

Kliensalkalmazások

Android 2 – Intent, UI, View

2023. 05. 05.

Gazdi László

gazdi.laszlo@aut.bme.hu



Automatizálási és
Alkalmazott
Informatikai Tanszék

Miről volt szó az előző órán?

- Android szerkezete
- Alkalmazás komponensek
- Manifest állomány
- Erőforrások
- Projekt felépítése
- Fordítás mechanizmusa
- Activity Életciklus
- Activity Back Stack
- ViewBinding



Tartalom

- Komponensek közti kommunikáció, Intent
- Erőforrás típusok
- Felhasználói felület alapok
- View/ViewGroup-ok
- Menü kezelés, Toolbar
- Felugró ablakok
- Stílusok&Témák

Komponensek közti kommunikáció, Intent

Bevezetés

- A legtöbb platformon az alkalmazások egymástól elkülönítve futnak
 - > Minden app a saját „homokozójában” (*Sandbox*)
 - > Szigorú korlátozások a sandbox-ból kinyúló műveletekre
 - Hardver elérés, pl kamera, szenzorok, stb
 - Rendszerszintű adatok, háttértár
 - Szálak, alk. komponensek közti kommunikáció
 - > Cél: adatvédelem, alkalmazások védelme egymástól

Lazán csatolt komponensek és köztük a kommunikáció

- Mi a helyzet Androidon?
 - > Alkalmazások külön ART VM példányokban (ez is sandbox)
 - > Kritikus műveletekhez engedély szükséges
 - > Alkalmazás = komponensek halmaza
 - > A komponensek akár alkalmazások között is kommunikálhatnak egymással (!)
- Két komponens között: Intent
 - > Egy komponensből mindenki másnak: Broadcast Intent
 - > Csak adat megosztása (ContentProvider)

Kommunikáció formái 1/3

- Egyik komponensből a másikba: *Intent*

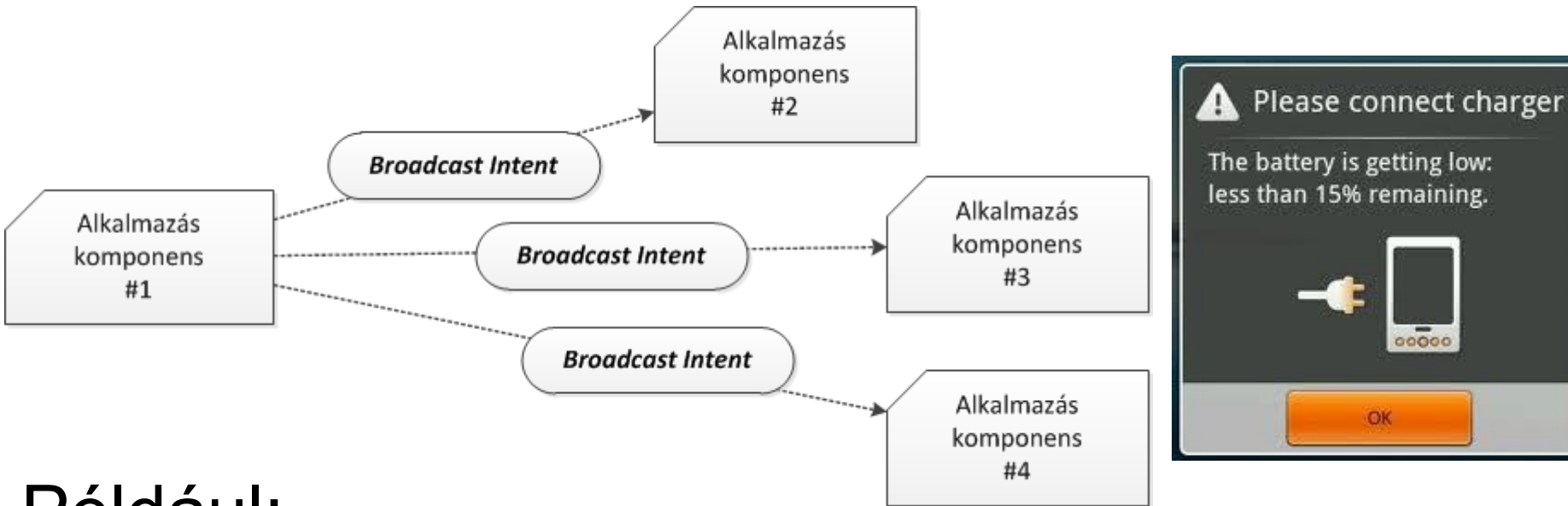


Például:

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

Kommunikáció formái 2/3

- Egy komponensből mindenki másnak: **Broadcast Intent**

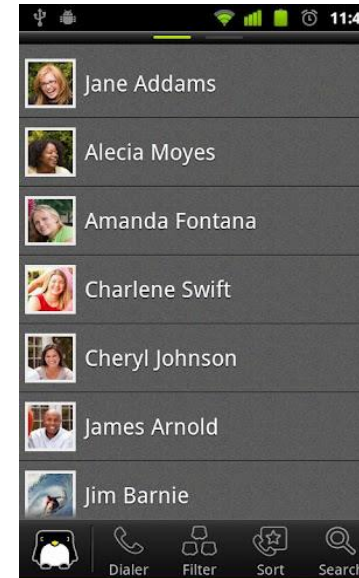
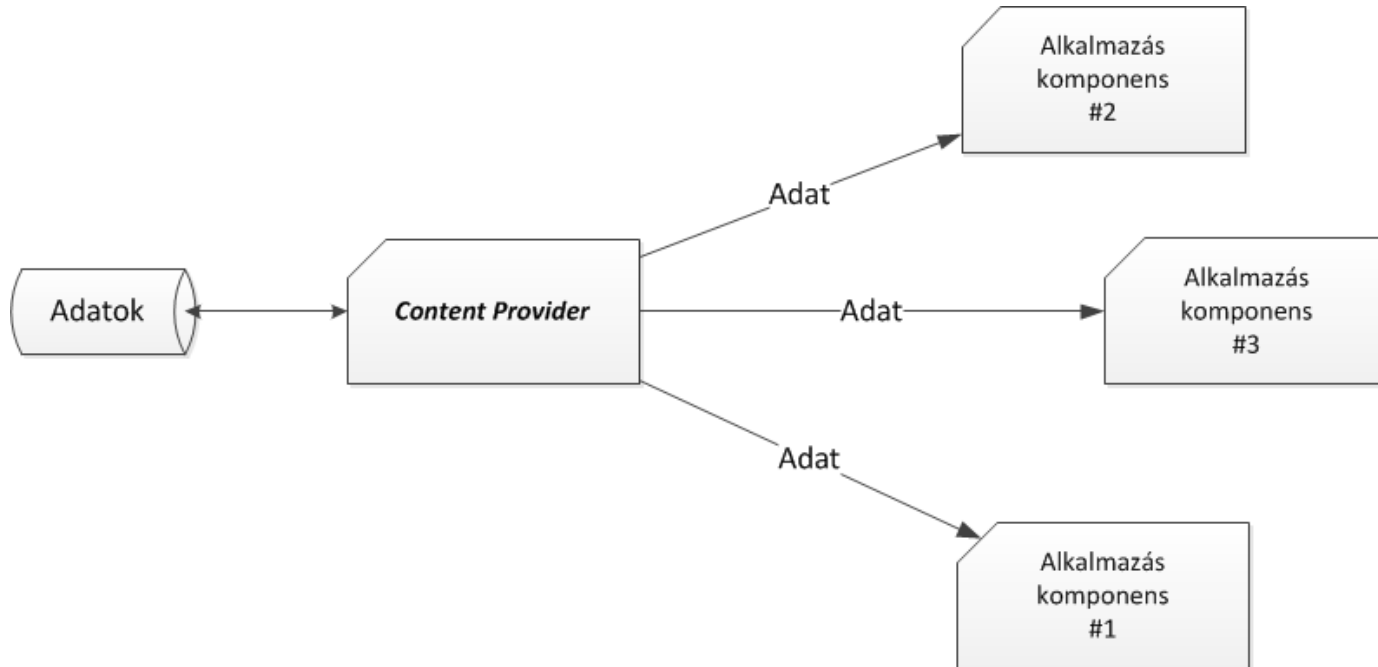


Például:

- „Akkufeszültség alacsony” rendszerüzenet

Kommunikáció formái 3/3

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

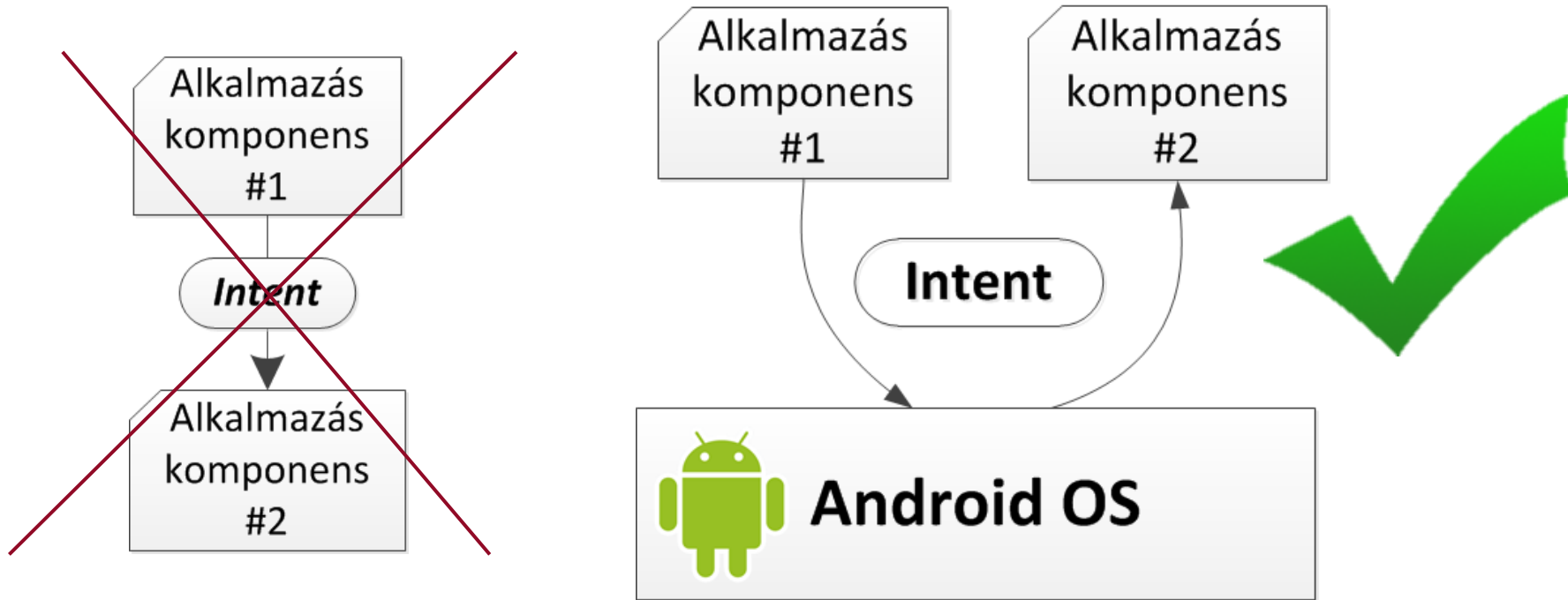


Például:

- Névjegyzék elérése saját alkalmazásból

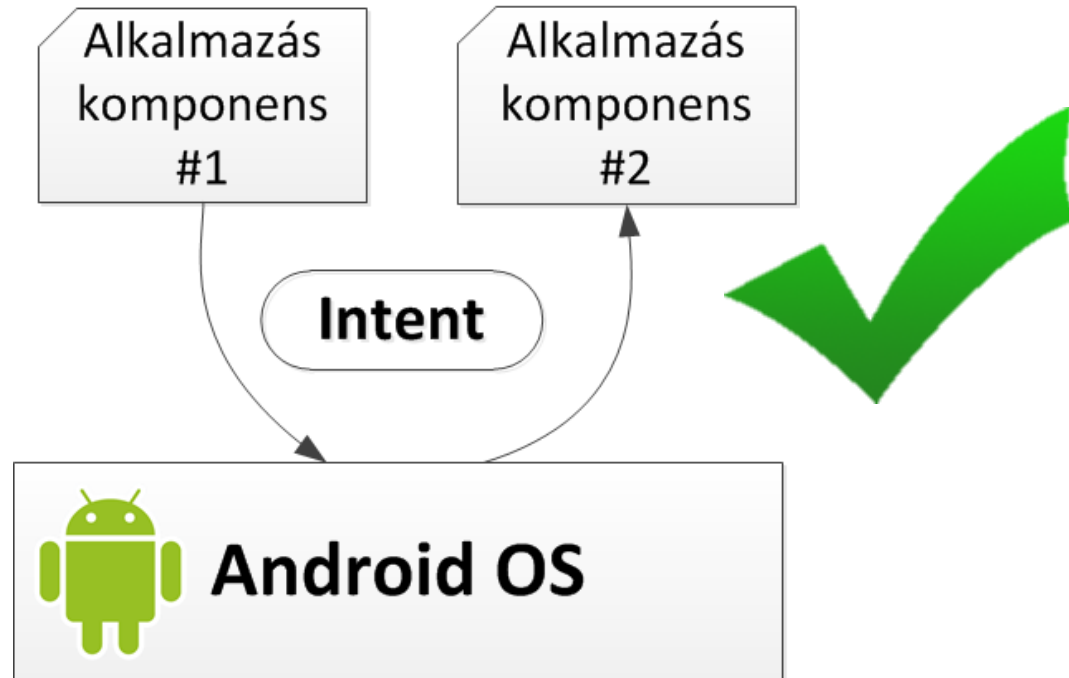
Intent átadása

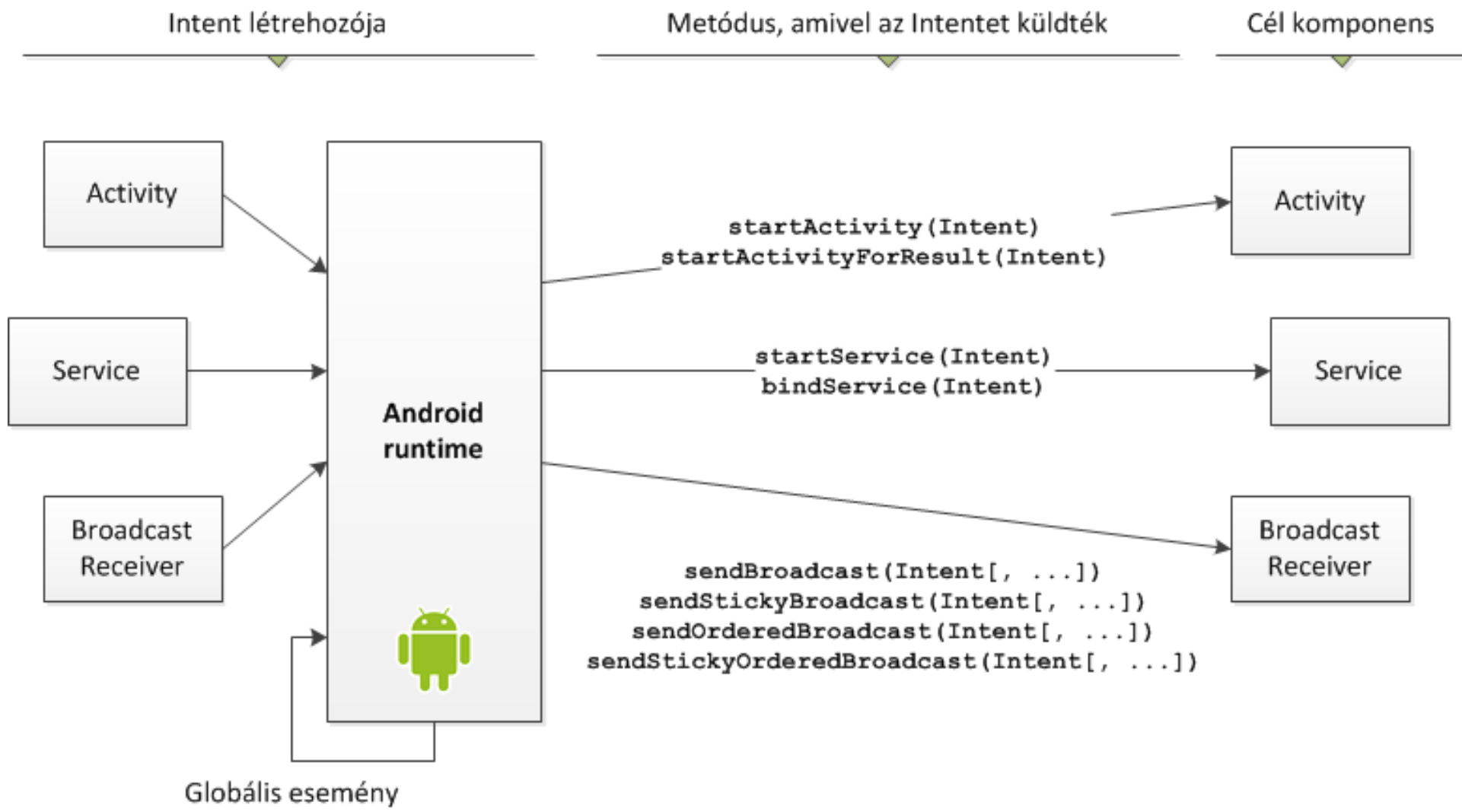
- Mindig az Android runtime-on keresztül!



Intent átadása

- Mindig az Android runtime-on keresztül!





Intent (*szándék*)

- Passzív adatstruktúra (~struct)
- Késői (futás idejű) kötést valósít meg alkalmazás komponensek között
 - > Komponensek: Activity, Service, Broadcast Receiver
- Az elvárt vagy bekövetkezett esemény absztrakt leírása
 - > Elvárt esemény leírása, ha az Intent hatására történik valami
 - Activity, Service, Broadcast Receiver regisztrálása / aktiválása
 - > Bekövetkezett esemény leírása, ha az Intent valamilyen esemény hatására jön létre
 - Broadcast üzenet (főleg rendszer események)

Intent típusai és részei

- **Intent típusok:**

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

- **Intent részei:**

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

Explicit Intent

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

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

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

Activity visszatérése

- Egy Android alkalmazás általában több Activity-ből épül fel, amik egymást indítják



- Gyakran szükséges visszajelzés arról, hogy a hívott Activity hogyan fejeződött be
 - > Melyik névjegyet választotta a felhasználó?
 - > Történt megrendelés?
 - > Bejelentkezés sikeres?

startActivityForResult

- **startActivity()** -vel indítva nem kapunk visszajelzést
- Megoldás:
`startActivityForResult(Intent, requestCode)`
- Több ilyen is lehet egy Activity-ben, a **requestCode** nevű integer különbözteti meg őket
- Az így indított Activity-k befejeződésük után vissza tudnak jelezni a hívónak
 - > **finish()** előtt **setResult(requestCode)** a hívott oldalon

onActivityResult

- Visszatérési érték kezelése a hívó oldalon (callback):

```
onActivityResult(requestCode, resultCode, extras) {...}
```

- Paramétereit:
 - > **requestCode**: integer, ugyanaz mint a **startActivityForResult**-ban megadott

onActivityResult

```
onActivityResult(requestCode, resultCode, extras) {...}
```

> **resultCode**: integer, eredmény számkódja

- **Activity.RESULT_OK** (= -1)
- **Activity.RESULT_CANCELED** (= 0, ezzel tér vissza akkor is, ha a Vissza gombra nyomott a user)
- Sajátot is definiálhatunk, például:

```
Companion object {
```

```
    const val RESULT_ORDER_SUCCESS = 2;
```

```
    const val RESULT_LOGIN_OK = 3
```

```
    const val RESULT_LOGIN_FAIL = 4
```

```
}
```


← Miért companion object?

Intent Extras

- Feltöltése:

> **intent.putExtra (name, value) ;**

```
var resultIntent: Intent = Intent  
resultIntent.putExtra (  
    "UserID", currentUser.getId()  
resultIntent.putExtra ("role", currentUser.getRole())  
setResult (RESULT_OK, resultIntent)  
finish()
```



Intent Extras

- Lekérdezése:

> `intent.getTypeName().getStringExtra(name, defaultValue)`

Implicit Intent

- Implicit hívás: azt mondjuk meg, hogy milyen akció történjen
 - > Ha szükséges, akkor azt is, hogy milyen adat(ok)on
- Hívás gomb megnyomásának szimulálása:

```
var i: Intent = Intent(Intent.ACTION_CALL_BUTTON)
startActivity(i)
```

Implicit Intent - Példa

- Telefonszám felhívása

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

Akció

Adat (URI)

- Névjegy kiválasztása

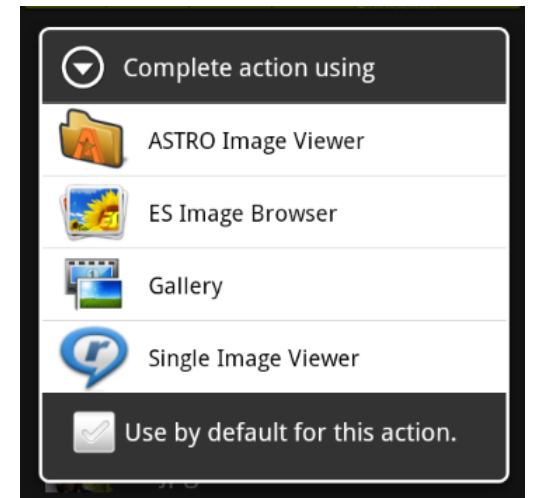
```
var i: Intent = Intent(Intent.ACTION_PICK,  
                      ContactsContract.Contacts.CONTENT_URI)  
startActivity(i)
```

Akció

Adat (URI)

Implicit Intent – Több célpont

- *ActivityNotFoundException*, ha nem talál megfelelőt
- Amennyiben több alkalmazás is képes a kért akcióra: „*Complete action using*” dialógusablak
 - *Ha nincs default megadva, akkor a felhasználó választ*
 - *Egyébként indul az alapértelmezett alkalmazás*
 - *(új app telepítése után újra rákérdez az alapértelmezésre)*



Intent képességek

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

Intent Filter

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

Mit nevezünk Explicit Intentnek?

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

Erőforrás típusok

Erőforrás használat előnyei

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

Gyakran használt erőforrások

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

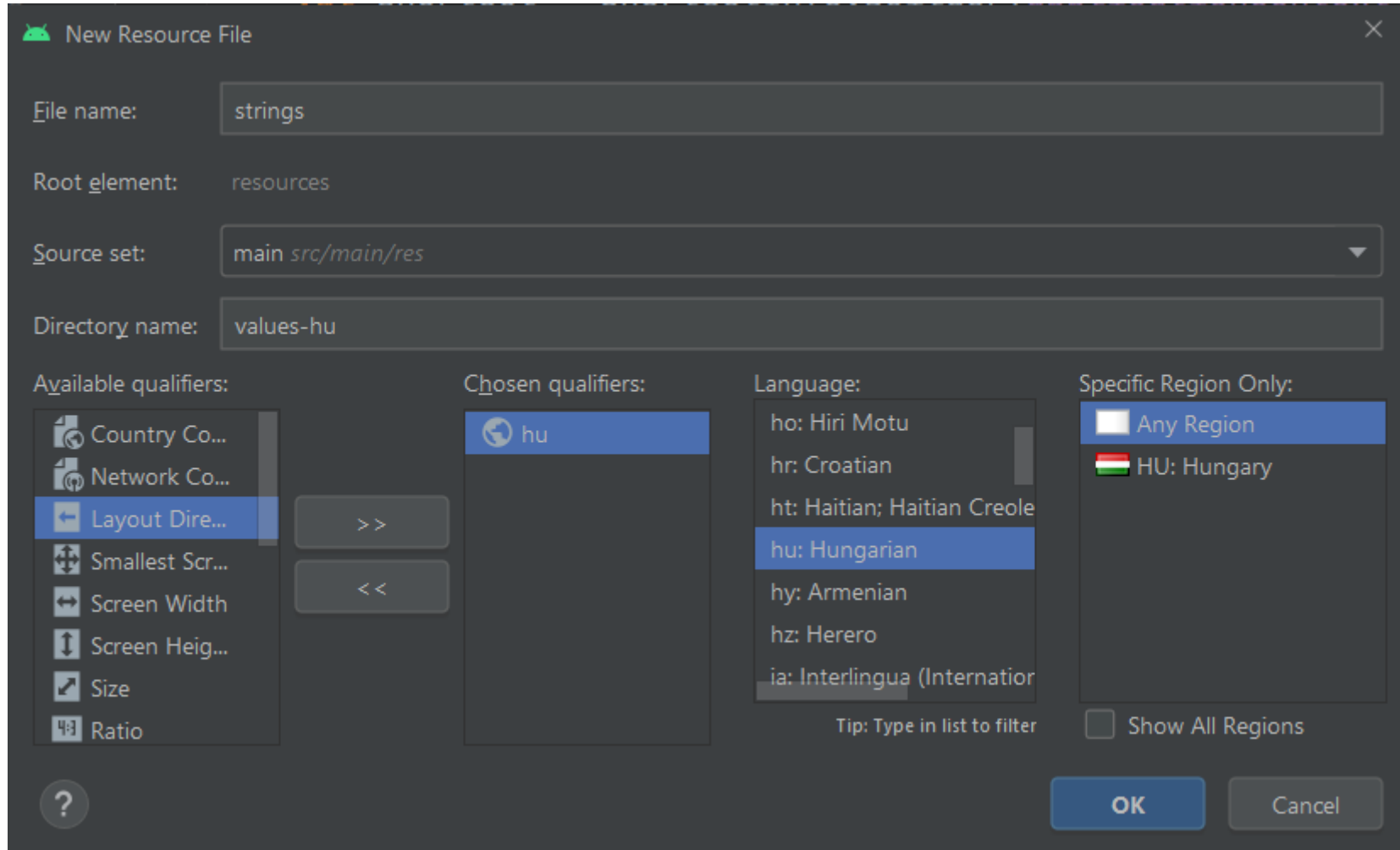
Szöveges erőforrások

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

Internalizáció/Lokalizáció

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

Lokalizált erőforrás hozzáadása



Felhasználói felület alapfogalmak

Különböző képernyők támogatása 1/2

- Az Android futtatható különböző felbontású és sűrűségű képernyőkön
- A rendszer egyfajta mechanizmust biztosít az eltérő képernyők támogatására (1.6-tól felfele)
- A fejlesztő válláról a legtöbb munkát leveszi
- Csak a megfelelő erőforrásokat kell elkészíteni
- Például egy mobiltelefon és egy tablet képernyője tipikusan eltérő
- 3.2-es tablet API-tól felfele újabb módszerek (lásd később)

Különböző képernyők támogatása 2/2

- A rendszer automatikusan is skálázza és átméretezi az alkalmazás felületét, hogy minden készüléket támogasson
- De! mindenképp fontos, hogy a felhasználói felület és az erőforrások (képek) optimalizálva legyenek az egyes felbontásokhoz és sűrűségekhez
- Ezzel nagy mértékben növelhető a felhasználói élmény
- Továbbá valóban az egyes készülékekhez igazítható a megjelenítés, ami növeli a felhasználói elégedettséget
- A módszer követésével minden készüléket támogató alkalmazás készíthető UI szempontjából egyetlen .apk-ba csomagolva

Legfontosabb fogalmak 1/2

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

Legfontosabb fogalmak 2/2

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

SP vs. DP

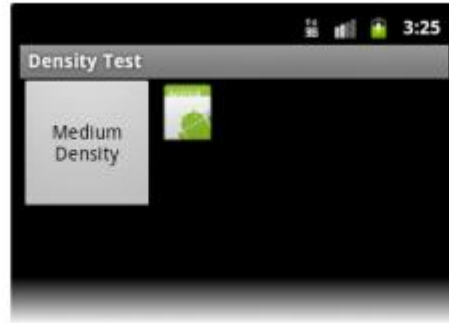
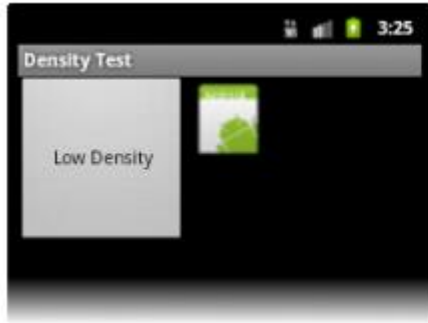
- „A dp is a density-independent pixel that corresponds to the physical size of a pixel at 160 dpi. An sp is the same base unit, but is scaled by the user's preferred text size (it's a scale-independent pixel), so you should use this measurement unit when defining text size (but never for layout sizes).”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

Sűrűség függetlenség

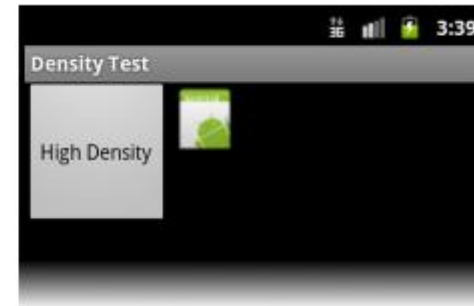
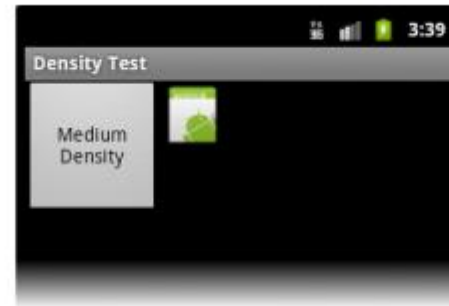
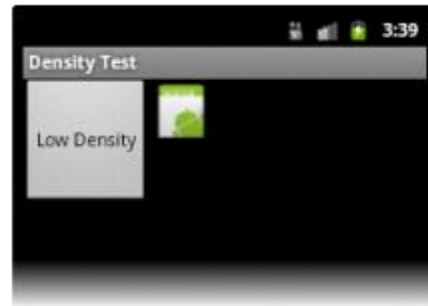
- Az alkalmazás akkor lehet „sűrűség független”, ha a felhasználói felületi elemek a felhasználó szemszögéből megőrzik a fizikai méretüket különböző sűrűségeken
- A „sűrűség függetlenség” fenntartása nagyon fontos, hiszen például egy gomb fizikailag nagyobbnak tűnhet egy alacsonyabb sűrűségű képernyőn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

Példa

- Sűrűség függetlenség támogatás nélkül:



- Sűrűség függetlenség támogatással:



Futás idejű működés

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

Erőforrás választó algoritmus

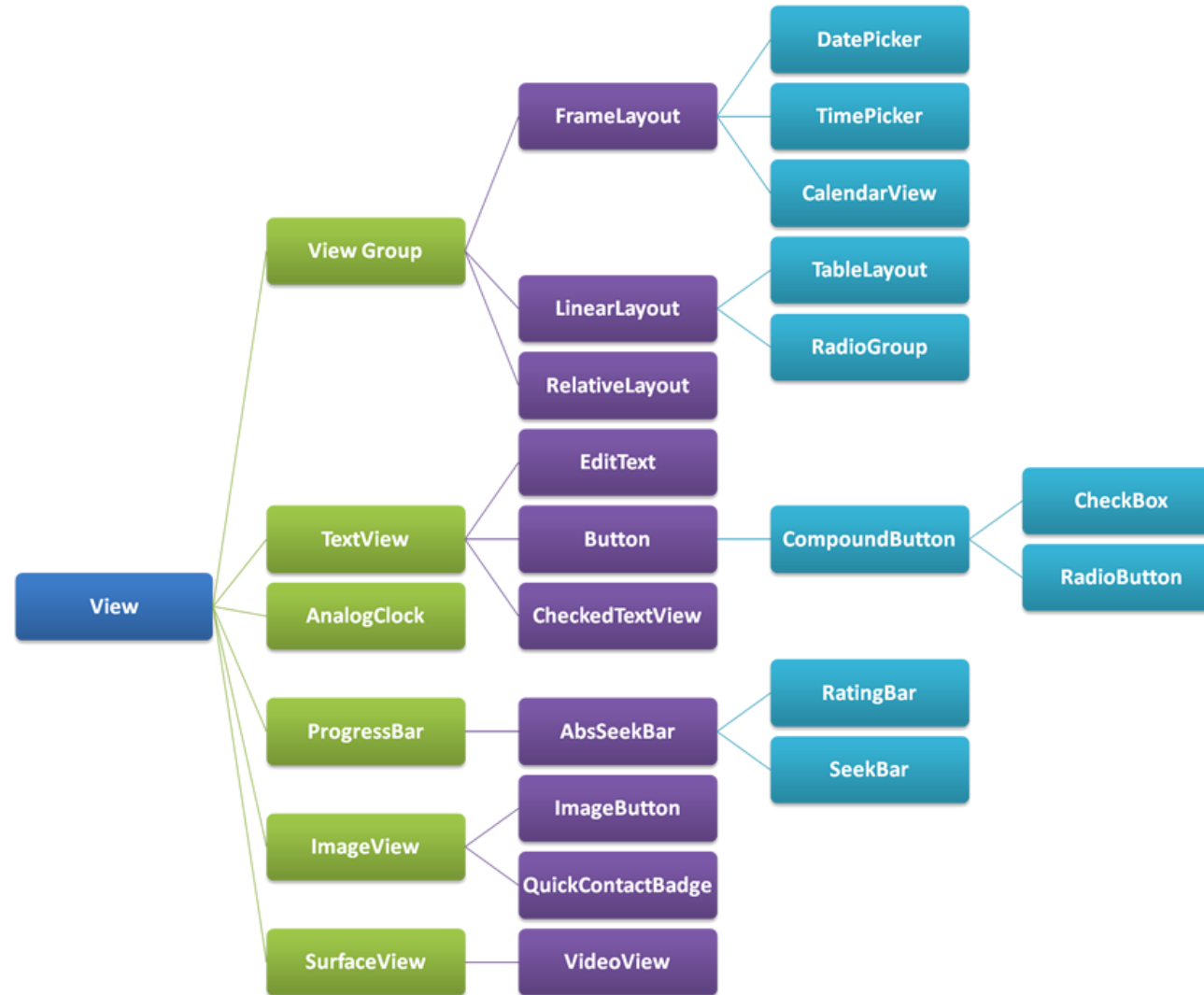
- Futás közben egy meghatározott logika alapján választ a rendszer
- Megkeresi a passzoló erőforrást az erőforrás minősítő alapján (könyvtár utoni postfix jelölő, pl `values-hu` vagy `layout-large`)
- Ha nincs az aktuálshoz passzoló, akkor egy kisebb/alacsonyabb sűrűségűt választ (pl. *large* mérethez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

Melyik állítás nem igaz?

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

Felhasználói felület felépítése

View hierarchia



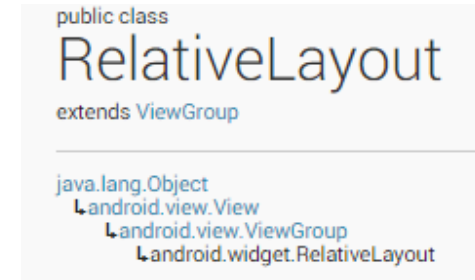
Android felhasználói felület felépítése

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

Layout-ok (ViewGroup)

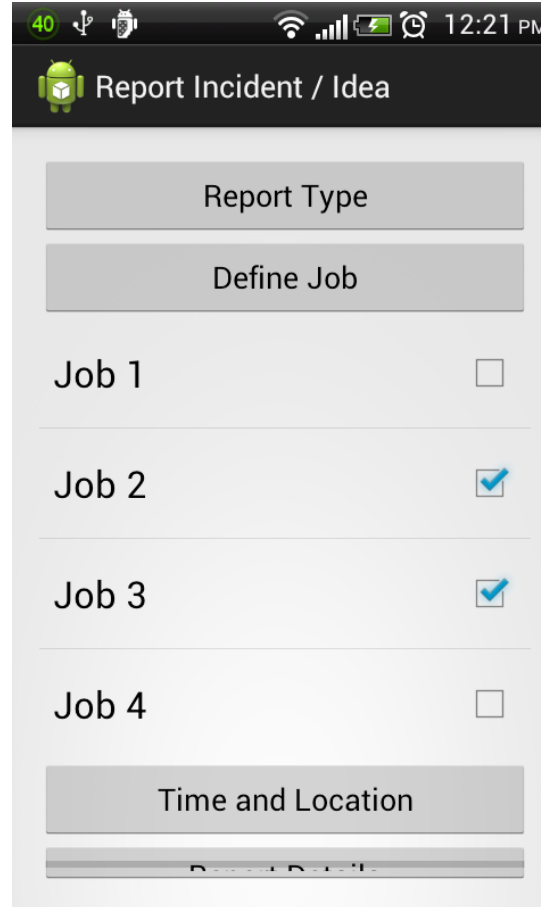
- **LinearLayout**
- RelativeLayout
- **ConstraintLayout** (~iOS AutoLayout)
- AbsoluteLayout (NEM használjuk!)
- GridLayout
- ...
- Teljes lista:

> <http://developer.android.com/reference/android/view/ViewGroup.html>



LinearLayout

- LinearLayout != Lista



Súlyozás Layout tervezéskor

- Megadható egy layout teljes súly értéke (weightSum)
- Elemek súly értéke megadható és az alapján töltődik ki a layout
 - > layout_weight érték
 - > A megfelelő width/height ilyenkor 0dp legyen!
- Hasonló, mint HTML-ben a %-os méret megadás

Layout súlyozás példa

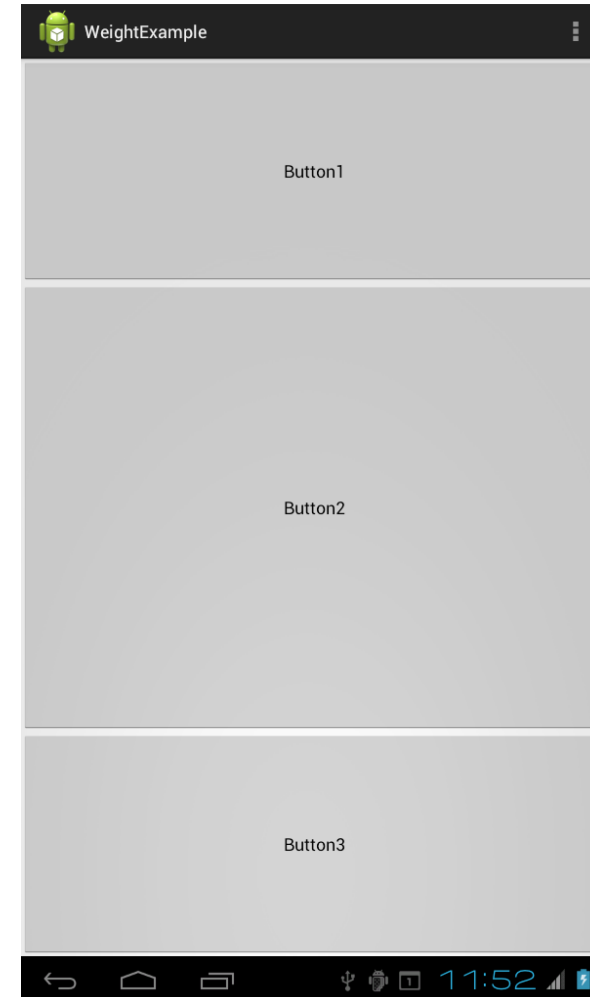
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="4"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Button1" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:text="Button2" />

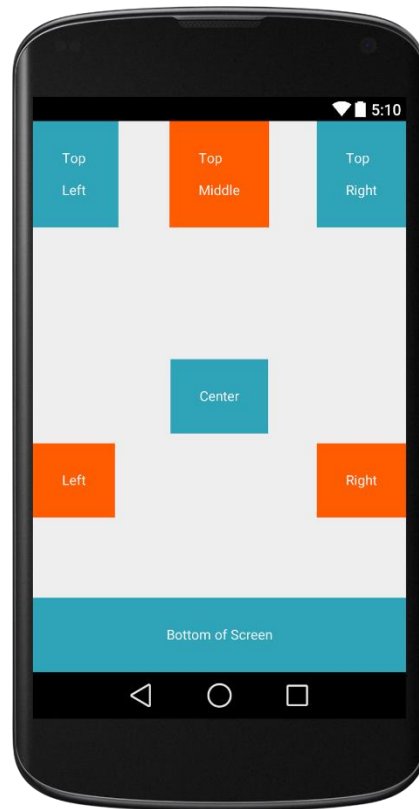
    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Button3" />

</LinearLayout>
```



RelativeLayout

- Elemek egymáshoz való viszonya definiálható



CONSTRAINTLAYOUT

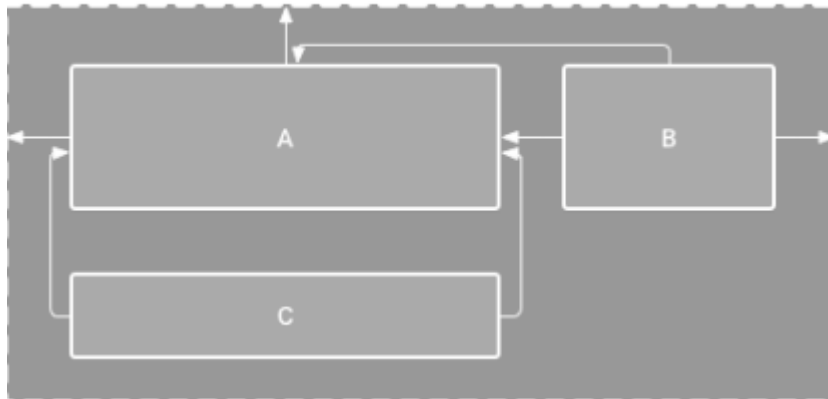
Reszponzív felületek ConstraintLayout-al

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

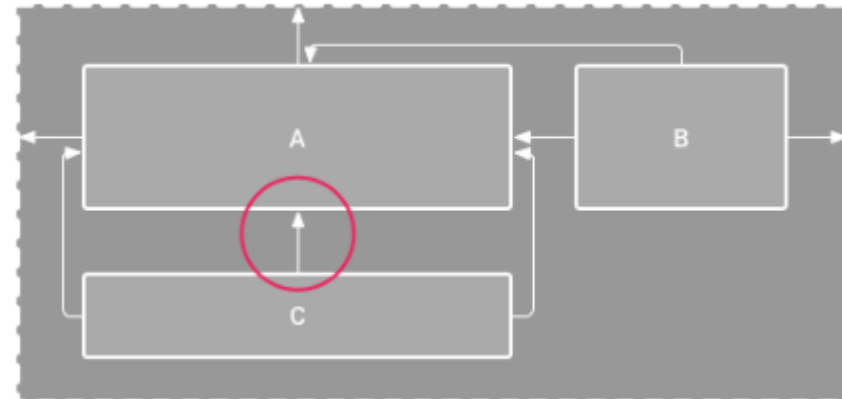
Reszponzív felületek ConstraintLayout-al

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

Hibás:

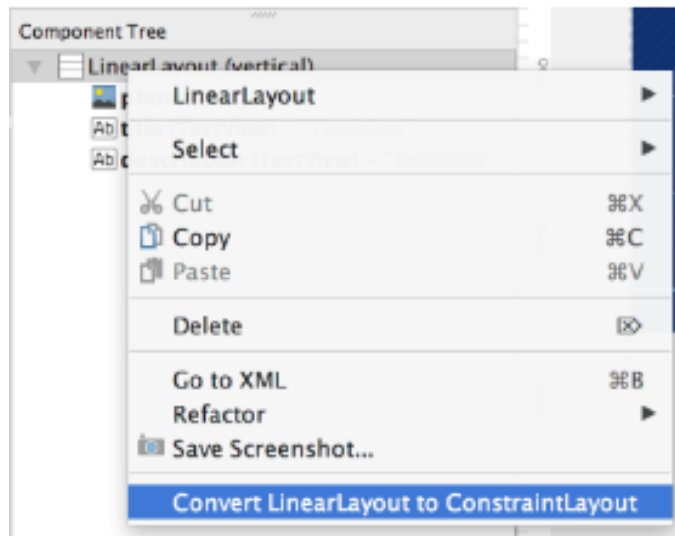


Helyes, mert C tudja, hogy A alatt van:



ConstraintLayout eszközök

- Gradle import:
 - > compile 'com.android.support.constraint:constraint-layout:2.2.0-alpha09'
- Automatikus átalakítás
 - > Nem tökéletes...

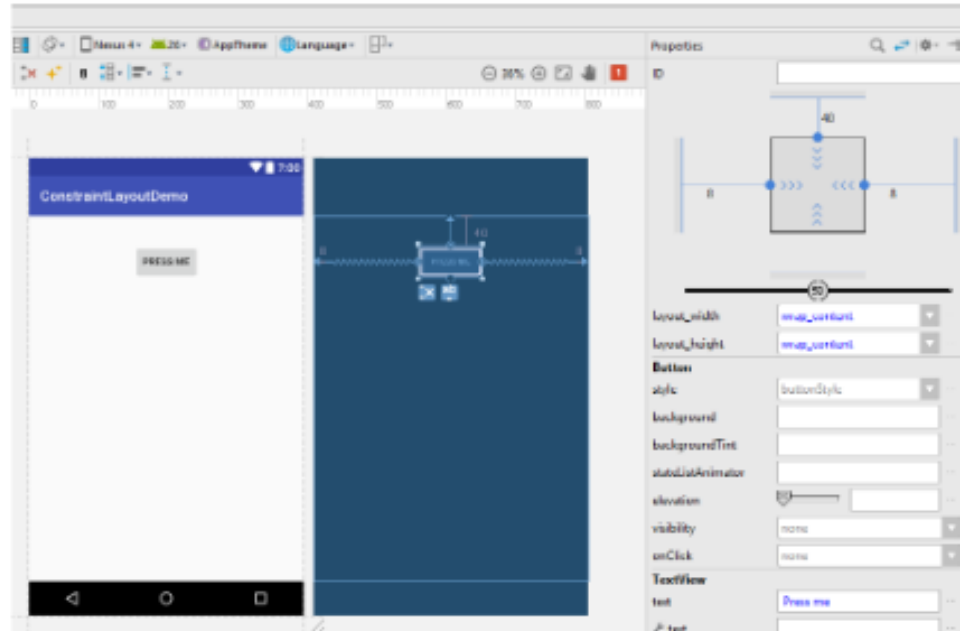


ConstraintLayout használat

- Kötelező legalább egy horizontális és vertikális „szabály”

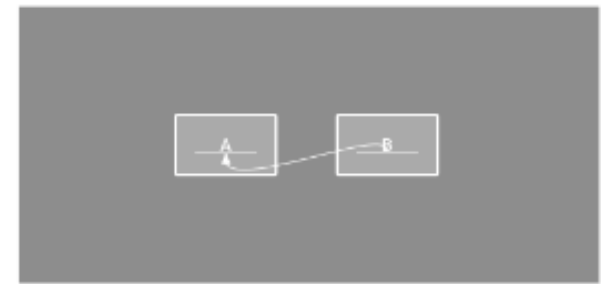
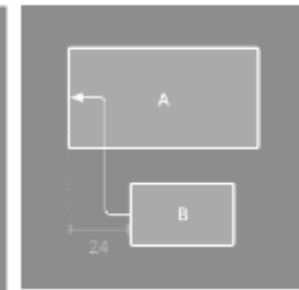
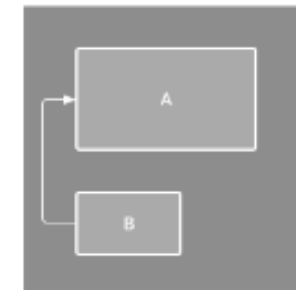
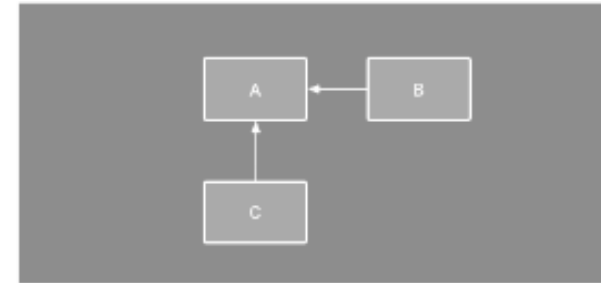
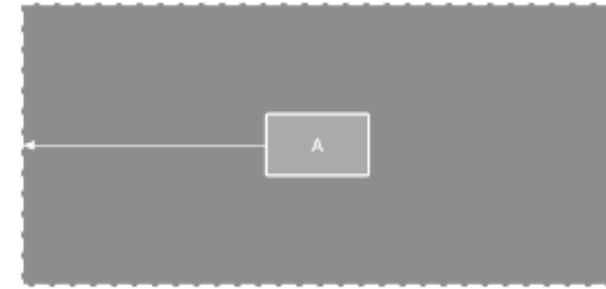
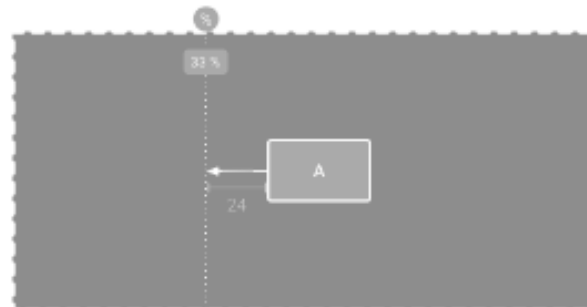
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
  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

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



Constraint lehetőségek

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



Padding és Margin



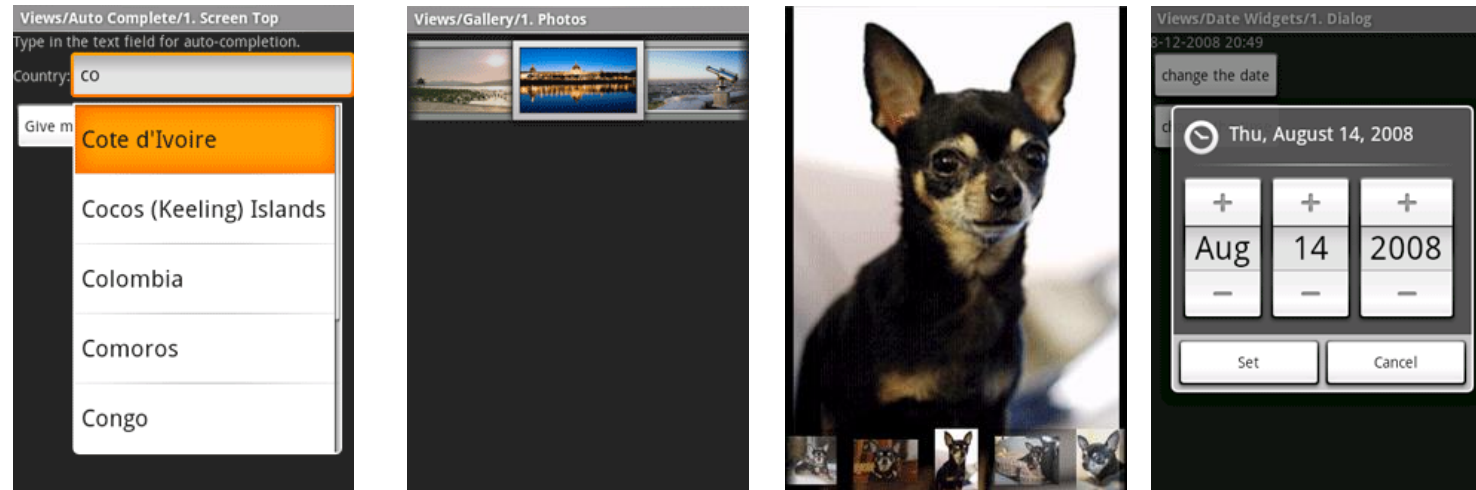
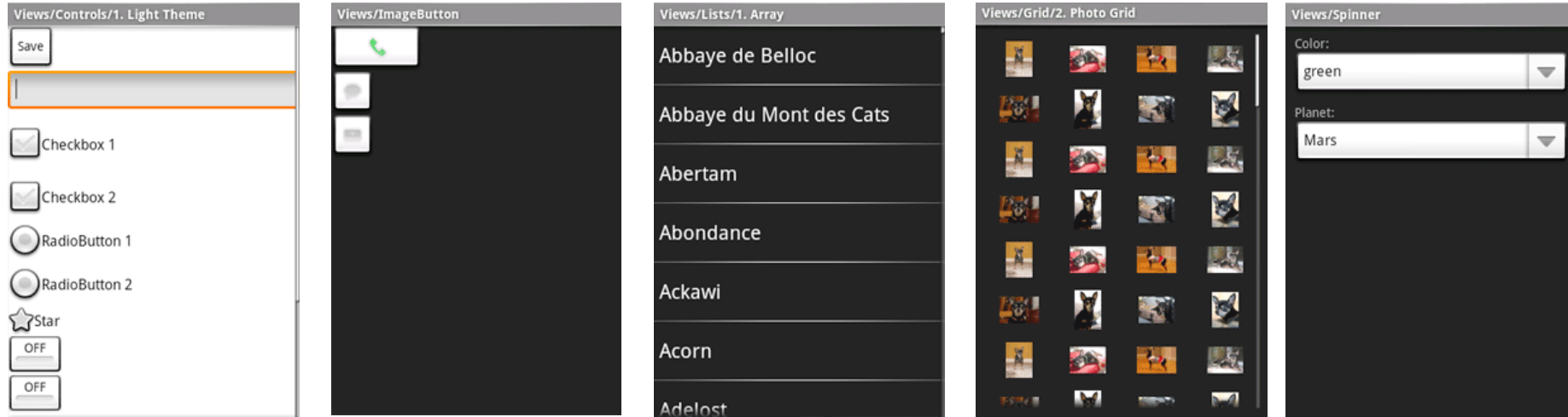
Nézetek (Widgetek/"View"-k)

View-k 1/2



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

View-k 2/2



Dinamikus UI kezelés - LayoutInflater

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

*// érdemes megadni a layout-ot ahova kerülni
// fog, hogy a méretezés optimális legyen*

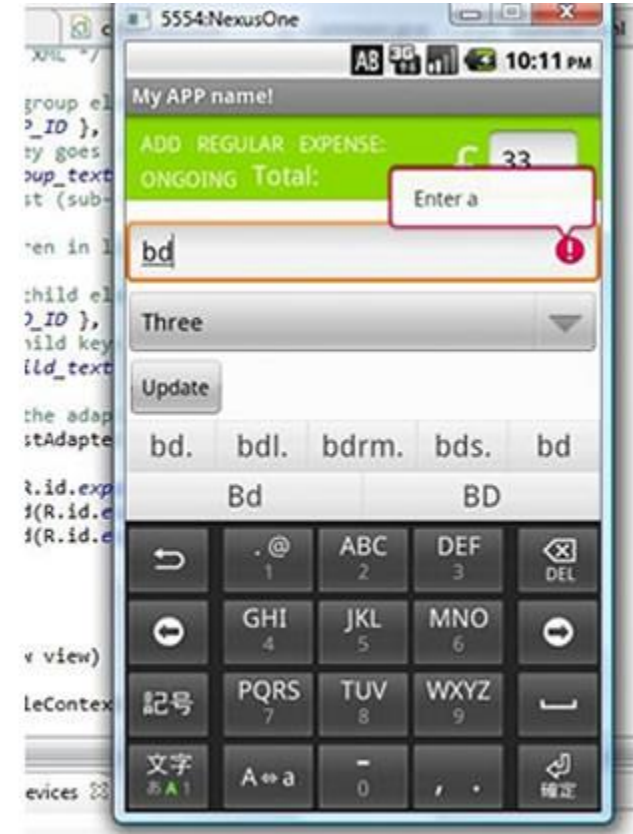
```
val myView = getLayoutInflater().inflate(  
    R.layout.activity_main, null)
```

Legfontosabb tényezők

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

Validáció támogatása

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



Menük

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

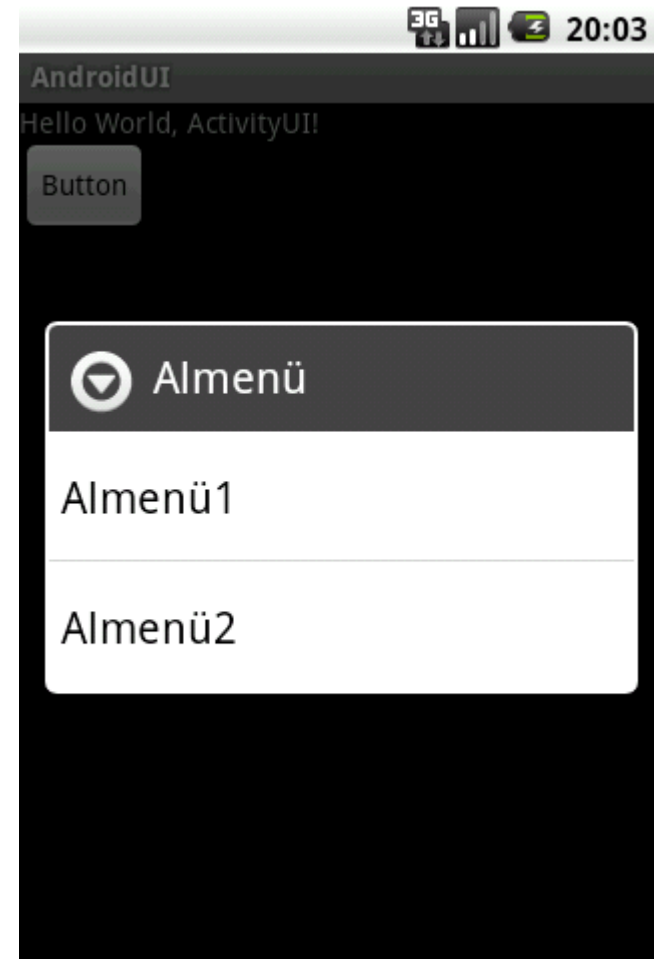
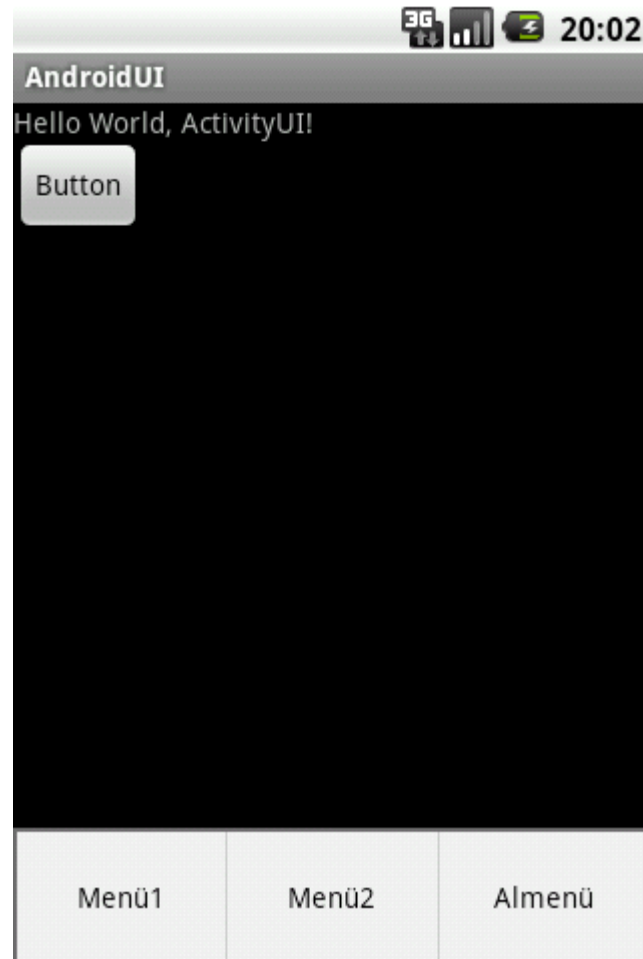
Menü erőforrás

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1"/>
    <item android:id="@+id/item2"
          android:title="@string/item2"/>
    <item android:id="@+id/submenu"
          android:title="@string/submenu_title">
        <menu>
            <item android:id="@+id/submenu_item1"
                  android:title="@string/submenu_item1" />
            <item android:id="@+id/submenu_item2"
                  android:title="@string/submenu_item2" />
        </menu>
    </item>
</menu>
```

Menü kezelése

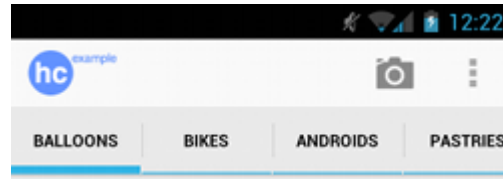
```
override fun onCreateOptionsMenu(menu: Menu): Boolean {  
    menuInflater.inflate(R.menu.mymenu, menu)  
  
    return true  
}  
  
override fun onOptionsItemSelected(item: MenuItem): Boolean {  
    Toast.makeText(this@ActivityUI,  
        item.getTitle(), Toast.LENGTH_LONG).show()  
  
    return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

Menü és almenü



ActionBar és Menük

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



ActionBar specifikus XML menü paraméterek

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

ActionBar -> Toolbar

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

Toolbar használat

- ActionBar nélküli téma(styles.xml):
 - > Theme.AppCompat.NoActionBar

- Layout erőforrás:

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

- Activity onCreate(...):

```
Lateinit var binding: ActivityMainBinding
Override fun onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    // Set a toolbar to replace the action bar.
    setSupportActionBar(binding.toolbar) ;
}
```

Mi igaz a felületi elemekre?

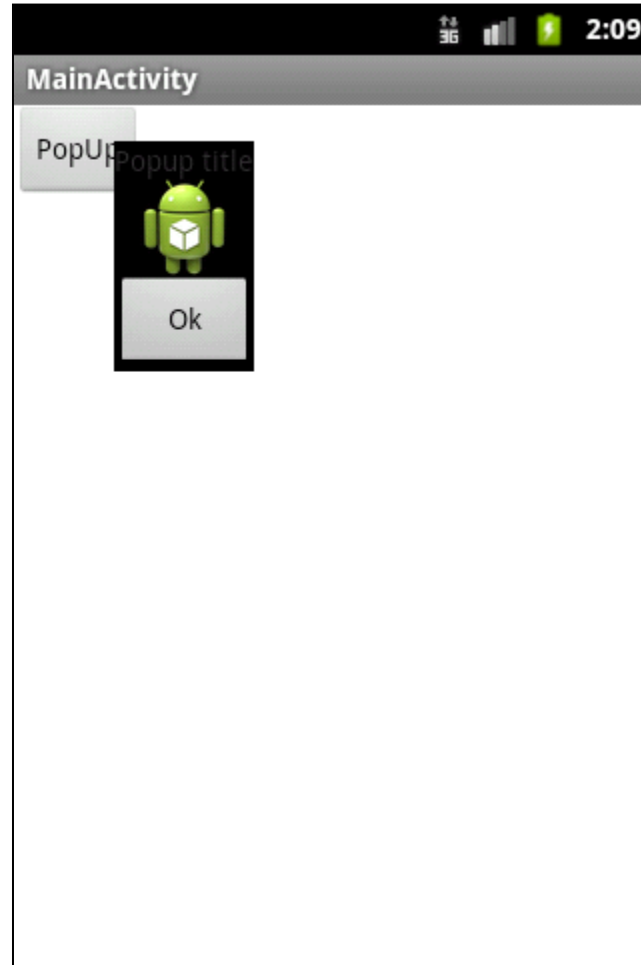
- A. A menü elemeket kódból kell felvennünk.
- B. A view-k méretét mindig pixelben adjuk meg.
- C. A szövegek méretezéséhez dp helyett sp-t használunk.
- D. A Toolbar nem támogatja a menü erőforrásokat.

FELUGRÓ ABLAKOK

Felugró ablakok

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

PopUpWindow példa



PopUp megjelenítése

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

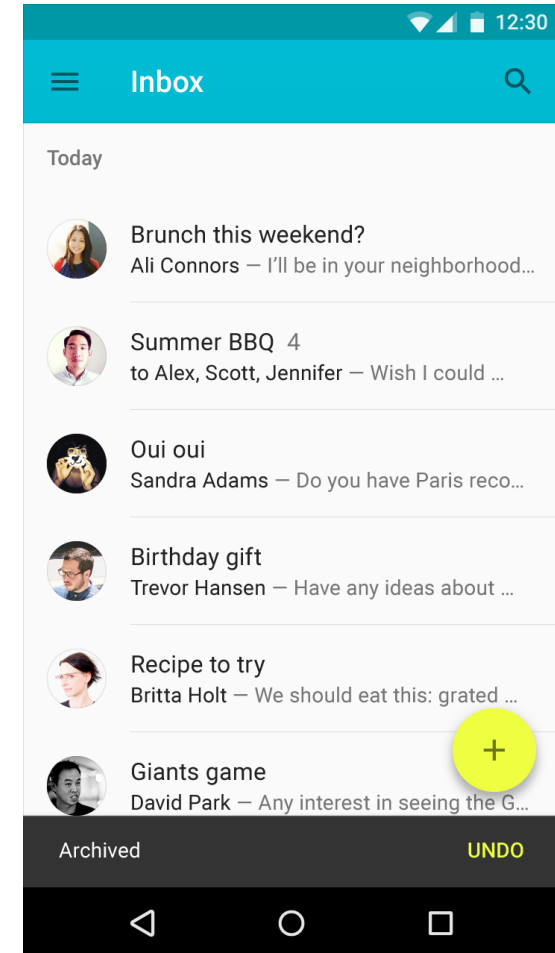
AlertDialog példa

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

SnackBar

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

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



STÍLUSOK, TÉMÁK

Stílusok készítése

- Stílus file: res/values/styles.xml

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

- Stílus használata:

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

Saját témák használata

- Stílusokhoz hasonlóan definiálhatók:

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

- A témák öröklődhetnek egymásból
- Manifestben állítható be:

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

Hogy is volt?

- Hogyan működik az implicit intent?
- Mit értünk a sűrűségfüggetlen pixel fogalom alatt?
- Egy 320 dpi-s képernyőn, 1 dp mennyi fizikai pixelnek felel meg ?
- Sorolja fel a legfontosabb Android Layout-okat!
- Hogy biztosítja az Android a lokalizáció támogatását?
- Mi a LayoutInflater feladata?
- Hogyan lehet elkészíteni és kezelni egy menüt Android platformon?
- Hogyan adható meg, hogy egy stílus az egész alkalmazásra érvényes legyen?

Összefoglalás

- Komponensek közti kommunikáció, Intent
- Erőforrás típusok
- Felhasználói felület alapok
- View/ViewGroup-ok
- Menü kezelés, Toolbar
- Felugró ablakok
- Stílusok&Témák

Köszönöm a figyelmet!

