Mobil- és webes szoftverek

Dr. Ekler Péter

ekler.peter@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

ZH és PótZH időpontok

- ZH időpont:
  - 2017. 11. 03. Péntek 14-16
  - QI, IB028 és QBF12-es termek (terembeosztás később)
- PótZH időpont:
- 2017. 11. 28. Kedd 8-10
  - IB028

Miről volt szó az előző órán?

- Grafikai e
- Animáció
- Élő háttér
- Fragment
  - > Statikus
  - > Worker
  - > DialogF

Miről volt szó az előző órán?

- Grafikai erőforrások
- Animációk
- Élő háttérkép, Widget
- Fragmentek
  - Statikus és dinamikus csatolás
  - Worker fragment
  - DialogFragment

Hogy is volt?

- Hogyan tudunk animációkat létrehozni, milyen lehetőségeket biztosít az Android API?
- Mit nevezünk Fragmentnek?
- Milyen Fragment típusokat ismer?
- Hogyan csatolhatók a Fragmentek az Activity-hez?
- Mire szolgál a FragmentTransaction?

Tartalom

- Android engedélyek (Permission modell)
- Perzisztens adattárolási lehetőségek
  - SQLite, ORM
  - SharedPreferences
  - File
- ContentProvider komponens

Android Engedélyek (Permission modell)

Mire szolgálnak az engedélyek?

- Veszélyes/kritikus/személyes adatokat érintő műveletekhez engedélyre van szükség a felhasználótól.
- Korábban:
  - Manifest állományban egyszerű bejegyzés:

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

  - Majd az összes engedély egyszeri elkérése telepítéskor
- Új permission modell 6-os Androidtól felfele:
  - Engedély elkérése futási időben Java kódból

Futási idejű permission kezelés

- Permission meglétének ellenőrzése:

```
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.WRITE_CALENDAR);
```

- Permission kérés Activity-ben:
  - Megvizsgálni, hogy már nem kaptuk-e meg az engedélyt
  - Ha szükséges elmagyarázni a felhasználónak, hogy miért kérjük, majd újra kérni
  - Permission kérés eredményének kezelése egyesével
    - Engedélyezés
    - Tiltás

Permission típusok

- Manifest állományban továbbra is érdemes felsorolni a permission-öket
  - Normál permission-öket automatikusan megkap az alkalmazás
  - Veszélyes permision-öket kérni kell Java kódból
- Normál és veszélyes permission-ök listája:
  - https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous
- Korábban megadott engedélyeket a felhasználó visszavonhatja!
- További részletek:
  - https://developer.android.com/training/permissions/requesting.html

Permission kérés példa 1/2

```
public static final int REQUEST_CODE_LOCATION_PERMISSION = 401;

public void requestNeededPermission() {
    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.ACCESS_FINE_LOCATION)) {
            // aszinkron módon magyarázat megjelenítése dialógusban,
            // majd újra kérés manuálisan
        }

        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_CODE_LOCATION_PERMISSION);
    } else {
        // megkaptuk az engedélyt, indíthatjuk a kívánt műveletet
    }
}
```

Permission kérés példa 2/2

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[],
                                       int[] grantResults) {
    switch (requestCode) {
        case REQUEST_CODE_LOCATION_PERMISSION: {
            // Ha mégsem (vissza gomb) lett választva a
            // kéréskor akkor üres a tömb
            if (grantResults.length > 0
                    && grantResults[0] ==
                      PackageManager.PERMISSION_GRANTED) {

                // Megkaptuk az engedélyt, indítható a művelet
                myLocationManager.startLocationMonitoring();
            } else {
                // Nem kaptuk meg az engedélyt,
                // üzenet jelzése a felhasználónak
            }
            return;
        }
    }
}
```

Külső könyvtár permission kérésre

- https://github.com/hotchemi/PermissionsDispatcher

```
@RuntimePermissions
public class MainActivity extends AppCompatActivity {

    @NeedsPermission(Manifest.permission.CAMERA)
    void showCamera() {
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.sample_content_fragment, CameraPreviewFragment.newInstance())
                .addToBackStack("camera")
                .commitAllowingStateLoss();
    }

    @OnShowRationale(Manifest.permission.CAMERA)
    void showRationaleForCamera(PermissionRequest request) {
        new AlertDialog.Builder(this)
                .setMessage(R.string.permission_camera_rationale)
                .setPositiveButton(R.string.button_allow, (dialog, button) -> request.proceed())
                .setNegativeButton(R.string.button_deny, (dialog, button) -> request.cancel())
                .show();
    }

    @OnPermissionDenied(Manifest.permission.CAMERA)
    void showDeniedForCamera() {
        Toast.makeText(this, R.string.permission_camera_denied, Toast.LENGTH_SHORT).show();
    }

    @OnNeverAskAgain(Manifest.permission.CAMERA)
    void showNeverAskForCamera() {
        Toast.makeText(this, R.string.permission_camera_neverask, Toast.LENGTH_SHORT).show();
    }
}
```

Legjobb gyakorlatok permission kezelésre

- Használjuk lehetőleg a rendszer beépített szolgáltatásait (pl. kamera)!
- Csak a szükséges engedélyeket kérjük el!
- Ne terheljük túl a felhasználót kérésekkel!
  - Csak akkor kérjük el az engedélyt, amikor szükség van rá!
- Magyarázzuk el a felhasználónak, hogy miért van szükség az adott engedélyre!
- Teszteljük mindkét permission kezelést (manifest és Java kód) korábbi és új Android verziókon!

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
  - **Privát lemezterület**: nem publikus adatok tárolása a fájlrendszerben
  - **SD kártya**: nagy méretű adatok tárolása, nyilvánosan hozzáférhető
  - **SQLite adatbázis**: strukturált adatok tárolására
  - **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 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 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 asszinkron módon végrehajtani (pl *AsyncTask* használlata v. *Loader*)

SQLite debug

- Az Android SDK „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;”)

Mi az ORM?

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

ORM könyvtárak Androidon

- Realm.io
  - http://realm.io/
- Sugar-ORM
  - http://satyan.github.io/sugar/index.html
- ORMLite
  - http://ormlite.com/
- GreenDAO
  - http://greendao-orm.com/
- MapDB:
  - http://www.mapdb.org/

SugarORM a gyakorlatban

- Gradle függőség:

```
compile 'com.github.satyan:sugar:1.4'
```

- Manifest-be:

```
<application … android:name="com.orm.SugarApp„/>
<meta-data android:name="DATABASE" android:value="sugar_example.db" />
<meta-data android:name="VERSION" android:value="2" />
<meta-data android:name="QUERY_LOG" android:value="true" />
<meta-data android:name="DOMAIN_PACKAGE_NAME" android:value="com.example" />
```

- Megfelelő package-be (pl. com.example) POJO:

```
public class Book extends SugarRecord<Book> {
  String title;
  String edition;

  public Book(){
  }

  public Book(String title, String edition){
    this.title = title;
    this.edition = edition;
  }
}
```

Forrás: http://satyan.github.io/sugar/getting-started.html

SugarORM használat

- Mentés:
  - Book book = new Book(ctx, "Title here", "2nd edition")
  - book.save();
- Betöltés:
  - Book book = Book.findById(Book.class, 1);
- Lekérdezés:
  - List<Book> books = Book.listAll(Book.class);
- Objektum törlése:
  - Book book = Book.findById(Book.class, 1);
  - book.delete();
- Összes törlése:
  - Book.deleteAll(Book.class);

Alap SQLite API

SQLite használata

- Az adatbázis megnyitásához és a séma létrehozásához egy segédosztály használható: SQLiteOpenHelper

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

SQLite használata

- Példányosítjuk a saját *SQLiteOpenHelper* osztályunkat
- A példányon meghívjuk a **`getWriteableDatabase()`** vagy **`getReadableDatabase()`** metódust, amik egy *SQLiteDatabase* objektumot adnak vissza. Ez a teljes adatbázisunkat reprezentálja.
- Ezek az objektumon futtathatunk query-ket
  - Ha nem akarjuk kézzel megírni, használhatjuk az *SQLiteQueryBuilder* segédosztályt
- Minden query egy *Cursor* objektummal tér vissza, ami az eredmény adathalmazra mutat
  - Rengeteg metódus a cursor használatára
  - Érdemes do-while ciklussal végigmenni az adathalmazon

SQLite használata

- INSERT és UPDATE: ContentValues objektumon keresztül

```
/* INSERT new record */
public long createOra(Ora ora){
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, ora.getName());
    values.put(KEY_START, ora.getStart());
    values.put(KEY_END, ora.getEnd());
    values.put(KEY_LOCATION, ora.getLocation());
    long rowId = mDb.insert(DATABASE_TABLE, null, values);
    return rowId;
}
```

SQLite használata

- Összes rekord lekérése

```
/* Minden rekord lekérése */
public Cursor fetchAll(){
    return mDb.query(
        DATABASE_TABLE, // tábla
        new String[]{ KEY_NAME, KEY_START, KEY_END, KEY_LOCATION
    }, // oszlopok
        null, null, null, null, // WHERE, GROUP, HAVING
        KEY_START); // ORDER BY
}
```

SQLite használata

- *Cursor* használata

```
OraDBAdapter oraDBAdapter = new OraDBAdapter(getApplicationContext());
Cursor cursor = oraDBAdapter.fetchAll();
Ora ora;
if(cursor.moveToFirst()){
    do{
        String oraStart = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_START));
        String oraEnd = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_END));
        String oraName = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_NAME));
        String oraLocation = cursor.getString(cursor.getColumnIndex(OraDBAdapter.KEY_LOCATION));
        ora = new Ora(oraStart, oraEnd, oraName, oraLocation);
        // rendelkezésre áll az óra objektum
        //...

    } while(cursor.moveToNext());
}
```

- Adapter is képes *Cursor*-ból feltölteni a UI-t
  - *BaseAdapter* helyett *CursorAdapter*-ből származtatunk
  - Lásd jövő órán

Loader használata aszinkron adatbetöltéshez

- Android 3.0-tól jelent meg, de a CompatibilityPack része
- *ContentProvider*-en is használható például a *CursorLoader*-en keresztül
- A *Cursor* lekérdezést háttér szálon oldja meg, így garantáltan nem blokkolódik a UI
- Az aszinkron adatbetöltés állapotáról az *LoaderManager.LoaderCallbacks* interface implementálásával értesülhetünk
- *Loader* létrehozás például Activity-ből a *getLoaderManager().initLoader(id, bundle, LoaderCallbacksImpl)* függvényhívással indítható
  - *Id*: Loader azonosítására szolgál (callback-ek esetén a *Loader* így azonosítható)
  - *Bundle*: A Callback osztálynak további adatok adhatók át *Bundle* formájában
  - *LoaderCallbackImpl*: Az objektum aki implementálja a *LoaderCallbacks* interface-t

Loader példa

```
public class LoaderDemoActivity extends ListActivity
  implements LoaderManager.LoaderCallbacks {

  private final int LOADER_ID_NOTES = 1;
  private SimpleCursorAdapter adapter;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getLoaderManager().initLoader(
      LOADER_ID_NOTES, null, this);
    // Oszlop ID-k
    String[] cols = new String[] {
      NoteTable.COLUMN_ID };
    // UI mező ID-k
    int[] fields = new int[] { R.id.label };
    adapter = new SimpleCursorAdapter(
      this, R.layout.todo_row, null, cols,
      fields, 0);
    setListAdapter(adapter);
  }
```

onCreateLoader() triggerelése

```
  @Override
  public Loader<Cursor> onCreateLoader(
    int id, Bundle args) {
    String[] projection = { NoteTable.COLUMN_ID };
    CursorLoader cursorLoader = new CursorLoader(
      this, NoteContentProvider.CONTENT_URI,
      projection, null, null, null);
    return cursorLoader;
  }

  @Override
  public void onLoadFinished(Loader loader,
   Object data) {
    adapter.swapCursor(data);
  }

  @Override
  public void onLoaderReset(Loader<Cursor> loader) {
    // adat már nem érhető el, referencia törlése
    adapter.swapCursor(null);
  }
} // osztály vége
```

ContentProvider elérése (később)

SQLite debug

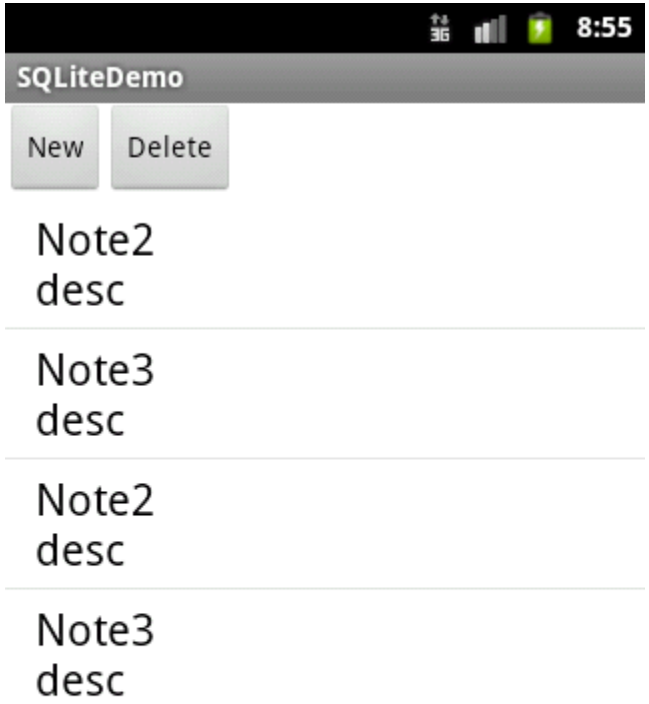
- Az Android SDK „tools” mappájában található egy konzolos adatbázis kezelő: sqlite3
- Ennek segítségével futás közben láthatjuk az adatbázist, akár emulátoron, akár telefonon
- Hasznos eszköz, de sajnos nincs grafikus felülete
- Használata (emulátoron, vagy root-olt eszközön):
  - Konzolban megnyitjuk 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;”)

SQLite Példa

Példa leírása

- Készítsünk egy alkalmazást, melyben jegyzeteket (Note) tudunk tárolni.
- Note mezői:
  - _id: long
  - title: String
  - desc: String
- Tegyük lehetővé új Note létrehozását, törlését és listázását

SQLite példa – Note készítő



Note POJO (Plain Old Java Object)

```
public class Note {
  private long id;
  private String title;
  private String desc;

  // Getterek és Setterek
  // …

  @Override
  public String toString() {
    return title+"\n"+desc;
  }
}
```

Note tábla külön osztályba

```
public class NoteTable {
  public static final String TABLE_NOTE = "note";
  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_TITLE = "title";
  public static final String COLUMN_DESC = "desc";

  private static final String DATABASE_CREATE = "create table " +
TABLE_NOTE
    + "(" + COLUMN_ID + " integer primary key autoincrement, "
    + COLUMN_TITLE + " text not null, " + COLUMN_DESC
    + " text not null" + ");";

  public static void onCreate(SQLiteDatabas
    database.execSQL(DATABASE_CREATE);
  }

  public static void onUpgrade(SQLiteDatabase database, int oldVersion,
    int newVersion) {
    Log.w(NoteTable.class.getName(), "Eredeti verzió: " + oldVersion
      + " új verzió" + newVersion);
    database.execSQL("DROP TABLE IF EXISTS " + TABLE_NOTE);
    onCreate(database);
  }
}
```

SQLite támogatja az AutoIncrement-et

Jelenleg csak újra létrehozzuk a táblát

Saját SQLiteOpenHelper

```
public class NoteDBHelper extends SQLiteOpenHelper {
  private static final String DATABASE_NAME = "notetable.db";
  private static final int DATABASE_VERSION = 1;

  public NoteDBHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  @Override
  public void onCreate(SQLiteDatabase database) {
    NoteTable.onCreate(database);
  }

  @Override
  public void onUpgrade(SQLiteDatabase database, int
oldVersion,
    int newVersion) {
    NoteTable.onUpgrade(database, oldVersion, newVersion);
  }
}
```

Adatbázis állomány neve és verziója

Tábla létrehozása

Tábla frissítése, ha változik a verzió

DAO osztály az adatok eléréséhez 1/3

```
public class NoteDataSource {
  // Database fields
  private SQLiteDatabase database;
  private NoteDBHelper dbHelper;
  private String[] allColumns = { NoteTable.COLUMN_ID,
    NoteTable.COLUMN_TITLE, NoteTable.COLUMN_DESC };

  public NoteDataSource(Context context) {
    dbHelper = new NoteDBHelper(context);
  }

  public void open() throws SQLException {
    database = dbHelper.getWritableDatabase();
  }

  public void close() {
    dbHelper.close();
  }
```

SQLiteOpenHelper példány

Adatbázis megnyitása és lezárása

DAO osztály az adatok eléréséhez 2/3

```
public Note createNote(String aTitle, String aDesc) {
  ContentValues values = new ContentValues();
  values.put(NoteTable.COLUMN_TITLE, aTitle);
  values.put(NoteTable.COLUMN_DESC, aDesc);
  long insertId = database.insert(NoteTable.TABLE_NOTE, null,
        values);
  Cursor cursor = database.query(NoteTable.TABLE_NOTE,
        allColumns, NoteTable.COLUMN_ID + " = " + insertId, null,
        null, null, null);
  cursor.moveToFirst();
  Note newNote = cursorToNote(cursor);
  cursor.close();
  return newNote;
}

private Note cursorToNote(Cursor cursor) {
  Note note = new Note();
  note.setId(cursor.getLong(0));
  note.setTitle(cursor.getString(1));
  note.setDesc(cursor.getString(2));
  return note;
}
```

ContentValues használata beillesztéshez

DAO osztály az adatok eléréséhez 3/3

```
  public void deleteNote(Note note) {
    long id = note.getId();
    database.delete(NoteTable.TABLE_NOTE, NoteTable.COLUMN_ID
        + " = " + id, null);
  }

  public List<Note> getAllNotes() {
    List<Note> notes = new ArrayList<Note>();
    Cursor cursor = database.query(NoteTable.TABLE_NOTE,
            allColumns, null, null, null,
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
      Note note = cursorToNote(cursor);
      notes.add(note);
      cursor.moveToNext();
    }
    // Ne felejtsük bezárni!
    cursor.close();
    return notes;
  }
}
```

Iteráció Cursor segítségével

ListActivity 1/2

```
public class MainActivity extends ListActivity {
  private NoteDataSource datasource;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    datasource = new NoteDataSource(this);
    datasource.open();
    List<Note> values = datasource.getAllNotes();
    // SimpleCursorAdapter használata az elemek egyszerű megjelenítéséhez
    ArrayAdapter<Note> adapter = new ArrayAdapter<Note>(this,
    android.R.layout.simple_selectable_list_item, values);
    setListAdapter(adapter);
  }
  @Override
  protected void onResume() {
    datasource.open();
    super.onResume();
  }
  @Override
  protected void onPause() {
    datasource.close();
    super.onPause();
  }
```

Adatbázis megnyitása

Lista feltöltése

Adatbázis kapcsolat bezárása

ListActivity 2/2

```
  // XML-ből állítjuk be az eseménykezelőt
  public void onClick(View view) {
    @SuppressWarnings("unchecked")
    ArrayAdapter<Note> adapter = (ArrayAdapter<Note>) getListAdapter();
    Note note = null;
    switch (view.getId()) {
      case R.id.newnote:
        String[] notes = new String[] { "Note1", "Note2", "Note3" };
        int nextInt = new Random().nextInt(3);
        note = datasource.createNote(notes[nextInt], "desc");
        adapter.add(note);
      break;
      case R.id.delnote:
        if (getListAdapter().getCount() > 0) {
          note = (Note) getListAdapter().getItem(0);
          datasource.deleteNote(note);
          adapter.remove(note);
        }
      break;
    }
    adapter.notifyDataSetChanged();
  }
}
```

Lista értesítése a változásról

Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <LinearLayout
        android:id="@+id/group"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/newnote"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="New"
            android:onClick="onClick"/>
        <Button
            android:id="@+id/delnote"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Delete one"
            android:onClick="onClick"/>
    </LinearLayout>
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="empty" />
</LinearLayout>
```

ListActivity ezen azonosító alapján azonosítja

Shared Preferences

SharedPreferences

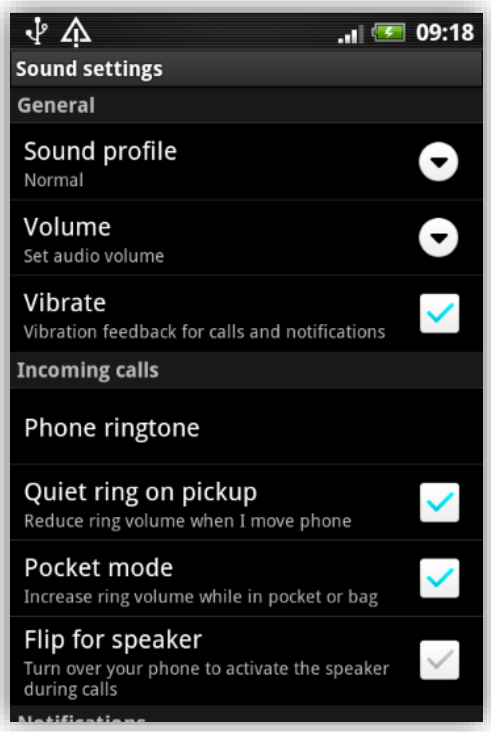
- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
  - Típusok: *int, long, float, String, boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
  - ***MODE_PRIVATE***: csak a saját alkalmazásunk érheti el
  - ***MODE_WORLD_READABLE***: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

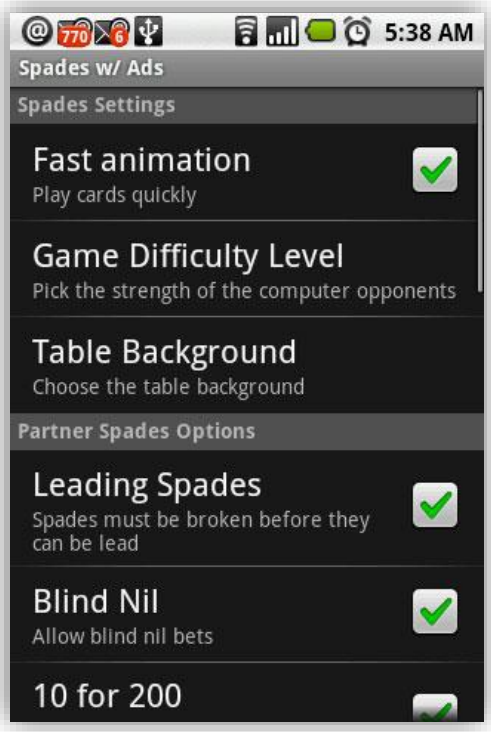
SharedPreferences

- 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(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();`**

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





Példa Preferences nézet - XML

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android=

"http://schemas.android.com/apk/res/and
roid" >
    <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>
```

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
    - (az „Android biztonság” előadáson meglátjuk hogy mennyire igaz ☺)
  - **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 (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(String filename)`** hívása (FileNotFound Exception-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
  - 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:

```
Resources resources = getResources();
InputStream inStream = 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

```
String state = Environment.getExternalStorageState();
// sokféle állapotban lehet, nekünk kettő fontos:
if (state.equals(Environment.MEDIA_MOUNTED)) {
    // Olvashatjuk és írhatjuk a külső tárat
} else if (state.equals(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**:

```
File filesDir = getExternalFilesDir(int type);
```

- 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*

CONTENT PROVIDER

Motiváció 1/2

Eddigi lehetőségeink adatok megosztására komponensek/alkalmazások között:

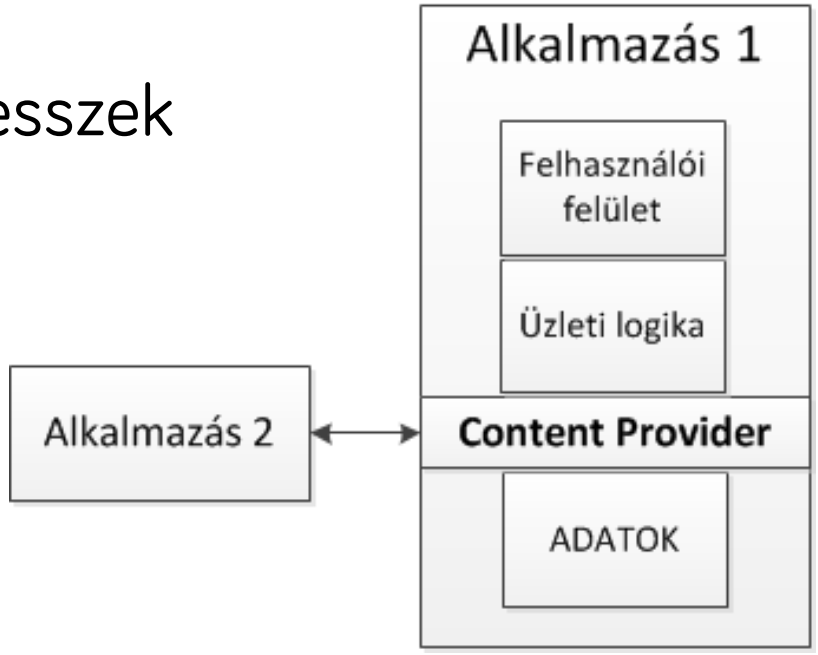
- **Intent Data**
  - Nem erre való
  - Intent kell hozzá, ami néha felesleges
- **SharedPreferences**
  - Nem kényelmes sok adat esetén
  - Ismerni kell a kulcsok nevét
  - Komplex adatstruktúrához használhatatlan
- **Fájlok a nyilvános lemezterületen**
  - Bármikor elérhetetlenné válhat
  - Látható és módosítható, akár törölhető a felhasználó által

Motiváció 2/2

- Egyik sem igazán jó megoldás
- A funkcióra azonban gyakran szükség van
  - Komplex alkalmazás fejlesztése esetén érdemes elválasztani az adat és az üzleti logika rétegeket (Miért?)
  - „Natív” adatok elérése - névjegyzék, naptár, SMS, felhasználói fiókok, stb...
  - Saját alkalmazásunk által létrehozott adatok elérhetővé tétele mások számára

Content Provider

- Megoldás: olyan mechanizmus, ami
  - Elérési réteget biztosít strukturált adatokhoz
  - Elfedi az adat tényleges tárolási módját
  - Adatvédelem biztosítható
  - Megvalósítható akár a processzek közti adatmegosztás is
- Neve: **Content Provider**



Adattárolás

- Konkrét adattárolási struktúra/módszer az alkalmazásra bízva
- Ami szükséges: a megfelelő interfész megvalósítása
  - Leszármaztatás az absztrakt *ContentProvider* osztályból és a kötelező metódusok implementálása
- A legegyszerűbb SQLite adatbázissal csinálni, de nem kötelező ezzel

Content Provider elérése

- Content Provider-től kérdezhetjük le az adatokat
  - „content://contacts”
- A komponens ami képes a lekérdezések futtatására és a válasz feldolgozására: **ContentResolver**
  - Csak ez tudja lekérdezni a Content Providert
  - Lehet akár ugyanabban, akár másik alkalmazásban (processzben)
  - A kommunikációhoz szükséges IPC-t az Android elintézi a fejlesztő helyett, teljesen átlátszó
  - Egy Content Providerből egyszerre egy példány futhat (singleton), ezt éri el az összes Resolver

ContentProvider műveletek

- Nem csak adatlekérés lehet, hanem teljes CRUD funkcionalitás:
  - SELECT: **`getContentResolver().query(...)`**
    - Visszatérés: Cursor az eredményhalmazra
  - INSERT: **`getContentResolver().insert(...)`**
    - Visszatérés: a beszúrt adatra mutató URI
  - UPDATE: **`getContentResolver().update(...)`**
    - Visszatérés: az update által érintett sorok száma
  - DELETE: **`getContentResolver().delete(...)`**
    - Visszatérés: a törölt sorok száma

Select példa 1/2

```
Cursor c = getContentResolver().query(
        UserDictionary.Words.CONTENT_URI,
        new String[]{ UserDictionary.Words.WORD,
                UserDictionary.Words.FREQUENCY }, // projekció
        UserDictionary.Words.LOCALE + " LIKE 'en_%' AND "
                + UserDictionary.Words.FREQUENCY + " > ?", // szelekció
        new String[]{ "50" }, // szekeciós változók értéke
        UserDictionary.Words.FREQUENCY + " DESC"); // rendezés
```

SELECT *word*, *frequency*

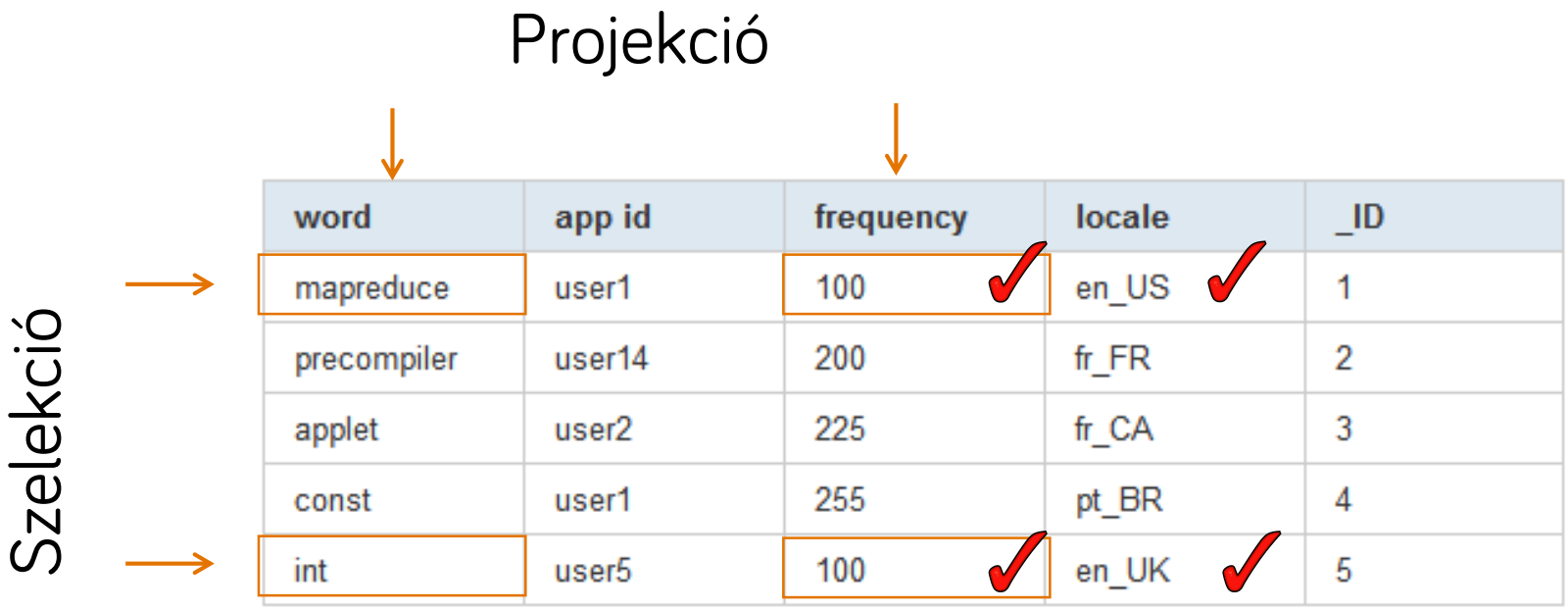
FROM *CONTENT_URI*

WHERE *locale* LIKE *'en_%'* AND *frequency* > 50

ORDER BY *frequency* DESC

Select példa 2/2

Visszatérés:



word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

```
SELECT word, frequency
FROM CONTENT_URI
WHERE locale LIKE 'en_%' AND frequency > 50
ORDER BY frequency DESC
```

Visszatérés:

word	frequency
mapreduce	100
int	100

CONTENT_URI

- Azonosítja a Content Provider-t, és azon belül a táblát
- Pl. **`UserDictionary.Words.CONTENT_URI = content://user_dictionary/words`**
- Felépítése:
  - **`content://`** – **séma**, ez mindig jelen van, ebből tudja a rendszer hogy ez egy Content URI
  - **`user_dictionary`** – „**authority**”, azonosítja a Providert, globálisan egyedinek kell lennie
  - **`words`** – „**path**”, az adattábla (NEM adatbázis tábla!) neve amelyre a lekérés vonatkozik, egy Provider több táblát is kezelhet

Nézzünk néhány példát!

- Telefonkönyv listázás
- Naptár bejegyzés készítés

Összefoglalás

- Android engedélyek (Permission modell)
- Perzisztens adattárolási lehetőségek
  - SQLite, ORM
  - SharedPreferences
  - File
- ContentProvider komponens

Kérdések