

iOS alapú szoftverfejlesztés

Dr. Blázovics László

Blazovics.Laszlo@aut.bme.hu

2021. Szeptember 21.



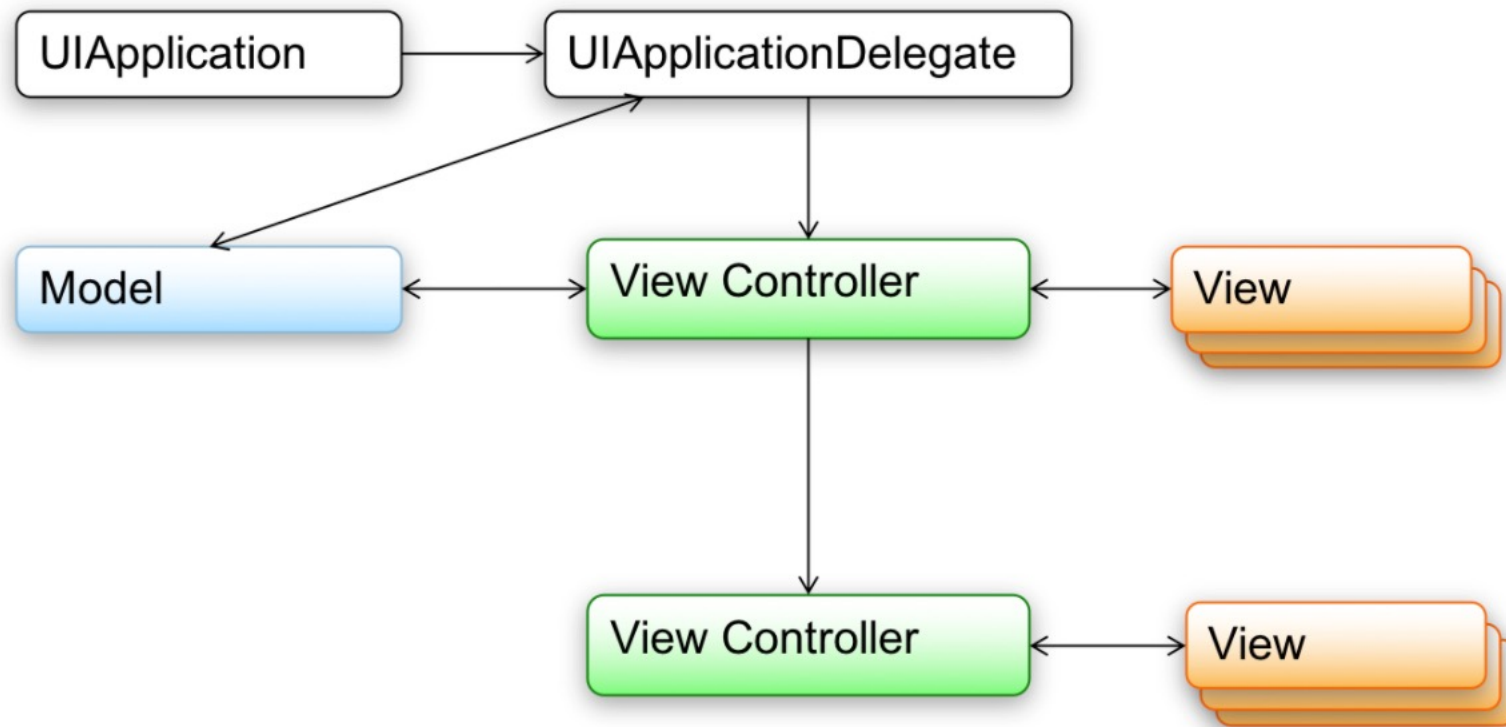
Automatizálási és
Alkalmazott
Informatikai Tanszék

Labor – QB115

- Laborbeosztás a Github-on
 - > L1 - 12:15-13:45
Blázovics László
 - > L2 14:15-15:45
Gavrillás Kristóf
- Aki akar, hozhat saját gépet -> Lehetőleg oda üljön, ahol nincs gép
- Segédletek szintén a ~~GitHub-on~~ Teams Assignment-ben
- Beadás is Teams Assignment-en keresztül

iOS alkalmazások felépítése, a felhasználói felület alapjai

Egy iOS alkalmazás elemei (UIKit)



UIApplication, AppDelegate, SceneDelegate

- **UIApplication:** minden app-hoz egyetlen példány tartozik, rendszer hozza létre
- **Application Delegate:** az alkalmazás „gyökér” osztálya (UIApplicationDelegate protokollt valósítja meg), amelyben:
 - > Reagálás alkalmazás életciklus eseményeire (Pl.: app elindult)
 - > Alkalmazáshoz beérkező külső események kezelése (Pl.: Push Notification fogadása)
- **Scene Delegate:** Feladata a WindowScene és a Window-k életciklusának a kezelése
 - > Reagálás alkalmazás életciklus eseményeire (Pl.: scene megjelent)

SceneDelegate vs AppDelegate

- Az alkalmazás életciklusát érintő változásokról iOS 12-ig az AppDelegate metódusain keresztül lehetett értesülni
- iOS 13-tól lehetőség van több-ablakos alkalmazások fejlesztésére
 - > Elsősorban iPad (Split View), AirPlay, CarPlay támogatására
 - > Az ablakok külön „életet élnek” -> SceneDelegate
- Az AppDelegate helyett használjuk a SceneDelegate-et az egyes scene-ek állapotváltozásainak kezeléséhez
 - > Az AppDelegate ugyanilyen metódusai is megmaradnak, de azok nem fognak meghívódni, ha a SceneDelegate is megvalósítja azokat
 - > iOS 13 előtti verziók támogatására az AppDelegate-et kell használni, de az új projektek már SceneDelegate-tel generálódnak

Model

- Model objektumok: az alkalmazás üzleti logikája és a hozzá tartozó adatok
- Nincs külön *model típus*
- Bármilyen osztály, struktúra vagy enum lehet a model része, ami az üzleti logikához kapcsolódik
 - A perzisztens tárból betöltött dolgok (adatbázis entitás osztályok, fájlokból betöltött osztályok, stb.)

View

- Minden, ami közvetlenül a képernyőre rajzol és érzékeli a felhasználói interakciót
 - > View elkapja a felhasználói interakciót (érintés), de reagálni nem fog, hanem jelzi a tényt a ViewController felé
- Alap nézet (ős)osztály: **UIView**
 - > Bármilyen, ami közvetlenül rajzol a képernyőre
 - > Interakció a felhasználóval: vezérlőelemek (UIControl)
 - > Egymásba ágyazhatók: egy UIView tartalmazhat gyerek UIView-kat
- Nézeteket kódból is felépíthetjük, de legtöbbször az Interface Buildert (IB) használjuk

UIWindow

- Az ablak: **UIWindow** egy speciális nézet (ősszánya a **UIView**)
- Minden iOS alkalmazás rendelkezik legalább egy ablakkal
- Az esetek túlnyomó többségében az iPhone alkalmazások csak eggyel (ha rádugunk egy külső monitort vagy CarPlay-t használunk, akkor lesz két ablak)
- Létrehozhatjuk az AppDelegate-ben kódból, vagy Storyboard használata esetén automatikusan létrejön
- iOS-en és iPadOS-en az ablak csak egy üres tároló (nem tartozik hozzá keret, menüsor, stb.)
- Az alkalmazások ablaka alapesetben a teljes kijelzőt kitölti
- A képernyő tetején lévő Status Bar egy külön ablak, ami rálóg az alkalmazások ablakára

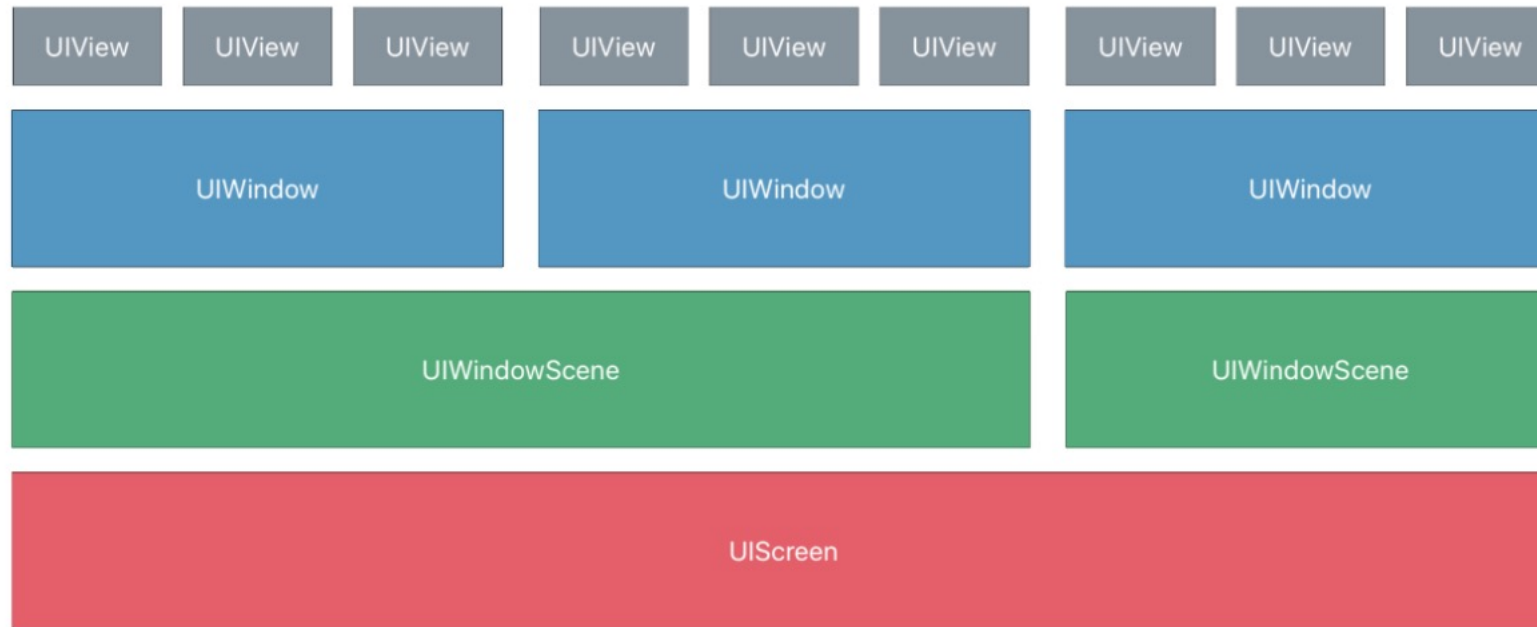
UIWindow II.

- Az ablak az egész alkalmazás gyökér nézete, ehhez alnézetként hozzáadva tudjuk megjeleníteni a saját nézeteinket

```
window.addSubview(view)
```

- > Az alkalmazás "képernyőinek" váltása az ablak gyerek nézeteinek cserélgetésével történik
 - > Ezt általában a rendszer végzi helyettünk
- Az alkalmazás indulásakor a Main.storyboard kezdeti View Controllerének tartalomnézete hozzáadódik az ablakhoz, mint gyerek nézet

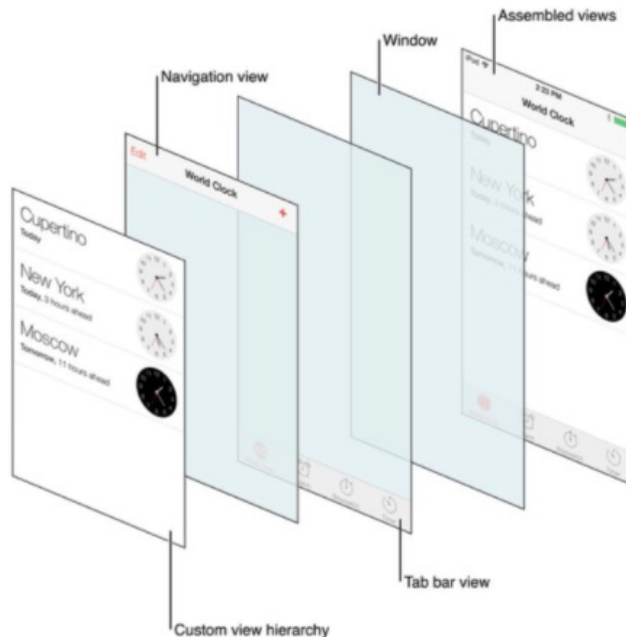
Többképernyős alkalmazások



Nézetek tulajdonságai

Nézet hierarchia

- A nézeteket egymásba ágyazhatjuk: egy nézetnek tetszőleges számú gyerek nézete lehet
 - > Szülő-gyerek viszony
- Az alkalmazásunk egy képernyője általában egy szülő nézet, sok gyerek nézettel



Nézet hierarchia

- A nézetek fa struktúrába rendeződnek (szülő-gyerek hierarchia)
- Alnézet hozzáadása (szülő View-n):

```
func addSubview(_ view: UIView)
```

- Alnézet levétele a szülőről (alnézeten):

```
func removeFromSuperview()
```

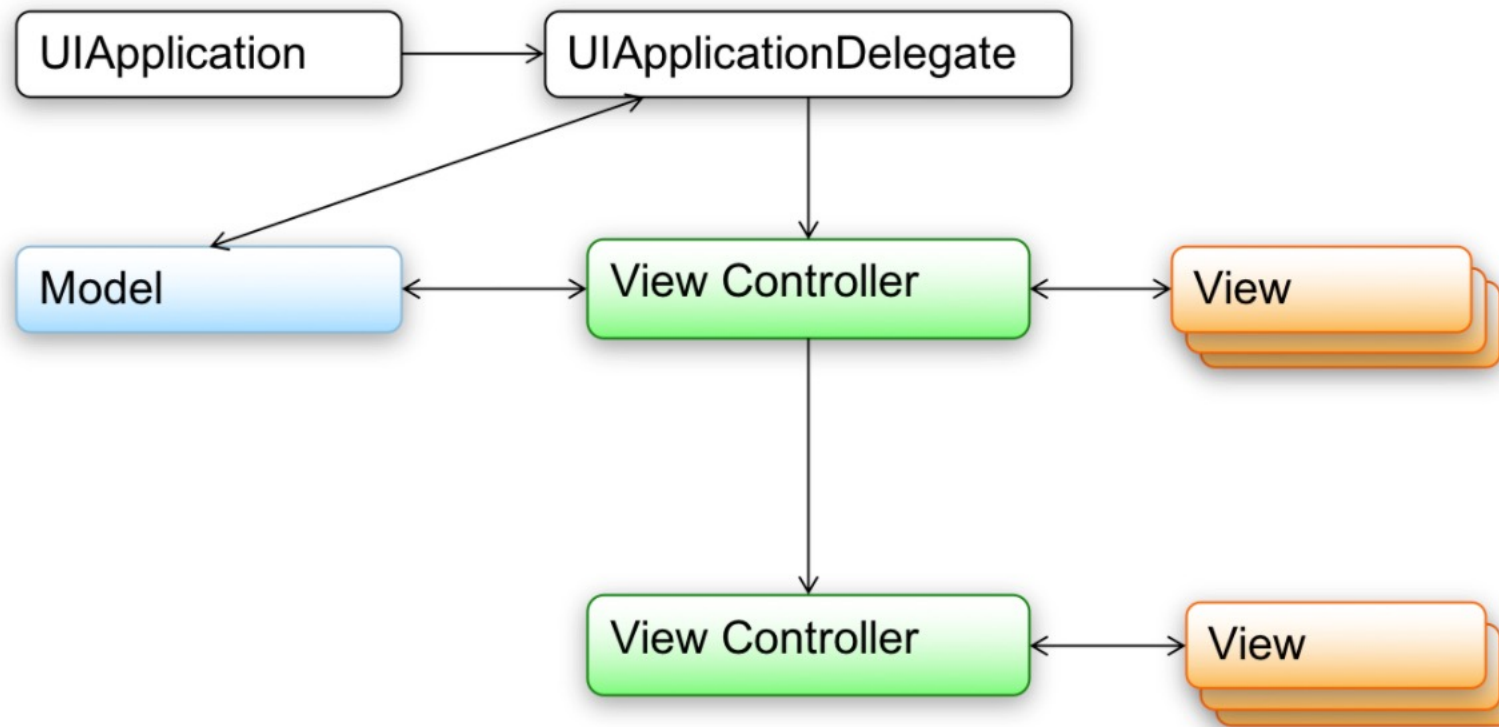
- Szülő nézet lekérdezése:

```
var superview: UIView? { get }
```

- Alnézetek lekérdezése:

```
var subviews: [UIView] { get }
```

Egy iOS alkalmazás elemei (UIKit)



ViewController

- Egy összetartozó nézet hierarchiát felügyel
 - > Maga a kontroller sosem rajzol a képernyőre, az mindig a nézet feladata
 - > Létrehozza és konfigurálja a nézeteket
 - > Feldolgozza a nézetek eseményeit
- Minden *ViewController* őssosztálya a **UIViewController**
- A View Controllerhez tartozó nézethierarchia gyökér nézetét a View Controller **view** propertyjén keresztül érhetjük el
 - > A gyökér nézet alnézeteit (gyerekeit) is ugyanazon View Controller felügyeli

Nézetek létrehozása, *view* property

- UIViewController nem rajzol semmit a képernyőre, azt a hozzárendelt nézetek teszik
- A *view* property-hez rendelt nézet a view hierarchia *gyökér nézete*

> Ez akár kódból is létrehozható, a `loadView()` metódus felüldefiniálásával:

```
override func loadView() {  
    view = UIView() // A gyökér nézet létrehozása  
}
```

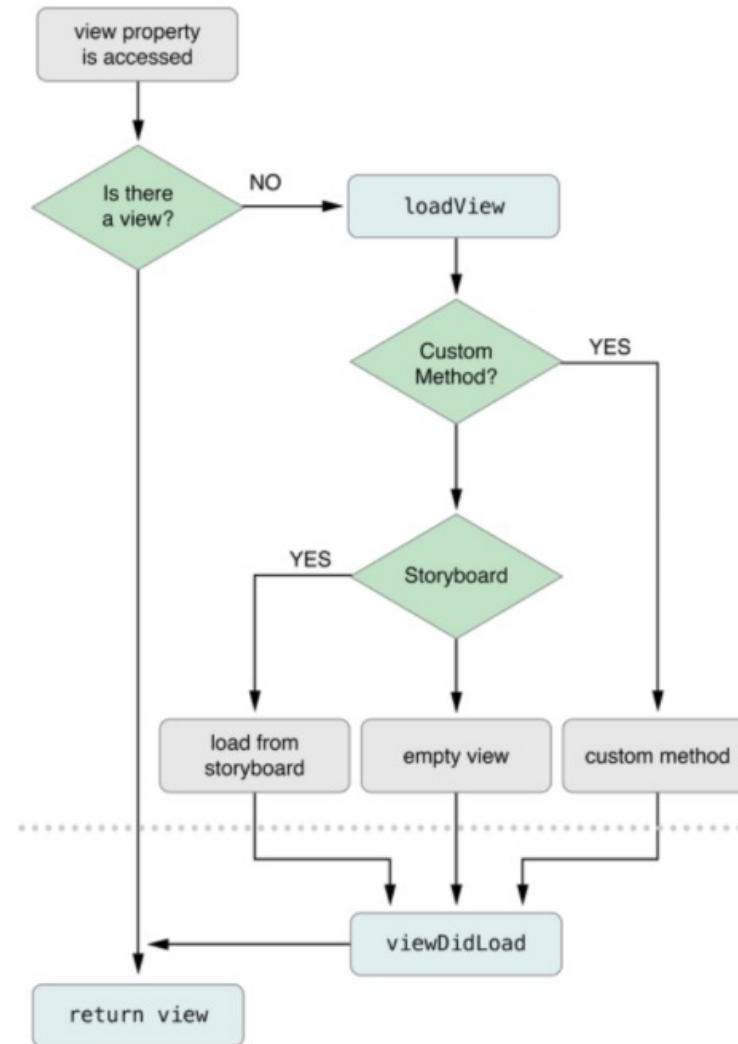
> A gyökér nézethez tetszőleges számú gyerek nézetet rendelhetünk hozzá

- A gyökér nézet és ennek alnézeteinek létrehozása több módon is lehetséges:
 - > Kódból (`loadView` metódus felüldefiniálásával)
 - > Grafikus szerkesztővel (Interface Builder~IB)
 - > Storyboard és XIB fájlokban

Nézet betöltése

- Az alkalmazás valamely része lekérdezi a View Controller view property-jét
- Ha a view még nincs betöltve, meghívódik a `loadView()` metódus
 - > Vagy felüldefiniáljuk és megvalósítjuk kódból a nézet betöltését/felépítését
 - > Vagy nem definiáljuk felül és ilyenkor Storyboard/XIB fájl alapján töltődik be
- Meghívódik a `viewDidLoad()` metódus (itt opcionálisan további inicializálást végezhetünk)
- A már betöltött nézetek addig nem törlődnek, amíg a View Controller létezik (erős referencia)

Nézet betöltése



UIViewController nézetspecifikus értesítései

- A View Controller view property-jének a nézethierarchiába kerülése **előtt** kerül meghívásra (mielőtt a VC megjelenik)

```
func viewWillAppear(_ animated: Bool)
```

- A View Controller view property-jének a nézethierarchiába kerülése **után** kerül meghívásra (miután a VC megjelent)

```
func viewDidAppear(_ animated: Bool)
```

- A View Controller view property-jének a nézethierarchiából való törlése **előtt** kerül meghívásra (mielőtt a VC eltűnne)

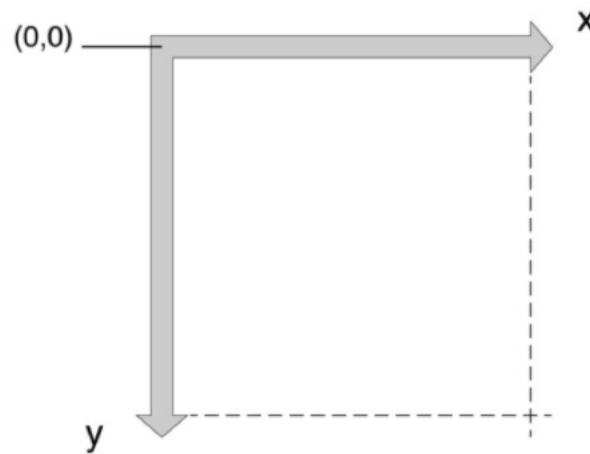
```
func viewWillDisappear(_ animated: Bool)
```

- A View Controller view property-jének a nézethierarchiából való törlése **után** kerül meghívásra (miután a VC eltűnt)

```
func viewDidDisappear(_ animated: Bool)
```

Nézetek koordinátarendszere

- Minden nézet saját logikai koordinátarendszerrel rendelkezik
 - > A (0,0) pont a bal felső sarokban van
- Koordináták és méretek kezelése a **Core Graphics** (prefix: CG) framework típusaival



Core Graphics alaptípusok

- **CGFloat** – egy lebegőpontos szám
- **CGPoint** – struktúra, ami egy pontot reprezentál a 2D koordinátarendszerben (két *CGFloat*: **x** és **y**)

```
var point = CGPoint(x: 3.0, y: 7.0) point.y = 17.0
```

- **CGSize** – struktúra, ami egy szélességet és egy magasságot határoz meg (két *CGFloat*: **width** és **height**)

```
var size = CGSize(width: 5.0, height: 10.0) size.width -= 5.0
```

- **CGRect** – struktúra, ami meghatározza egy négyszög elhelyezkedését és méretét (egy *CGPoint*: **origin** és egy *CGSize*: **size**)

```
var r = CGRect(x: 8.0, y: 8.0, width: 100.0, height: 200.0)
```

Nézetek pozíciója és mérete

var frame: **CGRect**

- A nézet befogó téglalapjának mérete, a **szülő** nézet koordinátarendszerében
- Transzformációk (pl.: elforgatás) után is a szülő nézet koordinátarendszerében lesz értelmezve! (ld. következő dia)

var bounds: **CGRect**

- A nézet által elfoglalt téglalap, a nézet **saját** koordinátarendszerében
- Nézet elforgatása sem befolyásolja

var center: **CGPoint**

- A nézet középpontja, a **szülő** nézet koordinátarendszerében

Frame vs bounds



- center: (160, 300)
- frame: ((10, 100), (200, 300))
- bounds: ((0, 0), (150,250))

Nézetek transzformálása CoreGraphics-szel

- UIView transform property-je tartalmazza a transzformációs mátrixot
 - > Elforgatás (radiánban):
`myView.transform = CGAffineTransform(rotationAngle: CGFloat.pi/180.0 * 30.0)`
 - > Átméretezés:
`myView.transform = CGAffineTransform(scaleX: 0.5, y: 2.0)`
- Általában elég a UIView példányunk következő függvényeit hívni:
 - > `translateBy(x:y:)`
 - > `scaleBy(x:y:)`
 - > `rotate(by:)`

Pontok és pixelek

- A nézetek koordinátarendszerének az egysége a **pont (point)**
- Lebegőpontos értékek, pl.: 0.5 point, 1.3 point
- 1 pont nem mindig egy pixel
 - > Korai display 1 pixel / point (pl.: iPhone 2G, 3G, 3GS, első iPad)
 - > Retina display 2 pixel / point (pl.: iPhone 11, iPad Mini, iPad Air, iPad Pro 10.5”, Apple Watch)
 - > Retina HD / Super Retina 3 pixel / point (pl.: iPhone 13 Plus, X, iPad Pro 12.9”)
- <https://www.ios-resolution.com>
- A View Controller **displayScale** property-je visszatér az aktuális pixel/pont aránnyal **traitCollection.displayScale**

UIView alaptulajdonságok

- Háttérszín

```
var backgroundColor: UIColor? { get set }
```

- Megjelenítés / elrejtés

```
var isHidden: Bool { get set }
```

- Nézet "alatti" területek kirajzolásának kapcsolgatása

```
var isOpaque: Bool { get set }
```

> Ha az isOpaque true (ez az alapértelmezett) gyorsabb lesz a kirajzolás, nem kell a rendszernek foglalkozni az eltakart View-kkal

- Átlátszóság mértéke

```
var alpha: CGFloat { get set }
```

> 0.0 és 1.0 közötti érték

> Ha az isOpaque true, akkor nem számít az értéke

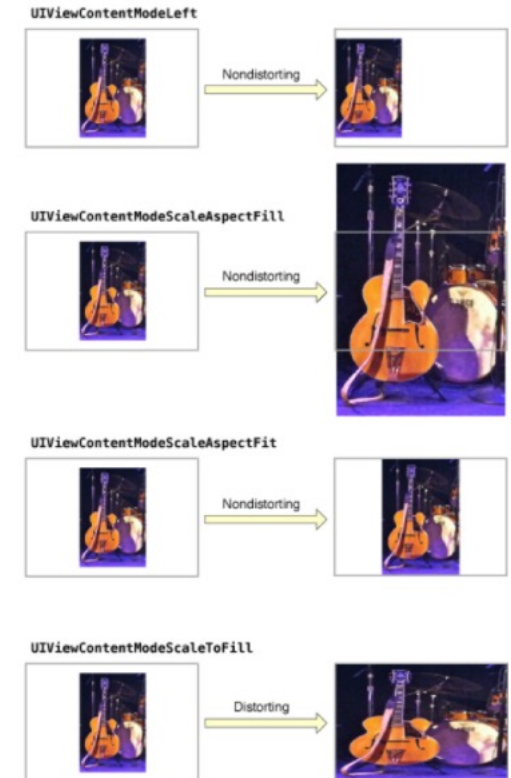
UIView alaptulajdonságok II.

- Kivágás (clipping)
 - > Alapból a nézetek azon része is kirajzolódik a képre, ami kilóg a szülő nézetből
 - > Ugyan kirajzolódik, de érintés eseményeket nem fogad!
`var clipsToBounds: Bool { get set }`
 - > `true` esetén nem rajzolódik ki a nézet azon része, mely a szülő nézetből kilóg
- Felhasználói interakció (érintés, billentyűzet) engedélyezése
`var userInteractionEnabled: Bool { get set }`
 - > Ha `false`, az eseményeket figyelmen kívül hagyja



UIView alaptulajdonságok III.

- A nézetek cache-elik a tartalmukat (egy képben)
 - > Méret- és pozícióváltás nem mindig igényli a nézet újrarajzolását
- A **contentMode** property határozza meg, hogy méret- vagy pozícióváltozáskor hogyan méretezze át a nézet tartalmát
 - > pl.: *UIImageView* a megjelenítendő képet is ez alapján skálázza
 - > **scaleToFill** a default
 - > **redraw** mindig újrarajzoltat!



Színek

- UIKitben a színek megadásához a UIColor osztályt használjuk
 - > Alpha (átlátszóság) komponenst is tartalmaz
- Több különféle inicializáló a színek megadásához (pl.: RGBA, HSBA)

```
let redColor = UIColor(red: 1.0, green: 0, blue: 0, alpha: 1.0)  
//intervallum 0.0 – 1.0
```

- Definiált színek *computed property*-kként

```
let redColor = UIColor.red
```

- Tudunk AssetCatalog-ba is felvenni színeket, erőforrásként, lásd később

Beépített nézetek



Automatizálási és
Alkalmazott
Informatikai Tanszék

UILabel

- Csak olvasható szöveges nézet
- Ha több soros szövegre van szükség:
 - > `numberOfLines`: 0 esetén a rendszer dönti el hány soros lesz
 - > `lineBreakMode`: `byWordWrapping` vagy `byCharacterWrapping` kell
- Szöveg automatikus átméretezése (Adjust to Fit)
 - > Lines -> 0-ra állítani – Ellenkező esetben annyi sorba próbálja beletenni a szöveget, amennyit megadtunk
 - > StoryBoard:
 - Autoshrink beállítása
 - > Kód:
 - `adjustsFontSizeToFitWidth = true`
 - `minimumScaleFactor`

UITextField

- Egysoros szövegbeviteli mező
- Ha többsoros bevitel kell: UITextView
- Szöveges property-k
 - > `text`: a beírt szöveg
 - > `placeholder`: a szöveg, ami megjelenik üres mező esetén
- Plusz nézetek rendelhetők a Text Field bal és jobb oldalához
 - > `clearButtonMode`: megjelenjen-e egy plusz X gomb, ami törli a tartalmat
- Hasznos események
 - > *Value Changed*
 - > *Did End On Exit*: a felhasználó Return-t nyomott a billentyűzeten

UIButton

- Érintések hatására eseményeket generál
 - > Gombnyomás: "Touch Up Inside" esemény
- Típusai:
 - > System
 - > Detail Disclosure
 - > Add Contact
 - > Custom
- Az egyes vezérlőelem állapotokhoz megadható, hogy milyen képet/háttérrel rajzoljon ki

UIControl

- A UIControl azon nézetek őssztálya, melyek felhasználói interakcióra eseményeket generálnak
 - > Ezeket az eseményeket kezeljük le, pl.: akció metódusokat rendelhetünk hozzájuk
- A UIControl definiálja az eseménytípusokat, melyeket a vezérlőelemek kibocsáthatnak, pl.:
 - `UIControlEvents.valueChanged`
 - `UIControlEvents.touchUpInside`
- Példák UIControl leszármazott osztályokra:
 - > `UIButton`
 - > `UISlider`
 - > `UITextField`

UIControlState

- Minden UIControlhoz tartozik egy **state property**

```
var state: UIControl.State { get }
```

- Konstansként vannak definiálva az értékei

```
static var normal: UIControl.State { get }
```

```
static var highlighted: UIControl.State { get }
```

```
static var selected: UIControl.State { get }
```

- A leszármazott vezérlőelem osztály dolga, hogy mire használja fel a property aktuális értékét

> Pl.: *UIButton*-nél külön kép állítható be az egyes állapotokhoz

```
button.setImage(image, for: .highlighted)
```

UIControl eseménykezelés

- A vezérlőelemek eseményeihez akciómetódusok kapcsolhatók
 - > Interface Builderből vagy
 - > Kódból (target-action minta)

```
button.addTarget(self, action: #selector(MyClass.doSomething(with:)),  
for: .touchUpInside)  
@objc func doSomething(with string: String) { ... }
```

UITextView

- Többsoros szövegek megjelenítésére és bevitelére alkalmas nézet
 - > UITextField csak egysoros
 - > Labellel ellentétben nem kell explicit jelezni a többsorosságot
- Automatikusan scrollozódik, ha szükséges
- Képes felismerni a szövegben automatikusan telefonszámokat, linkeket, címeket
- Fapadosabb design, több munka testreszabni

Lorem ipsum dolor sit er elit lamet,
consectetur cillum adipiscing
pecu, sed do eiusmod tempor
incididunt ut labore et dolore
magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea

Virtuális billentyűzet

- Beállítási lehetőségek:
 - > Típus:
 - Interface Builder-ben: **Keyboard** property
 - Kódban: `keyboardType` property
 - > Return gomb felirat:
 - Interface Builder-ben: **Return Key** property
 - Kódban: `returnKeyType` property
- Billentyűzet elrejtése szerkesztés végén:
 - > A `TextField` *Did End On Exit* eseményt generál a billentyűzet Return gombjának megnyomásakor. Ezt összekapcsolva egy akció metódussal, a „first responder státusz” lemondásával eltüntethető a billentyűzet

First responder

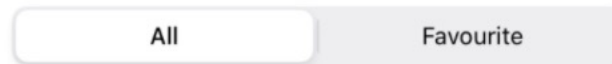
- Az az objektum, mely a felhasználói eseményeket először megkapja
 - > Mondhatjuk, hogy "ő van fókuszbán" - Pl.: megérintünk egy TextField-et
- A virtuális billentyűzet is a First Responder alapján jelenik meg
 - > Ha egy olyan nézet lesz a First Responder, ami szövegbevitelt igényel, akkor automatikusan megjelenik a billentyűzet
- Ha el szeretnénk tüntetni a billentyűzetet, akkor át kell állítani a First Responder-t
 - > Lemondás a First Responder státuszról:
`textField.resignFirstResponder()`
 - > First Responder státusz "kézi" beállítása:
`textField.becomeFirstResponder()`
 - > Egyszerre lemondani összes alnézet first responder státuszáról:
`view.endEditing(true)`

Number Pad elrejtése

- Probléma: A Number Pad virtuális billentyűzeten nincs Return gomb
- Egy megoldás: rejtjük el a billentyűzetet háttér megérintésekor
 - > A konténer nézet (háttér) típusát állítsuk át UIControl-ra, hogy tudjon eseményeket generálni
 - > Kössük össze a *Touch Down* eseményt egy akció metódussal, melyben minden TextField-en meghívjuk a `resignFirstResponder()`-t

UISegmentedControl

- Egy elem többit kizáró kiválasztása (Apple „Radio button”)
- Gombsor, mindig egy gomb aktív
- Gombváltáskor *Value Changed* esemény generálódik
- Az éppen kiválasztott gomb sorszámát a `selectedSegmentIndex` property-vel lehet lekérdezni



UIImage

- Egy képet reprezentál
- Betölthető képfájlból
 - > Projektstruktúrához adott fájlból (fájlnév kell csak)
`let image = UIImage(named: "background.jpg")`
 - > AssetCatalog-ból (felhasználói felületi elemeknél ezt használjuk)
- Betölthető dinamikusan fájlból, byte tömbből, stb.
- Támogatott formátumok: PNG, JPG, GIF, TIF, BMP, stb.
- A rendszer cache-eli a képeket, ugyanazon névhez tartozó képek betöltését és ilyenkor ugyanazt az objektumot adja vissza

Képek és a Retina kijelző

- Retina / Retina HD / Super Retina felbontású készülékekhez az alkalmazáshoz adott képekből is biztosítani kell nagyobb felbontású változatot
- Minden képből több változatot kell hozzáadni a projekthez (sima "1x", @2x és @3x)
 - > image.png
 - > image@2x.png
 - > image@3x.png
- A `UIImage(named:)` inicializáló gondoskodik róla, hogy a képből az adott kijelzőhöz tartozó változat töltsődjön be (ha nincs, akkor átskáláz egy másikat)
 - > Nem kell kiírni a @2x-et, Retinán automatikusan ez fog betöltődni
 - > PNG formátum esetén a .png-t is elhagyhatjuk

UIImageView

- Megjelenít egy UIImage-et
- Inicializálása egy UIImage-dzsel:

```
let imageView = UIImageView(image: image)
```

- A UIView-ból örökölt **contentMode** property-ben állítható, hogy hogyan skálázódjon a benne lévő kép a nézet átméretezésekor
pl.: *Aspect Fit, Aspect Fill, Scale to Fill, Center*, stb.
- Lehet animációt is megjeleníteni vele:
 - > Az **animationImages** property által tárolt tömbbe kell UIImage-eket tölteni
 - > Az UIImageView ezeket fogja váltogatni

További UI elemek I.

- UISlider

- > Egy adott szám beállítására szolgáló elem
- > A min és max értékek közötti intervallumból vesz fel értéket
- > A `value` property-ben tárolja az aktuális értéket
- > A *Value Changed* eseményben reagálhatunk a változásra



- UISwitch

- > Két állapot: ki vagy be van kapcsolva
- > Az `isOn` paraméterrel tudjuk lekérdezni az állapotot
- > A *Value Changed* eseményben reagálhatunk a változásra



További UI elemek II.

- **UIProgressView**
 - > Folyamat haladásának jelzésére
 - > 0 és 1.0 közötti érték
- **UIActivityIndicatorView**
 - > Animálható töltő képernyő nézet
 - > Két méretben
- **UIPageContol**
 - > Adott számosságú elemből az aktuális elem jelzése
 - > `currentPage`: aktuális elem beállítása


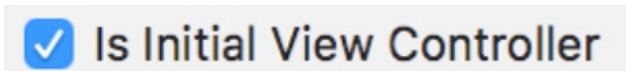


Grafikusan szerkeszthető fájlok

Storyboard

- **Storyboard:** több ViewControllerből álló nézethierarchia, mely egyben tartalmazza az alkalmazás nézeteit és a közöttük való navigációt
 - > Megadhatjuk benne a ViewControllerek nézeteit
 - > Megadhatjuk a ViewControllerek kapcsolatait és átmeneteit
- Xcode-ban grafikusan szerkeszthető (*Interface Builder*)
- Kódból is példányosíthatók a benne definiált ViewControllerek

Storyboard II.

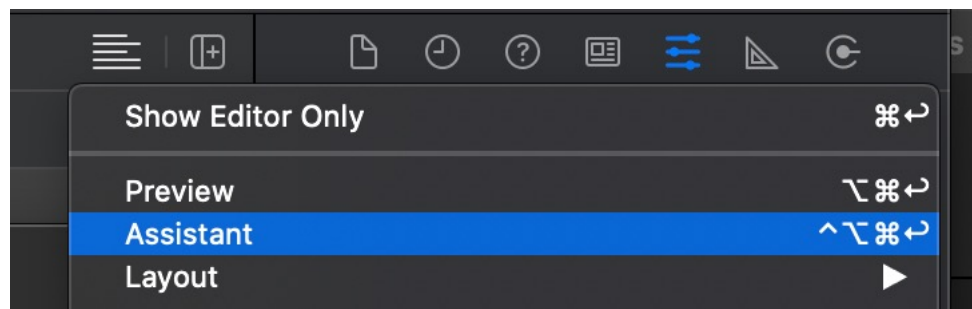
- .storyboard kiterjesztésű fájl 
 - > Valójában egy XML
- A *target* Info.plist fájljában adható meg, hogy az alkalmazás indításakor melyik Storyboard töltődjön be
 - > *Main storyboard file base name*: Main
- A Storyboard kezdeti jelenetét (*initial scene*) egy semmiből előtűnő nyíl jelöli
 - > A ViewController attribútumai között állítható be 

Storyboard III.

- Ha megadunk egy *Main Storyboard*-ot, akkor nem kell a kódból létrehoznunk az alkalmazás ablakát, ezeket a keretrendszer elvégzi helyettünk
- Hogyan adjuk meg a Storyboard-ban megadott ViewController osztályát?
 - > A Storyboard editorban kiválasztjuk a *View Controller*-t, majd az *Identity inspector*-ban a „Class” attribútumot

Storyboard összekötése a kóddal

- Minden elem, ami a storyboard-unkon van, összeköthető a hozzá tartozó ViewController-ben definiált nézettel
- Az összekötésre több megoldás is van, a legkényelmesebb az Assistant editor használata



Storyboard és a kód

- A grafikus editorban létrehozott elemek elérése a kódból: **Outlet (@IBOutlet)**
 - > Egy property, melynek értéke egy, az Interface Builderben-ben definiált, objektumra mutat
 - > Pl.: szeretnénk egy felirat (UILabel példány) szövegét a programkódból átírni, ezért a UILabel-re ráállítunk egy outlet-et
- A grafikus editorban létrehozott objektumok által generált események lekezelése: **Action (@IBAction)**
 - > Egy metódus, mely meghívódik adott esemény hatására
 - > Pl.: hívódjon meg egy metódus (ez az *akció*), ha a felhasználó megérint egy gombot
 - > Pl.: hívódjon meg egy metódus, ha megváltozik egy UITextField értéke

IBOutlet és a memória

- Az outletek legtöbbször **weak** property-k
 - > A nézetet, melyre az Outlet mutat, már felügyeli ("birtokolja") egy szülő nézet vagy a gyökér nézet esetén a ViewController
- Az outletek **implicitly unwrapped optional**-ok
 - > Amikor a ViewController példányosodik, a hozzá tartozó nézetek még nem léteznek (ezért az outletek értéke ilyenkor `nil` lesz, tehát optional-nek kell definiálni őket)
 - > Onnantól, hogy betöltődnek a nézetek, az Outletek-nek is mindig lesz értéke (ezért implicitly unwrapped: "!")

```
@IBOutlet weak var titleLabel: UILabel!
```

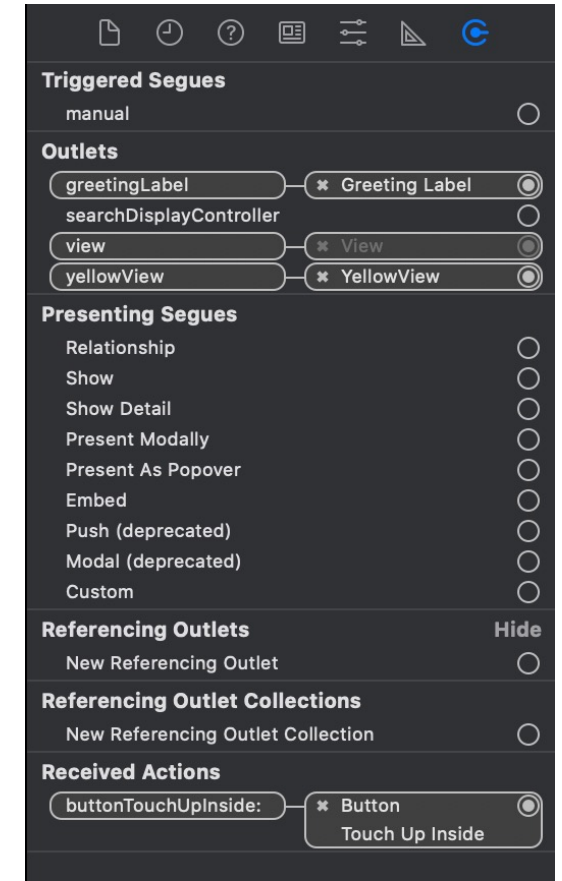
Akció metódusok

- Bizonyos nézetek felhasználói interakció hatására eseményeket bocsátanak ki
 - > Azokat a nézeteket, amik eseményeket bocsátanak ki, vezérlőelemeknek (control lásd. UIControl) nevezzük
- Ha egy objektum eseménye össze van kapcsolva egy akció metódussal, akkor az akció metódus meghívódik az esemény bekövetkeztekor
- Az események lekezeléséhez akció metódusokat definiálhatunk, ezek szignatúrája fix:

```
@IBAction func buttonDidTap(sender: AnyObject){}
```

Összekötések ellenőrzése

- Az Interface Builder *Connection inspector* nézetében láthatjuk, hogy a kijelölt ViewController-nek milyen kapcsolatai vannak a kóddal
- Outlets és Received Actions
- Ha valamilyen kapcsolat tönkrement, akkor azt teli karika helyett felkiáltó jellel jelzi a rendszer



Xib és Nib fájlok

- A XIB fájlok tetszőleges objektum fákat tárolhatnak, az Interface Builderben grafikusán szerkeszthetők
 - > Legtöbbször UI-hoz, de nincs ilyen megkötés, bármilyen osztály példánya eltárolható benne
- A rendszer futási időben képes példányosítani a XIB-ben eltárolt objektumokat
- XIB vs. NIB
 - > XIB: XML alapú, szerkeszthető, Xcode használja
 - > NIB: a XIB-ből fordításkor előállított bináris formátum, futási időben ezt használja a rendszer
 - > Dokumentációk/leírások általában NIB-ként hivatkoznak rájuk attól függetlenül, hogy épp melyik reprezentációval dolgozunk
- Felhasználói felületekhez Storyboard fájlokat érdemes használni, apróbb nézet elemekhez (pl. cellák, gombok, stb.) jók a Xib fájlok

Xib vs Storyboard

- XIB/NIB
 - > Egy ViewController vagy egy nézethierarchia
- Storyboard
 - > Több ViewController-ből álló hierarchiák
 - > A jelenetek között az átmenetek és a tartalmazás viszonyok definiálhatók
 - > Több olyan funkció, ami XIB/NIB fájlokon keresztül nem elérhető
 - UITableView cella prototípusok definiálása
 - Statikus lista (UITableView)

Nézet definiálása Xibként

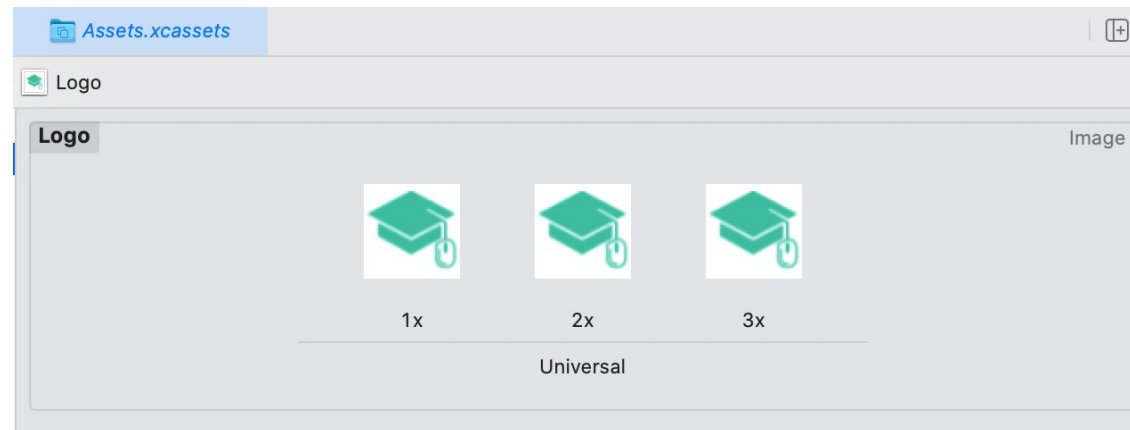
- Xib fájl: tetszőleges objektumokat tartalmazhat, de legtöbb esetben egy nézethierarchiát írunk le vele
- A *CustomButton.xib* példányosítás kódból:

```
let nib = UINib(nibName: "CustomButton", bundle: Bundle.main)
let customButton = nib.instantiate(withOwner: nil, options: nil) as? CustomButton
```




AssetCatalog

- Erőforrások (főleg képek) tárolására szolgáló katalógus
 - > Összerendeli ugyanazon kép különböző felbontású verzióit
 - > Nem kötelező speciális fájlneveket (@2x, @3x) alkalmazni
 - > Meghatározhatjuk, hogy mely device típusokon használja a képeket
 - > Külön megadhatjuk a light és dark mode képeit



AssetCatalog II.

- Átméretezhető képeket is kezel (pl. pdf, svg)
- A képekre ugyanúgy név szerint hivatkozhatunk
- xcassets mappában:  Assets.xcassets
- Egy projekthez több AssetCatalog is tartozhat

AssetCatalog III.

- AssetCatalogában nevesített színeket is felvehetünk
- Megadható, hogy külön dark és light mode-ban mi legyen a szín

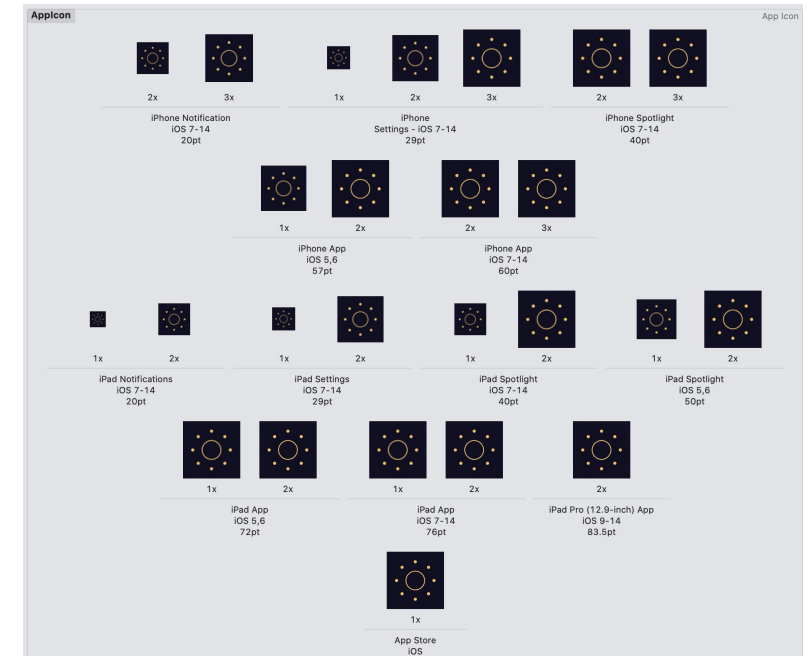


- Használata:

```
view.backgroundColor = UIColor(named: "background")
```

Alkalmazás ikon

- Az alkalmazás ikont is AssetCatalog-ban tároljuk
- A projekt App Icon Source beállításánál kell megadnunk, hogy melyik AssetCatalog tartalmazza az ikonokat
- Meg kell adni az összes felhasználási méretre az ikont
- Több weboldalon lehet generáltatni kész xcassets mappát, egyetlen nagyobb ikonból, pl.: <https://appicon.co/>



Gesture Recognizers

Gesture recognizer

- Egy-egy érintés-esemény (Touch event) feldolgozása könnyen megoldható (lásd. később), DE
 - > A bonyolultabb gesztúrák felismerése már sok programozással jár
 - > A legtöbb ilyen gesztúra hasonló
- A bonyolultabb érintés-események magasabb szintű kezelésére a UIGestureRecognizer-eket, illetve ennek leszármazottjait használhatjuk

Gesture recognizer működése

- UIView-hoz lehet hozzárendelni
 - > Hasonló esemény, mint egy UIControl event
 - > A Touch események ilyenkor alapesetben nem váltódnak ki
- Beállíthatjuk, hogy egyszerre több recognizer is detektálódjon (pl.: pinch-to-zoom és forgatás)
 - > Ehhez implementálni kell a megfelelő delegate metódust és persze az adott Gesture Recognizer-nek be kell állítani a delegate-jét is
- Interface Builderben hasonlóan lehet beállítani, mint a UIControl-okat
 - > A sender típusát a megfelelő Gesture Recognizerre kell állítani

Gesture recognize – működtetése kódból

- Létre kell hozni és be kell állítani az eseménykezelőt

```
tapGestureRecognizer = UITapGestureRecognizer(  
    target: self, action: #selector( handleTap(gesture:)))
```

- Hozzá kell adni a “kezelni” kívánt nézethez

```
self.view.addGestureRecognizer(tapGestureRecognizer)
```

- Le kell kezelni az eseményt

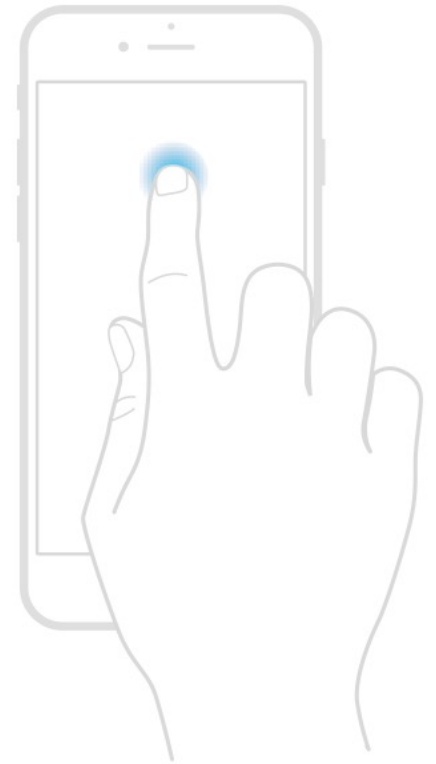
```
@objc func handleTap(gesture: UITapGestureRecognizer){  
    //Handle the gesture  
}
```

Speciális recognizerek

- Beépített, UIGestureRecognizer leszármazottak
- Egy-egy speciális, de gyakran használt gesztúra felismerésére alkalmasak
- Egyszeri eseményt kiváltók:
 - > UITapGestureRecognizer
 - > UISwipeGestureRecognizer
 - > UILongPressGestureRecognizer
- Folyamatos eseményeket kiváltók:
 - > UIPinchGestureRecognizer
 - > UIRotationGestureRecognizer
 - > UIPanGestureRecognizer

UITapGestureRecognizer

- Egyszerű érintést detektál.
 - > Pont mint a Touch event-ek, csak kicsit többet tud
- Egyedi property-k
 - > **numberOfTapsRequired:**
Mennyi érintés után jelezzen
 - > **numberOfTouchesRequired:**
Mennyi ujjal kell egyszerre a kijelzőhöz érni
 - > **buttonMaskRequired:**
Milyen fizikai gombot kell nyomni, ha egeret használunk, hogy jelezzen
- Pozíció visszaadása:
 - > **location(in view: UIView?) -> CGPoint**

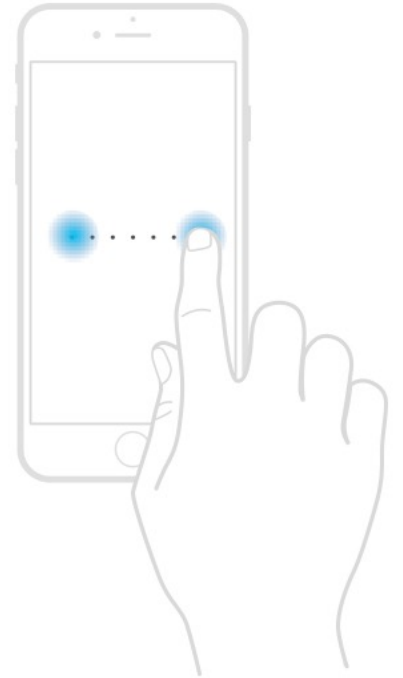


Tap
(~0.1 second)

UISwipeGestureRecognizer

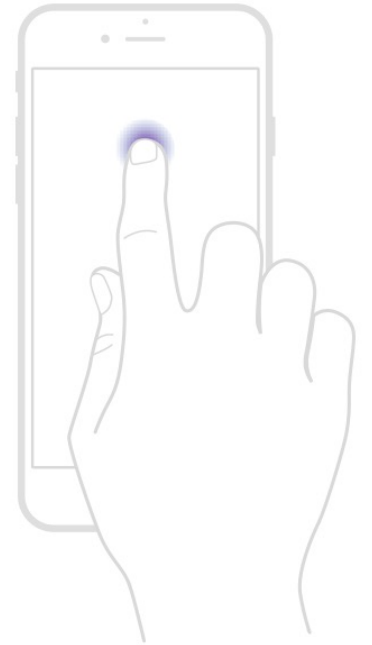
- Egy egyenes, vízszintes vagy függőleges ujjmozdulat detektálására
- Egyedi property-k
 - > **direction**: Milyen irányba történhet a swipe (balra, jobbra, fölfele, lefele)
 - > **numberOfTouchesRequired**:
Mennyi ujjal kell egyszerre a kijelzőhöz érni

Swipe



UILongPressGestureRecognizer

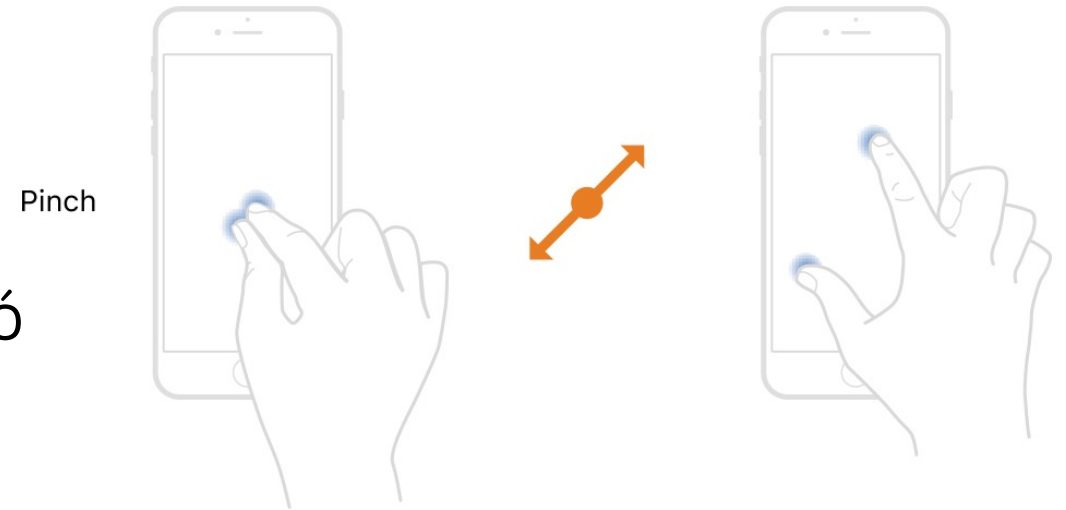
- Egy hosszabb ideig tartó érintést detektál
 - > Kb. mint egy speciálisabb Tap Gesture Recognizer, de nem leszármozottja
- Egyedi property-k
 - > **minimumPressDuration:**
Mennyi ideig kell tartania az érintésnek
 - > **allowableMovement:**
Mekkora távolságot mozdulhat el az ujj közben
 - > A Tap Gesture Recognizer-nél megismertek



Long press
(>0.5 seconds)

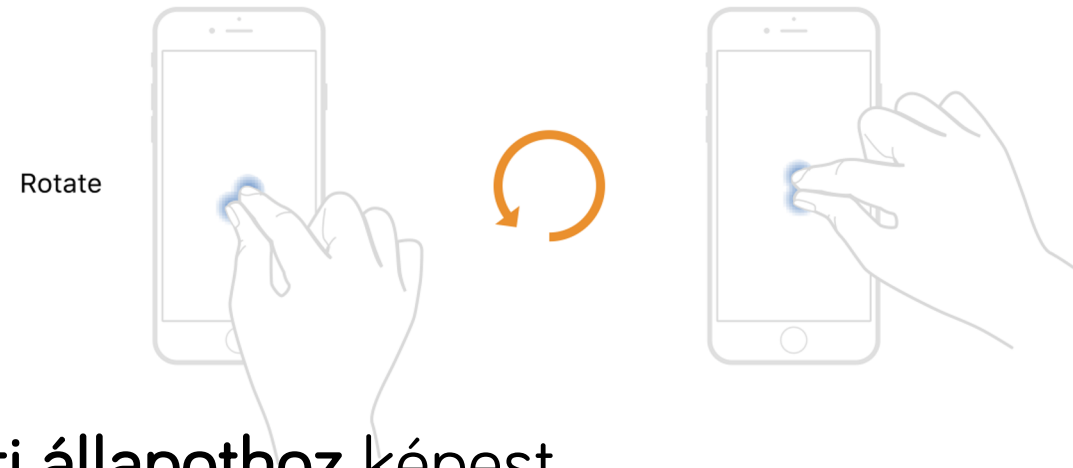
UIPinchGestureRecognizer

- “csippentő ujjmozdulat”
 - > Általában a nagyítás/kicsinyítés funkció tartozik hozzá, amit nekünk kell implementálni
- Speciális property-k
 - > **scale**: A két ujj között távolság milyen mértékben változott
 - > **velocity**: Milyen gyors volt a változás. (scale/sec)



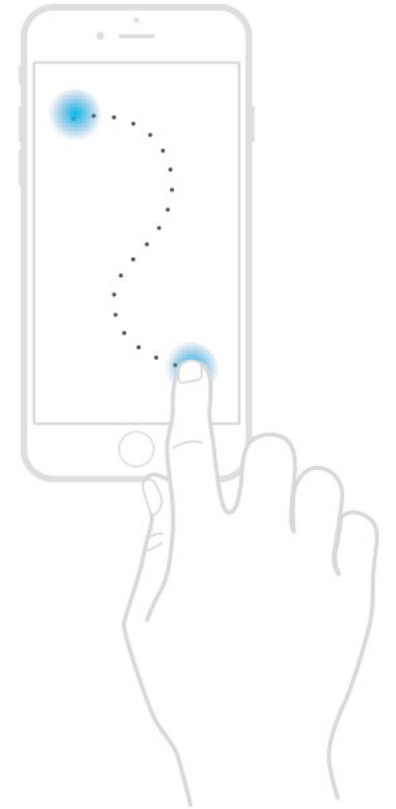
UIRotationGestureRecognizer

- A két ujj által meghatározott szakasz elfordulását detektálja
- Egyedi property-k
 - > **rotation**: Az elfordulás szöge a **kezdeti állapothoz** képest
 - > **velocity**: Milyen gyors volt a forgatás. (radian/sec)



UIPanGestureRecognizer

Pan



- A nyomva tartott ujj elmozdulását követi, amíg azt fel nem engedik
 - > Tipikusan a drag & drop műveleteknél lehet hasznos
- Speciális property-k
 - > **minimumNumberOfTouches**: Minimum hány ujjat lehet követni
 - > **maximumNumberOfTouches**: Maximum hány ujjat lehet követni
 - > **translation**: Az ujj(ak) aktuális helyzete (csak egy pont)
 - > **velocity**: Milyen gyors volt a változás. (point/sec)