Bevezetés

Szoftverfejlesztési eszközök:

- SDK (IDE, emulátor, Java/Kotlin)
- NDK (natív C++, Eclipse plugin)
- ADK (accessory, dokk és társai, Android Open Accessory protokoll USB-n/BT-n)

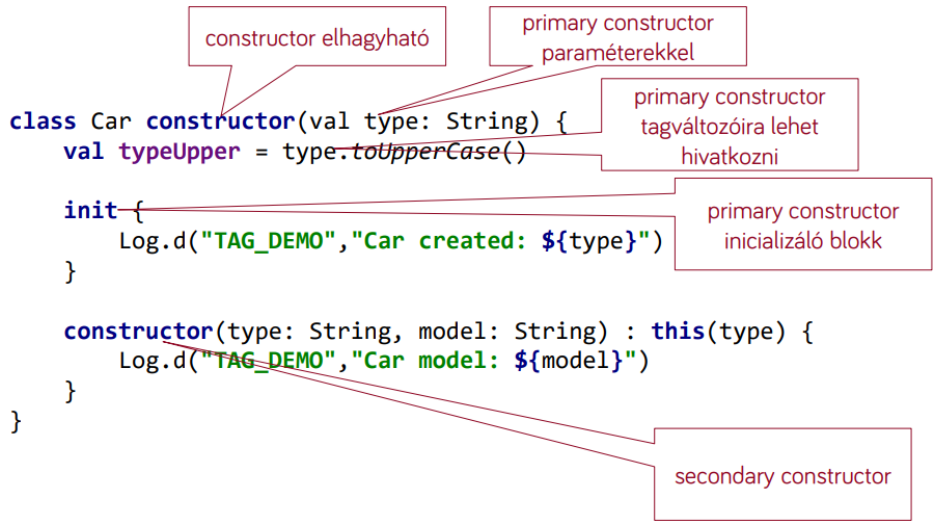
SDK komponensek: doksi, példakód, ADB, third party API-k, pl. Google-től a Maps
APK: kb. egy jar, tömörített, van benne:

- META-INF: CERT.RSA (tanúsítvány), MANIFEST.MF (kulcs-érték meta-k), CERT.SF (erőforrások és SHA-1-ük)
- res: erőforrások
- AndroidManifest.xml: név, verzió, jogosultság, library-k
- classes.dex: osztályok Dalvik bytecode-jai
- resources.asrc

Debuggolás: teljesen megy eszközön, minden alkalmazás saját VM-ről, azok egyedi porttal, de van egy base port, amiről minden debugolható
Alkalmazás elemei: Activity, Service, Content Provider, Broadcast Receiver
Manifest: XML, komponensek, minimális rendszerkövetelmény, telepítéskor ellenőrizve, engedélyek, HW/SW funkciók, API-k, alkalmazás neve és ikonja

Kotlin

Egy sziget neve
JetBrains, 2011, open source, 2017 Google IO: hivatalos támogatás droidon, statikusan típusos, OO és funkcionális
JVM bytecode vagy JS lesz belőle, automatikus Java-Kotlin konverzió
Nyelvismeret nélkül is olvasható
Egyszeri értékadás: val, típus elhagyható, később is lehet értéket adni, változó: var
String sablon: “$score pont”
Null-safe, alapból semmi nem lehet null, csak pl. az Int?
Null check az elvis operátorral (?.), ha null a változó, nem hívódik meg, null esetére egy ?: operátor megadja az értéket
Null kivételes működést a !!. operátor hoz vissza
Függvény szintaxis: fun add(a: Int, b: Int): Int = a + b - return elhagyható, a return value is, ha érték nélkül térne vissza, az Unit, de elhagyható



Alapesetben minden class final, open-t lehet leszármaztatni
Láthatóság: public (default), private, internal, protected
Van property:

```
var type: String? = null
    set(type) {
        Log.d("TAG_CAR", "type SET")
        field = type
    }
```

Data class (automatikus equals, hashCode, toString, component<N>, copy):

```
data class Ship(val name: String, val age: Int)
val discovery = Ship("Discovey", 31)
val (name, age) = discovery
```

Delegate-ek ilyen undor módon: class Gomb(pressable: Pressable) : Pressable by pressable
Függvény típus: ()->T, a ::fgv módon adható át függvény
Van range, iterálható is: for (nr in 1..y+1 step 2)

Android UI

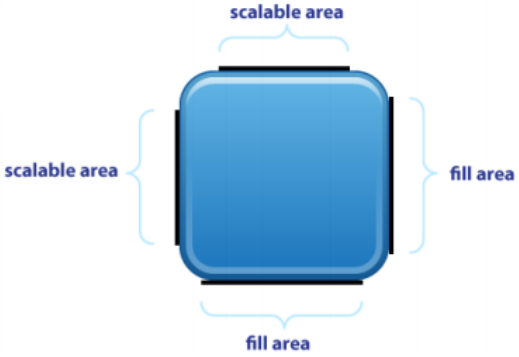
Egyedi nézet: View leszármazott, felül kell írni az onDraw-t és az onTouchEvent-et, XML layout-ban használható, beépített nézetek és LayoutGroup-ok is felüldefiniálhatók
Activity: képernyő, amin a user műveleteket végez, kb. egy ablak, full screen vagy popup
Egy app = több lazán csatolt activity, bármelyik indíthat újabbat, tipikusan fő activity indításkor, amit a manifestben jelölünk
Activity állapotok: resumed (running), paused (él, de van valami kicsi vagy átlátszó felette), stopped (él, de nem látszik, és a rendszer felszabadíthatja)
Override-olható:

- onCreate - létrejött
- onStart - látható lesz
- onResume - látható lett
- onPause - most veszik át a fókuszt
- onStop - már nem látható
- onDestroy - rip előtt
- onSave/RestoreInstanceState - amit mond, erre nincs garancia, hogy hívódik

Kép forgatása, billentyűzet, nyelv újratölt minden activity-t (onDestroy, onCreate)
Activity back stack: activity-k LIFO-ja, a vissza gomb kukázza a felsőt, és alá lép
Task: amihez több activity kell, nem feltétlenül ugyanabban az app-ban, kaphat külön back stacket, ami miatt egy korábbit lelőhet a rendszer
String res paraméterezése:
<string name="txtPageViews">%1$d oldalmegtekintés</string>
getString(R.string.txtPageViews, 5)
Android LogCat: Log.i (info), warning, error, debug, verbose, minden két string parammal, átirányítható fájlba is

```
fun runSecondActivity() {
    val myIntent: Intent = Intent()
    myIntent.setClass(this@MainActivity,
                      SecondActivity::class.java)
    // Adat átadása
    myIntent.putExtra("KEY_DATA", "Hi there!")
    startActivity(myIntent)
}
```

Screen size: fizikai képátló (small, normal, large, xlarge)
Screen density: DPI (ldpi, mdpi, hdpi, xhdpi, nodpi, tvdpi)
Orientation: portrait, landscape, futás közben változik, lehet rögzíteni
Resolution: px, ezzel nem foglalkozunk UI-nál
Density-independent pixel: dp, px = dp * (dpi / 160)
9-patch képek:



A sűrűség függetlenség fontos, hogy fizikailag legyenek azonosak a méretek.
Minősítő string a res könyvtár után: <drawable vagy layout>-<large, hdpi, ilyenek>
Képarány: long és notlong a nagyon széleseknél
Általában a kisebb erőforrást választja, ha nincs adott méretű, de ha nincs ilyen, az error
Tablet minősítők: sw<N>dp (smallest width), w<N>dp, h<N>dp
scale-independent pixel: olyan, mint a dp, csak szöveghez
Menü: XML-ben menu tag alatti item-ek id-vel és title-el, egy itemben lehet menu is
Action Bar: tipikusan menük, logók, konzisztens navigáció
Toolbar: dinamikusabb, támogat menüket, custom elemeket, manuális pozícionálást
Toolbar onCreate-ben:
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
Minden View leszármazott, a ViewGroup is, ami Layout-ok szülője és egymásba ágyazhatók
LinearLayout: egymás után dobált vezérlők, megadható egy layout súlya, ekkora arányban tölti ki a képet, jellemző paraméterek: margin, padding, gravity, ScrollView, weight
ScrollView: ha a tartalma nagyobb a méreténél, scrollozható
RelativeLayout: viszonyokat ír le, pl. toRightOf, egy jobb verziója a ConstraintLayout
CoordinatorLayout: felső szintű UI irányelv, material design-t segít
LayoutInflater: XML-ben összeállított layout-okat épít: val myView = getLayoutInflater().inflate(R.layout.activity_main, null)
Animációk: scale, rotate, translate, alpha alapján, ObjectAnimator és Animation osztály
MotionLayout: MotionScene-ekből áll, van rá editor is
Validáció: setError (pl. text-nél)
TextInputLayout: login mezőhöz meg ilyenekhez
Lokalizáció ilyen: van
Szálakat a Thread osztályból leszármazás művel
UI csak a fő szálból basztatható: runOnUiThread(runnable: Runnable)

Fragmentek

A képernyő nagyobb részéért felelős objektumok, de lehet háttérmunka is, több funkció lehet velük egy képernyőn, nem mindig fullscreen, és más életciklus Activity-hez képest, nem alkalmazás komponens, hanem mi hozzuk létre

Egy Activity-hez kapcsolódik, és hatással van rá az Activity életciklusa
Tartalmaz onAttach, onDetach, onCreate/DestroyView ciklusokat is
Statikusan csatolható fragment XML taggel, vagy dinamikusan tölti be az Activity egy adott ViewGroup-ba, ahol tranzakciókkal módosítható
FragmentManager: tranzakciókat indít (add, remove, replace, commit), meglévők közt keres, stack-et kezel
Két fragment közt az Activity közvetít
Fragment.arguments (Kotlin) - paramétereket gyűjt, túléli a forgatást, setArguments, getArguments (Java)
DialogFragment: dialógusként is megjelenhet, rákerül a back stack-re
A ViewPager (több nézet közt swipe) háttérben futó fragment segítségével működik

RecyclerView

Listák hatékony kezelése, gyors scroll, érintéses gesztusok támogatása, hatékony elem újrafelhasználás
onCreateViewHolder: egy sor nézetét hozza létre, van benne egy inflate, ami ViewHolder-t csinál
onBindViewHolder: adatok átadása az n-edik elemnek (második paraméterként)
getItemCount: kötelező override
Van újfajta: listakezelés és getItemCount nélkül, LiveData-kompatibilitással
Nézetek: linear, staggered grid (pl. Keep), grid
Lista készítése: data class, egy sor layout-ja, RecyclerView, adapter - tartalmat csinál
ItemTouchHelper.Callback: mindenféle, pl. easy drag & drop gesztus

- isLongPressDragEnabled()
- isItemViewSwipeEnabled()
- onMove(…)
- onSwipe(…)

Komponensek közti kommunikáció

Minden app sandboxban fut, szigorú korlátozások pl. kamerára vagy háttértárra, adat- és eszközvédelem miatt
Androidon a sandbox neve Dalvik VM, kritikus műveletekhez engedély kell
Két komponens beszél: Intent (pl. új Activity-re navigálás, zene player service indítása)
Egy komponens mindenkinek: Broadcast Intent (pl. 15% aksi van)
Adatmegosztás: ContentProvider (pl. névjegyzék)
Intent átadás mindig az OS-en keresztül megy
Intent: struct, késői (futásidejű) kötést valósít meg komponensek (Activity, Service, Broadcast Receiver) közt, az esemény absztrakt leírása, az Android megkeresi a címzettjét, és példányosítja, ha kell
Intent elemei: címzett osztály, akció, adat, kategória, extrák (kulcs-érték párok), kapcsolók
Explicit intent: tudjuk a címzettet, implicit: csak azt döntjük el, hogy mit szeretnénk (pl. hívás)
startActivityForResult: visszatérés előtt az indító requestCode-ra tudnak választ adni setResult módon, amit visszatérés után az onActivityResult kezel
Implicit intentet úgy fogad egy app, hogy intent filtert publikál a manifestben
Intent filter category, ha nem default: launcher (másik belépési pont), home (custom launcher)

PendingIntent is van, hogy felkészüljünk a fogadására, de később, vagy máshonnan (pl. widgetről) érkezik
Rendszerszintű broadcast intentekre fel lehet iratkozni, de lehet egyedit is csinálni
Broadcast Receiver kapja el ezeket, futás közben vagy manifestből készül, intent filterrel
A továbbdobást megakadályozhatjuk az abortBroadcast hívással
BOOT_COMPLETED broadcast is van, indítás után

Perzisztens adattárolás

- SQLite: strukturált
- SharedPreferences: kulcs-érték párok láthatósággal (csak app, mindenki olvas/ír)
- Privát lemezterület: nem publikus része a fájlrendszernek
- SD-kártya: nagy méretű adatok nyilvánosan
- Hálózat: saját szerver, felhő

SQLite a Droidon alap, sémát és query-ket mi írunk, mindenhol kötelező primary key (_id)
Az SQLite fájlt kezel, ami lassú, ezért érdemes aszinkron csinálni
sqlite3 a konzolos debug tool akár telefonon
ORM: Java objektumok a db-ben: osztály - tábla, objektum - sor, mező - oszlop
Room: absztrakciós réteg az SQLite felett, abból mindent használni tud

```
@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-ban a DAO kezeli a db-t, minden függvénye egy annotációs query alapján jön létre
@Database osztály kezeli az egészet, tartalmazza a db verzióját
Architecture Components:

- Lifecycle (ettől függő komponensek) - magától végrehajtódik pl. onStart-on belül, nem kell külön hívni az activity-ből egy objektum életciklusait
- LiveData (megfigyelhetővé teszi az adatot)
- ViewModel (UI-on megjelenő adatok egyszerű kezelése)
- Room Persistence Library (Google saját ORM-je)

Javasolt, hogy a UI ViewModel-t használ, ami LiveData-t mutat kifelé és Repository-ba rejti az adatforrást
SharedPreference-ből több tartozhat egy apphoz, ha egy activity-nek egy van, név se kell
Preferences Framework: gyári keretrendszer saját beállítások oldalakhoz, kell hozzá egy XML, egy Activity, és opcionálisan ChangeListener, SharedPreference az alapja
Fragment alapú beállítások nézet: bal oldalt címkék, jobb oldalt preference oldalak: PreferenceActivity
InternalStorage: alkalmazás saját, védett területe, openFileInput olvas, FileInputStream-et ad, még cache is van, amit a getCacheDir ad, amit kevés lemezterület esetén az OS üríthet
Nagy, read-only statikus fájlok a res/raw mappában, ezek ugyanúgy szűrhetők, pl. nyelvre
Nyilvános lemezterület: lehet kívül, belül, mindenki az egészhez hozzáfér (azért permission kell), USB mount-nál read only lesz, a user mindent lát
MediaScanner a nyilvános mappákat típusonként bejárja

Futásidejű engedélyek

Régen manifest, Android 6 óta futás közben kérjük
Ellenőrzés, majd ha nincs, felhasználónak szólunk az indokkal

A felhasználó settings alól visszavonhatja
Van pár külső könyvtár a kezelésükre, pl. Dexter

Backend as a Service

Adatkezelés felhőben
A szerver full saját, lehet PHP, JS, ami mehet Azure-ban, Heroku-n, stb.
Firebase: realtime JSON NoSQL, autentikáció, storage, crash report, analytics, notification
FCM (Firebase Cloud Messaging) - szerver-kliens push notification topic subscribe alapján

Helymeghatározás

GPS (pontos, de kültéren sok árammal) és hálózat (cella/wifi, gyors és pontatlan), általában a kettő egyszerre működik
ACCESS_COARSE_LOCATION permission a hálózat, ACCESS_FINE_LOCATION a GPS
LocationManager rendszerszolgáltatás, LocationListener-t használ, van egy halom callback-je
Google Play Services-en át megosztják alkalmazások egymással a helyet
A pozíció fontos eleme az accuracy, ami alapján érdemes mérlegelni
Geocoder: GPS koordináták címből, van reverse is, internet kell hozzá
Places API: Geocoder, place picker, autocomplete helyekhez
ProximityAlert: értesít egy hely megközelítésekor (koordináta és sugár alapján)
Geofencing: helyek körberajzolása, belépéskor, bolyongáskor, és kilépéskor trigger
Activity recognition: vezetés, biciklizés, gyaloglás…, engedélyköteles, változásra feliratkozás
MapFragment a Maps API-ban, internet permission kell hozzá, van a mapra onClickListener, és markerek is, lehet a térképre firkálni
MapView a konténer, GoogleMap objektumon keresztül

Hálózatkezelés

HTTP + REST - standard Java vagy Apache
Van JSON és XML API is beépítve
Retrofit: HTTP kérések leírása annotációkkal
WebView: WebKit alapú renderer, WebSettings tartozik hozzá, pl. a szövegmérethez
TrafficStats: adatforgalom figyelése, külön TCP/UDP, adatforgalom tiltása reseteli
TCP/UDP Java Socketekkel vagy DatagramSocket

Multimédia

Kamera API: több kamera, típus, funkció (zoom, vaku), engedélyköteles
Beépített kamera apptól lehet kérni eredményt Intenten keresztül, a mentés helyét adja át
Egyedi kamera nézet is készíthető a Camera API-val
Play Services, Vision API: arcot, QR-kódot, szöveget olvas - áttér az ML Kitre (Firebase), ami már tárgyakat, állatokat, stb.-t is felismer, ismert helyeket azonosít
RingtoneManager: figyelmeztető hangok, MediaPlayer: sima hangok, ToneGenerator
Csak kimeneten tudunk hangot kiadni, hívásba nem lehet keverni
AudioManager: forrás és kimenet megadása
Hangfelismerés: SpeechRecognition API
TextToSpeech: szövegfelolvasó, indítása előtt várni kell az inicializálásra

Service-ek

Hosszabb ideig tartó háttérfeladat, UI nélkül, más számára is adhat funkciót, akkor is nyitva marad, ha a hívó activity rip
Intent indítja, pl. zenelejátszó, hosszabb hálózati művelet (torrent), GPS, stb.
Lehet komponensből indított (started), ezen belül foreground (kötelező notification-nel), background, vagy IntentService, illetve kapcsolt (bound)
A started általában egy feladatra való, csak a hívónak van rá referenciája, magát kell leállítania, mert az OS nem fogja
Bound: nem indítjuk kézzel, ha valakinek (akár többeknek) kell, akkor indul, Android 8-tól felfelé a rendszer idle-be rakhatja pár perc után
Nem kap külön szálat, 5 mp után ugyanúgy nem válaszol
Újraindítást kérni kell, lehet sticky (amint van neki RAM, már indul is), not sticky (amúgy is lelőné magát), redeliver (ha nem lőtte le magát, akkor sticky)
Előtéri service visszalőhető a háttérbe, ha végzett, de fut tovább
IntentService: külön szálat indít, lelövi magát a kérések után, sorosít
Service-ből értesítés: toast, notification, vagy broadcast

Content Provider

Elérést biztosít strukturált adathoz, elfedi a tényleges tárolás módját, adatvédelem és processek közti adatátadás, pl. copy-paste, widget
A Droid gyárilag is használ, pl. médiafájlok, legutóbbi keresések, beállítások, könyvjelzők
ContentResolver lekérdezések férnek az adathoz, a Provider singleton
CONTENT_URI alapján is lehet hivatkozni
A rendszer által nyújtott contenthez általában engedély kell
A query mindig Cursorral tér vissza, véletlen hozzáférés az eredményhalmazhoz
Az insert a beszúrt elem URI-ját adja vissza
Az update visszaadja a módosított sorok számát, a delete a töröltekét
Manifestben jegyzendő dolog

Szenzorok

Absztraktok: gyorsulásmérő, giroszkóp, fény, mágnes, elforgatás, közelség
További egzotikus és származtatott pl. hő, pára, nyomás, lineáris gyorsulás
Egy típusból több is lehet egy eszközben
SensorManager-től lehet elkérni, általában a default-ot akarjuk, lehet listázni is
Eseménykezelten a SensorEventListener mondja, ha új értéke van a szenzornak: onSensorChanged
Az eseménykezelőben beállítható, hogy milyen gyakran kérjük (fastest, game, normal, UI)

Jetpack

Teljes support library, core funkciók (AppCompat), UI (fragment, layout, stb.), architecture (perzisztencia, életciklus, stb.), behavior (engedélyek, értesítések, megosztás, stb.)
Navigation Component: egyszerűbb navigáció activity-k és fragmentek közt
Navigation graph: XML-ben leírja a navigációs útvonalakat
NavHost: nézet, amiben váltakoznak a nézetek
NavController: navigációt vezérel
Data binding: felület és kód kapcsolása, rövidebb kód
View binding: findViewById kiváltása, inflate után kapott objektumból minden UI elem látható

Paging: nagy mennyiségű adat szakaszos betöltése (végtelen listából csak ami látszik)
Work manager: ütemező, kényszerekkel (pl. épp tölt a telefon), háttérszálon

Near-Field Communication

NFC tag és mobil közti max 4 cm-en kis payload-ok
NDEF: NFC Data Exchange Format
Permission kell hozzá, és le lehet tiltani a nem NFC-képes készülékekről

Egyéb

Statikus kódelemzés - Lint - strukturális problémák felderítése, hatékonyság és teljesítmény növelése, nem használt XML namespace-ek szűrése, deprecated kódra javaslatok
Spell Checker API - helyesírás ellenőrzés