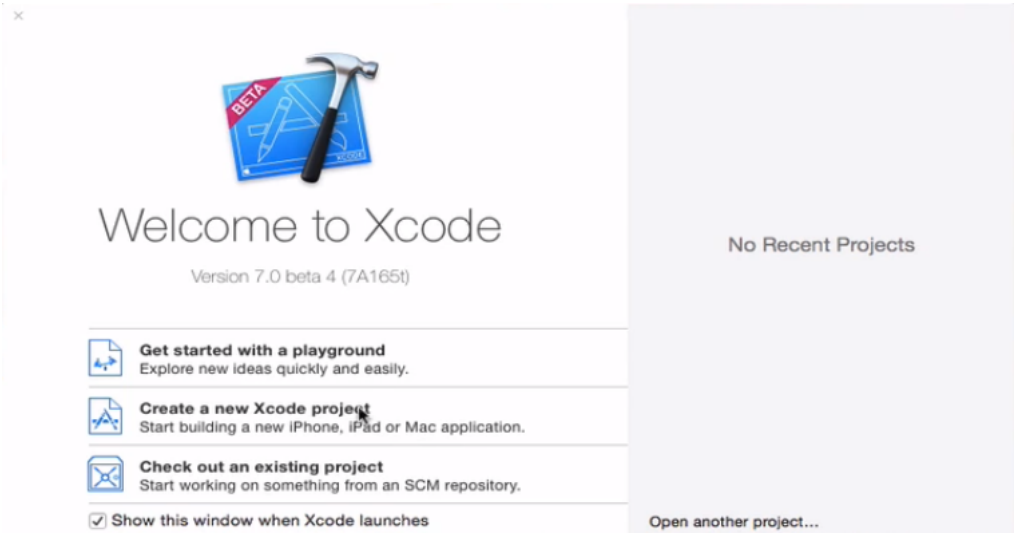
08

Storyboard

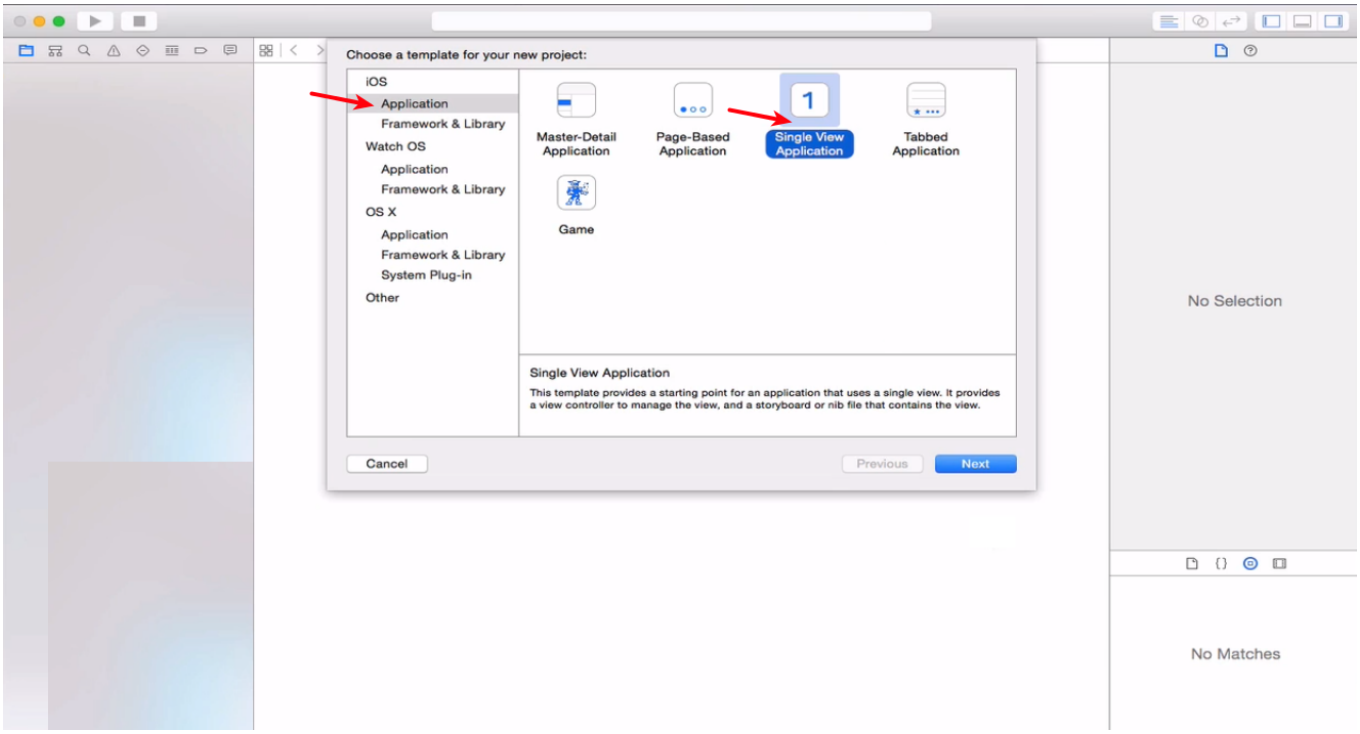
Storyboard

Sejam bem-vindos ao curso de Objective-C! Neste curso criaremos uma agenda de contatos com várias funcionalidades para conseguirmos aprender diversas coisas da linguagem, como fazemos aplicativos para iOS usando o Objective-C.

A primeira coisa que faremos será instalar o *Xcode* (https://developer.apple.com/xcode/). Feito isso, criaremos o nosso primeiro projeto clicando em "Create a new Xcode project":

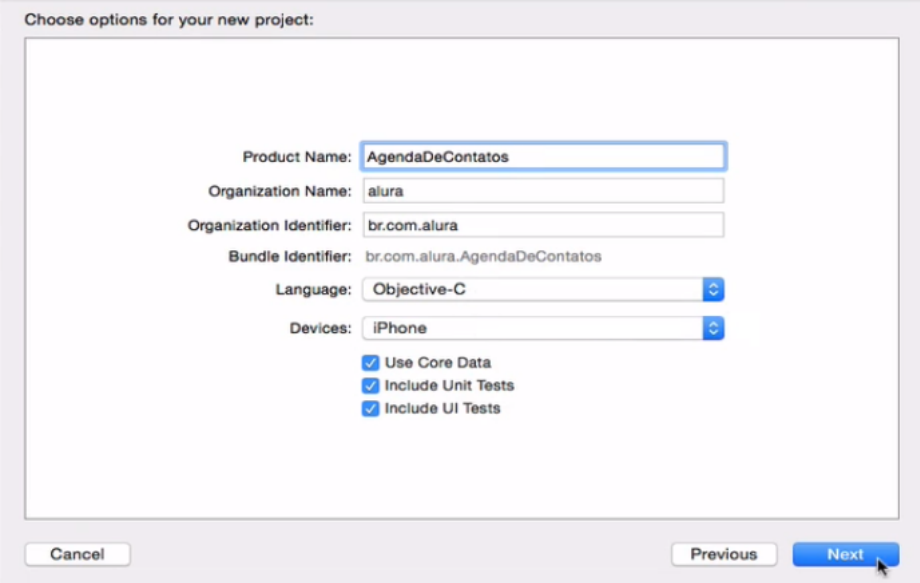


Com o Xcode aberto abrirá uma janela com opções para criarmos para iOS, Watch OS ou OS X. Começaremos para iOS selecionando "Application" e depois em "Single View Application", pois esta opção vem com a menor quantidade de coisas que precisaremos:



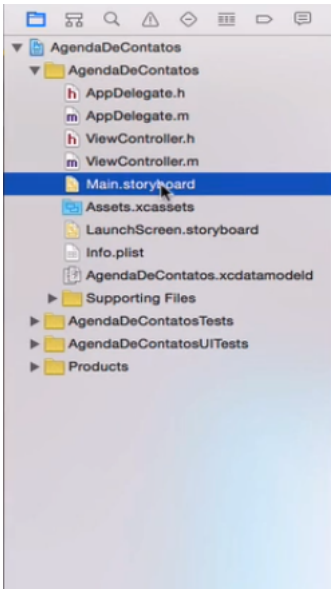
Apertamos "Next" e a próxima tela nos pede alguns dados do projeto. Vamos dar o nome de `AgendaDeContatos` , o *Identifier* é como se fosse o domínio e a linguagem que usaremos é *Objective-C* e o dispositivo é o *iPhone*. O resto deixaremos como o

padrão:

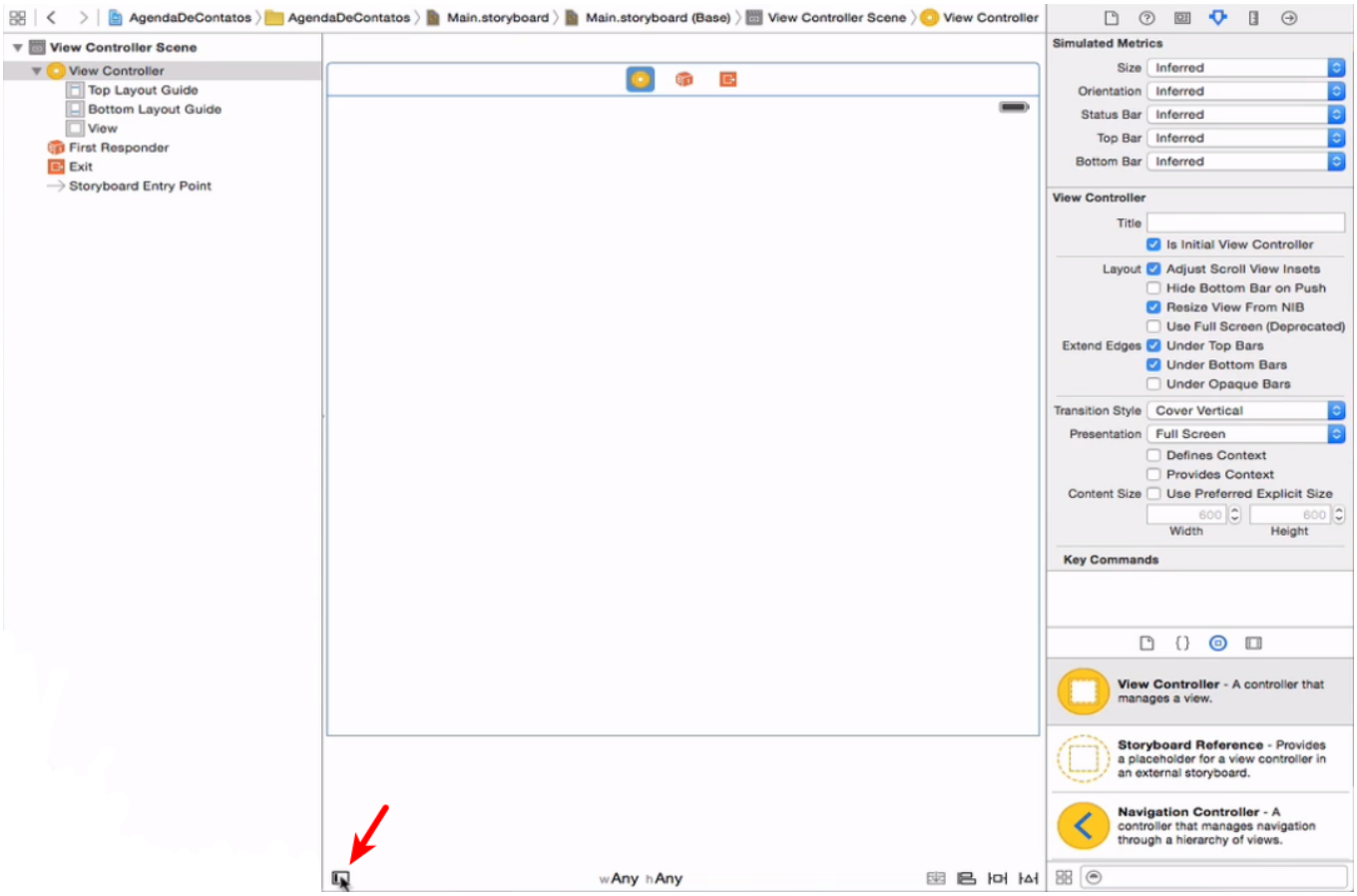


Dando "Next", o programa irá pedir o diretório onde a aplicação será salva. Escolhemos a pasta e clicamos em "Create". Entramos na tela de configuração do projeto, com diversas opções. Não mudaremos nada aqui.

Do lado esquerdo da tela podemos ver alguns arquivos em cascata, vamos utilizar o `Main.storyboard` :

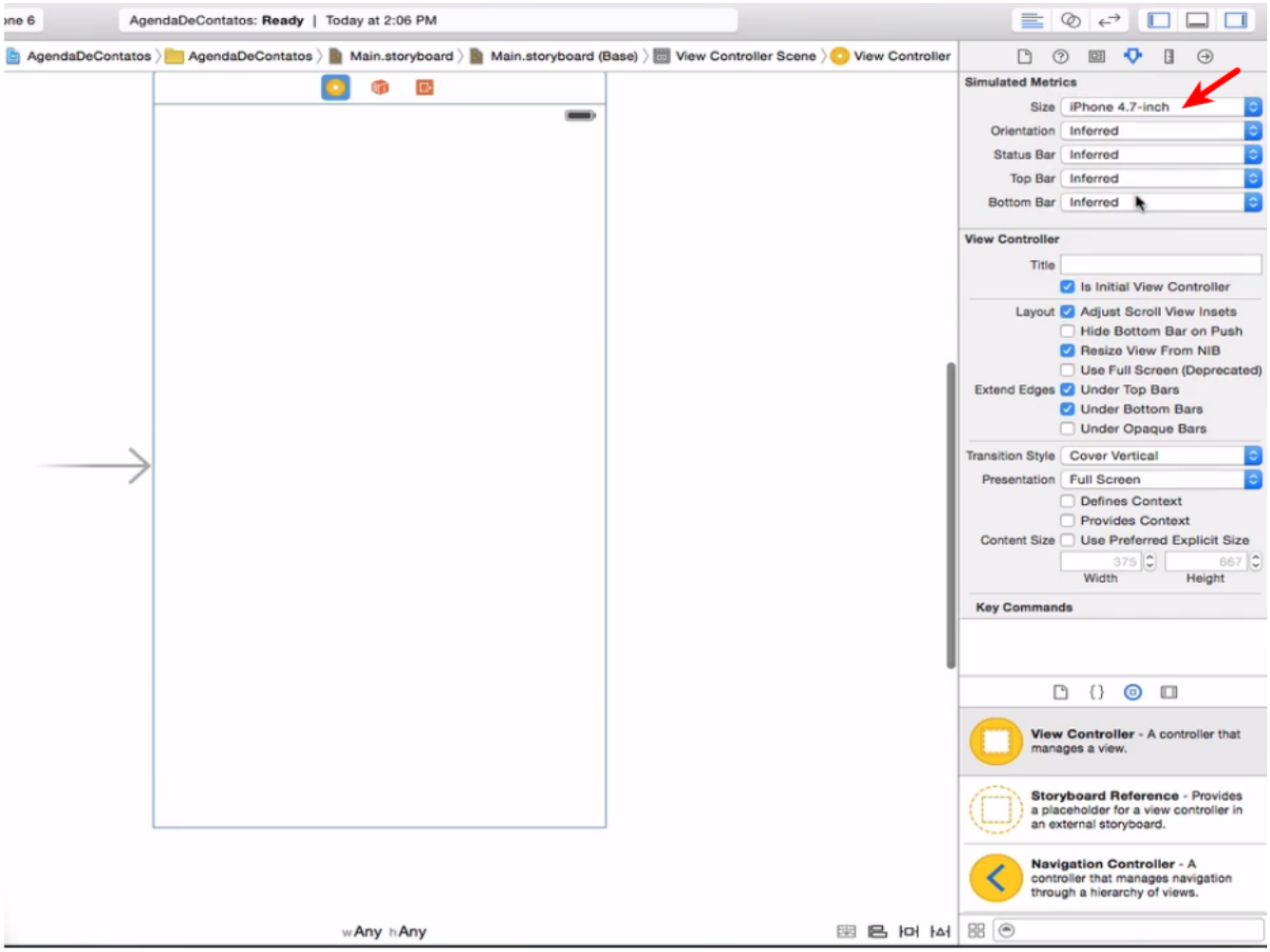


É ele que representa a tela de nossa aplicação, aqui desenharemos as informações. A tela do *View Controller* está um pouco apertada aqui, então clicamos no botão que fica abaixo dela:

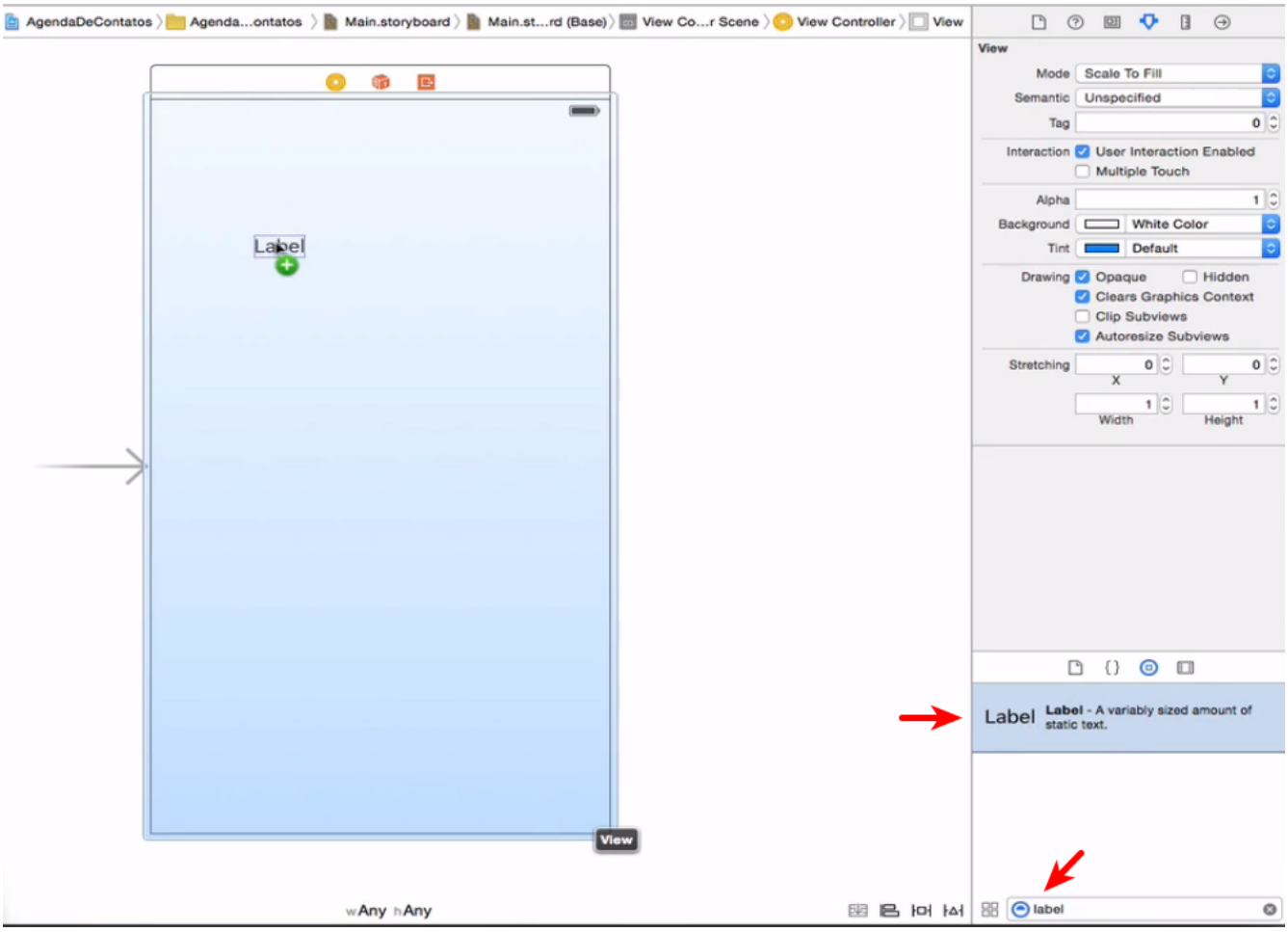


Agora poderemos trabalhar melhor. A tela, como podemos ver, é quadrada. Vamos mudar para deixar do tamanho de um iPhone 5, por exemplo, fazemos isso no menu da direita, após clicar no botão de opções

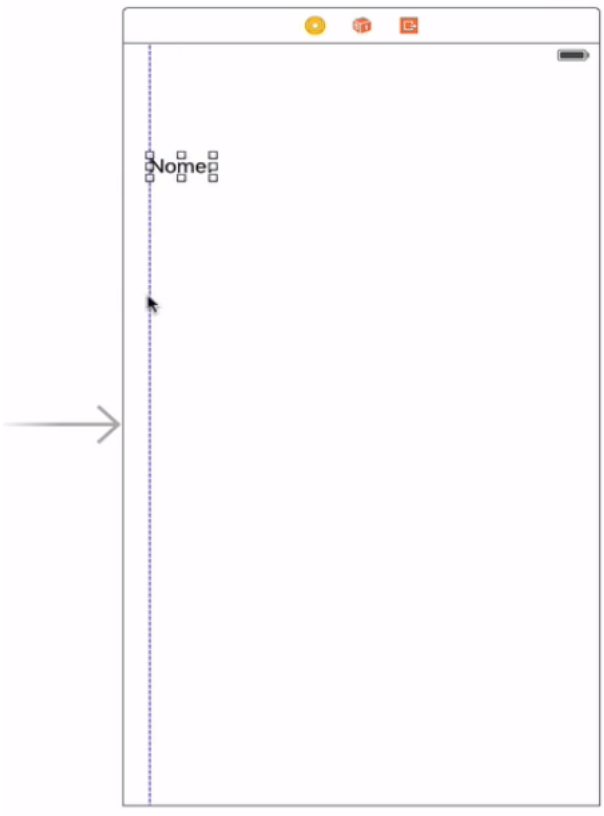
:



No canto inferior direito do programa vemos os componentes que podemos usar para criar a aplicação. Quermos fazer um formulário, que será nossa primeira tela. Lá é possível filtrar e faremos isso para criar o campo do nome. Escrevemos, então, *Label* e a arrastamos para dentro da tela:

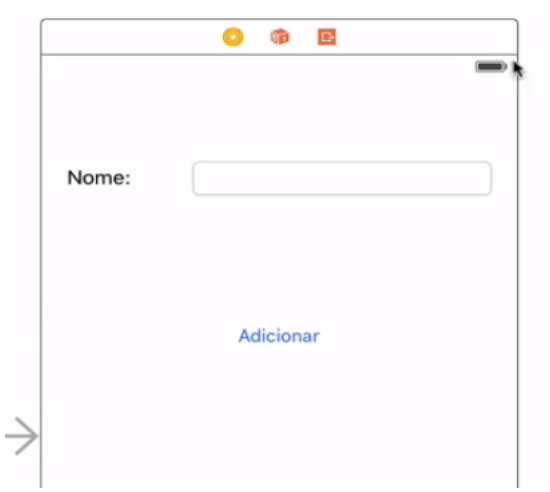


Irá aparecer "Label" e podemos mudar para "Nome" e posicionar utilizando guias inteligentes:

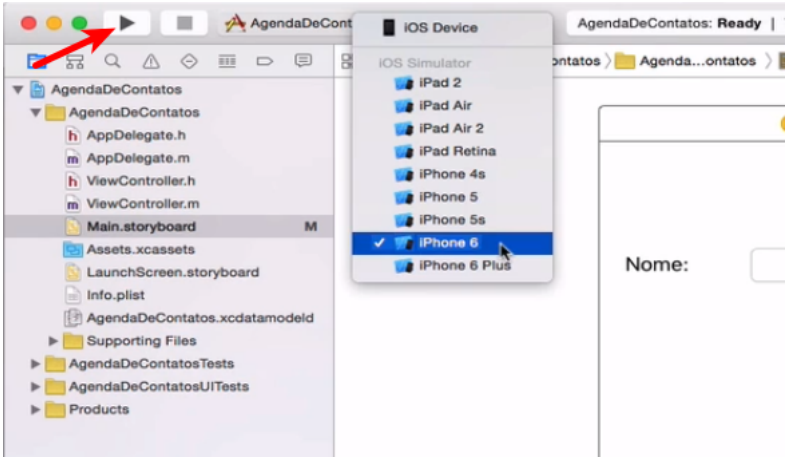


Vamos inserir a caixa de texto onde o usuário irá digitar seu nome. Este campo se chama "Text Field". Vamos adicioná-lo também arrastando, vamos posicioná-lo e redimensioná-lo usando as guias:

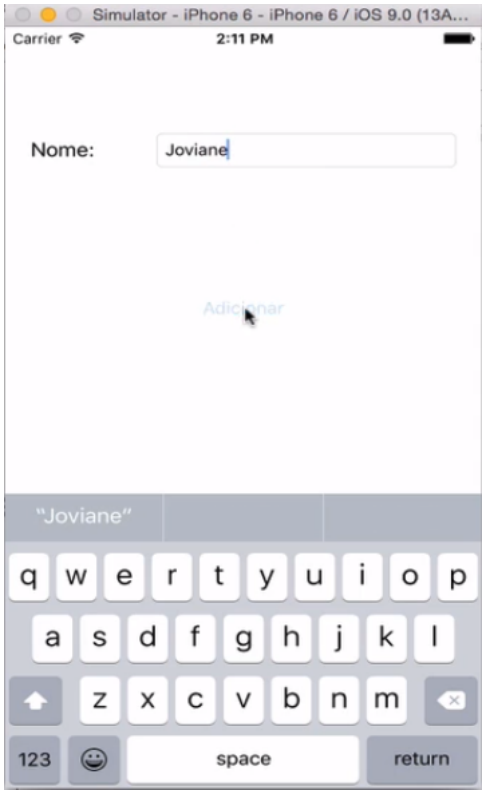
Outra forma de encontrar os campos é varrer manualmente por eles no menu inferior direito, caso não soubermos o nome. O próximo campo é um botão (procuramos por *Button*) de adicionar. O posicionamos como fizemos com os outros campos e o renomeamos:



Agora vamos rodar para ver se está tudo certo fazendo um simulação. Escolhemos o dispositivo e apertamos o *play*:



Após o programa compilar, será aberto o simulador do iOS (a primeira vez demora alguns minutinhos). Perceba que poderemos interagir com o simulador e escrever, por exemplo, o nome em seu campo e apertar o botão - que, no momento, não fará nada:



Perceba que poderemos digitar as informações tanto no teclado físico quanto no do simulador. Se não quisermos usar este último, basta ir no menu superior e selecionar "Hardware > Keyboard > Toggle Software Keyboard" ou "Ctrl + K".

Comportamento do botão

Lembre-se que no simulador, ao apertarmos o botão, nada ocorre. Vamos, então adicionar um comportamento para essa ação. Tal comportamento irá guardar o nome digitado. Cada tela adicionada ao Storyboard possui um controlador, o **View Controller**

.

Podemos perceber que no menu em cascata à esquerda existem dois arquivos:

- "ViewController.h", com a declaração da classe:

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

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@end
```

- "ViewController.m", com as implementações:

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

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resourses that can be recreated.
}

@end
```

O mínimo que iremos precisar neste segundo arquivo é

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

#import "ViewController.h"

@implementation ViewController


@end
```

O `@implementation ViewController` representa a nossa classe do Objective-C. Comecemos, então a implementação. Como falamos acima, queremos que uma ação seja executada quando clicamos no botão. Por exemplo, que o nosso nome seja impresso no *log*. Fazemos isso através de um **método**, que começa com `-` , dentro do **objeto** *View Controller*. Passamos o tipo de retorno dentro de parênteses `(void)` e o nome desse método abrindo e fechando chaves:

```
@implementation ViewController

-(void) adiciona {
```

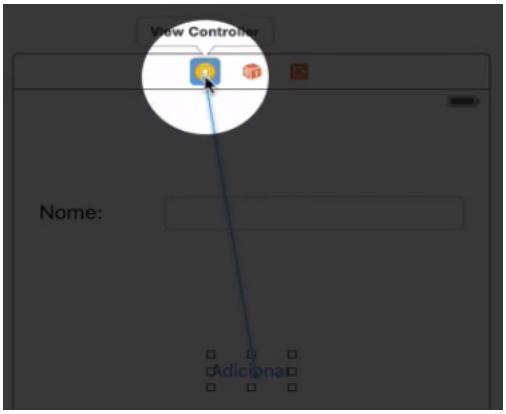
```
}

@end
```

Se quisermos imprimir uma mensagem no *log* usamos o comando `NSLog` seguido de `@mensagem` :

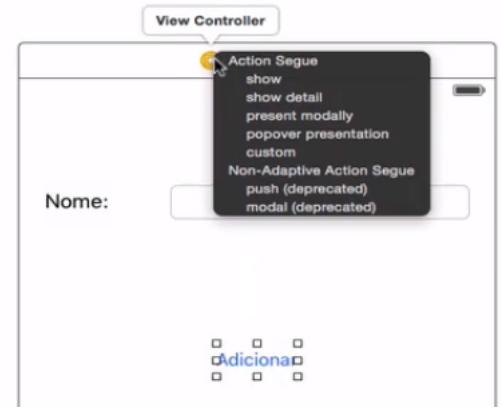
```
@implementation ViewController

-(void) adiciona {
    NSLog(@"Clicou no botão");
}

@end
```

A primeira coisa é saber se o usuário clicou realmente no botão. Agora vamos implementar para que quando ele clicar no botão "Adiciona" o método que acabamos de escrever seja chamado. para podermos vincular essa ação com o botão. Isso acontece da seguinte forma:

O botão avisa o *View Controller* que ele foi clicado. Fazemos isso segurando a tecla "Ctrl", clicando no botão e arrastando até o *View Controller":



Irá aparecer um menu com os métodos possíveis de serem executados:

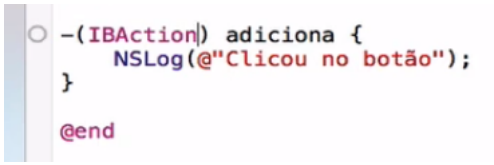


Porém, perceba que o método que escrevemos anteriormente, o `adiciona` , não está listado, não está visível para o *Interface Builder* (o storyboard que vínhamos trabalhando até agora). Para que ele fique visível, em vez de usarmos `void` como tipo de retorno, usaremos `IB Action` , ou seja, "I*nterface *Builder, execute essa ação":

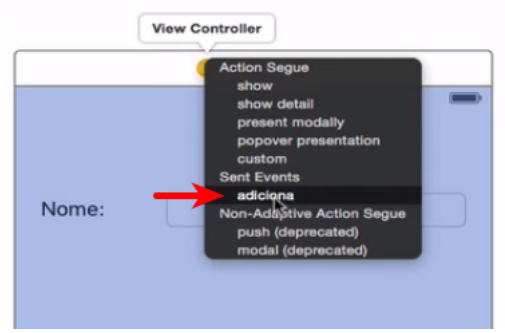
```
@implementation ViewController
```

```
-(IBAction) adiciona {
    NSLog(@"Clicou no botão");
}

@end
```

Perceba que apareceu um círculo ao lado do método indicando se existe uma ligação ou não com um botão:

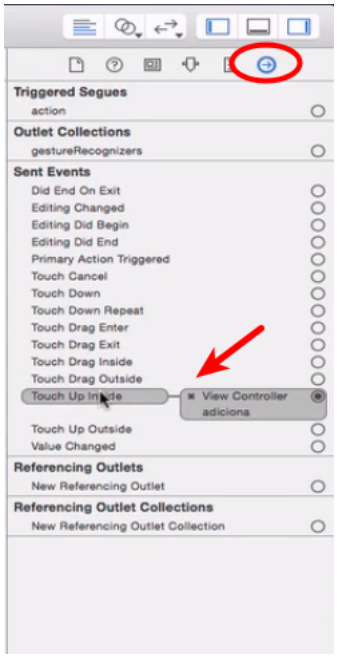


Veja que o círculo está sem preenchimento, isso significa que ainda não existe ligação. Então voltamos no Storyboard e repetimos o mesmo processo, conectando o botão ao *View Controller*. Agora nosso método aparece:

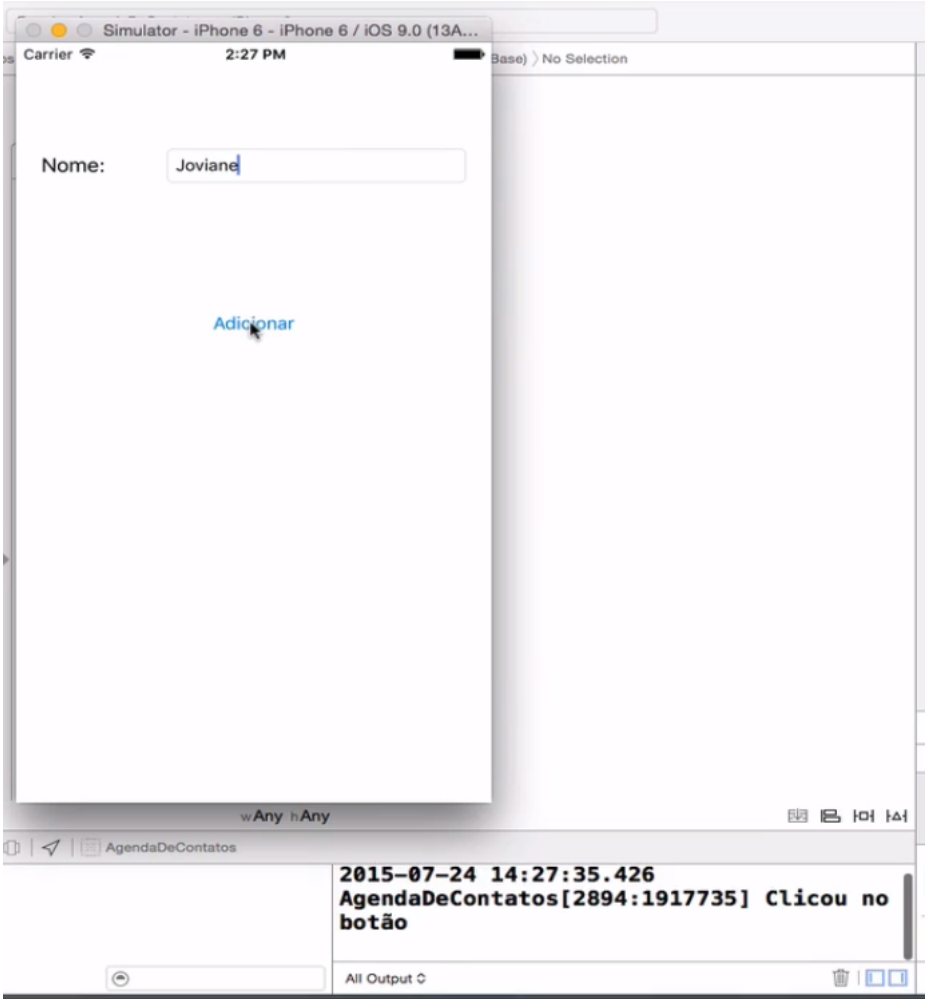


Para sabermos se a conexão foi de fato feita, clicamos no botão e selecionamos as "Conections"

. Logo abaixo dela irão aparecer todas as conexões do botão:



Vamos simular a aplicação dando "Play" e clicamos em "Adicionar":



Perceba que, de fato, o output da aplicação foi a frase que escrevemos dentro do método (Clicou no botão). Ele aparece em uma caixa do lado direito inferior. Todas as saídas que escrevermos serão mostradas nessa caixa.

Capturando o Nome

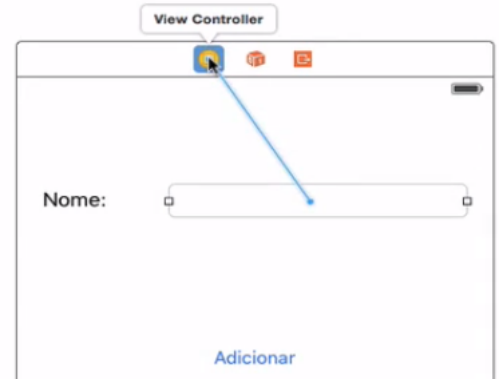
Agora que já conseguimos clicar no botão e executar a ação, precisamos capturar o texto que for digitado no campo do Nome. Ou seja, quando clicarmos em "Adicionar" haverá um lugar, uma propriedade que armazena as informações do texto escrito.

O arquivo "ViewController.h" separa a parte pública da parte privada do nosso código. É lá que declaramos essa propriedade (`@property`) e a informação representa um campo de texto (`UITextField`) com o nome (`*nome`):

```
@interface ViewController : UIViewController

@property UITextField *nome;

@end
```

Então declaramos uma propriedade, cujo tipo é de texto, e o `*` , seguido do nome da variável, representa um ponteiro, uma referência para o nome da mesma.

Da mesma forma que ligamos o botão "Adicionar" com o método `adiciona` , precisaremos ligar o campo do nome com a propriedade:



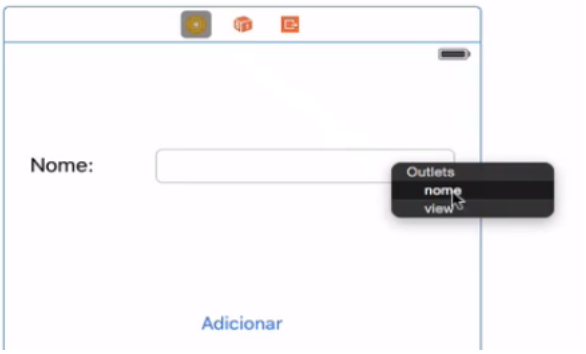
Vamos recomeçar e pensar sobre essa ligação. Quando, anteriormente, ligamos o botão "Adiciona" ao View Controller estávamos dando um input de que o usuário havia clicado no botão, por isso puxamos do botão ao View Controller.

Agora, o código está perguntando para o campo qual o texto digitado, através do View Controller. Portanto, o caminho será o inverso: puxamos do View Controller para o campo:

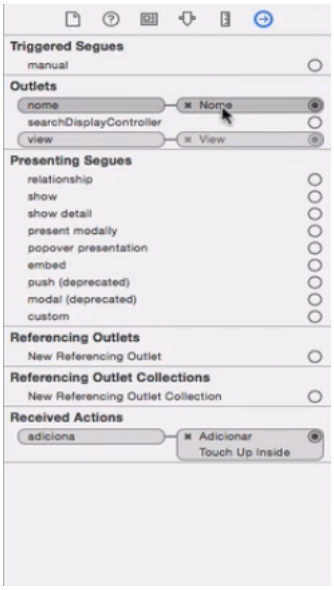
Também não teremos a propriedade listada no menu. Para podermos vincular o texto com o campo nome, teremos que passar o tipo de vínculo, no caso um **Outlet** (saída), para o Interface Builder:

```
@interface ViewController : UIViewController

@property IBOutlet UITextField *nome;

@end
```

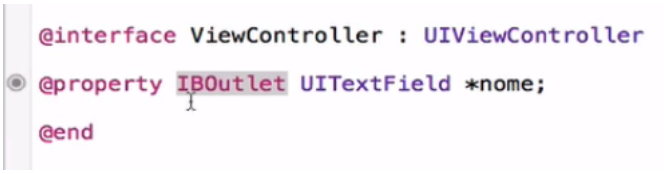
Agora ele está pronto para fazer uma ligação:



E, de fato, vendo nas conections, ele foi vinculado:



E aquele círculo está preenchido, indicando o mesmo:



Agora queremos imprimir essa informação. Fazemos isso dentro do logo no método `adiciona` :

```
@implementation ViewController

-(IBAction) adiciona {
    NSLog(@"Clicou no botão %@", self.nome);
}

@end
```

Usamos o símbolo `%` pois o Objective-C trabalha com interpolação e o `@` pois é o tipo da *String*. Como queremos pegar a informação do **próprio** nome, fazemos `self.nome` .

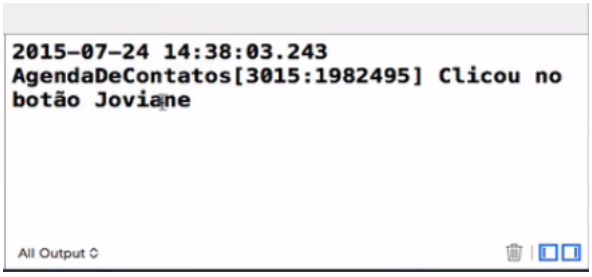
Vamos simular novamente a aplicação, escrevendo um nome no campo e clicando em "Adicionar". Agora serão impressas diversas informações na caixa de output, além da frase "Clicou no botão":

```
2015-07-24 14:36:03.755
AgendaDeContatos[2986:1966037] Clicou no
botão <UITextField: 0x7facca623c80; frame
= (119 93; 236 30); text = 'Joviane';
clipsToBounds = YES; opaque = NO;
autoresize = RM+BM; gestureRecognizers =
<NSArray: 0x7facca5252b0>; layer =
<CALayer: 0x7facca624210>>
```

Nós ligarmos o `IBOutlet` com o `UITextfield` , porém este não imprime só o texto - `text = 'Joviane';` -, também aparecem as dimensões - `frame (119 93; 236 30);` . Não estamos interessados nesta e em outras informações, apenas no texto digitado pelo usuário. Vamos mudar o código. Perceba que passamos `self.nome` , isso indica tudo relacionado ao campo. Então, para imprimir apenas o texto fazemos:

```
@implementation ViewController

-(IBAction) adiciona {
    NSLog(@"Clicou no botão %@", [self.nome text]);
}

@end
```

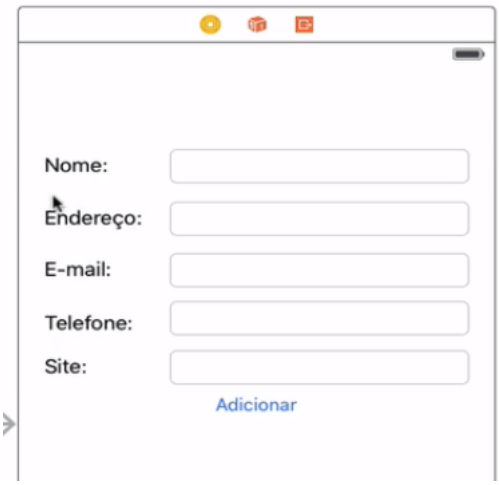
Usamos os colchetes para passar essa mensagem para o método. Se simularmos novamente a ação:



Acrescentando outros campos e declarando variáveis

Agora que já conseguimos inserir um campo do formulário e fazer as ligações necessárias, façamos o mesmo para os outros campos. Podemos copiar e colar

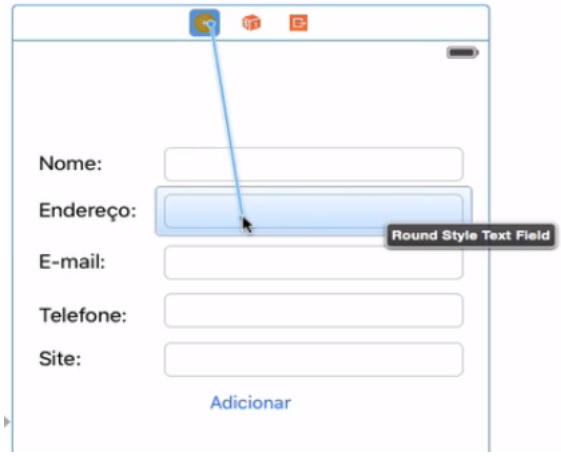
com "Ctrl + C" e "Ctrl + V" e apenas mudar os nomes das *labels*:



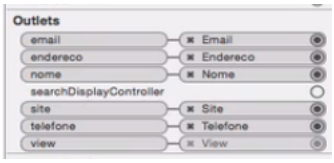
Vamos criar, agora, as propriedade correspondentes:

```
@interface ViewController : UIViewController

@property IBOutlet UITextField *nome;
@property IBOutlet UITextField *endereco;
@property IBOutlet UITextField *email;
@property IBOutlet UITextField *telefone;
@property IBOutlet UITextField *site;

@end
```

Sabemos que os circulos do lado de cada propriedade estão sem preenchimento, pois elas não estão conectadas. Como fizemos com o campo do Nome, ligamos o View Controller com o campo onde inserimos as informações:



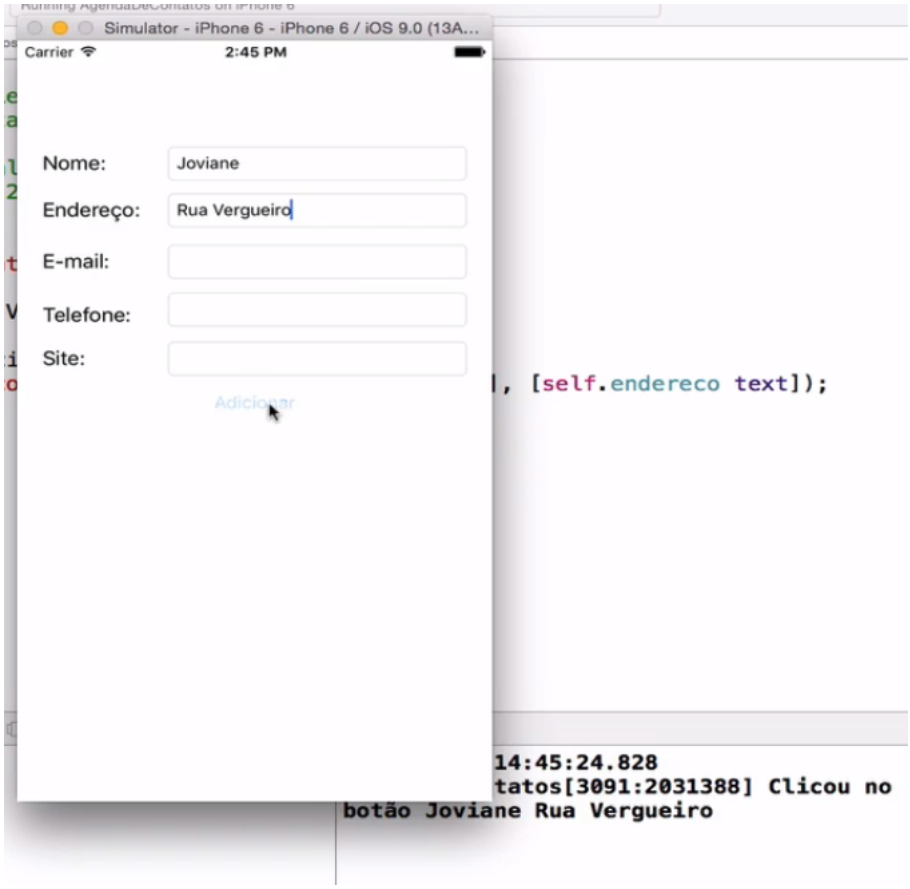
E selecionamos o *textfield* correspondente no menu que aparecerá. Lembre-se que para confirmar que tudo está vinculado é só ver na aba das *connections*:



Para mostrarmos as informações, implementamos no `NSLog()` :

```
@implementation ViewController

-(IBAction) adiciona {
    NSLog(@"Clicou no botão %@ %@", [self.nome text], [self.endereco text]);
}

@end
```

Testaremos com os campos de nome e de endereço primeiramente. Rodando a aplicação, adicionando as informações nos campos de nome e endereço e clicando em Adicionar:



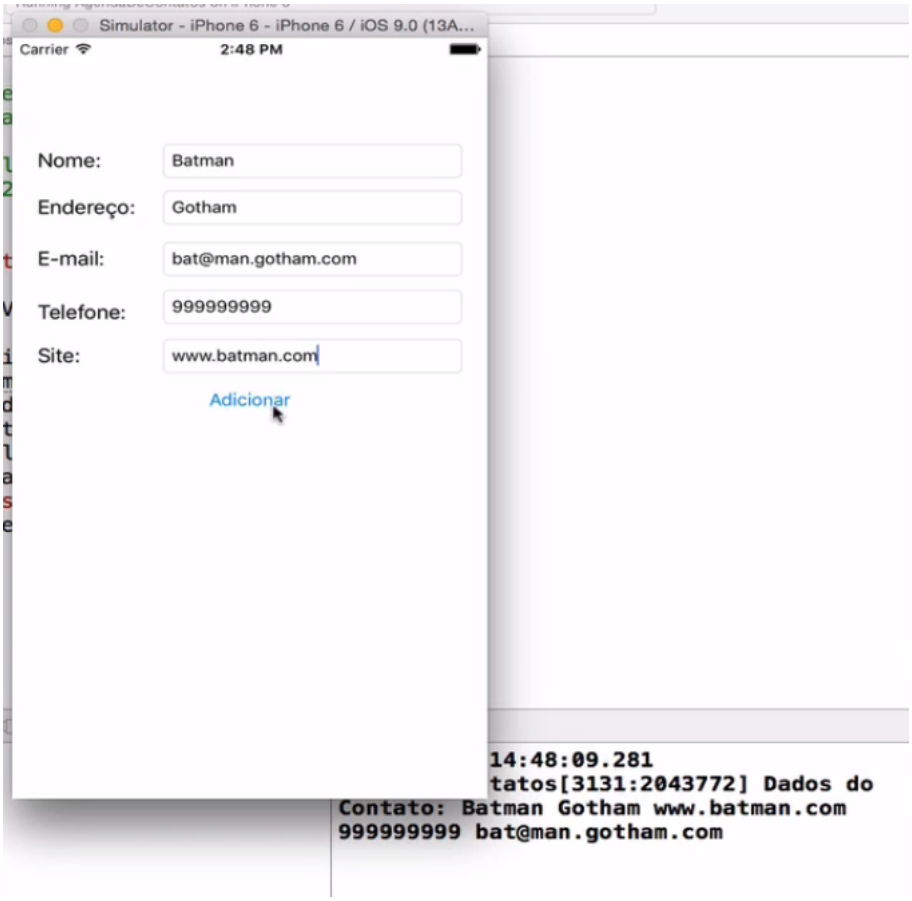
De fato as informações foram salvas e apareceram na caixa. Agora vamos melhorar o código. Primeiramente, trocaremos `Clicou no botão` por `Dados do contato:` , pois faz mais sentido:

```
...
NSLog(@"Dados do contato: %@ %@", [self.nome text], [self.endereco text]);
...
```

Perceba também que o código ficará sujo se dermos sequência e adicionarmos `[self.email text], [self.telefone text]` etc. Para solucionar isso, declaramos variáveis acima do tipo `NString` que irão guardar o texto de cada um dos campos:

```
@implementation ViewController

-(IBAction) adiciona {
    NString *nome = [self.nome text];
    NString *endereco = [self.endereco text];
    NString *email = [self.email text];
    NString *telefone = [self.telefone text];
    NString *site = [self.site text];
    NSLog(@"Dados do contato: %@ %@ %@ %@ %@", nome, endereco, email, telefone, site);
}

@end
```

Perceba que, no momento de interpolar, o código fica muito mais limpo. Vamos rodar novamente a aplicação e adicionar as informações:



Funcionou para todos os campos!

Refatorando o código

Toda vez que queremos acessar o texto de uma propriedade, por exemplo `*nome` , fazemos `[self.nome text]` . Porém, vamos olhar para dentro da Classe `UITextField` . Fazemos isso clicando nela onde foi declarada com o "Ctrl" apertado, por exemplo em:

```
@property IBOutlet UITextField *nome;
```

Um código se abrirá e poderemos ler as seguintes linhas:

```
@property(nullabe, nonatomic, copy)  NSString
    *text;                // default is nil
```

Podemos observar que o `*text` também é uma propriedade do `UITextField` ! Quando isso acontece, quando um elemento é uma propriedade, não precisamos mandar uma mensagem. Podemos usar diretamente o ponto entre as propriedades e sem os colchetes:

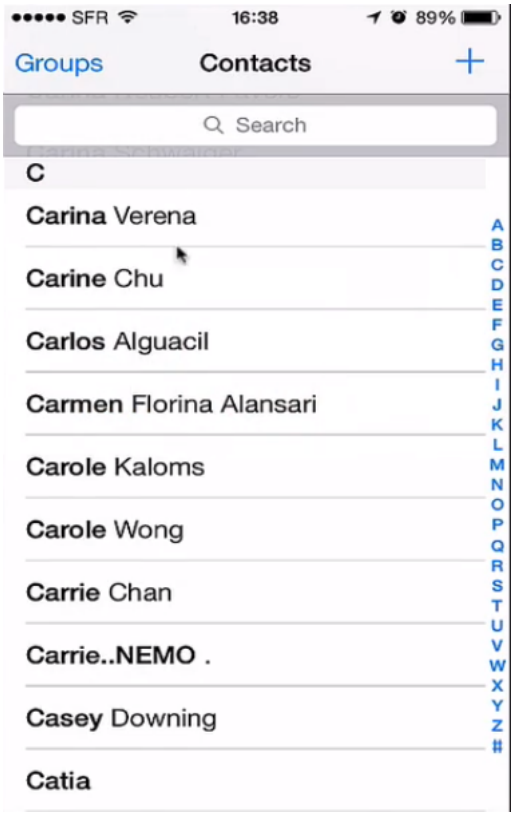
```
@implementation ViewController

-(IBAction) adiciona {
    NString *nome = self.nome.text;
    NString *endereco = self.endereco.text;
    NString *email = self.email.text;
    NString *telefone = self.telefone.text;
    NString *site = self.site.text;
```

```
    NSLog(@"Dados do contato: %@ %@ %@ %@ %@", nome, endereco, email, telefone, site);
}

@end
```

Se rodarmos, tudo funcionará como antes.

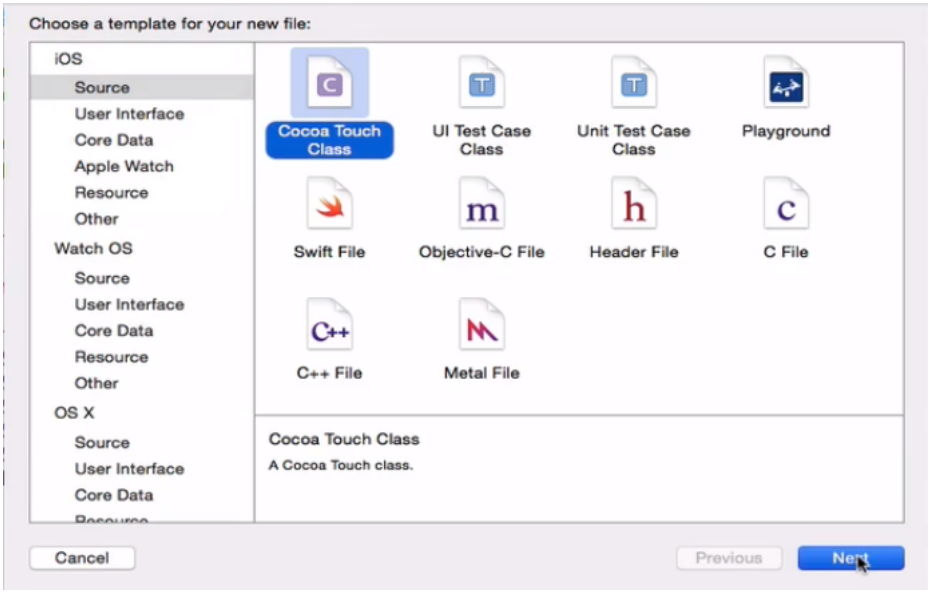
A Classe `contato`

Não nos limitaremos a apenas adicionar contatos, mas também os listaremos, como no exemplo abaixo:



Toda vez iremos precisar das informações adicionadas pelo formulário e guardá-las em outro lugar. Para não fazermos um *copy-paste* do código já implementado, vamos armazenar `*nome` , `*endereco` , `*email` , `*telefone` e `*site` em uma Classe chamada `contato` , pois é isso que eles representam.

Para criar uma classe usamos o atalho "Ctrl + N" ou no menu superior vamos em "File > New > File...". Então escolhemos o template "Cocoa Touch Class":



Clicamos em "Next" e na tela de opções:

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

Todas as classes de Objective-C são subclasses de *NSObject*. Clicamos em "Next" mais uma vez. Será pedido para escolhermos o diretório que o novo arquivo será armazenado. Escolhemos o mesmo onde estão os outros: "AgendaDeContatos". E a Classe "Contato" foi criada, tendo dois arquivos, o "Contato.m", contendo todas as implementações ou aquilo que será privado:

```
//
//  Contato.m
//  AgendaDeContatos
//
//  Created by alura on7/24/15.
//  Copyright © 2015 alura. All rights reserved.
//

#import "Contato.h"

@implementation Contato

@end
```

E o "Contato.h", que mostra tudo que é público, sendo que o : representa a herança do NSObject:

```
//
//  Contato.h
//  AgendaDeContatos
//
//  Created by alura on7/24/15.
//  Copyright © 2015 alura. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Contato : NSObject
```

```
@end
```

Agora no "ViewController.m" criamos o Contato com sua variável `*contato` , a qual irá armazenar:

```
//codigo
-(IBAction) adiciona {
    Contato *contato;
    NString *nome = self.nome.text;
    ...
//codigo
```

Só que da forma que está não irá compilar, pois o programa ainda não sabe quem é esse `*contato` . Falta dizermos no "Contato.h" que queremos usá-lo dentro de "ViewController.m". Ou seja, da mesma forma que o "ViewController.m" importa o "ViewController.h", ele pode fazer o mesmo com o "Contato.h":

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

#import "ViewController.h"
#import "Contato.h"

@implementation ViewController

//codigo
```

Agora, se quisermos criar um novo **objeto** contato, passamos uma mensagem para a Classe "Contato" dizendo:

```
Contato *contato = [Contato new];
```

Faltou agora indicarmos que as propriedades `self.nome.text` , `self.endereco.text` , etc serão armazenadas dentro de `*contato` e não dentro das variáveis com mesmo nome. Para isso precisaremos criar os atributos dentro da Classe "Contato.m":

```
//...

#import "Contato.h"

@implementation Contato

NSString *nome;

@end
```

Falta conseguirmos colocar o valor para a *String*. Tudo que está no "Contato.m" é privado, ou seja, não adianta acessar direto pelo "ViewController.m":

```
-(IBAction) adiciona {
    Contato *contato = [Contato new];
    contato.nome
    //...
}
```

Precisaremos, então, declarar dentro do "Contato.h" um método para que consigamos colocar o nome, o endereço, etc... Fazemos isso com a assinatura deles. Recebe um *setter*, que costuma ser *void* pois não devolve nada, e, para falar que estamos recebendo um parâmetro, usamos `:` e o tipo que recebe, este guarda uma referência que guarda o `novoNome`:

```
@interface Contato : NSObject

-(void) setNome: (NSString *) novoNome;

@end
```

Dentro de "Contato.m" declaramos o método:

```
@implementation Contato

NSString *nome;

-(void) setNome: (NSString *) novoNome {
    nome = novoNome;

}

@end
```

Agora, podemos pedir para setar o nome no "ViewController.m":

```
-(IBAction) adiciona {
    Contato *contato = [Contato new];
    [contato setNome:self.nome.text];
    //...
}
```

E podemos deletar a linha `NSString *nome = self.nome.text;`. E, para imprimir o nome, ainda dentro do `NSLog()` fazemos:

```
...
NSLog(@"Dados do Contato: %@ %@ %@ %@ %@", [contato nome], ...);
```

Para fazer o *getter*, é o mesmo processo. No "Contato.h" fazemos:

```
@interface Contato : NSObject
```

```
-(void) setNome: (NSString *) novoNome;
-(NSString *) nome;

@end
```

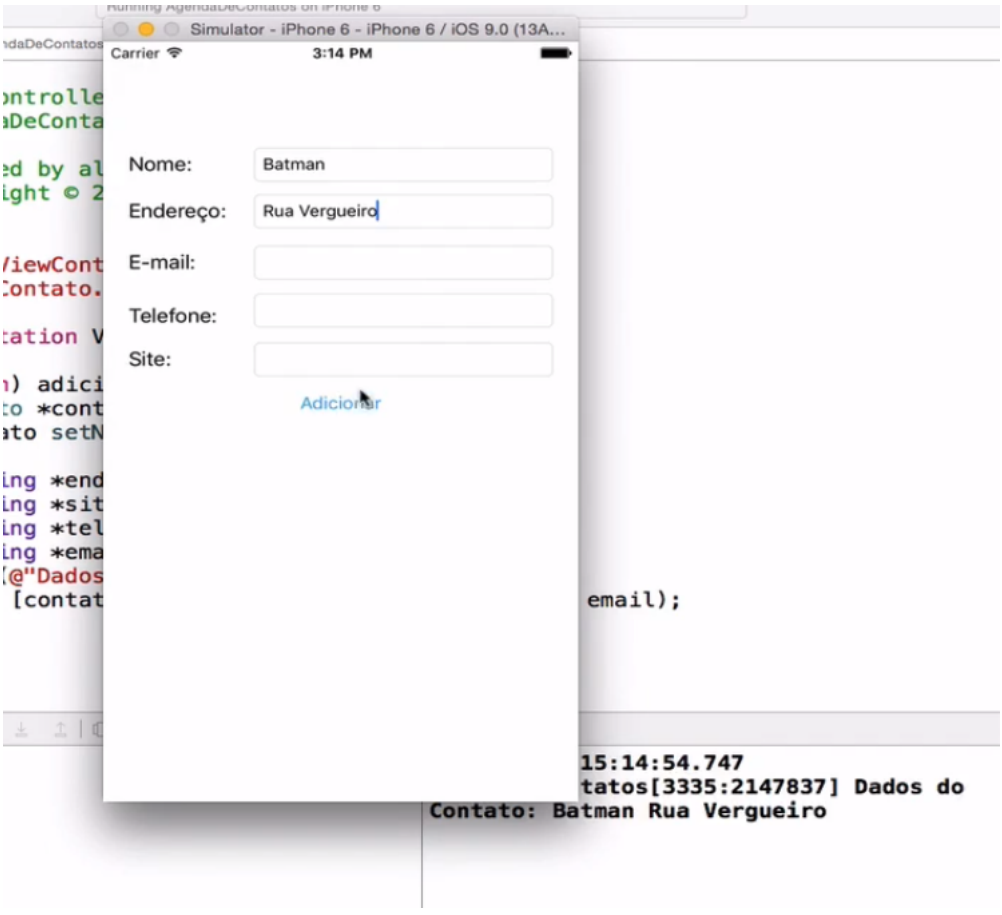
Por convenção o *getter* é o próprio nome do atributo, no caso, `nome`. O "Contato.m" usará esse *getter*:

```
@implementation Contato

...

- (NSString *) nome {
    return nome;
}

@end
```

Recapitulando. Criamos um *getter*, que possui o mesmo nome do atributo, e um *setter*. Para que eles fiquem visíveis, foram declarados em "Contato.h". Se não fizermos isso o "ViewController.m"não compila o `NSLog`.

##Declarando como `property`

Simulemos novamente e ver o que acontece. Apertamos o *play* e vamos tentar armazenar o nome do contato e um outro campo apenas como comparação:



De fato, foi impresso o nome do contato! Agora só falta declarar as outras propriedades. Perceba que o código ficará bem extenso!

Pensando nisso, os desenvolvedores do Objective-C, para não ficarmos declarando o tempo todo, implementaram uma propriedade que pega o atributo, o *setter* e o *getter* e declara tudo junto, é o `@property` . Vejamos como ela fica no "Contato.h" para o atributo `nome` :

```
@interface Contato : NSObject

@property NSString *nome;

@end
```

Com apenas uma linha estamos declarando o *getter*, o *setter* e o atributo. Dessa forma, não precisamos chamar o *setter explicitamente no ViewController:

```
-(IBAction) adiciona {
    Contato *contato = [Contato new];
    contato.nome = self.nome.text;
    //...
}
```

A mesma coisa vale para o *getter*:

```
...
NSLog(@"Dados do Contato: %@ %@ %@ %@ %@", contato.nome, ...);
```

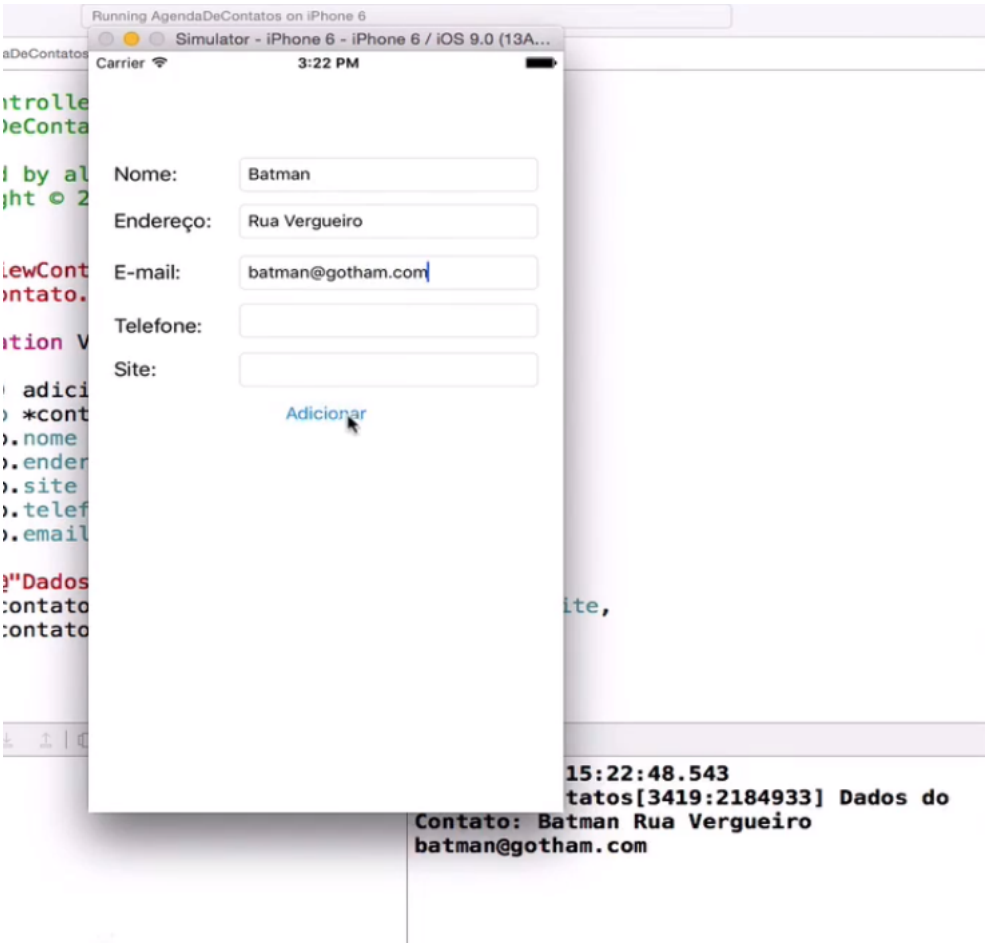
Se estamos usando a propriedade, podemos usar a notação com o ponto, a chamada **dot notation**. E seguiremos com o mesmo processo para os outros atributos. No "Contato.h":

```
@interface Contato : NSObject

@property NSString *nome;
@property NSString *endereco;
@property NSString *telefone;
@property NSString *site;
@property NSString *email;

@end
```

E no "ViewController.m":

```
-(IBAction) adiciona {
    Contato *contato = [Contato new];
    contato.nome = self.nome.text;
    contato.endereco = self.endereco.text;
    contato.site = self.site.text;
    contato.telefone = self.telefone.text;
    contato.email = self.email.text;

    NSLog(@"Dados do contato: %@ %@ %@ %@ %@", contato.nome, contato.endereco, contato.site, contato
}
```

Vamos simular novamente nossa aplicação (lembrando que podemos usar o atalho "Ctrl+R" para isso). Inserimos os dados e:



De fato funcionou! As informações estão aparecendo.

Então, revendo: sempre que queremos instanciar um objeto fazemos `Contato new` , ele cria através de mensagem, pois ele não é uma propriedade. A *dot notation* é usada apenas para propriedades. Cada propriedade será guardada dentro de "Contato.h" (para serem públicos), gerando o *getter*, o *setter* e o atributo, como por exemplo, `@property NSString *telefone;` .