

# Android alkalmazásfejlesztés

Szálkezelés  
Rajzolás képernyőre  
Bevitel kezelése  
- Érintő képernyő  
- Billentyűzet

OE-NIK

2012. szeptember 25.

**Sicz-Mesziár János**

sicz-mesziar.janos@  
nik.uni-obuda.hu



# Szálkezelés

- ⊙ A fő szál az UI szál, ami automatikusan létrejön.
- ⊙ Egyes folyamatok lassúak és bizonytalanok. pl.:

- hálózati forgalom
- adatbázis lekérések

→ **UI szálban a folyamatok fennakadnak!**

Ha kb. 5mp-ig nem válaszol: „application not responding” dialógus megjelenik.



- ⊙ Legyen egy másik szál: background (worker) thread
- ⊙ Például:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork();
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

**De ez sem jó! Cross-thread probléma:  
háttérszál manipulálja az UI szálát.  
Nem „thread-safe” megoldás!**

# Szálkezelés a gyakorlatban (2)

- © Küldjünk értesítést az UI elemnek. Így amint biztonságos állapotba kerül az UI szál lefuttatja a grafikai felületet érintő módosításokat.

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap b = loadImageFromNetwork();
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(b);
                }
            });
        }
    }).start();
}
```

- © **Vagy:** `class MyThread extends Thread{ ... }`
- © Alternatívák: [postDelayed](#), [Activity.runOnUiThread](#), [Handler](#), [AsyncTask](#)

# Rajzolás a képernyőre

- ⊙ A klasszikus 2D rajzolás Canvas-en történik.
- ⊙ A Canvas eredhet például:
  - Bitmap-ből, vagy
  - egy View leszármazott onDraw() implementálásából
- ⊙ Erre a célra kialakított felület: SurfaceView

## ⊙ Példa:

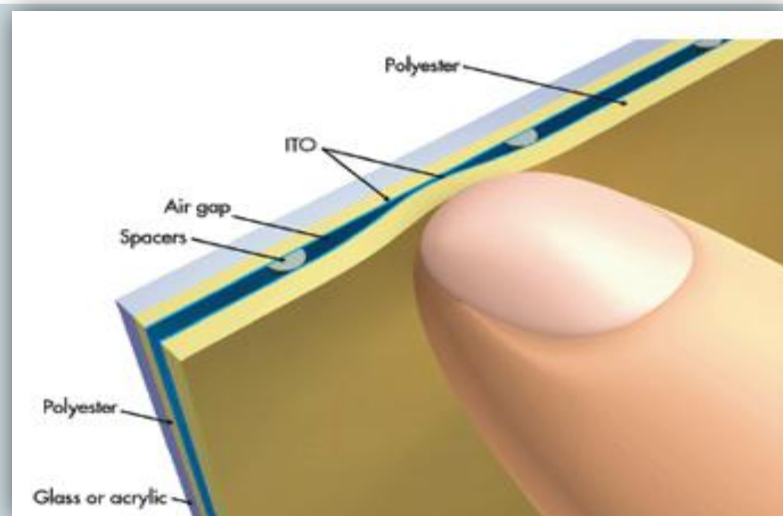
```
public class Rajzpapir extends View{
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.save();
        Paint p = new Paint();
        p.setColor(Color.RED);
        canvas.drawCircle(0, 0, 50, p);
        canvas.restore();
    }
}
```

- ⊙ OpenGL ES: hardveres gyorsítás, 2D és 3D grafika.

# Érintő képernyő (touchscreen)

Két technikai megoldás jellemző a piacon:

## Rezisztív



Ellenállás mechanikai megváltozása.  
(lassabb)

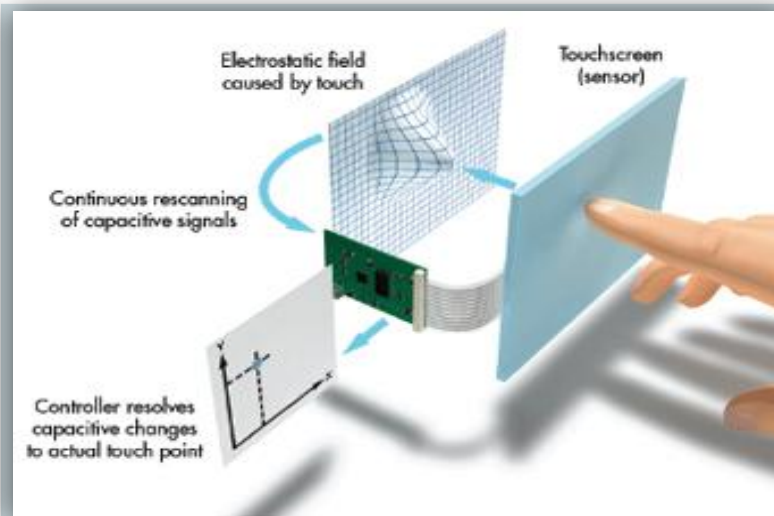
Meghatározható az érintés időtartama,  
erőssége, irányvektora.

Ütésre érzékeny, elektromos hatásra  
érzéketlen.

Forrás, nagyon jó leírással:

<http://www.seria.hu/cikkeink/Rezisztiv%20vs%20kapacitiv/03%20resz/Rezisztiv%20vs%20kapacitiv%203.html>

## Kapacitív



Referencia töltéstől való eltérés.  
(gyorsabb)

Érintések száma, terület nagysága,  
időtartama, irányvektora,

Elektromos hatásra érzékeny,  
páratartalomra, hőre kevésbé.

Forrás:

<http://www.seria.hu/cikkeink/Rezisztiv%20vs%20kapacitiv/04%20resz/Rezisztiv%20vs%20kapacitiv%204.html>

# Touchscreen kezelése

- ⦿ Jó lenne információ az érintésről. Pl.: X, Y koordináta
- ⦿ Adott UI elem érintésekor az egész kijelző felület a mienk → **Nem csak az adott UI elem méretére!**
- ⦿ 3 tipikus állapotról v. eseményről beszélhetünk:
  - DOWN : rátesszük az ujjunkat, azaz első érintés
  - MOVE : folyamatos mozgatás
  - UP : felemeljük az ujjunkat

## ⦿ Megvalósítás:

```
main.setOnTouchListener(new View.OnTouchListener() {  
    public boolean onTouch(View v, MotionEvent e) {  
        if(e.getAction() == MotionEvent.ACTION_MOVE) {  
            float x = e.getX();  
            float y = e.getY();  
        }  
    }  
});
```

# Több ujjas érintés (multitouch)

⦿ Android API elvben 250 ujjat képes kezelni egyszerre.

⦿ De limitált => gyártófüggő. Pl.:

- HTC Desire → 2 ujj
- Samsung Galaxy S → 5 ujj
- HTC Evo → 5 ujj

☺ Emulátor → 1 ujj

Tablet → ? ujj

⦿ Kód szinten:

```
for(byte i=0; i<event.getPointerCount(); i++){  
    float x = event.getX(i);  
    float y = event.getY(i);  
    // int id = event.getPointerId(i);  
    // int ind = event.findPointerIndex(pointerId);  
}
```

⦿ Bővebb leírás:

<http://android-developers.blogspot.com/2010/06/making-sense-of-multitouch.html>

# Gesztusok kezelése

- ◎ GestureDetector:

Egy ujjas gesztusok kezelése, mint: scrolling, flinging, long press

- ◎ ScaleGestureDetector: **(Android 2.2 óta!)**

Két ujjas gesztusok kezelése, mint: pinch zooming

## 1. Megvalósítunk egy OnGestureListener-t:

```
OnGestureListener gestListener = new  
    OnGestureListener() { ... onFling() ... onScroll() ... }
```

## 2. GestureDetector példányosít:

```
GestureDetector gd = new GestureDetector(context,  
    gestListener);
```

## 3. onTouchEvent() esemény átadása:

```
public boolean onTouch(View v, MotionEvent event) {  
    gd.onTouchEvent(event);  
}
```



# Billentyűzet és gombok

- ⊙ Hardveres és szoftveres billentyűzet is lehet:  
onKeyDown(), onKeyUp(), onKeyLongPress(), ...
- ⊙ D-pad (direction-pad) : 4-5 irányú gomb, kezelése szintén a fenti metódussokkal.



Ha onTrackballEvent() metódust nem valósítjuk meg, akkor az Android átfordítja D-pad eseményre!

- ⊙ Speciális gombok is felüldefiniálhatóak:

Hangerő gomb, Menü gomb, Vissza gomb

- ⊙ Gyakorlatban:

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    switch(keyCode) {
        case KeyEvent.KEYCODE_DPAD_CENTER:
            if(event.getAction() == KeyEvent.ACTION_DOWN) {
                Log.d("NIK", "D-PAD középső enter gomb.");
                return true;
            }
        default: return super.onKeyDown(keyCode, event);
    }
}
```

