CS193E
Lecture 2

Object Oriented Programming
The Objective C Language
Foundation Classes

Announcements

- Assignment 1A - download updated copy from web site
- Assignment 1B - available tonight on class website
  - Will send email to class when available
- Both Assignment 1A and 1B due next Friday 1/18, by 5:00 PM
  - After these, due date will be the Weds 11:59 PM following the date assignment goes out
- If you finish early, try submission script out

Today's Topics

- Questions from Tuesday or Assignment 1A?
- Object Oriented Programming Overview
- Objective-C Language
- Common Foundation Classes

Finding things out

- The assignment walks you through it
- Key spots to look
  - API & Conceptual Docs in Xcode
  - Class header files
  - Docs, sample code, tech notes on web
    - http://developer.apple.com
    - Dev site uses Google search

Objects

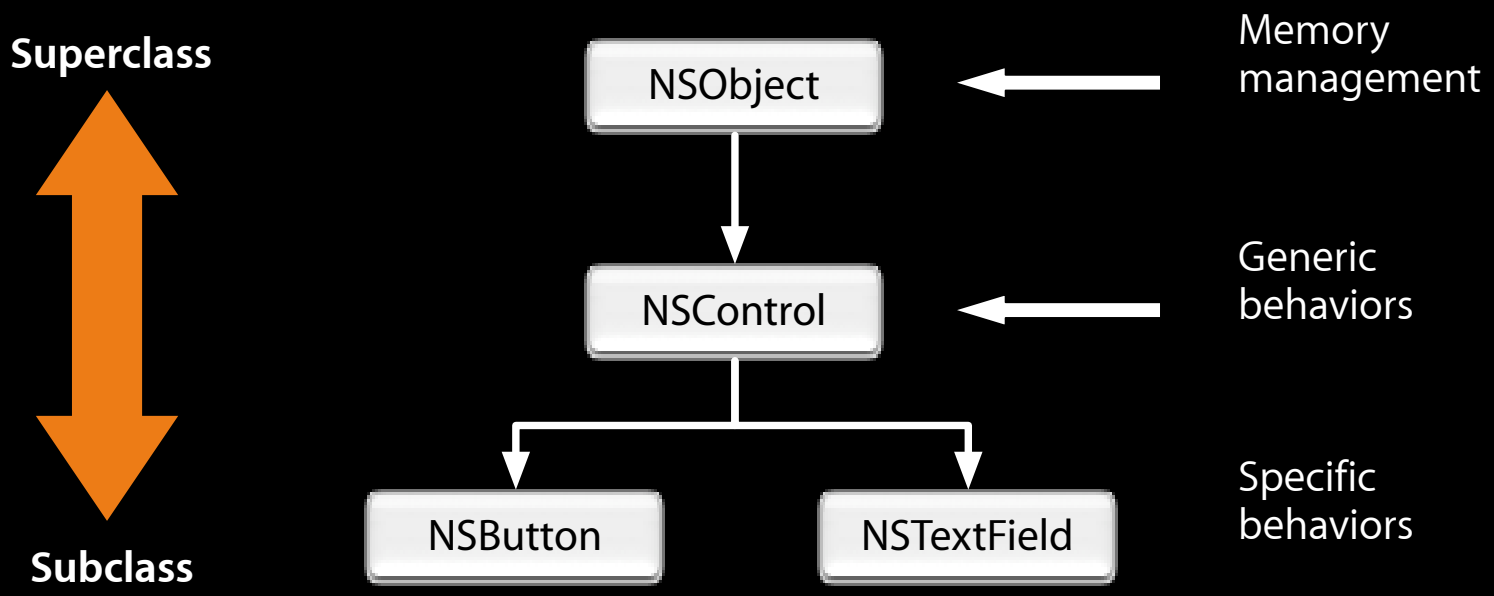
OOP Vocabulary

- **Class**: defines the grouping of data and code, the "type" of an object
- **Instance**: a specific allocation of a class
- **Method**: a "function" that an object knows how to perform
- **Instance Variable (or "ivar")**: a specific piece of data belonging to an object

OOP Vocabulary

- Encapsulation
  - keep implementation private and separate from interface
- Polymorphism
  - different objects, same interface
- Inheritance
  - hierarchical organization, share code, customize or extend behaviors

Inheritance



- Hierarchical relation between classes
- Subclass "inherit" behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

More OOP Info?

- Drop by David's office hours to cover basics of OOP
- Tons of books and articles on OOP
- Most Java or C++ book have OOP introductions
- http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Objective-C

Objective-C

- Strict superset of C
- A very simple language, but some new syntax
- Single inheritance, classes inherit from one and only one superclass.
- Protocols define behavior that cross classes
- Dynamic runtime
- Loosely typed, if you'd like

Defining a class

A public header and a private implementation

Header File

Implementation File

Defining a class

A public header and a private implementation

Header File

Implementation File

Class interface declared in header file

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

@interface Person : NSObject
{
   // instance variables
   NSString *name;
   int age;
}

// method declarations
- (NSString *)name;
- (void)setName:(NSString *)value;
- (int)age;
- (void)setAge:(int)age;
- (BOOL)canLegallyVote;

// alternative setter
- (void)setName:(NSString *)name age:(int)age;
@end
```

Class interface declared in header file

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

@interface Person : NSObject
{
   // instance variables
   NSString *name;
   int age;
}

// method declarations
- (NSString *)name;
- (void)setName:(NSString *)value;
- (int)age;
- (void)setAge:(int)age;
- (BOOL)canLegallyVote;

@end
```

Class interface declared in header file

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

@interface Person : NSObject
{
   // instance variables
   NSString *name;
   int age;
}

// method declarations
- (NSString *)name;
- (void)setName:(NSString *)value;
- (int)age;
- (void)setAge:(int)age;
- (BOOL)canLegallyVote;

@end
```

Class interface declared in header file

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

@interface Person : NSObject
{
   // instance variables
   NSString *name;
   int age;
}

// method declarations
- (NSString *)name;
- (void)setName:(NSString *)value;
- (int)age;
- (void)setAge:(int)age;
- (BOOL)canLegallyVote;

@end
```

Class interface declared in header file

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

@interface Person : NSObject
{
   // instance variables
   NSString *name;
   int age;
}

// method declarations
- (NSString *)name;
- (void)setName:(NSString *)value;
- (int)age;
- (void)setAge:(int)age;
- (BOOL)canLegallyVote;

@end
```

Defining a class

A public header and a private implementation

Header File

Implementation File

Private implementation defines methods

```
#import "Person.h"

@implementation Person

// method implementations
- (int)age {
   return age;
}
- (void)setAge:(int)value {
   age = value;
}
- (BOOL)canLegallyVote {
   return (age > 17);
}
// and others as declared in header...

@end
```

Private implementation defines methods

```
#import "Person.h"

@implementation Person

// method implementations
- (int)age {
   return age;
}
- (void)setAge:(int)value {
   age = value;
}
- (BOOL)canLegallyVote {
   return (age > 17);
}
// and others as declared in header...

@end
```

Private implementation defines methods

```
#import "Person.h"

@implementation Person

// method implementations
- (int)age {
   return age;
}
- (void)setAge:(int)value {
   age = value;
}
- (BOOL)canLegallyVote {
   return (age > 17);
}
// and others as declared in header...

@end
```

self and super

Methods have an implicit local variable named “self”

(like “this” in C++)

```
- (void)doSomething {
  [self doSomethingElseFirst];
    ...
}
```

Also have access to methods of the superclass using super

```
- (void)doSomething {
  [super doSomething];
    ...
}
```

Messaging syntax

Message examples

```
Person *voter; //assume this exists

[voter castBallot];

int theAge = [voter age];

[voter setAge:21];

if ([voter canLegallyVote]) {
   // do something voter-y
}

[voter registerForState:@"CA" party:@"Independant"];

NSString *name = [[voter spouse] name];
```

Objective-C Types

Dynamic and static typing

- Dynamically-typed object

```
id anObject
```

- Statically-typed object

```
Person *anObject
```

- Objective-C provides compile-time, not runtime, type checking

- Objective-C always uses dynamic binding

The null object pointer

- Test for nil explicitly

```
if (person == nil) return;
```

- Or implicitly

```
if (!person) return;
```

- Can use in assignments and as arguments if expected

```
person = nil;
[button setTarget: nil];
```

- Sending a message to nil?

```
person = nil;
[person castBallot];
```

BOOL typedef

- When ObjC was developed, C had no boolean type (C99 introduced one)
- ObjC uses a typedef to define BOOL as a type

```
BOOL flag = NO;
```

- Macros included for initialization and comparison: YES and NO

```
if (flag == YES)
if (flag)
if (!flag)
if (flag != YES)
flag = YES;
flag = 1;
```

Selectors identify methods by name

- A selector has type SEL

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

- Selectors include the name and all colons, for example:

```
-(void)setName:(NSString *)name age:(int)age;
```

would have a selector:

```
SEL sel = @selector(setName:age:);
```

- Conceptually similar to function pointer

Working with selectors

- You can determine if an object responds to a given selector

```
id obj;
SEL sel = @selector(start:);
if ([obj respondsToSelector:sel]) {
   [obj performSelector:sel withObject:self]
}
```

- This sort of introspection and dynamic messaging underlies many Cocoa design patterns

```
-(void)setTarget:(id)target;
-(void)setAction:(SEL)action;
```

Working with Classes

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:[NSControl class]]) {
   // something
}
```

- Testing for specific class membership (subclasses excluded):

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

Class Methods

- Instance methods operate on a specific object
- Class methods are global and have no specific data associated with them
- '-' denotes instance method

```
- (void)printName;
```

- '+' denotes class method

```
+ (NSApplication *)sharedApplication;
```

- You invoke a class method by messaging the class itself

```
[NSApplication sharedApplication];
```

Working with Objects

Identity versus Equality

- Identity—testing equality of the pointer values

```
if (object1 == object2) {
  NSLog(@"Same object instance");
}
```

- Equality—testing object attributes

```
if ([object1 isEqual: object2]) {
  NSLog(@"Logically equivalent");
}
```

-description

- NSObject implements -description

```
- (NSString *)description;
```

- Whenever an object appears in a format string, it is asked for its description

```
[NSString stringWithFormat: @"The answer is: %@", myObject];
```

- You can log an object's description with:

```
NSLog([anObject description]);
```

- Your custom subclasses can override description to return more specific information

Foundation Classes

Foundation Framework

- Value and collection classes
- User defaults
- Archiving
- Notifications
- Undo manager
- Tasks, timers, threads
- File system, pipes, I/O, bundles

NSObject

- Root class
- Implements many basics
  - Memory management
  - Introspection
  - Object equality

NSString

- General-purpose Unicode string support
  - Unicode is a coding system which represents all of the world's languages
- Consistently used throughout Cocoa instead of "char *"
- Without doubt the most commonly used class
- Easy to support any language in the world with Cocoa

String Constants

- In C constant strings are

  "simple"

- In ObjC, constant strings are

  @"just as simple"

- Constant strings are NSString instances

  NSString *aString = @"Hello World!";

Format Strings

- Similar to printf, but with %@ added for objects

```
NSString *log = [NSString stringWithFormat: @"It's '%@'", aString];
```

- Also used for logging

```
NSLog(@"I am a %@, I have %d items", [array className], [array count]);
```

NSString

- Often ask an existing string for a new string with modifications

```
- (NSString *)stringByAppendingString:(NSString *)string;
- (NSString *)stringByAppendingFormat:(NSString *)string;
- (NSString *)stringByDeletingPathComponent;
```

- Example:

```
NSString *myString = @"Hello";
NSString *fullString;
fullString = [myString stringByAppendingString:@" world!"];
```

NSString

- Common NSString methods

```
- (BOOL)isEqualToString:(NSString *)string;
- (BOOL)hasPrefix:(NSString *)string;
- (int)intValue;
- (double)doubleValue;
```

- Example:

```
NSString *myString = @"Hello";
NSString *otherString = @"449";
if ([myString hasPrefix:@"He"]) {
      // will make it here
}
if ([otherString intValue] > 500) {
      // won't make it here
}
```

NSMutableString

- NSMutableString subclasses NSString
- Allows a string to be modified
- Common NSMutableString methods

```
- (void)appendString:(NSString *)string;
- (void)appendFormat:(NSString *)format, ...;
+ (id)string;
```

```
NSString *newString = [NSMutableString string];
[newString appendString:@"Hi"];
[newString appendFormat:@", my favorite number is: %d",
   [self favoriteNumber]];
```

Collections

- Array - ordered collection of objects
- Dictionary - collection of key-value pairs
- Set - unordered collection of unique objects
- Common enumeration mechanism
- Immutable and mutable versions
  - Immutable collections can be shared without side effect
  - Prevents unexpected changes
  - Mutable objects typically carry a performance overhead

NSArray

- Common NSArray methods

```
- arrayWithObjects:(id)firstObj, ...;  // nil terminated!!!
- (unsigned)count;
- (id)objectAtIndex:(unsigned)index;
- (unsigned)indexOfObject:(id)object;
```

- NSNotFound returned for index if not found

```
NSArray *array = [NSArray arrayWithObjects:@"Red", @"Blue",
@"Green", nil];
if ([array indexOfObject:@"Purple"] == NSNotFound) {
      NSLog (@"No color purple");
}
```

- Be careful of the nil termination!!!

NSMutableArray

- NSMutableArray subclasses NSArray
- So, everything in NSArray
- Common NSMutableArray Methods
  - `- (void)addObject:(id)object;`
  - `- (void)removeObject:(id)object;`
  - `- (void)removeAllObjects;`
  - `- (void)insertObject:(id)object atIndex:(unsigned)index;`

```
NSMutableArray *array = [NSMutableArray array];
[array addObject:@"Red"];
[array addObject:@"Green"];
[array addObject:@"Blue"];
[array removeObjectAtIndex:1];
```

NSDictionary

- Common NSDictionary methods
  - `- dictionaryWithObjectsAndKeys: (id)firstObject, ...;`
  - `- (unsigned)count;`
  - `- (id)objectForKey:(id)key;`
- nil returned if no object found for given key

```
NSDictionary *colors = [NSDictionary
      dictionaryWithObjectsAndKeys:@"Color 1", @"Red",
      @"Color 2", @"Green", @"Color 3", @"Blue", nil];

NSString *firstColor = [colors objectForKey:@"Color 1"];


if ([colors objectForKey:@"Color 8"]) {

    // won't make it here

}
```

NSMutableDictionary

- NSMutableDictionary subclasses NSDictionary
- Common NSMutableDictionary methods

```
- (void)setObject:(id)object forKey:(id)key;
- (void)removeObjectForKey:(id)key;
- (void)removeAllObjects;
```

```
NSMutableDictionary *colors = [NSMutableDictionary dictionary];

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

NSSet

- Unordered collection of objects
- Common NSSet methods

```
- setWithObjects:(id)firstObj, ...;   // nil terminated
- (unsigned)count;
- (BOOL)containsObject:(id)object;
```

NSMutableSet

- NSMutableSet subclasses NSSet
- Common NSMutableSet methods

```
- (void)addObject:(id)object;
- (void)removeObject:(id)object;
- (void)removeAllObjects;
- (void)intersectSet:(NSSet *)otherSet;
- (void)minusSet:(NSSet *)otherSet;
```

NSEnumerator

- Consistent way of enumerating over objects in collections
- Use with NSArray, NSDictionary, NSSet, etc.

```
NSEnumerator *e;
id object;

e = [someCollection objectEnumerator];
while ((object = [e nextObject]) != nil) {
    ...
}
```

NSNumber

- In Objective-C, you typically use standard C number types
- NSNumber is used to wrap C number types as objects
- Subclass of NSValue
- No mutable equivalent!
- Common NSNumber methods

```
+ (NSNumber *)numberWithInt:(int)value;
+ (NSNumber *)numberWithDouble:(double)value;
- (int)intValue;
- (double)doubleValue;
```

Other Classes

- NSData / NSMutableData
  - Arbitrary sets of bytes
- NSDate / NSCalendarDate
  - Times and dates
- NSAttributedString
  - Basis of the Cocoa rich text system
  - Attributes are fonts, colors, etc.

Getting some objects

- Until we talk next time:
  - Use class factory methods
    - NSString's +stringWithFormat:
    - NSArray's +array
    - NSDictionary's +dictionary
  - Or any method that returns an object except alloc/init or copy.

More ObjC Info?

- Chapter 3 of Hillegass textbook
- http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC
- Concepts in Objective C are applicable to any other OOP language

Questions?