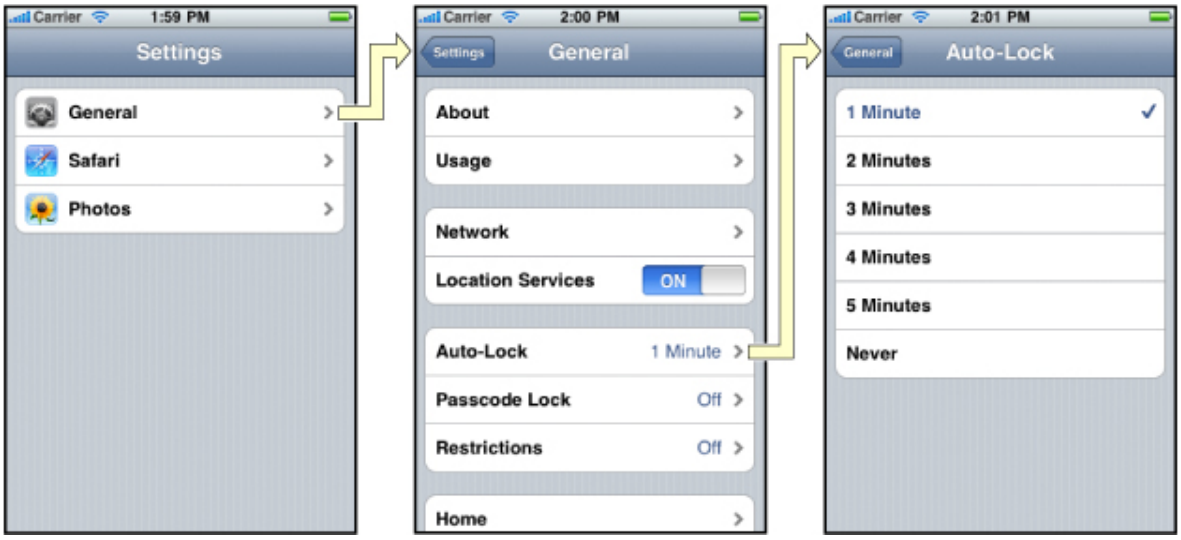
Using UINavigationController for fun and profit:
Costruiamo un menu di navigazione personalizzato con UINavigationController

Costantino Pistagna <costantino.pistagna@gmail.com>

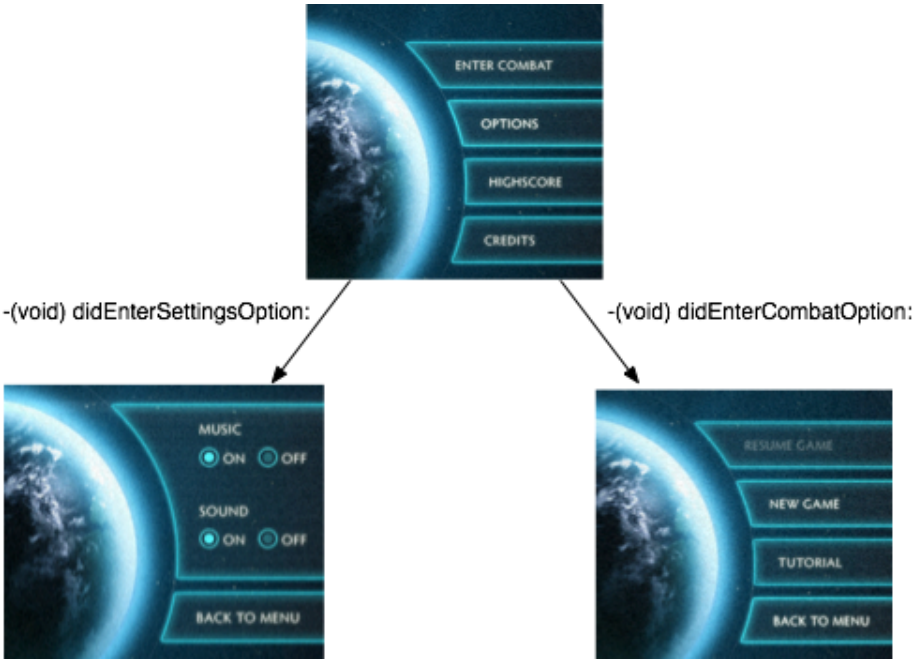
Una delle richieste che mi arriva più spesso è quella di gestire un'applicazione che utilizza più UIViewControllers. Detto in altri termini, come costruire un menu di navigazione tra piu' controllori resistente, riutilizzabile e privo di memory leaks. In quest'articolo vedremo come utilizzare uno degli oggetti già presenti nell'UIKit per ottenere il nostro scopo.

Un po' di teoria

UINavigationController è un oggetto fornito in bundle con UIKit framework che permette di navigare gerarchicamente e con poco sforzo per il programmatore una serie di UIViewControllers che risultano arrangiati secondo uno schema padre->figlio->nipote come quello rappresentato sotto.



Tipicamente, il suo utilizzo e' quello di fornire una barra di navigazione in alto che permetta di effettuare le operazioni di *push* e *pop* dei viewcontrollers figli, a partire da un elemento radice. Tuttavia, come vedremo in quest'articolo, è possibile manipolare l'oggetto UINavigationController per svolgere il suo comportamento in maniera diversa da quella standard; senza, cioè, l'ausilio di una barra di navigazione mantendendo, al contempo, tutte le sue funzionalità di controllore. Quello che vorremo ottenere alla fine del nostro articolo è qualcosa di questo tipo:

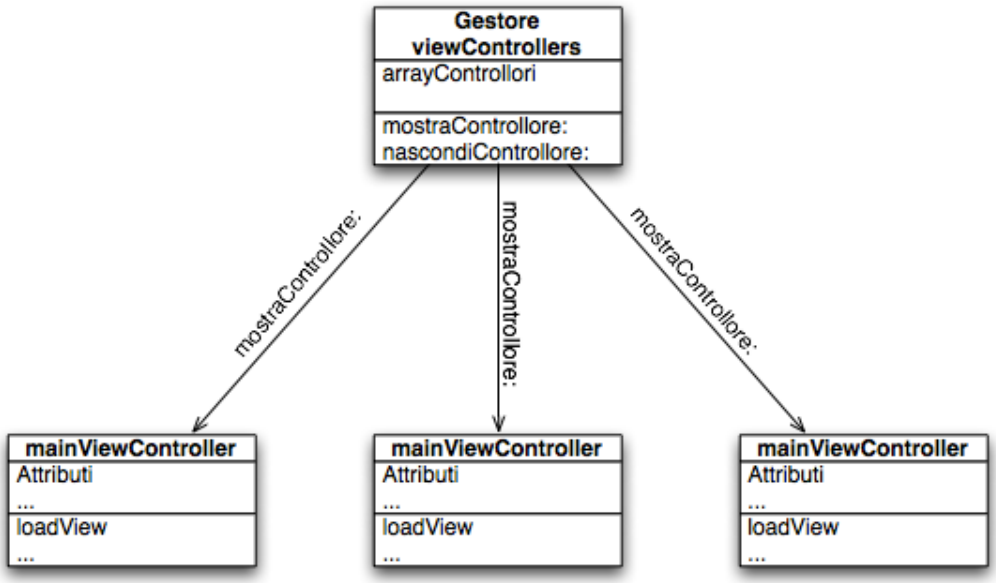


Come si puo' vedere dalla figura sopra, il menu principale della nostra applicazione prevede scelte multiple che riportano l'utente a viewControllers sempre differenti. Uno per ogni task.

Implementare questa funzionalità è, forse, uno dei task più tediosi e soggetti a memory leaks durante la scrittura di un applicativo. Il problema è di natura concettuale e risiede nel fatto che per gestire un insieme di viewControllers, deve esserci "qualcuno" o "qualcosa" che abbia una visione d'insieme. Qualcuno, cioè, che conosca chi è il viewController padre, chi sono i viewControllers figli, nasconda di volta in volta il viewController che esce di scena, si occupi di rimuovere dalla memoria gli oggetti non più utilizzati e molto altro.

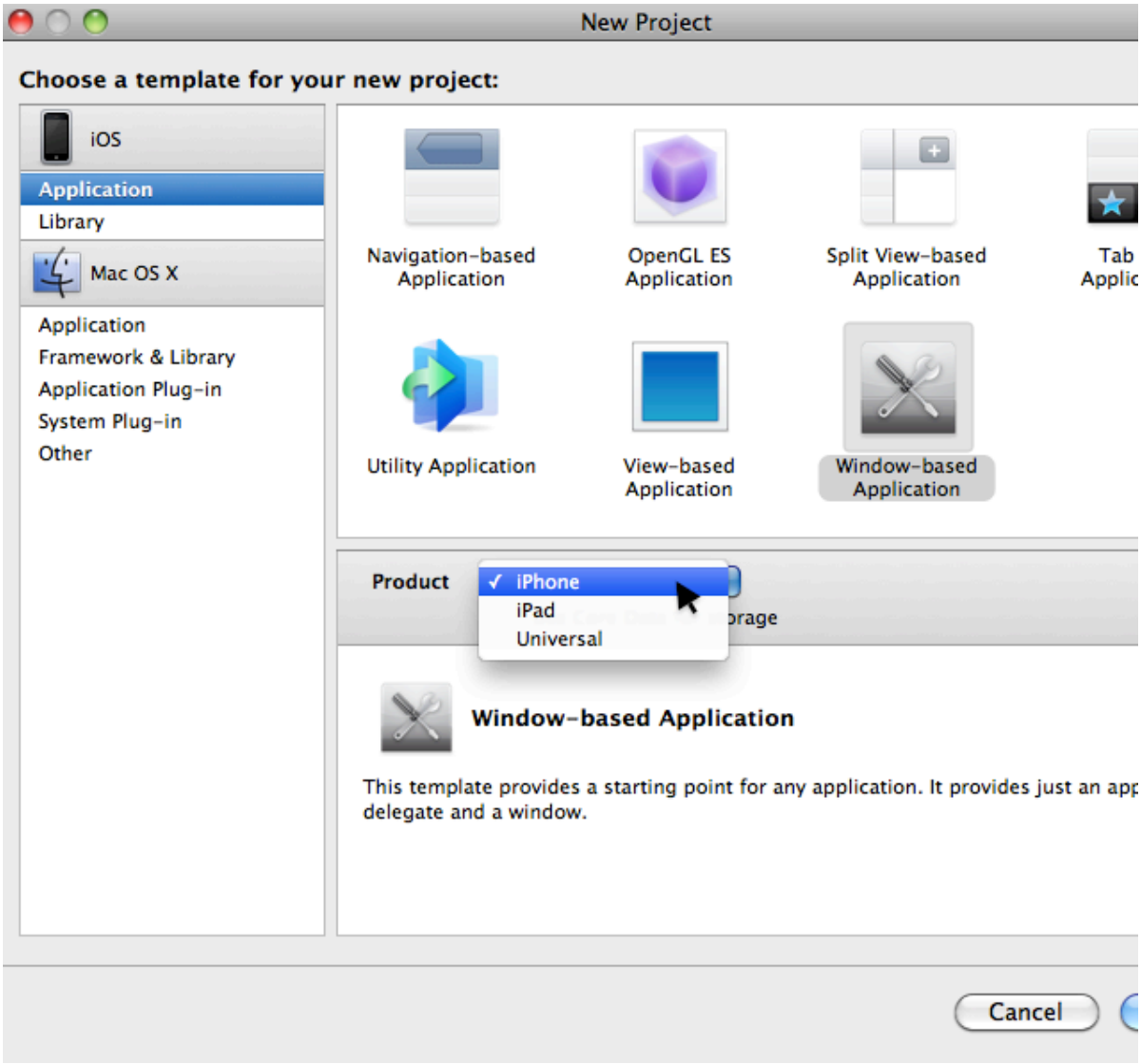
Uno degli errori tipici che si commette è quello di delegare questo lavoro alla classe AppDelegate. Niente di più sbagliato. Come descrive la documentazione developer e come ogni buon programmatore dovrebbe sapere, l'AppDelegate è una classe che poco c'entra con le rappresentazioni "visive" dei nostri oggetti e dei nostri controllori. Piuttosto, essa dovrebbe occuparsi di gestire eventi di sistema ed interazione da e verso i sensori del nostro apparato. Assegnargli un compito relativo alla mera visualizzazione di viewControllers e relative views a schermo è, quantomeno, off-topics. Sarebbe come chiedere all'autista del bus 405 di fermarsi a comprare il pane, solo perchè si trova a passare davanti ad un panificio.

La soluzione ideale è quella di costruire un'oggetto controllore che piloti lo scambio dei viewControllers e tenga traccia di chi è presente e quando lo è. Un lavoro non indifferente. Apple, ci ha messo a disposizione l'oggetto UINavigationController che, come dice il nome stesso, si occupa di gestire la navigazione sequenziale di due o più viewControllers. L'oggetto si occupa anche della dismissione dallo schermo e dalla memoria (se gestiti correttamente) degli oggetti non più utilizzati. Inoltre, permette con estrema naturalezza di gestire un approccio Lazy-Load dei nostri controllori, in maniera da preservare lo stato della memoria del nostro dispositivo.

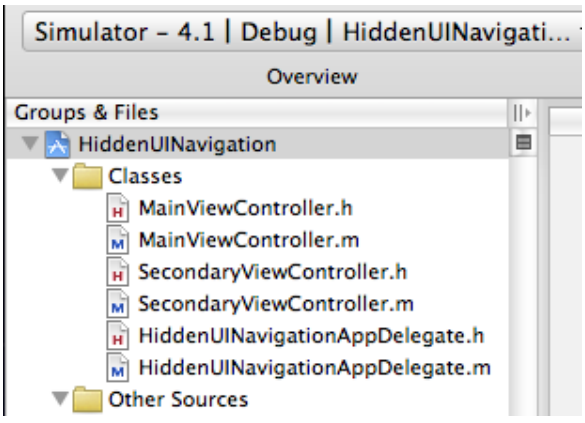


Il lavoro sopra sintetizzato può essere riassunto nel ruolo dell'oggetto UINavigationController che, non a caso, ha anche una property per abilitare o disabilitare la sua visibilità a schermo, preservando al contempo le sue funzionalità di controllore. Andiamo a vedere come operare passo per passo, in maniera da creare il nostro primo menu personalizzato.

Prima di tutto, creiamo un nuovo progetto e chiamiamolo "HiddenUINavigation":



Una volta creato il progetto ricreiamo la struttura di classi seguenti:



Oltre al nostro AppDelegate, quindi, avremo una classe MainViewController ed una SecondaryViewController. Ovviamente, per ogni nuova voce del menu che andremo a costruire, dovremo creare la corrispondente classe che eridita da UIViewController. Abbiate l'accortezza di rimuovere tutti gli XIB automaticamente creati dal template wizard

di XCode (MainWindow.xib, nel mio caso) e modificate il .plist di avvio in maniera che rifletta i cambiamenti apportati alle risorse.

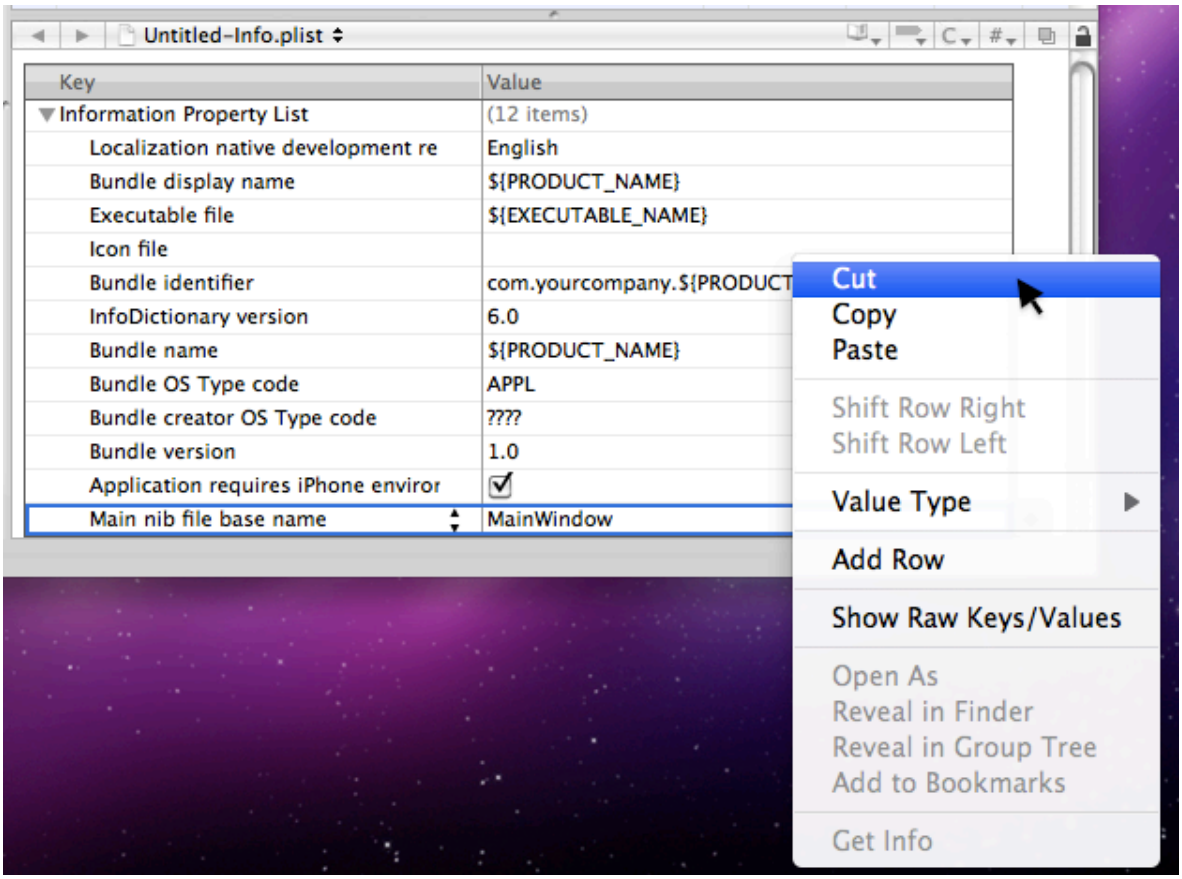
main.m

```
#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, @"HiddenUINavigationAppDelegate");
    [pool release];
    return retVal;
}
```

Per eliminare il file MainWindow.xib dall'elenco delle risorse necessarie all'avvio della vostra applicazione, dovrete selezionare nella cartella risorse, il file .plist con il nome del vostro progetto. Nel nostro caso HiddenUINavigation-Info.plist.



Una volta individuata la voce **Main nib file base name**, selezionate premendo il tasto destro del vostro mouse e scegliete l'opzione: CUT.

A questo punto, la nostra applicazione è completamente slegata dai files di risorse di interface builder. Inoltre, abbiamo inizializzato il ciclo del mainLoop con l'istanza (unica) della nostra classe AppDelegate. Non ci resta che procedere nell'implementazione di quanto sopra descritto.

Prima di tutto aggiungiamo una variabile d'istanza al nostro AppDelegate. Questa variabile ci servirà per referenziare l'oggetto UINavigationController. Questo sarà l'unico oggetto gestito dal nostro AppDelegate. Modifichiamo il nostro file di interfaccia come segue:

HiddenUINavigationAppDelegate.h

```
@interface HiddenUINavigationAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UINavigationController *tmpNavController;
}
```

Come si può notare sono stati eliminati anche i riferimenti alla property window. Essi servono solo quando utilizziamo interface builder per la gestione della prima finestra o abbiamo esigenza di referenziare in maniera diretta l'oggetto UIWindow da fuori appDelegate. Visto che abbiamo fatto tutto programmaticamente, possiamo eliminare i metodi setter e getter per l'oggetto UIWindow.

Spostiamoci nel file d'implementazione ed all'interno del metodo **didFinishLaunchingWithOptions:**, aggiungiamo la logica di inizializzazione della window e degli oggetti necessari a popolare la nostra applicazione.

HiddenUINavigationAppDelegate.m
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

    MainViewController *mainViewController = [[MainViewController alloc] init];
    tmpNavController = [[UINavigationController alloc]
initWithRootViewController:mainViewController];
    tmpNavController.navigationBarHidden = TRUE;
    [mainViewController release];

    [window addSubview:tmpNavController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

Inizializziamo il nostro oggetto mainViewController (la classe principale responsabile del menu principale, ad esempio) assegnandolo come rootViewController all'oggetto UINavigationController appena creato. Nota come il retainCounter di mainViewController, una volta assegnato alla property dell'UINavigationController, sarà stato incrementato di una unità. Possiamo quindi operare un release e lasciare che sia l'UINavigationController ad occuparsene da ora in poi. Una gestione oculata e sempre attiva della memoria è il modo migliore per evitare memory leak.
Aggiungiamo la property **view** di tmpNavController come view principale della window sopra creata. A questo punto, il compito del nostro delegato è terminato. Esso ha inizializzato correttamente la window contenitore ed i suoi oggetti principali (UINavigationController e MainViewController). L'ultimo task che dobbiamo sincerarci di compiere e' quello del rilascio di tmpNavController, ossia dell'oggetto UINavigationController. Visto che il suo compito si esaurirà solo alla fine dell'applicazione, è sensato rilasciarlo solo in fase di dealloc della nostra applicazione. Modifichiamo il metodo dealloc come segue:

HiddenUINavigationAppDelegate.m
```
- (void)dealloc {
    [tmpNavController release];
    [window release];
    [super dealloc];
}
```

Passiamo adesso alla classe controllore per il menu principale: MainViewController. Dotiamo la sua interfaccia di un oggetto UIView che utilizzeremo per renderizzare e costruire la property view. Modifichiamo il file di interfaccia come segue:

MainViewController.h
```
@interface MainViewController : UIViewController {
    UIView *contentView;
}
```

Spostiamoci, quindi, sul file di implementazione e costruiamo la nostra view in maniera programmatica, implementando il metodo **loadView**.

MainViewController.m
```
- (void)loadView {
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] applicationFrame]];
    [contentView setBackgroundColor:[UIColor redColor]];

    UIButton *tmpButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [tmpButton setFrame:CGRectMake(0, 0, 200, 60)];
    [tmpButton setTitle:@"pushViewController" forState:UIControlStateNormal];
    [tmpButton addTarget:self \
                  action:@selector(gotoSecondaryViewController) \
        forControlEvents:UIControlEventTouchUpInside];
    [contentView addSubview:tmpButton];
```

```
    [tmpButton setCenter:CGPointMake(160, 240)];
    contentView.alpha = 0.0;
    self.view = contentView;
    [contentView release];
}
```

Nulla di nuovo sotto il sole. Allochiamo ed inizializziamo un oggetto UIView che assegniamo alla property view della classe controllore. Inoltre, aggiungiamo un bottone utile ai fini del nostro esempio, per muoverci agevolmente tra un viewController e l'altro. Al bottone è associato un selettore: **gotoSecondaryViewController**.

MainViewController.m

```
- (void)gotoSecondaryViewController {
    SecondaryViewController *aController = [[SecondaryViewController alloc] init];
    [self.navigationController pushViewController:aController animated:NO];
    [aController release];
}
```

Unica responsabilità del metodo sopra è quella di allocare ed inizializzare un oggetto SecondaryViewController e *pusharlo* sullo stack dell'oggetto UINavigationController. Nota come ogni oggetto UIViewController eredita, implicitamente, una property *navigationController* che punta al nostro oggetto istanziato dall'AppDelegate.

Analogamente, l'interfaccia della classe SecondaryViewController sarà così popolata:

SecondaryViewController.h

```
@interface SecondaryViewController : UIViewController {
    UIView *contentView;
}
```

Il suo metodo loadView, nel file di interfaccia rifletterà i ragionamenti sopra esposti per la classe MainViewController:

SecondaryViewController.m

```
- (void)loadView {
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] applicationFrame]];
    [contentView setBackgroundColor:[UIColor greenColor]];
    UIButton *tmpButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [tmpButton setFrame:CGRectMake(0, 0, 200, 60)];
    [tmpButton setTitle:@"pushViewController" forState:UIControlStateNormal];
    [tmpButton addTarget:self \
                  action:@selector(gotoMainViewController)\
        forControlEvents:UIControlEventTouchUpInside];
    [contentView addSubview:tmpButton];
    [tmpButton setCenter:CGPointMake(160, 240)];
    contentView.alpha = 0.0;
    self.view = contentView;
    [contentView release];
}
```

Questa volta, il bottone piazzato al centro della view, selezionerà il metodo **gotoMainVewController**.
L'implementazione del metodo risulta immediata:

SecondaryViewController.m

```
-(void)gotoMainViewController {
    [self.navigationController popViewControllerAnimated:NO];
}
```

Questa volta, il selettore dovrà semplicemente rimuovere dallo stack dell'oggetto UINavigationController l'attuale viewController, in maniera da poter ritornare al menu principale.

A questo punto, non ci resta che compilare e provare il nostro codice. Se tutto è stato trascritto correttamente, l'applicazione dovrebbe mostrarci un sistema di navigazione a due ViewControllers (MainViewController e SecondaryViewController). Nota come il lavoro lato AppDelegate è molto semplice: esso si limita a dare voce e fare intervenire il controllore per i ViewControllers, inizializzando il tutto con il menu principale (MainViewController). Quest'ultimo, avrà l'unica responsabilità di allocare e "pushare" sullo stack dei controllori gestiti da UINavigationController la sottovoce di menu scelta dall'utente. Nel nostro caso, SecondaryViewController.

In questo modo, il nostro sistema di navigazione è resistente ad errori di allocazione della memoria ed inoltre, risulta facilmente espandibile con ulteriori sottovoci. In un prossimo articolo, vedremo come dotare questo menu di un sistema di animazione per l'entrata e l'uscita dei viewcontrollers personalizzato ed altamente modulare.