

ViewControllers

IZA, Martin Hrubý, FIT VUT, 2020

Co je ViewController (VC)

- Je to prvek konceptu M-V-C.
- **Není** zobrazovaným prvkem, ale **vlastní View**.
- Programování aplikací = programování VC.
 - (ve Storyboardu..., SwiftUI se tváří jinak)
 - ... a modelů. Data a procesy reagují na potřeby VC.
 - Neuvažujeme tvorbu vlastních *elementárních* View.
 - Předpokládáme tvorbu vlastních *složených* View, typicky děděním. Velmi typicky vlastní *UITableViewCell*.

Smysl VC

- Aktualizuje obsah *Views* podle změn v *Modelu*.
- Reaguje na události pocházející z *Views*.
- Aktualizuje geometrii *Views* (iOS / macOS).
 - Rotace, vkládání do superview, dynamika rozložení prvků.
- Spolupracuje s dalšími VC aplikace.
 - Je UIResponder — je v řetězci zpracování události.
 - Předávání řízení — kdo je adresátem toku událostí.

Návrh UI pro ViewController

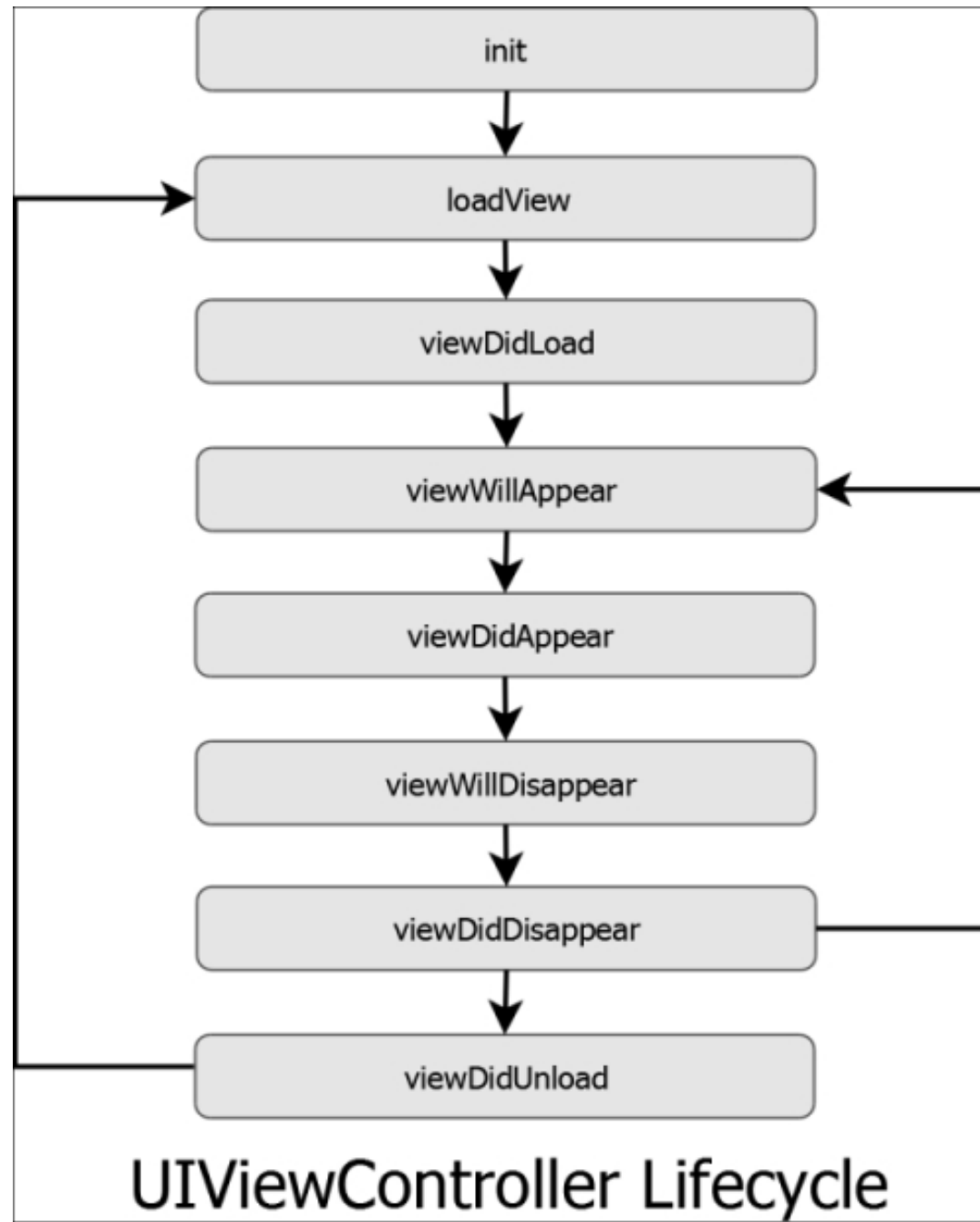
- Rozmístění UI prvků (layout, geometrie, constraints). Interface Builder (IB).
 - IB: návrh VC a jeho View! IB tvoří dva prvky.
 - Propojení na kódovou část (IBOutlet, IBAction).
- Je to filozofie přístupu k projektu:
 - Programově konstruovat (komponenty & layout),
 - Speciálně v macOS aplikacích! (složeniny V-C v oknech)
 - Interface builder (XIB / NIB) — pro jednotlivé VC.
 - StoryBoard — komfortní pro jednoduché aplikace.

Instanciacie VC

- Programem v kódu.
- Storyboard — XIB (NIB), Storyboard.
 - Automatizace správy viewControllerů.
 - XIB (XML doc), NIB (NextStep IB) — kompilát z XIB.

```
//  
let story = UIStoryboard(name: "Main", bundle: nil)  
let controller = story.instantiateViewController(withIdentifier:  
"mujModalek")  
  
// instanciacie MUJV : UIViewController  
// XIB: mujv.xib  
let mujv = MUJV(nibName: "mujv", bundle: nil)  
  
//  
self.navigationController?.pushViewController(mujv, animated: true)
```

Životní cyklus ViewController



loadView

- Instanciace VC. Pak fáze *loadView*:
 - Z metadat (NIB / Storyboard) se instancují další komponenty.
 - Propojí se komponenty s VC (*IBOutlet*, *IBAction*).
- Pak je VC ve stavu *viewDidLoad*:
 - Do *IBOutlets* už lze zapisovat.
- **LoadView se provádí až při aktivaci (present)**
VC (je to lazy...).

Poznámka: kdo koho vlastní?

- Mít přehled o vlastnictví objektů v aplikaci rovná se mít aplikaci pod kontrolou.
- *UIWindow* vlastní *rootViewController*,
- každý VC vlastní View, View tvoří hierarchii (subviews včetně vlastnictví),
- VC vlastní jiný VC,
- data vlastní buď AppDelegate nebo VC,
- specialita bude vlastnictví procesů (GCD, Operation, fronty).
- Budeme vlastnictví stále zkoumat. ARC.

Forma referencování Outlets

- Vlastníkem Outletu je View, ne VC.
 - (Co se stane, když outlet odstraním ze superView?) setHidden.
- @IBOutlet var textik : UILabel!
 - strong reference s garantovanou / předpokládanou hodnotou
- @IBOutlet var textik : UILabel?
 - je v XCode výchozí. Zřejmě reaguje na lazy loadView.
 - @IBOutlet weak var textik : UILabel?
- Totálně v rozporu s myšlenkou Swiftu.

Forma referencování Outlets

- Bylo by možné referencovat IBOutlety styly:
 - unowned, unowned(unsafe) — ??
 - VC (zprostředkovaně) vlastní Outlets, dostane je až po `loadView()`, vždy je bude mít, přesto se má ?. přistupovat.
- Odložené načtení (finální konstrukce) VC je v UIKit tradiční (instanciovat / prezentovat VC).
 - Tady můžeme **pochybovat, zda-li je to souladu s filozofií Swiftu**: `IBOutlet let x`, hodnoty známé při konstrukci VC.
 - Na tento koncepční rozpor odpovídá SwiftUI.

UI* třídy odvozují z NSObject

- Jak asi může *UIViewController* v metodě *loadView()* zapisovat do IBOutlet navazující uživatelské třídy?
- KVC — `setValue(_ val: Any, forKey: String)`
- Koncepty Foundation jsou věčné :)

```
// po instanci DetailVC máme hodnotu typu DetailVC,  
// do které UIViewController::loadView() zapisuje  
class DetailVC: UIViewController {  
    //  
    @IBOutlet var textik: UITextField?
```

Zahájení činnosti VC

- Někdo ho "musí nějak spustit".
 - *UIWindow* — *rootViewController*. Implicitně.
 - Explicitně — VC na nějakou událost prezentuje jiný VC.
- *Spustí ho jiný VC* (Navigation, TabBar).
 - *show / present VC*.
 - modální (účelový) — přebírá kontrolu.
 - *navigation / tabbar* — VC je zakomponován do *nav / tab VC*.

Modální prezentace

- Spuštění: programově nebo Storyboard.
 - Lze pomocí "segue" ([segvej]), plynule přejít.
 - VC přebírá kontrolu, tj. musí sám sebe ukončit a vrátit řízení.
- Ukončení: programově.
 - `self.dismiss(animated: completion:),`
 - `dismiss(animated: ) { escaping-closure }`
 - Předat zprávu původnímu VC, např. data.

Prezentace modálně

```
class MujModal: UIViewController {  
    //  
    @IBAction func butt() {  
        ///  
        self.dismiss(animated: true, completion: nil)  
    }  
}  
  
class ViewController: UIViewController {  
  
    @IBAction func buttRUN() {  
        //  
        let story = UIStoryboard(name: "Main", bundle: nil)  
        let controller = story.instantiateViewController(withIdentifier:  
"mujModalek")  
  
        present(controller, animated: true, completion: nil)  
    }  
}
```

```
private extension UIStoryboard {  
  
    static func main() -> UIStoryboard { return UIStoryboard(name: "Main", bundle: Bundle.main) }  
  
    static func leftViewController() -> SidePanelViewController? {  
        return main().instantiateViewController(withIdentifier: "LeftViewController") as? SidePanelViewController  
    }  
  
    static func rightViewController() -> SidePanelViewController? {  
        return main().instantiateViewController(withIdentifier: "RightViewController") as?  
SidePanelViewController  
    }  
  
    static func centerViewController() -> CenterViewController? {  
        return main().instantiateViewController(withIdentifier: "CenterViewController") as? CenterViewController  
    }  
}
```

Přechod na jiný VC: předání dat

- Předpokládáme režim Master-Detail:
 - MasterVC spustí DetailVC a **předá do něj data**,
 - DetailVC vykoná svou akci, **předá data do MasterVC**,
 - DetailVC předá řízení zpátky do MasterVC.
- Implementačně:
 - Ručně — instanciací VC,
 - StoryBoard a `prepareForSegue`,
 - maximální znovu-použitelnost kódu! Specificky v macOS!


```

class DetailVC: UIViewController {
    //
    @IBOutlet var textik: UITextField!
    var obsahModelModal: String!
    // rusim DetailVC, vracim rizeni zpatky
    @IBAction func buttDone(sender: AnyObject) { dismiss(animated: true) }
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // Predpokladam mi nekdo nastavil
        textik.text = obsahModelModal
    }
}
//
class MasterVC: UIViewController {
    // datovy obsah
    var obsahModel: String = "Ahoj"
    //
    @IBAction func butt(sender: AnyObject) {
        //
        let sb = UIStoryboard(name: "Main", bundle: nil)
        // pokud nelze as! DetailVC, at to slitne
        let vc = sb.instantiateViewController(withIdentifier: "mujModalek") as! DetailVC
        // 1) jeste neprobello viewDidLoad na DetailVC
        vc.obsahModelModal = obsahModel
        // prezentovat = 1) Nacist UI, 2) predat rizeni
        /// !!!! pozor, NENI synchronni !!!!
        present(vc, animated: true) {
            //
            print("Ted je DetailVC viditelny")
            // 2) Tady uz zapisujeme do vnitriho objektu DetailVC
            vc.textik.text = "Ahoj podruhe"
        }
        print("!!!! jeste tu neni !!!!")
    }
}

```

VC jako funkce(vstup) -> výstup

- Znovupoužitelnost VC, tj. nezávislost na okolním kódu aplikace.
 - Máme VC-A volající VC-X,
 - Máme VC-B volající VC-X,
 - Do kterého VC (A / B) má programově X vracet odpověď?
- VC pracuje nad strukturou vstupních dat.
- VC delegátsky vrací strukturu výstupních dat.

```

protocol DetailVCDelegate : AnyObject {
    //
    func finished(result: DetailVCData);
}

struct DetailVCData {
    //
    var obsah: String
    weak var delegate: DetailVCDelegate?
}

class DetailVC: UIViewController {
    //
    @IBOutlet weak var textik: UITextField!
    var data: DetailVCData = DetailVCData(obsah: "None", delegate: nil)
    // rusim DetailVC, vracim rizeni zpatky
    @IBAction func buttDone(sender: AnyObject) {
        //
        data.obsah = textik.text ?? "none"
        // callback je async!!!
        dismiss(animated: true) { self.data.delegate?.finished(result: self.data) }
    }
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // Predpokladam mi nekdo nastavil
        textik.text = data.obsah
    }
}

```

```

struct DetailVCData {
    //
    var obsah: String
    var delegate: ((DetailVCData) -> ())?
}
//
class DetailVC: UIViewController {
    //
    @IBOutlet weak var textik: UITextField!
    var data: DetailVCData = DetailVCData(obsah: "None", delegate:
nil)
    // rusim DetailVC, vracim rizeni zpatky
    @IBAction func buttDone(sender: AnyObject) {
        //
        data.obsah = textik.text ?? "none"
        //
        dismiss(animated: true) { self.data.delegate?(self.data) }
    }
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // Predpokladam mi nekdo nastavil
        textik.text = data.obsah
    }
}

```

```

class MasterVC: UIViewController {
    // datovy obsah
    var obsahModel: String = "Ahoj"
    //
    @IBAction func butt(sender: AnyObject) {
        //
        let sb = UIStoryboard(name: "Main", bundle: nil)
        // pokud nelze as! DetailVC, at to slitne
        let vc =
sb.instantiateViewController(withIdentifier: "mujModalek")
as! DetailVC
        // 1) jeste neproběhlo viewDidLoad na DetailVC
        vc.data = DetailVCData(obsah: "Ahoj") {
            // Typ "dt" je zřejmy, [weak self]!
            dt in self.obsahModel = dt.obsah
        }
        // prezentovat = 1) Nacist UI, 2) predat rizeni
        present(vc, animated: true)
    }
}

```

Pozn.: předávání zpráv mezi VC

- Synchronní / asynchronní — typicky přes `DispatchQueue`.
- Velmi typicky předáváte zprávu do VC, který:
 - není viditelný — je "pod vámi" v zásobníku `NavVC`.
 - už nemusí existovat — delegáti / closures — `weak` / `weak self`.
 - logické napojení na akce do Modelu — klást důraz na oddělení Modelu od VC (globální správa otevřených souborů, síťových spojení apod.).
 - Pozn.: closures jsou fajn, ale můžou totálně rozbít kód aplikace.

Storyboard, Seque

- Obecnější mechanismus definice přechodů mezi VC:
 - na stisk tlačítka, označení buňky TableView, ...
 - metadata přechodu VC->VC, tzv. segue,
 - `prepareForSegue(segueMetadata, sender)`.
- Segue: metadata přechodu:
 - source, destination (**nově instancovaný!!!**), identifier.
 - Styl Master-Detail: segue generuje novou instanci Detail.

Segue tam...

```
//  
class MasterVC: UIViewController {  
    // datovy obsah  
    var obsahModel: String = "Ahoj"  
  
    // akce byla navazana na tlacitko (namisto IBAction)  
    override func prepare(for segue: UIStoryboardSegue, sender:  
Any?) {  
        // test: nazev segue a podmienky downcast na DetailVC  
        if segue.identifier == "gotoDetail",  
            let _vc = segue.destination as? DetailVC {  
            // _vc je ve stavu init (neni jeste "loaded")  
            _vc.data = DetailVCData(obsah: obsahModel) {  
                // Typ "dt" je zrejmy, [weak self]!  
                dt in self.obsahModel = dt.obsah  
            }  
        }  
        /// else????  
    }  
}
```


Segue zpět...

- Konvenčně: tlačítko, IBAction, předání dat, dismiss(...)
- Lze i pomocí segue (pozor: znovu se alokuje cíl).

```
// uzivatelsky definovany segue
class DismissSegue: UIStoryboardSegue {
    // akce na segue
    override func perform() {
        //
        let controller = self.source as UIViewController
        controller.dismiss(animated: true, completion: nil)
    }
}
```

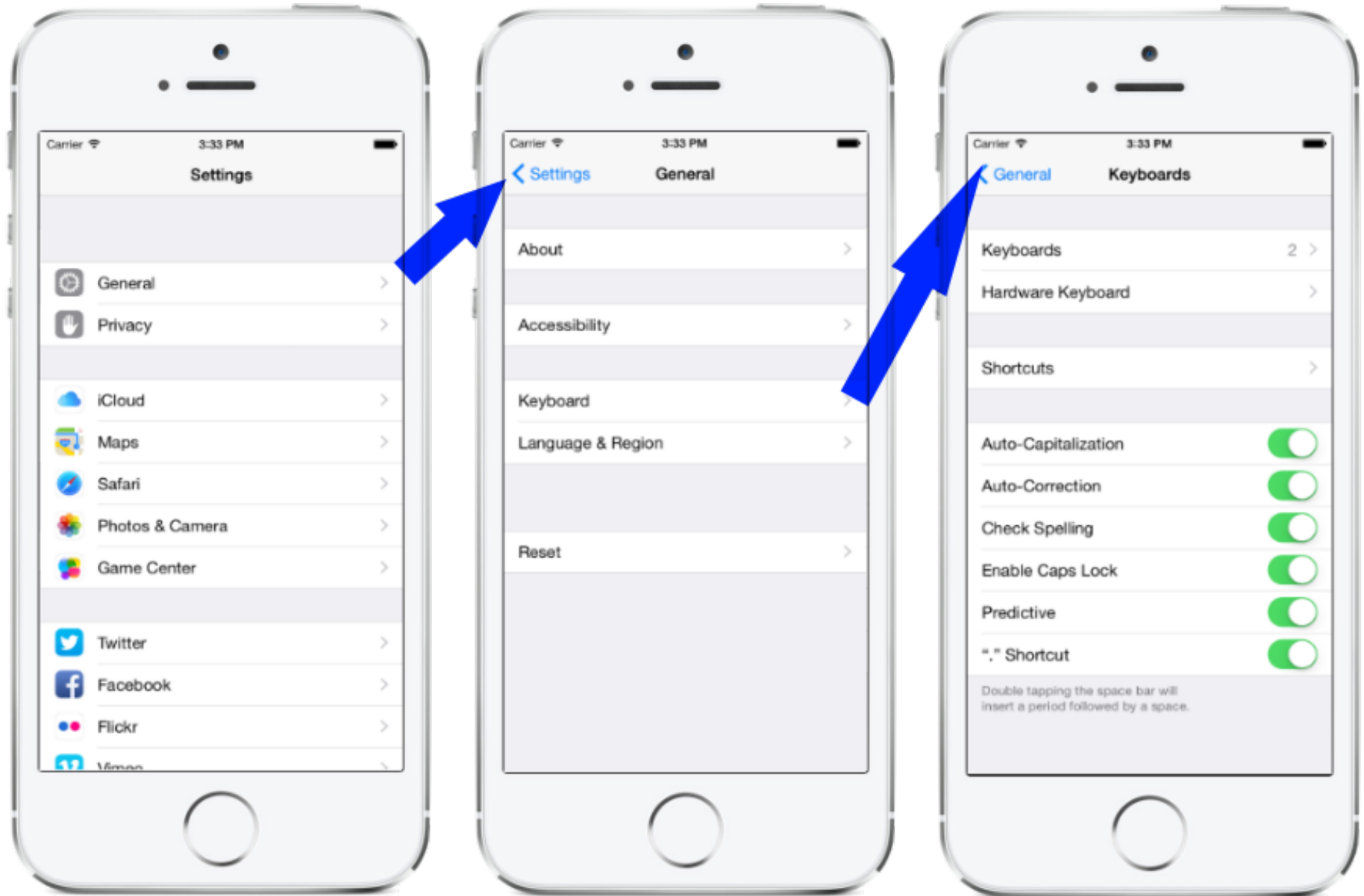
VC nad sdíleným objektem (ref)

- *MasterVC* předá do *DetailVC* referenci na sdílený objekt.
- *DetailVC* zapíše do sdíleného objektu.
 - Typicky nad `NSManagedObject` (CoreData).
- Je potřeba, aby *MasterVC* dostal explicitní zprávu o návratu z *DetailVC*?
 - Uvidíme. `NSFetchedResultsController`. `NSArrayController`.

Kontejnerové VC

- **NavigationController: zásobníkový.**
 - push / pop.
 - Přidává horní tlačítkovou lištu. Tlačítko pro návrat (!!!).
 - **Kdykoli chcete horní lištu, chcete NavVC.**
- **TabBarController:**
 - spodní lišta s ikonkami (tlačítka).
 - Ikonka (TabBarItem) — název, obrázek, VC.
- **Kombinace TabBarController / Navigation.**
 - Horní a spodní lišta. NVC pro každý VC TabBarItem.

Navigation VC



NavigationController

- Tlačítková lišta. Přidání tlačítka (UI Builder, programově).
 - Dynamika tlačítek. *navigationItem.left/right*
- Uživatelský VC je "embedded" v NVC.
 - *UIViewController* dodává vypočtené property:
 - *navigationItem* — přístup na tlačítkovou lištu
 - *navigationController: UINavigationController?*
 - Segue: typ "Show (e.g. push)".

NavigationController

- Pole odložených VC (stack), *rootViewController*.
- Operace:
 - `_mount(vc)` — `view.addSubview(vc.view)`
 - `_unmount(vc)` — `vc.view.removeFromSuperview()`
 - `push(vc)` — `unmount(původní)`, `mount(vc)`
 - `pop()` — `unmount(současný)`, `mount(poslední ve stack)`.

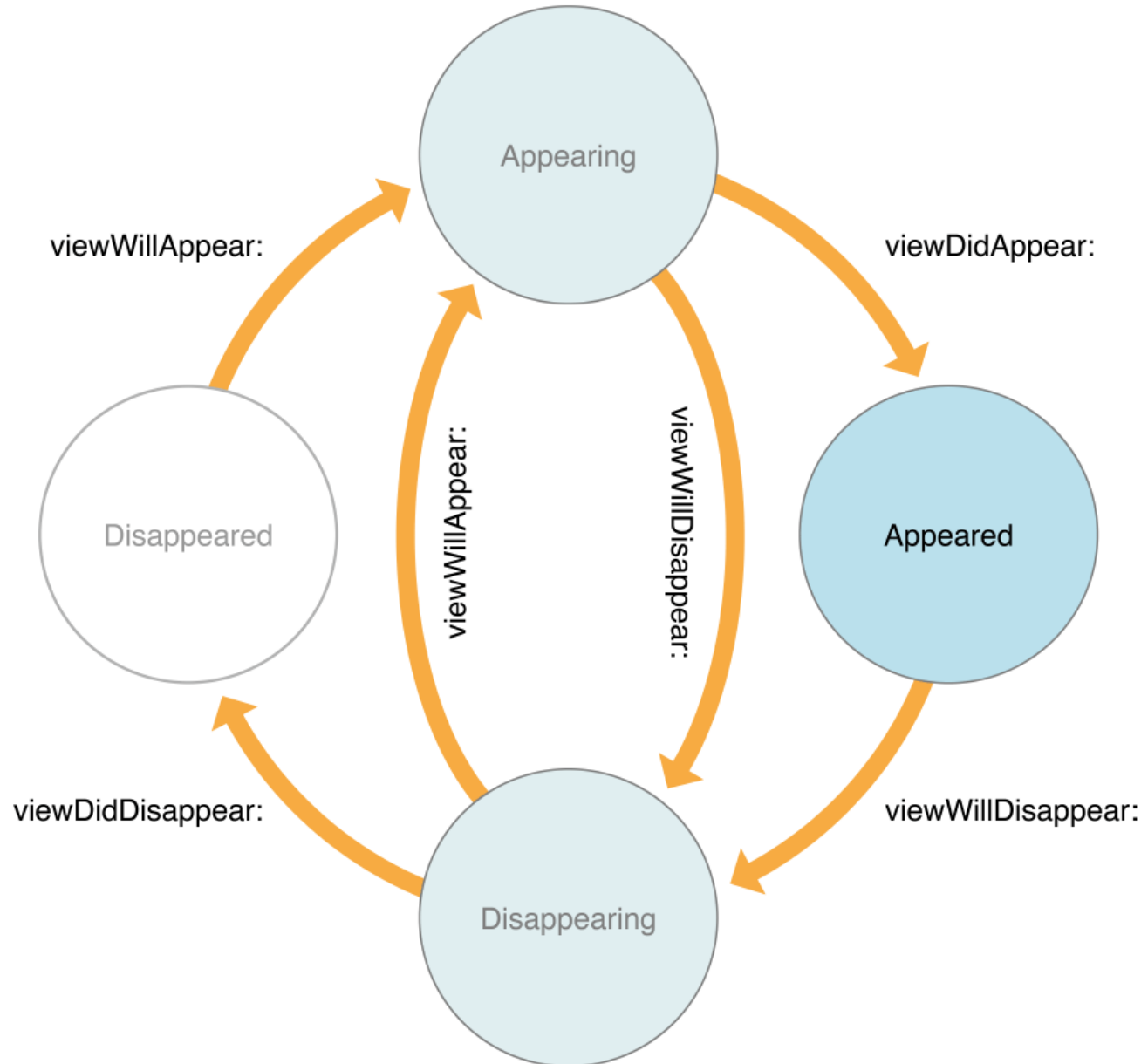
NVC, _unmount(vc)

- Deaktivace původního vc : UIViewController:
 - vc.willMove(toParentViewController: nil)
 - vc.view.removeFromSuperview()
 - vc.removeFromParentViewController()
- Pozn.: výměnu views (starý / nový) lze provést animovaně.

NVC, _mount(nvc)

- Aktivace nového VC, nvc:
 - addChildViewController(nvc)
 - view.addSubview(nvc.view)
 - nvc.view.frame = view.bounds
 - nvc.view.autoresizingMask = [.flexibleWidth, .flexibleHeight]
 - nvc.didMove(toParentViewController: self)

Životní cyklus zobrazování VC



viewWill / Did...

- *viewDidLoad()* — dodělavky do UI, inicializace hodnot UI-komponent.
- *viewWillDisappear* — zpráva o tom, že VC.view bude odstraněn ze svého superView.
 - Uklidit po sobě. Předat data. (*isMovingFromParent*)
 - Korektně: **func** willMove(toParent parent: **UIViewController?**)
 - Korektně: explicitní "back-Button" s vlastní obsluhou.
- *viewWillAppear* — nepoužívat pro převzetí dat.

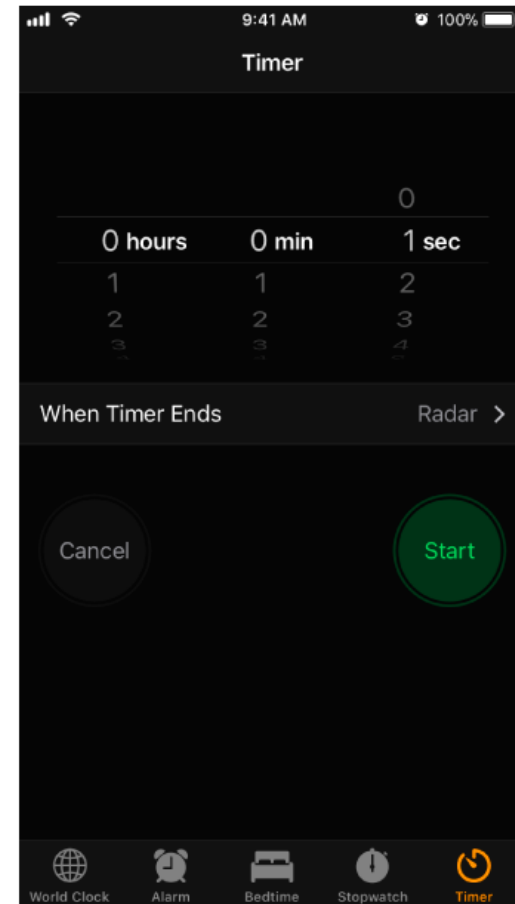
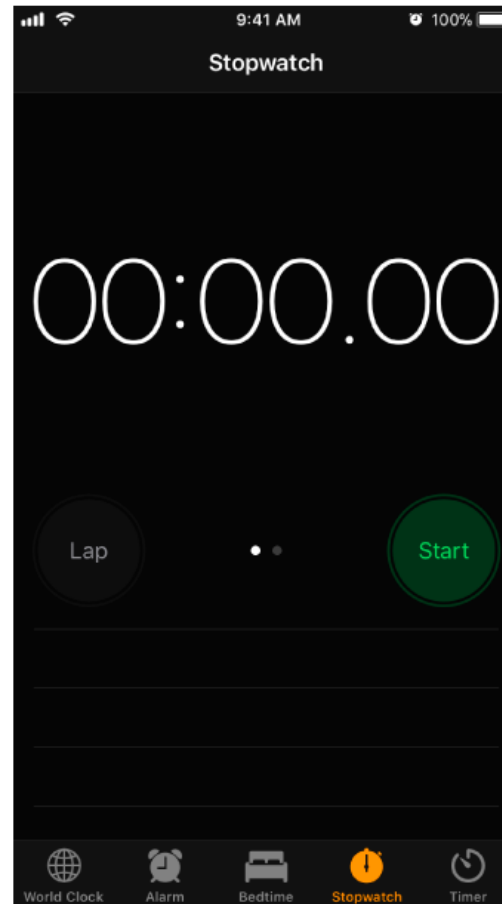
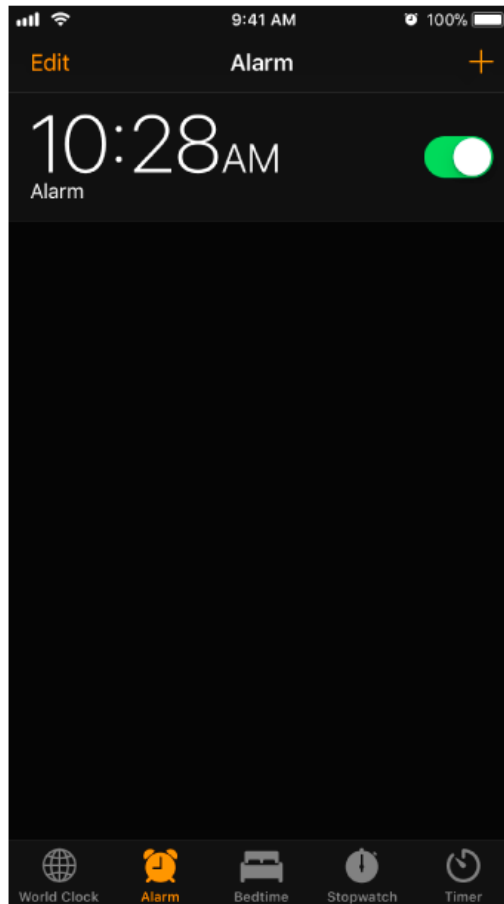
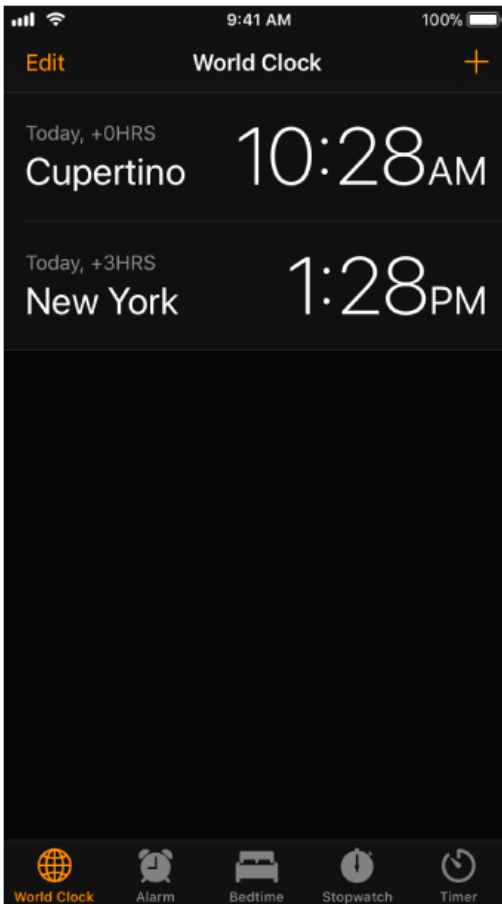
Navigation, tlačítka

```
class MasterVC: UIViewController {
    //
    var _buttDone: UIBarButtonItem?
    //
    @objc func loadDoc(_ sender: AnyObject) {
        _buttDone?.isEnabled = false
        // naplanuj nejakou praci do vedlejsiho vlakna
        DispatchQueue.global().async {
            // ... simulujeme si praci 3s delay
            Thread.sleep(forTimeInterval: 3)
            // do hlavniho naplanuj znovu-povoleni tlacitka
            DispatchQueue.main.async {
                //
                self._buttDone?.isEnabled = true
            }
        }
    }

    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // programova instanciace
        _buttDone = UIBarButtonItem(barButtonItemSystemItem: UIBarButtonItem.SystemItem.action,
                                     target: self,
                                     action: #selector(loadDoc(_:)))

        // navigationItem je vypoctena property v UIViewController
        self.navigationItem.rightBarButtonItem = [_buttDone!]
    }
}
```

TabBar VC



Přehled základních VC

- *NavigationVC* — kontejner VC. Zavádí horní lištu (tlačítka). Obaluje jiný VC.
- *TabBarVC* — pevně stanovený rozsah VC. Spodní tlačítková lišta.
- *TableViewController* — tabulka / seznam.
- *SplitViewVC* — řídí geometrii více VC.
 - Zejména v aplikacích pro macOS.
- VC má properties na ref příp. kontextu VC.

TabBarVC

- StoryBoard. Programově (TabBarItem).
- VC jednotlivých "tabs" se načítají (loadView) odloženě (lazy).
- Lze detekovat přepnutí tabů:
 - UITabBarControllerDelegate. Ne pomocí viewWill/did.
 - Detekovat přepnutí, odchytit, poslat do tabu zprávu.

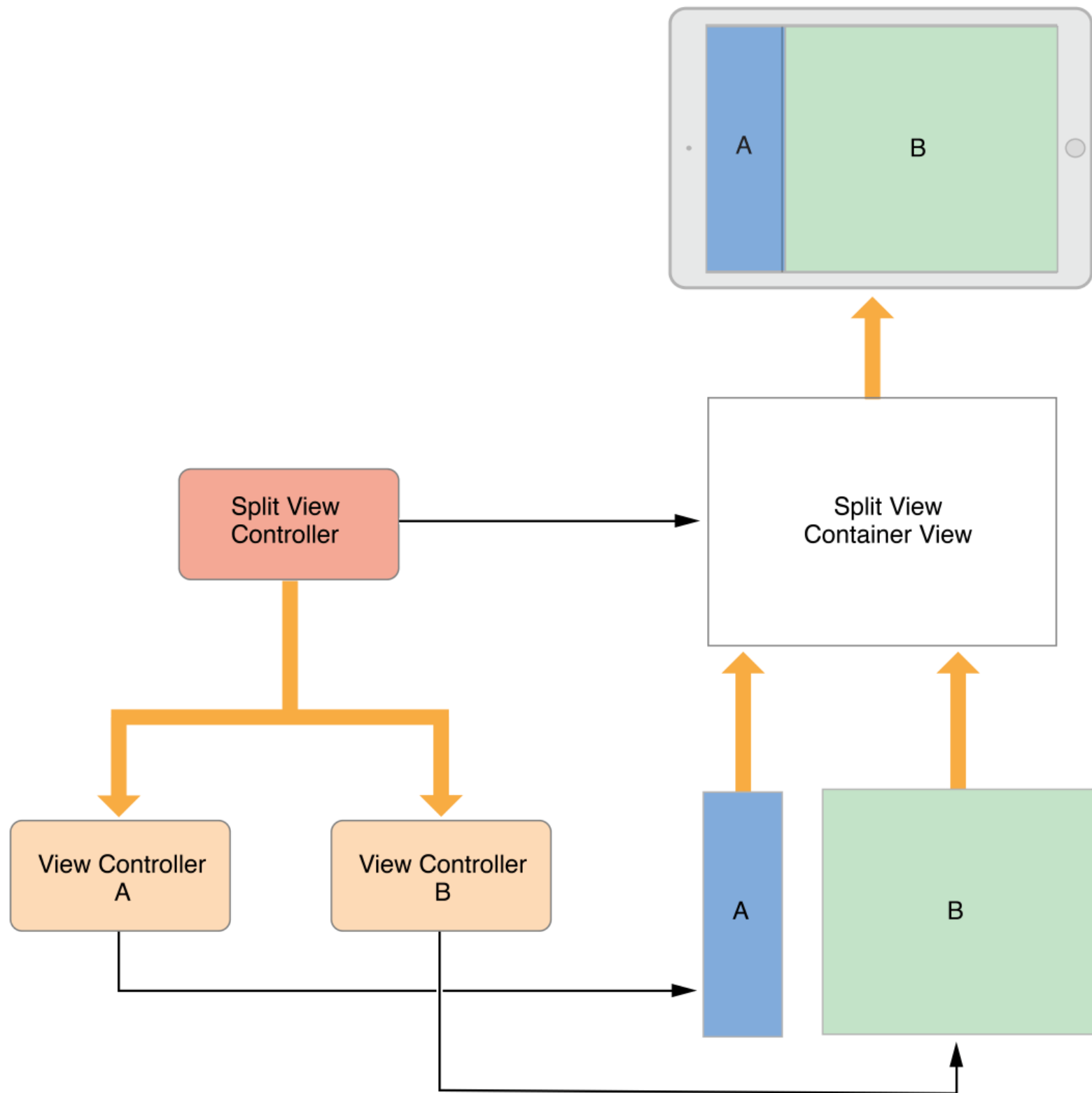
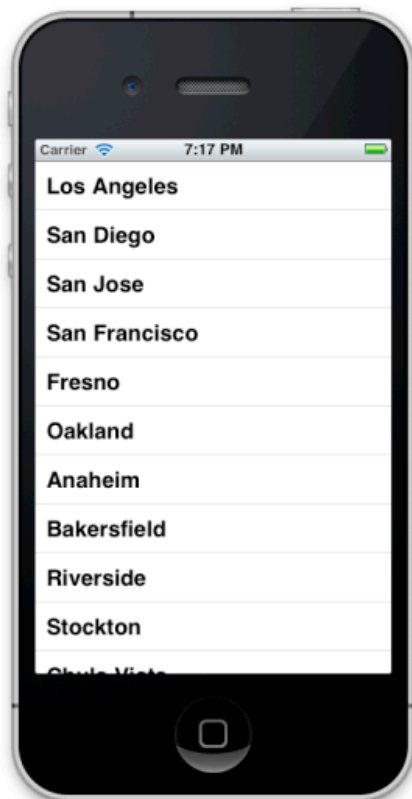
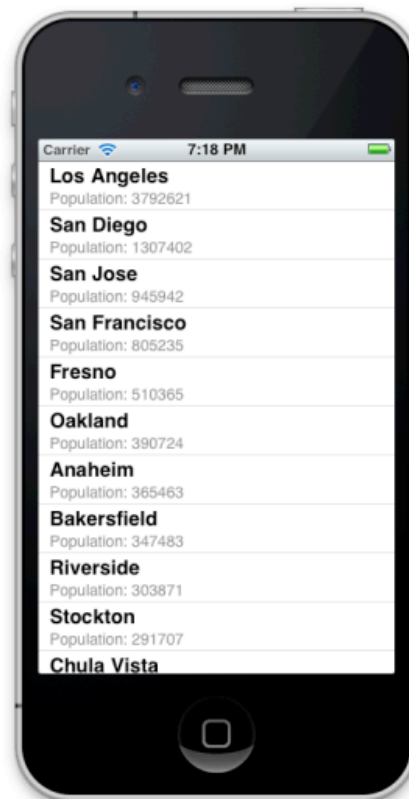


Table View Controller

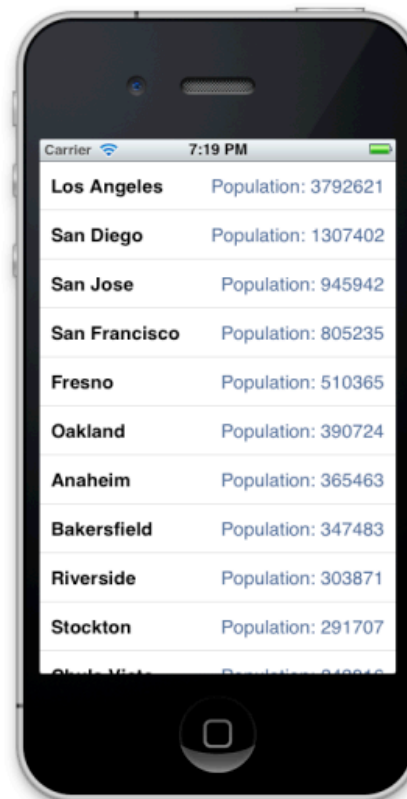
- Nejdůležitější prvek UI iOS.
- Seznam buněk (TableViewCell).



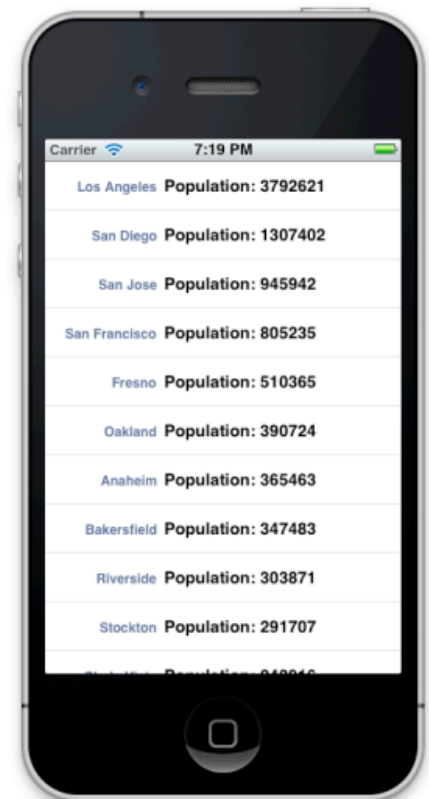
UITableViewCellStyleDefault



UITableViewCellStyleSubtitle



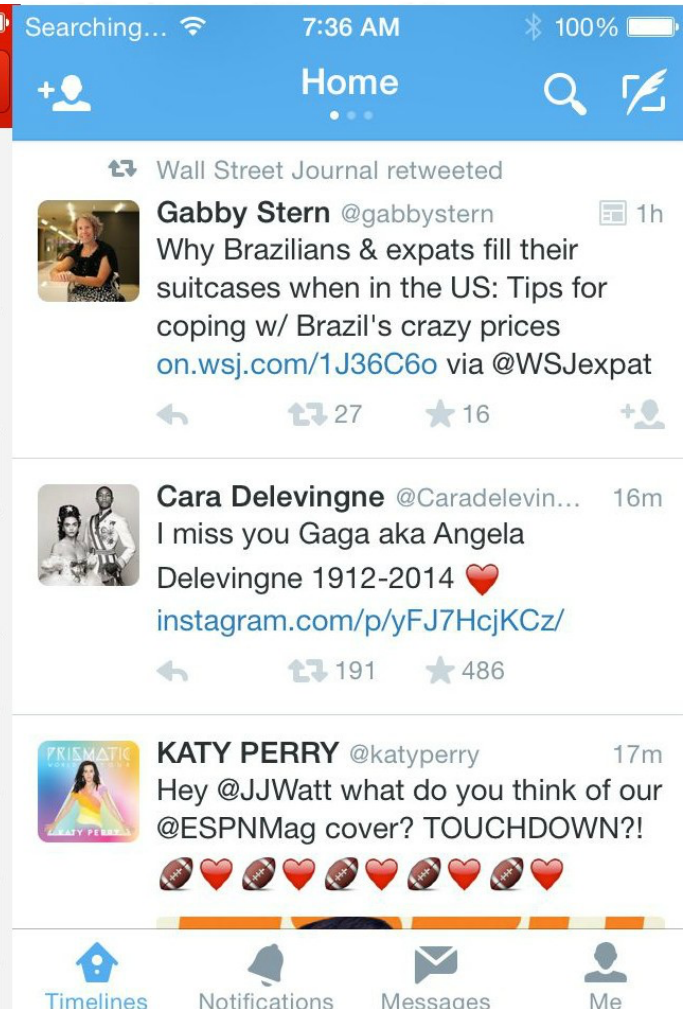
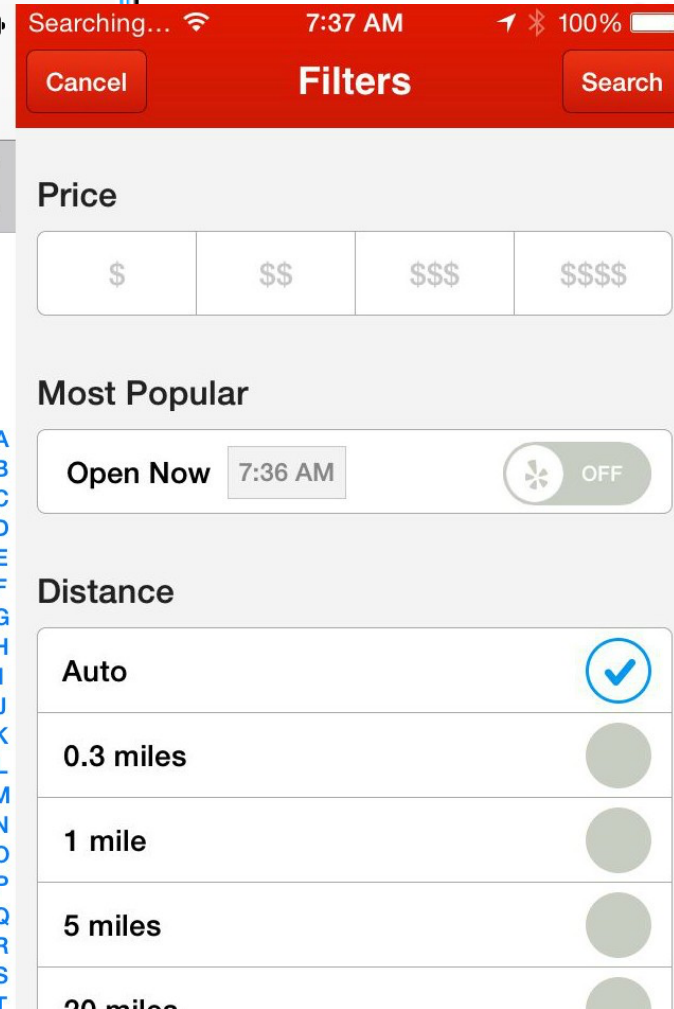
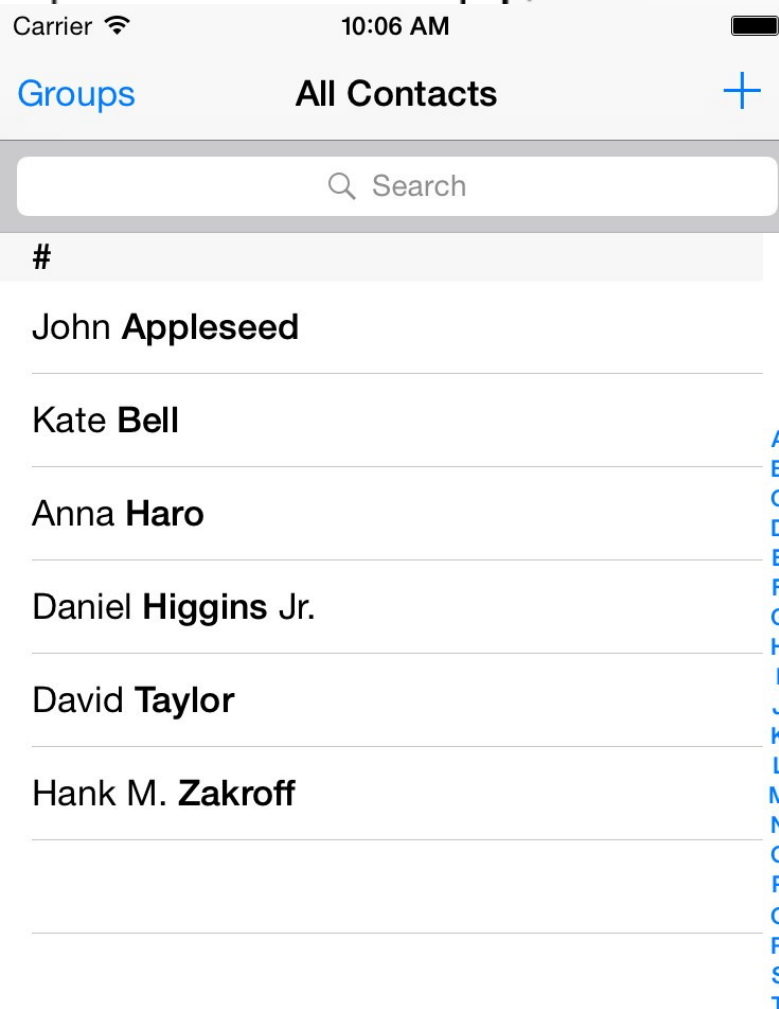
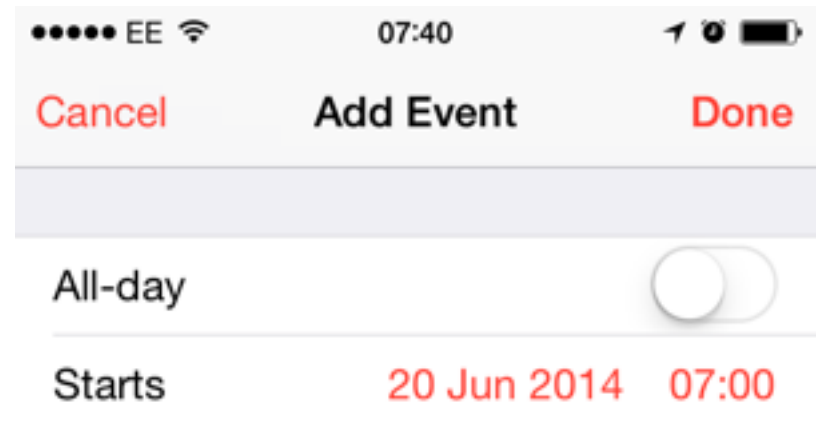
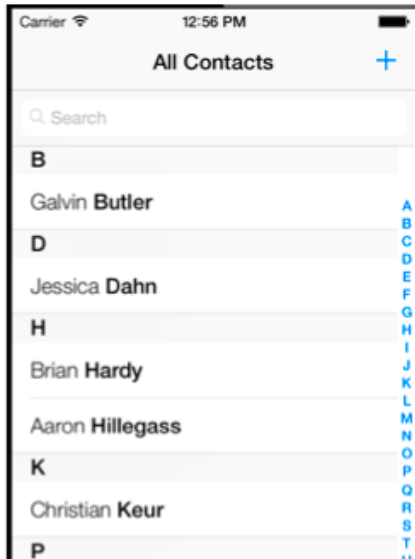
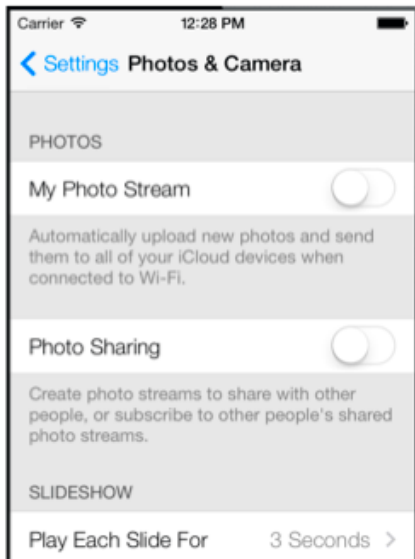
UITableViewCellStyleValue1



UITableViewCellStyleValue2

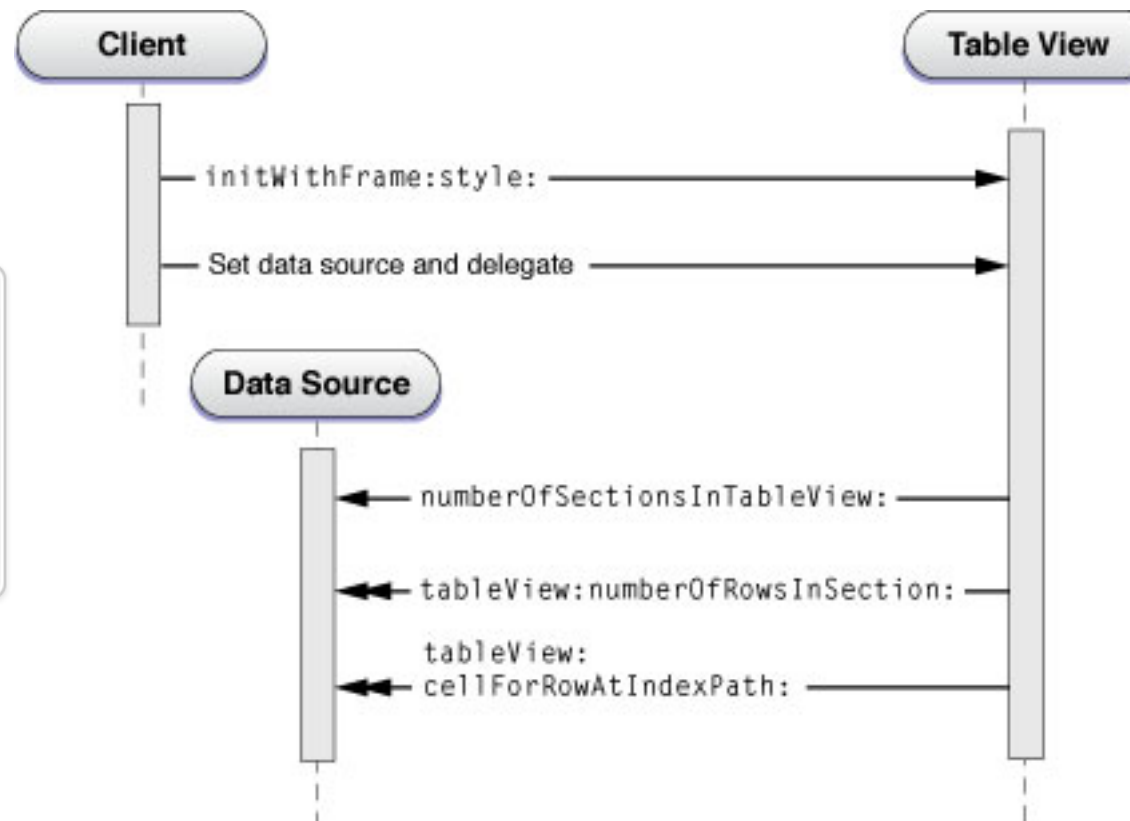
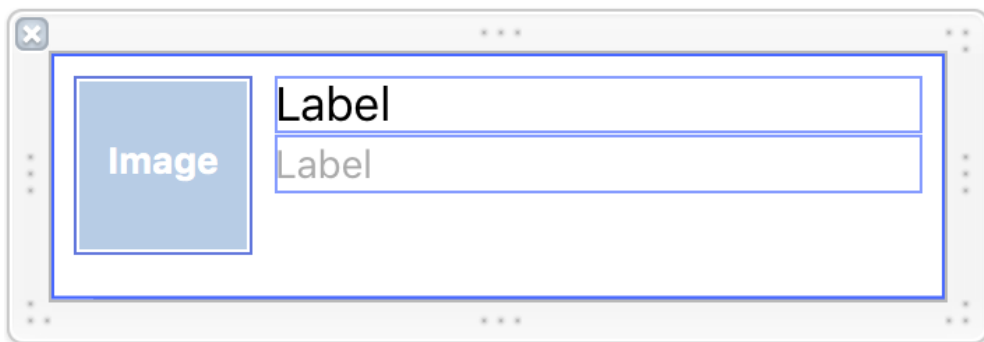
Tabulka jako koncept

- *TableView* — *View*, dotazuje se přes *dataSource* / *delegate*.
- *TableViewDataSource* — topologie tabulky, obsah.
- *TableViewDelegate* — geometrie, události, povolení k ...
- *TableViewController* — VC implementující *dataSource* a *delegate*.
- Zapojení *TableView* v aplikačním VC:
 - Aplikační VC odvozuje od TVC, nahrazuje vybrané metody *dataSource* / *delegate*.
 - Aplikační VC má TV jako *subView* a tvoří *dataSource*. Co *delegate*?



Řízení — dataSource

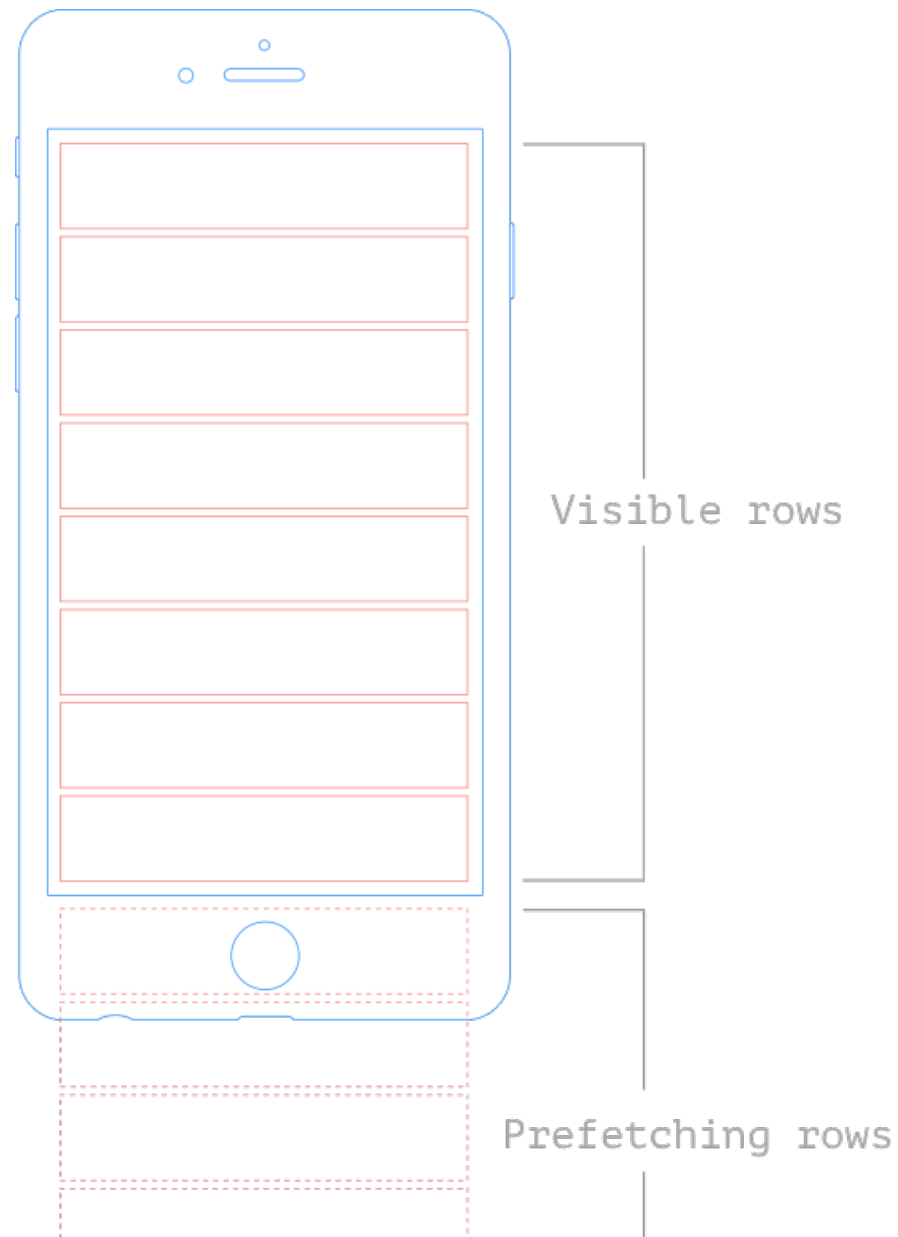
- TVC zprostředkovává obsah (Model) do View.
- Model zde vystupuje jako "dataSource" API:
 - počet sekcí, počet řádků v sekci.
 - sestavení TableViewCell pro zadanou souřadnici (IndexPath).



Řízení TV, dynamika

- TV zjistí datový rozsah (sekce, řádky).
- Předpokládá se, že viditelná je pouze část buněk.
- Ty jsou alokovány a inicializovány.
- Při posuvu tabulkou se dynamicky volá dataSource na doplňování obsahu buněk.
- Znovupoužití objektů buněk (pool). Identifikátory buněk.

Dynamika přístupu na dataSource



Demo

```
class SimpleTab: UITableViewController {  
    var seznam = ["Jeden", "Druhy", "Treti"];  
  
    override func tableView(_ tableView: UITableView,  
        numberOfRowsInSection section: Int) -> Int  
    {  
        return seznam.count  
    }  
  
    override func tableView(_ tableView: UITableView,  
        cellForRowAt indexPath: IndexPath) -> UITableViewCell  
    {  
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",  
for: indexPath)  
  
        cell.textLabel?.text = seznam[indexPath.row]  
  
        return cell  
    }  
}
```

Demo, oddělený dataSource

```
class MyArrayModel: NSObject, UITableViewDataSource {
    let seznam : [String]
    init(withStrings : [String]) {
        self.seznam = withStrings;
        super.init();
    }
    func tableView(_ tableView: UITableView,
                   numberOfRowsInSection section: Int) -> Int
    {
        return seznam.count
    }
    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
for: indexPath)

        cell.textLabel?.text = seznam[indexPath.row]

        return cell
    }
}
```

Demo, oddělený dataSource

```
class ModelTab: UITableViewController {  
    override func viewDidLoad() {  
        super.viewDidLoad();  
        //  
        self.tableView.dataSource = MyArrayModel(withStrings:  
["1", "2", "3"]);  
    }  
}
```


Dynamika, posuv v Table

- Chod aplikace musí být hladký.
- Inicializace buněk může brzdit chod tabulky (rychlý posuv).
- Při náročnější inicializaci se přechází do bočních vláken (GCD).

```

override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) ->
UITableViewCell
{
    let cell = tableView.dequeueReusableCell(withIdentifier: "tabik", for:
indexPath)

    cell?.imageView?.image = placeholder;
    // do vedlejsiho/globalniho vlakna naplanuju...
    DispatchQueue.global().async {
        // sestaveni obsahu pro bunku tabulky
        if let loaded = myModel.load(cosi) {
            // pokud mame obsah, jdeme zpatky do mainThread
            DispatchQueue.main.async {
                // pokud na indexPath stale existuje bunka
                if let ncell = tableView.cellForRow(at: indexPath) {
                    // aktualizuji
                    ncell.imageView?.image = loaded;
                }
            }
        }
    }
    //
    return cell;
}

```

Život jedné buňky...

- Buňka UITableViewCell je instanciována a je:
 - používaná tabulkou, pak má indexPath (dotazy na TableView).
 - nepoužívaná tabulkou, (nemá indexPath, nemá superview, je v poolu tabulky).
- Přechody:
 - prepareForReuse() — při alokaci z poolu.
 - Zařazení do tabulky.
 - (removeFromSuperview()) — přechází do poolu.)
 - since iOS 6.0>>> (void)tableView:(UITableView *)tableView didEndDisplayingCell:(UITableViewCell *)cell

Kontext buňky

- Objekt tabulky — analýza superviews.
- IndexPath v tabulce — ukládat v buňce jen velmi obezřetně nebo vůbec.
- VC tabulky.

```
extension UIView {  
    //  
    func getsv<T:UIView>(type: T.Type) -> T? {  
        //  
        guard let _sv = superview else { return nil }  
        //  
        return (_sv as? T) ?? _sv.getsv(type: type)  
    }  
}
```

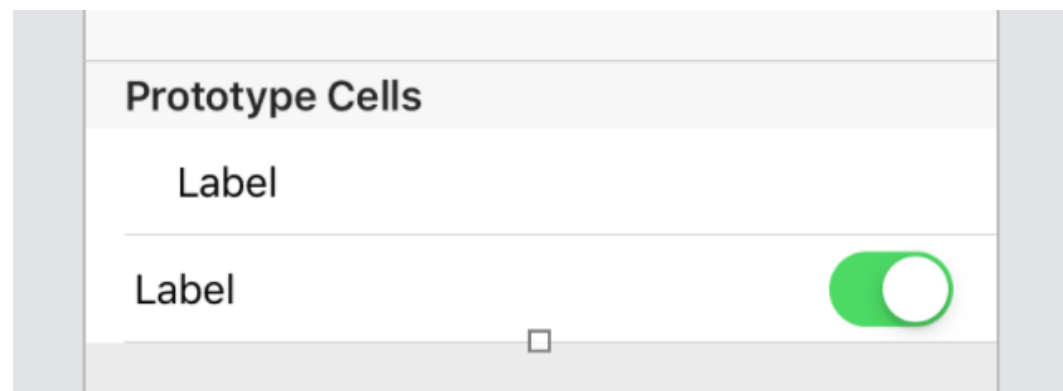
Datový model Tabulky

- `table.reloadData()`:
 - Tabulka zjistí 1) sekce, 2) počty řádků — model obsahu.
 - Pak je její model obsahu **dogmatem** do dalšího **`reloadData()`**.
 - Dualita `dataSource` a vnitřní model tabulky. Šéfuje tabulka.
 - Možno tabulce zadávat změny: `add / remove row / section`.

```
extension UITableViewCell {  
    //  
    var myIndexPath: IndexPath? {  
        //  
        return getview(type: UITableView.self)?.indexPath(for: self) ?? nil  
    }  
}
```

Uživatelská buňka

- Odvozeno od UITableViewCell.
- Prototyp v IB, v kontextu Tabulky:
 - Její třída. Její reuse-id.
 - IBOutlets, napojení. Konfigurace ve VC.
- (Datový) model pro buňku.



Model pro buňku

- Buňka má zobrazovat nějakou hodnotu, tj. hodnot lze předat, uložit...

```
// struct/class
class DatabaseItem {
    //
}

// bunka
class CellForDI: UITableViewCell {
    //
    weak var model: DatabaseItem?

    // volano z dataSource TVC
    func selfConfig(withDI: DatabaseItem) {
        //
        model = withDI
        // inicializuj IBOutlets
    }
}
```

Události z buňky

- Posílat do buňky. Pak z buňky:
 - delegátství (protokol), closure.
 - Zprávy jdou pochopitelně z MainThread.

```
//  
class MySwitchCell : UITableViewCell {  
    //  
    @IBOutlet weak var textik: UILabel!  
    @IBOutlet weak var swi: UISwitch!  
    // udalost od objektu "swi" pri zmene hodnoty  
    @IBAction func hasChanged(_ sender: AnyObject) {  
        // aktivuju kod v kontextu ....  
        delegate?()  
    }  
  
    //  
    var delegate: (() -> ())?  
}
```


Události z buňky

```
//
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    //
    let cell = tableView.dequeueReusableCell(withIdentifier: "cellSegm") as!
MySwitchCell

    //
    cell.textik?.text = "Nova \(indexPath)"
    cell.swi.setOn(false, animated: false)
    cell.delegate = {
        // kontext closure s pristupem na cell, indexPath, ...
        print("Index \(indexPath) udalost")
    }

    //
    return cell
}
```

Heterogenní tabulky

- DataSource chce pro každou indexPath jiný prototyp buňky.
 - tableView...cellForRowAt: může být značně košatá,
 - strukturované programování,
 - oddělené dataSource pro různé sekce.
- Formuláře s pevně stanovenou strukturou:
 - statické tableView. Prototypové buňky.

Další studium UIKitu

- Pickers — různé Views pro zadání hodnoty.
Alerts
- Množství knihoven pro přístup na uživatelská data (obrázky, hudba, kalendář, poznámky).
- Moderní koncepty UI (side-menu).

Příště

- Umíme programovat uživatelské rozhraní iOS aplikací (tu snadnější část tvorby aplikací).
- Closures.
- Vlákna, operace, fronty.
- Grand Central Dispatch.