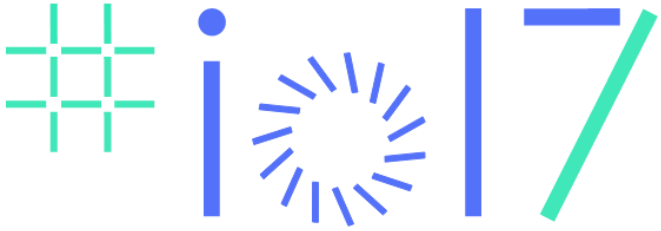


Újdonságok a Google műhelyéből

Péter Ekler

peter.ekler@aut.bme.hu

Android O újdonságok

?

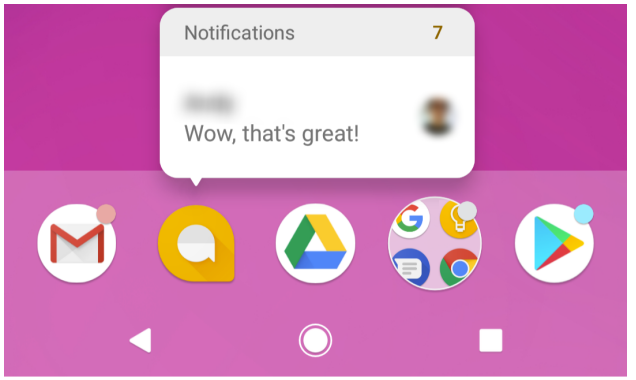
Alkalmazás viselkedés változások 1/2

- Korlátozott háttérben futás
  - Alacsonyabb prioritás a háttér szolgáltatásoknak
- Korlátozott helymeghatározás a háttérben
  - Ritkább pozíció frissítés
  - Állítólag csak COARSE_LOCATION
- Shortcut és Widget elhelyzeés egyszerűbben
  - *requestPinShortcut()* hívás
- URL-ek vége automatikusan kiegészül „/”-el
- Security
  - SSLv3 már nem támogatott
  - WebView külön process-ként fut, illetve *EnableSafeBrowsing*
  - ...

Alkalmazás viselkedés változások 2/2

- *ANDROID_ID*:
  - Alkalmazásonként egyedi és nem felhasználónként
  - Nem változik package uninstall/install-kor
  - Nem változik, ha a package sign key változna fordításkor
- Kontakt-ok *Content Provider*-en keresztüli elérésekor a kontakt „használati” mutatói (*TIMES_CONTACTED*, *TIMES_USED*, stb.) nem pontos számot adnak vissza, hanem csak nagyságrendit
- A kattintható *View*-k alapértemezetten *focusable-k*
- Permission kezelés permission csoportonként már nem automatikus
- További részletek:
  - https://developer.android.com/preview/behavior-changes.html

API újdonságok 1/4

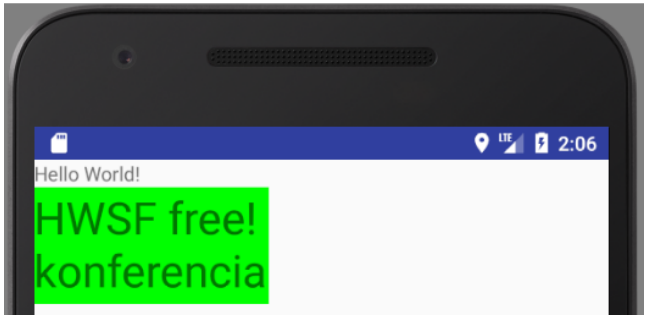


- Notification Badges
- Notification Channels
- Autofill keretrendszer
  - Gépelési hibák minimalizálása
  - Autofill service
  - requestAutofill(), cancel()

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="@font/bariol"/>
```

- res/font

API újdonságok 2/4



- TextView automatikus méretezés

```
<TextView
    android:id="@+id/tvAutoSize"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoSizeTextType="uniform"
    android:autoSizeMinTextSize="12sp"
    android:autoSizeMaxTextSize="100sp"
    android:autoSizeStepGranularity="2sp"
    android:text="HWSF free! konferencia"    />
```

- Picture in Picture
  - > *enterPictureInPictureMode(...);*
- Mi az újdonság?

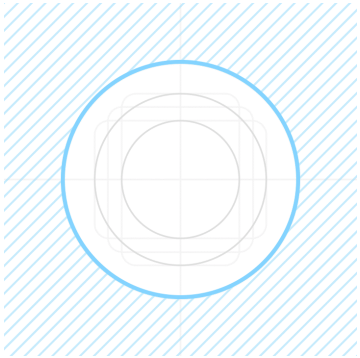
```
TextView tvUserName = findViewById(R.id.tvUserName);
```

API újdonságok 3/4

- Saját kulcs érték tárolás

```
public class MySpecialDataStore implements PreferenceDataStore {
    @Override
    public void putString(String key, @Nullable String value) {
        // TBD
    }
    @Override
    @Nullable
    public void getString(String key, @Nullable String defValue) {
        // TBD
    }
}
```

- Adaptív ikonok



API újdonságok 4/4

- Pointer „capture”
  - view.requestPointerCapture();
- VolumeShaper
  - Automatikus hang „tranzíciók”
- MediaPlayer újdonságok
  - Seek módok
  - Fejlett DRM támogatás
- Wi-Fi Aware API

```
SubscribeConfig config = new SubscribeConfig.Builder()
        .setServiceName(AWARE_FILE_SHARE_SERVICE_NAME)
        .build();

mAwareSession.subscribe(config, new DiscoverySessionCallback() {…});
```

Android O API „diff”

- https://developer.android.com/sdk/api_diff/26/changes/changes-summary.html

- Néhány érdekesség:
  - *java.nio.file.Files*
  - *android.service.autfill* és *android.view.autofill* csomagok
  - *android.icu.text*, *ScientificNumberFormatter*
  - *android.app.admin.DnsEvent*
  - *android.app.KeyguardManager*

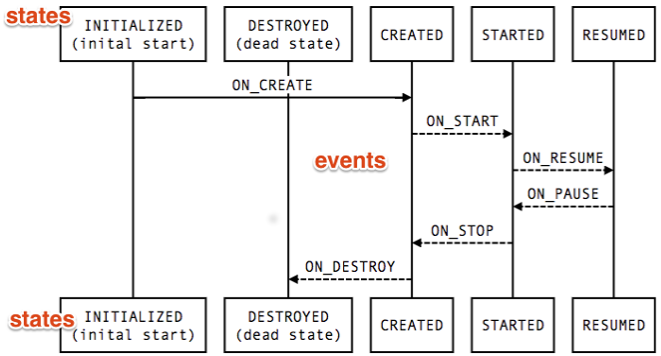
Alkalmazás architektúra

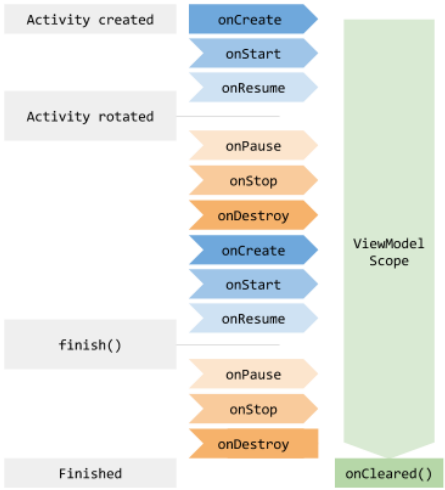
Miért kell beszélni az architektúráról?

- Mi kerüljön az Activity/Fragment osztályba?
- Hol tároljuk az adatokat?
- Hogyan kezeljük a konfiguráció változást (pl. képernyő elforgatás)?
- Hogyan érjük el, hogy az adatok változásakor frissüljön a UI?
- Milyen ORM-et használjunk?
- Hogyan válasszuk szét az adatforrást az adatoktól?

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





LifeCycleObserver

```
public class AcceleroSensorListener implements LifecycleObserver, SensorEventListener {

    private Context context;
    private SensorManager sensorManager;
    private Sensor acceleroSensor;

    public AcceleroSensorListener(Context context) {
        this.context = context;
        sensorManager = (SensorManager) context.getSystemService(SENSOR_SERVICE);
        acceleroSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    public void onResume() {
        sensorManager.registerListener(this, acceleroSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void onPause() {
        sensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        Toast.makeText(context, "Sensor: "+sensorEvent.values[0], Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {

    }
}
```

LifeCyclerObserver használat

- Activity-ben, Fragment-ben, vagy egyedi nézeteken:

```
getLifecycle().addObserver(new AcceleroSensorListener(this));
```

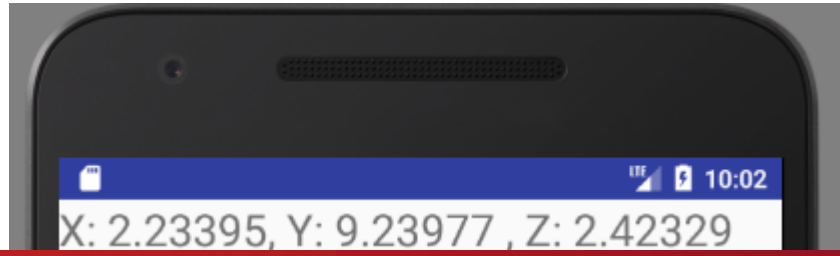
LiveData

```
public class AcceleroSensorLiveData extends LiveData<SensorEvent> implements SensorEventListener {
    private Context context;
    private SensorManager sensorManager;
    private Sensor acceleroSensor;

    public AcceleroSensorLiveData(Context context) {
        this.context = context;
        sensorManager = (SensorManager) context.getSystemService(SENSOR_SERVICE);
        acceleroSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    @Override
    protected void onActive() {
        sensorManager.registerListener(this, acceleroSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onInactive() {
        sensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        setValue(sensorEvent);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {}
}
```

LiveData használat

- LiveData gyakran Singleton
- Activity-ben, Fragment-ben, vagy egyedi nézeteken:

```
LiveData<SensorEvent> acceleroSensorLiveData =
  new AcceleroSensorLiveData(this);
acceleroSensorLiveData.observe(this, new Observer<SensorEvent>() {
    @Override
    public void onChanged(@Nullable SensorEvent sensorEvent) {
        tvStatus.setText("X: "+sensorEvent.values[0]+", Y: "+
          sensorEvent.values[1]+" , Z: "+sensorEvent.values[2]);
    }
});
```



ViewModel és használat

```
public class AcccceleroSensorViewModel extends ViewModel{

    private AcceleroSensorLiveData acceleroSensorLiveData;

    public LiveData<SensorEvent> getAcceleroLiveData(Context context) {
        if (acceleroSensorLiveData == null) {
            acceleroSensorLiveData = new AcceleroSensorLiveData(context);
        }
        return acceleroSensorLiveData;
    }
}
```

- Activity-ben, Fragment-ben, vagy egyedi nézeteken:

```
AcccceleroSensorViewModel acceleroViewModel = ViewModelProviders.of(this).get(
  AcccceleroSensorViewModel.class);
acceleroViewModel.getAcceleroLiveData(this).observe(this,
  new Observer<SensorEvent>() {
    @Override
    public void onChanged(@Nullable SensorEvent sensorEvent) {
        tvStatus.setText("X: "+sensorEvent.values[0]+", Y: "+
          sensorEvent.values[1]+" , Z: "+sensorEvent.values[2]);
    }
});
```

Room példa - Entity

```
@Entity
public class User {
    @PrimaryKey
    private int uid;

    @ColumnInfo(name = "first_name")
    private String firstName;

    @ColumnInfo(name = "last_name")
    private String lastName;

    // Getter-ek és setter-eks kötelezők
}
```

Room példa - DAO

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND "
           + "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

Room használat

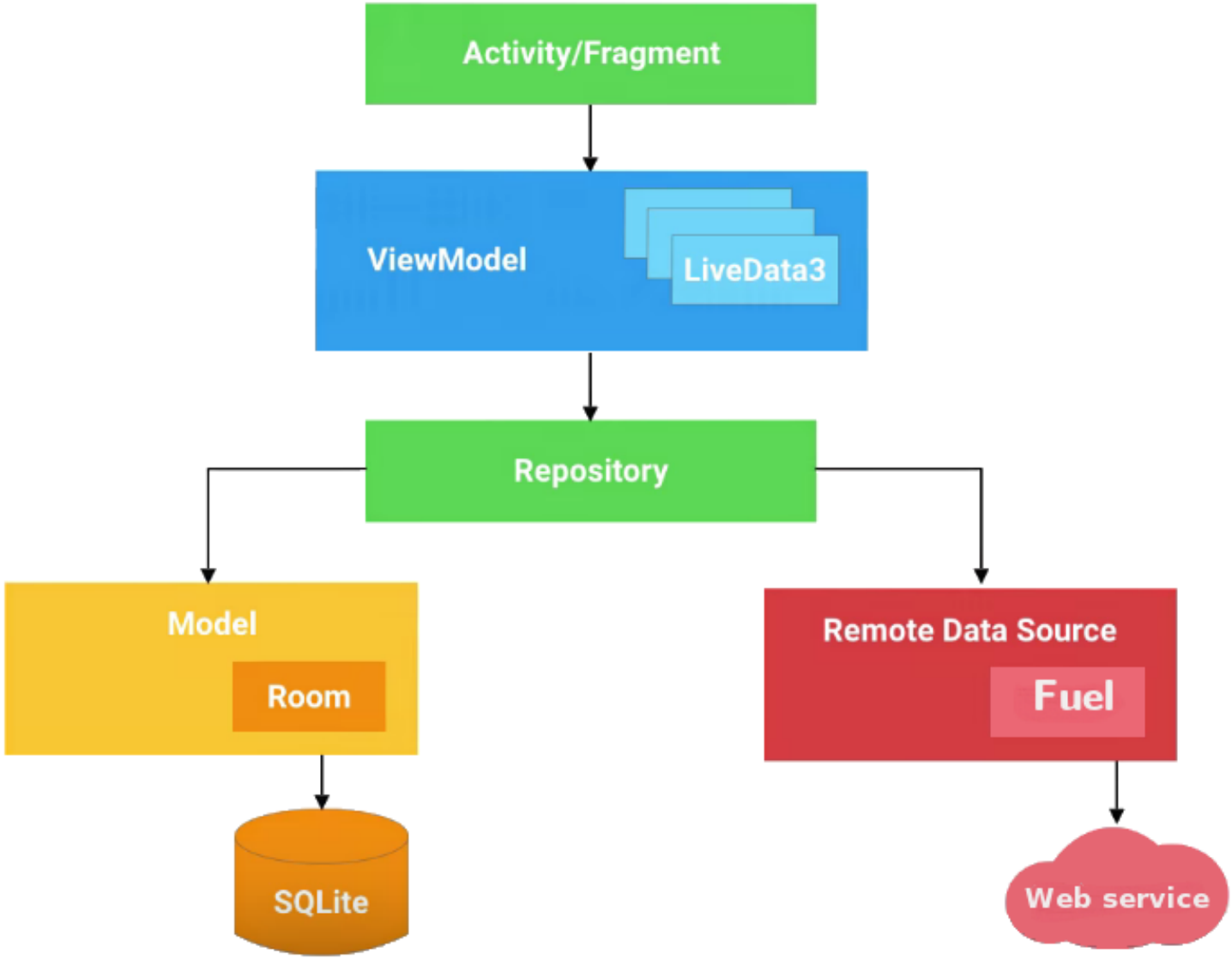
- Database:

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

- DB példány:

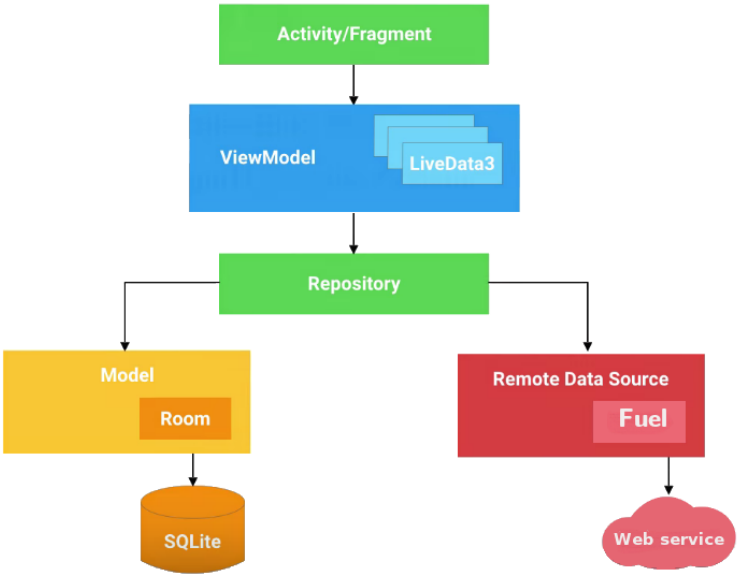
```
AppDatabase db = Room.databaseBuilder(
        getApplicationContext(),
        AppDatabase.class, "database-name").build();
```

Javasolt architektúra



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



Tanácsok

- A komponensek nem adatforrások
- Számos best practice létezik már, pl. MVP, de nem Android specifikusak
- Ha van egy bevált architektúránk, nem kell változtatni
  - De érdemes megnézni, hol segíthetnek/egyszerűsíthetnek az architektúra komponensek
- Tervezéskor a tesztelhetőség mindig fontos szempont legyen

Kotlin


Holnaptól Kotlint kell tanítanunk
Bazzeg

Kotlin érdekességek

- Extension függvények

```
fun String.thxEmailMessage():String {
    return this.plus(" , köszönjük, hogy részt vett a konferencián.")
}
"Zoltán".greet()
```

- Null safety

```
val text:String = "demo"
text = null
text?.length
```

- Default és elnevezett argumentumok

```
fun enableFeature(id: Int, desc: String = "Open") {
}

enableFeature(15, "Hey")
enableFeature(desc = "Name", id = 45)
enableFeature(45)
```

Az Android jövője

Pletykák: Fuchsia

A B C D E F G H I J

K L M N O P Q R S

T U V W X Y Z

Még van ~9 évünk ☺

Köszönöm a figyelmet!

https://github.com/peekler/AndroidDemos

peter.ekler@aut.bme.hu

AutSoft