Overview

- Part 1: Threading Flickr TableView

Submission

Lab evaluation form
Lab feedback form

Note: If you are unable to complete the lab during this week's lab period, please be ready to demonstrate it at the beginning of the next lab.

Part 1: Threading Flickr TableView

In this lab, you will take your Flickr TableView application and make the loading of images occur on a separate thread (so that scrolling the TableView is very responsive).

Step 1) Read the following about NSOperationQueue:

NSOperationQueue is one way to perform many operations on a separate thread without the overhead of creating and destroying new threads for each operation. To use NSOperationQueue:

```
// Allocate the queue (in viewDidLoad)

NSOperationQueue *queue = [[NSOperationQueue alloc] init];

[queue setMaxConcurrentOperationCount:1];

// Create an operation
```

Implement a class that extends NSOperation (uses NSOperation as a subclass). Create a **– (void) main** method (it will be called when on a separate thread when you add your operation to the queue). Also, create a custom initialization function that saves a url, target, and action. At the end of the main method, you can run: [target performSelectorOnMainThread…]; to pass back your image data and URL.

```
// Add an operation to the queue (when an image is not in the cache)

MyCustomOperation *operation = [[MyCustomOperation alloc] initWithURL:url target:self action:@selector(finished)];
[queue addOperation:operation];
```

WARNING! – If any object of your main method has to be **autoreleased**, you must wrap the main method with @autoreleasepool (since it is running on a separate thread).

Step 2) Reuse your project from lab7 (FlickrTableView)
Step 3) Implement your own CustomOperation class that extends NSOperation.
Step 4) Create a cache (a NSMutableArray, NSDictionary, etc) for the images. Whenever you scroll on the table, if an image is not in the cache, create a new operation and add it to the queue. In the example above, the custom operation class has a custom init function that saves an NSURL *url, an id target, and a SEL action. This is so the **finished** method of the view controller can be run after the operation is complete; this finished method could be a place where you call the **reloadData** method of the UITableView instance or where you call **reloadRowsAtIndexPath:withAnimation:**.
Step 5) If you are looking for a challenge, make it so your cache only stores the images currently being displayed and frees the memory for those that are not visible.
Step 6) Demo your application to the TA.

Tip: Make sure your initializer for the CustomOperation class follows the default initializer format:

```
- (id) init {
  self = [super init];
  return self;
}
```