Homework #5: Top Places

Due: Tuesday, March 10 @ 12:30 PM

Submission:

Please turn in all files on Canvas before the deadline. You should compress your submission into a single file, do not submit a large number of individual files. If you know you are going to miss a deadline, contact the TA **before** the deadline. If you do not do this, skip days will automatically be deducted. Canvas has been known to be quirky, so it is not advised to wait until 5 minutes before it is due to make your submission.

Please include a text file called "README" at the top level of your main project directory. Include the following:

- Your name
- Your email address
- How long this project took you to complete
- Any comments or notes for the grader
- The type of iphone I should use to run your project

Overview:

This is not a group assignment. It is acceptable to consult with other class members, **but your code must be your own**.

In this assignment, you will create an application that presents a list of popular Flickr photo spots and then allow the user to see some photos taken in those spots.

The primary work to be done in this assignment is to create a tab-based user-interface with two tabs: Top Places and Recents. The first tab will allow the user to view which places on Earth have been the most popular for taking photos posted to Flickr and then look at some photos from those places. The second tab will let the user go back and see his or her most recently-viewed (inside your application) photos.

The goals are to get familiar with table views, scroll view, image view and multithreading and to learn how to build a Universal application that runs on both iPhone and iPad (with appropriate UIs on each).

All the data you need will be downloaded from Flickr.com using Flickr's API. Code will be provided which can build URLs for the Flickr queries you need for this assignment.

Be sure to review the Hints section below!

Materials:

• This is a completely new application, so you will not need anything (but the knowledge you gained) from your first four homework assignments.
• 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).
• This FlickrFetcher utility class is very useful for this assignment!

Required Tasks:

Download the data at the URL provided by the FlickrFetcher class method

1. URLforTopPlaces to get an array of the most popular Flickr photo spots in the last day or so. See the Hints for how to interpret data returned by Flickr.

2. Create a UITabBarController-based user-interface with two tabs. The first tab shows a UITableView listing the places obtained above divided into sections by country and then alphabetical within each section. The second tab shows a UITableView with a list of the 20 most recently viewed (in your application) photos (in chronological order with the most-recently-viewed first and no duplicates).

3. Anywhere a place appears in a table view in your application, the most detailed part of the location (e.g. the city name) should be the title of the table view's cell and the rest of the name of the location (e.g. state, province, etc.) should appear as the subtitle of the table view cell. The country will be in the section title.

4. When the user chooses a place in a table view, you must query Flickr again to get an array of 50 photos from that place and display them in a list. The URL provided by FlickrFetcher's URLforPhotosInPlace:maxResults: method will get this from Flickr.

5. Any list of photos should display the photo's title as the table view cell's title and its description as the table view cell's subtitle. If the photo has no title, use its description as the title. If it has no title or description, use "Unknown" as the title. Flickr photo dictionary keys

are #defined in FlickrFetcher.h.

6. When the user chooses a photo from any list, show it inside a scrolling view that allows the user to pan and zoom (a reasonable amount). You obtain the URL for a Flickr photo's image using FlickrFetcher's URLForPhoto:format:.

7. Make sure the photo's title is somewhere on screen whenever you are showing a photo image to the user.8. Whenever a photo's image appears on screen, it should automatically zoom to show as much of the photo as possible with no extra, unused space. Once the user zooms in or out on a photo by pinching, though, you can stop auto-zooming that image.

9. Your application's main thread should never be blocked (e.g. Flickr fetches must happen in a different thread). If your application is waiting for something over the network, it should give feedback to the user that that is happening.

10. Your application must work in both portrait and landscape orientations on both the iPhone and the iPad. Use appropriate platform-specific UI idioms (e.g. don't let your iPad version look like a gigantic iPhone screen).

11. The list of recent photos should be saved in NSUserDefaults (i.e. it should be persistent across launchings of your application). Conveniently, the arrays you get back from the FlickrFetcher URLs are all property lists (once converted from JSON).

Hints:

1. Put your own Flickr API key into FlickrAPIKey.h or your queries will not work.

2. It is possible to start off this assignment with the Tabbed Application template (or even the Master-Detail Application template) and you are welcome to play with doing so, however, it will probably (certainly) be less confusing to just start with the Single View Application template as usual and drag in the UITableViewControllers you need and use the Embed menu item as needed and then ctrl-drag to set up Relationships and Segues. It is an important part of this assignment to reinforce your understanding of how these storyboard-construction mechanisms all relate to each other.

3. The data returned from Flickr is in JSON format. iOS has a JSON parser built right in. Simply create an NSData containing the information returned from Flickr (using NSData's class method dataWithContentsOfURL:), then turn the JSON into a property list (i.e. NSArray and NSDictionary objects) using the class method in NSJSONSerialization called JSONObjectWithData:options:error:. You can pass 0 for the options argument.

4. The very first thing you're probably going to want to do once you have copied the FlickrFetcher code into your application (and set your API key) is to do fetch the URLforTopPlaces, parse the JSON and then NSLog() the results. That way you can see the format of the fetched Flickr results. Ditto when you query Flickr for the list of photos at a given place.

5. The top level of a query from Flickr is a dictionary. Inside that dictionary is the array of your results. So, for example, to get the array of places out of the data returned by URLForTopPlaces (let's assume you've converted the Flickr results from JSON into an NSDictionary called results), you could first use NSDictionary *placesResults = results[@"places"] and then NSArray *places = placesResults[@"place"]. Alternatively, you can do this with one method invocation: NSArray *places = [results valueForKeyPath:@"places.place"]!

6. There are #defines in FlickrFetcher.h for all of the interesting keys in the data returned from Flickr. Some of these have dots in them and so can only be accessed using valueForKeyPath:, for example, FLICKR_PHOTO_DESCRIPTION.

7. The key id (in a photo's dictionary of info) is a unique, persistent photo identifier (this is #defined to FLICKR_PHOTO_ID).

8. To create a table-view-based MVC, drag a Table View Controller out of the Object Library into your storyboard and change its class to be a custom subclass of UITableViewController (don't forget to set the superclass to UITableViewController in the dialog that New File ... brings up).

9. Some of you are still a little fuzzy on using object-oriented programming to encapsulate functionality in your program. For example, you are probably going to want 5 different UITableViewController subclasses in your application so that you can appropriately share code amongst them and have nicely-designed, reusable Controllers with clear APIs and Models. It is perfectly fine to create a subclass of UITableViewController to do something, then create a subclass of that class to do something slightly more refined.

10. In the same "good object-oriented design" vein, you will want to collect all of your NSUserDefaults calls into a single utility class somewhere rather than sprinkling the knowledge of the format of the recents data you store there around in multiple classes.

11. There are awesome sorting methods (ones that use blocks are particularly useful) in

NSArray and NSMutableArray. Be sure to check those out!

12. You will need more than an array of places as the internal data structure of your places table view controller, but it can easily be made up entirely of common Foundation classes (like NSDictionary and NSArray).

13. Each MVC should be prepared with the information it needs before it is pushed and then allowed to go do its thing on its own. In other words, an MVC should never depend on an MVC that segues to it (except for the preparation that MVC does to it in prepareForSegue:sender:).

14. Note that all the UITableViewCells in this assignment require subtitles, so you must set that as the type of the cell in Xcode for your dynamic prototypes.

15. Don't forget that the UITableViewCell reuse identifiers that you set in Xcode for dynamic prototype cells must match what is in your tableView:cellForRowAtIndexPath: methods. This can be a little confusing if you choose to have subclasses of subclasses of UITableViewController (since you are then inheriting tableView:cellForRowAtIndexPath:), so pick good reuse identifier names (that succinctly and generically describe what the cell is displaying).

16. The UIRefreshControl does not always seem to appear when you call beginRefreshing programmatically. If you are intrepid, you can work around this by making the UITableView scroll up by setting it's contentOffset (remember that a UITableView is a UIScrollView) to have a negative y value (setting it equal to the height of the refreshControl would probably be best). However, doing this workaround is not a Required Task (starting the refresh control going is, though, even if it does not scroll to appear).

17. Turning an image URL from Flickr into a UIImage is easy. Just create an NSData with the contents of that URL ([NSData dataWithContentsOfURL:theURL]), then create a UIImage using that NSData ([UIImage imageWithData:imageData]).

18. The scroll view zooming requires some calculations involving the UIScrollView's bounds and the size of the photo. Thus you will have to recalculate this every time the UIScrollView's bounds changes or the frame of the UIImageView inside it changes. Don't forget from lecture where (in the View Controller Lifecycle) geometry calculations for a view have to occur.

19. You can get a quick-and-dirty title bar for the detail view controller of a split view controller simply by embedding the detail view controller inside a UINavigationController.

However, when you go to "find" the detail view controller, you will have to know how to look inside a UINavigationController (to find its rootViewController) to get at it.

20. By the time viewDidLoad is called, the UISplitViewController's delegate methods have already been called (especially the one that gives you a UIBarButtonItem to put in the detail view controller's UI somewhere that brings up the master). So you will need to set your UISplitViewController's delegate before that. awakeFromNib is a perfect place to do it.

21. If you want to update the detail view controller in a split view on the iPad from a master view controller which is a table view controller, you'll probably want to implement the tableView:didSelectRowAtIndexPath: method in the master (it's sort of the "target/action" method of a table view) rather than segueing. You'll want to do the same things in that method that you do in prepareForSegue:sender: (i.e. set the Model (and any "how to display this" properties) of the destination view controller).

22. Because your image view controller is probably on-screen at all times on the iPad, it needs to be able to respond properly to having the URL for the image it displays changing over time (to images which are different sizes, for example).

23. If you are resetting the image of your image-displaying MVC (e.g. it's the detail view controller in a split view), be careful to reset your UIScrollView's zoomScale back to 1 before you reset the scroll view's contentSize for a new image. The zoomScale affects the contentSize (e.g., when you zoom in the contentSize is automatically adjusted to be larger and when you zoom out, it gets smaller), so if you have a zoomScale other than 1 and you start mucking

with the contentSize, you'll get results you're probably not anticipating.

24. The method mutableCopy in NSArray might come in handy when you want to add something to a data structure already stored (immutably) in NSUserDefaults.

25. As always, the amount of code required to implement this application is not huge (it can be done in under 150 lines of code, if you only count the ones added between curly braces). If you find yourself needing dozens of lines of code for any one feature, there's probably a better way to go about it. In general, "brawn over brains" solutions (i.e. "just keep typing in code until it works") are bug-prone and a pain to maintain, so avoid them like the plague! We use object-oriented programming for a reason. Use its mechanisms to the fullest.

26. Just use the text that comes after the last comma in the name of a place as the place's country.