

Kliensalkalmazások

Android 3 – Fragment, RecyclerView, Persistence

2023. 05. 08.

Gazdi László

gazdi.laszlo@aut.bme.hu



Automatizálási és
Alkalmazott
Informatikai Tanszék

Miről volt szó az előző órán?

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



Tartalom

- Grafikai erőforrások
- Animációk
- Fragmentek
- Navigation Component
- Listakezelés: RecyclerView
- Perzisztens adattárolási lehetőségek
 - > Egyszerű kulcs-érték tár: SharedPreferences
 - > Adatbázistámogatás, SQLite
 - > ORM megoldások
 - > Room használata a gyakorlatban
 - > Filekezelés
- Adattárolás a felhőben

GRAFIKAI ERŐFORRÁSOK XML-BEN

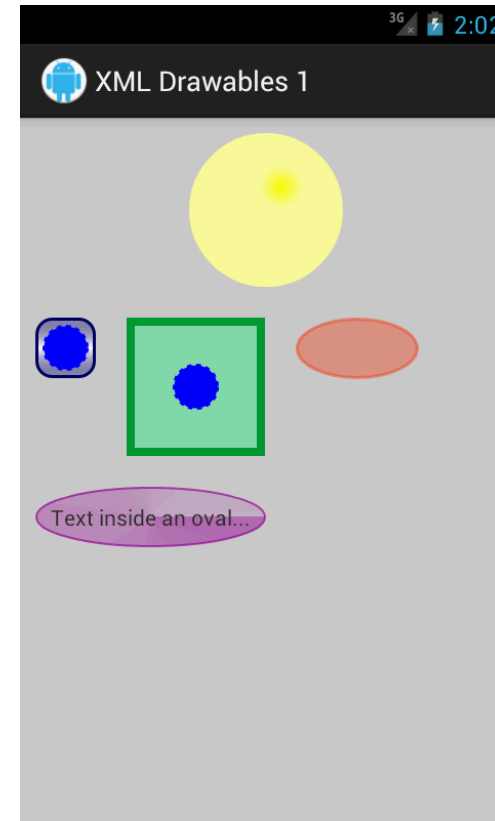
Grafikai erőforrások 1/2

- Grafikai elemek, hátterek is leírhatók XML-ben

```
<?xml version="1.0" encoding="utf-8"?>
<shape
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="oval">

  <gradient
    android:type="radial"
    android:gradientRadius="20"
    android:centerX=".6"
    android:centerY=".35"
    android:startColor="#FFFF00"
    android:endColor="#FFFF99" />

  <size
    android:width="100dp"
    android:height="100dp"/>
</shape>
```



Grafikai erőforrások 2/2

- Zöld szaggatott vonal:

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

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  android:shape="line">
```

```
  <stroke
```

```
    android:width="2dp"
```

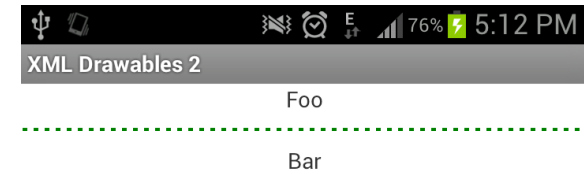
```
    android:color="#008000"
```

```
    android:dashWidth="3dp"
```

```
    android:dashGap="4dp"/>
```

```
  <size android:height="20dp" />
```

```
</shape>
```



Állapotok leírása grafikai erőforrásokban

- Például gombfelirat színe:

> res/color/button_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true"
    android:color="#ffff0000"/> <!-- pressed -->
  <item android:state_focused="true"
    android:color="#ff0000ff"/> <!-- focused -->
  <item android:color="#ff000000"/> <!-- default -->
</selector>
```

- Használat:

```
<Button
  android:layout_width=„match_parent”
  android:layout_height="wrap_content"
  android:text="@string/button_text"
  android:textColor="@color/button_text" />
```

ANIMÁCIÓK

Animációk

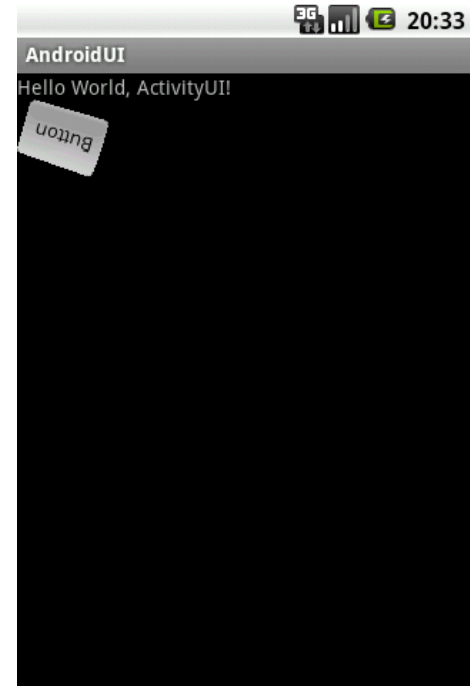
- Animációk támogatása
 - > XML erőforrás (*res/anim*)
 - > Programkód
- Layout animáció
 - > *Scale*
 - > *Rotate*
 - > *Translate*
 - > *Alpha*
- Három fő típus:
 - > Tween animáció
 - > Frame animáció
 - > Property animator

Tween animáció erőforrás

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
        android:interpolator=
            "@android:anim/accelerate_interpolator"
        android:fromXScale="0.0"
        android:toXScale="1.0"
        android:fromYScale="0.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="1000" />

    <alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:duration="5000"/>

    <rotate
        android:interpolator="@android:anim/accelerate_interpolator"
        android:fromDegrees="0.0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
</set>
```



Tween animáció lejátszása

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        var demoAnim = AnimationUtils.loadAnimation(this,  
            R.anim.demo_anim)  
        btnAnim.setOnClickListener {  
            btnAnim.startAnimation(demoAnim)  
        }  
    }  
}
```

Melyik állítás igaz?

- A. Egyszerű alakzatok megjelenítése is csak képi erőforrással lehetséges.
- B. Nem készíthető saját stílus, csak a beépítettek használhatók.
- C. A gombok állapotától függő színét selectorokkal egyszerűen állíthatjuk.
- D. Activity nem jelenhet meg dialógus formájában.

FRAGMENT-EK

Fragmentek

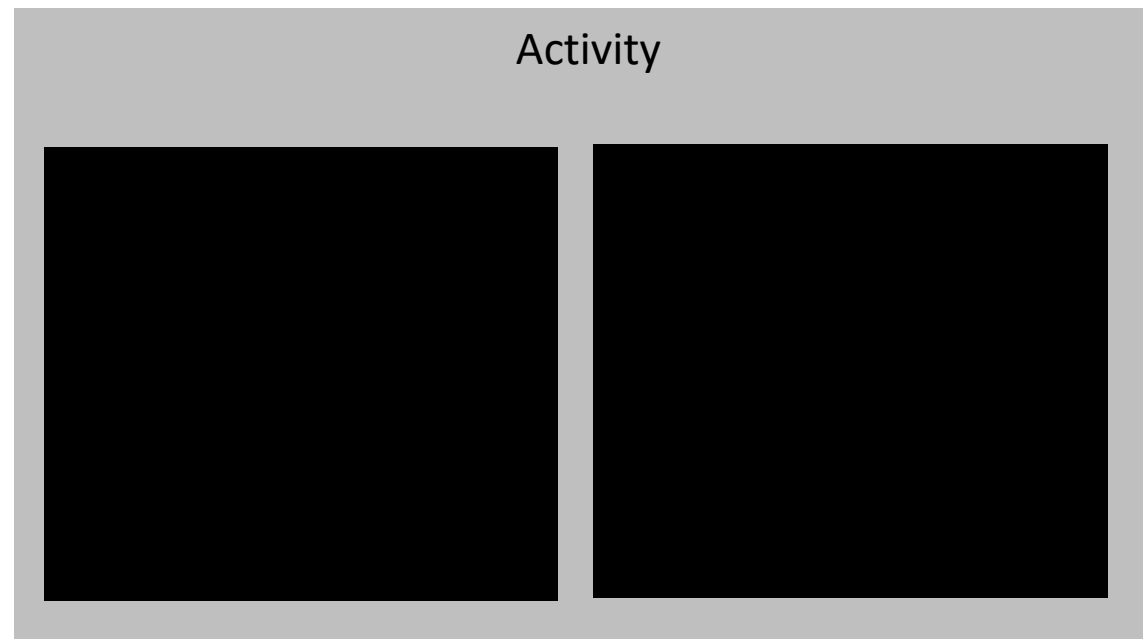
- Mik azok a Fragment-ek?
 - > Elsősorban: A képernyő egy nagyobb részéért felelős objektumok
 - > Továbbá: A háttérben munkát végző objektumok
- Miért kellene nekünk?
 - > Nagy képernyőméret = több funkció egy képernyőn = bonyolultabb Activity-k
 - > Fragment-ekkel modulárisabb, rugalmasabb architektúra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

Fragmentek

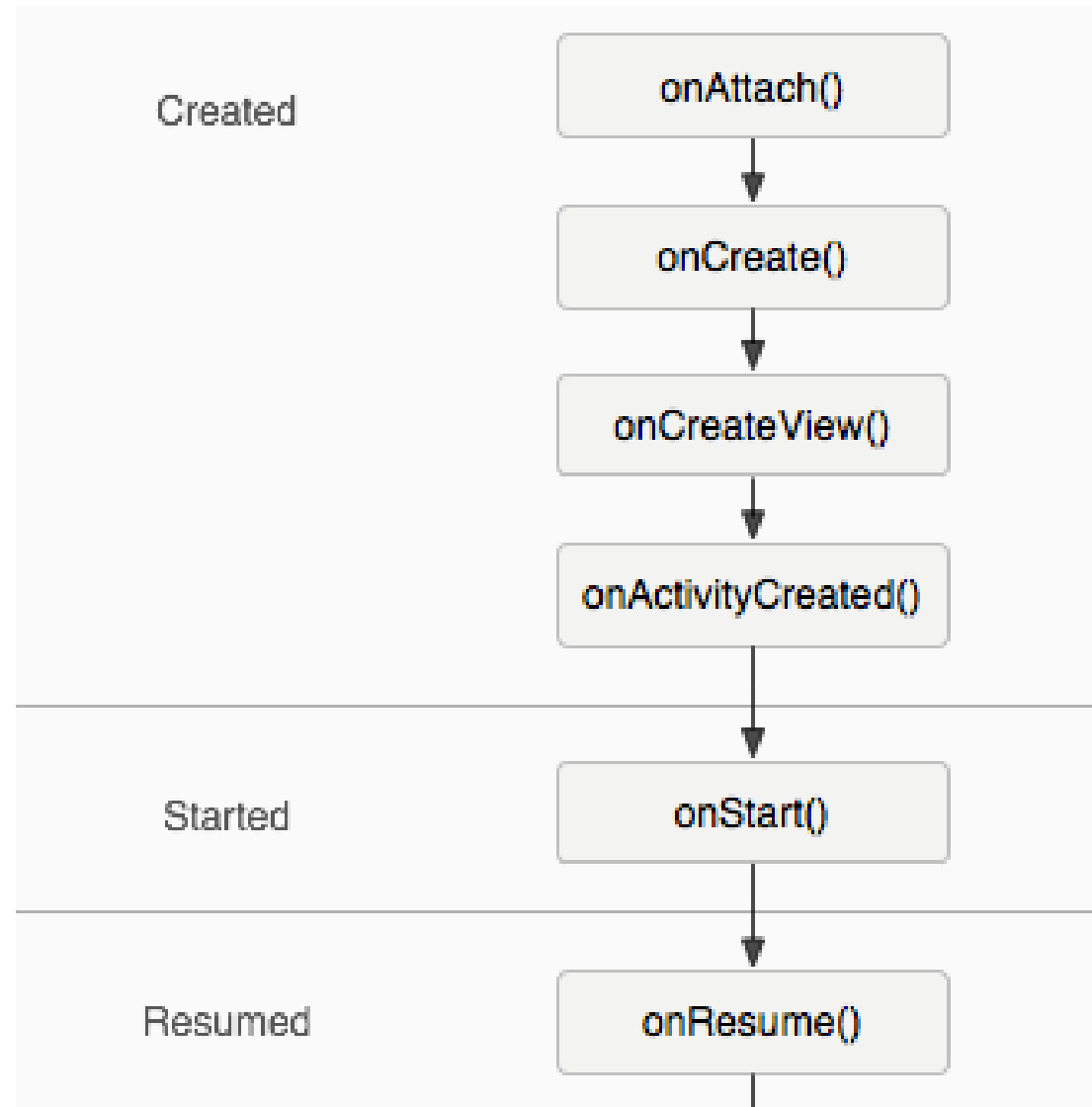
- Miben másabb mint az **Activity**?
 - > Kisebbs granualitás, nem mindig teljes képernyő egy fragment
 - > Az életciklusa nem mindig egyezik, pl. le lehet csatolni egy fragmentet úgy, hogy az activity előtérben marad.
- Miben másabb mint egy **Custom View**
 - > Összetett életciklus, mely az activity-t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

Fragment és Activity

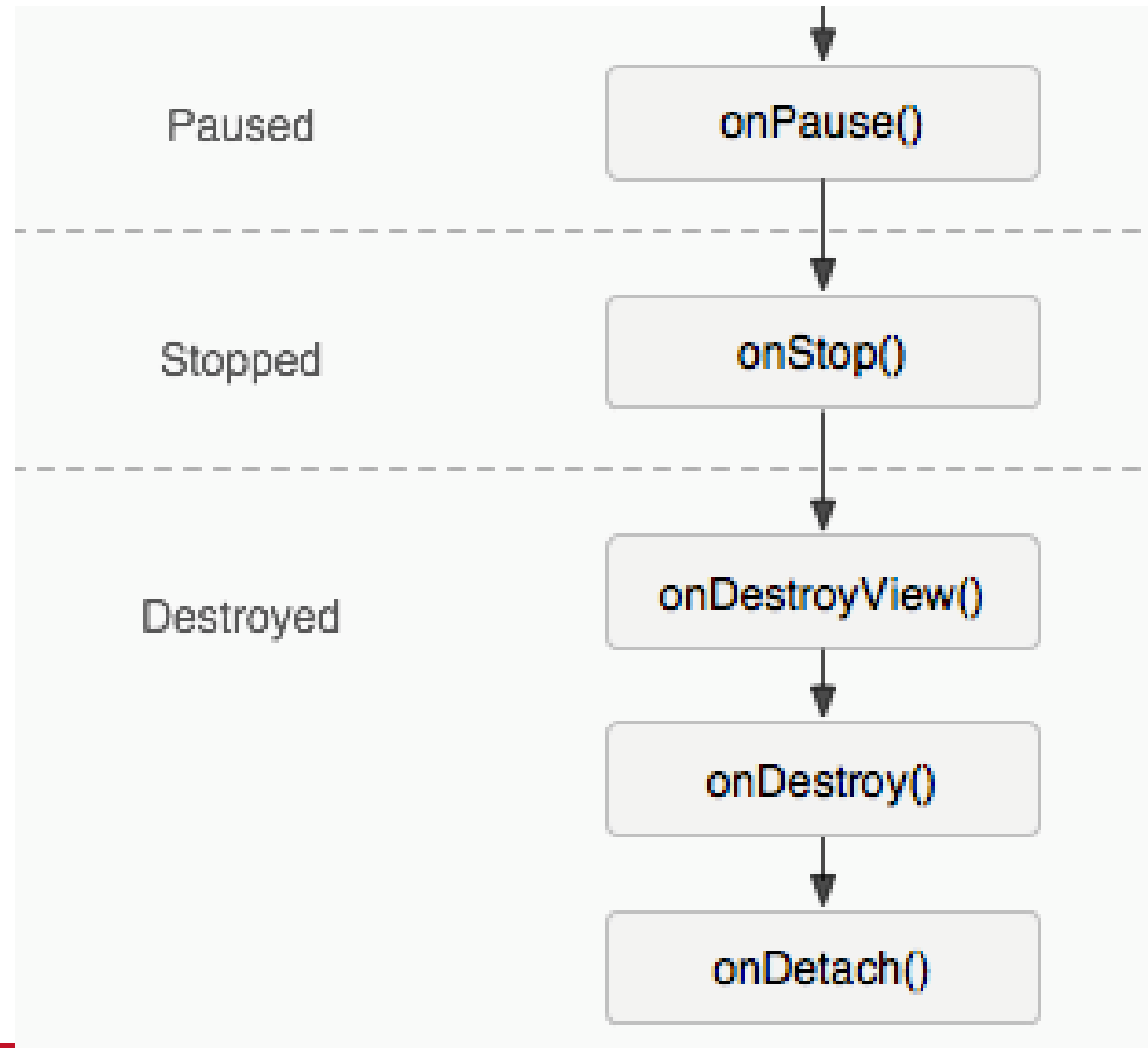
- Egy Fragment mindig egy Activity-hez csatoltan jelenik meg
- Az élelciklusaik nagyrészt megegyeznek



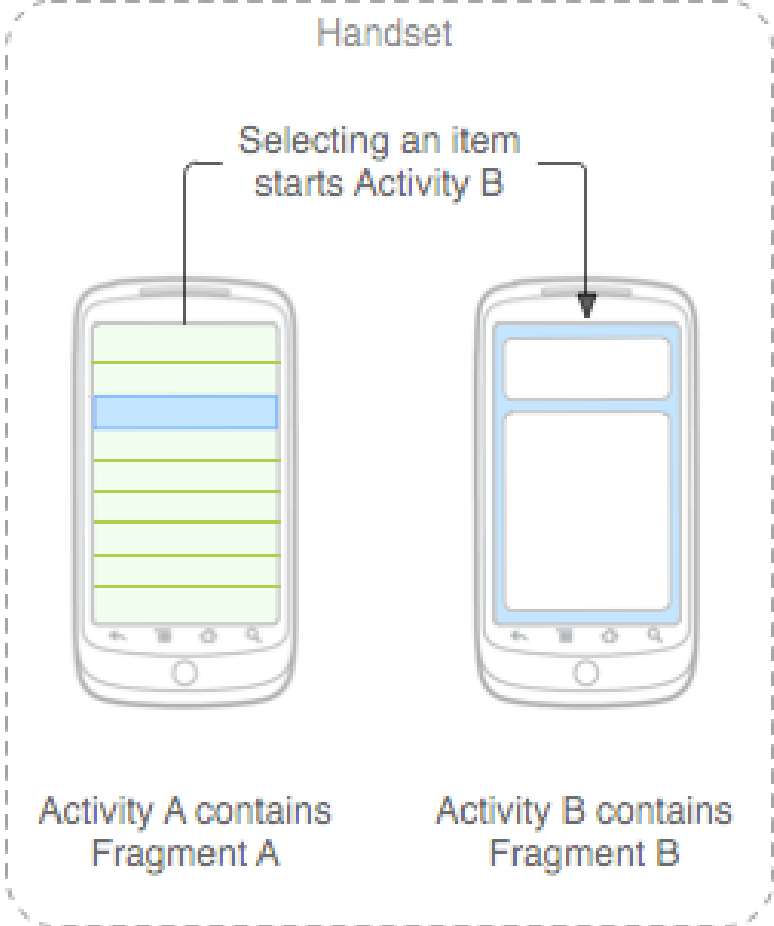
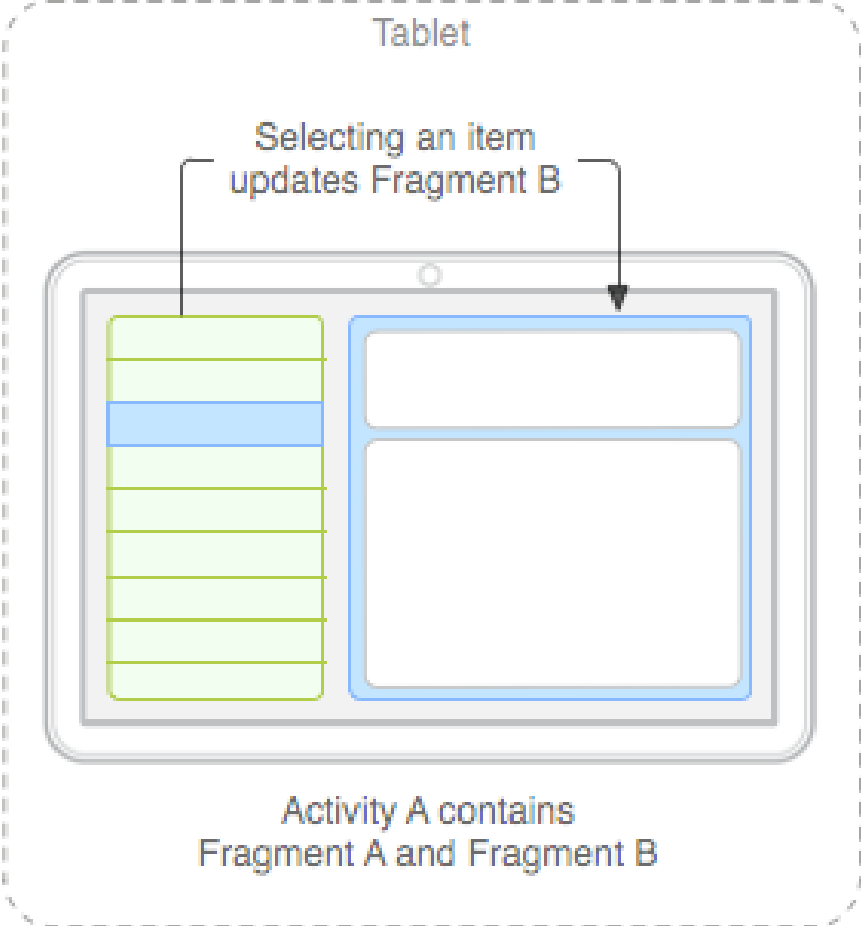
Fragment életciklus I.



Fragment életciklus II.



Eltérő képernyőméretek



UI Fragment készítése...

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

```
class FragmentProfile : Fragment() {
    private var binding: FragmentProfileBinding? = null
    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?):
View? {
    super.onCreateView(inflater, container, savedInstanceState)
    _binding = ResultProfileBinding.inflate(inflater, container, false)
    val view = binding.root
    return view
}

    override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
}
```

... 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*-rel menedzselhetők a *Fragment*-ek
 - > Activity: `.getManager()`
 - > *FragmentManager* indítása
 - > Aktív *Fragment*-ek közt keres
 - Tag alapján
 - ID alapján
 - > **FragmentManager-et** menedzseli

FragmentManager osztály I.

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

FragmentManager osztály II.

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

FragmentManager példa I.

- Fragment kicserélése:

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

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

FragmentManager 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)
```

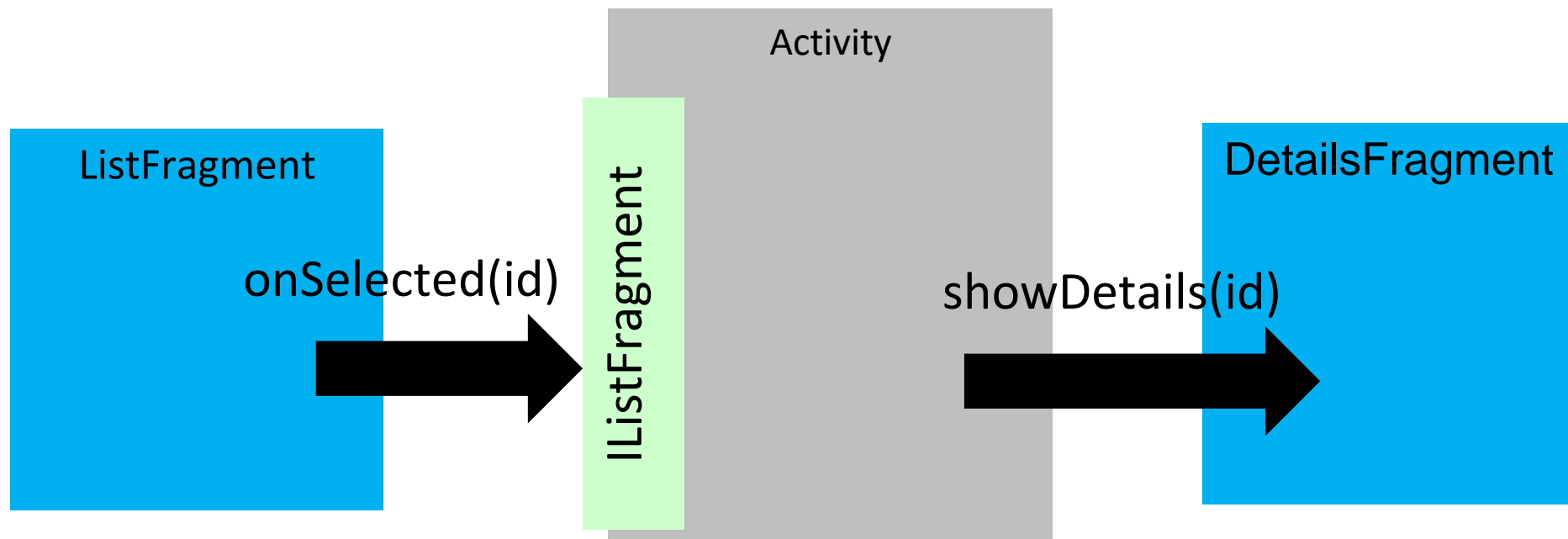
```
ft.setCustomAnimations(R.anim.slide_in_top,R.anim.slide_out_bottom)
```

```
ft.addToBackStack(null)
```

```
ft.commit()
```

Fragment kommunikáció

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



DialogFragment I.

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

DialogFragment II.

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

Melyik állítás nem igaz a Fragmentekre?

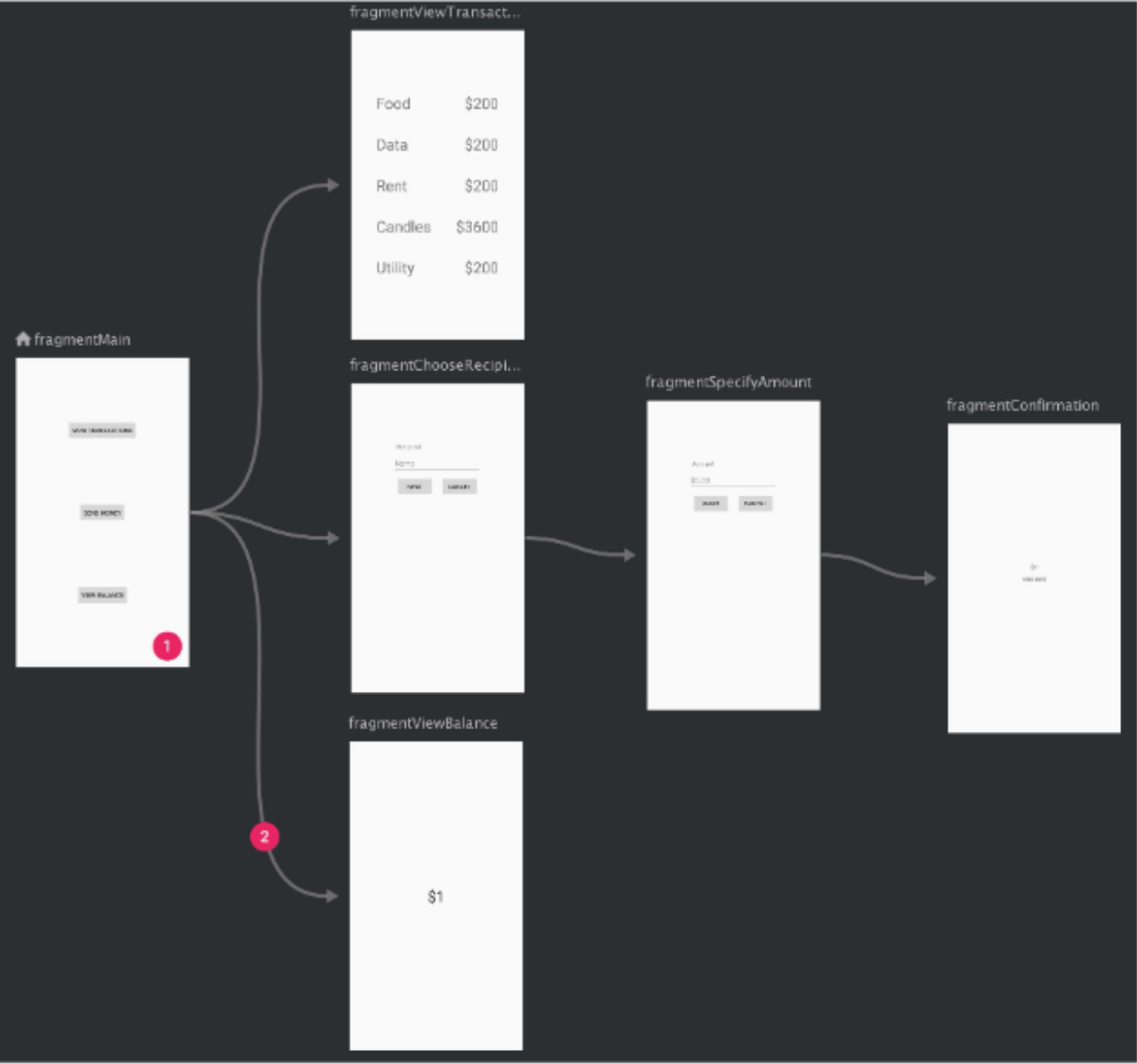
- A. Saját élelciklus függvényekkel rendelkeznek.
- B. Dialógusként nem jeleníthetők meg.
- C. Dinamikusan és statikusan is csatolhatók.
- D. A tabletek felhasználói felületének kialakításakor különösen hasznosak.

Navigation Component

Navigation Component

- Egyszerűsített navigáció *Activity*-k, *Fragment*ek és nézetek közt grafikus felületen.
- **Navigation graph:** XML erőforrás ami leírja a navigációs útvonalakat, vizuális megjelenítéssel is rendelkezik
- **NavHost:** Egy üres konténer amin belül a navigáció megvalósul (itt váltakoznak a nézetek), tipikusan egy *NavHostFragment*
- **NavController:** Egy objektum ami a navigációt vezérli és megvalósítja.

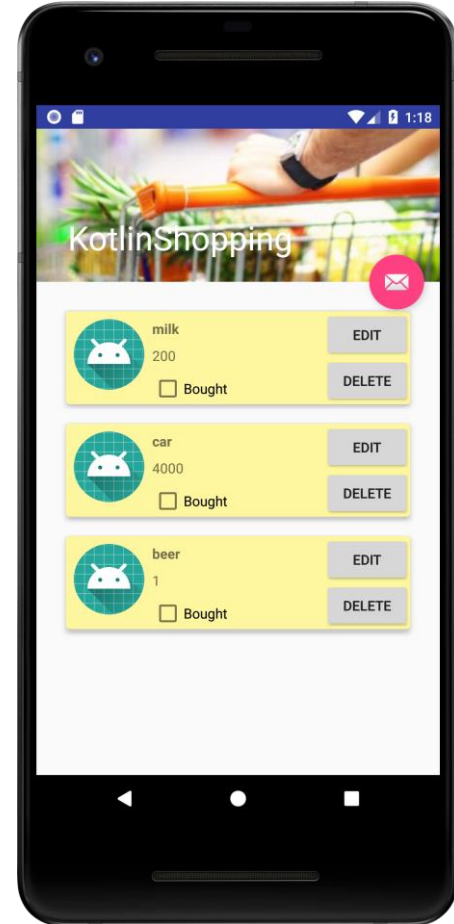
Navigation graph



LISTÁK KEZELÉSE

RecyclerView

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



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> 1/3

- Inicializálás, konstruktor

```
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)
}
```

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
```

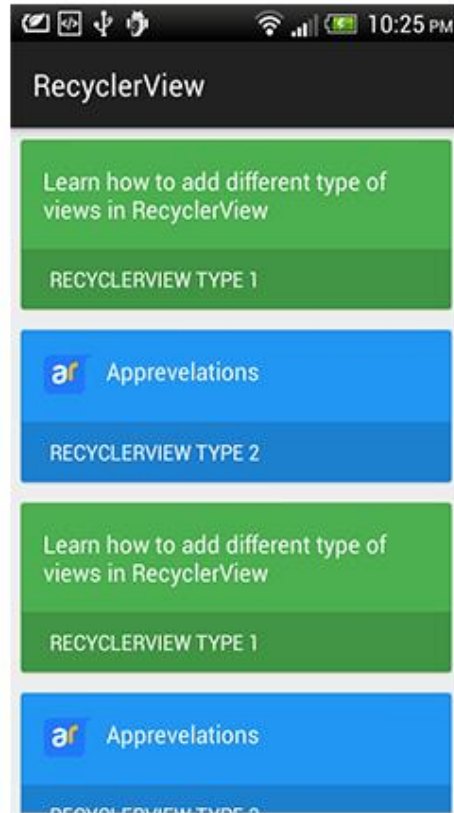
```
fun addItem(item: ShoppingItem) {  
    items += item  
    notifyItemInserted(items.lastIndex)  
}
```

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

ViewHolder pattern előnyei

- Statikus *ViewHolder* objektum, cache támogatás
- Nincs folyamatos *findViewById(...)* hívás
- Gyors működés

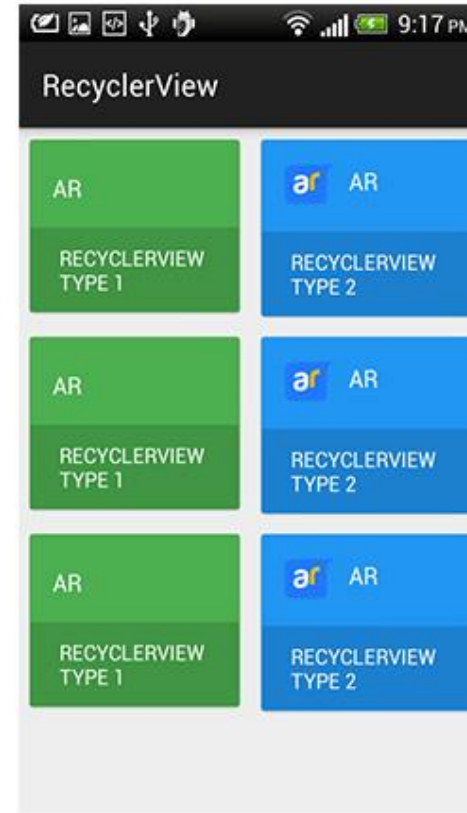
RecyclerView Layoutmanager-ek



Linear Layout View



Staggered Grid View



Grid View

Lista készítés

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

Mi nem igaz a RecyclerView-ra?

- A. Csak egymás alatti elrendezést támogat.
- B. Kikényszeríti a ViewHolder mintát.
- C. Támogatja a scrollozást.
- D. El kell készítenünk egy sor layout-ját.

Perzisztens adattárolás

Bevezetés

- Gyakorlatilag minden Android alkalmazásnak kell perzisztensen tárolnia bizonyos adatokat
 - > Beállítások szinte mindig vannak
 - > Kamera alkalmazások: új fénykép fájl mentése
 - > Online erőforrásokat használó appok: lokális cache
 - > Email alkalmazások: levelek indexelt adatbázisa
 - > Bejelentkezést tartalmazó appok: be van-e jelentkezve a felhasználó
 - > Első indításkor tutorial megjelenítése: első vagy későbbi indítás?
 - > Picasa, Dropbox: elsődleges tárhely a felhőben

Bevezetés

- Androidon minden igényre van beépített megoldás:
 - > **SharedPreferences**: alaptípusok tárolása kulcs-érték párokban
 - **DataStore** leváltja a SharedPreferences-t
 - > **File kezelés**:
 - **Privát lemezterület**: nem publikus adatok tárolása a fájlrendszerben
 - **Publikus lemezterület / SD kártya**: nagy méretű adatok tárolása, nyilvánosan hozzáférhető
 - > **SQLite adatbázis**: strukturált adatok tárolására
 - > **Hálózat**: saját webserveren vagy felhőben tárolt adatok
 - -BaaS: Backend as a Service

Shared Preferences

SharedPreferences

- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
 - > Típusok: *int, long, float, String, boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
 - > **MODE_PRIVATE**: csak a saját alkalmazásunk érheti el
 - > **MODE_WORLD_READABLE**: csak a saját alkalmazásunk írhatja, bárki olvashatja
 - > **MODE_WORLD_WRITABLE**: bárki írhatja és olvashatja
- Megőrzi tartalmát az alkalmazás és a telefon újraindítása esetén is

SharedPreferences

- Ideális olyan adatok tárolására, melyek primitív típusal könnyen reprezentálhatók, pl:
 - > Default beállítások értékei
 - > UI állapot
 - > Settings-ben megjelenő adatok (innen kapta a nevét)
- Több ilyen *SharedPreferences* fájl tartozhat egy alkalmazáshoz, a nevük különbözteti meg őket
 - > `getSharedPreferences (String name, int mode) ;`
 - > Ha még nem létezik ilyen nevű, akkor az Android létrehozza
- Ha elég egy SP egy Activity-hez, akkor nem kötelező elnevezni
 - > `getPreferences () ;`

SharedPreferences használata

- Olvasás:

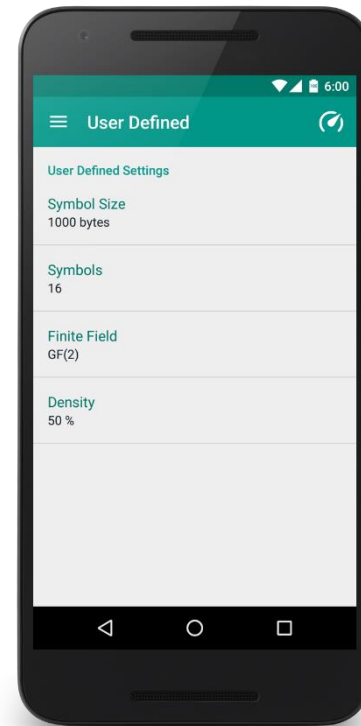
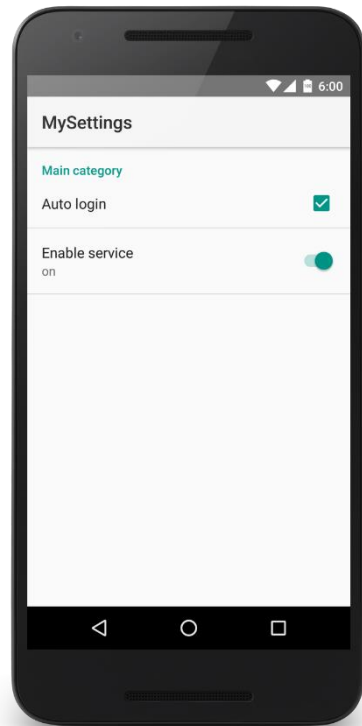
```
val sharedPref =  
activity?.getPreferences (Context.MODE_PRIVATE)  
  
val highScore = sharedPref.getInt (KEY, defaultValue)
```

- Írás:

```
val sharedPref =  
activity?.getPreferences (Context.MODE_PRIVATE)  
  
sharedPref.edit ().putInt (KEY, 100) .commit ()
```

Preferences Framework

- Az Android biztosít egy XML alapú keretrendszert saját Beállítások képernyő létrehozására
 - > Ugyanúgy fog kinézni mint az alap Beállítások alkalmazás
 - > Más alkalmazásokból, akár az op.rendszerből is átemelhető részek



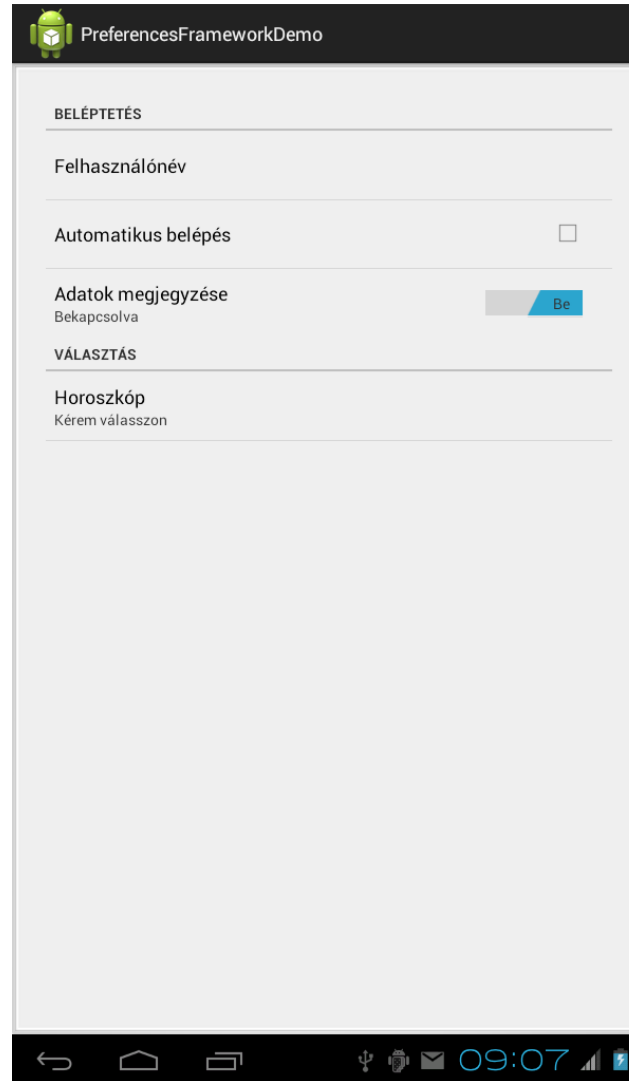
Preferences Framework

- Megvalósításához szükséges
 1. **XML**, ami leírja a megjelenítendő beállításokat
 2. **Activity**, ami a *PreferenceActivity* leszármazottja
 3. **SharedPreferencesChangeListener**: eseménykezelő a beállítások megváltozásának figyelésére (opcionális)
- Teljesen testre szabható struktúra
- Kinézetet a Framework adja
- Csak *SharedPreferences*-ben tárolt adatokkal működik
- Érdeemes ezt használni, ha Beállítások képernyőt szeretnénk az alkalmazásunkba

Példa Preferences nézet - XML

```
<?xml version="1.0" encoding="utf-8"?> <PreferenceCategory android:title="Választás" >
<PreferenceScreen xmlns:android=
                                <ListPreference
                                android:title="Horoszkóp"
                                android:summary="Kérem válasszon"
                                android:key="listPref"
                                android:entries="@array/listDisplayMarks"
                                android:entryValues="@array/listReturnMarks"/>
                                </PreferenceCategory>
                                </PreferenceScreen>
    <PreferenceCategory android:title=
        "Beléptetés" >
        <EditTextPreference
            android:defaultValue="empty"
            android:key="name"
            android:title="Username" />
        <CheckBoxPreference
            android:defaultValue="false"
            android:key="autologin"
            android:title="Automatikus
belépés" />
        <SwitchPreference
            android:title="Adatok
megjegyzése"
            android:key="remember"
            android:summaryOff="Kikapcsolva"
            android:summaryOn="Bekapcsolva"/>
    </PreferenceCategory>
```

Példa Preferences nézet - UI



Preferences Framework - XML

- Integrálhatjuk a rendszerszintű beállítások képernyőit is

```
<?xml version="1.0" encoding="utf-8"?>
  <PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="Beléptetés" >
      <Preference android:title="GPS beállítások" >
        <intent android:action=
          "android.settings.LOCATION_SOURCE_SETTINGS" />
      </Preference>
    </PreferenceCategory>
  </PreferenceScreen>
```

SQLite

SQLite

- Az Android alapból tartalmaz egy teljes értékű relációs adatbáziskezelőt
 - > SQLite – majdnem MySQL
- Strukturált adatok tárolására ez a legjobb választás
- Alapból nincs objektum-relációs réteg (ORM) fölötte, nekünk kell a sémát meghatározni és megírni a query-ket
- Külső ORM osztálykönyvtárak
- Mivel SQL, érdemes minden táblában elsődleges kulcsot definiálni
 - > autoincrement támogatás
 - > Ahhoz, hogy *ContentProvider*-el ki tudjuk ajánlani (később), illetve UI elemeket Adapterrel feltölteni (pl. list, grid), **kötelező egy ilyen oszlop**, melynek neve: „_id”

Android SQLite jellemzői 1/2

- Standard relációs adatbázis szolgáltatások:
 - > SQL szintaxis
 - > Tranzakciók
 - > Prepared statement
- Támogatott oszlop típusok (a többit ilyenekre kell konvertálni):
 - > TEXT (Java String)
 - > INTEGER (Java long)
 - > REAL (Java double)
- Az SQLite nem ellenőrzi a típust adatbeírásakor, tehát pl Integer érték automatikusan bekerül Text oszlopba szöveggként

Android SQLite jellemzői 2/2

- Az SQLite adatbázis elérés file rendszer elérést jelent, ami miatt lassú lehet!
- Adatbázis műveleteket érdemes asszinkron módon végrehajtani (pl *AsyncTask* használata v. *Loader*)

Object Relation Mapping (ORM)

Mi az ORM?

- Objektumok tárolása relációs adatbázisban
- Alapelvek:
 - > Osztálynév -> Tábla név
 - > Objektum -> Tábla egy sora
 - > Mező -> Tábla oszlopa
 - > Stb.

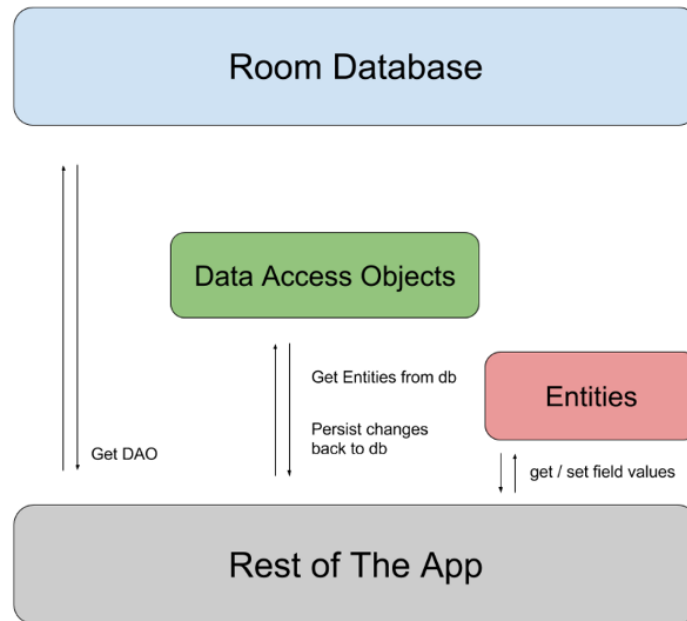


ORM példa – objektumok tárolása

```
Class Person {  
Private String name;  
Private String address;  
Private String tel;  
}
```


Room Persistence Library

- Absztrakciós réteg az SQLite felett
- SQLite teljes képességeinek használata
- Room architektúra:



Room példa - Entity

```
@Entity(tableName = "grade")
data class Grade(
    @PrimaryKey(autoGenerate = true) var gradeId: Long?,
    @ColumnInfo(name = "studentid") var studentId: String,
    @ColumnInfo(name = "grade") var grade: String
)
```

Room példa - DAO

```
@Dao
interface GradeDAO {
    @Query("""SELECT * FROM grade WHERE grade="B" """)
    fun getBGrades(): List<Grade>

    @Query("SELECT * FROM grade")
    fun getAllGrades(): List<Grade>

    @Query("SELECT * FROM grade WHERE grade = :grade")
    fun getSpecificGrades(grade: String): List<Grade>

    @Insert
    fun insertGrades(vararg grades: Grade)

    @Delete
    fun deleteGrade(grade: Grade)
}
```

RoomDatabase

```
@Database(entities = [Grade::class], version = 1)
abstract class AppDatabase : RoomDatabase() {

    abstract fun gradeDao(): GradeDAO

    companion object {
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                INSTANCE ?: Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "grade_db")
                    .fallbackToDestructiveMigration()
                    .build()
                .also { INSTANCE = it }
            }
        }
    }
}
```

Room használat

- Insert

```
val grade = Grade(null, etStudentId.text.toString(),  
    etGrade.text.toString())
```

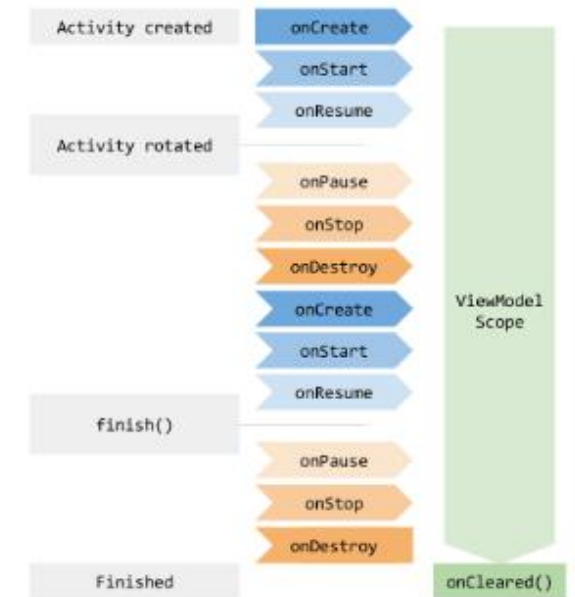
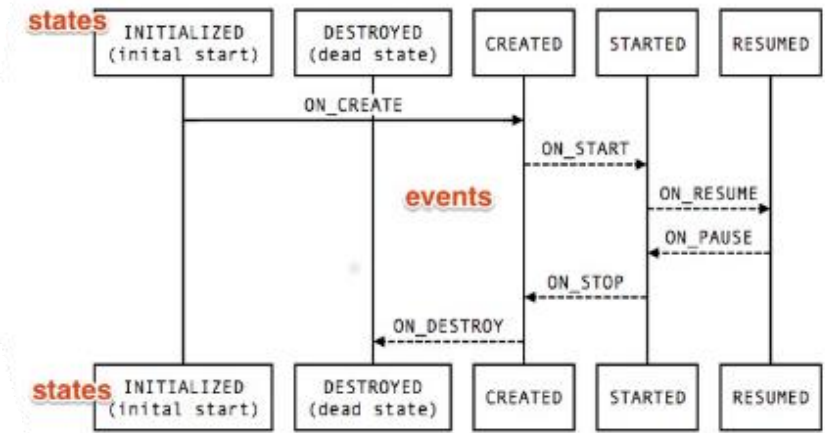
```
val dbThread = Thread {  
    AppDatabase.getInstance(this@MainActivity).gradeDao().insertGrades(grade)  
}  
dbThread.start()
```

- Query

```
val dbThread = Thread {  
    val grades = AppDatabase.getInstance(this@MainActivity).gradeDao()  
        .getSpecificGrades("A+")  
    runOnUiThread {  
        tvResult.text = ""  
        grades.forEach {  
            tvResult.append("${it.studentId} ${it.grade}\n")  
        }  
    }  
}  
dbThread.start()
```

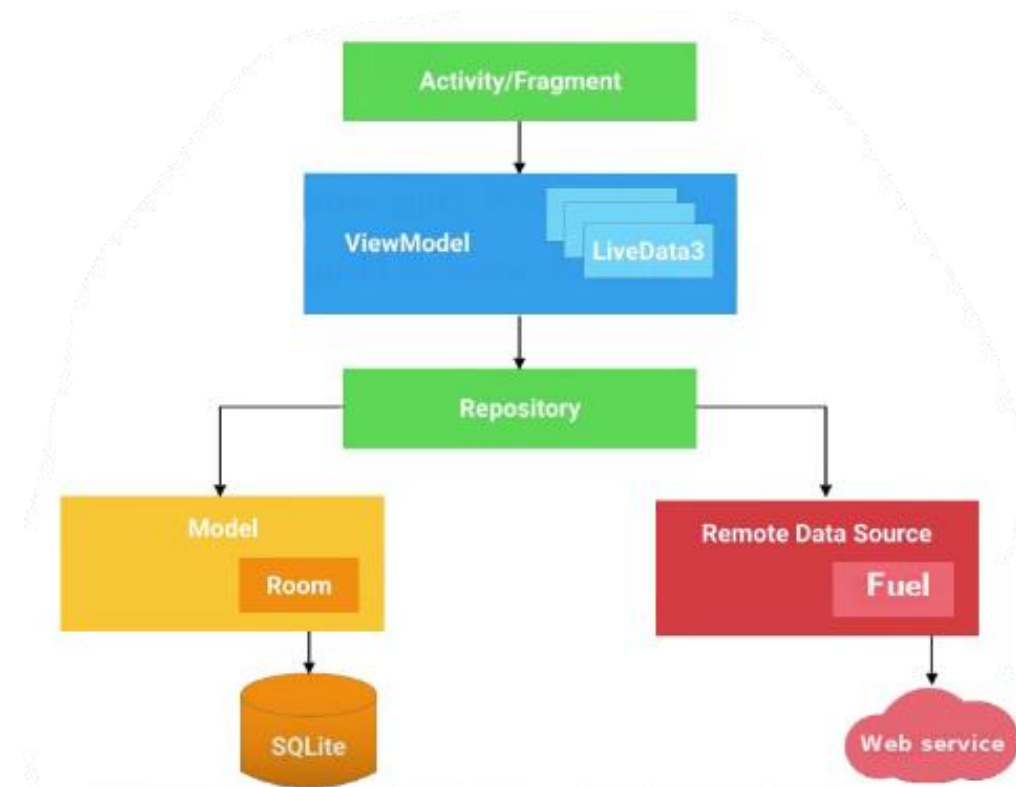
Architecture Components elemek

- Lifecycle / LifecycleObserver
 - > Lifecycle-függő komponensek, objektumok hozhatók létre
- LiveData
 - > Adat kezelő, megfigyelhetővé teszi az adatot
- ViewModel
 - > UI-on megjelenő adatok egyszerű kezelése, mely független a konfiguráció változástól
- Room Persistence Library
 - > Google saját ORM megoldása, a teljes SQLite képességeket kihasználja



Javasolt architektúra

- Az Activity-k, Fragmentek, egyedi nézetek *ViewModel*-eket használnak
- A *ViewModel*-ek *LiveData*-kon keresztül teszik megfigyelhetővé az adatokat
- A *ViewModel*-ek *Repository*-kat használnak az adatforrások elrejtéséhez
 - Perzisztencia használata offline működés támogatására



Mi igaz az Androidos adattárolásra?

- A. Nincs beépített adatbáziskezelő.
- B. Egy Activityhez csak egy SharedPreferences tartozhat.
- C. Adatbázis műveletet mindig aszinkron módon kell végezni.
- D. Az Android tartalmaz beépített ORM réteget.

File kezelés

Fájlkezelés Androidon

- Ugyanaz mint sima Java esetén
- Néhány specialitás Android környezetben
- Két lemezterületet különböztet meg
 - > **Internal storage:** az a védett tárhely, amit kizárólag az alkalmazás érhet el, se a user, se más appok nem fér hozzá
 - > **External storage:** felhasználó által is írható-olvasható terület (~SD kártya)
- *External storage* is lehet belső memóriában, ha nincs SD kártya a készülékben (mondjuk mert tablet)

Internal storage

- *openFileOutput(String filename, int mode)*

- > filename-ben nem lehet „\”, egyébként kivételt dob

- > Támogatott módok:

- Context.MODE_PRIVATE: alapértelmezett megnyitási mód, felülírja a fájlt ha már van benne valami
 - Context.MODE_APPEND: hozzáfűzi a fájlhoz amit beleírunk
 - Lehet WORLD_READABLE vagy WORLD_WRITEABLE is, ha szükséges, de nem ez a javasolt módja az adatok kiajánlásának, hanem a ContentProvider (később)

- > Privát vagy Append mód esetén nincs értelme kiterjesztést megadni, mert máshonnan úgysem fogják megnyitni

- > Ha nem létezik a fájl akkor létrehozza, a **WORLD_*** módok csak ekkor értelmezettek

Internal storage

- Fájl olvasása ugyanígy:
 - > **openFileInput(String filename)** hívása (FileNotFoundException-t dobhat)
 - > Byte-ok kiolvasása a visszakapott *FileInputStream*-ből a **read()** metódussal
 - > **Stream bezárása close() metódussal!**
- Cache használata
 - > Beépített mechanizmus arra az esetre, ha cache-ként akarunk fájlokat használni
 - > **getCacheDir()** metódus visszaad egy File objektumot, ami a cache könyvtárra mutat
 - > Ezen belül létrehozhatunk cache fájlokat
 - > Kevés lemezterület esetén először ezeket törli az Android
 - Nem számíthatunk rá, hogy mindig ott lesznek!
 - > Google ajánlás: maximum 1MB-os fájlokat rakjunk ide

Statikus fájlok egy alkalmazáshoz

- Szükséges lehet a fejlesztett alkalmazáshoz statikusan fájlokat linkelni
 - > Kezdeti, nagy méretű, feltöltött bináris adatbázis fájl
 - > Egyedi formátumú állomány
 - > Bármi ami fájl, de nem illik a res könyvtár mappáiba (drawable, xml, stb)
- Fejlesztéskor a **res/raw** mappába kell raknunk őket
- Ezek telepítéskor szintén az internal storage-be kerülnek
- Read-only lesz telepítés után, nem tudjuk utólag módosítani
- Olvasásuk futásidőben:

```
val inStream: InputStream =  
    resources.openRawResource(R.raw.myfile)
```

Nyilvános lemezterület

- Lehet akár SD kártyán, akár belső (nem kivehető) memóriában
- Bárki által írható, olvasható a teljes fájlrendszer
- Amikor a felhasználó összeköti a telefont a számítógépével, és „*USB storage*” módra vált (mount), a fájlok hirtelen csak olvashatóvá válnak az alkalmazások számára
- Semmilyen korlátozás/tiltás nincs arra, hogy a nyilvános területen lévő fájljainkat a felhasználó letörölje, lemásolja vagy módosítsa!
 - > Amit ide írunk, az bármikor elveszhet

Nyilvános lemezterület

- Legfontosabb tudnivalók
 - > Használat előtt ellenőrizni kell a tárhely elérhetőségét
 - > Fel kell készülni arra, hogy bármikor elérhetetlenné válik

```
val state: String = Environment.getExternalStorageState()
// sokféle állapotban lehet, nekünk kettő fontos:
when (state) {
    Environment.MEDIA_MOUNTED -> {
        // Olvashatjuk és írhatjuk a külső tárat
    }
    Environment.MEDIA_MOUNTED_READ_ONLY -> {
        // Csak olvasni tudjuk
    }
    else -> {
        // Valami más állapotban van, se olvasni,
        // se írni nem tudjuk
    }
}
```

Nyilvános lemezterület

- Fájlok elérése a nyilvános tárhelyen **2.2 verziótól felfelé**:

```
val filesDir: File = getExternalFilesDir(type: Int)
```

- type: megadhatjuk milyen típusú fájlok könyvtárát akarjuk használni, például:
 - > null: nyilvános tárhely gyökere
 - > DIRECTORY_MUSIC: zenék, ahol az zenelejátszó keres
 - > DIRECTORY_PICTURES: képek, ahol a galéria keres
 - > DIRECTORY_RINGTONES: csengőhangok, ez is hang fájl, de nem zenelejátszóban akarjuk hallgatni
 - > DIRECTORY_DOWNLOADS: letöltések default könyvtára
 - > DIRECTORY_DCIM: a kamera ide rakja a fényképeket
 - > DIRECTORY_MOVIES: filmek default könyvtára

Nyilvános lemezterület

- Média típusonként külön alapértelmezett könyvtárak
- Így az azokat lejátszó/kezelő alkalmazásoknak nem kell az egész lemezt végigkeresni, csak a megfelelő könyvtárakat
- Indexelésüket a MediaScanner osztály végzi
 - > Ez mindenhol keres, és ha a talált média fájlok nem default könyvtárban vannak, akkor megpróbálja kategorizálni őket kiterjesztésük és MIME típusuk szerint
 - > Ha nem szeretnénk beengedni egy könyvtárba, akkor egy üres fájlt kell elhelyezni, melynek neve: ".nomedia"
 - Így például egy alkalmazás által készített fotók nem fognak látszódni a galériában
 - > A megfelelő default könyvtárba rakjuk az alkalmazásunk által létrehozott fájlokat, ha meg akarjuk osztani a userrel
- ***android.permission.WRITE_EXTERNAL_STORAGE***

Mi igaz az Androidos adattárolásra?

- A. Az External Storage-ba mentett fájljaink mindig elérhetőek.
- B. Az Internal Storage-ban lévő cache korlátlan ideig elérhető.
- C. A háttértár eléréséhez engedély szükséges.
- D. A saját statikus fájljainkat nekünk kell „kézzel” felmásolni az Internal Storage-ba.

Adatkezelés a felhőben

Backend as a Service



Mi található a szerver oldalon?

1. Saját implementáció

- > PHP, Java, .NET, Node.JS, etc.
- > Software as a service (SaaS)

2. Felhő szolgáltatás használata

- > Platform as a Service: saját implementáció futtatása egy cloud megoldásban
 - OpenShift, Heroku, Azure, Amazon, etc.
- > Backend as a Service: háttér szolgáltatások használata, melyek elrejtik a bonyolult DB műveleteket és kommunikációt
Parse, Kumulus, Backendless

BaaS képességek

- Felhasználó kezelés
- Perzisztencia, táblák, backup
- Offline működés támogatása
- File kezelés
- Verzió kezelés
- Analytics
- Kód generálás
- Dinamikus kód futtatás
- Media streaming
- Geolocation
- Social Networks
- Push értesítés

BaaS példák

- Firebase: <https://firebase.google.com/>
- Backendless: <http://backendless.com/>
- Kumulos: <http://www.kumulos.com/>

BaaS hátrányok

- Félelem a felhő szolgáltatásokkal szemben
- „Lock-in” effektus:
 - > Körülményes, ha váltani szeretnénk egyik felhő szolgáltatásról a másikra, nincs interfész a kettő között
- BaaS specifikus UI elemek használata, pl. Parse UI elemek; váltáskor probléma
- Adat szuverenitás (EU vs. USA)
- Számlázás

Firebase képességek



Firestore fő funkciók

- https://www.youtube.com/watch?v=QcsAb2RR52c&ab_channel=Firestore
- Real time adatbázis: JSON alapú NoSQL tárolás
 - > Perzisztens
 - > Eseményvezérelt, minden változásról értesítés
- Cloud Firestore (új generációs real-time adatbázis fejlett lekérdezés támogatással)
- Authentikáció:
 - > E-mail/közösségi hálózatok/egyedi
- Storage:
 - > file/kép tárolás
- Crash reporting
- Analytics
- Notifications
- ...

Hogy is volt?

- Mire használhatók a Fragmentek?
- Hogyan csatolhatók a Fragmentek?
- Hogyan kommunikálhatnak a Fragmentek egymással?
- Mire jó a RecyclerView? Mik az előnyei?
- Milyen adattárolási lehetőségeket ismer Android platformon?
- Mire használható a SharedPreferences?
- Hogyan támogatja az Android az adatok adatbázisban tárolását?
- Sorolja fel az ORM alapelveit!
- Milyen lehetőségek vannak Androidon a fájlrendszerben való adattárolásra?

Összefoglalás

- Grafikai erőforrások
- Animációk
- Fragmentek
- Navigation Component
- Listakezelés: RecyclerView
- Perzisztens adattárolási lehetőségek
- Egyszerű kulcs-érték tár: SharedPreferences
- Adatbázistámogatás, SQLite
- ORM megoldások
- Room használata a gyakorlatban
- Filekezelés
- Adattárolás a felhőben

