iPhone Programming
CMSC 498i – Fall 2010

Introduction

Adam Porter
aporter@cs.umd.edu
Introduction

Course Details
Course Goals

• Core CS concepts
  - e.g. design patterns, OO programming

• Practical experience
  - e.g. focused labs, projects

• Mobile platform knowledge
  - design / engineering implications

• iPhone OS Programming
Course Layout

• Lectures cover smartphone concepts & iPhone APIs

• Significant reading assignments
  - No required book
  - Links to assigned reading will be provided

• Labs & Projects
  - Weekly labs & semester team project

• Exams
  - Midterm (Friday, Oct. 15, 2:00-2:50)
  - Final (Thursday, Dec 16, 1:30-3:30)
Semester Project

• Small teams of 2 or 3 students
  - Teams randomly assigned around week 3

• Choosing a project
  - Will start with a set of suggested project topics
  - Teams will extend suggestions into a project proposal
  - Industrial Partners Board will provide feedback
    • Professor makes go/no-go decision

• Projects presentations at end of semester
Equipment

• Owning a Mac computer is not required
  - Will have access to Mac laptops during lab sessions
  - On-campus computing labs can be used at other times

• Owning an iPhone or iPod Touch is not required
  - Assignments can be done with iPhone Simulator
  - OIT will provide some iPod Touch devices for team projects

• Provisioning profile
  - University development profile will be provided to all students
  - We’ll set this up during the first lab
Administrivia

• Lab attendance is vital
  - Assignments posted to class website before lab
  - Should be possible to finish in 1-2 hours

• Submissions
  - Will use SubmitServer system for lab & project submissions
    • http://submit.cs.umd.edu
  - Due at 11:59:59 EST of the due date (unless otherwise specified)
  - Late submissions accepted until 9am next morning
    • (score multiplied by 0.8)
Getting Help / Info

• Course website - http://www.cs.umd.edu/class/fall2010/CMSC498I
  - Lecture and Lab material
  - Resources, Syllabus, Contacts, etc…

• 498I forum
  - Questions & Answers
  - Supplemental info
  - Internship/job opportunities

• Office Hours
  T 10:00-11:00am, W 1:00-2:00pm in AV Williams 4125
This Week’s Topics

• iPhone SDK Overview

• Development tools
  - Xcode (IDE), Interface Builder (UI builder)

• Development language
  - Basic Objective-C

• Lecture will be posted online later today!
Introduction

iPhone SDK
iPhone SDK
iPhone SDK

Tools

Xcode

Interface Builder
iPhone SDK

Tools

Xcode

Interface Builder

Frameworks

(Cocoa Touch)

Foundation

UIKit
iPhone SDK

Tools
- Xcode
- Interface Builder

Frameworks (Cocoa Touch)
- Foundation
- UIKit

Language (& Runtime)
- Objective-C
- `textView` setStringValue: @“Hello”};
iPhone SDK

• Cocoa Touch
  - Based on Cocoa APIs from Mac OS X
  - Many, but not all components are identical, so be careful when reading online documentation
  - Uses many well-known design patterns

• Objective-C
  - Adds OOP concepts to C
  - Minimal syntax additions
  - Expect to spend some time learning this new language
iPhone OS Components

- Cocoa Touch Layer
- Media Layer
- Core Services
- Core OS
Core OS

- OS X Kernel
- File System
- Sockets
- Power Management
- Security
- Keychain
- Certificates
- Bonjour
Core Services

• Collections
• Address Book
• Networking
• URL Utilities
• File Access
• SQLite

• Core Location
• Threading
• Preferences
• Block Objects
• Grand Central Dispatch
Media Layer

- AV Foundation
- Core Audio
- Open Audio Lib (OpenAL)
- Core Graphics

- Core Animation
- Core Text
- Media Player
- OpenGL ES
Cocoa Touch Layer

- Multi-Touch
- Accelerometer
- View Hierarchy
- Alerts
- Web View
- Camera
- Address Book UI
- Game Kit
Introduction

Development Tools
Xcode

- **IDE** - Integrated Development Environment
- Code editor
- Compiler
- Run code in simulator or on device
- GDB Debugger - UI / command line available
- Tools
  - Static analysis
  - Performance: Shark, Instruments
- Source control integration
Xcode – main window
Xcode – main window

Files & Resources
Xcode – main window

Find, SCM, Build
Xcode – main window

Build Settings
Xcode – organizer

[Image of Xcode organizer window]
Xcode – SCM
Simulator

• Benefits
  - No need for physical devices
  - Fast turnaround
  - Work with faster processor

• Caveats
  - Doesn’t simulate all H/W features
    • No accelerometer or camera
  - Performance may be misleading
  - User experience on devices will be different
Interface Builder (IB)

- Use IB to layout UI and connect it to your code
- IB is an object editor (.xib files)
  - Creates archives (vs. code generation)
  - System unarchives .xib files at run time
- Strengths
  - Helps enforce UI consistency
  - Localizable
  - Often easier than typing
- Anything done in IB can be done in code
- Some things done in code, may be hard to do in IB
Interface Builder

Displays an element that shows the user the boolean state of a given value. By tapping the control, the state can be toggled.
Interface Builder

Object Palette

Displays an element that shows the user the boolean state of a given value. By tapping the control, the state can be toggled.
Interface Builder

Design Window
Interface Builder
Tool Summary

- IDE: Xcode
- Simulator
- Performance: Instruments, Shark
- UI: Interface Builder
- Compiler: gcc
- Debugger: gdb
- Unit Testing: SenTest
Hello World! demo
Introduction

Basic Objective-C
Objective–C

• Very thin layer on top of C
  - Adds OOP syntax for messaging, classes

• Dynamic runtime component
  - powerful introspection
  - dynamic typing
  - easy to extend classes
Objective–C

• Mature programming language

• History
  - Invented in 80s by Brad Cox and Tom Love
  - Inspired by ‘smalltalk’
  - Popularized by NeXT during 80s, 90s – now Mac OS X

• Actively under development
  - Garbage collection
  - ObjC 2.0 properties
C Programming

• Need to know a little C
C Programming

- Need to know a little C
- Reference to an object declared using pointer syntax
C Programming

- Need to know a little C
- Reference to an object declared using pointer syntax

```c
NSString *aString = ...;
```
C Programming

• Need to know a little C

• Reference to an object declared using pointer syntax

```c
NSString *aString = ...;
```

• C is valid in an Objective-C program
C Programming

• Need to know a little C

• Reference to an object declared using pointer syntax

```c
NSString *aString = ...;
```

• C is valid in an Objective-C program

```c
#define BOILING 100

int celsius;
for (int celsius = 0; celsius < BOILING; ....
{
    float fahrenheit = (celsius * 1.8) + 32.0;
    printf("%d Celsius: %1.2f Fahrenheit\n", ...");
}
```
Basic Types
Basic Types

```java
int anInt = 10;
float aFloat = 10.0f;
char aChar = 'c';
...
```
Basic Types

```c
int anInt = 10;
float aFloat = 10.0f;
char aChar = 'c';
...
```

```c
// a basic objective-c object create / init
Person *person = [[Person alloc] init];
```
int anInt = 10;
float aFloat = 10.0f;
char aChar = 'c';
...

// a basic objective-c object create / init
Person *person = [[Person alloc] init];

// string objects receive special syntax / handling
NSString *aStringObject = @"A String"
Objective-C

• “.h” – header files contain declarations
• “.m” – objc implementation files
• “.mm” – objc++ implementation files
  - mix and match with c++
OOP With Objective-C

• Class
  - Blueprint to create instances
  - Defines encapsulation of state and behavior
OOP With Objective-C

• Class
  - Blueprint to create instances
  - Defines encapsulation of state and behavior

• Instance
  - Specific allocation of a particular class
OOP With Objective-C

• Class
  - Blueprint to create instances
  - Defines encapsulation of state and behavior

• Instance
  - Specific allocation of a particular class

• Method
  - A procedure that can be executed by an object
OOP With Objective-C

• Class
  - Blueprint to create instances
  - Defines encapsulation of state and behavior

• Instance
  - Specific allocation of a particular class

• Method
  - A procedure that can be executed by an object

• Message
  - The “ID” of a method
Messaging
Messaging

- Combination of method name, arguments, and instance
- Tell the receiving instance to do something
Messaging

- Combination of method name, arguments, and instance
- Tell the receiving instance to do something
Messaging

- Combination of method name, arguments, and instance
- Tell the receiving instance to do something

```objectivec
[person setName: aName]
```
Messaging

• Combination of method name, arguments, and instance
• Tell the receiving instance to do something

Takeaway: At a high-level sending a message is just calling a method. Instead of writing `foo.m(param)`, you send a message “m:” to the receiver “foo”, `[foo m: param]`
Messaging Syntax
Messaging Syntax

C++

```cpp
Person *person = // ...create object...
person->setName(aName);
```
Messaging Syntax

C++

Person *person = // ...create object...
person->setName(aName);

Java

Person person = ...
person.setName(aName);
Messaging Syntax

C++

Person *person = // ...create object...
person->setName(aName);

Java

Person person = ...
person.setName(aName);

Obj-C

Person *person = ...
[person setName:aName];
Person *person = // create it...

[person setName: aName];

message receiver
Messaging Syntax

Person *person = // create it...
[person setName: aName];
Messaging Syntax

Person *person = // create it...
[person setName: aName];
Named Parameters
Named Parameters

C++

```cpp
image->drawAtPointWithBlendModeAndAlpha(pt, mode, 0.5)
```
Named Parameters

C++
```cpp
image->drawAtPointWithBlendModeAndAlpha(pt, mode, 0.5)
```

Obj-C
```objc
[image drawAtPoint:pt blendMode:mode alpha:0.5];
```
Named Parameters

C++

```cpp
image->drawAtPointWithBlendModeAndAlpha(pt, mode, 0.5)
```

Obj-C

```objc
[image drawAtPoint:pt blendMode:mode alpha:0.5];
```

Advice

- Choose clarity over brevity
- Strive for self documenting code
@interface Person : NSObject {
    @private
        NSString *name;
}

+ (Person *)personWithName:(NSString *)aName;

- (void)setName:(NSString *)aName;
- (NSString *)name;

@end
Class / Instance

Person Class

+personWithName:

Person Instances

name
-name:
-name:
-name:
-name:
-name:
-name:
Class Declaration

@interface Person : NSObject {

begins class declaration

“.h” – often called an interface file
Class Declaration

@interface Person : NSObject {

class being declared

}
@interface Person : NSObject {
    superclass
}
Class Declaration

@interface Person : NSObject {
@private

 instance variable visibility

Options

• @private – visible only to the implementing class
• @protected - visible to subclasses (default)
• @public - visible to all classes
• @package - public within your project, private outside
@interface Person : NSObject {
@private
    NSString *name;
}

instance variable == "ivar"
@interface Person : NSObject {
@private
    NSString *name;
}

+ (Person *)personWithName:(NSString *)aName;

class methods – begin with “+”

ObjC – class method
C++ / Java – static function / method
@interface Person : NSObject {
    @private
    NSString *name;
}

+ (Person *)personWithName:(NSString *)aName;

- (void)setName:(NSString *)aName;
- (NSString *)name;

instance methods – begin with “-”

ObjC – instance method
C++ /Java – member function / method
@interface Person : NSObject {
    @private
    NSString *name;
}

+ (Person *)personWithName:(NSString *)aName;

- (void)setName:(NSString *)aName;
- (NSString *)name;

@end

ends a class declaration
Comparison
Declaring classes
Comparison

Declaring classes

ObjC

```objc
@interface MyObject : Object {
    @public
        MyCoolClass *var;
}
- (void)setCoolObject:(MyCoolObject *)coolObj;
- (MyCoolObject *)coolObject;
@end
```
Comparison

Declaring classes

C++

```cpp
class MyObject : Object {
    public:
        // instance variables
        MyCoolObject *mVar;
        
        void setCoolObject(MyCoolObject *)object;
        MyCoolObject *coolObject(void);
};
```

ObjC

```objc
@interface MyObject : Object {
    @public
        MyCoolClass *var;
} 
- (void)setCoolObject:(MyCoolObject *)coolObj;
- (MyCoolObject *)coolObject;
@end
```
Naming Instance Variables

• Standard Styles

```c
// no additional adornment
MyCoolClass *var;

// add a "_" at the end
MyCoolClass *var_;

// do it like C++
MyCoolClass *mVar;
```
Naming Instance Variables

• Standard Styles

```c
// no additional adornment
MyCoolClass *var;

// add a "_" at the end
MyCoolClass *var_;

// do it like C++
MyCoolClass *mVar;
```

• Reserved for Apple
Naming Instance Variables

• Standard Styles

```c
// no additional adornment
MyCoolClass *var;

// add a "_" at the end
MyCoolClass *var_;

// do it like C++
MyCoolClass *mVar;
```

• Reserved for Apple

```c
// Apple gets to use "_"
MyCoolClass *_var;
```
Naming Instance Variables

• Standard Styles

```cpp
// no additional adornment
MyCoolClass *var;

// add a “_” at the end
MyCoolClass *var_;

// do it like C++
MyCoolClass *mVar;
```

• Reserved for Apple

```cpp
// Apple gets to use “_”
MyCoolClass *_var;
```

- To avoid naming collisions
- Don’t use “_” in front of instance variables or method names
@implementation Person

- (void)setName:(NSString *)name {
    if (name != name) {
        [name release];
        name = [name retain];
    }
}

- (NSString *)name {
    return name;
}

@end
Object Definition

Initialization and Deallocaton

@implementation Person
-
-(id)init {

}
-
-(void)dealloc {

}
@end
@implementation Person

- (id)init {
    if ((self = [super init])) {
        name = @"No Name";
    }
    return self;
}

- (void)dealloc {
    [name release];
    [super dealloc];
}

@end
@implementation Person

- (id)init {
    if ((self = [super init])) {
        name = @”No Name”;  
    }
    return self;
}

- (void)dealloc {
    [name release];
    [super dealloc];
}

@end
Comparison

Member functions / methods
Comparison

Member functions / methods

C++

```cpp
void Person::foo(void) {
    String *string = this->getName();
}
...
```
Comparison

Member functions / methods

C++

```cpp
void Person::foo(void) {
    String *string = this->getName();
}
...
```

Obj-C

```objc
@implementation Person...

- (void)foo {
    NSString *string = [self name];
}
...
```
ObjC 2.0

• New features recently added to ObjC
  - Syntax for declared properties
  - Ability to auto–magically define property implementation
  - Simplified “dot” notation

• More on this in the next lecture…
Assignment – Reading
Assignment – Reading

• Lecture will be posted online later today!
Assignment – Reading

- Lecture will be posted online later today!
- Learn to navigate Apple’s developer docs
  http://developer.apple.com/iphone
Assignment – Reading

- Lecture will be posted online later today!
- Learn to navigate Apple’s developer docs
- Tools and Xcode
Assignment – Reading

• Lecture will be posted online later today!

• Learn to navigate Apple’s developer docs
  http://developer.apple.com/iphone

• Tools and Xcode

• Learning Objective–C
Assignment – Reading

• Download and start looking at...
  - CocoaFundamentals.pdf
  - ObjC.pdf
Stanford iPhone Class

• Stanford Class
  - http://cs193p.stanford.edu
  • Link seems to be broken at the moment?
  - iTunesU : iPhone Application Programming
Questions?