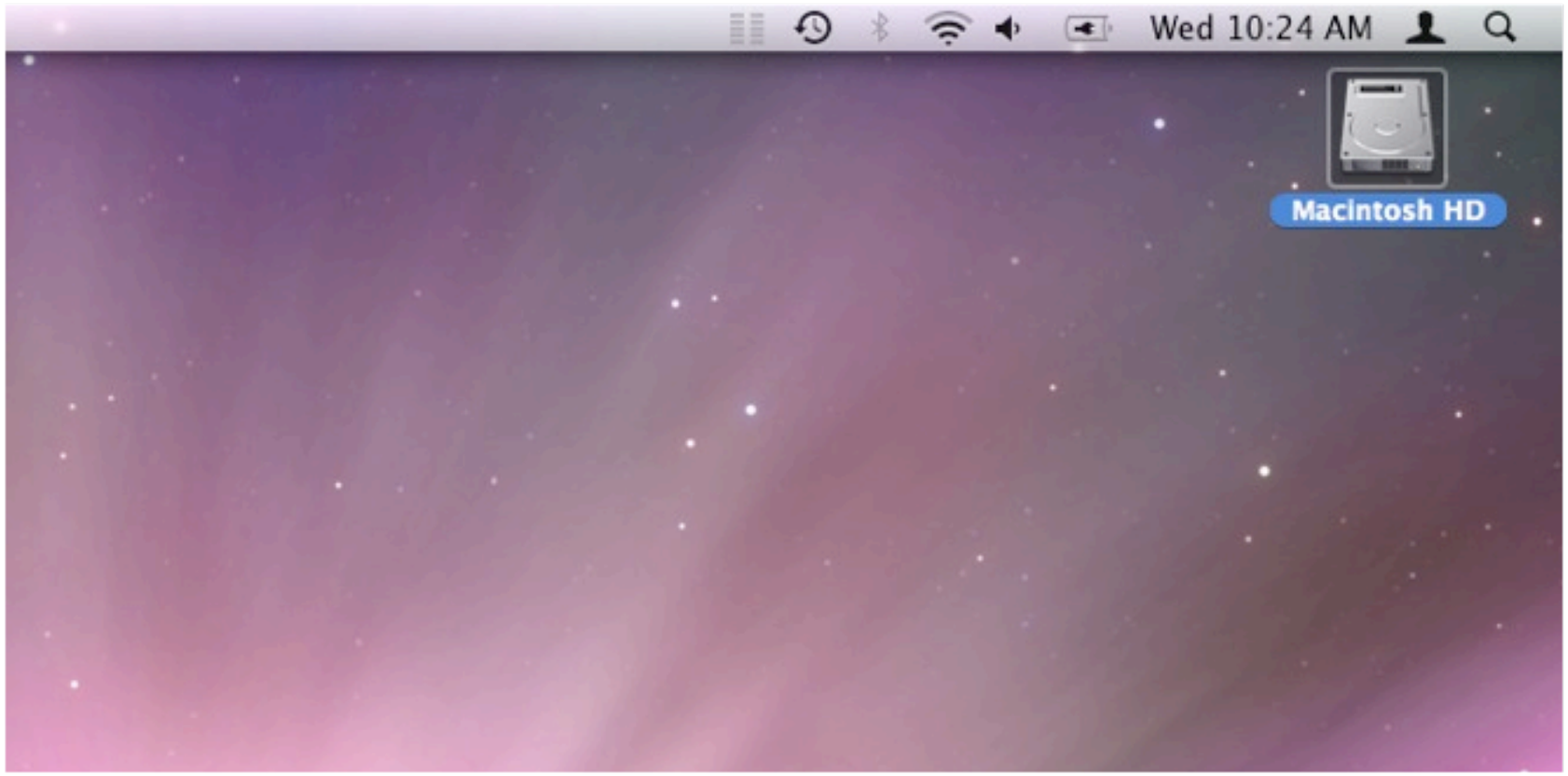
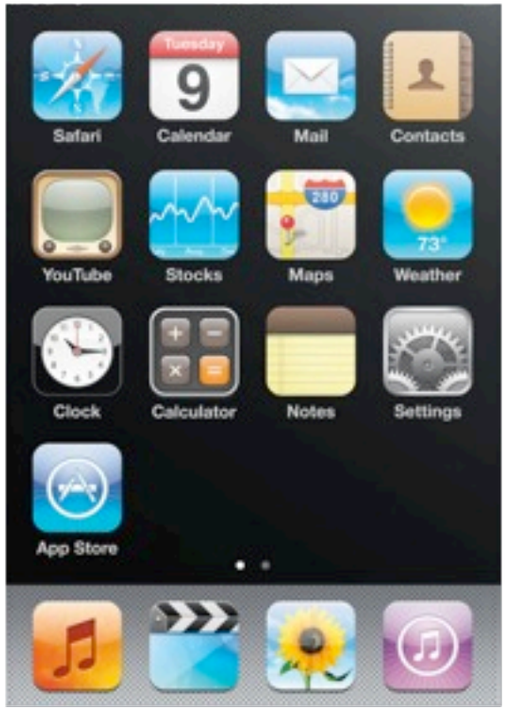




Intro to Cocoa: Part One

A crash course in programming Macs, iPhones, and iPod Touches

Stephen Voida, August 2009

Workshop Objectives

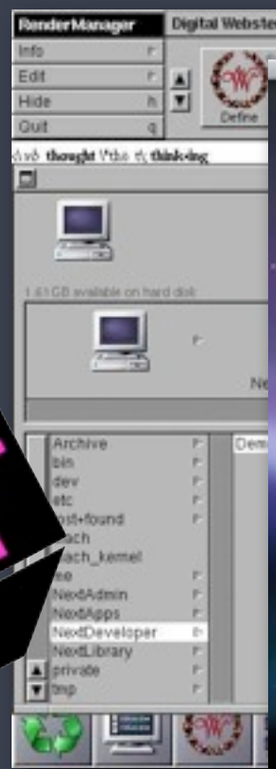
- High-level understanding of how the OS X platform is organized and how Cocoa applications are implemented
- Introduction to the OS X and iPhone SDKs and development tools
- Quick-and-dirty overview of Objective-C
- Brief tour of the Cocoa and UIKit frameworks, focusing on rapid application development and UI software prototyping
- Illustration of how all the pieces come together with code samples

Agenda: Day 1 of 2

- A bit of history and perspective on OS X
- Intro to the Apple development tools (Xcode, Interface Builder)
- Intro to the Objective-C programming language
- Intro to the Cocoa frameworks
- Putting it all together: A simple UI app from start to finish
- *Next week: Making the leap from the desktop to the iPod*

Just a bit about OS X


hello


RenderManager
Digital Webster
Info
Edit
Hide
Quit
Define
Archive
bin
dev
etc
NextAdmin
NextApps
NextDeveloper
NextLibrary
private
tmp


Finder File Edit View Go Window Help
Wed 10:24 AM
Macintosh HD

OS X: Vital Stats

- Derived from the NextStep operating system
- Based on the BSD variant of UNIX and the Mach-O kernel
  - All your favorite command-line tools: sh, grep, man, perl, **gcc, gdb**
- PostScript- and PDF-based graphics subsystems
- Smalltalk-inspired object-oriented programming tools
- User-facing Aqua UI layer hides the complexity of UNIX
- Same underpinnings for the Mac and the iPhone/iPod Touch

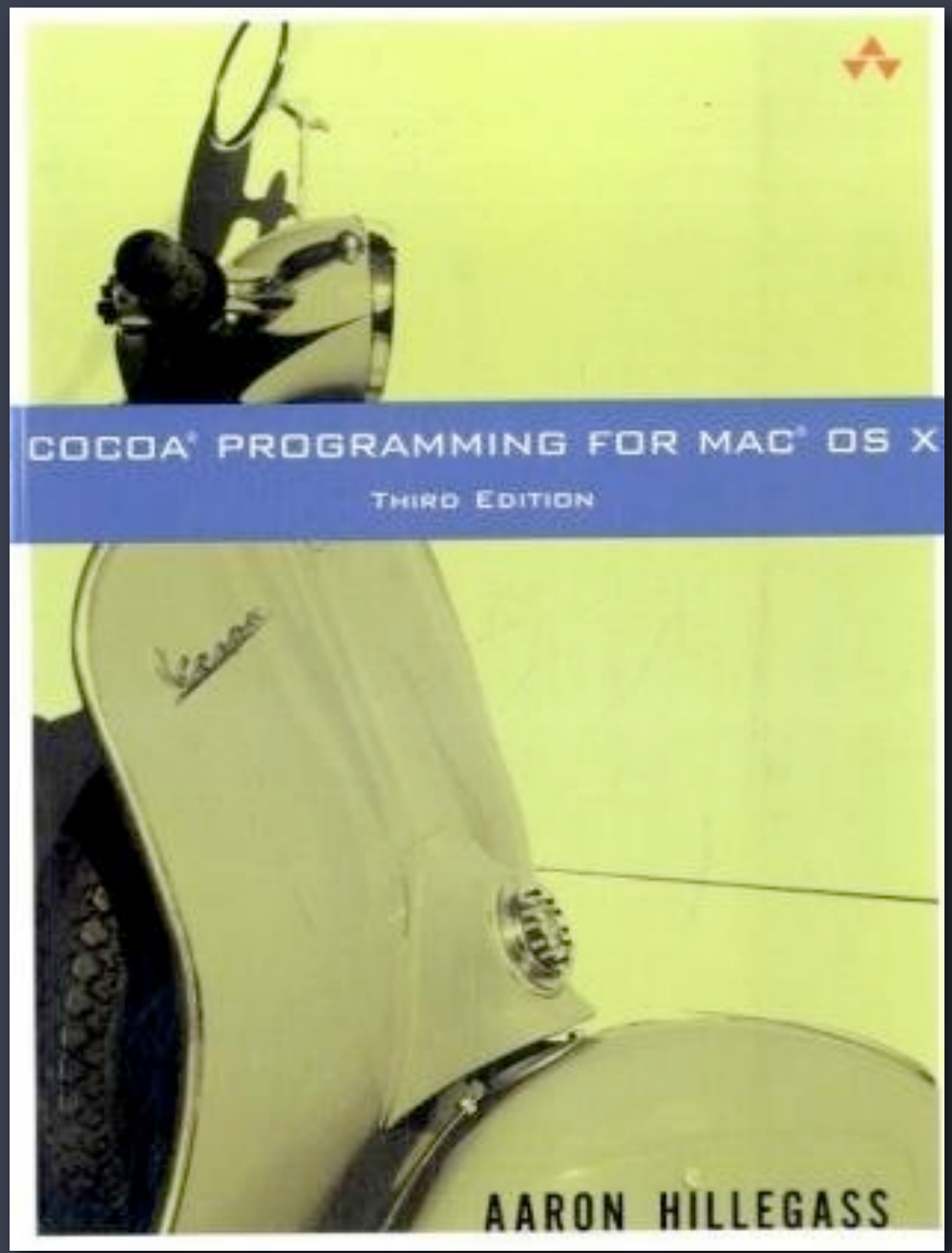
Intro to the Dev Tools

Xcode

Interface Builder

(Disclaimer)

Borrowing heavily here from Hillegass, A. *Cocoa Programming for Mac® OS X* (3rd Ed.). Addison-Wesley, Upper Saddle Ridge, NJ, 2008.



The Objective-C Language

- Simple set of object-oriented extensions to regular C
- Compiled using the same compiler (gcc) as regular C programs
- Can combine Objective-C code, C++ code, and/or regular C code in the same program and even in the same source code file
- Not the *only* option for programming Macs, iPhones, and iPods, but by far the most used and best supported

Notice: Objective-C looks stranger than it really is...

- In Java:

```
if (x.intersectsArc(35.0, 19.0, 23.0, 90.0, 120.0)) {
```

- In Objective-C:

```
if ([x intersectsArcWithRadius:35.0
                      centeredAtX:19.0
                                Y:23.0
                        fromAngle:90.0
                          toAngle:120.0]) {
```

(Tangent) Really Important Keyboard Shortcuts for Xcode!

- <Esc> : Activate CodeSense—provides a list of valid completions *and* a full stub for the selector you want to use
- Ctrl+/ : Jump to the next replaceable parameter in a selector and select it for replacement
- Command+] : Indent the current line (block) one tab
- Command+[: Outdent the current line (block) one tab
- Command+/ : Comment out the current line (block)

Instantiating and Referring to Objects

- Objects are stored using pointers

```
NSMutableArray *array;
```

- `alloc` allocates memory for an object; `init` initializes it

```
NSMutableArray *array;
array = [NSMutableArray alloc];
[array init];
```

- Method calls can be chained together

```
NSMutableArray *array = [[NSMutableArray alloc] init];
```

```
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[NSMutableArray alloc] init];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [[NSNumber alloc] initWithInt:(i * 3)];
        [array addObject:newNumber];
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@“The number at index %d is %@”, i, numberToPrint);
    }

    [pool drain];
    return 0;
}
```

Other Objective-C Quirks

- `id` is the miscellaneous object pointer type (kind of equivalent to `void*` in C++)
- `BOOL` is the built-in Boolean type and is equivalent to a `char`
  - `YES = 1`
  - `NO = 0`
- `nil` is Objective-C's `NULL` when referring to objects

Nil-Targeted Messages: The Smalltalk Legacy

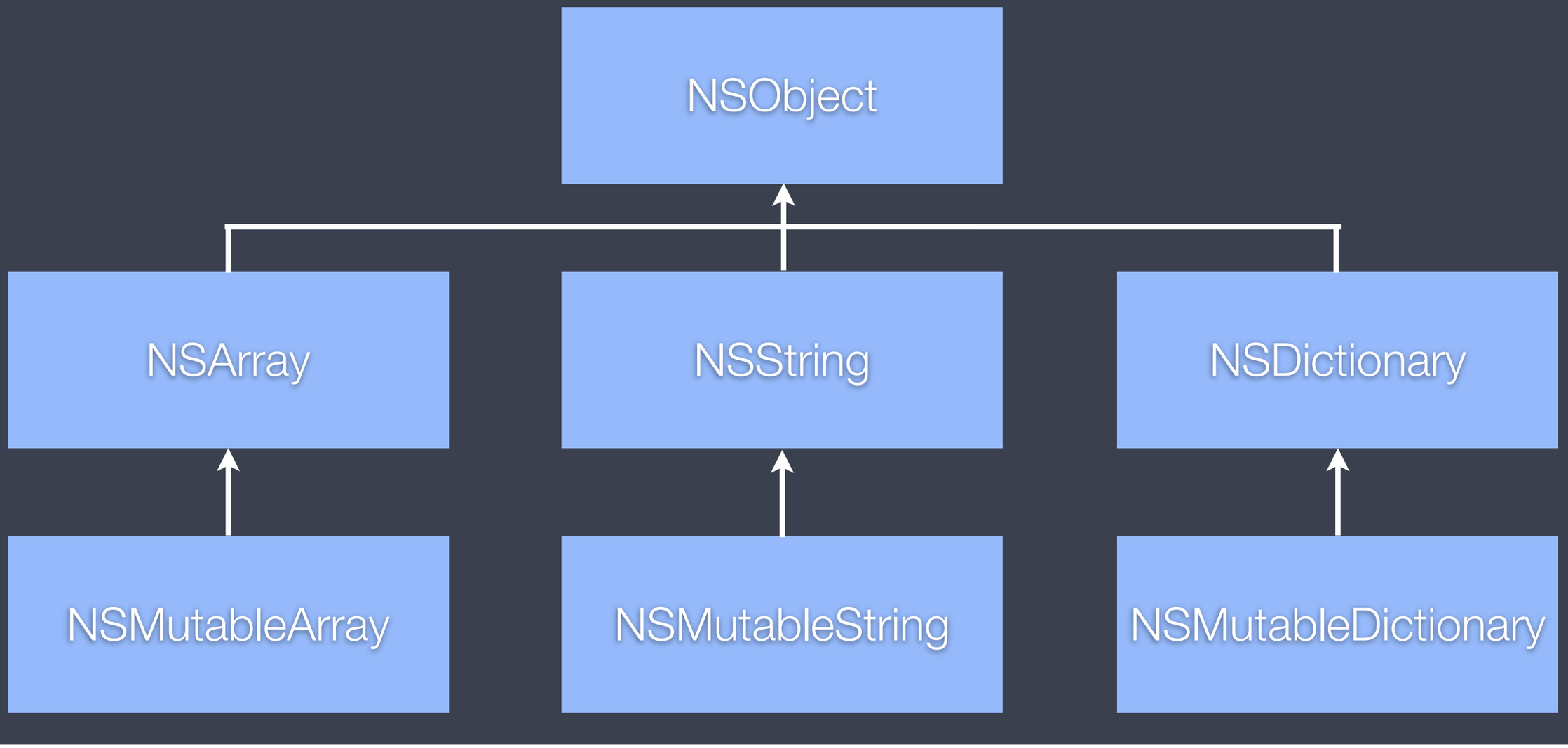
- Sending messages to `nil`
- Can't do this in Java/C++!

```
java.lang.Vector v = null;
int vSize = v.size();          // Crash!
```

- Smalltalk doesn't care…so neither does Objective-C

```
NSMutableArray *v = nil;
int vSize = [v count];         // vSize = 0
```

Some Very Common Classes



NSObject

- Base class for everything else in Cocoa
- A few common methods:

```
- (id)init                          obj = [[TheClass alloc] init];


- (NSString *)description          NSLog(@“Obj: %@”, [obj description]);
                                   NSLog(@“Obj: %@”, obj);
- (BOOL)isEqual:(id)anObject       if ([x isEqual:y]) ...
                                   if (x == y) ...
```

NSArray

- Equivalent to a Java Vector *but contents cannot be changed*
- 0-based indexing

```
- (id)initWithObjects:(id)firstObj, ...
```

```
ar = [ar initWithObjects:x, y, z];
```

```
- (unsigned)count
```

```
int i = [ar count];
```

```
- (id)objectAtIndex:(unsigned)index
```

```
id obj = [ar objectAtIndex:0];
```

```
- (id)lastObject
```

```
id obj = [ar lastObject];
```

```
- (BOOL)containsObject:(id)anObject
```

```
if ([ar containsObject:x]) ...
```

NSMutableArray

- Same as NSArray, but includes ability to change the array contents

```
- (void)addObject:(id)anObject                [mar addObject:w];

- (void)insertObject:(id)anObject atIndex:(unsigned)index
                                              [mar addObject:w
                                                 atIndex:3];

- (void)removeObject:(id)anObject             [mar removeObject:w];

- (void)removeObjectAtIndex:(unsigned)index
                                              [mar removeObjectAtIndex:3];

- (void)removeAllObjects                      [mar removeAllObjects];
```

NSString

- Sequence of Unicode characters (distinct from a typical C string)
  - NSString *anNSString = @“ABCDE”; char *aCString = “abcde”;

```
- (id)initWithFormat:(NSString *)format, ...

        s = [s initWithFormat:@“The current value is %d”, someInt];

- (unsigned int)length                    int i = [s length];

- (NSString *)stringByAppendingString:(NSString *)aString

        NSString *s = @“before”;
        s = [s stringByAppendingString:@“after”];
        NSLog(s);                              // Prints “beforeafter”
```

NSMutableString

- Adds ability to change characters to a regular NSString

```
- (void)appendWithFormat:(NSString *)format, …

        [ms appendWithFormat:@“ and its value is %d”, someInt];

- (void)insertString:(NSString *)aString atIndex:(unsigned)index

        [ms insertString:@“((Steve was here))” atIndex:3];

- (void)deleteCharactersInRange:(NSRange)aRange

        NSMutableString *ms = @“abcdefg”;
        [ms deleteCharactersInRange:NSMakeRange(1,2)];
        NSLog(ms);                          // Prints “adefg”
```

NSDictionary

- Equivalent to a Java Hashtable *but contents cannot be changed*

```
- (NSArray *)allKeys
```

```
NSArray* ar = [dict allKeys];
```

```
- (unsigned)count
```

```
int i = [dict count];
```

```
- (id)objectForKey:(NSString *)aKey
```

```
id obj = [dict objectForKey:@“someKey”];
```

```
- (NSEnumerator *)keyEnumerator
```

```
NSEnumerator *e = [dict keyEnumerator];
for (NSString *s in e) {
    NSLog(@“key is %@, value is %@”, s, [dict objectForKey:s]);
}
```

NSMutableDictionary

- You can probably guess where this is going...
- Keys are always NSStrings, but stored objects can be any Objective-C class (NSString, NSNumber, NSArray, NSDictionary…)

```
- (void)setObject:(id)anObject forKey:(NSString *)aKey
                                [mdict setObject:x forKey:@“myKey”];
- (void)removeObjectForKey:(NSString *)aKey
                                [mar removeObjectForKey:@“myKey”];
```

Creating Classes

- First, you need a header file…

```
#import <Foundation/Foundation.h>

@interface LuckyNumbers : NSObject {
    NSMutableString *personName;
    int firstNumber;
    int secondNumber;
}
- (void)prepareRandomNumbers;
- (void)setPersonName:(NSString *)name;
- (NSString *)personName;
- (int)firstNumber;
- (int)secondNumber;
@end
```

- Then, you need the implementation…

```
#import “LuckyNumbers.h”

@implementation LuckyNumbers

- (void)prepareRandomNumbers {
    firstNumber = random() % 100 + 1;
    secondNumber = random() % 100 + 1;
}

- (void)setPersonName:(NSString *)name {
    personName = name;
}

- (NSString *)personName {
    return personName;
}

- (int)firstNumber {
    return firstNumber;
}

- (int)secondNumber {
    return secondNumber;
}

@end
```

What about an initializer?

- Could add the following code to *override* the default initializer from NSObject…

```
- (id)init {
    self = [super init];
    if (self) {
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}
```

Or, alternatively...

```
// Override the default initializer, providing default values
- (id)init {
    return [self initWithName:@“Jane Doe”];
}

// Provide a “designated initializer” to do all the work
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = name;
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}
```

- Now, you can call `[obj initWithName:@“Sheelagh”]` as well. (The regular, no-parameter version is also still available…)

⚠ Memory Management

- Cocoa decides if an object's memory can be freed based on a system of *retain counts*.
- When object *y* is created using `alloc`, its retain count is set to 1.
- When object *x* is interested in object *y*, it increments object *y*'s retain count by sending it `retain`.
- When object *x* is finished with object *y*, it decrements object *y*'s retain count.
- When object *y*'s retain count becomes 0, it is deallocated and its memory is freed.

Memory Management: Manipulating Retain Counts

- Setting the retain count to 1

```
[object alloc]
[object copy]
[object mutableCopy]
```

- Increasing the retain count

```
[object retain]
```

- Decreasing the retain count

```
[object release]
[object autorelease]
```

Memory Management: Rules of Thumb

- Rule #1: If you `alloc`, `retain`, or `copy` it, it's your job to `release` it. Otherwise, it isn't.
- Rule #2: If you `alloc`, `retain`, or `copy` it, it's your job to `release` it. Otherwise, it isn't.
- Rule #3: `autorelease` and methods that return `autoreleased` objects are your friend.
- Rule #4: If a Cocoa app crashes, it's almost certainly because you `released` something that you shouldn't have.

Shamelessly stolen from http://www.cocoadev.com

A few fixes...

```
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = name;
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}

- (void)setPersonName:(NSString *)name {
    personName = name;
}
```

A few fixes...

```
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = [name retain];              // need to retain it!
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}

- (void)setPersonName:(NSString *)name {
    personName = [name retain];                  // need to retain it!
}
```

A few fixes...

```
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = [name retain];
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}

- (void)dealloc {
    [personName release];          // Deallocate what we retained
    [super dealloc];               // Before dealloc'ing ourselves
}

- (void)setPersonName:(NSString *)name {
    personName = [name retain];
}
```

A few fixes...

```
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = [name retain];
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}

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

- (void)setPersonName:(NSString *)name {
    [name retain];                  // “Retain-then-release” prevents
    [personName release];           // inadvertent deallocation
    personName = name;              // if someone sends a ref for the
}                                   // active personName to the setter
```

A few fixes...

```
- (id)initWithName:(NSString *)name {
    self = [super init];
    if (self) {
        personName = [name copy];
        firstNumber = random() % 100 + 1;
        secondNumber = random() % 100 + 1;
    }
    return self;
}

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

- (void)setPersonName:(NSString *)name {
    [name copy];                    // copy is even safer than retain
    [personName release];           // (and doesn't add all that much
    personName = name;              // overhead for NSStrings)
}
```

NSObject

- Base class for everything else in Cocoa
- A few common methods:

```
- (id)init                          obj = [[TheClass alloc] init];


- (NSString *)description          NSLog(@“Obj: %@”, [obj description]);
                                   NSLog(@“Obj: %@”, obj);
- (BOOL)isEqual:(id)anObject       if ([x isEqual:y]) ...
                                   if (x == y) ...
```

Returning a description

```
- (NSString *)description {
    NSString *result;
    result = [[NSString alloc] initWithFormat:@“%@’s lucky numbers
            are %d and %d”, personName, firstNumber, secondNumber];
    return result;
}
```

Returning a description

```
- (NSString *)description {
    NSString *result;
    result = [[NSString alloc] initWithFormat:@“%@’s lucky numbers
            are %d and %d”, personName, firstNumber, secondNumber];
    return result;
}
```

Returning a description

```
- (NSString *)description {
    NSString *result;
    result = [[NSString alloc] initWithFormat:@“%@’s lucky numbers
            are %d and %d”, personName, firstNumber, secondNumber];
    [result release];                  // would deallocate before return
    return result;
}
```

Returning a description

```
- (NSString *)description {
    NSString *result;
    result = [[NSString alloc] initWithFormat:@"%@'s lucky numbers
            are %d and %d", personName, firstNumber, secondNumber];
    [result autorelease];          // decrements retain count but
    return result;                 // doesn't deallocate right away
}
```

```
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[NSMutableArray alloc] init];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [[NSNumber alloc] initWithInt:(i * 3)];
        [array addObject:newNumber];
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@"The number at index %d is %@", i, numberToPrint);
    }

    [pool drain];
    return 0;
}
```

```
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[NSMutableArray alloc] init];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [[NSNumber alloc] initWithInt:(i * 3)];
        [array addObject:newNumber];
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@"The number at index %d is %@", i, numberToPrint);
    }

    [pool drain];
    return 0;
}
```

```objc
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[NSMutableArray alloc] init];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [[NSNumber alloc] initWithInt:(i * 3)];
        [array addObject:newNumber];
        [newNumber release];
        newNumber = nil;                // Not strictly necessary here...
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@"The number at index %d is %@", i, numberToPrint);
    }

    [array release];                    // ... but it's generally good
    array = nil;                        // practice to nil released vars

    [pool drain];
    return 0;
}
```

```objc
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[[NSMutableArray alloc] init] autorelease];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [[NSNumber alloc] initWithInt:(i * 3)];
        [newNumber autorelease];
        [array addObject:newNumber];
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@"The number at index %d is %@", i, numberToPrint);
    }

    [pool drain];
    return 0;
}
```

```objc
#import <Foundation/Foundation.h>

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableArray *array;
    array = [[[NSMutableArray alloc] init] autorelease];
    int i;
    for (i = 0; i < 10; i++) {
        NSNumber *newNumber = [NSNumber numberWithInt:(i * 3)];
        [array addObject:newNumber];
    }

    for (i = 0; i < 10; i++) {
        NSNumber *numberToPrint = [array objectAtIndex:i];
        NSLog(@"The number at index %d is %@", i, numberToPrint);
    }

    [pool drain];
    return 0;
}
```

A few other great resources...

- Hillegass book
- Apple Developer Documentation
  - Inside of Xcode
  - http://developer.apple.com
- CocoaDev (http://www.cocoadev.com)
- Me!

Agenda: Day 1 of 2

- A bit of history and perspective on OS X
- Intro to the Apple development tools (Xcode, Interface Builder)
- Intro to the Objective-C programming language
- Intro to the Cocoa frameworks
- Putting it all together: A simple UI app from start to finish
- *Next week: Making the leap from the desktop to the iPod*