Android

Felhasználói felület tervezés és készítés Android platformon

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

BME AUT
Department of Automation and Applied Informatics

Mobil alkalmazásfejlesztői verseny

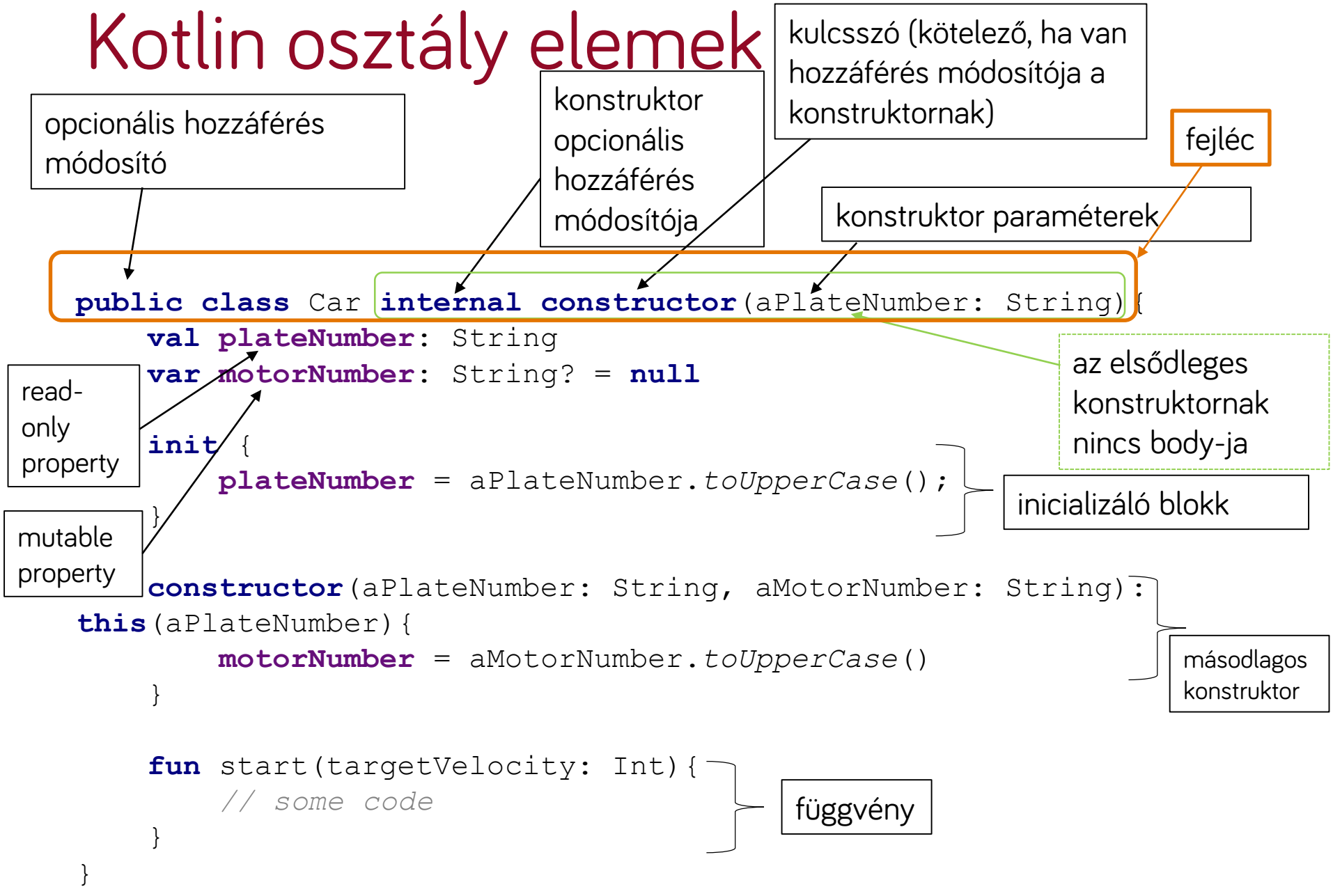
- Ütemezés:
  - Ötletek leadása: Október 15.
  - Short list kihirdetése: Október 22.
  - Prototípus béta leadása: November 25.
  - Prototípus végleges leadása: December 2.
  - Pályaművek elbírálása: December 17.

- Tetszőleges platformon megvalósítható prototípus
- **500.000 Ft összdíj!**
- Regisztráció és információk:
  - https://www.aut.bme.hu/Events/Granit2019

Tartalom

- További Kotlin nyelvi elemek
- Activity komponens
- Különböző felbontású és méretű kijelzők támogatása
- Sűrűség függetlenség
- Felhasználói felület tervezés
- UI alapok
- Egyedi nézetek

Kotlin osztály elemek



Java field vs. Kotlin property

Java

```java
public class Car {
    private String type;

    public String getType() {
        return type;
    }

    public void setType(String type) {
        Log.d("TAG_CAR", "type SET");
        this.type = type;
    }
}
```

Kotlin

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

Data class

```
data class Ship(val name: String, val age: Int)
```

- Automatikusan létrejön:
  - equals()/hashCode()
  - toString(): "Ship(name=Discovery, age=31)";
  - componentN() metódusok
  - copy() metódus

```
val discovery = Ship("Discovey", 31)
val (name, age) = discovery

//val name = discovery.component1()
//val age = discovery.component2()
```

- Követelmények Data osztályokkal szemben:
  - Primary constructor legalább 1 paraméterrel
  - Minden primary constructor paraméter *val* vagy *var*
  - *Data* classes nem lehet *abstract, open, sealed, vagy inner*

Property-k

- **var** <propertyName>[: <PropertyType>] [= <property_initializer>] [<getter>] [<setter>]

```
class Ship(var name: String, var age: Int) {
    var detailedName: String
        get() = "$name $age"
        set(value) {
            name = value
        }
}
```

Delegate

```
interface Pressable {
    fun press()
}

class MyButton(val x: Int) : Pressable {
    override fun press() { Log.d("TAG_MINE","press $x") }
}

class SpecialButton(pressable: Pressable) : Pressable by pressable

fun main(args: Array<String>) {
    val btn = MyButton(10)
    SpecialButton(btn).press() // press 10
}
```

- *by* kulcsszó miatt tovább hív a btn implementációba

Higher order functions

- Olyan metódus, amely metódust kap paraméterül, vagy metódussal tér vissza

```
fun <T> lock(lock: Lock, body: () -> T): T {
    lock.lock()
    try {
        return body()
    }
    finally {
        lock.unlock()
    }
}
```

- Függvény típus: ()->T
- Használat:

```
fun toBeSynchronized() = sharedResource.operation()
val result = lock(lock, ::toBeSynchronized)
```

Extensions

- Új funkció hozzáadása egy osztályhoz anélkül, hogy leszármaztatnánk belőle

```
private fun TicTacToeView.resetGame() {
    TicTacToeModel.resetModel()
    invalidate()
}
```

- *Valójában nem ad hozzá egy új függvényt az osztályhoz, csak lehetővé teszi ezt a függvény meghívását az adott típusú objektumokon*

Néhány érdekesség

- Range-k

```
val x = 4
val y = 3
if (x in 1..y+1) {
    Log.d("TAG_DEMO", "x benne van")
}
```

- Range iteráció

```
for (nr in 1..10 step 2) {
    Log.d("TAG_DEMO", "szam $nr")
}
```

- Lambda műveletek kollekciókon

```
val fruits = listOf("alma", "mango", "mandarin", "narancs")
fruits
    .filter { it.startsWith("m") }
    .sortedBy { it }
    .map { it.toUpperCase() }
    .forEach { Log.d("TAG_DEMO", "$it") }
```

Gyakoroljunk!

- Készítsünk egy Pong alkalmazást!
- Érintett témák
  - Egyedi nézetek
  - Rajzolás
  - Szálkezelés
  - Képek kezelése

Activity bevezetés

- Egy Activity tehát tipikusan egy képernyő, amin a felhasználó valamilyen műveletet végezhet (login, beállítások, térkép nézet, stb.)
- Az Activity leginkább egy ablakként képzelhető el
- Az ablak vagy teljes képernyős, vagy pop-up jelleggel egy másik ablak fölött jelenik meg
- Egy alkalmazás tipikusan több Activity-ből áll, amik lazán csatoltak
- Legtöbb esetben létezik egy „fő” Activity, ahonnét a többi elérhető
- Bármelyik Activity indíthat újabbakat
- Tipikusan a „fő” Activity jelenik meg az alkalmazás indulása után elsőként

A „fő” Activity jelölése

- Manifest állományban jelölnünk kell melyik Activity induljon el elsőként

```
<activity
    android:name=".ExampleActivity"
    android:icon="@drawable/app_icon" >
    <intent-filter>
        <action android:name=" android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- *action* tag: jelzi, hogy ez az alkalmazás fő belépési pontja
- c*ategory* tag: jelzi, hogy az Activity jelenjen meg az indítható programok listájában
- Intentekről (szándék) később…

Activity állapotok

- Egy Activity 3 fő állapotban lehet:
  - **Resumed (running)**: az Activity előtérben van és a focus rá irányul
  - **Paused**: az Activity él, de egy másik Activity előrébb van, de ez még látszik (transparens a felső, vagy pop-up jellege miatt nem fedi el teljesen). A rendszer extrém alacsony memóriaállapot esetén felszabadíthatja.
  - **Stopped**: az Activity még él, de már egy másik Activity van teljesen előtérben és a Stopped állapotban lévőből semmi nem látszik. Alacsony memóriaállapot esetén a rendszer felszabadíthatja.

Activity skeleton 1/2

```
class ExampleActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Most jön létre az Activity
    }

    override fun onStart() {
        super.onStart()
        // Most válik láthatóvá az Activity
    }

    override fun onResume() {
        super.onResume()
        // Láthatóvá vált az Activity
    }
```

Activity skeleton 2/2

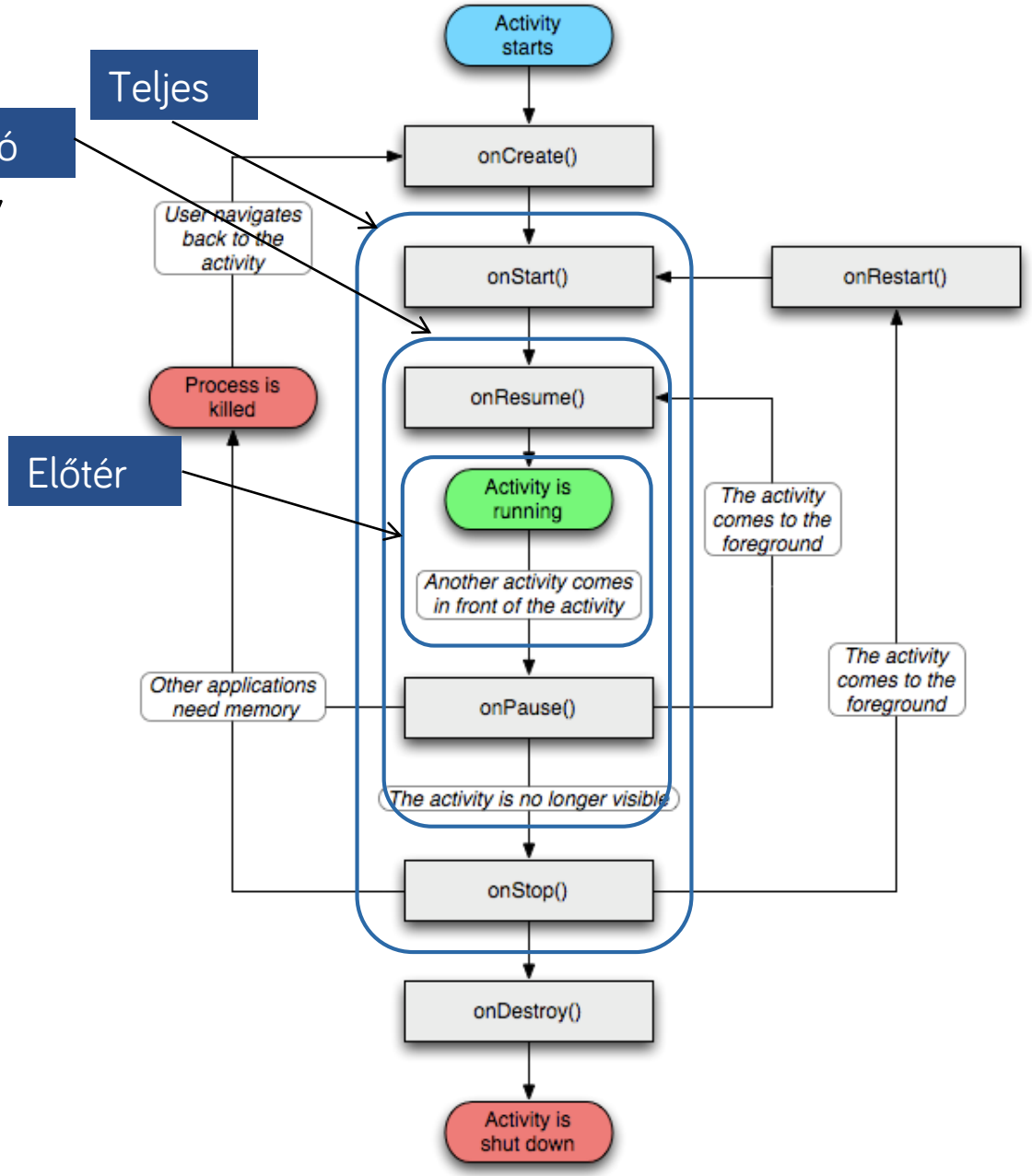
```
    override fun onPause() {
        super.onPause()
        // Másik Activity veszi át a focus-t
        // (ez az Activity most kerül „Paused” állapotba)
    }

    override fun onStop() {
        super.onStop()
        // Az Activity már nem látható
        // (most már „Stopped” állapotban van)
    }

    override fun onDestroy() {
        super.onDestroy()
        // Az Activity meg fog semmisülni
    }

}
```

Activity állapot típusok

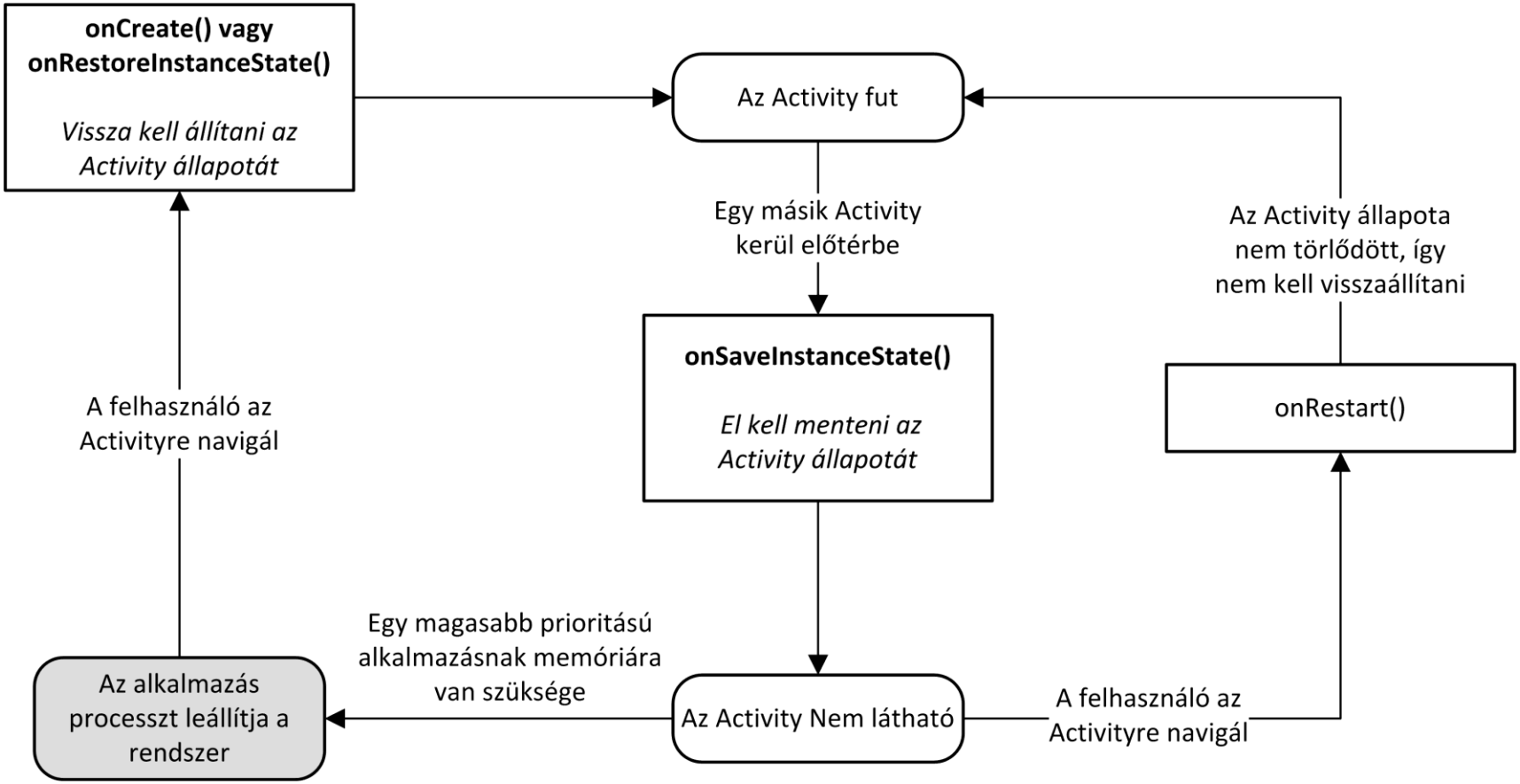
- Teljes (**entire**) életciklus:
  - *onCreate()* és *onDestroy()* közti állapot
- Látható (**visible**) életciklus:
  - *onStart()* és *onStop()* közti állapot
- Előtér (**foreground**) életciklus:
  - *onResume()* és *onPause()* közti állapot

Activity életciklus

- A megfelelő életciklus függvény hívódik meg állapotváltáskor
- Az életciklus függvények felüldefiniálhatók
  - Az ősosztály függvényét kötelező meghívni (pl.: *super.onCreate();*)
- Fejlesztők felelőssége!



Activity állapotának elmentése 2/3



Activity állapotának elmentése 3/3

- Az *onSaveInstanceState()* tipikusan az *onPause()* és *onStop()* előtt hívódik meg
- Nincs rá garancia, hogy mindig meghívódik, például ha a felhasználó a „Vissza” gombbal lép ki (jelzi, hogy végzett ezzel az Activity-vel, nincs mit elmenteni)
- Belső változók és UI elemek értékét szokás ilyenkor elmenteni
- Semmiképp se használjuk perzisztens adatok mentésére!
- Az ős (super) implementációját mindig hívjuk meg
- A rendszer alapértelmezetten is menti az Activity és a rajta lévő UI elemek állapotát bizonyos szinten (lásd UI előadás)
- Tesztelés: képernyő elforgatásával

Konfiguráció változások kezelése az Activity-ben

- A készülék fontos paramétere néha változhat futás közben (képernyő orientáció, külső billentyűzet, nyelv, stb.)
- Ezen változások esetén a rendszer újraindítja az Activity-t (*onDestroy()* és egyből *onCreate()* hívás)
- Ok: a rendszer új erőforrásokat tölthet be az új konfigurációhoz (pl. háttér más lesz, ha változik az orientáció)
- Ilyenkor az állapot elmentésére az *onSaveInstanceState()* a legkézenfekvőbb
- Visszatöltéshez használható még az *onRestoreInstanceState()*, de az *onCreate()*-ben a jellemzőbb

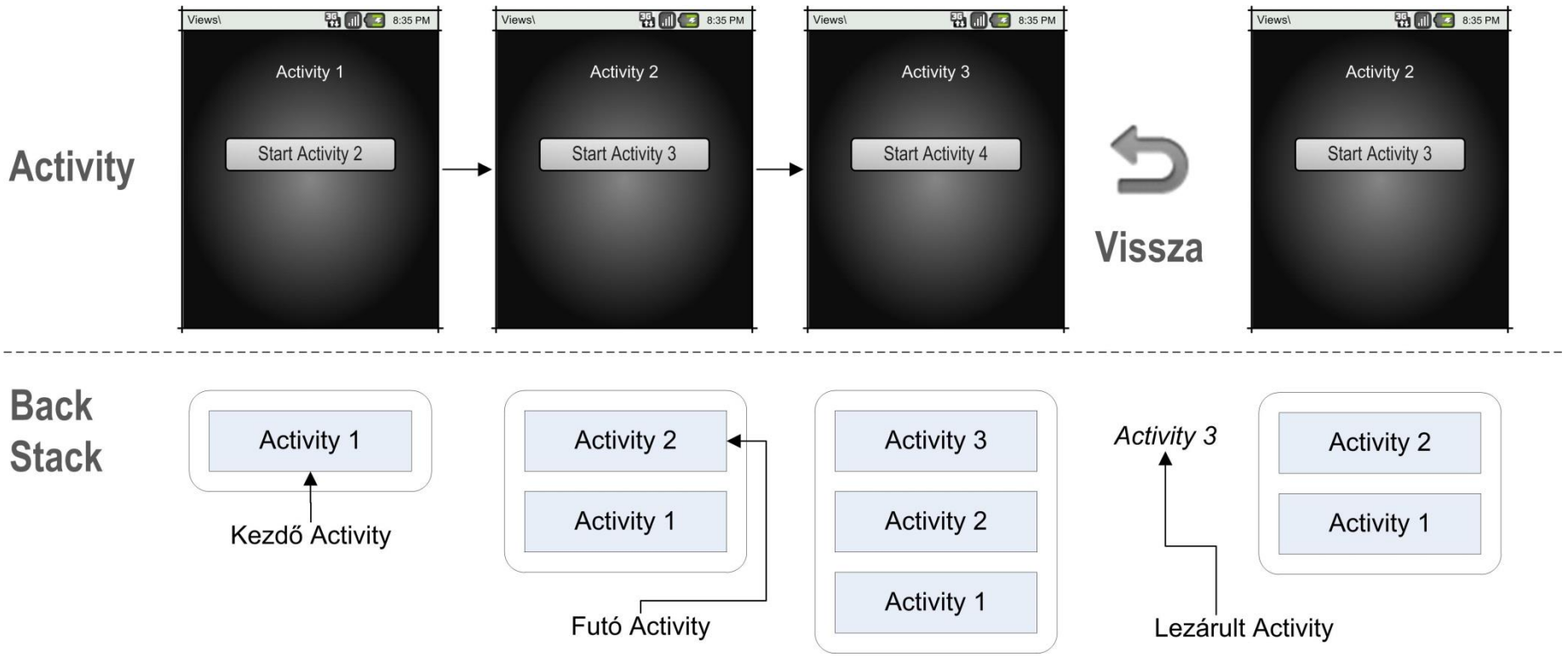
Activity váltás

- Életciklus callback függvények meghívási sorrendje:
  - A Activity *onPause()* függvénye
  - B Activity *onCreate()*, *onStart()* és *onResume()* függvénye (B Activity-n van már a focus)
  - A Activity *onStop()* függvénye, mivel már nem látható
- Ha a B Activity valamit adatbázisból olvas ki, amit az A ment el, akkor ez a mentés A-nak az *onPause()* függvényében kell megtörténjen, hogy a B aktuális legyen, mire a felhasználó előtt megjelenik

Activity Back Stack 1/2

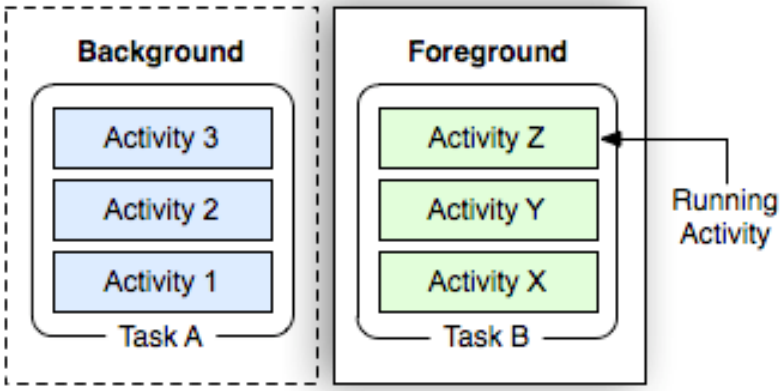
- Egy feladat végrehajtásához a felhasználó tipikusan több Activity-t használ
- A rendszer az Activity-ket egy ún. Back Stack-en tárolja
- Az előtérben levő Activity van a Back Stack tetején
- Ha a felhasználó átvált egy másik Activity-re, akkor eggyel lejjebb kerül a Stack-ben és a következő lesz legfelül
- Vissza gomb esetén legfelülről veszi ki a rendszer az megjelenítendő Activity-t
- Last in, first out

Activity Back Stack 2/2



Multitasking

- Task: Egy elvégzendő feladat, ami több Activity-t használ, de nem mindegyik Activity kell hogy ugyanabban az alkalmazásban szerepeljen
- A HOME gomb lenyomásával a rendszer visszatér a kezdő képernyőre és új taskot indíthatunk
- Új task indításakor a rendszer megőrzi az előző task Back Stack-jét, de memória gondok esetén bezárhat Activity-ket
- Az új task egy új Back Stack-et kap



Memória kezelés, processzek

- Activity külön processzben is futtatható (több memória)

```
<activity
        android:name="hu.bme.aut.JobActivity"
        android:process=":extrajob"
        android:label="@string/app_name"/>
```

Activity LAUNCH módok

- https://itnext.io/the-android-launchmode-animated-cheatsheet-6657e5dd9b0f
- https://www.youtube.com/watch?v=m8sf0UkJkxo

Az első Android alkalmazás Kotlin-ban – emlékeztető ☺

Egyszerű esemény kezelés

Kotlin extensions miatt használható

Lambda hívás

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        myTextView.append("#");

        myTextView.setOnClickListener{
            myTextView.append("\n--CLICKED--")
        }
    }
}
```

Függvény mint paraméter

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnTime.setOnClickListener(::click)
    }

    private fun click(view: View) {
        Toast.makeText(this,
                Date(System.currentTimeMillis()).toString(),
                Toast.LENGTH_LONG).show()
    }
}
```

Eseménykezelő megadása layout-ban

```
<Button
    android:id="@+id/btnTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="click"
    android:text="Show" />
```

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun click(view: View) {
        Toast.makeText(this,
                Date(System.currentTimeMillis()).toString(),
                Toast.LENGTH_LONG).show()
    }
}
```

Szöveges erőforrás paraméterezhetősége

- *strings.xml*-be:

```
<string name="txtPageViews">%1$d oldal
megtekintés</string>
```

- Használat:

```
var pvCount = 5
var pv = getString(R.string.txtPageViews, pvCount);
```

- További példa:
  - `<string name="txtPageViews">%1$d oldal %2$s szerző</string>`
  - `$d: decimális paraméter`
  - `$s: szöveges paraméter`
- *String.format(...)* is használható

Android LogCat

- Rendszer debug kimenet
- Beépített rendszer üzenetek is monitorozhatók
- Beépített Log osztály
  - v(String, String) (verbose)
  - d(String, String) (debug)
  - i(String, String) (information)
  - w(String, String) (warning)
  - e(String, String) (error)
- Log.i("MyActivity", "Pozíció: " + position);
- Átirányítható file-ba is

Új Activity indítása

- SecondActivity indítása:

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

Különböző képernyők támogatása 1/2

- Az Android futtatható különböző felbontású és sűrűségű képernyőkön
- A rendszer egyfajta mechanizmust biztosít az eltérő képernyők támogatására (1.6-tól felfele)
- A fejlesztő válláról a legtöbb munkát leveszi
- Csak a megfelelő erőforrásokat kell elkészíteni
- Például egy mobiltelefon és egy tablet képernyője tipikusan eltérő
- 3.2-es tablet API-tól felfele újabb módszerek (lásd később)

Különböző képernyők támogatása 2/2

- A rendszer automatikusan is skálázza és átméretezi az alkalmazás felületét, hogy minden készüléket támogasson
- De! mindenképp fontos, hogy a felhasználói felület és az erőforrások (képek) optimalizálva legyenek az egyes felbontásokhoz és sűrűségekhez
- Ezzel nagy mértékben növelhető a felhasználói élmény
- Továbbá valóban az egyes készülékekhez igazítható a megjelenítés, ami növeli a felhasználói elégedettséget
- A módszer követésével minden készüléket támogató alkalmazás készíthető UI szempontjából egyetlen .apk-ba csomagolva

Legfontosabb fogalmak 1/2

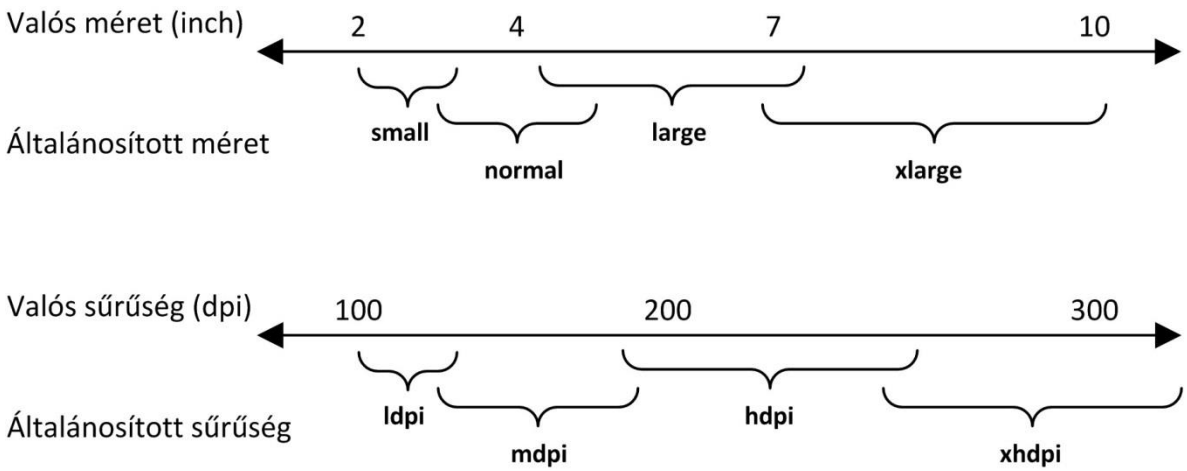
- Képernyő méret (*screen size*):
  - Fizikai képátló
  - Az egyszerűség kedvéért az Android 4 kategóriát különböztet meg: small, normal, large, és extra large
- Képernyő sűrűség (*screen density – dpi*): A pixelek száma egy adott fizikai területen belül, tipikusan inchenkénti képpont (dpi – dots per inch)
  - Az Android 6 kategóriát különböztet meg: low, medium, high és extra high
- Orientáció (*orientation*): A képernyő orientációja a felhasználó nézőpontjából:
  - Álló (*portrait*)
  - Fekvő (*landscape*)
  - Az orientáció futási időben is változhat, például a készülék eldöntésével
  - Lehetőség van rögzíteni az orientációt

Legfontosabb fogalmak 2/2

- Felbontás (*resolution – px*): Képernyő pixelek száma
  - A UI tervezésekor nem felbontással dolgozunk, hanem mérettel és pixel sűrűséggel
- Sűrűség független pixel (*density-independent pixel – dp*)
  - Virtuális pixel egység, amit UI tervezéskor célszerű használni
  - Egy dp egy fizikai pixelnek felel meg egy 160 dpi-s képernyőn (160 az egységes középérték)
  - A rendszer futási időben kezel minden szükséges skálázást a definiált dp-nek megfelelően
  - *px = dp * (dpi / 160)*
  - Például egy 240 dpi-s képernyőn, 1 dp 1.5 fizikai pixelnek felel meg

Általánosított képernyő méretek 1/2

- 6 általánosított méret:
  - small, normal, large és xlarge, stb.
- 6 általánosított sűrűség:
  - ldpi (low), mdpi (medium), hdpi (high), és xhdpi (extra high), stb.

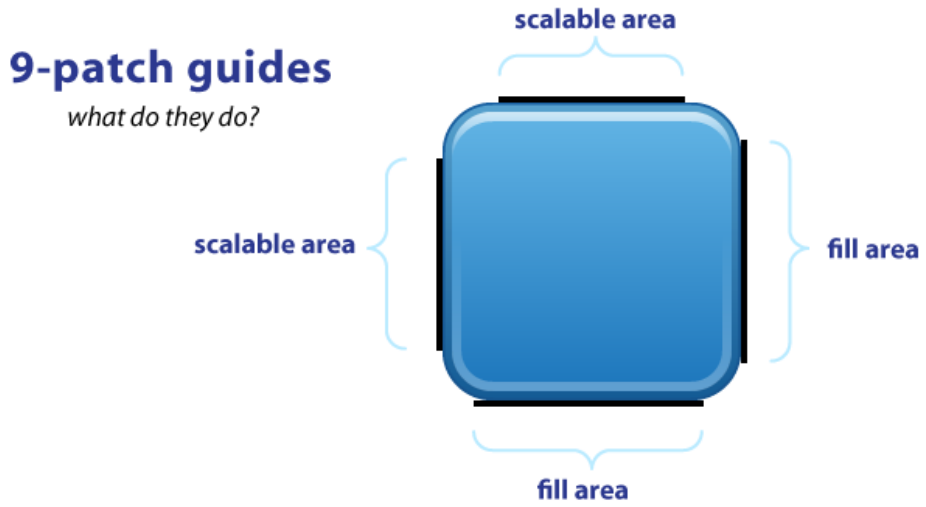


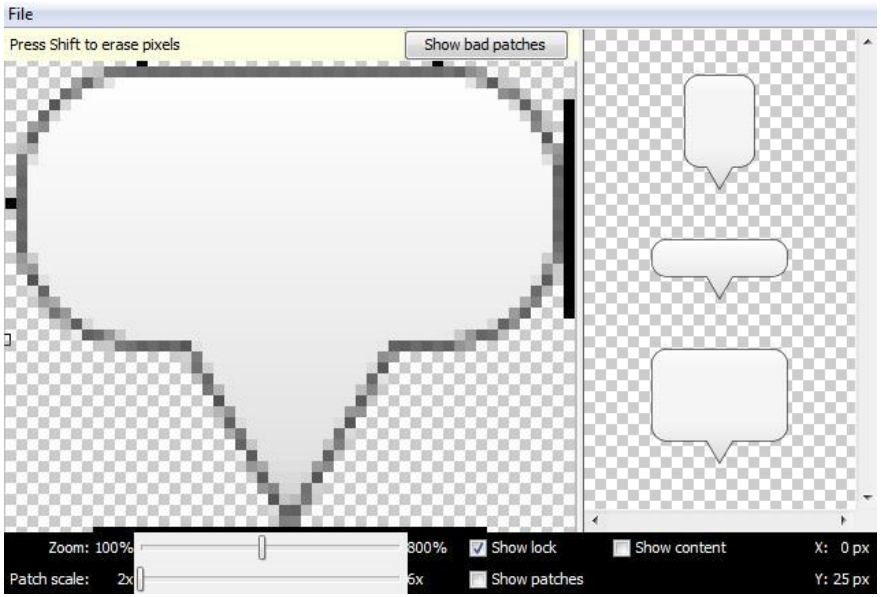
Általánosított képernyő méretek 2/2

- Definiált minimum küszöbök:
  - *xlarge*: legalább 960dp x 720dp
  - *large*: legalább 640dp x 480dp
  - *normal*: legalább 470dp x 320dp
  - *small*: legalább 426dp x 320dp
- 3.0-ás verzió alatt lehetnek bugok a normal és large megkülönböztetésében

NinePatch képek

- PNG képek skálázási szabályainak meghatározása
- SDK része: draw9patch.bat





Futás idejű működés

- A megjelenítés optimalizálása érdekében lehetőség van alternatív erőforrások megadására a különböző méretek és sűrűségek támogatásához
- Tipikusan különböző layout-ok és eltérő felbontású képek definiálása szükséges
- A rendszer futási időben kiválasztja a megfelelő erőforrást
- Általában nincs szükség minden méret és sűrűség kombináció megadására

Mi nem igaz az Android UI támogatására?

A. Az Android automatikusan átméretezi a képet, ha nincs megfelelően illeszkedő.

B. Az Anroid támogatja a sűrűségfüggetlen megjelenítést.

C. px = dp * (dpi / 160)

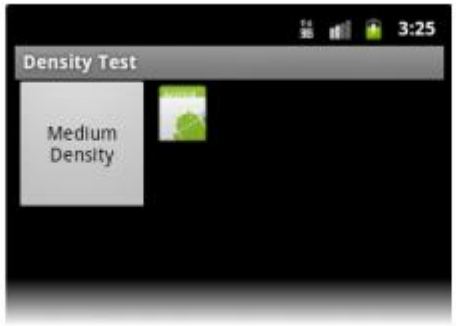
D. Közvetlenül pixelben nem adhatók meg a méretek.

Sűrűség függetlenség

- Az alkalmazás akkor lehet „sűrűség független”, ha a felhasználói felületi elemek a felhasználó szemszögéből megőrzik a fizikai méretüket különböző sűrűségeken
- A „sűrűség függetlenség” fenntartása nagyon fontos, hiszen például egy gomb fizikailag nagyobbnak tűnhet egy alacsonyabb sűrűségű képernyőn
- A képernyő sűrűséghez kapcsolódó problémák jelentősen befolyásolhatják az alkalmazás felhasználhatóságát.
- Az Android kétféle módon is segít elérni a sűrűség függetlenséget:
  - A rendszer a **dp** kiszámítása alapján skálázza a felhasználói felületet az aktuális képernyő sűrűségnek megfelelően
  - A rendszer a képernyő sűrűség alapján automatikusan átskálázza a kép erőforrásokat

Példa

- Sűrűség függetlenség támogatás nélkül:



- Sűrűség függetlenség támogatással:

Kép erőforrások átméretezése

- Nem szerencsés, ha a rendszerre bízzuk az átméretezést, hiszen így elmosódottak lehetnek a képek nagy felbontáson
- Az Android úgynevezett minősítő „string” (configuration qualifier)-ek segítségével teszi lehetővé, különböző erőforrások használatát
- A minősítő „string”-et az erőforrás könyvtár (res/) neve után kell fűzni (<resources_name>-<qualifier>, pl. *layout-xlarge*):
  - <resources_name>: standard erőforrás típus, pl. *drawable*, vagy *layout*
  - <qualifier>: minősítő a képernyőre vonatkozólag, pl. *hdpi*, vagy *large*
  - Több minősítő is szerepelhet egymás után kötőjellel elválasztva

Legfontosabb minősítő értékek

- Méret:
  - small, normal, large, xlarge
- Sűrűség:
  - ldpi, mdpi, hdpi, xhdpi, nodpi (a rendszer az ebben lévőket nem méretezi át), tvdpi
- Irány:
  - land, port
- Képarány:
  - long (a jelentősen szélesebb, vagy magasabb kijelzőkhöz), notlong

Példák

- ***res/layout/my_layout.xml***
- *res/layout-small/my_layout.xml*
- *res/layout-large/my_layout.xml*
- *res/layout-xlarge/my_layout.xml*
- *res/layout-xlarge-land/my_layout.xml*

- *res/drawable-mdpi/my_icon.png*
- *res/drawable-hdpi/my_icon.png*
- *res/drawable-xhdpi/my_icon.png*

Erőforrás választó algoritmus

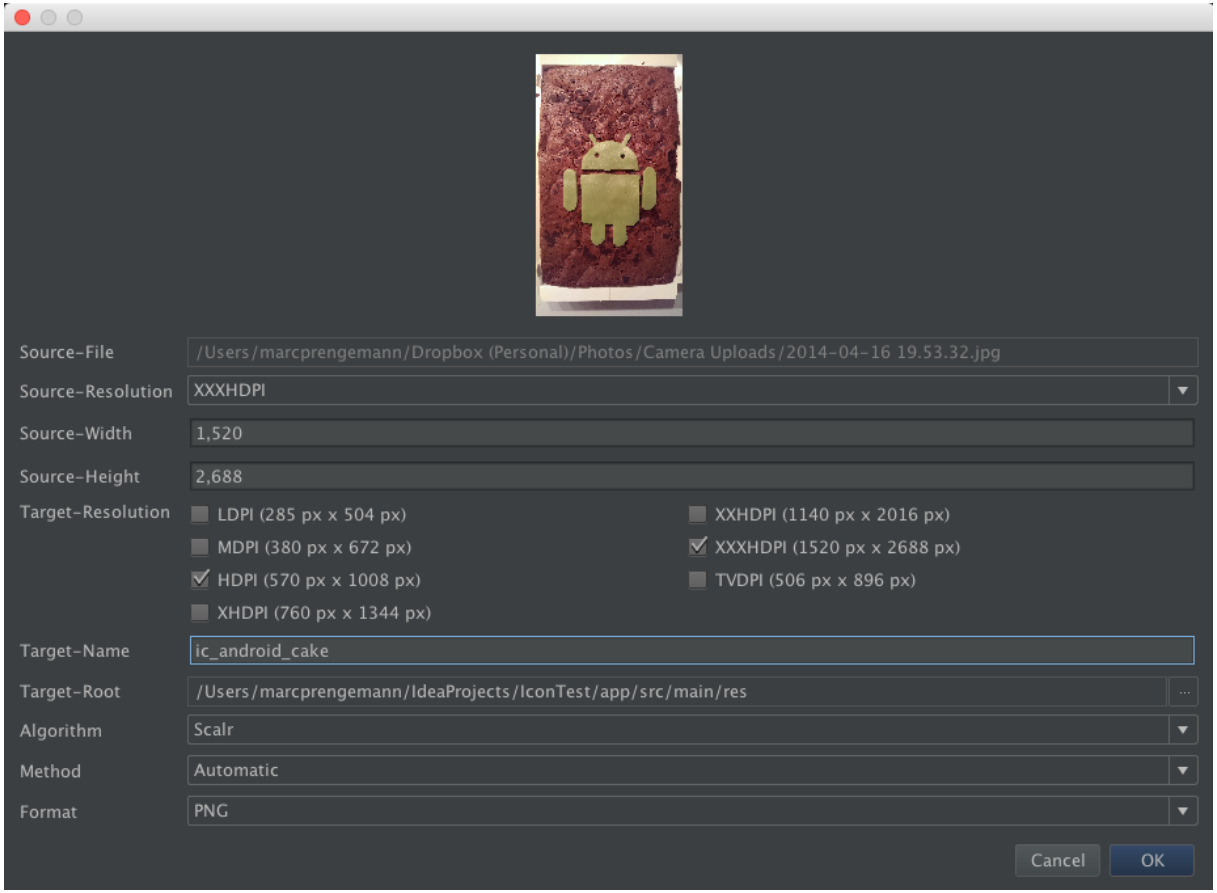
- Futás közben egy meghatározott logika alapján választ a rendszer
- Megkeresi a passzoló erőforrást
- Ha nincs az aktuálishoz passzoló, akkor egy kisebb/alacsonyabb sűrűségűt választ (pl. *large* mérethez *normal* méretet választ)
- Amennyiben az elérhető erőforrások csak nagyobb képernyőkhöz vannak, mint a készülék képernyője, akkor hibát jelez az alkalmazás
- Például ha az összes egy típusú erőforrás *xlarge*-al van megjelölve, akkor *normal* képernyős eszközökön hiba keletkezik

Mire érdemes figyelni?

- Összefoglalva, az alkalmazásnak biztosítani kell, hogy:
  - Elfér és használható kis képernyőkön
  - Kihasználja a nagy képernyőket és a rendelkezésre álló teret
  - Álló és fekvő mód megfelelően kezelve van
- Bitmap-ok esetén érdemes a 3:4:6:8 skálázási arányt követni, pl:
  - 36x36 low-density
  - 48x48 medium-density
  - 72x72 high-density
  - 96x96 extra high-density

Android Drawable Importer

- https://plugins.jetbrains.com/plugin/7658?pr=



Felület kialakítás tableteken

- Az első tabletek megjelenésekor (Android 3.0), leginkább az *xlarge* minősítőt lehetett használni, pl. *res/layout-xlarge*
- Azonban például a 7”-os tabletek ugyanúgy a large kategóriába tartoztak, mint az 5”-os telefonok, azonban látványos volt köztük a méretbeli különbség
- Ezért Android 3.2-ben egy sokkal diszkrétebb módszert vezettek be: mi az a méret (dp-ben), amekkorára legalább szükség van
- Így tehát egyszerűen beállítható, hogy mi az a méret, ami fölött már a tabletek számára tervezett elrendezést használja az alkalmazás
- Fejlesztési módszerek:
  - Minimum szélesség választása és ennek megfelelő tervezés
  - Mi az a legkisebb szélesség, amit a tervezett elrendezés már támogat
- *Fragment API (Android 3.0-tól): több nézet definiálása egy képernyőn*

Új méret-minősítők tableteken

- *sw<N>dp* (smallestWidth):
  - Például *sw600dp, sw720dp*
  - A képernyőn látható legkisebb dimenziót specifikálja
  - Másképp: a képernyőn elérhető legkisebb magasság és szélesség
  - A minősítővel biztosítható, hogy a képernyő orientációjától függetlenül biztos, hogy van <N> dp szélesség
- *w<N>dp* (available screen width):
  - A legkisebb szélesség, amivel az erőforrást lehet használni
  - Orientáció változáskor értesül a rendszer
- *H<N>dp* (available screen height):
  - A legkisebb magasság, amivel az erőforrást lehet használni
  - Orientáció változáskor szintén értesül a rendszer

Tipikus méretek

- **320dp**: tipikus telefon képernyő (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, stb).
- **480dp**: kisebb tabletek(480x800 mdpi).
- **600dp**: 7” tablet (600x1024 mdpi).
- **720dp**: 10” tablet (720x1280 mdpi, 800x1280 mdpi, stb.).

Például:

> res/layout/main_activity.xml # Mobilok számára (kevesebb mint 600dp szélességgel)
> res/layout-sw600dp/main_activity.xml # 7” tabletek számára (600dp széles, vagy nagyobb)
> res/layout-sw720dp/main_activity.xml # 10” tabletek számára (720dp széles, vagy nagyobb)

Alkalmazás szintű képernyő követelmények

- *android:requiresSmallestWidthDp*
  - A legkisebb dimenzió, amivel a képernyőnek rendelkeznie kell
  - <manifest ... >
    <supports-screens android:requiresSmallestWidthDp="600" />
    ...
    </manifest>
- *android:compatibleWidthLimitDp*
  - Maximum legkisebb szélesség, amit még az alkalmazás támogat
- *android:largestWidthLimitDp*

Legfontosabb tényezők

- Használjuk a **wrap_content**, **match_parent**, vagy **dp** egységeket, amikor egy felületet készítünk!
- Súlyozás
- Ne használjunk beégetett pixel értékeket!
- Ne használjuk az AbsoluteLayout-ot (elavult)!
- Mindenképp készítsünk különböző kép erőforrásokat az eltérő képernyősűrűségekhez
- Szövegek méretezéséhez érdemes használni az sp (scale-independent pixel) mértéket, dp-hez hasonlóan működik

Mi igaz az Android UI támogatására?

A. Az Android nem támogatja a sűrűségfüggetleneséget.

B. Ha nincs a képernyő tulajdonságaihoz illeszkedő erőforrás direkt megadva, akkor kivétel dobódik.

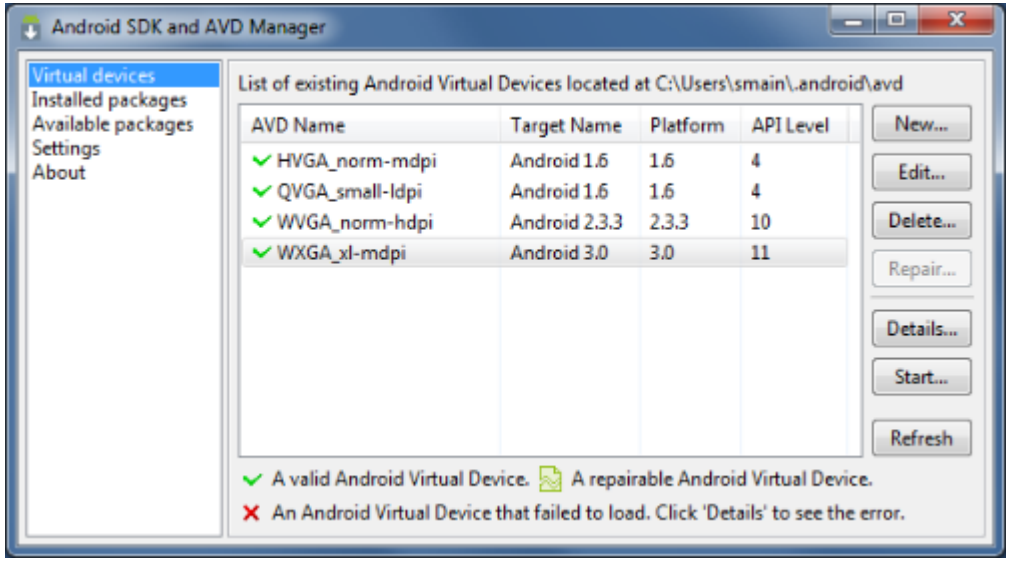
C. A dp mértékegység helyett a dpi-t javasolt használni.

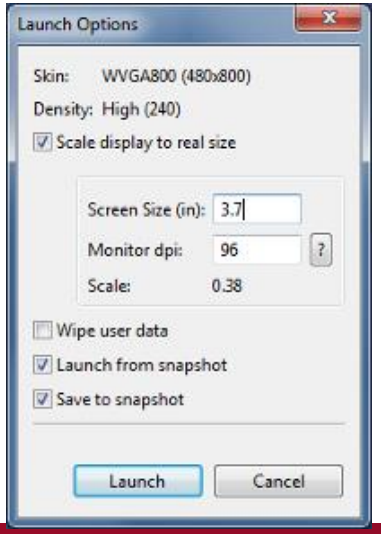
D. Az Android futás közben tudja kikeresni a leginkább illeszkedő erőforrást.

http://babcomaut.aut.bme.hu/votes/

Felhasználói felület tesztelése

- Az alkalmazás kiadása előtt mindenképp tesztelni kell a felhasználói felületet
- Az Android SDK támogat különféle emulátor skin-eket
- Tetszőlegesen beállítható az emulátor mérete, felbontása és pixelsűrűsége
- A teszteléshez létre kell hozni több AVD-t
- Skálázás és sűrűség megadása parancssorból indítás esetén (scale: 0.1-3):
  - emulator -avd <avd_name> -scale 96dpi





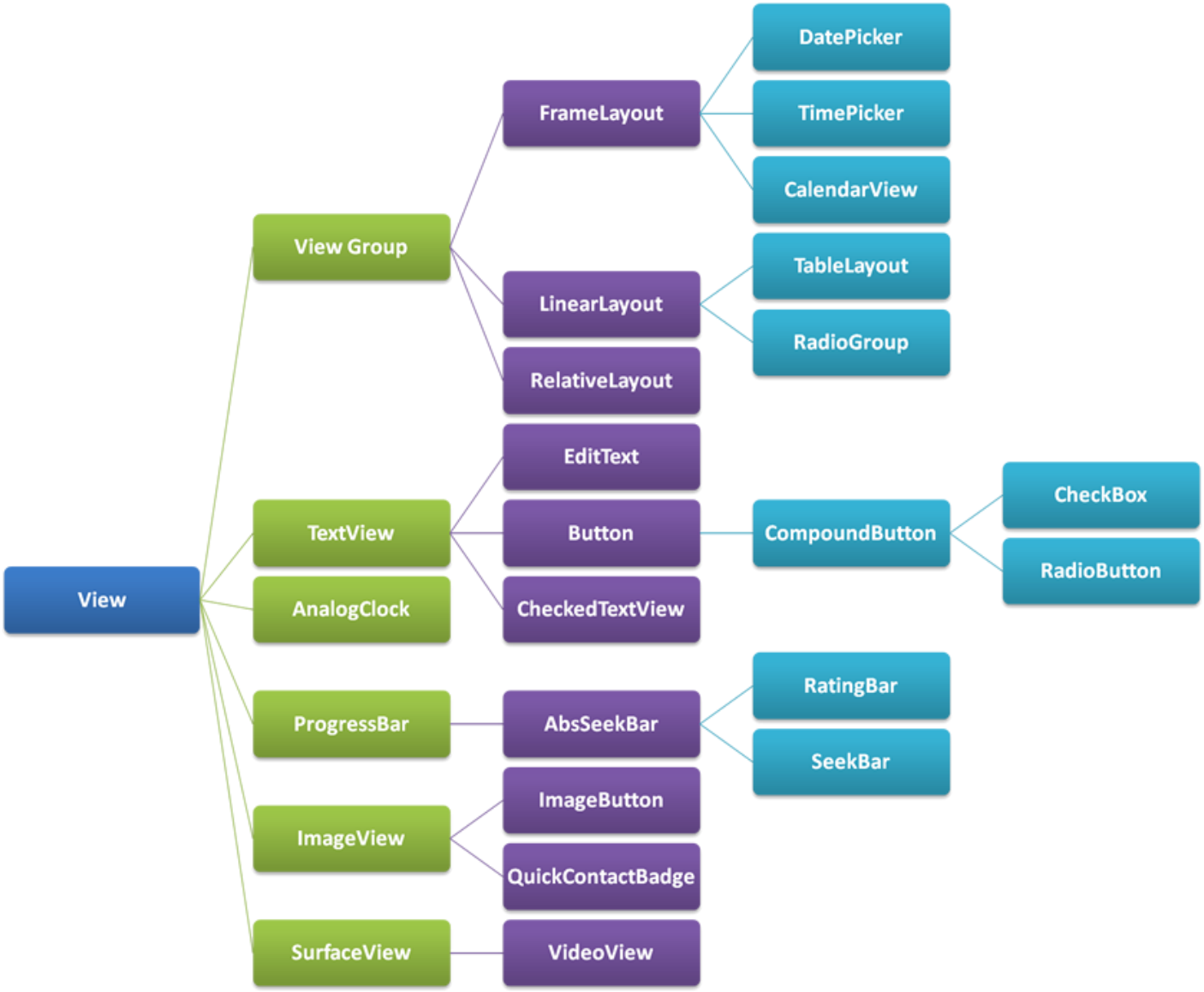
Felhasználói felület tervezése

- Elrendezés és erőforrások definiálása XML-ben
- UI erőforrás hozzárendelése Activity-hez
- Felületei elemek elérése forráskódból
  - *findViewById([erőforrás azonosító – R.id.X])*
- Dinamikus felhasználói felület kezelés
- Animációk támogatása

Felhasználói felület erőforrások

- Felületek
  - *res/layout*
- Szöveges erőforrás:
  - *res/values/strings.xml*
- Kép erőforrások:
  - *res/drawable-xyz/[kep].[ext]*
- Animáció erőforrások
  - *res/anim*
- *További erőforrások: animator, szín, menü, nyers (raw), xml fileok*

Android UI architektúra



Egyedi nézetek

- View leszármazott
- Beépített nézetek és *LayoutGroup*-ok is felüldefiniálhatók, pl. saját nézet *RelativeLayout*-ból leszármaztatva
- *<merge>* XML elem
- XML-ek egymásba ágyazhatósága: *<include>*

Egyedi felületi nézet

- Teljesen egyedi felületi elemek definiálása
- Meglévő felületi elemek kiegészítése
- Érintés események kezelése
- Dinamikus rajzolás
  - Színek, rajzolási stílus
  - Gyakori alakzatok: vonal, négyzet, kör stb.
  - Szöveg rajzolása
  - Képek megjelenítése
- Megjelenítési méretthez való igazodás
- XML-ből is használható!

Összefoglalás

- Magas szintű támogatás különböző méretű és felbontású kijelzők kezelésére
- XML alapú felületleírás és egyéb erőforrás kezelés
- Gazdag Layout és UI vezérlő készlet
- Animációk támogatása
- Grafikai elemek leírása XML-ben
- Lokalizáció támogatás XML alapokon

Hogy is volt?

- Egy Android alkalmazás milyen komponensekből épülhet fel?
- Mi a Service komponens?
- Miket kell tartalmaznia a manifest állománynak?
- Az Activity callback életciklus-függvények felüldefiniálásakor meg kell-e hívni kötelezően az ős osztály implementációját?
- Ha A Activity-ből átváltunk B Activity-re, milyen sorrendben hívódnak meg az életciklus függvények?
- Magyarázza el az Activity Back Stack működési elvét!

Hogy is volt?

- Mit értünk a sűrűségfüggetlen pixel fogalom alatt?
- Egy 320 dpi-s képernyőn, 1 dp mennyi fizikai pixelnek felel meg ?
- Sorolja fel a legfontosabb Android Layout-okat!
- Vázolja fel egy Android alkalmazás kódját, mely egy gombot jelenít meg és a gombot lenyomva a „Clicked!” szöveg jelenik meg egy Toast-ban!
- Hogy biztosítja az Android a lokalizáció támogatását?

Kérdések