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

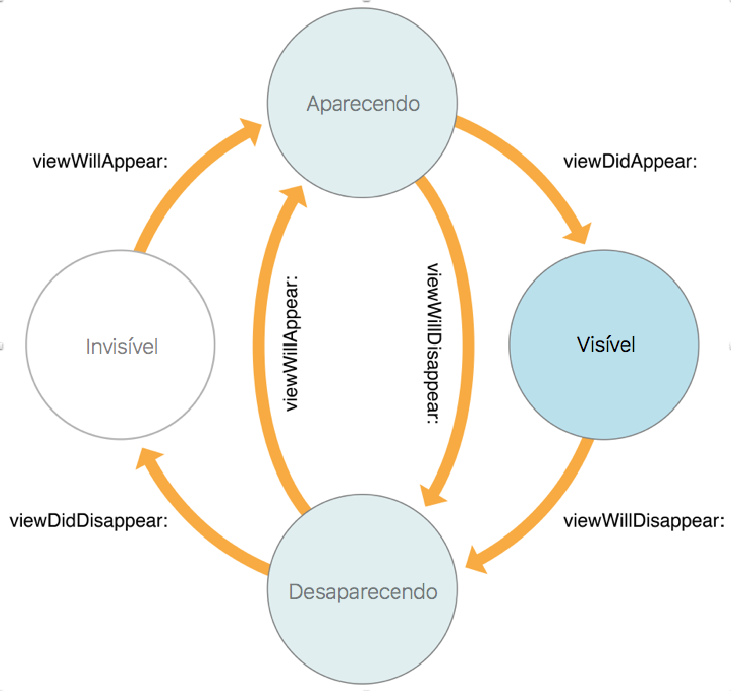
A classe UIViewController fornece a infra-estrutura apropriada para a gestão views dos seus aplicativos iOS. Um View Controller gere um conjunto de views que compõem uma parte da interface do usuário do seu aplicativo. É responsável por carregar e eliminar de tais views, por gerenciar interações entre 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()**.

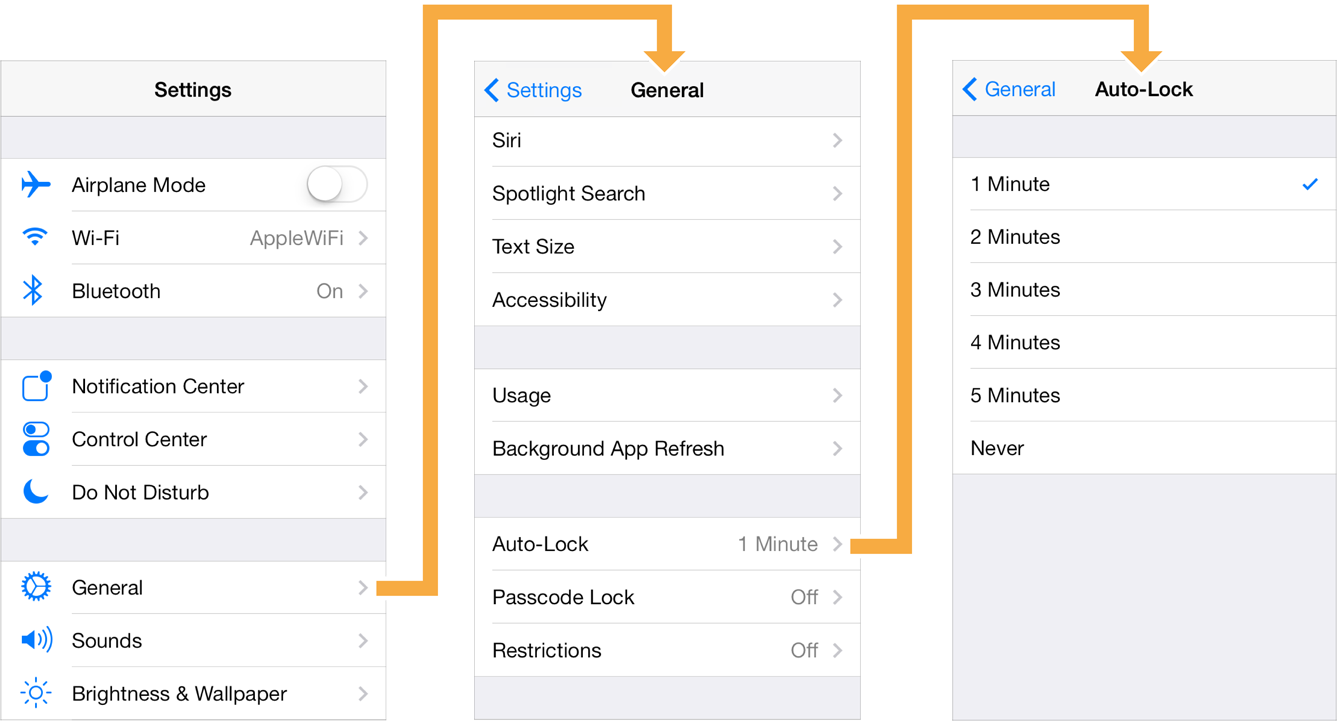
Estes são os métodos principais e que você provavelmente fará uso massivo dele, para os métodos **viewWillDesapear()** e **viewDidDisapear(),** mostrados no diagrama, vale o análogo dos métodos **viewWillAppear()** e **viewDidAppear()** respectivamente, ou seja, quando os dois últimos valem para quando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



### Definindo a navegação do app

## 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.

### Criando uma navegação em abas

## Resumo

Aula 2

# Listas com Table Views

Já sabemos que aplicativos predominantemente fazem uso excessivo de listas. Isto para apresentar dados dinâmicos na tela que podem variar ou simplesmente para organizar o conteúdo em um layout. A TableView é a classe responsável por colocar uma lista de células em nossa tela e, apesar da sua manipulação seja complicada no começo, ela é um componente poderoso para cuidar desta atividade.

## Table View Controllers

A Table View Controller, representada pela classe **UITableViewController**, é a classe convenientemente utilizada para gerenciar uma **Table View**. Ela possui algumas limitações, mas para uma introdução a exibição das listas, a Table View Controller é um bom começo. Além disso, ela ajuda na compreensão de alguns dos conceitos mais utilizados no iOS, como os **delegates** e **data sources.**

### Anatomia de uma Table View Controller

Como mencionado acima, a Table View Controller é uma classe de conveniência, isto é, ela não é fundamental no kit de desenvolvimento iOS, mas ela é feita de componentes existentes e cuidadosamente configurados para facilitar o uso. Uma representação da organização estrutural dos componentes de uma Table View Controller pode ser vista no desenho abaixo:

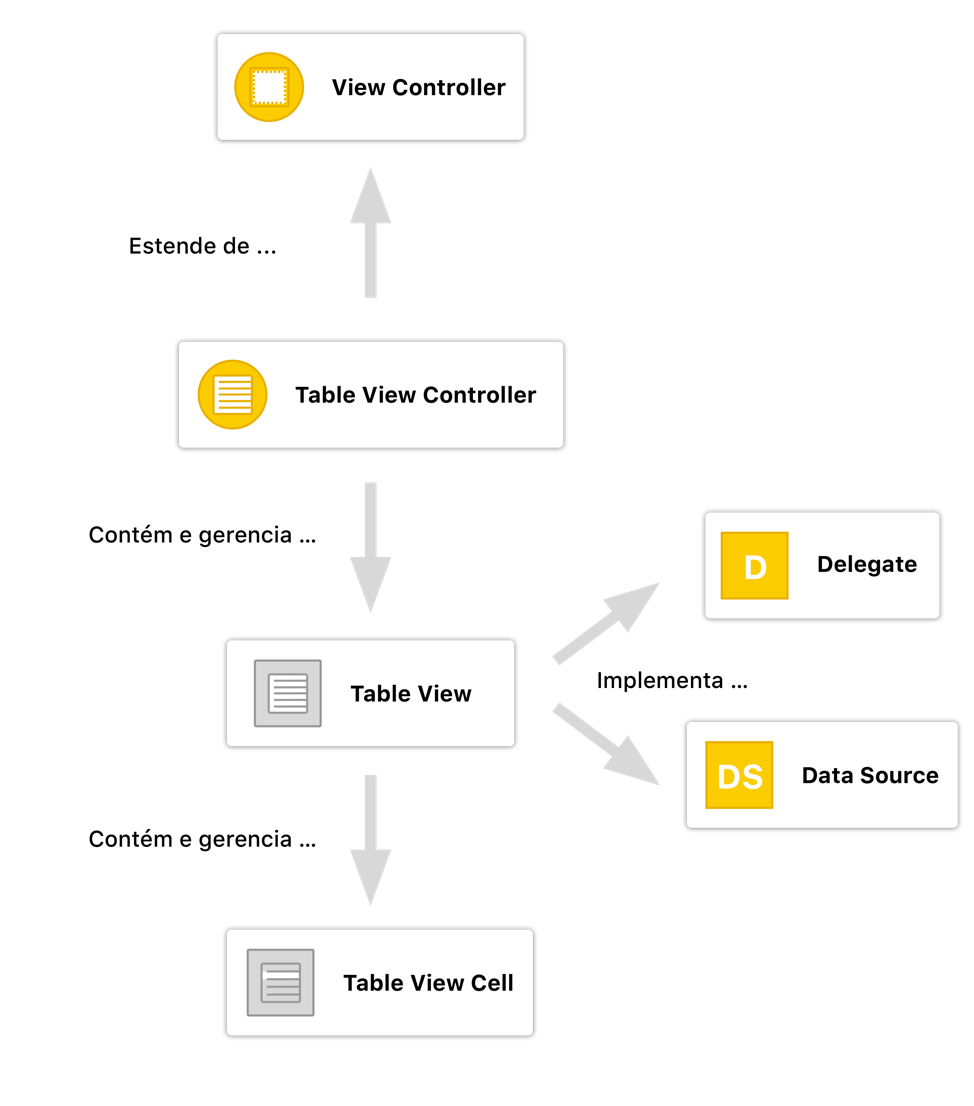


Figura - Esquema da estrutura de uma UITableViewController

## Table View

A Table View é o componente mais importante da UITableViewController. É nela que o gerenciamento de listas acontece realmente.

Para inserir dados em uma lista é preciso primeiro definir um layout para a Table View Cell, que na prática é uma linha (row) da tabela (lista). No UIKit já temos alguns layouts pré-definidos de Table View Cell que você pode utilizar, caso não esteja esperando nenhum visual mais complexo. Veja alguns exemplos de estilo de células disponíveis por padrão:

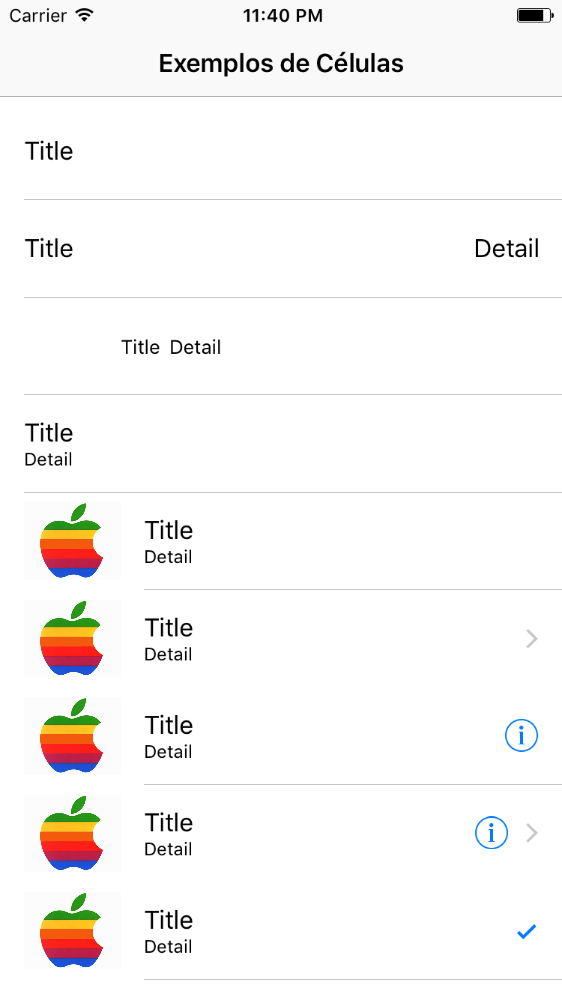


Figura - Exemplos de células padrões

Antes de continuar a explicar o funcionamento das Table Views vamos entender o que é uma Table View Cell.

### Table View Cell

A UITableViewCell é uma view suportada pela Table View. Para exibir qualquer coisa em uma Table View devemos exibir mediante a uma Table View Cell. Você pode optar por escolher células com estilos padrões ou personalizar a sua própria. Para existe a opção de prototipar o layout da célula direto no storyboard, onde a tabela está inserida. Feito isto basta criarmos uma subclasse de UITableViewCell e informar a UITableView que é esta célula que deverá ser renderizada.

Aprenderemos a fazer tudo isto na prática.

Voltando as Table Views, para definir a célula que será renderizada, o número de células o tamanho das células a quantidade de seções da tabela e outras coisas da configuração da tabela, é necessário implementar suas interfaces, ou melhor, protocolos, como é chamado no Swift.

Nesta aula iremos implementar alguns protocolos que levam o padrão de projeto Delegate, mas não entraremos em detalhes, pois ele será explicado com mais profundidade na unidade 5.

## Implementando uma Table View Controller

Para iniciar esta aula, abra o projeto no arquivo **Unidade\_4\_–\_Aula\_ 2\_–\_Exemplo\_1.zip**

Aula 3

# Animações básicas

Inserir animações nos nossos apps garante um visual fluído entre transições dos diferentes estados que nossa interface com o usuário pode se deparar. No iOS, animações são usadas exaustivamente para reposicionamento de views, mudança de seus tamanhos, inserção e remoção de views de uma hierarquia visual e ocultá-las.

## Por que utilizar animações?

Usar animações é opcional, mas ela garante uma experiência de uso por parte do usuário muito mais amigável do que um aplicativo que simplesmente troca os estados em um piscar de olhos.

Em computação, principalmente móvel, existe uma gigante área de estudo que estuda a fundo o comportamento e interação do homem com o computador. Se chama Interação Humano-Computador. Ela vai tão a fundo, que se preocupa em saber como o subconsciente do usuário reage a uma interface gráfica. Esta área dita como deverá ser a Experiência de Uso (UX) de um aplicativo. Com ela descobrimos que animações são de extrema importância.

Em um resumo, a utilização de animações garante um feedback visual mais natural e fluído, pois se comparado com a vida real, nada que sofre mudança, muda instantaneamente. Sempre há uma transição entre um estado e outro. Implementar esta semelhança com a vida real mexe com o poder cognitivo do usuário, e quanto menos ele precisar se adaptar, mas rápido ele vai aprender a utilizar seu app e o usará por mais tempo, pois não precisará lidar com frustrações e estranhezas que só o subconsciente dele percebe.

## O que pode ser animado

Como dizemos em aulas passadas, o kit de desenvolvimento do iOS possui um framework (ou biblioteca, ou SDK) chamado **UIKit**. Este framework possui todas as classes e tratamentos para lidar com a interface gráfica. É nela que encontramos os componentes UIButton, UIView, UIControl, UIViewController e todas as classes que possuem o prefixo **UI.** É daí que vem estes prefixos.

DICA: A linguagem utilizada para o desenvolvimento iOS era o Objective-C antes do Swift tomar o lugar. Nela não era possível definir **namespaces**, ou pacotes, então adotava-se o costume de adicionar um prefixo de duas letras maiúsculas no nome da classe para identificar o contexto que ela estava inserida. A cada atualização do Swift estes prefixos estão sendo deixados para trás, por exemplo a classe String e Dictionary, eram chamadas antigamente de NSString e NSDictionary pois faziam parte do “pacote” (entre aspas pois não existia pacotes mas separava-se as classes via prefixo) **NeXSTEP,** empresa criadora do Objective-C, fundada por Steve Jobs após ter sido demitido da Apple. A NeXSTEP estava ganhando espaço no mercado, enquanto a Apple estava cada vez mais na pior, foi então que a NeXSTEP foi agregada na Apple e Steve ganhou de volta o seu lugar de CEO.

No kit de desenvolvimento do iOS também temos o framework **Core Animation** que possui os tratamentos específicos e de mais baixo nível para manipulação de animações.

Ambos, UIKit e Core Animation suportam animações, mas o nível de suporte varia entre os dois. No UIKit, as animações são executadas por objetos de UIView. Views suportam um conjunto básico de animações, que felizmente cobrem a maioria das tarefas comuns que iremos nos deparar. Podemos animar as seguintes propriedades das Views.

* **frame:** Modifique esta propriedade para alterar o tamanho e posição da view relativamente a ao sistema de coordenadas de sua superview.
* **bounds:** Modifique esta propriedade para alterar o tamanho da view
* **center:** Modifique esta propriedade para mudar a posição de uma view, considerando o ponto de seu centro, relativamente ao sistema de coodenadas de sua superview
* **transform:** Modifique esta propriedade para escalar, rotacionar ou transladar uma view relativamente ao seu próprio centro.
* **alpha:** Modifique esta propriedade para mudar gradualmente a transparência de uma view
* **backgroundColor:** Modifique esta propriedade para mudar a cor de fundo de uma View.
* **contentStretch:** Modifique esta propriedade para mudar a maneira que uma view preenche os espaços disponíveis.

## Como animar as Views

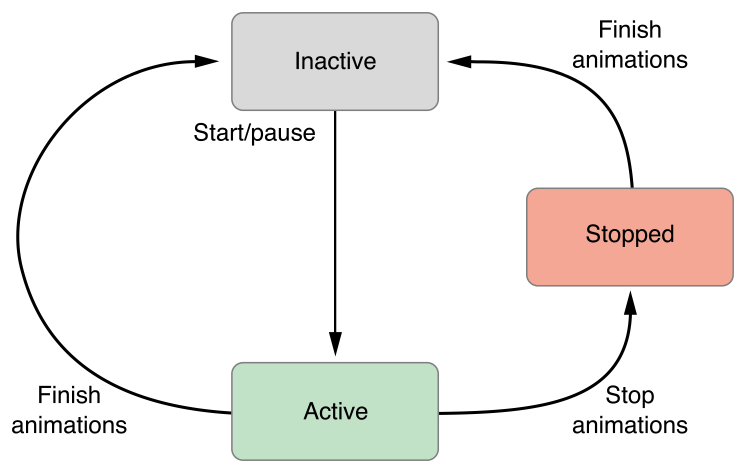
Existem duas maneiras de animar views. A primeira é utilizando métodos da UIView, que é utilizada para animações mais simples e a segunda, para animações mais complexas, utilizando as classes disponíveis no iOS 10: UIViewAnimating e UIViewPropertyAnimator

<inserir pequeno tutorial da maneira convencional da UIView.animate(with)>

### UIViewAnimating

A UIViewAnimating é um novo protocolo introduzido no iOS 10 para nos dar a possibilidade de animarmos views de uma forma muito mais interativa. Ela define alguns métodos a implementar por parte dos animadores. Os animadores são definidos pela classe UIViewPropertyAnimator. Estes métodos implementam o controle de fluxo da animação, dando a opção de **iniciar, paular e cancelar** animações.

Quando criamos um animador UIViewPropertyAnimator, este começa no estado **inativo** e para conseguirmos coloca-lo em execução, utilizamos os métodos **startAnimation()** e **pauseAnimation()** que vão leva-lo para o estado **ativo.** Com o método **stopAnimation()** cancelamos a animação em execução, e as propriedades do objeto animado conservam o exato estado em que a animação o deixou. Veja abaixo o ciclo de vida de uma animação.



A animação possui algumas propriedades importantes:

* **fractionComplete:** Mostra a porcentagem do avanço da animação (em um valor entre 0.0 e 1.0) em um determinado instante de tempo.
* **isReversed:** Indica se a animação está sendo executada ao contrário.
* **state:** Nos indica o estado UIViewAnimatingState atual da animação (ativo, inativo ou cancelado)
* **isRunning:** Nos permite identificar se a animação está em execução ou não em um instante de tempo.

Possui também alguns métodos importantes:

* **addAnimations:** permite incluir um bloco (ou closure, que iremos explicar em detalhes na unidade 5) com as animações que desejamos realizar e uma duração desta animação indicada na propriedade **duration** do animador. Esta é uma chamada assíncrona que não retorna nenhum valor, ou seja, não precisamos esperar a animação terminar para realizar outras tarefas no app, pois ela irá executar em paralelo.
* **addAnimations:delayfactor:** Este é semelhante ao método anterior, com a diferença que podemos incluir via parâmetro um delay, ou **atraso** entre as animações. Este atraso será inserido quando a animação que esta sendo adicionada começar a sua execução. Se introduzirmos um valor entre 0.0 e 1.0, ele se multiplicará pelo valor da propriedade **duration** do animador.