iPhone Programming
CMSC 498i – Spring 2010

Developing Complete Apps
Lecture #24 – Chuck Pisula
Announcements
Announcements

• Turn in Loaner iPods – Tuesday, May 11th in Class
• Course Evaluation – Now…
Tuesday Guest Lecture

- Todd Moore
  - Founded TMSOFT 2 years ago to build apps for smart phones
  - Best known for writing the app White Noise

- Guest Rules In Effect
Today’s Topics

• Application artwork and packaging
• Preferences and Settings
• Unit Testing
• Fun with Objective-C
Application Artwork
Application Artwork

- Application Icons
  - The app icon shown on the device
Application Artwork

- Application Icons
  - The app icon shown on the device
  - The app icon shown in iTunes / Store
Application Artwork

• Application Icons
  ▪ The app icon shown on the device
  ▪ The app icon shown in iTunes / Store

• Graphics and Icons inside your application

• Graphics shown during app launch
Application Artwork

• Application Icons
  ▪ The app icon shown on the device
  ▪ The app icon shown in iTunes / Store

• Graphics and Icons inside your application

• Graphics shown during app launch

• Screenshots used by App Store
Application Icon

• Get an icon
  ▪ Make your own using Photoshop, etc…
  ▪ Find free icons allowing commercial use
  ▪ Find someone to make them for you

• Worth the effort
  ▪ This may be all an AppStore user sees
  ▪ Customers assume icon quality == app quality!
Creative Commons License

- Creative Commons (CC)
- Not restricted to personal use only
- Categories
  - Attribution (CC-BY)
  - Attribution Share Alike (CC-BY-SA)
  - Attribution No Derivatives (CC-BY-ND)
Application Icon

- Some places I found for getting icons
  - Aggregating sites – some free, some personal use only (links back to artist)
    - http://www.iconarchive.com
    - http://www.veryicon.com
  - Pay sites
    - http://iconfactory.com
    - http://stockicons.com
    - http://www.apporacle.com
  - “Bake off Site” – you submit project with budget, designers compete
    - http://www.crowdspring.com
    - http://99designs.com
Application Icon

- PNG is the recommended format for any image on the device
- App Icon – square 57 x 57 pixel PNG
- Small icon for Spotlight search – 29 x 29 pixel PNG
- iPhone will automatically round the corners and make it “shiny”

- When submitting to the App Store – include a 512 x 512 icon
  - Named JPEG or PNG named “iTunesArtwork”
Application Icon

- Include in Xcode
  - As Copied Resources – Icon.png
  - Target > Your App > Icon File
Artwork

Launch Images
Example

- Settings Application
Launch Images

- Enhance the user’s experience at application launch
- Provides for better perceived responsiveness
  - You still need to make your launch time as fast as possible

- Launch Image Components
  - Default.png – 320x480 – Add to your Xcode projects Resources
  - Void of any text or aspects requiring localization
  - Void of any dynamic UI elements that might change
  - Goal – no flash / changes – only new UI once your app completes launching
Capturing Launch Images

- Launch on the device
- Use Xcode’s organizer “Screenshots” tab
- May need to customize drawing when launching to capture a screenshot
  - Example

```c
#define LAUNCHED_FOR_DEFAULT_PNG 1

#if LAUNCHED_FOR_DEFAULT_PNG
    [someView setHidden:YES]
#endif
```
Application Settings

Packaging
Info.plist

- Configure aspects of your application
Info.plist Keys

- Keys you may want to configure
  - UIPrerenderedIcon – can indicate you don’t want the automatic icon shine…
  - UIInterfaceOrientation – portrait or landscape initially?
  - UIStatusBarHidden – show status bar initially?
  - UIStatusBarStyle – gray, black, translucent

- UIRequiredDeviceCapabilities – dictionary specifying requirements
  - telephony, wifi, accelerometer, location-services, etc...
- CFBundleIdentifier – unique identifier like com.myCompany.myApp
Settings
Settings

- Application preferences displayed in Settings applications
- App Settings UI – No code!
  - You provide multiple plists describing the app settings UI
  - Organized into a specifically formatted bundle – Settings.bundle
    - Xcode provides Settings bundle template
    - Settings application loads, displays your bundles data
- Retrieving settings
  -NSUserDefaults or CFPreferences
Settings Bundle

• Contents
  - Added as a new file in Xcode project
  - Hierarchical plist schema – Root.plist, addition child plists
**Settings**

- Example

![Example Settings](image)

- Accessing the setting

```objective-c
NSUserDefaults *ud = [NSUserDefaults standardUserDefaults];
BOOL playSounds = [ud boolForKey:@"play_sounds_preferences"];```

```javascript
// Accessing the setting
let ud = NSUserDefaults.standardUserDefaults();
let playSounds = ud.boolForKey("play_sounds_preferences");
```
Settings HCI

• Apple very consciously limits the number of settings
  ▪ Avoid throwing in every switch “just because”

• Apple HIG makes 2 recommendations
  ▪ Put in Settings application
    ▪ Default behavior overrides, Infrequent options
    ▪ **Examples**: Mail account information, Safari search provider
  ▪ Keep in your application
    ▪ For frequently changed options
    ▪ **Examples**: Stock symbols, Map/Satellite/Hybrid in Maps
Unit Testing
Unit Tests

- Just as compiler checks syntax, unit tests check semantics
- Test specific areas of functionality frequently during development
- When should you write tests?
  - Ideally written along with new code
  - In conjunction with bug fixes
  - Test-driven development - write test and use as implementation spec
- When should you run unit tests?
  - Run tests every time you build
Why Unit Test?

• Fewer bugs
  ▪ More confidence that you're shipping a high quality product

• Find bugs **early**
  ▪ Bugs are easier (and cheaper) to fix early in development

• Avoid **regressions**
  ▪ Ensure that changing one piece of code doesn't break another

• **Document** your code
  ▪ How is a method intended to be used? Check out the tests...

• Encourage **good design**
  ▪ Spaghetti code is hard to test! Design with **testability** in mind
OCUnit – Basics

- iPhone uses the open-source SenTestingKit framework (OCUnit)

- SenTestCase is abstract test case superclass

- Automatically runs methods that begin with "test"

- Macros for asserting conditions during tests
  - STAssertNotNil(someObject, @"Some object was nil");
  - See SenTestCase.h for more

- -setUp and -tearDown methods run before and after each test
Defining A New Test Case Class

```objective-c
#import <SenTestingKit/SenTestingKit.h>

@class Foo;

@interface FooTests : SenTestCase {
  Foo *foo;
}
@end
```
Preparing Tests

@implementation FooTests

- (void)setUp {
    // Every test will have its own Foo instance
    foo = [[Foo alloc] init];
}

- (void)tearDown {
    [foo release];
}

...
@end
Example Tests

@implementation FooTests

- (void)testCreateFoo {
    Foo *fooObj = ...;
    STAssertNotNil(fooObj, @"Couldn't create Foo");
}

- (void)testSetBar {
    Bar *barObj = ...;
    Foo *fooObj.bar = bar;
    STAssertEqualObjects(fooObj.bar, barObj, @"Couldn't set foo.bar");
}

- (void)testLegalValue {
    STAssertTrue([foo barValue] <= 100, @"bar should always be <= 100");
}

@end
When Does Unit Testing Make Sense?

- Always be conscious of the return on investment
  - Benefit of the test versus time to create and maintain?
- Some types of code are notoriously difficult to test
  - Networking
  - User Interface
  - Often possible to test a subset of behavior and still benefit
Demo

Unit Testing an iPhone App
Fun with Objective-C
Fun With Objective-C

• **Introspection** – Dynamically figure out classes details including methods, instance variables, and more...

  ```c
  -(BOOL)respondsToSelector:(SEL)sel;
  -(BOOL)isKindOfClass:(Class)cls;
  ```

• **Swizzling** – Dynamically replace a class / method with your own

  ```c
  // objc/runtime.h
  IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types)
  ```

• **Message Forwarding** – forward method invocations to other objects

  ```c
  -(void)forwardInvocation:(NSInvocation *)invocation {
  ...
  }
  ```

• **Invocations** – freeze-dried representation of a method + arguments

  ```c
  NSInvocation *invocation = [NSInvocation invocationWithMethodSignature: ms];
  [invocation setTarget: calculator];
  [invocation setArgument: &intValue atIndex:2];
  [invocation invoke];
  [invocation getReturnValue: &result];
  ```
Related APIs

- Objective–C runtime
  - `<objc/objc.h>` – id, Nil, nil, BOOL, YES, NO
  - `<objc/message.h>` – objc_msgSend() and friends
  - `<objc/runtime.h>`
    - Inspect and manipulate classes, protocols, methods
    - Add and replace methods at runtime

- Foundation framework methods
  - NSObject.h
  - NSInvocation.h
  - NSMethodSelector.h
Objective-C Runtime

#include <objc/runtime.h>
#include <objc/message.h>
...

BOOL class_respondsToSelector(Class cls, SEL sel)

objc_allocateClassPair(Class superclass, const char *name, size_t extraBytes)
void objc_registerClassPair(Class cls)

const char * ivar_getName(Ivar ivar)
Ivar object_getInstanceVariable(id obj, const char *name, void **outValue)
Ivar object_setInstanceVariable(id obj, const char *name, void *value)

BOOL class_addMethod(Class cls, SEL name, IMP imp, const char *types)
IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types)

Class class_setSuperclass(Class cls, Class newSuper)
BOOL class_addIvar(Class cls, char *name, size_t s, uint8_t align, const char *types)

void objc_setAssociatedObject(id obj, void *key, id value, objc_AssociationPolicy p)
id objc_getAssociatedObject(id object, void *key)
Oh, The Possibilities

- Delegation
  - **Introspection** (-respondsToSelector:) to know if a method is implemented

- Key Value Coding
  - KVC’s –setValue:forKey:
  - Runtime **introspection** determines which getters/setters/ivars to be used

- Key Value Observing
  - Observed properties setters are overridden by the runtime by **swizzling** individual methods
  - Posts “will change”, calls **real** implementation, the posts “did change” observation callbacks
Oh, The Possibilities

• Multiple Inheritance
  ▪ “Inherit” through composition and forwarding messages to other objects

• NSZombies
  ▪ Debug feature to help track down retain/release imbalances (over releases)
  ▪ Instead of being deallocated, objects are swizzled into NSZombie objects which raise exceptions when messaged…

• Proxy objects
  ▪ Allow different objects to stand in for another
  ▪ Example – Via swizzling, KVC/KVO returns proxy array object from NSMutableViewArray returning methods. Proxy can capture changes…
  ▪ Example – On Mac OS X, NSConnection returns proxy objects that forward messages to the real objects on potentially remote machines
Oh, The Possibilities

• Undo

  ▪ Simple Syntax

    ```
    [[undoMgr prepareInvocationWithTarget: foo] setTitle:@"oldTitle"];
    ```

  ▪ Like freeze-drying a context that will execute

    ```
    [foo setTitle:@"oldTitle"];
    ```

  ▪ `prepareInvocationWithTarget:` returns a **proxy** object, in this case, sometimes called a **trampoline**

  ▪ This particular proxy object does not forward, and instead pushes an **NSInvocation** onto the undo stack

    ```
    -(void)forwardInvocation:(NSInvocation *)invocation {
    // save invocation for later, when user selects “Undo”!
    }
    ```
Oh, The Possibilities

• Trampolines – easy to make your own, just like NSUndoManager
  ▪ Example: add a “do” method to NSArray

```objective-c
for (MyObject *obj in array) {
    [obj setEnabled:NO];
}
[[myArray do] setEnabled:NO]; // calls setEnabled:NO on each item
```
Trampolines – easy to make your own, just like NSUndoManager

- Example: add a “do” method to NSArray

Implement “do” method in a NSArray category

```
- (id)do {
    return [MyArrayProxy proxyWithArray:self];
}
```

MyArrayProxy implements no methods, but will forward invocations

```
@implementation (MyArrayProxy)

- (void)forwardInvocation:(NSInvocation *)invocation {
    for (id object in array_) {
        [invocation invokeWithTarget:object];
    }
}
```

Oh, The Possibilities

- **Trampolines** – easy to make your own, just like NSUndoManager
  - Example: add a “do” method to NSArray

- Implement “do” method in a NSArray category

```
- (id)do {
    return [MyArrayProxy proxyWithArray:self];
}
```

- MyArrayProxy implements no methods, but will forward invocations

```
@implementation (MyArrayProxy)

- (void)forwardInvocation:(NSInvocation *)invocation {
    for (id object in array_) {
        [invocation invokeWithTarget:object];
    }
}
```
Oh, The Possibilities

- Unit Testing with SenTesting framework
  - How does OCUnit find all the methods that begin with “test”?  
  - Use ObjC runtime API to dig through classes, and methods

- Testing with mock objects – OCMock
  - Example: Create a mock text field

```c
UITextField *textField;
urlTextField = [OCMockObject mockForClass:[UITextField class]];
```

- Convenient proxy-trampoline style way to stub out the -text method

```c
NSString *testableURL = @"http://foo/bar";
[[[urlTextField stub] andReturn: testableURL] text];

STAssertEqualObjects([urlTextField text], @"http://foo/bar", ....);
```
Runtime Exposes Everything

- Runtime inspection – means nothing is really private at runtime!
The Problem With Using Private APIs

- Framework APIs are kept private for one of a few reasons:
  - They’re not done yet (and *will probably change*)
  - They’re never going to be public (and *may disappear*)
- Not just because Apple wants to hide cool stuff from you
- If your app depends on a private API that goes away...
  - At best, your app won’t work correctly anymore
  - More often, your app will just crash
- Apps submitted to Apple can be rejected if using private API
Reading

• iPhone Development Guide.pdf
  ▪ Chapter 7, Unit Testing – p.61 - 73

• iPhone Application Programming Guide.pdf
  ▪ Application Settings – p.191 - 202

• MobileHIG.pdf
  ▪ Creating Custom Icons and Images – p.121 - 126