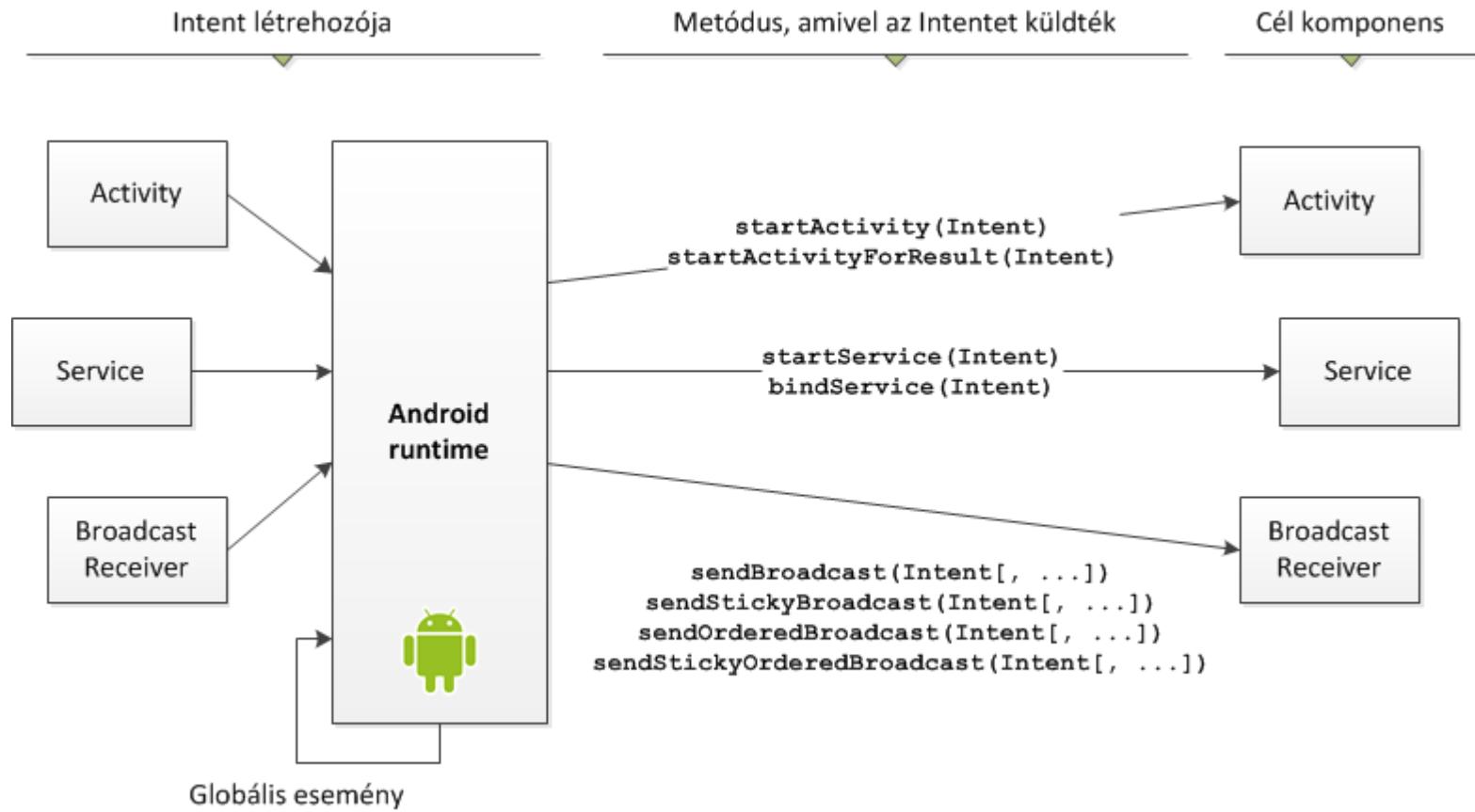
- Mindig az Android runtime-on keresztül!



# Intent átadása

- Mindig az Android runtime-on keresztül!





# Intent típusai és részei

- Intent típusok:

- > Explicit Intent:
  - konkrétan meg van nevezve a cél komponens
- > Implicit Intent:
  - a végrehajtandó feladat kerül leírásra

- Intent részei:

- > Címzett komponens osztályneve (*Component name*): ha üres akkor az Android megkeresi a megfelelőt
- > Akció (*Action*): az elvárt vagy megtörtént esemény
- > Adat (*Data*): az adat (URI-ja és MIME típusa), amin az esemény értelmezett
- > Kategória (*Category*): további kritériumok a feldolgozó komponesszel kapcsolatban
- > Extrák (*Extras*): saját kulcs-érték párok, amiket át akarunk adni a címzettnek
- > Kapcsolók (*Flags*): Activity indításának lehetőségei

# Explicit Intent

- Mindkét esetben a `startActivity()` függvényt használjuk:
  - > `startActivity(Intent)`
- **Explicit hívás:** az Intent-ben kitöltjük a címzett komponens nevét (konstruktorból vagy setterrel)

```
var i: Intent = Intent(getApplicationContext(),  
                           ListProductsActivity::class.java)  
startActivity(i)
```

- Ha a `ListProductsActivity`-ből már van példány a memóriában akkor folytatódik, ha nincs akkor az Android példányosítja és elindítja

# Implicit Intent - Példa

- Telefonszám felhívása

```
var i: Intent = Intent(Intent.ACTION_DIAL,  
                      Uri.parse(„tel:0630-123-4567”))  
startActivity(i)
```

Akció

Adat (URI)

- Névjegy kiválasztása

```
var i: Intent = Intent(Intent.ACTION_PICK,  
                      ContactsContract.Contacts.CONTENT_URI)  
startActivity(i)
```

Akció

Adat (URI)

# Intent képességek

- Android alkalmazás komponensek:
  - > Activity, Service, Broadcast Receiver
- Kommunikáció köztük: Intentekkel
- Nem csak alkalmazáson belül, hanem azok között is lehetséges
  - > Használhatunk más alkalmazásban lévő komponenst
  - > Kiajánlhatjuk a sajátunkat
- Rendszerszintű eseményeket kezelhetünk

# Intent Filter

- Lehetséges a saját alkalmazásunk funkcióinak kiajánlása mások számára
  - > Az Androidban beépítve vannak ilyenek, ld. Intent Action (pl. ACTION\_CALL, ACTION\_IMAGE\_CAPTURE)
- Az AndroidManifest-ben kell deklarálni (miért?)
- Ha nincs Intent filter beállítva, akkor a komponens kizárolag explicit intentet képes fogadni
- Ha van Intent filter, akkor explicit és implicit intenteket is ki tud szolgálni

# Mit nevezünk Explicit Intentnek?

- A. Ami kihív az alkalmazásból.
- B. Ami explicit képet ad vissza hívás után.
- C. Ami konkrét telefonszámot hív fel.
- D. Amikor megadjuk a konkrét osztályt (komponenst) akinek a kérést küldjük.

# Összefoglalás

- Perzisztens adattárolási lehetőségek
  - > SQLite, ORM
  - > SharedPreferences
  - > File
- Futási idejű engedélyek
- ContentProvider
- Az Intent fogalma

# Kérdések



# Mobil- és webes szoftverek

Dr. Ekler Péter

[ekler.peter@aut.bme.hu](mailto:ekler.peter@aut.bme.hu)



Department of  
Automation and  
Applied Informatics

# Mobil alkalmazás fejlesztői verseny ☺

A legjobb 3 megoldást  
díjazzuk: HONOR MagicBook



HONOR

# HONOR

- Nemzetközi technológiai márka
  - A Huawei Cégecsoporthoz tartozó e-brand elsősorban fiatalok számára
  - A legújabb technológiák jó ár-érték arányban
  - Az új mobilok a HMS (Huawei Mobile Services) ökoszisztemára épülnek, Huawei AppGallery alkalmazásáruházzal.
- @honorhungary  
hihonor.com/hu



## A feladat:

Olyan kreatív mobil alkalmazást készíteni a HMS (Huawei Mobile Services) platformra, a Huawei AppGallery-re, amely megkönnyíti a fiatalok életét.

## Technikai követelmények:

- Legalább 3 HMS AppService-t fel kell használni a megvalósításhoz
- Olyan minőségben kell létrehozni, hogy bekerülhessen a Huawei AppGallery alkalmazásáruházba
- A magyar nyelvet kell támogatnia, az angol pluszban előny.

Forrás: [developer.huawei.com/consumer/en/hms](https://developer.huawei.com/consumer/en/hms)

## Tesztkészülékek

7nm Kirin 810 AI processzor  
Hatalmas teljesítmény



HONOR 9X<sup>PRO</sup>

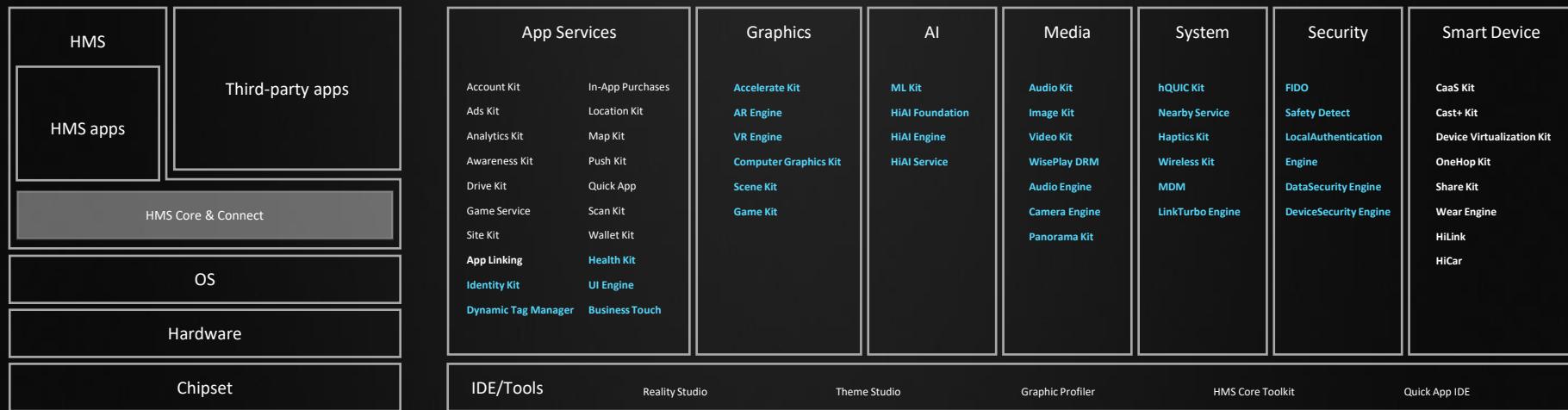
5000 mAh akkumulátor  
Akár 2 napos használat egyetlen töltéssel



HONOR 9A

# Huawei Mobile Services

## HMS Core 5.0



# Five HMS basic service engines nurture global developer innovation



## Payment engine

61% increase in overseas apps that have integrated this kit



## Ads engine

3,100+ overseas apps have integrated this kit



## Browsing engine

330 million MAUs  
Information feeds for 120+ countries/regions



## Map engine

2,000+ overseas apps have integrated this kit



## Search engine

20+ vertical sectors  
3,000+ partners

The data comes from Huawei's internal statistics as of August 31, 2020.



## Map engine



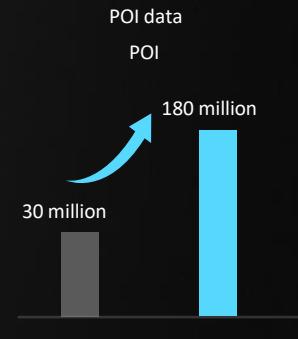
Website



Mobile device



Cross-platform



The data comes from Huawei's internal statistics as of August 31, 2020.



## ML Kit

Provides text, voice, language, image, facial, and biometric-related AI services



### HiAI Foundation

Optimizes performance through heterogeneous scheduling and NPU acceleration



### HiAI Engine

Incorporates AI capabilities into apps, for example, document skew correction, image super-resolution, and intelligent recognition



### HiAI Service

Aggregates developer content and services for instant access

## ML Kit



### Facial/Body Recognition

Detection speed: 70%↑



### Text Recognition

Recognition rate : up to 99%



### Speech/Language

Speech recognition rate: higher than 95%  
Text-to-speech MOS: higher than 4.2



### Custom Model

AI inference speed: 20%↑



### MT

Translation fluency score:  
higher than 4.2

The data comes from Huawei's Lab.

# Mobil alkalmazásfejlesztői verseny

- Ütemezés:
  - > Ötletek leadása: HF specifikáció ☺
  - > Feltöltés határideje: NagyHF határidő ☺
- Nevezés, feltöltés:
  - > Tárgy oldalon külön űrlap
- Értékes díjak!

# Miről volt szó az előző órán?

- Perziszter
  - > SQLite,
  - > SharedF
  - > File
- Futási ide
- ContentP
- Az Intent



# Miről volt szó az előző órán?

- Perzisztens adattárolási lehetőségek
  - > SQLite, ORM
  - > SharedPreferences
  - > File
- Futási idejű engedélyek
- ContentProvider
- Az Intent fogalma

# Kvíz!

QUIZ  
TIME!

<https://kahoot.it/>



# Tartalom

- BroadcastReceiver
- Hálózati kommunikáció
- NFC
- Bluetooth
- TCP/IP, UDP
- HTTP(s) kapcsolatok kezelése
  - > Retrofit
- BaaS
- Push értesítések
- Service komponens
- Érdekességek, Android újdonságok

# BroadcastReceiver komponens

# Broadcast események

- Rendszerszintű eseményekre fel lehet iratkozni – Broadcast Üzenet
- Az Intent alkalmas arra hogy leírja az eseményt
- Sok beépített Broadcast Intent, lehet egyedi is

`ACTION_TIME_TICK`

`ACTION_TIME_CHANGED`

`ACTION_TIMEZONE_CHANGED`

`ACTION_BOOT_COMPLETED`

`ACTION_PACKAGE_ADDED`

`ACTION_PACKAGE_CHANGED`

`ACTION_PACKAGE_REMOVED`

`ACTION_PACKAGE_RESTARTED`

`ACTION_PACKAGE_DATA_CLEARED`

`ACTION_UID_REMOVED`

`ACTION_BATTERY_CHANGED`

`ACTION_POWER_CONNECTED`

`ACTION_POWER_DISCONNECTED`

`ACTION_SHUTDOWN`

# Broadcast események

- Nem csak az Android, hanem alkalmazások (Activity-k és Service-ek) is dobhatnak Broadcast Intentet
  - > Telephony service küldi az ACTION\_PHONE\_STATE\_CHANGED Broadcast Intentet, ha a mobilhálózat csatlakozás megváltozott
  - > android.provider.Telephony.SMS\_RECEIVED
  - > Sok más, érdemes tájékozódni ha valamit szeretnénk lekezelní
  - > Saját alkalmazásunkból is dobhatunk a **sendBroadcast (String action)** metódussal
  - > Ez is Intent, lehet Extra és Data része

# Broadcast intentek elkapása

- Broadcast Receiver nevű komponens segítségével
  - > Kód ból vagy manifestben kell regisztrálni
    - (bizonyos Action-ök esetén nem mindegy, tájékozódni!, pl. TIME\_TICK)
  - > Intent filterrel állíthatjuk be hogy milyen Intent esetén aktivizálódjon
- Nem Activity, nincs felhasználói felülete
- Azonban képes Activity-t indítani
- Használata: BroadcastReceiver osztályból származtatunk, és felüldefiniáljuk az onReceive() metódust, majd intent-filter

# Broadcast intentek kezelése

```
class OutgoingCallReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val outNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER)
        Toast.makeText(context, outNumber, Toast.LENGTH_LONG).show()
    }
}
```

## AndroidManifest.xml:

```
<receiver android:name=".OutgoingCallReceiver">
    <intent-filter>
        <action android:name=
            "android.intent.action.NEW_OUTGOING_CALL"/>
    </intent-filter>
</receiver>
```

# Broadcast intentek kezelése

Broadcast továbbdobásának megakadályozása:

- `abortBroadcast();`

Például ha a fülhallgató média gombjait kell kezelni és nem akarjuk, hogy a zenelejátszó is megkapja a Broadcast-ot ☺

# Gyakoroljunk!

- Készítsünk egy AirPlane mód változásra figyelő BroadcastReceivert!
- Készítsünk egy kimenő hívásra figyelő BroadcastReceivert!
  - > Változtassuk meg a hívott számot!
  - > Egészítsük ki SMS figyeléssel!
    - Valósítsuk meg, hogy ne kerüljön be inbox-ba a bejövő SMS

# SMS Receiver 1/2

- Manifest:

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />  
...  
<receiver android:name=".SMSReceiver" android:enabled="true">  
    <intent-filter android:priority="1000">  
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>  
    </intent-filter>  
</receiver>
```

# SMS Receiver 2/2

```
class SMSReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val extras = intent.extras ?: return
        val pdus = extras.get("pdus") as Array<ByteArray>
        for (pdu in pdus) {
            val msg = SmsMessage.createFromPdu(pdu)
            val origin = msg.originatingAddress
            val body = msg.messageBody
            Toast.makeText(context,
                "SMS caught, number: $origin body: $body", Toast.LENGTH_LONG)
                .show()
        }
    }
}
```

# Alkalmazáskomponens indítása Boot után

- Néha olyan szolgáltatásokra van szükség, amelyek minden futnak a készüléken
- Ilyen esetben fontos, hogy a készülék indítása esetén ezek automatikusan is el tudjanak indulni
- Az Android lehetőséget biztosít arra, hogy feliratkozzunk a „*Boot befejeződött*” eseményre és valamilyen alkalmazás komponenst elindítsunk:
  - > *BroadcastReceiver* definiálása Manifest-ben
  - > `android.intent.action.BOOT_COMPLETED`
- A *BroadcastReceiver onReceive()* függvényében elindíthatjuk a megfelelő komponenst

# Hálózati kommunikáció

# Hálózati kommunikáció Android platformon

- „Rövid” távú kommunikáció
  - > NFC
  - > Infra
  - > Bluetooth
  - > WiFi Direct / WiFi P2P (\*)
- „Hosszabb távú” / internet alapú kommunikáció
  - > UDP
  - > TCP/IP, Socket, ServerSocket
  - > HTTP (REST)
  - > SOAP WebService (nem ajánlott!, kSoap2 osztálykönyvtár)

# Near Field Communication (NFC)

- Rövidtávú vezeték nélküli kommunikációs technológia
- <4cm távolságon belül működik
- NFC tag és mobil telefon közti kis méretű adatátvitel (payload)
- Mobil telefon és mobil telefon közti kis méretű adatátvitel
- NFC Forum által meghatározott formátum:  
NDEF (NFC Data Exchange Format)

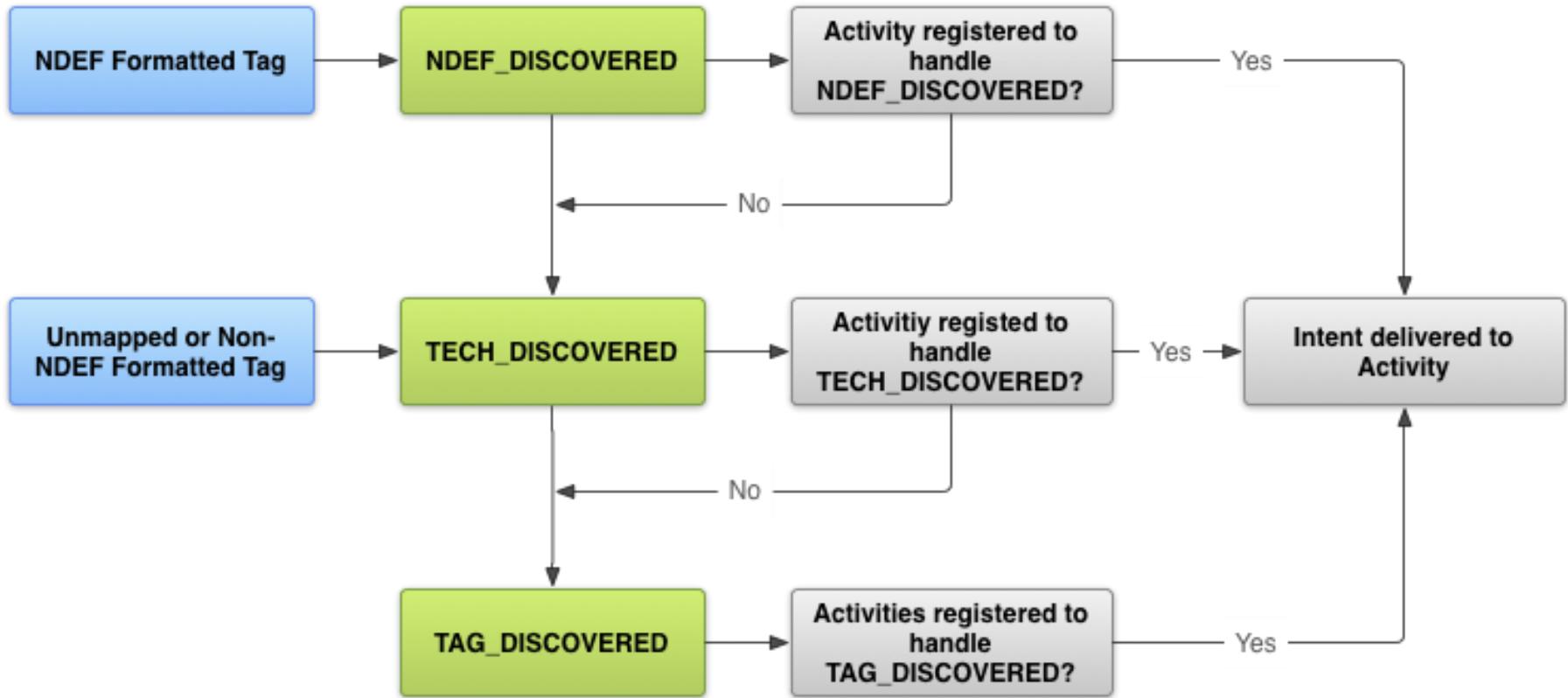
# NFC Tag-ek jellemzői

- Írható/olvasható/egyszer írható
- Komplex Tag-ek tartalmazhatnak matematikai műveleteket is és lehet külön kriptográfia hardver egységük authentikáció céljából
- Még bonyolultabb Tag-ek akár saját működési környezettel is rendelkezhetnek és egyszerűbb kód végrehajtására is alkalmasak

# NFC Beam

- NDEF üzenet küldése két Android készülék között
- Nincs szükség készülék felderítésre
- A kapcsolat automatikusan felépül, ha két eszköz hatótávon belülre kerül
- Beépített alkalmazások is támogatják adatcserére (pl.: Browser, Névjegyzék, YouTube stb.)

# Tag Dispatch System működése



# NFC használat lépései

- Permission
  - > <uses-permission android:name="android.permission.NFC" />
- Minimum Android verzió
  - > <uses-sdk android:minSdkVersion="10"/>
- NFC hardware ellenőrzése (így nem jelenik meg azon eszközök számára, amik nem támogatják az NFC-t)
  - > <uses-feature android:name="android.hardware.nfc" android:required="true" />

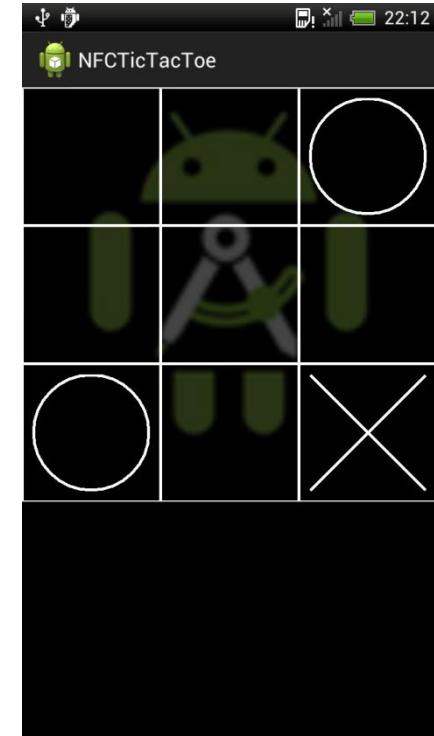
# Szöveges tartalom detektálása

- Manifest-en belül a megfelelő `<activity>`-elembe:

```
<intent-filter>  
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="text/plain" />  
</intent-filter>
```

# NFC TicTacToe

- Készítsünk egy TicTacToe alkalmazást, ahol az NFC tag a játék állását és a következő játékoszt tárolja.
- A Tag érzékelésekor az alkalmazás automatikusan induljon el és jelenítse meg a játék állását, valamint egy lépést engedjen lépni.
- Sikeres lépés után az állást mentse rá az alkalmazás a Tag-ra.
- A játék végén üres állást mentsünk a Tag-ra.
- Extra: tárolja a Tag az X-el és O-val nyert játékok számát globálisan.
  - > [https://github.com/peekler/Android-BME/tree/master/10\\_Multimedia\\_Extras/NFCTicTacToe AS](https://github.com/peekler/Android-BME/tree/master/10_Multimedia_Extras/NFCTicTacToe_AS)



# Android Bluetooth APIk

- Hagyományos Bluetooth kommunikáció
  - > Részletes Bluetooth API-k
- Bluetooth LE támogatás
  - > Bluetooth LE API

# Legfontosabb Bluetooth osztályok

- *BluetoothAdapter*: felderítés, párosítások lekérdezése, *BluetoothDevice* példányosítása, server socket létrehozás
- *BluetoothDevice*: távoli Bluetooth eszközt jelképezi
- *BluetoothSocket/BluetoothServerSocket*
- *BluetoothClass*: eszköz tulajdonságai (read-only)
- *BluetoothProfile*: profil jellemzői
- Chat példa:
  - > <https://github.com/android/connectivity-samples/tree/master/BluetoothChat>

# TCP/IP Socket

- Szabványos *Socket* implementáció
- Jól ismert *java.net.Socket* osztály a kapcsolatok megnyitására
- *java.net.ServerSocket* osztály a bejövő kapcsolatok fogadására
  - > Localhoston az alkalmazások egymás közti kommunikációja is megoldható
- *InputStream* és *OutputStream* támogatás az adatok olvasására és írására

# Socket példa

```
val socket = Socket("192.168.2.112", 8787)
val inputStream = socket.getInputStream()
val isr = InputStreamReader(
    inputStream,
    "UTF-8"
)
val resultBuffer = StringBuilder()
var inChar: Int
while ((inChar = isr.read()) != -1) {
    resultBuffer.append(inChar.toChar())
}
val result = resultBuffer.toString()
// result kezelése
// ...
inputStream.close()
socket.close()
```

# UDP Üzenetek küldése

```
val msg = "UDP Test"
val socket = DatagramSocket()
// In case of broadcast
socket.setBroadcast(true)
// InetAddress localAddr =
//   InetAddress.getByName("192.168.0.110");
val message = msg.toByteArray()
val p = DatagramPacket(
    message,
    msg.length, localAddr, 10100
)
socket.send(p)
```

# UDP Üzenet fogadása

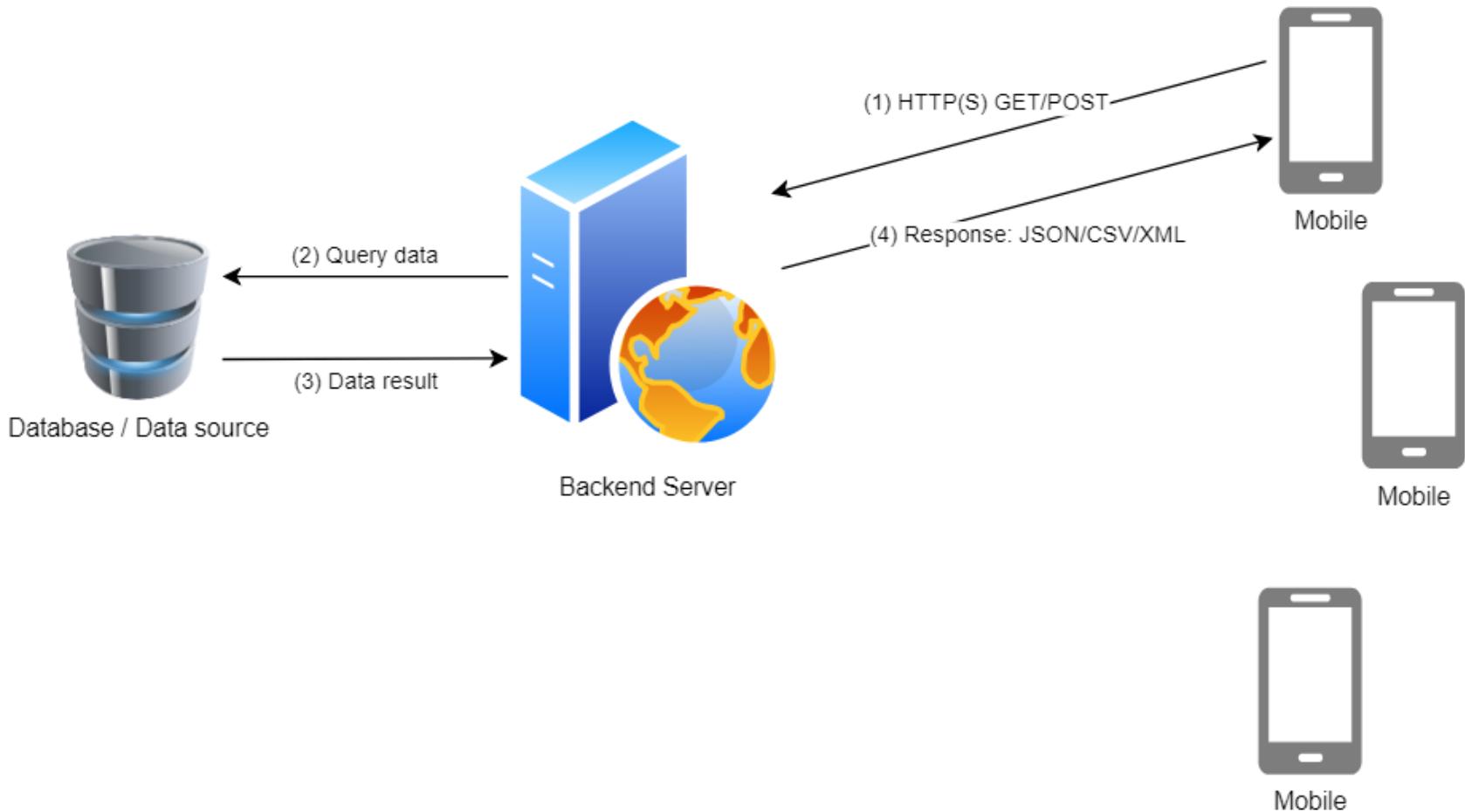
```
val message = ByteArray(1500)
val packet = DatagramPacket(message, message.size)
val socket = DatagramSocket(10100)
socket.receive(packet)
val msg = String(
    message, 0, packet.getLength()
)
Log.d("MYTAG", "received: $msg")
socket.close()
```

# TCP/IP és UDP library: Kryonet

- Java könyvtár TCP és UDP kliens-szerver kommunikációhoz
- Kryo sorosító könyvtár
  - > Objektumok küldése hálózaton keresztül
- Mobil és asztali környezet támogatása
- Hatékony
- Játékokhoz kiváló
- Szolgáltatás felderítést támogat
- <https://github.com/EsotericSoftware/kryonet>

# HTTP(s) kommunikáció

# 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:
  - > <http://www.w3.org/Protocols/HTTP/HTRESP.html>
- HTTP REST
  - > [http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)
- Ügyeljünk az alapos hibakezelésre
- HTTP GET példa:
  - > <http://numbersapi.com/3/math>

# Példa API

><http://api.openweathermap.org/data/2.5/weather?q=Budapest&units=metric&appid=f3d694bc3e1d44c1ed5a97bd1120e8fe>

# URL felbontása

> Host: http://api.openweathermap.org/

> Path: data/2.5/weather

> Params: ?  
q=Budapest  
&  
units=metric  
&  
appid=f3d694bc3e1d44c1ed5a9  
7bd1120e8fe

# 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.getInputStream()))
        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.getOutputStream()
        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")
```

# UI módosítása más szálból

- Az alkalmazás indításakor a rendszer létrehoz egy úgynévezett *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* – *deprecated*
  - > Külső libek, pl. *EventBus* v. *Otto*
  - > *Retrofit*
  - > *Coroutine - Kotlin*

# AsyncTask példa

```
class AsyncTaskUploadVote : AsyncTask<String, Void, String>() {  
  
    override fun onPreExecute() {  
        // ...  
    }  
  
    override fun doInBackground(vararg params: String): String? {  
        val result = null  
        // Hálózati kommunikáció, válasz mentése result-ba  
        return result  
    }  
  
    override fun onPostExecute(result: String?) {  
        // Eredmény használata UI szálón  
        Log.d("RESULT", result)  
    }  
  
}  
  
AsyncTaskUploadVote().execute("Yes")
```

# Tipikus üzenet formátumok

- CSV, TSV
- XML
  - > Beépített API/parser
  - > SimpleXML külső könyvtár
- JSON:
  - > Beépített API/parser
  - > Moshi, GSON külső könyvtár
- REST API tesztelésére:
  - > Chrome alkalmazás: PostMan

# JSON

- Structural characters: '{', '}', '[', ']', '.', ','
- Example:

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "age": 25,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": 10021  
    },  
    "phoneNumber": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "fax",  
            "number": "646 555-4567"  
        }  
    ]  
}
```

# 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

# Külső osztálykönyvtárak XML és JSON feldolgozásra

- XML:
  - > SimpleXML
- JSON:
  - > Moshi vagy GSON
  - > <http://www.jsonschema2pojo.org/>
- REST API tesztelésére:
  - > PostMan (Chrome plugin)

# JSON – Kotlin osztály

```
{  
    "base": "USD",  
    "rates": {  
        ...  
        "HUF": 289.8969072165,  
        "EUR": 0.896458987  
    },  
    "date": "2019-05-04"  
}
```



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

```
data class Rates(  
    ...  
    val HUF: Number?,  
    val EUR: Number?  
)
```

# 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>

# Moshi POJO példa

```
@JsonClass(generateAdapter = true)
class MoneyResult(val rates: Rates?, val base: String?, val date: String?)

@JsonClass(generateAdapter = true)
class Rates(val CAD: Double?, val HKD: Double?, val ISK: Double?, ...)
```

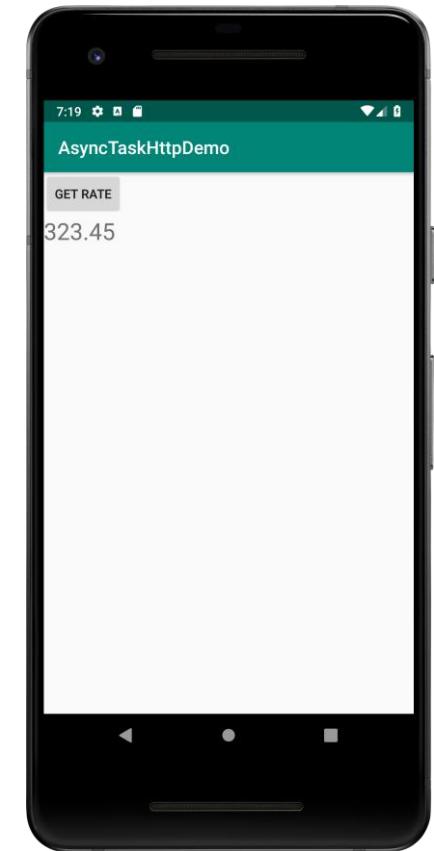
# REST libraryk

- Volley
- RESTLet
- OkHttp
- Retrofit
  - > OkHttp-re épül

# REST API példa

# Példa

- <https://api.exchangeratesapi.io/latest?base=EUR>
- Retrofit használata



# Retrofit

<https://api.exchangeratesapi.io/latest?base=EUR>

- 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:2.9.0'
- További információk:
  - > <http://square.github.io/retrofit/>

# Retrofit – Moshi támogatás

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

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

- Gradle:

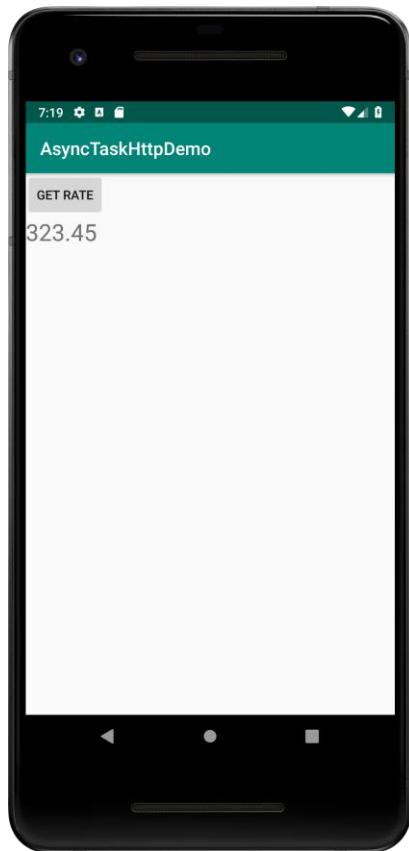
```
def retrofit_version = '2.8.1'  
implementation "com.squareup.retrofit2:retrofit:$retrofit_version"  
implementation "com.squareup.retrofit2:converter-moshi:$retrofit_version"  
def moshi_version = '1.9.2'  
implementation "com.squareup.moshi:moshi:$moshi_version"  
kapt "com.squareup.moshi:moshi-kotlin-codegen:$moshi_version"
```

- További információk:

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

# Példa - Retrofit

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



# Gradle

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
apply plugin: 'kotlin-kapt'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "hu.ait.httpmoneydemo"
        minSdkVersion 26
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

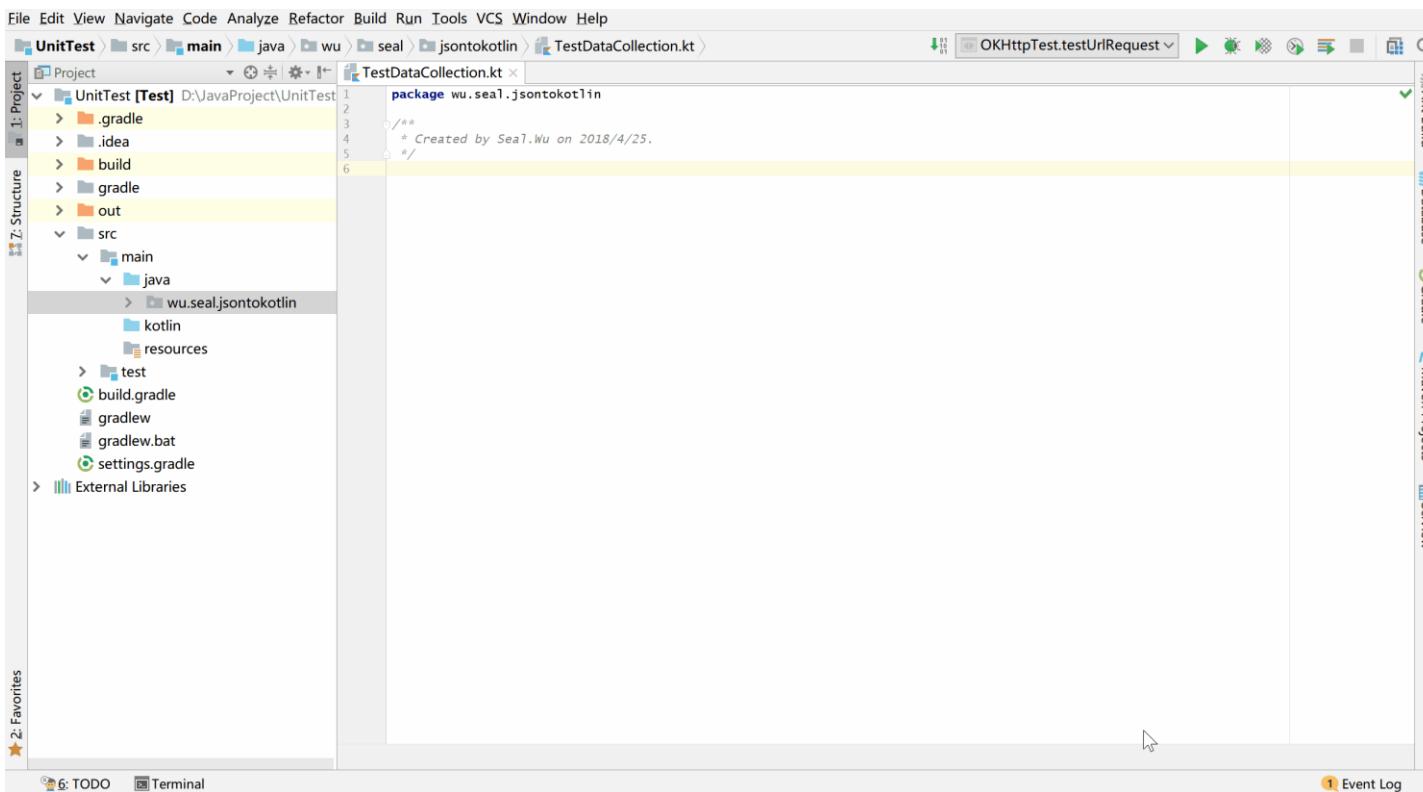
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    ...

    compileOptions {
        sourceCompatibility 1.8
        targetCompatibility 1.8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}
dependencies {
    ...
    def retrofit_version = '2.9.0'
    implementation "com.squareup.retrofit2:retrofit:$retrofit_version"
    implementation "com.squareup.retrofit2:converter-moshi:$retrofit_version"
    def moshi_version = '1.9.2'
    implementation "com.squareup.moshi:moshi:$moshi_version"
    kapt "com.squareup.moshi:moshi-kotlin-codegen:$moshi_version"
}
```

# 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

```
@JsonClass(generateAdapter = true)
class MoneyResult(val rates: Rates?, val base: String?, val date: String?)

@JsonClass(generateAdapter = true)
class Rates(val CAD: Double?, val HKD: Double?, val ISK: Double?, val PHP: Double?,
           val DKK: Double?, val HUF: Double?, val CZK: Double?, val AUD: Double?, val RON: Double?,
           val SEK: Double?, val IDR: Double?, val INR: Double?, val BRL: Double?, val RUB: Double?,
           val HRK: Double?, val JPY: Double?, val THB: Double?, val CHF: Double?, val SGD: Double?,
           val PLN: Double?, val BGN: Double?, val TRY: Double?, val CNY: Double?, val NOK: Double?,
           val NZD: Double?, val ZAR: Double?, val USD: Double?, val MXN: Double?, val ILS: Double?,
           val GBP: Double?, val KRW: Double?, val MYR: 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(MoshiConverterFactory.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()
            }
        })
    }
}
```

# Push notification - Android FCM

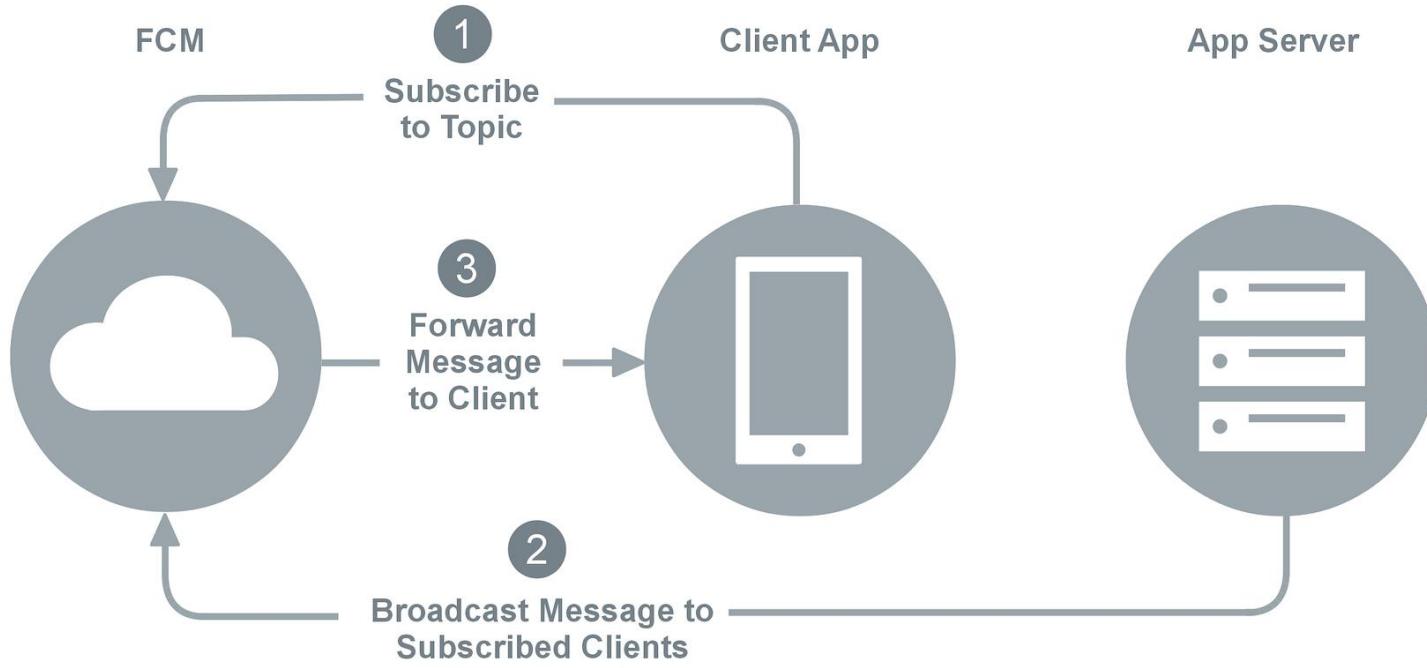
- Sokszor szükség lehet rá, hogy a szerver tájékoztassa a mobil klienseket valamilyen eseményről
- Jelenlegi eszközünk:
  - > Poll-ozás
  - > Kliens időközönként lekérdezi a szervertől, hogy van-e számára üzenet
  - > Lassú és nem real-time
- Fordított irány: valahogy a szervernek kéne tájékoztatni a klienseket
- Megoldás: FCM (Firebase Cloud Messaging)
- Szerver és kliens oldali implementáció szükséges
- Regisztráció és további információk a használati feltételekről:
- <https://firebase.google.com/docs/cloud-messaging>



# Android FCM

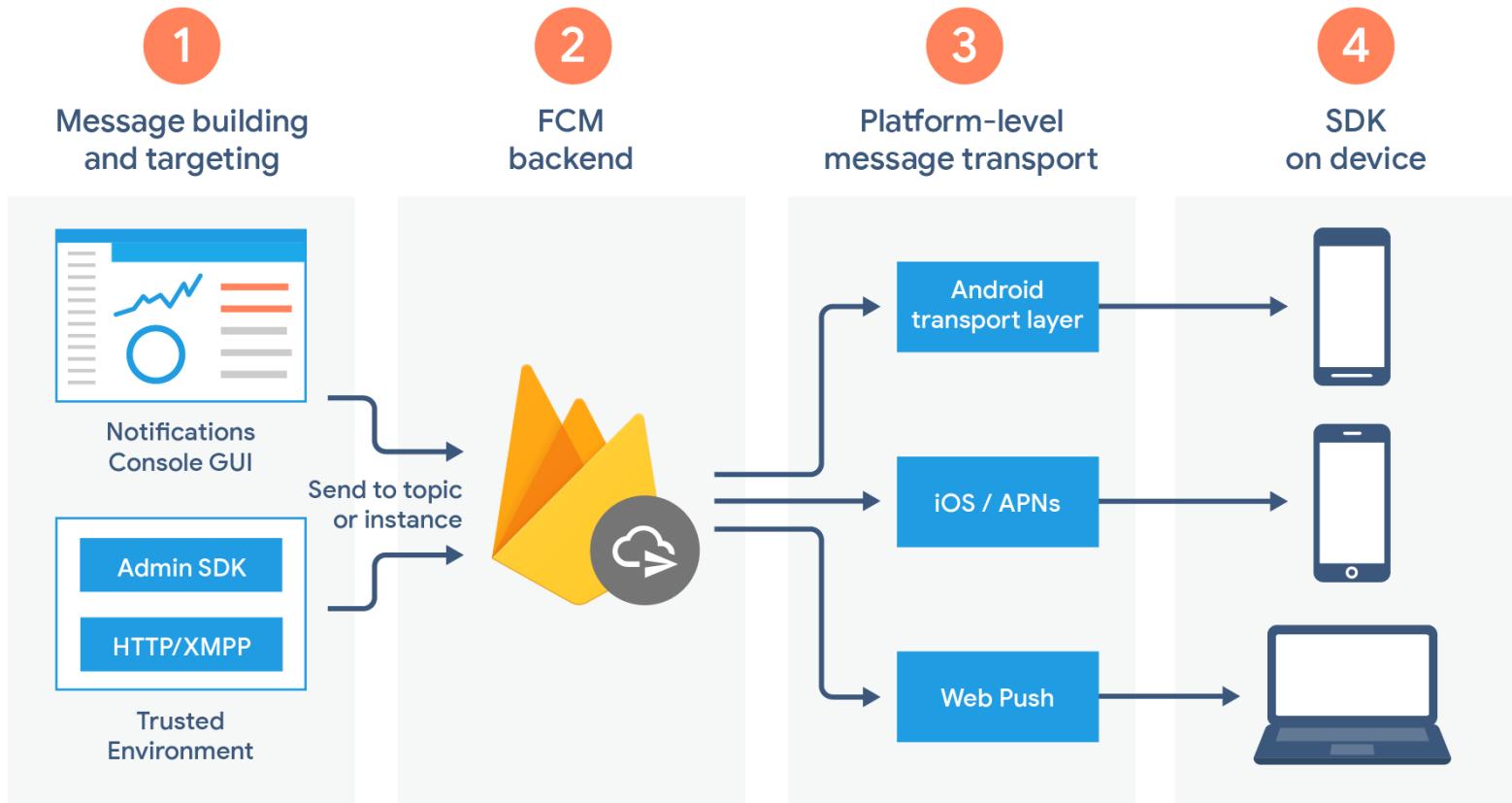
- Cél: szerver -> kliens kommunikáció
- Rövid üzenetek (~4kb) a kliensek értesítésére
- Ingyenes használat
- FCM komponensek:
  - > Android készülék
  - > Application server
  - > FCM cloud

# Push értesítés folyamata



**Simplify - Firebase Cloud Messaging (FCM)**

# FCM általános architektúra



# BaaS jellemzők

- ~2011 körül jelentek meg először
- Szerver és háttér feladatok automatikus ellátása
- Mára sokkal többet tudnak
- Széles körű mobil támogatás, mobil fejlesztők „tehermentesítése”
- Néhány népszerű BaaS szolgáltatás:
  - > FireBase, Backendless, Buddy, StackMob, Kinvey, Appcelerator Cloud Service, Kumulos, Google Mobile Backend Starter
- Árazás példa:
  - > <https://firebase.google.com/pricing/>
  - > <https://backendless.com/pricing/>

# BaaS képességek

- Felhasználó kezelés
- Perzisztencia, táblák, backup
- Offline működés támogatása
- File kezelés
- Verzió kezelés
- Analytics
- Kód generálás
- Dinamikus kód futtatás
- Media streaming
- Geolocation
- Social Networks
- Push értesítés

# BaaS példák

- Firebase: <https://firebase.google.com/>
- Backendless: <http://backendless.com/>
- Kinvey: <http://www.kinvey.com/>
- Kumulos: <http://www.kumulos.com/>
- Buddy: [http://buddy.com/developers/#Platform\\_Overview](http://buddy.com/developers/#Platform_Overview)
- Google Mobile Backend:  
<https://developers.google.com/cloud/samples/mbs/>

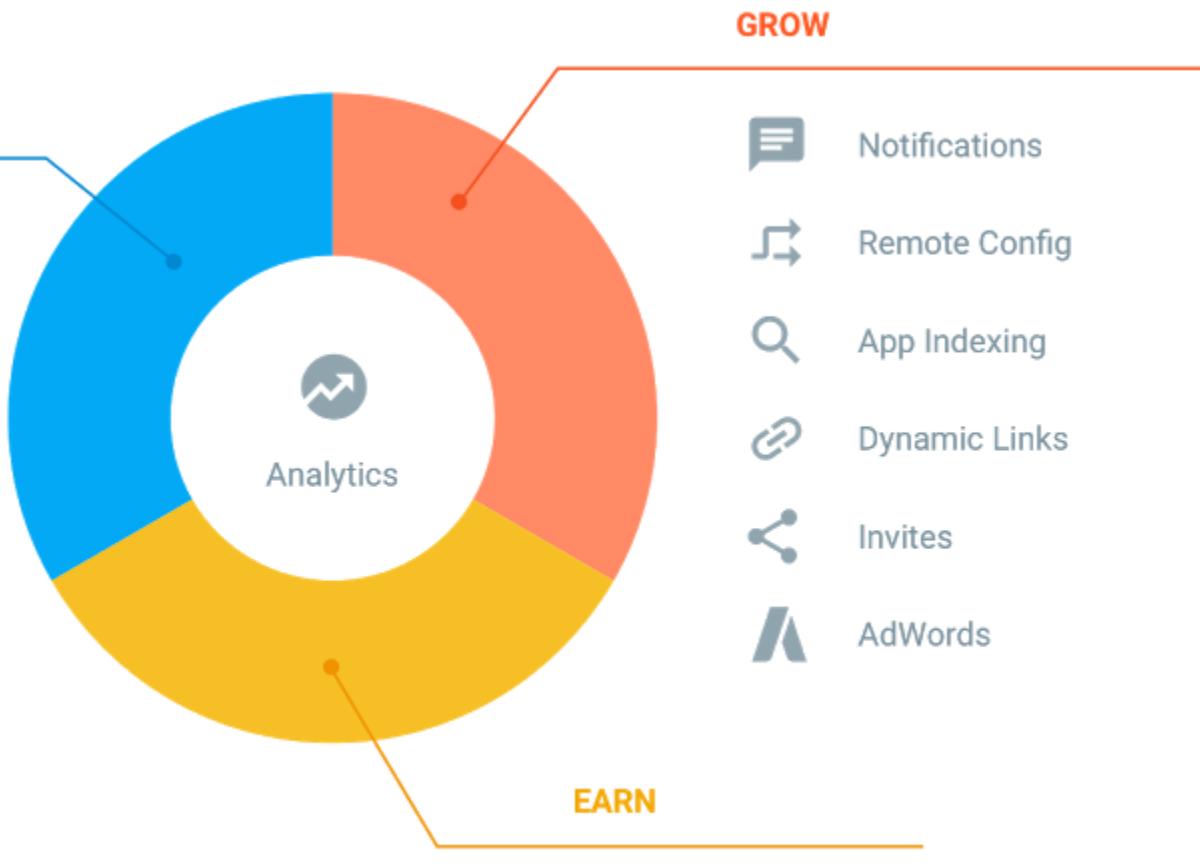
# BaaS hátrányok

- Félelem a felhő szolgáltatásokkal szemben
- „Lock-in” effektus:
  - > Körülményes, ha váltani szeretnénk egyik felhő szolgáltatásról a másikra, nincs interfész a kettő között
- BaaS specifikus UI elemek használata, pl. Parse UI elemek; váltáskor probléma
- Adat szuveneritás (EU vs. USA)
- Számlázás

# Firebase képességek

## DEVELOP

-  Realtime Database
-  Authentication
-  Cloud Messaging
-  Storage
-  Hosting
-  Test Lab
-  Crash Reporting



# Firebase SignIn példa

```
 FirebaseAuth.getInstance().signInWithEmailAndPassword(
    etEmail.text.toString(), etPassword.text.toString()
).addOnCompleteListener {
    if (it.isSuccessful) {
        startActivity(Intent(this@LoginActivity, MainActivity::class.java))
    } else {
        Toast.makeText(this@LoginActivity, "Error: "+ 
            it.exception?.message,
            Toast.LENGTH_SHORT).show();
    }
}.addOnFailureListener {
    Toast.makeText(this@LoginActivity,
        "Error: ${it.message}",
        Toast.LENGTH_SHORT).show();
}
```

# Service komponens

# Service bemutatása

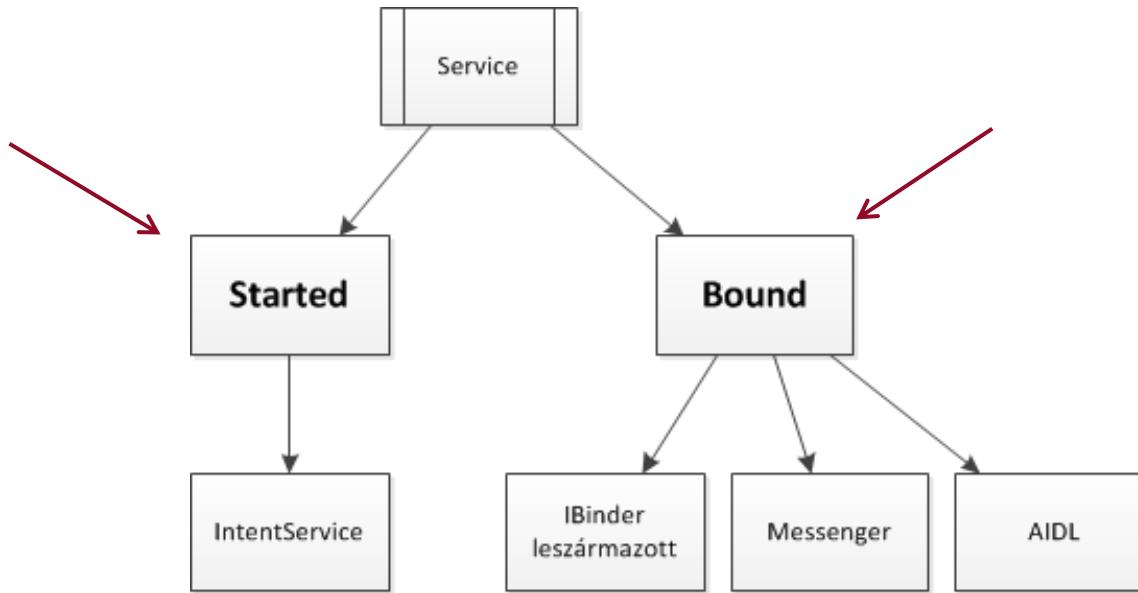
- Valamilyen hosszabb ideig tartó, háttérben futó feladatot lát el
- Felhasználói beavatkozás nélkül működik
- Nincs felhasználói felülete
- Más komponensek/alkalmazások számára is szolgáltathat funkciókat
- Bármilyen alkalmazáskomponens elindíthatja és **akkor is futva maradhat, amikor a hívó (pl. Activity) megáll,** tipikusan ha más alkalmazásra váltunk

# Service bemutatása

- Más alkalmazások komponensei is kapcsolódhatnak hozzá és kommunikálhatnak vele (*Interprocess Communication, IPC*)
- (ahogy egy Activity is elindítható másik alkalmazásból, de a **Service**-nél ez letiltható)
- Ugyanúgy **Intent**-el indítjuk el
- Példa szolgáltatások:
  - > Zenejátszó
  - > Hosszabb hálózati kommunikáció (pl. torrent)
  - > GPS pozíció követés - rendszerszolgáltatás
  - > Stb...

# Service típusok

- Egy Service kétféle módon képes működni:
  - > Komponensből indított / *Started*
  - > Kapcsolt / *Bound*



# Service működési típusok

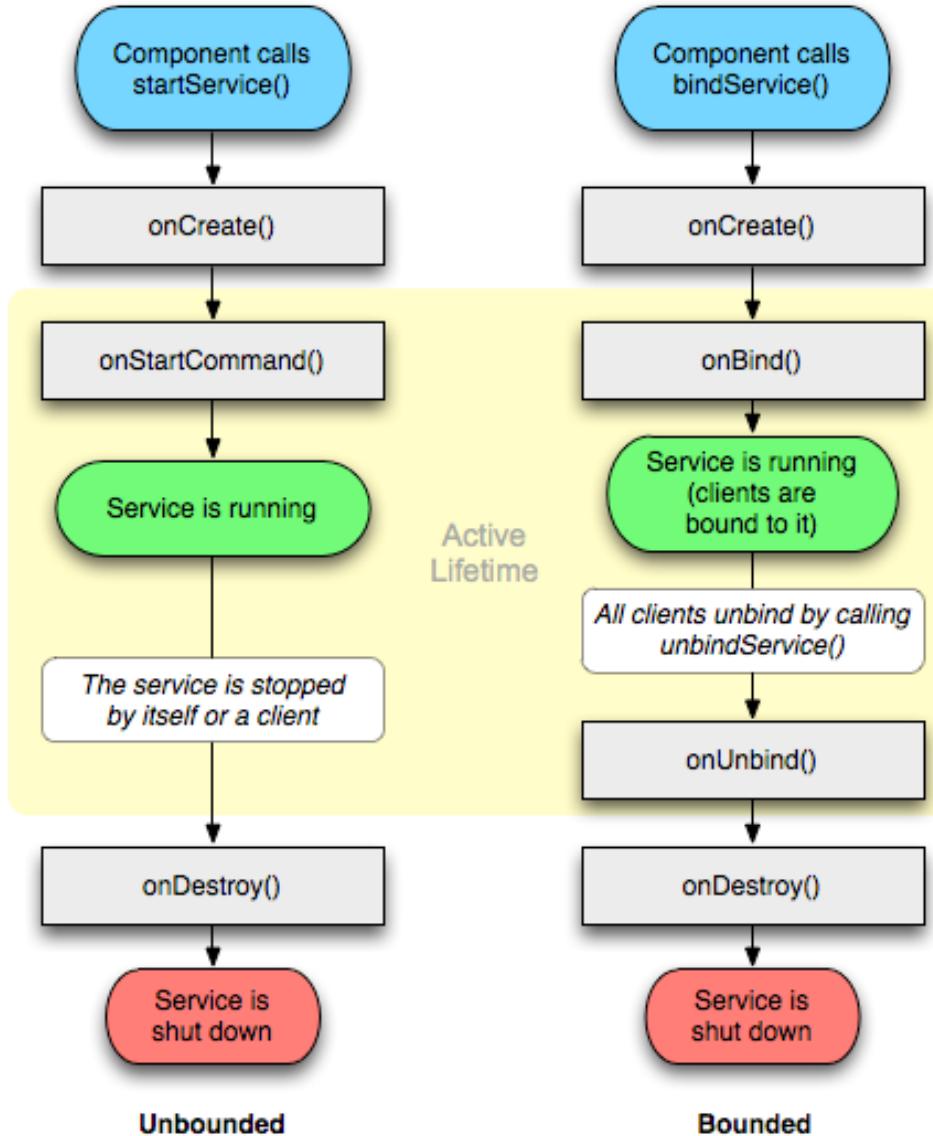
## 1. Komponensből indított (*Started/Unbound*):

- > Valamilyen alkalmazás komponens (tipikusan Activity) elindítja a ***startService(intent)*** metódussal
- > A Service akkor is folytathatja a futását, amikor a hívó komponens már megsemmisült
  - A hívó megállítása nem törli a Service-t is!
- > Általában az így indított Service-ek **egy feladatot hajtanak végre**, nem folytonos szolgáltatást nyújtanak (például egy darab fájl feltöltése vagy letöltése)
- > Csak a hívónak van rá referencia
- > Ha végzett, **magát kell leállítania** a *stopSelf()* metódussal vagy a hívónak *stopService(intent)*-el, az op.rendszer nem fogja!

# Service megvalósítás

- Service (vagy beépített gyerek) osztályból származtatunk
- Megvalósítjuk a megfelelő callback függvényeket:
  - > Ha *Started*-ként (is) akarjuk használni: *onStartCommand()* implementálása
  - > Ha *Bound*-ként (is) akarjuk használni: *onBind()* implementálása

# Service életciklus modell



# Műveletek végrehajtása Service-n belül

- A Service alapértelmezetten a processze fő szálában fut, **nem kap külön szálat!!**
- Erőforrás igényes műveletek esetén „kézzel” kell új szálat indítanunk, pl.:
  - > CPU intenzív feladatok (pl. titkosítás, enkódolás, zene lejátszás)
  - > Hálózati kommunikáció, stb...
- Ellenkező esetben 5mp után ugyanúgy megjelenik az *Application Not Responding* (ANR) ablak, mint Activity esetén

# Gyakoroljunk!

- Készítsünk egy Service-t, mely 5 másodpercenként kiírja az aktuális időt!

# ZH információk

- 2020-11-17, K. 8-10
  - > Online
- Igaz-hamis kérdések
- Ábra, architektúra magyarázat
- Összehasonlítás
- Koncepció/működés magyarázata
- XML <-> UI
- Hibakeresés kódban
- Gyakorlati rész (kis alkalmazás fejlesztése)

# További érdekességek

- APK visszafejtés
- iOS-Android crossplatform
- USB API
- CI/CD
- Dependency Injection
- Architecture Components
- Navigation Component
- MotionLayout

# APK Visszafejtés

- Eszközök:
  - > Play Store -> myBackup
  - > Dex2jar
  - > JDGUI
- APK hely:
  - > ..\[projekt  
mappa]\app\build\outputs\apk\debug\app-  
debug.apk
- Megoldás: Obfuszkáció

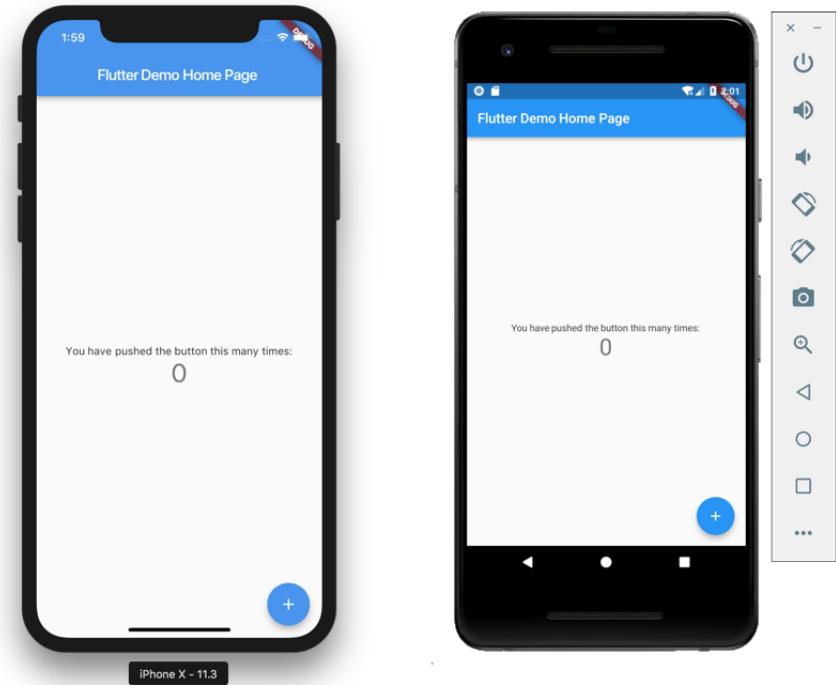
# Crossplatform fejlesztés

- (reszponzív weboldal)
- Flutter
- Xamarin
- React Native
- NativeScript
- ...

# Flutter koncepció

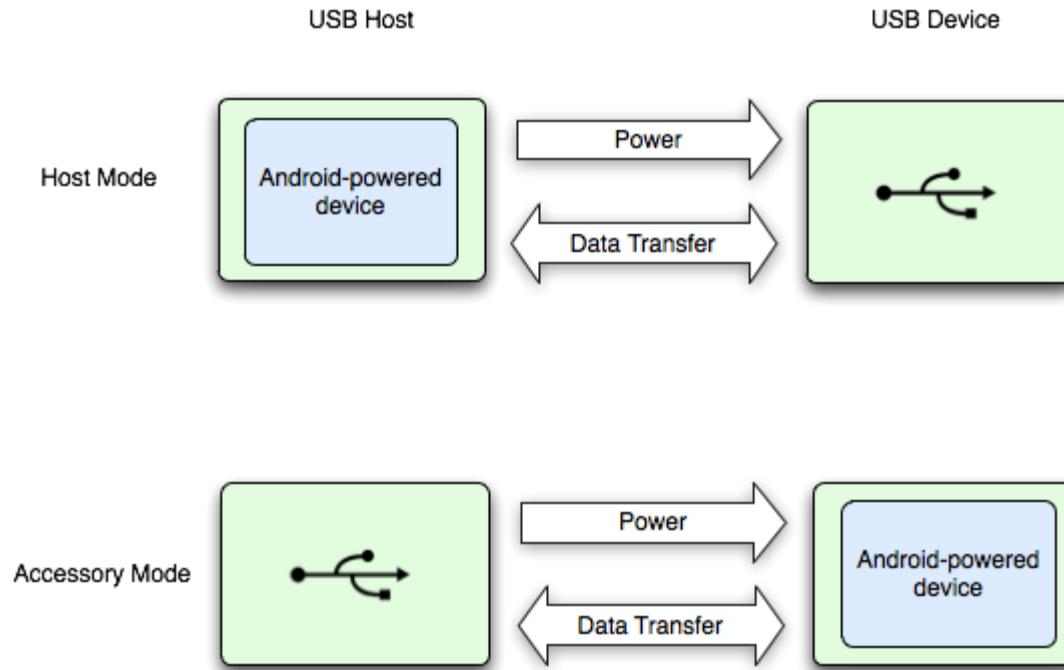
Az alapkoncepció hasonló a játék motorokhoz

- A natív alkalmazás egy nagy **rajzvászon**
- Ebbe rajzolja bele a Flutter alkalmazás, mint pixeleket
- A UI és az üzleti logika is ugyanazon a nyelven
  - > **Dart**



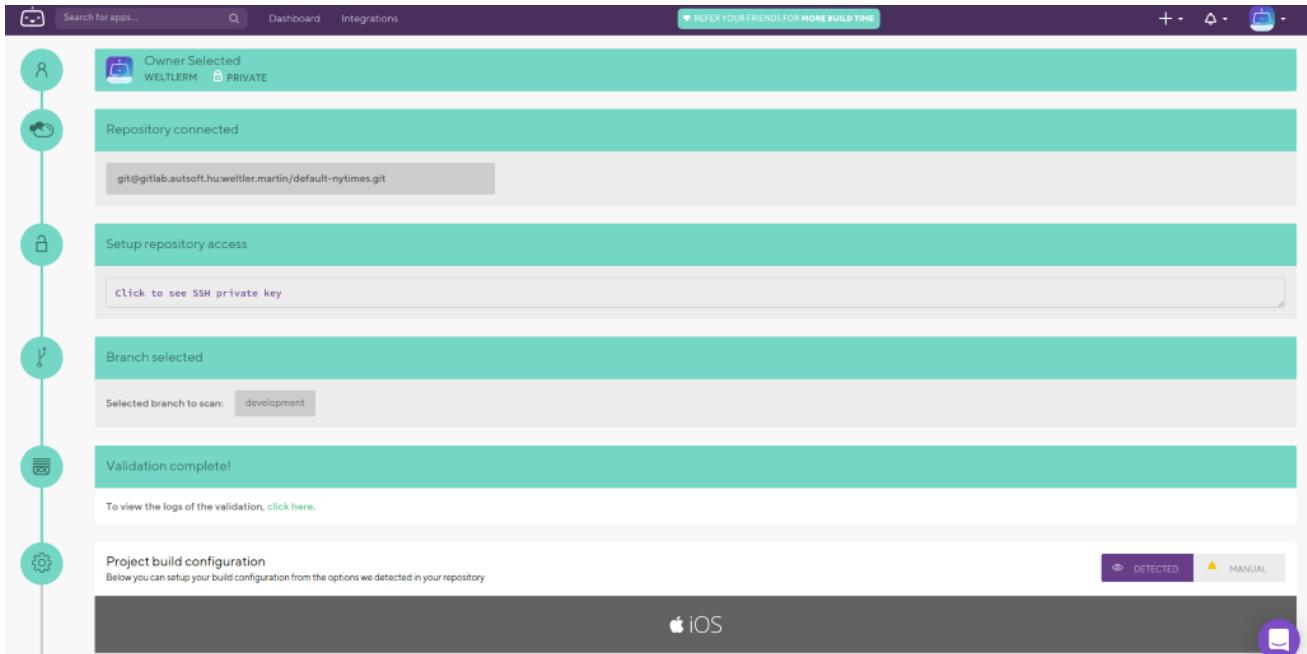
# USB Host API

- <https://developer.android.com/guide/topics/connectivity/usb/host>

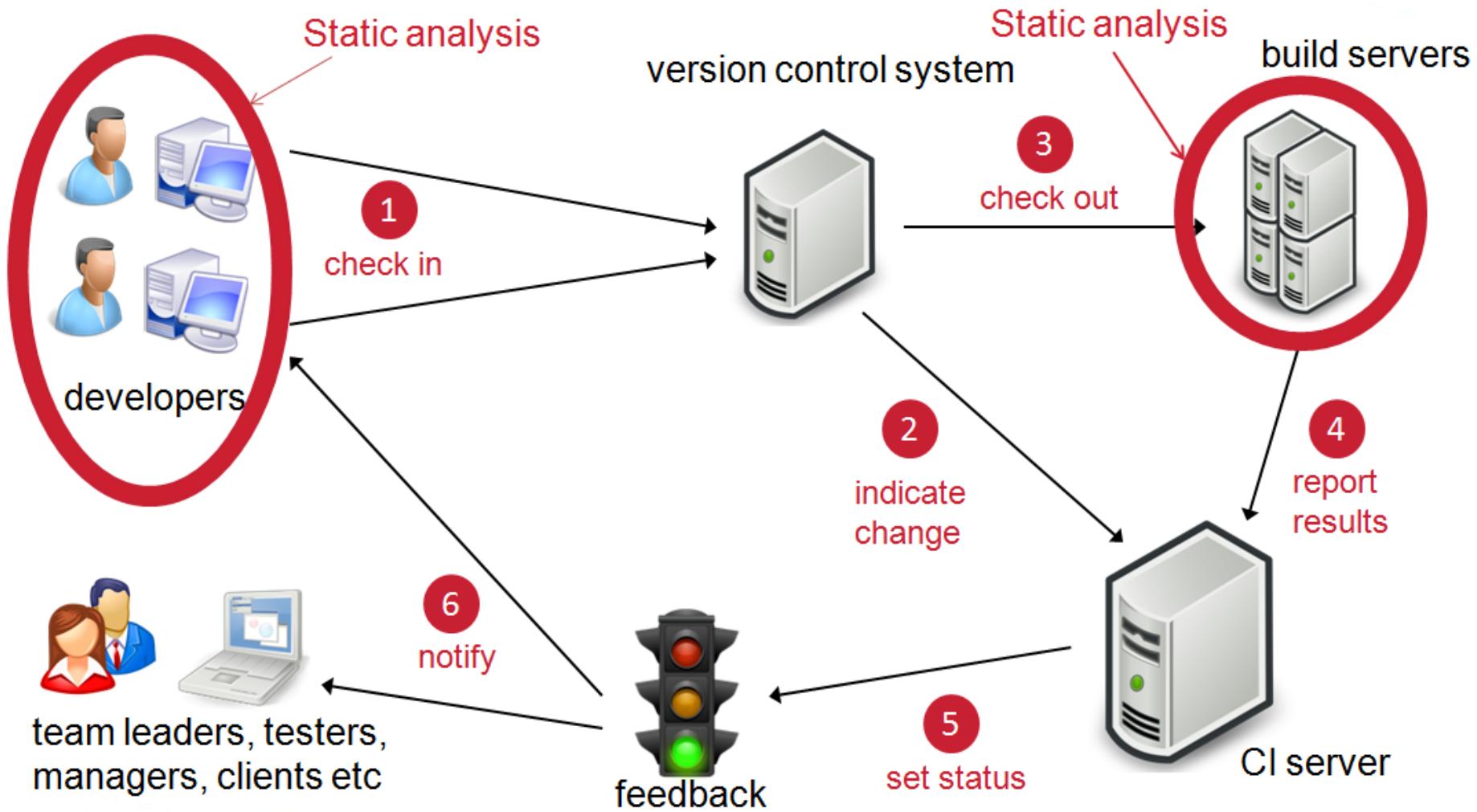


# CI/CD

- Jenkins
- GitLab CI
- Bitrise



# Egy tipikus CI rendszer



# Dependency Injection

- Az objektumot használó objektumnak/komponensnek nem szükséges tudnia az objektum létrehozásáról, annak életciklusáról, mindenzt kezeli a dependency injection framework

# DI alapok

- Üzleti logikát ellátó osztályok, melyeket használunk
  - > EncryptEngine, FaceDetector, stb.
- Segéd osztályok, melyek ezeket felépítik
  - > EncryptEngineFactory, FaceDetectorLoader, stb.
  - > Sokszor boilerplate
  - > Állapot kezelés?
  - > pl. Contextet minden rétegen „át kell fúrni”
- Ne az configuralja aki felhasználja!

# DI előnyei

- Lehetővé teszi, hogy a „fontos” kód részekre koncentrálunk
- Könnyebb tesztelés
  - > Könnyen lecserélhető a „*NetworkApi*” egy „*FakeNetworkApi*”-ra teszt íráskor
- Könnyen készíthető vele újrafelhasználható és kicserélhető modul

# DI előnyei

- Ugyanaz a modul használható az alkalmazás bármelyik részében
- Könnyen igazítható a modul használat a build type (debug/release)-hoz
  - > Pl más loggoló modul debug és release esetén
- Modulok életciklusa transzparent a felhasználó osztályok számára
  - > Scopeok
  - > Singleton

# Hilt - Android

- Tegyük fel, hogy a *NetworkCall* segít valamilyen kommunikáció lebonyolításában és több helyen használjuk, de nem akarjuk tudni hogyan kell létrehozni/mik a függőségei
- Egyszerű használata:

```
class MainActivity : AppCompatActivity() {  
  
    @Inject  
    NetworkCall networkCall;  
  
    ...  
}
```

# De ki tudja hogy jön létre a NetworkCall?

- A modul osztály feladata a NetworkCall létrehozása:

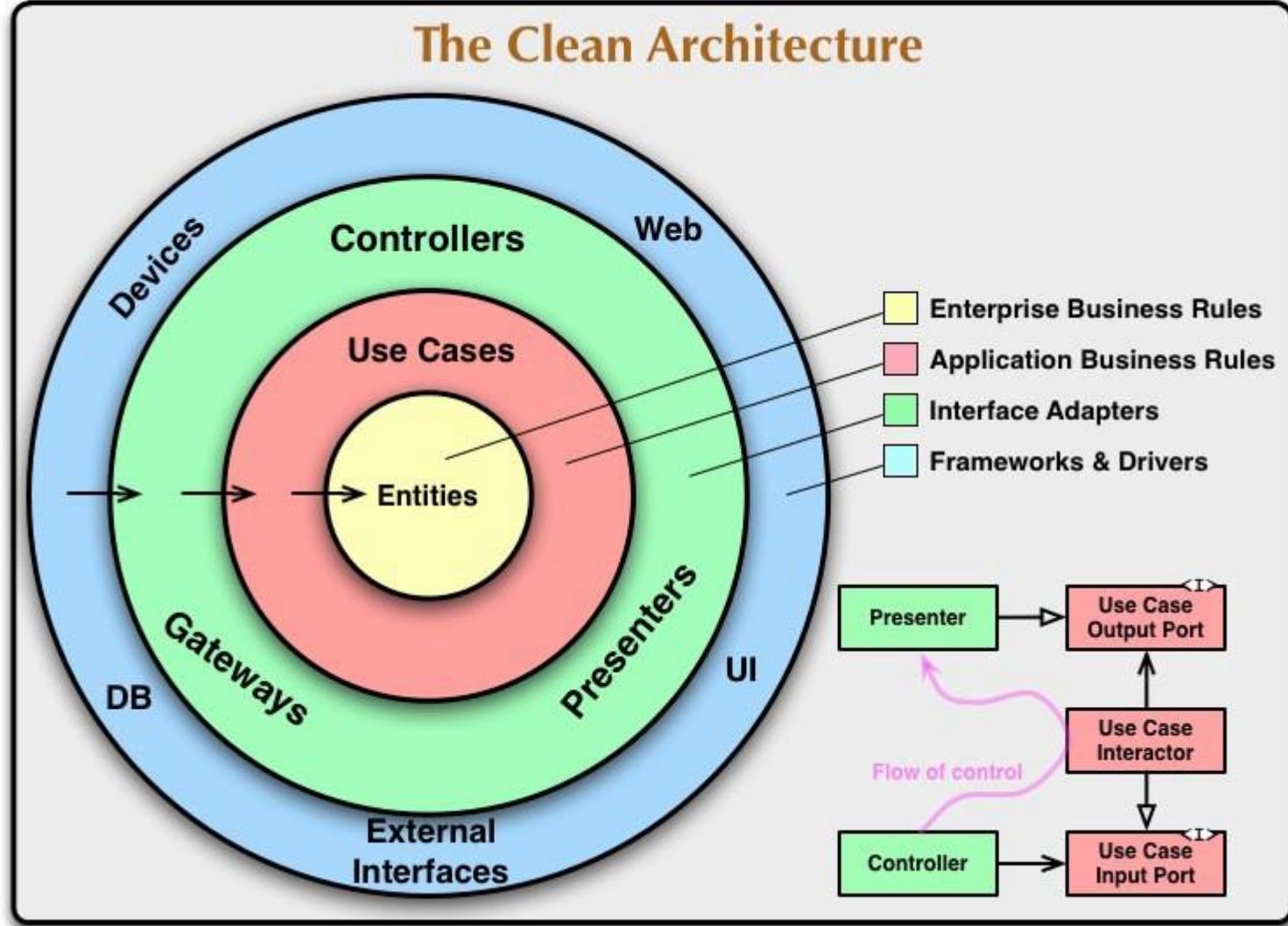
```
@Module
@InstallIn(ApplicationComponent::class)
class NetworkModule {
    @Provides
    @Singleton
    fun provideNetworkCall() : NetworkCall() {
        return NetworkCall(...)
    }
}
```

- *Singleton* megvalósítás csak egy annotációba kerül

# Room – thread safe

- SQLite thread safe
- Room -hoz készült AppDatabase osztály  
INSTANCE része viszont nem
- Megoldás:
  - > Dependency Injection - @Singleton
  - > Manuális szálbiztosítás tétele:
    - [https://www.reddit.com/r/androiddev/comments/c26qd4/room\\_database\\_creation\\_in\\_kotlin/](https://www.reddit.com/r/androiddev/comments/c26qd4/room_database_creation_in_kotlin/)

# Clean Architektúra



Forrás: <https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html>

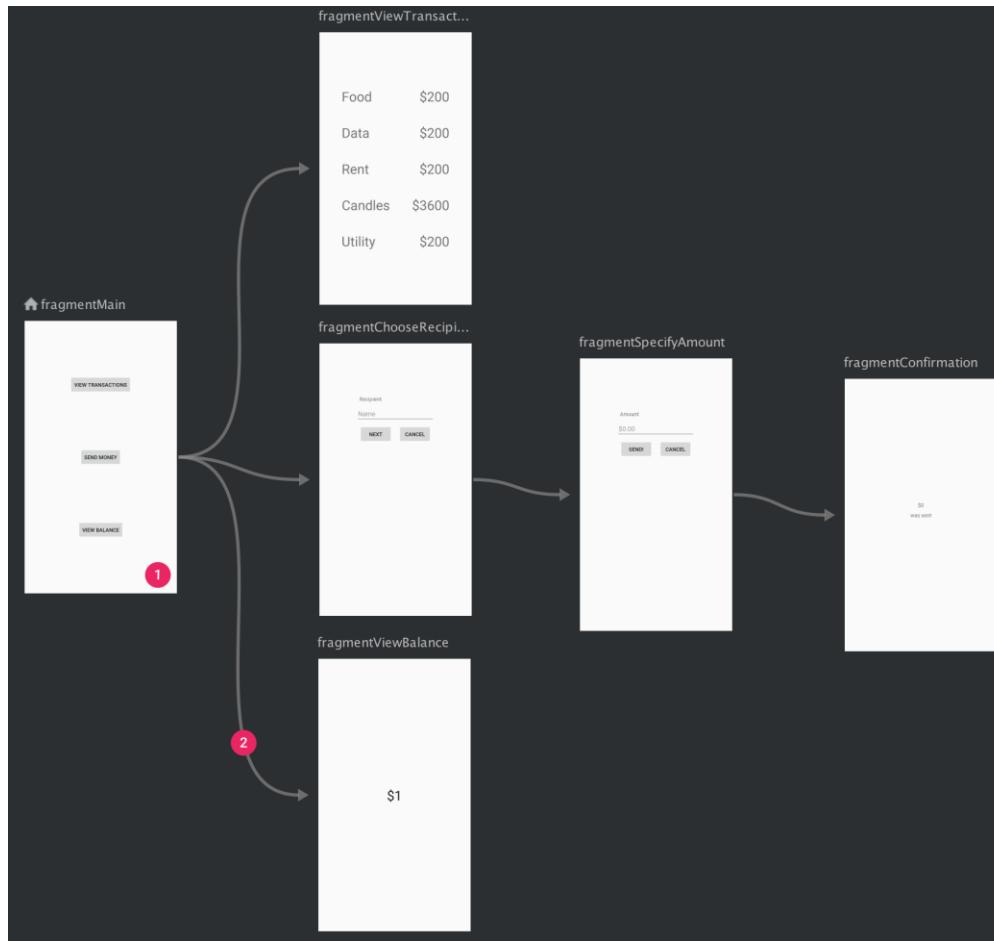
# Android architektúra

- Android Architecture Components - MVVM
  - > LifecycleObserver
  - > LiveData
  - > ViewModel
  - > <https://developer.android.com/topic/libraries/architecture>
- MVP
- Rainbowcake
  - > <https://rainbowcake.dev/about/>

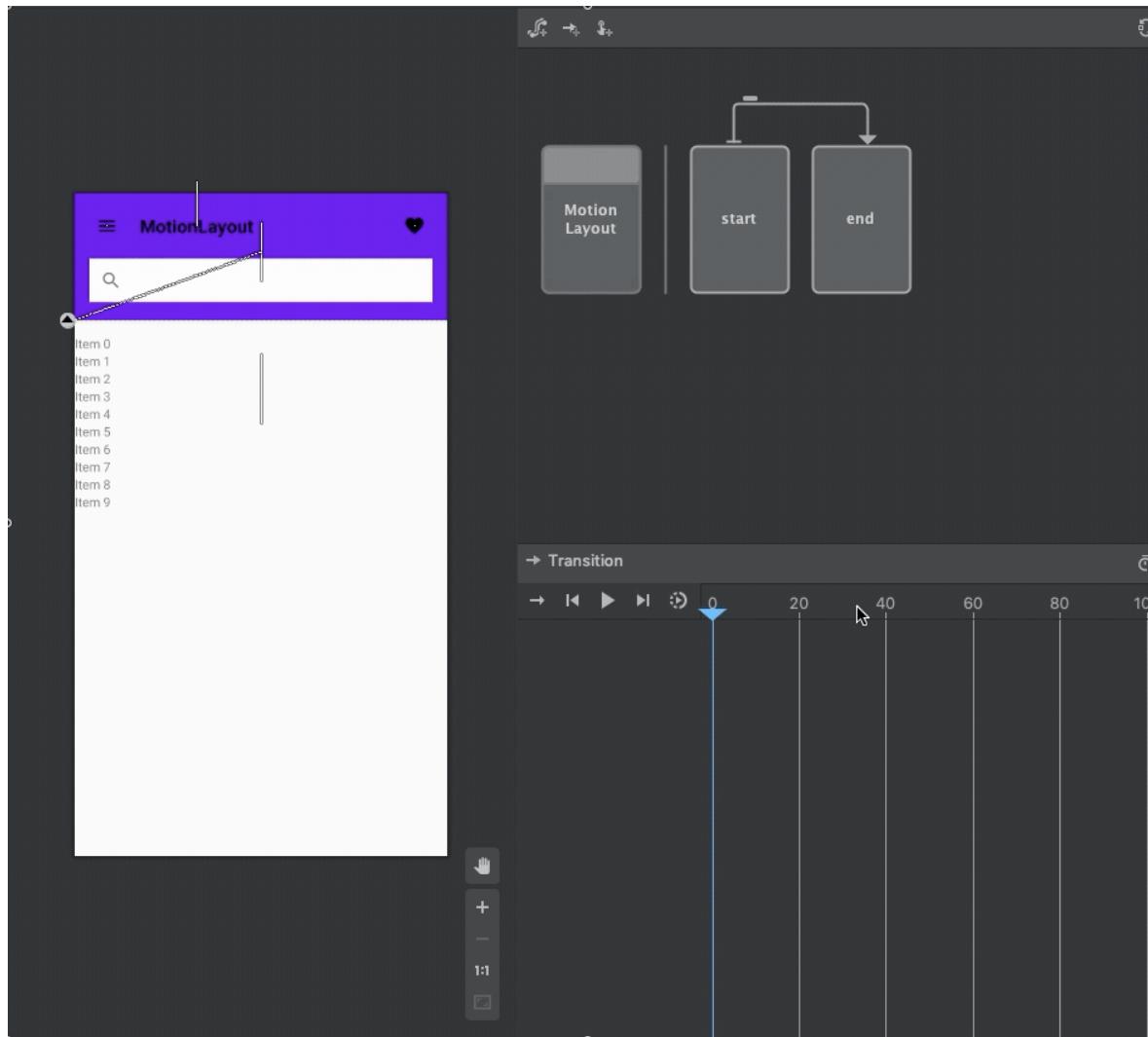
# Navigation Component

- Egyszerűsített navigáció Activity-k, Fragmentek és nézetek között grafikus felületen.
- **Navigation graph:** XML erőforrás ami leírja a navigációs útvonalakat, vizuális megjelenítéssel is rendelkezik
- **NavHost:** Egy üres konténer amin belül a navigáció megvalósul (itt váltakoznak a nézetek), tipikusan egy *NavHostFragment*
- **NavController:** Egy objektum ami a navigációt vezérli és megvalósítja.

# Navigation graph



# MotionLayout

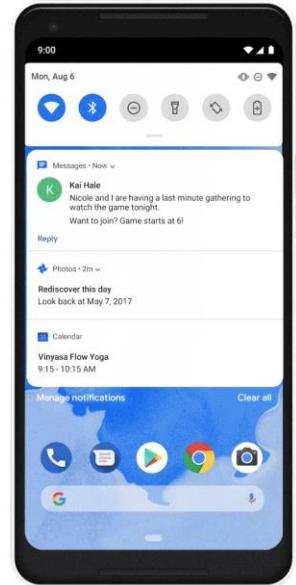
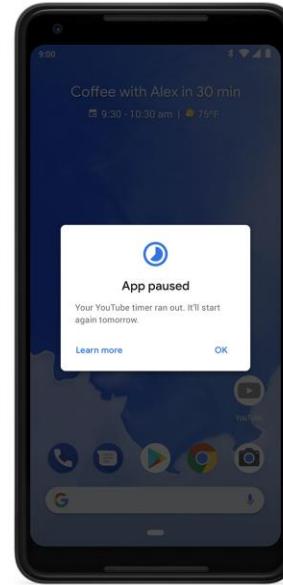
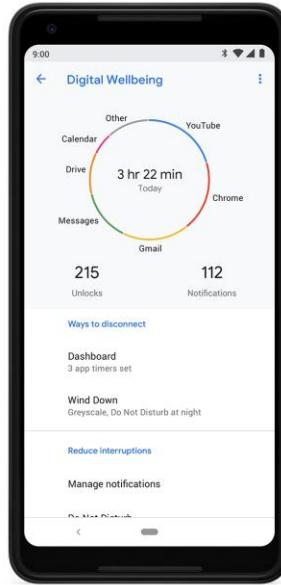


# Android újdonságok



ANDROID

- Felhasználó megismerése
- Slices
- App Actions
- Google Assistant
- Megújult navigáció
- ...



# Változó az alkalmazás (szolgáltatás) fejlesztés



# Útravaló – szoftverfejlesztési elvek



# Code rot (kód romlás)

- Az alkalmazások tipikusan letisztult, tiszta architektúrával indulnak
- Mi történik egy bizonyos idő után?
  - > A kód elkezd „rothadni” (romlani): kicsi hack itt-ott, egyre több *if* elágazás, mígnem az egész kódban ezek dominálnak -> átláthatatlan viselkedés
- Nehéz karbantartani, nehéz új funkciókat hozzáadni -> a fejlesztők egy idő után áttervezésért könyörögnek

# Kód romlás

- A forráskód bizonyos szempontból a terv (design)
- A romlott design és a rossz architektúra tipikus tünetei
  - > Merevség
    - Folyamatosan nehezebb a kód módosítás
    - A változtatás költsége magas
  - > Törékenység
    - Apró változtatás egy modulon egy másik modulban okozhat hibás viselkedést
    - Például: egy bug javítás elront egy látszólag független részt
  - > Mozdulatlanság
    - Egy rendszer mozdulatlan, ha a részeit nem lehet könnyedén modulokba kiszervezni és máshol újra hasznosítani
    - Például: a login modul újra felhasználható legyen
    - Mozdulatlanság elkerülési stratégiák: rétegek kialakítása (adatbázis és UI különválasztása)
  - > Nyúlékonyság
    - A kód struktúra nyúlékonysága
    - Új feature implementálását könnyebb megoldani hackeléssel, mint új kód írásával/új osztály bevezetésével
    - A környezet nyúlékonysága
    - Fordítás, teszt futtatás és becheckolás körülményes és sok ideig tart

# Kód romlás – Mi az okozója?

- Változó követelmények
  - > Ha olyan a kódunk/architektúránk, hogy nehéz a változásokat kezelni, az a mi hibánk
  - > A kód/architektúra rugalmas kell legyen a változások követésére és meg kell akadályozza a kód romlást
- Milyen változások miatt kezd romlani a kód? *Olyan változások, amelyek új, nem tervezett dolgokat hoznak az osztály függőségek szintjén.*
- A legtöbb tünet direkt, vagy indirekt módon a modulok közti nem megfelelő függőségre vezethető vissza.
- Az objektum orientált tervezési elvek segítenek a modulok közti függőségek kezelésében.
  - > SOLID elvek

# Függőség kezelő OO elvek: SOLID elvek

- Single Responsibility Principle (S.R.P.)
- Open Closed Principle (O.C.P.)
- Liskov Substitution Principle (L.S.P.)
- Interface Segregation Principle (I.S.P.)
- Dependency Inversion Principle (D.I.P.)

# Android fejlesztési javaslatok

- Hibamentes, hatékony működés
  - Megfelelő osztálykönyvtárak ismerete és használata
  - Fejlesztőkörnyezet kialakítása
    - > Verziókezelés
    - > Continous Integration
    - > Tesztelés
      - Unit tesztek
      - Integrációs tesztek
      - Teszt környezet
  - Clean code
    - > Kódminőség
    - > Kód újrafelhasználhatóság
    - > SOLID tervezési elvek
    - > Refaktor
  - Optimalizálás
    - > Memória, CPU használat és ... energiafogyasztás
  - Tesztelés (Unit/UI tesztek) ++



# Összefoglalás

# Megismert téma körök

- Mobil platformok helyzete
- Android platform felépítése, verziók, struktúra
- Alkalmazás komponensek
- Felhasználói felület elemek, sűrűség függetlenség
- Adattárolás, perzisztencia
- Content Provider
- Intent, BroadcastReceiver
- Hálózati kommunikáció
- Service

# Android interjú kérdések

1. Describe the APK format.
2. What are the different phases of the Activity life cycle?
3. What is the significance of the .dex files?
4. What is the difference between Service and Thread?
5. What is a Content Provider?
6. When does ANR occur?
7. What is a BroadcastReceiver?

# Android architekt feladatai

- Megfelelő Android verzió kiválasztása
- Alkalmazás architektúrájának megtervezése
- Android alkalmazás viselkedése (komponensek)
- UI tervezek ellenőrzése (mockup vs. specifikáció), tablet
- Hálózati kommunikáció tervezése (protokoll, biztonság)
- Teljesítmény kérdések tisztázása (natív támogatás kell-e?)
- Speciális igények, funkciók, algoritmikai kérdések felmérése
- Megfelelő külső könyvtárak kiválasztása
- Teszt környezet tervezése
- Kód minőség ellenőrése, best practice-k javasolása
- Java, Java, Java, Kotlin, Kotlin, Kotlin... (hatékonyan!)
  - > Joshua Block: Effective Java

# Az architekt további feladatai

- Continous Integration tervezése, felügyelete
- Verziókezelő konvenciók betartatása
- Library függőségek kezelése
- Licence kérdések tisztázása
- Alkalmazás publikációs körülményeinek tisztázása
- Összefoglalva: mindig talál megoldást!

# Hogyan tovább...

- Googel I/O Extended események
- Android fejlesztői lista
  - > ~700 tag
  - > Android fejlesztői közösség
  - > Ötletek megosztása
  - > Problémák közös megoldása
  - > Értesítés különféle eseményekről
- <https://groups.google.com/group/bme-android>

# Kérdések

