Objectives

- Use the Leaks Performance Tool to find a memory leak in a provided application*
- Use the Time Profiler to increase performance in a provided application*
- Create a battery monitor application*

* Show TA these steps when completed

Submission

Answer the questions in this manual on the lab evaluation form.
Complete an online lab feedback form.
If you are unable to complete a battery monitor application before the end of lab, please have it ready to show by the beginning of the next lab.

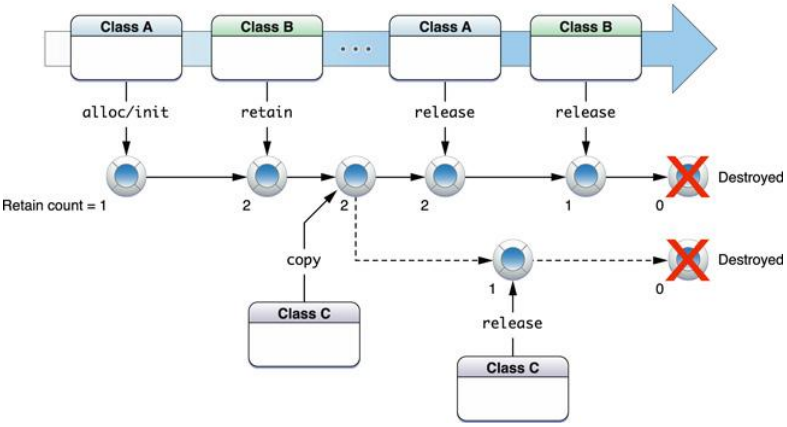
Prelab

The broadcast project on the course website has functionality you may wish to use in future applications. Review the MainViewController.m code file of the broadcast project from the course website. See if you can find the statements in code that create the following functionality, and then write the statement on your evaluation form:

1. Using NSTimer to call a function every x seconds
2. Using NSLog to write text to the console
3. Converting an int to an NSString

References

Memory Management:
http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/MemoryMgmt/Articles/mmRules.html#//apple_ref/doc/uid/20000994-BAJHFBGH
https://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.html



UIDevice Class:
http://developer.apple.com/iphone/library/documentation/UIKit/Reference/UIDevice_Class/Reference/UIDevice.html

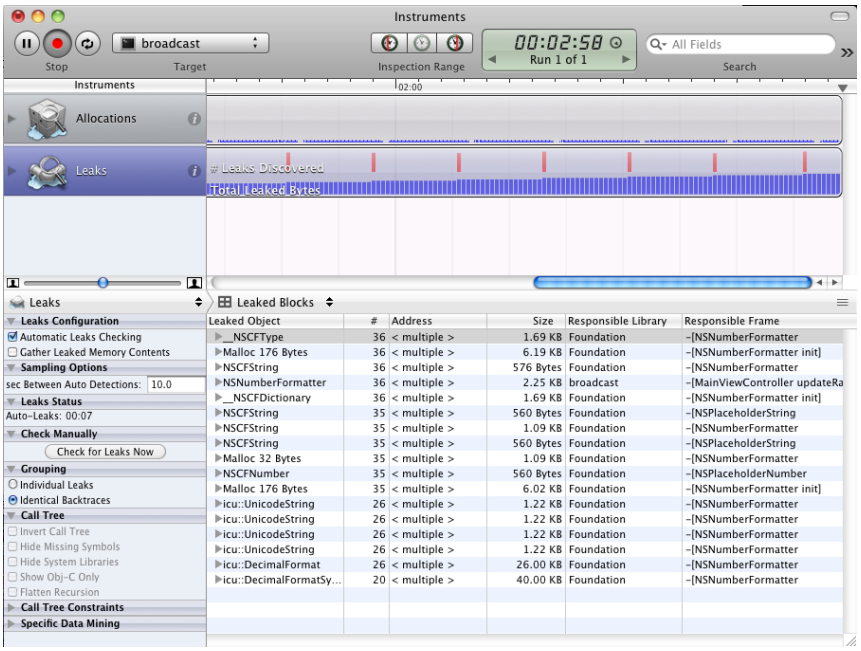
NSNotificationCenter Class:
http://developer.apple.com/iphone/library/documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter_Class/Reference/Reference.html

Memory Management

Step 1) Read Apple's documentation on memory management and answer the following questions on the lab evaluation sheet.

1. When do you need to use **release** and **autorelease**?
2. What is reference counting? Describe how **release**, **autorelease**, and **retain** affect the retain count of an object. What occurs when the retain count reaches 0?
3. What is the proper usage of the **NSAutoreleasePool** class? You may answer with example code.

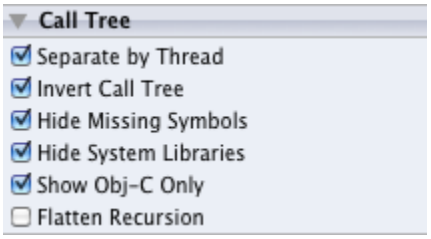
Step 2) Download the "broadcast" project from the course website.
Step 3) Launch the project in Xcode (open the broadcast.xcodeproj file).
Step 4) Ensure that the project is set to run in the simulator by selecting Simulator in the drop down box on the top-left corner of Xcode's main window.
Step 5) Build your project (⌘ + B). Running a performance tool will use the last build of your project.
Step 6) From the Product file menu, select *Profile*. Then select the Leaks tool. The application will run in the simulator and memory information will be displayed in the *Instruments* window. The performance tool will check for a leak every 10 seconds. Take note of the interface:



Step 6) When a leak is discovered, you can find out more information by clicking on the Leaks bar chart, which will list the leaks. Furthermore, you can select *View->Extended Details* (⌘ + E) and select one of the memory leaks to view a stack trace, which will allow you to see the method from which the memory leak originated.
Step 7) In Xcode 4, a new feature was created to analyze your code for possible memory leaks. From the Product file menu, select *Analyze*.
Step 8) Find and fix the memory leak(s) in the project's code. Rerun the Leaks Performance test and show the TA when you have successfully fixed the memory leak(s).

CPU Management

Step 1) Keep open the fixed broadcast project.



Step 2) Profile your project with the *Time Profiler*. The application will run in the simulator and CPU utilization information will be displayed in the *Instruments* window. Since we mostly care about the performance of our application, you can check the box next to *Hide System Libraries* to show the CPU utilization of the project's methods.

Step 3) When we do our programming, the event loop (a loop that handles responding to messages that the OS sends) is hidden from view. This event loop uses the CPU to handle messages, and thus consumes most of the CPU when the program is idle. Check the box next to *Show Obj-C Only* to hide the event loop in the samples list.

Step 4) You may wish to turn on the *# Samples* column. Notice the broadcast method increases when the program is sending packets. Toggle the packet sending off. Now the broadcast method's running count stops increasing.

Step 5) One of the Objective-C methods seems to be consuming CPU resources even when there is no user action to respond to and no packets being broadcast. Alter the program so that this method still does its job, but without using resources when the user commits no actions.

Note: Currently, an NSTimer is constantly polling the function. This constant polling is wasting CPU resources. Think about pushing the data to the method like you did with the Touch Down event of the UIButton in Lab 1.

Step 6) Build, then rerun the *CPU Sampler* test and show the TA when you have successfully fixed the code.

Battery Monitor Application

In this section of the lab you will create your own application. The requirements are listed in the bullets below, but you may add more features to your application as you see fit. You will need to extensively rely on Apple's documentation to implement the requirements of this application.

Step 1) Open the documentation on the UIDevice class and NSNotificationCenter class.
Step 2) Read the summary of UIDevice class and NSNotificationCenter. Determine which methods and properties of UIDevice you will most likely need to get the battery state and level.

Hint: Both classes use the Singleton paradigm; that is, there is one instance of UIDevice and once instance of NSNotificationCenter. To get the singleton instance (and thus be able to call instance methods), use:

[UIDevice currentDevice];
[NSNotificationCenter defaultCenter];

Step 3) Create a Battery Monitor application that has the following features:

- Displays whether the battery is charging, full, or unplugged.
- Displays the level of the battery as a percentage (0% to 100%).
- Updates the information on the display when it changes.
- Includes your name.

Notes:
You will need to use a device for accurate information to be displayed. The simulator will not show accurate battery state information.

- You should be able to do all of your coding in your ViewController. The viewDidLoad method gets called after the viewController's view is loaded from the .xib; you may put any setup instructions in this method.
- There are two ways to update the display. The best way is to push the data to a method by registering a method with NSNotificationCenter to be called whenever the singleton instance of UIDevice sends out a notification relating to the battery state or level. Another way is to constantly poll the singleton instance of UIDevice for the information using a timer.