

Android

Service-ek fejlesztése

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



Department of
Automation and
Applied Informatics

Tartalom

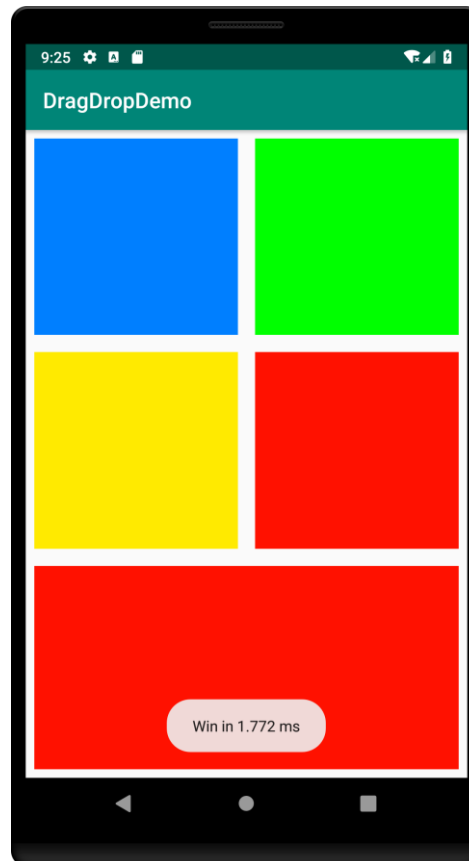
- Érdekesség: Drag&Drop
- Service típusok
- Background Service
- Foreground Service
- IntentService
- Bind Service

Drag & Drop támogatás

- Gyakorlatilag bármelyik komponens „drag”-elhető:
 - > `View.startDrag(...)`
- Felületek/komponensek feliratkozhatnak „drag” esemény érzékelésre:
 - > `setOnDragListener(...)`
- Többféle „drag” esemény:
 - > `ACTION_DRAG_STARTED`
 - > `ACTION_DRAG_ENTERED`
 - > `ACTION_DRAG_EXITED`
 - > `ACTION_DROP`

Gyakorljunk!

- Készítsünk egy színválasztó Drag&Drop játékot reakció idő mérésre!



SERVICE KOMPONENTENS

Bevezetés

- Összetett alkalmazások esetén gyakran szükség van **háttérben futó folyamatokra**, amelyek **felhasználói felület nélküli** funkcionalitást valósítanak meg (pl. hálózati kommunikáció, monitorozás, zene lejátszás, fájl feltöltés, stb...)
- Android alkalmazáskomponens: **Service**
- Többféle Service típus és viselkedési modell
- Ügyeljünk a megfelelő leállításra és az erőforrás felszabadításra!
- Minden Service létrehozás komoly felelősség a fejlesztő részéről!

Service bemutatása

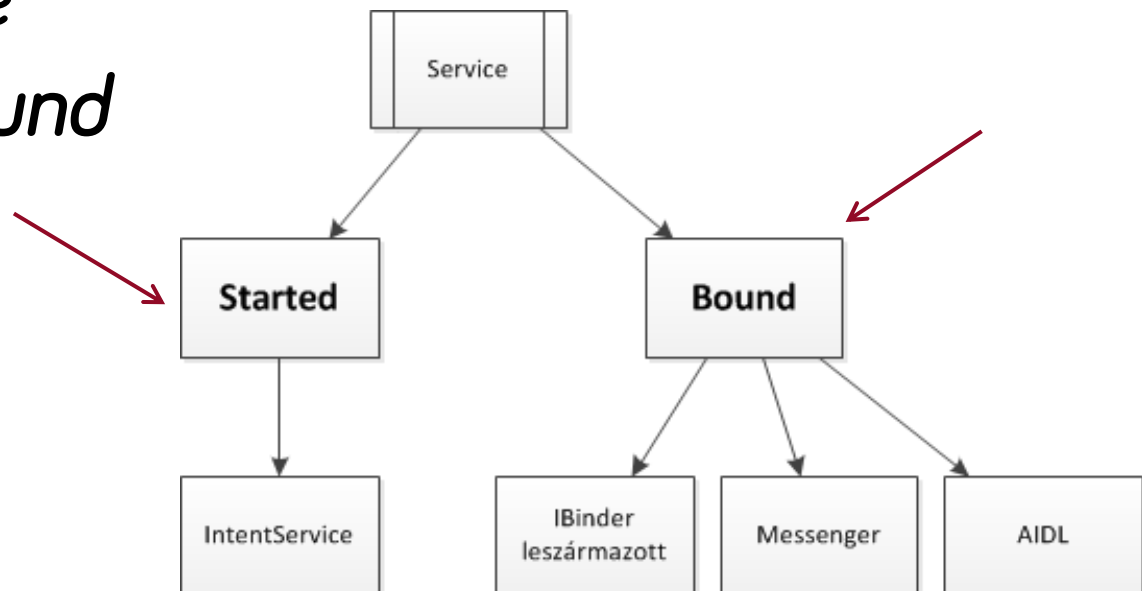
- Valamilyen hosszabb ideig tartó, háttérben futó feladatot lát el
- Felhasználói beavatkozás nélkül működik
- Nincs felhasználói felülete
- Más komponensek/alkalmazások számára is szolgáltathat funkciókat
- Bármilyen alkalmazáskomponens elindíthatja és **akkor is futva maradhat, amikor a hívó (pl. Activity) megáll**, tipikusan ha más alkalmazásra váltunk

Service bemutatása

- Más alkalmazások komponensei is kapcsolódhatnak hozzá és kommunikálhatnak vele (*Interprocess Communication, IPC*)
- (ahogy egy Activity is elindítható másik alkalmazásból, de a **Service**-nél ez letiltható)
- Ugyanúgy **Intent**-el indítjuk el
- Példa szolgáltatások:
 - > Zenelejátszó
 - > Hosszabb hálózati kommunikáció (pl. torrent)
 - > GPS pozíció követés - rendszerszolgáltatás
 - > Stb...

Service típusok/működési módok

- Egy Service kétféle módon képes működni:
 - > *Komponensből indított / **Started***
 - *Foreground (kötekező Notification)*
 - *Background*
 - *IntentService*
 - > *Kapcsolt / **Bound***



Background Service limitációk

- Android 8-tól felfele
 - > De korábbi verziókban is beállítható manuálisan
- Limitált végrehajtás, sokkal nagyobb valószínűséggel leállítja a rendszer
- Néhány perc után idle-be állítja az alkalmazást, ami olyan mintha a `stopSelf()` meghívódott volna

Service működési típusok

1. Komponensből indított (***Started/Unbound***):
 - > Valamilyen alkalmazás komponens (tipikusan Activity) elindítja a ***startService(intent)*** metódussal
 - > A Service akkor is folytathatja a futását, amikor a hívó komponens már megsemmisült
 - A hívó megállítása nem törli a Service-t is!
 - > Általában az így indított Service-ek **egy feladatot hajtanak végre**, nem folytonos szolgáltatást nyújtanak (például egy darab fájl feltöltése vagy letöltése)
 - > Csak a hívónak van rá referenciája
 - > Ha végzett, **magát kell leállítania** a *stopSelf()* metódussal vagy a hívónak *stopService(intent)*-el, az op.rendszer nem fogja!

Service működési típusok

2. Kapcsolt szolgáltatás (*Bound*):

- > Nem indítjuk „kézzel”, **magától indul** ha kapcsolódni próbálunk hozzá
- > Addig fut amíg van hozzá kapcsolódó komponens
- > **Egyszerre többen is kapcsolódhatnak**, akár más alkalmazásból is
- > **Hosszabb, folyamatosan tartó feladatokhoz**
 - Pl. Torrent, zenelejátszó
- > **Nem kell önmagát leállítania**, az Android gondoskodik róla
- > Gyakorlatilag minden **rendszer szolgáltatás** ilyen

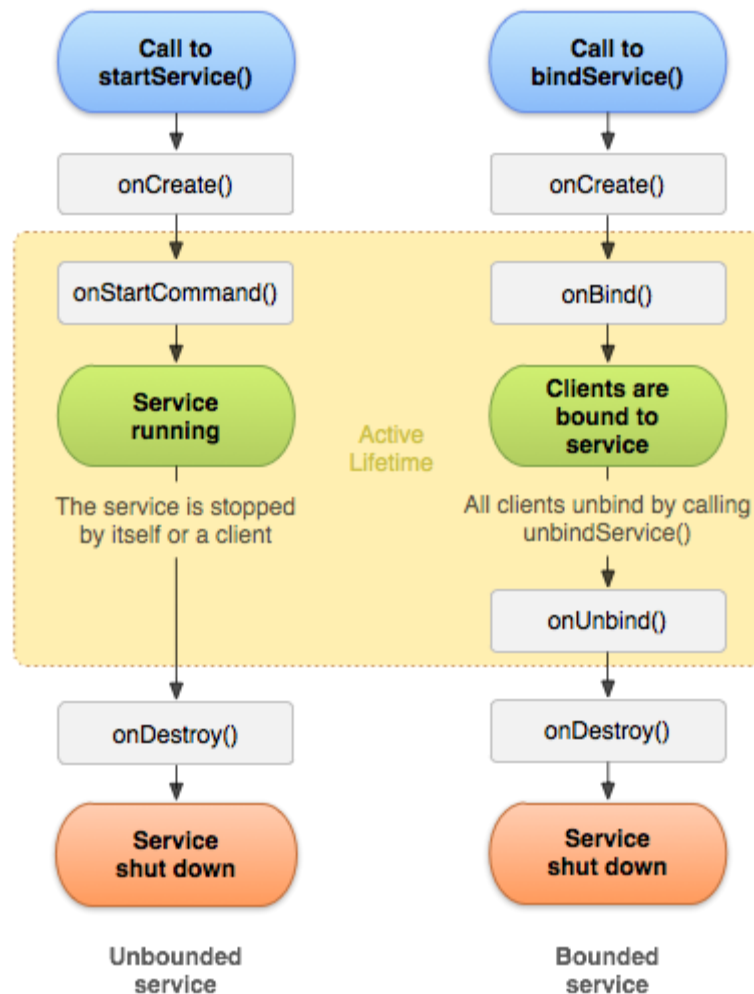
Service működési típusok

- A Service támogathatja **egyszerre mind a két típusú működést is** (!), néha szükséges is
- Started esetben nem tudunk referenciát szerezni a Service-re, ha a hívó (Activity) már nem fut
- Példa: zenelejátszó ami mindkét módot igényli
 1. A lejátszó UI-ról indítjuk a playlist lejátszást `startService()`-el
 2. Kilépünk az alkalmazásból, a Service fut a háttérben és megy a zene
 3. Visszatérünk a lejátszóba hogy számot váltsunk. Ekkor az új Activity példányból már nem tudunk referencát szerezni a Started Service-re, és megkérni hogy váltson a következő számra
 4. Tudunk viszont kapcsolódni a Service-hez, ha implementáltuk a Bound működési módot is
 5. Kapcsolódunk a Service-hez, és utasítjuk a számváltásra

Service megvalósítás

- Service (vagy beépített gyerek) osztályból származtatunk
- Megvalósítjuk a megfelelő callback függvényeket:
 - > Ha *Started*-ként (is) akarjuk használni: ***onStartCommand()*** implementálása
 - > Ha *Bound*-ként (is) akarjuk használni: ***onBind()*** implementálása

Service élelciklus modell



Forrás: <https://developer.android.com/guide/components/services>

Service kezelés

- Mindkét működés esetén más alkalmazás is használhatja a szolgáltatást
- Ugyanúgy, mint az Activity-t is – **Intentek** segítségével, akár más alkalmazásból is
- De! Lehetőség van **privát szolgáltatások** létrehozására is:
 - > Manifest attribútumban megadva (ld. később)
 - > Ekkor más alkalmazásból nem férnek hozzá a classname megadásával sem

Műveletek végrehajtása Service-n belül

- A Service alapértelmezetten a processze fő szálában fut, **nem kap külön szálát!!**
- Erőforrás igényes műveletek esetén „kézzel” kell új szálát indítanunk, pl.:
 - > CPU intenzív feladatok (pl. titkosítás, enkódolás, zene lejátszás)
 - > Hálózati kommunikáció, stb...
- Ellenkező esetben 5mp után ugyanúgy megjelenik az *Application Not Responding* (ANR) ablak, mint Activity esetén

Service készítése

1. Leszármaztatunk az *android.app.Service*-ből

```
import android.app.Service

class DemoService : Service() {
    ...
}
```

2. Szükséges callback függvények megvalósítása

```
override fun onCreate() {
    super.onCreate()
}

override fun onBind(intent: Intent): IBinder? {
    return null
}
```

3. Regisztráció a Manifestben

Fejlesztő által felüldefiniálандó

CALLBACK METÓDUSOK

Callback metódusok

1. onCreate() – *mindkét működési módnál*
2. onStartCommand() – *Started esetben*
3. onBind()/onUnbind()/onRebind() – *Bound esetben*
4. onDestroy() – *mindkét működési módnál*

Service példa

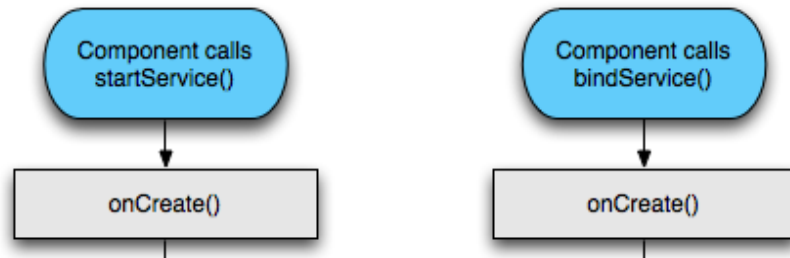
```
class DemoService : Service() {  
    private var mStartMode: Int = 0           // service leállítás esetén a működési mód  
    private var mBinder: IBinder? = null      // bind interfész  
    private var mAllowRebind: Boolean = false // újra bind-olás engedélyezett-e  
  
    override fun onCreate() {  
        // Elkészül a service  
    }  
  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        // A service elindul külső startService() hívás miatt  
        return mStartMode  
    }  
  
    override fun onBind(intent: Intent): IBinder? {  
        // Kliens kapcsolódik a Service-hez bindService()-el  
        return mBinder  
    }  
  
    override fun onUnbind(intent: Intent): Boolean {  
        // Minden kliens lecsatlakozott unbindService()-el  
        return mAllowRebind  
    }  
  
    override fun onRebind(intent: Intent) {  
        // Kliens újra csatlakozik bindService()-el, onUnbind() után  
    }  
  
    override fun onDestroy() {  
        // A service nem használható tovább és meg fog semmisülni  
    }  
}
```

Callbacks 1 -onCreate

- onCreate()

```
override fun onCreate() {  
    super.onCreate()  
    // inicializálás, szálak indítása, stb  
}
```

- > A Service legelső létrehozásakor fut le, elsőként
- > Akár *Started*, akár *Bound* módban indítják



- > Ha már elindult a Service, soha nem fog újra lefutni ez a függvény
- > Feladat: inicializálás

Callbacks 2 - onStartCommand

- onStartCommand()

```
override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {  
    // feladat inicializálás, szál létrehozás, stb.  
    return super.onStartCommand(intent, flags, startId)  
}
```

- > Ez hívódik *Started* esetben (ha *startService()*-el hozták létre)
- > Lefutása után a Service futó állapotban van, a háttérben bármeddig folytatható, itt **végzi el a feladatát**
- > **Innen kell új szálát indítani!** (javasolt megoldás: *AsyncTask*)
- > A **fejlesztő felelőssége** a zárása *stopSelf()*-el a Service-en belülről, vagy *stopService()*-el más komponensből
- > Ha csak *Bound*-ként akarjuk használni, akkor nem kell megvalósítani ezt a metódust
- > Visszatérési értéke egy int, ami megmondja **hogyan induljon újra** a Service, ha a rendszer megállította

Service újraindítás

- Service: háttérben kell futnia, hosszabb távon
- Az OS kilövi ha kevés a memória
- Kritikus feladat jól kezelni ezt az eseményt, előbb-utóbb biztosan megtörténik!
- Androidon 3 különböző módon kérhetünk újraindítást
- A Service kéri az OS-től, az *onStartCommand()* visszatérési értékével

onStartCommand – újraindítási mód

> Újraindítási módok

Service.START_STICKY:

Az OS mindig megpróbálja újraindítani ha kevés memória miatt le kellett állítani. Ebben az esetben újraindításkor az onStartCommand() kapott Intent paramétere *null* értékű.

Service.START_NOT_STICKY:

Csak akkor indítja újra az Android, ha az elpusztítása óta valaki megpróbálta indítani *startService()* hívással. Tipikusan olyan esetben használatos, ha a Service-ben *egy darab* (hosszú ideig tartó) műveletet végzünk, pl. feltöltés, letöltés, stb. Alapesetben *stopSelf()*-el fejezi be a futást.

onStartCommand – újraindítási módok

> Újraindítási módok

Service.START_REDELIVER_INTENT:

Az előző kettő kombinációja. Akkor indul újra, ha vannak függő *startService()* hívások rá, vagy még nem futott le a *stopSelf()* metódusa. Ha újraindul, akkor hívódik az *onStartCommand()* az eredeti Intent-el. Akkor érdemes használni, ha visszajelzést várunk a műveletvégzés sikeréről

onStartCommand – újraindítási mód

Mindhárom mód esetén a fejlesztőnek kell gondoskodnia a Service leállításáról **stopSelf()** vagy **stopService()** hívással!

Tipikus implementáció: **új szál**at indítunk az *onStartCommand()*-on belül, **ami már független a UI-tól**, és abban végezzük az időigényes feladatot. Használjunk AsyncTask-ot!

onStartCommand – újraindítási mód

- > **Android 2.0 előtt** (API level 5) **nincs ilyen metódus**, helyette *onStart()*. Ennek implementációja ugyanaz mint az *onStartCommand()*, ami mindig `START_STICKY`-vel tér vissza
- > Az újraindítási mód beállítása **befolyásolja a többi paraméter értékét is**: újraindítás után csak `REDELIVER_INTENT` esetben kapjuk vissza az eredeti Intent-et

onStartCommand - Intent

- Miért **nagyon fontos** hogy újraindításkor visszakapja-e az Intent-et?

Mert az Intent-ben adatot kap(hat), ami megmondja mit kell csinálnia

- Melyik fájlt kell letölteni
- Melyik zenét kell lejátszani
- stb...

onStartCommand - flags

- > Értéke lehet **START_FLAG_REDELIVERY** ha újraindult, **START_FLAG_RETRY** ha **STICKY**-re volt állítva, valamilyen hiba miatt állt meg, és az Android megpróbálja újra futtatni.

Callbacks 3– onBind

- *onBind()*

- > Akkor hívja meg a rendszer ezt a függvényt, ha egy másik komponens hozzá akar csatlakozni a szolgáltatáshoz a *bindService()* hívás segítségével
- > Kötelező egy *IBinder* interfész implementációval visszatérni, amivel majd a szolgáltatással tud a másik fél kommunikálni (ld később)
- > **Kötelező felüldefiniálni**, de ha nem akarjuk ezt a funkciót támogatni, *null*-al térjünk vissza

Callbacks 4 – onDestroy

- ***onDestroy()***:

- > A szolgáltatás végleges bezárása előtt hívódik meg
- > Feladat: erőforrások felszabadítása.
 - Indított szálak
 - Regisztrált listener-ek, receiver-ek
 - Hálózati kapcsolatok
 - Minden ami felszabadítható
- > Ez az utolsó értesítés, amit a *Service* kap, mindent itt kell elengedni

Service élettartama

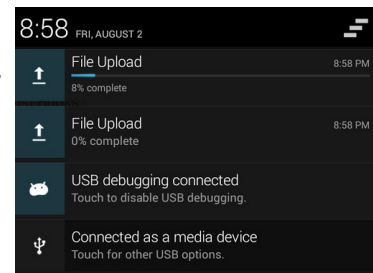
- A hosszú ideje futó szolgáltatások prioritását az Android folyamatosan csökkenti
- Kevés memória esetén az alacsony szintű komponenseket kezdi el kilőni
- Ha olyan Service-t írunk, ami valószínűleg pár percnél tovább fog futni, akkor fel kell készíteni a rendszer általi újraindításokra
- Kivéve az előtérben futó Service-ek

Background Service limitációk

- Android 8-tól felfele
 - > De korábbi verziókban is beállítható manuálisan
- Limitált végrehajtás, sokkal nagyobb valószínűséggel leállítja a rendszer
- Néhány perc után idle-be állítja az alkalmazást, ami olyan mintha a `stopSelf()` meghívódott volna

Előtérben futó Service - Foreground

- Olyan szolgáltatás, mely futásának a felhasználó „tudatában van”
 - > Pl. Zene lejátszás, torrent, USB kapcsolat
- Ezért a rendszernek csak a végső esetben szabad megállítania
- Kérhetünk magas prioritást a szolgáltatásunknak
- „*Előtérben futó Service*” (*Foreground Service*)
- ***startForeground()*** hívással **helyezhető** előtérbe
- Kötelezően meg kell adni egy Notification-t, ami „*Ongoing*”-ként látszik



Előtérben futó Service

- Ha már nincs szükség arra, hogy előtérben fusson...
 - > Végetért a playlist, letöltődött a fájl, kihúzták az USB kábelt, stb
- *...stopForeground()* -al visszatehető háttérbe
- Ez **nem állítja meg a Service-t**, csak kiveszi az előtérből
- Paraméterben átadható, hogy a Notification-t is törölje, vagy az maradjon
- (a Notification mindenképp törlődik, ha előtérben lévő Service-t megállítunk)

Service leállítása

- A rendszer csak akkor kezdi leállítani a szolgáltatásokat, ha az előtérben lévő *Activity*-nek memóriára van szüksége
- Ha egy Service hozzá van kötve egy *Activity*-hez (*bind*), akkor kisebb eséllyel kerül leállításra
- *Foreground* típusú szolgáltatást csak a legvégső esetben állít le a rendszer
- Amennyiben újra rendelkezésre áll a memória, a rendszer megpróbálja újraindítani a szolgáltatást!
- A megfelelő leállás, újraindulás kezelése és erőforrás felszabadítás a **fejlesztő felelőssége!**

MANIFEST ATTRIBÚTUMOK

Service attribútumok 1/2

```
<service android:enabled=["true" | "false"]  
        android:exported=["true" | "false"]  
        android:icon="kép erőforrás"  
        android:label="szöveg erőforrás"  
        android:name="név"  
        android:permission="engedély"  
        android:process="process név" >  
    . . .  
</service>
```

Service attribútumok 2/2

- ***enabled***: A rendszer indíthatja-e (miért jó hogy állítható?)
- ***exported***: Más alkalmazás komponensei kapcsolódhatnak-e hozzá, privát lesz ha *false*-ra állítjuk
- ***icon***: Szolgáltatás ikon, ha launcher-ből indítható
- ***label***: *Settings/Running services*-ben megjelenő felirat
- ***name(*)***: Melyik osztályban található az implementáció
- ***permission***: Milyen engedély szükséges ehhez a szolgáltatáshoz történő kapcsolódáshoz
- ***process***: Melyik processzben fusson a szolgáltatás. Ha kettősponttal kezdődő nevet adunk meg, akkor újat hoz létre a rendszer a szolgáltatáshoz

STARTED SERVICE PÉLDA

Started Service létrehozás

- Tipikus scenárió, amikor **Started** típusú Service szükséges:

Activity-ből szeretnénk adatot feltölteni egy szerverre

1. Az Activity becsomagolja az adatot egy Intent-be, beállítja az általa ismert (ugyanazon alkalmazásban lévő) Service osztálynevét
2. Meghívja a `startService()` metódust aminek átadja az Intentet

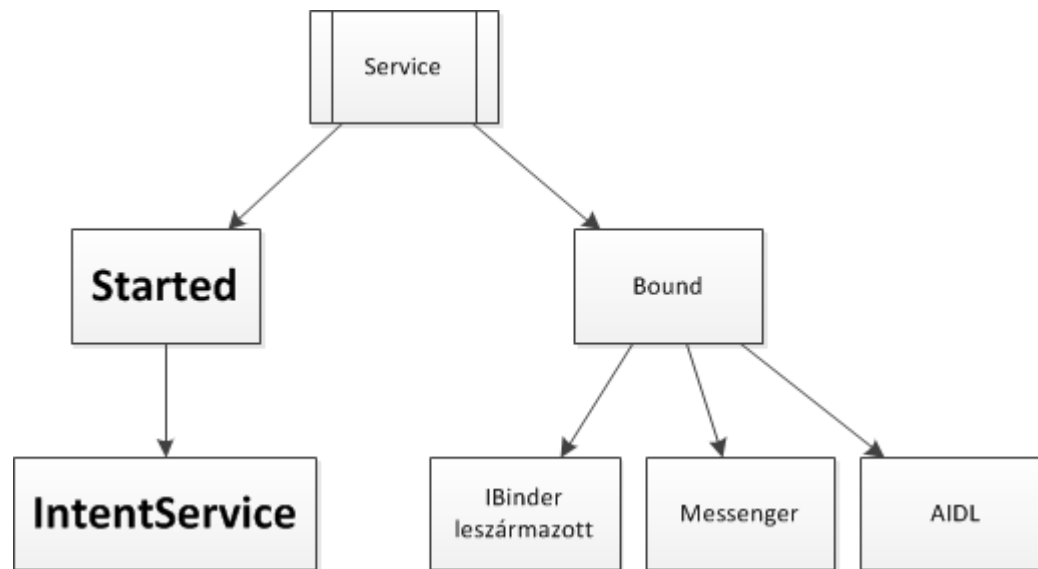
Started Service létrehozás

3. A Service-ben lefut az *onStartCommand()* eseménykezelő, ami megkapja az Intentet
4. Az Intent-ből kibányássza hogy mit kell feltölteni (esetleg hogy hova)
5. A Service létrehoz egy Thread-et vagy AsyncTask-ot, amiben elvégzi a feltöltést
6. Ha végezett, *stopSelf()*-el leállítja magát

Gyakoroljunk!

- Készítsünk egy Background service-t, mely megjeleníti az aktuális időt 5 másodpercenként
- Fejlesszük tovább a megoldást Foreground service-é kötelező notification-al




INTENTSERVICE



IntentService

- A Service leszármazottja
- Problémák a sima Service osztállyal:
 - Nem kap új szálát
 - Kézzel kell leállítani
 - Ha egyszerre többen használják (minden eseménykezelőben vizsgálni kell, hogy épp melyik Intent kiszolgálása folyik)

IntentService

- Megoldás: **IntentService** használata a Service helyett
 - > Külön szálát indít, a fejlesztőnek már nem kell 
 - > Ha elfogytak a kérések, **megállítja magát** 
 - > Sorosítja a bejövő Intenteket 

IntentService

- Használata:
 - > IntentService-ből származtatunk, konstruktorból `super()`-t hívunk
 - > Az aktuális Intentet feldolgozzuk az `onHandleIntent()` megvalósításával
 - > A többi eseménykezelő implementálása nem kötelező, ha mégis megvalósítjuk, akkor az IntentService ősszotály megfelelő metódusával térjünk vissza!

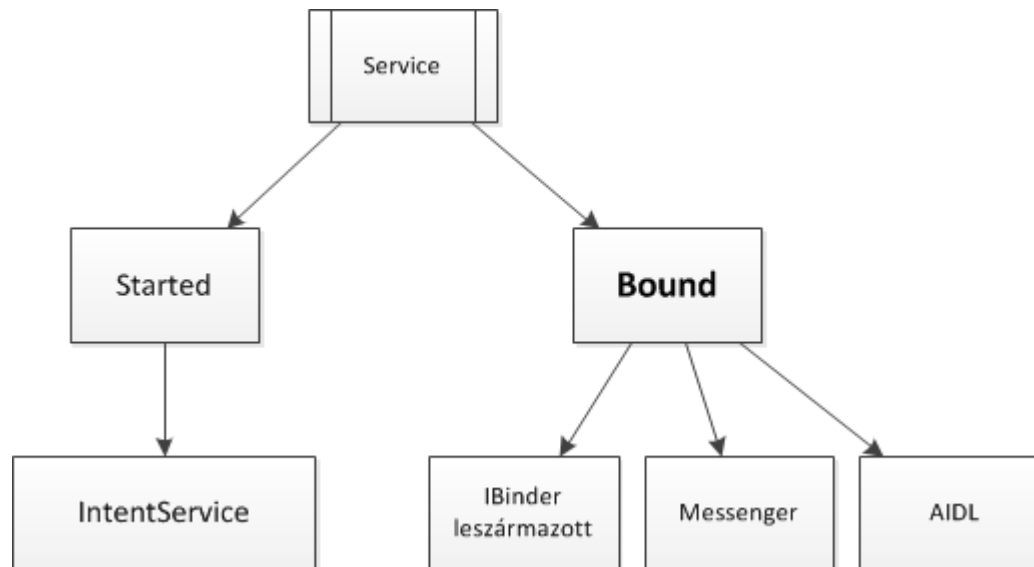
IntentService skeleton

```
class MyIntentService(var serviceName: String) : IntentService(serviceName) {  
  
    override fun onHandleIntent(p0: Intent?) {  
        // Service feladatának ellátása  
        // 1. Egyszerre egy Intent-et dolgoz fel - sorosítás.  
        // 2. Ha végez az összes kéréssel, az IntentService leállítja magát,  
        //    - nincs szükség külön stopSelf hívásra.  
        // 3. Ez a kódrész már külön szálban fut  
  
    }  
  
}
```

Gyakoroljunk

- Készítsünk egy *IntentService*-t, mely véletlen fényképeket tölt le hálózatról
- Vizsgáljuk meg a sorosított végrehajtást
- Vizsgáljuk meg a szálkezelést

BOUND SERVICE



Bound service

- Akkor használjuk, ha nem egy taszk egyszeri elvégzése a feladat...
- ...hanem **folyamatos szolgáltatást** akarunk nyújtani és kiajánlani...
- ...amihez nem kell UI
- Kliens-szerver architektúrát követ
- Ahol a szerver a Service, a kliensek kapcsolódnak hozzá (ettől *Bound*)

Bound service

- Bármilyen komponens csatlakozhat
- Akár más alkalmazásból
- Üzenet alapú kommunikáció vagy közvetlen metódushívás
- Akkor él, ha valaki használja
- Az Android megállítja, ha már nem csatlakozik hozzá senki

BOUND SERVICE MEGVALÓSÍTÁS

Bound service megvalósítása

- Service-ből származtatunk
- Megvalósítjuk az *onBind()* metódust (ez kötelező Started esetben is)
- IBinder interfész implementációjával térünk vissza belőle
- Kliensek *bindService()*-el kapcsolódnak
- Ennek hatására hívódik a Service *onBind()* metódusa

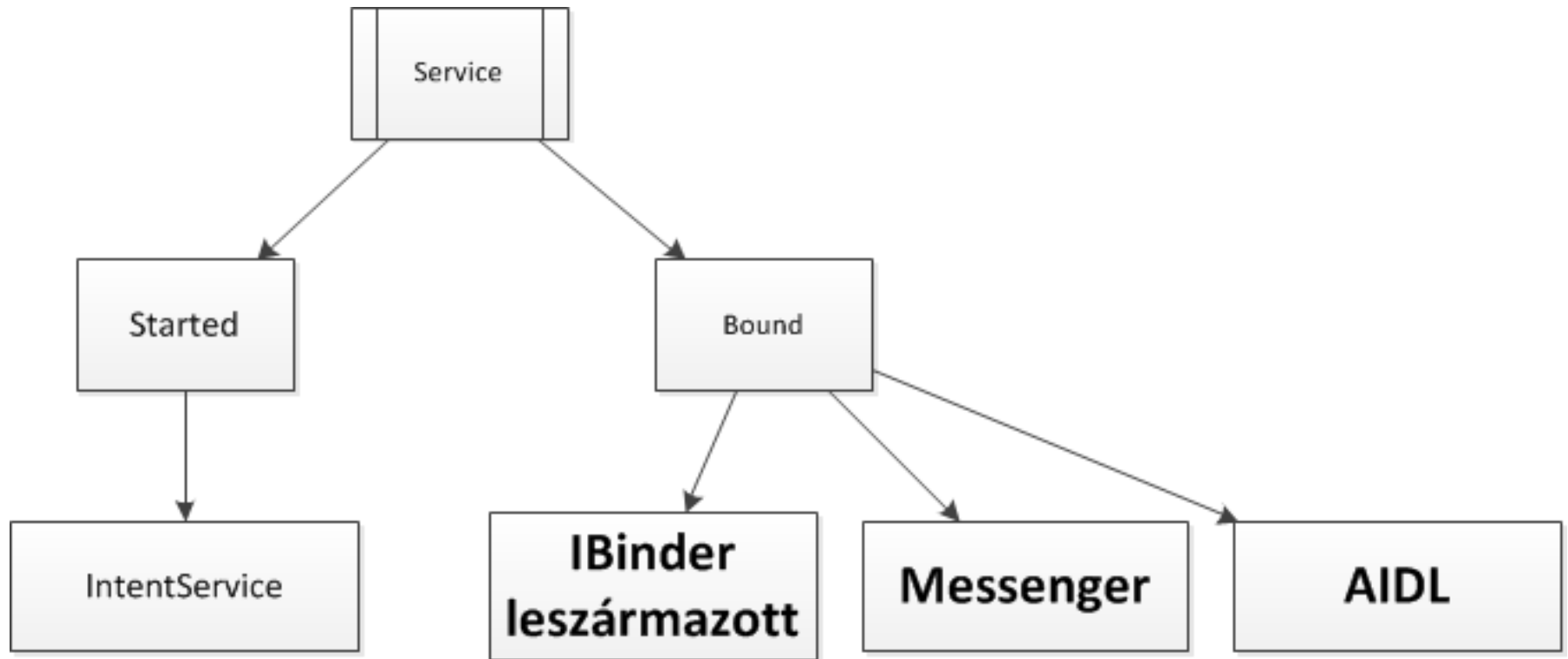
Bound service megvalósítása

- Többen is kapcsolódhatnak egyszerre
- Ekkor ugyanazt az `IBinder` objektumot kapják (Singleton)
 - > Csak az első bind-nál fut le a létrehozása
- Ha már senki sem csatlakozik hozzá, akkor megáll, ilyenkor ismét el kell indítani bejövő kérés esetén (Android végzi)
- Legfontosabb feladat az **`IBinder`** interfész megvalósítása, amint az *onBind* visszaad

IBinder implementáció

- Három lehetőség az interfész megadására
 1. Binder osztály leszármazott
 2. Messenger használata
 3. AIDL (Android Interface Description Language) használata

Szolgátatás interfész megadása



IBinder implementáció

> IBinder-ből származtatás

- Ha nem ajánljuk ki a Service-t más alkalmazások számára, akkor ezt használjuk
- Ezen keresztül elérhető lesz minden **public** metódus a saját Binder osztályunkban, és a Service-ben egyaránt
- Közvetlen metódushívás

IBinder leszármozott

```
private val timeServiceBinder = TimeServiceBinder()

override fun onBind(intent: Intent): IBinder {
    return timeServiceBinder
}

fun changeInterval(newInterval: Long) {
    interval = newInterval
}

inner class TimeServiceBinder: Binder() {
    internal val service: TimeService
        get() = this@TimeService
}
```

IBinder implementáció

> **Messenger** osztály használata

- Különböző processzek közti kommunikációra (IPC)
- **Nincsenek közvetlenül meghívható metódusok, csak üzenetek!**
- A Messenger sorosítja a bejövő üzeneteket
- Írnunk kell egy saját **Handler** leszármazottat, ami az üzeneteket feldolgozza
- Ezzel példányosítjuk a Messenger osztályt, ami kezeli az üzenetváltást
- Hívóból (pl. Activity-ből) **ServiceConnection** segítségével csatlakozunk

IBinder implementáció

- Messenger osztály használata a **hívó oldalon**
 1. ServiceConnection osztályból leszarmaztatunk
 - **onServiceConnected()** megvalósítása: itt kell lekérnünk az IBinder objektumot, az Android hívja
 - **onServiceDisconnected()**: Az op.rendszer hívja ha megszakadt a kapcsolat a Service-el (nem unBind-kor!)
 2. *bindService()* hívása a *ServiceConnection* leszarmazott példánnyal *onStart()*-ban

IBinder implementáció

3. A rendszer hívja az *onServiceConnected()* metódust, ha lefutott akkor kezdhetjük használni a szolgáltatást (érdemes egy bool-t fenntartani ennek jelzésére)
4. Használjuk a Service-t (üzeneteket küldünk neki)
5. *unbindService()* ha végeztünk, vagy az Activity *onStop()* hívódik

IBinder implementáció

- > AIDL (*Android Interface Description Language*) interfészleíró használata
 - C jellegű (IDL) leíró
 - Akkor használjuk, ha ki akarunk szolgálni konkurrens kéréseket sorosítás nélkül (miért jó?)
 - Szálbiztos implementáció kritikus!!
 - Nagyon ritkán kell alkalmazni, nem tárgyaljuk részletesen
 - Minden rendszerszolgáltatás ilyen

AIDL példa

```
// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things with it. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,
        double aDouble, String aString);
}
```

Forrás: <https://developer.android.com/guide/components/aidl>

Felhasználó értesítése Service-ből

A Service-nek nincs felhasználói felülete

Mégis szükség lehet arra, hogy értesítse a usert

- Pl. sikerült letölteni a fájlt
- Milyen lehetőségeink vannak?
 - **Toast** – csak szöveg, nem perzisztens, de néha elég ez is
 - **Notification** – gazdagabb felület, ott marad amíg a felhasználó el nem tünteti, PendingIntent-el interaktívvá tehető
 - **Broadcast** a Service-ből, amit a hívó Activity feldolgoz és megjelenít valamit

Gyakoroljunk!

- Fejlesszük tovább az előző Foreground service-t úgy, hogy lehessen hozzá csatlakozni és módosítani az idő frissítés gyakoriságát

Service vs. Szál

- A Service olyan komponens ami akkor is futhat, ha a felhasználó épp nincs interakcióban az alkalmazással
 - > Zene lejátszás
 - > Torrent
 - > Fájl le-feltöltés
 - > Stb...
- Nincs felhasználói felülete
- Szolgáltatást nyújt más komponenseknek
- Akár más alkalmazásoknak
 - > Lásd rendszer szolgáltatások, pl. TelephonyService
- **Ha erre van szükségünk, használjunk Service-t**

Service vs. Szál

- Amennyiben hosszan tartó feladatot akarunk végezni...
- ...de **csak akkor, amikor a felhasználó interakcióban van az alkalmazással**
 - > = látszik az alkalmazásunk
 - > Például játék alatti zene lejátszás
- Akkor **NEM a Service a jó megoldás!**
- Indítsunk egy új **szálat**, használjuk az **AsyncTask**-ot vagy **HandlerThread**-et
- Figyelem! A Service is alapból a fő (UI) számban fut. Ha a szolgáltatásunk erőforrás igényes, blokkoló műveletet végez, annak is új szál kell indítani!

Gyakorló feladat!

- Készítsünk egy Service-t (például ami időt számol, vagy pozíciót tölt fel Firebase-be) és indítsuk el egyből Boot után
 - > BroadcastReceiver-el BOOT_COMPLETE-elkapás

Köszönöm a figyelmet!



peter.ekler@aut.bme.hu



Department of
Automation and
Applied Informatics