Introduction to OS X development with Cocoa

…and how web developers can cheat

George Brocklehurst
http://georgebrock.com
@georgebrock on Twitter

Before we start…

- Have you done any…
  - HTML?
  - Javascript?
  - PHP?
  - Any object oriented programming?
  - C?

What is Cocoa?

- Framework for Mac OS X and iPhone
- Heavily MVC focused
- Usually written in Objective-C
- Can also be written in Python or Ruby

Getting Started

- You will need: Apple's Xcode Tools
- Available from:
  - http://developer.apple.com/technology/xcode.html
  - Your OS X install DVDs

Objective C Syntax: Classes

- Declaring a class interface (.h file)

```
@interface MyClass : NSObject
{
  int anInteger;
}
- (void)doStuff;
@end
```

Objective C Syntax: Classes

- Declaring a class interface (.h file)

```
@interface MyClass : NSObject
{
    NSInteger;
}
- (void)doStuff;
@end
```

Interface only here!

Objective C Syntax: Classes

- Declaring a class interface (.h file)

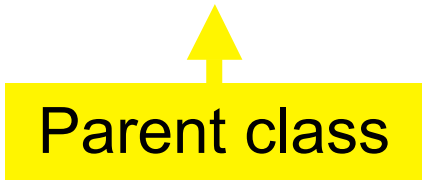
```
@interface MyClass : NSObject
{
  int anI
}
- (void)doStuff;
@end
```

Class name

Objective C Syntax: Classes

- Declaring a class interface (.h file)

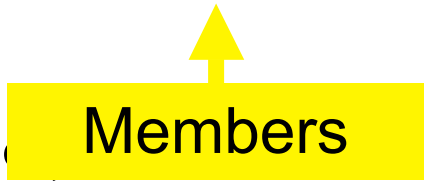
```
@interface MyClass : NSObject
{
  int anInteger;
}
- (void)doStuff;
@end
```



Objective C Syntax: Classes

- Declaring a class interface (.h file)

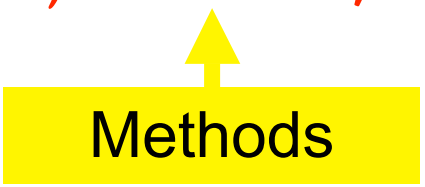
```
@interface MyClass : NSObject
{
  int anInteger;
}
- (voi
@end
```



Objective C Syntax: Classes

- Declaring a class interface (.h file)

```
@interface MyClass : NSObject
{
  int anInteger;
}
- (void)doStuff;
@end
```



Objective C Syntax: Classes

- Implementing a class (.m file)

```
#import “MyClass.h”
@implementation MyClass
- (void)doStuff
{
  // Do stuff here
}
@end
```

Objective C Syntax: Classes

- Implementing a class (.m file)

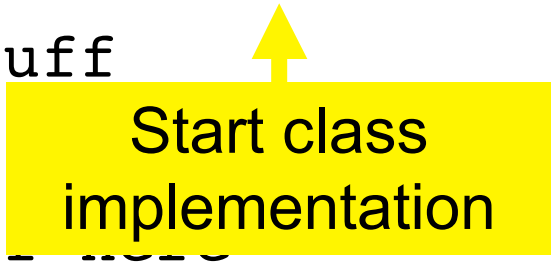
```
#import "MyClass.h"
@implementation MyClass
- (void)doStuff
{
  // Do stuff here
}
@end
```

Interface

Objective C Syntax: Classes

- Implementing a class (.m file)

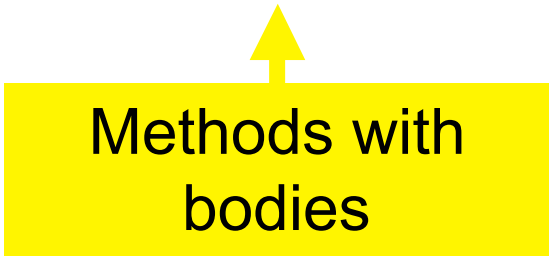
```
#import “MyClass.h”
@implementation MyClass
- (void)doStuff
{
  // Do stuff here
}
@end
```



Objective C Syntax: Classes

- Implementing a class (.m file)

```
#import “MyClass.h”
@implementation MyClass
- (void)doStuff
{
  // Do stuff here
}
@end
```



Objective C Syntax: Methods

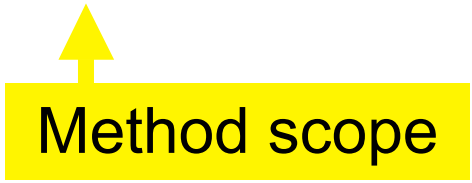
```
-(void)myMethod:(int)arg
```

In other languages, might be:

```
function myMethod(arg)
```

Objective C Syntax: Methods

```
-(void)myMethod:(int)arg
```



Can be either:

- \+ for a class method
- \- for an instance method

Objective C Syntax: Methods

```
-(void)myMethod:(int)arg
```

Can be any valid data type, including:

- `void` returns nothing
- `id` a pointer to an object of any class
- `NSString*` a pointer to an NSString
- `BOOL` a boolean (`TRUE`, `FALSE`, `YES` or `NO`)

Objective C Syntax: Methods

```
-(void)myMethod:(int)arg
```

Method name

Colons precede arguments, but are considered part of the method name

Objective C Syntax: Methods

```
-(void)myMethod:(int)arg
```

Come after or within the method name

For multiple arguments:

```
-(void)myMethod:(int)arg andAlso:(int)arg2
```

(Method name is “myMethod:andAlso:”)

Objective C Syntax: Methods

```
-(void)myMethod:(int)arg1
     andAlso:(int)arg2;
```

How to call this method:

```
[myObject myMethod:10 andAlso:20];
```

In other languages this might be:

```
myObject->myMethod(10, 20); //or
myObject.myMethod(10, 20);
```

Objective C Syntax: Properties

- New short hand in Objective-C 2.0 (Xcode 3 / Leopard)
- Access class members without writing getters and setters
- Convenient, but nasty difficult to remember syntax

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

In the class implementation:

```
@synthesize propertyName;
```

To access:

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

Storage method:
assign, **retain**
or **copy**

In the cl tion:

```
@synthesize propertyName;
```

To access:

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

Access permissions: **readwrite** or **readonly**

In the class implem

```
@synthesize pr
```

To access:

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

In the class implementatio

```
@synthesize propert
```

Can add other things: **nonatomic**, **getter=…** and **setter=…**

To access:

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

Property variable declaration, must also be declared as a class member

In the cla…on:

```
@synthe…tyName;
```

To acces…

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

In the class implementation:

```
@synthesize propertyName;
```

Tell Objective-C precompiler to make getter & setter

```
propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

In the class implementation:

```
@synthesize propertyName;
```

To access:

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Properties

In the class interface (with methods):

```
@property(copy, readwrite) NSString
  *propertyName;
```

In the class implementation:

```
@synthesize propertyName;
```

Sensible syntax!
Yay!

To ac

```
myInstance.propertyName = @"Foo";
```

Objective C Syntax: Selectors

```
SEL callback = NULL;
callback =
  @selector(myMethod:andAlso:);
```

- Like function pointers
- Useful for callback type behavior

Objective C Syntax: Selectors

```
SEL callback = NULL;
callback =
                Method:andAlso:);
```

Data type for selectors

- Like function pointers
- Useful for callback type behavior

Objective C Syntax: Selectors

```
SEL callback = NULL;
callback =
  @selector(myMethod:andAlso:);
```

Macro to create selector

- L nters
- Useful for callback type behavior

Objective C Syntax: Selectors

```
SEL callback = NULL;
callback =
  @selector(myMethod:andAlso:);
```

- Like function pointers
- Useful for callback type behavior

Objective C Syntax: Strings

```
NSString *myString;
myString = @"This is a string";
```

- Note the `@` prefix
- Tells the Objective-C precompiler to instantiate an NSString instead of just creating a C string (aka. A nasty character array)

Managing Memory

- Easier than vanilla C, honest!
- Reference counting
- Increment with `retain`, decrement with `release`
- Rule of thumb: `release` anything you `alloc`, `copy` or `retain`

Managing Memory: Example

```
NSString *str;
str = [NSString alloc];    //1
[str retain];              //2
[str release];             //1
[str release], str = nil; //0
```

Managing Memory: Autorelease

```
[str autorelease];
```

This will automatically release an object when it's finished with

Another rule of thumb:

```
[NSThing thingWith:…]
```

returns an autoreleased instance

```
[[NSThing alloc] initWith:…]
```

needs manually releasing

Wasn’t there something about cheating?!

- WebKit is part of Cocoa
- Can build your UI with HTML, CSS and Javascript and only use Objective-C when you really need to

Demo

I Can Haz Questions?

K, Thx, Bai!

George Brocklehurst
http://georgebrock.com
@georgebrock on Twitter