iOS: Objective-C

Tommy MacWilliam

Harvard University

March 22, 2011

Announcements

- Lectures: http://cs76.net/Lectures
- Sections: http://cs76.net/Sections
- n-Puzzle feedback this week

Today

- XCode + GDB
- Data types
- Classes and objects
- Foundation collections
- Designing a Class

XCode

- download from the Mac App Store, $4.99
  - :(

XCode

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

Getting Help

- installing documentation: XCode → Preferences → Documentation → Check and install now
- view all documentation: Organizer (in the top right) → Documentation
- view documentation for class/method: option-click

Debugging

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

Debugging

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

The Language

- strict superset of C
  - any C program is also an Objective-C program
- major implementations: Clang (with LLVM) and GCC
  - not just for OS X! see: GNUstep

Primitives

- `int`: integers like `1, -2, 123`
- `float`: floating point decimals like `1.0f, 3.14f, -5.f`
- `double`: larger-capacity floats
- `char`: single character like `'a','Z', '8'`
- `id`: object of any type

Strings

- not a primitive type (just like in Java)
- implemented by `NSString`
- strings defined via `@"the string"`

Formatting

- `NSLog` is the new `Log.i` or `console.log`
- special characters in `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> <ivar name>;
}
- (<type>) <method name>;
@end
```

Implementation

- defines class methods
- `.m` file

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

Properties

- getters and setters are necessary to access class member variables
- getter

```
- (int) bar { return bar; }
```

- setter

```
- (void) setBar:(int)newBar {
  bar = newBar;
}
```

Properties

- starting with Objective-C 2.0, getters/setters can be generated for you
- interface: `@property (attributes) <property name>`
- implementation: `@synthesize <property name>`
- `foo.bar = 4;`

Property Attributes

- `nonatomic`: unsynchronized, but faster access
- `readonly`: only getter generated
- `readwrite`: both getter/setted generated (default)

Method Arguments

- no arguments:
  - `- (void) foo`
- single argument:
  - `- (void) foo:(int)bar`
- multiple arguments:
  - `- (void) foo:(int)bar baz:(int)qux`

Calling Methods

- message-passing used to “call” methods
  - message sent to object, and object responds to message
- message receiver resolved at runtime
  - no type-checking at compile time
  - object may not respond to message!
- `[object method:argument another: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

Memory Management

- no automatic garbage collection :(
- reference counting
  - `alloc` or `retain`: `count++`
  - `release`: `count--`
- `dealloc` called when the reference count is 0
  - `release` object's fields

More Property Attributes

- `assign`: nothing extra, just assignment
- `retain`: `retain` sent to new value, `release` to previous value
- `copy`: new object is allocated via `copy` messaged (old value `released`)

Using Other Classes

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

NSString

- `initWithString`: create a new `NSString` object from `@"string"`
- `length`: number of characters in the string
- `subStringFromIndex, substringToIndex`: get a substring from a string
- `isEqualToString`: compare strings
- `stringByReplacingOccurrencesOfString`: replace substring with another string
  - told you Apple liked long method names

NSMutableArray

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

Designing a Class

- example time!