Objective C for iPhone Developement

June 2013

Objective

- Basic development skill in Objective-C (Beginner level)
- The syntax of the Objective-C programming language

Prerequisite

- Basic programming language skill

Assessment Disciplines

- Class Participation: 40%
- Assignment: 60%
- Final Exam: 0%
- Passing Scores: 70%

Agenda

- Introduction
- Classes, Objects, and Methods
- Working with Declared Properties
- Control Structures in Objective-C
- Objective-C Data Type
- Basic Memory Management

Introduction

Introduction

- Objective-C is an object-oriented programming language.
- Implemented as set of extensions to the C language. Everything that works in C also works in Objective-C.
- A very simple language, but some new syntax.
- The primary language used to write Mac software.
- The latest version of objective-C is 2.0.

Classes, Objects, and Methods

Objective-C Files: .h and .m

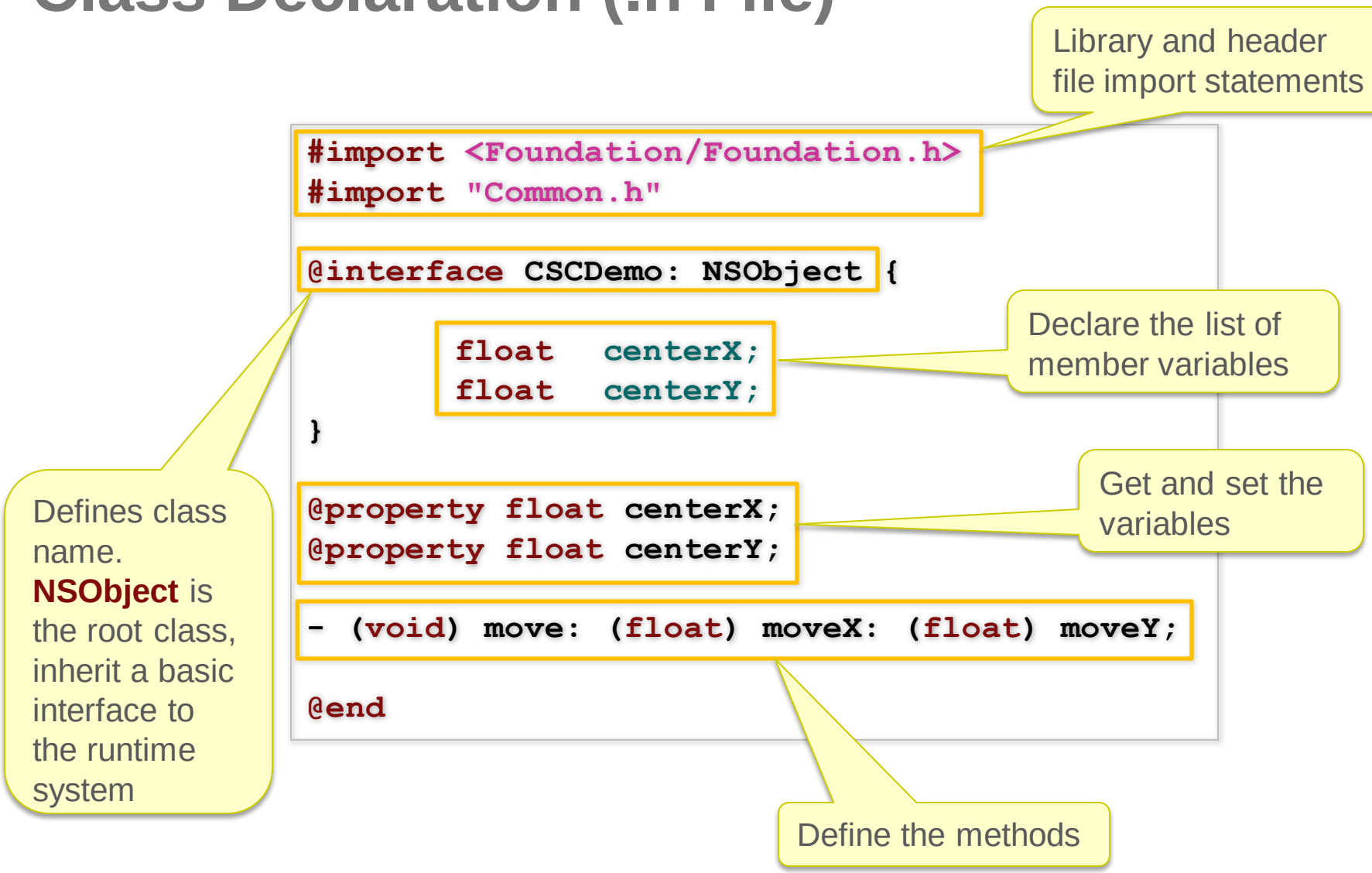
- **<FileName>.h**
  - The header file that contains the declaration of a class.
  - Contains declaration of member variables and methods of the class.
  - Use the **"#import"** statement to include other .h files (e.g. Foundation.h)
- **<FileName>.m**
  - Contains the definition (implementation) of the methods of the corresponding class, manipulation of data and objects.
  - We need to use the **"#import"** statement to include the header <fileName>.h file

Examples

HelloWorld.h

HelloWorld.m

Class Declaration (.h File)



Class Definition (.m File)

```
#import "CSCDemo.h"

@implementation CSCDemo
@synthesize centerX, centerY;


- (void) move: (float) moveX: (float) moveY
{
        self.centerX = moveX;
        self.centerY = moveY;
}

- (void) dealloc {
        [supper dealloc];
}

@end
```

Standard implementation syntax (with @end)

Corresponds to the @property statements of the .h file

Contain implementation of the method

Must be included in all classes, related to object de-allocation

Scoping Instance Variables

```
@interface User : NSObject
{
      @private
            bool isActivated;

      @public
            NSString *userName;

      @protected
            NSString *userId;

}
// …
@end
```

The instance variable is accessible only within the class that declares it.

The instance variable is accessible everywhere.

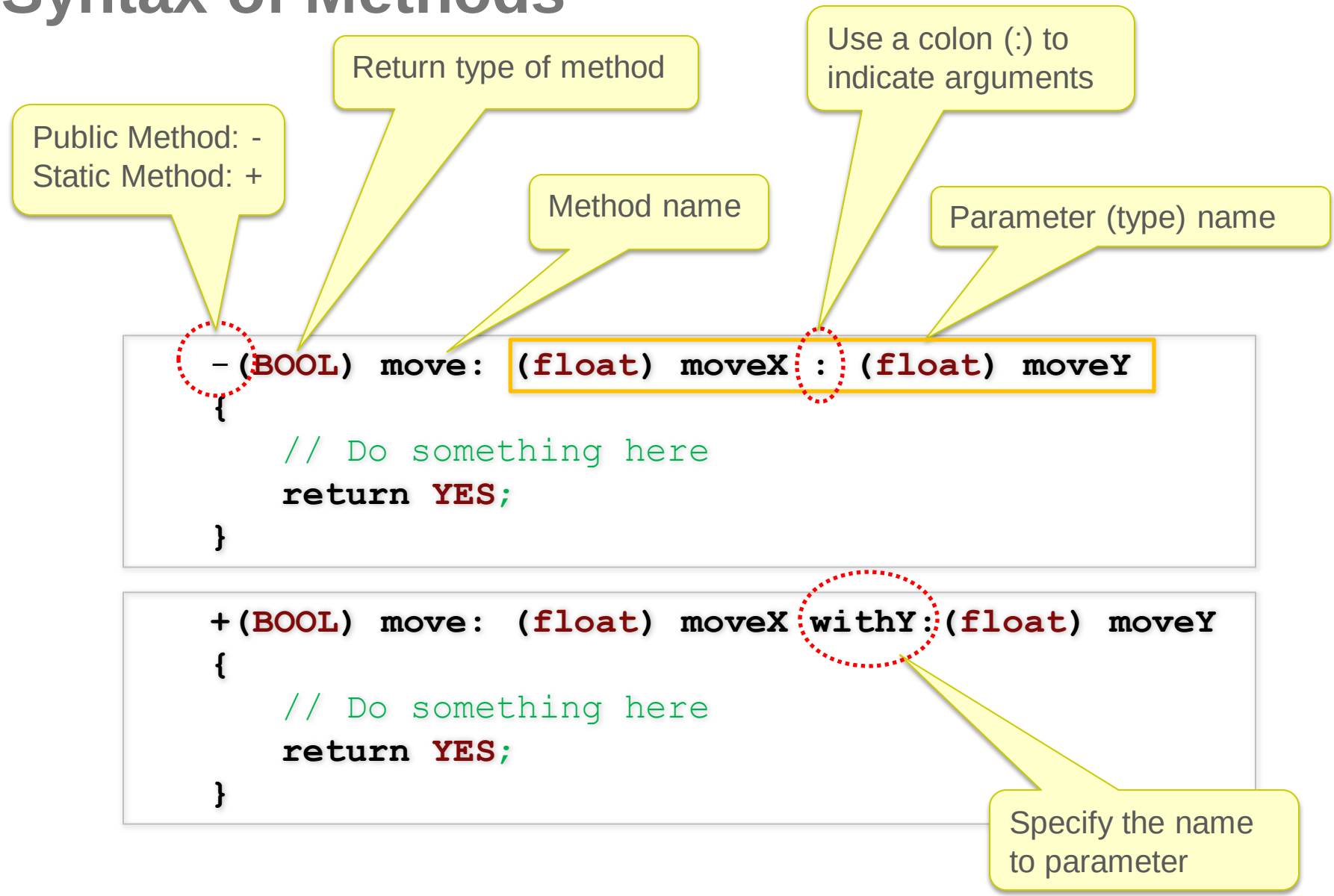
The instance variable is accessible within the class that declares it and within classes that inherit it.
This is the default if you don't specify anything.

Intrinsic Variable Types

- Intrinsic variable types we use in C/C++ also work in Objective-C . Below is some of the supported intrinsic variables types

Type	Description	Size	Ranges
char	A character	1 byte	-128 → 127
int	Integer	4 bytes	-2147483648 → 2147483647
float	Single precision floating point number	4 bytes	7 digits
double	Double precision floating point number	8 bytes	15 → 16 digits
short	A short integer	2 bytes	-32768 → 32767
BOOL	Boolean variable	1 byte	

Syntax of Methods



The id type

- Objective-C has a type called **id**, that acts in some ways like a void*, though it's meant strictly for objects.
- It doesn't need to know the object type.

Examples

```
@interface CSCDemo: NSObject
{
  //Declare Variables
}
-(IBAction)changeText:(id)sender;
@end
```

Allocating Objects

- You must do the following:
  1) Dynamically allocate memory for the new object.
  2) Initialize the newly allocated memory, as described in the upcoming “Initialization” section.
- You can use the new method to combine **alloc** and **init**

Examples

```
//using alloc/init
id anObject = [[Rectangle alloc] init];
trip = [[Trip alloc] initWithName:@”Road Trip”];
//using new
Budget *europeBudget = [Budget new];
```

Calling Methods

- **[myDemo ...];**
  [... move<parameters>];
  Calls the "**move**" method with parameters **moveX**, and **moveY** are float variables already defined
- **[myDemo move];**
  If the "**move**" method accepts no parameters, then simply write
- **myPhoto.caption = @"Day at the Beach";**

  The dot syntax for getters and setters is new in Objective-C 2.0

Examples

```
[myDemo move:10 20];
NSLog(@"My name: %@, age: %d", @"David", 20);
```

Constant strings are @"My String"
NSLog: Show results to debug console

Working with Declared Properties

Working with Declared Properties

Declare instance variables

```
//.h header file
@interface Destination : NSObject {
        NSString* country;

}
@property (nonatomic, retain) NSString* country;

@end
```

Add @property declaration of that instance variables

```
//.m implementation file
@implementation Destination
@synthesize country;
```

Add @synthesize statement in the implementation file

Working with Declared Properties

- nonatomic: Setter and getter won't guarantee a complete, viable value is returned in a threaded environment. Faster than atomic and generally what you want to use when programming for iOS. By default, accessors are "atomic".
- retain: directs the compiler to create an accessor method that sends a retain message to any object that is assigned to this property.

Accessing the properties

Examples

```
[self setCountry:theCountry];

//Or
self.country = theCountry;

//Releasing the object assigned to a property
[country release];
```

Selectors identify methods by name

- A selector has type SEL

```
SEL action = [button action];
[button setAction:@selector(start:)];
```

- Conceptually similar to function pointer
- Selectors include the name and all colons

Examples

```
- (void) setName:(NSString *)name age:(int)age;
//would have a selector
SEL sel = @selector(setName:age:);
//
SEL sel = @selector(start:);
if ([obj respondsToSelector:sel]){
  [obj performSelector:sel withObject:self]
}
```

Exercise #1

Working With Methods

Control Structures in Objective-C

Control Structures in Objective-C

- We can use C control structures in developing iPhone: while, do-while, for loops, if, switch statement

Examples

```
if ( [userType isEqualToString:@"new"] )
{
        for (int i=0; i < numToDraw; i++ )
        {
                [[displayList objectAtIndex: i] draw];
        }
}

while ( !found )
{
        [graphicObject draw];
}
```

Exceptions

- Objective-C provides a mechanism to handle exceptions with the compiler directives: @try, @catch, and @finally.

Examples

```
id hamburger = @"hamburger";

@try {
    hamburger = [hamburger addCheese];
}
@catch( NSException *exc ){
    NSLog( @"Exception: %@ : %@", [exc name], [exc reason]);
    NSException* myException = [NSException
        exceptionWithName:@"MyException"
        reason:@"Testing Exception" userInfo:nil];
    @throw myException;
}
@finally {
        NSLog( @"This block is always executed.");
}
```

Class Introspection

- You can ask an object about its class

```
Class myClass = [myObject class];

NSLog(@"My class is %@", [myObject className]);
```

- Testing for general class membership (subclasses included):

```
if ([myObject isKindOfClass:[UIControl class]]) {
   // something
   }
```

- Testing for specific class membership (subclasses excluded):

```
if ([myObject isMemberOfClass:[NSString class]]) {
   // something string specific
   }
```

Exercise #2

Working With Control Structures

Objective-C Data Type

Objective-C String: NSString

Examples

```
int age = 20;
float tall = 1.8;

NSString * nameStr = @"David";

NSString * string1 = [NSString stringWithFormat:@"%@
age %d, %@ tall %f ", nameStr, age, nameStr, tall];

//or
NSString * string2 = [NSString stringWithFormat:@"%1$@
age %2$d, %1$@ tall %3$f ", nameStr, age, tall];

// NSLog is to output string to the Console
NSLog(@"Show log: %@",string1);

The output on the console is:
    Show log: David age 20, David tall 1.8
```

Objective-C String: NSString

- Common NSString methods

```
// Append string
- (NSString *)stringByAppendingString:(NSString *)aString;
- (NSString *)stringByAppendingFormat:(NSString *)aString;

// Split string to array
- (NSArray *)componentsSeparatedByString:(NSString *)aString;

// Find characters
- (NSRange)rangeOfString:(NSString *)aString;

// Compare
- (BOOL)isEqualToString:(NSString *)aString;
- (BOOL)hasPrefix:(NSString *)aString;

// Convert
- (int)intValue;
- (double)doubleValue;
```

Objective-C String: NSString

Examples

```
// Append string
NSString *greet = @"Hello";
NSLog(@"%@", [greet stringByAppendingString:@", World"]);

// Split string to array
NSArray *strings = [coords componentsSeparatedByString:@","];

// Compare
if ([myString isEqualToString:@”mydata”]){
   //Do something
}

// Convert
int age = [ageStr intValue];
```

Objective-C String: NSString

Examples

```
//substring
NSString *searchKeyword = @"your string";
NSRange rangeOfYourString = [string rangeOfString:searchKeyword];

if (rangeOfYourString.location == NSNotFound)
{
   NSLog(@"Not found!");
}
else{
   NSString *subString = [string
                substringToIndex:rangeOfYourString.location];
   NSLog(@"Found!");
}
```

Objective-C String: NSString

Examples

```
// Formatting
//      %@: Any Object-C object including NSString objects
//      %d: int
//      %f: float/double,
//      %.xf: float/double with decimal places "x" number

NSString *searchKeyword = @"your string";
NSRange rangeOfYourString = [string rangeOfString:searchKeyword];

if (rangeOfYourString.location == NSNotFound)
{
   NSLog(@"Not found!");
}
else{
   NSString *subString = [string
                substringToIndex:rangeOfYourString.location];
   NSLog(@"Found!");
}
```

Objective-C Number: NSNumber, NSNumberFormatter

Examples

```
// Number with 2 decimal round up
NSNumberFormatter *formatter = [NSNumberFormatter new];
[formatter setNumberStyle: NSNumberFormatterDecimalStyle];
[formatter setMaximumFractionDigits:2];
[formatter setRoundingMode: NSNumberFormatterRoundUp];

NSString *numberString = [formatter stringFromNumber:
                [NSNumber numberWithFloat:123456789.3685]];

NSLog(@"Result...%@",numberString);//Result 123,456,789.37
```

Objective-C DateTime: NSDate

Examples

```
// Convert Date/Time to string
NSDateFormatter *dateFormat = [NSDateFormatter new];
[dateFormat setDateFormat:@"MM/dd/yyyy"];

NSDate *now = [NSDate date]; //Current date

NSString *theDate = [dateFormat stringFromDate:now];

// Add one day to current date
int daysToAdd = 1;
NSDate *newDate1 = [now
        dateByAddingTimeInterval:60*60*24*daysToAdd];

// Compare date: newDate1 > now
if ([newDate1 compare:now] == NSOrderedDescending)
{
    // Do something
}
```

Exercise #3

Working With Basic Data Type

Objective-C Array

- We can simply use C/C++ style of creating array

Examples

```
int myArray[100];
myArray[0] = 11;
myArray[1] = 22;

int integers[5] = { 0, 1, 2, 3, 4 };
int a[] = { 0, 1, 2, 3, 4, 5};

int M[4][5] = {
                        { 10, 5, -3, 17, 82 },
                        { 9, 0, 0, 8, -7 },
                        { 32, 20, 1, 0, 14 },
                        { 0, 0, 8, 7, 6 }
                };
```

Objective-C Array: NSArray

- NSArray is static and cannot be changed at runtime. We have to define all the elements when we initialize by using the arrayWithObjects method.
- The array has to be terminated by “nil”.
- Get/Find element by objectAtIndex, indexOfObject

Examples

```
NSArray *array = [NSArray
        arrayWithObjects:@”Red”, @”Blue”, @”Green”, nil];
for (int i=0; i < [array count]; i++){
        NSLog(@"array[%d] = %@", i,
                [array objectAtIndex:i]);
}
if ([array indexOfObject:@”Purple”] == NSNotFound) {
        NSLog (@”No color purple”);
}
```

Objective-C Array: NSMutableArray

- By using NSMutableArray, we can dynamically modify the array elements by using the methods addObject, removeObjectAtIndex, removeAllObjects, replaceObjectAtIndex

Examples

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

[array addObject:@"Red"];
[array addObject:@"Green"];
[array addObject:@"Blue"];
[array removeObjectAtIndex:1];

[name release];
```

Objective-C Array: NSDictionary

- Manage immutable associations of keys and values.
- Common NSDictionary methods: dictionaryWithObjectsAndKeys, count, objectForKey

Examples

```
NSDictionary *colors = [NSDictionary
       dictionaryWithObjectsAndKeys:
              @"Color Red", @"KeyColor1",
              @"Color Green", @"KeyColor2",
              @"Color Blue", @"KeyColor3", nil];

NSString *firstColor = [colors
       objectForKey:@"KeyColor1"];
if ([colors objectForKey:@"KeyColor3"]) {
   // Do something
}
```

Objective-C Array: NSMutableDictionary

- Manage mutable associations of keys and values
- Common NSMutableDictionary methods: setObject, removeObjectForKey, removeAllObjects

Examples

```
NSMutableDictionary *colors =
        [[NSMutableDictionary alloc] init];

[colors setObject:@"Orange" forKey:@"HighlightColor"];

[colors release];
```

Enumeration

- Consistent way of enumerating over objects in collections: NSArray, NSMutableArray, NSDictionary...

Examples

```
NSArray *array = [NSArray
        arrayWithObjects:@"Red", @"Blue", @"Green", nil];

for (NSString *color in array) {
        NSLog(color);
}

[array release];
```

Basic Memory Management

Basic Memory Management

- Objective-C provides two methods of application memory management
  - In the method described in this guide, referred to as "manual retain-release" or MRR, you explicitly manage memory by keeping track of objects you own. This is implemented using a model, known as reference counting, that the Foundation class NSObject provides in conjunction with the runtime environment
  - In Automatic Reference Counting, or ARC, the system uses the same reference counting system as MRR, but it inserts the appropriate memory management method calls for you at compile-time. You are strongly encouraged to use ARC for new projects. If you use ARC, there is typically no need to understand the underlying implementation described in this document, although it may in some situations be helpful.

Basic Memory Management

- To manage its memory, Objective-C (actually Cocoa) uses a technique known as reference counting. Every object has its own reference count, or retain count.
- If you create an object using the manual alloc style, you need to release the object later. You should not manually release an autoreleased object (or using ARC) because your application will crash if you do.

Examples

```
// string1 will be released automatically
NSString *string1 = [NSString
       stringWithFormat:@"%@",@"myString"];

// NSMutableArray
NSMutableArray *array = [[NSMutableArray alloc] init];
[array addObject:@"myArray"];
[array release]; //cannot use this code in mode ARC
```

Exercise #4

Working With Array

Q & A