Assignment V: Fast SPoT

Objective

In this series of assignments, you will create an application that lets users tour Stanford through photos. The first assignment was to create a navigation-based application to let users browse different categories of spots at Stanford, then click on any they are interested in to see a photo of it.

The primary work to be done in this assignment is to improve the performance of the application by using GCD to move all Flickr fetching outside the main thread and by caching the image data for photos that comes back from Flickr.

All the data you need will be downloaded from Flickr.com using Flickr's API.

This assignment is due by the start of lecture next Tuesday. Remember that there is a holiday in between, so plan your work week accordingly.

Be sure to check out the **Hints** section below!

Materials

- This is a relatively minor modification to last week's assignment, so you'll probably be best off just extending what you did last week.
- You will need the `FlickrFetcher` helper code (see download instructions on iTunes U).
- You will need to obtain a Flickr API key. A free Flickr account is just fine (you won't be posting photos, just querying them).

Required Tasks

1. Your application must implement all the required tasks from the last assignment (and all the required tasks in this assignment) without doing any Flickr fetching or file system interactions in the main thread. Your user-interface should be responsive to the user at all times (i.e. the main thread should never be blocked).
2. The user-interface should always be giving some indication (e.g. a `UIActivityIndicatorView`) to the user when the currently displaying user-interface is going to show something when some active thread finishes. The network activity indicator in the status bar is not sufficient for this, but your application should *also* turn on the `networkActivityIndicator` whenever it is accessing the network (and only then).
3. Your primary table view must now have Refreshing enabled (i.e. you can pull down on it to activate the `UIRefreshControl`). Doing so should refetch the Stanford photo information from Flickr (of course, it's unlikely to have changed, but refetch anyway).
4. Cache photo image data viewed by the user into files in your application's sandbox. Each photo's image should be a separate file in the sandbox. Limit the cache's size to something testable. When this limit is reached, the least recently viewed photos in the cache should be evicted (deleted) to make room for new photos coming in (remember that you can look at a file URL's resource values to find out when it was last accessed). Your application should never query Flickr for the image data for a photo that it has in its cache (obviously). This cache should persist between application launches.
5. Keep as little of the photos' image data in the heap as you can (i.e. don't keep `strong` pointers to any `NSData` and/or `UIImage` objects that you are not currently displaying). You should have at most 2 photos' image data in memory at a given time, preferably only 1 (or 0 if none are in the process of being viewed or fetched).
6. Sort all tables alphabetically (and case insensitively) except the Recents table (which should still be sorted with the most recently viewed first). A good solution to this might involve some of the array sorting methods that use blocks.
7. Your application must work properly in portrait and landscape and on iPhone 4 & iPhone 5 and also must also work using appropriate idioms on the iPad.
8. You must get your application working on a physical device this week. Obviously we are not able to check this, so it is on you (and the honor code) to check yourself on this required task.

Hints

These hints are not required tasks. They are completely optional. Following them may make the assignment a little easier (no guarantees though!).

1. Be careful to pick the appropriate directory from `NSSearchPathDirectory` to use to store your cached photo image data.
2. Unless you did the iPad extra credit last week, you're probably using the photo size `FlickrFetcherPhotoFormatLarge` (that's 1024x768). On the iPad, you might want to move up to `FlickrFetcherPhotoFormatOriginal`.
3. A 3MB cache size is about 8 photos at `FlickrPhotoFormatLarge` (used on iPhone), but only about 2 photos at `FlickrPhotoFormatOriginal` (used on iPad), so might want to have the cache size vary depending on platform (maybe based on screen size?). Remember, the point is for you (and for us) to be able to test that your cache is working, so don't set the cache so big that you (or we) have to click on dozens of photos to start the eviction happening (nor so small that it's evicting on every photo).
4. You can try to use `NSCache` if you want, but it should not be necessary (it might be simpler to write your own cacheing class). In either case, you'll have to sync up the state of your cache object with what is in your sandbox each time your application launches.
5. If you want to clear out your image cache completely, you can do so by deleting your entire application from the device or simulator. You do this by pressing and holding on the application icon on the home screen of the device or simulator until it jiggles, then pressing the `X` that appears in the corner of the application's icon. Or put a secret (very uncommon) gesture somewhere in your UI that clears the cache.
6. `NSFileManager` can create directories, find out if files exist, delete files, etc., and `NSURL` can give you various information about a given file. Part of the intent of this assignment is for you to read through the documentation for these classes and figure out how to use them.
7. While a thread is off doing its work, things might be changing in the UI (i.e. the user might be touching on other things) so make sure that when the thread is ready to report its results back to the main thread that the user is still interested in what the thread was sent off to do. If not, you can just discard the results of the work the thread did.
8. The class `NSFileManager` is thread-safe (as long as you don't use the same instance in two different threads) and so is `NSData`'s `writeToFile:atomically:` and `initWithContentsOfURL:` so you can safely use these outside of the main queue.
9. With a few exceptions (`UIImage`, `UIFont`, `UIColor` and manipulating a drawing context directly), `UIKit` is NOT thread-safe. It relies on everything happening on the main queue. So do not access `UIKit` outside of the main thread. Doing so will have unpredictable results.

10. The `hidesWhenStopped` property on `UIActivityIndicatorView` is something you probably will want to set if you drag one in via Xcode.

11. You probably do not want an `UIActivityIndicatorView` to be a *subview* of a scroll view. If you just drag it out and drop it on top of a scroll view in Xcode, it will be. Use the Document Outline to drag it out from the scroll view to be a *sibling* of the scroll view. The Document Outline is a good place to wire it up as an outlet too.

12. It might be a good idea to introduce some simulated network latency to be sure that your multithreaded code is actually working. We'll probably do it when we grade your assignment, so beware! You can make the current thread sleep by using the call `[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:2]]` (which would sleep for 2 seconds). It's okay to pop this inside `FlickrFetcher.m` if you want to.

13. Even though we will start to learn about Core Data before this assignment is due, please do not try to use Core Data in this assignment (we'll do that next time).

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the **Required Tasks** section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc.
- UI is a mess. Things should be lined up and appropriately spaced to "look nice." Xcode gives you those dashed blue guidelines so there should be no excuse for things not being lined up, etc. Get in the habit of building aesthetically balanced UIs from the start of this course.
- Assignment was turned in late (you get 3 late days per quarter, so use them wisely).
- Incorrect or poor use of object-oriented design principles. For example, code should not be duplicated if it can be reused via inheritance or other object-oriented design methodologies.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

Extra Credit

Sorry, no extra credit available this week.