CS164 Section 2: OOP / Objective-C

Chris Gerber

Harvard University

February 29, 2012

Agenda

- Xcode + LLDB
- Data types
- Classes and Objects
- Foundation collections
- Designing a Class

Xcode

- We are using version 4.3 on Lion
- Downloadable from the Mac App Store
- (Read the FAQ if on Snow Leopard)

Xcode

- Navigator View
  - Project: files and groups
  - Symbol: classes and methods
  - Search: search classes, methods, implementations
  - Issue: compilation errors and warnings
  - Debug: debug information
  - Breakpoint: view/remove breakpoints
  - Log: build/run list

Getting Help

- Install the documentation

  Xcode -> Preferences -> Documentation
  -> Check and install now

- View the documentation

  Organizer -> Documentation

- Documentation for class/method

  Option-click

Debugging

- LLDB built into Xcode (embedded console)
  - Print value: p <variable>
  - Print object: po <object>
  - Breakpoint: b <line>
  - List breakpoints: br l
  - Delete breakpoint: br del <id>
  - Next instruction: n <count>
  - Step into: s <count>
  - Continue: c

The Language

- Strict superset of C
  - Any C program is also an Objective-C program
- Major implementations:
  - Clang (with LLVM)
  - GCC
- Not just for OS X (see GNUstep)

Primitives

- `int`: integers like 1, -2, 123
- `float`: floating point decimals like 1.0f, 3.14f, -5.0f
- `double`: larger-capacity floats
- `char`: single character like ‘a’, ‘Z’, ‘8’
- `BOOL`: YES or NO
- `id`: object of any type

  `nil`: a null object

Strings

- Not a primitive type (like in Java)
- Implemented by `NSString`
- Strings defined via `@“the string”`

Formatting

- `NSLog` is the parallel to `Log.i` and `console.log`
- Uses replacement patterns similar to `printf`

```
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

- 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
- `readonly`: only generate a getter
- `readwrite`: generate both a getter and setter (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

More Property Attributes

- assign: straight assignment of value
- copy: new object allocated via copy message (old object released)
- strong: a reference that retains the object
- weak: a reference that does not retain the object

Using Other Classes

- Interfaces and implementations need to know about the other classes
- Interface: `@class <class>`
  - Forward class declaration: tells compiler that `<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
- `stringByReplacingOccurancesOfString`: replace substring with another string

NSMutableArray

- `initWithObjects`: create a `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

NSMutable Dictionary

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

Designing a Class

- Demo