Kliensalkalmazások

Android 1 - Bevezetés, alapok, Activity

2023. 04. 24.

Gazdi László

gazdi.laszlo@aut.bme.hu

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Követelmények

- Labor = 10 pont
  - Webes laborok: +1 pont / alkalom = 6 pont
  - Mobilos laborok: +1 pont / alkalom = 4 pont
  - Legfeljebb 1-1 hiányzás lehetséges
- 2 db házi feladat = 30 pont
  - Webes házi bemutatása 10. héten (20 pont, 50% minimum)
  - Mobilos házi bemutatása 14. heti laboron (10 pont, 50% minimum)
- ZH nincs
- Vizsgaidőszakban írásbeli vizsga: 36 (W) + 24 (A) = 60 pont
  - Számonkért anyag: minden, ami az előadáson, laborokon elhangzik
  - Minimum 30 pont

Android témakör

- Nagyvonalakban
  - Kotlin nyelv
  - Android alapok
  - **XML alapú felületek**
  - **Jetpack Compose (dekleratív UI)**
  - **Adatbáziskezelés**
  - **Hálózatkezelés**

Kotlin

- Nyelv ismertető:
  - https://kotlinlang.org/
- Kotlin Koan-ok gyakorolni:
  - https://kotlinlang.org/docs/tutorials/koans.html
- Java-ról Kotlinra tutorial:
  - https://github.com/Zhuinden/guide-to-kotlin

A mobilpiac szereplői

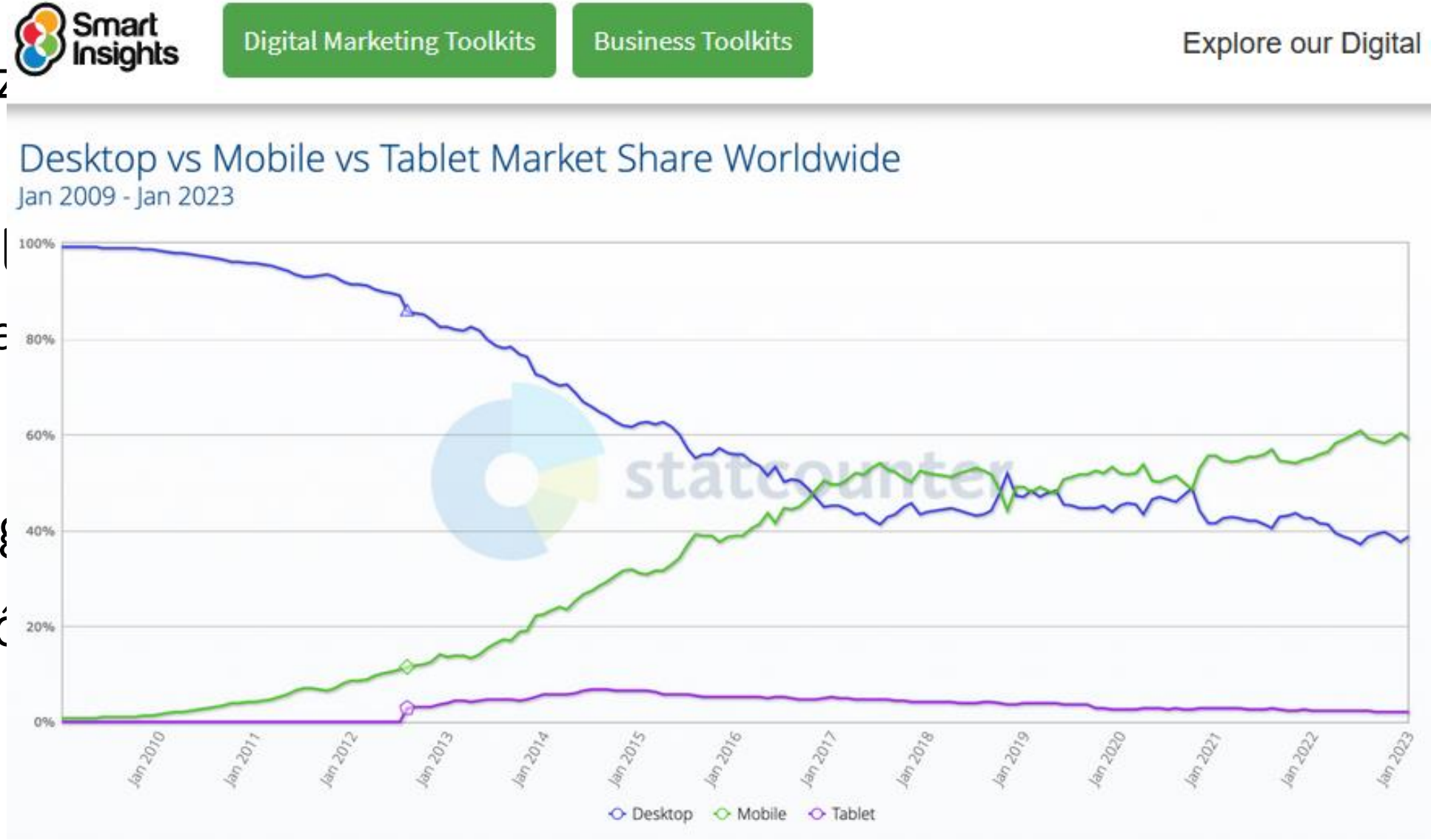
- Hálózat operátor
  - Kiépíti és karbantartja a hálózatot, lehetővé téve a készülékek közötti kommunikációt
  - Pl. T-Mobile, Yettel, Vodafone
- Szolgáltatók
  - Különféle szolgáltatásokat nyúlt a mobilkészülékek ill. a –hálózat felhasználóinak
  - Hangátvitel: Jelenleg még sokszor azonos az operátorral, de egyre inkább szétválik: VMNO (VirtualMobile Network Operator)
  - Pl. Skype, Google, alkalmazásfejlesztők, stb…
- Készülékgyártók
  - Pl. Apple, Samsung, Asus, Acer, OnePlus, Elephone, Doogee
- Felhasználók

Korlátozott erőforrások

- Asztali gépekhez képest a mobilkészülékek erőforrásai jelentősen korlátozottak
- CPU: Alacsonyabb feszültség (hőtermelés)
  - Csökkentett utasításkészlet, órajel csökkentés, részleges kikapcsolás
- Kevés memória
- Korlátozott energiaellátás (akkumulátor véges)
- Kisméretű kijelző, korlátozott felhasználói felület

Korlátozott erőforrások

- Asztali gépekhez
  korlátozottak
- CPU: Alacsonyal
  - Csökkentett uta
- Kevés memória
- Korlátozott energ
- Kisméretű kijelző



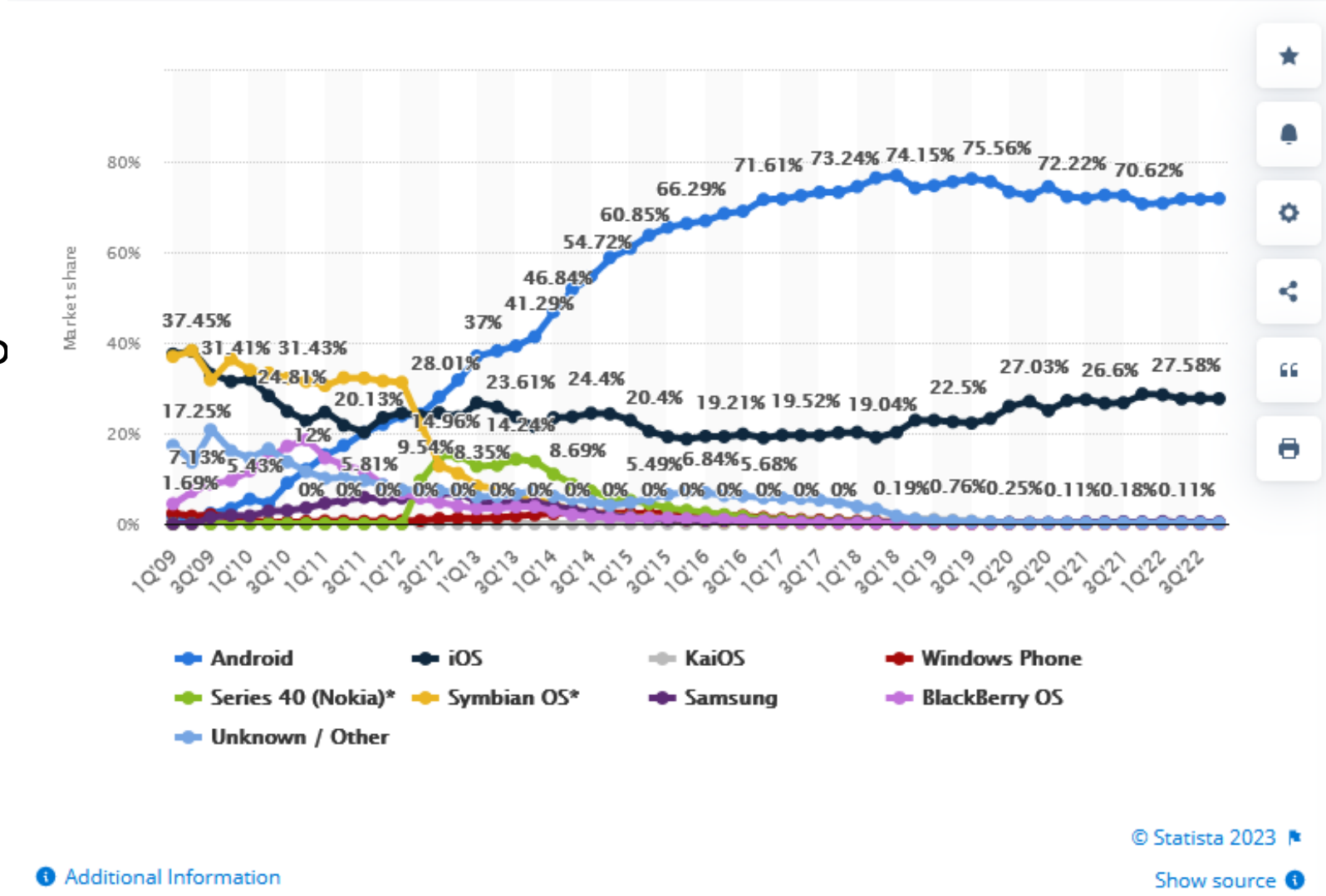
Tartalom

- Mobil platformok áttekintése
- Android bevezetés
- Android platform szerkezete és a fejlesztőkörnyezet
- Android alkalmazás komponensek
- Manifest állomány, jogosultságok
- Erőforrások kezelése
- Android projekt felépítése
- Fordítás mechanizmusa
- Activity Életciklus
- Activity állapot mentése
- Activity Back Stack
- Navigálás Activity-k között
- ViewBinding

Mobil platformok áttekintése

Mobil platformok áttekintése

- Symbian OS
- Java ME
- Python
- Windows Mobile / Windows P
- Meego/Tizen
- Android
- iPhone OS
- Egyéb (ami kimaradt)



Symbian OS - A történelem kezdete

- Kifejezetten mobilkészülékekre kifejlesztett operációs rendszer
- Hardver erőforrásokban szegény készülékekre
  - Gyenge processzor
  - Kevés memória
  - Korlátozott üzemidő (akkumulátor)
- Magas rendelkezésre állásra tervezve
  - Reboot csak ritkán megengedett
- Személyi adatkezelő funkciók OS szintű támogatása (kontaktok, naptár, stb.)
- Csak alap - Erre minden esetben ráépül egy UI platform
  - Hogy minden gyártó egyedi megjelenést biztosíthasson
- Fejlesztés: C, C++, Qt, Java ME, Python

Java ME

- Java Mobile Edition - A Java technológia mobilkészülékekre kifejlesztett változata
- Rengeteg mobilkészülék által támogatott szoftverplatform volt
- Java alaptulajdonságok
  - A programkódból a speciális fordító úgynevezett „object code”-ot készít (nem gépi kód!)
  - Az object code-ot a készüléken futó Java virtuális gép (JVM) futtatja, interpretált módon
- Az eltérő képességű mobilkészülékekhez külön konfigurációk
  - Virtuális gép
  - Alacsony szintű API-k (általában Java SE API-k részhalmaza + mobil specifikus API-k)
- Egy adott konfigurációra további, magasabb szintű API csomagok épülnek, ezek a **profilok**

Windows Mobile/Phone

- Windows Mobile:
  - A Windows mobil mutációja volt
  - Okostelefonokra, PDA-kra
  - Windows CE-re épül
  - Az operációs rendszerhez hozzá tartozik számos ismert alkalmazás mobil verziója (Excel Mobile, Word Mobile, stb.)
  - Fejlesztés: Java ME, .NET CF
- Windows Phone 7/8:
  - Könnyen testreszabható és mindig friss home screen
  - Az úgynevezett „live tile”-ok mindig a legfontosabb információkat jelenítik meg (hívás, sms, naptár, időjárás, stb.)
  - Fejlesztés: Silverlight alapon (C#, VB) vagy natív C++

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

Meego, Tizen

- Meego:
  - Linux alapú operációs rendszer Internettáblákra, mobilokra, set-top boxokra
  - Nagyméretű érintőképernyő
  - Fejlesztés: Régen csak scratchbox linux alól, Python, C/C++
  - Linux alkalmazások viszonylag könnyen portolhatók rá
- Tizen:
  - MeeGo fejlesztők (Linux Foundation) és Intelesek áttértek rá
  - Samsung, Intel beálltak mögé
  - Fejlesztés:
    - Natív appok: C, C++, Python, Lua
    - HTML5 és JavaScript alapú appok
    - .NET support
    - Android alkalmazások (módosított Dalvik VM)

iOS

- Eredetileg iPhone OS, az Apple-től
  - Specialitás: Exkluzívan az ő hardvereikre
- iOS 14 jelenleg
- Apple újdonsága volt: Központilag kontrollált terjesztés, app modellek
- iPod Touch (2007), iPhone (2007), iPad (2010), Watch (2014)…
- Speciális UX ötlet: multi-touch gesztúrák, közvetlen manipulálás
- OS X-el közös alapok (Core Foundation, Foundation Kit), de speciális UI (Cocoa Touch)
- Fejlesztés: Objective C, Swift
- Jól bevált eszközök
  - XCode, Interface Builder, …

Android bevezetés

Mi is az Android?

- Egyrészt: operációs rendszer (nem csak mobiltelefonokra!)
- Másrészt: 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

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, ahol:
  - Alapvetően kicsi a kijelző (Google TV megcáfolja!)
  - Más jellegű erőforrások
  - Az adatbevitel nem tipikusan egérrel és/vagy billentyűzettel történik
  - Android@Home

Android eszközök 2/2

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 kódnév: valamilyen édesség ☺

Android verziók

- 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 (Red Velvet Cake)
- Android 12 (Snow Cone)
- Android 13 (Tiramisu)
- (Android 14) (Upside Down Cake)
- (Android 15) (Vanilla Ice Cream)

Mi nem igaz az Androidra?

A. Linux kernel alapú.

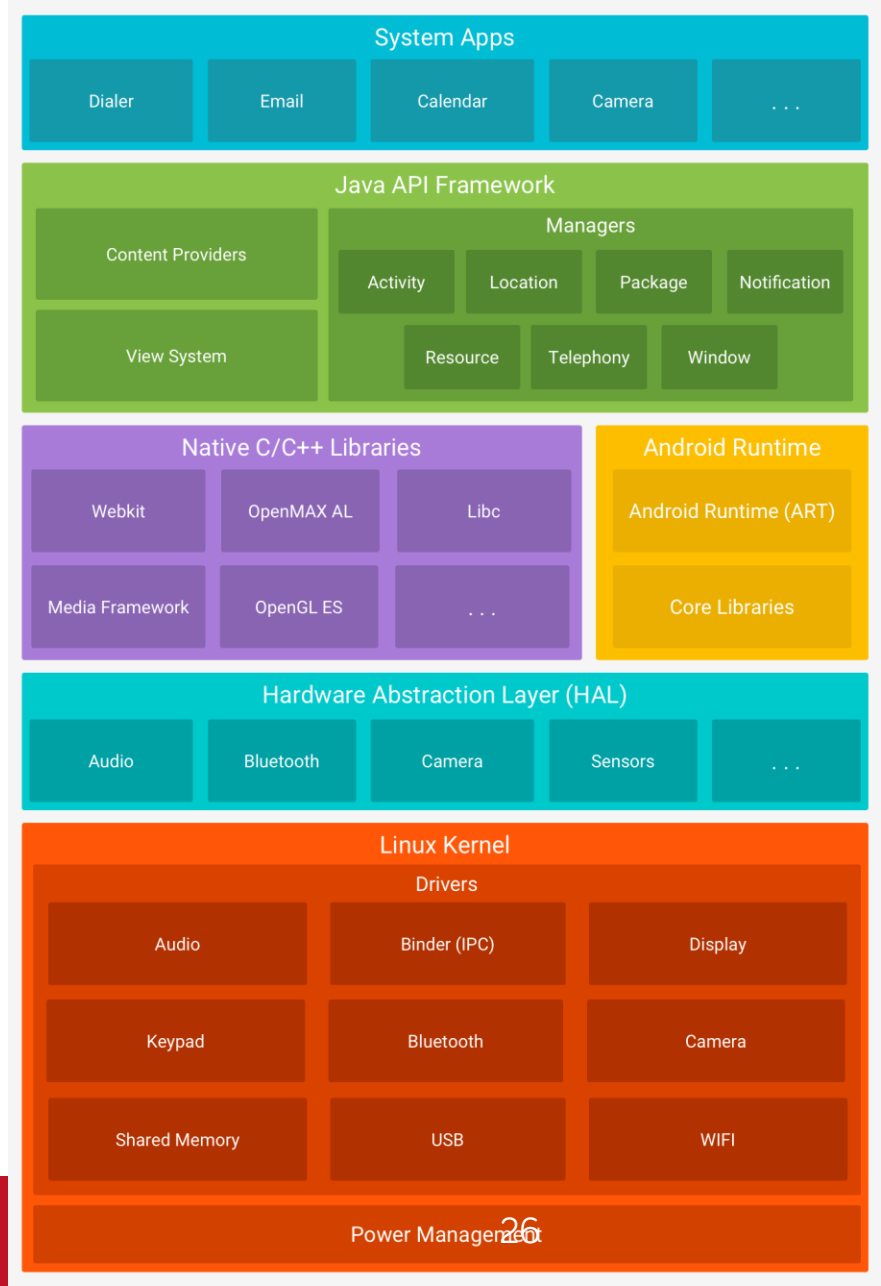
B. A verzió kódneve általában valamilyen édesség.

C. Csak okostelefonokon fut.

D. Nyílt forráskódú.

Android platform szerkezete és a fejlesztőkörnyezet

Az Android platform szerkezete



ART: virtuális gép
(Android RunTime)

Régebben: DVM
Dalvik Virtaul Machine

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)

SDK komponensek

- SDK minden Android verzióra
- Dokumentáció
- Példakódok
- USB Driverek (ADB)
- Third party kiegészítők
  - Google APIs (Térkép)
  - Galaxy Tab API
  - Stb.
- Konzolos felhasználás is támogatott, pl projekt létrehozás:
  - android create project --target android-16 --name MyFirstApp --path D:\tmp\MyFirstApp --activity MainActivity --package com.example.myfirstapp

Az SDK felépítése

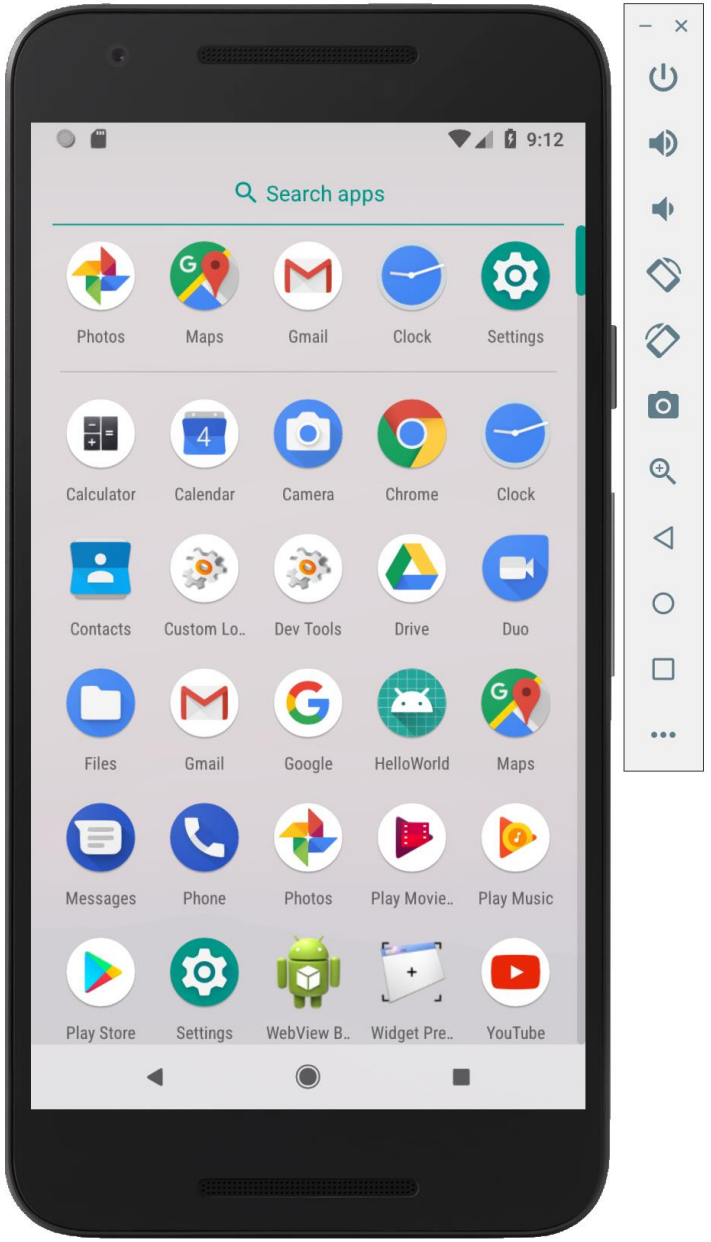
- **add-ons/:** Kiegészítők külső könyvtárak használatához, pl. Google APIs
- **docs/:** Offline dokumentáció
- **platform-tools/:** Eszközök, mint például az adb az emulátorok, készülékek vezérléséhez
- **platforms/:** Az egyes platform verziók
- **samples/:** Példa kódok platform verziónként
- **tools/:** 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
- 2022.2.1 (Flamingo)

Emulátor

- Teljes operációs rendszer emulálása (lassú)
  - > Beépített alkalmazások elérhetők
  - > Ctrl+F11 (screen orientáció állítás)
- Elérhető konzolból is



Debugolás folyamata

- On-device debug teljes mértékben támogatott
  - Megfelelő USB driver szükséges!
  - Készüléken engedélyezni kell az USB debugolást
- Minden alkalmazás önálló process-ként fut
- Minden ilyen process saját virtuális gépet (VM) futtat
- Minden VM egy egyedi portot nyit meg, melyre a debugger rácsatlakozhat (8600, 8601, stb.)
- Létezik egy úgynevezett „base port” is (8700), mely minden VM portot figyel és erre csatlakozva az összes VM-et debugolhatjuk

Az első Android alkalmazás (Java)

Az első Android alkalmazás

Ősosztály

```
public class HelloAndroid extends AppCompatActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello Android!");
        setContentView(tv);
    }
}
```

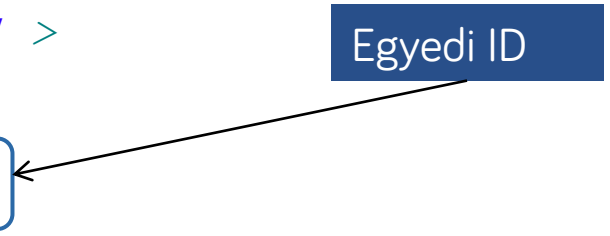
Ősosztály implementáció meghívása

TextView megjelenítése

Android HelloWorld XML alapú UI-al 1/2

Hello Android XML (*res/layout/activity_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>
```



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

Mivel anonim osztályból férünk hozzá

Egyszerű érintés esemény kezelés

Az első Android alkalmazás Kotlin-ban ☺

Egyszerű esemény kezelés

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

        myTextView.setOnClickListener{
            myTextView.append("\n--CLICKED--")
        }
    }
}
```

Lambda hívás

Kotlin extensions miatt használható

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final TextView myTextView =
      (TextView)findViewById(R.id.tvHello);
    myTextView.append("#");
    myTextView.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            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

```
<Button
    android:id="@+id/btnTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="click"
    android:text="Show" />
```

```
class MainActivity : AppCompatActivity() {

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

    fun click(view: View) {
        Toast.makeText(this,
                Date(System.currentTimeMillis()).toString(),
                Toast.LENGTH_LONG).show()
    }
}
```

Melyik igaz az Androidra?

A. Androidos appot csak telefonon lehet debuggolni.

B. Fejlesztéshez használható natív C++.

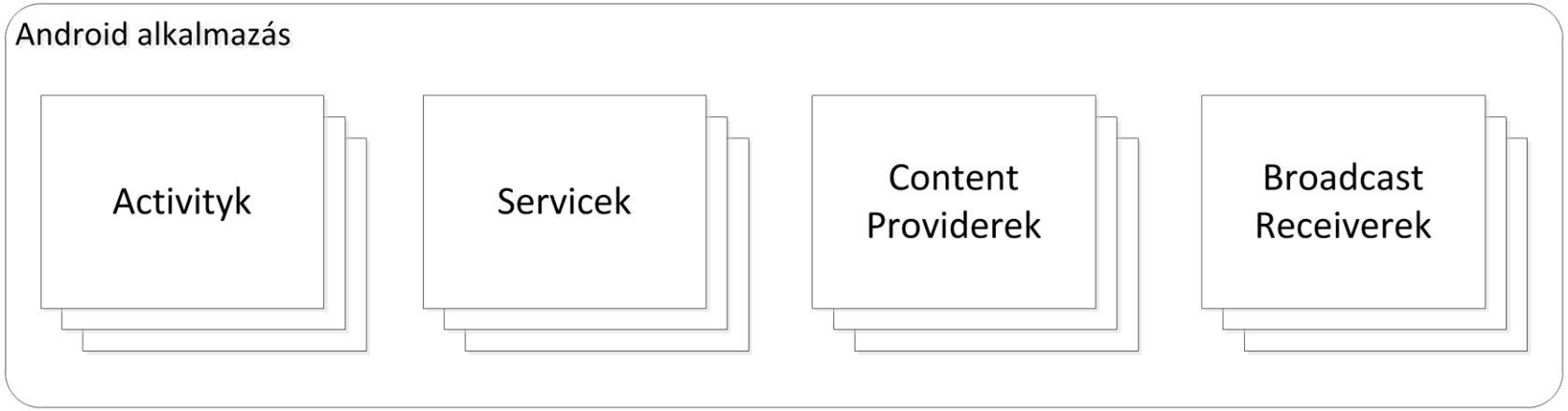
C. Az Androidos SDK fizetős.

D. Eseménykezelőt csak layout-ban lehet megadni.

Android alkalmazás komponensek

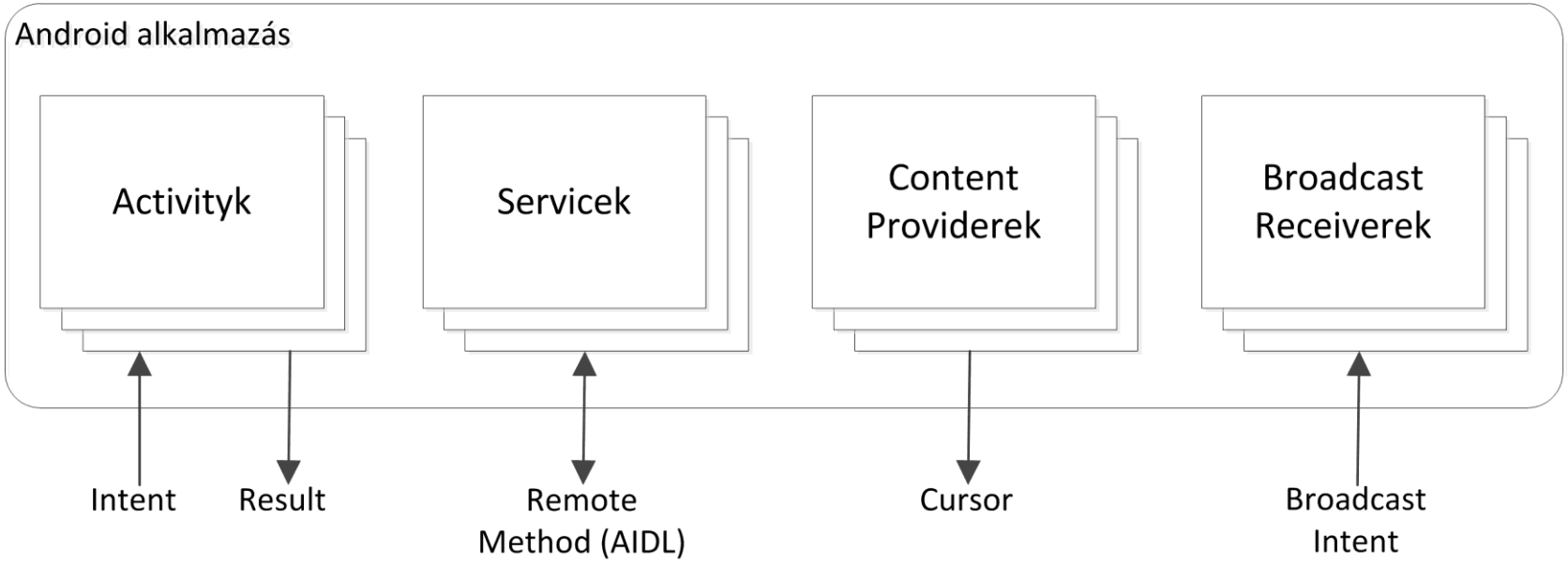
Android alkalmazás felépítése 1/3

- 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
- Bármelyik 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-el
- 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.bme.aut.android.examples"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion„21" />
    <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 cimke

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>
```

Activity osztály és cím

Alkalmazás belépési pont jelölő

Megjelenik a futtatható alkalmazások listájában (Launcher)

Manifest attribútumok és tag-ek

- `android:icon`: alkalmazás ikonja
- `android:name`: Activity teljes neve package-el együtt
- `android:label`: A készülék felületén, a felhasználók által látható név
- Komponensek:
  - `<activity>`: Activity
  - `<service>`: Service
  - `<provider>`: Content provider
  - `<receiver>`: Broadcast receiver
- A manifest-ben nem szereplő Activity-k, Service-k és Content provider-ek nem láthatók a rendszer számára
- Broadcast receiver-ek viszont dinamikusan is ki/be-regisztrálhatnak (kódból – `registerReceiver()`)

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, ikonok, szövegek 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

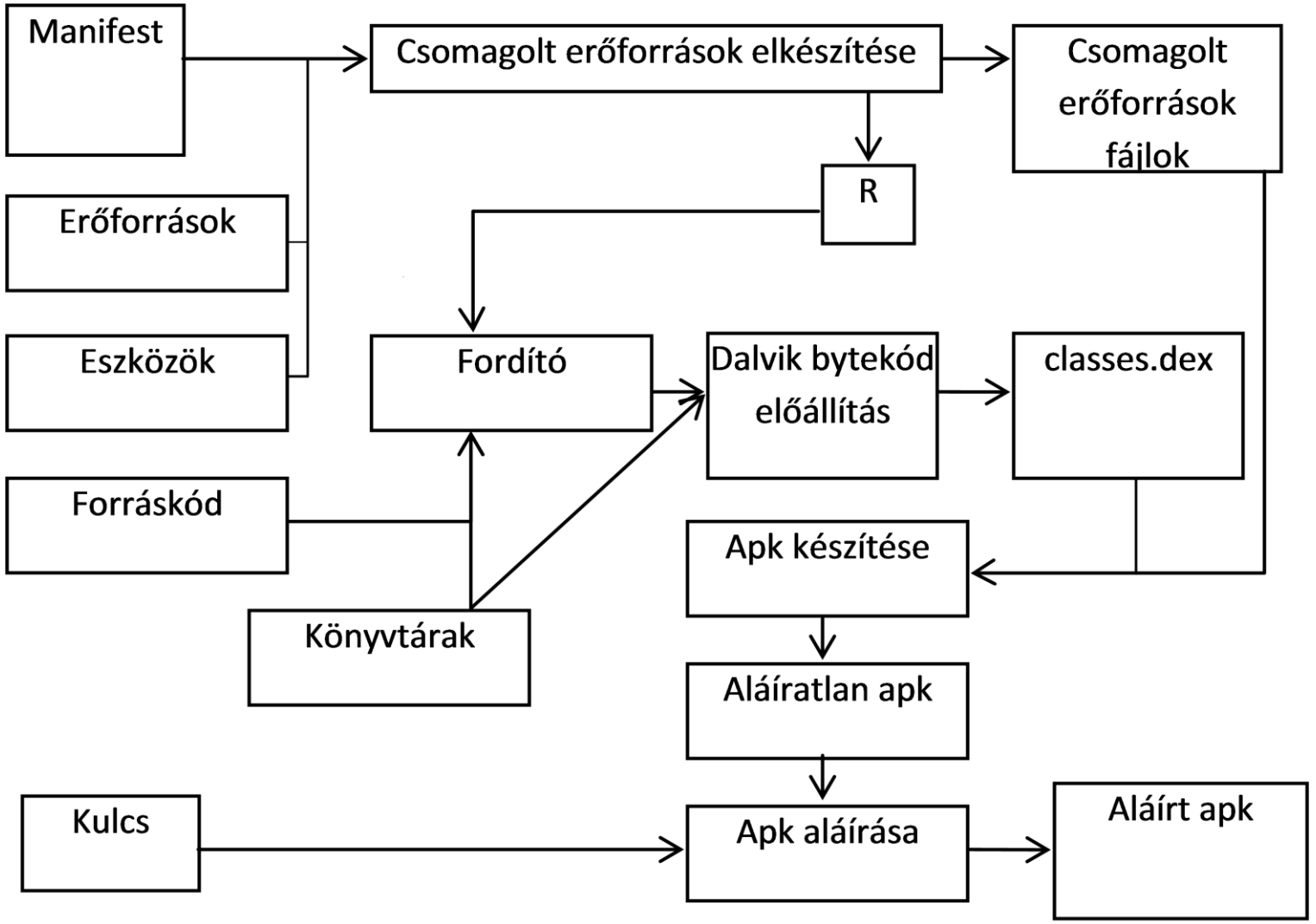
- Tegyük fel, hogy készítettünk egy *logo.png*-t és elmentettük a *res/drawable/* könyvtárba
- Az SDK eszköz előállít egy egyedi erőforrást hozzá mentés után automatikusan
- Az azonosító: *R.drawable.logo*
- Ezzel az azonosítóval lehet hivatkozni bárhol az erőforrásra
- Az azonosítók az *R.java* állományban 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 (akár többet is) 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/*

Android projekt felépítése

A fordítás menete (forrás->.apk)



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 úgynevezett *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

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: wxqnEAI0UA5nO5QJ8CGMwjkGGWE=
...
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 érthető formátumban
  - resources.arsc

APK visszafejtés

- MyBackup
  - > APK megszerzése
- dex2Jar
- JD-Gui

- →obfuszkálás

Melyik igaz az Androidra?

A. Csak egy erőforrásminősítőt használhatunk.

B. Az R.java állományt nekünk kell karbantartani.

C. A megírt kódunk biztonságban van az apk-ban.

D. A kódunk java bytekódra fordul.

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

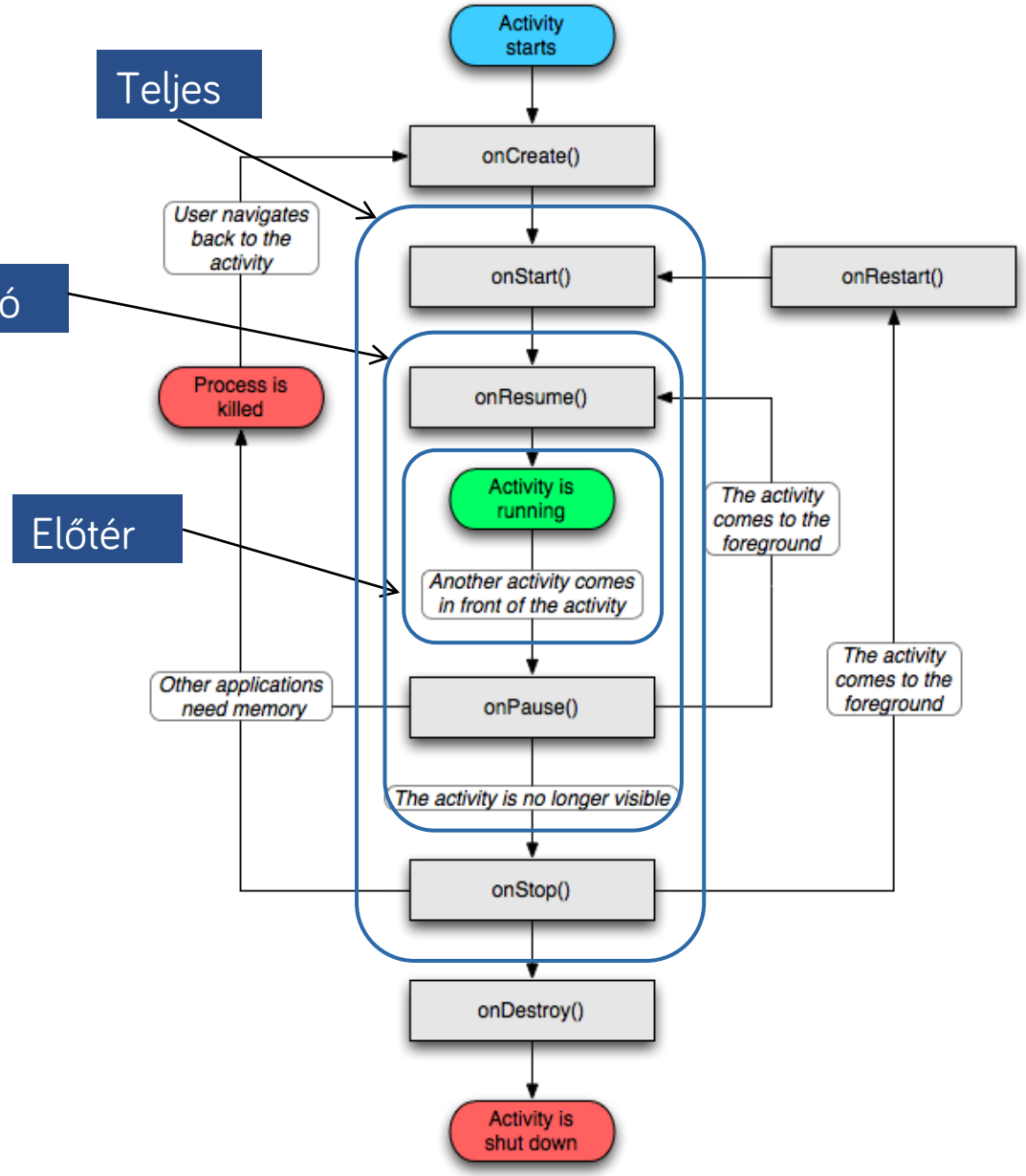
- Egy megbízható és flexibilis alkalmazás esetén kritikus fontosságú az Activity életciklus-callback függvények megfelelő felüldefiniálása
- Az Activity életciklusát a vele együttműködő többi Activity határozza meg
- Elengedhetetlen az Activity működésének tesztelése a különböző életciklus állapotokban

Activity állapotok

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

Activity életciklus

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



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 ős osztály implementációját is (pl. `super.onCreate();`)!**
- A rendszer felelőssége meghívni ezeket a függvényeket, de a fejlesztő felelőssége a helyes implementáció

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 kezelni 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

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
    }
}
```

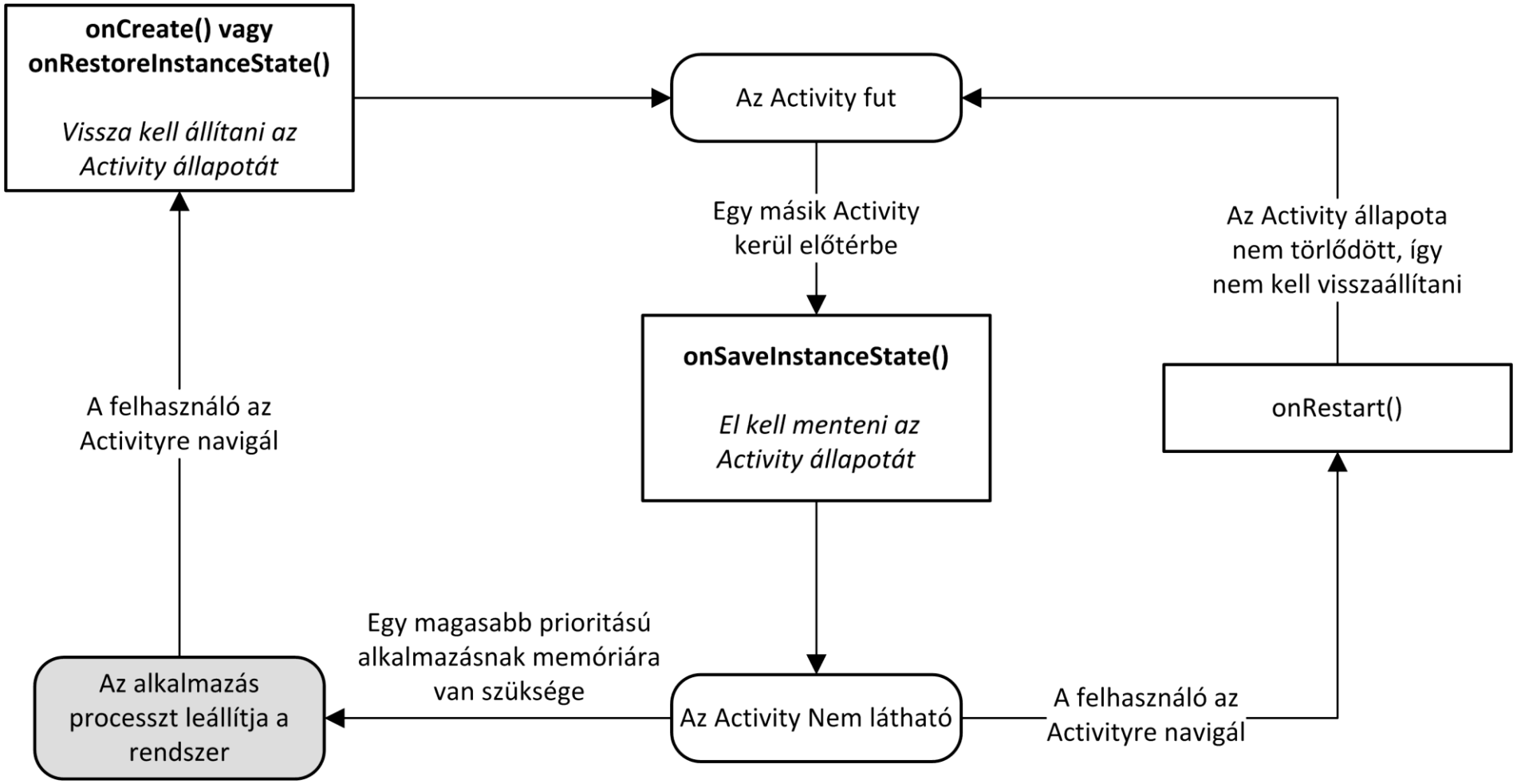
Activity bezárása a rendszer által

- *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

Activity állapotának elmentése 1/3

- A felhasználó nem tudja, hogy amikor visszalép egy Activity-re, akkor azt a rendszer újra létrehozta-e, vagy csak megnyitotta a memóriából
- *onSaveInstanceState()*: callback függvény, akkor hívódik meg, mielőtt az Activity sebezhetővé válna a rendszer általi bezárásra
- Bundle objektumba lehet elmenteni az értékeket, melyet *onCreate()*-kor megkap az Activity

Activity állapotának elmentése 2/3



Activity állapotának elmentése 3/3

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

Konfiguráció változások kezelése az Activity-ben

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

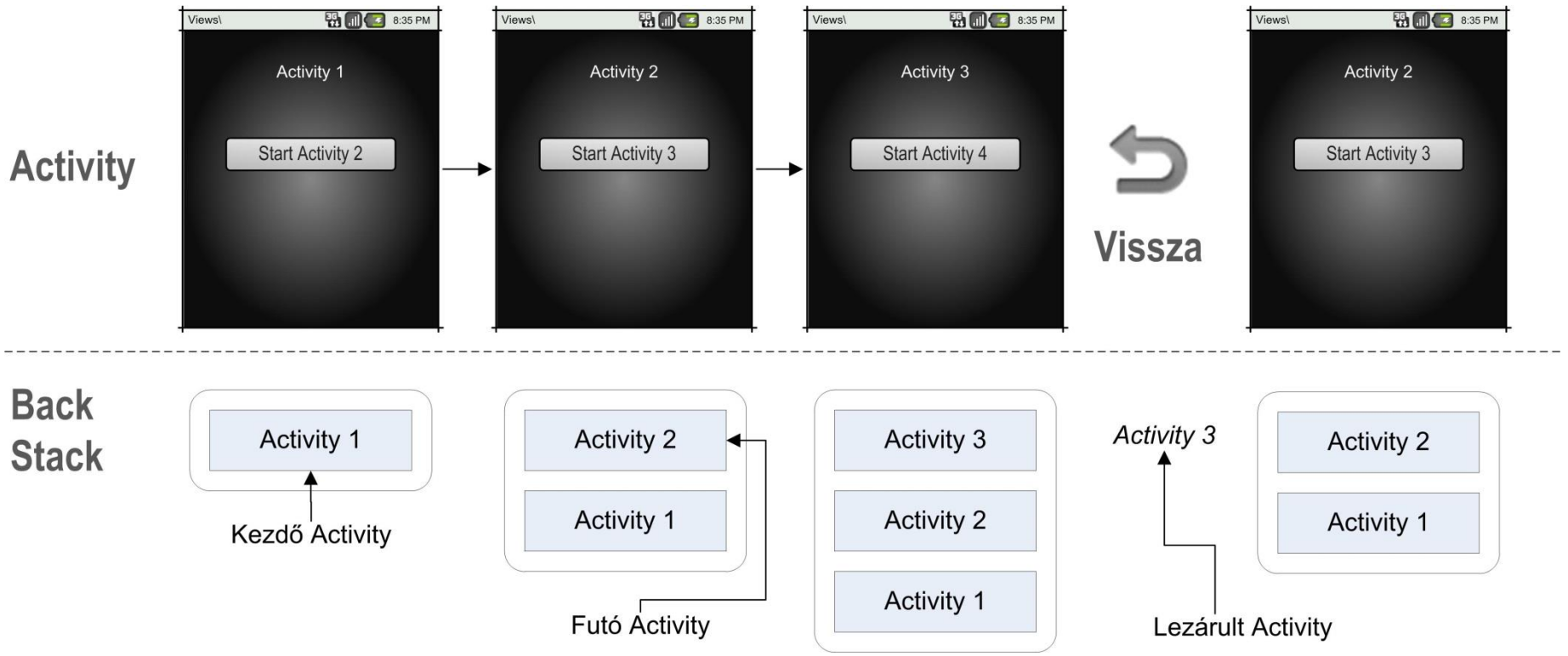
Activity váltás

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

Activity Back Stack 1/2

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

Activity Back Stack 2/2



Activity vezérlés

- 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 mindig egy kezdő Activity-re kell visszalépni
- Az alapértelmezett viselkedés felülírása:
  - Manifest állományban az <activity>-be
  - *startActivity(...)* fv. Paramétereként
- Amennyiben az alapértelmezett viselkedést módosítjuk, mindenképp teszteljük az alkalmazást navigálás és felhasználói élmény szempontjából, mert sokszor a programozó szempontjából jó megoldás nem ideális felhasználói szempontból

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.

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)
}
```

ViewBinding

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

View Binding

- findViewById (és Kotlin synthetic) kiváltása
- Amint aktiváltuk akkor a modulban található összes layouthoz generálódik egy binding class
  - > Binding class referenciát tartalmaz a root-hoz és minden View-hoz aminek van ID-je

```
buildFeatures {
    viewBinding true
}
```

- Layout file ignore-álható

```
<LinearLayout
    ...
    tools:viewBindingIgnore="true" >
  ...
</LinearLayout>
```

View Binding példa

- Tegyük fel, hogy adott egy *activity_main.xml*
- Ebből generálja a rendszer az *ActivityMainBindig* osztályt, ami tartalmazza a *View*-kat, melyeknek van *id*-ja

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.tvHello.text="DEMO"
    }
}
```

Hogy is volt?

- Milyen mobilplatformokat ismer?
- Milyen lehetőségeink vannak a fejlesztés közbeni debugolásra?
- Egy Android alkalmazás milyen komponensekből épülhet fel?
- Miket kell tartalmaznia a Manifest állománynak?
- Mit értünk erőforrás-minősítő alatt?
- Magyarázza el a fordítás mechanizmusát!
- Mit jelent az obfuszkálás?
- Az Activity callback életciklus-függvények felüldefiniálásakor meg kell-e hívni kötelezően az ős osztály implementációját? Miért?
- Ha A Activity-ből átváltunk B Activity-re, milyen sorrendben hívódnak meg az életciklus függvények?
- Magyarázza el az Activity Back Stack működési elvét!
- Hogy kell Activity-t indítani, ha vissza akarunk kapni adatot belőle?

Összefoglalás

- Történelem: mobil platformok áttekintése
- Android platform szerkezete és a fejlesztőkörnyezet: Android Studio, SDK
- Android alkalmazás komponensek: Activity, Service, Broadcast Receiver, Content Provider
- Manifest állomány
- Erőforrások kezelése
- Android projekt felépítése
- Fordítás mechanizmusa
- Activity Életciklus
- Activity állapot mentése
- Activity Back Stack
- Navigálás Activity-k között
- ViewBinding

Köszönöm a figyelmet!