

# Android

Fragment, RecyclerView, Komponens közi  
kommunikáció, Intent

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



Department of  
Automation and  
Applied Informatics

# Miről volt szó az előző órán?

- UI építő elemek
  - Layout (ViewGroup)
  - View-k
- Animációk
- MotionLayout
- Érdekesség:
  - > Shared element:  
<https://www.uplabs.com/posts/shared-element-transition-kotlin>

# Tartalom

- Fragment-ek
- RecyclerView és kiegészítések
- Komponens közi kommunikáció
- Intent

# Fragmentek

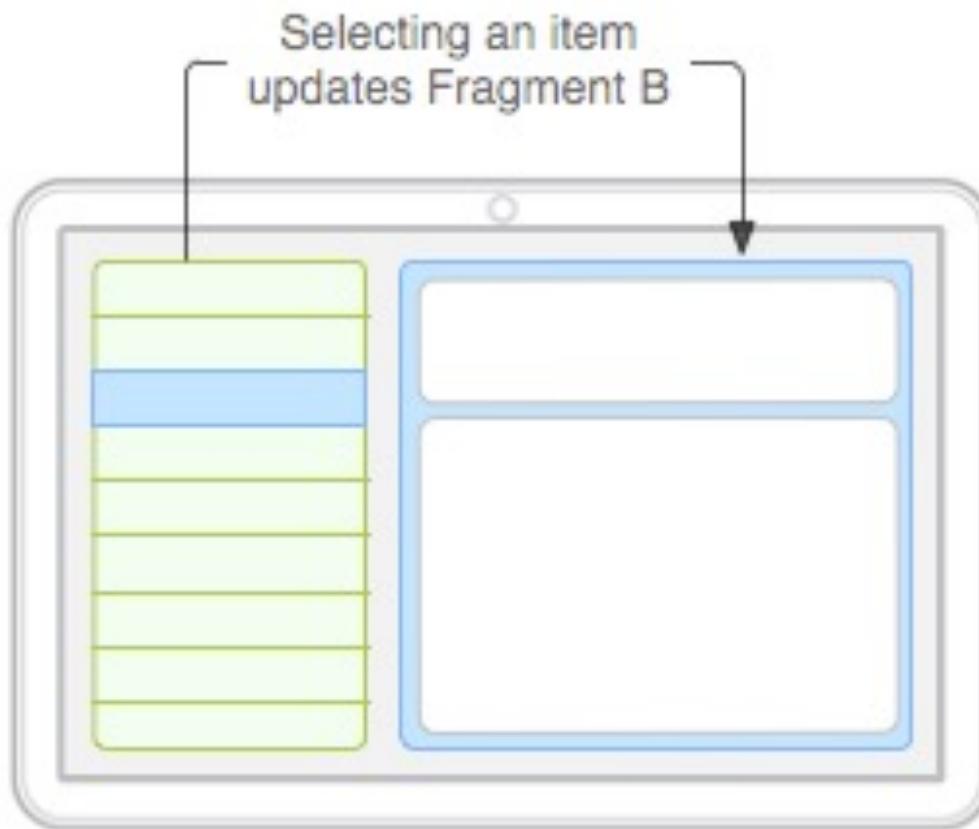
Csatolás, DialogFragment, ViewPager

# Fragment-ek

# Fragmentek

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

# Fragmentek



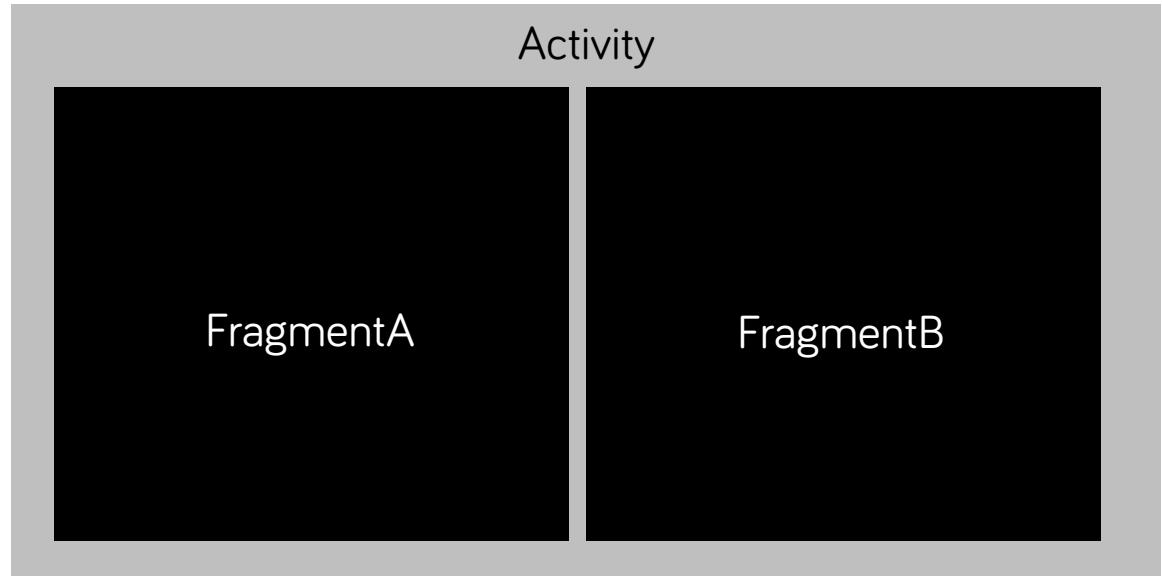
Activity A contains  
Fragment A and Fragment B

# Fragmentek

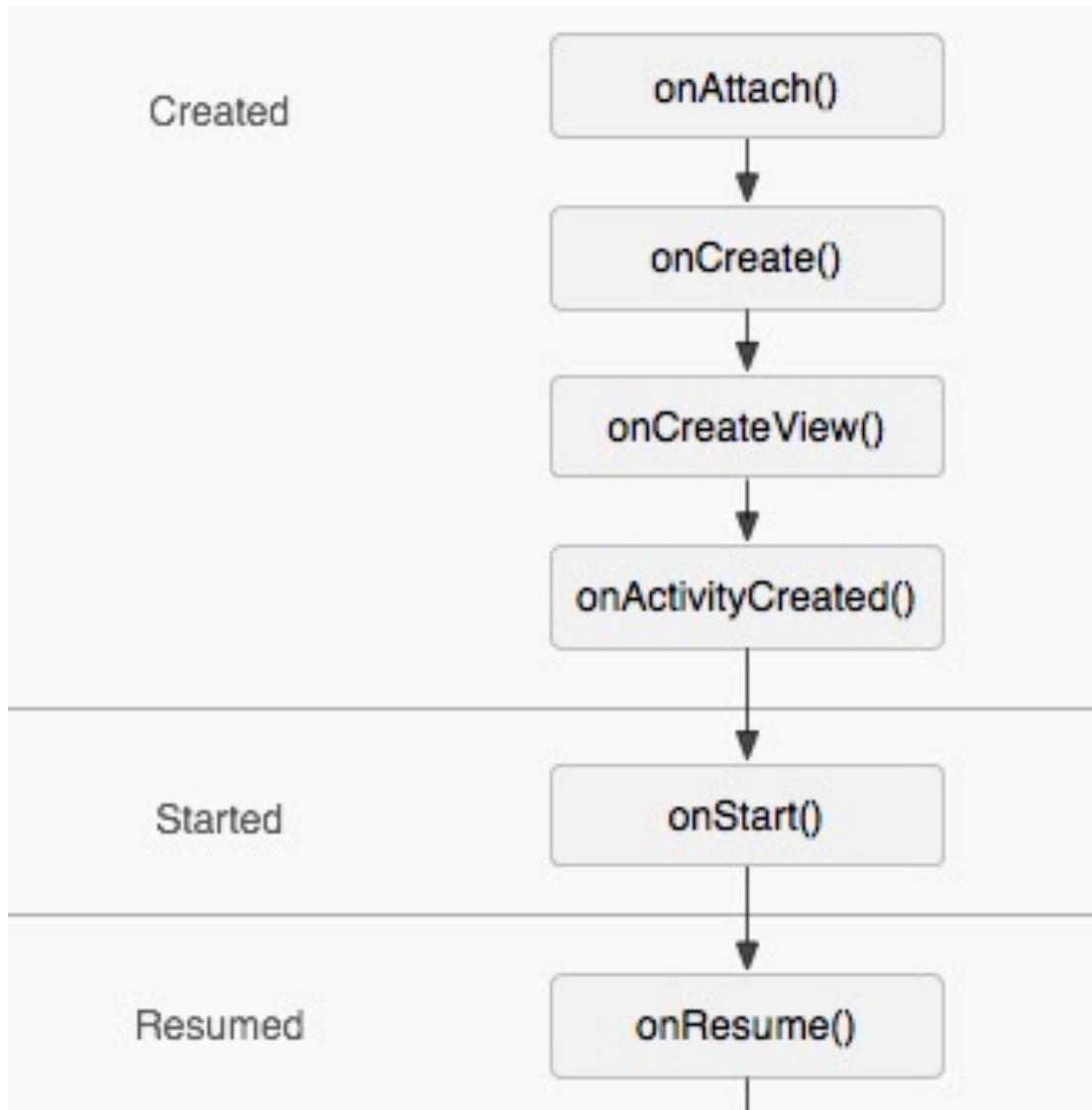
- Miben másabb mint az **Activity**?
  - > Kisebb granualitás, nem minden teljes képernyő egy fragment
  - > Az életciklusa nem minden egyszerű, pl. le lehet csatolni egy fragmentet úgy, hogy az activity előtérben marad.
- Miben másabb mint egy **Custom View**
  - > Összetett életciklus, mely az activity-t is figyelembe veszi
  - > Előny, de hátrány is lehet!

# Fragment és Activity

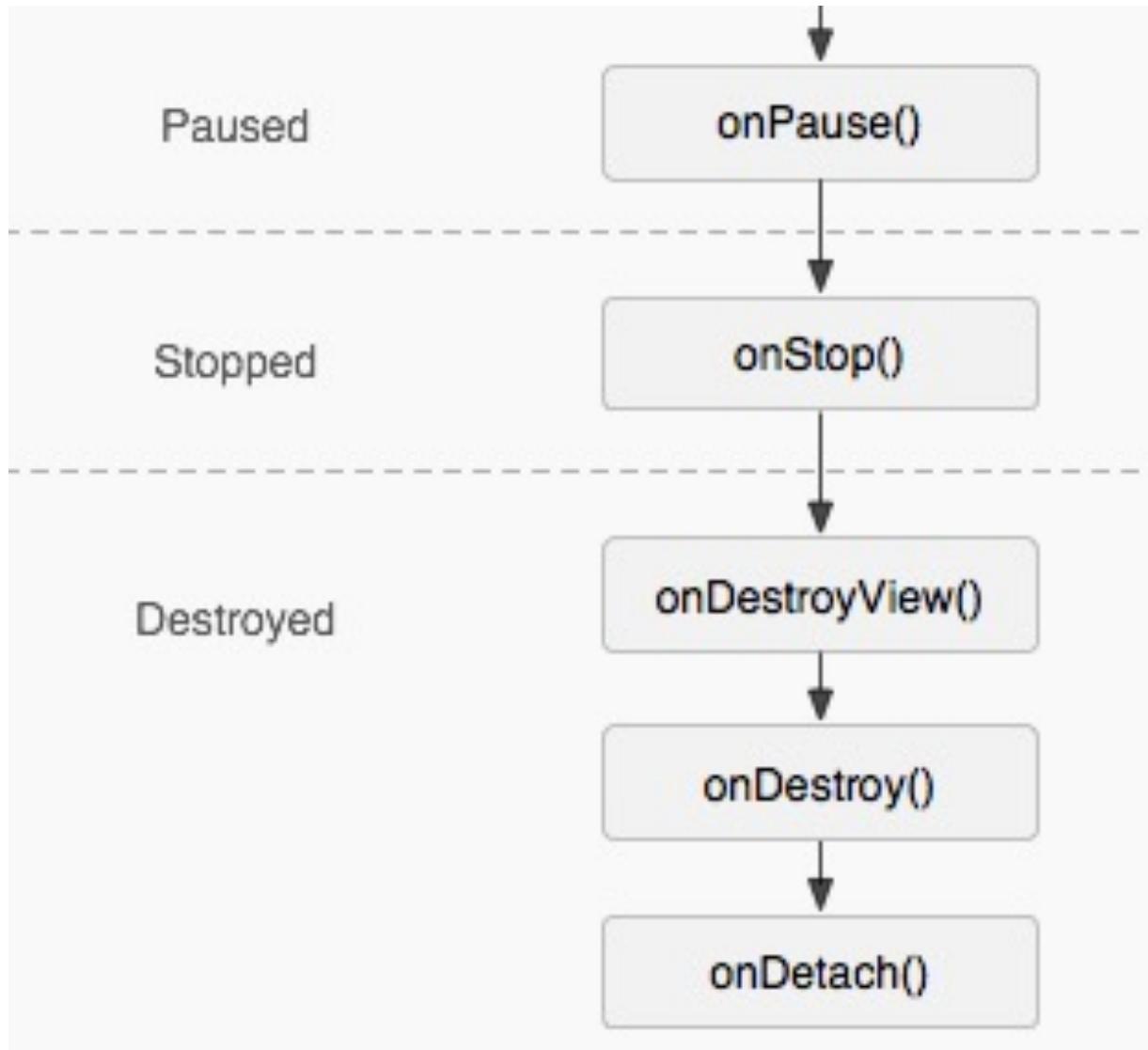
- Egy Fragment mindig egy Activity-hez csatoltan jelenik meg
- Az Activity életciklusa ráhatással van a Fragmentére

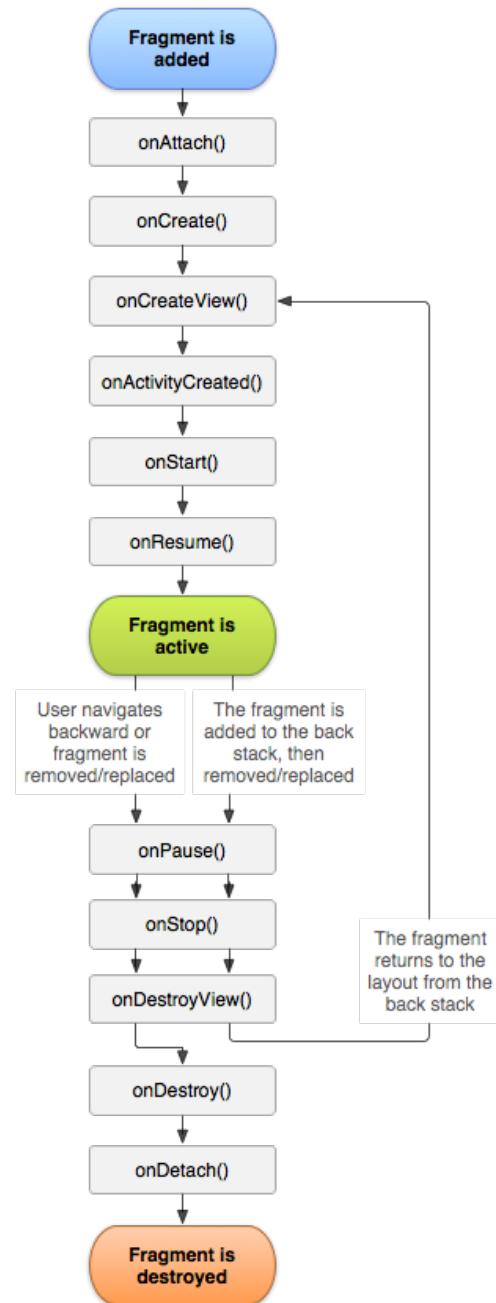


# Fragment életciklus I.



# Fragment életciklus II.





# Support Library

- Fragment API az Android API része **Android 3.0** óta
- **Support Library** – Fragmentek Android 3.0 előtt
  - > AndroidX csomagból érdemes már használni
- Ma már 3.0 alá nem is targetáluk, miért használjuk?
  - > Android verzionként eltér a beépített Fragment api (~~android.app.Fragment~~)
  - > A support környtárral minden Android verzióon ugyan azt kapjuk (**androidx...**)

# Nem alkalmazás komponens

- Mi hozzuk létre, nem a rendszer
- Nem kell feltüntetni a manifestben
- Nem intentekkel kommunikálunk
  - > Mezei függvényhívás az objektumon
  - > pl. Activity hívja a Fragment objetumon
  - > getActivity()/activity property – szülő activity

# UI Fragment készítése...

- A megjelenítendő View-hierarchiát az .onCreateView() metódusban kell visszaadni

```
class FragmentProfile : Fragment() {  
  
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?): View? {  
        val rootView = inflater.inflate(R.layout.fragment_profile, container, false)  
        return rootView  
    }  
  
}
```

# ... és csatolása

- Statikusan
  - > Az Activity-hez tartozó layout-ban beégetjük a Fragment-et, nem módosítható később
  - > <fragment .../> tag
- Dinamikusan
  - > Az Activity futás közben tölti be a megfelelő Fragment-eket, adott ViewGroup-okba
  - > Fragment-Tranzakciókkal módosítható

# Statikus csatolás példa

```
<fragment class="hu.bme.aut.fragment.MenuListFragment"
    android:tag="MenuListFragment"
    android:layout_width="0dip"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

# A FragmentManager

- A FragmentManager-el menedzselhetők a Fragment-ek
  - > Activity: `supportFragmentManager` property
  - > FragmentTransaction indítása
  - > Aktív Fragment-ek között keres
    - Tag alapján
    - ID alapján
  - > Fragment-stack-et menedzseli

# A FragmentManager

- Az activitynek a FragmentActivityból kell származnia – Neki van FragmentManagerje
- Activity.supportFragmentManager == fragment.fragmentManager
- Kezeli a fragmenteket, backstacek, állapotmentést ... stb.
- Fragmenten belül fragmentek, lehet: Fragment.childFragmentManager

# FragmentTransaction osztály I.

- Ezen keresztül módosíthatók az aktív Fragmentek
- A FragmentManager .beginTransaction()  
metódusával indítható
- Fontosabb műveletek:
  - > .add(...) / .remove(...) / .replace(...)
    - Fragment példányok le- és felcsatolása az adott Activity-re
  - > .commit()
    - Tranzakció végrehajtása

# FragmentTransaction osztály II.

- > `.show(...)` / `.hide(...)`
  - Fragment példány elrejtése / újra megjelenítése
- > `.setTransition(...)` / `.setCustomAnimations(...)`
  - A tranzakció végrehajtásakor lejátszandó animáció beállítása
- > `.addToBackStack(...)`
  - Rákerüljön-e a FragmentTransaction backstack-re a tranzakció?
- > `.commit()`
  - Tranzakció végrehajtása

# FragmentTransaction példa I.

- Fragment kicserélése:

```
val fragment=DetailsFragment.newInstance()
```

```
val ft = supportFragmentManager.beginTransaction()
ft.replace(R.id.fragmentContainer, fragment, DetailsFragment.TAG)
ft.commit()
```

# FragmentTransaction példa II.

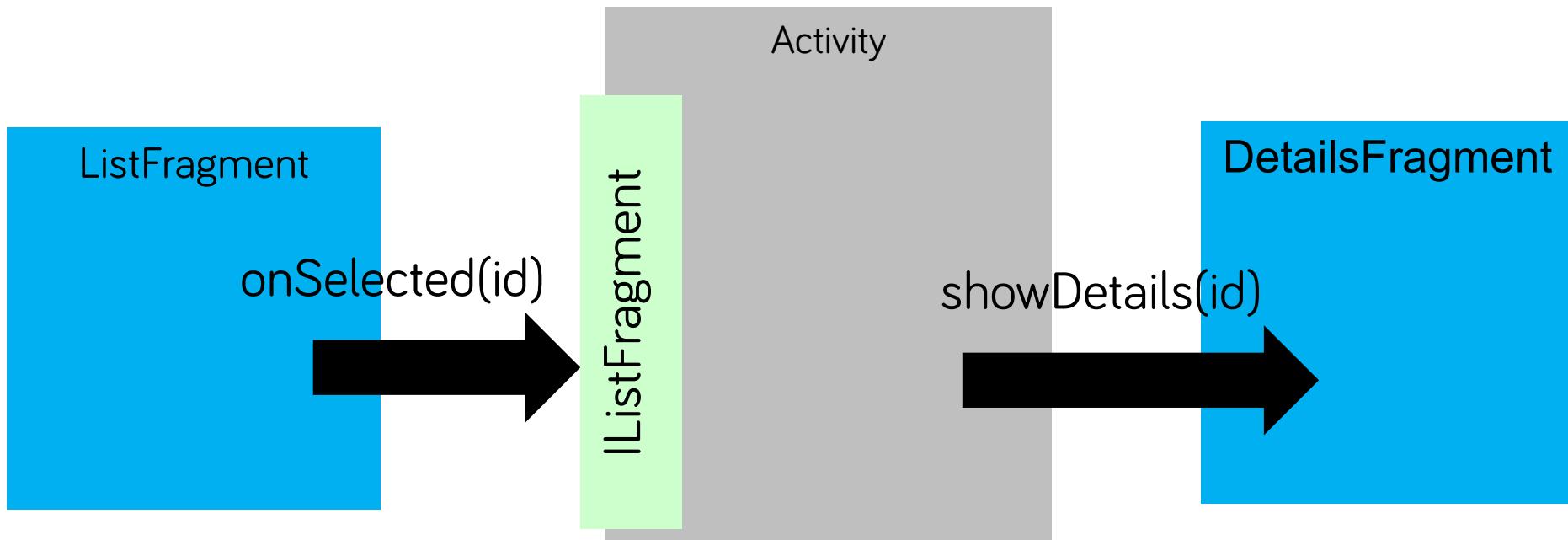
- Fragment hozzáadása, a tranzakciót a backstack-re téve:

```
val fragment=DetailsFragment.newInstance()
```

```
val ft = supportFragmentManager.beginTransaction()
ft.add(R.id.fragmentContainer, fragment, TAG)
ft.setCustomAnimations(R.anim.slide_in_top,R.anim.slide_out_bottom)
ft.addToBackStack(null)
ft.commit()
```

# Fragment kommunikáció I.

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



# Fragment kommunikáció II.

- Ha mégis szükség van a közvetlen Fragment-kommunikációra:
  - > `Fragment.targetFragment` propertyvel állítható/elérhető
- Az így létrejövő kapcsolat túléli a forgatásokat is

# Fragment paraméterek

- Szükség lehet paraméter átadás
  - > pl. Melyik cikk részleteit jelenítse meg a hírolvasó, stb..
- Első ötlet
  - > ~~Mi inicializáljuk – Konstruktor paraméter~~
  - > Nem csak mi inicializálhatjuk! – pl. Elforgatás
- Használjuk a Factory pattern-t
  - > `Fragment.arguments:Bundle` property
  - > Mentésre kerül a Fragment példánnyal

# Fragment paraméterek

- Egy Fragmentet a publikus, paraméter nélküli konstruktorával példányosítunk
- Ha paramétereket kell átadnunk:
  - > Példányosításkor Bundle-ben adjuk át őket
    - .setArguments(...) (Kotlinban arguments property)
  - > Inicializáláskor ezt a Bundle-t kérjük el
    - getArguments()
  - > A forgatást is túléli az Arguments Bundle

# DialogFragment I.

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

# DialogFragment II.

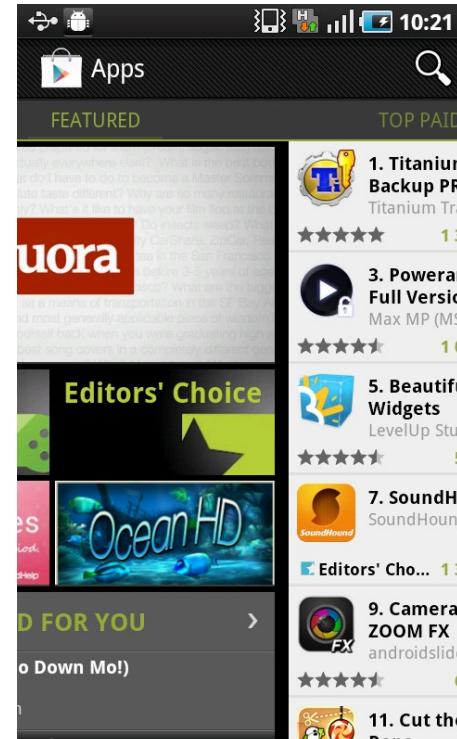
- .onCreateDialog()
  - > Ez a metódus is visszatérhet a megjelenítendő Dialog-gal
- .onCreateView()
  - > Ha nem használjuk az .onCreateDialog()-ot
  - > Tetszőleges megjeleníthető tartalom
- Egy DialogFragment egyben Fragment is!
  - > Ha kell, akár Activity-be ágyazottan is megjeleníthető

# Fragment-ek fej nélkül

- Ha nem adjuk hozzá a Fragmentet egy ViewGroup-hoz -> a háttérben dolgozik
- .setOnRetainInstanceState(true)
  - > Az .onRetainNonConfigurationInstance() helyett használható
  - > Forgatásnál csak lecsatolja a megsemmisülő Activity-ről a Fragmentet, majd rácsatolja az újra
    - A memóriában marad, nincs szükség állapotmentésre

# ViewPager

- ViewGroup, ahol az elemek közt swipe-al lehet mozogni
  - > Pl.: Google Play



# FragmentPagerAdapter I.

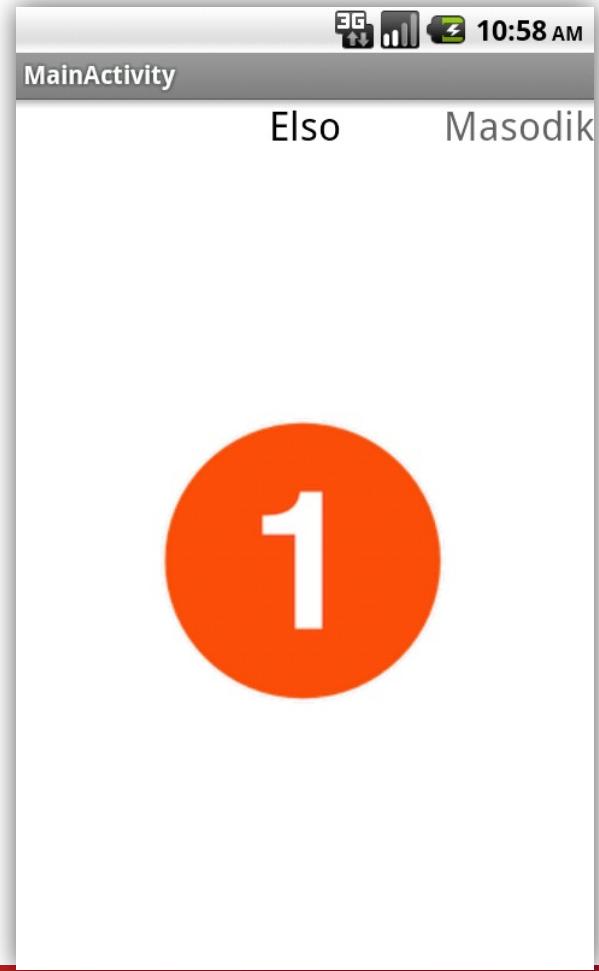
- Általában Fragment-eket adják a ViewPager oldalait
- Ekkor egy FragmentPagerAdapter példány szolgáltatja az egyes oldalakat
  - > Hasonló elven működik, mint a BaseAdapter
  - > Fragment getItem(int position):
    - Visszaadjuk a megfelelő Fragment példányt
  - > int getCount():
    - Visszaadjuk, hogy összesen hány oldalunk van

# FragmentPagerAdapter II.

- > String getTitle(int position):
  - Visszaadjuk az adott oldal címét
- Demo!

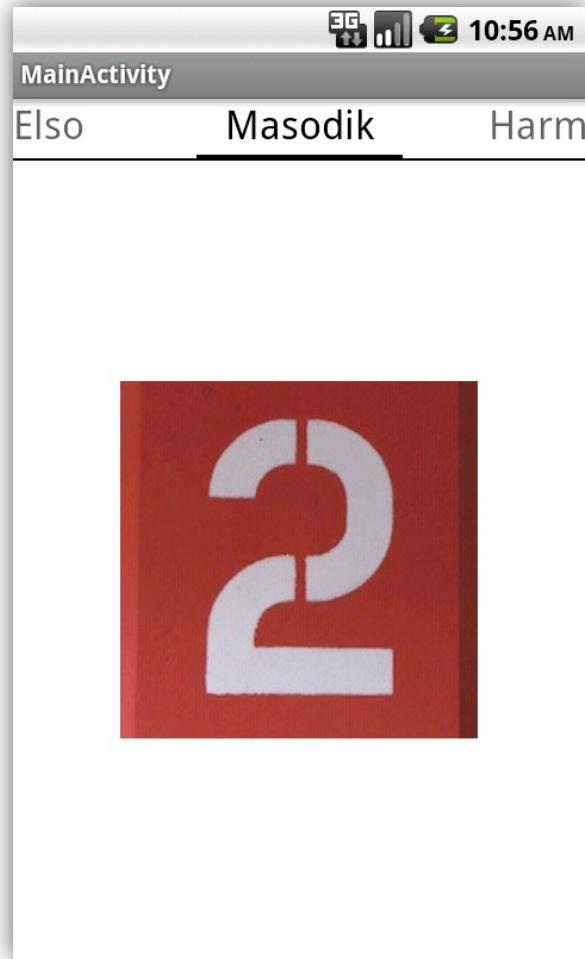
# PagerTitleStrip

- Widget, ami mutatja a ViewPager-ben szereplő oldalak címeit
  - > A Support lib. tartalmazza
  - > Nem interaktív



# PagerTabStrip

- Widget, ami mutatja a ViewPager-ben szereplő oldalak címeit
  - > A Support lib. tartalmazza
  - > Interaktív!



# ViewFlipper komponens

- View közti egyszerű váltás
- Több view található rajta, de mindig csak egyet jelenít meg
- XML-ben felsorolhatók a tartalmazott view-k
- Dinamikusan vezérelhető
- Fragmenteket is kezel
- Ha sok view-t teszünk rá, lassú lehet, mivel mindenkomponens előre előkészíti!

# Próbáljuk ki!

FragmentDemo projekt

# Mi nem igaz a Fragment-ekre?

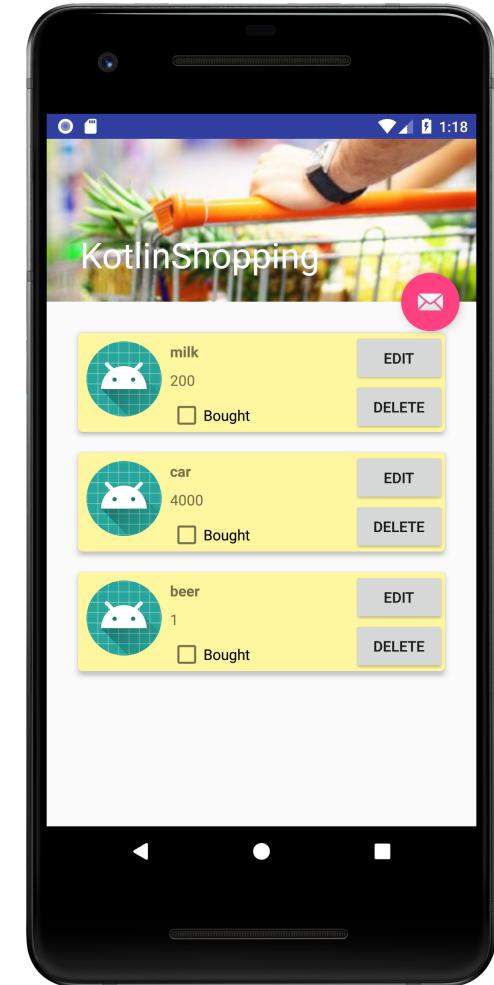
- A. Önálló életciklussal rendelkeznek.
- B. Kötelező felhasználói felületet tartalmazniuk.
- C. Dinamikusan és statikusan is csatolhatók.
- D. A tableték felhasználói felületének kialakításakor különösen hasznosak.

# Listák kezelése - RecyclerView

RecyclerView, Gesztusok, ListAdapter

# RecyclerView

- Listák hatékony kezelése
- Gyors scrollozás
- Általános érintés gesztusok támogatása (swipe, move, stb.)
- *ViewHolder* minta a gyors működés érdekében
- Hatékony elem újrafelhasználás
- *Flexibilis*



# RecyclerView.Adapter<ViewHolder> 1/3

- Inicializálás, konstruktur

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

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

- Egy sor nézetének beállítása: onCreateViewHolder

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

# ViewHolder implementáció

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

# RecyclerView.Adapter<ViewHolder> 2/3

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

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

# RecyclerView.Adapter<ViewHolder> 3/3

- Elemek száma, hozzáadás, törlés

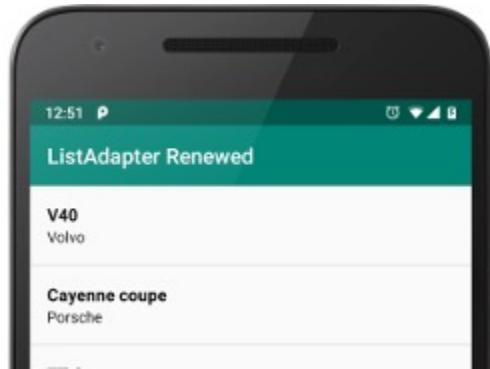
```
override fun getItemCount() = items.size

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

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

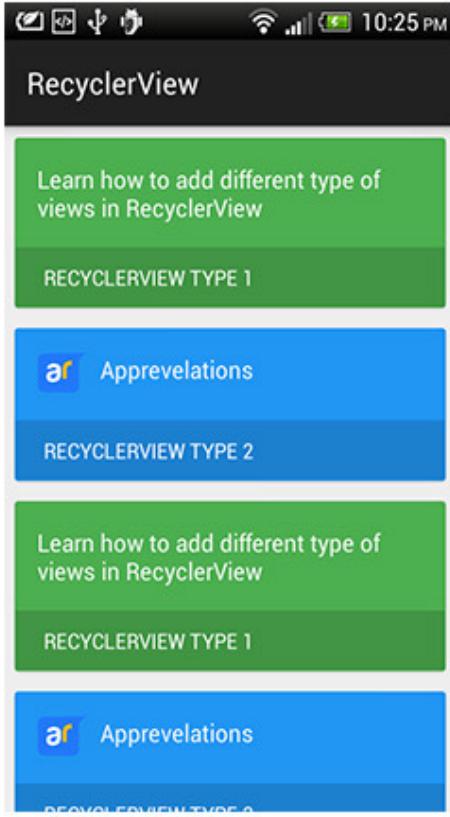
# Újfajta RecyclerView adapter - ListAdapter

- *LiveData* kompatibilitás (adatbázis kezeléskor elemezzük)
- Nem kell saját listát kezelní benne, megoldja az implementációt belül
- *DiffUtil* használata a listaelemek összehasonlítására és az optimális újrarendezésre
- `getItemCount()` használatára már nincs szükség



- További részletek:
  - > <https://medium.com/@hackathon.blog.42/listadapter-renewed-9b5b496198e2>

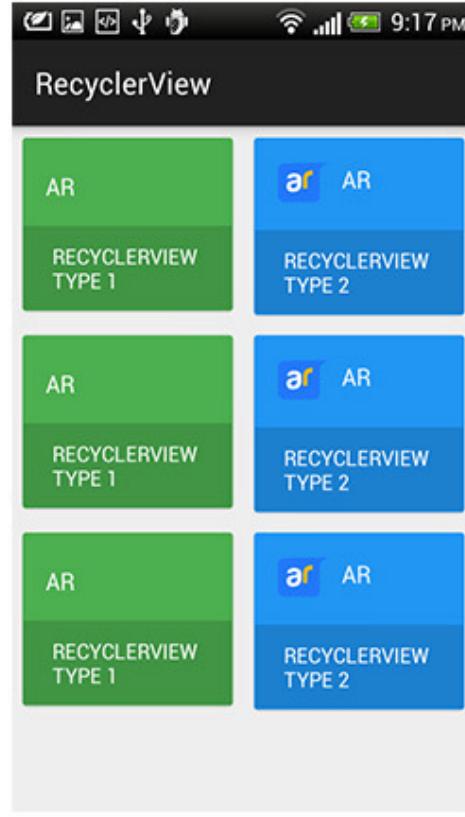
# RecyclerView LayoutManager-ek



Linear Layout View



Staggered Grid View



Grid View

# Különböző elem megjelenítés a RecyclerView-ban

- Elemek/sorok típusa megadható pozíció alapján a *getItemViewType(...)* felüldefiniálásával
- *viewType* parancs jelzi a megfelelő függvényekben a sor/elem típusát, amely alapján a megjelenítés szabályozható
- *ViewHolder* ismeri a *viewType*-ját

```
// determine which layout to use for the row
@Override
public int getItemViewType(int position) {
    Item item = itemList.get(position);
    if (item.getType() == Item.ItemType.ONE_ITEM) {
        return TYPE_ONE;
    } else if (item.getType() == Item.ItemType.TWO_ITEM) {
        return TYPE_TWO;
    } else {
        return -1;
    }
}

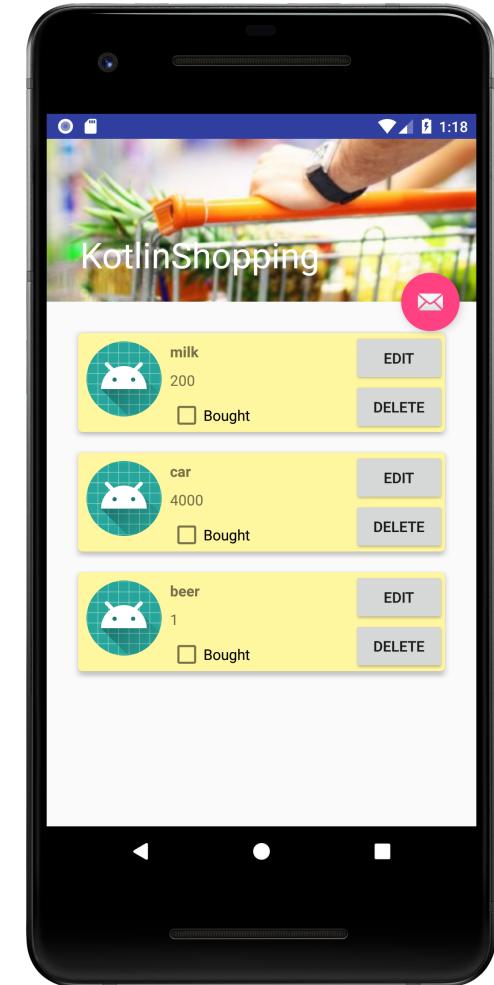
// specify the row layout file and click for each row
@Override
public RecyclerView.ViewHolder onCreateViewHolder(
    ViewGroup parent, int viewType) {
    if (viewType == TYPE_ONE) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(
            R.layout.list_item_type1, parent, false);
        return new ViewHolderOne(view);
    } else if (viewType == TYPE_TWO) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(
            R.layout.list_item_type2, parent, false);
        return new ViewHolderTwo(view);
    } else {
        throw new RuntimeException(
            "The type has to be ONE or TWO");
    }
}
```

# Lista készítés – fő lépések

1. Data class
2. Egy sor layout-ja
3. RecyclerView – lista hol legyen
4. Adapter – megmondja hogy mi legyen a RecyclerView-ba

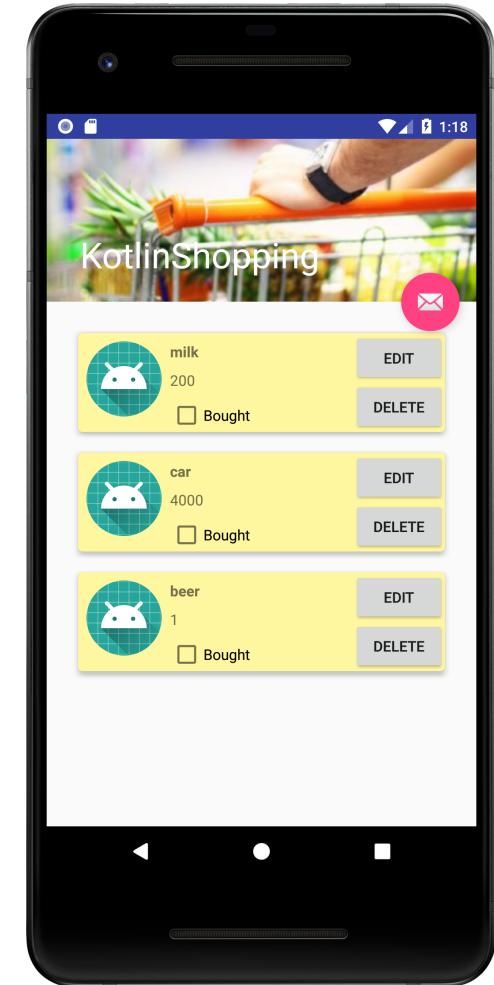
# Bevásárló lista példa - RecyclerView

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



# Bevásárló lista példa - RecyclerView

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



# Swipe és drag&drop gesztusok

- Távolítsuk el az elemeket swipe hatására
- Tegyük lehetővé az elemek átrendezését drag&drop-pal
- *RecyclerView* támogatás:
  - `ItemTouchHelper.Callback`

# ItemTouchHelper.Callback 1/2

- *isLongPressDragEnabled():*
  - > True visszatérés ha a drag&drop támogatott
- *isItemViewSwipeEnabled():*
  - > True visszatérésé ha a swipe támogatott
- *onMove(...):*
  - > Elem mozgatás esetén hívódik meg
- *onSwipe(...):*
  - > Swipe esetén hívódik meg

# ItemTouchHelper.Callback 2/2

- Drag és swipe irányok beállítása:

```
override fun getMovementFlags(recyclerView: RecyclerView,  
    viewHolder: RecyclerView.ViewHolder): Int {  
    val dragFlags = ItemTouchHelper.UP or ItemTouchHelper.DOWN  
    val swipeFlags = ItemTouchHelper.START or ItemTouchHelper.END  
    return ItemTouchHelper.Callback.makeMovementFlags(dragFlags, swipeFlags)  
}
```

# További UI lehetőségek

# Validáció támogatása

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



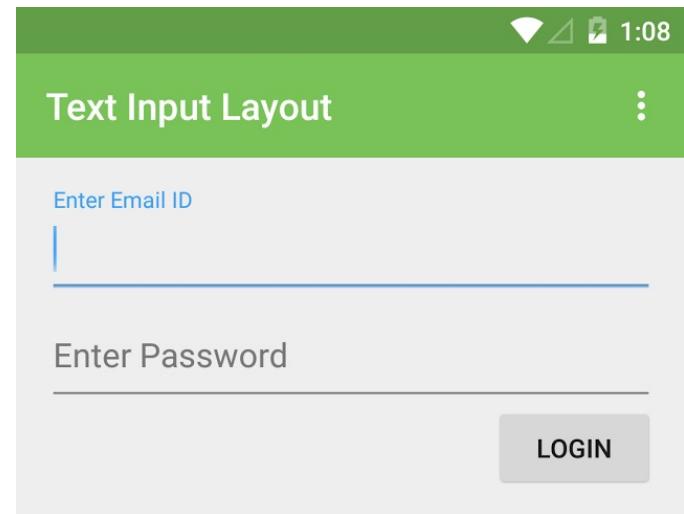
# TextInputLayout

- A *Material Design Library* része
- Használat:

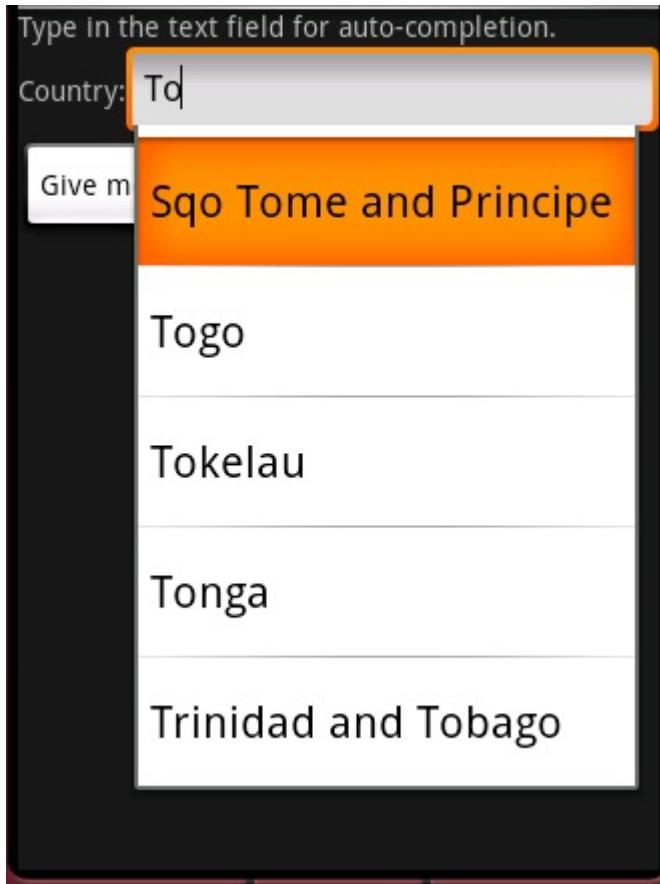
```
<android.support.design.widget.TextInputLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">
```

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Enter Email ID"  
    android:id="@+id/etName" />
```

```
</android.support.design.widget.TextInputLayout>
```



# Tipikusan Google-s: Autocomplete 1/3



# Tipikusan Google-s: Autocomplete 2/3

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <AutoCompleteTextView
        android:id="@+id/autoCompleteTextView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="" >

        <requestFocus />

    </AutoCompleteTextView>

</LinearLayout>
```

# Tipikusan Google-s: Autocomplete 3/3

```
var cityNames = arrayOf("Budapest", "Bukarest", "Krakkó", "Bécs")  
  
override fun onCreate(savedInstanceState: Bundle) {  
  
    super.onCreate(savedInstanceState)  
  
    setContentView(R.layout.main)  
  
    var cityAdapter: ArrayAdapter<String> =  
        ArrayAdapter<String>(this,  
            android.R.layout.  
                simple_dropdown_item_1line, cityNames)  
  
    autoCompleteTextView1.setAdapter(cityAdapter)  
}
```

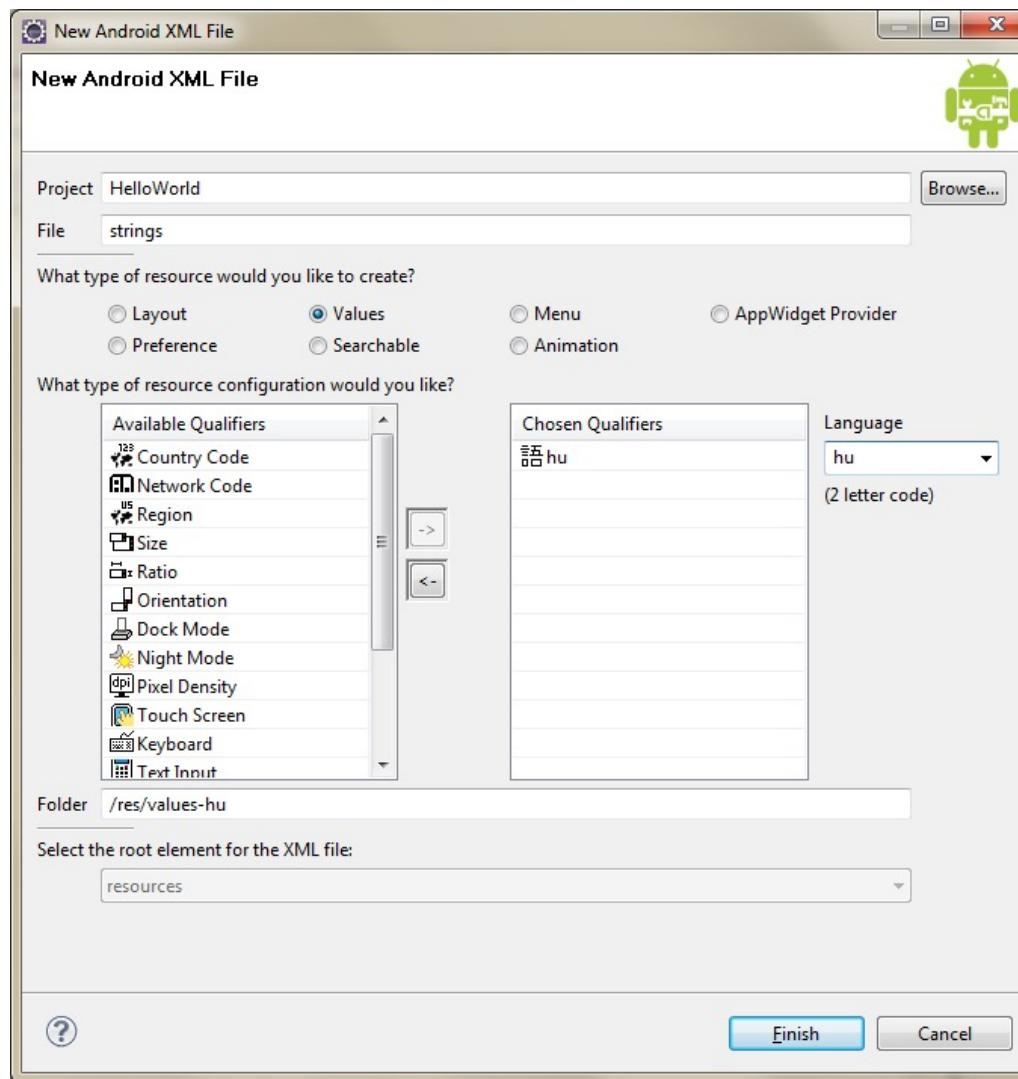
# Szöveges erőforrások

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

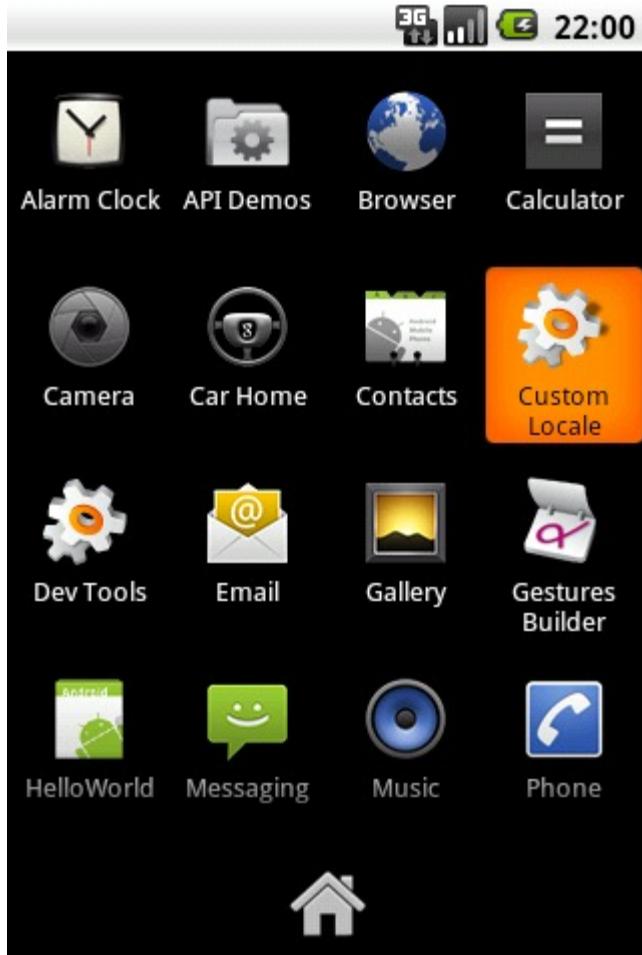
# Lokalizáció

- 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



# Lokalizáció tesztelése Emulátoron



# Egyedi nézetek

- View leszármazott
- Beépített nézetek és *LayoutGroup*-ok is felüldefiniálhatók, pl. saját nézet *RelativeLayout*-ból leszármaztatva
- <merge> XML elem
- XML-ek egymásba ágyazhatósága: <include>

# Egyedi felületi nézet

- Teljesen egyedi felületi elemek definiálása
- Meglévő felületi elemek kiegészítése
- Érintés események kezelése
- Dinamikus rajzolás
  - > Színek, rajzolási stílus
  - > Gyakori alakzatok: vonal, négyzet, kör stb.
  - > Szöveg rajzolása
  - > Képek megjelenítése
- Megjelenítési mérethez való igazodás
- XML-ből is használható!

# CustomView feladatai

- Törekedjünk az Android szabványok betartására
- Támogassuk a szükséges attribútumok XML-ben való megadását is
- Eseménykezelés, paraméterekhez való hozzáférés biztosítása
- Képernyőményet és eszköz függetlenség biztosítása

# CustomView példa

- Forráskód:

```
class MyView(context: Context?, attrs: AttributeSet?) :  
    View(context, attrs)
```

- XML:

```
<hu.bme.aut.amorg.examples.customview.MyView  
    android:id="@+id/myView"  
    android:layout_width="fill_parent"  
    android:layout_height="100dp"/>
```

# XML tulajdonságok megadása

- values/attrs.xml:

```
<resources>  
    <declare-styleable name="MyView">  
        <attr name="myBgColor" format="color" />  
    </declare-styleable>  
</resources>
```

- Namespace jelölés (pl. gyökér layout elemben):

```
xmlns:custom="http://schemas.android.com/apk/res/hu.bme  
.aut.amorg.examples.customview"
```

Paraméter beállítása:

```
<hu.bme.aut.amorg.examples.customview.MyView  
    ...  
    custom:myBgColor="#ff00ff00" />
```

# XML tulajdonságok lekérdezése

```
class MyView(ctx: Context, attrs: AttributeSet) : View(ctx, attrs) {  
    private val bgPaint: Paint = Paint()  
  
    init {  
        bgPaint.setStyle(Paint.Style.FILL)  
        val a = ctx.getTheme().obtainStyledAttributes(  
            attrs, R.styleable.MyView, 0, 0)  
        try {  
            bgPaint.setColor(a.getColor(  
                R.styleable.MyView_myBgColor,  
                Color.RED))  
        } finally {  
            a.recycle()  
        }  
    }  
  
    override fun onDraw(canvas: Canvas) {  
        super.onDraw(canvas)  
        canvas.drawRect(30F, 30F, width.toFloat() - 30,  
            height.toFloat() - 30, bgPaint)  
    }  
}
```

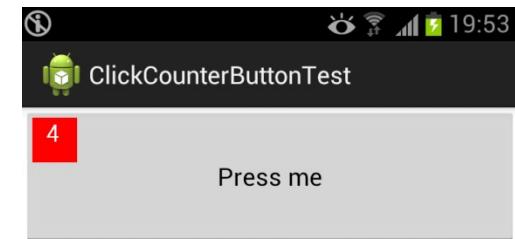
# Méret beállítása

- XML-ben megadható a méret
- Dinamikusan is módosítható
- Például négyzetes megjelenítés szélességhez igazítva:

```
override fun onMeasure(widthMeasureSpec: Int,
                      heightMeasureSpec: Int) {
    val w = View.MeasureSpec.getSize(widthMeasureSpec)
    val h = View.MeasureSpec.getSize(heightMeasureSpec)
    val d = if (w == 0) h else if (h == 0) w else if (w < h) w else h
    setMeasuredDimension(d, d)
}
```

# Gyakoroljunk!

- Készítsünk egy alkalmazást, amely egy saját gomb vezérlőt jelenít meg!
- Valósítsuk meg, hogy a gombra kattintáskor egy számláló a gomb bal felső sarkában mutassa hányszor kattintottunk a gombra!
- A számláló mérete és láthatósága legyen XML paraméterből állítható!



# Komponens közi kommunikáció

# 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

# Bevezetés

## Mi a helyzet Androidon?

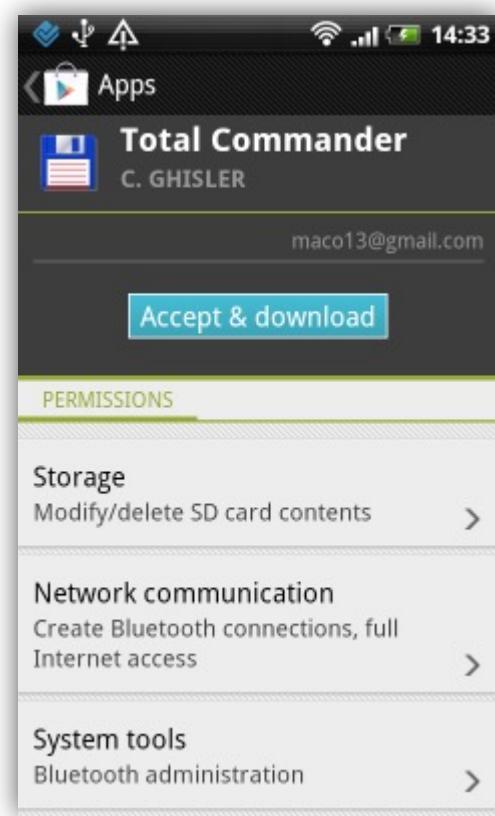
- Alkalmazások külön Dalvik VM példányokban (ez is sandbox)



# Bevezetés

## Mi a helyzet Androidon?

- Alkalmazások külön Dalvik VM példányokban (ez is sandbox)
- Kritikus műveletekhez engedély szükséges



# Bevezetés

Mi a helyzet Androidon?

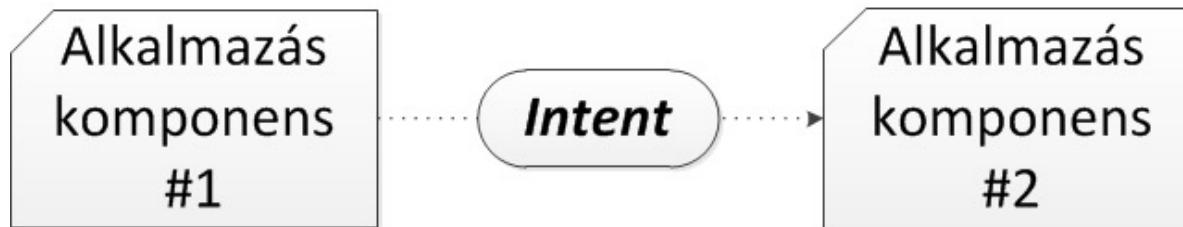
- Alkalmazások külön Dalvik VM példányokban (ez is sandbox)
- Kritikus műveletekhez engedély szükséges
- Alkalmazás = komponensek halmaza

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*

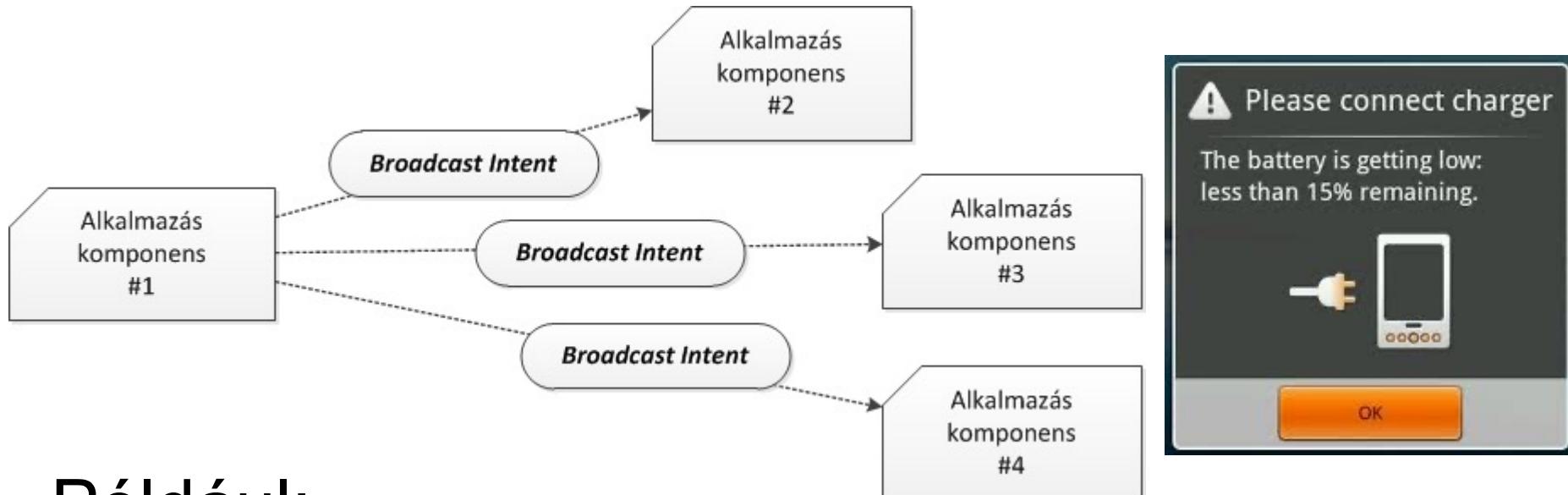


Például:

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

# Kommunikáció formái 2/3

- Egy komponensből mindenki másnak: *Broadcast Intent*



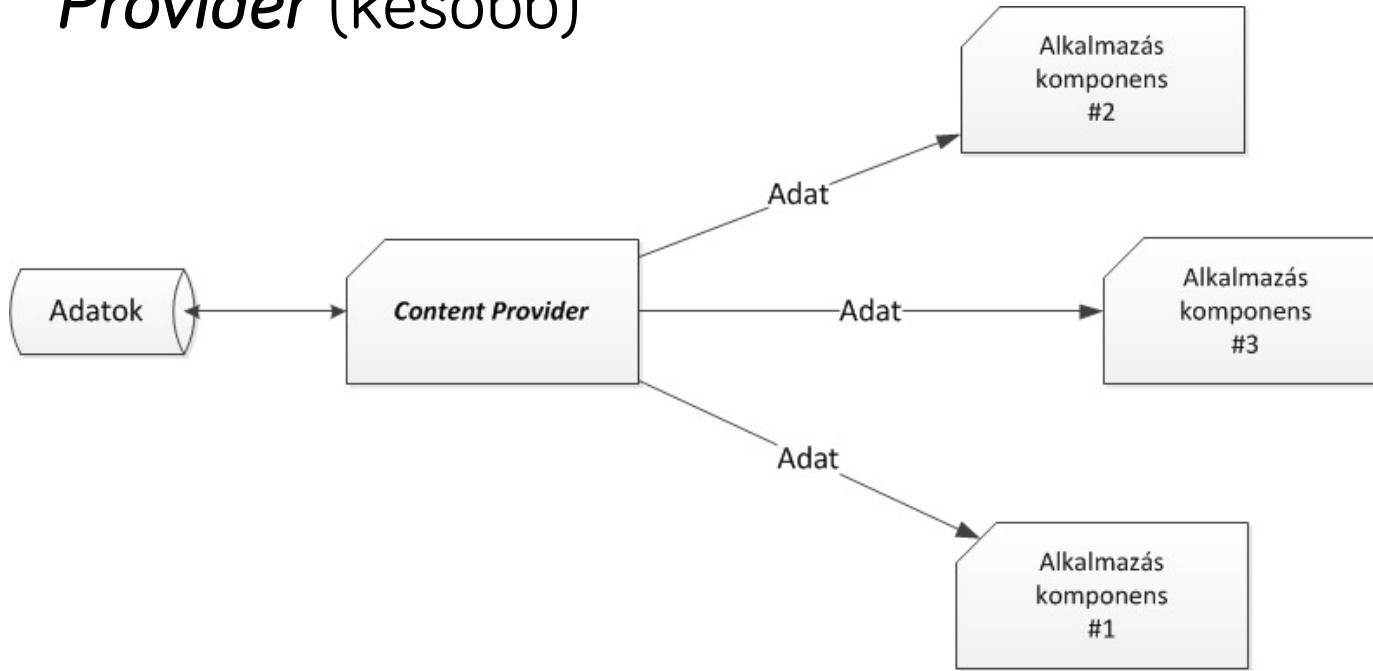
Például:

- „Akkufeszültség alacsony” rendszerüzenet

- val intentStart: Intent = Intent(this,  
„hu.bme.aut.demoapp.DetailActivity”)
- startActivity(intentStart)

# Kommunikáció formái 3/3

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

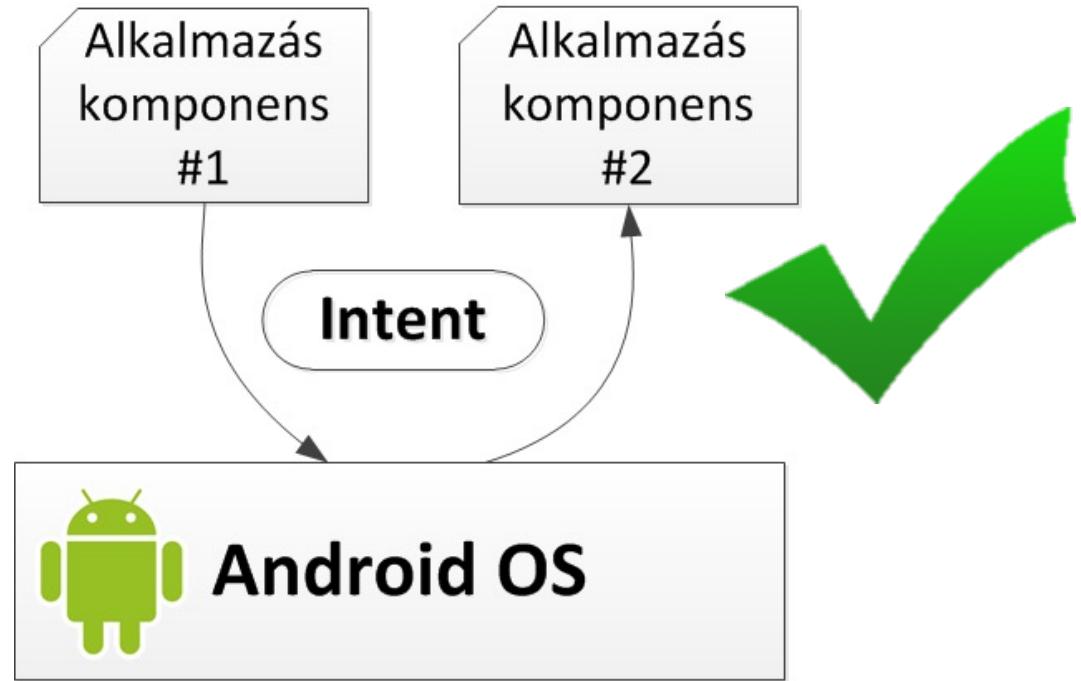
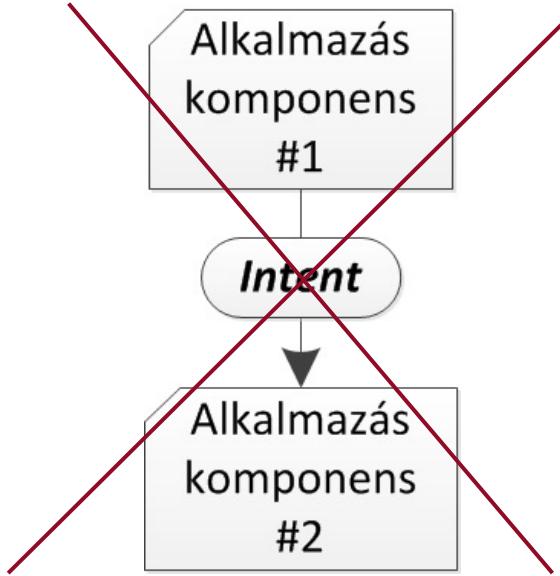


Például:

- Névjegyzék elérése saját alkalmazásból

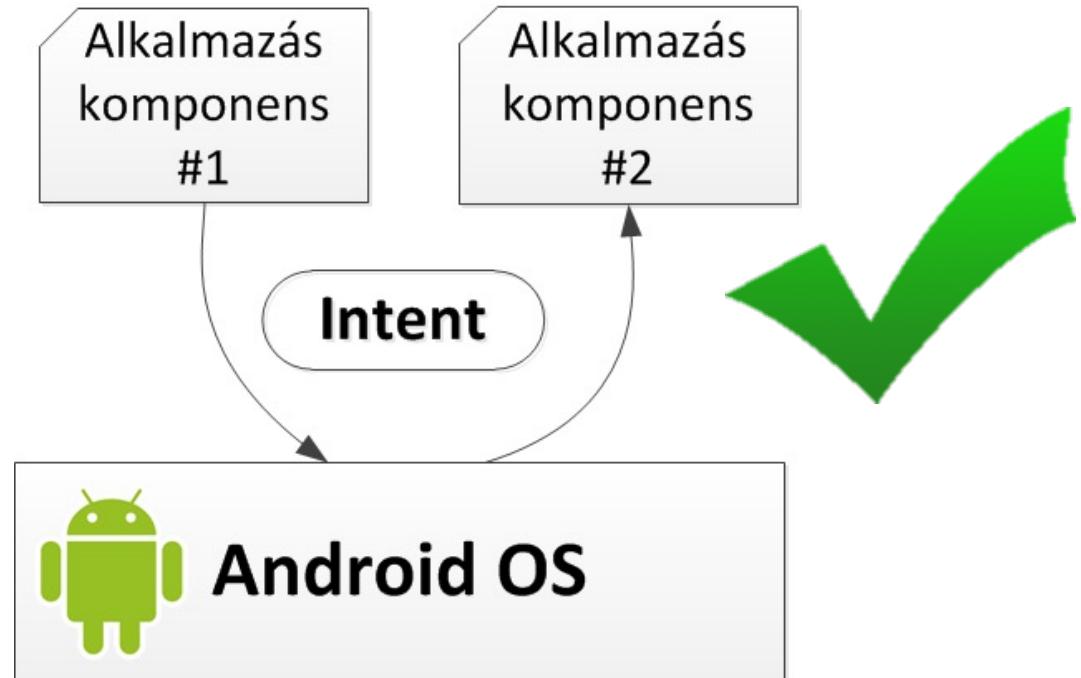
# Intent átadása

- Mindig az Android runtime-on keresztül!



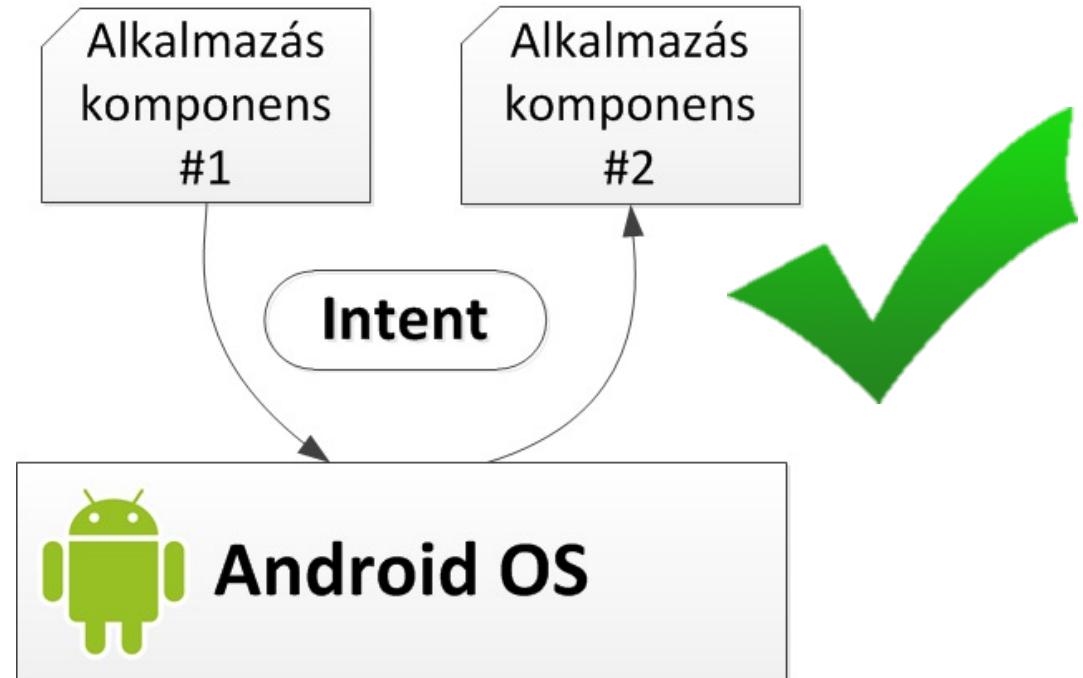
# Intent átadása

- Mindig az Android runtime-on keresztül!



# Intent átadása

- Mindig az Android runtime-on keresztül!



# Intent

# Intent (szándék)

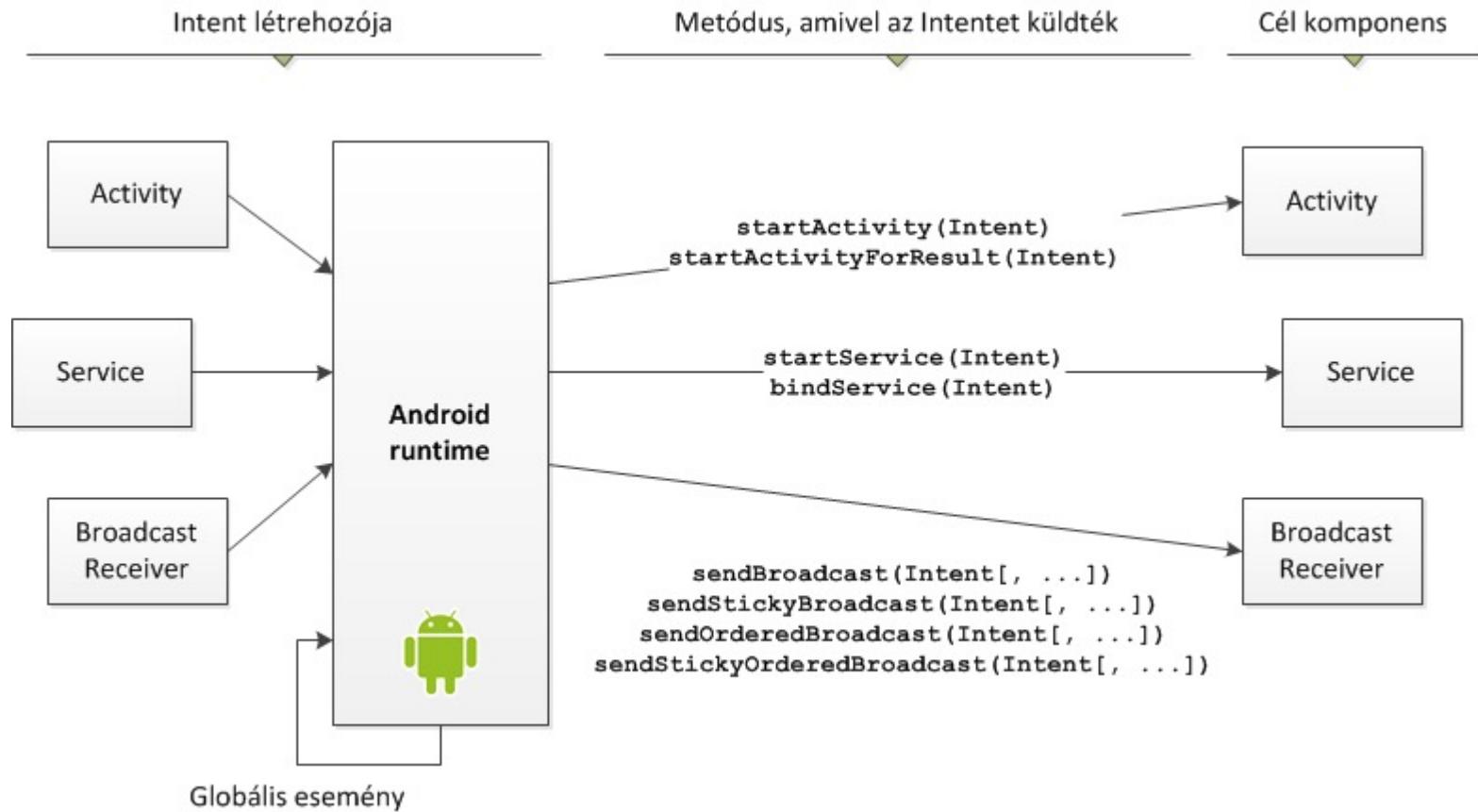
- Passzív adatstruktúra (~struct)
- Késői (futás idejű) kötést valósít meg alkalmazás komponensek között
  - > Komponensek: Activity, Service, Broadcast Receiver
- Az elvárt vagy bekövetkezett esemény absztrakt leírása
  - > Elvárt esemény leírása, ha az Intent hatására történik valami
    - Activity,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

# Intent kézbesítés

- Az Intent objektum tehát háromféle komponensnél köthet ki:
  - > **startActivity [ForResult] (Intent)** : Activity indítása vagy folytatása
  - > **startService (Intent)** ,  
**bindService (Intent)** : Service indítása vagy futásidéjű kötés megvalósítása
  - > **send [Ordered | Sticky] Broadcast (Intent)** : minden érdekelt BroadcastReceiver(BR)-hez eljut

# Intent kézbesítés

- Az Android minden esetben megkeresi a megfelelő címzettet, és példányosítja ha szükséges
- Egy Intent objektum mindenkor csak egyfél komponenshez jut el
  - > `startActivity()`-vel átadott Intent-et nem kapják meg a Service-ek és a BR-ek
  - > Rendszerszintű esemény értesítését (Broadcast Intent) csak BR-ek kapják
  - > `startService(Intent)` csak Service-hez kerülhet



# Intent részei

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

# Felépítése



# Intent felépítése

- Intent = Adatcsomag
- Információkat tartalmaz...
  - > ...az operációs rendszernek, ami eldönti hogy milyen komponenst és hogyan kell indítani/folytatni
    - Komponens neve
    - Category
    - Flags
  - > ...és annak a komponensnek ahova kézbesítődik
    - Action
    - Data
    - Extras

# Intent használata

- Leggyakoribb használata: új Activity indítása
- Két módon lehetséges
  - > *Explicit* – ismerjük a hívandó Activity osztálynevét (tipikusan alkalmazáson belül)
  - > *Implicit* – nem tudjuk / nem akarjuk eldöntení, hogy melyik Activity szükséges, csak azt hogy mire akarjuk használni (tipikusan másik alkalmazást használunk)
    - Pl. kép megjelenítése, PDF megnyitása, videó lejátszása, névjegy kiválasztása, stb...

# Explicit Intent

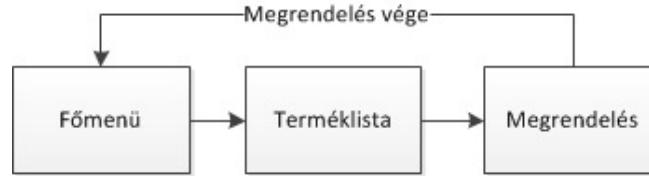
- Mindkét esetben a `startActivity()` függvényt használjuk:
  - > `startActivity(Intent)`
- **Explicit hívás:** az Intent-ben kitöljük a címzett komponens nevét (konstruktorból vagy setterrel)

```
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) {...}
```

- Paraméterei:
  - > **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:

```
public static int RESULT_ORDER_SUCCESS  
= 2;  
public Miért static? RESULT_LOGIN_OK = 3;  
public static int RESULT_LOGIN_FAIL =  
4;
```

# onActivityResult

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

- > **extras**: ha nem elég a resultCode, akkor egy Intent objektumot is visszaadhatunk, amiben adatot helyezünk el
  - Intent Extras: tetszőleges kulcs-érték párok
    - Egy Intent objektumban akármennyi elhelyezhető
    - Kulcs: String, aminek prefixe a package name
    - Érték: akármilyen beépített típus, de lehet saját is, ha megvalósítja a Serializable vagy a Parcelable interfészt

# 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()
```

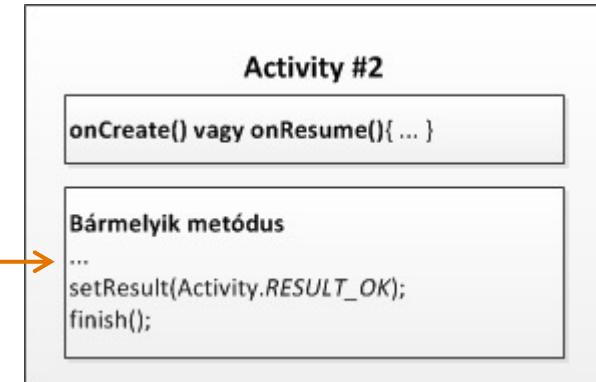
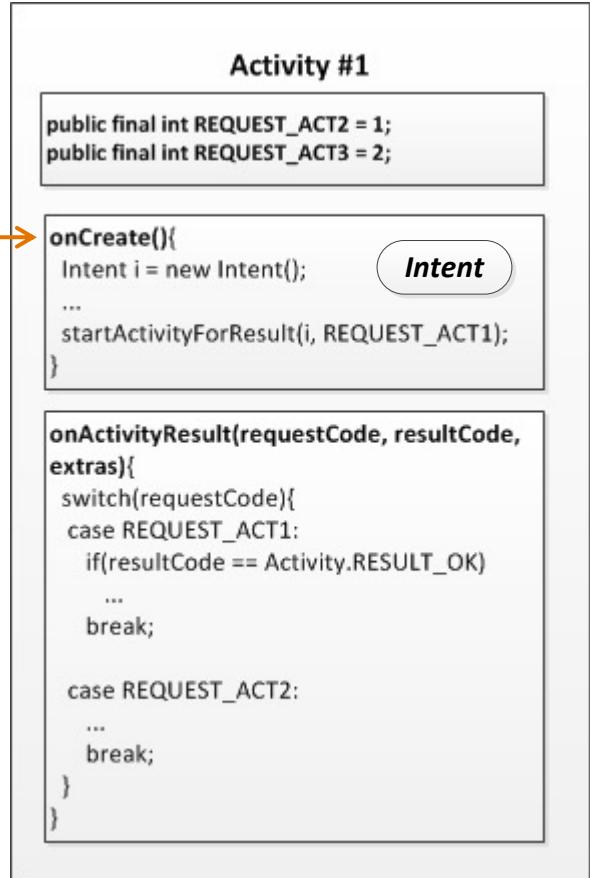


# Intent Extras

- Lekérdezése:

```
intent.get[Típusnév]Extra(name[, defaultValue])
```

# Activity visszatérése



# Új Activity Result API

- Activity indítás:

```
private val startForResult = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result: ActivityResult ->
    if (result.resultCode == Activity.RESULT_OK) {
        val intent = result.data
        Toast.makeText(this, "Result: ${intent?.getStringExtra("KEY_ANS")}", Toast.LENGTH_LONG).show()
    } else if (result.resultCode == Activity.RESULT_CANCELED) {
        Toast.makeText(this, "CANCELLED", Toast.LENGTH_LONG).show()
    }
}
```

```
binding.btnStart.setOnClickListener {
    var intentStart = Intent(this, DetailsActivity::class.java)
    intentStart.putExtra("KEY_DATA", binding.etData.text.toString())
    startForResult.launch(intentStart)
}
```

- További részletek:

> <https://developer.android.com/training/basics/intents/result>

# 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)
```

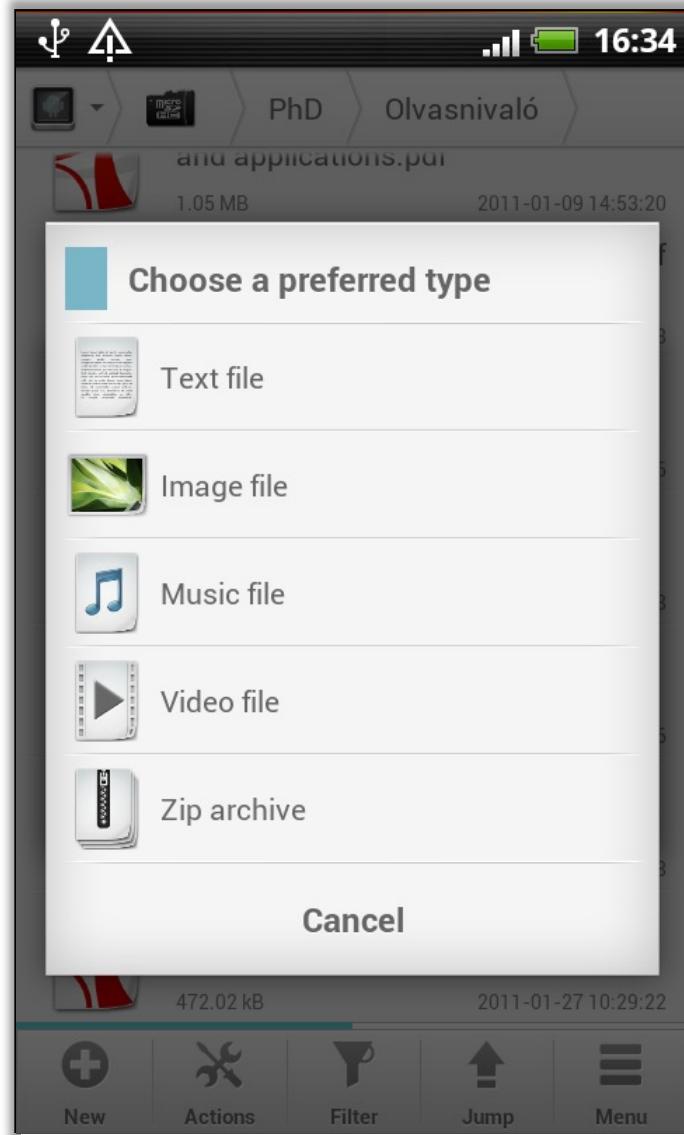
Akció

Adat (URI)

# Implicit Intent - Példa

- Pdf megnyitása

```
var sdCard: File = Environment.getExternalStorageDirectory()  
var pdf:File = File(sdCard+"/docs", "sample.pdf")  
var i: Intent = Intent(Intent.ACTION_VIEW)  
i.setDataAndType(Uri.fromFile(pdf), "application/pdf")  
startActivity(i)
```



# Implicit Intent – Több célpont

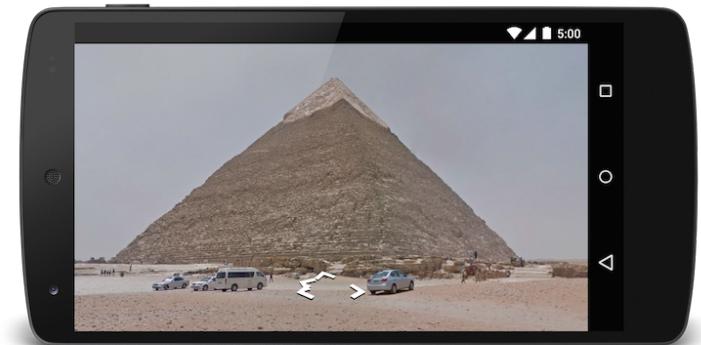
- *ActivityNotFoundException*, ha nem talál megfelelőt
- Amennyiben több alkalmazás is képes a kért akcióra: „*Complete action using*” dialógusablak
  - Ha nincs default megadva, akkor a felhasználó választ
  - Egyébként indul az alapértelmezett alkalmazás
  - (új app telepítése után újra rákérdez az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



# Implicit Intent - Térkép

- <https://developers.google.com/maps/documentation/urls/android-intents>

```
// Opens Street View between two Pyramids in Giza. The values passed to the  
// cbp parameter will angle the camera slightly up, and towards the east.  
val gmmlIntentUri = Uri.parse("google.streetview:cbll=29.9774614,31.1329645&cbp=0,30,0,0,-15")  
val mapIntent = Intent(Intent.ACTION_VIEW, gmmlIntentUri)  
mapIntent.setPackage("com.google.android.apps.maps")  
startActivity(mapIntent)
```



# Implicit Intent – Akciók

- Nem csak megnyitni és kiválasztani lehet
- Rengeteg „beépített” akció, a fontosabbak:
  - > ACTION\_EDIT: szerkesztés, pl. névjegy, txt, kép
  - > ACTION\_DELETE: adat törlése
  - > ACTION\_WEB\_SEARCH: Google keresést indít a kapott szövegre
  - > ACTION\_CALL: kapott telefonszám felhívása
  - > ACTION\_PICK: kiválasztás listából, pl. névjegy, fénykép, zene, videó
  - > ACTION\_VIEW: megnyitás a megfelelő alkalmazással, pl: telepítő fájl, pdf, txt, fénykép

# Implicit Intent – Beépített akciók

A majdnem teljes lista (új verzióval általában bővül!):

<code>ACTION_MAIN</code>	<code>ACTION_SEND</code>	<code>ACTION_WEB_SEARCH</code>
<code>ACTION_VIEW</code>	<code>ACTION_SENDTO</code>	<code>ACTION_FACTORY_TEST</code>
<code>ACTION_ATTACH_DATA</code>	<code>ACTION_ANSWER</code>	
<code>ACTION_EDIT</code>	<code>ACTION_INSERT</code>	
<code>ACTION_PICK</code>	<code>ACTION_DELETE</code>	
<code>ACTION_CHOOSER</code>	<code>ACTION_RUN</code>	
<code>ACTION_GET_CONTENT</code>	<code>ACTION_SYNC</code>	
<code>ACTION_DIAL</code>	<code>ACTION_PICK_ACTIVITY</code>	
<code>ACTION_CALL</code>	<code>ACTION_SEARCH</code>	

# Implicit Intent

Az Intent nem csak arra jó, hogy a következő képernyőre ugorjunk

Intent általánosan: „*Kérés egy akció elvégzésére valamelyen adaton*”

- startActivity esetén nem fogalmazunk meg sem akciót, sem adatot, csak a cél komponenst.
- „*Az elvárt vagy bekövetkezett esemény absztrakt leírása*”

# Implicit Intent

- Ha az Intentben nem kötelező megadni a célpontot (Activity-t), akkor honnan tudja az Android hogy mit indítson?
- Az Intent objektum információkat tartalmaz arról, hogy mit kell csinálni
  - > *Action, Data, Category* mezők
- Az Android megkeresi a legjobb célpontot
- Neve: Intent feloldás (*Intent Resolution*)

# Intent feloldás

- Az Intent-ben lévő Action, Data és Category mezők tartalma alapján
- Mindháromra teszteli az összes alkalmazás összes komponensét
- Regisztráció Intentek kezelésére: Intent filterek segítségével
  - > Egy XML node az AndroidManifest-ben
  - > Ezzel történik a szolgáltatások kiajánlása is más komponensek / alkalmazások számára

# Intent Filter

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

# Intent Filter

- Egy komponenshez több filtert is beállíthatunk
  - > Mindegyik leírja a komponens egy képességét, amivel a hozzá érkező implicit intenteket kezelní tudja
  - > Explicit intentek mindenkorán eljutnak a komponenshez, függetlenül a beállított filterektől
  - > minden filter egy kijánlott funkciónak, képességnak felel meg
  - > Pl: névjegy szerkesztő activityhez két intent filter:
    - Kiválasztott névjegy szerkesztése
    - Új (üres) névjegy létrehozása
  - > A megfelelő Activity/Service/BR node-ban **<intent-filter>** node bevezetése

# Intent Filter – Action

- Action: az az akció, amit a komponens le tud kezelní
  - > Vagy egy default ACTION
    - Pl. „android.intent.action.ACTION\_CAPTURE\_IMAGE”, ha egyedi kamera alkalmazást csinálunk
  - > Vagy teljesen egyedi név kell
    - Package name prefix

```
<action android:name="com.amorg.notepad.SHOW_NOTE" />
```

- Legalább egy Action kell az intent filterbe
- Ha az Intent-ben nincs kitöltve az Action, akkor bármilyen action-el rendelkező komponens átmegy a teszten

# Intent Filter – Data

- Milyen típusú **adat** kezelésére képes a komponens
- Data mező részei:
  - >  **mimeType**: megszokott MIME típus (pl. `text/*`), de egyedit is definiálhatunk. Ekkor a package name előtt vnd (*Vendor*) prefix szükséges  
`<data android:mimeType="vnd.amorg.notebook.note/*"/>`
  - > **URI**: scheme://host:port/path
- Intent URI és type nélkül csak ugyanilyen filterre illeszthető
- Intent csak URI-val: csak type nélküli, min. egy illeszthető URI-val rendelkező filterre (pl. `mailto:` vagy `tel:`)
- Intent csak type-al: A filter is csak egy megfelelő type-ot ír elő

# Intent Filter – Category

- Category: milyen körülmények között szolgálható ki az Action. Fontosabbak:
  - > **LAUNCHER**: csak Activity-re állítható, ekkor megjelenik az alkalmazások között a launcher-ben (így lehet több belépési pont, lásd Google+ kliens)
  - > **HOME**: csak Activity-re állítható, ha be van állítva és nincs Action, akkor a natív home screen alternatívája lesz (így lehet saját home képernyőt csinálni)
  - > **DEFAULT**: Data és Action beállítással együtt érvényes, alapértelmezett akciót állít be az adat típuson (pl. kép, videó, pdf, doc)

# Intent feloldás

- Csak implicit intent esetén (!)
- Sok lépésből álló folyamat
- Az Intentnek mindenkomponensnek (*Action, Data, Category*) teszten meg kell felelnie!
- Hogy juthat el mégis a komponenshez?
  - > Explicit intent
  - > Ugyanennek a komponensnek van egy másik intent filtere, amin átment

# Gyakoroljunk!

- Készítsünk egy alkalmazást, amely alapértelmezett alkalmazás *txt* állományok kezelésére!

# Intent Filter példa

- Készítsünk egy alkalmazást, amely alapértelmezett alkalmazás kép állományok kezelésére, a kapott képet egy imageView-n megjeleníti

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="hu.aut.android.intentdemo"

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <intent-filter tools:ignore="AppLinkUrlError">
                <action android:name="android.intent.action.SEND"/>
                <action android:name="android.intent.action.EDIT" />
                <action android:name="android.intent.action.PICK" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT"/>
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:mimeType="image/*" />
                <!-- <data android:mimeType="text/plain" /> -->
            </intent-filter>

        </activity>
    </application>
</manifest>
```

```
    val b = intent.extras
    if (b != null && b.containsKey(Intent.EXTRA_STREAM)) {
        val uri = b[Intent.EXTRA_STREAM] as Uri
        binding.imageView.setImageURI(uri)
    }
}
```

# Intent FLAG-ek

- Új Activity alapértelmezetten a Back Stack tetejére jön létre
- Ha szükséges, változtathatunk ezen Intent Flag-ek segítségével
- Ezek módosítják az alapértelmezett viselkedést (pl. a Back gomb hatását), használatuk csak indokolt esetben javasolt, és tesztelni kell a használhatóságot! (pl. böngészőkben mit csinál a Back?)

# Intent FLAG-ek

- A leggyakrabban használtak:
  - > **FLAG\_ACTIVITY\_NEW\_TASK**: A létrehozott Activity nem a hívó taszkban indul, hanem újat kap (launcher jellegű appok)
  - > **FLAG\_ACTIVITY\_SINGLE\_TOP**: Ha a hívott Activity-ből már van egy példány a stack tetején, akkor nem hoz létre egy újabbat, hanem a meglévőt folytatja
  - > **FLAG\_ACTIVITY\_CLEAR\_TOP**: Ha a hívott Activity már ott van valahol a stack-ben, akkor letakarítja a fölötte lévőket és azt folytatja

# Intent FLAG-ek

- Beállítása:

```
var i:Intent = Intent(Intent.ACTION_VIEW)
i.setFlags( Intent.FLAG_ACTIVITY_CLEAR_TOP |
            Intent.FLAG_ACTIVITY_NEW_TASK )
```

- Nagyon ritka az az eset, amikor Intent Flag-et kell használni
- Rengeteg más Flag van még, lásd a setFlags() metódus dokumentációjánál

# Intent lekérése

- Az Activity/Service/BR lekérheti azt az Intentet, ami őt indította
  - > Extrák kibányászása
  - > Data kinyerése
  - > ( minden más részére is van getter)
- `getIntent()` metódus szolgáltatja

```
var dataUri: Uri = intent.getData()  
var extras: Boundle = intent.getExtras()  
var action: String = intent.getAction()
```

# Linkify

- Automatikusan linket készít a TextView-ban lévő szövegrészekből
- Rákattintva a megfelelő Intentet küldi a rendszernek

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:text="@string/linkify_me"  
    android:autoLink="phone|email"  
/>
```

# Linkify saját típusok

- Beépített típusok
  - > Web, email, phone, map
- Saját típus is definiálható, amiből link készül
  - > RegExp + séma megadásáva



Ebből gyártja majd az Intent data URI-t

- > Mi lesz az Intent Action?
  - android.intent.action.VIEW

# Pending Intent

- Intentet nem csak azonnal lehet küldeni
- Előre definiálhatunk olyan Intentet, ami majd később, vagy akár egy másik alkalmazásból fog küldődni
  - > Pl. widgetre kattintáskor vagy időzítve küldődik
- Azért, hogy előre felkészülhessünk a fogadására
- Neve: PendingIntent

# Intent - összefoglaló

- 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

# Hogy is volt?

- Hogy kell Activity-t indítani, ha vissza akarunk kapni adatot belőle?
  - > `startActivityForResult()`
- Mit kell beállítani a manifestben, ha saját Home Screen-t akarunk csinálni?
  - > Intent filter: `category=HOME`

# Kérdések

