iOS: Objective-C Primer

Jp LaFond
Jp.LaFond+e76@gmail.com
TF, CS76

Announcements

- n-Puzzle feedback this week (if not already returned)
- iOS Setup project released
- Android Student Choice project due

Tonight

- XCode and GDB
- Objective-C Primitive Data Types
- Objective-C Classes and Objects
- Objective-C Foundation Collections
- Objective-C Designing a Class

XCode

- Download from the Mac App Store
  - http://itunes.apple.com/us/app/xcode/id497799835?mt=12

XCode

- View project information in navigator view
    - project: files
    - symbol: classes and methods
    - search: search classes, methods, and implementations
    - issues: compilation errors and warnings
    - debug: debug information
    - breakpoint: view/remove breakpoints
    - log: build/run list

Help and Simulators

- Installing documentation:
  XCode → Preferences → Downloads → Documentation → Check and Install Now
- Viewing all documentation:
  Organizer (Windows → Documentation)
- View documentation by class/method:
  option/click

Debugging

- GDB built into XCode
    - print object: `po <object>`
    - create breakpoint by clicking on line number
        - or in the console: `b function` or `b line`
    - list breakpoints with: `info b`
    - delete nth breakpoint with: `delete <n>`

Debugging

- At breakpoint, go to the next line
  - `next` (execute any called function)
  - `step` (go into any called function)
- At breakpoint, continue to the next breakpoint:
  `continue`

Objective-C: The Language

- Strict superset of C
    - Any C program is also an Objective-C program
- Major implementations: Clang (with LLVM) and GCC
    - These are changed in the Build Settings

Primitive Data Types

- `int`: integers like `-17, 26, 341`
- `float:` floating point decimals like `1.0f, 3.14f, -7f`
- `double:` larger-capacity floats
- `char:` single character like `‘1’, ‘J’, ‘p’`
- `id:` object of any type
    - `nil`: any empty id

Strings

- not a primitive type (just like Android's String)
- implemented by `NSString`
- strings are defined as `@“a string constant”`

Logging

- `NSLog` is the equivalent of Android's `Log.d` or `console.log`
- Special characters in the `NSLog` string can be replaced with values:
  - `int: %d`
  - `float: %f`
  - `char: %c`
  - `NSObject: %@`

Interface

- declares class instance variables and methods
- `.h` file

```
@interface <class> : <parent>
- (<type>) <method name>;
@end
```

Implementation

- defines class methods
- `.m` file

// optional private interface extension

```
@implementation <class>
- (<type>) <method name> {
    // implementation goes here
}
@end
```

Properties

- getters/setters to access class member variables
- getter
```
- (int) variableName { return variableName; }
```
- setter
```
- (void) setVariableName:(int)NewVariable {
    variableName = newVariable;
}
```

Properties

- Starting with Objective-C 2.0, getters/setters can be generated for you:
  - interface:
    `@property (attributes) <property name>`
  - implementation:
    `@synthesize <property name> [= <ivar name>]`
  - `self.variableName = 7;`
  - `[self setVariableName:7];`

Property Attributes

- `nonatomic`: unsynchronized, but faster access
- `readonly`: only getter generated
- `readwrite`: both getter/setter generated (default)
- `assign`: nothing extra, just assignment
- `retain`: `retain` sent to the new value
- `copy`: new object is allocated and the value copied

Method Arguments

- no arguments:
```
- (void) method
```
- single argument:
```
- (void) method:(int)argument
```
- multiple arguments:
```
- (void) method:(int)argument
otherArgument:(int)other
```

Calling Methods

- All calls are message passing:
  - Message sent to object, and object responds to the message
- Message receiver resolved at runtime:
  - No type checking at compile time.
  - Object may not respond to the message sent.
- `[class method:argument other:value];`

Instantiating Classes

- `alloc`: reserve memory for object (like `malloc` in C)
- `init`: set up the created object (like a constructor in Java)
  - initialize attributes via custom `initWith<Something>`: methods
- both return pointers to objects
- convenience method: `new`

Memory Management

- In iOS 5.0, Apple introduced ARC (Automatic Reference Counting), their answer to garbage collecting.
  - Gone of the days of having to use: `retain`, `release`, `autorelease`, or to deal with writing a specific `dealloc` method, where you `release` things.

Using Other Classes

- interfaces and implementations need to know about other classes
- interface `@class <class>`
  - forward class declaration: tell compiler `<class>` exists
- implementation: `#import “<class>.h”`
  - like `#include`, this uses interface to tell compiler what `<class>` looks like

NSString

- `initWithString`: creates a new `NSString` object from `@“string constant”`
- `length:` number of characters in the string
- `substringFromIndex`, `substringToIndex`: get a substring from a `NSString`.
- `isEqualToString`: string comparison
- `stringByReplacingOccurrencesOfString`: new string from replacing substring with another string.

NSMutableArray

- `initWithObjects`: create an `NSMutableArray` with a comma-separated list of objects
- `count`: number of elements in the array
- `containsObject`: whether or not an object is in the array
- `indexOfObject`: index of given object in array

NSMutableArray

- `objectAtIndex`: object at given index in array
- `addObject`, `removeObject`: add/remove an object from the array

NSMutableDictionary

- `initWithObjects`: create an `NSMutableDictionary` from a list of keys and values
- `count`: number of elements in the dictionary
- `objectForKey`: get value associated with key
- `allKeys`, `allValues`: get an `NSArray` of all keys/values.

NSMutableDictionary

- `setObject`, `removeObjectForKey`: add/ remove an object from the dictionary

Class Design

- Sample code!