lab 6

Objective C (optional)

cs198-001 : fall 2017

Announcements

- **Guest lecture this Monday (11/20)**
  - Past instructors Shawn (now working at Apple) and Gene (now working at Yelp)
  - Topic: Advanced Swift and cross language compatibility
  - Recommend getting familiar with Objective-C before lecture

Today

- Quick Objective-C Overview
- TA check-off
  - if your TA is not here, check-off with Chris, Paige, or Nithi

History and Philosophy

VS

Why Bother?

- Most Companies still have thousands of lines of legacy Objective-C code
- Cocoa and Cocoa Touch Frameworks still in Objective-C (including ability to mix in C/C++)
- Objective-C is still just…better

Object-Oriented + C = Objective C

- Designed by Brad Cox and Tom Love at Stepstone
- Combined Smalltalk philosophy (OO, message passing, etc.) with C backwards compatibility
- Licensed by NeXT, later adopted by Apple

Pointers and Mem. Management

- YES, you must alloc memory for objects!
  - How much? Done for you
- YES, Objective-C has pointers
- no, you don't have to free allocated memory
  - Automatic Reference Counting (ARC)

Why not Garbage Collection?

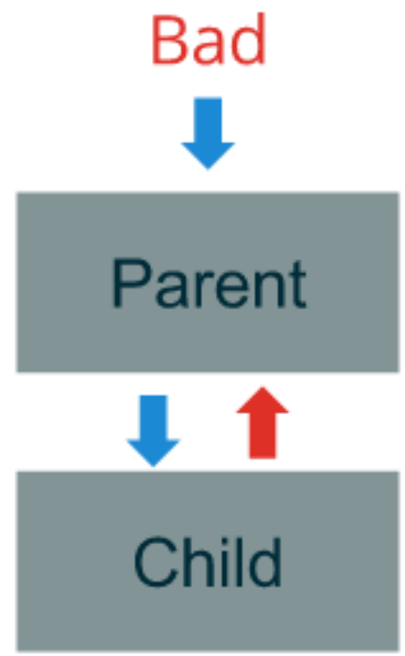
- What's wrong with Garbage Collection (GC)?
  - Stores object-dependency graph at run-time
  - Objects become candidates to free later (unpredictable, slow, inefficient)
- Improve?
  - Make the compiler analyze code to determine when objects need to be released (freed) or retained (not freed) at compile-time

GC vs ARC

- **Garbage Collection (GC)?**
  - Stores object-dependency graph at run-time
  - Objects become candidates to free later (unpredictable, slow, inefficient)
- **Automatic Reference Counting (ARC)**
  - Auto-generate release/retain code @ compile-time
  - Objects are freed IMMEDIATELY when not needed
  - Vulnerable to retain cycles

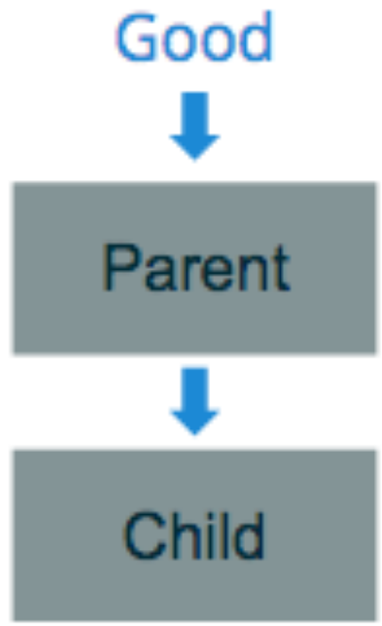
Retain Cycles

```
@class Child;
@interface Parent : NSObject {
    //instance variables implicitly __strong
    Child *child;
}
@end
@interface Child : NSObject {
    Parent *parent;
}
@end
```



Retain Cycles

```
@class Child;
@interface Parent : NSObject {
    //instance variables implicitly __strong
    Child *child;
}
@end
@interface Child : NSObject {
    //doesn't increase ref count
    __weak Parent *parent;
}
@end
```



Syntax

Head the Implementation!

- Return of the header (.h) and main (.m) file
- **Header** file used to define class
- **Main** file is the implementation
- No more .swift

Header File - Example

```
#import <UIKit/UIKit.h>
@interface ViewController: UIViewController


@property (nonatomic, retain) UILabel *label;
-(NSString *)showString;


@end
```

Header File - General

```
#import <UIKit/UIKit.h>
@interface <ClassName>: <SuperClassName>

//all properties and methods go here

@end
```

The Implementation File - Example

```
#import “ViewController.h”
@interface ViewController()
@end
@implementation ViewController
-(void)viewDidLoad {
    [super viewDidLoad];
    self.label = [[UILabel alloc] init];
}
-(NSString *)showString {
    return @”Obj-C >= Swift”;
}
@end
```

The Implementation File - General

```
#import “ClassName.h”
@interface <ClassName>()
@end
```

```
@implementation <ClassName>
//implement methods here
@end
```

Properties

- @property - tag for creating instance variables
- **Automatically** generates getter/setter methods
  - i.e. [self setPropertyName: Steve]
- Several **Attributes**

Attributes

- atomic - less prone to multithreading (concurrency) errors
- nonatomic - prone to multithreading errors
- strong - want to own the object
- weak - you don’t want control over object’s lifetime
- readonly - doesn’t create setter method
- copy - used for primitive data-types
- retain - pre-iOS 5 version of strong
- assign - pre-iOS 5 version of weak

Bracket and Dot Notation

- Two ways to call functions and access properties
- bracket notation is more common
  - (stay consistent / follow style guidelines)

```
self.aTextField = [[UITextField alloc] init];

// dot notation
self.aTextField.text.length;

// bracket notation
[[[self aTextField] text] length];
```

Bracket Notation - Instantiating

```
UILabel *label = [[UILabel alloc] init];
```

Getters and Setters

- Compiler automatically synthesizes getters and setters

```
@property (nonatomic, strong) UILabel *ageLabel;
….
//automatically generated
[self ageLabel];
//automatically generated
[self setAgeLabel:someLabel];
```

Dot Notation

- More similar to Swift
  - Less confusion when using both Swift / Obj-C
- Some Obj-C programmers still prefer getters/ setters
  - Easier to distinguish between getter and setter
  - Able to use CMD + f
- Use when dealing with properties
  - Keeps it clean and concise

Dot vs. Bracket (Setters)

```
@property (nonatomic, strong) UILabel *ageLabel;
....
//dot notation
self.ageLabel = someLabel;

//bracket with setter
[self setAgeLabel: someLabel];
```

Functions

- **Plus (+)**
  - class method
- **Minus (-)**
  - instance method

Creating Functions

```
-(NSString *)showString {
    return @"Obj-C >= Swift";
}
```

```
-(NSString *)showString:(NSString *)extra {
    return extra + @" says hello!";
}
```

```
-(NSString *)showString:(NSString *)extra _:(NSString *)var {
    return [NSString stringWithFormat:@"%@ %@", extra, var];
}
```

Calling Functions

```
[self showString];
```

```
[self showString:@”Obj-C”];
```

```
[self showString:@”Obj-C”
                withString:@”Functions”];
```

```
//general method of calling
[<Class> <Function in class>];
```

Strings

- There is no built-in class type String, must use NSString
- To use NSString, must prepend every string with @
- Printing to log is done with NSLog

Strings - Examples

```
//creating a string
NSString* str = @"Hello world!";
```

```
//combining strings
NSString* str = [NSString
    stringWithFormat:@"Hello %@", @"World!"];
```

```
//printing to log
NSLog(@"Printing %@”, @”to log!");
```

Arrays

- There is no built-in class type Array
- Use NSArray or NSMutableArray

```
//immutable
NSArray *staff = @[@"Sony", @"Gene", @"Shawn",
@"Allie", @"Lucy", @"Helena", @"Kyle"];
```

```
NSArray *staff1 = [NSArray
arrayWithObjects:@"Sony", @"Gene", @"Shawn",
@"Allie", @"Lucy", @"Helena", @"Kyle", nil];
```

```
//mutable
NSMutableArray *hello = [[NSMutableArray
alloc] init];
[hello addObject:@"Hello"];
```

UIView

```
//initWithFrame and CGRectMake
UIView *testView = [[UIView alloc] initWithFrame:
                    CGRectMake(0, 0, 200, 200)];
```

```
//addSubview is crucial!
[self.view addSubview:testView];
```

UIView - Another example

```
//getting the size of our screen
CGSize *screen = [UIScreen mainScreen].bounds.size;
```

```
//let's apply this now (look at CGRectMake)
UIView *testView = [[UIView alloc] initWithFrame:
                    CGRectMake(screen.width/2-100,
                               screen.height/2-100, 200, 200)];
//addSubview is crucial!
[self.view addSubview:testView];
```

Custom Initializer Example

```
#import “CustomLabel.h”
@implementation CustomLabel
-    (id) custInit {
     self = [super initWithFrame:CGRect];
     //Design your UILabel here
     return self;
}
@end

CustomLabel *l = [[CustomLabel alloc] custInit];
```

Additional Notes

Objective-C + Swift

- Use a **bridging header**
- Add Obj-C Class (.m file)
- Add Bridging Header
  - Click ‘Yes’ when asked to configure bridging header **(gone over in detail next lecture)**
- Add Obj-C class header (.h file)
- Fill out Obj-C code (both .m and .h)
- Import class in Bridging Header

Using Swift with Objective-C

- Add Swift class (.swift file)
- Import Swift file as:
  - #import “NameOfFile-Swift.h”

today's lab

Optional Lab: Obj-C Calculator
github.com/iosdecal/ios-decal-objc-lab

Progress Check-off with TA
Required for attendance
Free to leave once you have met with TA