

# ContentProvider, összetett felhasználói felület fejlesztés, külső UI library-k

Ekler Péter

BME VIK AUT, AutSoft

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



# Tematika

1. Service komponens
2. ContentProvider, Komplex felhasználói felületek
3. Játékfejlesztés
4. Multimédia megoldások
5. További kommunikációs megoldások
6. Biztonságos alkalmazások
7. Andorid TV és Wear fejlesztés
8. Android 9 újdonságok és további helyfüggő szolgáltatások
9. Tesztelési lehetőségek
10. Alkalmazás publikálás, karbantartás (CI/CD)

# Tartalom

- ContentProvider
- ConstraintLayout - haladó
- Drag&Drop általános kezelése
- Grafikonok rajzolása
- Gyakran használt külső UI library-k áttekintése

# 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**

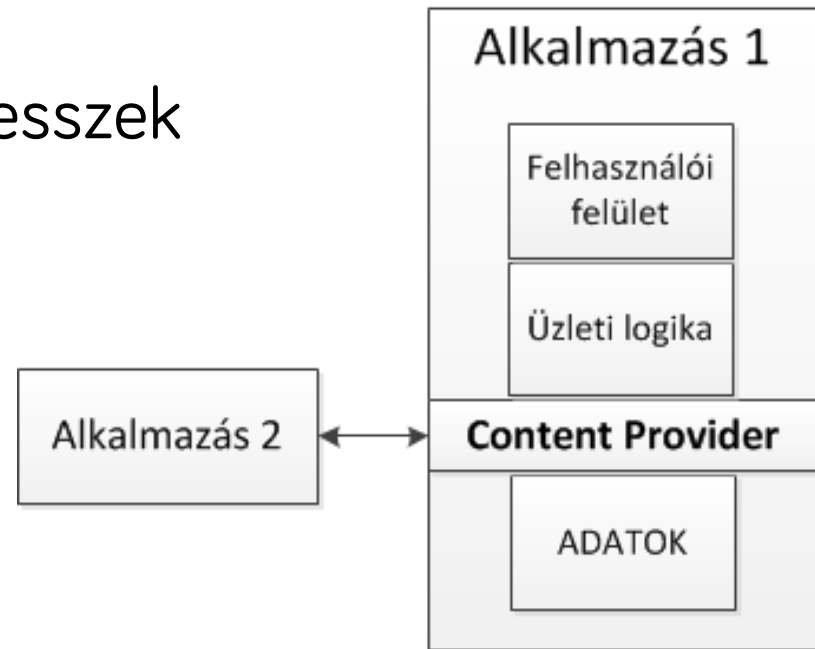
- > Bármikor elérhetetlenné válhat
- > Látható és módosítható, akár törölhető a felhasználó által

# 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**



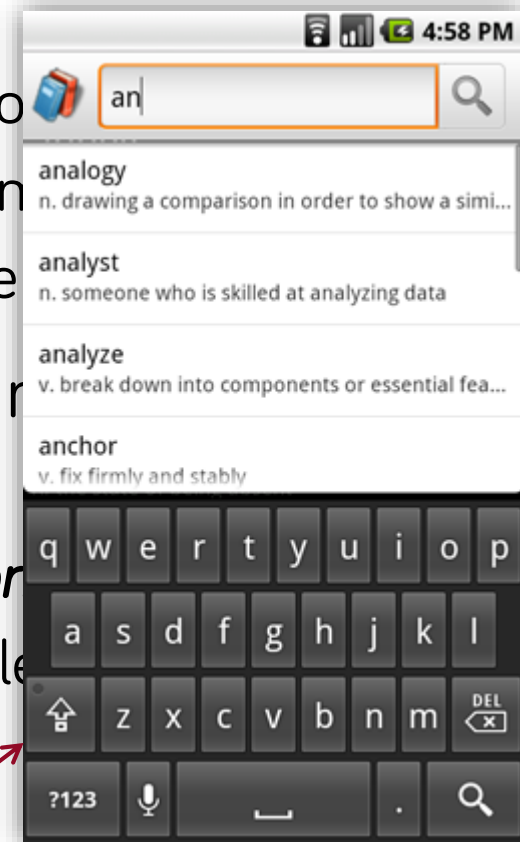
# Content Provider – Mikor?

- 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

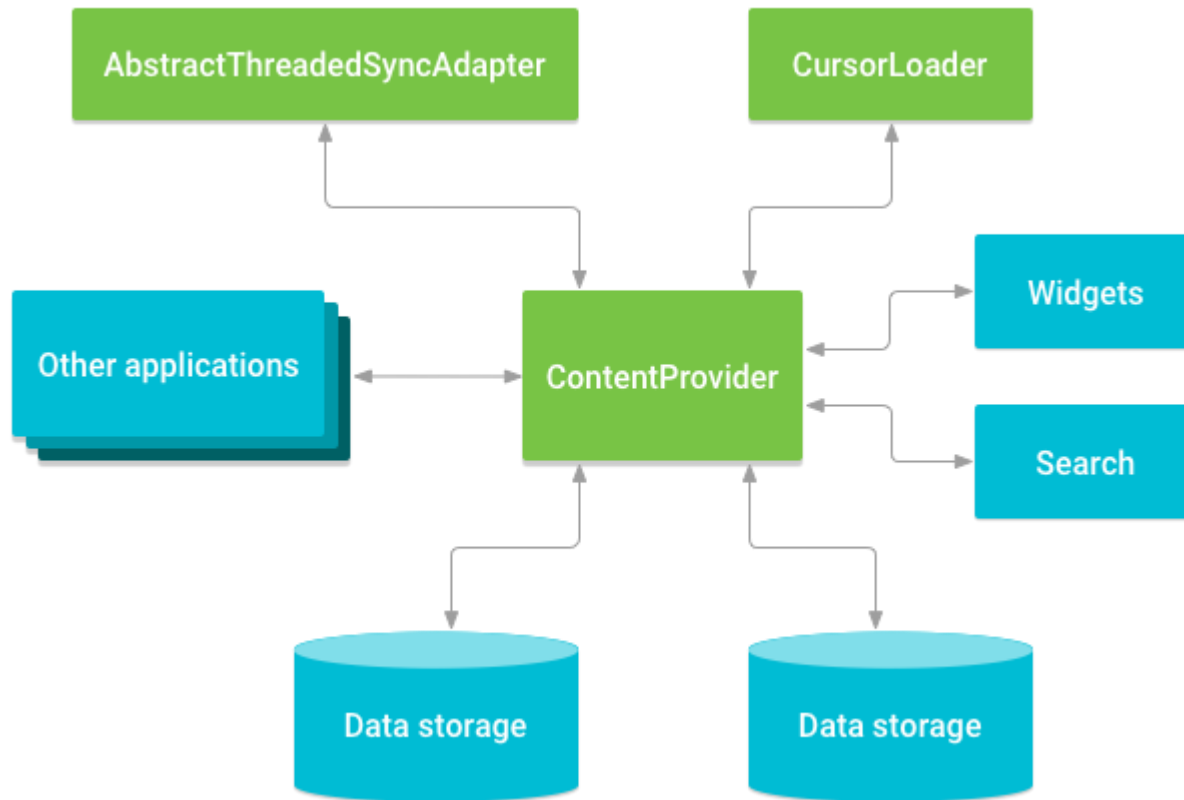


# Content Provider – Mikor?

- Nem kötelező Content Provider-t írni...
  - > Ha nem akarunk más alkalmazásokkal adatot osztani
  - > 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

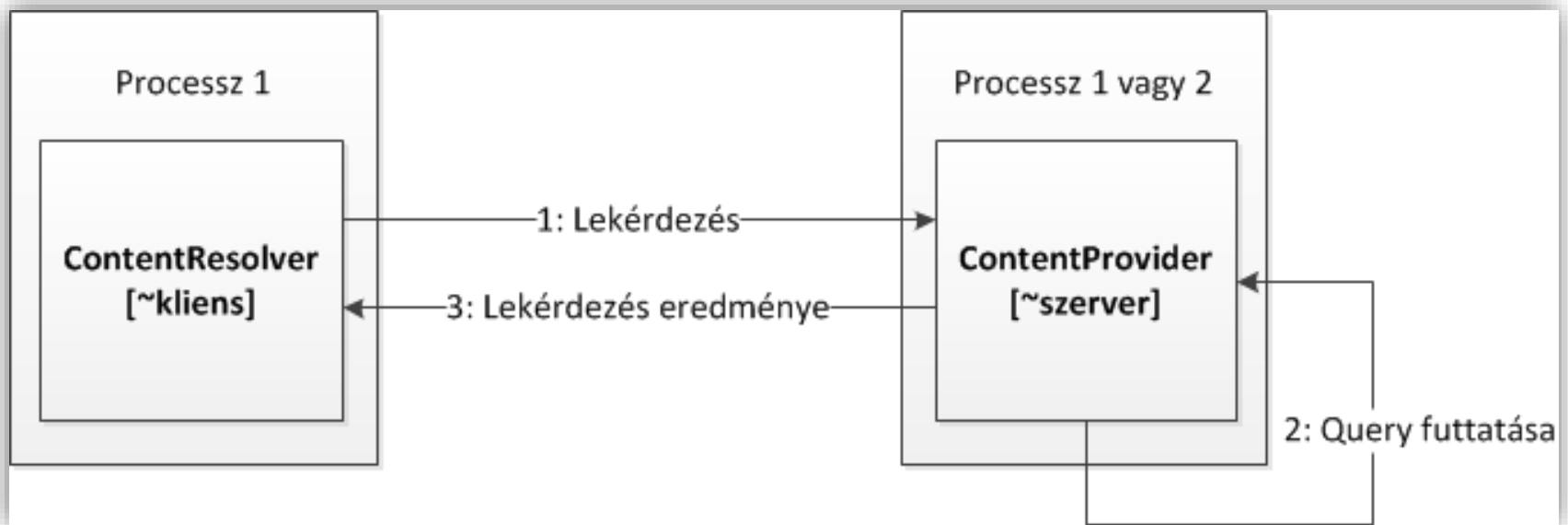
- Content Provider-től kérdezhetjük le az adatokat
- A komponens ami képes a lekérdezések futtatására és a válasz feldolgozására: **ContentResolver**
  - > Csak ez tudja lekérdezni a Content Providert
  - > Lehet akár ugyanabban, akár másik alkalmazásban (processzben)
  - > A kommunikációhoz szükséges 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.DESRIPTION, "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)
```

# CONTENT PROVIDER KÉSZÍTÉSE

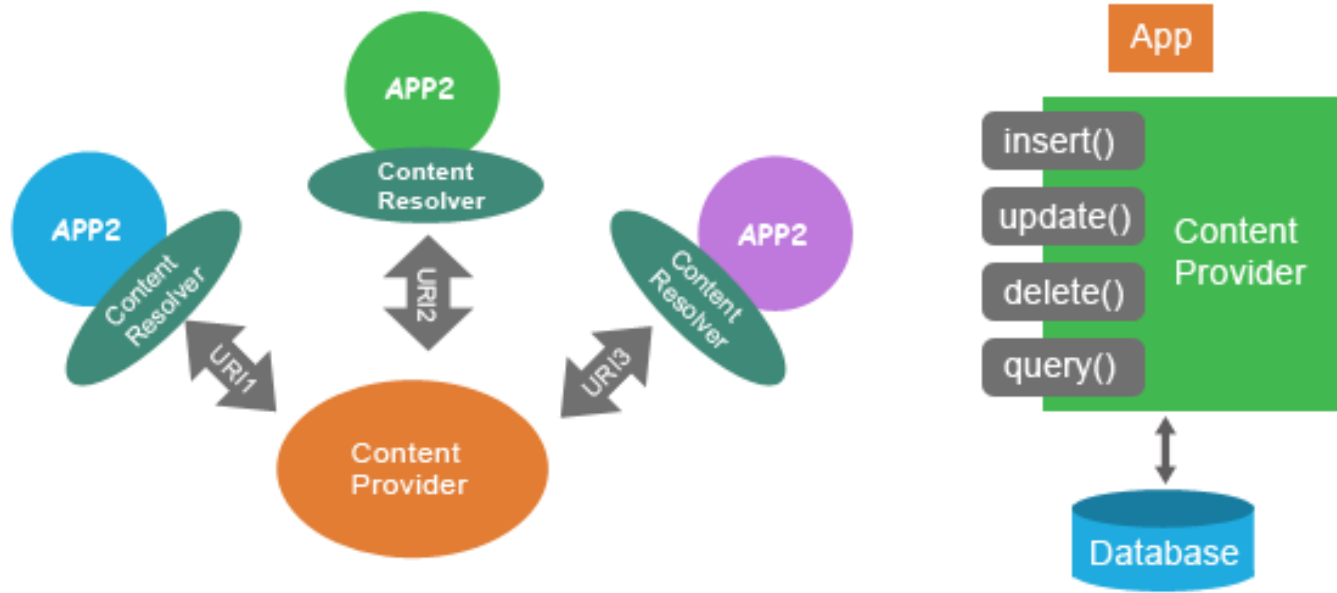
# ContentProvider készítés

- Mikor használjunk ContentProvidert?
  - > Ha más alkalmazásokkal komplex adatokat vagy fileokat kell megosztani
  - > Komplex adatokat kell egyik vagy másik alkalmazásba másolni
  - > Ha egyedi keresési javaslatokat akarunk ajánlani a search framework-ön keresztül
  - > Ha az alkalmazás adatait widget-ek számára elérhetővé akarjuk tenni
  - > AbstractThreadedSyncAdapter, CursorAdapter, vagy CursorLoader használata esetén
- Mikor nem kell használni?
  - > Ha az adatok használata teljesen az alkalmazáson belül marad



# Content Provider architektúra 2/2

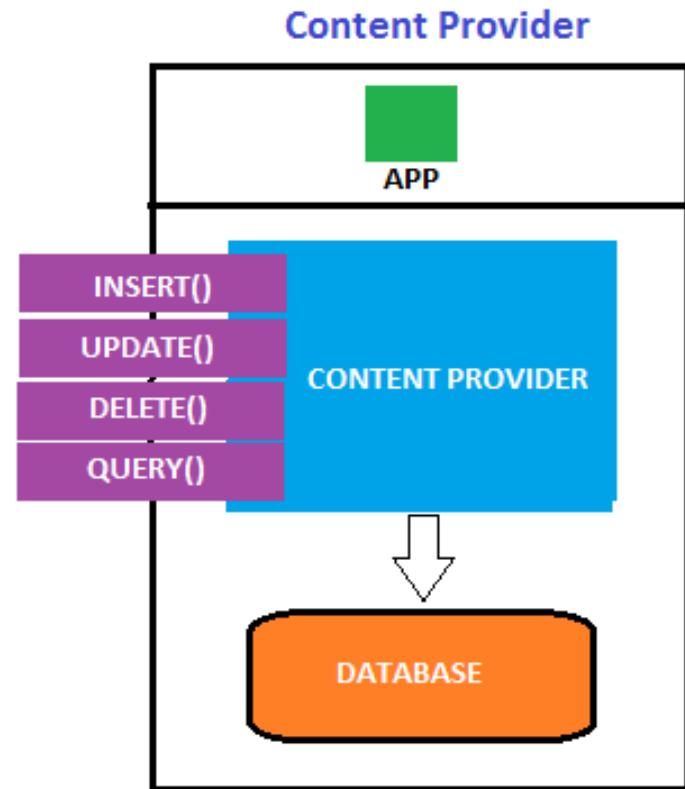
- ContentResolver osztály oldja fel az URI-kat
  - Emiatt kell minden ContentProvidert regisztrálni a Manifest fileba



forrás: <https://www.oodlestechnologies.com/blogs/Content-Providers-in-Android>

# Content Provider architektúra 1/2

1. Saját ContentProvider-ből leszármazott osztály
2. Contract meghatározása
  - > `CONTENT_AUTHORITY` (package név általában)
  - > `CONTENT_URI` (pl. `content://recipes`)
3. Insert, update, delete és query függvények felüldefiniálása
4. Adat elérés megvalósítása
5. Komponens regisztrálása Manifest-ben
  1. Provider osztály neve
  2. Authority



forrás: <https://resources.infosecinstitute.com/android-hacking-security-part-2-content-provider-leakage/#gref>

# Saját provider írása

```
public static final Uri CONTENT_URI =  
    Uri.parse("content://com.example.codelab.transporationprovider");
```

`content://com.example.codelab.transporationprovider/train`

`content://com.example.codelab.transporationprovider/air/domestic`

`content://com.example.codelab.transporationprovider/air/international`

# Saját provider írása

- Felüldefiniálható függvények:

- query()
  - insert()
  - update()
  - delete()
  - getType()

- Manifest.xml-ben be kell jegyezni a provider-ünket
  - > A name attribútum a Provider osztály minősített neve
  - > Az authorities attribútum a content:// uri része (path nélkül!)

# Saját provider írása

```
<provider
```

```
  name="com.example.railprovider.TransportationProvider"
```

```
  authorities="com.example.railprovider" ... />
```

```
</provider>
```

DE:

com.example.railprovider/trains/ nem kell

# Gyakoroljunk

- Készítsünk egy alkalmazást, amely:
  - > Recepteket kezel, tárol
  - > *ContentProvider*-en keresztül csatornát biztosít a receptekhez
- Készítsünk egy külön főző alkalmazást, amely eléri a recepteket a másik alkalmazás *ContentProvider*-én keresztül

# Hasznos linkek

- Saját Content Provider készítése
  - > <http://developer.android.com/guide/topics/providers/content-provider-creating.html>
- Naptár Provider részletes leírás
  - > <http://developer.android.com/guide/topics/providers/calendar-provider.html>
- Aszinkron Cursor használat Loader-rel (3.0-tól)
  - > <http://developer.android.com/guide/topics/fundamentals/loaders.html>
- Nem dokumentált Providerek használatának veszélyei
  - > <http://android-developers.blogspot.com/2010/05/be-careful-with-content-providers.html>
- <provider> AndroidManifest elem referencia
  - > <http://developer.android.com/guide/topics/manifest/provider-element.html>

# ConstraintLayout

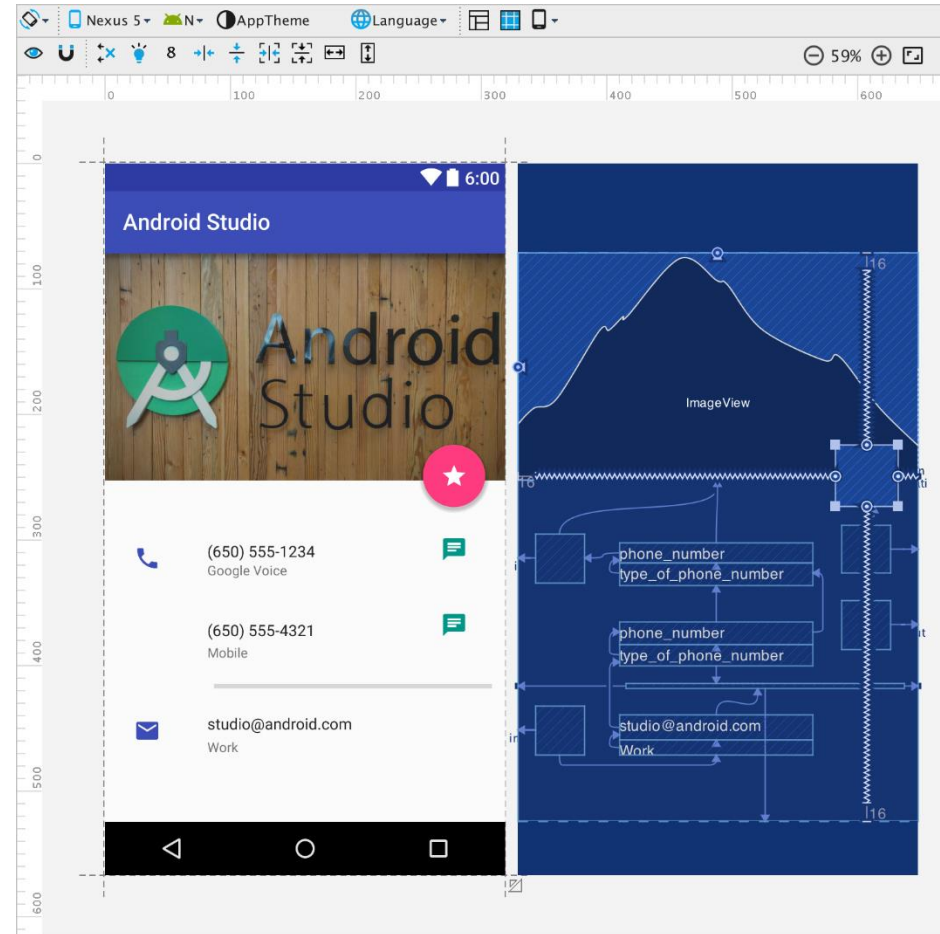


# ConstraintLayout bevezetés

- ConstraintLayout általános áttekintése
- View hozzáadása
- Szabálytípusok
- Szabályok törlése
- View méreteinek beállítása
- View láncok
- ConstraintLayout teljesítmény
- Gyakorlatok

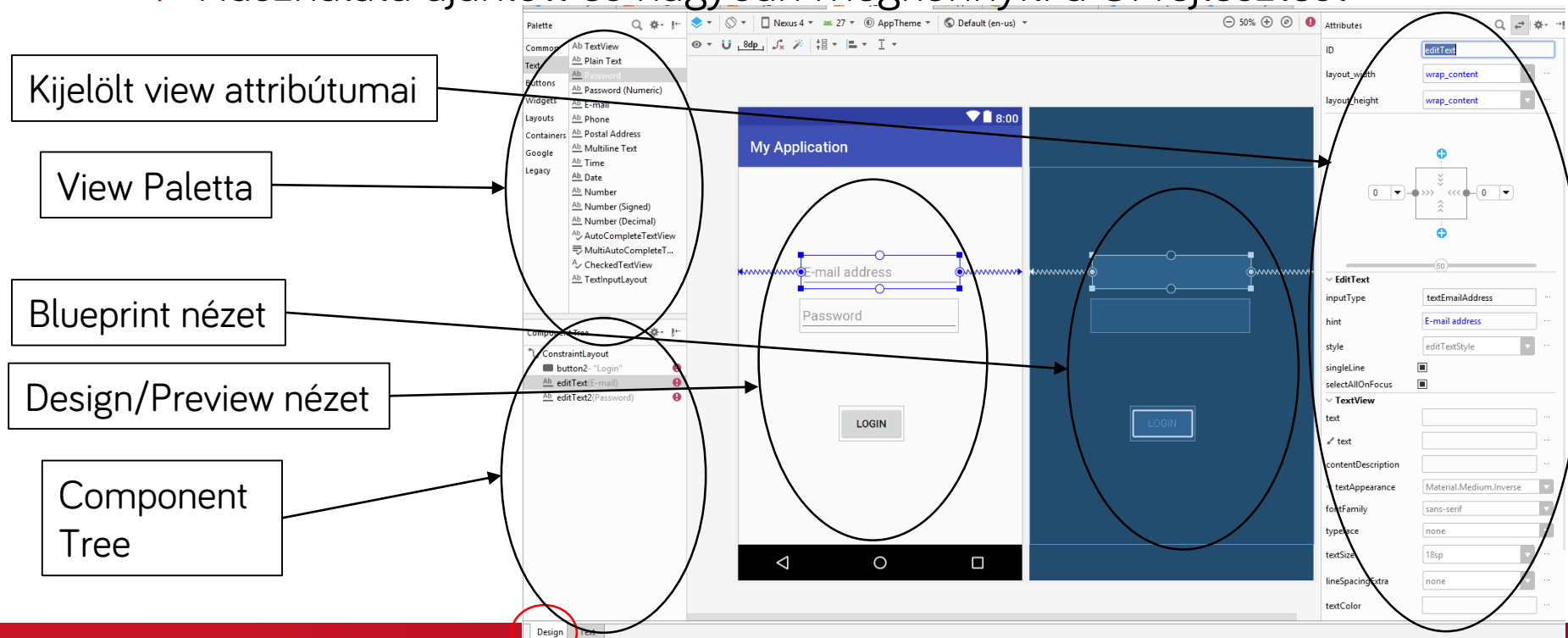
# ConstraintLayout

- Továbbgondolt RelativeLayout
- iOS AutoLayout-hoz hasonló
- Lapos view hierarchia
- Relatív pozíciók
- Erős Layout Editor támogatás
- Gyorsabb UI fejlesztés
- Android 2.3-tól (API Level 9)



# ConstraintLayout fejlesztő eszközök

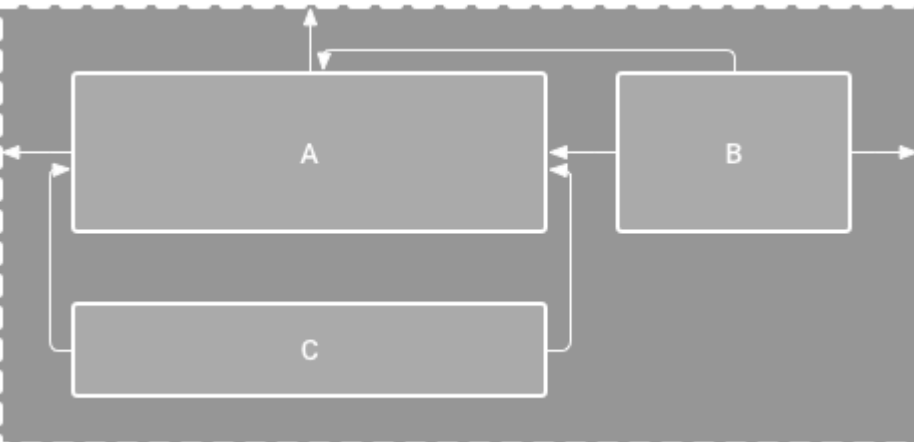
- Külön support library a ConstraintLayout-nak
  - > *gradle import: compile 'com.android.support.constraint:constraint-layout:1.1.3'*
- Layout Editor
  - > A Layout Editor és a ConstraintLayout egymásnak voltak fejlesztve
  - > Használata ajánlott és nagyban megkönnyíti a UI fejlesztést



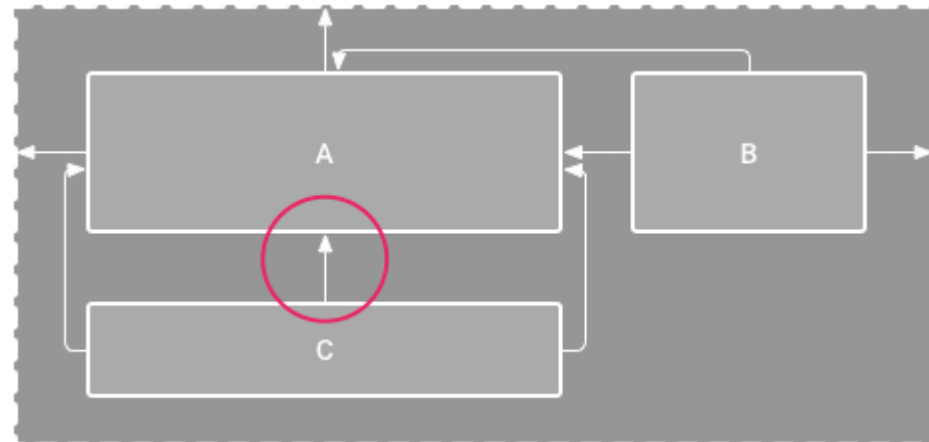
# Relatív view pozíciók

- Pozíció megadáshoz szükséges:
  - > **Legalább** egy horizontális és **legalább** egy vertikális „szabály” (constraint)
- Minden szabály egy **kapcsolat** (connection)/**igazítás** (alignment):
- Android Studio jelzi a hiányzó szabályokat

Hibás:



Helyes, mert C tudja, hogy vertikálisan hol kell elhelyezkedjen (A alatt van)



# View hozzáadása ConstraintLayouthoz

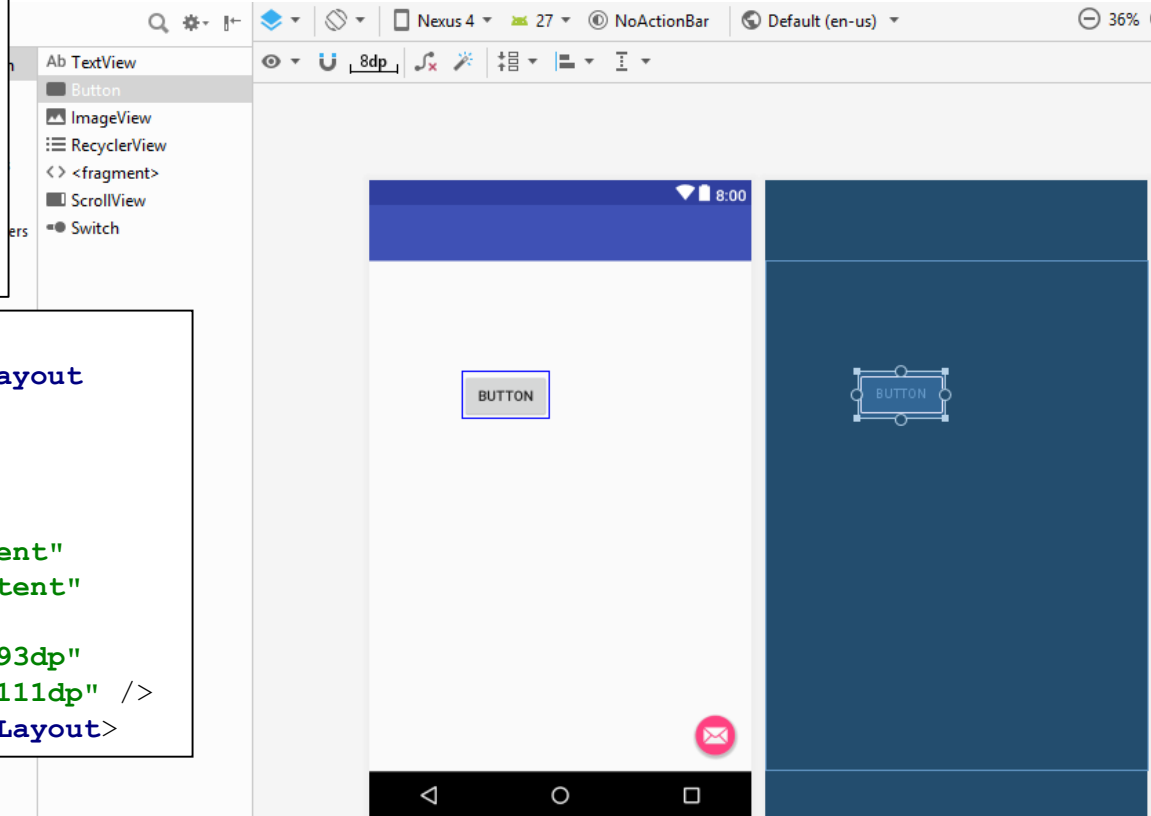
1. Húzz be egy view-t a *Palette* ablakból

2. Katt a hozzáadott view-ra. Figyeld meg:

- a) A sarkainál átméretező négyzet alakú kezelőket
- b) A kör alakú kezelőket az oldalakon a szabályok megadására

*This view is not constrained.  
It only has designtime  
positions, so it will jump to  
(0,0) at runtime unless you  
add the constraints*

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    . . .
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        tools:layout_editor_absoluteX="93dp"
        tools:layout_editor_absoluteY="111dp" />
</android.support.constraint.ConstraintLayout>
```



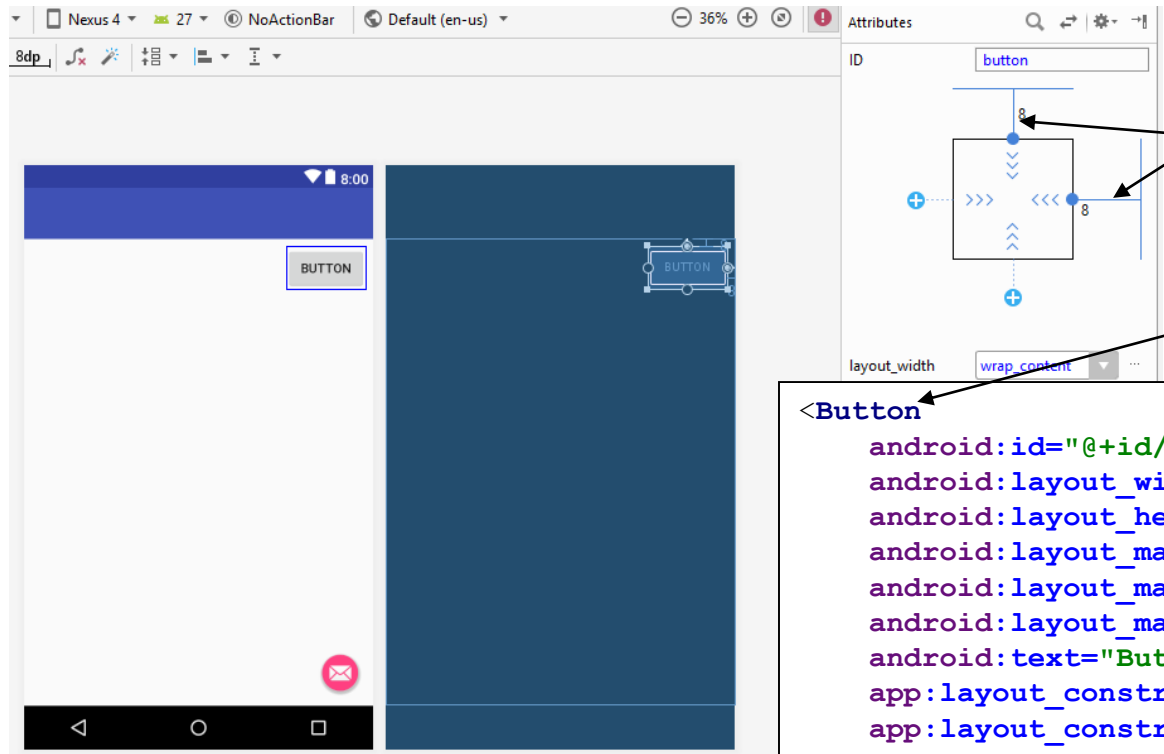
# Szabálytípusok

1. Függőleges/vízszintes szélek **kapcsolása** szülő függőleges/vízszintes **széleihez**
2. Két view **sorrendjének** megadása
3. **Igazítás** más view függőleges/vízszintes **széleihez**
4. **Szövegvonala igazítása** másik view szövegvonalaához
5. Láthatatlan függőleges/vízszintes **vezetővonalhoz** (guideline) való **kapcsolás**
6. Láthatatlan függőleges/vízszintes **akadályhoz** (barrier) való **kapcsolás**

# 1. Kapcsolási szabály szülő széleihez

**Feladat:** Pozícionáljuk a gombot a jobb felső sarokba (a jobb oldali sötétkék tervezőben)

1. **Horizontális szabály megadása:** a gomb felső oldalán fogjuk meg a kör alakú kezelőt és húzzuk a megjelenő nyilat szülő felső vízszintes széléig, azaz a horgonypontig (*anchor point*)
2. **Vertikális szabály megadása:** a gomb bal oldalán fogjuk meg a kör alakú kezelőt és húzzuk a megjelenő nyilat szülő bal oldali függőleges széléig



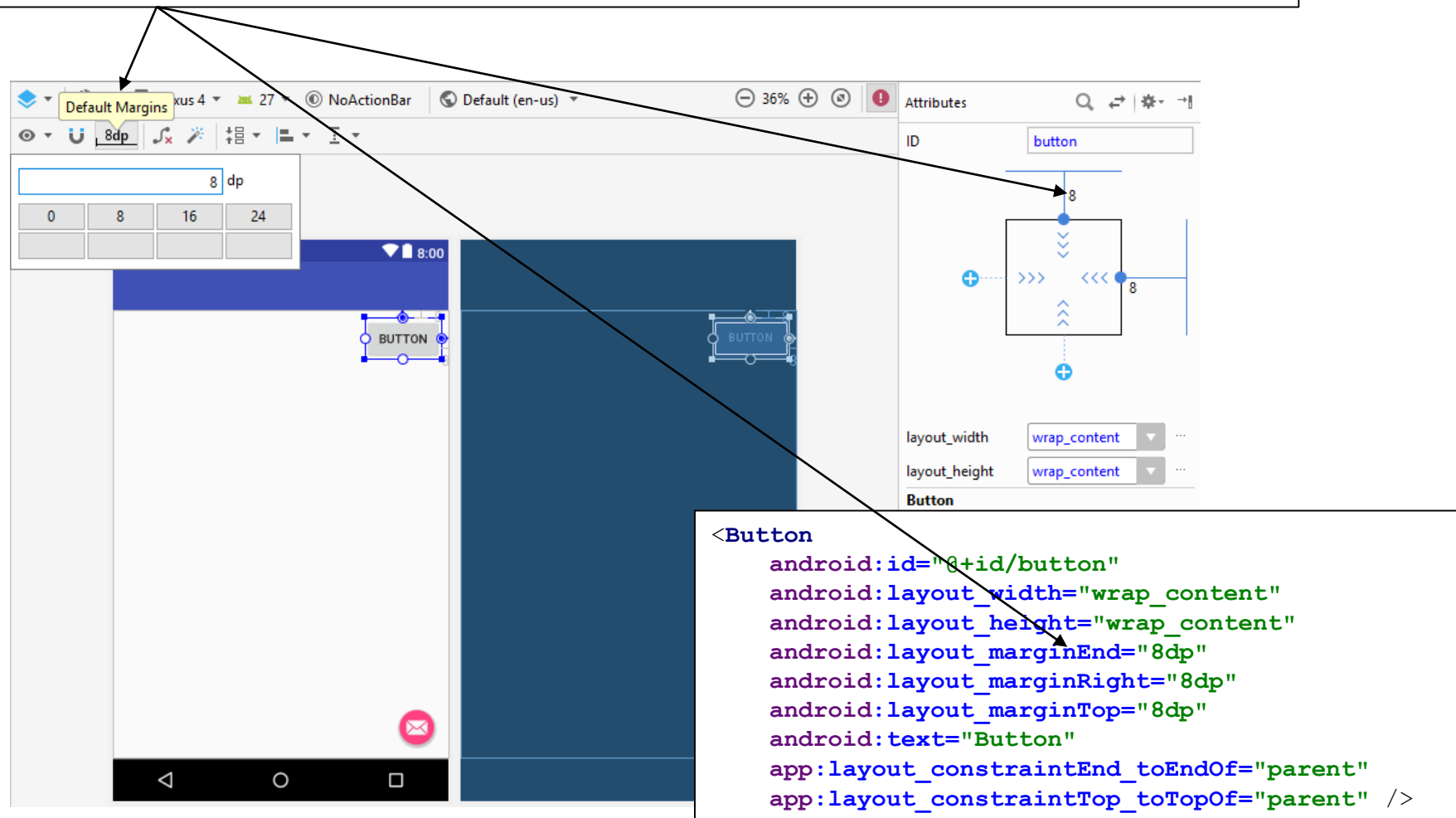
A vonalak jelzik, hogy van 1 vertikális és 1 horizontális szabály is megadva

Nincs több hibaüzenet

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

# Szabály hozzáadása view-hoz – default margó méret

Szabály létrehozásakor minden view margót kap, melynek default értéke állítható



The screenshot shows the Android Studio IDE with a 'Default Margins' dialog box open. The dialog has a text input field set to '8 dp' and a grid of buttons for values 0, 8, 16, and 24. The 'Attributes' panel on the right shows the 'button' view with its layout attributes. The 'layout\_marginEnd' and 'layout\_marginRight' attributes are both set to '8dp'. The 'layout\_marginTop' attribute is also set to '8dp'. The 'layout\_width' and 'layout\_height' attributes are set to 'wrap\_content'. The 'Button' text attribute is set to 'Button'. The XML code for the button is shown in a separate box below the dialog.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



# Szabály hozzáadása view-hoz – margók állítása

A margók view-nként is állíthatóak

The screenshot displays the Android Studio interface. On the left, a mobile emulator shows a blue header bar and a button labeled 'BUTTON'. On the right, the 'Attributes' panel for the selected 'button' widget is visible. The 'ID' is 'button'. Below it, a diagram shows the widget's margins: top (0), left (8), bottom (16), and right (24). The right margin is highlighted with a blue dot and a value of 8. Below the diagram, the 'layout\_width' and 'layout\_height' are set to 'wrap\_content'. Under the 'Button' section, 'style' is 'buttonStyle', 'background' is empty, 'visibility' is 'none', and 'onClick' is 'none'. Under the 'TextView' section, 'text' is 'Button', 'textAppearance' is 'mpat.Widget.Button', and 'contentDescription' is empty. The 'Favorite Attributes' section is at the bottom.

Attributes

ID button

layout\_width wrap\_content

layout\_height wrap\_content

**Button**

style buttonStyle

background

visibility none

onClick none

**TextView**

text Button

text

contentDescription

textAppearance mpat.Widget.Button

**Favorite Attributes**

## 2. Sorrend szabály 1 / 2

Feladat: Legyen egy TextView a gomb bal oldalán, egy Switch pedig a TextView fölött!

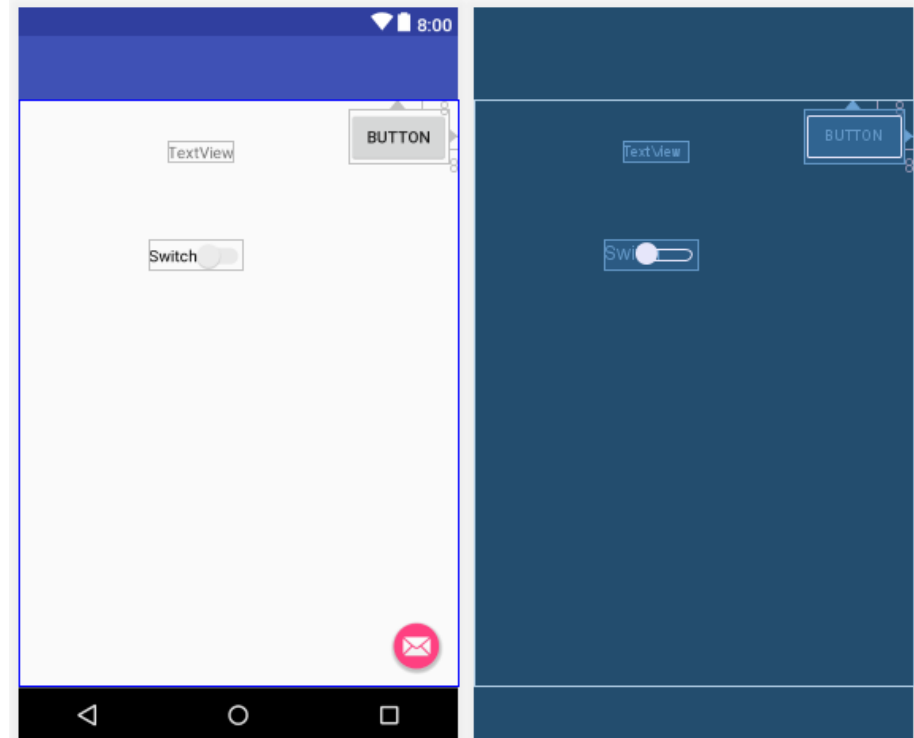
1. Behúzzunk egy TextView-t és egy Switch-et

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

    <Button
        . . . />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        tools:layout_editor_absoluteX="130dp"
        tools:layout_editor_absoluteY="36dp" />

    <Switch
        android:id="@+id/switch1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Switch"
        tools:layout_editor_absoluteX="114dp"
        tools:layout_editor_absoluteY="122dp" />
</android.support.constraint.ConstraintLayout>
```

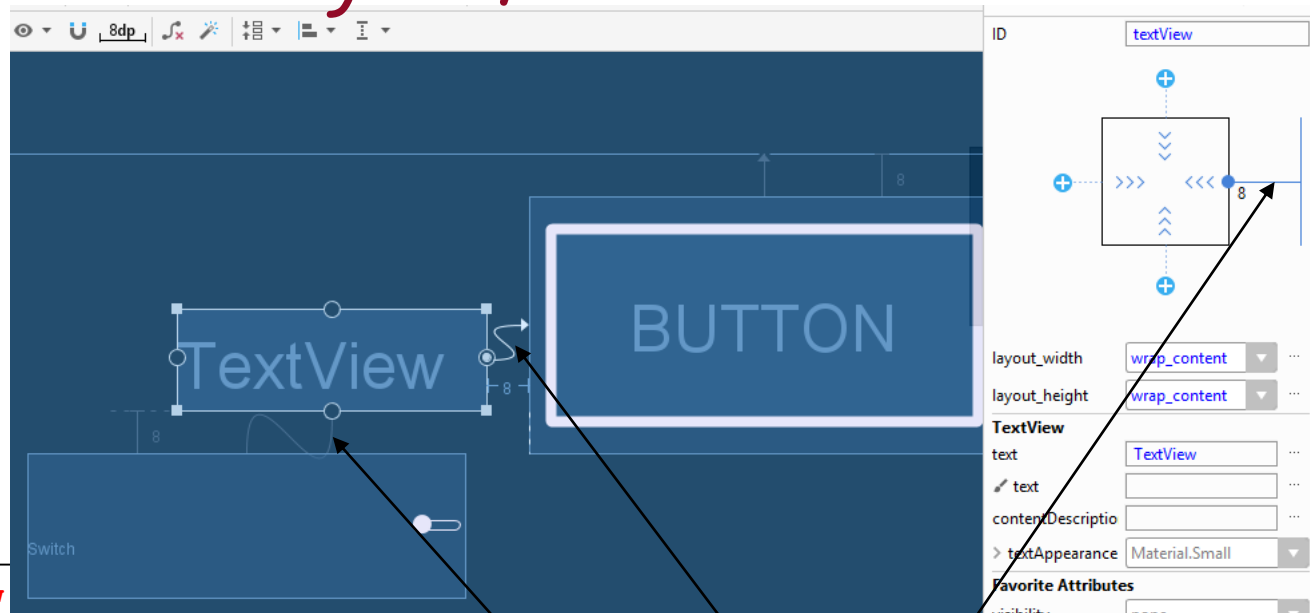


## 2. Sorrend szabály 2 / 2

**Feladat:** Legyen egy TextView a gomb bal oldalán, egy Switch pedig a TextView alatt

2. A TextView jobb oldalán levő kör alakú kezelőt megfogjuk, a megjelenő nyilat a gomb jobb oldalán levő kör alakú kezelőjéig húzzuk

3. A Switch felső oldalán levő kör alakú kezelőt megfogjuk, a megjelenő nyilat a TextView alsó oldalán levő kör alakú kezelőjéig húzzuk



<TextView

```
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginEnd="8dp"
android:layout_marginRight="8dp"
android:text="TextView"
app:layout_constraintEnd_toStartOf="@+id/button"
tools:layout_editor_absoluteY="29dp" />
```

<Switch

```
android:id="@+id/switch1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="8dp"
android:text="Switch"
app:layout_constraintTop_toBottomOf="@+id/textView"
tools:layout_editor_absoluteX="194dp" />
```

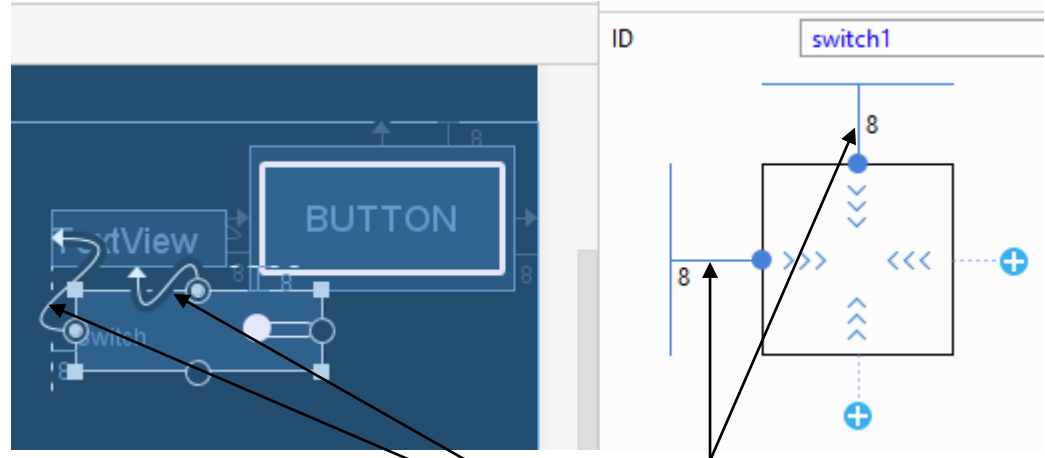
Látható, hogy a TextView-nak lett horizontális szabálya

a Switch-nek pedig vertikális szabálya

# 3. Igazítás más view széleihez

**Feladat:** Az előző példában a Switch-nek nincs horizontális, a TextView-nek nincs vertikális szabálya. Megoldásként először igazítsuk a Switch-et a TextView bal széléhez!

A Switch bal oldalán levő kör alakú kezelőt megfogjuk, a megjelenő nyilat a TextView bal oldalán levő kör alakú kezelőjéig húzzuk



A Switch-nek mostmár van vertikális és horizontális szabálya is

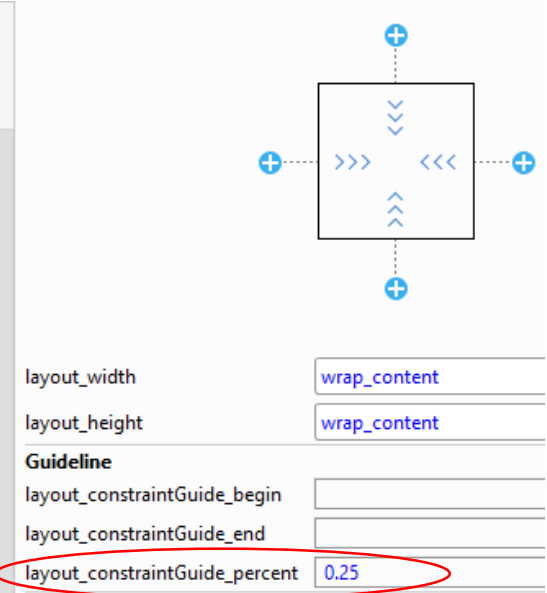
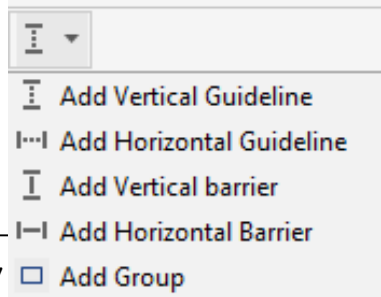
```
<Switch
    android:id="@+id/switch1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Switch"
    app:layout_constraintStart_toStartOf="@+id/textView"
    app:layout_constraintTop_toBottomOf="@+id/textView" />
```

# 5. Kapcsolási szabály vezetővonalhoz 1/2

**Feladat:** Az előző példában a TextView-nak nincs vertikális szabálya.  
Megoldásként igazítsuk a TextView-t a az alján egy vízszintes vezetővonalhoz (guideline), mely a képernyő felső részén, 25%-nál helyezkedik el!

1. Adjunk hozzá új vízszintes vezetővonalat az eszköztárban elérhető ikonra kattintva:

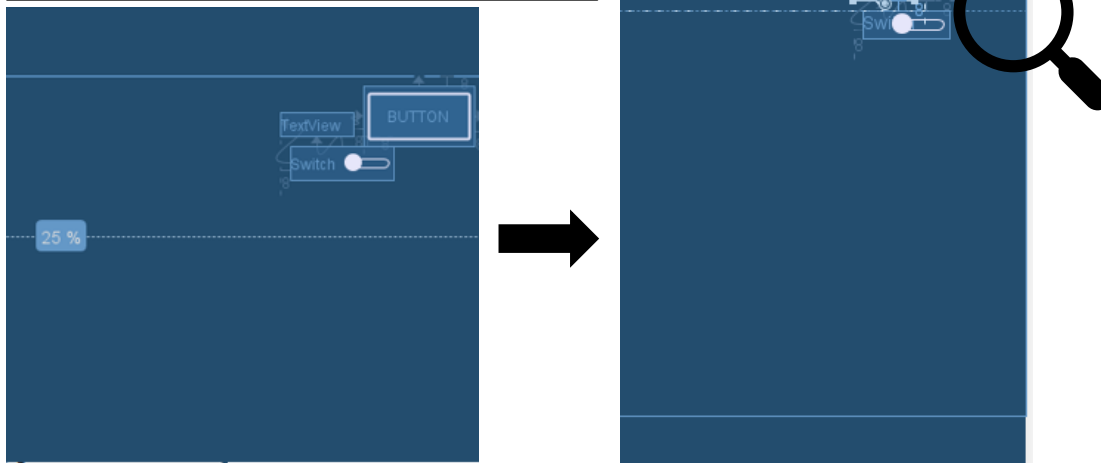
2. Jelöljük ki a vezetővonalat egy kattintással, aztán kattintsunk az elején levő körre, amivel kiválaszthatjuk, hogy milyen módon szeretnénk megadni a mértékegységet (dp valamelyik szélétől vagy %) (ez a művelet vízszintes vonalnál nem mindig működik, olyankor az attribútumoknál adjuk meg a kívánt méretet



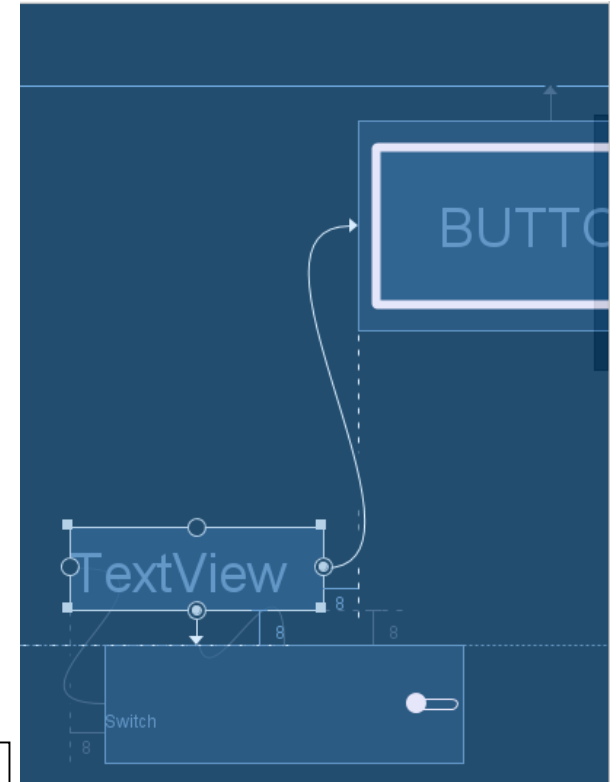
```
<android.support.constraint.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.25"
/>
```

# 5. Kapcsolási szabály vezetővonalhoz 2/2

3. A TextView alsó oldalán levő kör alakú kezelőt megfogjuk, a megjelenő nyilat a vezetővonalhoz húzzuk



```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:text="TextView"
    app:layout_constraintBottom_toTopOf="@+id/guideline"
    app:layout_constraintEnd_toStartOf="@+id/button" />
```



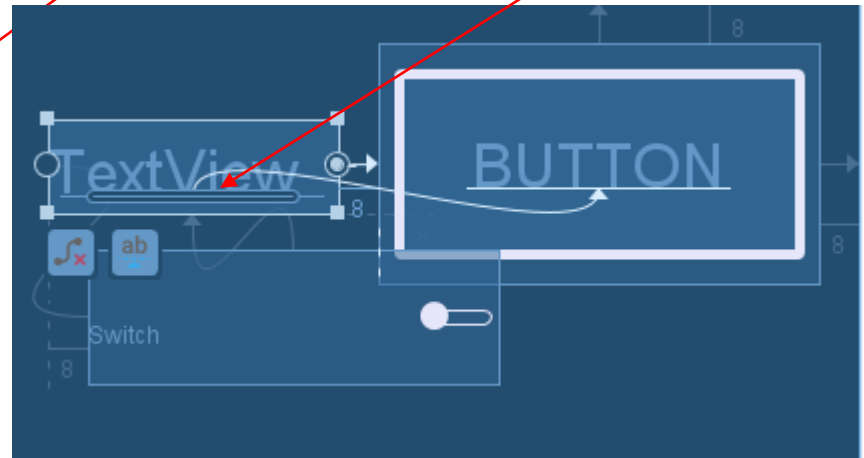
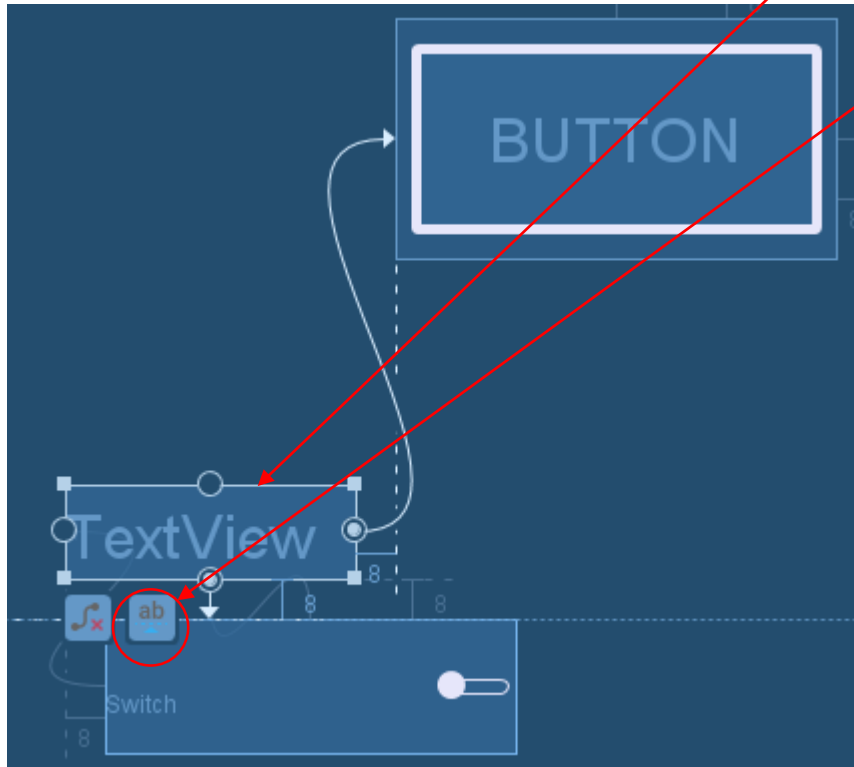
# 4. Szövegvonál igazítása más view szövegvonálához

**Feladat:** Igazítsuk a TextView szövegvonálát a gomb szövegvonálához!

1. Jelöljük ki a TextViewt rákattintással.

2. Kattintsunk a megjelent *Edit Baseline* ikonra

3. Fogjuk meg az egérrel a megjelent *text baseline*-t és húzzuk a megjelenő nyilat a gomb megjelenő *text baseline*-jához

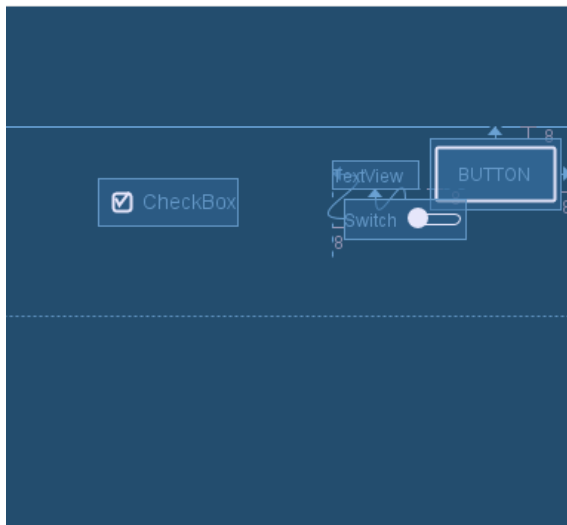


```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:text="TextView"
    app:layout_constraintBaseline_toBaselineOf="@+id/button"
    app:layout_constraintEnd_toStartOf="@+id/button" />
```

# 6. Kapcsolási szabály akadályhoz 1/5

**Feladat:** Adjunk hozzá az activity-hez egy CheckBox-ot, ami legyen mind a bal oldalán az három másik view-nak.

1. Húzzunk be egy CheckBox-ot a palettából

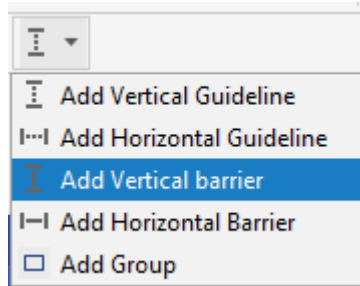


```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CheckBox"
    tools:layout_editor_absoluteX="64dp"
    tools:layout_editor_absoluteY="35dp" />
```



# 6. Kapcsolási szabály akadályhoz 2/5

2. Hozzunk létre egy függőleges akadályt (*barrier*)



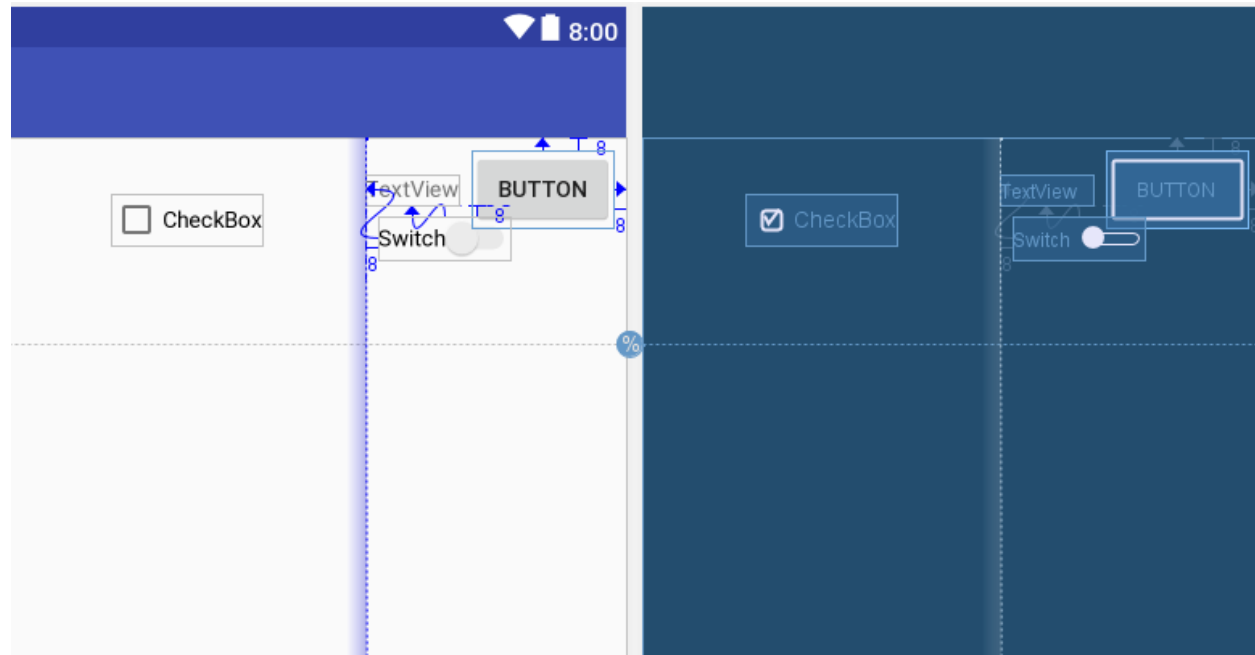
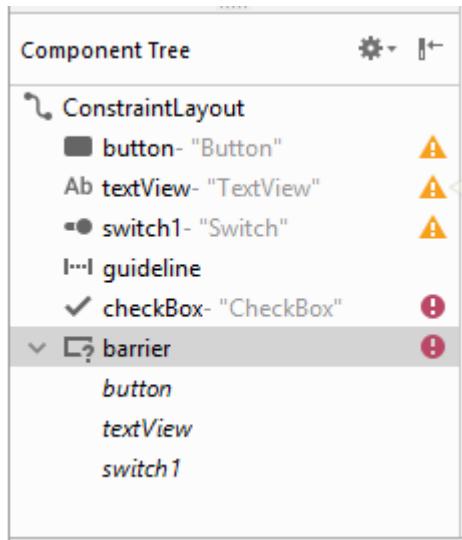
```
<android.support.constraint.Barrier  
    android:id="@+id/barrier"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:barrierDirection="left" />
```

Mi egy akadály (*barrier*)?

- Láthatalan vonal
- Nem definiálja a saját pozícióját mint egy vezetővonal, hanem mozog aszerint, hogy a benne levő view-k hova mozognak
- Akkor használatos, ha egy view-t egy view csoporthoz akarunk kapcsolni

# 6. Kapcsolási szabály akadályhoz 3/5

3. Jelöljük ki a barrier-t a *Component Tree*-ben, és húzzuk bele a textView-t, button-t és switch1-et



Ha nem látszik a Barrier a blueprint nézetben, a View-k behúzása után, akkorh Build->Clean szükséges

```
<android.support.constraint.Barrier
    android:id="@+id/barrier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="left"
    app:constraint_referenced_ids="button,textView,switch1"
    tools:layout_editor_absoluteX="384dp" />
```

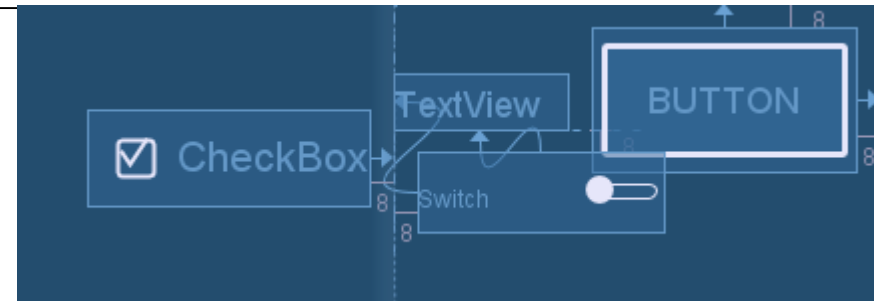
# 6. Kapcsolási szabály akadályhoz 4/5

4. Hogy ne legyen hibaüzenet az akadályhoz (*This view is not constrained horizontally: at runtime it will jump to the left unless you add a horizontal constraint*), töröljük a `tools:layout_editor_absoluteX` tulajdonságát az xml-ben

```
<android.support.constraint.Barrier
    android:id="@+id/barrier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="left"
    app:constraint_referenced_ids="button,textView,switch1"/>
```

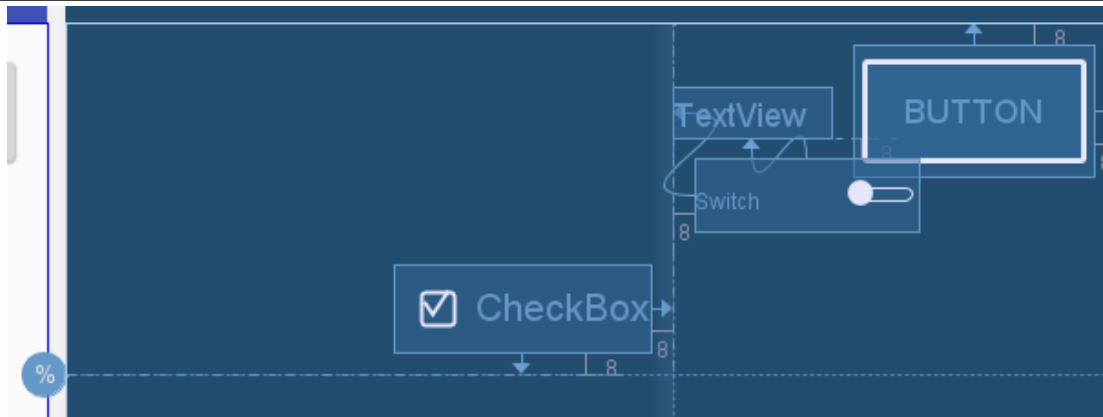
5. Kapcsoljuk a CheckBox bal oldalát a barrier-hez úgy, hogy megfogjuk az egérrel a CheckBox bal oldalán levő kör alakú kezelőt és húzzuk a megjelenő nyilat a barrier-ig

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:text="CheckBox"
    app:layout_constraintEnd_toEndOf="@+id/barrier"
    tools:layout_editor_absoluteY="35dp" />
```



# 6. Kapcsolási szabály akadályhoz 5/5

6. A CheckBox-nak még nincs vertikális szabálya. Kapcsoljuk az meglevő vezetővonalhoz az alját: megfogjuk az egérrel a CheckBox alsó oldalán levő kör alakú kezelőt és húzzuk a megjelenő nyilat a vezetővonalig

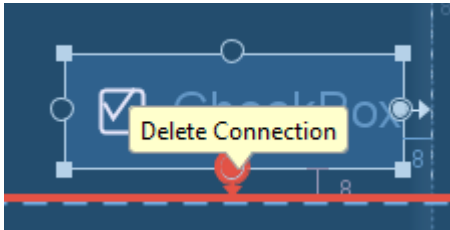
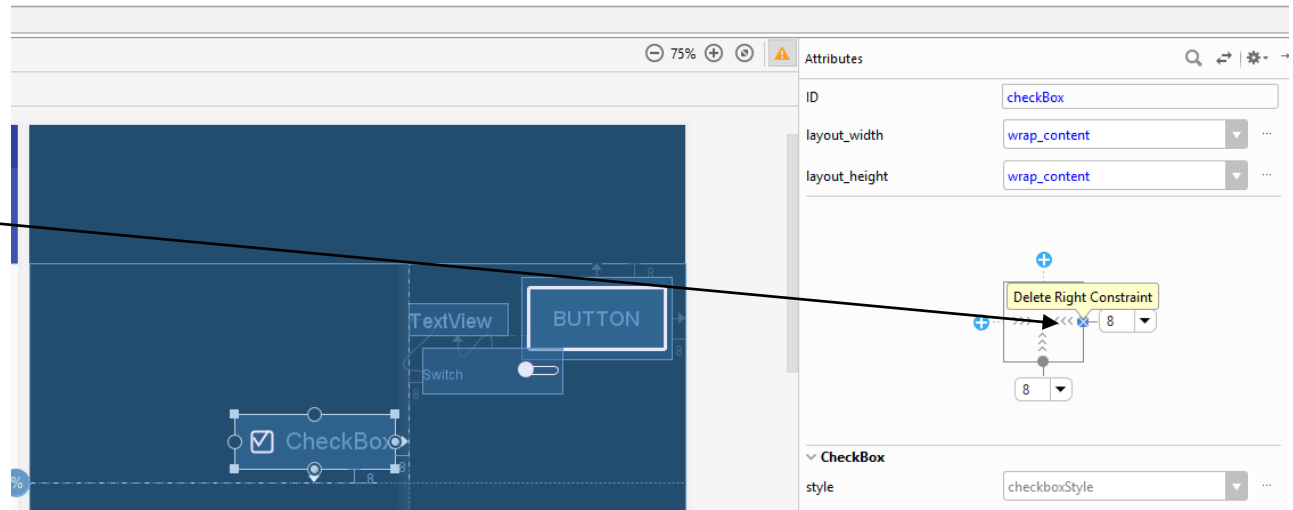


<CheckBox

```
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:text="CheckBox"
    app:layout_constraintBottom_toTopOf="@+id/guideline"
    app:layout_constraintEnd_toEndOf="@+id/barrier" />
```

# Szabály törlése

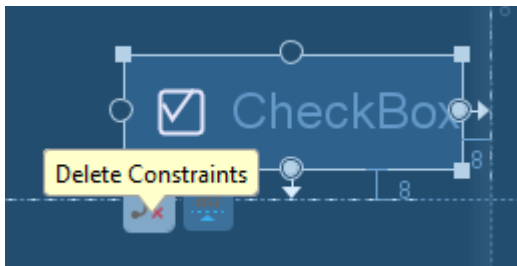
a. View kijelölése, majd az Attributes ablakban törlés ikon



b. Jelöljük ki a view-t, vigyük az egeret a kapcsolat fölé, katt az *Delete Connection* piros vonalra

c. Vagy xml-be megfelelő sor törlése

```
<CheckBox  
    . . .  
    app:layout_constraintBottom_toTopOf="@+id/guideline"  
    app:layout_constraintEnd_toEndOf="@+id/barrier" />
```



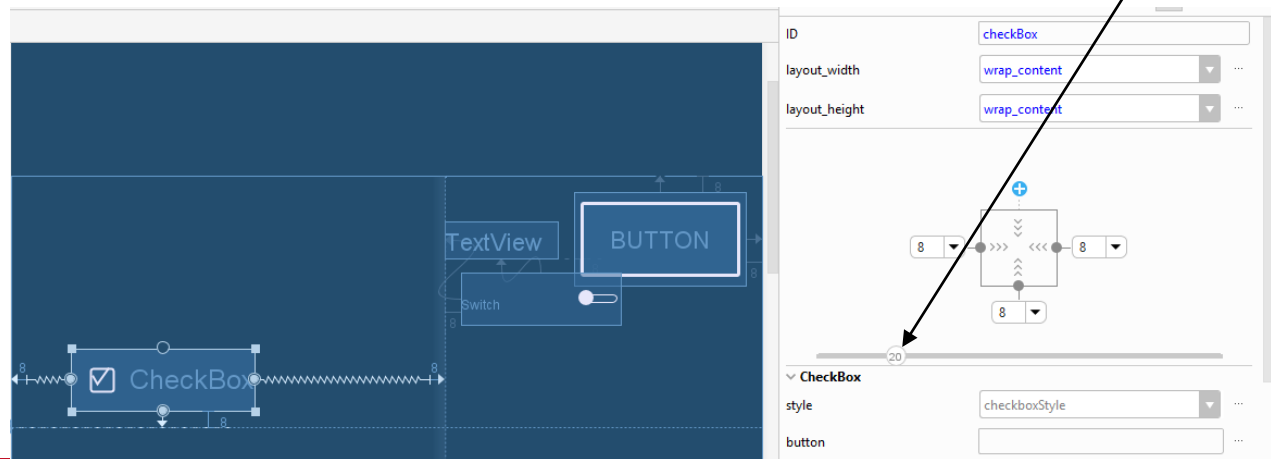
d. Összes szabály törlése: Jelöljük ki a view-t, vigyük az egeret fölé, katt az *Delete Constraints* ikonra

# Jegyezd meg (szabályok szabályok létrehozására)

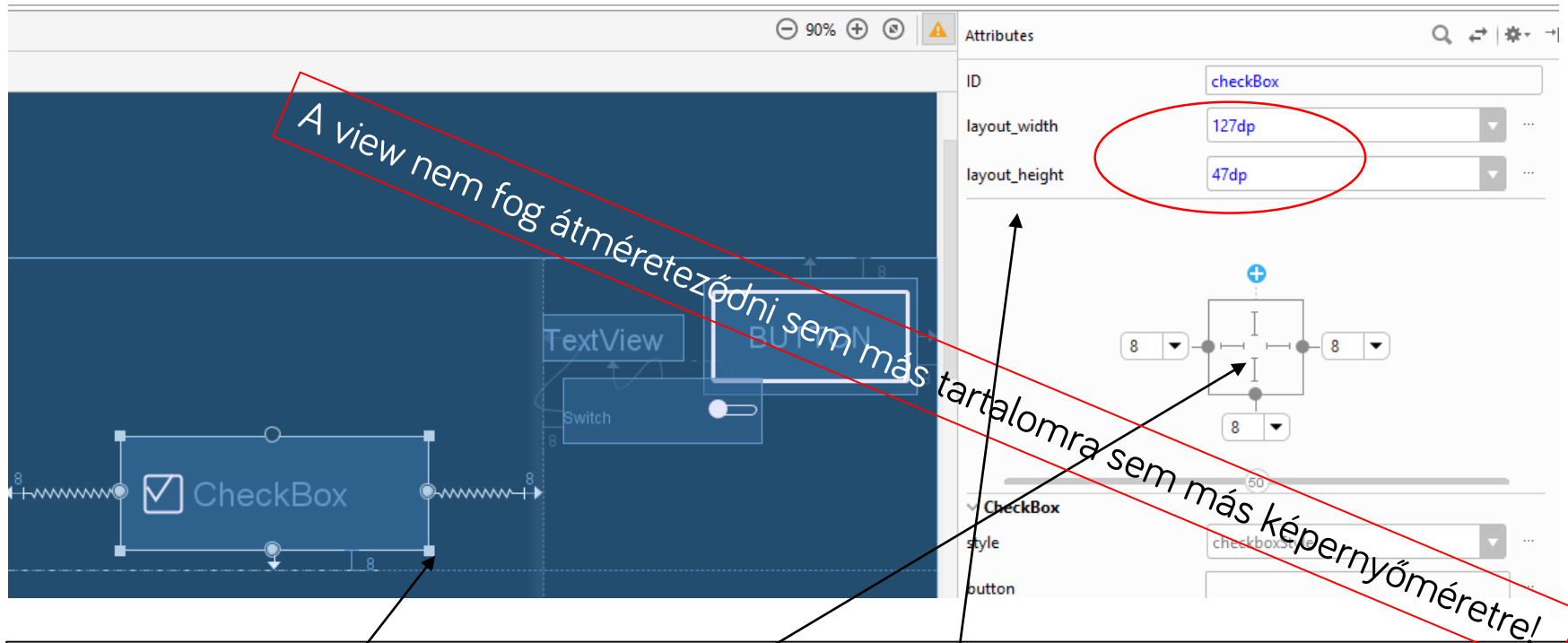
1. Vertikális szabályt csak vertikális síkkal lehet létrehozni
2. Horizontális szabályt csak horizontális síkkal lehet létrehozni
3. Szövegvonalat csak szövegvonálhoz lehet igazítani
4. Egy kör alakú kezelővel csak egy szabály hozható létre, viszont egy horgonypontra több szabály is készíthető

# Arányok beállítása több szabály esetén (Adjusting constraint bias)


- Ha egy view-nak két szabálya is van ugyanazon síkra az ellentétes oldalain, akkor középre fog helyeződni
- Ezt a default 50%-s arányt módosítani lehet
  - > Csúszka segítségével az attribútumok ablakában
  - > Kézzel elmozgatva a view-t



# View méreteinek beállítása – 1. nem átméreteződő view



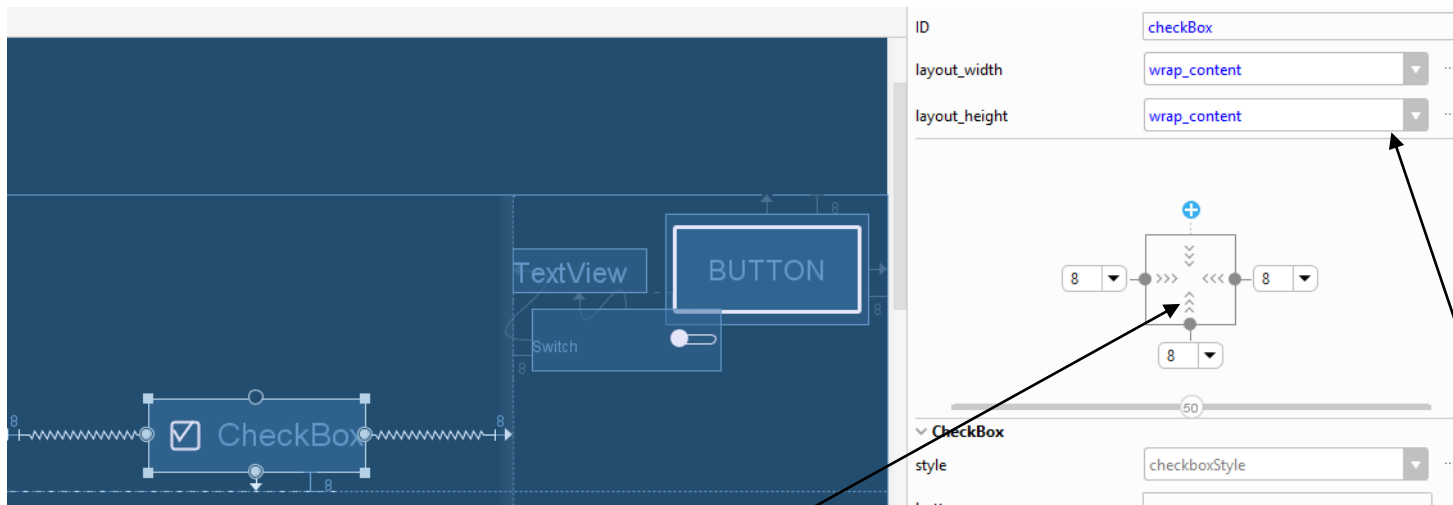
Fix méret megadható

1. View sarkain levő négyzet alakú kezelőkkel
2. Az attribútumok ablakában a  szimbólumra váltással (katt rá) és az attribútumok ablakában a layout\_width és layout\_height értékeket begépelve
3. Az layout xml fájljában a layout\_width és layout\_height értékeket megadva dp-ben



# View méreteinek beállítása – 2. wrap\_content

`wrap_content` szélesség és/vagy magasság esetén a view akkora lesz, hogy a tartalma pontosan elérjen benne



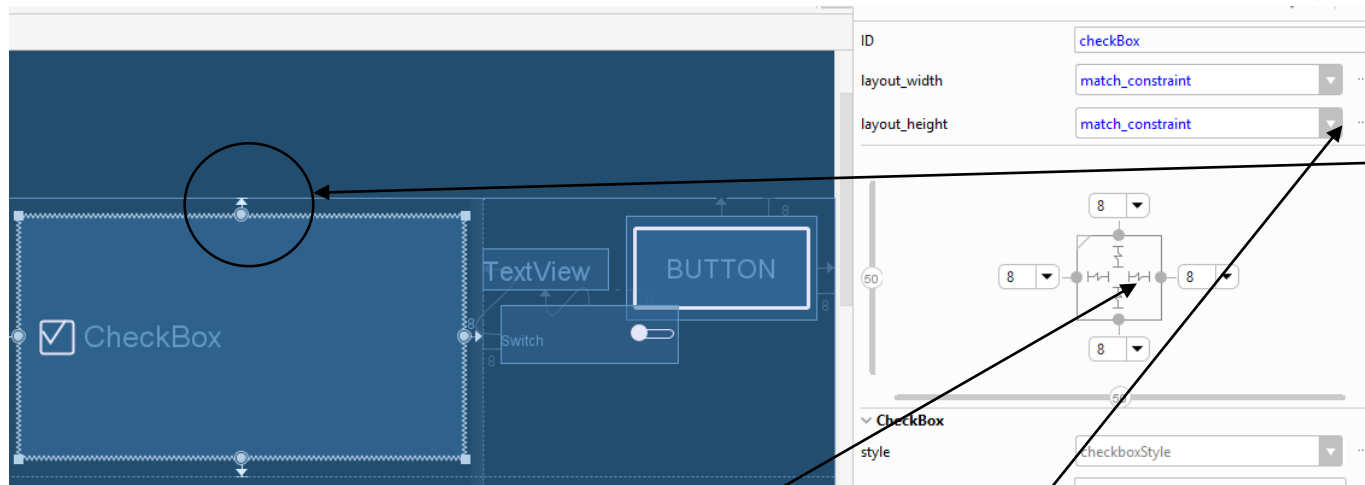
`wrap_content` méret megadható

1. Az attribútumok ablakában a `>>>` szimbólumra váltással
2. Az attribútumok ablakában a `layout_width` és `layout_height` értékeinek a `wrap_content`-et választva a legördülő menüből
3. Az `layout.xml` fájljában a `layout_width` és `layout_height` értékeknek `wrap_content`-et megadva

# View méreteinek beállítása – 3.


## match\_constraints 1/3

`match_constraints` szélesség és/vagy magasság esetén a view a lehető legnagyobb méretű lesz, amit a szabályai még megengednek (a margin-okat figyelembe véve)



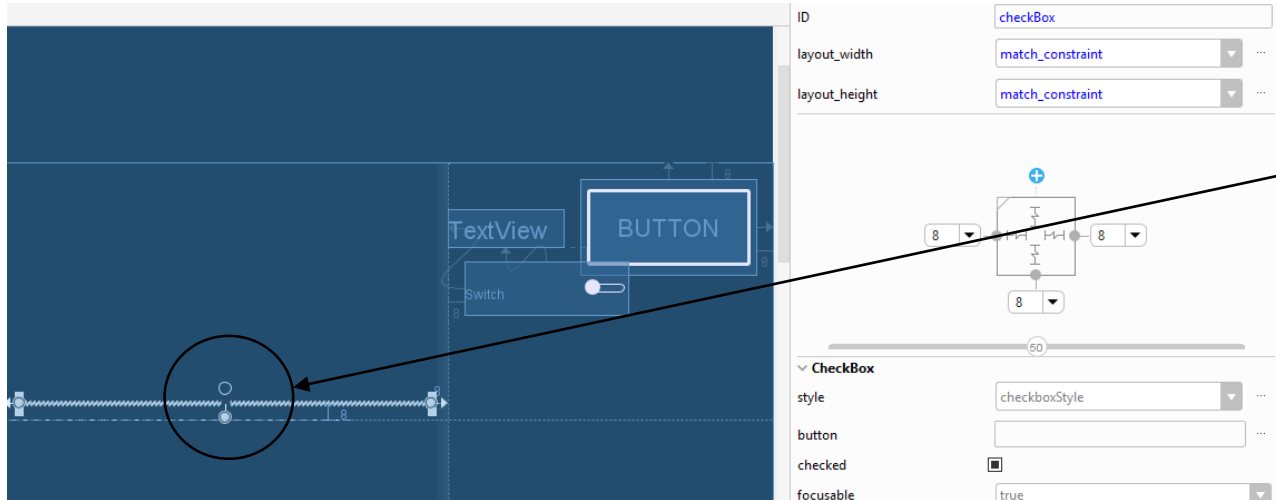
Figyeljük meg: a CheckBox felső széle hozzá van kapcsolva a szülőhöz az előző példához képest

`match_constraints` méret megadható

1. Az attribútumok ablakában a  szimbólumra váltással
2. Az attribútumok ablakában a `layout_width` és `layout_height` értékeinek a `match_constraints`-et választva a legördülő menüből
3. Az `layout.xml` fájljában a `layout_width` és `layout_height` értékeknek `0dp`-t megadva

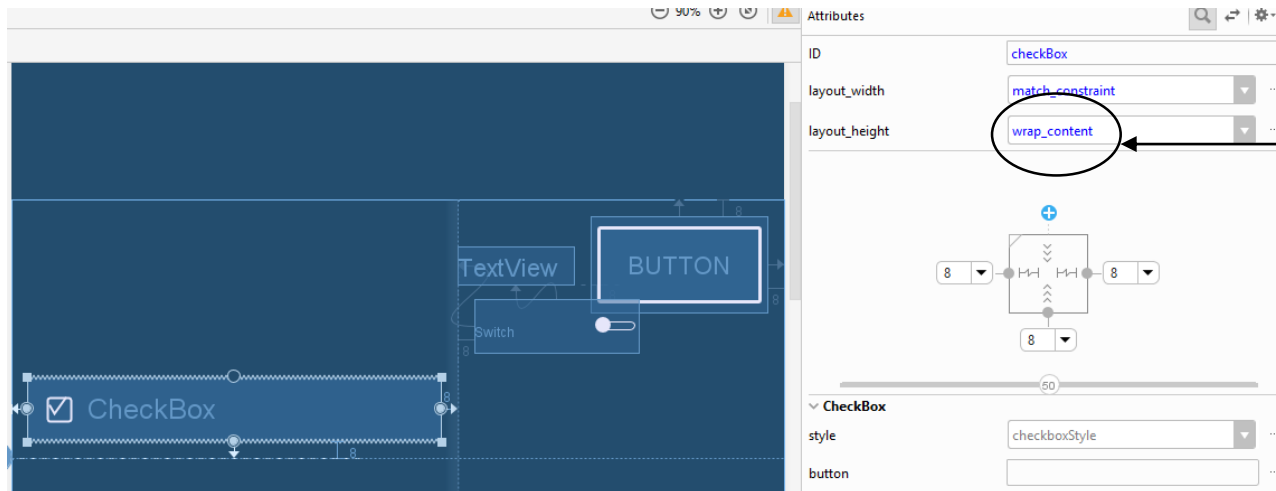
# View méreteinek beállítása – 3.

## match\_constraints 2/3



Töröljük a  
CheckBox felső  
szélén a  
kapcsolást

Figyeljük meg: a  
Checkbox nulla  
magasságú lett



Javítsuk meg a  
magasságot  
wrap\_content-re  
állítással

# View méreteinek beállítása – 3.

## match\_constraints 3/3

- Ha egy view szélessége `match_constraints`, akkor további attribútumok is használhatóak:

### 1. `layout_constraintWidth_default`

1. `spread` értékkel – default érték – kinyújtja a view-t szélességében, amennyire csak a szabályok engedik
2. `wrap` értékkel – csak annyira nyújtja ki a view-t, hogy a tartalma elférjen benne, de ha a szabályok megkívánják, kisebb is lehet a view, mint a tartalma (olyan mint a `wrap_content`, de az nem engedi kisebbnek a view-t, mint maga a tartalma)

### 2. `layout_constraintWidth_min` – meghatározza a view min szélességét dp-ben

### 3. `layout_constraintWidth_max` – meghatározza a view max szélességét dp-ben

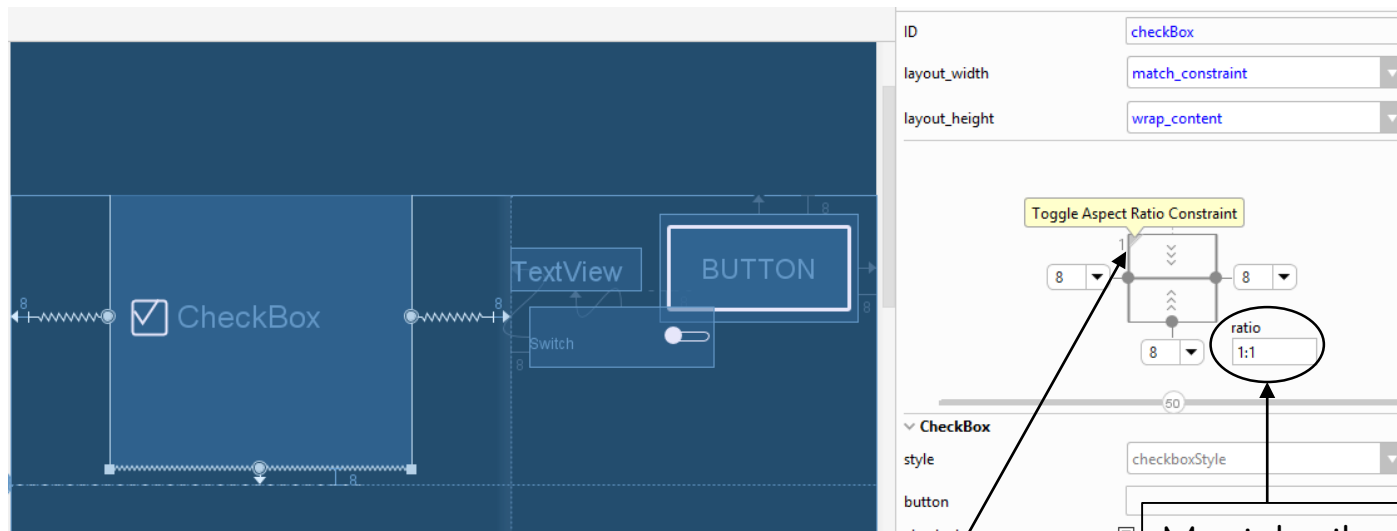
# View méreteinek beállítása – 4. match\_parent???

- A `match_parent` érték nem használható méretek `ConstraintLayout`-ban
- Helyette használjunk `match_constraints` (`0dp`) értéket

# View méreteinek beállítása – 5.

## szélesség:magasság arány 1/4

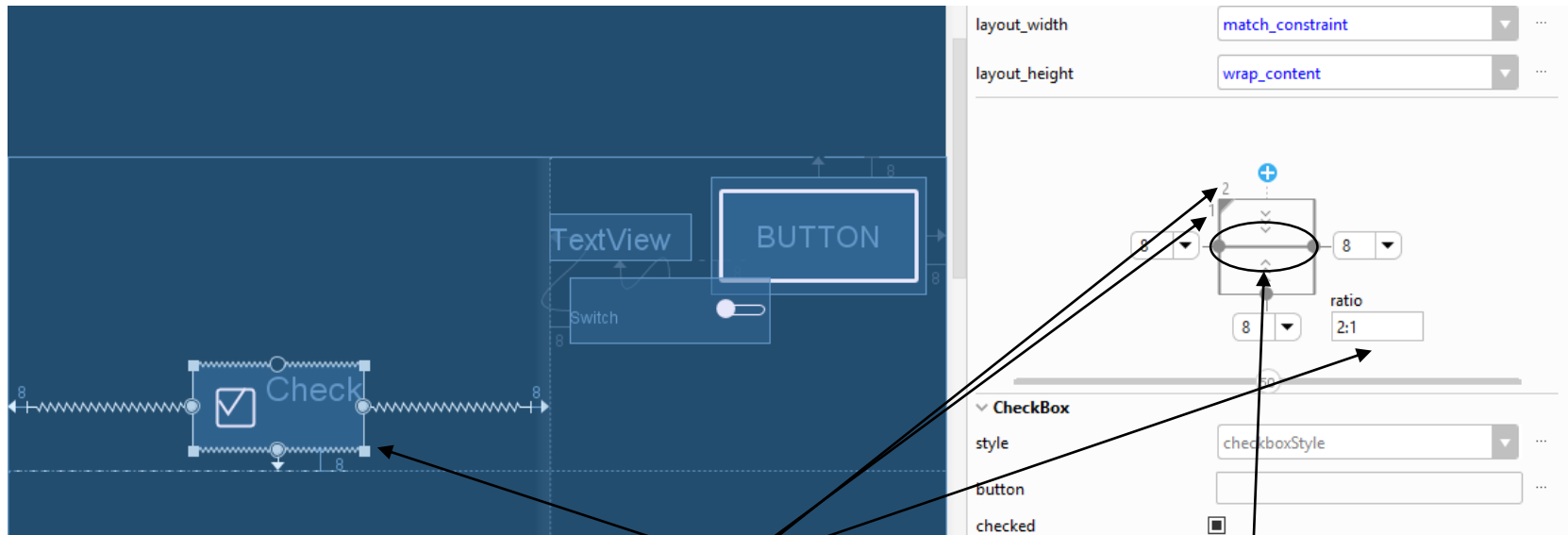
- Ha legalább a szélesség vagy a magasság értéke `match_constraint`, akkor a view méretét meg lehet adni **szélesség:magasság** arányban is



Katt a bal felső sarokban a *Toggle Aspect Ratio Constraint*-ra

Megjelenik a szélesség:magasság arány és a view átméreteződik alpból 1:1 arányra

# View méreteinek beállítása – 5. szélesség:magasság arány 2/4



Állítsuk a 2:1-re a szélesség:magasság arányt

```
<CheckBox  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    . . .  
    android:text="CheckBox"  
    app:layout_constraintDimensionRatio="w,2:1"  
>
```

Jelen esetben a szélesség függ a magasság méretétől

# View méreteinek beállítása – 5. szélesség:magasság arány 3/4

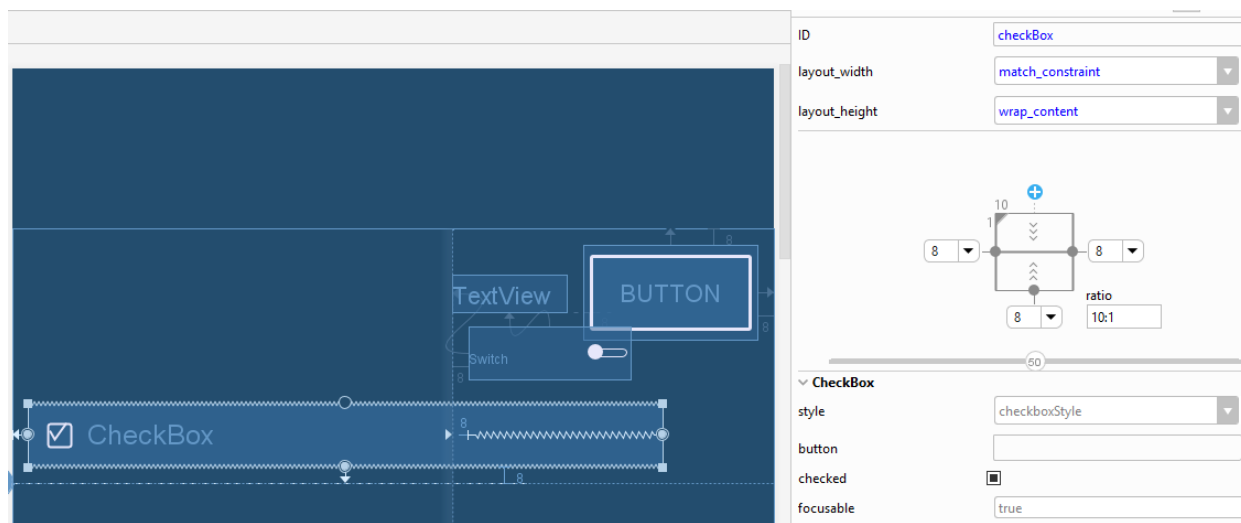
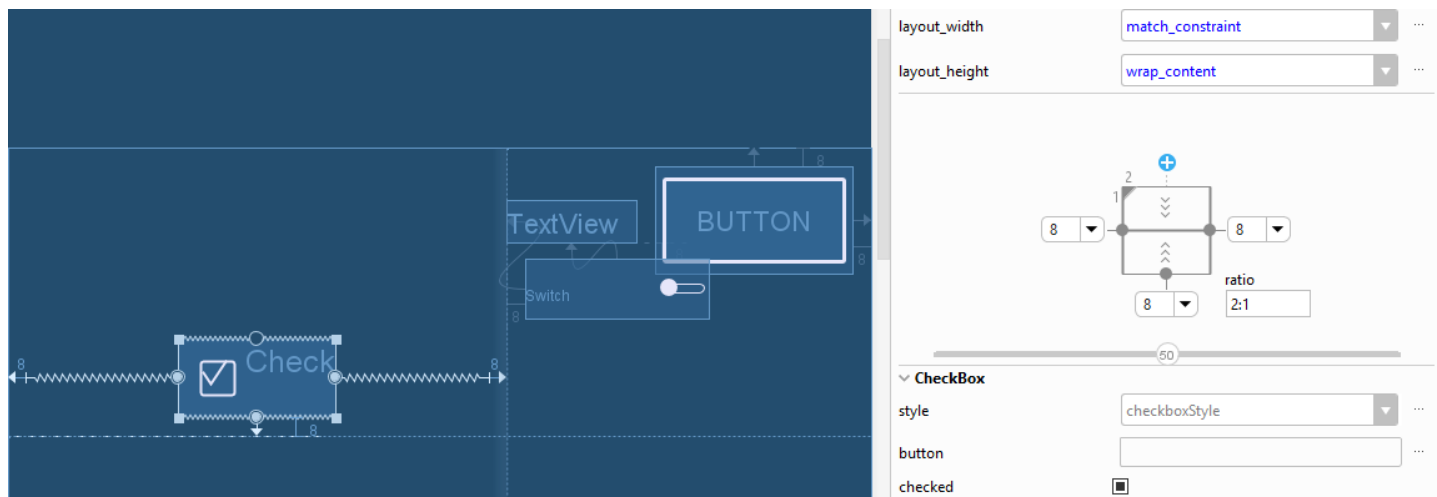
## Feladat:

Állítsuk át a  
szélesség:mag  
asság arányt 2:  
1-ről 10:1-re



-> a CheckBox  
túlnyúlik a  
vezetővonalon...

A probléma az, hogy  
a magasságtól függ a  
szélesség...

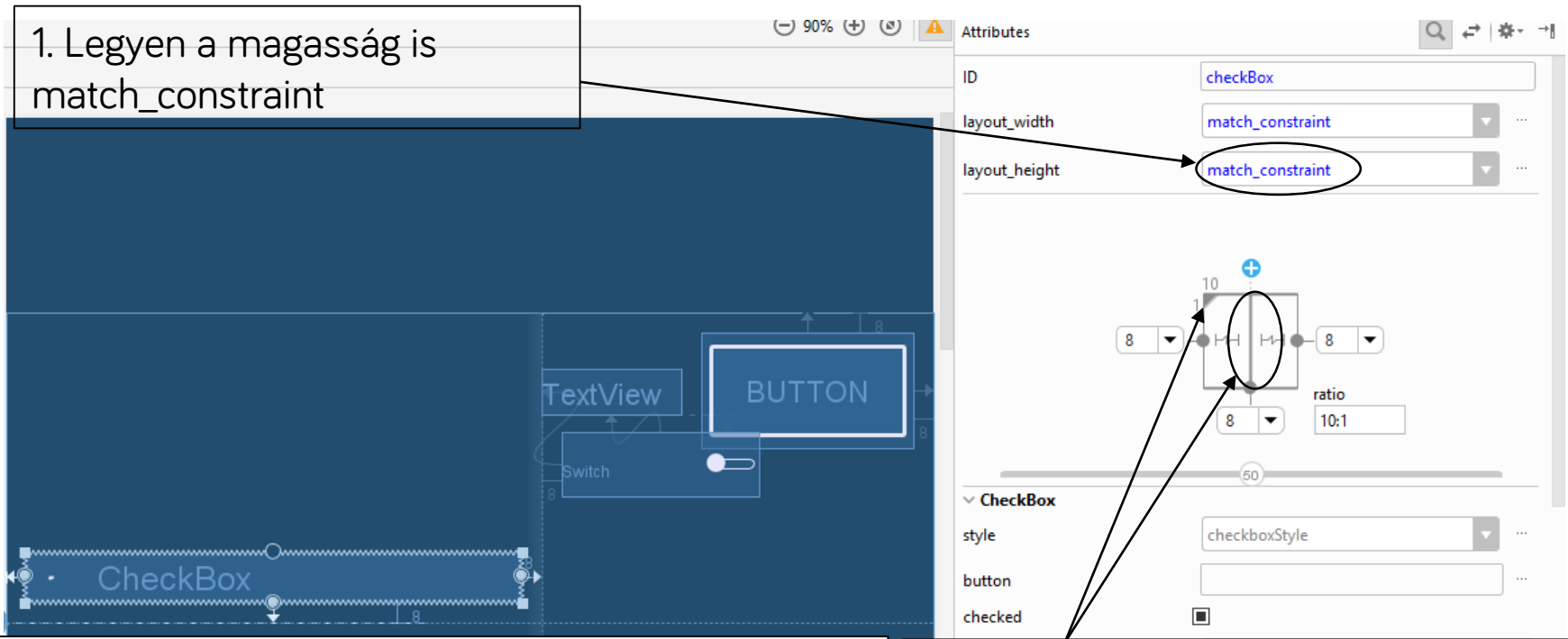




# View méreteinek beállítása – 5. szélesség:magasság arány 4/4

**Megoldás:** Fügjön inkább a magasság a szélességtől, hogy férjen bele a CheckBox a helyére és az arány is 10:1 legyen

1. Legyen a magasság is  
match\_constraint



```
<CheckBox  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    . . .  
    app:layout_constraintDimensionRatio="h,10:1"  
>
```

2. Fügjön a magasság a szélességtől:  
katt a bal felső sarokban a *Toggle Aspect Ratio Constraint-ra*, amíg megjelenik a függőleges vastag vonal

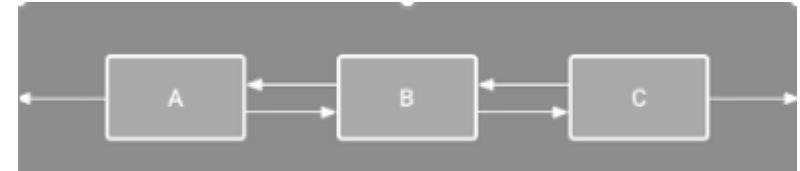
# View láncok

- Egy lánc (chain) view-k csoportja, melyben a view-k mindekét irányból kapcsolva vannak egymáshoz
- Vertikális és horizontális elrendezés is támogatott

# View lánc stílusok

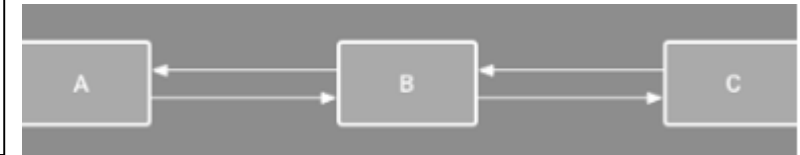
## Spread

A view-k egyenlő távolságban egymástól



## Spread inside

Az első és utolsó view a kapcsolt elemek mellett vannak közvetlenül, a több view egymástól egyenlő távolságra



## Weighted

Spread és Spread inside esetén megadható az egyes view-k súlya

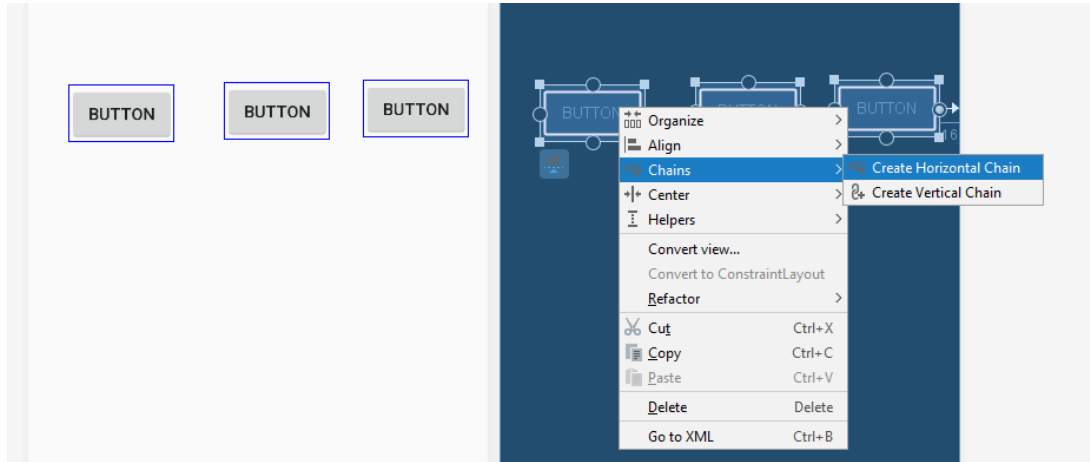


## Packed

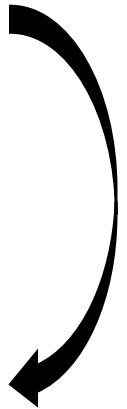
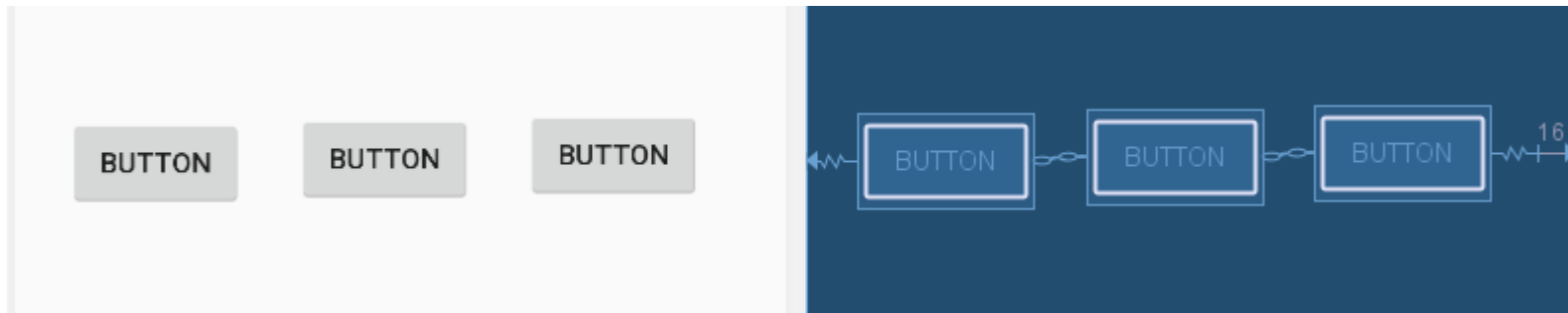
A view-k közvetlenül egymás mellett helyezkednek el (margin-ok figyelembe vételével). A lánc jobb és bal széleinek aránya állítható az első view arányának (bias) állításával



# View lánc létrehozása



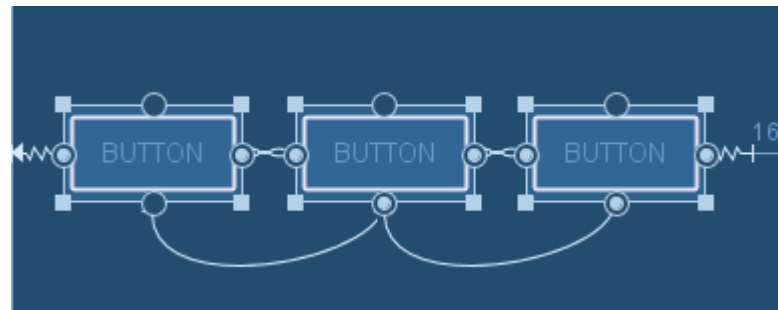
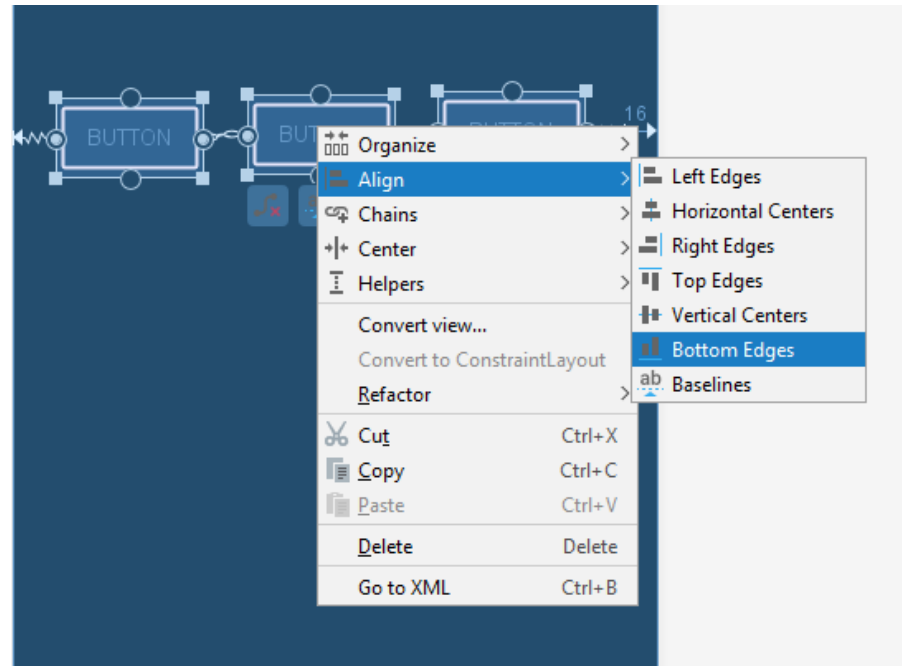
Jelöljük ki a láncba tenni kívánt view-kat -> jobb klikk -> Chains -> Create Horizontal Chain / Create Vertical Chain



# View-k súlyozása view láncban 1/4

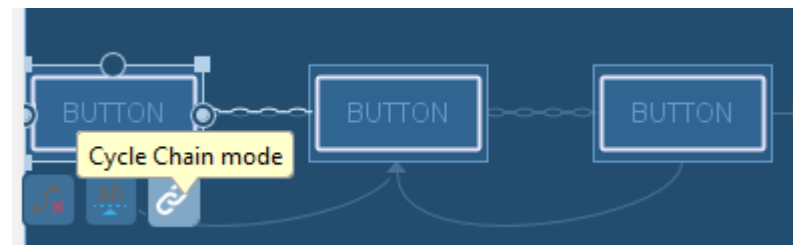
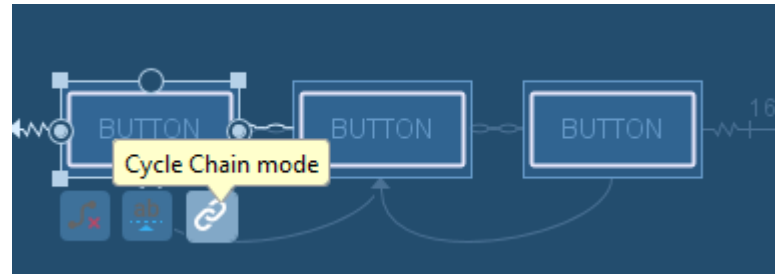
**Feladat:** Igazítsuk a gombok alját egy vonalba! Töltsék ki a gombok a képernyő teljes szélességét, az első gomb 20%-ban, a második 30%-ban, a harmadik pedig 50%-ban töltsse ki a képernyő szélességét!

1. Align bottom edges



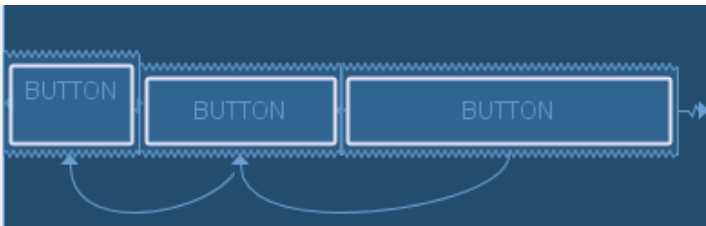
# View-k súlyozása view láncban 2/4

2. Állítsunk be *spread inside* stílust bármelyik view Cycle Chain mode ikonjának segítségével



# View-k súlyozása view láncban 3/4

3. Állítsunk be minden gombnak **0dp** (azaz `match_constraints`) szélességet és súlyt a **`app:layout_constraintHorizontal_weight`** segítségével (xml-ben vagy az attribútumoknál szintén kézzel írva)



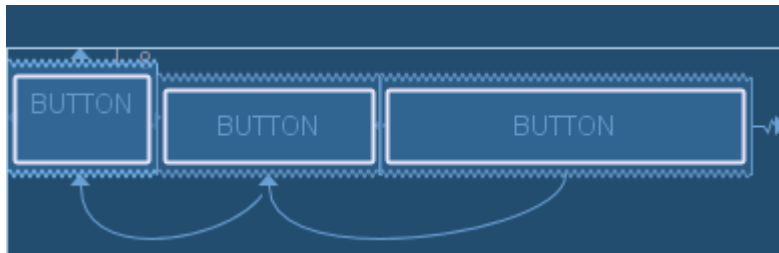
```
<Button
    android:id="@+id/button8"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintEnd_toStartOf="@+id/button9"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintHorizontal_chainStyle="spread_inside"
    app:layout_constraintHorizontal_weight="0.2"
    app:layout_constraintStart_toStartOf="parent"
    tools:layout_editor_absoluteY="90dp" />

<Button
    android:id="@+id/button9"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="@+id/button8"
    app:layout_constraintEnd_toStartOf="@+id/button10"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintHorizontal_weight="0.3"
    app:layout_constraintStart_toEndOf="@+id/button8" />

<Button
    android:id="@+id/button10"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="@+id/button9"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintHorizontal_weight="0.5"
    app:layout_constraintStart_toEndOf="@+id/button9" />
```

# View-k súlyozása view láncban 3/4 - példa befejezése

Az első gombnak nincs vertikális szabálya még, megoldásként kapcsoljuk a szülő tetejéhez!





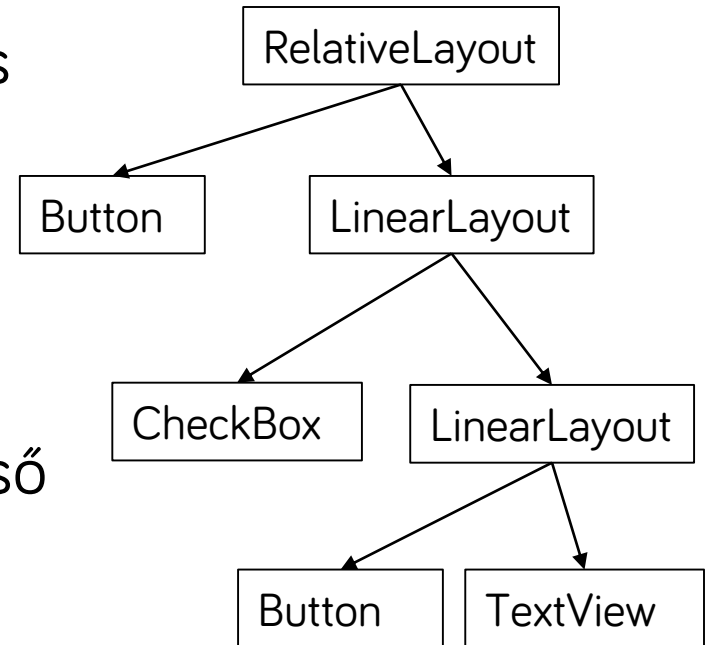
# Jegyezd meg a view láncokról

- Ugyanaz a view része lehet egy függőleges és egy vízszintes láncnak is

# ConstraintLayout teljesítmény 1/2

- View rajzolás fázisai Androidban

1. Measure (Mérés): View fa bejárása minden viewgroup és view méretének meghatározására
2. Layout (Elrendezés): View fa bejárása, hogy minden ViewGroup gyerekeinek pozícióját meghatározza az első fázisban kiszámolt méretek alapján
3. Draw (Rajzolás): View fa bejárása, hogy minden objektumról egy Canvas objektum készülhessen



-> Minél több view-t ágyazunk egymásba, a bejárás annál költségesebb!!!

# ConstraintLayout teljesítmény 2/2

- Mérések alapján egy ConstraintLayout akár 40%-al gyorsabb tud lenni egy sok viewgroup-ot tartalmazó RelativeLayout-tal szemben (a measure és layout fázisok kerültek mérésre)
- Bővebben a mérésről: <https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html>

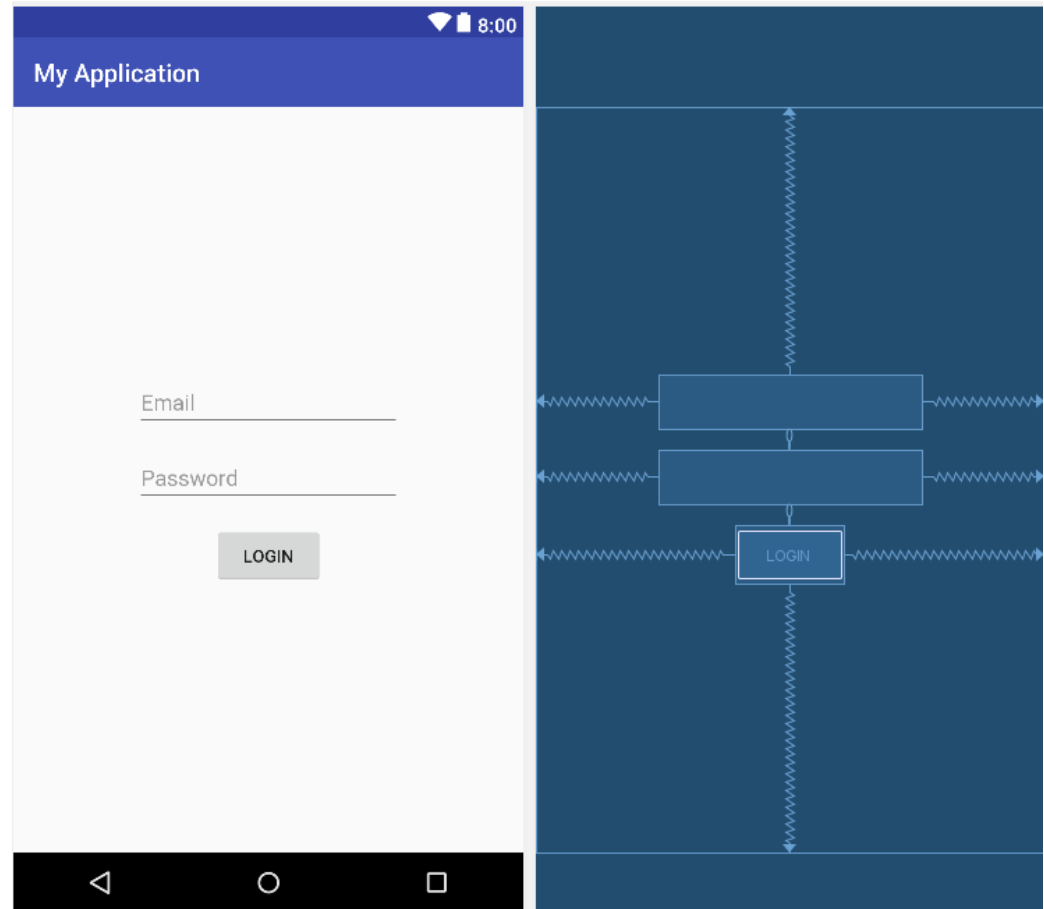
# UI performancia mérési módok

- Systrace command line (python) paranccsal
  - > [UI performancia mérése a systrace paranccsal](#)
- Programkódból FrameMetrics API-val

# Gyakoroljunk!

## – 1. Feladat

Készítsük el a felületét egy login activity-nek  
ConstraintLayout-ot és  
Layout Editor-t használva!

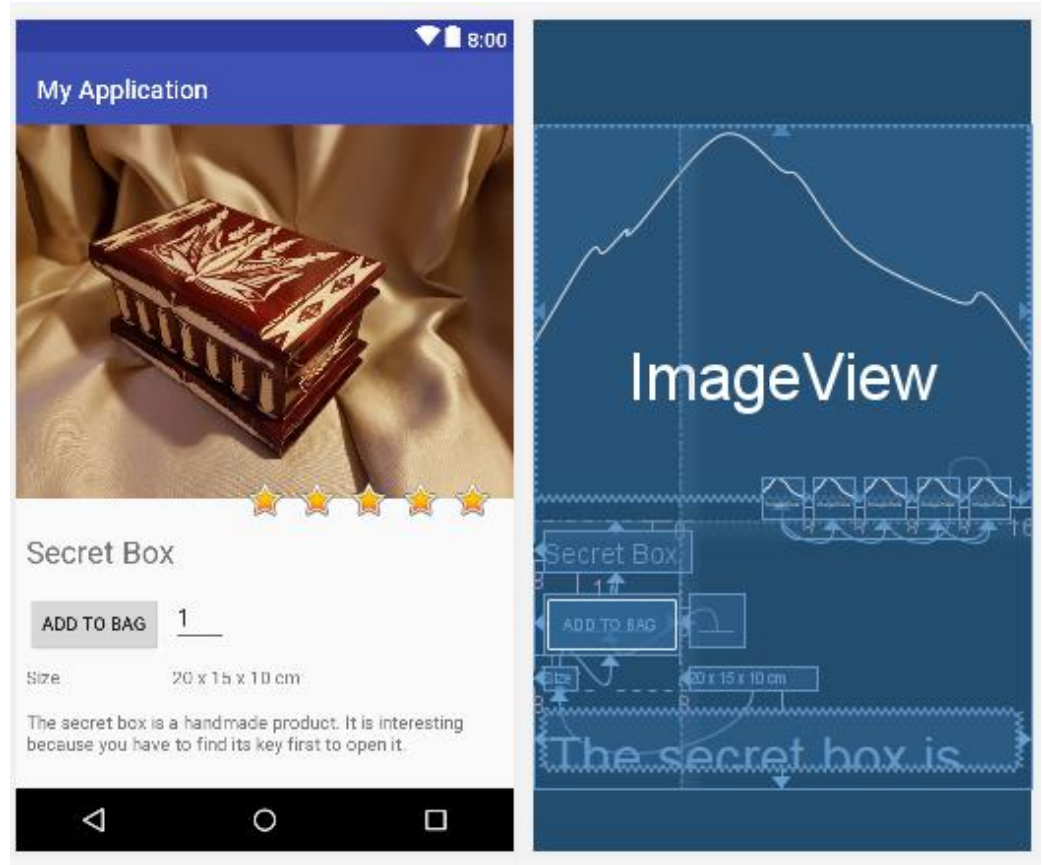


# Mi az **android:ems**?

- TextView-val, EditText-el, ChekBox-xal és más TextView-ből származó view-val használható attribútum
  - > Csak ha a layout\_width="wrap\_content" !!!
- Olyan szélesre állítja a view-t, hogy a beállított értékkel egyenlő darabszámú "M" betű férjen bele
- Tipográfiában használatos mértékegység
- 1 em egy 16 pontos typeface-ben 16 pont

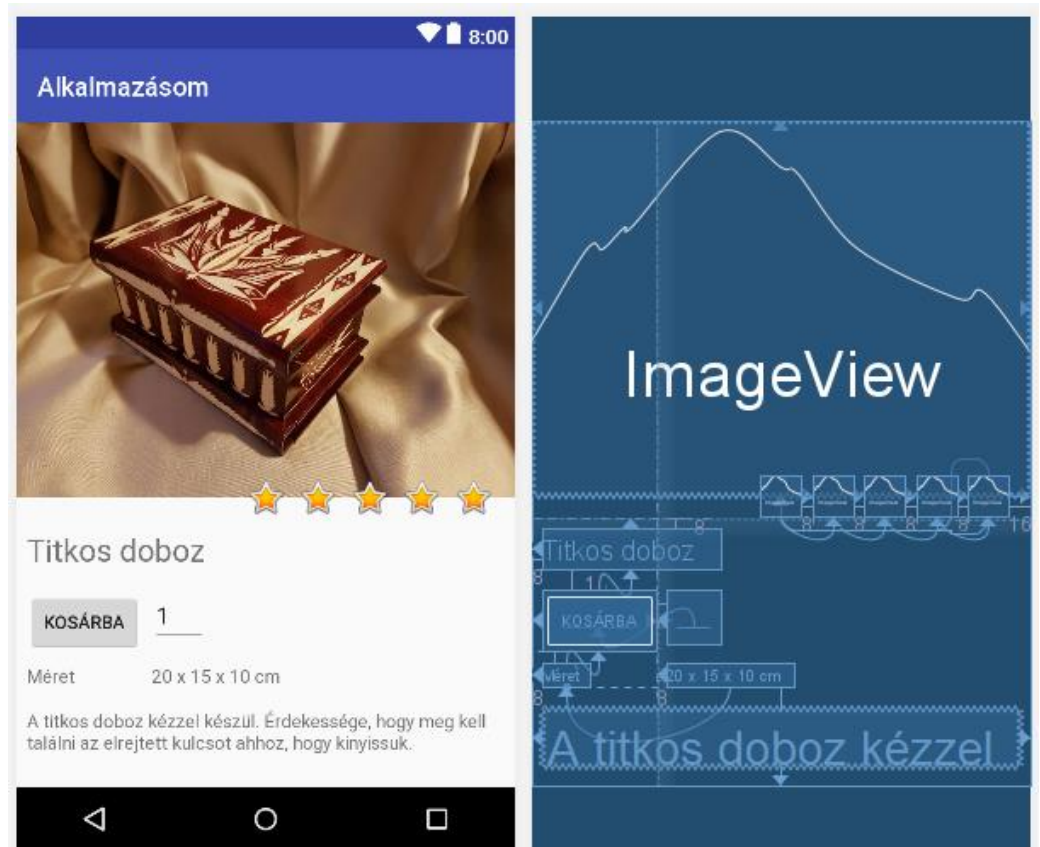
# Gyakoroljunk! – 2. Feladat

Készítsük el a felületét egy webáruházban levő termék oldalának ConstraintLayout-ot és Layout Editor-t használva!



# Gyakoroljunk! – 3. Feladat

Az előző feladatban elkészített felülethez adjunk magyar nyelvű fordítást is! Ellenőrizzük, hogy a magyar szövegekkel is megmarad az elrendezés!

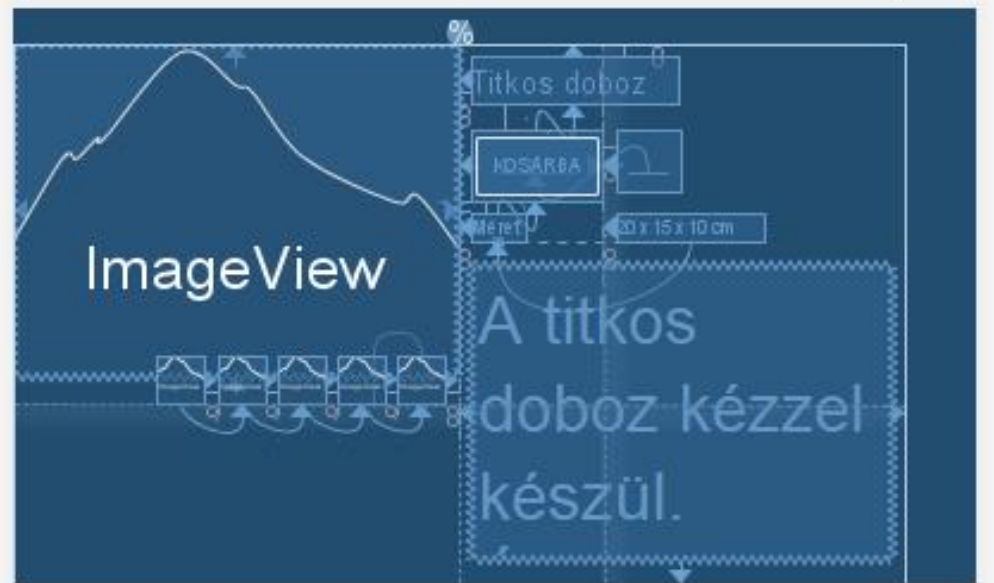




# Gyakoroljunk!

## – 4. Feladat

Az előző feladatban elkészített felülethez adjunk fekvő nézetet is!



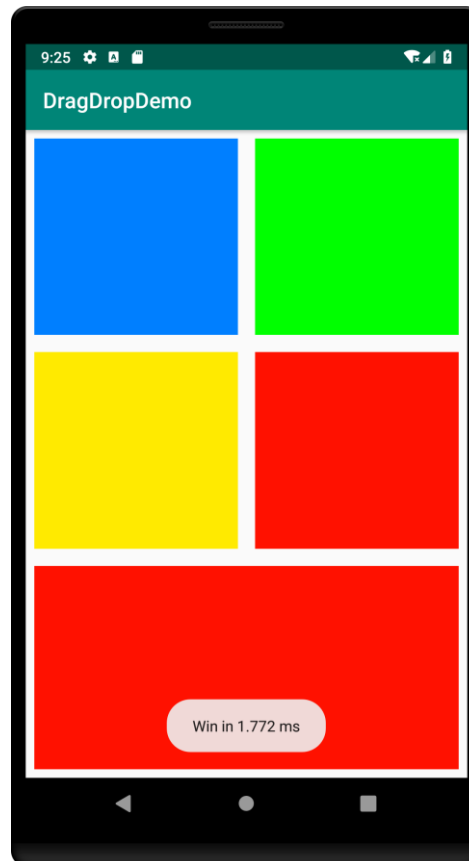
# DRAG & DROP ÁLTALÁNOSAN

# Drag & Drop támogatás

- Gyakorlatilag bármelyik komponens „drag”-elhető:
  - > `View.startDrag(...)`
- Felületek/komponensek feliratkozhatnak „drag” esemény érzékelésre:
  - > `setOnDragListener(...)`
- Többféle „drag” esemény:
  - > `ACTION_DRAG_STARTED`
  - > `ACTION_DRAG_ENTERED`
  - > `ACTION_DRAG_EXITED`
  - > `ACTION_DROP`

# Gyakorljunk!

- Készítsünk egy színválasztó Drag&Drop játékot reakció idő mérésre!



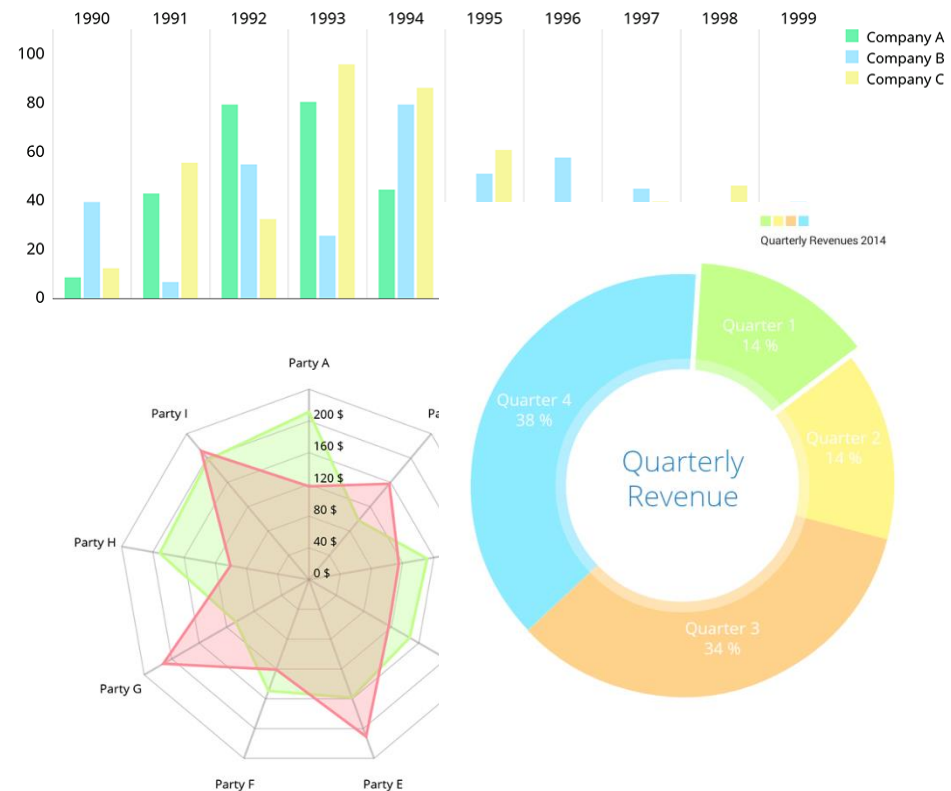
# GRAFIKONOK RAJZOLÁSA

# MPAndroidChart

- Gazdag grafikon rajzoló osztálykönyvtár
- Számptalan grafikon típus

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

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



# MPAndroidChart használat

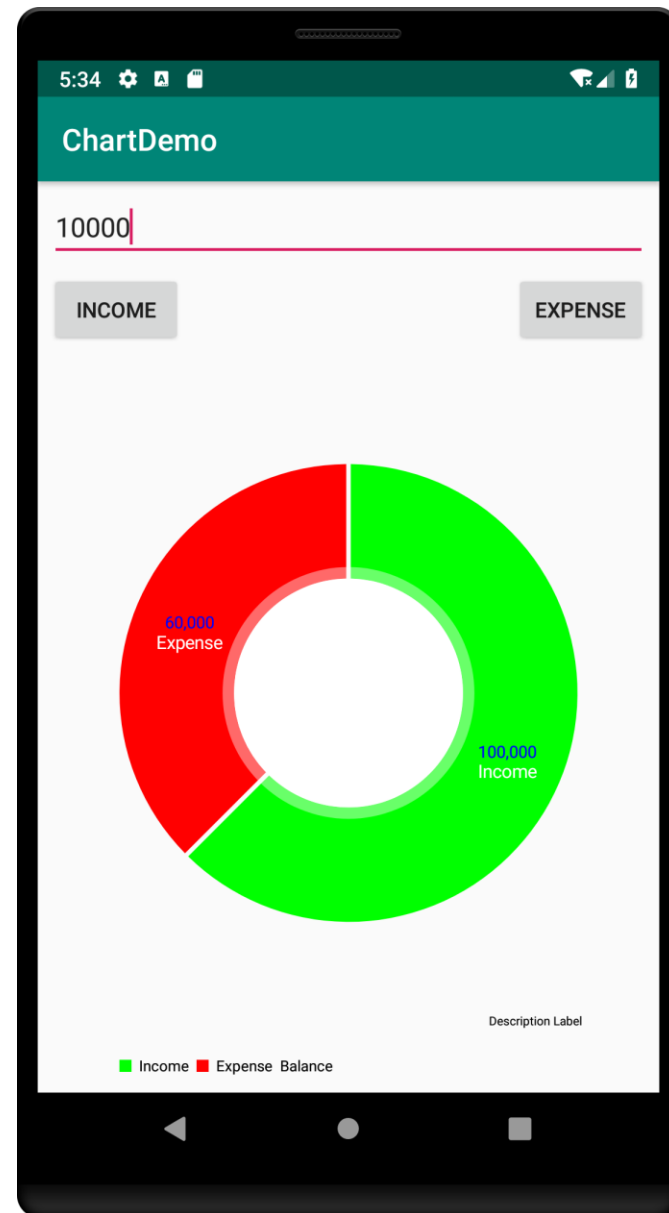
## 1. Grafikon elhelyezése layout file-ba

```
<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

# Gyakoroljunk!

Készítsünk egy kiadás/bevétel  
kezelő alkalmazást és az  
állapotot jelenítsük meg egy  
*PieChart*-on

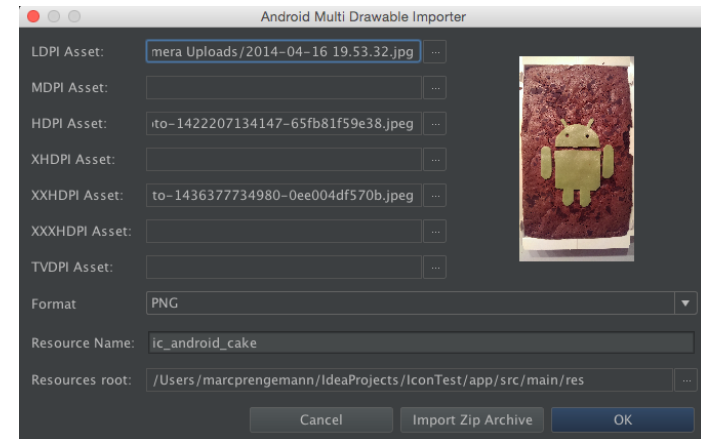




# NÉPSZERŰ UI LIBRARY-K ÉS PLUGINEK

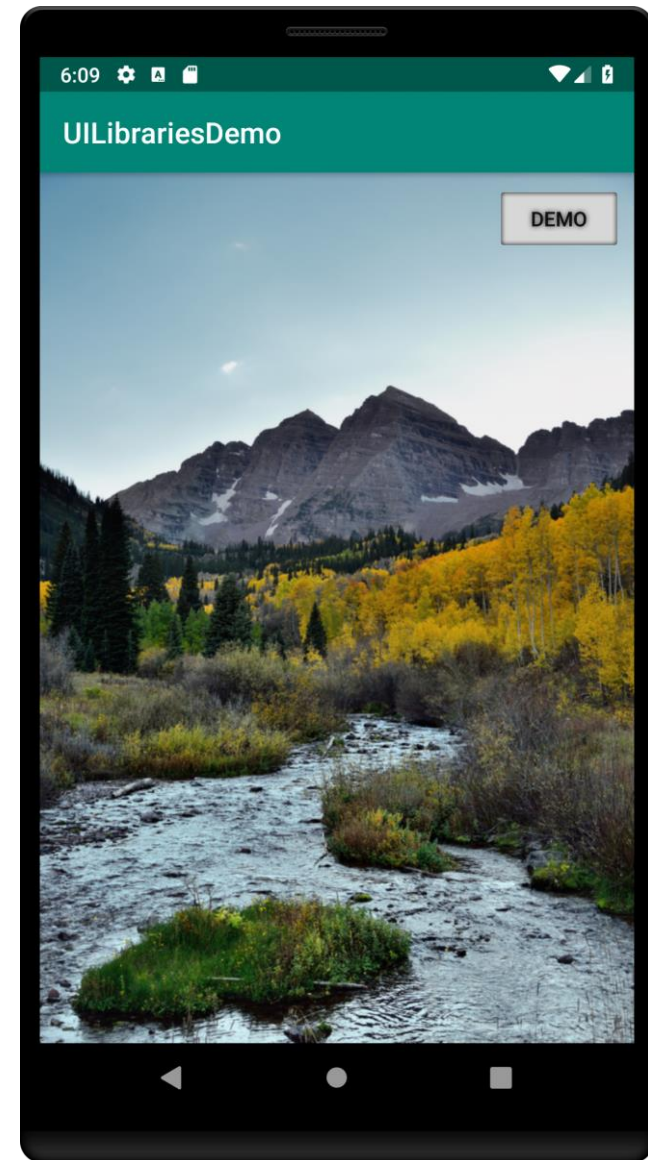
# Képek kezelése

- Android Drawable Importer Plugin
  - > <https://plugins.jetbrains.com/plugin/7658-android-drawable-importer>
- Ikonok gyors betöltése
  - > Gazdag ikon készlet
  - > Különböző méretek és színek
- Képek betöltése több méretben
  - > Automatikus átméretezés
- Egyszerű és gyors használat



# Blur effekt

- <https://github.com/mirrajabi/view-effects>

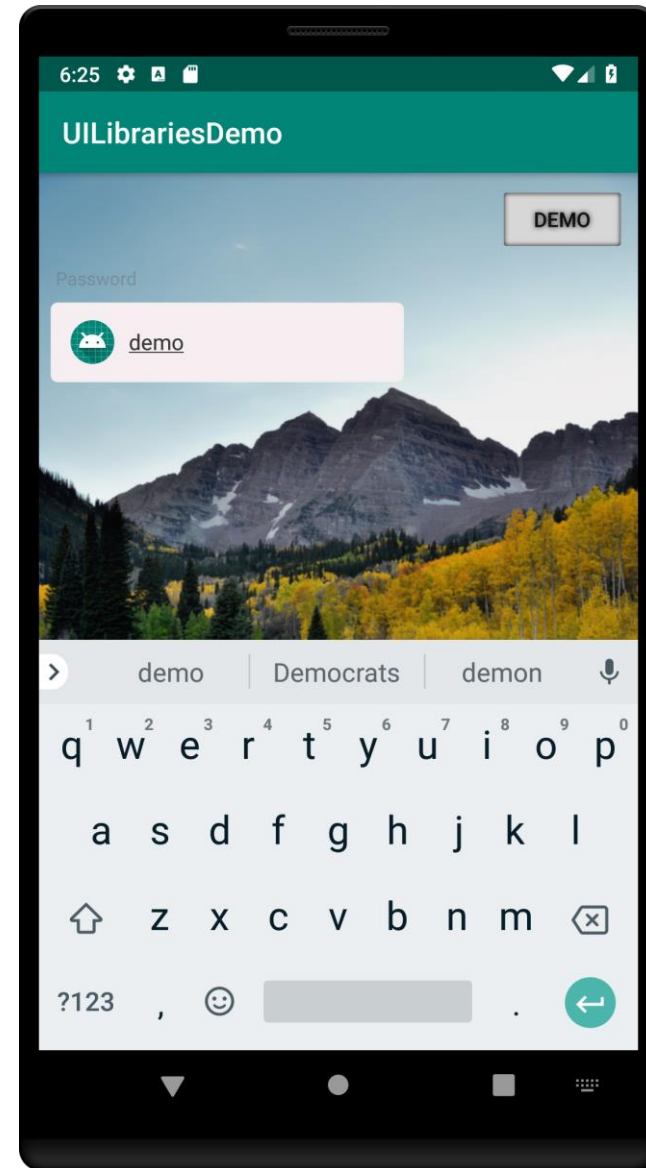


```
ViewFilter.getInstance(this)  
    //Use blur effect or implement custom IRenderer  
    .setRenderer(BlurRenderer(10))  
    .applyFilterOnView(btnDemo, rootView)
```

# MaterialTextField

- Látványos beviteli mező ikonnal
- <https://github.com/florent37/MaterialTextField>

```
<com.github.florent37.materialtextfield.MaterialTextField
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:mtf_animationDuration="500"
    app:mtf_cardCollapsedHeight="4dp"
    app:mtf_image="@mipmap/ic_launcher"
    app:mtf_labelColor="#666"
    app:mtf_openKeyboardOnFocus="true">
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Password"
        android:ems="10"
        android:textColor="#333"
        android:textSize="15sp" />
</com.github.florent37.materialtextfield.MaterialTextField>
```



# PhysicsLayout

- JBox2D motor által hajtott
- Nem játékra lett tervezve (arra libGdx pl)
- <https://github.com/Jawnnypoo/PhysicsLayout>



# PhysicsLayout példa

```
<?xml version="1.0" encoding="utf-8"?>
<com.jawnnypoo.physicslayout.PhysicsLinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/physics_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:physics="true"
    app:gravityX="0.0"
    app:gravityY="9.8"
    app:bounds="true"
    app:boundsSize="50dp">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher"/>

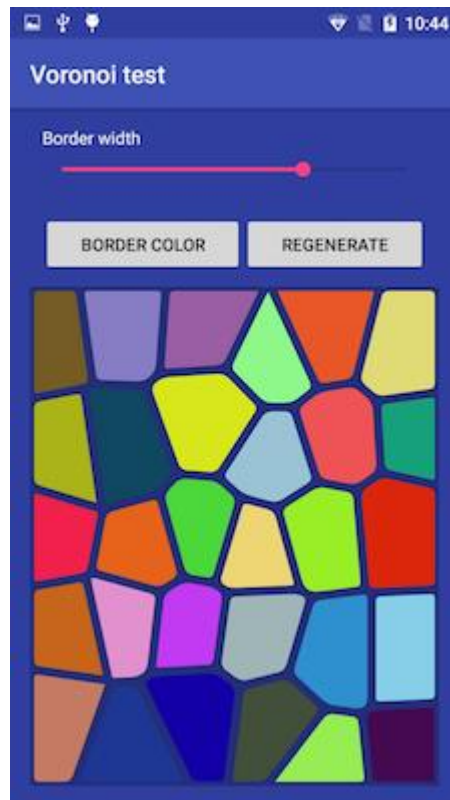
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello world, I have physics!"/>

</com.jawnnypoo.physicslayout.PhysicsLinearLayout>
```

# VoronoiView / Vorlay

- Márvány szerű elrendezés
- <https://github.com/Quatja/Vorolay>





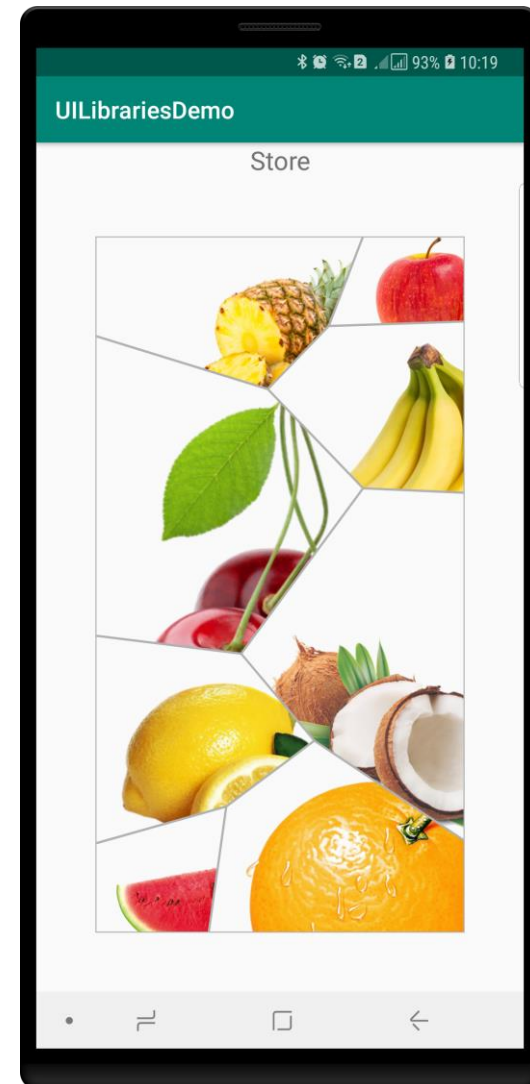
# VoronoiView példa

```
<quatja.com.vorolay.VoronoiView
    android:id="@+id/voronoiView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="50dp"
    custom:border_color="#b1b1b1"
    custom:border_round="false"
    custom:border_width="5"
    custom:generation_type="random"/>

class VoronoiActivityActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_voronoi_activity)

        addFruit(R.drawable.ananas)
        addFruit(R.drawable.apple)
        addFruit(R.drawable.banana)
        addFruit(R.drawable.cherry)
        addFruit(R.drawable.coconut)
        addFruit(R.drawable.lemon)
        addFruit(R.drawable.melone)
        addFruit(R.drawable.orange)

    }
    private fun addFruit(fruitDrawableId: Int) {
        val imageView = ImageView(this);
        imageView.setImageResource(fruitDrawableId);
        voronoiView.addView(imageView)
    }
}
```





# GifImageView

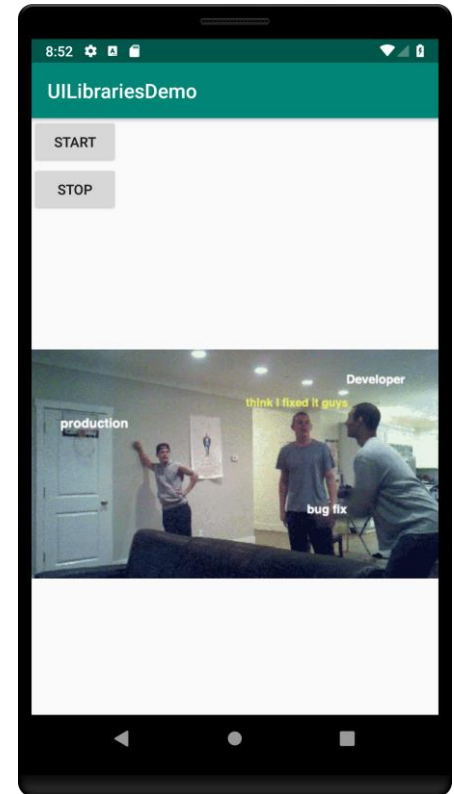
- GIF-ek lejátszása egyszerűen
- <https://github.com/felipecsl/GifImageView>

```
<com.felipecsl.gifimageview.library.GifImageView  
    android:id="@+id/gifImageView"  
    android:layout_gravity="center"  
    android:scaleType="fitCenter"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

```
btnStart.setOnClickListener {  
    gifImageView.setBytes(IOUtils.toByteArray(  
        resources.openRawResource(R.raw.developer))  
    gifImageView.startAnimation()  
}
```

- IOUtils – Apache Common IO

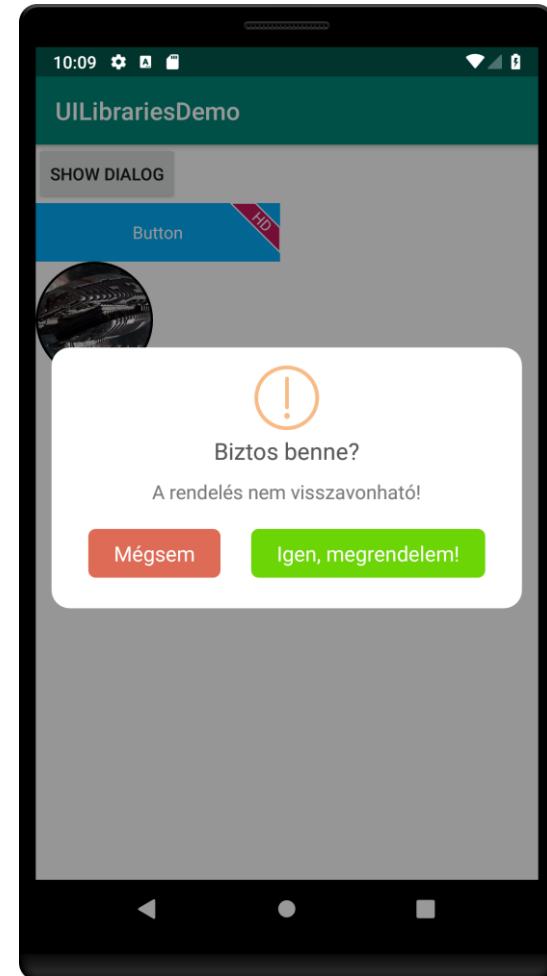
```
> implementation 'commons-io:commons-io:+'
```



# SweetAlertDialog

- Látványos, több funkciós dialógusok megjelenítése egyszerűen
  - > <https://jitpack.io/p/Leogiroux/sweet-alert-dialog>

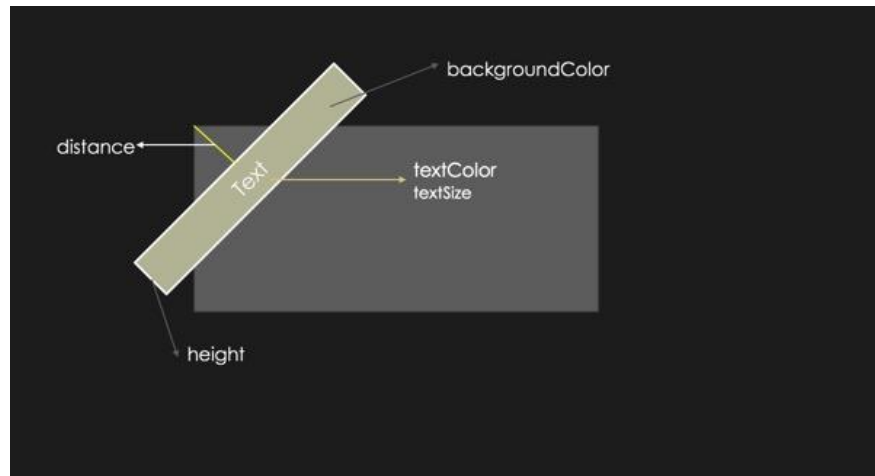
```
SweetAlertDialog(this, SweetAlertDialog.WARNING_TYPE)
    .setTitleText("Biztos benne?")
    .setContentText("A rendelés nem visszavonható!")
    .setConfirmText("Igen, megrendelem!")
    .setConfirmClickListener{
        it.dismissWithAnimation()
    }
    .setCancelButton("Mégsem") {
        sDialog -> sDialog.dismissWithAnimation() }
    .show()
```



# LabelView

- Címkék elhelyezése gombon, képen, szövegen

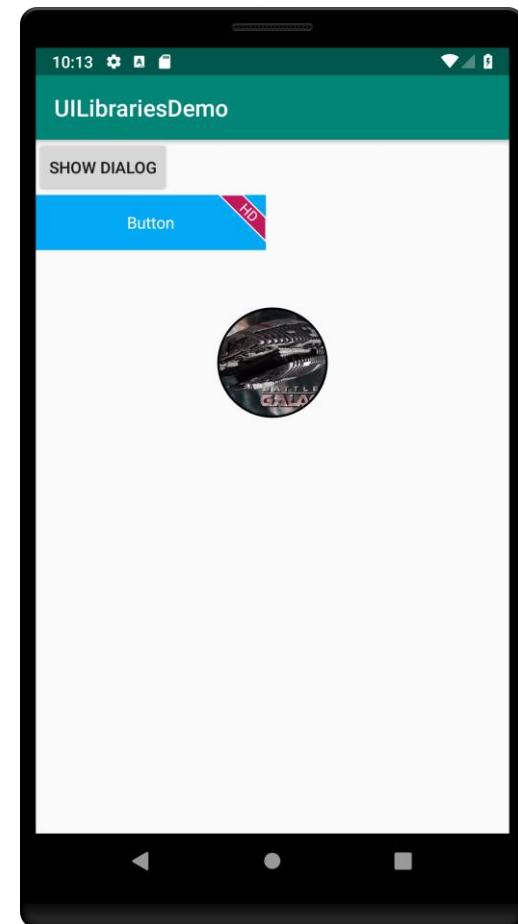
```
<com.lid.lib.LabelButtonView
    android:id="@+id/labelbutton"
    android:layout_width="200dp"
    android:layout_height="48dp"
    android:background="#03a9f4"
    android:gravity="center"
    android:text="Button"
    android:textColor="#ffffff"
    app:label_backgroundColor="#C2185B"
    app:label_distance="20dp"
    app:label_height="20dp"
    app:label_orientation="RIGHT_TOP"
    app:label_text="HD"
    app:label_textSize="12sp" />
```



# CircleImageView

- Kör formában kivágott képek (pl. profil képek)
  - > <https://github.com/hdodenhof/CircleImageView>

```
<de.hdodenhof.circleimageview.CircleImageView  
    android:id="@+id/circleImage"  
    android:layout_width="96dp"  
    android:layout_height="96dp"  
    android:src="@drawable/galactica"  
    app:civ_border_width="2dp"  
    app:civ_border_color="#FF000000"/>
```



# További UI gyűjtemények

- <https://github.com/wasabeef/awesome-android-ui>
- <https://android-arsenal.com/>
- <https://github.com/JStumpp/awesome-android>
- <https://www.uplabs.com/collections/library-b9e1d971-50e0-4600-ab9b-5f6b23fe3c0b>
- <https://medium.freecodecamp.org/25-new-android-libraries-which-you-definitely-want-to-try-at-the-beginning-of-2017-45878d5408c0>



# 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...

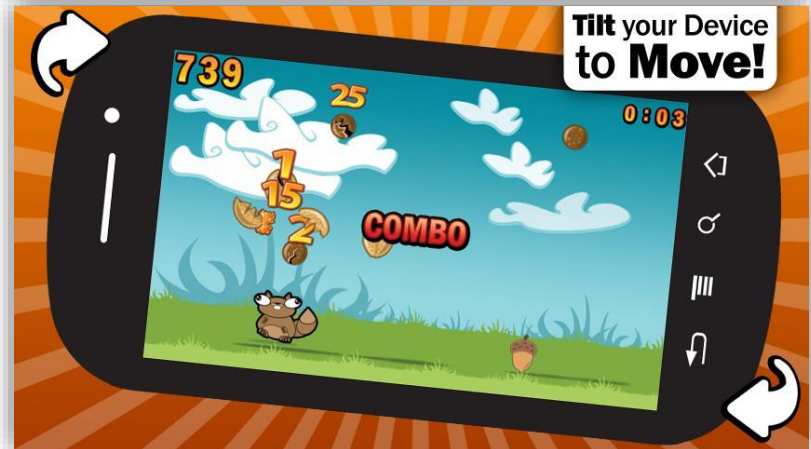
# Bevezetés

- Új lehetőségek az interakció megvalósítására

- > Kiterjesztett valóság

- > Mozgás alapú vezérlés

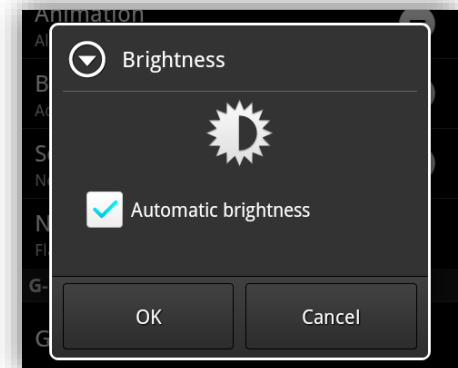
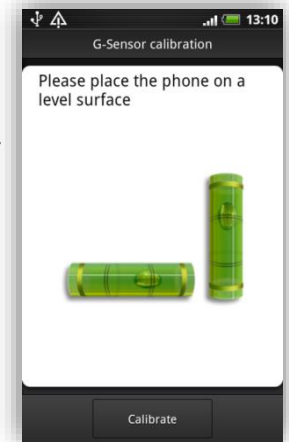
- > Stb...





# 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
```

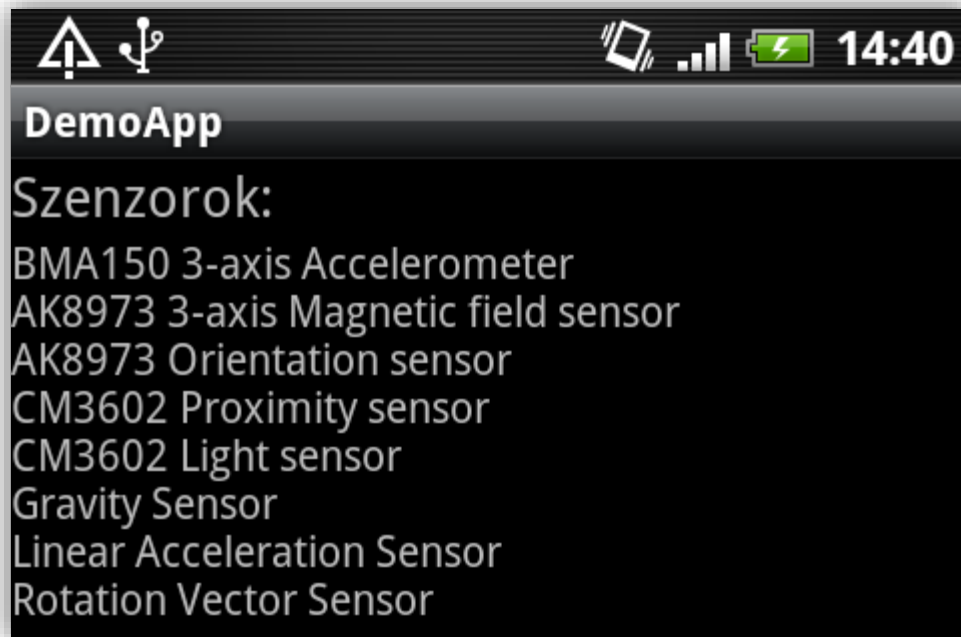
- Szenzor elkérése

```
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

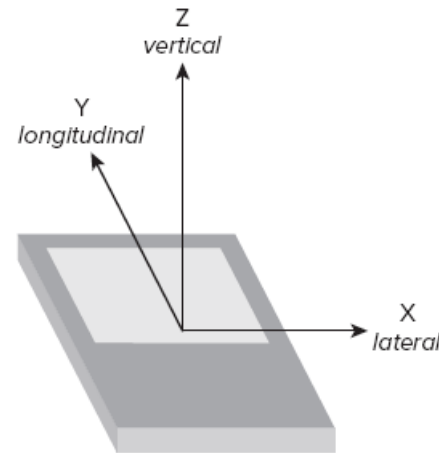
- Ezek segítségével
  - > Tudjuk, hogy mi az orientáció
  - > Mennyire van elforgatva a készülék
  - > Melyik égtáj felé néz a felhasználó
  - > Merre mozog a készülék
- További érzékelőkkel kiegészíthető
  - > Hol van (GPS)
  - > Mit lát a kamera
  - > Stb...

# 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 ( $9.81 \text{ m/s}^2$ )



# Gyorsulásmérő használata

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

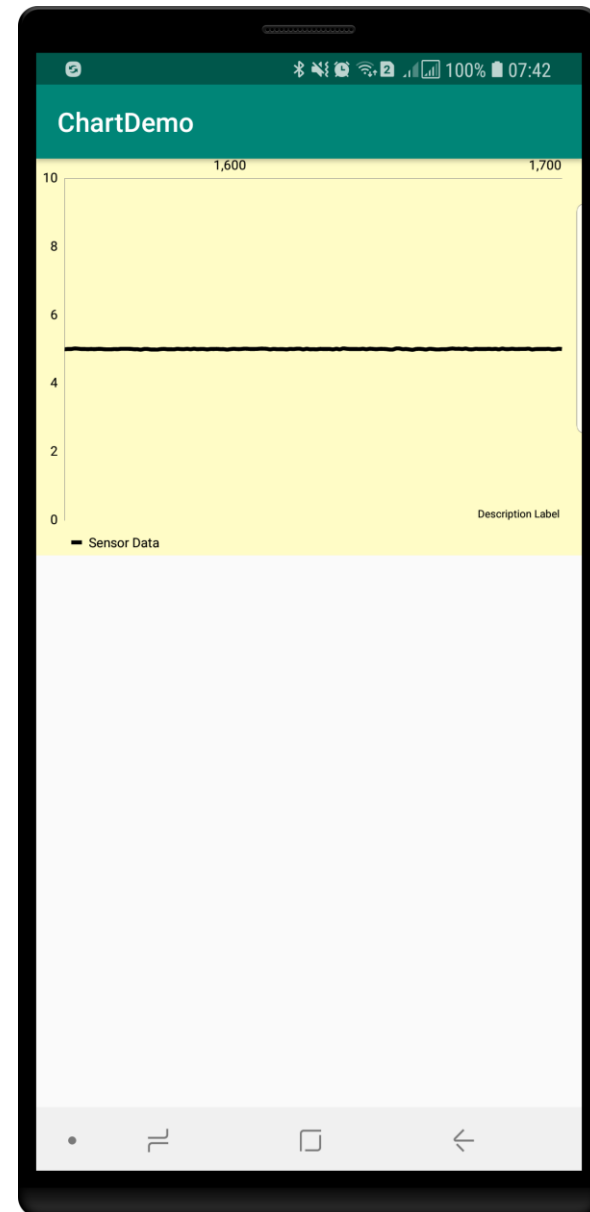
```
override fun onSensorChanged(event: SensorEvent) {  
  
    val accX = event.values[0].toDouble()  
    val accY = event.values[1].toDouble()  
    val accZ = event.values[2].toDouble()  
  
    var origin = Math.sqrt(  
        Math.pow(accX, 2.0) +  
        Math.pow(accY, 2.0) +  
        Math.pow(accZ,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



# Összefoglalás

- ContentProvider
- ConstraintLayout - haladó
- Drag&Drop általános kezelése
- Grafikonok rajzolása
- Gyakran használt külső UI library-k áttekintése



# Köszönöm a figyelmet!



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



**AutSoft**