Announcements

- **On Wednesday we will meet in the Whitaker Mac Lab**
- **Laptop students need to install SDK before class**
  - developer.apple.com
- **Due to the large size of this class we will have two lab sessions**
  - If you are unable to attend the 8:30 AM lab session please email me by tonight

Today's Topics

- **Object Oriented Programming Overview**
- **Objective-C Language**
- **Common Foundation Classes**

Object Basics

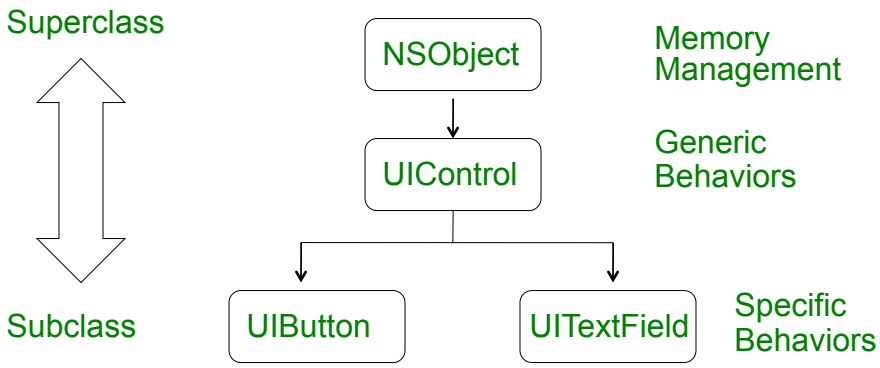
OOP Vocabulary

- **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

More Vocabulary

- **Encapsulation:**
  - keep implementation private and separate from 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**

Objective-C

- **Strict superset of C**
  - -Mix C with ObjC
  - -Or even C++ with ObjC (usually referred to as ObjC++)
- **A very simple language, but some new syntax**
- **Single inheritance, classes inherit from one and only one superclass**
- **Protocols define behavior that cross classes**
  - A protocol is a collection of methods grouped together
    - Indicates that a class implements a protocol
    - Similar to interfaces in Java

Syntax Additions

- **Small number of additions**
- **Some new types**
  - Anonymous object
  - Class
  - Selectors (covered later)
- **Syntax for defining classes**
- **Syntax for message expressions**

OOP with ObjC

Classes and Objects

- **In Objective-C, classes and instances are both objects**
  - Class is the blueprint to create instances
- **Classes declare state and behavior**
- **State (data) is maintained using instance variables**
- **Behavior is implemented using methods**
- **Instance variables typically hidden**
  - Accessible only using getter/setter methods

OOP From ObjC Perspective

- **Everybody has their own spin on OOP**
  - Apple is no different
- **For the spin on OOP from an ObjC perspective:**
  - Read the "Object-Oriented Programming with Objective-C" document:
    - http://developer.apple.com/iphone/library/documentation/ Cocoa/Conceptual/OOP_ObjC

Messaging syntax

Message syntax

- [receiver message]
- [receiver message:argument]
- [receiver message:arg1 andArg:arg2]

Class and Instance Methods

- Instances respond to instance methods

```
-(id)init;
-(float)height;
-(void)walk;
```

- Classes respond to class methods

```
+(id)alloc;
+(id)person;
+(Person *)sharedPerson;
```

Message examples

```
Person *voter; //assume this exists
[voter castBallot];

int theAge = [voter age];
[voter setAge:21];

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

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

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


Method definition examples (.h)

```
Person *voter; //assume this exists
-(void)castBallot;
[voter castBallot];

-(int)age;
int theAge = [voter age];

-(void)setAge: (int)age;
[voter setAge:21];

-(BOOL)canLegallyVote;
if ([voter canLegallyVote]) {
  // do something voter
}
-(void)registerForState: (NSString*) state party: (NSString*)party;
 [voter registerForState:@"CA" party:@"Independant"];

-(Person*)spouse;
-(NSString*)name;
NSString *name = [[voter spouse] name];
```

Addtional Example

- **Method calls**
  - [receiver message:arg1 andArg:arg2]

Consider a Shape class with a method called defineShape

```
-(void) defineShape:(int) sides height:(int) myHeight width:(int) myWidth {
   int tempSides = sides;
   int tempHeight = myHeight;
  //set shape properties….
}


Shape *basicShape;

[basicShape defineShape:4 height:3 width: 2];    //correct call

[basicShape defineShape:4];      //calls something else (or issues warning)
[basicShape defineShape:4:3:2];  //warning no matching signature, fail when running
```


I could create:

```
-(void) defineShape: (int) sides : (int) myHeight : (int) myWidth {

}
```

I could then call:

```
[basicShape defineShape:4:3:2];
```

Considered a bad idea… the parameters are not clearly labeled

Good!

```
-(void) defineShape:(int) sides height:(int) myHeight width:(int) myWidth {
```

Bad!

```
-(void) defineShape: (int) sides : (int) myHeight : (int) myWidth {
```

Terminology

- **Message expression**

  [receiver method: argument]

- **Message**

  [receiver method: argument]

- **Method**
  - The code selected by a message

Dot Syntax

- **Objective-C 2.0 introduced dot syntax**
- **Convenient shorthand for invoking accessor methods**

```
float height = [person height];
float height = person.height;

[person setHeight:newHeight];
person.height = newHeight;
```

- **Follows the dots...**

```
[[person child] setHeight:newHeight];
// exactly the same as
person.child.height = newHeight;
```

Objective-C Types

Dynamic and Static typing

- **Dynamically-typed object**
  id anObject
  - Just id
  - Not id * (unless you really, really mean it...)
- **Statically-typed object**
  Person *anObject
- **Objective-C provides compile-time, not runtime, type checking**
- **Objective-C always uses dynamic binding**
  - determining the exact implementation of a request based on both the request (operation) name and the receiving object at the run-time

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;
[person name];
```


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;
```

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
}
```


Working with Objects

Identity versus Equality

- Identity—testing equality of the pointer values

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

- Equality—testing object attributes

```
if ([object1 isEqual: object2]) {
NSLog(@"Logically equivalent, but may be different
   object instances");
}
```


-description

- NSObject implements –description
  -(NSString *)description;
- Objects represented in format strings using %@
- When 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
- Similar to a toString() in Java

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 Touch instead of "char *"**
- **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 *aString = @"Hello World!";
```


Format Strings

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

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

- **log would be set to**
  - It's 'Johnny'
- **Also used for logging**

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

- **would log something like:**
  - I am a NSArray, I have 5 items

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!"];
```

- **fullString would be set to**
  - Hello 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

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

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

```
NSMutableString *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");
}
```


NSMutableArray

- **NSMutableArray subclasses NSArray**
  - So, everything in NSArray
- **Common NSMutableArray Methods**

```
+ (NSMutableArray *)array;
- (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:@"Red", @"Color 1",
   @"Green", @"Color 2", @"Blue", @"Color 3", nil];

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

if ([colors objectForKey:@"Color 8"]) {
  // won't make it here
}
```


NSMutableDictionary

- **NSMutableDictionary subclasses NSDictionary**
- **Common NSMutableDictionary methods**

```
+ (NSMutableDictionary *)dictionary;
- (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 distinct objects
- Common NSSet methods

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


NSMutableSet

- NSMutableSet subclasses NSSet
- Common NSMutableSet methods

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

Enumeration

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

```
NSArray *array = ... ; // assume an array of People objects

// old school
 Person *person;
int count = [array count];
for (i = 0; i < count; i++) {
  person = [array objectAtIndex:i];
  NSLog([person description]);
}

// new school
for (Person *person in array) {
  NSLog([person description]);
}
```


Other Classes

- **NSData / NSMutableData**
  - Arbitrary sets of bytes
- **NSDate / NSCalendarDate**
  - Times and dates

More ObjC Info?

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