Università degli studi dell'Aquila

Introduzione allo sviluppo di applicazioni su dispositivi iOS

Ing. Tarquini Francesco Ph.D - Ing. D'Errico Leonardo - Ing. Franchi Fabio

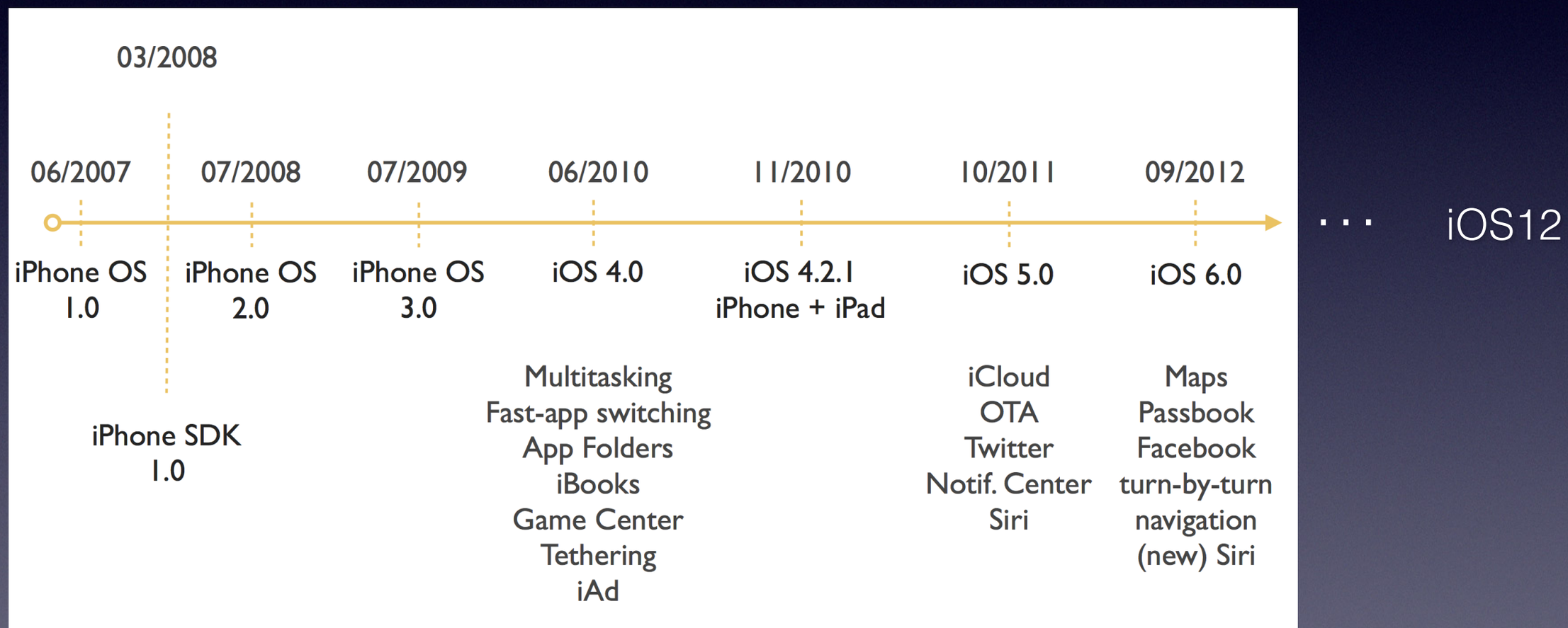
Di cosa parliamo?

- Introduzione
- Requisiti
- Apple Developer Program
- Device
- SDK
- Risorse

Di cosa parliamo?

- Design Patterns
- Xcode: introduzione alle funzioni base
- Sviluppiamo la nostra prima App:
  - AppDelegate
  - UIViewController
  - UINavigationController
  - UITableViewController

Timeline


03/2008
06/2007
07/2008
07/2009
06/2010
11/2010
10/2011
09/2012
. . .
iOS12
iPhone OS 1.0
iPhone OS 2.0
iPhone OS 3.0
iOS 4.0
iOS 4.2.1 iPhone + iPad
iOS 5.0
iOS 6.0
iPhone SDK 1.0
Multitasking
Fast-app switching
App Folders
iBooks
Game Center
Tethering
iAd
iCloud
OTA
Twitter
Notif. Center
Siri
Maps
Passbook
Facebook
turn-by-turn navigation
(new) Siri

Think Different

I contro del mondo Apple

- Hardware costoso
- Unico evento ufficiale la WWDC
- Politica di update di Xcode e SDK
- Tutte le risorse sotto NDA

Think Different

I PRO del mondo Apple

- Risorse di qualità
- Strumenti di sviluppo curati e performanti
- SDK

Requisiti

Requisiti

- Apple ID (Gratuito)
- Computer Apple (iMac, MacBook, …)
- Apple Developer Program (per *distribuire a pagamento*)
- OSX 10.8+ (Mountain Lion)
- Xcode (Gratis su AppStore)
- Device iPhone, iPad, iPod Touch (per testare)

Requisiti

- Conoscere il paradigma OOP
- Conoscere il linguaggio C
- Conoscere l’Objectve-C / Swift e iOS SDK
- Conoscere le Guideline Apple

Requisiti

Per fare delle prove NON è necessario

- Possedere un device
- Acquistare un Apple Developer Program

Apple Developer Program

Apple Developer Program

- Sottoscrizione annuale che da accesso a tutte le risorse tecniche e consente di pubblicare le proprie App sulla Apple App Store
- A pagamento (~€99/anno) ed è acquistabile online sullo store Apple
- acquistabile da Aziende, Privati ed Università

Apple Developer Program

iOS Developer Program	$99/year	App Store / 100 device
iOS Developer Enterprise Program	$299/year	NO App Store / ∞ device
iOS Developer University Program	free	NO App Store / 200 person

Apple Developer Program

- **Member Center**: gestione utenti, sottoscrizioni e rinnovi
- **iOS Provisioning Portal**: gestione certificati, device abilitati, app
- **iTunes Connect**: app submission, sales&trends, pagamenti

Concetti

- **Certificati**: Identificano univocamente un *“attore”*
- **Devices**: Elenco dei dispositivi abilitati al testing
- **APP IDs**: Identificativi univoci delle app
- **Provisioning**: “Entità” che raggruppa i tre concetti precedenti

Device

Device

- iPhone
- iPad
- iPod Touch

SDK

iOS

- **OS Mobile**: architetture ARM, multitasking, CocoaTouch, multitouch GUI
- **Kernel XNU**: sviluppato inizialmente da NeXT

iOS SDK

- Librerie e strumenti di sviluppo
- Prima release Marzo 2008
- Legata strettamente all ambiente Xcode e OSX

Strumenti

- **Xcode**: l’ambiente di sviluppo delle applicazioni iOS e OSX
- **Simulator**: Lo strumento di test e di Debug
- **Profiles**: Applicazione per l’analisi e la profilazione del codice

Xcode

- Ambiente di sviluppo (IDE)
- Swift / Objective-C / Objective-C++ / C / C++
- Source Control (SVN, GIT)
- Interface Builder

Simulator

- iPhone / iPhone retina / iPad / iPad retina
- gesture, orientation
- GPS
- Apps: Safari, Immagini, Contatti, Impostazioni, Game Center, Edicola

Just do it!

Apriamo Xcode e compiliamo...

…ma prima un po’ di
Ingegneria del Software!

Design Patterns

“In informatica, nell'ambito dell'ingegneria del software, un design pattern (traducibile in lingua italiana come schema progettuale, schema di progettazione, schema architetturale), è un concetto che può essere definito "una soluzione progettuale generale a un problema ricorrente". Si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale.”

http://it.wikipedia.org/wiki/Design_patterns

Design Patterns

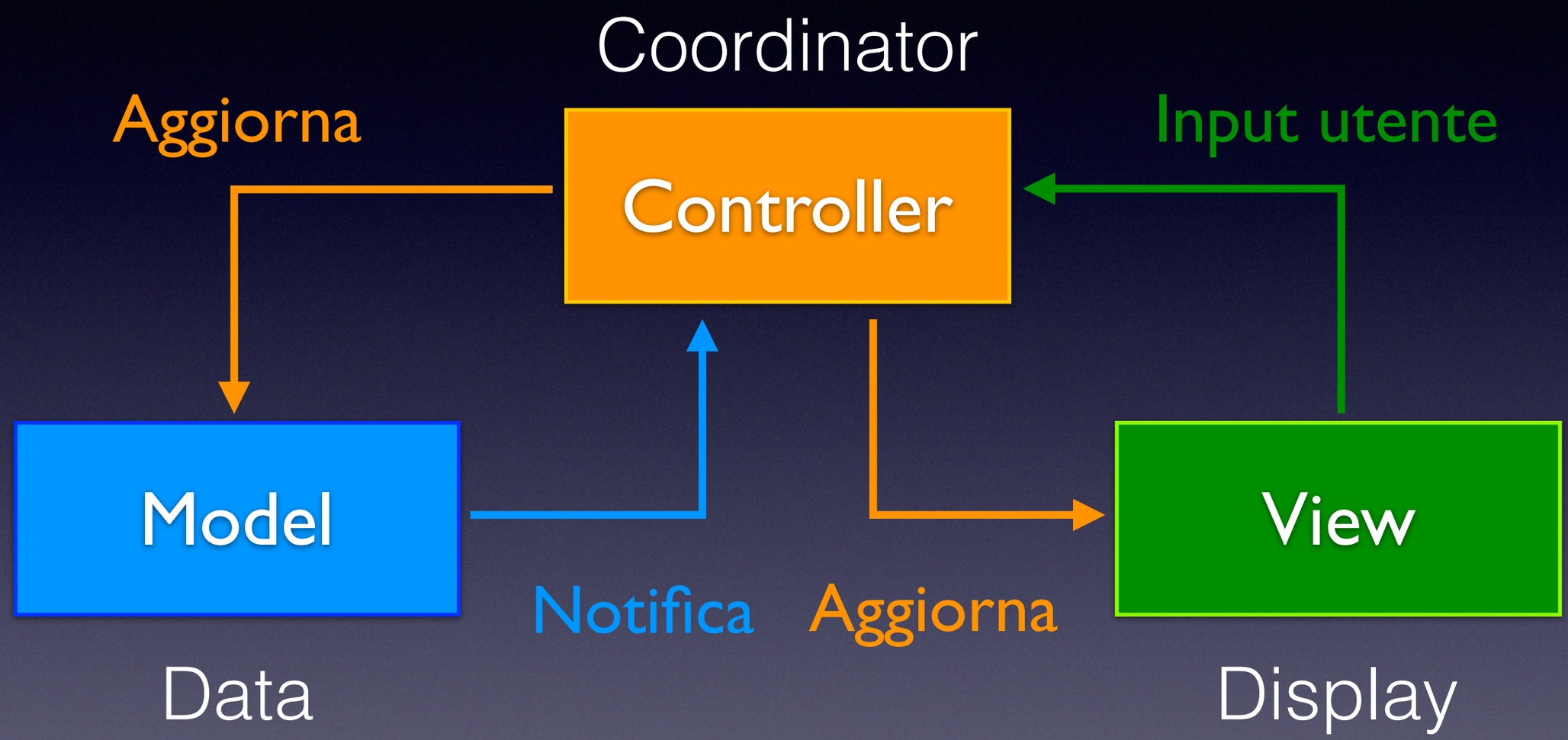
Nello sviluppo di applicazioni si utilizzano i più comuni Design Patterns:

- **Creazionali**: Singleton, Factory
- **Strutturali**: MVC, Decorator, Adapter, Facade, Composite
- **Funzionali**: Observer, Memento, Chain of responsibility, Command

MVC

The King of Design Patterns

MVC



Il design pattern MVC assegna ad ogni oggetto un ruolo preciso: Model, View o Controller

Model

- Descrive il modello dei dati utilizzato dalla nostra applicazione
- Definisce la logica con cui vengono manipolati questi dati: salvataggio, modifica, eliminazione

View

E ciò che l’utente vede:

- sa come disegnarsi
- risponde alle azioni dell’utente

Controller

E' l'intermediario tra una o più View e uno o più Model

- interpreta le azioni dell'utente eseguite nella View e comunica al Model eventuali cambiamenti da apportare
- segnala alla View le variazioni ai dati compiute dal Model
- gestisce il ciclo di vita degli altri oggetti

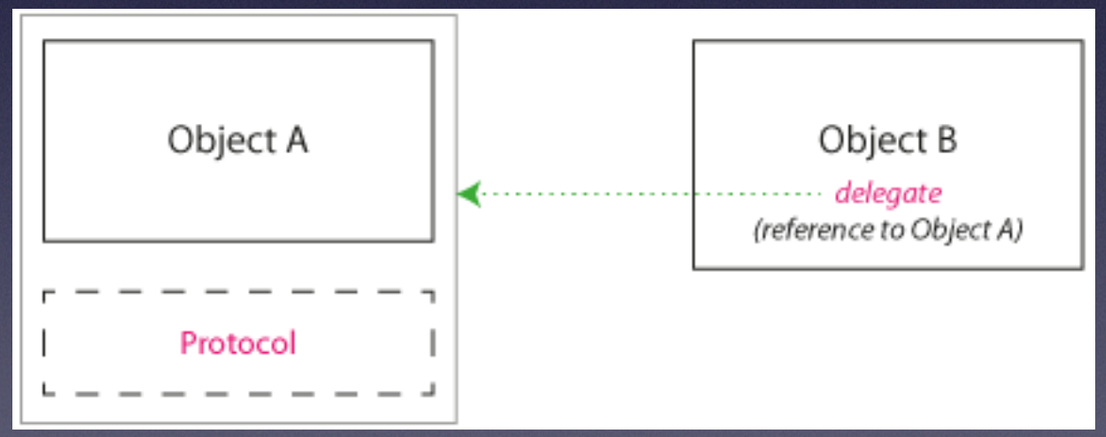
Best Practice

1. Utilizziamo l'MVC per dividere il lavoro
2. Impariamo a conoscere il Framework: **non reinvestiamo nulla**
3. Non abusiamo delle view
4. Minimizziamo le dipendenze mutuali
5. Dividiamo il codice in moduli

Delegation

La *delegazione* è un meccanismo attraverso il quale un oggetto opera assieme ad un'altro oggetto

- un oggetto chiama il suo unico delegato quando si verificano determinati eventi
- dal punto di vista del delegato è una *callback*
- dal punto di vista del delegante è uno scarico di responsabilità



Gestione della memoria

- La gestione della memoria sui dispositivi mobile è fondamentale
- E' necessario appena possibile liberare quanta più memoria possibile
- Ogni applicazione iOS deve gestirla correttamente
- MRC (Manual Reference Counting) e ARC (Automatic Reference Counting)

Swift

Concetti Fondamentali

Classi e Strutture

Le **classi** e le **strutture** sono costrutti generali che sono alla base del codice di un programma swift.

E' possibile definire proprietà e metodi per aggiungere funzionalità alle classi e alle strutture utilizzando la sintassi base swift.

A differenza di Obj-C in swift non è necessario definire differenti file per l'interfaccia e l'implementazione ma tutto avviene in un unico file *.swift*

Classi e Strutture

In swift le classi (**class**) e le strutture (**struct**) sono molto simili e differiscono solo per qualche dettaglio:

- Per le classi è possibile sfruttare l'ereditarietà
- Per le classi è possibile effettuare il *cast*
- Le *struct* sono <u>sempre</u> passate per valore, mentre le *class* sono <u>sempre</u> passate per riferimento

Dichiarare classi e strutture

dichiarazione classe

```
class SomeClass {
    // class definition goes here
}
struct SomeStructure {
    // structure definition goes here
}
```

nome della struttura

nome della classe

dichiarazione struttura

Struttura di un metodo

nome del metodo

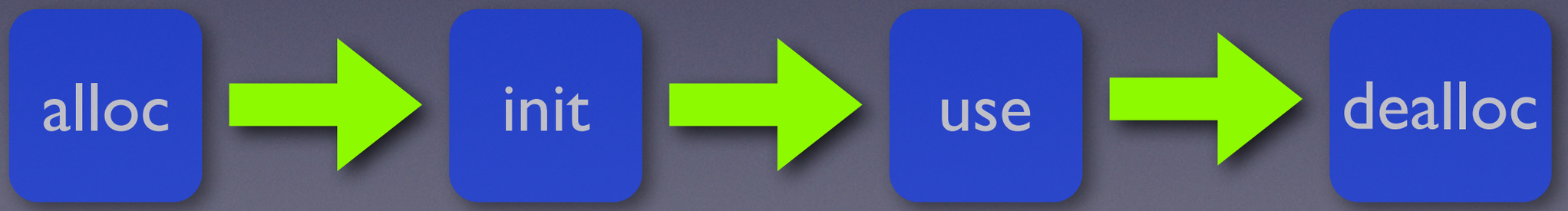
```
func eseguiMetodo (param: String) -> String;
```

tipo di ritorno

Oggetti e memoria

La creazione di un oggetto implica l'occupazione di memoria che successivamente dovrà essere liberata.

Possiamo riassumere il **ciclo di vita di un oggetto** nel seguente modo:

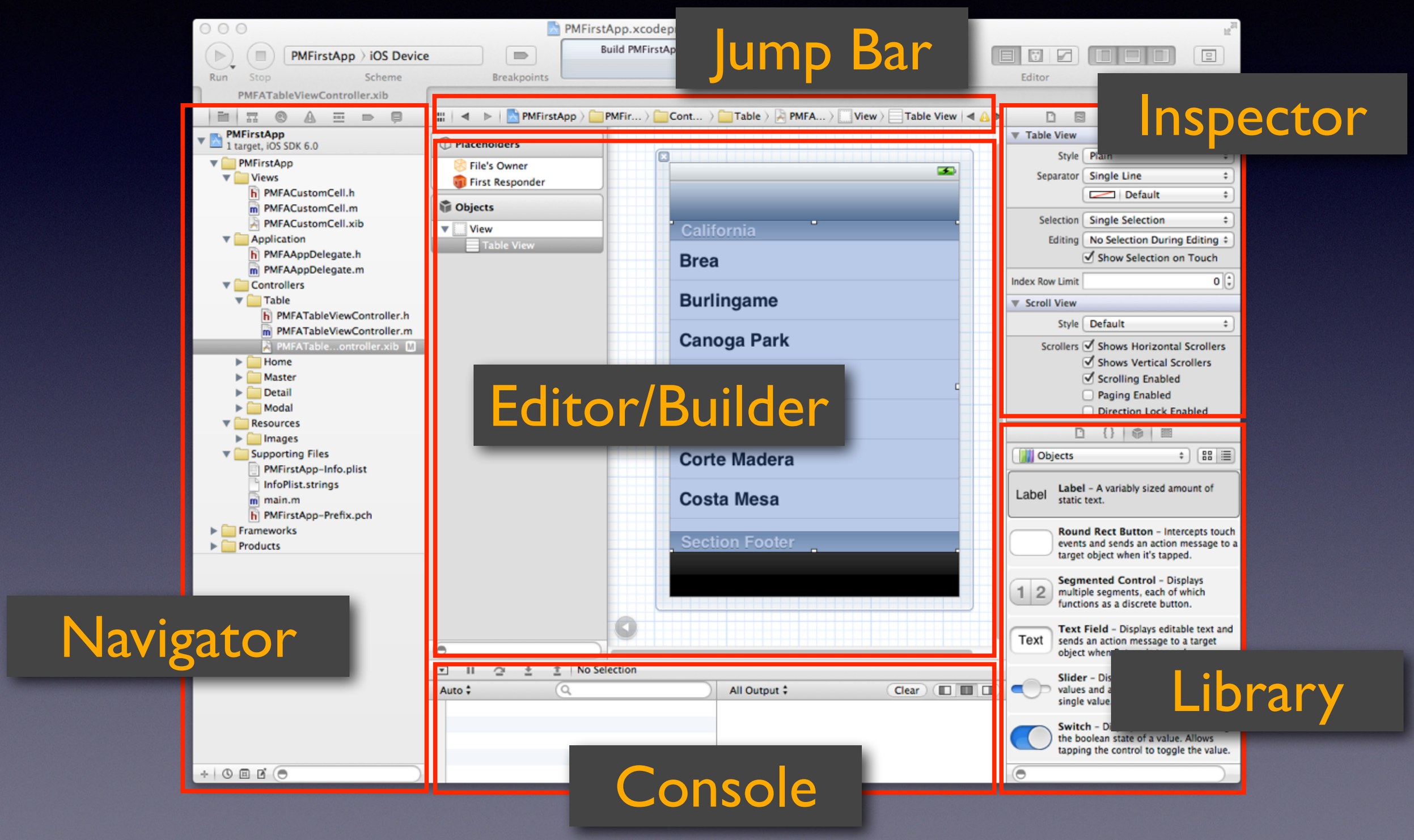


init & dealloc

La creazione di un oggetto avviene attraverso l'invocazione di un particolare metodo chiamato costruttore. Non è necessario implementare questo metodo in quanto ogni oggetto ha un **costruttore di default** definito implicitamente.

```
let my_object = MyClass();
```

Xcode

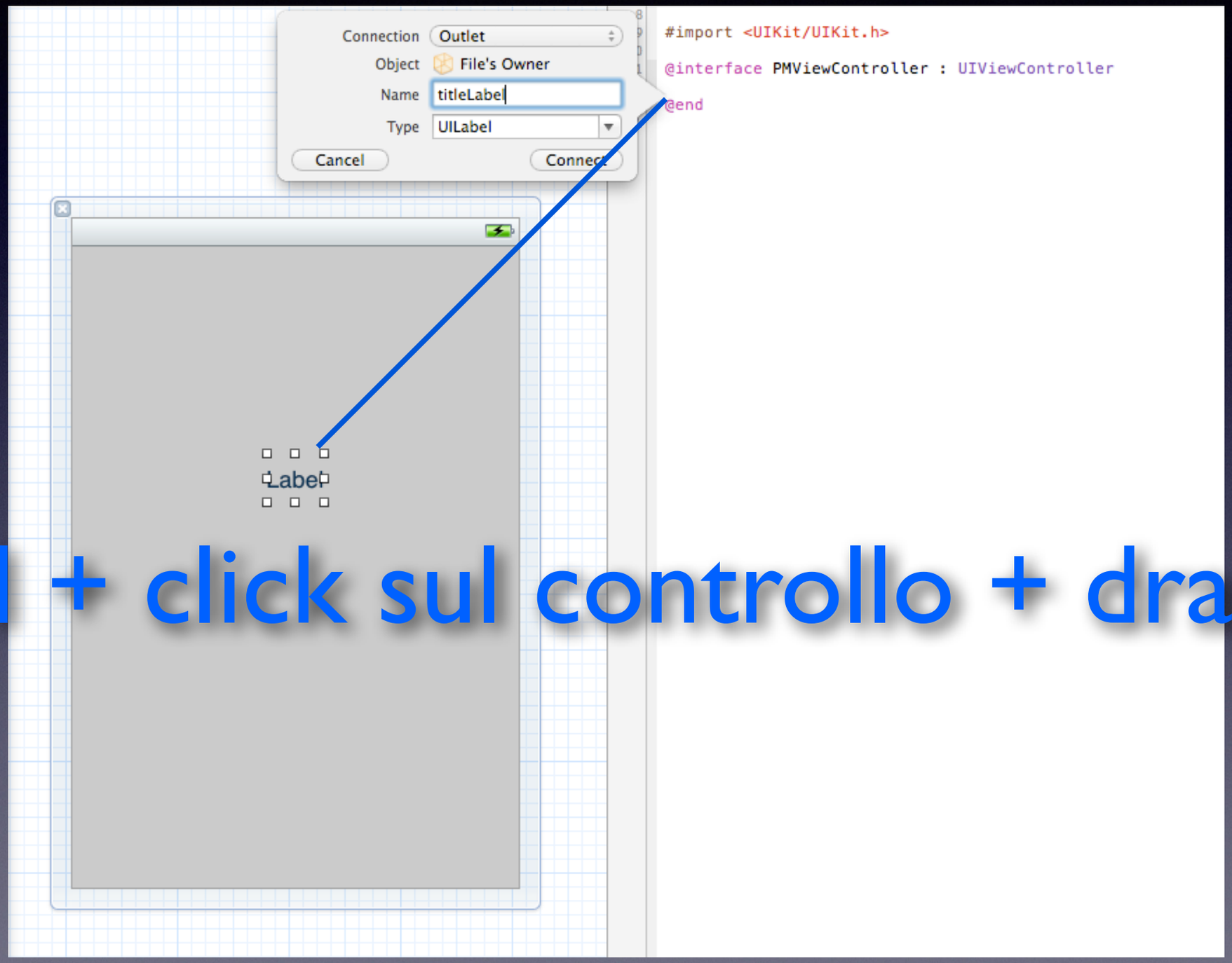

Jump Bar
Inspector
Editor/Builder
Navigator
Library
Console
PMFirstApp.xcodep
PMFirstApp 〉 iOS Device
Run
Stop
Scheme
Breakpoints
Build PMFirstAp
Editor
PMFATableViewController.xib
PMFirstApp 〉 PMFir... 〉 Cont... 〉 Table 〉 PMFA... 〉 View 〉 Table View
PMFirstApp
1 target, iOS SDK 6.0
PMFirstApp
Views
PMFACustomCell.h
PMFACustomCell.m
PMFACustomCell.xib
Application
PMFAAppDelegate.h
PMFAAppDelegate.m
Controllers
Table
PMFATableViewController.h
PMFATableViewController.m
PMFATable...ontroller.xib
Home
Master
Detail
Modal
Resources
Images
Supporting Files
PMFirstApp-Info.plist
InfoPlist.strings
main.m
PMFirstApp-Prefix.pch
Frameworks
Products
Placeholders
File's Owner
First Responder
Objects
View
Table View
California
Brea
Burlingame
Canoga Park
Corte Madera
Costa Mesa
Section Footer
Table View
Style
Separator
Single Line
Default
Selection
Single Selection
Editing
No Selection During Editing
Show Selection on Touch
Index Row Limit
0
Scroll View
Style
Default
Scrollers
Shows Horizontal Scrollers
Shows Vertical Scrollers
Scrolling Enabled
Paging Enabled
Direction Lock Enabled
Objects
Label
Label – A variably sized amount of static text.
Round Rect Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control – Displays multiple segments, each of which functions as a discrete button.
Text
Text Field – Displays editable text and sends an action message to a target object when
Slider – Di
values and a
single value
Switch – Di
the boolean state of a value. Allows tapping the control to toggle the value.
No Selection
Auto
All Output
Clear

IBOutlet

```
@IBOutlet var textFields: UITextField!
```

- Rappresenta una variabile d’istanza che referenzia un’altro oggetto
- Questa referenza è configurata ed archiviata attraverso l’Interface Builder

Connettiamo



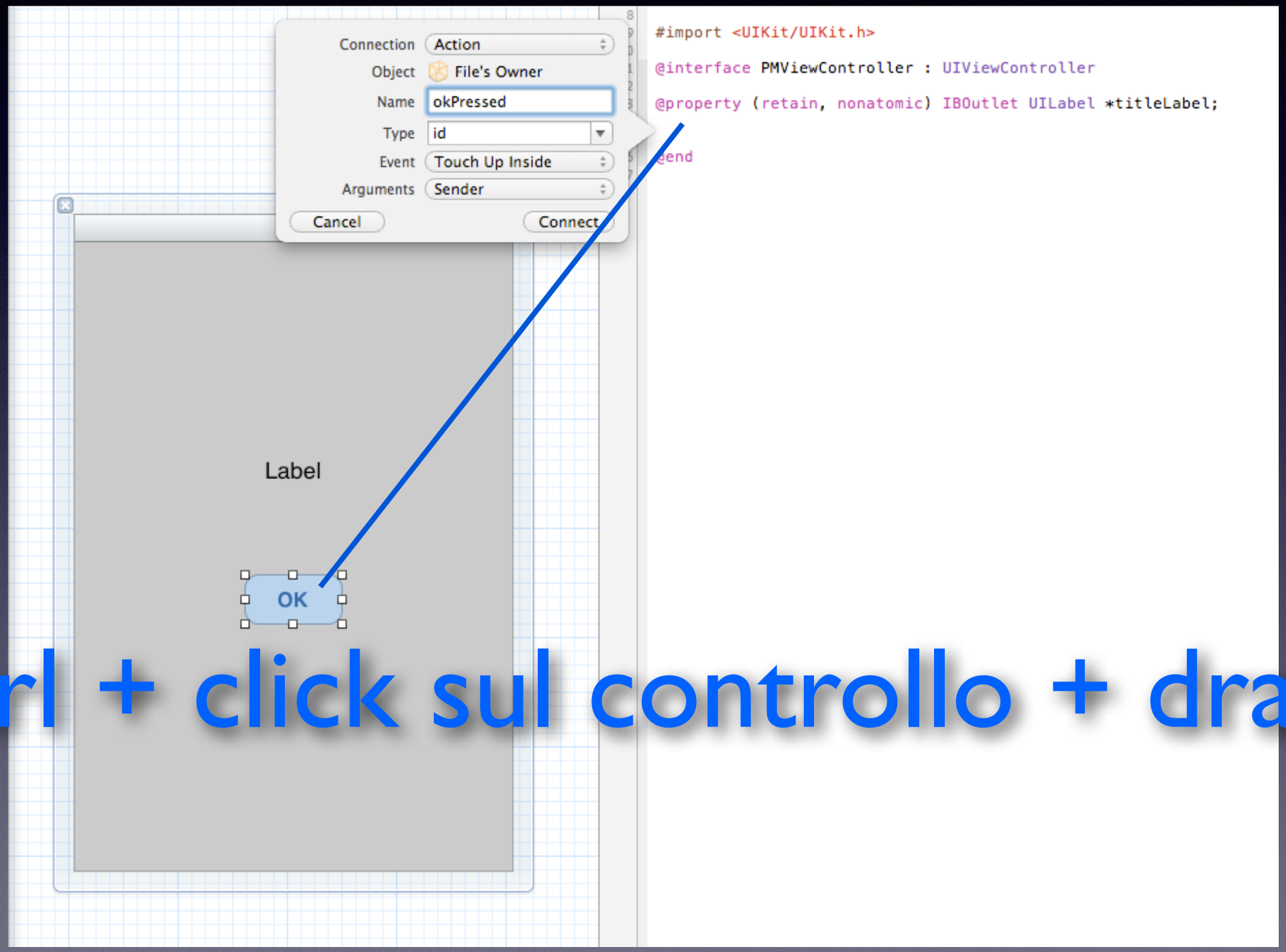
ctrl + click sul controllo + drag

IBAction

```
@IBAction func doSomething:(sender UIButton);
```

- Dichiara che il metodo è una “Action”. In questo modo IB, come per gli IBOutlet, è consapevole della sua presenza e può collegarlo ad un evento
- Il parametro `sender` è colui che manda il messaggio

Connettiamo



ctrl + click sul controllo + drag

La prima App...

dalla teoria alla pratica

AppDelegate Start & End

```
func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject:
AnyObject]?) -> Bool {
 // Il sistema notifica alla nostra applicazione che la fase di
 caricamento è terminata
 return true
}

func applicationWillTerminate(application: UIApplication) {
 // Il sistema notifica alla nostra applicazione che l'utente o
 il sistema stesso ha richiesto la chiusura della nostra
 applicazione. Possiamo usare questo metodo per salvare lo
 status della nostra applicazione.
}
```

Enter in Background

```
func applicationWillResignActive(application: UIApplication) {
{
// Il sistema notifica alla nostra applicazione che sta per
diventare inattiva. Disabilitare Timer, mettere in pausa i
Task, ...
}

func applicationDidEnterBackground(application: UIApplication)
{
// Se la nostra applicazione supporta il background, questo
metodo viene chiamato al posto del applicationWillTerminate.
Utilizzate questo metodo per rilasciare le risorse, salvare i
dati, invalidare i timers...
}
```

```
}
```

Enter in Foreground

```
func applicationWillEnterForeground(application: UIApplication)
{
 // Il sistema notifica che l’applicazione sta passando dallo
stato di background a quello di inattiva. In questo metodo è
possibile ripristinare la maggior parte delle cambiamenti
effettuati prima di entrare in background.
}
func applicationDidBecomeActive(application: UIApplication)
{
 // Il sistema notifica che l’applicazione è tornata attiva.
Utilizzare questo metodo per far ripartire i vari task attivi,
timers, eventualmente per effettuare un refresh
dell’interfaccia utente...
}
```

```
}
```

UIViewController

- Un UIViewController è controller generico utile per manipolare le UIView.
- Mette a disposizione diversi metodi che sono chiamati quando una vista appare o scompare.

Quale transizione utilizzare?

```
//    UIModalTransitionStyle:
//
//    UIModalTransitionStyleCoverVertical,
//    UIModalTransitionStyleFlipHorizontal,
//    UIModalTransitionStyleCrossDissolve,
//    UIModalTransitionStylePartialCurl,
```

Utilizziamo il ViewController

Presentiamo in modale il ViewController

```
self.presentViewController(viewControllerToPresent: UIViewController, animated:Bool, completion: (() -> Void)?)
```

Rimuoviamo il ViewController

```
self.dismissViewControllerAnimated(flag: Bool, completion: (() -> Void)?)
```

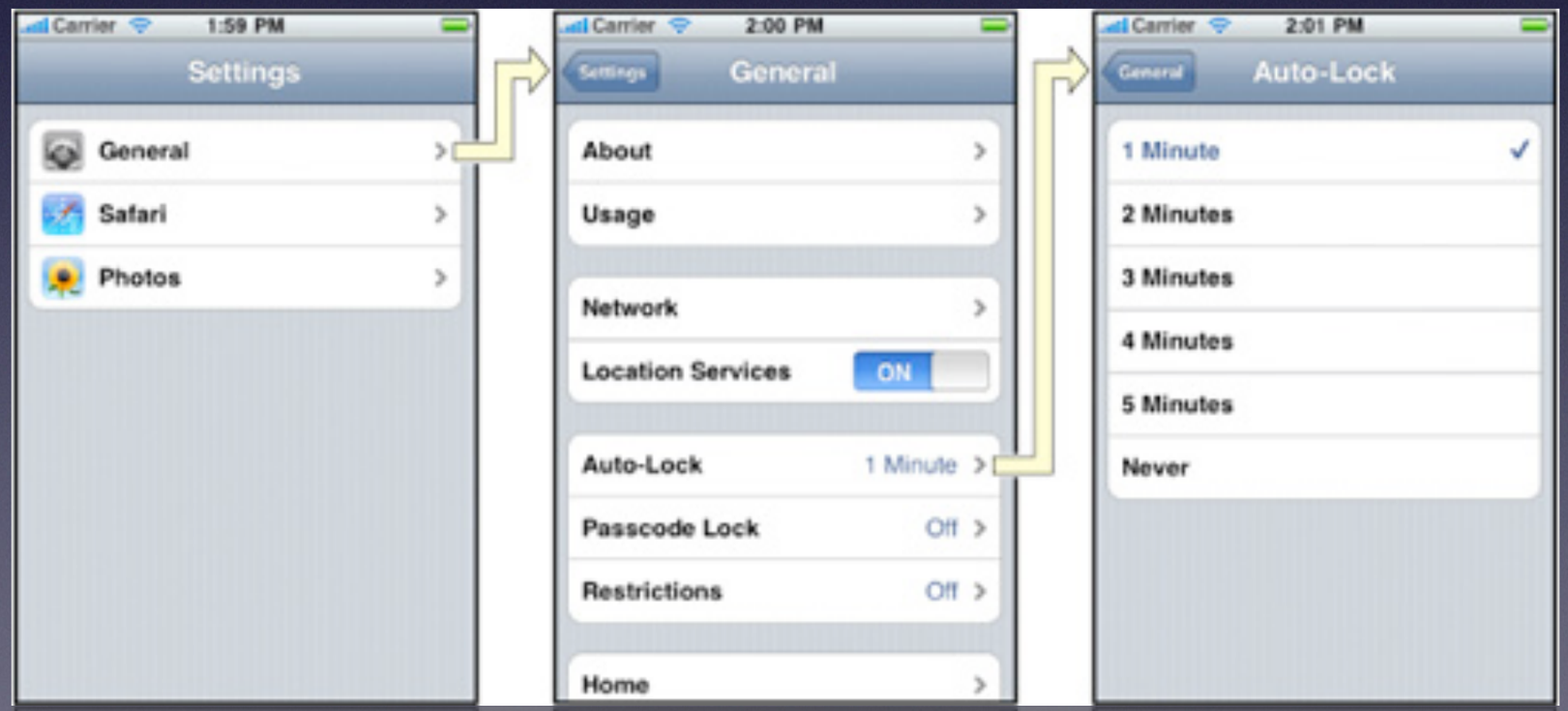
Appear/Disappear

Gestione visualizzazione view

- `viewWillAppear:(animated: Bool)`
- `viewDidAppear:(animated: Bool)`
- `viewWillDisappear:(animated: Bool)`
- `viewDidDisappear:(animated: Bool)`
- `viewDidLoad`

UINavigationController

Estende la classe UIViewController implementando la navigazione, semplice e veloce, tra ViewControllers



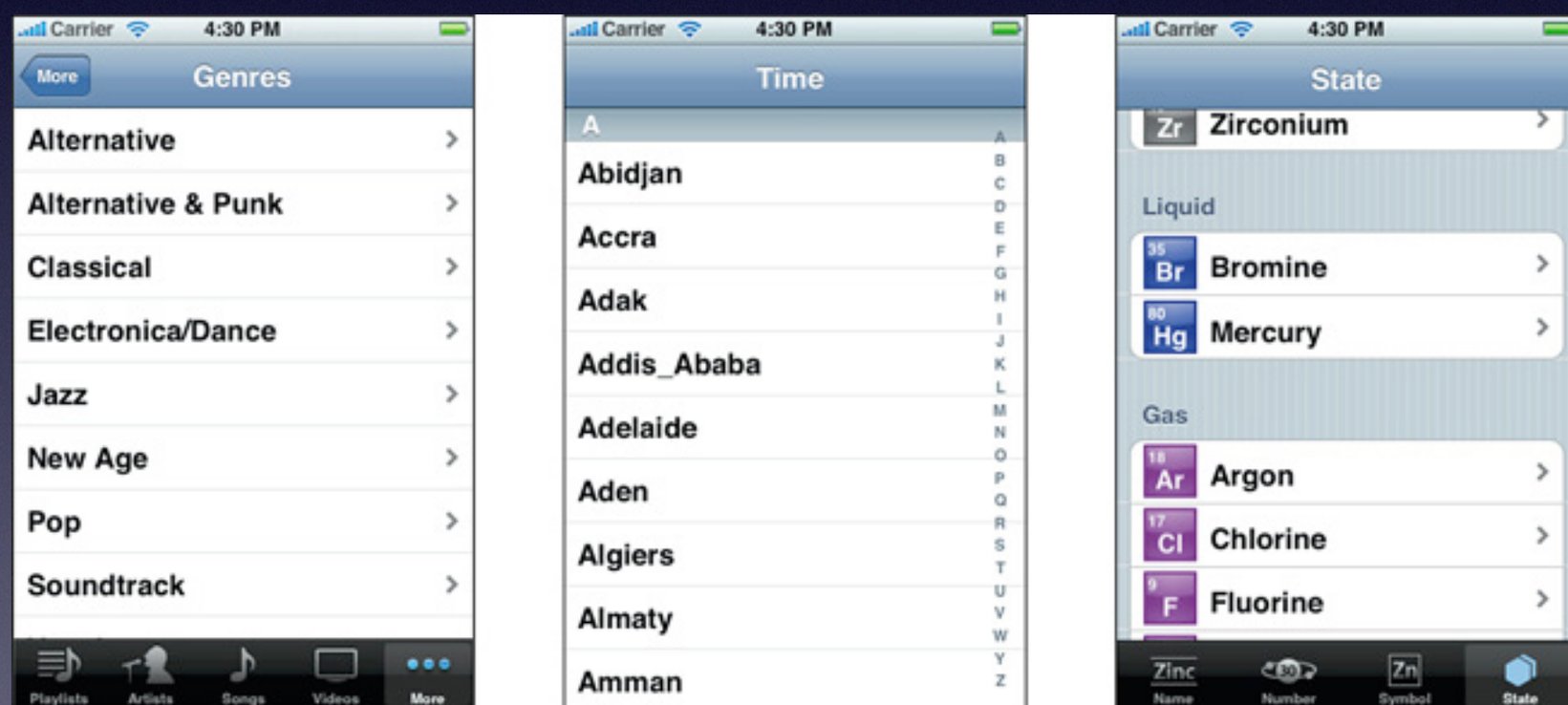
Push & Pop

Navigazione tra ViewControllers:
Gestione visualizzazione view

- `pushViewController`
- `popToViewController`
- `popToRootViewControllerAnimated`

UITableView

Ecco qualche esempio di TableView:



UITableView

- UITableViewController

oppure

- UIViewController

- UITableView

- implementare correttamente UITableViewDataSource (rappresentazione dei dati) e il UITableViewDelegate (visualizzazione contenuto celle)

UITableViewDataSource

Obbligatori

```
public func tableview(tableView: UITableView, numberOfRowInSection section:Int) -> Int
public func tableview(tableView: UITableView, cellForRowAtIndexPath indexPath:NSIndexPath) -> UITableViewCell
```

Opzionali

- ...

Demo

Ing. Tarquini Francesco Ph.D
Socio fondatore BiTE s.r.l.
e-mail: francesco.tarquini@univaq.it

Ing. D'Errico Leonardo
Socio fondatore BiTE s.r.l.
e-mail: leonardo@bitesrl.it

Ing. Franchi Fabio
Socio fondatore BiTE s.r.l.
e-mail: fabio@bitesrl.it