Android fejlesztés

Activity életciklus, több képernyős alkalmazások, felhasználói felület alapok

peter.ekler@aut.bme.hu

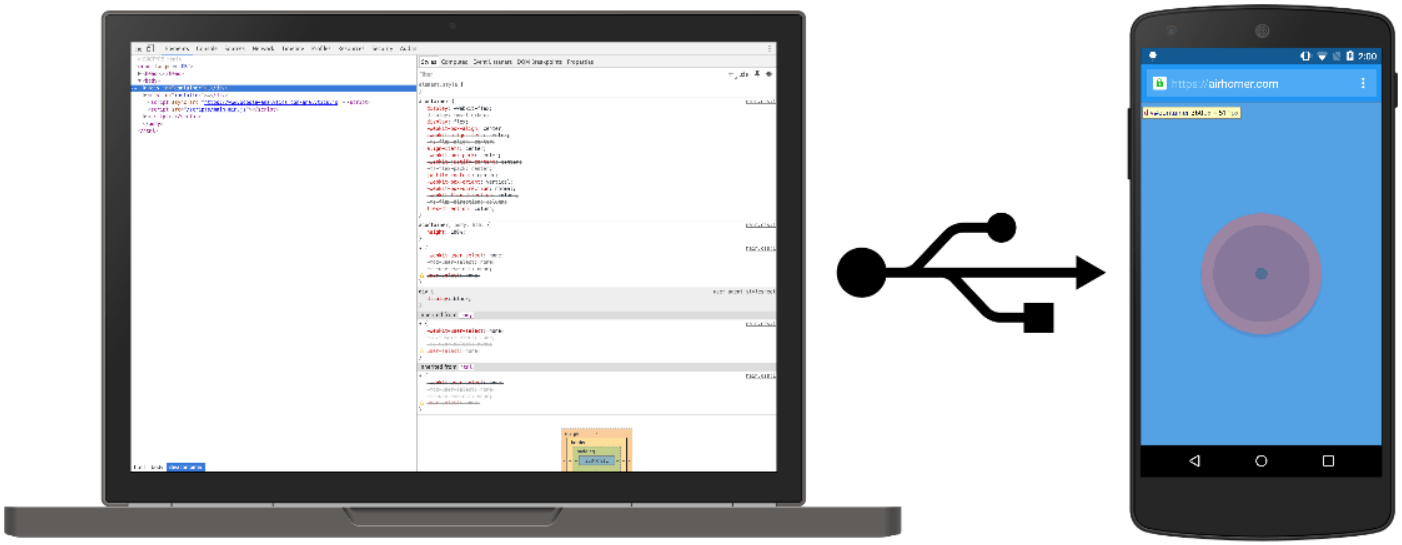
BME AUT Department of Automation and Applied Informatics

Miről volt szó az előző alkalommal? ☺

- Android platform szerkezete
- Android verziók
- Android projekt felépítése
- Android alkalmazás szerkezete
- Alapvető felhasználói felületi elemek
- Egyszerű alkalmazás fejlesztése

Tesztelés valós telefonon

- Fejlesztői mód bekapcsolása
- USB debugging engedélyezése
- Készülék összekötése és jóváhagyás



Tartalom

- Activity komponens
- Activity életciklus
- Több Activity-s alkalmazások
- Állapot megőrzés az életciklus során
- Menük kezelése
- Felhasználói felület tervezése, egyedi nézetek, rajzolás

Erőforrások kezelése

Alkalmazás erőforrások

- Egy Android alkalmazás nem csak forráskódból áll, hanem erőforrásokból is, úgy mint: képek, hanganyagok, stb.
- Emellett erőforrások az XML-ben definiált felületek is: elrendezés, animáció, menü, stílus, szín.
- Erőforrások használatával sokkal rugalmasabban változtatható az alkalmazás
- Minden erőforráshoz a rendszer automatikusan egy egyedi azonosítót generál, amin keresztül elérhető a forráskódból

Erőforrás hivatkozás példa

- Tegyük fel, hogy készítettünk egy *logo.png*-t és elmentettük a *res/drawable/* könyvtárba
- Az SDK eszköz előállít egy egyedi erőforrást hozzá mentés után automatikusan
- Az azonosító: *R.drawable.logo*
- Ezzel az azonosítóval lehet hivatkozni bárhol az erőforrásra
- Az azonosítók az *R.java* állományban tárolódnak (soha ne módosítsuk ezt az állományt!)

Erőforrás használat előnyei

- Az egyik legnagyobb előny, hogy a készülék képességeihez lehet igazítani az erőforrásokat
- A könyvtárak után „minősítő”-ket írhatunk, amellyel megadjuk hogy mely tulajdonságok teljesülése esetén vegye a rendszer ebből a könyvtárból az erőforrásokat
- Többnyelvűség támogatása:
  - *strings.xml*
  - *res/values/*
  - *res/values-fr/*
  - *res/values-hu/*

Activity életciklus

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

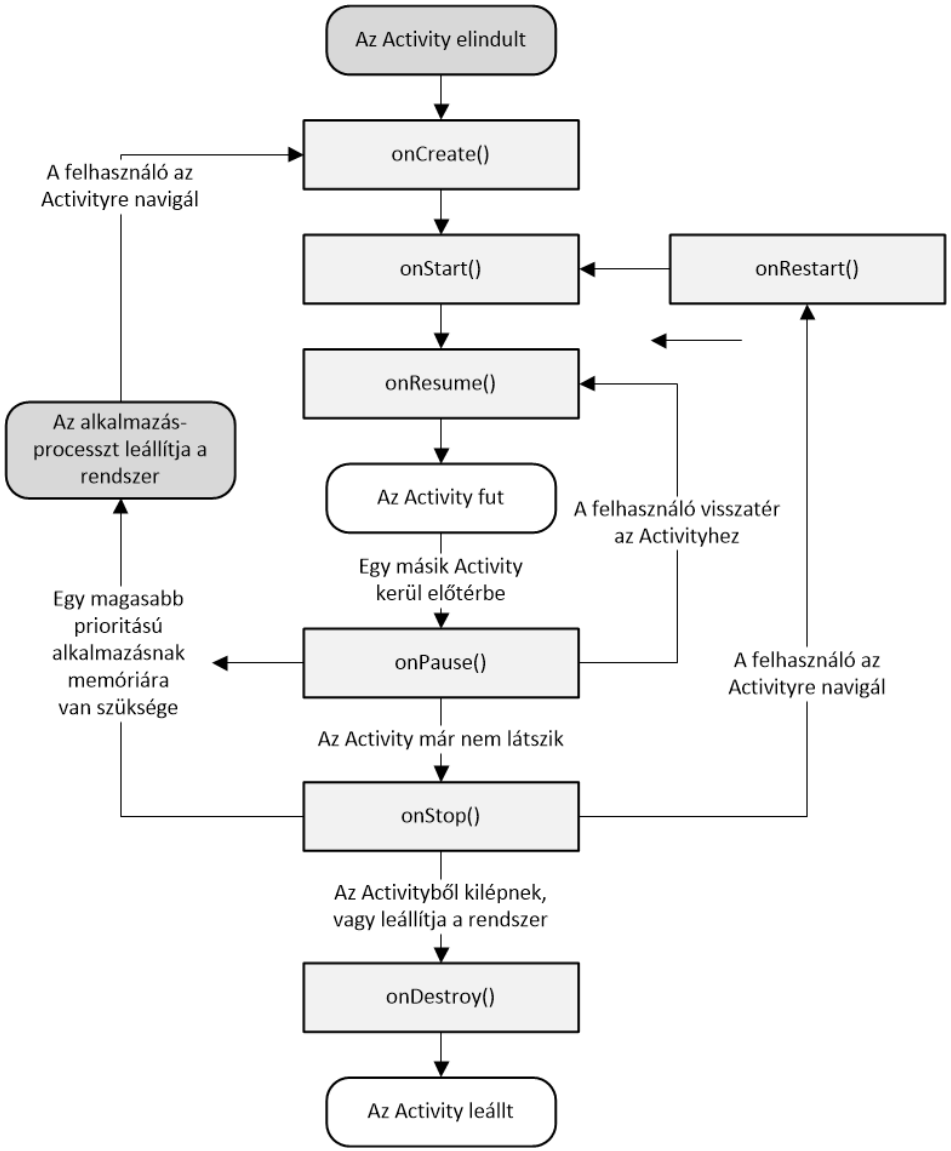
Activity életciklus-callback

- Amikor egy Activity leáll egy másik indulása miatt, az Activity az eseményről értesítést kap az úgynevezett életciklus-callback metódusokon keresztül
- Számos callback metódus támogatott (create, stop, resume, destroy, stb.), amikre megfelelően reagálhat az Activity
- Például stop esemény hatására tipikusan a nagyobb objektumokat érdemes elengedni (DB/hálózati kapcsolat)
- Amikor az Activity visszatér (resume), újra kell kérni az erőforrásokat
- Ezek az átmenetek tipikus részei az Activity életciklusának

Activity életciklus

- Egy megbízható és flexibilis alkalmazás esetén kritikus fontosságú az Activity életciklus-callback függvények megfelelő felüldefiniálása
- Az Activity életciklusát a vele együttműködő többi Activity határozza meg
- Elengedhetetlen az Activity működésének tesztelése a különböző életciklus állapotokban

Activity életciklus modell



Activity bezárása a rendszer által

- Paused, vagy Stopped állapotban a rendszer bármikor leállíthatja memória-felszabadítás céljából
- A leállítás történhet a *finish()* hívással, vagy kritikusabb esetben a Process leállításával
- Ha az Activity-t újra megnyitják (miután be lett zárva), a rendszer újra létrehozza

Életciklus callback függvények

- Amikor az Activity állapotot vált, megfelelő callback függvények hívódnak meg
- Ezek a callback függvények „hook” jellegű függvények, melyeket a rendszer hív
- Fontos a metódusok felül definiálása és a megfelelő részek implementálása
  - > **Mindig meg kell hívni az ős osztály implementációját is (pl. `super.onCreate()`)!**
- A rendszer felelőssége meghívni ezeket a függvényeket, de a fejlesztő felelőssége a helyes implementáció

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 életciklus callback függvények 1/2

- *onCreate()*: Activity létrejön és beállítja a megfelelő állapotokat (layout, munka szálak létrehozása, stb.)
- *onDestroy()*: Minden még lefoglalt erőforrás felszabadítása
- *onStart()*: Az Activity látható, a vezérlők is. Például BroadcastReceiverek-re feliratkozás, amik módosítják a UI-t
- *onStop()*: Az Activity nem látható. Például BroadcastReceiverek-ről leiratkozás
  - > Az Activity élete során többször válthat látható és nem látható állapotok között.

Activity életciklus callback függvények 2/2

- *onRestart()*: Az Activity leállítása (onStop()) majd újraindítása után hívódik meg, még az indítás (onStart()) előtt
- *onResume()*: Az Activity láthatóvá válik és előtérben van, a felhasználó eléri a vezérlőket és tudja kezelni azokat
- *onPause()*: Az Activity háttérbe kerül, de valamennyire látszik a háttérben, például egy másik Activity pop-up jelleggel előjön, vagy sleep állapotba kerül a készülék

Mi igaz az Activity életciklus függvényekre?

A. Kötelező minden életciklus függvényt felüldefiniálni, különben nem fordul az alkalmazás kódja.

B. Kötelező az ősosztály implementációjának meghívása.

C. Az Activity élete során minden függvény csak egyszer hívódhat meg.

D. Szükség esetén manuálisan is meg kell hívni.

Több Activity használata

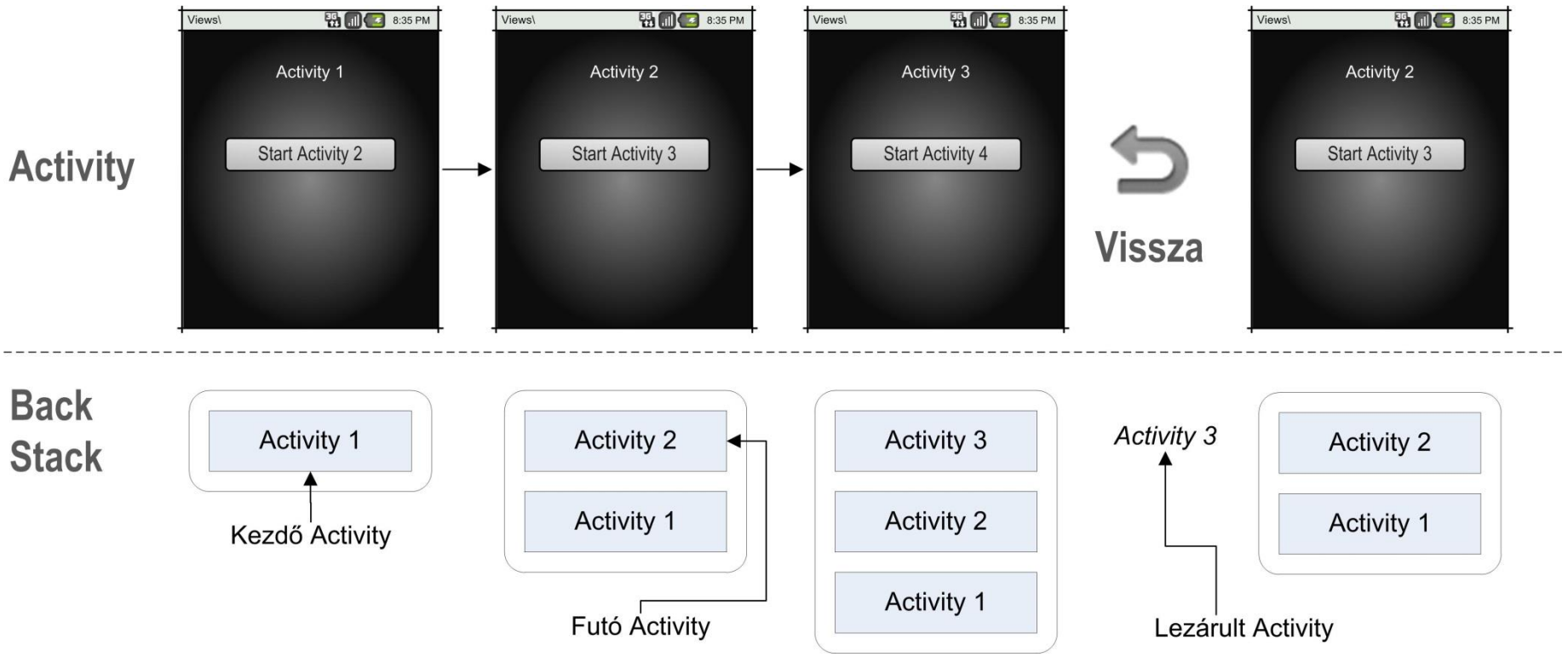
Activity váltás

- 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



Activity vezérlés 1/2

- Legtöbb esetben az alapértelmezett Back Stack viselkedés kielégíti az igényeket
- Néha azonban szükség lehet ezen alapértelmezett viselkedés felül definiálására
- Back Stack törlése, ha a Vissza hatására mindig egy kezdő Activity-re kell visszalépni
- Az alapértelmezett viselkedés felülírása:
  - Manifest állományban az <activity>-be
  - *startActivity(…)* fv. Paramétereként
- Amennyiben az alapértelmezett viselkedést módosítjuk, mindenképp teszteljük az alkalmazást navigálás és felhasználói élmény szempontjából, mert sokszor a programozó szempontjából jó megoldás nem ideális felhasználói szempontból

Activity vezérlés 2/2

- Az <activity> tag attribútumai (új Activity hogyan viselkedjen a többihez képest):
  - taskAffinity: melyik taskhoz tartozik
  - launchMode: indítási mód (mindig új példány, stb.)
  - allowTaskReparenting: új taskhoz kerül át
  - clearTaskOnLaunch: minden Activity-t töröl a task-ból
  - alwaysRetainTaskState: a rendszer kezelje-e a task állapotát
  - finishOnTaskLaunch: le kell-e állítani az Activity-t ha a felhasználó kilép a task-ból (pl. HOME gomb)
- *startActivity(…)* függvény paraméter értékei (az új Activity hogyan viselkedjen a most futóhoz képest):
  - FLAG_ACTIVITY_NEW_TASK
  - FLAG_ACTIVITY_CLEAR_TOP
  - FLAG_ACTIVITY_SINGLE_TOP
- További részletek az Intent előadásban

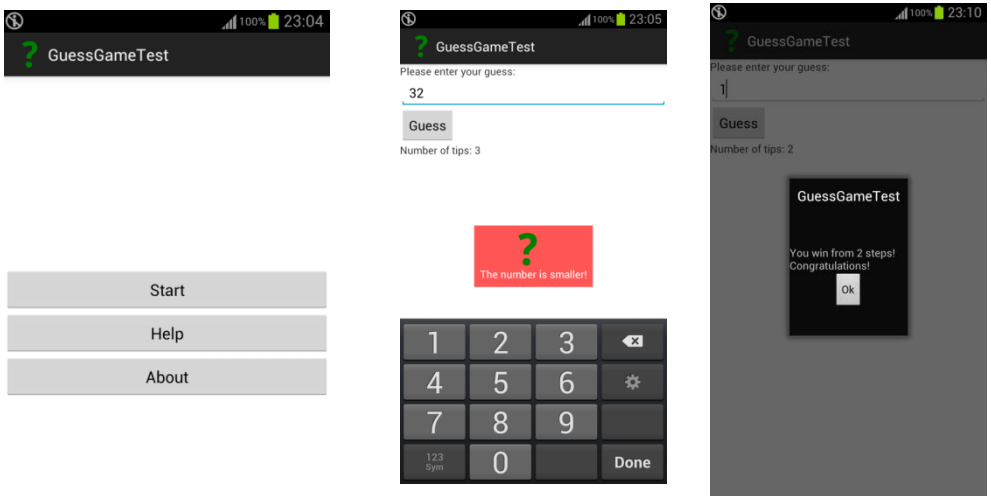
Új Activity indítása

- 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)
}
```

Egészítsük ki a Barkóba alkalmazást

- Készítsünk egy kezdő „menü” képernyőt
- Készítsünk egy eredmények képernyőt, ahol egy új Activity jelenik meg dialógus formában



Menü kezelés

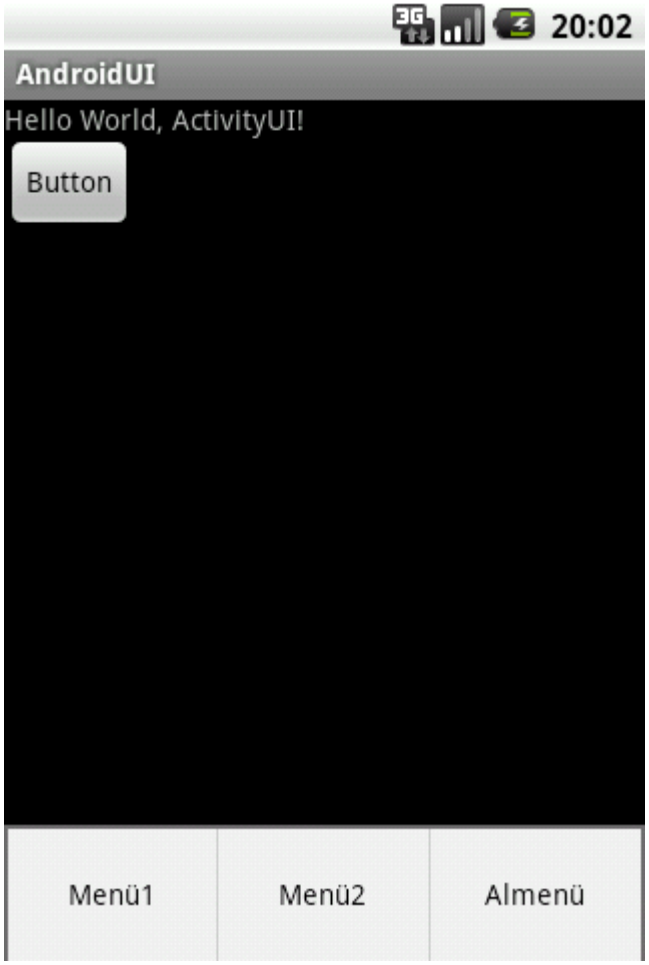
Menük

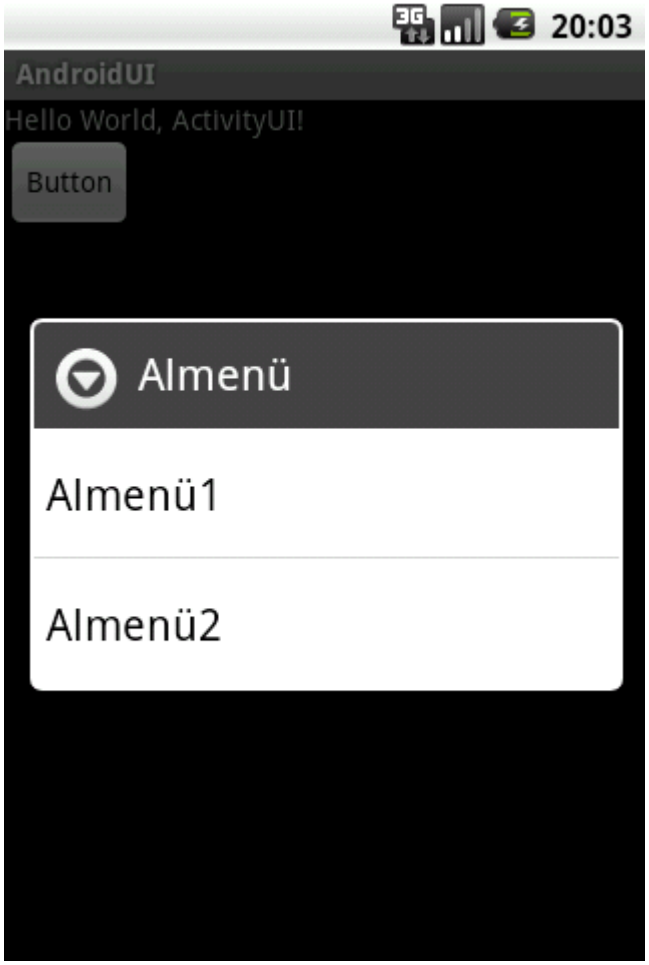
- *ActionBar->Toolbar* része
- Menü definiálása kódból
- Menü definiálása erőforrásból
- Dinamikus menük
  - Láthatóság beállítása
  - Manipuláció Java kódból
- Almenük támogatása

Menü erőforrás

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1"/>
    <item android:id="@+id/item2"
          android:title="@string/item2"/>
    <item android:id="@+id/submenu"
          android:title="@string/submenu_title">
        <menu>
            <item android:id="@+id/submenu_item1"
                  android:title="@string/submenu_item1" />
            <item android:id="@+id/submenu_item2"
                  android:title="@string/submenu_item2" />
        </menu>
    </item>
</menu>
```

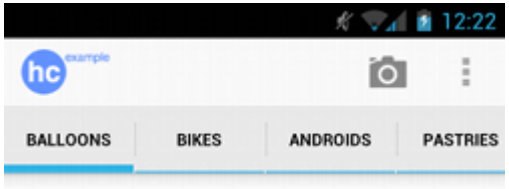
Menü és almenü





ActionBar és Menük

- Dedikált alkalmazás menü, logo és cím
- Tipikus felhasználás:
  - > Menü
  - > Branding (logo/background) és alkalmzás ikon
  - > Konzisztens alkalmazás navigáció
  - > Fő funkciók bemutatása



ActionBar specifikus XML menü paraméterek

```
<item android:id="@+id/action_time"
      android:title="@string/action_show_time"
      android:orderInCategory="5"
      android:icon="@drawable/clock_icon"
      android:showAsAction="always|withText"
/>
```

ActionBar -> Toolbar

- ActionBar helyett ToolBar
- Sokkal dinamikusabb viselkedés
- Menü erőforrások támogatása
- Custom elemek támogatása
- Manuális pozicionálás
- Toolbar tutorialok:
  - http://javatechig.com/android/android-lollipop-toolbar-example
  - http://www.101apps.co.za/index.php/articles/using-toolbars-in-your-apps.html

Toolbar használat

- ActionBar nélküli téma(styles.xml):
  > Theme.AppCompat.NoActionBar
- Layout erőforrás:

```
<android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:minHeight="?attr/actionBarSize"
        android:background="#2196F3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
</android.support.v7.widget.Toolbar>
```

- Activity onCreate(…):

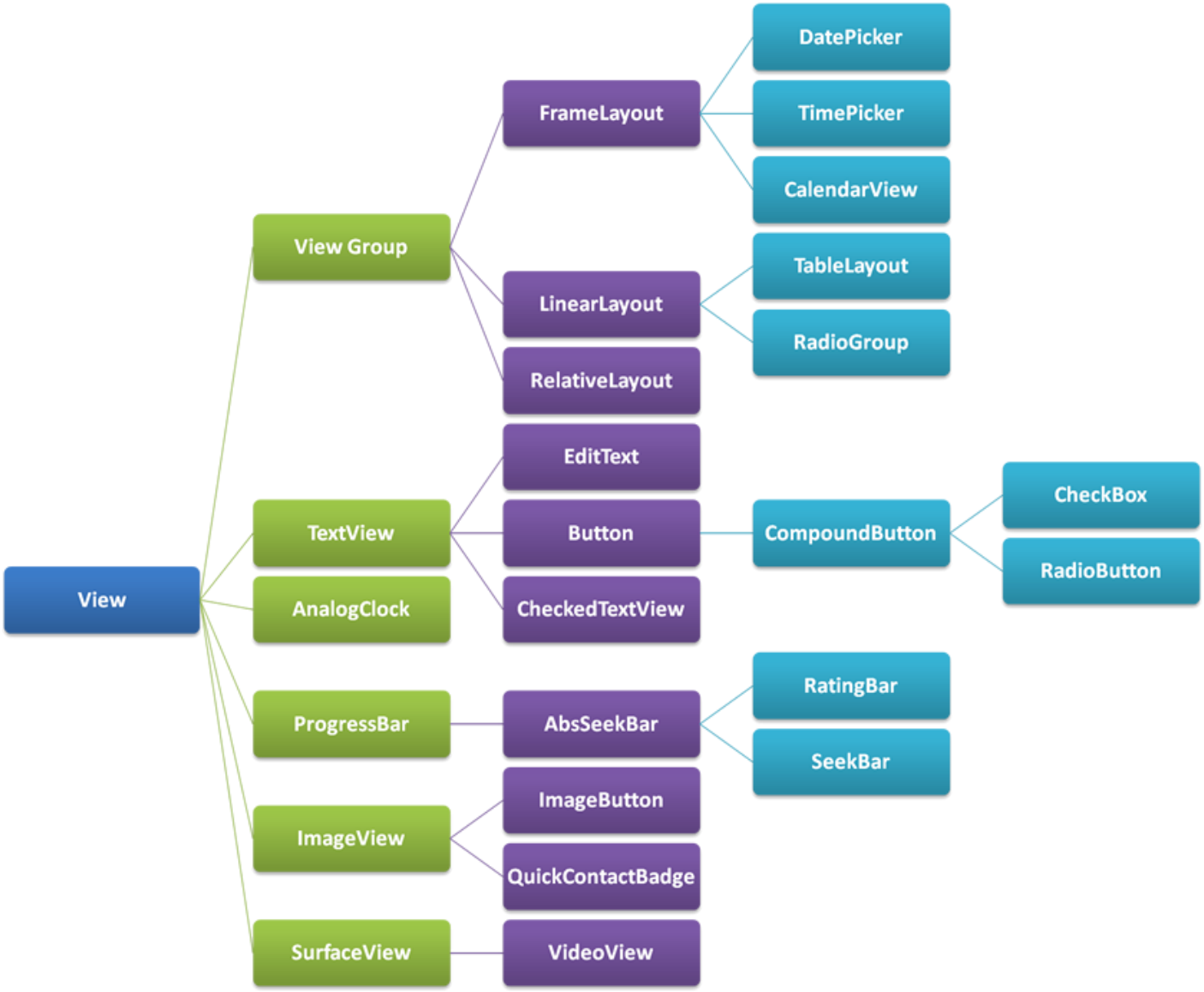
```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Set a toolbar to replace the action bar.
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
}
```

Felhasználói felület alapok

- *egyedi felületi elemek*

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*>

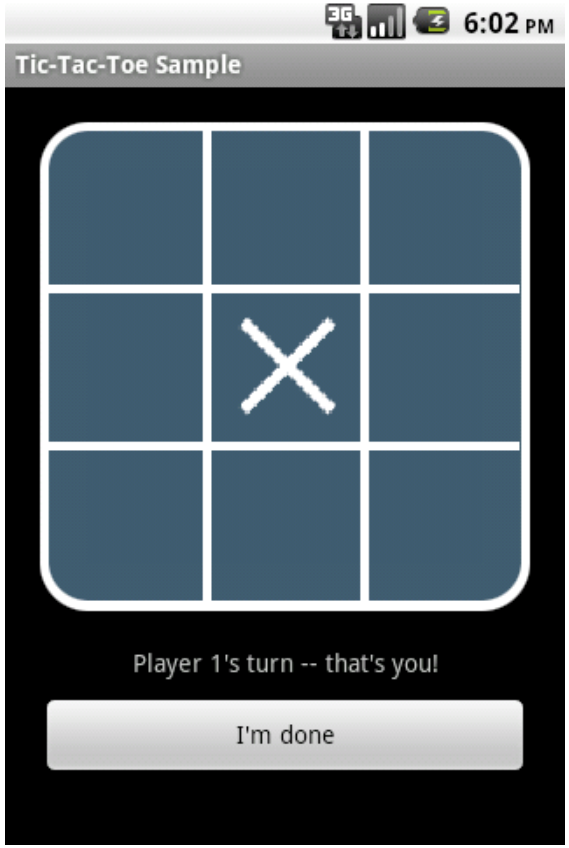
Egyedi felületi nézet

- 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 mérethez való igazodás
- XML-ből is használható!

TicTacToe

Gyakoroljunk

- Készítsünk egy TicTacToe játékot!



Singleton – object (Kotlin)

- Java:

```
public class TicTacToeModel {

    private static TicTacToeModel instance = null;

    public static TicTacToeModel getInstance() {
        if (instance == null) {
            instance = new TicTacToeModel();
        }

        return instance;
    }

    private TicTacToeModel(){

    }

}
```

- Kotlin:

```
object TicTacToeModel {


}
```

Játéktér elrendezése

- Próbáljunk *LinearLayout*-ba másik *LinearLayout*-ot elhelyezni és használjuk a „gravity” tulajdonságot

- Négyzet méret kikényszerítése a *TicTacToeView*-ban:

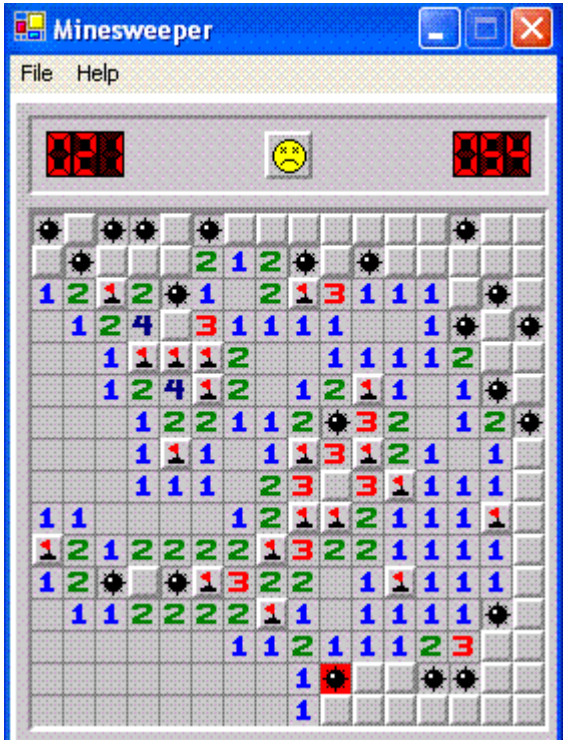
```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    val w = View.MeasureSpec.getSize(widthMeasureSpec)
    val h = View.MeasureSpec.getSize(heightMeasureSpec)
    val d = if (w == 0) h else if (h == 0) w else if (w < h) w else h
    setMeasuredDimension(d, d)
}
```

Házi feladat

1. Használjon különböző színeket az X-hez és az O-hoz
2. Fejezze be a TicTacToe algoritmust (győzelem/döntetlen)
3. Tegye lehetővé, hogy újra lehessen kezdeni a játékot

Szorgalmi feladat

- Aknakereső játék



Aknakereső tippek

- Field osztály

```
data class Field(var type: Int, var minesAround: Int,
                 var isFlagged: Boolean, var wasClicked: Boolean)
```

- Model - tárolás:

```
val fieldMatrix: Array<Array<Field>> = arrayOf(
        arrayOf(Field(1, 5, true, true),
                Field(2, 6, false, false)),
        arrayOf(Field(3, 7, true, true),
                Field(4, 8, false, false)))
```

- Használat:

```
fieldMatrix[0][0].minesAround = 2
fieldMatrix[0][0].isFlagged = true
```

Tömb létrehozás

```
object MineSweeperModel {

    lateinit var fieldMatrix: Array<Array<Field>>


    fun initGameArea(size: Int) {
        fieldMatrix = Array(size){ Array(size) {Field(0, 0, false, false)} }
    }

}
```

Hasznos olvasmányok

- Custom Views and Drawing:
  - https://developer.android.com/training/custom-views/create-view.html
  - https://developer.android.com/training/custom-views/custom-drawing.html
  - https://developer.android.com/training/custom-views/making-interactive.html
  - https://developer.android.com/training/custom-views/optimizing-view.html
- Toggle button:
  - https://developer.android.com/guide/topics/ui/controls/togglebutton.html
- Toast:
  - https://developer.android.com/guide/topics/ui/notifiers/toasts.html
- Snackbar:
  - https://developer.android.com/training/snackbar/action.html

További hasznos linkek

- Awsome Android UI
  - https://github.com/wasabeef/awesome-android-ui
- Design patterns (what is besides Singleton?)
  - https://github.com/iluwatar/java-design-patterns
- Best Android development resources:
  - http://www.anysoftwaretools.com/best-android-development-resources
- Android library collection:
  - https://android-arsenal.com/

Összefoglalás

- Activity komponens
- Activity életciklus
- Több Activity-s alkalmazások
- Állapot megőrzés az életciklus során
- Menük kezelése
- Felhasználói felület tervezése, egyedi nézetek, rajzolás

Köszönöm a figyelmet!

peter.ekler@aut.bme.hu