Android

BroadcastReceiver, Perzisztens Adattárolás

Dr. Ekler Péter
peter.ekler@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

Tartalom

- BroadcastReceiver
- Futás idejű engedélyek
- Perzisztens adattárolás
  - SharedPreferences
  - SQLite
  - ORM - Room

BroadcastReceiver komponens

Broadcast események

- Rendszerszintű eseményekre fel lehet iratkozni – Broadcast üzenet
- Az Intent alkalmas arra hogy leírja az eseményt
- Sok beépített Broadcast Intent, lehet egyedi is

```
ACTION_TIME_TICK
ACTION_TIME_CHANGED
ACTION_TIMEZONE_CHANGED
ACTION_BOOT_COMPLETED
ACTION_PACKAGE_ADDED
ACTION_PACKAGE_CHANGED
ACTION_PACKAGE_REMOVED
ACTION_PACKAGE_RESTARTED
ACTION_PACKAGE_DATA_CLEARED
```

```
ACTION_UID_REMOVED
ACTION_BATTERY_CHANGED
ACTION_POWER_CONNECTED
ACTION_POWER_DISCONNECTED
ACTION_SHUTDOWN
```

Broadcast események

- Nem csak az Android, hanem alkalmazások (Activity-k és Service-ek) is dobhatnak Broadcast Intentet
  - Telephony service küldi az ACTION_PHONE_STATE_CHANGED Broadcast Intentet, ha a mobilhálózat csatlakozás megváltozott
  - android.provider.Telephony.SMS_RECEIVED
  - Sok más, érdemes tájékozódni ha valamit szeretnénk lekezelni
  - Saját alkalmazásunkból is dobhatunk a **`sendBroadcast(String action)`** metódussal
  - Ez is Intent, lehet Extra és Data része

Broadcast intentek elkapása

- Broadcast Receiver nevű komponens segítségével
  - Kódból vagy manifestben kell regisztrálni
    - (bizonyos Action-ök esetén nem mindegy, tájékozódni!, pl. TIME_TICK)
  - Intent filterrel állíthatjuk be hogy milyen Intent esetén aktivizálódjon
- Nem Activity, nincs felhasználói felülete
- Azonban képes Activity-t indítani
- Használata: BroadcastReceiver osztályból származtatunk, és felüldefiniáljuk az onReceive() metódust, majd intent-filter

Broadcast intentek kezelése

```
class OutgoingCallReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val outNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER)
        Toast.makeText(context, outNumber, Toast.LENGTH_LONG).show()

    }
}
```

AndroidManifest.xml:

```
<receiver android:name=".OutgoingCallReceiver">
    <intent-filter>
        <action android:name=
          "android.intent.action.NEW_OUTGOING_CALL"/>
    </intent-filter>
</receiver>
```

Broadcast intentek kezelése

Broadcast továbbdobásának megakadályozása:

- abortBroadcast()

Például ha a fülhallgató média gombjait kell kezelni és nem akarjuk, hogy a zenelejátszó is megkapja a Broadcast-ot ☺

Gyakoroljunk!

- Készítsünk egy AirPlane mód változásra figyelő BroadcastReceivert!
- Készítsünk egy kimenő hívásra figyelő BroadcastReceivert!
  - Változtassuk meg a hívott számot!
  - Egészítsük ki SMS figyeléssel!
    - Valósítsuk meg, hogy ne kerüljön be inbox-ba a bejövő SMS

SMS Receiver 1/2

- Manifest:

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

...

<receiver android:name=".SMSReceiver"  android:enabled="true">

    <intent-filter android:priority="1000">

        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>

</receiver>
```

SMS Receiver 2/2

```
class SMSReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val extras = intent.extras ?: return
        val pdus = extras.get("pdus") as Array<ByteArray>
        for (pdu in pdus) {
            val msg = SmsMessage.createFromPdu(pdu)
            val origin = msg.originatingAddress
            val body = msg.messageBody
            Toast.makeText(context,
                  "SMS caught, number: $origin body: $body", Toast.LENGTH_LONG)
                  .show()
        }
    }
}
```

Alkalmazáskomponens indítása Boot után

- Néha olyan szolgáltatásokra van szükség, amelyek mindig futnak a készüléken
- Ilyen esetben fontos, hogy a készülék indítása esetén ezek automatikusan is el tudjanak indulni
- Az Android lehetőséget biztosít arra, hogy feliratkozzunk a „*Boot befejeződött*” eseményre és valamilyen alkalmazás komponenst elindítsunk:
  - *BroadcastReceiver* definiálása Manifest-ben
  - *android.intent.action.BOOT_COMPLETED*
- *A BroadcastReceiver onReceive()* függvényében elindíthatjuk a megfelelő komponenst

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

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
- Mit csinálhatunk a rendszerszintű események bekövetkezésekor?
  - > Bármit

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()
            }
        }
    }
}
```

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

SharedPreferences

Beállítások mentése hosszú távra

SharedPreferences

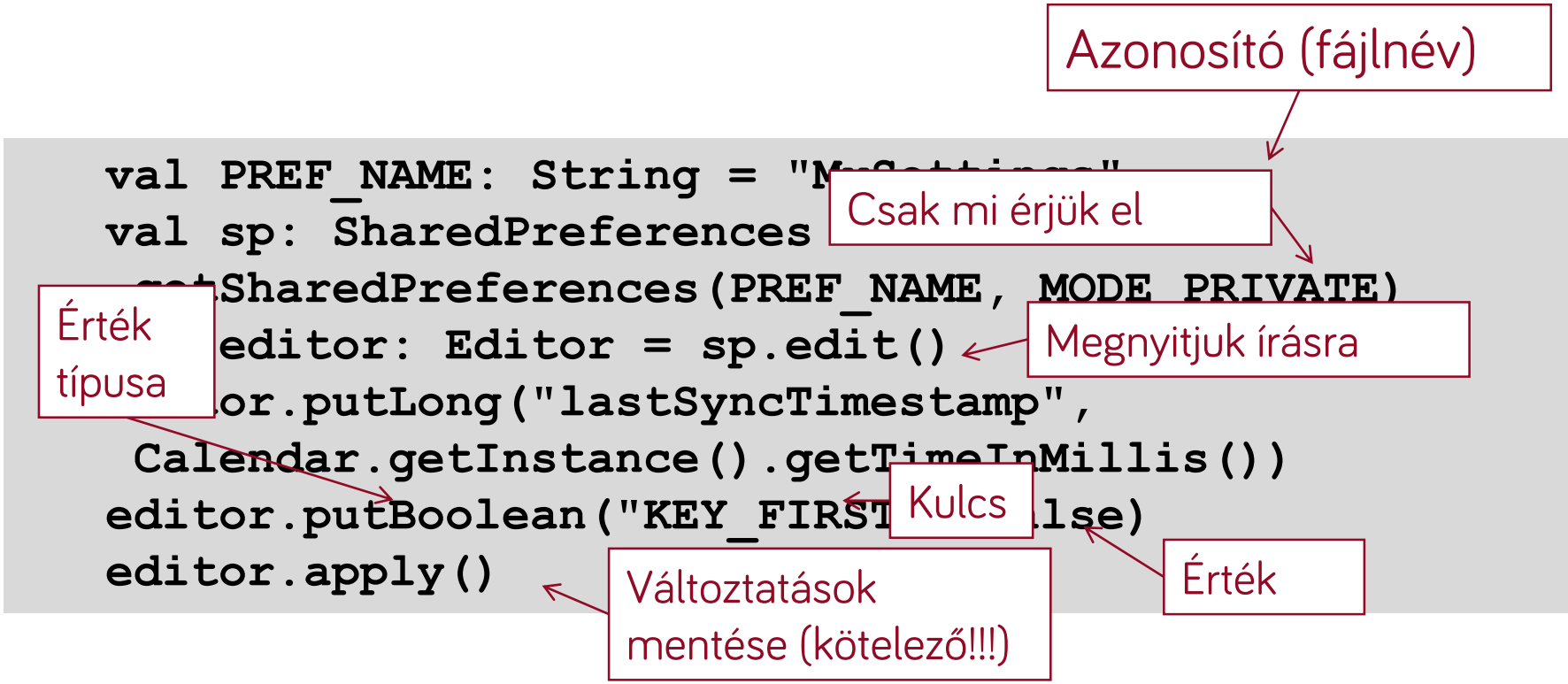
- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
  - Típusok: *int, long, float, String, boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
  - ***MODE_PRIVATE***: csak a saját alkalmazásunk érheti el
  - ***MODE_WORLD_READABLE***: 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 = "M
val sp: SharedPreferences
 etSharedPreferences(PREF_NAME, MODE_PRIVATE)
    editor: Editor = sp.edit()
  or.putLong("lastSyncTimestamp",
  Calendar.getInstance().getTimeInMillis())
editor.putBoolean("KEY_FIRST  lse)
editor.apply()
```

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

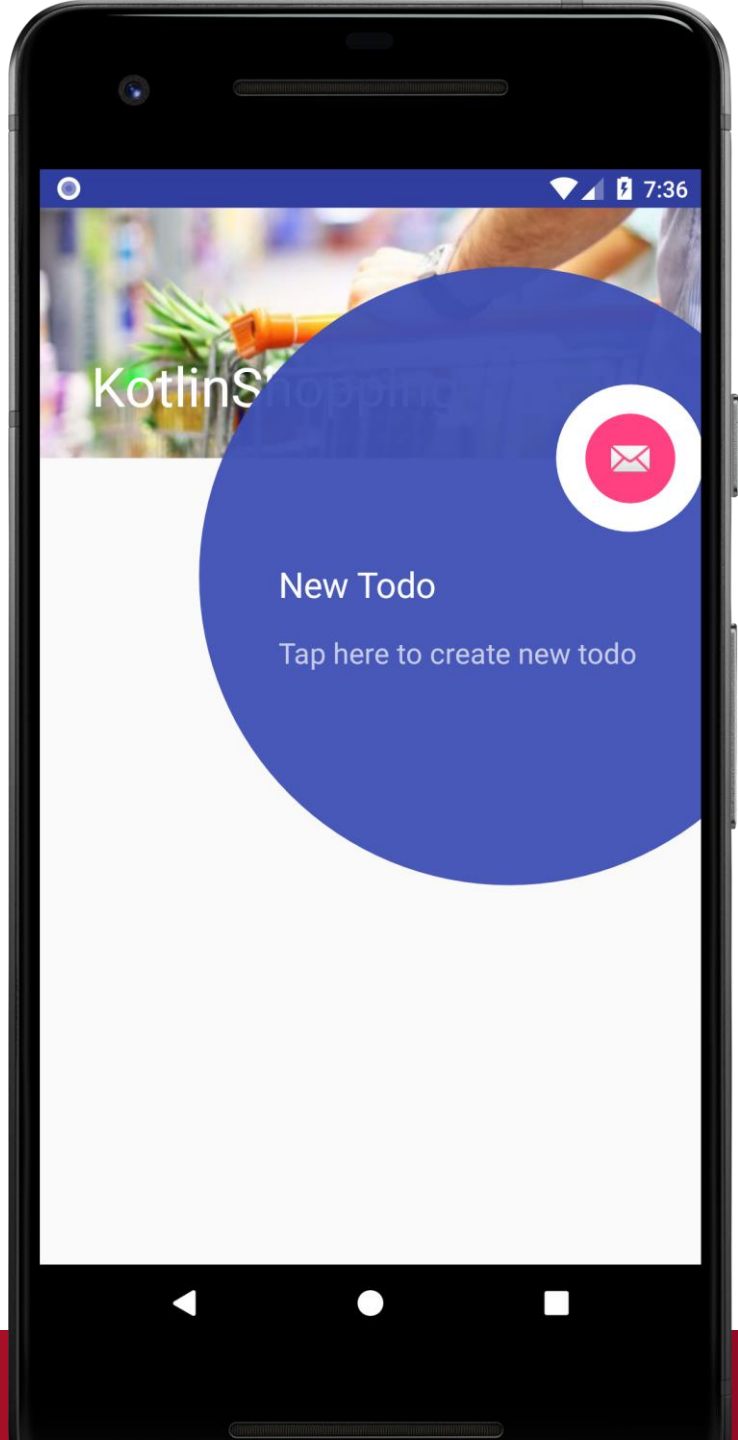
És a típust is!

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.

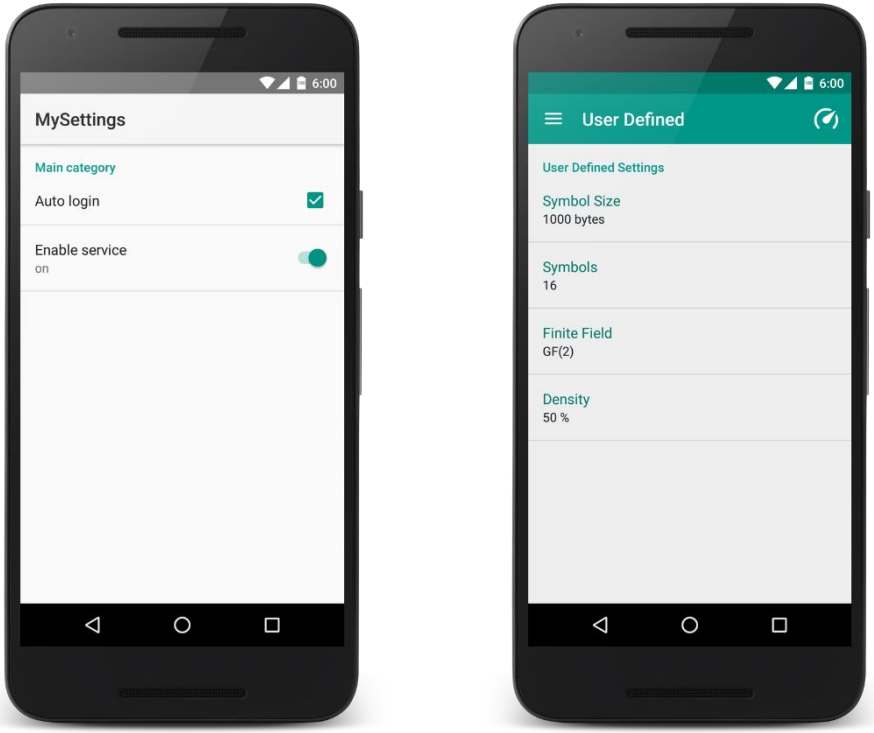


Beállítások képernyő készítéséhez

PREFERENCES FRAMEWORK

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



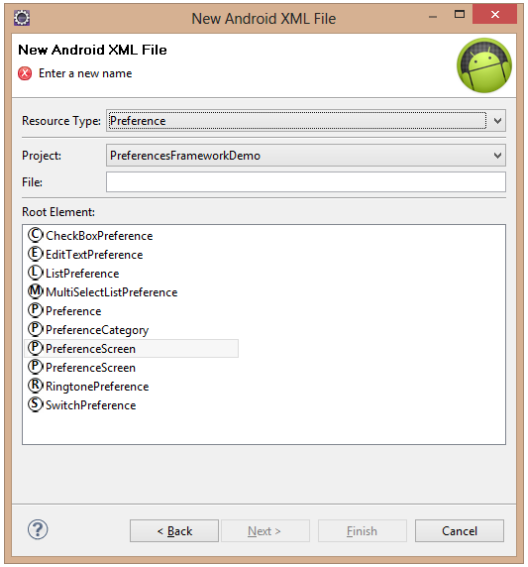
Preferences Framework

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

Preferences Framework - XML

- Ez nem *layout*, hanem azt írja le, hogy miket lehessen beállítani
- *res/xml* könyvtárba kell rakni, a gyökér elem

  kötelezően *PreferenceScreen*
- Azon belül:
  - **Preference**: egy beállítási lehetőség (egy sor)
  - **PreferenceCategory**: csoportosítás
  - **PreferenceScreen**: új képernyő (csak a neve látszik egy sorként, kattintásra új képernyőre ugrik, ahol a node tartalmát jeleníti meg)
- A fenti elemek tetszőlegesen egymásba ágyazhatók



Preferences Framework - XML

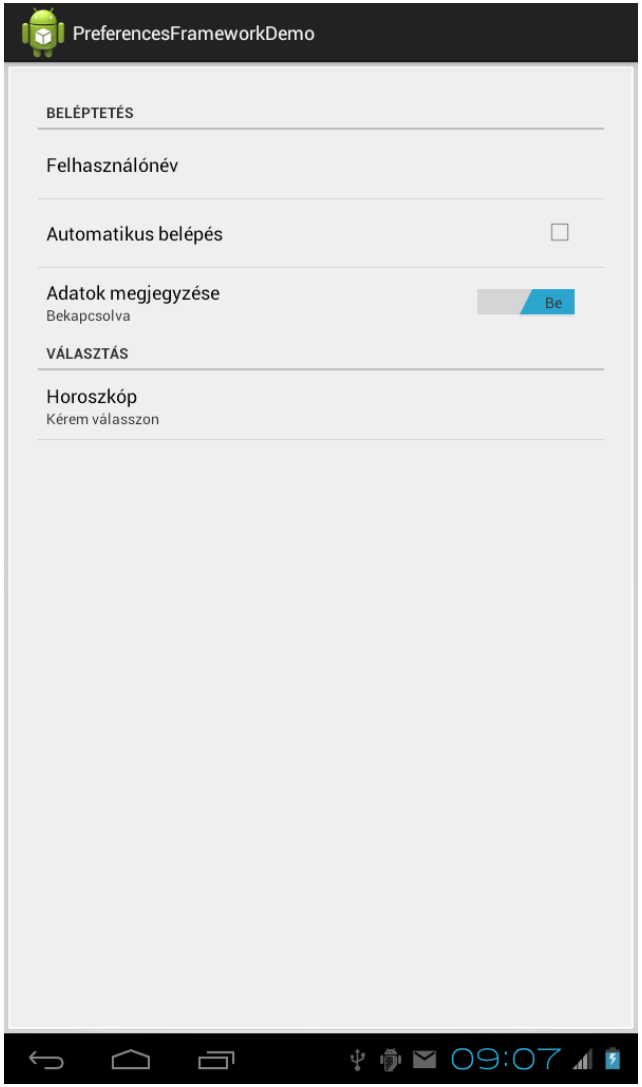
- **Preference** node tartalma
  - **`android:key`** – a SharedPreferences-ben lévő kulcs neve
  - **`android:title`** – a megjelenő UI-on szöveg
  - **`android:summary`** – bővebb leírás, ha szükséges (title alatt jelenik meg)
  - **`android:defaultValue`** – alapértelmezett érték, ez van kiválasztva ha a kulcshoz nincs semmilyen érték a SP-ben
- A megjelenő elem típusát az határozza meg, hogy melyik Preference node-ot használjuk az XML-ben:
  - *CheckboxPreference*, *EditTextPreference*, *ListPreference*, *MultiSelectListPreference*, *RingtonePreference*, *SwitchPreference*
  - Akár sajátot is készíthetünk, ha a Preference osztályból származtatunk

Példa Preferences nézet - XML

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

```
 <PreferenceCategory android:title="Választás" >
        <ListPreference
          android:title="Horoszkóp"
          android:summary="Kérem válasszon"
          android:key="listPref"
          android:entries="@array/listDisplayMarks"
          android:entryValues="@array/listReturnMarks"
/>
    </PreferenceCategory>
</PreferenceScreen>
```

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>
```

Beállítások kiajánlása

- Kiajánlhatjuk a saját beállítás képernyőinket más alkalmazások számára az *AndroidManifest*-ből

```
<activity android:name=".UserPreferences"
    android:label="Beállítások">
 <intent-filter>
  <action android:name="amorg.orarend.ACTION_USER_PREFERENCE" />
 </intent-filter>
</activity>
```

Preferences Framework - **Activity**

- Activity készítésének lépései:
  1. **PreferenceActivity**-ből származtatunk
  2. onCreate()-ben **`addPreferencesFromResource(R.xml.prefs);`**
  3. Activity regisztrálása az **AndroidManifest**-ben
- Egy alkalmazáshoz több SharedPreferences fájl is tartozhat, azonban nem tölthetjük fel bármelyikből a beállítások képernyőt, csak ebből:

```
var myPrefs:SharedPreferences =
    PreferenceManager.getDefaultSharedPreferences(
        applicationContext);
```

SharedPreferences megváltozása

- Az alkalmazásunknak reagálnia kell a beállítások változására
  - Pl. skin átállítása, súgószövegek megjelenítésének kikapcsolása után le kell venni azokat a UI-ról, felhasználói fiókot váltott a user, stb…
- **Eseménykezelőt** állíthatunk be, ami meghívódik ha valamelyik beállítás változik
- Megvalósítása:
  - Az Activity (amelyiknek reagálnia kell) implementálja az *OnSharedPreferenceChangeListener* interfészt
  - onCreate-ben ***registerOnSharedPreferenceChangeListener(this)*** hívás
  - ***onSharedPreferenceChanged()*** metódus felüldefiniálása

SharedPreferences megváltozása

```
class MySettings : PreferenceActivity(),
  SharedPreferences.OnSharedPreferenceChangeListener {

    override fun onCreate(savedInstanceState: Bundle
        super.onCreate(savedInstanceState)
    }

    override fun onStart() {
        super.onStart()

        PreferenceManager.getDefaultSharedPreferences(this).
          registerOnSharedPreferenceChangeListener(this)
    }

    override fun onStop() {
        super.onStop()
        PreferenceMana    ultSharedPreferences(this).
          unregisterOn    enceChangeListener(this)
    }

    override fun onSharedPreferenceChanged(
            sharedPreferences: SharedPreferences, key: String) {
        Toast.makeText(this, key, Toast.LENGTH_LONG).show()
    }
}
```

Az Acitivity megvalósítja a megfelelő interfészt

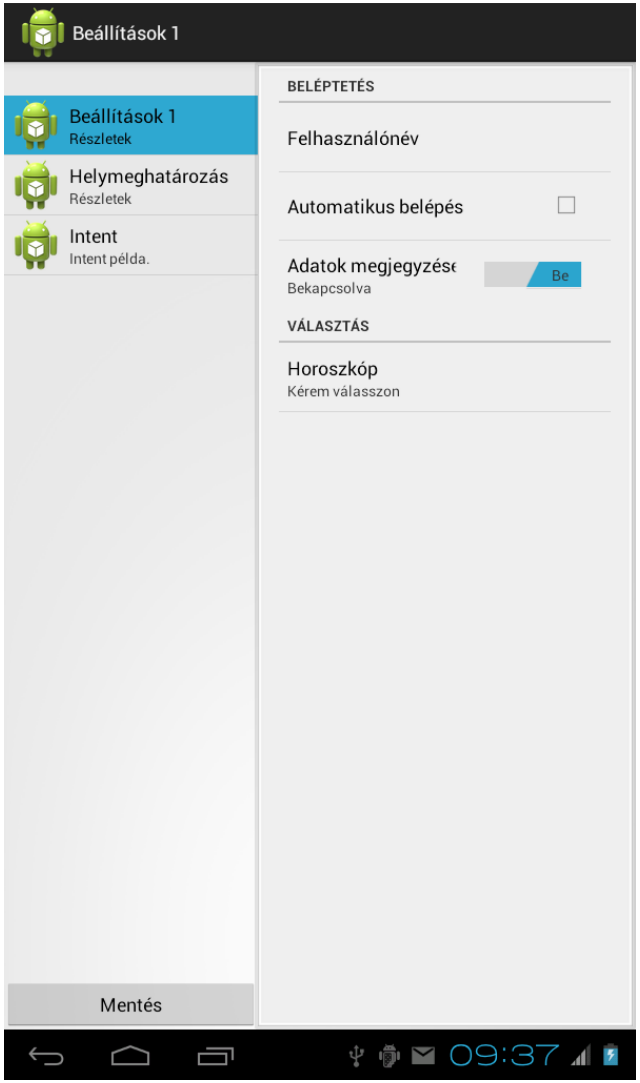
Beregisztráljuk saját magunkat Listener-ként

Értesítés változásról

Fragment alapú beállítások nézet

- Android 3.0-tól fejlettebb funkciók
- Beállítás header-ek (címkék) definiálása
- Minden címkéhez külön Fragment tartozhat
- Kis képernyőn a címkék egy listában jelennek meg
- Nagy képernyőn bal oldalt a címkék listája, jobb oldalt pedig a kiválasztott címkéhez tartozó Fragment jelenik meg

Fragment alapú beállítások példa 2/3



Fragment alapú beállítások példa 2/3

```
public class ActivityFragmentSettings
 extends PreferenceActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (hasHeaders()) {
      Button button = new Button(this);
      button.setText("Mentés");
      setListFooter(button);
    }
  }
  @Override
  public void onBuildHeaders(List<Header> target) {
    loadHeadersFromResource(
      R.xml.fragmentsettings, target);
  }
```

Header-ek betöltése

```
  public static class FragmentSettings1
   extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.mainsettings);
    }
  }

  public static class FragmentSettings2
   extends PreferenceFragment {
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     addPreferencesFromResource(R.xml.locsettings);
    }
  }
}
```

Fragment-hez tartozó beállítások nézet betöltése

Fragment alapú beállítások példa 2/3

```
class ActivityFragmentSettings : PreferenceActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        if (hasHeaders()) {
            val button = Button(this)
            button.setText("Mentés")
            setListFooter(button)
        }
    }

    override fun onBuildHeaders(target: List<PreferenceActivity.Header>) {
        loadHeadersFromResource(
                R.xml.fragmentsettings, target)
    }

    class FragmentSettings1 : PreferenceFragment() {
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            addPreferencesFromResource(R.xml.mainsettings)
        }
    }

    class FragmentSettings2 : PreferenceFragment() {
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            addPreferencesFromResource(R.xml.locsettings)
        }
    }
}
```

Header-ek betöltése

Fragment-hez tartozó beállítások nézet betöltése

Fragment alapú beállítások példa 3/3

```
<preference-headers
        xmlns:android="http://schemas.android.com/apk/res/android">
    <header android:fragment=

"hu.bute.daai.amorg.examples.preferencesframeworkdemo.ActivityFragmentSettings$Fragment
Settings1"
            android:icon="@drawable/ic_launcher"
            android:title="Beállítások 1"
            android:summary="Részletek">
        <!-- További kulcs/érték párok megadhatók a fragment argume
        <extra android:name="extraKey" android:value="extraValue" /
    </header>
    <header android:fragment=

"hu.bute.daai.amorg.examples.preferencesframeworkdemo.ActivityFragmentSettings$Fragment
Settings2"
            android:icon="@drawable/ic_launcher"
            android:title="Helymeghatározás"
            android:summary="Részletek">
    </header>

    <header android:icon="@drawable/ic_launcher"
            android:title="Intent"
            android:summary="Intent példa.">
        <intent android:action="android.intent.action.VIEW"
                android:data="http://www.aut.bme.hu" />
    </header>
</preference-headers>
```

Hivatkozás a Fragment-ekre

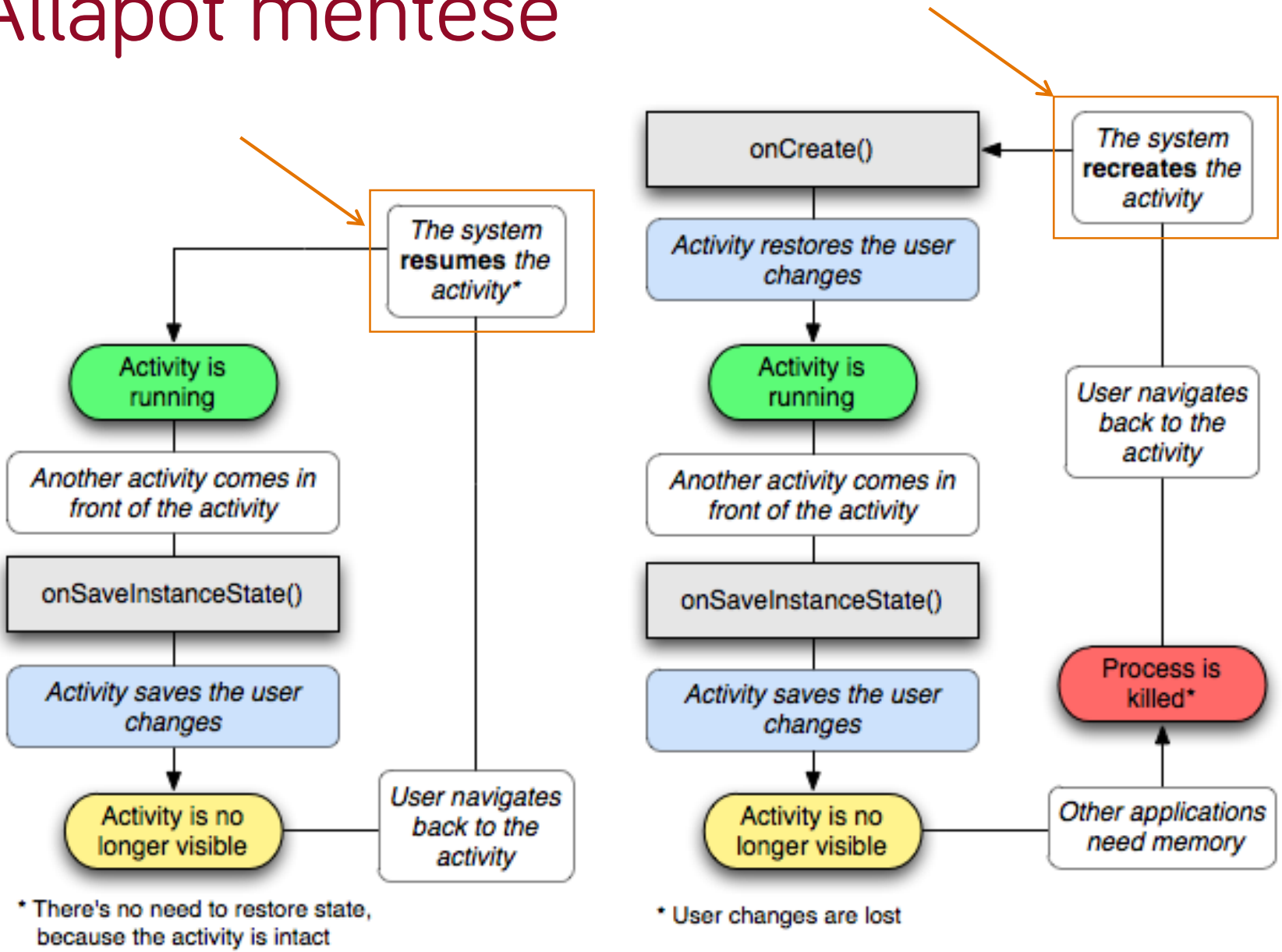
Állapotmentés

Activity visszaállításához szükséges adatok mentése

Állapot mentése

- Ha egy Activity *Paused* (részben takarja valami) vagy *Stopped* (egyáltalán nem látszik) állapotban van, akkor a memóriában tárolódik minden hozzá tartozó adat
- Folytatódáskor minden visszaáll „magától”, mert még ugyanaz a processz fut
- Azonban ha kevés memória miatt az Android felszabadította a területét, akkor a **processz meghal**, és vele együtt **minden elvész a memóriaterületről** (osztály tagváltozói, UI állapot információk, stb...)
- Újra előtérbe kerüléskor az *onCreate()* hívódik, gyakorlatilag újraindul az Activity
- **Képernyő elforgatáskor** is megsemmisül és újraépül az Activity!
- **A fejlesztőnek kell gondoskodnia az állapot mentéséről! – onSaveInstanceState()**

Állapot mentése



Állapot mentése

- Ezért kap az *onCreate()* paraméterként egy *savedInstaceState* nevű Bundle-t

```
@Override
public void onCreate(Bundle savedInstanceState) {...}
```

- Itt kell újra beállítani a változókat és a UI elemeket a kapott ***Bundle*** alapján
- Amit az ***onSaveInstanceState()*** -ben töltöttünk fel
- Ez az egész mechanizmus láthatatlan a felhasználó számára
  - Ha nem csináljuk meg, akkor csak azt veszi észre, hogy néha máshogy tér vissza az alkalmazás mint ahogy otthagyta
  - Emiatt **alapkövetelmény hogy a fejlesztő lekezelje az állapotmentést**, és mindig úgy folytatódjon az app, ahogy háttérbe került!

onSaveInstanceState()

- Az ***onSaveInstanceState()*** lefutása nem egyértelmű
  - Nem biztos hogy egyáltalán lefut, a Vissza gomb megnyomása esetén például az Android feltételezi, hogy a felhasználó ki akart lépni, így nincs szükség állapotmentésre (*onBackPressed()*-ben felülírható)
  - Nem meghatározott, hogy az *onPause()* előtt vagy után fut le (az biztos, hogy az *onStop()* előtt)
- Emiatt rossz gyakorlat perzisztens adat mentéséről (pl. adatbázisba, fájlba írás) gondoskodni az *onSaveInstanceState()*-ben! Helyette az *onPause()*-ban intézzük el, az biztosan lefut minden esetben
- A képernyő elforgatásakor az Activity elpusztul és újra létrejön (mert újra kell építenie a UI-t), ezzel jól tesztelhető az állapotmentés
  - A user szereti forgatni a telefont, emiatt is erősen ajánlott a helyes lekezelése!

onSaveInstanceState()

- Az onCreate()-ben hívott **`super.onCreate(savedInstanceState)`** magától visszaállít néhány dolgot, de nem mindent! Amit igen:
  - Majdnem minden UI Widget alapból visszáll
    - Pl. EditText, Checkbox, stb
    - **DE**: a Spinner például nem őrzi meg, hogy mi volt kiválasztva!
    - Nincs dokumentálva minden, ha kritikus akkor ki kell próbálni!
    - Csak akkor, ha van id-ja! (`android:id`)
  - Ezen kívül semmi mást nem állít vissza magától
  - UI elemre beállíthatjuk, hogy ne állítsa vissza az állapotát – **`android:saveEnabled="false"`**

Állapot mentése - HowTo

1. *onSaveInstanceState()*-ben a paraméterként kapott Bundle-be beleírunk mindent, ami a UI-on megjelenik, vagy egyéb okból szükséges visszaállítani
   - Listák aktuálisan kiválasztott elemei
   - Tagváltozók, amik futás közben változhattak
   - Stb…

2. *onCreate()*-ben visszatöltjük az értékeket, ha a kapott savedInstanceState nem null (akkor null, ha nem volt állapotmentés)
   - Ha nem az *onCreate()*-et akarjuk telenyomni a visszaállító kóddal, akkor tegyük az *onRestoreInstanceState()*-be, ami az *onCreate()* után hívódik

Állapotmentés Példa

```
class MainActivity : Activity() {
    private var string: String? = null
    private val KEY_STRING = "str"

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        string = "A tagváltozó kezdeti értéke változhat a futás során!"
        if (savedInstanceState != null) {
            if (savedInstanceState.containsKey(KEY_STRING)) {
                string = savedInstanceState.getString(KEY_STRING)
            }
        }
    }

    override fun onSaveInstanceState(outState: Bundle) {
        outState.putString(KEY_STRING, string)
        super.onSaveInstanceState(outState)
    }
}
```

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 szövegként

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?

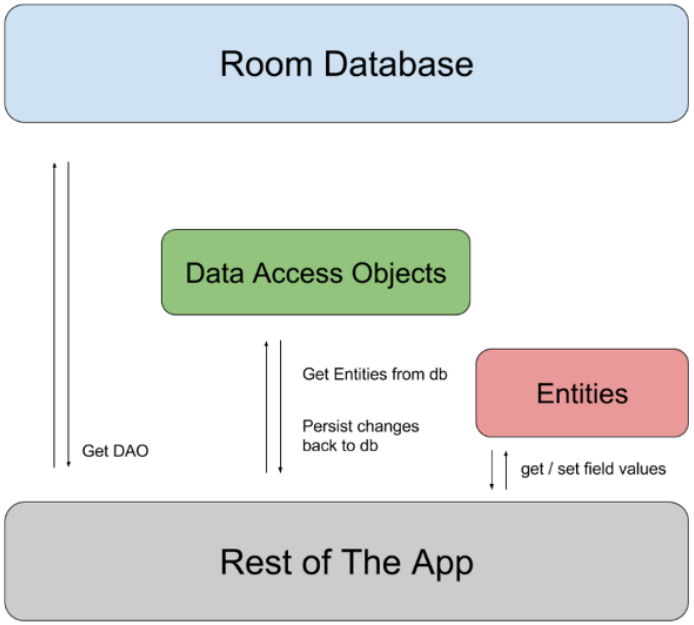
- Java objektumok tárolása relációs adatbázisban
- Alapelvek:
  - Osztálynév -> Tábla név
  - Objektum -> Tábla egy sora
  - Mező -> Tábla oszlopa
  - Stb.

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()
```

Összefoglalás

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

Kérdések