

# Android 9 újdonságok és további helyfüggő szolgáltatások, Coroutinok, Kotlin érdekességek

Ekler Péter

BME VIK AUT, AutSoft

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)

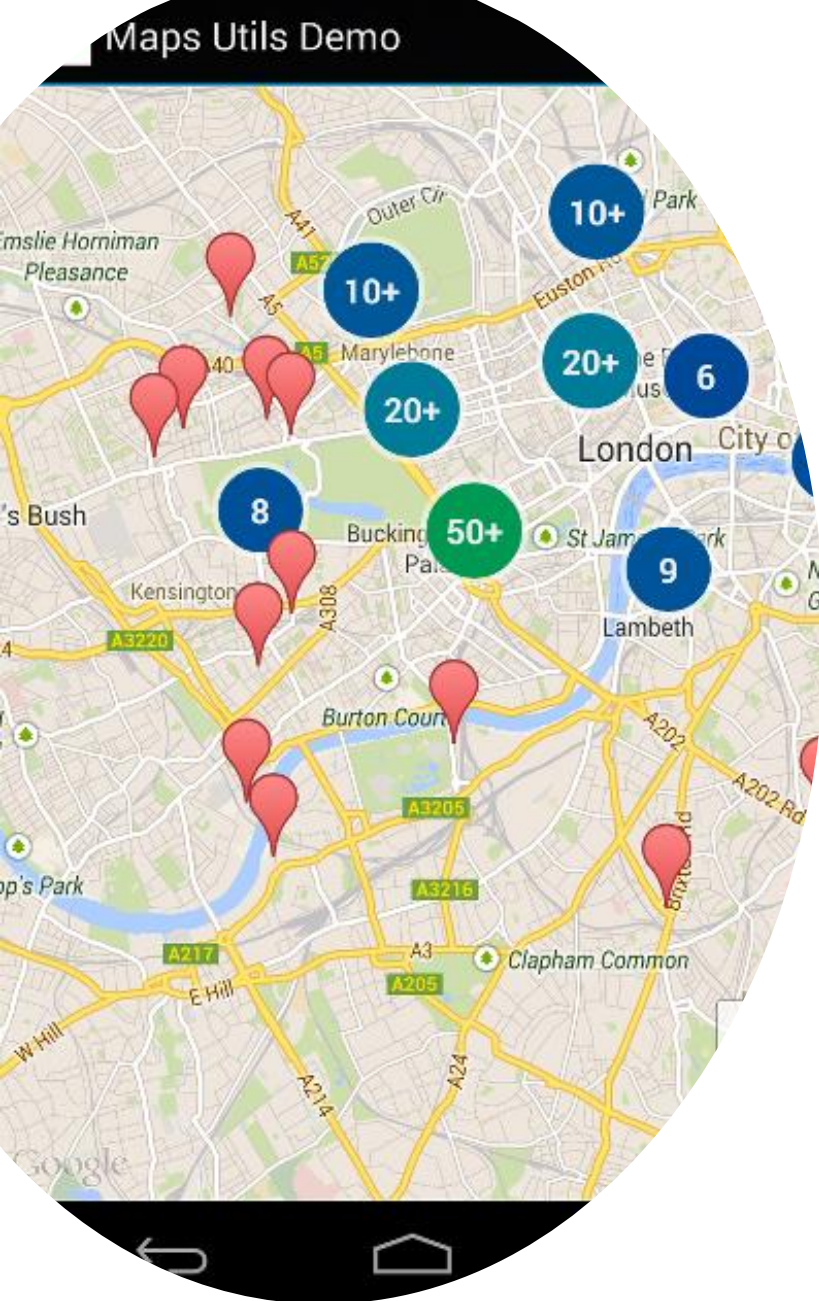


# Tematika

1. Service komponens
2. ContentProvider, Komplex felhasználói felületek
3. Játékfejlesztés
4. Grafikonok, Szenzorok, Külső osztálykönyvtárak, Multimédia alapok
5. Multimédia, további kommunikációs megoldások
6. Biztonságos alkalmazások
7. Android TV és Wear OS
8. Android 9 újdonságok és további helyfüggő szolgáltatások
9. Tesztelési lehetőségek
10. Alkalmazás publikálás, karbantartás (CI/CD)

# Tartalom

- Haladó térkép kezelés
- Activity recognition
- Geofence-ek kezelése
- Places API
- Android 9 API-k bemutatása
- Kotlin Coroutine-ok

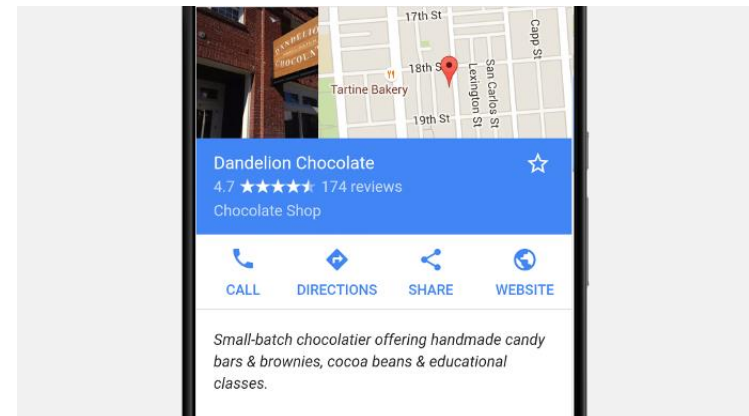
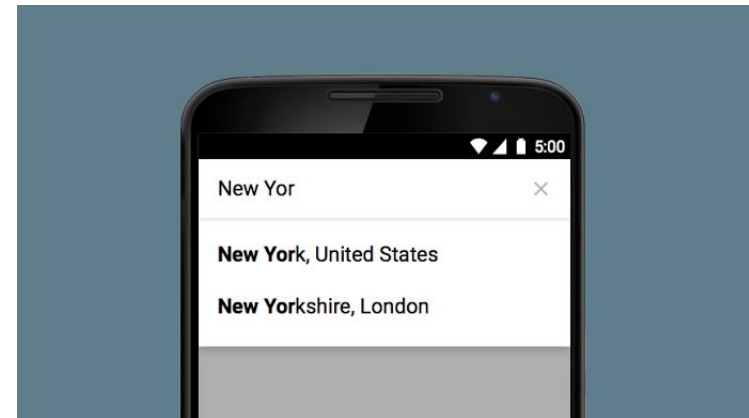
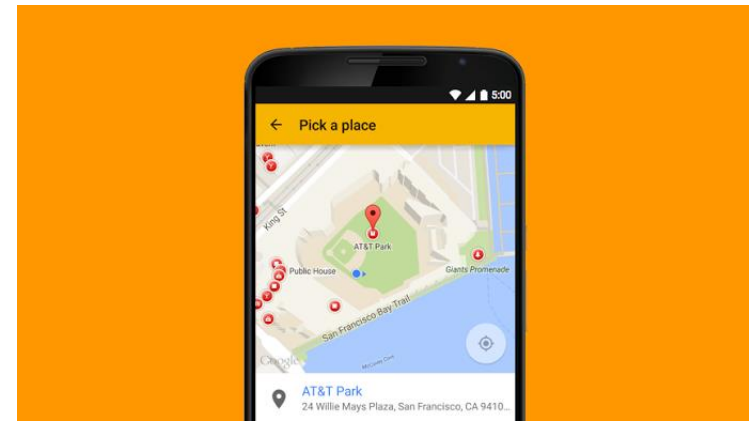


<https://developers.google.com/maps/documentation/android-api/utility/?hl=en>

# Map Utility Library

# Places API

- PlacePicker
  - > Hely kiválasztó dialógus
- GeoDataApi
  - > Google hely adatbázisához hozzáférés
- PlaceDetectionApi
  - > Hozzáférés az az aktuális helyhez és jelentések készítése
- AutoComplete
  - > Hely kiegészítő – beviteli mező
- További részletek:
  - > <https://developers.google.com/places/android-api/>



# PlacePicker Dialog demo

- build.gradle

- > implementation 'com.google.android.gms:play-services-location:16.0.0'
- > Implementation ,com.google.android.gms:play-services-places:16.0.0'
  - <https://developers.google.com/android/guides/setup>

- Manifest:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY" />
<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

- API kulcs:

- > <https://developers.google.com/places/android-sdk/signup>

# PlacePicker Dialog demo

- Google API client csatlakozás

```
googleApiClient = GoogleApiClient.Builder(this)
    .addApi(Places.GEO_DATA_API)
    .addApi(Places.PLACE_DETECTION_API)
    .enableAutoManage(this, this)
    .build()
```

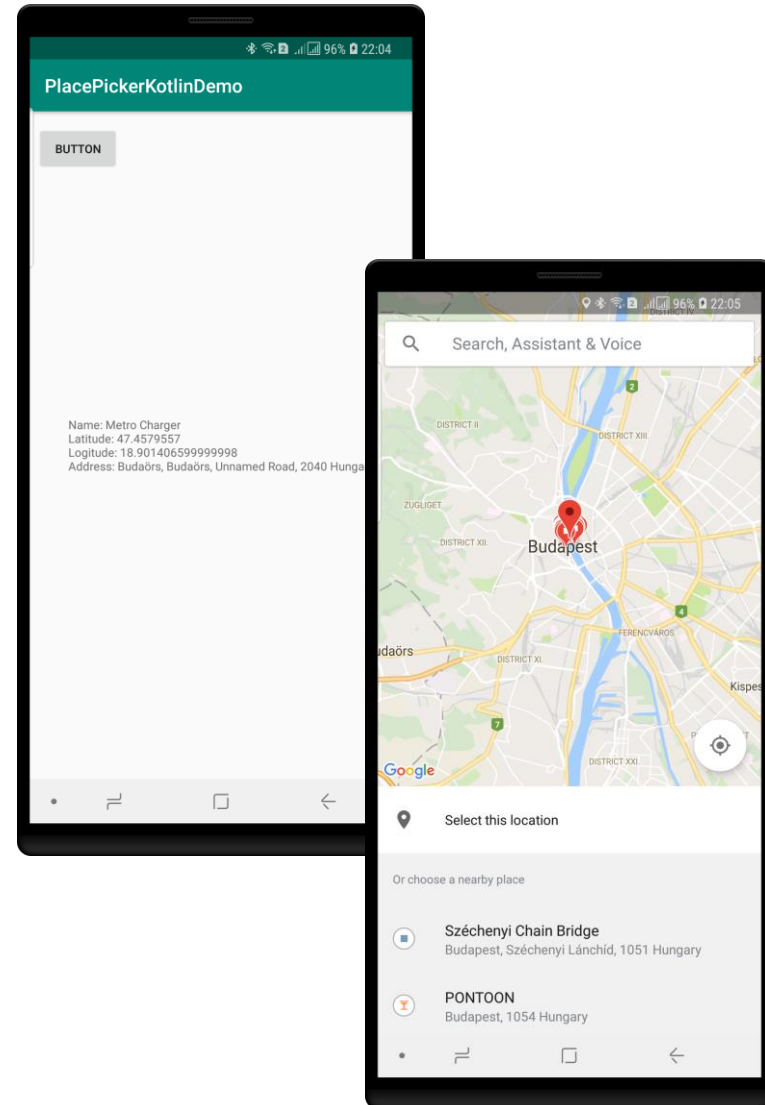
- PlacePicker indítás

```
val builder = PlacePicker.IntentBuilder()
startActivityForResult(builder.build(this@MainActivity), PLACE_PICKER_REQUEST)
```

- Válasz kezelése onActivityResult(...) -ban

# PlacePicker Dialog demo

- Készítsünk egy PlacePicker demot
- A hely kiválasztása után jelenítsük meg a fő adatokat





# ProximityAlert

- Értesítések egy adott környék megközelítésekor
  - > Koordináta és sugár

```
val intent = Intent(ACTION_PROXIMITY_ALERT)

val pendInt: PendingIntent = PendingIntent.getBroadcast(this, requestcode,
intent, flags)

locationManager.addProximityAlert(lat, long, radius, timeout, pendInt)

...

class ProximityIntentReceiver: BroadcastReceiver() {
    @Override
    override fun onReceive(context: Context, intent: Intent) {
        val key = LocationManager.KEY_PROXIMITY_ENTERING
        val entering = intent.getBooleanExtra(key, false)
        ...
    }
}
```

# További GEO API-k

- Google Play Services része
- Új API-k:
  - > Fused location provider (erről már volt szó):
    - <https://developer.android.com/training/location/retrieve-current.html>
  - > Geofencing API
  - > Activity Recognition

# GEOFENCING API

Mostoha Roland (mostoha.roland@autsoft.hu)

# Geofencing API

- Földrajzi határok beállítása és feliratkozás belépés vagy elhagyás események bekövetkezésekor



# Geofencing API

- Google Play Services része
- Egyszerű, de gazdag API
  - > *Geofence* lista gyors és kötegelt megadása vagy eltávolítása
  - > Több *Geofence* egyidejű kezelése
  - > Riasztások szűrése
  - > Hatékony helymeghatározás *Fused Location Provider* használatával
  - > Alacsony energifogyasztás: a helymeghatározási technológiák dinamikus használata a *Geofence* határától függően

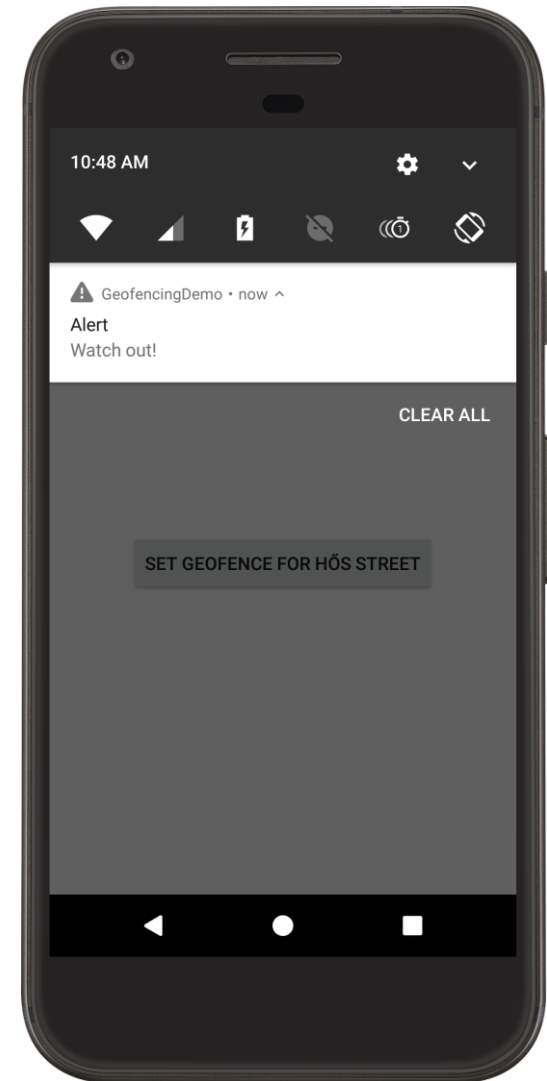
# Geofencing API

- Szükséges engedély (dangerous, el kell kérni):  
`android.permission.ACCESS_FINE_LOCATION`
- Szükséges függőség:  
`com.google.android.gms:play-services-location:X`
- API használata
  - > Kliens inicializálása: *LocationServices.getGeofencingClient(this)*
  - > *IntentService* létrehozása, amely feldolgozza és reagál az *Geofence API* eseményeire
  - > *IntentService* és a számunkra érdekes események regisztrációja a kliensnél
  - > *GeofenceTransition* alapján lekezelni az eseményeket
  - > Szükség esetén kliens lecsatolása és *Geofence* lista törlése

# Geofence API demo alkalmazás

- A hős utcát megközelítve dobjunk egy figyelmeztető értesítést, elhagyva pedig egy megnyugtatót

47.49671, 19.11177



# Tippek a teszteléshez

- Google Play Services legújabb verziója szükséges a teszteléshez
- Emulátoron teszteljünk, használjuk a Location tool-t a pozíciók beküldéséhez

The screenshot shows the 'Location' tool interface in Android Studio. On the left is a sidebar with icons for Location, Cellular, Battery, Phone, Directional pad, and Microphone. The main panel is titled 'GPS data point' and contains the following elements:

- Coordinate system:** A dropdown menu set to 'Decimal'.
- Currently reported location:** A text box displaying:  
Longitude: 19.1118  
Latitude: 47.4967  
Altitude: 0.0
- Longitude:** A text field with the value '19.11177646357646'.
- Latitude:** A text field with the value '47.49671108006458'.
- Altitude (meters):** A text field with the value '0.0'.
- SEND:** A button to submit the data.
- GPS data playback:** A section at the bottom for playing back recorded location data.



# Tippek a teszteléshez

- Ne használjunk túl pontos pozíciókat vagy túl kis távolságot
- *Settings- Location mode – High accuracy* módot használjunk, különben az API 1000-es error code-al tér vissza
- Wi-Fi legyen bekapcsolva
- Az esemény kiváltáshoz akár 2-3 perc szükséges lehet

# Kotlin kiegészítések

Pair, Triple beépített osztályok Kotlinban:

```
// Hős street (latitude, longitude, circular distance in meters)  
private val region = Triple(47.49671, 19.11177, 5000f)
```

by lazy delegate. Első híváskor inicializálódik, utána ugyanazt a példányt használja:

```
private val geofencePendingIntent: PendingIntent by lazy {  
    val intent = Intent(this, GeofenceIntentService::class.java)  
    PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT)  
}
```

Számok tagolása kódban. Növeli az olvashatóságot:

```
.setExpirationDuration(120_000L)
```

# Kotlin kiegészítések

Java `||` helyett `or`, javítja az olvashatóságot

(Geofence.*GEOFENCE\_TRANSITION\_ENTER* or Geofence.*GEOFENCE\_TRANSITION\_EXIT*)

Függvényhívások ugyanazon az objektumon: `apply`

```
return GeofencingRequest.Builder().apply {  
    setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)  
    addGeofences(geofenceList)  
}.build()
```

Mostoha Roland (mostoha.roland@autsoft.hu)

# ACTIVITY RECOGNITION

# Activity Recognition API

- Google Play Services része
- A felhasználók cselekvéseit detektálhatjuk és reagálhatunk rá
  - > IN\_VEHICLE
  - > ON\_BICYCLE
  - > ON\_FOOT
  - > STILL
  - > WALKING
  - > RUNNING

# Activity Recognition API

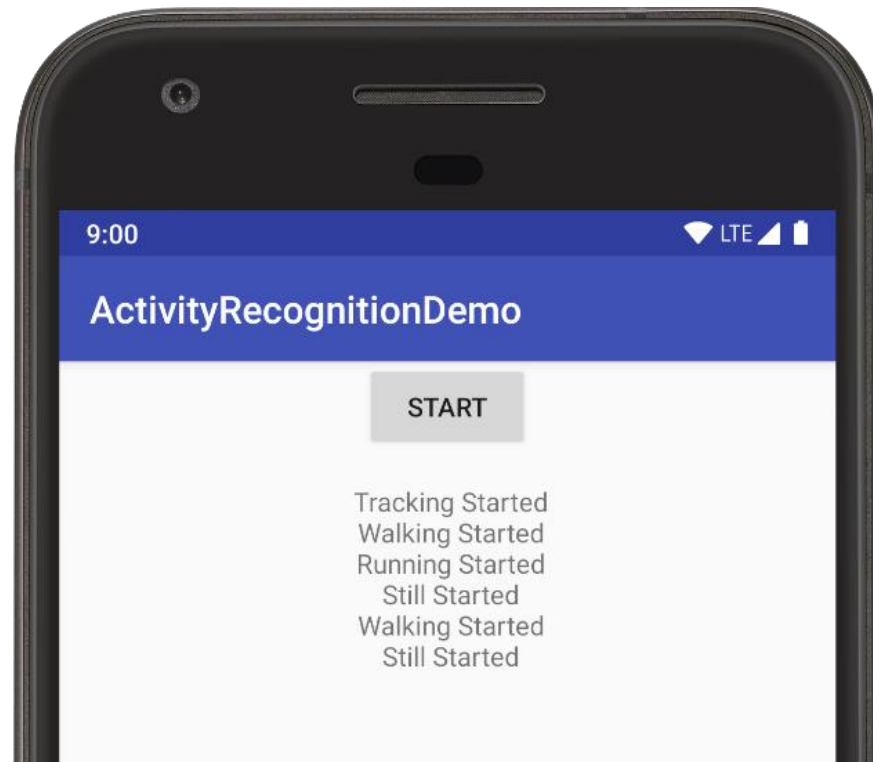
- Feliratkozhatunk a cselekvés megváltozását jelző ENTER és EXIT eseményekre
- Pl. Kalória számláló alkalmazás, amely automatikusan változtatja az elégetett kalória mennyiséget aszerint hogy gyaloglunk, futunk vagy állunk
- Pl. Egy chat alkalmazás, amely vezetés közben letiltja az értesítéseket

# Activity Recognition API

- Szükséges engedély:  
`com.google.android.gms.permission.ACTIVITY_RECOGNITION`
- Szükséges függőség:  
`com.google.android.gms:play-services-location:X`
- API használata
  - > Kliens inicializálása: *ActivityRecognition.getClient(this@MyActivity)*
  - > *IntentService* létrehozása, amely feldolgozza és reagál az *ActivityRecognition* eseményeire
  - > *IntentService* és a számunkra érdekes események regisztrációja a kliensnél
  - > *ActivityType* és *TransitionType* alapján lekezelni az eseményeket
  - > Kliens lecsatolása

# Activity Recognition demo alkalmazás

- Activity Log gyűjtése az események alapján





# Tippek a teszteléshez

- Valós eszközön teszteljük (az emulátor szenzorai csak ritkán triggerelik a szolgáltatást)
- Tiltsuk le az elforgatást
- Az újabb verziókban már automatikusan történik a mintavételezés, nem tudjuk befolyásolni a gyakoriságot
- Legyünk türelmesek és kitartóak 😊

# Kotlin kiegészítések

Előtte:

```
LocalBroadcastManager.getInstance(this).registerReceiver(  
    object : BroadcastReceiver() {  
        override fun onReceive(context: Context, intent: Intent) {  
            ...  
        }  
    },  
    statusIntentFilter)
```

---

Utána:

```
LocalBroadcastManager.getInstance(this).registerReceiver(statusIntentFilter) {  
    ...  
}
```

# ANDROID 9 ÚJDONSÁGOK

# Testreszabott rendszer

- Egyszerűbb, testre szabottabb és okosabb rendszer
  - > Gépi tanulási algoritmus

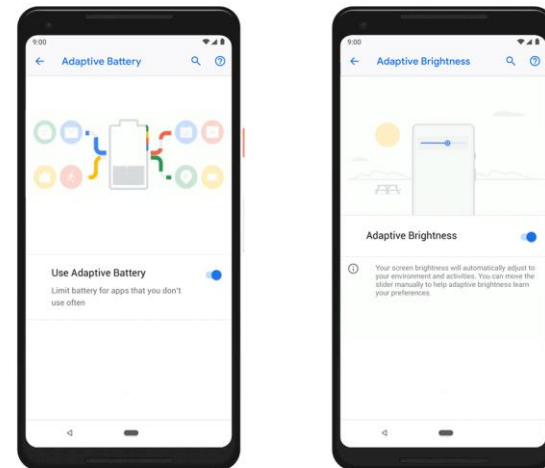


„We’ve built Android 9 to learn from you—and work better *for you*—the more you use it.”

„Android 9 adapts to your life and the ways you like to use your phone.”

# Energiafogyasztás optimalizálása

- Adaptív akkumulátor használat
- Adaptív fényerő állítás
- További optimalizációs megoldások



# Hogy használjuk a mobilunkat?

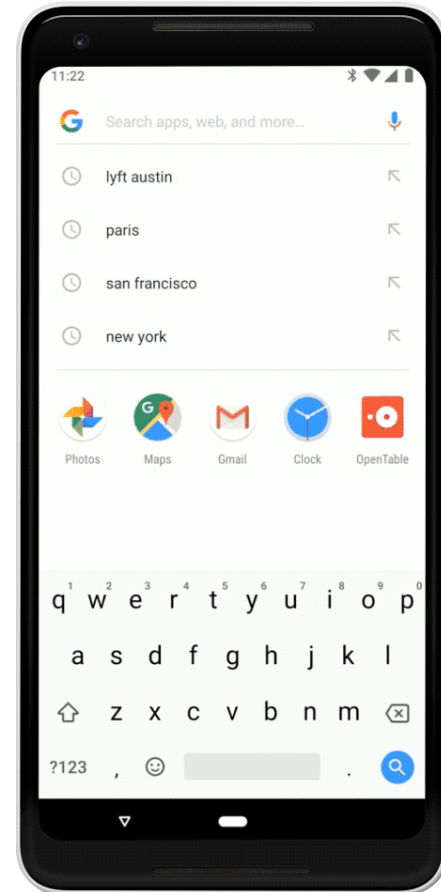


- Média felhasználási idő 69%-a mobilon
- Globális internet használat 80%-a mobilon
- Alkalmazás letöltések száma (2016): 115 milliárd
  - > iOS: 25 milliárd
  - > Android: 90 milliárd
- Keresések számában a mobil a vezető platform
- 88% keresés után elmegy az adott helyi üzletbe 24 órán belül

Forrás: <https://www.biznessapps.com/blog/2018-mobile-marketing-report-stats-need-know/>

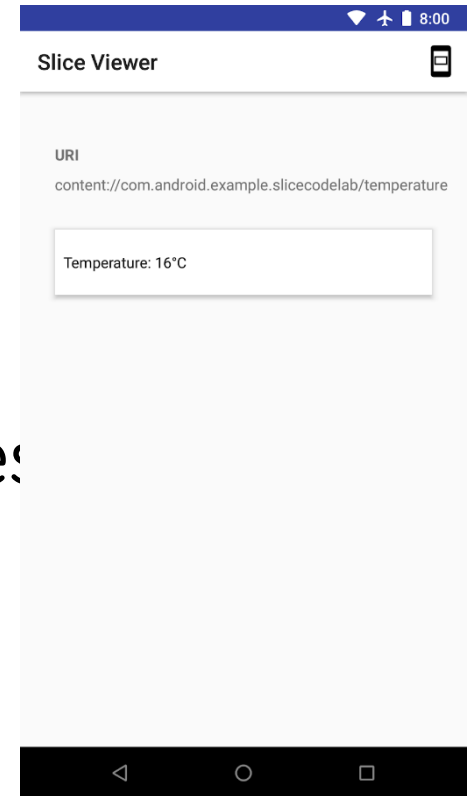
# Gyorsabb ügyintézés - Slices

- Alkalmazás specifikus információk megjelenítése például kereséskor



# Slices példa

- ContentProvider regisztráció
  - > Manifest: `<provider>...</provider>`
- Saját *SliceProvider* osztály
  - > Slice nézet elkészítése
- Interakció BroadcastReceiver-en keresztül
- Tesztelés Slice Viewer-el





# Slices példa

```
class MySliceProvider : SliceProvider() {  
  
    override fun onCreateSliceProvider() = true  
  
    override fun onBindSlice(sliceUri: Uri): Slice? {  
        return when (sliceUri.path) {  
            "/temperature" -> createTemperatureSlice(sliceUri)  
            else -> null  
        }  
    }  
  
    private fun createTemperatureSlice(sliceUri: Uri): Slice? {  
        TODO("implement my Slice")  
    }  
}
```

- Teljes példa: <https://codelabs.developers.google.com/codelabs/android-slices-basic/index.html>

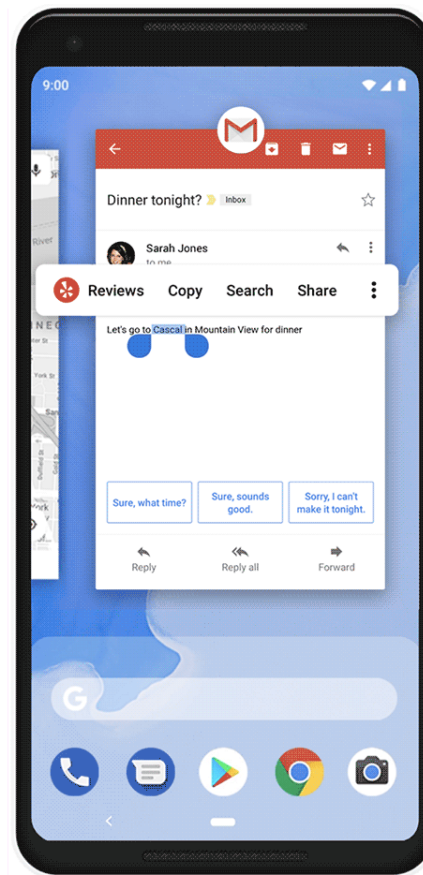
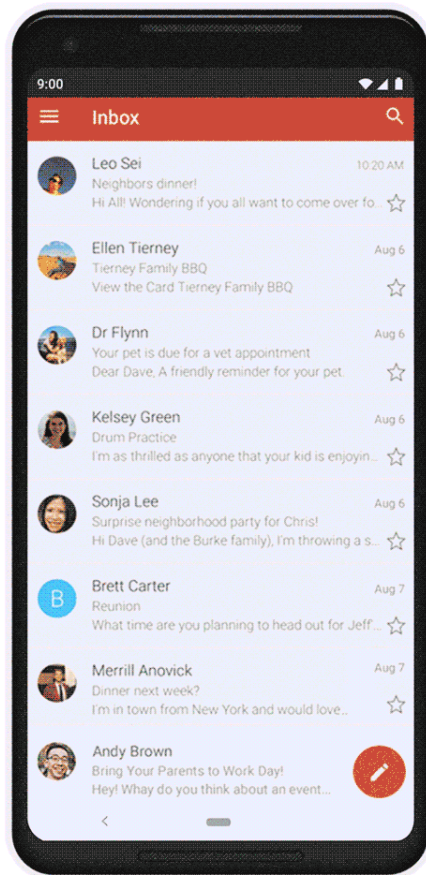
# Gyorsabb ügyintézés – App Actions

- Használati minták alapján kikövetkezteti a következő lépéseket és egyszerűen elérhetővé teszi



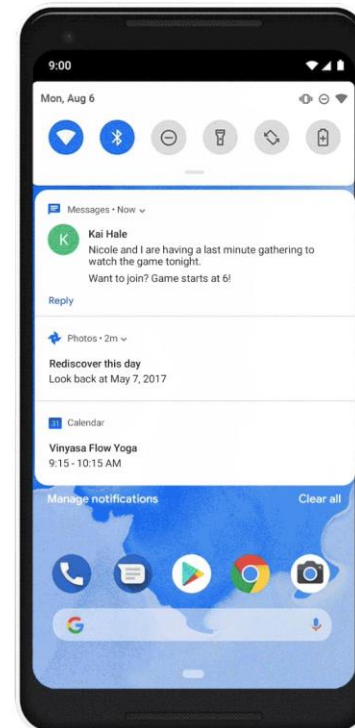
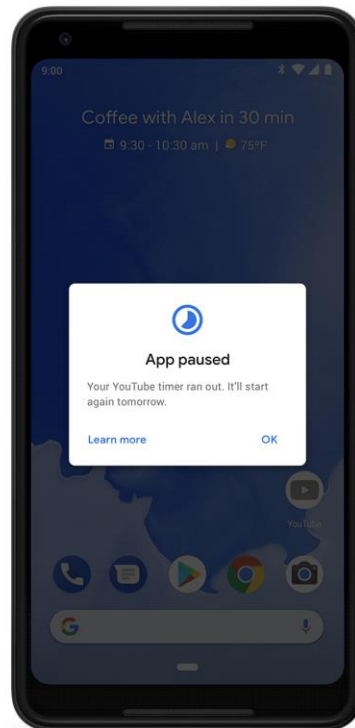
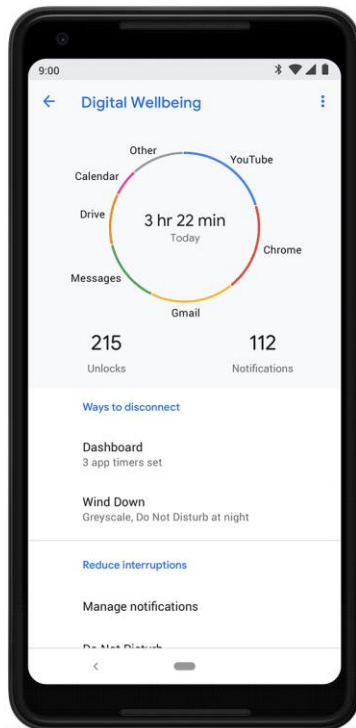
# Egyszerűbb navigáció

- Megújult navigáció alkalmazások között
  - > Home gomb új szerepe
- Könnyebben elérhető beállítások
- Intelligens szöveg kijelölés
  - > és akció felajánlás ...



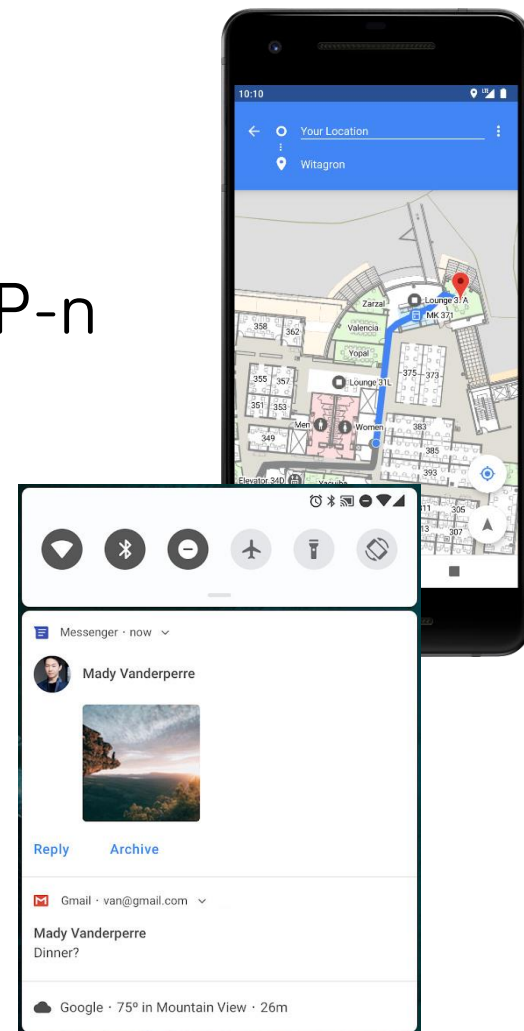
# „Balance your life”

- Dashboard
  - > Alkalmazásokkal töltött idő
- App Timer
  - > Felhasználási idők korlátozása
  - > Ikon kiszürkítése
- Do Not Disturb mód
  - > Minden vizuális jelzés tiltása
- Wind Down mód
  - > Éjszakai mód, Do Not Disturb és szürek árnyalatoss kép bekapcsolása



# API újdonságok

- Beltéri helymeghatározás Wi-Fi RTT
  - > Nem kell csatlakozni az AP-hez
  - > Csak a készüléken érhető el az info, AP-n nem (privacy)
  - > 1-2 méteres pontosság
- Értesítések továbbfejlesztése
  - > Person azonosítás (üzenetek)
  - > Képek támogatása
  - > Smart reply
  - > Csatornák továbbfejlesztése



# API újdonságok

- Több kamera használat
  - Kamerák egyidejű használata
- GIF-ek hatékonyabb kezelése
- JobScheduler hálózati jel alapján
- Neural Networks API 1.1
- Autofill framework
- Security
- On-device system tracing



# Fejlett szöveg kezelés

- Precomputed Text: szöveg renderelés hatékonyság növelése
- Magnifier: alkalmazáson belül használható
- Smart Linkify (TextClassifier osztály a Linkify helyett): gépi tanulás alapú
- Text layout
- Képernyő elforgatás kezelése manuálisan is



Search or type web address



That's all for now

More articles will appear soon. Enjoy your afternoon!

[REFRESH](#)

# API változások a motorháztető alatt

- Melyik API szint közt volt a legnagyobb változtatás?
  - A.25 → 26
  - B.26 → 27
  - C.27 → 28
- [https://developer.android.com/sdk/api\\_diff/28/changes](https://developer.android.com/sdk/api_diff/28/changes)
- [https://developer.android.com/sdk/api\\_diff/28/changes/jdiff\\_statistics](https://developer.android.com/sdk/api_diff/28/changes/jdiff_statistics)

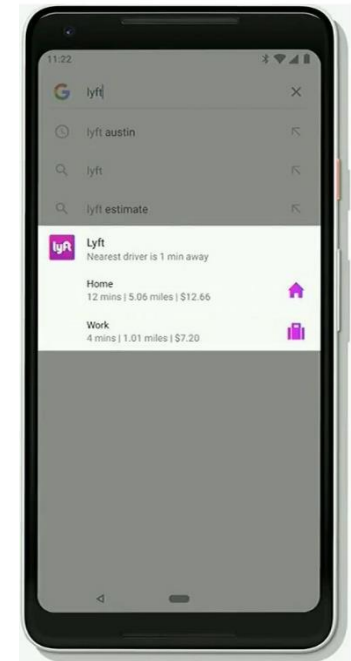
# API 26 → 27

„The overall difference between API Levels 27 and 28 is approximately **6.48%**”

Type	Additions	Changes	Removals	Total
Packages	8	87	7	102
Classes and Interfaces	133	489	9	631
Constructors	14	14	0	28
Methods	483	673	169	1325
Fields	469	272	8	749
<b>Total</b>	<b>1107</b>	<b>1535</b>	<b>193</b>	<b>2835</b>

# Változó az alkalmazás (szolgáltatás) fejlesztés

- UI/UX Design feladatok bővülése
- Alkalmazás viselkedési modellek
  - > Google Play Instant
  - > App Actions
  - > Slices
  - > Google Assistant
- Alkalmazások közti együttműködés
- Reszponzív felületek szerepe, integráció
- Mesterséges intelligencia, ajánlások



# COROUTINE ALAPOK

# Bevezetés

1. Mik azok a Coroutine-ok?
2. Miért érdemes használni őket?
3. Library támogatás
4. “Hello from Coroutine!”
5. Suspending függvények
6. Coroutine építők
7. Coroutine hierarchiák
8. Scope, Dispatcher
9. Androidos példa

# Mi egy coroutine?

- Felfüggeszthető művelet példánya
- Végre tud hajtani megadott kódblokkot és élelciklussal rendelkezik
- Konkurens módon egymással egyidőben képesek futni, várni egymásra, kommunikálni
- Light-weight thread-nek is szokták nevezni: nem egy natív thread, tehát nem von maga után context switch-et a processzoron, így sokkal gyorsabb
- Nincs egy valódi szálhoz (thread) csatolva, library által kezelt thread pool-okban fut
  - Felfüggesztheti a futást egyik szálban és folytathatja egy másikban
  - Nem nagyon van limit, hogy hány coroutine indítható, míg az igazi szálaknál van (a natív szálak száma, illetve a szálak által foglalt memóriára a JVM memória limitje is)



# Előnyök

- Coroutine létrehozása sokkal olcsóbb, mint egy szálé
  - > Kapu nyílik az egyszerű aszinkron programozáshoz
- A kód egyszerű, szekvenciális marad, csak a futtatás történik aszinkron módon
- Lehetőséget ad arra is, hogy egymástól független függvények párhuzamosan fussanak

# Aszinkron programozás coroutine-okkal

- Konkurens programozási stílusok
  - > Callback alapú (JavaScript)
  - > Promise és Future alapú (Java, JavaScript)
  - > Async és Await alapú (C#)
- coroutine-okkal mindegyik stílus megvalósítható

# Nyelvi támogatás

- A Kotlin nyelv a **suspend** függvényekkel támogatja a coroutine feature-t
- A suspending függvényekre építve library-k adhatnak magasabb szintű coroutine api-kat
  - > Pl. JetBrains által fejlesztett **kotlinx.coroutines** lib (további példák ezen a library-n alapulnak)

# kotlinx.coroutines library

- Library támogatás a JetBrains által:  
<https://github.com/Kotlin/kotlinx.coroutines>
- Gradle függőség: `implementation`  
`'org.jetbrains.kotlinx:kotlinx-coroutines-core:0.30.2'`
- Api: `kotlinx.coroutines.experimental` csomagban
  - Kotlin 1.3-ban az `.experimental` package nem kell (Kotlin 1.3 RC - 2018 szept. 20.)

# kotlinx.coroutines library Androidra

- Library támogatás a JetBrains által:  
<https://github.com/Kotlin/kotlinx.coroutines>
- Gradle függőség: `implementation`  
`'org.jetbrains.kotlinx:kotlinx-coroutines-android:0.30.2'`
- Api: `kotlinx.coroutines.experimental` csomagban
  - > Kotlin 1.3-ban az `.experimental` package nem kell (Kotlin 1.3 RC - 2018 szept. 20.)
- Extra funkcionalitások Android-ra
  - > `Dispatchers.Main`-t definiálja, ami az Android fő szálhoz tud indítani coroutine-t
  - > Biztosít a coroutine-okban történt nem elkapott kivételek kiloggolásáról

# “Hello from coroutine!”

```
import kotlinx.coroutines.experimental.GlobalScope
import kotlinx.coroutines.experimental.delay
import kotlinx.coroutines.experimental.launch

fun main(args: Array<String>) {
    GlobalScope.launch {
        delay(1000L)
        println("Hello from coroutine!")
    }
    println("Hello not from coroutine!")
    Thread.sleep(2000L)
}
```

Új coroutine indítása

Nem blokkoló késleltetés, csak felüggeszti (suspend) a coroutine-t

Kimenet:

Hello not from coroutine!  
Hello from coroutine!

Blokkoló késleltetés, hogy bevárjuk a coroutine-t


# Suspending függvények

- Felfüggesztődnek, majd folytatódni tudnak
  - > Nem blokkolják az aktuális szálát
- Végezhetnek hosszan tartó műveleteket
- Szabály: meghívhatóak egy másik suspending függvényből (azaz coroutine-ból)

# Suspending függvények – példa 1/2

```
suspend fun calcOne(): Int {  
    delay(1000L)  
    return 1  
}  
suspend fun calcTwo(): Int {  
    delay(1000L)  
    return 2  
}
```

```
suspend fun runSuspendingFunctions(): Int {  
    val one = calcOne()  
    var two = calcTwo()  
    return one + two  
}  
fun main(args: Array<String>) {  
    val executionTime = measureTimeMillis {  
        runBlocking { println("sum is: ${runSuspendingFunctions()}") }  
    }  
    println("Execution time: $executionTime")  
}
```



Kimenet:

```
sum is: 3  
Execution time: 2127
```

Futtat egy új coroutine-t, híd a blokkoló kód és a coroutine-okat használó kód között



# Suspending függvények – példa 2/2

```
suspend fun calcOne(): Int {  
    delay(1000L)  
    return 1  
}  
suspend fun calcTwo(): Int {  
    delay(1000L)  
    return 2  
}
```

kotlinx.coroutines-beli függvény  
párhuzamosan futó coroutine indítására

```
suspend fun runSuspendingFunctions(): Int {  
    val one: Deferred<Int> = GlobalScope.async { calcOne() }  
    var two: Deferred<Int> = GlobalScope.async { calcTwo() }  
    return one.await() + two.await()  
}  
  
fun main(args: Array<String>) {  
    val executionTime = measureTimeMillis {  
        runBlocking { println("sum is: ${runSuspendingFunctions()}") }  
    }  
    println("Execution time: $executionTime")  
}
```

Kimenet:

```
sum is: 3  
Execution time: 1117
```

# Hogy indítható egy coroutine?

Válasz: coroutine építőkkal

```
fun main(args: Array<String>) {  
    GlobalScope.launch {  
        delay(1000L)  
        println("Hello from coroutine!")  
    }  
    println("Hello not from coroutine!")  
    Thread.sleep(2000L)  
}
```

A launch coroutine builder-ről:

```
/**  
 * Launches new coroutine without blocking current thread and returns a reference  
 * to the coroutine as a [Job].  
 * The coroutine is cancelled when the resulting job is [cancelled] [Job.cancel].  
 * ...  
 */  
public fun CoroutineScope.launch(  
    context: CoroutineContext = EmptyCoroutineContext,  
    start: CoroutineStart = CoroutineStart.DEFAULT,  
    block: suspend CoroutineScope.() -> Unit  
) : Job { . . . }
```

# Gyakori coroutine építők

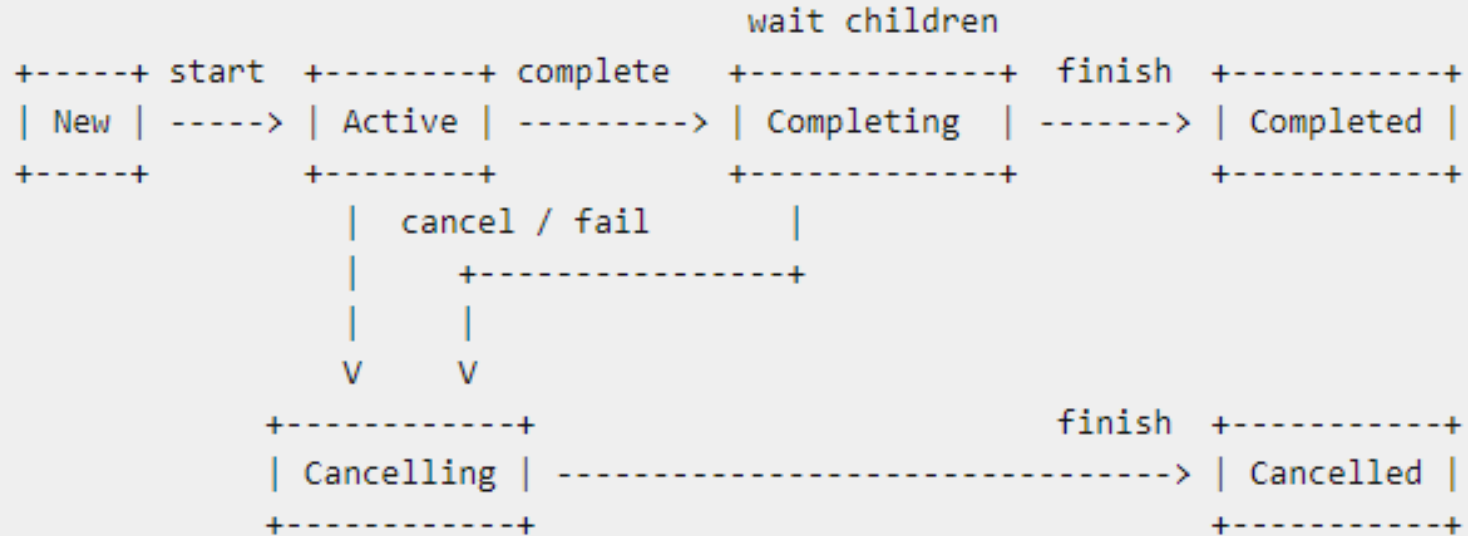
- Launch – indít egy coroutine-t és visszatérít rá egy Job referenciát, a coroutine megállítható `cancel` hívással
- Async – indít egy coroutine-t és visszatérít egy promise-t (Deferred objektumot, ami a Job-ból származik), a coroutine megállítható `cancel` hívással
- `runBlocking`
  - > használatos main függvényekben és tesztekben

# Mi a coroutine kód szintű reprezentációja?

```
/**
 * Launches new coroutine without blocking current thread and returns a
 * reference to the coroutine as a [Job].
 * The coroutine is cancelled when the resulting job is
 * [cancelled] [Job.cancel].
 *
 * ...
 */
public fun CoroutineScope.launch(
    context: CoroutineContext = EmptyCoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    val newContext = newCoroutineContext(context)
    val coroutine = if (start.isLazy)
        LazyStandaloneCoroutine(newContext, block) else
        StandaloneCoroutine(newContext, active = true)
    coroutine.start(start, coroutine, block)
    return coroutine
}
```

A launch építő egy Job referenciát ad vissza

# Job élelciklus



Forrás: <https://kotlin.github.io/...>

# Coroutine hierarchia

- A coroutine-ok szülő-gyerek hierarchiába szervezhetőek
  - > Ha egy coroutineból indítunk egy másikat, akkor az a gyereke lesz
- Szülő leállítása leállítja a gyerekeket is
- Gyerek leállása mással mint `CancellationException`, leállítja a szülőt is

# Példa: Kivétel gyerek coroutine-ban

```
fun main(args: Array<String>) {  
    try {  
        runBlocking { failedChildSuspendingFunctions() }  
    } catch (ex: SomeException) {  
        println("Exception thrown in parent. msg: " + ex.message)  
    }  
}  
  
suspend fun failedChildSuspendingFunctions(): Int {  
    val one = GlobalScope.async<Int> {  
        try {  
            delay(4000L)  
            1  
        } finally {  
            println("First child cancelled...")  
        }  
    }  
    val two = GlobalScope.async<Int> {  
        throw SomeException("Exception from second child")  
    }  
  
    return one.await() + two.await()  
}
```

First child cancelled..  
Exception thrown in parent. msg: Exception  
from second child

# Dispatchers

- Meghatározza, hogy mely szálon fog futni a blokk
- Gyakori dispatcher-ek
  - > **Dispatchers.Default**
    - Minden buildernek ez van beállítva default-ként
    - Közös thread pool-t használ a coroutine blokkjának futtatására
    - CPU intenzív feladatokra használatos
  - > **Dispatchers.IO**
    - Közös on-demand létrehozott thread pool-t használ
    - I/O intenzív műveletekre
  - > **Dispatcher.Main**
    - Android UI szál



# Példa – Androidban – Http GET 1/2

UI thread-en indítjuk a coroutine-t, hogy a UI-t módosítani tudjuk az eredmény alapján

Új coroutine-t indítunk, amin a háttérben, külön szálon fusson a hálózati kérés

```
GlobalScope.launch(context = Dispatchers.Main) {  
    var job = async(context = Dispatchers.Default) {  
        HttpGet(applicationContext).doGet(CURRENCY_URL)  
    }  
    val jsonResponse = job.await()  
    val hufValue = JSONObject(jsonResponse).getJSONObject("rates").getString("HUF")  
    tvResult.text = hufValue  
}
```

A háttérben futtatandó blokk egy http GET kérése lesz

Azonnal kapunk egy **Job** referenciát, mellyel ha szükséges lenne le tudnánk állítani a coroutine-t

Nem blokkoló módon fel van függesztve, amíg a blokk végez

# Példa – Androidban – Http GET 2/2

A `withContext` nem indít új coroutine-t, hanem a meglévőt teszi át másik kontextusba – ezért nem is `Job`-ot ad vissza, hanem a benne lévő blokk eredményét

```
GlobalScope.launch(context = Dispatchers.Main) {  
    var jsonResponse = withContext(context = Dispatchers.Default) {  
        HttpGet(applicationContext).doGet(CURRENCY_URL)  
    }  
    val hufValue =  
        JSONObject(jsonResponse).getJSONObject("rates").getString("HUF")  
    tvResult.text = hufValue  
}
```

Nem blokkoló módon fel van függesztve a fő szál, amíg a blokk le nem fut

# A színfalak mögött

- A coroutine-ok Kotlin implementációja nem JVM vagy OS feature-ön alapul
- A fordító átalakítja a suspending függvényeket és coroutine-okat egy állapot géppé

# Coroutines irodalom

- <https://kotlinlang.org/docs/reference>
- [Concurrent Programming in Kotlin](#)
- [Coroutines in Android](#)

# Köszönöm a figyelmet!



*[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)*



**AutSoft**