

Android

Perzisztens Adattárolás

Ekler Péter
peter.ekler@aut.bme.hu



Department of
Automation and
Applied Informatics

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:
 - > **SQLite adatbázis:** strukturált adatok tárolására
 - > ***SharedPreferences*:** alaptípusok tárolása kulcs-érték párokban
 - > **Privát lemezterület:** nem publikus adatok tárolása a fájlrendszerben
 - > **SD kártya:** nagy méretű adatok tárolása, nyilvánosan hozzáférhető
 - > **Hálózat:** saját webszerveren vagy felhőben tárolt adatok

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 osztálykönyvtár:
 - > http://ormlite.com/sqlite_java_android_orm.shtml
- Mivel SQL, érdemes minden táblában elsődleges kulcsot definiálni
 - > autoincrement támogatás
 - > Ahhoz, hogy *ContentProvider*-rel ki tudjuk ajánlani (később), illetve UI elemeket Adapterrel feltölteni (pl. list, grid), **kötelező egy ilyen oszlop**, melynek neve: „_id”

```
Class Person {  
    String name;  
    String address;  
    Int age;  
  
}
```

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 adatbeíráskor, tehát pl Integer érték automatikusan bekerül Text oszlopba szöveggént

Android SQLite jellemzői 2/2

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

SQLite debug

- Az Android SDK „platform-tools” mappájában található egy konzolos adatbázis kezelő: `sqlite3`
- Ennek segítségével futás közben láthatjuk az adatbázist, akár emulátoron, akár telefonon
- Hasznos eszköz, de sajnos nincs grafikus felülete
- Használata (emulátoron, vagy root-olt eszközön):
 - > Konzolban megnyitjuk a **platform-tools** könyvtárat
 - > „adb shell” futtatása, egy eszköz legyen csatlakoztatva
 - > „**sqlite3 data/data/[Package név]/databases/[DB neve]**” futtatása
 - > Megkapjuk az SQLite konzolt, itt már az adatbázison futtathatunk közvetlen parancsokat (Pl. „dump orak;”)

OBJECT RELATION MAPPING (ORM)

Mi az ORM?

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

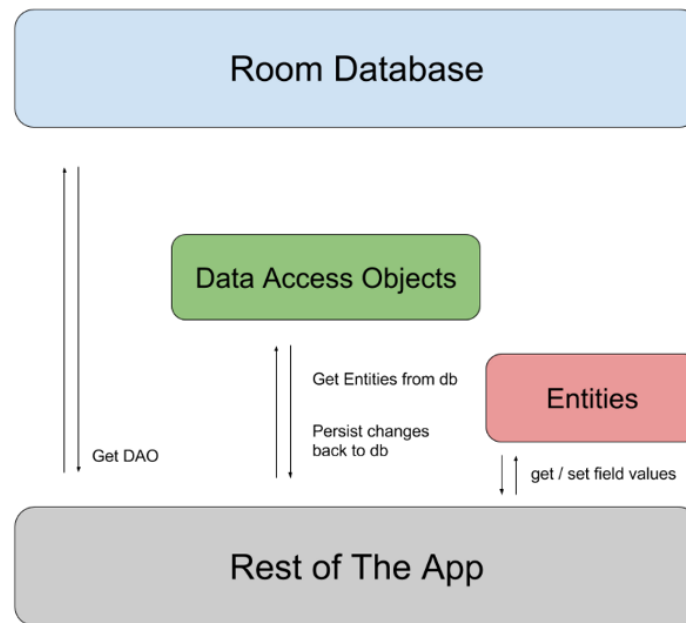


ORM könyvtárak Androidon

- Sugar-ORM
 - > <http://satyan.github.io/sugar/index.html>
- Realm.io (NoSQL), nem SQLite-ot használ
 - > <http://realm.io>
- Objectbox
 - > <https://objectbox.io/>
- ORMLite
 - > <http://ormlite.com/>
- GreenDAO
 - > <http://greendao-orm.com/>

Room Persistence Library

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



Szálkezelés

- *Thread*
 - > <https://developer.android.com/guide/components/processes-and-threads.html>
- A felhasználói felület csak a fő szálról módosítható:
 - > *runOnUiThread(runnable: Runnable)*
- Szálakat le kell állítani
 - > Biztosítani kell, hogy a *run()* függvény befejeződjön, ne maradjon végtelen ciklusban

Szál példa

```
private inner class MyThread : Thread() {  
    override fun run() {  
        while (threadEnabled) {  
            runOnUiThread {  
                Toast.makeText(this@MainActivity,  
                    "Message", Toast.LENGTH_LONG).show()  
            }  
            Thread.sleep(6000)  
        }  
    }  
}
```

```
MyThread().start()
```


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 = arrayOf(Grade::class), version = 1)
abstract class AppDatabase : RoomDatabase() {

    abstract fun gradeDao(): GradeDAO

    companion object {
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            if (INSTANCE == null) {
                INSTANCE = Room.databaseBuilder(context.applicationContext,
                    AppDatabase::class.java, "grade.db").build()
            }
            return INSTANCE!!
        }

        fun destroyInstance() {
            INSTANCE = null
        }
    }
}
```

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()
```

Gyakoroljunk!

- Egészítsük ki a Todo alkalmazást adatbázis támogatással

SharedPreferences

Beállítások mentése hosszú távra

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
 - > Miért?

SharedPreferences

- Ideális olyan adatok tárolására, melyek primitív típussal 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(name: String, mode: Int)**
 - > Ha még nem létezik ilyen nevű, akkor az Android létrehozza
- Ha elég egy SP egy Activity-hez, akkor nem kötelező elnevezni
 - > **getPreferences(mode: Int)**

SharedPreferences írás

- Közvetlenül nem írható, csak egy *Editor* objektumon keresztül

```
val PREF_NAME: String = "MySettings"
val sp: SharedPreferences =
    getSharedPreferences(PREF_NAME, MODE_PRIVATE)
    editor: Editor = sp.edit()
    editor.putLong("lastSyncTimestamp",
        Calendar.getInstance().getTimeInMillis())
    editor.putBoolean("KEY_FIRST", false)
    editor.apply()
```

Azonosító (fájlnév)

Csak mi érjük el

Érték
típusa

Megnyitjuk írásra

Kulcs

Érték

Változtatások
mentése (kötelező!!!)

SharedPreferences olvasás

- Az *Editor* osztály nélkül olvasható, közvetlenül a SharedPreferences objektumból
- Ismernünk kell a kulcsok neveit és az értékek típusát
 - > Emiatt sem alkalmas nagy mennyiségű adat tárolására

```
PREF_NAME = "MySettings"
```

```
val sp =
```

```
    getSharedPreferences(PREF_NAME, MODE_PRIVATE)
```

```
val lastSaved: Long = sp.getLong("lastSaved", 0)
```

```
val isFirstRun: Boolean =
```

```
    sp.getBoolean("KEY_FIRST", true)
```

Tudni kell a kulcsot

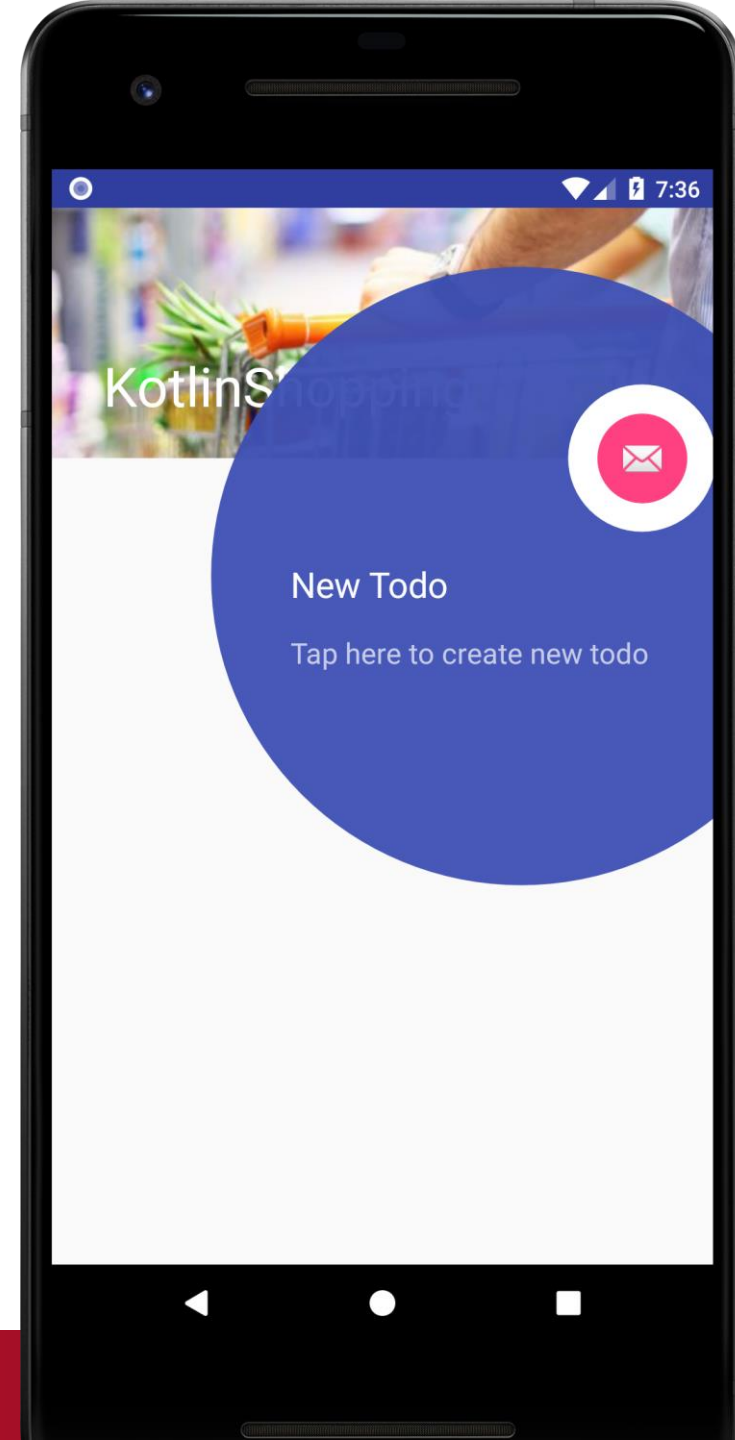
És a típust is!

Alapértelmezett
érték

- Egy hasznos metódus:
 - > **sp.getAll()** – minden kulcs-érték pár egy Map objektumban
 - > Tutorial lib: <https://github.com/sjwall/MaterialTapTargetPrompt>

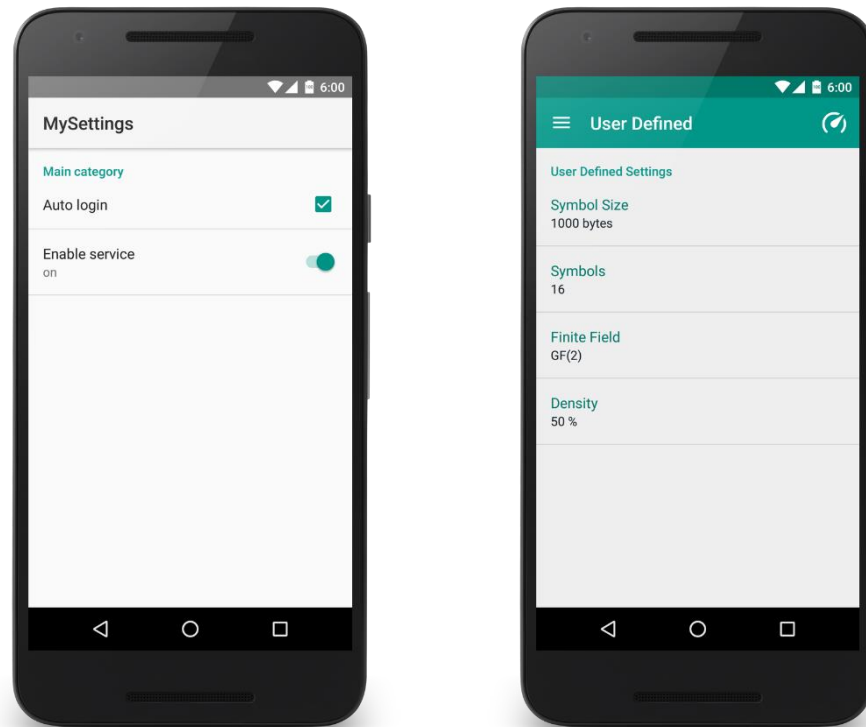
Gyakoroljunk!

Egészítsük ki a bevásárló lista alkalmazást, hogy csak legelső induláskor mutasson használati tippeket.



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



Futási idejű engedélyek

Mikor van rá szükség?

- Felhasználót „veszélyeztető” műveletek
- Engedély kérés régebben:

- > Manifest engedélyek:

```
<uses-permission android:name=  
    "android.permission.WRITE_EXTERNAL_STORAGE"/>
```

- Új Permission modell Android 6 óta:
 - > Veszélyes engedélyeket futási időben kell kérni

Engedély ellenőrzése

- Check permission:

```
ContextCompat.checkSelfPermission(thisActivity,  
    Manifest.permission.WRITE_CALENDAR)
```

- Engedély kérés Activity-ből:

- > Ellenőrizni, hogy megvan-e már az engedélye
- > Felhasználó tájékoztatása az engedély kérés okáról

Engedély típusok

- Típusok
 - > Normal permissions
 - > Dangerous permissions
 - > <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>
- A felhasználó visszavonhatja az engedélyeket a beállításokban bármikor
- További részletek:
 - > <https://developer.android.com/training/permissions/requesting.html>

Permission kérés 1/2

```
private fun requestNeededPermission() {  
    if (ContextCompat.checkSelfPermission(this,  
        android.Manifest.permission.CAMERA) !=  
        PackageManager.PERMISSION_GRANTED) {  
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,  
            android.Manifest.permission.CAMERA)) {  
            Toast.makeText(this,  
                "I need it for camera", Toast.LENGTH_SHORT).show()  
        }  
  
        ActivityCompat.requestPermissions(this,  
            arrayOf(android.Manifest.permission.CAMERA),  
            PERMISSION_REQUEST_CODE)  
    } else {  
        // már van engedély  
    }  
}
```

Permission kérés 2/2


```
override fun onRequestPermissionsResult(requestCode: Int,
    permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        PERMISSION_REQUEST_CODE -> {
            if (grantResults.isNotEmpty() && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "CAMERA perm granted",
                    Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "CAMERA perm NOT granted",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

Fájlkezelés - Internal storage

Internal storage

- Alkalmazás saját, védett lemezterülete
- `mnt/sdcard/data/data/[Package name]` könyvtár
- Fájl írása:

```
val FILENAME = "hello_file.txt"
val content = "Hello World!"
val out: FileOutputStream =
    openFileOutput(FILENAME, Context.MODE_PRIVATE)
out.write(content.getBytes())
out.close()
```



Internal Storage
könyvtárba ír

- `openFileOutput()`-tal csak a könyvtár gyökerébe tudunk fájlokat írni

Internal storage

- `openFileOutput(filename: String, mode: Int)`
 - > filename-ben nem lehet „\”, egyébként kivételt dob (Miért?)
 - > 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(filename: String)`** 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 (miért `File`?)
 - > 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 (Miért?)

Statikus fájlok egy alkalmazáshoz

- Szükséges lehet a fejlesztett alkalmazáshoz statikusan fájlokat linkelni
 - > Kezdeti, nagy méretű, feltöltött bináris adatbázis fájl
 - > Egyedi formátumú állomány
 - > Bármilyen 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)
```

Statikus fájlok egy alkalmazáshoz

- Mivel ugyanolyan resource mint az összes többi, különböző fájlok használhatók különböző konfigurációkhoz, mint például a UI finomhangolásnál:
 - > **res/layout**: felhasználói felületek alapértelmezett orientáció esetén (telefon: álló, tablet és Google TV: fekvő)
 - > **res/layout-port**: felhasználói felületek álló orientáció esetén
 - > **res/layout-land**: felhasználói felületek fekvő orientáció esetén
- Ugyanúgy lehet ezt is finomhangolni:
 - > **res/raw/initialDatabase.db**: kezdeti adatbázis
 - > **res/raw-hu_rHU/initialDatabase.db**: kezdeti adatbázis, ha a telefon magyar nyelvre van állítva
 - > **res/raw-hu_rHU-long-trackball/initialDatabase.db**: kezdeti adatbázis, ha a telefon magyar nyelv van állítva, a kijelző szélesvásznú és van trackball

Néhány hasznos metódus

- **getFilesDir()**
 - > Visszaadja az alkalmazás védett tárterületére mutató fájl objektumot (data/data/[Package név])
- **getDir()**
 - > Létrehoz vagy megnyit egy könyvárat az intenal storage-en belül
- **deleteFile()**
 - > Fájlt töröl az internal storage könyvárban
- **fileList()**
 - > Egy String tömbben visszaadja az internal storage-ben lévő fájlok neveit

Fájlkezelés - External storage

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***

Nyilvános lemezterület

Android 2.2 alatt

- External storage elérése: `getExternalStorageDirectory()`
- Ez a gyökerre ad referenciát
- Innen az `Android/data/[Package név]/files` könyvtárat használjuk
- Nincsenek konstansok a média típusokhoz, tudnunk kell hogy melyik default könyvtárnak mi a neve, és „kézzel” kell beleraknunk a média fájlokat
 - > `PL.DIRECTORY_MUSIC = „Music/”`
- A 2.1-es verzió még nem halt ki teljesen, érdemes felkészítenünk az alkalmazásunkat rá! (2.2 alatt jelenleg az eszközök 2 százaléka)

File írás (Java módra)

```
private fun writeFile(data: String) {  
    val file = File( "text.txt")  
  
    var outputStream: FileOutputStream? = null  
    try {  
        outputStream = FileOutputStream(file)  
        outputStream.write(data.toByteArray())  
        outputStream.flush()  
    } catch (e: IOException) {  
        e.printStackTrace()  
    } finally {  
        if (outputStream != null) {  
            try {  
                outputStream.close()  
            } catch (e: IOException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```


File írás (röviden)

```
private fun writeFile(data: String) {  
    val file = File(Environment.getExternalStorageDirectory(),  
                    "text.txt")  
  
    try {  
        file.writeText(data)  
    } catch (e: IOException) {  
        e.printStackTrace()  
    }  
}
```

File olvasás (Java módra)

```
private fun readFile(): String? {  
    val file = File(Environment.getExternalStorageDirectory(), "text.txt")  
  
    var reader: BufferedReader? = null  
    try {  
        reader = BufferedReader(InputStreamReader(FileInputStream(file)))  
        val builder = StringBuilder()  
        while (true) {  
            val line: String? = reader.readLine()  
            if (line != null) {  
                builder.append(line)  
                builder.append("\n")  
            } else {  
                return builder.toString()  
            }  
        }  
    } catch (e: IOException) {  
        e.printStackTrace()  
    } finally {  
        if (reader != null) {  
            try {  
                reader.close()  
            } catch (e: IOException) {  
                e.printStackTrace()  
            }  
        }  
    }  
    return null  
}
```

File olvasás (röviden)

```
private fun readFile(): String? {  
    val file = File(Environment.getExternalStorageDirectory(),  
                     "text.txt")  
  
    try {  
        return file.readText()  
    } catch (e: IOException) {  
        e.printStackTrace()  
        return null  
    }  
}
```

Összefoglalás

- Perzisztens adattárolási lehetőségek
- Adatbázistámogatás, SQLite
- ORM megoldások
- Room használata a gyakorlatban
 - > Komplex alkalmazás megvalósítása listakezeléssel és adatbázis-támogatással
- Egyszerű kulcs-érték tár: SharedPreferences
- File kezelés
- Futási idejű engedélyek

Kérdések

