

## iOS alapú szoftverfejlesztés - Labor 04

A laborsegédletet összeállította: Kelényi Imre - [imre.kelenyi@aut.bme.hu](mailto:imre.kelenyi@aut.bme.hu)

A labor témája:

- Nézetek példányosítása kódból
- Egyedi nézet osztály létrehozása kódból
- Rajzolás
- Gesztusfelismerők

A labor célja egyszerű "rajzprogram" megírása.

A laborhoz tartozó nagyobb kódrészletek a következő url-en érhetők el copy-paste barát formában:

```
https://gist.github.com/imrekel/1489f8bf479b6c1f6a66  
https://goo.gl/lqZ0RB
```

## 1. iPaint

### 1.1. Egyedi nézet osztály és rajzolás

Hozzunk létre egy új "Single View" application-t, "iPaint névvel" iPhone-ra!

Hozzunk létre egy új, UIView-ből leszármazó osztályt **EllipseView** névvel (a fájl létrehozásakor használhatjuk a Cocoa Touch templatet).

Definiáljuk felül a drawRect() metódust és rajzoljunk ki egy kék ellipszist, mely teljesen kitölti a nézet területét (4 pontnyi "peremet" hagyva neki):

```
class EllipseView: UIView {  
  
    override func drawRect(rect: CGRect) {  
        let context = UIGraphicsGetCurrentContext()  
  
        let ellipseRect = CGRect(x: 4, y: 4,  
                                width: self.bounds.width - 8, height: self.bounds.height - 8)  
        UIColor.blueColor().setFill()  
        CGContextFillEllipseInRect(context, ellipseRect)  
    }  
}
```

*A rajzoláskor a Core Graphics framework műveleteit használjuk. A rajzolás mindig egy grafikus contextus segítségével történik (CGContext). Mivel a Core Graphics C-ben lett megírva, az API hívások függvényeken keresztül történnek (nem objektum orientált).*

Próbáljuk ki az új ellipszis osztályunkat! A ViewController osztály viewDidLoad() metódusában hozzunk létre egy új ellipszist és adjuk hozzá a view controller gyökérnézetéhez:

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    let ellipse = EllipseView()  
    ellipse.frame = CGRect(x:100, y:100, width: 60, height: 60)  
    ellipse.opaque = false  
    view.addSubview(ellipse)  
}
```

*Ha kódból hozunk létre nézeteket a következő három lépésre figyeljünk:*

- 1. A nézet példányosítása*
- 2. A nézet pozíciójának és méretének megadása (frame, esetleg bounds + center beállításával) a leendő szülőnézetének koordinátarendszerében*

### *3. A nézet hozzáadása egy szülőnézethez (itt kerül be a nézet hierarchiába)*

*Ha bármelyik lépést kihagyjuk, a nézet nem fog megjelenni!*

*Az **opaque** property azt szabályozza, hogy a nézet "alatti" területet is ki kell-e rajzolni a rendszernek. **opaque = true** esetén a nézetnek teljesen ki kell töltenie a keretét, mert ellenkező esetben a "lyukaknál" nem fognak látszódni az alatta levő egyéb nézetek vagy a szülőnézete.*

Kicsit tuningoljuk fel EllipseView rajzoló kódját és adjunk hozzá egy körvonalat:

```
let strokeColor = UIColor(red: 0.5, green: 0.5, blue: 1.0, alpha: 1.0)
strokeColor.setStroke()
CGContextSetLineWidth(context, 8.0)
CGContextStrokeEllipseInRect(context, ellipseRect)
```

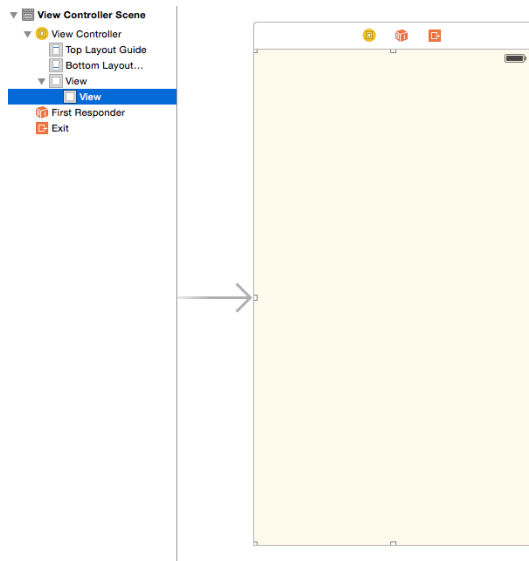


## 1.2. Egyedi nézet osztályok használata Interface Builderben

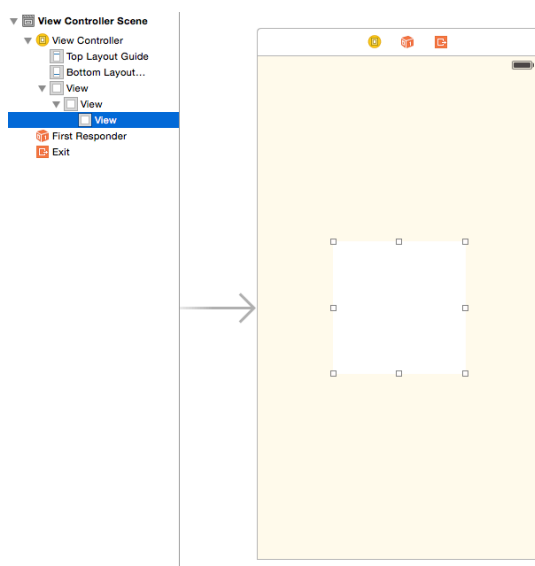
Nyissuk meg a **Main.storyboard**-ot és a File Inspectorban kapcsoljuk ki a "Use Size Class" opciót.

*Ezen a laboron is csak abszolút koordinátákkal és méretekkel dolgozunk vagyis csak azon a kijelzőméreten fog helyesen megjelenni a felhasználói felület, amire megszerkesztettük. Éppen ezért érdemes a teszteléshez iPhone 5 szimulátort használni (ennek a méretére szerkesztjük a felületet a storyboardban).*

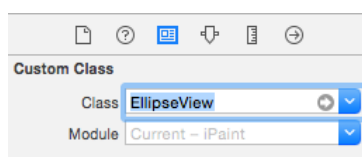
Adjunk hozzá egy üres View-t a gyökérnézethez. Ez lesz a "rajzlapunk", ehhez a nézethez fogjuk majd hozzáadni a rajzolás használt gyereknézeteket. Érdemes a rajzlap-nézet háttérszínét is megváltoztatni valami világos színre, hogy jobban elüssön a háttérétől:



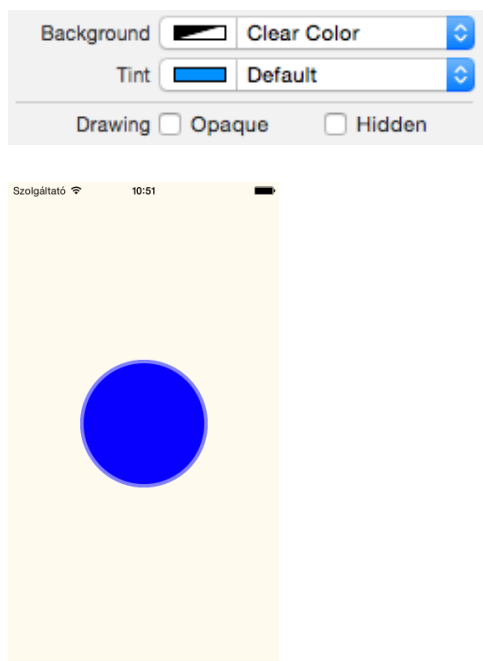
Most próbáljuk ki, hogy hogyan tudunk felvenni egy példányt az egyedi EllipseView nézet osztályunkból! Ehhez ismét csak egy sima View-t kell először felvennünk a rajzlap-nézet gyerekeként:



Majd ezek után át kell állítanunk a felvett nézet osztályát EllipseView-ra. Ehhez válasszuk ki a nézetet, majd az Identity Inspector-ban írjuk át a Class attribútumot:



Az Attribute Inspectorban állítsuk a **Background** attribútumot Clear Color-ra és kapcsoljuk ki az **Opaque**-t:



*Egyedi nézetosztályok használatánál mindig a nézet egyik ősztyájának megfelelő elemet kell behúznunk az Interface Builderben. Ez legáltalánosabb esetben a View (UIView), de ha például egy UITextFieldből leszármazott egyedi osztály egy példányát akarjuk létrehozni, akkor már egy Text Field elemet érdemes választani és ennek átállítani a Class beállítását az egyedi osztályra.*

### 1.3. Interface Builder támogatás egyedi osztályokhoz

Xcode 6 óta lehetőség van az Interface Builderrel is kirajzoltatni az egyedi osztályokat. Ehhez mindössze annyit kell tennünk, hogy az osztály forráskódjába, közvetlenül az osztály deklarációja elé felvesszük az **@IBDesignable** attribútumot:

```
@IBDesignable  
class EllipseView: UIView {
```

Ha megnézzük a storyboardot, meg fog jelenni a nézetünk.

Vegyünk fel egy új property-t EllipseView-hoz, ami a kitöltés színét adja meg és írjuk át a rajzoló kódot, hogy ez alapján színezzon:

```
@IBDesignable
class EllipseView: UIView {
    var color: UIColor = UIColor.blueColor()

    override func drawRect(rect: CGRect) {
        var context = UIGraphicsGetCurrentContext()

        let ellipseRect = CGRect(x: 4, y: 4,
                                width: bounds.width - 8, height: bounds.height - 8)
        color.setFill()
        CGContextFillEllipseInRect(context, ellipseRect)
    }
}
```

Ha szeretnénk, hogy körvonal színe is menjen a kiválasztott kitöltőszínhez, `strokeColor`-nak állítsuk be a kiválasztott színt egy kisebb alpha komponenssel:

```
let strokeColor = color.colorWithAlphaComponent(0.4)
```

A színválasztás azonban így még csak kódból elérhető. Szerencsére ezen is segíthetünk, prefixáljuk a `color` property-t **@IBInspectable** attribútummal:

```
@IBInspectable var color: UIColor = UIColor.blueColor()
```

Az **@IBInspectable**-ként megjelölt property-k az Interface Builderben megjelennek az Attribute Inspectorban mint az adott osztály szerkeszthető beállításai:



*A type inference valamiért nem mindig működik jól az @IBInspectable property-knél, ezért ezeknél mindig explicit adjuk meg a property típusát! (pl. `var color: UIColor = ...`)*

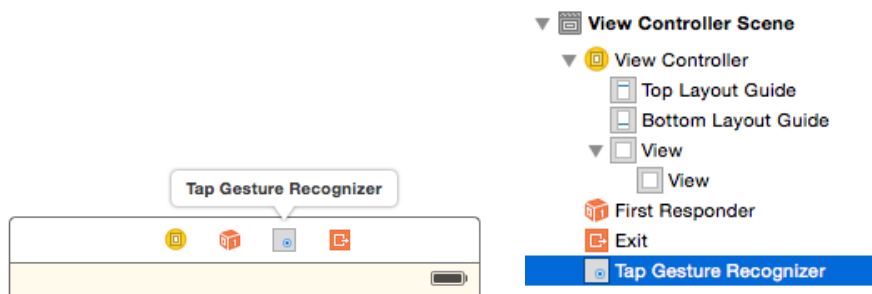
## 1.4. Gesztusfelismerés és rajzolás

Töröljük ki a storyboardba felvett nézetet és a `viewDidLoad()`-ban a példa `EllipseView`-t létrehozó kódot.

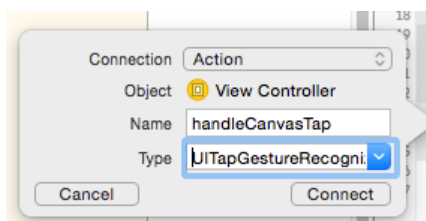
Vegyünk fel egy `canvas` nevű outletet a "rajzlapunkhoz":

```
@IBOutlet weak var canvas: UIView!
```

A storyboardban adjunk hozzá egy Tap gesztusfelismerőt `canvas`-hez. Ehhez drag-and-drop húzzunk rá egy Tap Gesture Recognizer elemet, mely a művelet elvégzése után megjelenik a view controller szerkesztő nézet felső részén található sávban, illetve a Document Outline-ban:



Az akció metódusok bekötésénél megismert módon, "Ctrl-klikk-drag-and-drop" húzzuk be a gesztusfelismerő akciómetódusát a kódba. A megjelenő popuban váltsunk Action-re, adjuk neki a `handleCanvasTap()` nevet és módosítsuk a metódus egyetlen paraméterének `UITapGestureRecognizer`-re:



A metódusban hozzunk létre egy új `EllipseView`-t a "koppintás" pozíciójában és adjuk hozzá a canvas nézethez:

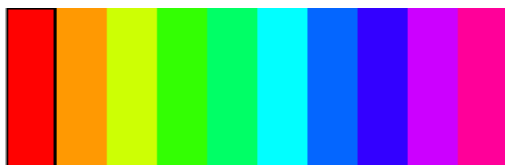
```
@IBAction func handleCanvasTap(sender: UITapGestureRecognizer) {
    let tapPoint = sender.locationInView(canvas)

    let ellipse = EllipseView()
    ellipse.bounds = CGRect(x:0, y:0, width: 60, height: 60)
    ellipse.center = tapPoint
    ellipse.opaque = false
    canvas.addSubview(ellipse)
}
```

Ezzel neki is állhatunk felhőket rajzolni!

## 1.5. Színválasztó nézet

Egy színválasztó nézetet fogunk írni, ami végül valahogy így fog festeni:



Hozzunk létre egy új osztályt **ColorPicker** néven, őssztályul válasszuk `UIView`-t (érdeemes használni a Cocoa Touch Class template-t).

Definiáljuk az osztályt rögtön `IBDesignable`-nek. Vegyünk fel egy új property-t, mely eltárolja, hogy hány színből választhat majd a felhasználó. A propertyt

tegyük IB-ből szerkeszthetővé az @IBInspectable attribútummal. CGFloat típust használunk, mert ez később jelentősen leegyszerűsíti a számműveleteket (kevésbé kell majd cast-olni):

```
@IBDesignable
class ColorPicker: UIView {

    @IBInspectable var colorCount: CGFloat = 10
}
```

Vegyünk még fel két property-t. Az egyik a kiválasztott szín sorszámát (a választható színekből hanyadik van épp kiválasztva), a másik pedig magát a színt fogja tartalmazni:

```
private var selectedColorIndex = 0

var selectedColor =
    UIColor(hue: 0.0, saturation: 1.0, brightness: 1.0, alpha: 1.0)
```

Vegyünk fel egy computed property-t, ami a nézet aktuális méretének függvényében megadja, hogy egy színhez "milyen széles" téglalap tartozik:

```
var colorWidth: CGFloat {
    return bounds.width / CGFloat(colorCount)
}
```

Definiáljuk felül az őssosztályból örökölt drawRect() metódust:

```
override func drawRect(rect: CGRect) {
    super.drawRect(rect)

    let context = UIGraphicsGetCurrentContext()

    for var i:CGFloat = 0; i<colorCount; i++ {
        let color = UIColor(hue: i*(1.0 / colorCount), saturation: 1.0,
            brightness: 1.0, alpha: 1.0)
        color.setFill()

        CGContextFillRect(context, CGRect(x:colorWidth * i, y:0,
            width: colorWidth, height:bounds.height))

        if Int(i) == selectedColorIndex {
            UIColor.blackColor().setStroke()
            CGContextSetLineWidth(context, 2.0)

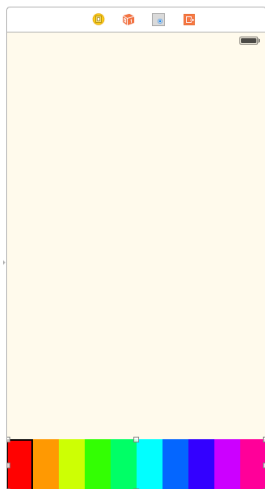
            CGContextStrokeRect(context, CGRect(x:colorWidth * i, y:1,
                width: colorWidth - 1, height:bounds.height - 2))
        }
    }
}
```

<https://gist.github.com/imrekel/1489f8bf479b6c1f6a66#file-colorpicker-drawrect-swift>

Próbáljuk ki a nézetet! Váltunk a storyboardra és a rajzfelület alá helyezzünk el egy új UIView-t (a canvas nézet magasságát előtte csökkentjük le, hogy alatta



elférjen az új nézet). Az új nézet osztályát az Identity Inspectorban állítsuk át ColorPicker-re:



Ha most kipróbáljuk, a ColorPicker még nem fogja kezelni az érintéseket. Ehhez hozzá fogunk adni egy gesztusfelismerőt ColorPicker inicializálójában.

Definiáljuk felül a ColorPicker UIView-ből örökölt két fontos inicializálóját az **init(frame: CGRect)** és **init(coder aDecoder: NSCoder)**-t. Mindkettőből egy közös inicializáló kódot fogunk hívni, amit egy **commonInit()** nevű metódusban fogunk definiálni:

```
override init(frame: CGRect) {  
    super.init(frame: frame)!  
  
    commonInit()  
}  
  
required init(coder aDecoder: NSCoder) {  
    super.init(coder: aDecoder)  
  
    commonInit()  
}
```

*Ha storyboard-ból vagy XIB-ből töltődik be egy nézet, akkor az **init(coder aDecoder:NSCoder)** hívódik meg.*

*Ha kódból hozzuk létre a nézetet, akkor az **init(frame: CGRect)**-et használjuk.*

*Ha egyedi nézetet készítünk, akkor célszerű mindkettőt feldefiniálni. Az NSCoder paraméterű inicializáló **required**, vagyis kötelező minden leszármazott osztályban felüldefiniálni (pont azért, hogy IB-ben használva az osztályt, helyes működést kapjunk).*

**commonInit()**-ben hozzunk létre a kódból egy Tap gesztus felismerőt és rendeljük hozzá a **handleTap()** akció metódust:

```
func commonInit() {  
    let tapRecognizer = UITapGestureRecognizer(target: self,  
        action: "handleTap:")  
    addGestureRecognizer(tapRecognizer)  
}
```

Mikor kódból hozzárendelünk egy akció metódust egy elemhez (pl. itt a gesztusfelismerőhöz a `handleTap()` nevű metódust), a **target-action** mintát használjuk. Ez Objective-C-ből ered, ahol a metódusok nevére egy speciális típussal, az úgynevezett **selector**-ral hivatkozhatunk. Ez Swift-ben ilyen formában nem létezik, ezért a target-action mintát használó API-knál string-ként kell hivatkozni a meghívandó akciómetódusok nevére. Itt is ez történik a "handleTap:" megadásával.

A kettőspont a metódus nevének végén "handleTap:" arra utal, hogy a metódusnak egy paramétere van. Kettőspont nélkül (pl. "handleTap") egy olyan metódusra hivatkoznánk aminek nincsenek paraméterei.

Valósítsuk meg a `handleTap()` metódust, mely az érintéspont függvényében beállítja a kiválasztott színt és újrarajzoltatja a nézetet:

```
func handleTap(gestureRecognizer: UITapGestureRecognizer) {  
    let tapPoint = gestureRecognizer.locationInView(self)  
    selectedColorIndex = Int(tapPoint.x / colorWidth)  
    selectedColor = UIColor(  
        hue: CGFloat(selectedColorIndex) * (1.0 / colorCount),  
        saturation: 1.0, brightness: 1.0, alpha: 1.0)  
  
    setNeedsDisplay()  
}
```

<https://gist.github.com/imrekel/1489f8bf479b6c1f6a66#file-colorpicker-handletap-swift>

Ha kipróbáljuk az alkalmazást, színválasztón már váltakozni fog a kiválasztott szín.

Váltsunk át a storyboard-ra és vegyünk fel egy outletet a színválasztóhoz `colorPicker` névvel:

```
@IBOutlet weak var colorPicker: ColorPicker!
```

Állítsuk be a kiválasztott színt az ellipszis nézeteket létrehozó akció metódus kódjában:

```
let ellipse = EllipseView()  
ellipse.color = colorPicker.selectedColor
```

## 1.6. Legutóbb rajzolt kör átméretezése

ViewController-be vegyünk fel egy új property-t, melyben eltároljuk a legutóbb létrehozott nézetet:

```
var currentEllipse: EllipseView?
```

Az rajzoló nézetek létrehozásakor állítsuk be a property értékét:

```
currentEllipse = ellipse
```

Nyissuk meg a storyboardot és az Object Library-ból húzzunk be egy Pinch gesztus felismerőt a canvas-hez (ügyeljünk rá, hogy nehogy a gyökér nézethez adjuk hozzá).

Kössünk be egy akciómetódust a gesztusfelismerőre, **handlePinch()** névvel:

```
@IBAction func handleCanvasPinch(recognizer: UIPinchGestureRecognizer) {  
    currentEllipse?.bounds.size = CGSize(width: 60 * recognizer.scale,  
        height: 60 * recognizer.scale)  
}
```

A pinch gesztust a szimulátorban az Option (Alt) billentyű nyomva tartásával tudjuk szimulálni (kapunk egy virtuális ujjpárt).

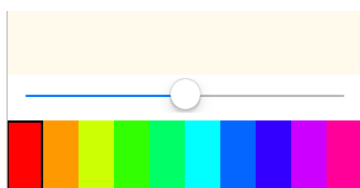
Láthatjuk, hogy nagyításkor egy "maszatos" átskálázott képet kapunk. Ez azért van, mert nem rajzolódik ki újra a nézet, csak a becachelt tartalmát (textúrát) nagyítja fel a rendszer. Ezen segíthetünk a **contentMode** property **.Redraw** értékre állításával, ami azt idézi elő, hogy a nézet minden méretváltozásakor újra ki fogja rajzoltatni a tartalmát. Gondoskodjunk róla, hogy az EllipseView példányosításakor beállítjuk:

```
ellipse.contentMode = .Redraw
```

## 2. Önálló feladatok

### 2.1. Ecsetméret állítása

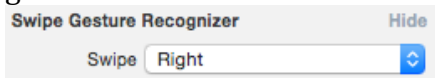
Adjunk hozzá egy Slider-t (UISlider) a "canvas" alá, amivel a kirajzolt körök kezdeti méretét tudja állítani a felhasználó!



- A Slider attribútumai között beállíthatjuk a minimális, maximális és kezdeti értékét (pl. 10-100 és 10-el indul)
- Vegyünk fel egy Outlet property-t a Slider-hez (pl. radiusSlider)
- A Slider értékét a "value" property-jével tudjuk kiolvasni. Ez alapján módosítsuk az EllipseView-k létrehozásánál a méretüket (bounds property)

## 2.2. Swipe to delete

Adjunk hozzá egy Swipe gesztusfelismerőt a canvas-hez és a gesztus bekövetkeztekor töröljük a canvas tartalmát, az összes alnézet eltávolításával!

- A Swipe gesztus felismerőnél meg kell adnunk, hogy milyen irányú "swipe"-ot detektáljon (pl. balról-jobbra). Ennek megfelelően, ha tetszőleges irányú swipe-ot észlelni szeretnénk, akkor 4 külön gesztusfelismerőt kellene létrehozni
- 
- Nézetek eltávolításához a removeFromSuperview() metódust kell meghívni az eltávolítandó gyerek nézeteken. Ehhez végig kell iterálni canvas subViews propertyjén (ami a gyerek nézeteket tartalmazó tömb). Sajnos subViews típusa [AnyObject] (valószínűleg ezt majd javítják egy későbbi Xcode verzióban), ezért a nézeteket át kell castolni UIView-ra:

```
for view in canvas.subviews {  
    let view = view as UIView  
    view.removeFromSuperview()  
}
```