Developing Applications for iOS

Lecture 8:
View Controller Lifecycle and UIKit

Prof. Dr. Radu Ionescu
raducu.ionescu@gmail.com
Faculty of Mathematics and Computer Science
University of Bucharest

Content

- View Controller Lifecycle

  When your controller hears about things and what you should do about it.

- Image View

  Kind of like “`UILabel`” for images.

- Web View

  Complete browser in a view.

- Scroll View

  Provides a moving “viewport” on a rectangular area that has views (the scroll view’s subviews) in it.

View Controller Lifecycle

View Controllers have a Lifecycle

- A sequence of messages is sent to them as they progress through it.

Why does this matter?

- You very commonly override these methods to do certain work:

```
func viewDidLoad()
func viewWillAppear(_ animated: Bool)
func viewDidAppear(_ animated: Bool)
func viewWillDisappear(_ animated: Bool)
func viewDidDisappear(_ animated: Bool)
```

and many other methods.

View Controller Lifecycle

We have already talked about the first part of the lifecycle

- Creation
- This is done mostly either via a segue or storyboard's:

  `instantiateViewController(withIdentifer:)`
- Because of this, we rarely have to override `UIViewController`'s designated initializer in iOS.
- Another option is `awakeFromNib`, but we rarely do that either.
- If you cannot define your views in a storyboard or a nib file, override the `loadView` method to manually instantiate a view hierarchy and assign it to the `view` property.
- There are better methods to initialize in: `viewDidLoad`, `viewWillAppear`, `viewDidAppear`.

View Controller Lifecycle

- After instantiation and outlet-setting, `viewDidLoad` is called:

```
func viewDidLoad()
```

- This method is called regardless of whether the view hierarchy was loaded from a nib file or created programmatically in the `loadView` method.
- This is an exceptionally good place to put a lot of setup code.
- But be careful because the geometry of your view (its `bounds`) is not set yet!
- If you need to initialize something based on the geometry of the view, use the next method.

View Controller Lifecycle

- Just before the view appears on screen, you get notified:

```
func viewWillAppear(_ animated: Bool)
```

- When this is called, your `bounds` has been set (via your `frame` by your `superview`).
- Your view will probably only get “loaded” once, but it might appear and disappear a lot. So don’t put something in this method that really wants to be in `viewDidLoad`.
- Otherwise, you might be doing something over and over unnecessarily.
- Use this to optimize performance by waiting until this method (i.e. just before view appears) to kick off an expensive operation (might have to put up a spinning “loading” icon though).
- This method is for geometry-related initialization and lazy execution for performance.

View Controller Lifecycle

- And you get notified when you will disappear off screen too. This is where you put “remember what’s going on” and cleanup code.

```
override func viewWillDisappear(_ animated: Bool)
{
    super.viewWillDisappear(animated)
    // Call super in all the viewWill/Did... methods.

    /* Let’s be nice to the user and remember
     * the scroll position they were at */
    self.rememberScrollPosition()
    // We will have to implement this, of course.

    /* Do some other clean up now that we have
     * been removed from the screen. */
    self.saveDataToPermanentStore()
    /* But be careful not to do anything
     * time-consuming here, or app will be sluggish.
     * Do it in the did version or in a thread. */
}
```

View Controller Lifecycle

- There are “did” versions of both of the previous methods.
- You can override this method to perform additional tasks associated with presenting the view:

```
func viewDidAppear(_ animated: Bool)
```

- You can override this method to perform additional tasks associated with dismissing or hiding the view:

```
func viewDidDisappear(_ animated: Bool)
```

- If you override these methods, you must call super at some point in your implementation.

View Controller Lifecycle

Frame changed?

- Here's a good place to layout subviews manually (if struts and springs are not enough):

```
func viewWillLayoutSubviews()
```

```
func viewDidLayoutSubviews()
```

- When a view's `bounds` change, the `view` adjusts the position of its `subviews`. Your View Controller can override these methods to make changes before/after the view lays out its subviews.
- Called any time a view's `frame` changed and its `subviews` were thus re-layed out.
- For example, autorotation.
- You can reset the `frame`s of your `subviews` here.

View Controller Initialization

Creating a `UIViewController` from a .xib file

- This is the old, iOS 4 way. Not covered in this class.
- You create a .xib file with the same name as your `UIViewController` subclass. Then use `init` to create it.
- Designated initializer (only if you need to override it; use `init` otherwise):

```
init(nibName nibNameOrNil: String?,
     bundle nibBundleOrNil: Bundle?)
```

View Controller Initialization

Creating a `UIViewController`'s UI in code (no .xib, no storyboard)

- Override the method `func loadView()` and set `self.view`.
- Do **NOT** implement `loadView` if you use a storyboard/.xib to create the `UIViewController`.
- Do **NOT** set `self.view` anywhere else besides in `loadView`.
- Do **NOT** implement `loadView` without setting `self.view` (i.e. you must set `self.view` here).
- You should **NEVER** call this method directly. The View Controller calls this method when its `view` property is requested but is currently `nil`.

View Controller Initialization

- Avoid `awakeFromNib` if possible.
- It is an acceptable place to initialize stuff for a `UIViewController` from a storyboard /.xib.
- You can also put stuff in `viewDidLoad`, `viewWillAppear` or the segue preparation code (`prepare(forSegue:sender:)`) instead.

UIView's frame

Who's responsible for setting a `UIView`'s `frame`?

- The object that puts the `UIView` in a view hierarchy.
- In Interface Builder, you set all view's `frame`s graphically.
- You do this by dragging on the little handles (or from Size Inspector).

What about the frame passed to `init(frame:)`?

- If you're putting it into a view hierarchy right away, pick the appropriate `frame`.
- If you are not, then it doesn't really matter what `frame` you choose (but avoid `CGRect.zero`).
- The code that eventually does put you in a view hierarchy will have to set the `frame`.

UIView's frame

Setting frames in viewDidLoad

- Recall that your final bounds are not set in viewDidLoad.
- If you create views in code in viewDidLoad, pick sensible frames based on the view's bounds then.
- But be sure to set struts/springs (UIView's autoresizingMask property).
- You specify the value of this mask by listing the constants described in UIViewAutoresizing in an array:

```
view.autoresizingMask = [.flexibleWidth, .flexibleHeight]
```

- Think of adding something in viewDidLoad as the same as laying it out in Interface Builder.
- In both cases, you have to anticipate that the top-level view's bounds will be changed.

UIImageView

A `UIView` subclass which displays a `UIImage`

- We covered how to create a `UIImage` in the lecture on Views.

How to set the `UIImageView`'s `UIImage`

Use this initializer:

```
init(image: UIImage?)
```

- It will set its `frame` size to match the `image` size. Note that the designated initializer is still `init(frame:)` (inherited from the `UIView` superclass).
- You can also set this property (but it will not adjust the `frame` size):

```
var image: UIImage? { get set }
```

UIImageView

Remember UIView's contentMode property?

- It can be set to UIViewContentMode{Redraw,Top,Left,...}.
- UIViewContentModeScaleToFill is the default.
- Determines where the image appears in the UIImageView's bounds and whether it is scaled.

Highlighted image

```
var highlightedImage: UIImage? { get set }
var isHighlighted: Bool { get set }
```

UIImageView

Sequence of images forming an animation

- For animating more images, set the following property:

```
var animationImages: [UIImage]? { get set }
```

- `UIImageView` class provides controls to set the duration and frequency of the animation:

```
var animationDuration: TimeInterval { get set }
var animationRepeatCount: Int { get set }
```

The default value is 0, which specifies to repeat the animation indefinitely.

- You can also start and stop the animation:

```
func startAnimating()
func stopAnimating()
var isAnimating: Bool { get }
```

The `startAnimating` method always starts the animation from the first image in the list.

WKWebView

A full Internet browser in a `UIView`

- Can use it not only to take your users to websites, but to display PDFs, for example.
- Built on WebKit, an open source HTML rendering framework (started by Apple).
- Supports JavaScript:

```
func evaluateJavaScript(_ javaScriptString: String,
    completionHandler: ((Any?, Error?) -> Void)? = nil)
```

- Example:

```
webView.evaluateJavaScript("document.getElementById('Id').
                            innerText")
{ (result, error) in

   if error != nil
   {
      print(result!)
   }
}
```

WKWebView

- WKWebView automatically scales the content to fit the screen, but you can also change the zoom level:

```
var allowsMagnification: Bool { get set }
```

Default is `false`. If `true`, then the user can use magnify gestures to change the web view's magnification. This can also be done programmatically:

```
func setMagnification(_ magnification: CGFloat,
                      centeredAt point: CGPoint)
```

- Property to get the scroll view it uses:

```
var scrollView: UIScrollView { get }
```

Can now set properties in the scroll view to control the scrolling behavior of the web view.

- If you allow the user to move back and forward through the webpage history, then you can use the `goBack` and `goForward` methods as actions for buttons.

WKWebView

- Three ways to load up content:

```
func load(_ request: URLRequest) -> WKNavigation?

func loadHTMLString(_ string: String,
                    baseURL: URL?) -> WKNavigation?

func load(_ data: Data,
          mimeType MIMEType: String,
          characterEncodingName: String,
          baseURL: URL) -> WKNavigation?

func loadFileURL(_ URL: URL,
                 allowingReadAccessTo readAccessURL: URL)
                 -> WKNavigation?
```

- Base URL is the “environment” to load resources out of (i.e., it’s the base URL for relative URL’s in the data or HTML string).
- MIME type (Multimedia Internet Mail Extension) says how to interpret the passed-in data.

  Standard way to denote file types (like PDF). Think “e-mail attachments” (that’s where the name MIME comes from).

URLRequest

`URLRequest`

```
init(url: URL,
     cachePolicy: URLRequest.CachePolicy = default,
     timeoutInterval: TimeInterval = default)
```

`URL`

- Basically like a `String`, but enforced to be “well-formed”.
- Examples: file://... or http://...
- In fact, it is the recommended way to specify a file name in the iOS API.

`URLRequest.CachePolicy`

- Ignore local cache; ignore caches on the Internet; use expired caches; use cache only (don’t go out onto the Internet); use cache only if validated.

WKWebView

WKWebViewDelegate

- You can set the `uiDelegate` property to an object conforming to the `WKWebViewDelegate` protocol if you want to control the opening of new windows, augment the behavior of default menu items displayed when the user clicks elements, and perform other user interface-related tasks.
- Methods in the `WKWebViewDelegate` are:

```
func webView(_ webView: WKWebView,
   createWebViewWith configuration: WKWebViewConfiguration,
   for navigationAction: WKNavigationAction,
   windowFeatures: WKWindowFeatures) -> WKWebView?

func webView(_ webView: WKWebView,
   runJavaScriptConfirmPanelWithMessage message: String,
   initiatedByFrame frame: WKFrameInfo,
   completionHandler: @escaping (Bool) -> Void)

func webViewDidClose(_ webView: WKWebView)
```

WKWebView

WKNavigationDelegate

- You can set the `navigationDelegate` property to an object conforming to the `WKNavigationDelegate` protocol if you want to implement custom behaviors that are triggered during a web view's process of accepting, loading, and completing a navigation request.
- Methods in the `WKWebViewDelegate` are:

```
func webView(_ webView: WKWebView,
             didCommit navigation: WKNavigation!)
func webView(_ webView: WKWebView,
             didFail navigation: WKNavigation!,
             withError error: Error)
func webView(_ webView: WKWebView,
             didFinish navigation: WKNavigation!)
```

UIScrollView

How do you create one?

- Just like any other `UIView`. Drag out in a storyboard or use `init(frame:)`.
- Or select a `UIView` in your storyboard and choose “Embed In > Scroll View” from Editor menu.
- Or add your “too big” `UIView` using `addSubview` like this:

```
let bigImage = UIImage(named: "bigImage.jpg")
let imageView = UIImageView(image: bigImage)
// now imageView.frame.size is equal to bigImage.size

let frame = CGRect(x: 0,
                   y: 0,
                   width: self.view.frame.size.width,
                   height: self.view.frame.size.width)
let scrollView = UIScrollView(frame: frame)

scrollView.addSubview(imageView)
// add more subviews if you want
```

UIScrollView

- All of the `frame`s of the `subview`s will be in the `UIScrollView`'s content area's coordinate system.
- (0,0) is the upper left corner of the scroll view.
- Width and height are given by `contentSize.width` and `contentSize.height`.
- Don't forget to set the `contentSize`!
- Common mistake is to do the previous lines of code (or embed in Interface Builder) and forget to say:

```
scrollView.contentSize = imageView.bounds.size
```

UIScrollView

Scrolling programmatically

```
func scrollRectToVisible(_ rect: CGRect,
                         animated: Bool)
```

Other things you can control in a scroll view

- Control whether scrolling is enabled through the `scrollEnabled` property.
- Lock scroll direction to user's first "move" by setting the `directionalLockEnabled` property.
- The style of the scroll indicators are set via the `indicatorStyle` property. (call `flashScrollIndicators` when your scroll view appears).
- Whether the actual content is "inset" from the scroll view's content area (`contentInset` property).
- Note that `UIScrollView` is the superclass of several UIKit classes including `UITableView` and `UITextView`.

UIScrollView

Zooming

- All `UIView`s have a property (`transform`) which is an affine transform (translate, scale, rotate). Scroll view simply modifies this transform when you zoom.
- Zooming is also going to affect the scroll view's `contentSize` and `contentOffset`.
- Will not work without minimum / maximum zoom scale being set

```
scrollView.minimumZoomScale = 0.5 // half its normal size
scrollView.maximumZoomScale = 2.0; // twice its normal size
```

- Will not work without delegate method to specify view to zoom:

```
func viewForZooming(in scrollView: UIScrollView) -> UIView?
```

If your scroll view only has one subview, you return it here.

More than one subview? It's up to you then.

UIScrollView

Zooming programmatically

```
var zoomScale: CGFloat { get set }
func setZoomScale(_ scale: CGFloat, animated: Bool)
func zoom(to rect: CGRect, animated: Bool)
```

Lots and lots of delegate methods!

- The scroll view will keep you up to date with what's going on.
- Example: delegate method will notify you when zooming ends

```
func scrollViewDidEndZooming(_ scrollView: UIScrollView,
                             with view: UIView?,
                             atScale scale: CGFloat)
```

- If you redraw your view at the new `scale`, be sure to reset the affine transform back to identity.

Next Time

iDevice Capabilities:

- Core Location: GPS + Compass
- Accelerometer
- Map Kit