Unidade 4

Aula 1

# View Controllers

Nas aulas anteriores aprendemos a compor a interface de usuário nos Storyboards sem ao menos ter que escrever uma linha de código, mas nem sempre isto será possível. Para criar aplicações dinâmicas e mais complexas precisaremos entender como funcionam as Views e View Controllers. Nesta aula iremos entender estes conceitos e aprender a interceptar os eventos de entrada dos nossos botões.

## Entendendo a hierarquia das Views

Objetivamente a View é representada pela classe UIView e define umas áreas retangulares que podemos posicionar na tela do nosso aplicativo. Nela podemos colocar outras UIView, criando uma estrutura de camadas de Views dentro da área de cada View.

A UIView é a classe mais primitiva das Views, mas podemos encontrar suas subclasses, como UIButton, UIImageView, UIControl que possuem funcionalidades e características específicas para cada uso. No decorrer do seu aprendizado você pode consultar a documentação oficial do iOS para entender o que cada subclasse faz (é uma variedade enorme de subclasses da UIView).

### Visão Geral

Em tempo de execução, um objeto de View lida com a prestação de qualquer conteúdo em sua área e também lida com todas as interações com esse conteúdo. Views com conteúdo mais sofisticado podem ser apresentadas por subclasses da UIView e implementam o desenho (apresentação) e códigos de manipulações dos eventos necessários para cada caso. A UIView faz parte do framework (ou grupo de classes) UIKit, que também inclui um conjunto de subclasses padrões que variam de simples botões até tabelas complexas. Por exemplo, um objeto UILabel um texto (string) na tela e um UIImageView desenha uma imagem.

Devido ao fato de que a View é a principal maneira com que seu aplicativo interage com o usuário, ela têm uma série de responsabilidades. Aqui estão apenas alguns:

* **Desenho e animação**
  + Visualizações desenhar o conteúdo em sua área retangular usando tecnologias como UIKit, Core Graphics, e OpenGL ES.
  + Algumas propriedades da View podem ser utilizadas para definirem animações desta View.
* **Gerenciamento de layout e subview**
  + Uma View pode conter zero ou mais subviews.
  + Cada View define o seu próprio comportamento de redimensionamento padrão em relação à sua superview.
  + Uma View pode definir o tamanho e posição de suas subviews conforme necessário.
* **Manipulação de eventos**
  + A View é um Responder e isto significa que pode lidar com (ou responde a) eventos de toque e outros eventos definidos pela classe UIResponder.
  + Views podem usar o método addGestureRecognizer(\_:) para instalar reconhecedores gesto para lidar com gestos comuns no smartphone.

Com as Views é possível incorporar outras Views, criando hierarquias visuais sofisticadas. Isto cria uma relação mãe-filha entre uma View que está sendo incorporada (conhecida como subview) e a View pai, que é a incorporadora (conhecida como superview). Normalmente, a área visível de uma subview não é podada dentro dos limites de sua superview, mas em iOS podemos utilizar a propriedade clipsToBounds para reverter este comportamento. Toda View Controller possui uma view principal, que pode conter um número indefinido de subviews mas não possui uma superview, pois ela está no ponto mais alto da hierarquia visual. Já que mencionamos aView Controller, vamos compreender seu funcionamento a seguir.

## Compreendendo a View Controller

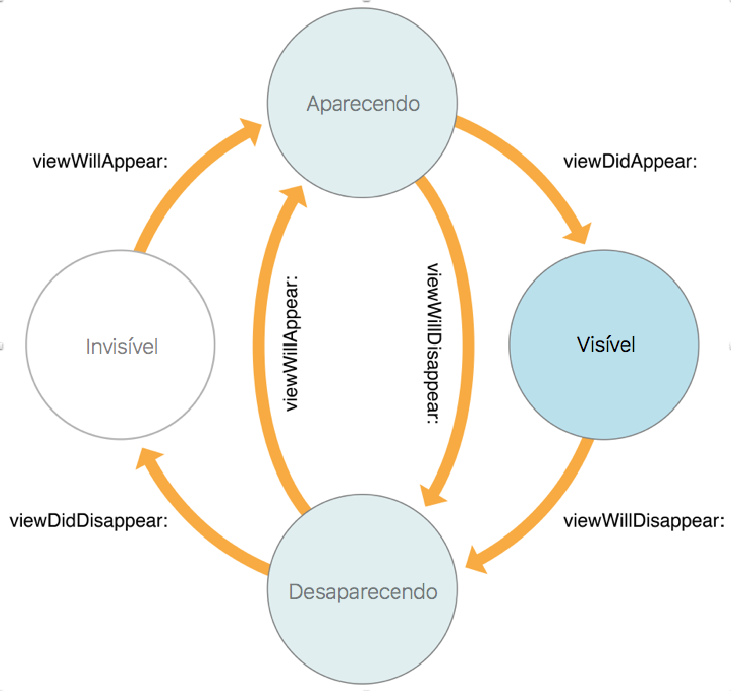
A classe UIViewController fornece a infra-estrutura apropriada para a gestão views dos seus aplicativos iOS. Um View Controller gere um conjunto de views que compõem uma parte da interface do usuário do seu aplicativo. É responsável por carregar e eliminar de tais views, por gerenciar interações entre estas views, e por mediar a interação entre quaisquer modelos de dados e as views. Resumidamente a View Controller controla e gerencia as views delegadas a ela. Comparando com o Android, podemos fazer um paralelo com a Activity.

### O ciclo de vida de uma View Controller

Um objeto de UIViewController (e suas subclasses) vêm com um conjunto de métodos que gerem a sua hierarquia de views. O iOS automaticamente chama estes métodos em momentos adequados, como em transições e mudanças de estados das View Controllers.

Quando criamos uma subclasse de UIViewController (sempre trabalharemos em alguma subclasse de UIViewController), ela herda os métodos definidos na superclasse e permite que adicionemos o comportamento que desejamos para cada um destes métodos. Isto porque esta estrutura de métodos está organizada no padrão de projeto Template Method, visto no curso de Android, e que é o mesmo padrão aplicado na Activity e seu ciclo de vida.

É importante entender que, para que você possa configurar certos comportamentos de views ou eliminá-las, é necessário estar adequado no método correto, ou seja, existe um método da View Controller apropriado para cada configuração de views a ser feita. Abaixo temos uma ilustração vinda da documentação oficial do iOS que exemplifica o ciclo de vida de uma View Controller e os métodos aqui em questão.



A UIViewController chama os métodos na sequência como se segue:

* **viewDidLoad()** – É chamado quando o conteúdo da view principal do View Controller (a que está no topo da hierarquia de visualização) é criado e carregado pelo storyboard. Este método destina-se a configurações iniciais. Entretanto, devido ao fato de que o smartphone possui recursos limitados, esta view pode ser destruída e reconstruída a depender do sistema operacional, então não é garantido que este método será chamado apenas uma vez no ciclo, mas o fato é que ele é o ponto inicial da criação de uma View Controller.
* **viewWillAppear()** – É destinado para realizar quaisquer operações que deseja-se que sejam realizadas quando a view principal se torna visível. Devido a possibilidade da visibilidade da view ser alterada por outras views, este método é chamado diversas vezes. Em uma explicação com nível de abstração mais alto, sempre que **a tela irá tornar-se visível**, este método é invocado, seja porque esta tela esta sendo criada, ou seja porque estamos retornando de outra tela.
* **viewDidAppear()** – Este é destinado para qualquer operação que você deseja que ocorra logo que a view se torna visível, como busca de dados ou mostrar animações. Pelo mesmo motivo retratado no **viewWillAppear()** este método pode ser invocado diversas vezes pela View Controller. Em uma explicação com nível de abstração mais alto, sempre que **a tela já está visível**, este método é invocado, e sempre após o método **viewWillAppear()**.

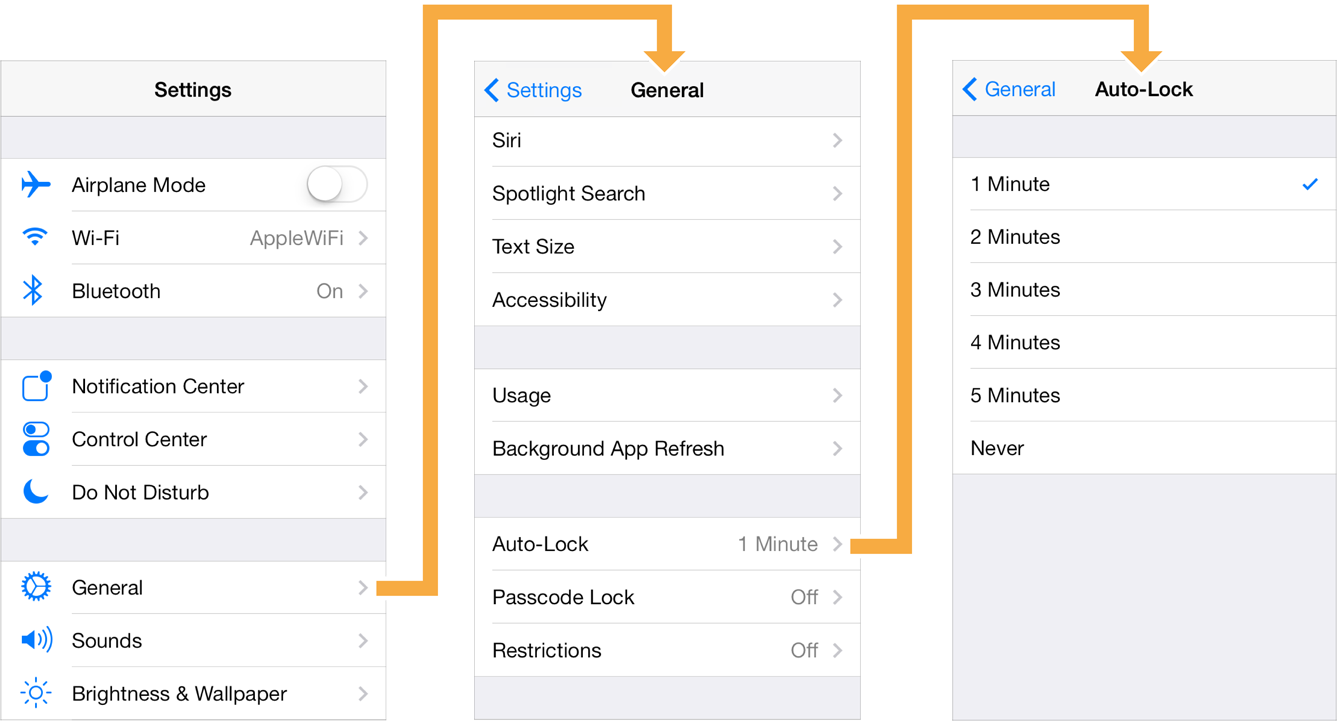
Estes são os métodos principais e que você provavelmente fará uso massivo dele, para os métodos **viewWillDesapear()** e **viewDidDisapear(),** mostrados no diagrama, vale o análogo dos métodos **viewWillAppear()** e **viewDidAppear()** respectivamente, ou seja, quando os dois últimos valem para quando a view irá **aparecer** os dois primeiros valem para quando a view irá **desaparecer.**

## UINavigationController

A classe UINavigationController implementa um View Controller que gere a navegação entre conteúdos. Esta interface de navegação torna possível apresentar os seus dados de forma eficiente e torna mais fácil para que o usuário navegue por estes conteúdos. Você pode utilizar esta classe exatamente como ela está, pois já oferece as funcionalidades mínimas para tais objetivos mencionados acima, mas você pode também criar subclasses para personalizar seu comportamento e aparência livremente.

As telas apresentadas por uma interface de navegação normalmente imitam a maneira como a hierarquia dos seus dados está organizada. A cada nível hierárquico, você fornece a tela apropriada (gerida por uma View Controller personalizada) para que o conteúdo deste nível (na estrutura dos seus dados) seja exibido.

A imagem abaixo mostra um exemplo da interface de navegação  do aplicativo Settings do simulador do iOS. A primeira tela apresenta ao usuário uma lista de preferências de aplicativos. Quando selecionamos um aplicativo, é revelado configurações e grupos de configurações deste aplicativo individualmente. Selecionando uma destas configurações é revelado mais configurações e por aí em diante. Para todas as telas, exceto para a tela raiz, é apresentado um botão de retorno, que permite que o usuário volte para cima na hierarquia.



## UITabBarController

A classe UITabBarController implementa um View Controller que gere uma interface de seleção no estilo rádio.

DICA: Uma seleção rádio é um tipo de seleção onde podemos escolher exclusivamente um item de uma lista que aplica este tipo de seleção.

Esta interface exibe guias ou abas em uma barra na parte inferior da tela. Em cada aba está relacionada um View Controller e, ao tocá-la, esta View Controller é exibida. Como a UINavigationController, podemos utilizar a UITabBarController exatamente como ela está e é possível fazer personalizações.

Cada aba de um Tab Bar Controller está associada a uma View Controller personalizada, como dito anteriormente. Quando o usuário seleciona uma aba específica, a Tab Bar Controller coloca a View Controller na raiz da estrutura de navegação, substituindo as View Controllers que estavam lá anteriormente. As Tab Bar Controllers são utilizadas comumente para apresentar tipos diferentes de informações ou apresentar a mesma informação utilizando estilos completamente diferentes. A figura abaixo mostra com o exemplo do nosso projeto um Tab Bar Controller. Cada aba representa um fluxo diferente do app, com informações não relacionadas entre si.

<imagem da tabbar>

Você nunca deve selecionar uma aba na Tab Bar Controller diretamente (via código). Isto é algo que o próprio iOS deve cuidar, mediante a interação do usuário. Para configurar as abas da Tab Bar Controller, basta associarmos uma View Controller. A ordem que as abas aparecerão na tela será definida de acordo com a ordem que as View Controllers são associadas a Tab Bar Controller. Ao definir as abas, também podemos definir qual será a aba inicialmente selecionada e por padrão a primeira aba (da esquerda para a direita) estará selecionada.

Outro ponto importante é que podemos associar Navigation Controllers, já que também são View Controllers. Sendo assim, a Tab Bar Controller gerencia um tipo de fluxo em abas e a Navigation Controller gerencia outro tipo de fluxo (sequencial) de View Controllers, então podemos fundir estes fluxos. Tendo esta possibilidade, é interessante saber que quando alternamos entre abas que possuem Navigation Controllers o estado destas são mantidos.