Lecture 8 Demo Code: Cassini Multithreading

Objective

Included below is the source code for the demo in lecture. It is provided under the same Creative Commons licensing as the rest of CS193p's course materials. The code which has not changed since the previous lecture is included but is grayed out. And here is the complete project.

```
//
//  ImageViewController.swift
//  Cassini
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class ImageViewController: UIViewController
{
    // MARK: Model

    var imageURL: URL? {
        didSet {
            image = nil
            if view.window != nil { // if we're on screen
                fetchImage()        // then fetch image
            }
        }
    }
```

```
// MARK: Private Implementation

// when we dropped this spinner into our storyboard
// it initially put it as a subview of our scrollView!
// we used the Document Outline to drag it out to be a sibling of scrollView
// instead of a subview
@IBOutlet weak var spinner: UIActivityIndicatorView!

private func fetchImage() {
    if let url = imageURL {
        // we're going to start something on another queue soon
        // so let's start a spinner going in the UI to let the user know
        // we'll stop this any time an image actually gets set
        // (we'd never want the spinner and an image appearing at the same time!)
        spinner.startAnimating()
        // try? Data(contentsOf:) blocks the thread it is on
        // we are currently on the main thread
        // so we must dispatch that call off to a background queue
        // we'll use one of the shared, global, concurrent background queues
        DispatchQueue.global(qos: .userInitiated).async { [weak self] in
            let urlContents = try? Data(contentsOf: url)
            // now that we're back from blocking
            // are we still even interested in this url (i.e. does it == self.imageURL)?
            if let imageData = urlContents, url == self?.imageURL {
                // now we want to set the image in the UI
                // but we are not on the main thread right now
                // so we are not allowed to do UI
                // no problem: just dispatch the UI stuff back to the main queue
                DispatchQueue.main.async {
                    self?.image = UIImage(data: imageData)
                }
            }
        }
    }
}

// MARK: View Controller Lifecycle

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    if image == nil { // we're about to appear on screen so, if needed,
        fetchImage()  // fetch image
    }
}
```

```
    // MARK: User Interface

    @IBOutlet weak var scrollView: UIScrollView! {
        didSet {
            // to zoom we have to handle viewForZooming(in scrollView:)
            scrollView.delegate = self
            // and we must set our minimum and maximum zoom scale
            scrollView.minimumZoomScale = 0.03
            scrollView.maximumZoomScale = 1.0
            // most important thing to set in UIScrollView is contentSize
            scrollView.contentSize = imageView.frame.size
            scrollView.addSubview(imageView)
        }
    }

    fileprivate var imageView = UIImageView()

    private var image: UIImage? {
        get {
            return imageView.image
        }
        set {
            imageView.image = newValue
            imageView.sizeToFit()
            // careful here because scrollView might be nil
            // (for example, if we're setting our image as part of a prepare)
            // so use optional chaining to do nothing
            // if our scrollView outlet has not yet been set
            scrollView?.contentSize = imageView.frame.size
            // now that we've set an image
            // stop any spinner that exists from spinning
            spinner?.stopAnimating()
        }
    }
}

// MARK: UIScrollViewDelegate
// Extension which makes ImageViewController conform to UIScrollViewDelegate
// Handles viewForZooming(in scrollView:)
// by returning the UIImageView as the view to transform when zooming

extension ImageViewController : UIScrollViewDelegate
{
    func viewForZooming(in scrollView: UIScrollView) -> UIView? {
        return imageView
    }
}
```

```
//
//  CassiniViewController.swift
//  Cassini
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class CassiniViewController: UIViewController, UISplitViewControllerDelegate
{
    // for the UISplitViewControllerDelegate method below to work
    // we have to set ourself as the UISplitViewController's delegate
    // (only we can be that because ImageViewControllers come and goes from the heap)
    // we could probably get away with doing this as late as viewDidLoad
    // but it's a bit safer to do it as early as possible
    // and this is as early as possible
    // (we just came out of the storyboard and "awoke"
    // so we know we are in our UISplitViewController by now)
    override func awakeFromNib() {
        super.awakeFromNib()
        self.splitViewController?.delegate = self
    }

    // MARK: - Navigation

    // we interpret our segue identifier
    // as a key into the DemoURL.NASA Dictionary (a [String:URL])
    // if we find a URL for that key
    // then we just set the public imageURL of the ImageViewController to it
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if let url = DemoURL.NASA[segue.identifier ?? ""] {
            if let imageVC = (segue.destination.contents as? ImageViewController) {
                imageVC.imageURL = url
                imageVC.title = (sender as? UIButton)?.currentTitle
            }
        }
    }

    // we "fake out" iOS here
    // this delegate method of UISplitViewController
    // allows the delegate to do the work of collapsing the primary view controller (the master)
    // on top of the secondary view controller (the detail)
    // this happens whenever the split view wants to show the detail
    // but the master is on screen in a spot that would be covered up by the detail
    // the return value of this delegate method is a Bool
    // "true" means "yes, Mr. UISplitViewController, I did collapse that for you"
    // "false" means "sorry, Mr. UISplitViewController, I couldn't collapse so you do it for me"
    // if our secondary (detail) is an ImageViewController with a nil imageURL
    // then we will return true even though we're not actually going to do anything
    // that's because when imageURL is nil, we do NOT want the detail to collapse on top of the master
   func splitViewController(
        _ splitViewController: UISplitViewController,
        collapseSecondary secondaryViewController: UIViewController,
        onto primaryViewController: UIViewController
    ) -> Bool {
        if primaryViewController.contents == self {
            if let ivc = secondaryViewController.contents as? ImageViewController, ivc.imageURL == nil {
                return true
            }
        }
        return false
    }
}

extension UIViewController
{
    // this var returns the "contents" of this UIViewController
    // which, if this UIViewController is a UINavigationController
    // means "the UIViewController contained in me (and visible)"
    // otherwise, it just means the UIViewController itself
    // could easily imagine extending this for UITabBarController too
    var contents: UIViewController {
        if let navcon = self as? UINavigationController {
            return navcon.visibleViewController ?? self
        } else {
            return self
        }
    }
}
```

```
//
//  DemoURL.swift
//
//  Created by CS193p Instructor.
//  Copyright (c) 2017 Stanford University. All rights reserved.
//

import Foundation

struct DemoURL
{
    static let stanford = URL(string: "http://stanford.edu/about/images/intro_about.jpg")

    static var NASA: Dictionary<String,URL> = {
         let NASAURLStrings = [
            "Cassini" : "http://www.jpl.nasa.gov/images/cassini/20090202/pia03883-full.jpg",
            "Earth" : "http://www.nasa.gov/sites/default/files/wave_earth_mosaic_3.jpg",
            "Saturn" : "http://www.nasa.gov/sites/default/files/saturn_collage.jpg"
        ]
        var urls = Dictionary<String,URL>()
        for (key, value) in NASAURLStrings {
            urls[key] = URL(string: value)
        }
        return urls
    }()
}
```