Android fejlesztés

Alapok

Tanács Attila, Kálmán Kornél

Android

- Az Android egy mobil eszközök számára kifejlesztett **szoftverrendszer**, amely magába foglal egy **operációs rendszert, középszintet** (middleware) **és célalkalmazásokat.**

- Az **Android SDK** olyan eszközöket és alkalmazásprogramozási felületeket (APIs) biztosít, melyek szükségesek az Android platformon való alkalmazásfejlesztés elkezdéséhez, mindezt **Java programozási nyelven**.

Főbb jellemzők

- Az **alkalmazás keretrendszer** lehetővé teszi az egyes alkotóelemek, komponensek ***újrafelhasználását*** és ***helyettesítését***.
- Mobil eszközökre optimalizált **Dalvik (újabban ART) virtuális gép**
- A nyílt forráskódú WebKit motoron alapuló **beépített böngésző**
- Egy 2D grafikai programkönyvtárra épülő **optimalizált grafikai műveletek**
  - OpenGL ES 1.1-re és 2.0-ra épülő 3D grafika
  - Hardver gyorsítás opcionálisan
- **SQLite** strukturált adattároláshoz
- **Médiatámogatás** a közismert audio, video és állókép formátumokhoz
  - MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF
- **GSM telefonálás** (hardver függő)
- **Bluetooth, EDGE, 3G és WiFi** (hardver függő)
- **Kamera, GPS, iránytű és gyorsulásmérő** (hardver függő)
- **Gazdag fejlesztőkörnyezet**
  - céleszköz emulációs program,
  - hibakereső eszközök (debug),
  - bővítmény az Eclipse integrált fejlesztői környezethez,
  - lehetővé teszi memória- és teljesítményprofilok létrehozását.

Könyvtárak

- Az Android magába foglal jó néhány **C/C++ programkönyvtárat**, amelyeket az Android rendszer különböző alkotóelemei használnak. Ezen képességek/funkcionalitások a fejlesztőkhöz az Android alkalmazás-keretrendszerén keresztül jut el. Néhány a könyvtárak közül az alábbi listán látható:
  - **C rendszerkönyvtár:** a C nyelv rendszerkönyvtárának (*libc*) egy „BSD-leszármazott” megvalósítása, beágyazott, Linux-alapú eszközökre optimalizálva.
  - **Média könyvtárak:** a PacketVideo OpenCORE multimédiás alrendszerén alapul; a könyvtárak támogatják több elterjedt audio, video és állókép formátumú fájl rögzítését és visszajátszását (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG stb.)
  - **Felület menedzser:** a kijelző alrendszerhez való hozzáférést kezeli, és zökkenőmentesen rak össze 2D és 3D grafikus rétegeket több alkalmazásból
  - **LibWebCore:** egy modern web böngésző motor, amely egyszerre „hajtja meg” az Android böngészőt és egy beágyazható web nézetet.
  - **SGL:** a mögöttes 2D grafikus motor
  - **3D könyvtárak:** egy OpenGL ES 1.0 és 2.0 API-kon alapuló megvalósítás; a könyvtárak használnak mind 3D gyorsítást (ha elérhető), mind az erősen optimalizált szoftveres 3D raszterezőt
  - **FreeType:** bittérkép és vektorgrafikus szöveg renderelés
  - **SQLite:** egy minden alkalmazás számára hozzáférhető hatékony, de egyszerű relációs adatbázis motor

Architektúra

Alkalmazások

- **Alap alkalmazások előre telepítve**
  - E-mail kliens, SMS program, naptár, térkép, web böngésző, névjegyek stb.
  - Java nyelven készülnek

- **Új alkalmazások**
  - Fejlesztők számára elérhetők a hardver elemek, hely információ; háttér szolgáltatások futtathatók, riasztások, értesítések adhatók stb.

Alkalmazás keretrendszer

- A fejlesztőknek **teljes hozzáférésük van ugyanahhoz a keretrendszerbeli alkalmazásprogramozási felülethez**, amelyet az **alapvető alkalmazások** is használnak.
- Az alkalmazás felépítése úgy van megtervezve, hogy **egyszerűsítse a komponensek újrafelhasználását**;
  bármely alkalmazás nyilvánossá tudja tenni képességeit, így bármely másik alkalmazás felhasználhatja azokat (a szükséges biztonsági megszorításokról a keretrendszer gondoskodik).
- Szintén ezen mechanizmus teszi lehetővé **a komponensek egymással való helyettesítését a felhasználó számára.**

Futtató környezet

- Az Android tartalmaz **néhány alapvető könyvtárat**, amelyek biztosítják a Java programozási nyelv magkönyvtáraiban található legtöbb funkcionalitást
- **Minden Android alkalmazás a saját processzét futtatja**, a **saját Dalvik Virtuális Gép** (DVM) példányával együtt
- A Dalvik úgy lett megírva, hogy egy eszköz hatékonyan futtathassa több virtuális példányát. A DVM a Dalvik Executable *(.dex*) formátumban futtatja a fájlokat, mely a lehető legkisebb memória-felhasználásra lett optimalizálva. A DVM regiszter alapú, és a Java nyelv által lefordított osztályokat a beépített DX eszköz által *.dex* formátumúra átalakítva hajtja végre.
- A DVM mögöttes funkcionalitásért, úgy mint végrehajtási szálak (thread) használata vagy alacsonyszintű memóriakezelés, a **Linux kernelhez** fordul.
- Az újabb **ART virtuális gép** az alkalmazás telepítésekor végi el a fordítást
  - Lassabb a telepítés és nagyobb helyet foglal el az alkalmazás
  - Gyorsabb az alkalmazások indítása és futtatása

Linux Kernel

- Az Android a Linuxra támaszkodik
  - 2.6 a korábbi Android verziók esetén
  - 3.x az ICS verzió óta
  - **Alapvető rendszer szolgáltatások**, mint például biztonság, memóriakezelés, processzuskezelés, hálózati protokoll verem, illesztőprogram modell.
  - A kernel egy, a hardver és a szoftverrendszer többi része közti **absztrakciós rétegként** is működik.

Alkalmazások alapjai

- Android alkalmazások a **Java programozási nyelven** íródnak
- A lefordított Java kód – bármilyen adat- és az alkalmazás által kért erőforrásfájllal egyetemben – egy ***Android package***-be van becsomagolva, amely egy **.apk** kiterjesztésű archív fájl. Egy *.apk* fájlban lévő össze kód tekinthető egy *alkalmazásnak*.

Sok tekintetben egy Android alkalmazás „a saját világában létezik”:

- **Minden alkalmazás a saját Linux processzét futtatja.** Az Android elindítja a processzt amikor valamely alkalmazás kódja végrehajtandó, és leállítja azt, amikor nem szükséges már és egyéb alkalmazásoknak szükségük van (rendszer) erőforrásra.
- **Minden processznek meg van a saját virtuális gépe (DVM),** azaz az alkalmazás kódja egyéb alkalmazásoktól elkülönítve hajtódik végre.
- **Alapértelmezésben minden alkalmazás számára ki lesz osztva egy (egyedi) Linux felhasználói azonosító.** Az engedélyek úgy vannak beállítva, hogy az alkalmazás fájljai csak az adott felhasználó és csak az alkalmazás számára legyenek láthatók – habár van mód egyéb alkalmazások számára elérhetővé tenni őket.

Alkalmazás komponensek

- **Aktivitások (Activities)**
  - Vizuális felület a felhasználó számára a feladat elvégzéséhez
  - Egy alkalmazás egy vagy több aktivitásból állhat
  - Nézet osztály leszármazottjai használhatók
- **(Háttér)szolgáltatások (Services)**
  - Nincs UI, háttérben fut
  - Pl. zenelejátszás, pozíciófigyelés
- **Broadcast-fogadók (Broadcast receivers)**
  - Nincs UI, de indíthat aktivitást, vagy adhat értesítést
  - Tetszőleges számú üzenetre reagálhat
  - Pl. időzóna változott, kép elkészült, bejövő hívás, rendszer nyelvi beállítása változott
- **Tartalomszolgáltatók (Content providers)**
  - Adatokat szolgáltatnak más alkalmazásoknak
  - Pl. kontakt információk

Komponensek aktiválása

- **„Intent” objektum**
  - Aszinkron üzenetek
    - Aktivitások, broadcast-fogadók, szolgáltatások aktiválására
    - Ezek közötti kommunikációra, paraméterátadásra
  - Fő típusai
    - **Direkt**: megmondjuk pontosan melyik csomag hajtódjon végre
    - **Indirekt**: szándékot közlünk egy feladat végrehajtására (pl. email küldése, FTP, …), választhatunk a rendszerben regisztrált alkalmas végrehajtó komponensek közül
- **Feladatok**
  - Aktivitások sorozata
    - Akár más alkalmazás aktivitása is lehet
  - Aktivitás verem
  - „Vissza” gomb: előző aktivitás a veremből

AndroidManifest.xml

- **Komponensek adatainak összefoglalására**
  - Minden APK csomag tartalmazza
  - Rögzített a neve
- **Tartalma**
  - A komponenshez tartozó aktivitás objektum neve, ikonja, címkéje
  - Intent szűrők (mire reagál a komponens)
  - Engedélyek (milyen engedélyköteles szolgáltatásokat használ: pl. telefonhívások kezdeményezése, GPS, kontakt adatokhoz hozzáférés)
  - …

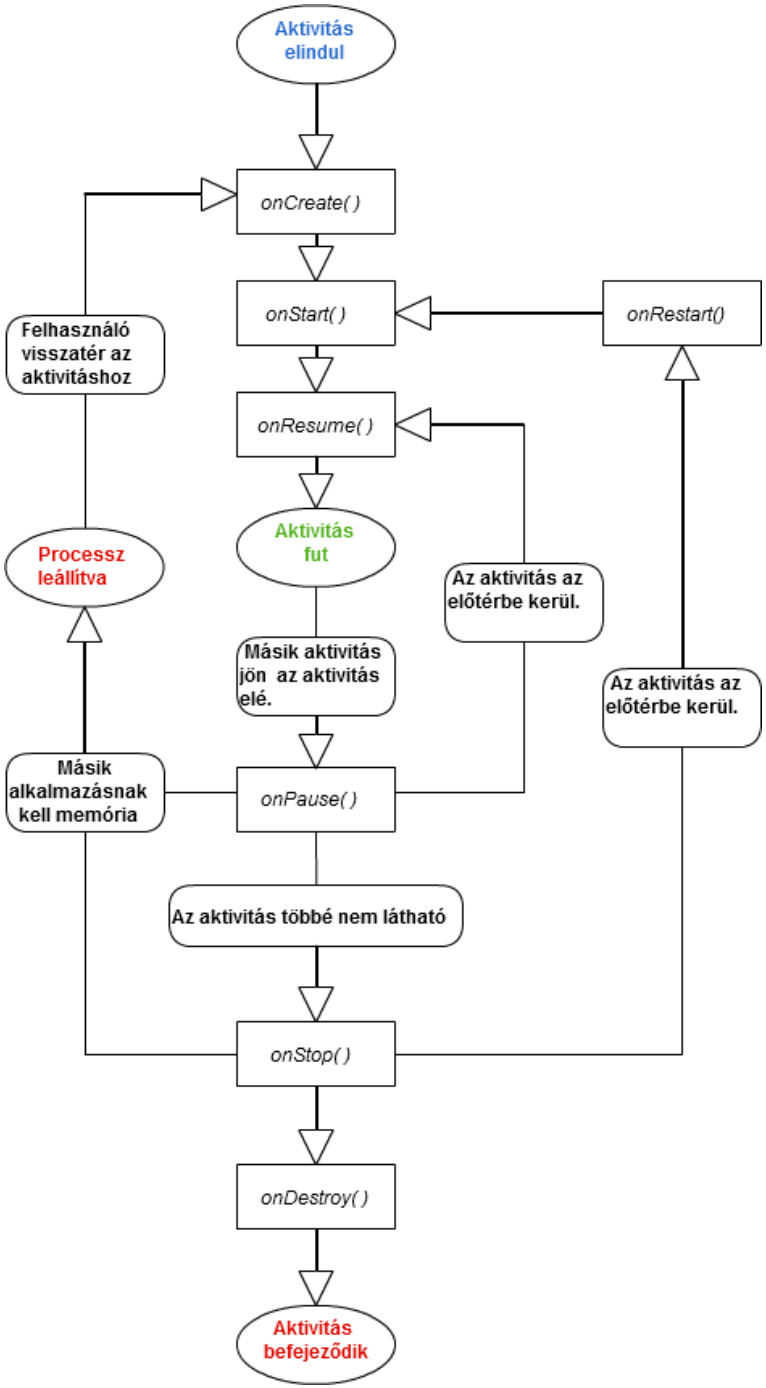
Komponensek életciklusa

- Egy aktivitásnak lényegében 3 állapota van:
- **Aktív vagy fut,** amikor **a képernyőn az előtérben van** (az aktivitás veremben legfelül helyezkedik el). Ez az aktivitás kezeli a felhasználói tevékenységet.
- **Felfüggesztett**, ha nem az aktivitás reagál a felhasználói tevékenységre, de még látható a felhasználó számára.
  Akkor van, amikor egy másik alkalmazás az addig aktív/futó felé kerül, és ezen aktivitás vagy átlátszó, vagy nem tölti ki az egész képernyőt, szóval néhány megállított alkalmazás látszódhat. Egy megállított aktivitás még teljes mértékben megmarad (fenntart minden állapot információt, és továbbra is csatlakozik az ablak kezelőhöz), de a rendszer véget vethet neki nagyon kevés rendelkezésre álló memória esetén.
- **Megállított**, ha teljesen háttérbe kerül egy másik aktivitás miatt. Továbbra is fenntartja minden állapot és belső információját. Többé nem látható a felhasználó számára, szóval a hozzá tartozó ablak rejtett, és **általában a rendszer megállítja**, amikor valahol szükség van memóriára.

- Ha egy aktivitás felfüggesztett vagy megállított állapotban van, a rendszer kitörölheti a memóriából anélkül, hogy **„megkérné a befejezésre”** (*finish( )* eljárás meghívása), **vagy egyszerűen csak megszüntetheti a processzét**. Amikor újra a felhasználó elé kerül, **teljesen újraindítandó és visszaállítandó az előző állapotába**.
- Mivel egy aktivitás állapotból állapotba vált, a változásokat a következő védett (*protected*) eljárásokkal van kezelve:

  void onCreate(Bundle *savedInstanceState*)
  void onStart()
  void onRestart()
  void onResume()
  void onPause()
  void onStop()
  void onDestroy()

- Ezen eljárások az állapotok nyomon követésére valók, így **felüldefiniálhatók**, hogy az elvárt módon reagáljanak az állapotváltozásokra.
- **Minden aktivitásnak implementálnia kell onCreate() eljárását**, hogy elvégezze a kezdeti beállításokat az objektum első példányosításakor.
- Sok alkalmazás az *onPause()* eljárást is implementálja, hogy kezelje az adatváltozásokat és ezen felül felkészül a felhasználóval való együttműködés befejezésére.



- *onCreate()*
  Az aktivitás első meghívásakor hívódik meg. Itt kell elvégezni az összes statikus beállítást: nézetek készítése, adatok lekötése listákhoz stb. Egy *Bundle* objektumot vár paraméterben, mely tartalmazhatja az aktivitás előző állapotát. Mindig *onStart()* követi.

- *onRestart()*
  Az aktivitás megállítása után hívódik meg, **közvetlenül azelőtt, hogy újra elindulna.** Mindig *onStart() követi.*

- *onStart()*
  Közvetlen azelőtt hívódik meg, hogy az aktivitás láthatóvá válna a felhasználó számára.

- *onResume()*
  Közvetlenül azelőtt hívódik meg, hogy az aktivitás interakcióba lépne a felhasználóval. Ilyenkor az aktivitás az aktivitás verem tetején van, a felhasználói bemenetelet ő fogadja. Mindig *onPause()* követi.

- *onPause()*
  Akkor hívódik meg, **amikor a rendszer egy másik aktivitást akar folytatni.** Általában adatváltozások mentésére, animációk és egyéb CPU-igényes feladatok leállítására használják. Bármit is csinál, azt **nagyon gyorsan kell tennie**, mivel a következő aktivitás addig nem folytatódik, amíg vissza nem tér.

- *onStop()*
  Akkor hívódik meg, **ha az aktivitás többé nem látható a felhasználó számára.** Ez történhet, ha az aktivitás megszűnik, vagy ha egy másik folytatódik, és elfedi ezt.

- *onDestroy()*
  Az aktivitás megszűnésekor hívódik. Ez történik, ha az aktivitás befejeződik, vagy ha a rendszer ideiglenesen megszűnteti ezt az aktivitáspéldányt. A két eset között az *isFinshing()* metódussal lehet különbséget tenni.

Aktivitások állapotmentése

- Ha a rendszer memória felszabadítása végett leállít egy aktivitást, **a felhasználó vissza akarhat térni az aktivitásba, annak legutóbbi használt állapotába.** Hogy megörökítse ezt az állapotot mielőtt az aktivitás megszűnne, implementálható az aktivitás *onSaveInstanceState()* metódusa. Az Android meghívja ezt az eljárást, azelőtt, hogy az aktivitás megszűnése fenyegetne, azaz mielőtt az *onPause()* eljárás meghívódna. A rendszer átad egy *Bundle* obejktumot, így az aktivitás állapota név-érték páronkként elmenthető. Ha az aktivitás újraindul, a *Bundle* átadódik mind az *onCreate()*, mind az *onStoreInstanceState()* eljárásnak, mely utóbbi az *onStart()* után hívódik meg, így bármelyikben vagy mindkettőben visszaállítható az állapot.
- Az *onPause()* és a többi fentebb felsorolt eljárással szemben *onSaveInstanceState()* és *onStoreInstanceState()* nem életciklus-metódusok. Például, az Android rendszer nem hívja meg az *onStoreInstanceState()* eljárást, ha az aktivitás felhasználói tevékenység következtében szűnik meg (pl. „Vissza” gomb). Ebben az esetben ugyanis a felhasználó nem kíván visszatérni az aktivitáshoz, így nincs ok annak állapotának elmentésére.
- Amiért *onSaveInstanceState()* nem hívódik meg mindig, csak átmeneti állapot mentésére használandó. Állandó adatok mentésére az *onPause()* metódus ajánlott.

Aktivitásváltások koordinálása

- Amikor egy aktivitás elindít egy másikat, **mindkettő életciklusában változások következnek be**. Az egyik felfüggesztődik vagy meg is állhat, míg a másik elindul.
- **Az életciklus visszahívások/eljárások sorrendje meghatározott:**
  - Az aktuális aktivitás *onPause()* metódusa meghívódik.
  - Ezután az új aktivitás *onCreate(), onStart()* és *onResume()* metódusa sorban meghívódnak.
  - Majd amikor az újonnan elindított aktivitás nem látható a kijelzőn, az *onStop()* metódusa meghívódik.

Processzusok életciklusa

Az Android rendszer igyekszik megtartani egy alkalmazás processzt ameddig csak lehetséges, de időnként szükséges eltávolítani régi processzeket, ha a memória elfogy. Meghatározandó mely processzeket tartsa meg vagy szüntesse meg, a rendszer „fontossági hierarchiába” sorolja a processzeket a bennük futó komponensek és azok állapota alapján. 5 szint van a hierarchiában, a következőkben ezek láthatóak fontosság szerint csökkenő sorrendben:

- **Előtérben lévő processz**, mely ahhoz szükséges, amit a felhasználó éppen csinál. Egy processz az előtérbe kerül bármely következő fennállása esetén.
  - Olyan aktivitást futtat, amellyel a felhasználó interakcióban van (ilyenkor az *Activity* objektum *onResume()* metódusa meghívódik).
  - Olyan szolgáltatást tartalmaz, amely szükséges a felhasználóval interakcióban lévő aktivitáshoz.
  - Tartalmaz egy *Service* objektumot, amely egy életciklus visszahívást hajt végre (*onCreate(), onStart()* vagy *onDestroy()).*
  - Tartalmaz egy *BroadcastReceiver* objektumot, amely az *onReceive()* eljárását hajtja végre.
  - Csak néhány előtérben lévő processz létezhet egy adott pillanatban. Legvégső esetben megszűnik egy, ha annyira kevés a memória, hogy nem tudja mind folytatni a futását. Általában ilyenkor a memória betelik, tehát a felhasználói interfész működőképessége érdekében meg kell szüntetni néhány processzt.
- **Látható processz** az, melynek nincs előtérbeli komponense, de hatással lehet arra, amit a felhasználó a képernyőn lát. Egy processz látható, ha az alábbiak valamelyike teljesül:
  - Olyan aktivitást nyújt, amely bár nincs a előtérben, látható a felhasználó számára (*onPause()* metódusa meghívódott). Ez történik például akkor, ha az előtérben lévő aktivitás egy párbeszéd ablak, amely mögött még látszódik az előző aktivitás.
  - Olyan szolgáltatást nyújt, amely szükséges egy látható aktivitás számára.

- **Szolgáltatás processz** az, mely futtat egy szolgáltatást, mely a *startService()* metódussal lett elindítva, de nem tartozik az előző két kategóriába. Bár a szolgáltatás processzek nem köthetők közvetlenül valamihez, amit a felhasználó lát, olyan dolgokat csinálnak, amivel a felhasználó törődik (pl. mp3 lejátszása a háttérben, adat letöltése a hálózaton), így a rendszer futtatja ezeket, hacsak nincs elég memória, hogy az előtérbeli és látható processzekkel együtt fussanak.
- **Háttér processz** az, mely olyan aktivitást tartalmaz, amely éppen nem látható a felhasználó számára (az aktivitás *onStop()* eljárása meg lett hívva). Ezen processzeknek nincs közvetlen hatásuk a felhasználói élményre, és bármikor megszüntethetők annak érdekében, hogy egy fontosabb processznek legyen memóriája. Általában sok háttér processz fut, így azok egy **LRH (legritkábban használt**, angolul Last Recently Used) **listában** vannak nyilvántartva biztosítva, hogy a felhasználó által a korábbiakban legtöbbet használt aktivitás processze legyen legkésőbb megszüntetve. Ha egy aktivitás életciklus eljárásait helyesen implementálja és megőrzi az aktuális állapotát, a processz törlésének nem lehet káros hatása a felhasználói élményre.
- **Üres processz** az, mely **nem tartalmaz aktív alkalmazás összetevőt**. Az üres processz olyan folyamat, amelynek nincs aktív komponense. Ilyen folyamatokat csak cache gyanánt használunk, hogy amikor egy komponens legközelebb futtatni akarja, gyorsabban induljon. A rendszer gyakran kilövi ezeket a folyamatokat, hogy a rendszererőforrások kiegyenlítetten legyenek elosztva a folyamatcache-ek és a mögöttes kernelcache-ek között.

Felhasználói felület (UI)

- Egy Android alkalmazásban **a felhasználói interfész** *View* (nézet) és *ViewGroup* (nézetcsoport) elemekből épül fel. Sokféle nézet és nézetcsoport létezik, mindegyik a *View* osztály leszármazottja.
- A nézet objektumok a felhasználói interfész alapegységei. A ***View*** osztály alapját képezi a **widget**-eknek (vezérlőknek) nevezett alosztályoknak, melyek kész grafikus elemeket jelentenek úgy, mint szövegmezők és gombok.
- A ***ViewGroup*** osztály alapját képezi a **layout**-oknak (elrendezéseknek) nevezett alosztályoknak, melyek különböző elrendezésű megjelenítést tesznek lehetővé úgy, mint sorban, táblázatban, valamihez képest relatívan elhelyezett elemek.
- **Egy *View* objektum egy olyan adatszerkezet, amelynek paraméterei tárolják az elrendezés tulajdonságait és tartalmát a képernyő egy adott téglalap alakú részéhez.** A *View* objektum kezeli a saját méretét, elrendezésének (layout) rajzolását, fókuszváltozását, görgetését/lapozását és a képernyő adott részéhez tartozó felhasználói beavatkozásokat (billentyűk, gesztusok). Egy felhasználói interfészen belüli objektumként, egy *View* szintén lehet a felhasználó számára egy visszajelzési pont és a felhasználói beavatkozásokhoz tartozó események kezelője.

Nézet hierarchia

- Egy tevékenység (activity) felhasználói interfésze egy hierarchiaként definiálható, melynek csomópontjai *View* és *ViewGroup* objektumok.
- Ez a hierarchia-fa olyan egyszerű vagy összetett lehet, amilyenre csak szükségünk van, és felépíthető mind az Androidban előre definiált widget és layout objektumok, mind egyéni vezérlők és elrendezések felhasználásával, melyeket mi készítünk el.
- Gyökérelem: ***setContentView(…)***
- Minden *ViewGroup* objektum felelős felszólítani *View* gyermekeit, hogy „rajzolják meg magukat”.
- **A gyermekei kérhetnek egy méretet és elhelyezést** a szülőn belül, és **a szülő objektum dönt**, hol és mekkora lehet minden gyerek objektum.

Elrendezés (Layout)

- Az elrendezést és nézet hierarchiát általában egy **XML elrendezés fájl** definiálja.
- **Minden elem az XML-ben vagy *View* vagy *ViewGroup* objektum** (vagy azok leszármazottja). A *View* objektumok levelek, a *ViewGroup* objektumok elágazások a fában.
- Egy XML elem neve megfelel az őt megvalósító Java osztálynak, például: TextView, LinearLayout.
- Amikor betöltünk egy elrendezést, az Android rendszer **inicializálja ezen futás idejű objektumokat**, a benne lévő elemeknek megfelelően.

Vezérlő elemek

- Egy **vezérlő elem** egy olyan *View* objektum, amely **a felhasználóval való közreműködésre szolgáló interfész**.
- Az Android biztosít **néhány teljesen implementált vezérlő elemet (gombok, választó mezők, szövegbeviteli mezők)**, ezekkel gyorsan építhető egy UI (felhasználói interfész).
- Az Android néhány vezérlő eleme **összetettebb**, mint például **egy adat begyűjtő, egy óra vagy egy zoom kezelő.**
- De nem vagyunk az Android nyújtotta widgetekre korlátozva. Létrehozható saját, specifikus elem **saját *View* objektum definiálásával**, vagy már létezők kiterjesztésével, kombinálásával.

UI események

- Egy hozzáadott View/widget elem esetén nyilván tudni akarunk a vele való felhasználói interakcióról, hogy egy bizonyos működést végrehajtva válaszolhassunk rá. Az UI események kezelésére két lehetőség van:
  - **Definiálni egy esemény figyelőt, és hozzárendelni az adott *View* objektumhoz.** Általában így figyeljük az eseményeket. A *View* osztály beépített interfészeket tartalmaz *On<valami>Listener* névvel, mindegyikhez egy *On<valami>()* visszahívható függvény tartozik.
    - Például *View.OnClickListener* („kattintások” kezelésére), *View.OnTouchListener* (érintőképernyő eseményeinek kezelése), és *View.OnKeyListener* (fizikai gombok kezelése). Tehát, hogy egy *View* objektum értesüljön arról, hogy rákattintottak (pl. egy gomb megnyomása által), implementálni kell *OnClickListener*-t definiálva az *onClick()* eljárását (ahol reagálhatunk a kattintásra) és hozzárendelve azt a *View*-hoz a *setOnClickListener()*-rel.
  - **Valamely *View*-hoz tartozó visszahívható eljárás felüldefiniálása.** Ez a teendő saját *View* típusú osztály megvalósításakor. Többek között kezelhető így a képernyő megérintése (*onTouchEvent()*), a joystick megmozdítása (*onTrackballEvent()*) vagy egy gomb megnyomása az eszközön (*onKeyDown()*). Ezáltal megadható minden eseményhez az alapértelmezett viselkedés, és meghatározható, hogy az esemény tovább legyen-e adva valamely gyerek *View*-nak. Tehát, mivel ezek a *View* osztály visszahívható eljárásai, az egyetlen lehetőség definiálni őket a saját komponens létrehozása.

Elrendezések (layout) megadása

Kétféleképpen adható meg az elrendezés:

- **UI elemek megadása XML-ben**
  - Az Android egyértelmű XML kulcsszavakkal rendelkezik a View osztályok és leszármazott osztályoknak megfelelően.
  - Ezen megoldás előnye, hogy az alkalmazás megjelenéséért felelős kódot elválasztja a viselkedésért felelőstől. Az UI leírás kívül van az alkalmazás kódján, azaz módosítható, átdolgozható a forráskód módosítása és újrafordítása nélkül.
- **Elrendezés elemek példányosítása futás közben**
  - Az alkalmazások képesek létrehozni *View* és *ViewGroup* objektumokat és befolyásolni azok tulajdonságait programból.

Az Android keretrendszer lehetővé teszi bármelyik, vagy mindkettő használatát is.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android=http://schemas.android.com/apk/res/android
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >
        <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
        <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

XML betöltése

- Az alkalmazás lefordításakor minden XML layout fájlból létrejön egy *View* erőforrás**.**
- Ezt az erőforrás fájlt be kell tölteni az alkalmazás kódjában, az *Activity.onCreate()* visszahívható eljárásból. Ez a *setContentView()* meghívásával történhet, amelynek át kell adni az erőforrásra való hivatkozást *R.layout.<elrendezés_fájl_neve>* alakban. Például, ha az XML main_layout.xml néven van elmentve, így tölthető be:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

- Az *onCreate()* visszahívható eljárást a Aktivitásban az Android keretrendszer hívja meg, amikor elindítjuk az alkalmazást.

XML attribútumok

- Minden *View* és *ViewGroup* objektum saját XML attribútumokat támogat.
- Vannak specifikus attribútumok egy View objektumhoz (például a *TextView* támogatja a *textSize* attribútumot), de ezek az attribútumokat örökli bármely View objektum, amely kiterjeszti az osztályt.
- Néhány közös minden View objektum számára, mert a gyökér View objektumból van örökölve (például az *id* attribútum).
- És vannak még az „elrendezés paraméterek”, amelyek a View objektum konkrét elhelyezkedését adják meg, ahogy a szülő ViewGroup objektumban meg van határozva.

- ID
  - **Bármely View objektum társulhat egy egész szám azonosító**, mely egyértelműen azonosítja az objektumot a fán belül. Az alkalmazás lefordításakor erre az ID-ra egész számként hivatkoznak, de tipikusan az XML fájlban van meghatározva karakterláncként, az id attribútumban. Ez egy minden View objektum számára elérhető attribútum (View osztályban által van megadva), és gyakran használatos. A szintaxis az XML-en belül a következő:

```
ANDROID:ID="@+ID/MY_BUTTON”
```

  - A kukac jel (**@**) az elején **azt jelzi, hogy az XML elemzőnek elemeznie kell és ki kell terjesztenie a sztring hátralévő részét, és egy ID erőforrásként kell beazonosítania.**
  - A plusz jel (**+**) **azt jelenti, hogy ez egy új forrás neve, amit létre kell hozni és hozzá kell adni az erőforrásainkhoz (az R.java fájlban).**
  - Számos, az Android keretrendszer által nyújtott, egyéb ID erőforrás létezik még. Amikor egy Android erőforrás ID-re hivatkozunk, nincs szükség a plusz jelre, de meg kell adni a csomag névterét, például:

```
ANDROID:ID="@ANDROID:ID/EMPTY"
```

Megadva a névteret, egy az android.R erőforrás osztályban található ID-re hivatkozunk, a helyi erőforrás osztály helyett.

Erőforrások a kódban

Nézetek készítésére és az alkalmazásból rájuk való hivatkozásra egy minta:

- Definiálni egy view/widget-et az elrendezés fájlban, és megadni hozzá egy azonosítót:

```
<Button android:id="@+id/my_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/my_button_text"/>
```

- Majd példányosítani a View objektumot és összekötni az erőforrás elemmel (tipikusan az *onCreate()* eljárásban):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Elrendezés paraméterek

- A *layout_<valami>* nevű XML elrendezés attribútumok a View számára elrendezés paramétereket határoznak meg a, amelyek megfelelnek a ViewGroup objektumnak, amelyben az megtalálható.
- Minden ViewGroup objektum implementál egy beágyazott osztályt, amely a *ViewGroup.LayoutParams*-ot terjeszti ki. Ez az alosztály különböző típusú tulajdonságokat tartalmaz, melyek minden gyermek objektum méretét és pozícióját megadják.
- Minden LayoutParams alosztály meg van a saját szintaxisa az értékek beállítására. Minden gyerek elemnek meg kell adnia a szülőnek megfelelő LayoutParams paramétereket, noha az szintén megadhat különböző paramétereket a saját gyerekeinek.
- **Minden nézetcsoport tartalmaz szélességet és magasságot** (*layout_width* és *layout_height*)**, és minden nézetnek definiálnia kell azokat.** Sok LayoutParams tartalmaz még margókat és szegélyeket is.
- Megadható egzakt mértékként a szélesség és a magasság, habár ez ritka. Gyakoribb a következő konstansok valamelyikének használata:
  - *wrap_content* a nézetet a tartalmának megfelelő méretűre méretezi
  - *fill_parent* (*match_parent* az API level 8-tól kezdődően) a nézetet akkorára méretezi, amekkorának hagyja a szülő nézetcsoportja.
- Általában, **egy elrendezés méreteinek abszolút egységekkel (pl. pixel) való megadása nem ajánlott.** Ehelyett relatív mértékek, mint például tömörség-független pixel egység (*dp*), *wrap_content, fill_parent*, használata egy jobb megközelítés, mert ez segít biztosítani, hogy az alkalmazás helyesen jelenjen meg különböző méretű kijelzőkön.

Tutorialok

- Android Wireless Application Development könyv
  - Chapter06_code.zip Chater07_code.zip, Chapter08_code.zip
- Példaprogramok
  - Elements.zip
  - AritmTest_01.zip, AritmTest_02.zip, AritmTest_03.zip
- CodeProject
  - Learn How to Develop Android Application
  - Android UI Layouts and Controls
  - Beginner's Guide to Organizing/Accessing Android Resources
  - Android Touch Gestures Capturing Interface
  - Article 5 - Android User Interactivity and Sensors
- Android Developer portál
  - Training for Android developers
  - Notepad Tutorial (Android Developer Portal)