

Android

ContentProvider, Szenzorok

Dr. Ekler Péter

peter.ekler@aut.bme.hu



Department of
Automation and
Applied Informatics

Tartalom

- ContentProvider
 - > Providerek használata
 - > Provider készítés
- Grafikonok rajzolása
- Szenzorok

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**

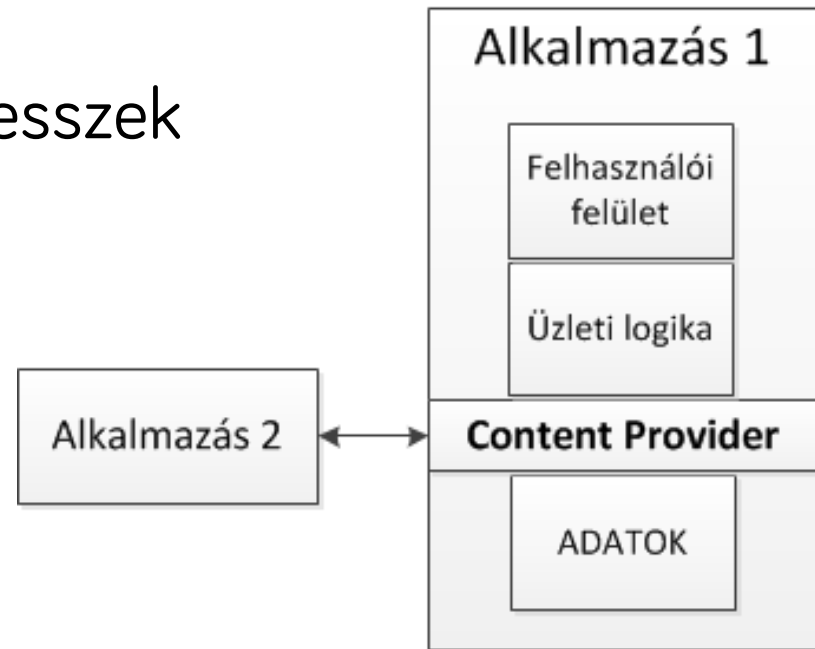
- > Bármikor elérhetetlenné válhat
- > Látható és módosítható, akár törölhető a felhasználó által

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
 - > Elfedí az adat tényleges tárolási módját
 - > Adatvédelem biztosítható
 - > Megvalósítható akár a processzek közti adatmegosztás is
- Neve: **Content Provider**

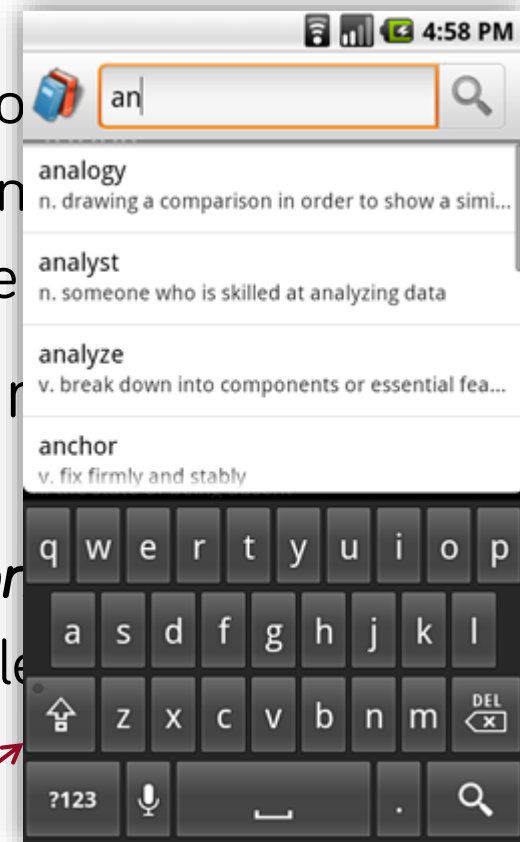


Content Provider – Mikor?

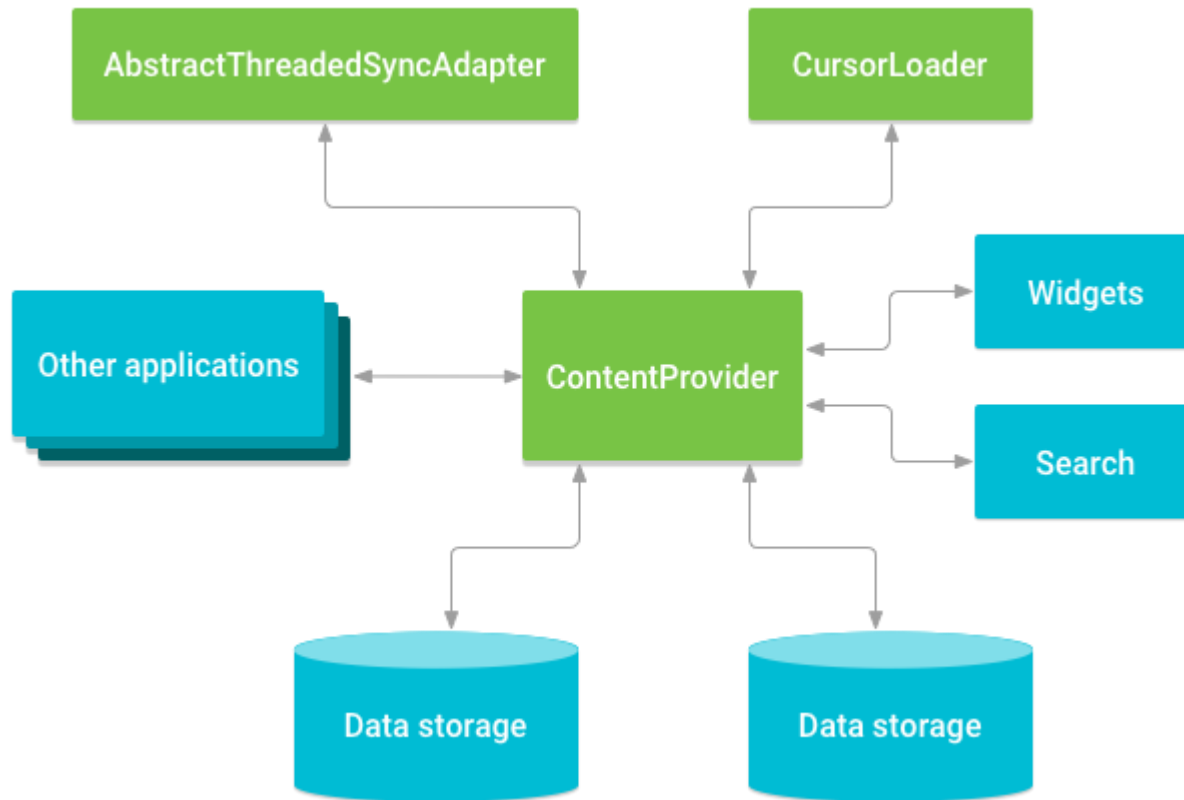
- Nem kötelező Content Provider-t írni...
 - > Ha nem akarunk más alkalmazásokkal adatot megosztani
 - > Nincs szükség a rétegek elkülönítésére alkalmazáson belül
 - > Vagy megfelel a többi megoldás valamelyike
- Bizonyos esetekben viszont mindenképp meg kell készíteni, pl:
 - > Egyedi keresési javaslatok *Search Framework*-el
 - > Komplex adatok vagy fájlok kimásolása-beillesztése (copy-paste) más alkalmazásba

Content Provider – Mikor?

- Nem kötelező Content Provider-t írni...
 - > Ha nem akarunk más alkalmazásokkal adatot cserélni
 - > Nincs szükség a rétegek elkülönítésére alkalmazások között
 - > Vagy megfelel a többi megoldás valamelyike
- Bizonyos esetekben viszont mindenképp szükséges csinálni, pl:
 - > Egyedi keresési javaslatok *Search Framework* használatával
 - > Komplex adatok vagy fájlok kimásolása-beillesztése (copy-paste) más alkalmazásba



Mikor használjunk *ContentProvider*-t?



Forrás: <https://medium.com/@sanjeevy133/an-idiots-guide-to-android-content-providers-part-1-970cba5d7b42>

Content Provider beépítve

- Az Android a globálisan elérhető adatok megosztására is ***Content Provider***-eket használ, például:
 - > Médiafájlok (zenék, képek, videók)
 - > Naptár, névjegyzék, hívásnapló
 - > Beállítások
 - > Legutóbb keresett kifejezések
 - > Böngészőben lévő könyvjelzők
 - > Felhasználói szótár, stb...

Adatok szolgáltatása

- Mintha egy vagy több adatbázis táblát látnánk a Content Provider-en keresztül
- Példa: felhasználói szótár Provider által használt tábla:

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

Content Provider elérése

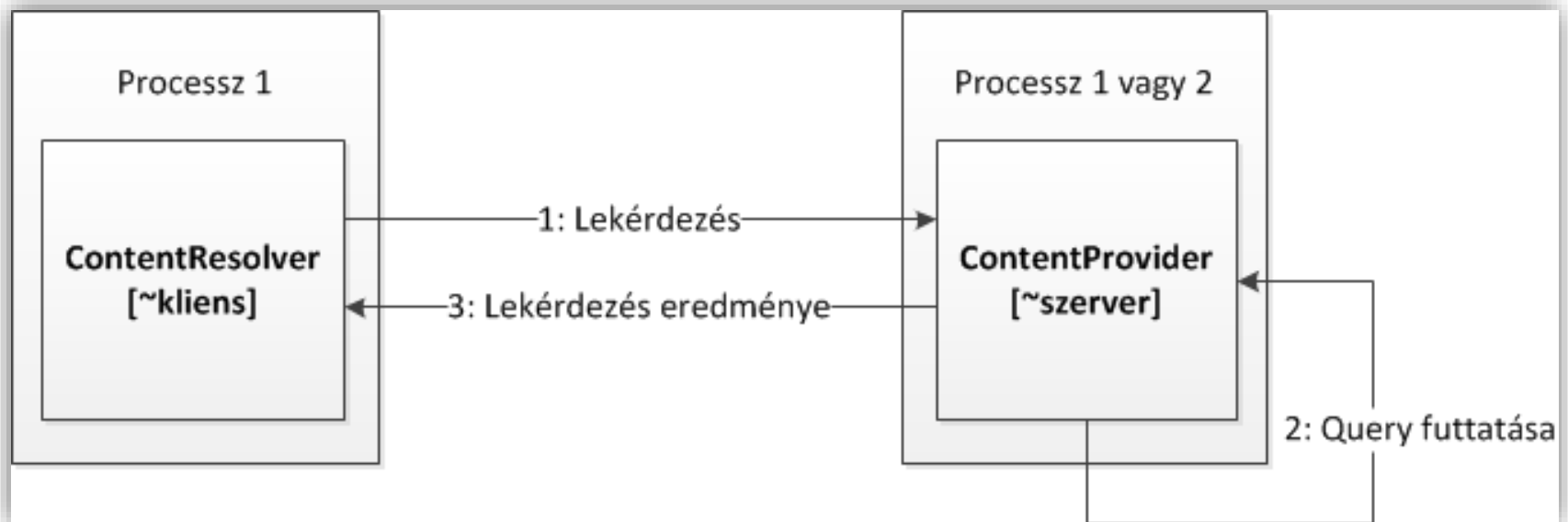
- Content Provider-től kérdezhetjük le az adatokat
- A komponens ami képes a lekérdezések futtatására és a válasz feldolgozására: **ContentResolver**
 - > Csak ez tudja lekérdezni a Content Providert
 - > Lehet akár ugyanabban, akár másik alkalmazásban (processzben)
 - > A kommunikációhoz szükséges IPC-t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

Content Resolver

- Kliens szerep, a szerver maga a Provider
- A *ContentResolver*-en hívhatjuk a lekérdezéshez használatos metódusokat, melyek hatására hívódik a megfelelő *ContentProvider* azonos nevű függvénye
- Nem példányosítjuk közvetlenül, hanem lekérhető (*contextResolver* bármilyen *Context*-ben, pl. *Activity*-ben).

ContentResolver

- Kliens szerep, a szerver maga a Provider
- A *ContentResolver*-en hívhatjuk a lekérdezéshez használatos metódusokat, melyek hatására hívódik a megfelelő *ContentProvider* azonos nevű függvénye



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 mindig jelen 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

CONTENT_URI felépítése

- (Emlékeztető) URI:
scheme://host:port/path
- CONTENT_URI:
content://authority/path[/id]

CONTENT_URI

- Sok Provider lehetővé teszi, hogy a CONTENT_URI végén megadjuk a keresett elem azonosítóját (elsődleges kulcsát), pl:

content://user_dictionary/words/4

- Több osztály is ad segédmetódust
 - > *Uri, Uri.Builder, ContentUris*

**ContentUris.withAppendedId(
UserDictionary.Words.CONTENT_URI, 4) ;**

CONTENT_URI

- Kötelező attribútum minden *ContentResolver*-en hívott metódusnál
- A végbemenő folyamat:
 1. A *ContentResolver* a paraméterben kapott CONTENT_URI-ből meghatározza az **authority**-t
 2. Egy globális, Android által kezelt táblából megkeresi az ehhez tartozó *ContentProvider*-t (innen jön a Resolver elnevezés, „feloldja” a nevet)
 3. A megfelelő *ContentProvider*-nek átadja a lekérés paramétereit
 4. A *ContentProvider* futtatja a query-t, és visszatér

Engedélyek

- A rendszer által nyújtott Providerek eléréséhez általában felhasználói engedély szükséges
- A konkrét engedély a Provider dokumentációjában található
- Pl. a felhasználói szótár olvasásához:
`android.permission.READ_USER_DICTIONARY`
- Telepítéskor el kell fogadni és futási időben is kell kérni a megfelelő engedélyeket

SQL Injection

- Amennyiben a *ContentProvider* által kiajánlott adatainkat SQLite adatbázisban tároljuk, számolnunk kell rosszindulatú bemenettel, pl:

```
SELECT * FROM words WHERE word = [user input]
```

```
[user input] = "; DROP TABLE *;"
```

- Ekkor minden tábla törlődik!
- Megoldás: a szelekciós paraméterben a változók helyére ?-et írunk, és az értékeket külön adjuk át
 - > Ekkor nem egy SQL utasításként kezeli a rendszer, hanem query paraméterként, így nem futthat le
 - > Minden szelekciós feltételnél ez az ajánlott megoldás, nem csak a felhasználói bevitelből származóknál, főleg ha SQLite-ban tároljuk a tényleges adatokat

SQL Injection fun



Cursor 1/2

- A query() mindig **Cursor**-al tér vissza
 - > Az egész eredményhalmazra mutat
 - > Nem csak szekvenciálisan járhatjuk végig, hanem bármilyen sorrendben (véletlen hozzáférésű – random access)
 - > Soronként tudjuk feldolgozni az eredményt
 - > Lekérhetjük az oszlopok típusát, az adatokat, és további információkat az eredményről (sorok/oszlopok száma, aktuális pozíció, stb...)
 - > Bizonyos Cursor leszármazottak automatikusan szinkronizálnak ha az eredményhalmaz változik
 - > Vagy képesek ekkor trigger metódust hívni egy beállított Observer objektumon

Cursor 2/2

- Eredményhalmaz feldolgozása
 - > Ha nincs találat, akkor `Cursor.getCount() == 0`
 - > Ha a query futtatása közben hiba lépett fel, akkor a Providerre van bízva annak kezelése, általában:
 - null-al tér vissza
 - Vagy kivételt dob
 - > Egyébként van eredmény

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()  
}
```

Adat beszúrás

- **ContentResolver.insert()** metódus
> (= SQL INSERT)
- Visszaadja a beszúrt elem Uri-ját
- Paramtérei:
 1. Provider CONTENT_URI
 2. A beszúrandó elem mezői egy ContentValues objektumba csomagolva

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)
```

Adatmódosítás

- **ContentResolver.update()** metódus
 - > (= SQL UPDATE)
- Visszaadja az érintett sorok számát
- Paraméterei:
 - > CONTENT_URI
 - > Új értékek egy **ContentValues** objektumban
 - > Szelekciós feltétel (változók helyén „?”)
 - > Szelekciós változók értékei

Adat törlése

- **ContentResolver.delete()**
 - > (= SQL DELETE)
- Visszaadja a törölt sorok számát
- Paraméterei:
 - > CONTENT_URI
 - > Szelekciós feltétel (változók helyén „?”)
 - > Szelekciós változók értékei

// naptárból törlés

```
contentResolver.delete(CalendarContract.Events.CONTENT_URI,  
    CalendarContract.Events._ID+"=599", null)
```

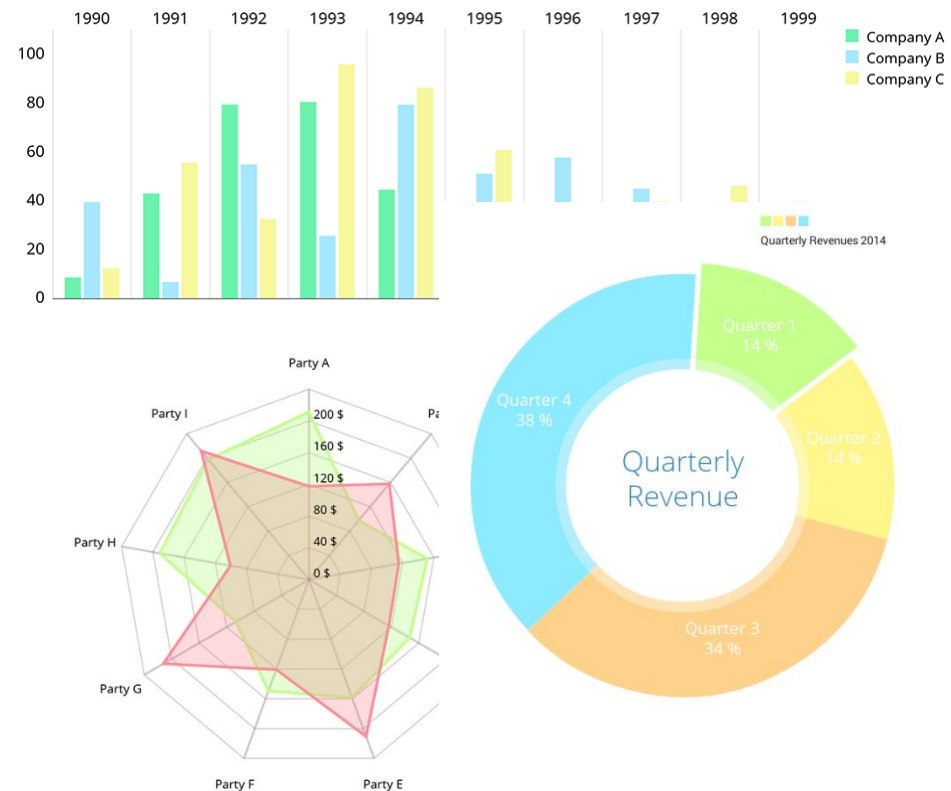
GRAFIKONOK RAJZOLÁSA

MPAndroidChart

- Gazdag grafikon rajzoló osztálykönyvtár
- Számptalan grafikon típus

- > LineChart
- > BarChart
- > PieChart
- > CandleStickChart
- > BubbleChart
- > ...

- <https://github.com/PhilJay/MPAndroidChart>



MPAndroidChart használat

1. Grafikon elhelyezése layout file-ba

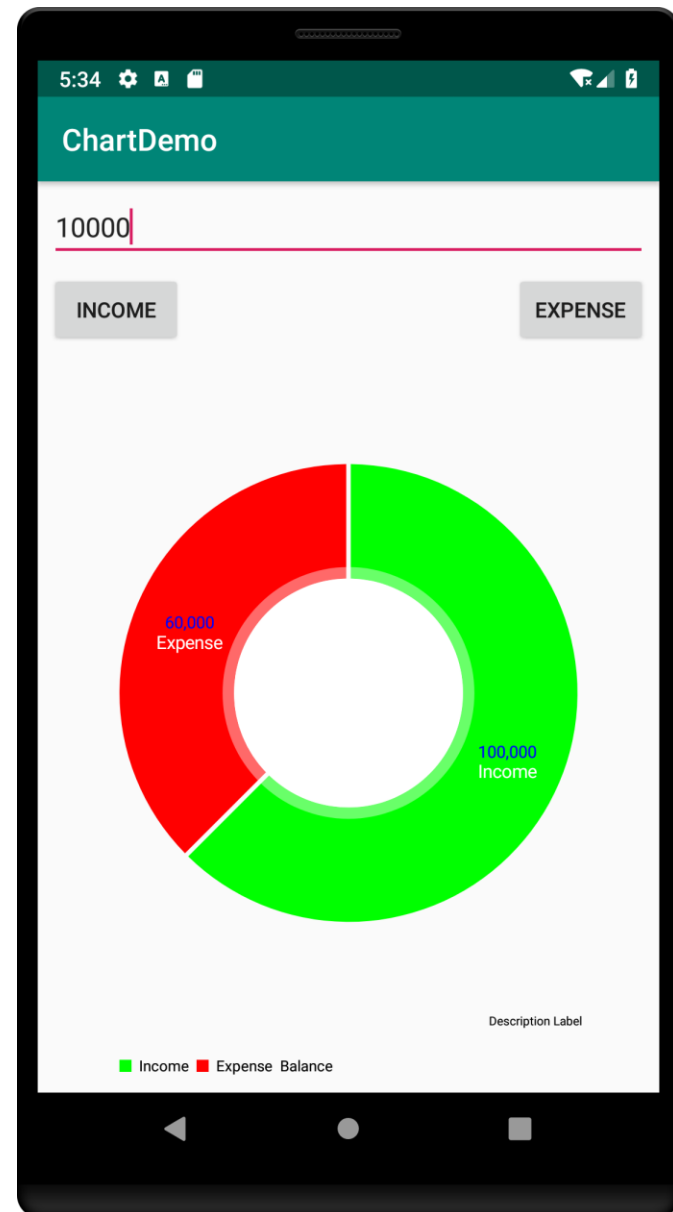
```
<com.github.mikephil.charting.charts.PieChart  
    android:id="@+id/chartBalance"  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    .../>
```

2. Grafikon beállítások: elforgatás, gesztusok kezelése, szöveg pozíciók, stb.

3. DataSet és entry-k (értékek) beállítása

4. Színsémák megadása értékekhez

5. Grafikon és DataSet összerendelése



Szenzorok használata

Kilépés a virtuális világból

Bevezetés

- A mai mobilok többre képesek a telefonálásnál és internet csatlakozásnál
- Rengeteg beépített szenzor
 - > Gyorsulásmérő
 - > Iránytű
 - > Fényerősség érzékelő
 - > Hőmérő
 - > Stb...

Bevezetés

- Új lehetőségek az interakció megvalósítására

- > Kiterjesztett valóság

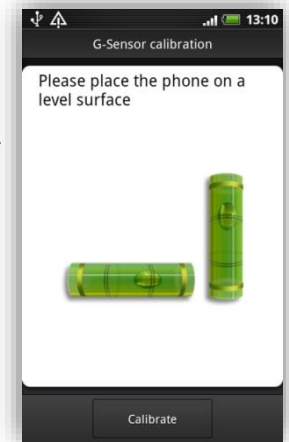
- > Mozgás alapú vezérlés

- > Stb...



Támogatott szenzorok

- Az Android absztrakt szenzor típusokat támogat
 - > **Sensor.TYPE_ACCELEROMETER**: háromtengelyes gyorsulásmérő, m/s^2 -ben adja vissza a pillanatnyi értékeket
 - > **Sensor.TYPE_GYROSCOPE**: elforgatás mértékét adja meg fokban, mindhárom tengelyre
 - > **Sensor.TYPE_LIGHT**: ambiens megvilágítást méri, egyetlen visszaadott értékének mértékegysége *lux*. Ezt használja az op.rendszer a képernyő fényerősségének automatikus beállításához



Támogatott szenzorok

- > **Sensor.TYPE_MAGNETIC_FIELD:** Mágneses erősség mérése három tengely mentén, microtesla egységekben. Iránytű alkalmazáshoz elengedhetetlen
- > **Sensor.TYPE_ORIENTATION:** elforgatás szenzor. Közvetlenül nem használjuk, a `SensorManager.getOrientation()` adja az orientációt
- > **Sensor.TYPE_PROXIMITY:** Visszaadja a telefon és a cél tárgy közti távolságot centiméterben. A telefon felvételekor (fülhöz emelés) az Android kikapcsolja a képernyőt, ezen szenzor segítségével



Támogatott szenzorok

- További egzotikus és származtatott szenzorok is támogatottak
 - Hőmérséklet, relatív páratartalom, légköri nyomás, elforgatás vektor, lineáris gyorsulás
- Speciális alkalmazás igények esetén használhatjuk őket
- Lekérdezhető, hogy milyen szenzorok elérhetők a telefonon
 - Egy absztrakt típusból akár több is jelen lehet ugyanazon a készüléken

Szenzorok elérése

- Rendszerszolgáltatás biztosítja a szenzorokkal történő kommunikációt, értékek lekérdezését

> **SensorManager**

- SensorManager elérése

```
val sensorManager =  
    getSystemService(SENSOR_SERVICE) as SensorManager
```

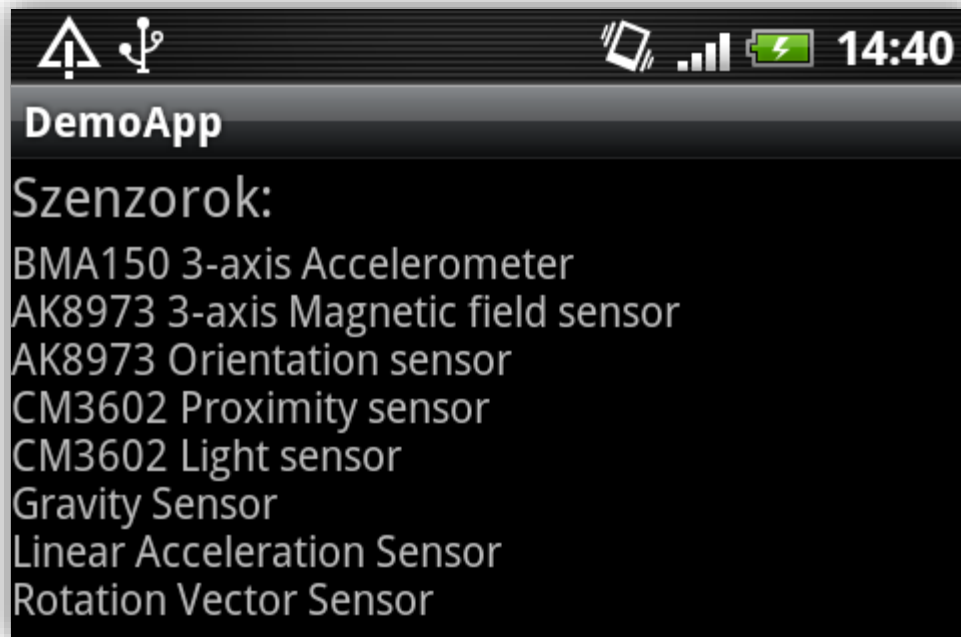
- Szenzor elkérése

```
val sensor = sensorManager.getDefaultSensor(  
    Sensor.TYPE_MAGNETIC_FIELD)  
sensorManager.registerListener(this, sensor,  
    SensorManager.SENSOR_DELAY_NORMAL)
```


Összes szenzor listázása

- Lekérhető a készülék összes szenzora

```
sensorManager.getSensorList(Sensor.TYPE_ALL).forEach {  
    tvStatus.append("${it.name}\n")  
}
```



Szenzorok használata

- Eseménykezelte módon,
SensorEventListener megvalósításával
 - > **onSensorChanged(SensorEvent)**: A szenzor által mért érték változásakor hívódik (új mérés). SensorEvent-ből kinyerhető értékek:
 - Szenzor, amelyik triggerelte
 - Pillanatnyi mérési pontosság (*alacsony, közepes, magas, nem megbízható* – kalibráció szükséges)
 - Mért értékek tömbje (FloatArray). A szenzor típusa határozza meg hogy hogyan kell értelmezni a tartalmát
 - A mérés nanosec pontosságú időbélyege

Szenzorok használata

- Az eseménykezelő beállításakor megadhatjuk, hogy milyen gyakran szeretnénk mérni a szenzor értéket

`SensorManager.DELAY_(FASTEST|GAME|NORMAL|UI)`

- Erőforrás igényes feladat a szenzor folyamatos lekérdezése, válasszuk a célunkhoz megfelelő legalacsonyabbat
- És állítsuk le a frissítést, ha nem szükséges tovább futnia (onPause-ban)

Példa

```
class MainActivity : AppCompatActivity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager

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

        sensorManager = getSystemService(SENSOR_SERVICE) as SensorManager

        ...
    }

    override fun onStop() {
        super.onStop()
        sensorManager.unregisterListener(this);
    }

    private fun listSensors() {
        sensorManager.getSensorList(Sensor.TYPE_ALL).forEach {
            tvStatus.append("${it.name}\n")
        }
    }

    private fun startSensor() {
        val sensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
        sensorManager.registerListener(this, sensor,
            SensorManager.SENSOR_DELAY_NORMAL)
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    }

    override fun onSensorChanged(event: SensorEvent) {
        tvStatus.setText("Magnetó: ${event.values[0]}")
    }
}
```

Kilépés a virtuális térből

- Mostanában az a trendi, ha az alkalmazás ki tud lépni a virtuális világból
- Nem (csak) úgy működik, hogy nézzük a kirajzolt pixeleket és nyomkodjuk a képernyőt
- Szenzorok használatával képes kapcsolatot teremteni a külvilággal
- Iránytű, gyorsulásmérő és elforgatás szenzor kombinációival érhető el a legtöbb

Kilépés a virtuális térből

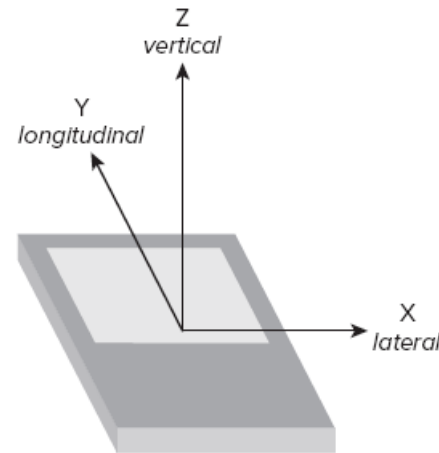
- Ezek segítségével
 - > Tudjuk, hogy mi az orientáció
 - > Mennyire van elforgatva a készülék
 - > Melyik égtáj felé néz a felhasználó
 - > Merre mozog a készülék
- További érzékelőkkel kiegészíthető
 - > Hol van (GPS)
 - > Mit lát a kamera
 - > Stb...

Kilépés a virtuális térből

- Rengeteg lehetőség az innovációra, a szenzorokat ügyesen használó alkalmazások általában átütő sikert érnek el, például:
 - > Iránytű, gyorsulásmérő, GPS és kamera használatával **kiterjesztett valóság** készíthető
 - > Gyorsulásmérő segítségével érzékelhetjük az **ütközéseket** – baleset esetén jelezhetünk
 - > Mozgás mint input – **Mobil Wii**

Gyorsulásmérő használata

- Három tengelyen méri a gyorsulást (nem sebességet!)
- A telefon háton fekvő helyzetében
 - > X tengely: jobbra-balra
 - > Y tengely: előre-hátra
 - > Z tengely: fel-le
- Fontos tudni: nyugvó állapotban a Z tengely a gravitációs gyorsulást méri (9.81 m/s^2)



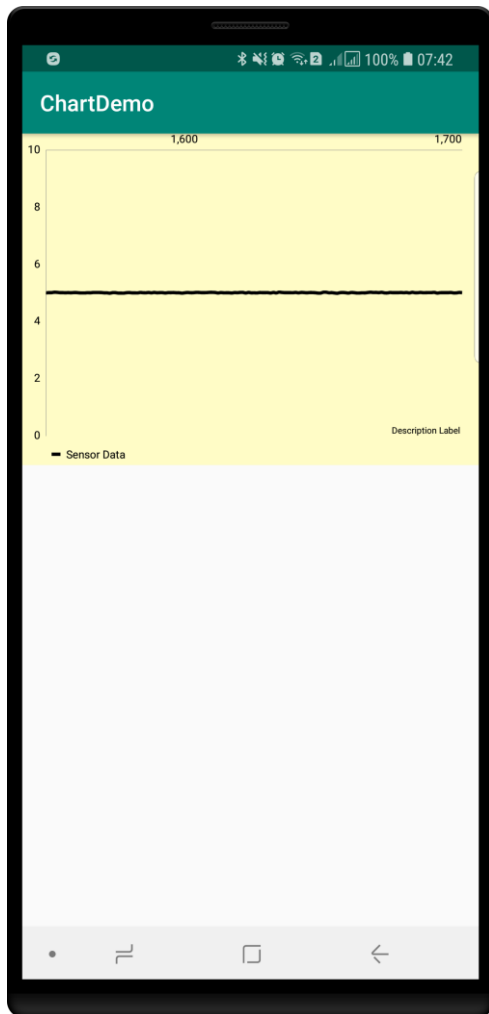
Gyorsulásmérő használata

- Mérési adatok értelmezése (G-erő számítása = három gyorsulási érték négyzetösszegének gyöke mínusz gravitáció)

```
override fun onSensorChanged(event: SensorEvent) {  
  
    val accX = event.values[0].toDouble()  
    val accY = event.values[1].toDouble()  
    val accZ = event.values[2].toDouble()  
  
    var origin = Math.sqrt(  
        Math.pow(accX, 2.0) +  
        Math.pow(accY, 2.0) +  
        Math.pow(accZ, 2.0)  
    )  
    origin = Math.abs(origin - SensorManager.STANDARD_GRAVITY)  
}
```

Szenzorok használata

- Absztrakt szenzor típusok
- Mérési eredmények eseménykezelte módon
- Legritkább szükséges frissítést használjuk
- Állítsuk le onPause()-ban
- Nem a szenzor érték kinyerése a nehéz, hanem az értelmes felhasználása
- http://developer.android.com/guide/topics/sensors/sensors_motion.html



Gyakoroljunk!

- Készítsünk egy alkalmazást, amely megjeleníti a gyorsulásmérő szenzor adatait.
- Ábrázoljuk az értékeket egy grafikonon!

Köszönöm a figyelmet!



peter.ekler@aut.bme.hu



Department of
Automation and
Applied Informatics