OBJECTIVE-C

All the weird syntax and stuff

PIAZZA

- Join our Piazza page at **git.to/195piazza**

C

- Objective-C is a strict superset of C
  - Everything that is true in C is also true in Objective-C
  - Some of the iOS API uses pure C as opposed to Objective-C. That means you'll be calling C functions and using/creating structs.
  - You can use C code anywhere in Objective-C code, since any Objective-C compiler can compile C code as well.

POINTERS

- Instance references are pointers.
  - Don't think *too* much about this, since pointers are used consistently throughout Objective-C.
    - A message sent to an instance is directed to the pointer, so no need to worry about dereferencing.
- Example:

```
NSString* s = @"This is a string of length 29";
int length = [s length];
```

- FYI: "NS" prefix stands for the NeXTSTEP operating system, from which OS X and iOS were derived.

NIL

- Let's say we have:

```
NSString* s;
```

  - What is the value of `s` at this moment?

- Examples:

```
if (nil == s) // do something
```

- What happens when you send a message to nil?

```
NSString* s;
NSString* upper = [s uppercaseString];
```

METHOD NAMING

- You'll find that Objective-C methods tend to be very verbose.

- NSString:
  - stringByAppendingString:

- UIColor:
  + colorWithHue:saturation:brightness:alpha:

- These are the just the method names.

METHOD DECLARATION

- Example from UIColor and NSString:

```
+ (UIColor*) colorWithRed: (CGFloat) red green: (CGFloat) green
                     blue: (CGFloat) blue alpha: (CGFloat) alpha

- (unichar) characterAtIndex:(NSUInteger)index
```

- `+` signifies class methods
- `-` signifies instance methods

PARAMETER LISTS

- Example from NSArray:

```
+ (id)arrayWithObjects:(id)firstObj, ...
```

- Usage:

```
NSArray* teas = [NSArray arrayWithObjects: @“earl grey”,
@“prince of wales”, @“genmai-cha”, nil];
```

- iOS Developer Library: **git.to/ios**

ID

- You can typecast anything to `id`.
- You can typecast `id` to anything.
- It can legally receive any message.
  - That doesn't mean your program won't crash.
  - Unless `id` happens to be `nil`.
- Example:
```
+ (id)arrayWithObjects:(id)firstObj, ...
```
- Be aware.

CLASSES

- No method overloading for methods of the same type (class vs. instance).
- Global namespace
  - Problems?
- 2 parts:
  - Interface
  - Implementation

PARTS OF A CLASS

- *@interface*
  - global method declarations
- *@implementation*
  - instance variables
  - method implementations

FORMAT

- Example:

MyClass.h

```
#import “MySuperClass.h”
#import “MyProtocol.h”

@interface MyClass : MySuperClass <MyProtocol>
- (NSString*) sayHello;
@end
```

MyClass.m

```
#import “MyClass.h”

@implementation {
  // instance variables go here.
}
- (NSString*) sayHello {
  return @“Hello!”
}

@end
```

@CLASS VS. #IMPORT

- Use *@class* when you just need to mention a class in the header. For example, if OtherClass is a return type of one of the public methods. It only tells the compiler that this is a valid class.

- Use *#import* when you are subclassing OtherClass, or other situations where you need to know more about a class, like its members.

NSOBJECT

- Like "Object" in Java.
- Must be declared explicitly, unlike in Java, where Object is implied as a superclass if there is none specified.
  - This is because Objective-C allows for different classes to be roots of different object hierarchies.
- Example:

```
@interface MyClass : NSObject
```

ALLOC, INIT

- *alloc* sets aside memory for the instance

- *init* actually initializes the new instance.

- Example:

```
NSArray* array = [[NSArray alloc] initWithObjects: @“pirate”,
@“ninja”, nil];
```

SELF

- Like "this" in Java.
- Refers to what the instance really is, not the class where *self* is mentioned.

SUPER

- Class-based (not instance based).

ACCESSORS

- In Java, they would be called something like "getName" and "setName". In Objective-C, they would be "name" and "setName" by convention.

- Example:

```
[person setName: @"Alfred"];
NSString* name = [person name];
```

PROPERTIES

- Syntactic sugar for accessors.
- Example:

```
person.name = @“Alfred”;
NSString* name = person.name;
```

- Why is this nice?
- Note: Properties do *not* allow access to instance variables directly.

INITIALIZER

- Example:

```
- (id) initWithName: (NSString*) name {
  self = [super init];
  if (self) {
    self.name = name;
  }
  return self;
}

Person* person = [[Person alloc] initWithName: @“Alfred”];
```