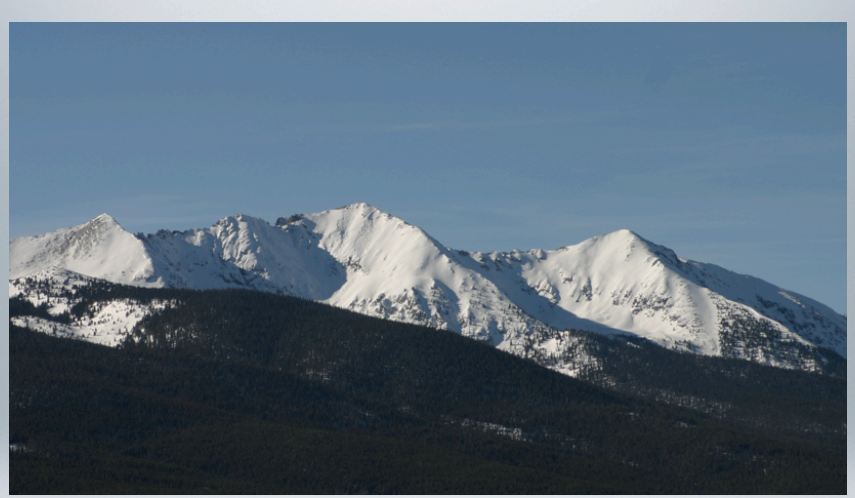
iPhone DEVELOPMENT

Getting Started with the iPhone SDK

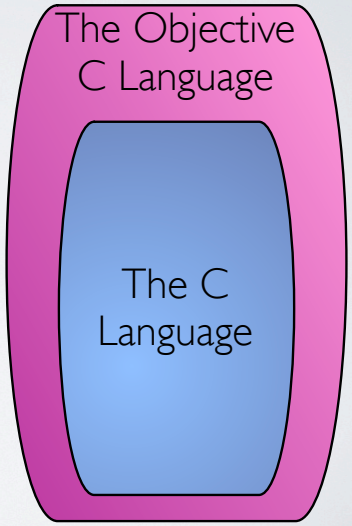


OBJECTIVE-C

The Big Picture

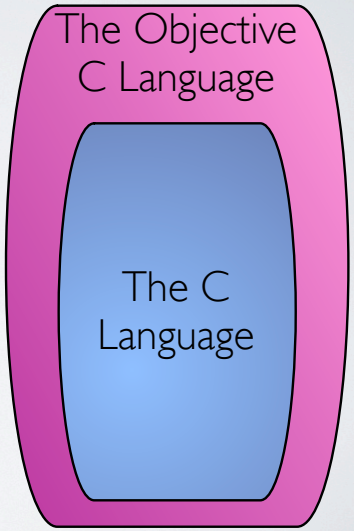
STRICT SUPERSET OF C

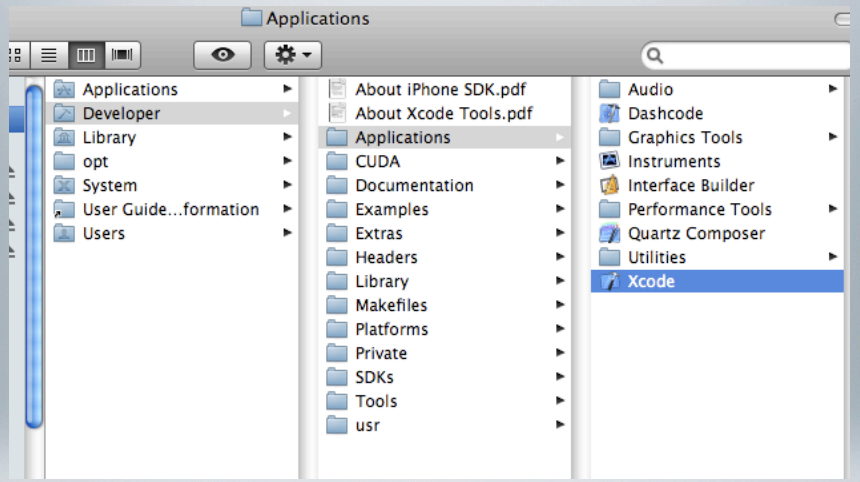
- Any C stuff applies
- Standard libs are here (time, sqrt etc)



SMALL SET OF EXTENSIONS

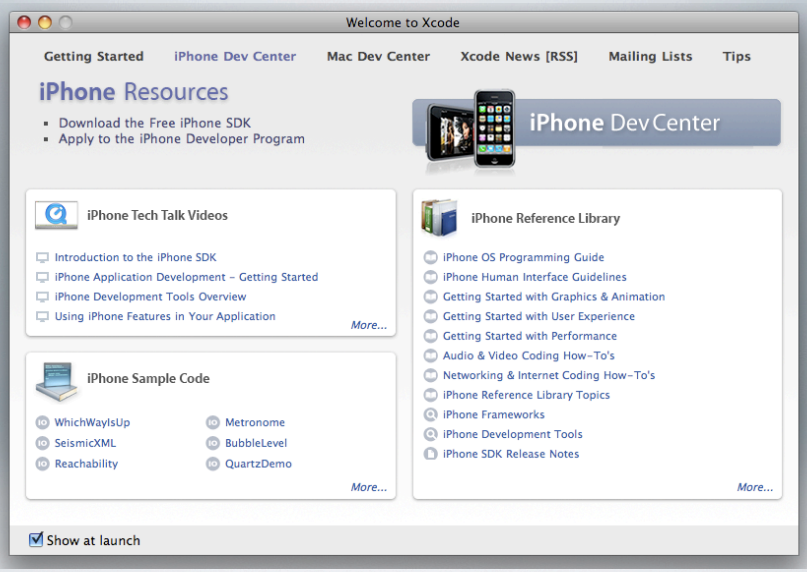
- Compiler Directives
  - @class, @property etc.
- Message Sending
  - [receiver message]



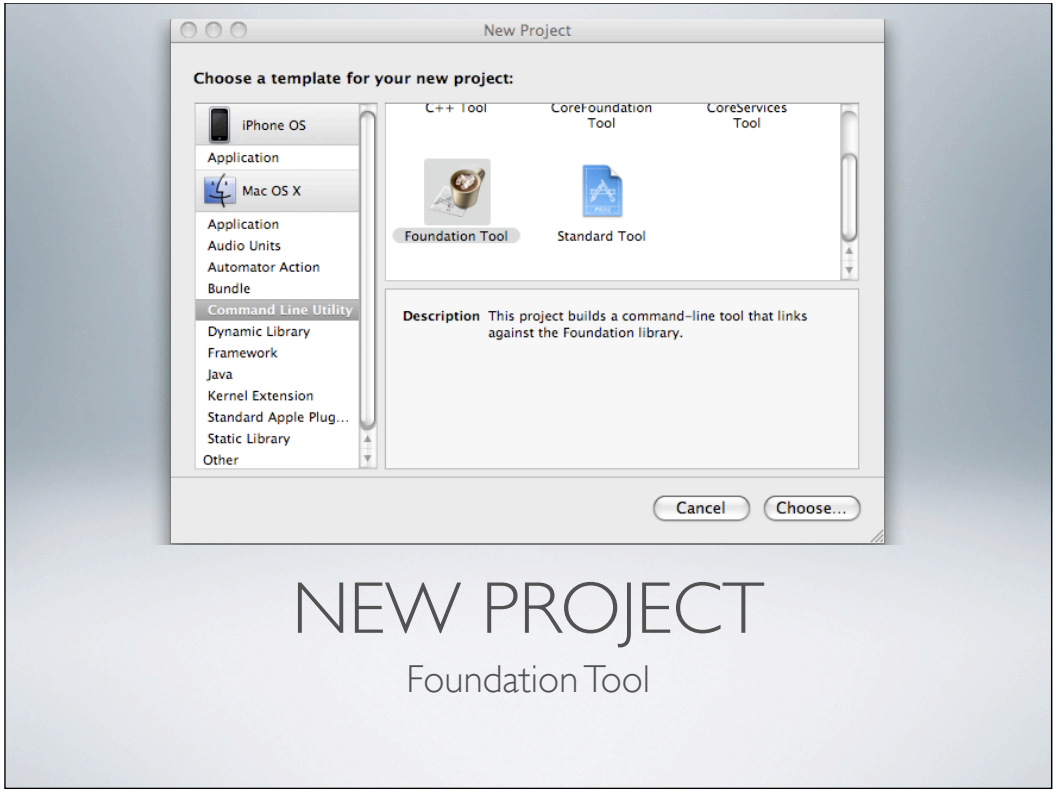


XCODE

Launch when we Live Code

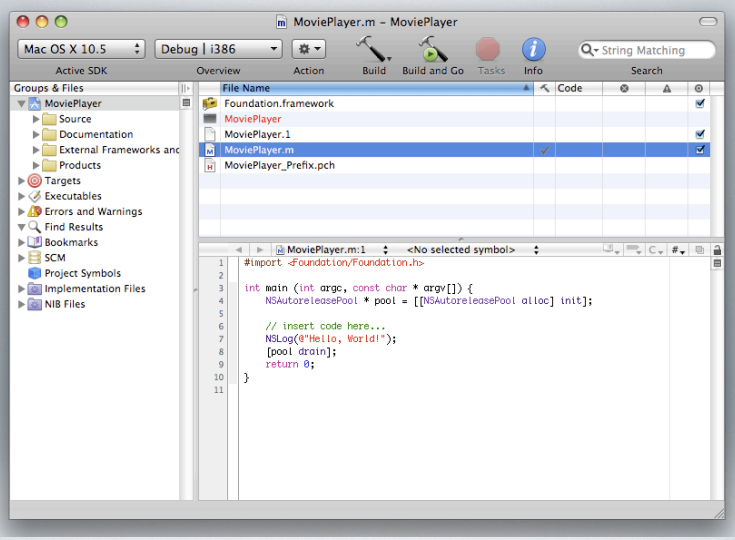


WELCOME TO XCODE



NEW PROJECT

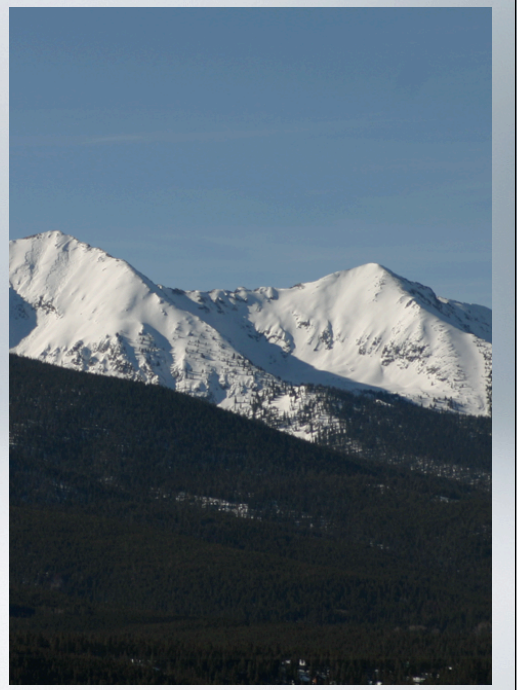
Foundation Tool

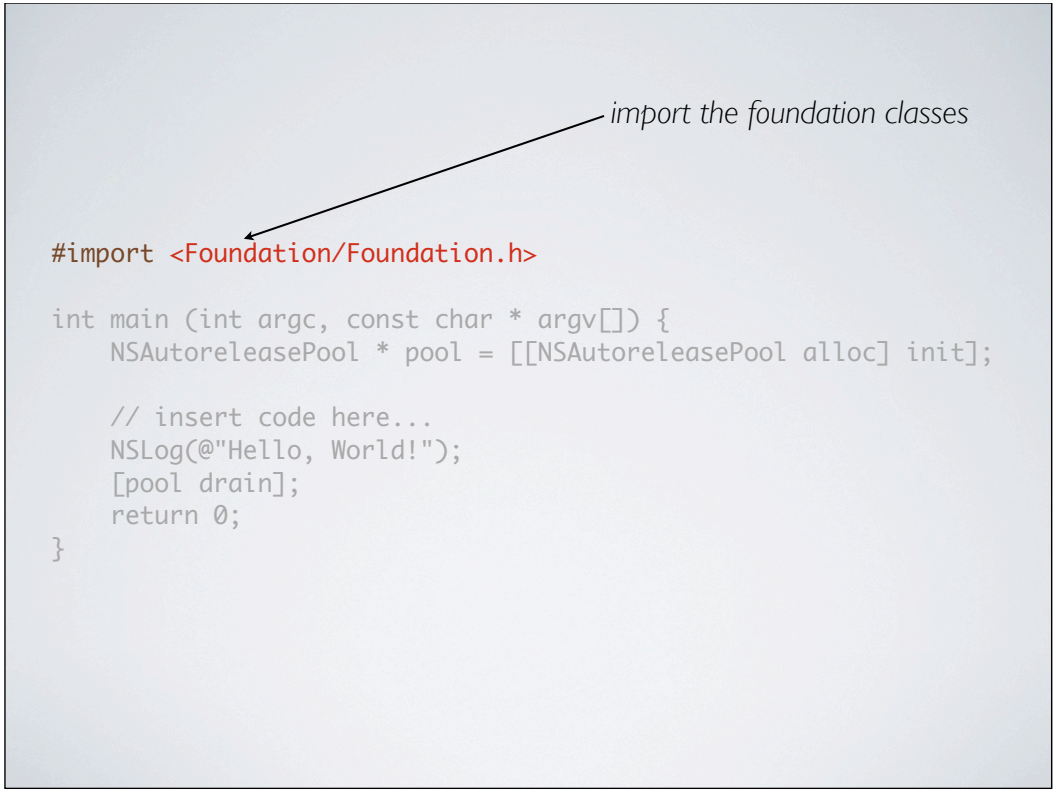

MoviePlayer.m – MoviePlayer
Mac OS X 10.5
Debug | i386
Active SDK
Overview
Action
Build
Build and Go
Tasks
Info
String Matching
Search
Groups & Files
MoviePlayer
Source
Documentation
External Frameworks and
Products
Targets
Executables
Errors and Warnings
Find Results
Bookmarks
SCM
Project Symbols
Implementation Files
NIB Files
File Name
Code
Foundation.framework
MoviePlayer
MoviePlayer.1
MoviePlayer.m
MoviePlayer_Prefix.pch
MoviePlayer.m:1
<No selected symbol>
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]) {
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
// insert code here...
NSLog(@"Hello, World!");
[pool drain];
return 0;
}

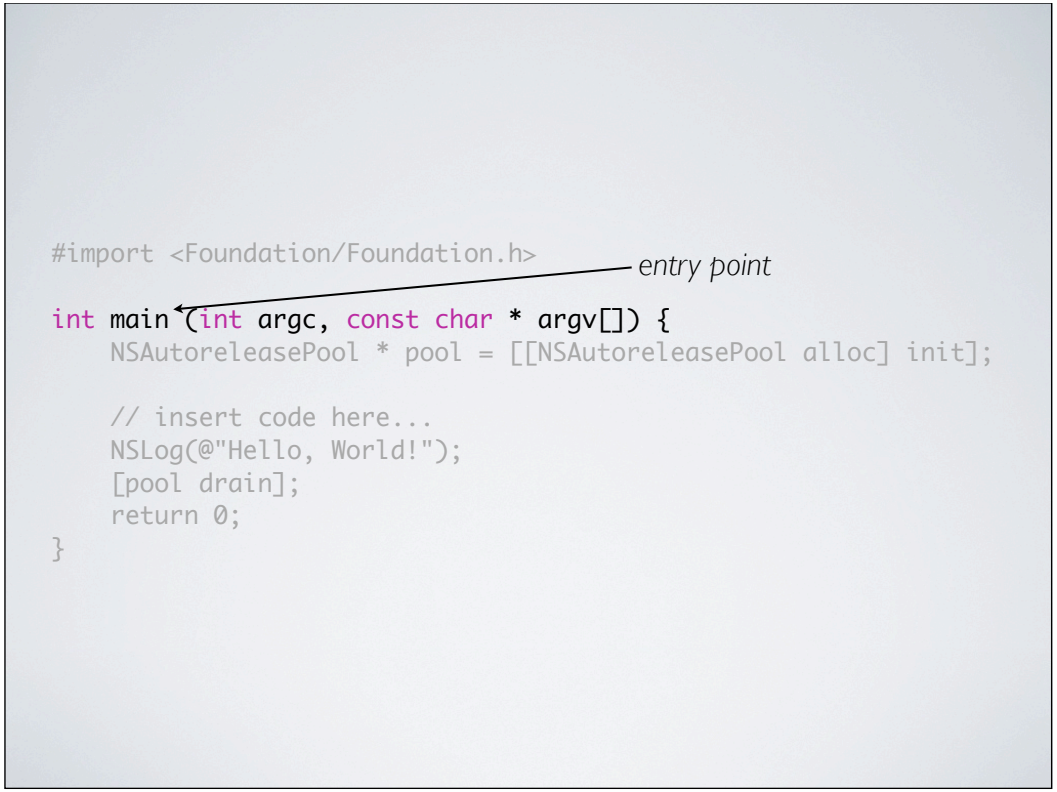
HELLO WORLD!

HELLO WORLD

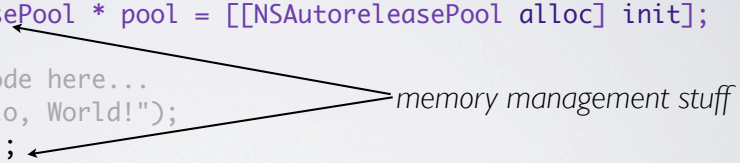
The Ubiquitous 'First Program'




import the foundation classes
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]) {
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
// insert code here...
NSLog(@"Hello, World!");
[pool drain];
return 0;
}


#import <Foundation/Foundation.h>
entry point
int main (int argc, const char * argv[]) {
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
// insert code here...
NSLog(@"Hello, World!");
[pool drain];
return 0;
}

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

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // insert code here...
    NSLog(@"Hello, World!");
    [pool drain];
    return 0;
}
```



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

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // insert code here...
    NSLog(@"Hello, World!");
    [pool drain];
    return 0;
}
```

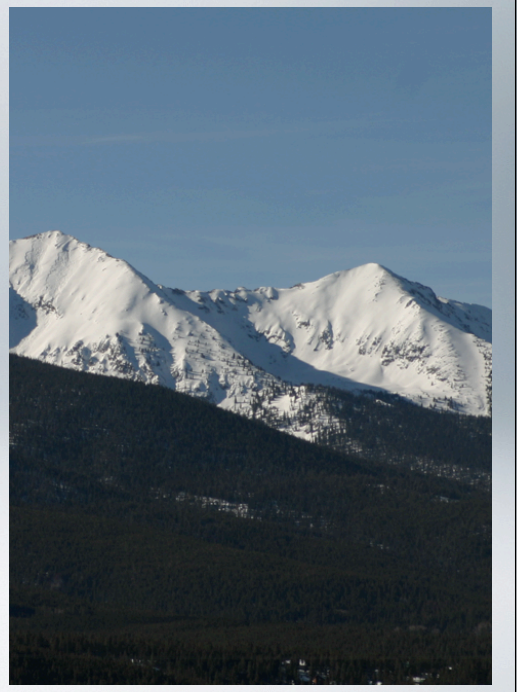

single line comment
ObjC also supports multi-line comments with / and */ denoting the beginning and end respectively*

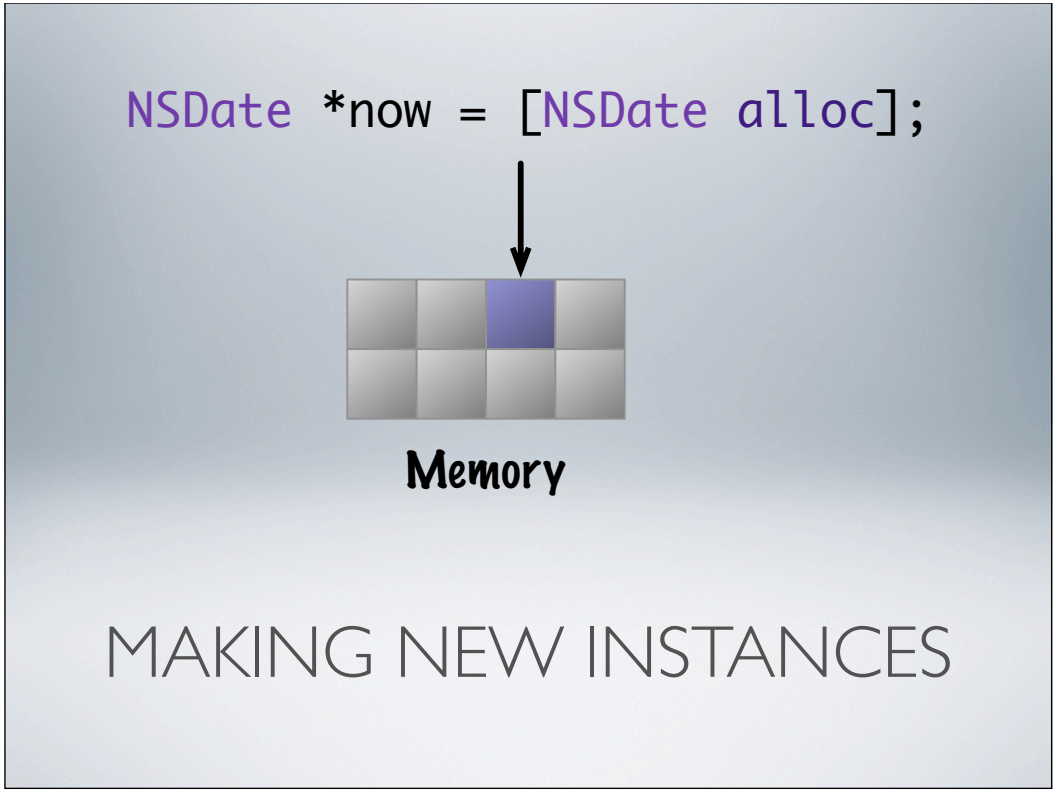
```objc
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // insert code here...
    NSLog(@"Hello, World!");
    [pool drain];
    return 0;
}
```

Objective-C String Object
like C strings but the '@' creates an object instead of an array of C characters

USING

Sending Messages to Existing Classes





alloc is a class method

inherited from NSObject

grabs memory

sets up objects class

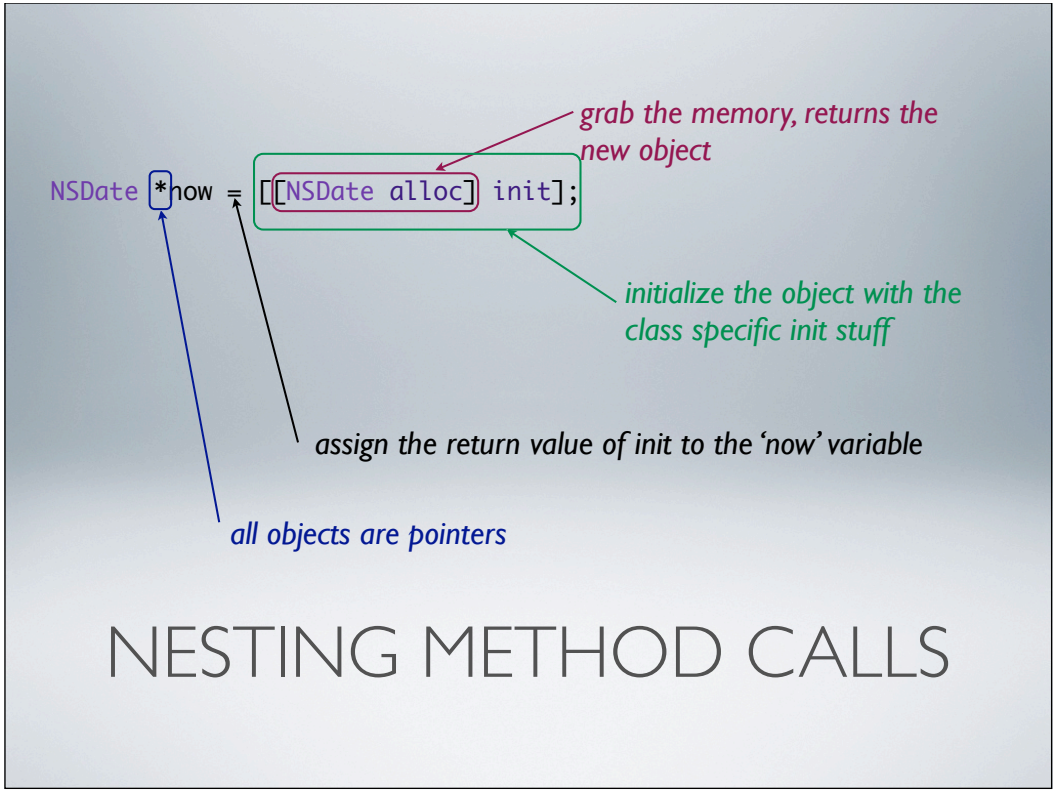
NSDate in this case

zero initializes the rest

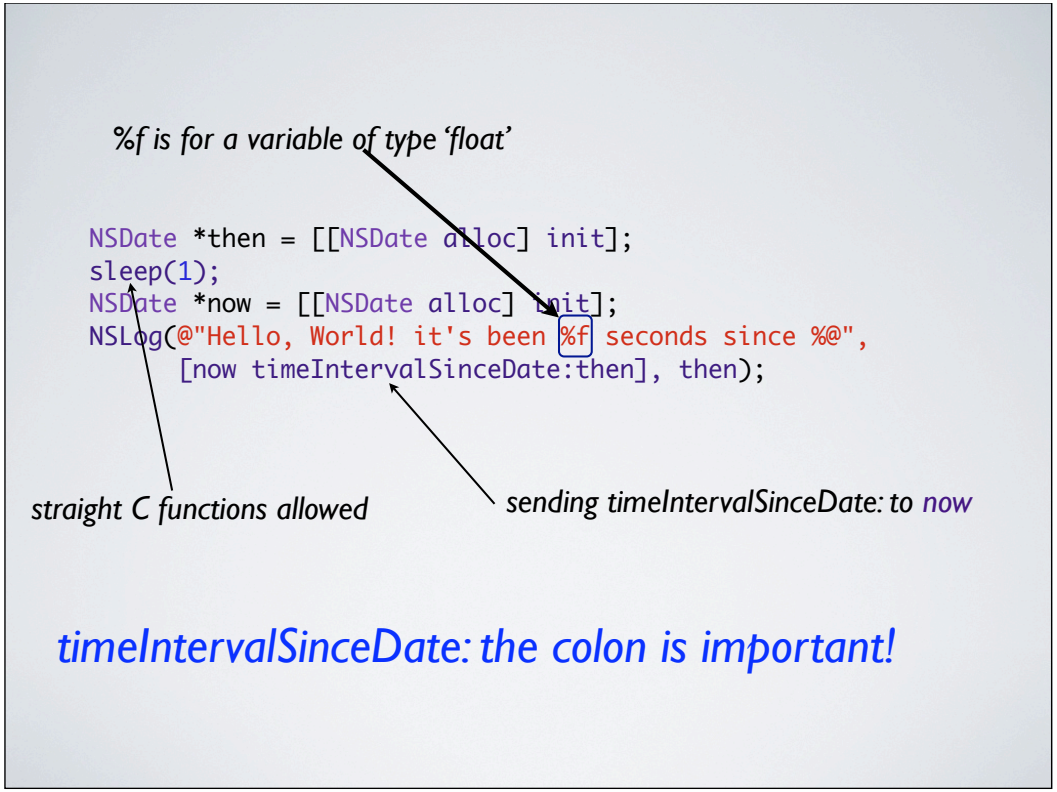
Must call class specific initialize to get a real object

```
now = [now init];
```

INITIALIZE INSTANCES



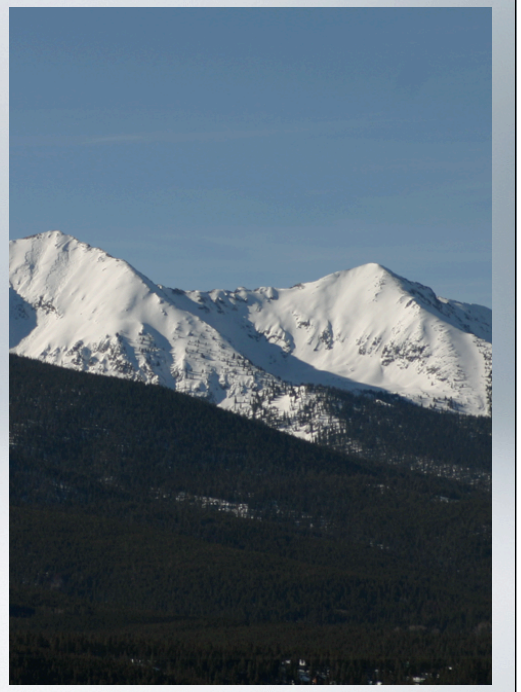
NESTING METHOD CALLS

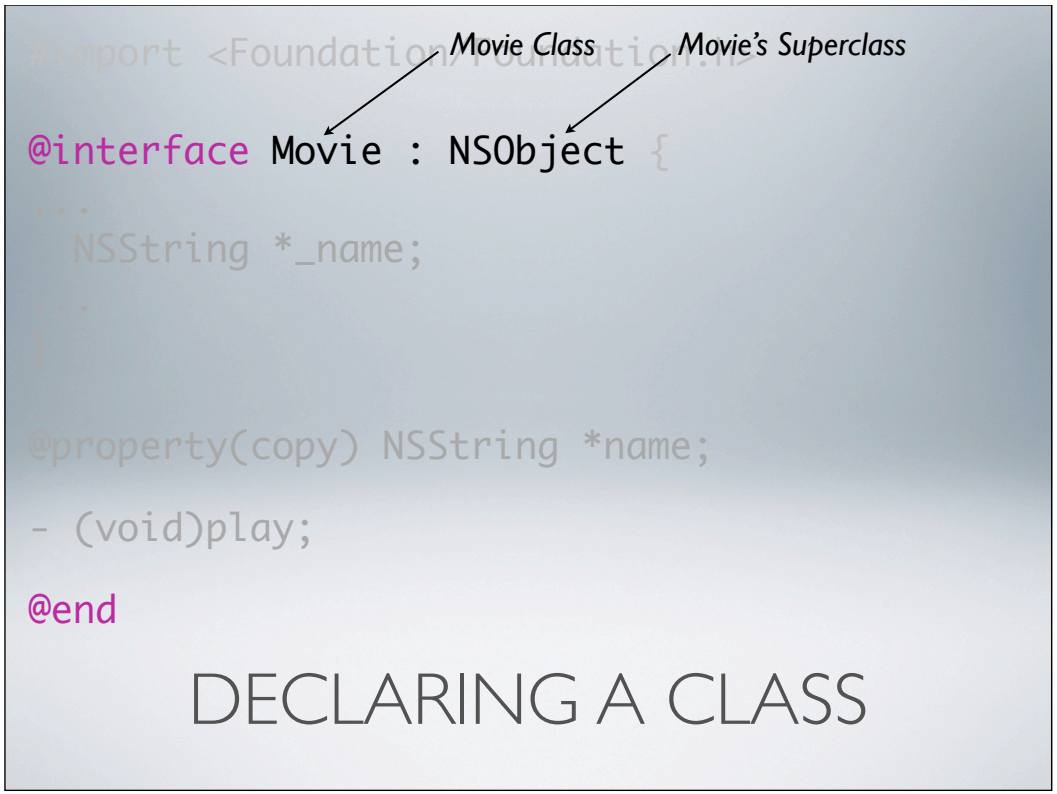


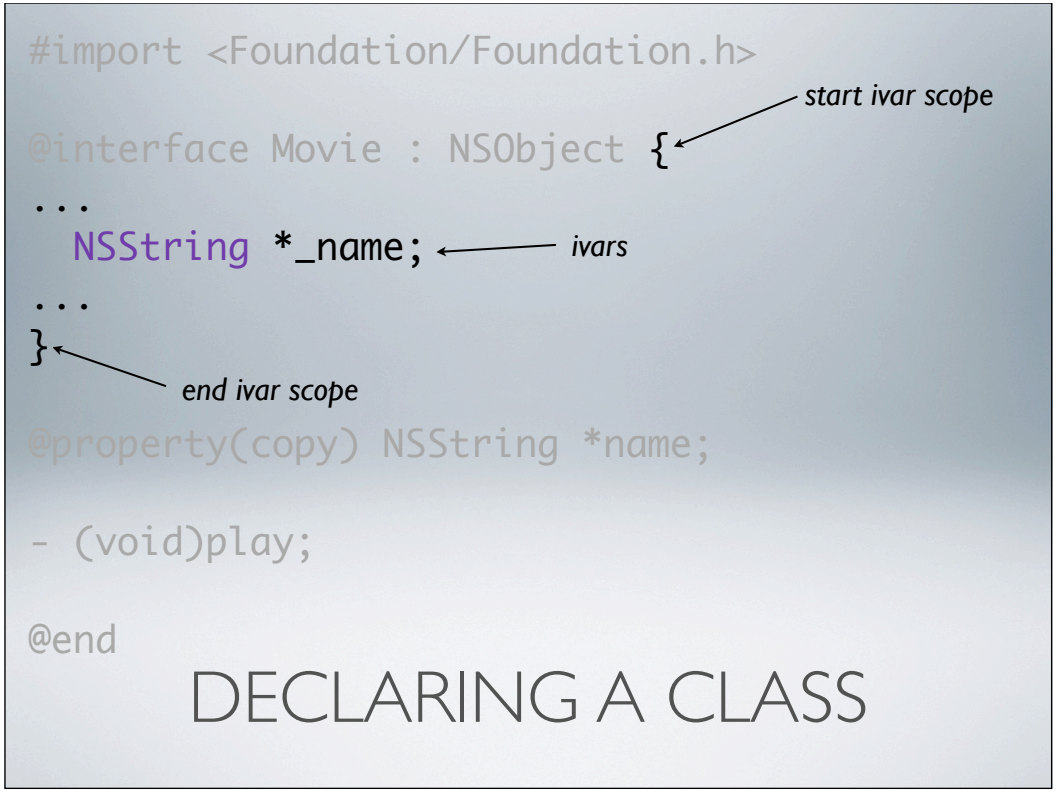
timeIntervalSinceDate: the colon is important!

CREATING

Making New Classes to Model your Space

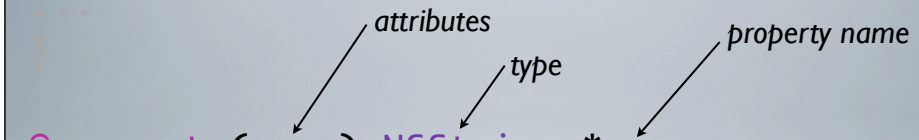



Movie Class
Movie's Superclass
#import <Foundation/Foundation.h>
@interface Movie : NSObject {
...
NSString *_name;
...
}
@property(copy) NSString *name;
- (void)play;
@end
DECLARING A CLASS



DECLARING A CLASS

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

@interface Movie : NSObject {
...
  NSString *_name;
...
}

@property(copy) NSString *name;

- (void)play;

@end
```



PROPERTIES

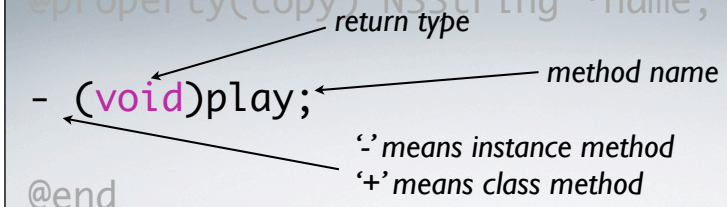
```
#import <Foundation/Foundation.h>

@interface Movie : NSObject {
...
  NSString *_name;
...
}

@property(copy) NSString *name;

- (void)play;

@end
```

to link must share the same type

naming convention is to name ivar with an '_' in front of property name

PROPERTIES

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

@interface Movie : NSObject {
...
  NSString *_name;
...
}

@property(copy) NSString *name;

- (void)play;

@end
```



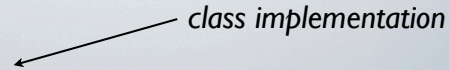
METHODS

```
#import "Movie.h"

@implementation Movie

@synthesize name = _name;

- (void)play {
  NSLog(@"playing %@", self.name);
}

@end
```

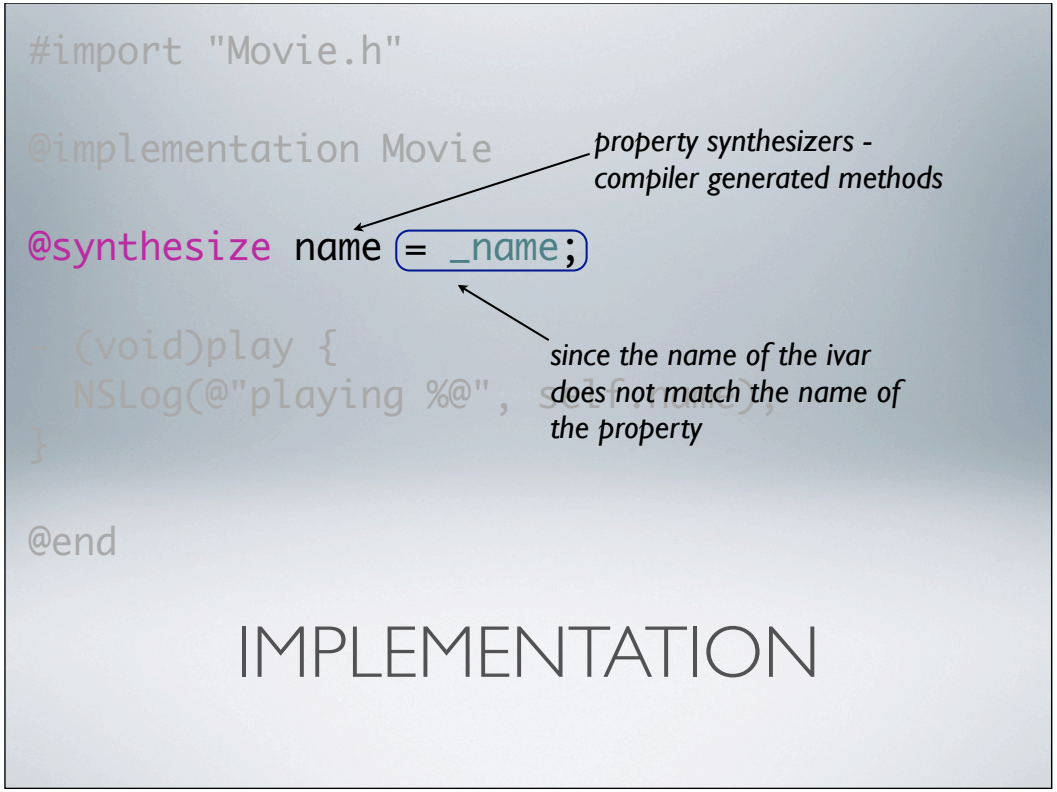
import the header

IMPLEMENTATION

```
#import "Movie.h"

@implementation Movie

@synthesize name = _name;

- (void)play {
  NSLog(@"playing %@", self.name);
}

@end
```



IMPLEMENTATION



IMPLEMENTATION

```
#import "Movie.h"

@implementation Movie

@synthesize name = _name;

- (void)play {
  NSLog(@"playing %@", self.name);
}

@end
```

method implementation, any valid C or ObjC can go here

recall - this is a format string, %@ is replaced by invoking the description method on the argument

IMPLEMENTATION

```objc
#import <Foundation/Foundation.h>
#import "Movie.h"

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Movie *movie = [[Movie alloc] init];
    movie.name = @"Iron Man";
    [movie play];
    [movie release];
    [pool drain];
    return 0;
}
```

import the header

create a movie

set the name

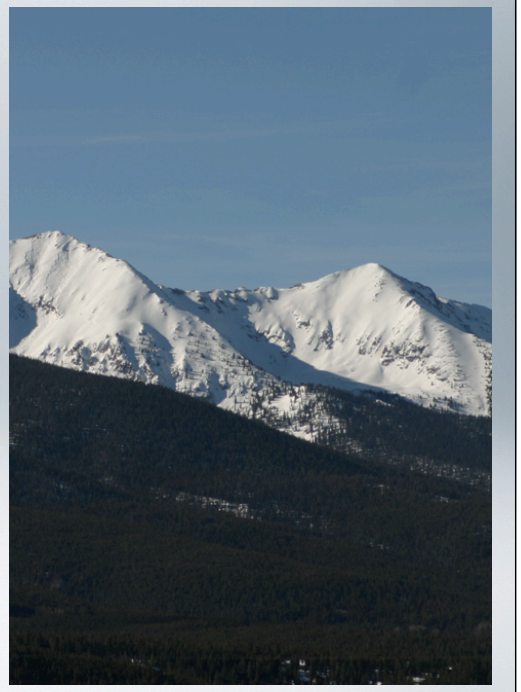
play the movie

clean up the memory

USE THE CLASS

MEMORY MANAGEMENT

Reference Counting Objects



OBJECT OWNERSHIP

- If you own an object you must release it when you are done
- If you don't own an object, don't release it

3 SIMPLE RULES

you 'Own' an Object when

- You create it with 'alloc' or 'new'
- You copy another object
- You explicitly retain it

the system automatically frees the object when its retain count goes to zero

```
#import <Foundation/Foundation.h>
#import "Movie.h"

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Movie *movie = [[Movie alloc] init];
    movie.name = @"iron man";
    [movie play];
    [movie release];
    [pool drain];
    return 0;
}
```

we called 'alloc' so we own the object

must call release when we are done with it

FOR EXAMPLE

```
- (void) dealloc {
  NSLog(@"in dealloc");
  [self setName:nil];
  [super dealloc];
}
```

since setName: releases the old object this does both set the ivar to nil and release the old object

self.property = nil; would be equivalent
This syntax is an idiom, not required you can instead just release _name

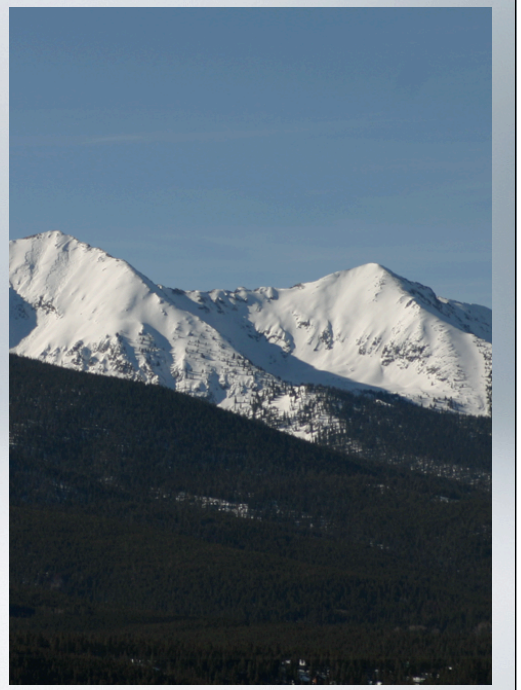
Called by the system when the ref-count goes to zero

Never *call dealloc on an object, instead call release, let the system deal with freeing the memory*

RELEASING OWNED OBJECTS

AUTORELEASE

Release Objects Later



no copy or new in this method name so caller does not know to release it

```
- (NSString *)description {
  return [[NSString alloc] initWithFormat:
              @"Movie: %@", self.name];
}
```

we own it because of the alloc, but we don't release

MEMORY LEAK

AUTORELEASE POOLS

- Autorelease pools do not retain the objects they contain
- An autorelease pool is a place to put objects that you want to release but can't for whatever reason release them now
- Objects go into an autorelease pool via the autorelease method
- Objects are removed and released when the pool is drained

```
- (NSString *)description {
  NSString *desc = [[NSString alloc]
     initWithFormat:@"Movie: %@", self.name];
  [desc autorelease];
  return desc;
}
```

put the object in the autorelease pool so it will get released, later

returning a valid object here

AUTORELEASE

iPhone APP

Enough Theory, Let's Build!

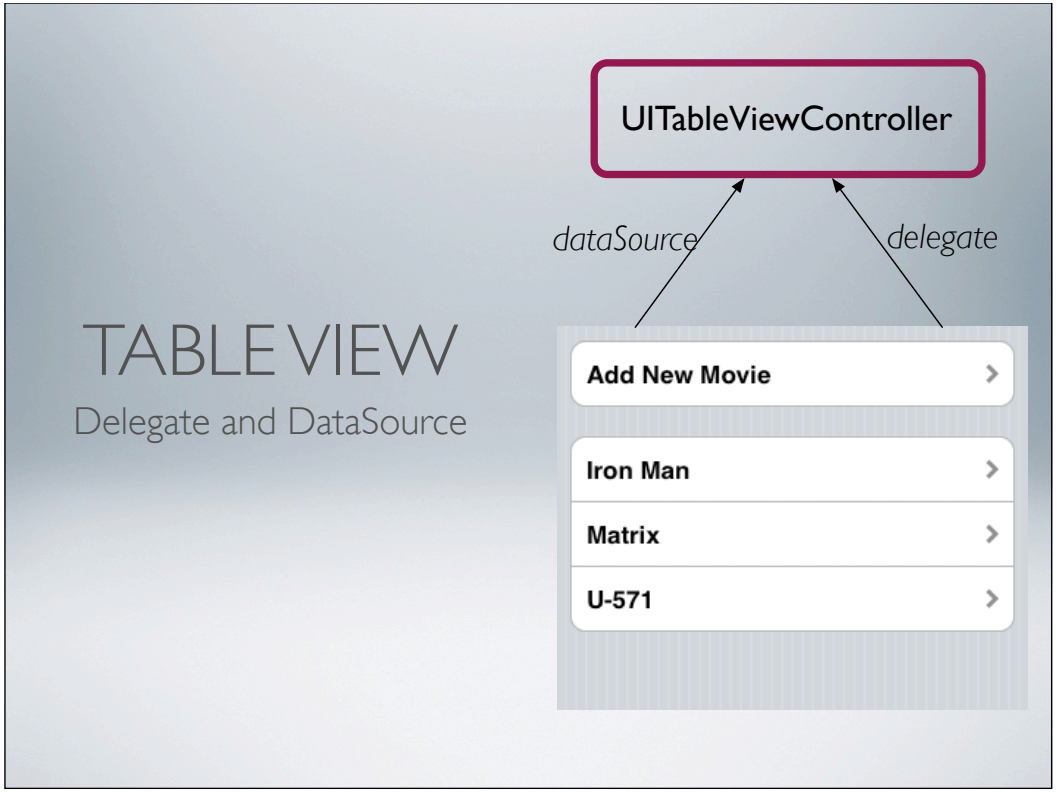
DEMO APP

In the next few minutes we are going to build this app

UINavigationController

UITableViewController

NAVIGATION CONTROLLER



dataSource provides how many sections and how many rows
delegate responds to events like ‘this row was clicked on’

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 2;
}
```

HOW MANY SECTIONS

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section {
  NSInteger count = 1;
  if(1 == section) {
    count = self.movies.count;
  }
  return count;
}
```

HOW MANY ROWS

```
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath {

  static NSString *CellIdentifier = @"Cell";

  UITableViewCell *cell =
     [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
    cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
                                   reuseIdentifier:CellIdentifier]
            autorelease];
  }

  if(indexPath.section == 0) {
    cell.text = @"Add New Movie";
  } else {
    cell.text = [[self.movies objectAtIndex:indexPath.row] name];
  }

  return cell;
}
```

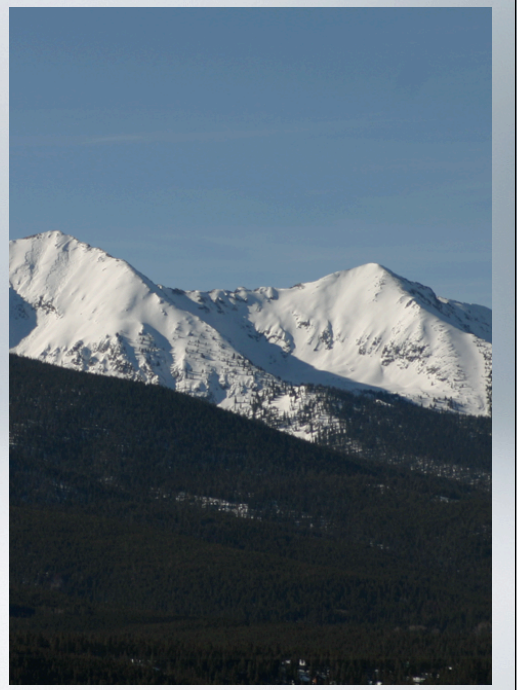
CONFIGURE THE CELLS

dequeue - muy importante!

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
  Movie *movie = [self.movies objectAtIndex:indexPath.row];
  if(0 == indexPath.section) {
    movie = [[[Movie alloc] initWithName:@"New Movie"]
                            autorelease];
    [self.movies addObject:movie];
  }
  self.movieDetailViewController.movie = movie;
  [self.navigationController
   pushViewController:self.movieDetailViewController
             animated:YES];
}
```

NAVIGATION

LIVE CODE



Now we have seen the basics of ObjC coding, we are using

RESOURCES

- http://pragprog.com/titles/amiphd
- http://bill.dudney.net/roller/objc
- http://pragmatic.tv/