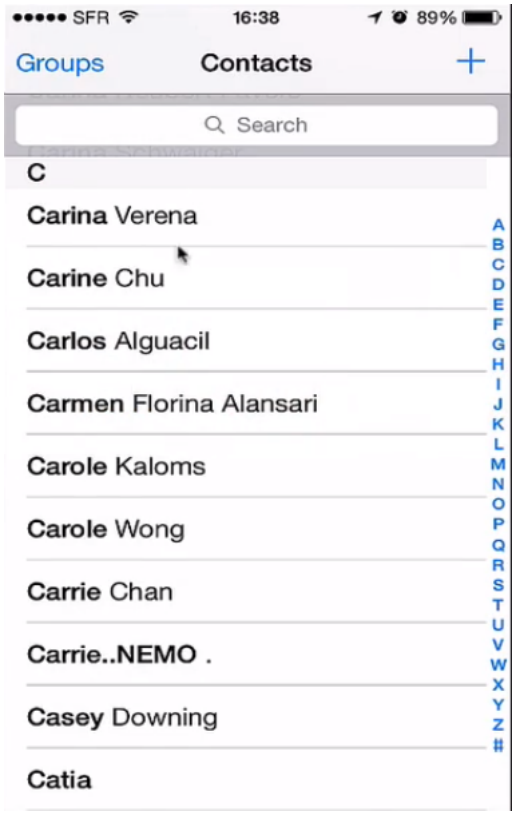
07

O TableViewController

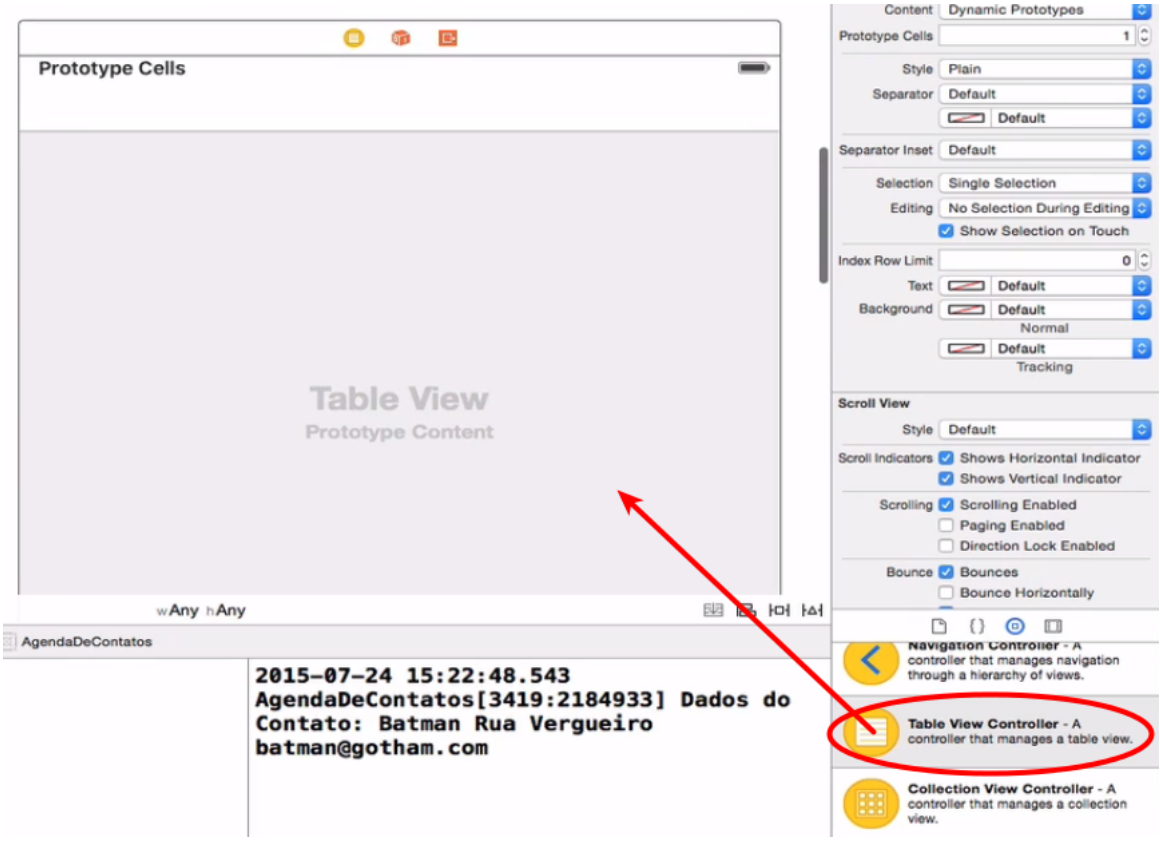
O `TableViewController`

Criando um `TableViewController`

Já temos nosso formulário pronto e conseguimos gravar as informações passadas nos campos. Precisamos, além de cadastrar, conseguir visualizar os contatos na aplicação em uma tela parecida com esta:



Já existe um componente pronto do iOS chamado **Table View Controller**. Basta arrastá-lo para dentro do ambiente de trabalho:



Porém, existem outras formas de criarmos telas no iOS, em vez de inserir tudo no Storyboard via o *Interface Builder*, vamos criar manualmente, via código, o **Table View Controller**. Essa é uma abordagem diferente, interessante de ser estudada.

Criemos uma nova Classe, também *Cocoa Touch Class*, com as seguintes informações:

- Class: **ListaContatosViewController**
- Subclass of: **NSObject**
- Language: **Objective-C**

Salvamos no mesmo diretório e teremos, da mesma forma que antes, dois arquivos: "ListaContatosViewController.h" e "ListaContatosViewController.m". Adicionamos "ViewController" no final por convenção. Vejamos ambos:

```
//
//  ListaContatosViewController.h
//  AgendaDeContatos
//
//  Created by alura on 7/24/15.
//  Copyright © 2015 alura. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface ListaContatosViewController : NSObject

@end
```

```
//
//  ListaContatosViewController.m
//  AgendaDeContatos
//
```

```
//  Created by alura on 7/24/15.
//  Copyright © 2015 alura. All rights reserved.
//

#import "ListaContatosViewController.h"

@implementation ListaContatosViewController

@end
```

Toda Classe que controla tela herda um *ViewController*. No caso do "ViewController.h", ele herdava de `UIViewController`:

```
@interface ViewController : UIViewController
```

Para nossa "ListaContatosViewController.m" existe um *ViewController* específico, que é o `UITableViewController`:

```
@interface ListaContatosViewController : UITableViewController

@end
```

E, para encontrar a Classe do `UITableViewController`, em vez de importar a biblioteca *Foundation*, importamos a parte de UI:

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

Veremos que só de colocar a Classe como filha de `UITableViewController` ela já ganhará a tabela mostrada no começo desse capítulo. Falta-nos ver se ela está realmente funcionando. Vamos, então, rodar a simulação. Ao fazer isso, cairemos na tela de preenchimento de contato, e isso acontece porque a criamos via Storyboard. As telas criadas manualmente, por código, não serão mostradas automaticamente.

A Classe que decide quais telas e em que ordem elas aparecem é o "AppDelegate.m". É a ela que a aplicação delega essa função de decidir qual tela mostrar. Abrindo este arquivo podemos ver muito código pronto, a maioria não nos interessa. Mas vamos olhar para essa parte:

```
@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishlaunchingWithOptions:(NSDictionary *)launc
    // Override point for customization after application launch.
    return YES;
}

//...
@end
```

O `didFinishlaunchingWithOptions` é que tem o trabalho de carregar aplicação. já dentro da "AppDelegate.h" podemos ver a propriedade que representa a janela

```
@property (strong, nonatomic) UIWindow *window;
```

Vamos chamá-la dentro da aplicação para pedir que a sua raiz (*root*), sua tela padrão, seja a lista de contatos:

```
- (BOOL)application:(UIApplication *)application didFinishlaunchingWithOptions:(NSDictionary *)launc
    self.window.rootViewController = ListaContatosViewController

    return YES;
}
```

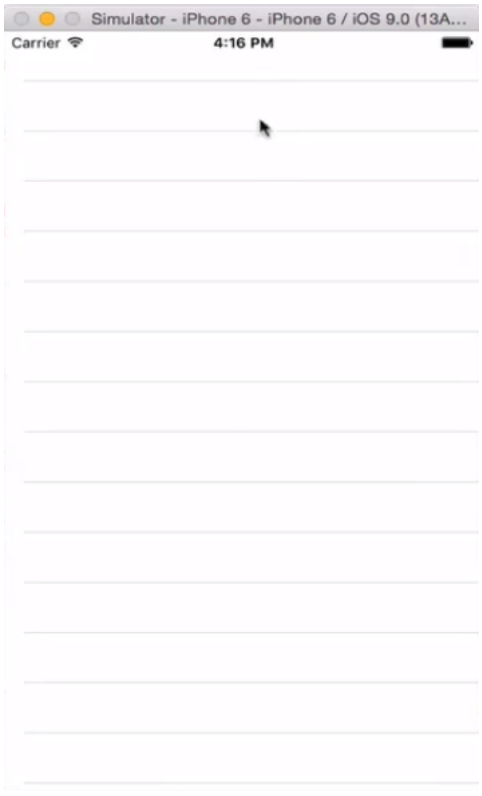
Porém, é necessário instanciar essa lista e, para tal, a importaremos:

```
#import "AppDelegate.h"
#import "ListaContatosViewController.h"
```

E, instanciamos:

```
- (BOOL)application:(UIApplication *)application didFinishlaunchingWithOptions:(NSDictionary *)launc

    ListaContatosViewController *lista = [ListaContatosViewController new];

    self.window.rootViewController = lista

    return YES;
}
```

Quando mandarmos rodar, esta tela que aparecerá:



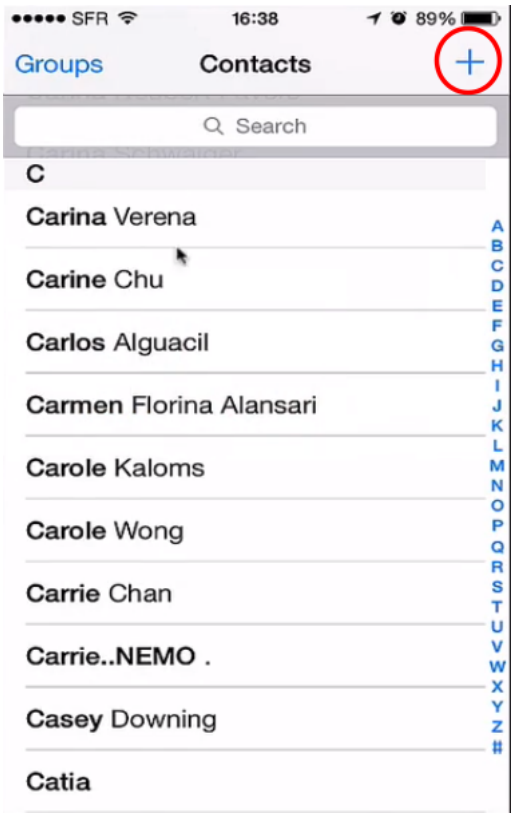
Lembrando:

- Podemos criar telas tanto pelo Storyboard, como vimos na aula passada, quanto manualmente, via código.
- Neste caso acima, indicamos qual a tela inicial através da propriedade `rootViewController` passando uma instância dele, no caso `lista`.
- Só pelo fato da Classe *ListaContatosViewController* ser um filho de `UITableViewController`, ela já ganha algumas capacidades, por exemplo, ter desenhada na tela a tabela de listagem de contatos cadastrados.

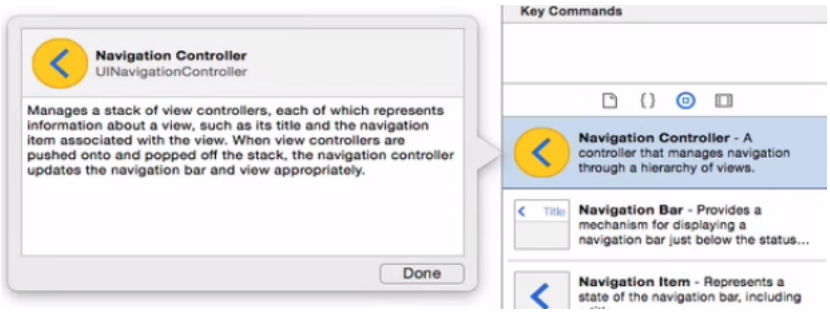
Falta agora enviar, armazenar e exibir as informações dos contatos na nossa lista.

NavigationController

Ainda não podemos navegar da tela da lista até a tela do formulário e vice-versa. É muito comum nas aplicações para iOS haver um botão no canto superior que abre outra tela, no caso, de formulário. Após processar ele volta para a tela inicial:



Perceba que ele faz parte de uma barra superior. Quem a representa é o `NavigationController`:



Da mesma forma que fizemos com o TableViewController, essa barra será implementada via código direto dentro da *AppDelegate*. A primeira coisa que faremos é instanciar o `UINavigationController` dentro do `- (BOOL)application`:

```
- (BOOL)application:(UIApplication *)application didFinishlaunchingWithOptions:(NSDictionary *)launc

    ListaContatosViewController *lista = [ListaContatosViewController new];

    UINavigationController *navLista = [UINavigationController new];

    self.window.rootViewController = lista

    return YES;
}
```

O *NavigationController* consegue navegar de uma tela para outra. Ele funciona como se fosse a janela do nosso navegador da internet: conseguimos entrar e sair e diversas páginas, mas ela continua a mesma. Ou seja, ele fica por trás da janela da lista. Para conseguirmos fazer isso, devemos indicar que sua página inicial seja a lista de contatos. Então, em vez de passarmos o `new` para ele simplesmente instanciar, passaremos informações:

```
UINavigationController *navLista = [[UINavigationController alloc] initWithRootViewController:lista]
```

Alocamos (`alloc`) o `UINavigationController` na memória e o inicializamos com a informação de qual é seu *RootView*, ou seja, a lista. O `initWithRootViewController` funciona como um construtor do *NavigationController*.

No fundo, fazer `new` nada mais é do que alocar com o `init` simples:

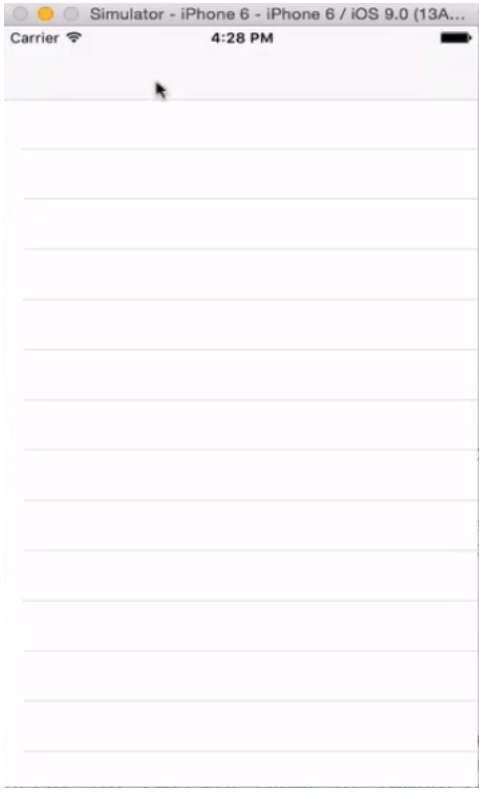
```
ListaContatosViewController *lista = [[ListaContatosViewController alloc] init];
```

É muito comum ver este tipo de comportamento em códigos antigos, pois antes não havia o `new` .

Agora a tela inicial da aplicação deve ser a navLista`:

```
self.window.rootViewController = navLista;
```

Vamos rodar novamente:



De fato, apareceu uma barra superior na aplicação!

Lembrando:

- Para criarmos um objeto sem precisar configurá-lo, usamos `new`, senão usamos `alloc` seguido de um `init`. Isso pode ser feito em duas linhas: >

```
UINavigationController *navLista = [UINavigationController alloc];
navLista = [navLista initWithRootViewController:lista];
```

Nosso próximo passo será acrescentar o botão de `+`.

`init` e `bar button item`

Quem "manda" no botão de `+` da barra é a tela de lista, pois apenas nela que faz sentido ele aparecer. No formulário podemos querer outro botão.

Essa informação do botão é passada dentro da Classe "ListaContatosViewController.m" com o método `init`, o qual retorna um objeto do tipo "ListaContatosViewController":

```
#import "ListaContatosViewController.h"

@implementation ListaContatosViewController

-(ListaContatosViewController *) init {

    return self;
}

@end
```

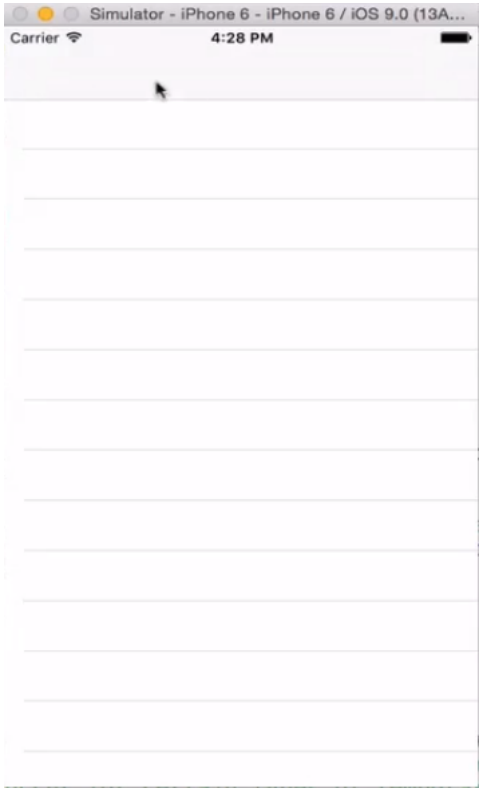
Usamos `return self` pois a resposta é para a própria lista. Porém `ListaContatosViewController` é filho de `UITableViewController`, como podemos observar em "ListaContatosViewController.h". Vimos anteriormente que uma das coisas que o `UITable` faz automaticamente é mostrar a tabela e queremos que ele continue fazendo, então vamos chamar o inicializador da Classe mãe (`super`):

```
@implementation ListaContatosViewController

-(ListaContatosViewController *) init {
    self = [super init];

    return self;
}

@end
```

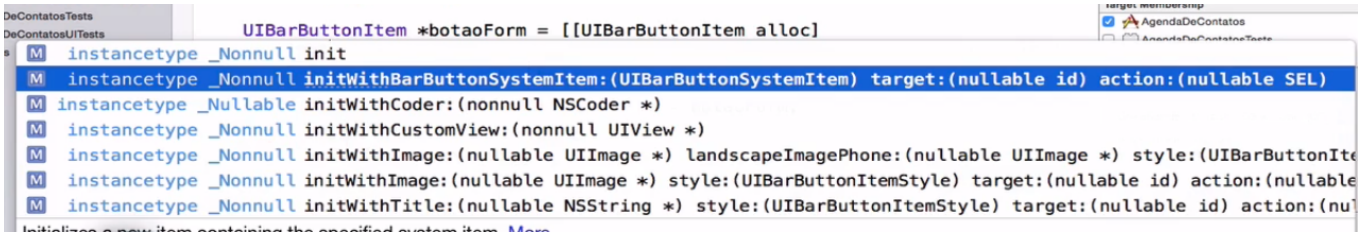
Falta indicar que queremos que apareça um botão que exibe o formulário na `NavigationController`. Mais uma vez, sendo filho de `UITable`, ele possui a propriedade `navigationItem` que é exatamente este botão, `UIBarButtonItem`, este item de navegação, colocado à direita (*right*):

```
@implementation ListaContatosViewController

-(ListaContatosViewController *) init {
    self = [super init];

    UIBarButtonItem *botaoForm = [UIBarButtonItem new];

    self.navigationItem.rightBarButtonItem = botaoForm;

    return self;
}

@end
```

Vamos rodar:



Não apareceu nada. Isso acontece porque, como fizemos tudo via código, devemos passar todas as informações do botão, como tamanho e conteúdo. Vamos configurar esse botão usando `alloc` e `init`. Um dos inicializadores é este que aparece selecionado nas opções do `init`:



É exatamente o botão `+` (`UIBarButtonSystemAdd`), que é padrão do sistema:

```
UIBarButtonItem *botaoForm = [[UIBarButtonItem alloc] initWithbarButtonSystemItem:UIBarButtonSystemA
```

Com isso não perdemos tempo configurando suas dimensões ou colocando texto.

Lembremos que ao criarmos o botão de "Adicionar" na primeira aula, fizemos diretamente no storyboard o linkando com o ViewController. Ou seja, precisamos dizer quem é o alvo do clique e qual o método a ser executado em seguida. Via código existem essas duas propriedades:

- Quem será chamado:

```
target:(nullable id)
```

- A ação que será executada:

```
action:(nullable SEL)
```

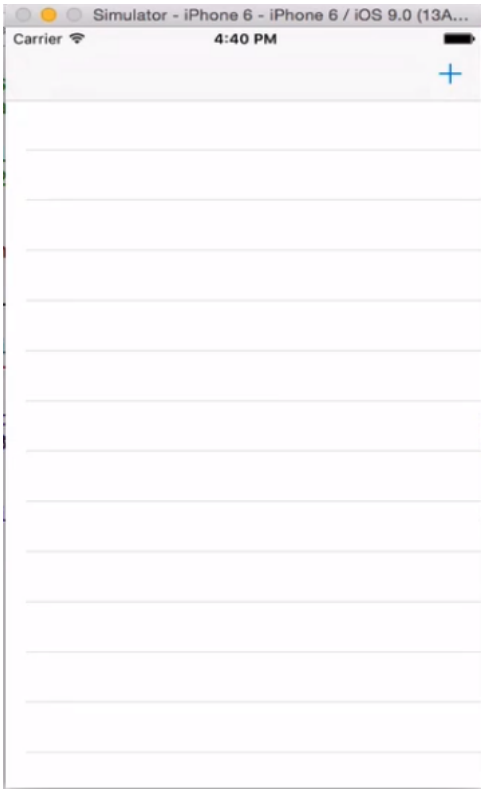
Comecemos pelo `target`. Queremos que ele chame um método da própria Classe de lista:

```
target:self
```

Mas qual método queremos que execute a ação? Por enquanto deixaremos nulo para testar o `target` e vermos o botão ser renderizado:

```
action:nil
```

Rodando novamente:



De fato, o botão está renderizando! Mas se clicarmos nada acontece. Ainda não passamos a ação para ele.

`Selector push` e *Storyboard*

Queremos que o botão `+` exiba o formulário. Criemos, então, um outro método, logo depois do `-(ListaContatosViewController *)` que será responsável por exibir o formulário:

```
-(void) exibeFormulario {

}
```

E, no método da lista, pedimos para executar a ação de `exibeFormulario`:

```
action:exibeFormulario
```

Porém, solto assim, não irá funcionar. Quem precisa executar o método é o botão. Então indicamos a ele que **selecione** o método `exibeFormulario` com o `@selector` :

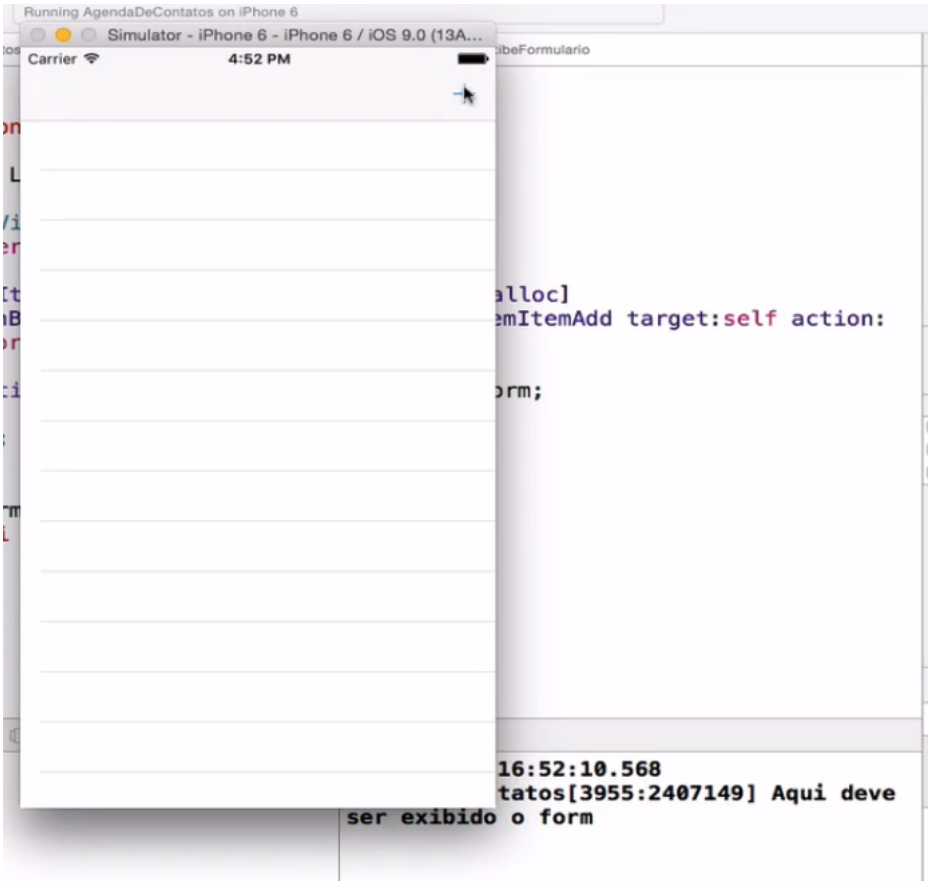
```
action:@selector(exibeFormulario)`
```

Lembre-se quando vinculamos o botão "Adicionar" ao ViewController via Storyboard. Essas duas ações de `target` e `action` são a mesma coisa, porém fizemos manualmente.

Vamos inserir um `MSLog` no método `exibeFormulario` para ver se a seleção está acontecendo:

```
-(void) exibeFormulario {
    NSLog(@"Aqui deve ser exibido o form");
}
```

Na hora que apertarmos o botão, deve aparecer a frase no log. E, de fato:



Agora falta exibir realmente o formulário. O responsável por isso é o `navigationController` , suas telas vão sendo trocadas. Então indicamos para ele **empurrar** para a tela o formulário, usando o método `pushViewController` :

```
-(void) exibeFormulario {
    [self.navigationController pushViewController:(nonnull UIViewController *) animated:(BOOL)];
}
```

Então precisaremos ter o ViewController instanciado dentro do método e importado:

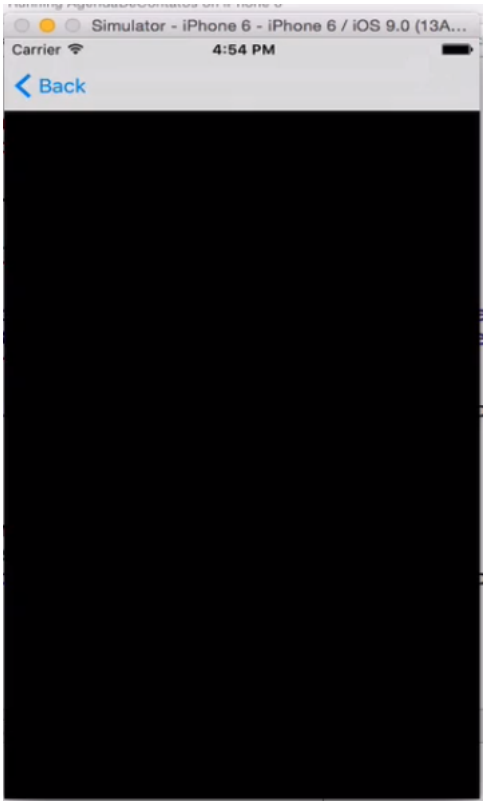
```
#import "ViewController.h"
```

```
//...

-(void) exibeFormulario {
    ViewController *form = [ViewController new];
    [self.navigationController pushViewController:(nonnull UIViewController *) animated:(BOOL)];
}
```

Ele será "empurrado" para o *form* com uma animação de deslize, passando um booleano dizendo que "sim":

```
-(void) exibeFormulario {
    ViewController *form = [ViewController new];
    [self.navigationController pushViewController:form animated:YES];
}
```

Vamos mandar rodar novamente e testar:



Parece que não funcionou, apareceu uma tela preta. Voltemos ao código: o `new` é a mesma coisa que se tivermos feito uma `alloc` seguido de `init`:

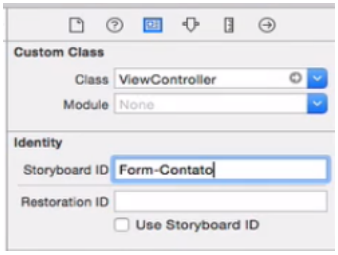
```
ViewController *form = [[ViewController alloc] init];
```

Acontece que a tela do ViewController está dentro de um Storyboard que é quem instancia aquele. Dessa forma, não conseguimos instanciar via código com o `new`. Então apagamos a linha anterior e pedimos diretamente para o Storyboard:

```
-(void) exibeFormulario {
    UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
    [storyboard instantiateViewControllerWithIdentifier:@"Form-Contato"];
    [self.navigationController pushViewController:form animated:YES];
}
```

Precisamos dizer que o `Form-Contato` é o nosso formulário. Fazemos isso selecionando o botão do ViewController

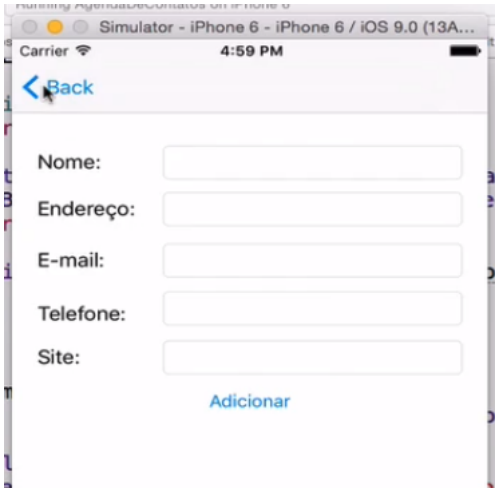
e clicando na terceira opção do menu da direita ("Show the Identity Inspector") e passamos que o *ID* do Storyboard é `Form-Contato`:



Agora guardamos a referência dele dentro de uma variável do tipo `UIViewController` chamada de `form`:

```
-(void) exibeFormulario {
    UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
    UIViewController *form = [storyboard instantiateViewControllerWithIdentifier:@"Form-Contato"];
    [self.navigationController pushViewController:form animated:YES];
}
```

Testando de novo, percebemos que funciona! Ao clicar no botão de `+` o formulário aparece! E, melhor ainda, automaticamente aparece um botão para voltar para a lista de contatos:



Isso acontece porque o NavigationController guarda o fluxo de navegação.

Agora, o que precisamos é que os dados digitados no formulário apareçam na lista.

Navigation pop

Precisamos mudar o método ao qual o botão de "Adicionar" está ligado, que, neste momento, é o método `adiciona` na Classe "ViewController.m":

```
-(IBAction) adiciona {

    //...
```

```
    [self.navigationController popViewControllerAnimated:YES];
}
```

O `pop` faz o inverso do `push`, ou seja, tira a tela atual do fluxo de navegação. Então falamos para o NavigationController, assim que apertarmos o botão, para voltar para a lista. Ao rodarmos e testarmos, de fato o botão "Adiciona"nos leva para a lista.

Perceba que a navegação entre as telas de formulário e de lista está funcionando corretamente. Mas ainda não estamos adicionando o contato à lista.

initWithCoder

Como estamos utilizando o NavigationController, é interessante que o botão de "Adicionar" também fique lá posicionado, afinal, após clicado queremos que volte para a tela da lista de contatos. O botão `+` foi implementado no "ListaContatosViewController.m". Para o `Adicionar` faremos no "ViewController.m" e será parecido, porém com o comportamento diferente e com outro texto:

```
@implementation ViewController

-(UIViewController *)init {
    self = [super init];
    if (self){
        UIBarButtonItem *botao = [[UIBarButtonItem alloc] initWithTitle:@"Adicionar" style:UIBarButt
    }
    return self;
}
```

Para o botão ficar visível com seu nome correto dentro do NavigationController do lado direito fazemos:

```
-(UIViewController *)init {
    self = [super init];
    if (self){
        UIBarButtonItem *botao = [[UIBarButtonItem alloc] initWithTitle:@"Adicionar" style:UIBarButt
        self.navigationItem.rightBarButtonItem = botao;
        self.navigationItem.title = @"Novo Contato";
    }
    return self;
}
```

Ao rodarmos podemos perceber que o botão ainda não aparece no NavigationController, além disso também não excluímos o botão antigo, abaixo do formulário. Isso acontece porque quem instancia a Classe "ViewController" é o Storyboard que não deve chamar o `init` padrão da Classe, mas sim o `initWithCoder, o qual recebe um parâmetro chamado de` NSCoder `com decodificador` aDecoder`:

```
-(UIViewController *)initWithCoder: (NSCoder *) aDecoder {
    self = [super initWithCoder:aDecoder];
    if (self){
        UIBarButtonItem *botao = [[UIBarButtonItem alloc] initWithTitle:@"Adicionar" style:UIBarButt
        self.navigationItem.rightBarButtonItem = botao;
```

```
        self.navigationItem.title = @"Novo Contato";
    }
    return self;
}
```

Agora sim, se mandarmos rodar novamente:

Se é o Storyboard que instancia, usamos o `initWithCoder` diferentemente de quando instanciamos na mão.

No *Objective-C* podemos pedir para retornar o objeto mais genérico possível, nesse caso, a classe mãe `NSObject` de todas as classes:

```
-(NSObject *)initWithCoder: (NSCoder *) aDecoder
```

Como é muito comum de se usar o `NSObject *` , podemos substituí-lo por `id` que teremos o mesmo tipo de retorno:

```
-(id)initWithCoder: (NSCoder *) aDecoder
```

A mesma coisa para `ListaContatosViewController *` :

```
-(id) init
```

Falta tirar o botão de Adicionar lá de baixo do formulário. Primeiro deletamos simplesmente o botão no Storyboard que sua ligação com o ViewController já é desfeita. Sendo assim, o método `adiciona` não precisa ficar mais visível para o Storyboard, ou seja, voltamos para `void` :

```
-(void) adiciona
```

Outra implementação que podemos fazer é colocar um título para a lista, já que colocamos um para o formulário ("Novo Contato"):

```
-(id) init {

//...

    self.navigationItem.title = @"Contatos";
return self;

}
```

Os contatos ainda não estão sendo inseridos na lista. Veremos isso na próxima aula.