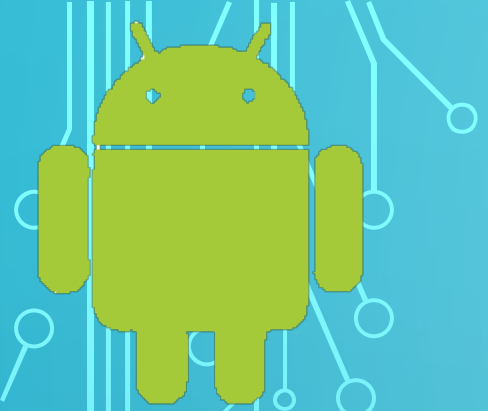


SZOFTVERFEJLESZTÉS ANDROID PLATFORMRA

DR. ISZÁLY GYÖRGY BARNA

ERŐFORRÁSOK

- Az alkalmazások által használt erőforrások jól különválnak az őket kezelő forráskódoktól
- A készülék egyedi tulajdonságaihoz lehet igazítani az erőforrásokat
- Típusuk alapján az erőforrások lehetnek:
  - Képek
  - Hanganyagok
  - Animációk
  - Menük
  - Stílusok
  - Színek
  - Szöveges tartalmak
  - A felhasználói felületet leíró *layout* xml állományok

ERŐFORRÁSOK

- Segítségükkel a kód változtatása nélkül készülhetünk fel az alkalmazásunk különböző környezetben való futtatására
- Valamennyi erőforrás hivatkozhatóvá válik forráskódból az SDK által (Android Asset Packaging Tool - *aapt*) kezelt R.java nevű állományon keresztül
- Az R.java állományban statikus int típusú változókkal leírható valamennyi erőforráselem
- A fájl frissítése az erőforrások mentésekor történik
- A forráskódban az erőforrásokra a következőképen lehet:

```
Button okButton = (Button) findViewById(R.id.okButton);
```

PÉLDA R.JAVA ÁLLOMÁNYRA

```
package com.example.file_teszt;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070005;
        public static final int okButton=0x7f070002;
        public static final int text1=0x7f070000;
    }...
```

ERŐFORRÁSOK

- Az erőforráselemek a projekt gyökérmappájában, a *res* mappában helyezkednek el
- Típusuk alapján külön mappákba elhelyezkedve:
  - *drawable* mappák: a képek, bitmap-ek, ikonok számára
  - *layout*: a felhasználói felületek, activityk leírására
  - *menu*: menük leírására
  - *values*: témák, szöveges erőforrások tárolására
  - *animator*: animációk leírására
  - *color*: színek megadásához

- FONTOS! Az erőforrások neve csak kis betűsek lehetnek, és nem tartalmazhatnak szóközt vagy speciális karaktereket!

ERŐFORRÁSOK

- A mappák elnevezése egy, vagy több *qualifier* (minősítő) segítségével módosítható
- A minősítők használatának szabályai:
  - Kötőjellel választjuk el őket az eredeti névtől, vagy egymástól, ha több is van. Például: *values-hu-v11* jelentése magyar lokalizációjú, API 11-es verziójú eszközök számára használható values mappa.
  - A különböző típusú minősítők csak előre meghatározott, kötött sorrendben használhatók.
    A *values-v11-hu n*em használható!
    https://developer.android.com/guide/topics/resources/providing-resources
  - A minősítővel ellátott mappát ne ágyazzuk az eredetibe be!
    Ne használjuk a values/values-hu mappaszerkezetet!
  - Ugyan nem érzékenyek a kis-nagybetű eltérésekre, azonban a fordító kisbetűs változatra fordítja az elnevezéseket.
  - Minden minősítő fajtából egyszerre csak egy érték használható az erőforrásmappa elnevezésében.
    Ne használjunk pl. *values-en-hu* elnevezést!

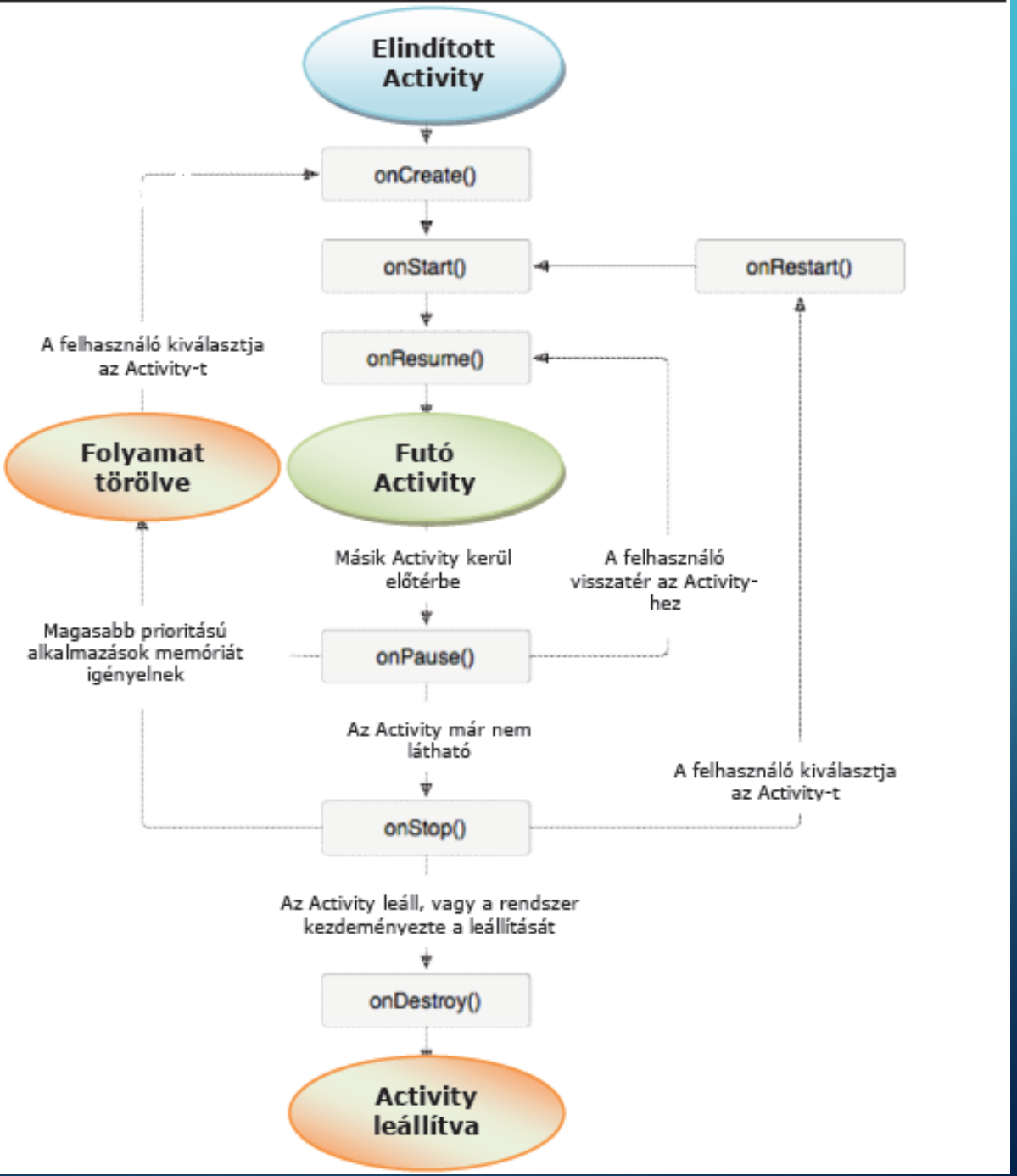
ACTIVITY

- Az alkalmazásnak egy olyan fontos része, amivel a program felhasználója interakcióba léphet
- Minden Activity egy ablakot biztosít a számunkra
- Egy alkalmazás általában több Activity-ből áll
- Van egy kijelölt Activity, amelyik a futtatáskor először jelenik meg
- Egy Activity elindíthat más Activity-ket
- Valahányszor egy Activity elindul, az előző Activity leáll, de a rendszer megőrzi ennek az előző tevékenységnek az állapotát a háttér veremben (*back stack*), ha erről kapcsolókkal másképpen nem rendelkezünk
- Az új Activity indulásakor bekerül a stack tetejére, majd a rendszer elkezdi futtatni.
- A verem LIFO elvű – a *Back (Vissza)* gomb hatására az ezt megelőző tevékenység kerül elő a veremből, és annak futása folytatódik

ACTIVITY

- Az Activity megvalósítása az *Activity* osztály egy alosztályaként deklarálva lehetséges.
- Amikor egy tevékenység egy másik tevékenység indulása miatt leállításra kerül, értesítést kap erről úgynevezett *callback* metódusok segítségével.
- Számos callback metódus létezik:
  - kaphat ilyet az operációs rendszertől létrehozásakor,
  - leállításakor,
  - folytatása,
  - vagy törlése esetén.
- Így az állapotváltozásra egy megfelelő eseménnyel reagálhat a tevékenység
- A tevékenység folytatásakor a félbeszakított tevékenységek folytathatók

AZ A ÉLET


Elindított Activity
onCreate()
onStart()
onRestart()
onResume()
A felhasználó kiválasztja az Activity-t
Folyamat törölve
Futó Activity
Másik Activity kerül előtérbe
A felhasználó visszatér az Activity-hez
Magasabb prioritású alkalmazások memóriát igényelnek
onPause()
Az Activity már nem látható
A felhasználó kiválasztja az Activity-t
onStop()
Az Activity leáll, vagy a rendszer kezdeményezte a leállítását
onDestroy()
Activity leállítva

AZ ACTIVITY ÉLETCIKLUSA

1. **Running**
   - Az Activity az előtérben fut (a stack tetején helyezkedik el), ez az aktív, vagy futó állapot (*running*).
   - Ilyenkor az Activity fókuszban van.
2. **Paused**
   - Ha az Activity elvesztette a fókuszt, de még látható, akár részben is
   - Teljes mértékben élő maradhat, abban az értelemben, hogy az összes állapot- és taginformációját megőrizheti és csatlakoztatva marad az ablakkezelőhöz.
   - Az operációs rendszer extrém alacsony memóriaállapotot érzékel akkor törölheti az Activityt.

AZ ACTIVITY ÉLETCIKLUSA

3. **Stopped**
   - Ha az Activityt egy másik tevékenység teljes egészében letakarja, akkor leállítódik (*stopped*).
   - Minden állapot- és taginformációja megőrződik ugyan, de nem lesz látható a felhasználó számára az ablaka
   - Gyakran törlésre kerülhet az operációs rendszer által
4. **Finished, vagy killed**
   - Szünetelő, vagy leállított állapotú Activity esetén tehát az operációs rendszer bármikor kérheti az Activity befejezését (*finished*), vagy egyszerűen kitörölheti azt (*killed*).
   - Az Activity újbóli megnyitásához ilyenkor újra teljes egészében létre kell hozni azt.

ACTIVITY ÉLETTARTAMA

1. **Teljes élettartam (entire lifetime)**
   - Az Activity azon életciklusa, amely az *onCreate()* metódus első hívása és az *onDestroy()* utolsó hívása között zajlik.
   - Egy Activitynek minden globális beállítást el kell végezni az *onCreate()* híváskor és minden esetleg megmaradt erőforrást el kell engedjen az *onDestroy()* esetében.
2. **Látható élettartam (visible lifetime)**
   - Az *onStart()* és az *onStop()* metódusok meghívása között rész.
   - Az operációs rendszer többször is meghívhatja ezeket a metódusokat az Activity teljes élettartama során, hiszen az Activity láthatósága változhat, ahogy változnak a tevékenységek.
3. **Előtérben töltött élettartam (foreground lifetime)**
   - Az *onResume()* és *onPause()* metódusok hívása között rész.
   - Az Activity nemcsak látható, de előtérben van, birtokolja a fókuszt is.
   - Mivel ilyen váltás gyakran előfordulhat, ezért ajánlott, hogy az ehhez tartozó kód kicsi, gyorsan lefutó legyen, elkerülendő a felhasználói élmény csorbulását.

CALLBACK METÓDUSOK

Metódus	Leírás	Leállítható?	Következő metódus
onCreate()	Az Activity létrehozásakor hívódik. Ekkor kell az összes statikus beállítást elvégezni – nézetek létrehozása, adatok kötése listákhoz, stb. A metódus egy *Bundle* objektumot kap, ami tartalmazza az Activity korábbi állapotát, ha volt ilyen.	Nem	*onStart()*
onRestart()	A leállított Activity újra elindítása előtt közvetlenül hívódik meg. Mindig az *onStart()* metódus követi.	Nem	*onStart()*

Leállítható? – az operációs rendszer törölheti-e bármikor az Activityt kiszolgáló folyamatot az adott *callback* metódus lefutása után egyetlen további sor futtatása nélkül

CALLBACK METÓDUSOK

Metódus	Leírás	Leállítható?	Következő metódus
onStart()	Az Activity felhasználója számára láthatóvá válása előtt kerül meghívásra. Ha az Activity előtérbe kerül, az *onResume()* követi, ha rejtetté válik, akkor az *onStop()* hívódik meg.	Nem	*onResume()*
onResume()	Akkor hívódik meg, amikor egy tevékenység interaktívvá válik a felhasználó számára, azaz az előtérbe kerül. Ekkor az Activity a stack tetején van, készen áll a felhasználói interakcióra. Mindig az *onPause()* metódus követi őt.	Igen	*onPause()*

CALLBACK METÓDUSOK

Metódus	Leírás	Leállítható?	Következő metódus
onPause()	Akkor hívódik, amikor az operációs rendszer egy másik Activityt készül folytatni, vagy újraindítani. Általában a mentetlen adatok rögzítésekor, animációk leállításakor, és más processzor igényes tevékenységekhez köthető. A megvalósításnak nagyon gyorsnak kell lennie, mivel a következő Activity nem folytatódik addig, amíg ez be nem fejeződik. Vagy az *onResume()* metódus követi, ha az *Activity* az előtérbe tér vissza, vagy az *onStop()*, ha láthatatlanná válik a felhasználó számára.	3.0 verzió előtt	*onResume()*, vagy *onStop()*

CALLBACK METÓDUSOK

Metódus	Leírás	Leállítható?	Következő metódus
onStop()	Az Activity nem látható már a felhasználó számára. Ez fakadhat abból, hogy az Activity megszűnt, de abból is, hogy egy másik új, vagy már létező Activity (újra)indítódott és eltakarja ezt a képernyőn. Az *onRestart()* metódus követi, ha a tevékenység a felhasználó számára interaktív állapotba kerül vissza, vagy az *onDestroy()*, ha az Activity végleg meg fog szűnni.	igen	*onRestart()*, vagy *onDestroy()*
onDestroy()	Az utolsó metódus az Activity megszűnése előtt. Bekövetkezhet a *finish()* metódus hívása okán (be akarjuk fejezni az Activityt), vagy azért, mert az operációs rendszer hely megtakarítása végett ideiglenesen megszünteti. Az *isFinishing()* metódussal lehet különbséget tenni a kettő között (*true*, vagy *false*).	Igen	nincs

ACTIVITY ÁLLAPOTÁNAK TÁROLÁSA

- *onSaveInstanceState()* – Amikor a felhasználó elhagyja az Activity-t akkor hívódik meg. Ennek a callback metódusnak a felülírásával lehet további adatokat elmenteni az Activity-nk állapotáról. Egy Bundle osztálybeli elemet ment el.
- Az Activity újra létrehozásakor a Bundle objektum kerül átadásra az onCreate() és az onRestoreInstanceState() metódusoknak.

PÉLDA AZ ACTIVITY ÁLLAPOTÁNAK MENTÉSÉRE

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

PÉLDA AZ ACTIVITY ÁLLAPOTÁNAK VISSZAÁLLÍTÁSÁRA

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
```