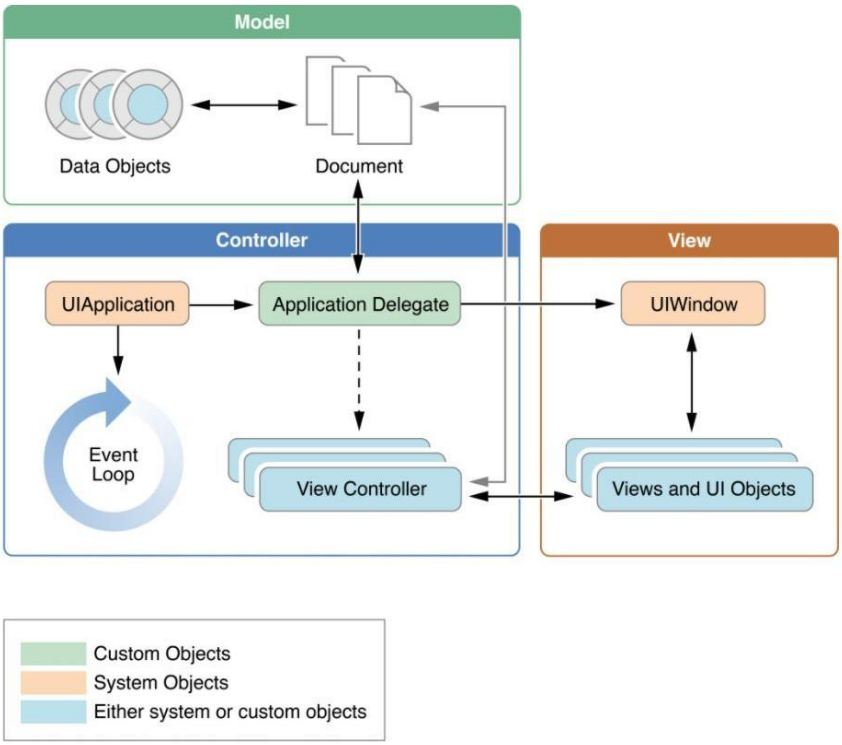
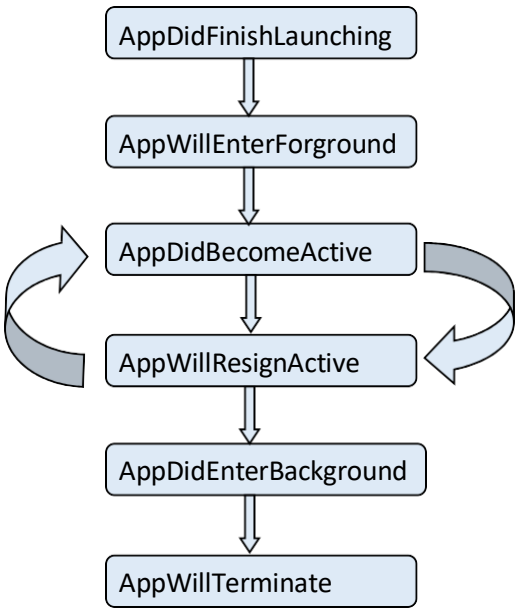
iOS Application Life Cycle

At the heart of every iOS app is the UIApplication object, whose job is to facilitate the interactions between the system and other objects in the app.



Application lifecycle helps to understand overall app behaviour. The main point of entry into iOS apps is **UIApplicationDelegate**. **UIApplicationDelegate** is a protocol that your app has to implement to get notified about user events such as app launch, app goes into background or foreground, app is terminated, a push notification was opened, etc.



When an iOS app is launched the first thing called is

application: willFinishLaunchingWithOptions:-> Bool. This method is intended for initial application setup. Storyboards have already been loaded at this point but state restoration hasn't occurred yet.

Launch

application: didFinishLaunchingWithOptions: -> Bool is called next. This callback method is called when the application has finished launching and restored state and can do final initialization such as creating UI.

applicationWillEnterForeground: is called after application: didFinishLaunchingWithOptions: or if your app becomes active again after receiving a phone call or other system interruption.

applicationDidBecomeActive: is called after applicationWillEnterForeground: to finish up the transition to the foreground.

Termination

applicationWillResignActive: is called when the app is about to become inactive (for example, when the phone receives a call or the user hits the Home button).

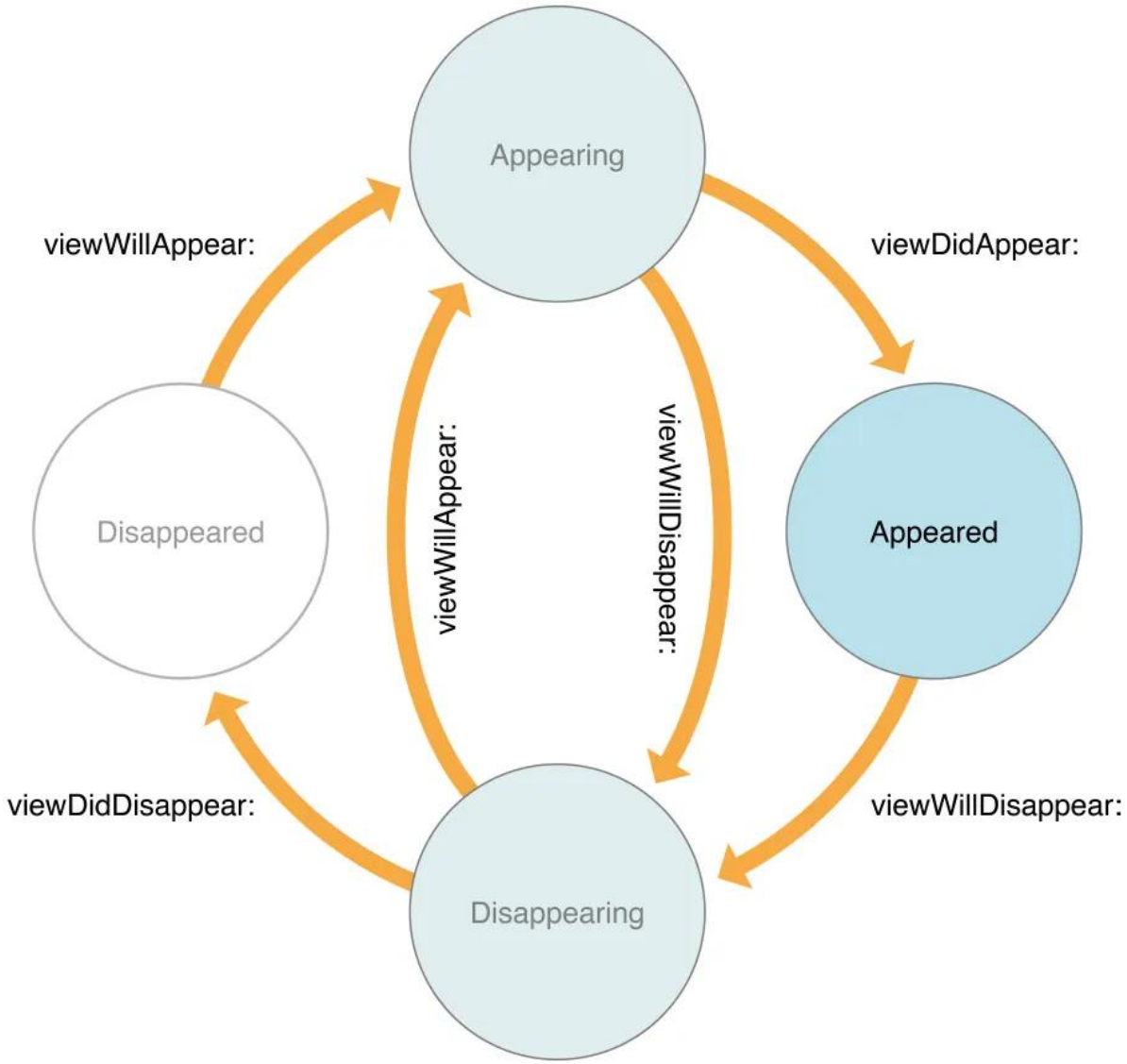
applicationDidEnterBackground: is called when your app enters a background state after becoming inactive. You have approximately five seconds to run any tasks you need to back things up in case the app gets terminated later or right after that.

applicationWillTerminate: is called when your app is about to be purged from memory. Call any final cleanups here.

Both application: willFinishLaunchingWithOptions: and application:didFinishLaunchingWithOptions: can potentially be launched with options identifying that the app was called to handle a push notification or url or something else. You need to return true if your app can handle the given activity or url.

ViewController Life Cycle

In iOS development, the view controllers are the foundation of the Application's internal structure. The View Controller is the parent of all the views present on a storyboard. Each UIKit application has at least one ViewController. It facilitates the transition between various parts of the user interface.



Functions are called in the following order when a View Controller is added to the view hierarchy:

1. init()
2. loadView()
3. viewDidLoad()
4. viewWillAppear()
5. viewIsAppearing()
6. viewWillLayoutSubviews()
7. viewDidLayoutSubviews()

8. viewDidAppear()
9. viewWillTransition()
10. viewWillDisappear()
11. viewDidDisappear()
12. deinit()

1. **init() :** This is the designated initializer for UIViewController. It's called when the view controller is being initialized.
2. **loadView() :** This method is responsible for creating or loading the view hierarchy of the view controller. You override this method when you want to create your view hierarchy programmatically without using a storyboard or a nib file. If you don't override this method, the view controller will automatically load its view from a nib or storyboard. If you want to perform any additional initialization of your views, do so in the viewDidLoad() method.
3. **viewDidLoad() :** This method is called after the view controller has loaded its view hierarchy into memory. This method is called regardless of whether the view hierarchy was loaded from a nib file or created programmatically in the loadView() method. This method is called only once during the view controller's lifecycle.
4. **viewWillAppear() :** This method is called use before the View Controller is added to the view hierarchy. The frame and bounds of the view controller are all set. This function is called each time the view is presented on screen.
5. **viewIsAppearing() :** The system calls this method once each time a view controller's view appears after the viewWillAppear(_:) call.
6. **viewWillLayoutSubviews() :** This method is called just before the view controller's view is about to update its layout to fit the current device's orientation or size. You can override this method to make adjustments to your view's layout before it occurs. For example, you can override this method to fix issues when the user changes orientation to landscape and gets back because often occurs issues when the users change application orientation.
7. **viewDidLayoutSubviews() :** This method is called after the view controller's view has finished updating its layout. You can use this method to perform additional layout-related tasks or animations that depend on the final layout of your view.
8. **viewDidAppear() :** This method is called when the view has appeared on the screen. It's often used to start animations, load additional data, or perform other tasks that should happen after the view is fully visible.
9. **viewWillTransition():** This method is called when the view controller's trait collection is changed by its parent. UIKit calls this method before changing the size of a presented view controller's view. You can override this method in your own objects and use it to perform additional tasks related to the size

change.

10. **viewWillDisappear() :** This method is called just before the view is about to disappear from the screen, typically when you navigate away from the current view controller. It's often used to perform cleanup tasks, save data, or pause ongoing processes that are specific to the current view.
11. **viewDidDisappear() :** This method is called after the view has disappeared from the screen. It's often used to perform additional cleanup tasks or to release resources that were allocated when the view was visible. For example, you might release memory-intensive resources, unregister from notifications, or stop observing changes.
12. **deinit() :** This method is automatically called when an instance of a class is being deallocated or destroyed. For view controllers, deinit() can be useful for cleaning up any resources that were allocated during the view controller's lifecycle but are no longer needed. For example, you might use deinit() to release memory, close network connections, or perform any other necessary cleanup before the view controller is removed from memory.