THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2A

© 2020 T.A. Binkowski

MODERN OBJECTIVE-C

MODERN OBJECTIVE-C

SUBTITLE

- Programming language
- A library of objects
- Runtime environment
  - Defers many operations from compile time to runtime
  - Executes the compiled code

Simpler, safer though automation

MODERN OBJECTIVE-C

SUBTITLE

- Popularity by TIOBE Programming Community Index
  - 2007 - 45th
  - 2011 - 6th
  - 2012-16 - 4th

Simpler, safer though automation

MODERN OBJECTIVE-C

- Based on Smalltalk language
  - Messaging an object instead of ‘calling’ a function
- In "messaging"
  - The runtime decides which code gets executed
  - Performs a ‘lookup’ to identify method by name
  - Compiler does not check type of object being messaged
    - Dynamic binding at runtime

Looks up method to run

Code can compile without warnings/ errors and crash.

MODERN OBJECTIVE-C

- A strict-superset of ANSI C
  - Can use C code interspersed with objective-C code (in .m file)
    - Include header files using `#import`
  - Structs are used in many frameworks (for efficiency)
    - There is overhead to creating objects
    - Structs can only hold non object types (int, float, double, char, etc.)

```
struct CGRect {
  CGPoint origin;
  CGSize size;
};
typedef struct CGRect CGRect;
```

```
CGRect frame;
frame.origin.x = 0.0f;
frame.origin.y = 10.0f;
frame.size.width = 100.0f;
frame.size.height = 150.0f;
```

MODERN OBJECTIVE-C

OBJECTIVE-C ADD THE FOLLOWING SYNTAX AND FEATURES TO ANSI C

- Definition of new classes
  - Class and instance methods
  - Method invocation (called messaging)
  - Declaration of properties (synthesizing of accessor methods)
- Static and dynamic typing
- Blocks—encapsulated segments of code
- Extensions to the base language such as protocols and categories

MODERN OBJECTIVE-C

- NEXT was the most significant adopter of Objective-C
- Next added
  - Full implementation of object-oriented C
  - Retain and Release memory management model
  - Interface library and project builder

MODERN OBJECTIVE-C

OBJECTIVE-C 2.0 (2006)

- Apple introduced Obj-C 2.0 and added
  - Properties
  - Fast Enumeration
  - Blocks
  - Literal syntax
  - ARC (automated memory management)

Objective-C

MODERN OBJECTIVE-C

- Many of the 2.0 features are strictly compiler automation
  - The underlying language/runtime features are the same

Simpler, safer though automation

Objective-C

OBJECTIVE-C TYPES

OBJECTIVE-C

TYPES

```
typedef struct objc_object {
    Class isa;
} *id;
```

isa variable of what class it "isa" an instance of

- "id"
  - General type for any kind of object (e.g. instance of a class or class objects)
  - Pointer to an object data structure

OBJECTIVE-C

TYPES

```
if ( [anObject isKindOfClass:someClass] )
...

if ( [anObject isMemberOfClass:someClass] )
…
```

- "isa" variable allows objects to perform introspection
  - `isKindOfClass:` - includes subclasses
  - `isMemberOfClass:` - specific to a class (no subclasses)
  - `respondsToSelector:` - implements a selector (method); does not care what class it belongs to (polymorphism)

OBJECTIVE-C

TYPES

- `id` is the generic type for dynamic binding at runtime
  - Can be cast to a pointer (*)
  - Does not know (or care) what objects is until it needs it
  - Warning: Compiler does not check

OBJECTIVE-C

TYPES

- Use case examples for `id`
  - Identify which button is pressed in a group of buttons
  - Iterate through an NSArray of different objects
- Advantages of statically type objects (why not to use `id`)
  - Compile-time checking
  - Autocompletion in Xcode

OBJECTIVE-C

TYPES

- SEL (selector) type represents a way to refer to a method by name
- @selector() keyword return a SEL

```
// Test if a class implements a method
[object respondsToSelector:@selector(methodName)]

// Store a method and call on object
SEL method = @selector(methodName);
[object performSelector:method];

// Set method to call for target-action
[aButton addTarget:self action:@selector(eatCake)

forControlEvents:UIControlEventTouchDown];
```

OBJECTIVE-C

TYPES

```
if ([aObject respondsToSelector:@selector(myMethodName)])
{
    ...
}
```

Does aObject implement a method named 'myMethodName'?

- `@selector` turns a method into a name

```
[button addTarget:self action:@selector(digitPressed:)]
```

Has a parameter

When button is tapped, execute method named 'digitPressed' that is implemented in this class (self)

OBJECTIVE-C

TYPES

- `nil` object pointer to nothing
- Example: Test if it points to an object

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

- Instance variables can be set to nil

```
myObject.delegate = nil;
```

OBJECTIVE-C

SUBTITLE

- It is safe to send a message to nil
  - Invoking a method on nil returns a zero value

```
myObject = nil;
[myObject doSomething];
```

Warning!!! Would crash other languages

OBJECTIVE-C

TYPES

- Sending messages to nil simplifies expressions
  - Limits having to check for nil before doing anything
  - Use as a convenience feature of Obj-C

```
// For example, this expression...
if (name != nil && [name isEqualToString:@"Steve"]) { ... }

// ...can be simplified to:
if ([name isEqualToString:@"steve"]) { ... }
```

It was so much work to do this…

OBJECTIVE-C

TYPES

Symbol	Value	Meaning
`NULL`	`(void *)0`	literal null value for C pointers
`nil`	`(id)0`	literal null value for Objective-C objects
`Nil`	`(Class)0`	literal null value for Objective-C classes
`NSNull`	`[NSNull null]`	singleton object used to represent null

- Values representing nothing that every Objective-C programmer should know about

Collections can not hold nil, but can hold NSNull object

OBJECTIVE-C

TYPES

http://nshipster.com/

nil / Nil / NULL / NSNull

Written by Mattt Thompson on Jan 7th, 2013

Understanding the concept of nothingness is as much a philosophical issue as it is a pragmatic one. We are inhabitants of a universe of *somethings*, yet reason in a logical universe of existential uncertainties. As a physical manifestation of a logical system, computers are faced with the intractable problem of how to represent *nothing* with *something*.

In Objective-C, there are several different varieties of *nothing*. The reason for this goes back to a common NSHipster refrain, of how Objective-C bridges the procedural paradigm of C with Smalltalk-inspired object-oriented paradigm.

C represents *nothing* as `0` for primitive values, and `NULL` for pointers (which is equivalent to `0` in a pointer context).

OBJECTIVE-C

TYPES

- `BOOL` types to encode 'truth'
  - typedef of a signed char, with the macros YES and NO
  - YES = #define 1
  - NO = #define 0

```
// Test for true
if (myObject.value == YES) {...}
if (myObject.value) {...}

// Test for false
if (myObject.value == NO) {...}
if (!myObject.value) {...}

// Set BOOL type variable
myObject.updated = NO;
myObject.updated = YES;
```

OBJECTIVE-C

TYPES

- Truth types and values in Objective-C

Name	Typedef	Header	True Value	False Value
BOOL	signed char	objc.h	YES	NO
bool	_Bool(int)	stdbool.h	true	false
Boolean	unsigned char	MacTypes.h	TRUE	FALSE
NSNumber	__NSCFBoolean	Foundation.h	@(YES)	@(NO)
CFBooleanRef	struct	CoreFoundation.h	kCFBooleanTrue	k

OBJECTIVE-C
OBJECT MESSAGING

OBJECTIVE-C

OBJECT MESSAGING

- Send a message (method) to an object (receiver)

```
[receiver message];

[Car start];

[[[Car alloc] init] anotherMessage];
```

- When a message is sent, the runtime system selects method from the receiver
  - Method names in message are “selectors”

OBJECTIVE-C

OBJECT MESSAGING

MESSAGE EXPRESSION

MESSAGE

```
[receiver method:argument];
```

SELECTOR

OBJECTIVE-C

OBJECT MESSAGING

- Method parameters (arguments)
  - Not optional

```
[myObject setName:@"Jane"];
```

  - Order must be preserved

```
[myObject setName:@"Jane" andDate:[NSDate now] andEmail:jd@mail.com];
[myObject setName:@"Jane" andEmail:jd@mail.com andDate:[NSDate now]];
```

```
setName:andEmail:andDate
```

OBJECTIVE-C

OBJECT MESSAGING

- Polymorphism
  - Objects can respond differently to the same message

```
[personObject description];
```

  - Example: send a `description` message to an `id` variable

```
[carObject description];
```

NSObject provides 'description' to convert object to string. It varies depending on object. Good method to override in custom classes.

OBJECTIVE-C

OBJECT MESSAGING

```
myInstance.value = 10;
[myInstance setValue:10];


self.age = 10      // Access object's ivar
                   // using accessor methods
[self setAge:10]; // Equivalent to above
age = 10;          // Access ivar directly
```

- Dot Syntax
  - Alternative to square bracket notation "[ ]"
  - Compiler transforms to accessor methods

OBJECTIVE-C

OBJECT MESSAGING

- Method types:
  - Class methods “+” directly to class
  - Instance methods “-” scoped to instance of a class

```
[[myClass alloc] init];
```

CLASS METHOD

INSTANCE METHOD

OBJECTIVE-C

OBJECT MESSAGING

- Method declaration

image via App

OBJECTIVE-C

OBJECT MESSAGING

```
// Allocate memory for the object (always pair w/init)
+ (id)alloc;

// returns the one and only, shared (singleton) instance
+ (id)sharedObject;
```

- Class methods
  - Used for allocation, singletons, utilities
  - No access to local variables

OBJECTIVE-C

OBJECT MESSAGING

```
// Setter method
-(void)setString:(NSString *)newString
{
...
}
```

- Instance methods
  - Most common methods to implement
  - Access variables as if they were local
  - Can call methods on `self` and `super`

OBJECTIVE-C CLASSES

OBJECTIVE-C

CLASSES

- Classes declare state and behavior
  - State is maintained using instance variables
  - Behavior is implemented using methods

Interface files `.h`

Class name

Parent class name

Member variable declarations

Method declarations

```
@interface MyClass : NSObject
{
    int         count;
    id          data;
    NSString*   name;
}
- (id)initWithString:(NSString*)aName;
+ (MyClass*)createMyClassWithString:(NSString*)aName;
@end
```

OBJECTIVE-C

CLASSES

- Instance variables should only be accessed using getter/setter methods
  - Prefer use of @properties

```
@interface MyClass : NSObject
{
    int         count;
    id          data;
    NSString*   name;
}
- (id)initWithString:(NSString*)aName;
+ (MyClass*)createMyClassWithString:(NSString*)aName;
@end
```

OBJECTIVE-C

CLASSES

Subclass of UIViewController

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

@interface MyViewController : UIViewController {

}
@end
```

- Inheritance
  - Class definitions are additive
  - Methods and instance variables
  - Single inheritance
- Overwrite existing methods
  - Custom ‘description’ method

OBJECTIVE-C

CLASSES

- Every Foundation framework class inherits from `NSObject`
  - Memory management
  - Runtime cooperation

image via Apple Inc.

OBJECTIVE-C

CLASSES

- Creating a class instance by sending `alloc`
  - Dynamically allocates memory

```
//  Alloc and init
Class *myClass = [[myClass alloc] init];

//  Init with parameters
Class* myClass = [[myClass alloc] initWithInt:1];
```

- Initialize a class instance's instance variables by sending `init`
  - Can customize or override `init` method
  - Must call [super init]

OBJECTIVE-C

CLASSES

- All NSObjects implement the method -description
  - [NSString stringWithFormat:@"This is my class:%@", anObject];
- Called in format string using %@
  - [NSString stringWithFormat:@"This is my class:%@", anObject];
- Called in format string using %@
  - NSLog([anObject description]);
- Common to overwrite it for custom classes
-

```
-(NSString *)description
{
  return [NSString stringWithFormat:@"<GameObject: %@, Position: %f, %f>",
                  [self objectID], [self position].x, [self position].y];
}
```

OBJECTIVE-C

CLASSES

- Convenience methods to create a class
  - Automatically alloc, init and autorelease (pre-ARC) an object
  - Pro: Less code, will not leak (pre-ARC)
  - Cons: Less control

```
//  Convenience method
NSString* myString = [NSString string];

//  Manual
NSString* myString = [[NSString alloc] init];
```

OBJECTIVE-C

CLASSES

- Destroying a class by sending message `release`

```
//  Pair alloc-init with a release
Class *myClass = [[myClass alloc] init];
...
[myClass release];

//  Best practices
[myClass release], myClass = nil;
```

PRE-ARC

- Setting the variable to `nil` prevents bad access later
  - Sending a message to `nil does nothing

OBJECTIVE-C

CLASSES

- Class equalities of pointer values (e.g. class instances are identical)

```
if ([objectA class] == [objectB class]) { //...
```

- Class instance values are equivalent

```
if ([object1 isEqual: object2]) { //...
```

OBJECTIVE-C MEMORY MANAGEMENT

FOR HISTORICAL PURPOSES

OBJECTIVE-C

MEMORY MANAGEMENT

- Reference counting - keep track of all references to an object
  - Typical pattern: alloc, retain, release

image via http://cocoadevcenttral.com

- If you create an object with `alloc` or `copy` you have to send it `release` or `autorelease`
  - Calls must be balanced or will leak memory or crash

OBJECTIVE-C

MEMORY MANAGEMENT

- Reference counting part of NSObject
  - Objects exist in memory if count > 0
- Object lifecycle
  - `+alloc` and `-copy` create objects with count = 1
  - `-retain` increments retain count
  - `-release` decrements retain count
- When retain count == 0
  - Object is destroyed and memory release
  - `-dealloc` method invoked automatically; do not call

OBJECTIVE-C

MEMORY MANAGEMENT

- Query the retain count of an object

```
NSString *myString = [[NSString alloc] init];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString retain];
  // NSLog(@"Count:%@",[myString count]);
  // 2

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 0 (Dealloced)
```

OBJECTIVE-C

MEMORY MANAGEMENT

- Query the retain count of an object

```
NSString *myString = [[NSString alloc] init];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString retain];
  // NSLog(@"Count:%@",[myString count]);
  // 2

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 0 (Dealloced)

  [myString release];
  // Crash
```

OBJECTIVE-C

MEMORY MANAGEMENT

```
@interface MyAppDelegate_iPhone : MyAppDelegate {
    UILabel *theLabel;
}
@property (nonatomic, retain) IBOutlet UILabel *theLabel;
```

- Balance retain and release of instance variables

```
- (void)dealloc
{
    [theLabel release];
    [super dealloc];
}
```

OBJECTIVE-C

MEMORY MANAGEMENT

- Creating and initializing NSString

```
– initWithCharacters:length:
– initWithString:
– initWithCString:encoding:
– initWithUTF8String:
...
+ stringWithFormat:
+ localizedStringWithFormat:
+ stringWithCharacters:length:
+ stringWithString:
+ stringWithCString:encoding:
```

Convenience methods

OBJECTIVE-C

MEMORY MANAGEMENT

- Using `autorelease`
  - Objects are added to the autorelease pool
  - The pool is “drained” after each event
- Call autorelease on object you alloc-init
- Autorelease is built-in to convenience methods

Table of Contents

Jump To...

- Overview
- Tasks
- Class Methods
- Instance Methods
- Revision History

COMPANION GUIDE

Memory Management Programmi...

NSAutoreleasePool Cl

Inherits from	NSObject
Conforms to	NSObject (NSObjec
Framework	/System/Library/F
Availability	Available in Mac O
Companion guide	Memory Managem
Declared in	NSAutoreleasePool
Related sample code	CocoaSpeechSynth OpenCL NBody Sim SpellingChecker Ca SpellingChecker-C SpellingChecker-C

Overview

The `NSAutoreleasePool` class is used to sup
management system. An autorelease pool sto
message when the pool itself is drained.

In a reference-counted environment (as oppo
an `NSAutoreleasePool` object contains obje
message and when drained it sends a `releas`
sending `autorelease` instead of `release` to

OBJECTIVE-C

MEMORY MANAGEMENT

- Methods with alloc, copy or new in name need to be released or autorelease

```
[[NSString alloc] initWithString:@"hi"] autorelease];
```

- Convenience methods
  - Cover entire lifecycle of object
  - Autorelease built-in

```
[NSString stringWithString:@"Hi"];
```

OBJECTIVE-C

MEMORY MANAGEMENT

- Each application creates an NSAutoreleasePool in main.m
  - The pool sends a release message (e.g. drains) at the end of every event loop cycle
- Create local ones for use in loops, multithreading or other processes

```
int main(int argc, char *argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

OBJECTIVE-C AUTOMATIC REFERENCE COUNTING

OBJECTIVE-C

ARC

SUBTITLE

- Automatic Reference Counting
  - Introduced in iOS 5
  - Implements automatic memory management for Objective-C objects and blocks
  - Frees the programmer from the need explicitly insert retains and releases

OBJECTIVE-C

ARC

- ARC is a compiler feature that provides automatic memory management of objective-c objects

OBJECTIVE-C

ARC

- Follows the same memory management convention but “adds code at compile time”
- Evaluates the lifetime of an object
  - Inserts memory management into your compiled code
  - Generates dealloc methods
- How it works
  - Pointers keep objects in memory
  - If pointer gets a new value or is “zeroed” (nil) it is released

OBJECTIVE-C

ARC

```
@property (weak, nonatomic) IBOutlet UILabel *sliderValue;
@property (strong, nonatomic) NSString *name;
```

- Property attributes (pointer types)
  - strong
    - Keeps objects alive
  - weak
    - Point to object, but do not keep it around because another object is managing it
    - IBOutlets are weak because they are managed by parent (e.g. view controller's view)
    - Parent-child relationships (parent - strong; child - weak)

These concepts are in swift

OBJECTIVE-C

ARC

- New rules because of ARC
  - Cannot explicitly invoke retain, release, dealloc, retainCount, autorelease
  - Cannot use object pointers in C structs
  - No custom autorelease pools
  - Cannot prefix a property accessor with “new” (e.g. newLabel)

OBJECTIVE-C DECLARED PROPERTIES

OBJECTIVE-C

DECLARED PROPERTIES

```
@property (strong, nonatomic) UIView *myView;
```

- `@property` is data encapsulated or stored by an object
  - Attributes of an object (e.g. label.text, view.backgroundColor)

OBJECTIVE-C

DECLARED PROPERTIES

- Properties simplify classes
  - Expose variable and methods through *accessor methods (e.g. getters and setters)*
  - Compiler automatically generates getters and setters
- Simplified memory management if using old method
  - Redundant if using ARC (but still common practice and seen in large frameworks)

```
@property (strong, nonatomic) UIView *myView;
```

OBJECTIVE-C

DECLARED PROPERTIES

```
@property (strong, nonatomic) UIColor *color;
```

- Class of the object defines an interface to set and get the values of the property
  - Property Name is `color`
  - Setter created named ‘setColor’
  - Getter created named ‘color’
  - Backing variable `_color`

Naming conventions

OBJECTIVE-C

DECLARED PROPERTIES

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}

// Declare accessor methods
- (void) name;
- (void) setName;

@end
```

```
//
// Leaf.m
//

@implementation Leaf

// Setter
- (void) setName:(NSString *)newName {
    if (_name != newString) {
        [_name release];
        _name = [newName copy];
    }
}

// Getter
- (NSString*)name {
    return _name;
}
@end
```

- Class creation in 2005

OBJECTIVE-C

DECLARED PROPERTIES

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}
@property(nonatomic, retain)
                NSString *name;

// Declare accessor methods
- (void) name;
- (void) setName;

@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;

// Setter
- (void) setName:(NSString *)newName {
    if (_name != newString) {
        [_name release];
        _name = [newName copy];
    }
}

// Getter
- (NSString*)name {
    return _name;
}
@end
```

SYNTHESIZE A PROPERTY "NAME"
IVAR NAME = _NAME
PROPERTY NAME = NAME
SETTER = SETNAME
GETTER = NAME

- Class creation in 2010 (Objective-C 2.0)

OBJECTIVE-C

DECLARED PROPERTIES

```
//
// Leaf.h
//

@interage Leaf : NSObject {
    NSString *_name; // ivar
}
@property(nonatomic,retain)
                 NSString *name;





@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;
@end
```

- Class creation in 2010 (Objective-C 2.0)

OBJECTIVE-C

DECLARED PROPERTIES

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}
@property(strong)NSString *name;

@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;
@end
```

- Class creation in 2012

OBJECTIVE-C

DECLARED PROPERTIES

```
//
// Leaf.h
//

@interface Leaf : NSObject
@property(strong)NSString *name;
@end
```

```
//
// Leaf.m
//

@implementation Leaf
@end
```

```
// @synthesize name = _name
// done by default
```

OBJECTIVE-C

DECLARED PROPERTIES

```
@property (nonatomic, weak) IBOutlet UILabel *theLabel;
```

- Atomicity
  - Multi-threading option
  - ALWAYS set to (`nonatomic`) unless you mean it; atomic is default
- Writeabilty
  - `readwrite` - default behavior; creates getters and setters
  - `readonly` - creates getter only
- Memory management:
  - Applies to setters only
  - Options: `strong`,`weak`, `copy`

OBJECTIVE-C

DECLARED PROPERTIES

```
// Generate getters/setters and retain the incoming object
@property (nonatomic, strong) NSString *name;

// Same
@property (nonatomic, strong) NSString *title;

// Generate getters/setters and assigns the incoming value
@property (nonatomic) int age;

// Generate getters/setters and assigns the incoming value
@property int age;

// Generate the getters only
@property (nonatomic, readonly) NSDate *created;

// Generate the getters/setters and create a new instance
// with the same values as the original
@property (nonatomic, copy) NSString *backupData;
```

OBJECTIVE-C

DECLARED PROPERTIES

```
// Generate getters/setters and retain the incoming object
@property (nonatomic, strong) NSString *name;

// Same
@property (nonatomic, strong) NSString *title;

// Generate getters/setters and assigns the incoming value
@property (nonatomic) int age;

// Generate getters/setters and assigns the incoming value
@property int age;

// Generate the getters only
@property (nonatomic, readonly) NSDate *created;

// Generate the getters/setters and create a new instar
// with the same values as the original
@property (nonatomic, copy) NSString *backupData;
```

copy is required when the object is mutable. Use this if you need the value of the object as it is at this moment, and you don't want that value to reflect any changes made by other owners of the object.

OBJECTIVE-C

DECLARED PROPERTIES

```
// Getter for name property
[self name];

// Call method "doSomething" in current class
[self doSomething];

// Self as parameter to method
[someClass doWorkOn:self];
```

- A class' `self`
  - Implicit local variable
  - Points to the object that was sent the message
  - Allows an object to send a message `[self addDate]`
- `self` uses the accessor methods to access instance variables

OBJECTIVE-C

DECLARED PROPERTIES

- Allows use of dot notation
  - Different notation `[myObject name]` or `myObject.name`

```
// Create property
@property int year;

// Bracket notation
[myCar year];           // Getter
[myCar setYear: 1962]; // Setter

// Dot notation
myCar.year = 1962;          // Setter
NSLog(@"%d", myCar.year);  // Getter
```

OBJECTIVE-C

DECLARED PROPERTIES

- Invoking accessor methods (i.e. getters and setters)

```
[[object theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
[object.theArray insertObject:myAppObject.objectToInsert atIndex:0];
```

dot

dot

- Use for assignment

```
myAppObject.theArray = aNewArray;
[myAppObject setTheArray:aNewArray];
```

OBJECTIVE-C BLOCKS

BLOCKS

- Objects that encapsulate a unit of work
  - Anonymous functions
  - Passed as parameters to methods; returned from methods

```
int result = myBlock(4); // result is 28
```

Xcode helps

BLOCKS

- Simplify execution of a single task versus writing a collection of methods
  - Scoped locally and variables can be read-only or read-write
- Used throughout UIKit
  - Callbacks
  - Animations
  - Multi-threading
  - Parameters

```
- (id)addObserverForName:(NSString *)name
                  object:(id)obj
                   queue:(NSOperationQueue *)queue
              usingBlock:(void (^)(NSNotification *note))block
```

OBJECTIVE-C CATEGORIES

CATEGORIES

SUBTITLE

- Allow you to add methods to an existing class
- Why not subclass?
  - Would not have to change existing codebase
  - Can be applied to other methods that return NS and UI classes

CATEGORIES

Choose options for your new file:
Class
NSArray+Shuffle
Subclass of
NSObject
Cancel
Previous
Next

CATEGORIES

- Declaring the interface for a category

```
//
//  NSArray+Shuffle.h
//  MapKitDemonstration
//
#import <Foundation/Foundation.h>

@interface NSArray (Shuffle)
- (NSArray*) shuffle;
@end
```

CATEGORIES

```
//
//  NSArray+Shuffle.m
//  MapKitDemonstration

#import "NSArray+Shuffle.h"

@implementation NSArray (Shuffle)
- (NSArray*)shuffle
{
    // create temporary mutable array
    NSMutableArray *tmpArray = [NSMutableArray arrayWithCapacity:[self count]];

    for (id anObject in self) {
       NSUInteger randomPos = arc4random()%([tmpArray count]+1);
       [tmpArray insertObject:anObject atIndex:randomPos];
    }

    return [NSArray arrayWithArray:tmpArray];  // non-mutable autoreleased copy
}

@end
```

- Implementing a category

CATEGORIES

- Using a category

```
// Test NSSArray+Shuffle Category
   NSArray *places = [NSArray arrayWithObjects:@"Universal Studios",
                                               @"Disney World", @"Sea World", nil];
   NSLog(@"Places (before): %@",places);
   NSLog(@"Places (after): %@",[places shuffle]);
```

CATEGORIES

- Tips & Tricks
  - Categories are inherited (i.e. categories on NSObject are available to every class)
  - The names must be unique across all namespaces
- A couple useful categories
  - coredata-easyfetch
    - https://github.com/halostatue/coredata-easyfetch
  - Image utilities
    - http://mattgemmell.com/2010/07/05/mgimageutilities/

OBJECTIVE-C PROTOCOLS

PROTOCOLS

- Declare methods that can be implemented by any class
- Not a class; define an interface that other objects can implement
  - "Conforming to a protocol"
  - Implement in header file

```
@protocol MyProtocol

@required
- (void)myProtocolMethod;
@option
- (void)anotherMethod;

@end
```

PROTOCOLS

- Classes that wish to implement a protocol declare it in their interface

```
@interface MyClass : NSObject <AProtocol> {}
...
@end
```

- All non-optional methods must be implemented.
- Used to specify the interface for a delegate object
  - Can be done programmatically or in Storyboard

PROTOCOLS

It's A Zoo
iPhone 5
It's A Zoo | Build It's A Zoo: Succeeded | Today at 3:54 PM
It's A Zoo
2 targets, iOS SDK 8.1
It's A Zoo
AppDelegate.h
AppDelegate.m
Animal.h
Animal.m
ViewController.h
ViewController.m
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files
Info.plist
main.m
It's A ZooTests
Products
View Controller Scene
View Controller
View
Scroll View
Label
First Responder
Exit
UIScrollView
Outlets
delegate
View
Outlet Collections
gestureRecognizers
Referencing Outl
View Controller

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Protocol definition

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Declare a delegate that adopts a protocol

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Conform to a protocol

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scr
- (void)scrollViewDidEndDragging:(UIScrollView *)scroll
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Set delegate when methods are implemented

PROTOCOLS

```
@interface MyViewController : UIViewController <UIScrollViewDelegate> {}

- (void)loadView {
...
  // Add the scrollview
  scroll = [[UIScrollView alloc] initWithFrame:CGRectMake(0,30,320,400)];
}


- (void)scrollViewDidScroll:(UIScrollView *)scrollView {
    static NSInteger previousPage = 0;
    CGFloat pageWidth = scrollView.frame.size.width;
    float fractionalPage = scrollView.contentOffset.x / pageWidth;
    NSInteger page = lround(fractionalPage);
    if (previousPage != page) {
        self.currentPage = page;
        previousPage = page;
        NSLog(@"PAGE %d",self.currentPage);
        // Do something with page
    }
}
```

Method declared in protocol. Sends messages every time the scrollview scrolls

OBJECTIVE-C CHEAT SHEET

OBJECTIVE-C

header File

```
//
//  MyClass.h
//  Objective-C Class
//

#import <Foundation/Foundation.h>

@interface MyClass : NSObject
{
    // Declare instance variables (optional)
}

// Define properties

// Define methods

@end
```

Implementation File

```
//
//  MyClass.m
//  Objective-C-hristmas
//

#import "MyClass.h"

@interface MyClass ()
// Define private properties
// Define private methods
@end



@implementation MyClass
// Implement methods
@end
```

OBJECTIVE-C

```
// Defining methods
- (void)message;
- (void)message:(id)parameter;
- (BOOL)message:(id)parameter1 anotherMessage:(id)parameter2;


// Implementing methods
- (void)message {
     // Do some stuff
}

- (BOOL)message:(id)parameter {
    // Do some stuff with parameter
    return YES;
}
```

OBJECTIVE-C

```
// Defining Properties
@property (attribute1, attribute2) propertyName;


// Synthesizing Properties
_propertyName   // Instance variable created
propertyName    // Accessor getter method
setPropertyName // Accessor setter method


// Bracket syntax
value = [MyClass propertyName];
// Dot syntax for getter
value = MyClass.propertyName;

// Bracket syntax
[myClass setPropertyName:value];
// Dot syntax for setter
myClass.propertyName = value;
```

OBJECTIVE-C

```
// Creating a class instance
MyClass *classObject = [[MyClass alloc] init];


// Creating a custom init method
- (id) initWithParameter:(id)parameter {
     if ( self = [super init]) {
         self.someProperty = parameter
   }
    return self;
}
```

NSOBJECT DECLARES INIT PROTOTYPE

CALLS INIT ON SUPER CLASS

OBJECTIVE-C

```
// Messaging - invoking methods on an object

// Fancy objective-c terms
[receiver message];

// Message expression
[myClass method];

// Pass an argument as part of the message
[myClass message:argument];

// Pass two argument2 as part of the message
[myClass message:argument anotherMessage:anotherArgument];

// Nesting Methods
id output = [receiver message];
[myClass message:output];
[myClass message:[object method]];
```

OBJECTIVE-C

```
// Declare a custom protocol
@protocol MyProtocol
@required
- (void)myProtocolMethod;
@optional
- (void)anotherMethod;
@end
```

DECLARE PROTOCOL

```
// .h
@interface MyClass : NSObject <MyProtocol>
@end
```

CONFORM TO IT

```
// .m
@implementation MyClass
- (void)myProtocolMethod  {
   // Do some stuff
{
@end
```

IMPLEMENT PROTOCOL

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2A

© 2020 T.A. Binkowski

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2B

© 2020 T.A. Binkowski

A CUSTOM OBJECTIVE-C CLASS

A CUSTOM OBJECTIVE-C CLASS

A CUSTOM OBJECTIVE-C CLASS

A CUSTOM OBJECTIVE-C CLASS

A CUSTOM OBJECTIVE-C CLASS

Header file

Properties

Method

It's A Zoo
2 targets, iOS SDK 8.1 M

- It's A Zoo
  - AppDelegate.h
  - AppDelegate.m
  - ViewController.h
  - ViewController.m
  - Animal.h A
  - Animal.m A
  - Main.storyboard
  - Images.xcassets
  - LaunchScreen.xib
  - Supporting Files
- It's A ZooTests
- Products

```
//
//  Animal.h
//  It's A Zoo
//
//  Created by T. Andrew Binkowski on 1/12/15.
//  Copyright (c) 2015 The University of Chicago. All rights rese
//

#import <Foundation/Foundation.h>
@import UIKit;

@interface Animal : NSObject

// Declared properties
@property (strong, nonatomic) NSString *name;
@property (strong, nonatomic) NSString *species;
@property (strong, nonatomic) NSNumber *age;
@property (strong, nonatomic) UIImage *image;

// Instance method
- (NSString*)describeAnimal;

@end
```

Identity and
Nar
Ty
Locati
Full Pa
Target Memb
It's A
It's A
Text Settings
Text Encodi
Line Endin
Indent Usi
Widt
Source Cont
Reposito
Ty
Current Bran
Versi

A CUSTOM OBJECTIVE-C CLASS

We don't have to write [init] methods; only if you want to

```
#import "Animal.h"

@implementation Animal


- (NSString*)describeAnimal
{
    NSString *descriptionString = [NSString stringWithFormat:@"Animal Name:%@ Species:%@ Age:%@ \n",
                                   self.name,self.species,self.age];
    return descriptionString;
}

@end
```

A CUSTOM OBJECTIVE-C CLASS

- Animal.h
- Animal.m
- Main.storyboard
- Images.xcassets
- LaunchScreen.xib
- Supporting Files
- It's A ZooTests
- Products

```
#import "ViewController.h"
#import "Animal.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the nib.
}


- (void)viewWillAppear:(BOOL)animated
{
    // Create new dog object and fill it with data
    Animal *dog = [[Animal alloc] init];
    dog.name = @"Max";
    dog.species = @"Great Dane";
    dog.age = @1;
    dog.image = [UIImage imageNamed:@"ImageInAssetCatalog"];

    NSLog(@"Dog Description:%@",[dog describeAnimal]);
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
```

import class

create new class

call our method; message our object

A CUSTOM OBJECTIVE-C CLASS

Images.xcassets

iOS Simulator - iPhone 5 - iPhone 5 / iOS 8.1 (...

Carrier 12:27 PM

```
@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}


- (void)viewWillAppear:(BOOL)animated
{
    // Create new dog object and fill it with data
    Animal *dog = [[Animal alloc] init];
    dog.name = @"Max";
    dog.species = @"Great Dane";
    dog.age = @1;
    dog.image = [UIImage imageNamed:@"ImageInAssetCatalog"];

    NSLog(@"Dog Description:%@",[dog describeAnimal]);
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
```

console output

OBJECTIVE-C IN A SWIFT PROJECT

CREATE A PROJECT

On This Page

Swift and Objective-C in the Same Project

Swift's compatibility with Objective-C lets you create a project that contains files written in either language. You can use this feature, called *mix and match*, to write apps that have a mixed-language codebase. Using mix and match, you can implement part of your app's functionality using the latest Swift features and seamlessly incorporate it back into your existing Objective-C codebase.

Mix and Match Overview

https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/MixandMatch.html

CREATE A PROJECT

AN OBJECTIVE-C VIEW CONTROLLER

CREATE A PROJECT

AN OBJECTIVE-C VIEW CONTROLLER

AN OBJECTIVE-C VIEW CONTROLLER

AN OBJECTIVE-C VIEW CONTROLLER

2016-SwitchToObjectiveC 〉 2016-SwitchToObjectiveC 〉 SwitchViewController.h 〉 No Selection

- 2016-SwitchToObjectiveC M
  - 2016-SwitchToObjectiveC
    - AppDelegate.swift A
    - ViewController.swift A
    - Main.storyboard A
    - SwitchViewController.h A
    - SwitchViewController.m A
    - Assets.xcassets A
    - LaunchScreen.storyboard A
    - Info.plist A
    - 2016-SwitchT...dging-Header.h A
  - Products

```objc
//
//  SwitchViewController.h
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Department o
//

#import <UIKit/UIKit.h>

@interface SwitchViewController : UIViewController

@end
```

Counterparts 〉 SwitchViewController.m 〉 No Selection

```objc
//
//  SwitchViewController.m
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Department o
//

#import "SwitchViewController.h"

@interface SwitchViewController ()

@end

@implementation SwitchViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(i
    // Get the new view controller using [segue destination
```

AN OBJECTIVE-C VIEW CONTROLLER

- 2016-SwitchToObjectiveC
  - 2017-SwitchToObjectiveC
    - AppDelegate.swift
    - ViewController.swift
    - SwitchViewController.h
    - SwitchViewController.m
    - Assets.xcassets
    - Main.storyboard
    - LaunchScreen.storyboard
    - Info.plist
    - 2016-SwitchToObjectiveC-Bridging-Header.h
    - GreetingObject.h
    - GreetingObject.m
  - Products

```
//
//  Use this file to import your target's public headers that
//

#import "GreetingObject.h"
```

IMPORT THE OBJECTIVE-C HEADER INTO A SWIFT PROJECT FOR CUSTOM CLASSES

AN OBJECTIVE-C VIEW CONTROLLER

AN OBJECTIVE-C VIEW CONTROLLER

IBOUTLETS WILL AUTOMATICALLY CREATE METHOD DEFINITIONS DURING DRAG/DROP

2016-SwitchToObjectiveC
- 2016-SwitchToObjectiveC
  - AppDelegate.swift
  - ViewController.swift
  - Main.storyboard
  - SwitchViewController.h
  - SwitchViewController.m
  - Assets.xcassets
  - LaunchScre....storyboard
  - Info.plist
  - 2016-Switc...g-Header.h
- Products

2016-SwitchToObjectiveC 〉 2016-SwitchToObjectiveC 〉 SwitchViewController.m 〉 No Selection

```
//

#import "SwitchViewController.h"

@interface SwitchViewController ()

@end

@implementation SwitchViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Do any additional setup after loading the view.
    NSLog(@"The view did load");

}


// Toggle the values of the switch and store its values in `NSUserDefa
// Dump the user defaults to the console for debugging
- (IBAction)tapSwitch:(UISwitch *)sender {
  NSLog(@"Switch Value:%d", sender.isOn);

  NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
  NSLog(@"Defaults: %@",defaults.dictionaryRepresentation);

  [defaults setBool:sender.isOn forKey:@"kSwitchPreference"];
  NSLog(@"Defalts: %@", defaults.dictionaryRepresentation);
}

@end
```

Counterparts 〉 SwitchVie Controller.h 〉 No Selection

```
//
//  SwitchViewController.h
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Dep
//

#import <UIKit/UIKit.h>

@interface SwitchViewController : UIViewController
- (IBAction)tapSwitch:(UISwitch *)sender;

@end
```

USING CUSTOM CLASSES

CREATE A PROJECT

BRIDGING HEADER ALLOWS OBJECTIVE-C TO BE CALLED FROM SWIFT (AND VICE VERSA)

CREATE A PROJECT

2016-SwitchToObjectiveC › 2016-SwitchToObjective-C › GreetingObject.h › @interface GreetingObject

- 2016-SwitchToObjectiveC (M)
  - 2016-SwitchToObjectiveC
    - AppDelegate.swift (A)
    - ViewController.swift (A)
    - Main.storyboard (A)
    - SwitchViewController.h (M)
    - SwitchViewController.m (A)
    - Assets.xcassets (A)
    - LaunchScre....storyboard (A)
    - Info.plist (A)
    - 2016-Switc...g-Header.h (A)
    - GreetingObject.h (A)
    - GreetingObject.m (A)
  - Products

```
//
//  GreetingObject.h
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Department of Computer
//

#import <Foundation/Foundation.h>

@interface GreetingObject : NSObject

@property (strong, nonatomic) NSString *message;
@property (strong, nonatomic) NSDate *today;

- (id)initWithMessage:(NSString*)message;
- (void)greetWithDate;

@end
```

Counterparts › GreetingObject.m › No Selection

```
//
//  GreetingObject.m
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Depa
//

#import "GreetingObject.h"

@implementation GreetingObject


- (id)initWithMessage:(NSString*)message {
  if ( self = [super init]) {
    self.message = message;
    self.today = [NSDate date];
  }
  return self;
}

- (void)greetWithDate {
  NSLog(@"%@ on %@", self.message, self.today);
}

@end
```

CUSTOM INITIALIZATION METHOD

CREATE A PROJECT

2016-SwitchToObjectiveC 〉 2016-SwitchToObjectiveC 〉 2016-SwitchToObjectiveC-Bridging-Header.h 〉 No Selection

- 2016-SwitchToObjectiveC M
  - 2016-SwitchToObjectiveC
    - AppDelegate.swift A
    - ViewController.swift A
    - Main.storyboard A
    - SwitchViewController.h M
    - SwitchViewController.m A
    - Assets.xcassets A
    - LaunchScre....storyboard A
    - Info.plist A
    - 2016-Switc...g-Header.h A
    - GreetingObject.h A
    - GreetingObject.m A
  - Products

```
//
//  Use this file to import your target's public headers that you would like to expose to Swift.
//

#import "GreetingObject.h"
```

THAT'S IT

CREATE A PROJECT

THE BRIDGING HEADER CONVERTS TO SWIFT (DOES NOT ACTUALLY GENERATE THE FILE)

2016-Swit...bjectiveC › GreetingObject.h › No Selection

```objc
//
//  GreetingObject.h
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Department of
    Computer Science. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface GreetingObject : NSObject

@property (strong, nonatomic) NSString *message;
@property (strong, nonatomic) NSDate *today;

/// Custom initialization that sets the message and timestamp
/// - parameters message: An `NSString` that holds the message to
    display
- (id)initWithMessage:(NSString*)message;

/// Print the message and date to the console
- (void)greetWithDate;

@end
```

DECLARE CLASS METHODS AND PROPERTIES IN HEADER

Counterparts › GreetingObject.h (Interface) 2

```swift
//
//  GreetingObject.h
//  2016-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/3/16.
//  Copyright © 2016 The University of Chicago, Department of
    Computer Science. All rights reserved.
//

import Foundation

open class GreetingObject : NSObject {


    open var message: String!

    open var today: Date!


    /// Custom initialization that sets the message and timestamp
    /// - parameters message: An `NSString` that holds the messag
        display
    public init!(message: String!)


    /// Print the message and date to the console
    open func greetWithDate()
}
```

CREATE A PROJECT

```swift
class ViewController: UIViewController {

  //
  // MARK: - Lifecycle
  //


  override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)

    // Load up the defaults and print them the console (we should be able to see
    // that we changed the values from Objectice-C
    let defaults = UserDefaults.standard
    print("Defaults (from Swift ViewController:\n \(defaults.dictionaryRepresentation())")

    // Call our objective-c code
    let message = GreetingObject(message: "This is from Swift")
    message?.greetWithDate()

  }

}
```

```objc
- (id)initWithMessage:(NSString*)message;
- (void)greetWithDate;
```

CREATE A PROJECT

```swift
let d = Dog()
let numberOfDogs = d.number(ofDogs: 4)
print("Number of Dogs: \(numberOfDogs)")
```

```objc
//
//  Dog.m
//  2018-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/9/18.
//  Copyright © 2018 The University of Chicago,
    Department of Computer Science. All rights reserved.
//

#import "Dog.h"

@implementation Dog
- (int)numberOfDogs:(int)number {
  return number;
}
@end

```

```objc
//
//  Dog.h
//  2018-SwitchToObjectiveC
//
//  Created by T. Andrew Binkowski on 4/9/18.
//  Copyright © 2018 The University of Chicago,
    Department of Computer Science. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Dog : NSObject
@property (strong, nonatomic) NSString *message;

- (int)numberOfDogs:(int)number;

@end

```

CREATE A PROJECT

- https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/InteractingWithObjective-CAPIs.html#//apple_ref/doc/uid/TP40014216-CH4-ID35

```
@property (nullable) id nullableProperty;
@property (nonnull) id nonNullProperty;
@property id unannotatedProperty;
```

```
var nullableProperty: Any?
var nonNullProperty: Any
var unannotatedProperty: Any!
```

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2B

© 2020 T.A. Binkowski

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2C

© 2020 T.A. Binkowski

UIKIT DYNAMICS

ANIMATIONS AND INTERACTIONS ON IOS

- Core Animation
- UIView animations
- Motion effects
- Gesture driven interactions

WHAT IS UIKIT DYNAMICS?

- A composable, reusable, declarative, real-world inspired animation and interaction system
- Allow physics based interactions
  - Gravity
  - Collisions
  - Attachments
  - Snap
  - Forces
  - Item properties

UIKIT DYNAMICS

SUBTITLE

- Simplify animations that were complex
  - Math
  - Physics
  - Core Animation
- Underlying 2D physics engine
  - Based on Box2D
  - Not “accurate”

Same as SpriteKit

UIKit Dynamics

Apply physics-based animations to your views.

Framework
UIKit

Topics

Dynamic Animators

`class UIDynamicAnimator`
An object that provides physics-related capabilities and animations for its dynamic it
and provides the context for those animations.

Dynamic Items

`protocol UIDynamicItem`
A set of methods that can make a custom object eligible to participate in UIKit Dynar

`class UIDynamicItemBehavior`
A base dynamic animation configuration for one or more dynamic items.

`class UIDynamicItemGroup`
A dynamic item that comprises multiple other dynamic items.

Behaviors

`class UIDynamicBehavior`
An object that confers a behavioral configuration on one or more dynamic items, for
participation in 2D animation.

`class UIAttachmentBehavior`
A relationship between two dynamic items, or between a dynamic item and an ancho
point.

`class UICollisionBehavior`

UIKIT DYNAMICS

- Not for games
  - Flappy bird using UIKit Dynamics on github
  - Use Sprite Kit instead

BaneWong / FlappyBird-UIKitDynamics

Watch

Code | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights

Joke demo app implementing popular Flappy Bird game with UIKit Dynamics

7 commits | 1 branch | 0 packages | 0 releases

Branch: master | New pull request | Create new file | Upload

chordogg removed tap gesture recognizer and hooked up touchesbegan for more re...

FlappyBird-UIKitDynamics.xcode...	Added initial gameplay
FlappyBird-UIKitDynamics	removed tap gesture recognizer and hooked up touchesbegan for mo
FlappyBird-UIKitDynamicsTests	Added initial gameplay
.gitignore	Initial commit
README.md	Initial commit

README.md

FlappyBird-UIKitDynamics

Joke demo app implementing popular Flappy Bird game with UIKit Dynamics

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHu

WHY?

SUBTITLE

- Real world inspired interactions
  - Combining predefined and interactive animations
- Designed for increased user interface interactions
- Full featured demonstration
  - UIKit Dynamics Catalog from Apple WWDC 2013 (in Objectice-C)
  - WWDC 2015, 2016

API hasn't changed much

UIKITDYNAMICS CLASSES

UIKIT DYNAMICS CLASSES

- UIDynamicAnimator
- UIDynamicBehavior
- UIDynamicItems

Image via Apple

UIKIT DYNAMICS CLASSES

- Reference View is the UIView to act on

Image via Apple

UIKIT DYNAMICS CLASSES

- Reference view can act on multiple subviews

Image via Apple

UIKIT DYNAMICS CLASSES

SUBTITLE

- UIDynamicAnimator
  - Provides the overall context
  - Control engine
  - Define the coordinate system
  - Monitor behaviors

UIKIT DYNAMICS

```
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

- Instantiate a UIDynamicAnimator
- Add behaviors

View that provides the context for the dynamic animator

UIKIT DYNAMICS

```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
[animator addBehavior:…];
[animator addBehavior:…];
```

- Instantiate a UIDynamicAnimator
- Add behaviors

View that provides the context for the dynamic animator

UIKIT DYNAMICS

Running 2018-session2-uikitdynamics

2018-session2-uikitdynamics 〉 Untitled Page

```
//: A UIKit based Playground for presenting user interface

import UIKit
import PlaygroundSupport

class MyViewController : UIViewController {

  var animator:UIDynamicAnimator? = nil;
  let gravity = UIGravityBehavior()


  override func loadView() {
    let view = UIView()                                                    UIView
    view.backgroundColor = .white                                          UIView
    self.view = view                                                       .MyViewController: 0x7...
    play()                                                                 .MyViewController: 0x7...
  }


  func play() {

    let square = UIView(frame: CGRect(x: 100, y: 100, width: 40, height: 40))   UIView
    square.backgroundColor = UIColor.gray                                  UIView
    self.view.addSubview(square)                                           UIView

    self.animator = UIDynamicAnimator(referenceView: self.view)            .MyViewController: 0x7...

    let gravity = UIGravityBehavior(items: [square])                       <UIGravityBehavior: 0x...
    gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)                    <UIGravityBehavior: 0x...
```

Li...iew 〉 Untitled Page.xcplaygroundpage (Live View)

DYNAMIC BEHAVIORS

DYNAMIC BEHAVIOR

SUBTITLE

- Core unit of composition for a dynamic animation
  - Define the way items interact with animations
- Behaviors are
  - Declarative, describe “influences” on views
  - Added and removed at any time
  - Composable
  - Subclassable

Class

UIDynamicBehavior

An object that confers a behavioral configuration on one or more dynamic i
for their participation in 2D animation.

Overview

A *dynamic item* is any iOS or custom object that conforms to the `UIDynamicItem` proto
The `UIView` and `UICollectionViewLayoutAttributes` classes implement this prot
starting in iOS 7.0. You can implement this protocol to use a dynamic animator with custo
objects for such purposes as reacting to rotation or position changes computed by an ani
—an instance of the `UIDynamicAnimator` class.

This parent class, `UIDynamicBehavior`, is inherited by the primitive dynamic behavior
classes `UIAttachmentBehavior`, `UICollisionBehavior`, `UIGravityBehavior`,
`UIDynamicItemBehavior`, `UIPushBehavior`, and `UISnapBehavior`.

You can subclass `UIDynamicBehavior`. By using the `addChildBehavior(_:)` metho
an instance of this class or in a custom subclass, you can create composite behaviors of y
own design.

When you subclass `UIDynamicBehavior`, you typically need to provide one or more
initializers, along with other housekeeping methods such as those implemented in the iOS
primitive dynamic behaviors.

To perform per-step logic in a dynamic animation, provide a block object using the `actio`
property.

DYNAMIC BEHAVIOR

SUBTITLE

- Predefined behaviors
  - Gravity
  - Collisions
  - Attachments
  - Snap
  - Forces

Class

UIDynamicBehavior

An object that confers a behavioral configuration on one or more dynamic
for their participation in 2D animation.

Overview

A *dynamic item* is any iOS or custom object that conforms to the `UIDynamicItem` proto
The `UIView` and `UICollectionViewLayoutAttributes` classes implement this pro
starting in iOS 7.0. You can implement this protocol to use a dynamic animator with custo
objects for such purposes as reacting to rotation or position changes computed by an ar
—an instance of the `UIDynamicAnimator` class.

This parent class, `UIDynamicBehavior`, is inherited by the primitive dynamic behavior
classes `UIAttachmentBehavior`, `UICollisionBehavior`, `UIGravityBehavior`,
`UIDynamicItemBehavior`, `UIPushBehavior`, and `UISnapBehavior`.

You can subclass `UIDynamicBehavior`. By using the `addChildBehavior(_:)` meth
an instance of this class or in a custom subclass, you can create composite behaviors of
own design.

When you subclass `UIDynamicBehavior`, you typically need to provide one or more
initializers, along with other housekeeping methods such as those implemented in the iO
primitive dynamic behaviors.

To perform per-step logic in a dynamic animation, provide a block object using the `acti`
property.

DYNAMIC BEHAVIOR

```
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

Add gravity to an Animator

- Instantiate a UIDynamicBehavior
  - Initialize with view(s)
- Add behaviors to animator

```
myBehavior = [[MyBehavior alloc] initWith...];
[animator addBehavior:myBehavior];
```

DYNAMIC BEHAVIOR

- Add views to behaviors
- Remove views from behaviors
  - The influence stops when the behavior is removed
- Caveats
  - Possible to create impossible scenarios
  - Not physics-accurate

DYNAMIC BEHAVIORS

- UIGravityBehavior
  - A simple gravity vector
  - UIKit coordinate system
    - (0,1) by default
    - Items can be added and removed at any time
  - UIKit gravity
    - 1000 p/s^2

DYNAMIC BEHAVIORS

Add gravity to the animator

```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc]
                                initWithReferenceView:self.view];

UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc]
                                initWithItems:@[self.square1]];

[animator addBehavior:gravityBehavior];
```

- UIGravityBehavior

```
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

DYNAMIC BEHAVIORS

- UICollisionBehavior
  - Between a view and a boundary
  - Or between views associated to the same behavior
  - Or both, by default

```
@property (nonatomic, readwrite) UICollisionBehaviorMode collisionMode;
```

DYNAMIC BEHAVIORS

```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
```

```
UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc]
                                                        initWithItems:@[self.square1]];

// Creates collision boundaries from the bounds of the dynamic animator's
// reference view (self.view).
collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
collisionBehavior.collisionDelegate = self;
[animator addBehavior:collisionBehavior];
```

Delegate provides information about the collision

Set boundary based on reference view

- UICollisionBehavior

DYNAMIC BEHAVIORS

- UICollisionBehavior boundaries
  - Set automatically from frame

```
@property (nonatomic, readwrite) BOOL translatesReferenceBoundsIntoBoundary;
```

  - Optional inset

```
-(void)setTranslatesReferenceBoundsIntoBoundaryWithInsets:(UIEdgeInsets)insets;
```

inset

DYNAMIC BEHAVIORS

- Adding collision behavior

```
let square = UIView(frame: CGRect(x: 100, y: 100, width: 40, height: 40))
square.backgroundColor = UIColor.gray
self.view.addSubview(square)

// Instantiate the Animator
self.animator = UIDynamicAnimator(referenceView: self.view)

// Setup a behavior
let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
// Add the behavior to the animator
animator?.addBehavior(gravity)

// Setup up another behavior
let collision = UICollisionBehavior(items: [square])
collision.translatesReferenceBoundsIntoBoundary = true
// Add to the animator
animator?.addBehavior(collision)

// Create a new view for the boundary
let barrier = UIView(frame: CGRect(x: 0, y: 300, width: 110, height: 20))
barrier.backgroundColor = UIColor.red
self.view.addSubview(barrier)

let rightEdge = CGPoint(x: barrier.frame.origin.x + barrier.frame.size.width,
                        y: barrier.frame.origin.y)
collision.addBoundary(withIdentifier: "barrier" as NSCopying,
                      from: barrier.frame.origin, to: rightEdge)
```

DYNAMIC BEHAVIORS

```
c = [[UICollisionBehavior alloc] initWithItems:@[view];
[c addBoundaryWithIdentifier:@”Wall1”
                              fromPoint:p1 toPoint:p2];
[c addBoundaryWithIdentifier:@”Wall2”
                              fromPoint:p3 toPoint:p4];
```

- Explicitly with segments
  - - (void)addBoundaryWithIdentifier: (id)identifier fromPoint:(CGPoint)p1 toPoint: (CGPoint)p2;
- • Or paths (approximated)
  - - (void)addBoundaryWithIdentifier: (id)identifierforPath:(UIBezierPath*)p;

DYNAMIC BEHAVIORS

- UICollisionBehaviorDelegate
  - Between views
    - - collisionBehavior: beganContactForItem: withItem: atPoint:
    - - collisionBehavior: endedContactForItem: withItem:
  - • Or boundaries
    - - collisionBehavior: beganContactForItem: withBoundaryIdentifier: atPoint:
    - - collisionBehavior: endedContactForItem: withBoundaryIdentifier:
  - The reference boundary identifier is always nil
  - Conform to protocol <UICollisionBehaviorDelegate>

DYNAMIC BEHAVIORS

- Handling contacts

begin Contact

```
func collisionBehavior(_ behavior: UICollisionBehavior,
                       beganContactFor item: UIDynamicItem,
                       withBoundaryIdentifier identifier: NSCopying?,
                       at p: CGPoint) {
  print("Item: \n\t\(item) hit point: \n\t\(p)")

  // Cast the colliding item as a Box
  let collidingView = item as! Box

  // Highlight the collision
  collidingView.backgroundColor = UIColor.yellow
  UIView.animate(withDuration: 0.3, animations: {
    collidingView.backgroundColor = collidingView.color
  })

}
```

DYNAMIC BEHAVIORS

- Contact protocol methods

begin Contact

```
//| ----------------------------------------------------------------------
//  This method is called when square1 begins contacting a collision boundary.
//  In this demo, the only collision boundary is the bounds of the reference
//  view (self.view).
//
- (void)collisionBehavior:(UICollisionBehavior*)behavior beganContactForItem:
(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier atPoint:
(CGPoint)p
{
    // Lighten the tint color when the view is in contact with a boundary.
    [(UIView*)item setTintColor:[UIColor lightGrayColor]];
}
```

end Contact

```
//| ----------------------------------------------------------------------
//  This method is called when square1 stops contacting a collision boundary.
//  In this demo, the only collision boundary is the bounds of the reference
//  view (self.view).
//
- (void)collisionBehavior:(UICollisionBehavior*)behavior endedContactForItem:
(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier
{
    // Restore the default color when ending a contcact.
    [(UIView*)item setTintColor:[UIColor darkGrayColor]];
}
```

DYNAMIC BEHAVIORS

```
a1 = [[UIAttachmentBehavior alloc]
      initWithItem:v1 attachedToAnchor:ap];
a2 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 attachedToItem:v2];
```

- UIAttachmentBehavior
  - Between a view and an anchor point
  - Optional inset

DYNAMIC BEHAVIORS

SUBTITLE

- UIAttachmentBehavior
  - Offset

```
a1 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 point:p1
                attachedToAnchor:ap];
a2 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 point:p2
                attachedToItem:v2];
```

DYNAMIC BEHAVIORS

- UIAttachmentBehavior
  - An attachment can act as a spring
    - [a setFrequency:4.0];
    - [a setDamping:0.5];
  - An anchor point can be modified later
  - Attachments are invisible!

DYNAMIC BEHAVIORS

- UISnapBehavior
  - Snap a view to place
  - Ensure position and angle

```
s = [[UISnapBehavior alloc] initWithItem:v
                            snapToPoint:p];
[animator addBehavior:s];
```

DYNAMIC BEHAVIORS

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  if (snap != nil) {
    animator.removeBehavior(snap)
  }

  let touch = touches.first! as UITouch
  snap = UISnapBehavior(item: square, snapTo: touch.location(in: view))
  animator.addBehavior(snap)
}
```

- UISnapBehavior
  - Snap move view to a touch

DYNAMIC BEHAVIORS

- UIPushBehavior
  - Snap a view to place
  - Ensure position and angle
  - Damping is customizable
- A simple force vector
  - @property (readwrite,nonatomic) CGFloat xComponent;
  - @property (readwrite,nonatomic) CGFloat yComponent;
  - @property (readwrite,nonatomic) CGFloat angle;
  - @property (readwrite,nonatomic) CGFloat magnitude;

DYNAMIC BEHAVIORS

UIPushModeContinuous

- UIPushBehavior
  - Target point can be customizable

```
p = [[UIPushBehavior alloc]
                initWithItems:@[view]
                mode:UIPushBehaviorModeContinuous];
 [p setTargetPoint:x forItem:view];
```

DYNAMIC BEHAVIORS

- UIPushBehavior
  - UIKit newton 100 p/s2
  - Instantaneous mode
  - Velocity change is instantaneous

UIPushModeInstantaneous

```
p = [[UIPushBehavior alloc]
                initWithItems:@[view]
                mode:UIPushBehaviorModeInstantaneous];
[p setActive:TRUE]
```

DYNAMIC ITEM BEHAVIOR

DYNAMIC ITEM BEHAVIOR

- UIDynamicItemProtocol
  - Applied to one or many items
  - Change item-level properties
    - friction
    - resistance
    - angularResistance
    - elasticity
    - density
    - allowsRotation
  - Directly add angular or linear velocities
    - i.e. map with a previous gesture

DYNAMIC ITEM BEHAVIOR

- A protocol for items associated to predefined behaviors
- Describe what UIKit needs to animate an item
- UIView implements it
- You can implement it

DYNAMIC BEHAVIOR

- UIDynamicItem
  - A protocol for items associated to predefined behaviors
  - Describe what UIKit needs to animate an item
  - UIView implements it
  - You can implement it

```
@protocol UIDynamicItem <NSObject>
@property (nonatomic, readwrite) CGPoint center;
@property (nonatomic, readonly)  CGRect bounds;
@property (nonatomic, readwrite) CGAffineTransform transform;
@end
```

ONLY 2D

DYNAMIC ITEM BEHAVIOR

```
behavior.action =  ^{
    NSLog(@"%@, %@",
          NSStringFromCGAffineTransform(square.transform),
          NSStringFromCGPoint(square.center));
};
```

- What else
  - Each dynamic behavior has an action property where you supply a block to be executed with every step of the animation

DYNAMIC ITEM BEHAVIOR

- Put them all together to have custom behavior and effects for your views
- Don't forget about SpriteKit

DEMO PROJECT UIKITDYNAMICS

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2C

© 2020 T.A. Binkowski

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2D

© 2020 T.A. Binkowski

VIEW PROPERTY ANIMATOR

VIEW PROPERTY ANIMATOR

- Multiple ways to animate views
  - Begin/commit
  - Block based API (iOS4+)
  - Property animators (iOS10+)

VIEW PROPERTY ANIMATOR

UIVIEW ANIMATIONS

- Built-in animatable properties on UIView
- Different variations for customization, callbacks, timing, etc.

```
// Add a rotation transform to our red box view
UIView.animate(withDuration: 3.0, animations: { () -> Void in
  let rotationTransform = CGAffineTransform(rotationAngle: 3.14)
  redView.transform = rotationTransform
})

// Fade out the green
UIView.animate(withDuration: 2) {
  greenView.alpha = 0
}
```

VIEW PROPERTY ANIMATOR

UIVIEW ANIMATIONS

- Pros
  - Easy to use
  - Customizable to an extent
- Cons
  - Limited to predefined timing curves (fadeinout, etc.)
  - No way to cancel/interupt animations
    - Dirty solution: Create an animation to replace the current one

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

- New API that is highly extensible (iOS10+)
- Can behave the same as the old UIViewAnimations
- Provides fine grained control over the animations
  - Pause, restart, modify, cancel animations

UIViewPropertyAnimator

A class that animates changes to views and allows the dynamic modification of those animations.

Overview

A `UIViewPropertyAnimator` object lets you animate changes to views and dynamically modify your animations before they finish. With a property animator, you can run your animations from start to finish normally or you can turn them into interactive animations and control the timing yourself. The animator operates on animatable properties of views, such as the `frame`, `center`, `alpha`, and `transform` properties, creating the needed animations from the blocks you provide.

When creating a property animator object, you specify the following:

- A block containing code that modifies the properties of one or more views.
- The timing curve that defines the speed of the animation over the course of its run

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

API can be similar to UIViewAnimation

```
/// Move and Fade (similiar to UIViewAnimation)
UIViewPropertyAnimator(duration: 1, curve: .easeInOut) {
  heart.alpha = 0
  heart.center = heart.superview!.center
}.startAnimation()
```

Animations start in inactive state

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

- Property animator gives you programmatic control over the timing and execution of the animations
  - Start, pause, resume, and stop animations
  - Add animation blocks after the original animations start
  - Scrub through a paused animation by modifying the `fractionComplete` property
  - Change the animation's direction using the `isReversed` property.
  - Modify the timing and duration of a partially complete animation

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

SUBTITLE

- Animations are additive

- Add animation blocks after the original animations start using the addAnimations(_:) and addAnimations(_:delayFactor:) methods.

```
/// Alternative call
let animator = UIViewPropertyAnimator(duration: 3,
                                      curve: .easeInOut)
// Add animation block
animator.addAnimations {
  heart.alpha = 0
}

// Now here goes our second
animator.addAnimations {
  heart.center = heart.superview!.center
}
animator.startAnimation()
```

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

- Add a completion block
- Monitor the state

```
/// Alternative call
let animator = UIViewPropertyAnimator(duration: 3,
                                      curve: .easeInOut)
// Add animation block
animator.addAnimations {
  heart.alpha = 0
}

// Now here goes our second
animator.addAnimations {
  heart.center = heart.superview!.center
}
animator.startAnimation()

/// Add completion block
animator.addCompletion { (position) in
  switch position {
  case .end: print("Completion handler called at end of animation")
  case .current: print("Completion handler called mid-way through animation")
  case .start: print("Completion handler called  at start of animation")
  }
}
```

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

- Additional properties to monitor or change the animation

fractionComplete—This property sets or returns the animation's completion percentage.

isReversed—This property sets or returns a Boolean value that determines if the animation is running in reverse.

state—This property returns a value that determines the current state of the animation. It is an enumeration of type **UIViewAnimatingState** with the values **inactive**, **active**, and **stopped**.

isRunning—This property returns a Boolean value that determines whether the animation is running or not.

startAnimation()—This method starts the animation.

pauseAnimation()—This method pauses the animation.

stopAnimation(Bool**)**—This method stops the animation. The attribute indicates whether the animator will be deactivated or it will stay active for further actions.

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

```
/// Reverse the animation
let button = UIButton(frame: CGRect(origin: .zero, size: CGSize(width: 100,
    height: 30)))
button.setTitle("Reverse", for: .normal)
button.setTitleColor(.black, for: .normal)
button.setTitleColor(.gray, for: .highlighted)
let listener = EventListener()
listener.eventFired = {
  animator.isReversed = true
}
```

Reverse the animation.
Callback at start.

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

```
/// Start/stop the animation
let button = UIButton(frame: CGRect(origin: .zero, size: CGSize(width: 100, height: 30)))
button.setTitle("Pause", for: .normal)
button.setTitleColor(.black, for: .normal)
button.setTitleColor(.gray, for: .highlighted)

let listener = EventListener()
listener.eventFired = {
  if animator.isRunning {
    animator.pauseAnimation()
  } else {
    animator.startAnimation()
  }
}
```

Check state

Pause/restart

VIEW PROPERTY ANIMATOR

UIVIEWPROPERTYANIMATOR

- Additional properties to monitor or change the animation

fractionComplete—This property sets or returns the animation's completion percentage.

isReversed—This property sets or returns a Boolean value that determines if the animation is running in reverse.

state—This property returns a value that determines the current state of the animation. It is an enumeration of type **UIViewAnimatingState** with the values **inactive**, **active**, and **stopped**.

isRunning—This property returns a Boolean value that determines whether the animation is running or not.

startAnimation()—This method starts the animation.

pauseAnimation()—This method pauses the animation.

stopAnimation(Bool**)**—This method stops the animation. The attribute indicates whether the animator will be deactivated or it will stay active for further actions.

VIEW PROPERTY ANIMATOR

- Nothing wrong with UIView animations
- In our book
  - Might pause/cancel animations when the page is being turned
  - Mini-game

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2D

© 2020 T.A. Binkowski

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2E

© 2020 T.A. Binkowski

PARENTAL GATE

PARENTAL GATE

SUBTITLE

- Features are intended for parents, not kids
  - In-app purchase
  - Signup for news
  - Social Media
- Prevent kids from accessing inappropriate content

PARENTAL GATE

Building Apps for Kids

You can help kids learn, grow, and have fun with games, interactive stories, educational materials, and more. Find out how to create kids' apps that are safe and trusted by parents.

PARENTAL GATE

PARENTAL GATE

- Draw with a finger on the screen to validate
  - Circle
  - Check mark
  - Figure 8
  - ...

PARENTAL GATE

- Track and save touches
  - Began
  - Moved
  - End

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Began")
  drawing = false
  if let touch = touches.first {
    lastPoint = touch.location(in: self.view)
  }
}

override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Moved")

  drawing = true
  if let touch = touches.first {
    let currentPoint = touch.location(in: view)
    drawLineFrom(fromPoint: lastPoint, toPoint: currentPoint)


    lastPoint = currentPoint
    print(currentPoint)
  }
}

override  func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Ended")
  if !drawing {
    drawLineFrom(fromPoint: lastPoint, toPoint: lastPoint)
  }
  tempImageView.image = nil
}
```

PARENTAL GATE

- When touch point moves, connect the existing line segment to the new point

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Began")
  drawing = false
  if let touch = touches.first {
    lastPoint = touch.location(in: self.view)
  }
}

override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Moved")

  drawing = true
  if let touch = touches.first {
    let currentPoint = touch.location(in: view)
    drawLineFrom(fromPoint: lastPoint, toPoint: currentPoint)


    lastPoint = currentPoint
    print(currentPoint)
  }
}

override  func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  print("Ended")
  if !drawing {
    drawLineFrom(fromPoint: lastPoint, toPoint: lastPoint)
  }
  tempImageView.image = nil
}
```

PARENTAL GATE

- Draw small line segments with CG

```
/// Draw a line segment
func drawLineFrom(fromPoint: CGPoint, toPoint: CGPoint) {

  // Create the canvas
  UIGraphicsBeginImageContext(view.frame.size)
  let context = UIGraphicsGetCurrentContext()
  tempImageView.image?.draw(in: CGRect(x: 0, y: 0,
                                       width: view.frame.size.width,
                                       height: view.frame.size.height))

  // Create line segment
  context!.move(to: CGPoint(x: fromPoint.x, y: fromPoint.y))
  context!.addLine(to: CGPoint(x: toPoint.x, y: toPoint.y))

  // Set the `pen`
  context!.setLineCap(.round)
  context!.setLineWidth(brushWidth)
  context!.setStrokeColor(UIColor.black.cgColor)
  context!.setBlendMode(.normal)

  // Stroke the path with the pen
  context!.strokePath()

  // Copy the canvas on the imageview
  tempImageView.image = UIGraphicsGetImageFromCurrentImageContext()
  tempImageView.alpha = opacity
  UIGraphicsEndImageContext()

}
```

PARENTAL GATE

- Draw small line segments with CG

```
/// Draw a line segment
func drawLineFrom(fromPoint: CGPoint, toPoint: CGPoint) {

  // Create the canvas
  UIGraphicsBeginImageContext(view.frame.size)
  let context = UIGraphicsGetCurrentContext()
  tempImageView.image?.draw(in: CGRect(x: 0, y: 0,
                                       width: view.frame.size.width,
                                       height: view.frame.size.height))

  // Create line segment
  context!.move(to: CGPoint(x: fromPoint.x, y: fromPoint.y))
  context!.addLine(to: CGPoint(x: toPoint.x, y: toPoint.y))

  // Set the `pen`
  context!.setLineCap(.round)
  context!.setLineWidth(brushWidth)
  context!.setStrokeColor(UIColor.black.cgColor)
  context!.setBlendMode(.normal)

  // Stroke the path with the pen
  context!.strokePath()

  // Copy the canvas on the imageview
  tempImageView.image = UIGraphicsGetImageFromCurrentImageContext()
  tempImageView.alpha = opacity
  UIGraphicsEndImageContext()

}
```

PARENTAL GATE

- This implementation is limited
  - Single stroke
  - Raw point capture
- It collects all the points for you

PARENTAL GATE

Distance from point to center will them same in a perfect circle

A hand drawn one should be close

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2E

© 2020 T.A. Binkowski

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2F

© 2020 T.A. Binkowski

APP PREVIEWS

APP VIDEOS

- Up to 3, 30 second videos of app
- Autoplay in App Store
  - Without sound

App Store

Overview Developer Insights Planning Guidelines Submi

Show More with App Previews

Engage users with short videos of your iOS or tvOS app in action, directly on the App Store. Since app previews autoplay on the App Store on iOS 11, they're key in helping users discover and learn about your app.

APP VIDEOS

- Can overlay sound, text, voice overs
- Can not show users interacting with the application

App Store

Overview Developer Insights Planning Guidelines Submi

Show More with App Previews

Engage users with short videos of your iOS or tvOS app in action, directly on the App Store. Since app previews autoplay on the App Store on iOS 11, they're key in helping users discover and learn about your app.

APP VIDEOS

- Creating Great App Previews
- https://developer.apple.com/videos/play/design/11/

Overview

Creating Great App Previews

Spark interest in your app by showing a compelling video preview highlighting the magic moments that make your app unique. Learn how to plan, create and

APP VIDEOS

- Introducing the New App Store
- https://developer.apple.com/videos/play/wwdc2017/301/

APP VIDEOS

- iOS11+ Screen recorder
- https://support.apple.com/en-us/HT207935

How to record the screen on your iPhone, iPad, or iPod touch

In iOS 11, you can create a screen recording and capture sound on your iPhone, iPad, or iPod touch.

Record your screen

1. Add Screen Recording to Control Center. Go to Settings > Control Center > Customize Controls, then tap + next to Screen Recording.
2. Open Control Center.
3. Tap ◉, then wait for the three-second countdown. To capture sound while you record, press deeply on ◉ and tap Microphone Audio.
4. To stop recording, open Control Center and tap ●. Or tap the red status bar at the top of your screen and tap Stop.

APP VIDEOS

- Quicktime recording from connected device

APP VIDEOS

- Hidden Xcode command line tool to get video from simulator

Not officially supported, but works

RSS Feed
Contact Me

Recording iOS Simulator Video

At the latest meeting of the Redmond Xcoders, there was a talk by Lori Hill about creating App Preview videos. During the Q&A period after her talk, questions were raised about recording such a video from the iOS Simulator rather than an actual device.

The first few comments were about doing an actual screen capture of the simulator, and recording the simulator *directly* was an option it seemed most people didn't know about. I've attempted to find out what I can about the subject and put it somewhere that might be helpful to Google users in the future.

Disclaimer: It is unclear whether these videos can be used for App Preview. Some differences between device recordings and simulator recordings are discussed below. Apple's App Preview documentation has no real stance on the matter; while they discuss how to record with a physical device, they don't explicitly state that a device *must* be used. If you attempt this and find out one way or another, please e-mail me and I will update this post.

Video from the Simulator can be recorded using the `simctl` Xcode tool, like so:

```
xcrun simctl io <device> recordVideo <path>
```

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2F

© 2020 T.A. Binkowski

ASSIGNMENT 2

POP QUIZ

What do John Travollta, Barack Obama, Shaquille O'Neil, Whoopi Goldberg, Glenn Beck, Jay Leno, Cheech Marin, Jerry Seinfeld, Sting, Queen Latifah, Jimmy Buffett, Madonna, Spike Lee, Jerry Garcia, Paul McCartney, Jamie Lee Curtis, Bob Dylan, The Dutchess of York and Steve Martin have in common?

ASSIGNMENT 1

IF A CELEBRITY CAN DO IT, SO CAN YOU

SUBTITLE

- Create a children's book using `UIPageViewController
- A hilarious, heartwarming and/or educational tale
  - At least 6 pages of unique content
- Must be better than "Shaq and Beanstalk"

ASSIGNMENT 1

Settings

About the author

ASSIGNMENT 1

Page forward

Page Backward

interactive element

ASSIGNMENT 1

ASSIGNMENT 1

Tap this button to return to the

Tap this button to have the boo

ASSIGNMENT 1

OVERALL REQUIREMENTS

OVERALL REQUIREMENTS

- “Home Page” that represents the beginning of the book
- “Settings page” for “Read to Me” options (objective-c)
  - Automatically play on each page
  - Tap-to-play
  - Store these preferences in `UserDefaults`
- “About the Author” page protected by Parental Gate (external .xib file)
- Store currently reading page so that book opens to last view paged
- Create an App video of your book

BOOK PAGES

- Every content page must implement:
  - “Read to me” functionality
    - Record sound using Audacity, Garage Band, iPhone and email to self (UITextKit,AVSpeechSynthesis, next week)
    - Words should highlight as they are read
    - Start automatically or by button tap (depending on user preference in settings)
  - Picture and text
    - You must use `UIDynamics` for on-screen element (next week)
    - `UIGestureRecognizer` of on-screen element
    - UIViewPropertyAnimator based animations
    - Tap to hear sound effects
  - Button to return to home page

PARENTAL GATE

SUBTITLE

- Features are intended for parents, not kids
  - In-app purchase
  - Signup for news
  - Social Media
- Prevent kids from accessing inappropriate content

PARENTAL GATE

- Implementation used to be a best practice, now it is required

```
— 24. Kids Apps
  • 24.3- Apps primarily intended for use by
 kids under 13 must get parental permission
 or use a parental gate before allowing the
 user to link out of the app or engage in
 commerce
 This requires gating when linking out of the
 app or engaging in commerce such as selling
 in-app items.
```

PARENTAL GATE

PARENTAL GATE

PARENTAL GATE

REQUIREMENTS

- Implement a Parental Gate for your book app
  - Triggered when the user tries to view the “About the Author” screen
  - In practice the credits screen would typically have a link to social media
- Custom finger drawing view for validator
- Needs to use custom error handling do-try-throw
  - Custom ErrorType

SETTINGS PAGE

REQUIREMENTS

- Settings page in objective-c 😳
- Allow user to select preference to automatically read the page
  - Default should be "on"

BOOK PAGE OPTIONS

SUBTITLE

- Implementations options
  - View controller per page
  - Create view controller templates
    - Design 3 different styles of pages
    - Each page is a dictionary that includes images paths, text, sound file paths, animation start/stop frames

DUE DATE

- Due April 16th at 5:29 PM
  - 2 Weeks
  - No late homework accepted
- “It’s all about story…story, story, story.” ~John Lasseter

RECOMMENDED WORK FOR THIS WEEK

- Page View Controller
  - Get basic layout
  - Back to home button
  - Save current page
  - How to handle interrupted page turns
- Book pages
  - Draw them out
  - Sound effects
  - Interactive element
  - Text-to-speech

GOING OVER NEXT WEEK

- Settings screen
  - Obj-c first app from Apple
  - Obj-c bridging header
- Parental gate
- App Preview
- Picture and text
  - `UIDynamics` for on-screen element
  - UIViewPropertyAnimator

(JUST ENOUGH) OBJECTIVE-C

HISTORY

- Developed by Brad Cox and Tom Love in 1980s
  - Influenced by Smalltalk
  - Focus on reusability
  - Backwards compatible with C
- Licensed by NeXT
  - NSObject, NSString
- Apple acquired NeXT

COMPONENTS OF OBJECTIVE-C

- Programming language
- A library of objects
- Runtime environment
  - Defers many operations from compile time to run
  - Executes the compiled code

- Popularity by TIOBE Programming Community Index
  - 2007 - 45th
  - 2011 - 6th
  - 2012 - 4th
  - 2014 - 3rd
  - 2015 - Something called Swift appeared

OBJECTIVE-C

- “Objective-C provides the basic construct for encapsulating some data with the actions that operate on that data.”
- Evolution made it simpler, safer though automation
  - NEXT
    - Object-oriented C
    - Retain and Release
  - Apple:
    - Properties
    - Fast Enumeration
    - Blocks
    - ARC (automated memory management)

A STRICT-SUPERSET OF ANSI C

- Include header files using #import instead of #include
- Objective-C add the following syntax and features to ANSI C:
  - Definition of new classes
  - Class and instance methods
  - Method invocation (called messaging)
  - Declaration of properties (synthesizing of accessor methods)
  - Static and dynamic typing
  - Blocks—encapsulated segments of code
  - Extensions to the base language such as protocols and categories

OBJECT

- Object is a runtime instance of a class
  - Its own in-memory copy of the instance variables
  - Pointers to methods of the class
- Everything is an object
  - Interface objects (e.g. UIView, UIButton)
  - Data objects (e.g. NSDictionary, NSSets)
- Developer is responsible for memory management
  - Reference counting
  - Compiler makes it automatic
  - Automatic Reference Counting (ARC)

CLASSES

- Methods
  - Associates data with an operation that uses it (functions)
- Instance
  - Allocation of a class
- Instance variables (“ivar”)
  - An object’s data

image via Apple, Inc.

CLASS DECLARATION

- Class specification:
  - .h - header files
  - .m - implementation files
  - .mm - source files containing C++
- Example MyClass (a subclass of of NSObject)

image via Apple, Inc.

TYPING

- Supports strong and weak typing

```
MyClass *aStrongObject      // S
id      aWeakObject         // W
```

- Strong:
  - Include class name in type declaration
  - Checked by compiler
- Weak:
  - Use id for the object
  - Increase flexibility and dynamism

```
- (IBAction)sliderMoved:(id)sender
```

METHODS

SUBTITLE

- Method types:
  - Instance methods “-” scoped to instance of a class
  - Class methods “+” directly to class
- Method declaration:

image via Apple, Inc.

Objective-C Messaging

- "Messaging an object" is a call to a method on an object
- Messaging expression

```
[receiver messageSignatures:parameter];
[myArray insertObject:anObject atIndex:0];
```

- Nesting

```
[[myAppObject theArray]
         insertObject:[myAppObject objectToInsert]
              atIndex:0];
```

Objective-C Messaging

image via Apple, Inc.

Objective-C Declared Properties

- Property is data encapsulated or stored by an object
  - Attributes (e.g. label.text, view.backgroundColor)
- Class of the object defines an interface to set and get the values of the property
  - Property Name - color
  - Setter - setColor
  - Getter - color
- Declare a

```
@property (strong, nonatomic) UIView *myView;
```

Objective-C Dot Syntax

- Invoking accessor methods (i.e. getters and setters)

```
[[object theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
[object.theArray insertObject:myAppObject.objectToInsert atIndex:0];
```

dot

dot

- Use for a

```
myAppObject.theArray = aNewArray;
[myAppObject setTheArray:aNewArray];
```

Objective-C Blocks

- Objects th
  - Anonyr
  - Passed

```
int result = myBlock(4); // result is 28
```

Objective-C Blocks

- Scoped locally and variables can be read-only or read-write
- Used throughout UIKit
  - Callbacks
  - Animations
  - Multi-threading
  - Para

```
- (id)addObserverForName:(NSString *)name
                  object:(id)obj
                   queue:(NSOperationQueue *)queue
              usingBlock:(void (^)(NSNotification *note))block
```

Objective-C Protocols

- Protocol declares methods that can be implemented by any class
- Define the interface for another class to implement ("conform")

```
@interface ViewController : UIViewController <UITextFieldDelegate>
```

- Custom classes can define their own protocols

```
@protocol MyProtocol
- (void)myProtocolMethod;
@end
```

OBJECTIVE-C CHEAT SHEET

OBJECTIVE-C

```
//
//  MyClass.h
//  Objective-C-hristmas
//

#import <Foundation/Foundation.h>

@interface MyClass : NSObject
{
    // Declare instance variables (optional)
}

// Define properties
// Define methods

@end
```

Objective-C

```
//
//  MyClass.m
//  Objective-C-hristmas
//

#import "MyClass.h"

@interface MyClass ()
// Define private properties
// Define private methods
@end

@implementation MyClass
// Implement methods
@end
```

Objective-C

```
// Defining methods
- (void)message;
- (void)message:(id)parameter;
- (BOOL)message:(id)parameter1 anotherMessage:(id)parameter2;


// Implementing methods
- (void)message {
     // Do some stuff
}

- (BOOL)message:(id)parameter {
    // Do some stuff with parameter
    return YES;
}
```

Objective-C

```
// Defining Properties
@property (attribute1, attribute2) propertyName;

// Synthesizing Properties
_propertyName   // Instance variable created
propertyName    // Accessor getter method
setPropertyName // Accessor setter method

// Dot syntax for getter
value = [MyClass propertyName];
value = MyClass.propertyName;

// Dot syntax for setter
[myClass setPropertyName:value];
myClass.propertyName = value;
```

Objective-C

```
// Creating a class instance
MyClass *classObject = [[MyClass alloc] init];


// Creating a custom init method
- (id) initWithParameter:(id)parameter {
     if ( self = [super init]) {
         self.someProperty = parameter
   }
    return self;
}
```

NSObject declares init prototype

Objective-C

```
// Messaging - invoking methods on an object

// Fancy objective-c terms
[receiver message];

// Message expression
[myClass method];

// Pass an argument as part of the message
[myClass message:argument];

// Pass two argument2 as part of the message
[myClass message:argument anotherMessage:anotherArgument];

// Nesting Methods
id output = [receiver message];
[myClass message:output];
[myClass message:[object method]];
```

Objective-C

```
// Declare a custom protocol
@protocol MyProtocol
@required
- (void)myProtocolMethod;
@optional
- (void)anotherMethod;
@end
```

```
// .h
@interface MyClass : NSObject <MyProtocol>
@end
```

```
// .m
@implementation MyClass
- (void)myProtocolMethod  {
   // Do some stuff
{
@end
```

CREATE A CUSTOM OBJECTIVE-C CLASS

CUSTOM CLASSES

OBJECTIVE-C PLAYGROUND

OBJECTIVE-C PLAYGROUND

OBJECTIVE-C PLAYGROUND

OBJECTIVE-C PLAYGROUND

```
3   //  It's A Zoo
4   //
5   //  Created by T. Andrew Binkowski on 1/12/15.
6   //  Copyright (c) 2015 The University of Chicago. All rights reserved.
7   //
8
9   #import "Animal.h"
10
11  @implementation Animal
12
13
14  - (NSString*)describeAnimal
15  {
16      NSString *descriptionString = [NSString stringWithFormat:@"Animal Name:%@ Species:%@ Age:%@ \n",
17                                     self.name,self.species,self.age];
18      return descriptionString;
19  }
20
21  @end
22
```

OBJECTIVE-C PLAYGROUND

```
2   //  Animal.m
3   //  It's A Zoo
4   //
5   //  Created by T. Andrew Binkowski on 1/12/15.
6   //  Copyright (c) 2015 The University of Chicago. All rights reserved.
7   //
8
9   #import "Animal.h"
10
11  @implementation Animal
12
13
14  - (NSString*)describeAnimal
15  {
```

Declaration	- (NSString *)describeAnimal;
Description	Return a string that describes the animal traits
Returns	An NSString with animal desscription
Declared In	Animal.h

```
withFormat:@"Animal Name:%@ Species:%@ Age:%@ \n",
species,self.age];
```

```
21  @end
22
```

Option-Click shows information about method if using HeaderDoc syntax

OBJECTIVE-C PLAYGROUND

- Images.xcassets
- LaunchScreen.xib
- Supporting Files
- It's A ZooTests
- Products

```
12  @interface ViewController ()
13
14  @end
15
16  @implementation ViewController
17
18  - (void)viewDidLoad {
19      [super viewDidLoad];
20      // Do any additional setup after loading the view, typically from a nib.
21  }
22
23
24  - (void)viewWillAppear:(BOOL)animated
25  {
26      // Create new dog object and fill it with data
27      Animal *dog = [[Animal alloc] init];
28      dog.name = @"Max";
29      dog.species = @"Great Dane";
30      dog.age = @1;
31      dog.image = [UIImage imageNamed:@"ImageInAssetCatalog"];
32
33      NSLog(@"Dog Description:%@",[dog describeAnimal]);
34  }
35
36  - (void)didReceiveMemoryWarning {
37      [super didReceiveMemoryWarning];
38      // Dispose of any resources that can be recreated.
39  }
40
```

import class

create new class

call our method;
message our object

OBJECTIVE-C PLAYGROUND

OBJECTIVE-C

MODERN OBJECTIVE-C

Modern Objective-C

- Programming language
- A library of objects
- Runtime environment
  - Defers many operations from compile time to runtime
  - Executes the compiled code
- Popularity by TIOBE Programming Community Index
  - 2007 - 45th
  - 2011 - 6th
  - 2012 - 4th
  - 2014 - 4th

Modern Objective-C

- Based on Smalltalk
  - Messaging an object instead of ‘calling’ a function
- In messaging
  - The runtime decides which code gets executed
  - Performs a ‘lookup’ to identify method by name
  - The compiler does not check the type of object being messaged
    - Dynamic binding at runtime

Code can compile without warnings/errors and crash.

Modern Objective-C

- A strict-superset of ANSI C
  - Use C code interspersed with objective-C code (file)
    - Include header files using #import instead of #include
  - Structs are used in many frameworks (for efficiency)
    - There is overhead to creating objects
    - Structs can only hold non object types
      - int, float, double, char, etc.

```
struct CGRect {
  CGPoint origin;
  CGSize size;
};
typedef struct CGRect CGRect;
```

```
;
0f;
.y = 10.0f;
idth = 100.0f;
eight = 150.0f
```

Modern Objective-C

- Objective-C add the following syntax and features to ANSI C:
  - Definition of new classes
  - Class and instance methods
  - Method invocation (called messaging)
  - Declaration of properties (synthesizing of accessor methods)
  - Static and dynamic typing
  - Blocks—encapsulated segments of code
  - Extensions to the base language such as protocols and categories

Modern Objective-C

- Objective-C 2.0 (2006)
- Simpler, safer though automation
  - NEXT added
    - Object-oriented C
    - Retain and Release memory management model
    - Interface library and project builder
  - Apple added
    - Properties
    - Fast Enumeration
    - Blocks
    - Literal syntax
    - ARC (automated memory management)

Modern Objective-C

- Many of the 2.0 features are strictly compiler automation
  - The underlying language/runtime features are the same

OBJECTIVE-C TYPES

OBJECTIVE-C TYPES

- id
  - General type or class objects)
  - Pointer to an

```
typedef struct objc_object {
    Class isa;
} *id;
```

isa variable of what class it
"isa" n instance of

OBJECTIVE-C TYPES

- "isa" variable allows objects to perform introspection
  - `isKindOfClass:` - includes subclasses
  - `isMemberOfClass:` - specific to a class (no subclasses)
  - `respondsToSele` what class it belongs to (polymorph

```
if ( [anObject isKindOfClass:someClass] )
...

if ( [anObject isMemberOfClass:someClass] )
…
```

OBJECTIVE-C TYPES

- id is the generic type for dynamic binding at runtime
  - Can be cast to a pointer (*)
  - Does not know (or care) what objects is until it needs it
  - Warning: Compiler does not check
- Use case examples
  - Identify which button is pressed in a group of buttons
  - Iterate through an NSArray of different objects
- Advantages of statically type objects (why not to use id)
  - Compile-time checking
  - Autocompletion in Xcode

OBJECTIVE-C TYPES

- SEL (selec e
- @selector

```
// Test if a class implements a method
[object respondsToSelector:@selector(methodName)]

// Store a method and call on object
SEL method = @selector(methodName);
[object performSelector:method];

// Set method to call for target-action
[aButton addTarget:self action:@selector(eatCake)
              forControlEvents:UIControlEventTouchDown];
```

OBJECTIVE-C TYPES

- @selecto

```
if ([aObject respondsToSelector:@selector(myMethodName)])
{
    ...
}
```

Does aObject implement a method named ‘myMethodName’?

```
[button addTarget:self action:@selector(digitPressed:)]
```

When button is tapped, execute method named ‘digitPressed’ that is implemented in this class (self)

OBJECTIVE-C TYPES

- nil object pointer to nothing
- Example: Test if it points to an object

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

- Instance variables can be set to nil

```
myObject.delegate = nil;
```

OBJECTIVE-C TYPES

- It is safe to send a message to nil
  - Invoking a me
  - Warning!!! W

```
myObject = nil;
[myObject doSomething];
```

OBJECTIVE-C TYPES

- Sending messages to nil simplifies expressions
  - Limits
  - Use as

```
// For example, this expression...
if (name != nil && [name isEqualToString:@"Steve"]) { ... }

// ...can be simplified to:
if ([name isEqualToString:@"steve"]) { ... }
```

It was so much work to do this…

OBJECTIVE-C TYPES

- Values rep er should know abou

Symbol	Value	Meaning
NULL	(void *)0	literal null value for C pointers
nil	(id)0	literal null value for Objective-C objects
Nil	(Class)0	literal null value for Objective-C classes
NSNull	[NSNull null]	singleton object used to represent null

Collections can not hold nil, but can hold NSNull object

OBJECTIVE-C TYPES

http://nshipster.com/

nil / Nil / NULL / NSNull

Written by Mattt Thompson on Jan 7th, 2013

Understanding the concept of nothingness is as much a philosophical issue as it is a pragmatic one. We are inhabitants of a universe of *somethings*, yet reason in a logical universe of existential uncertainties. As a physical manifestation of a logical system, computers are faced with the intractable problem of how to represent *nothing* with *something*.

In Objective-C, there are several different varieties of *nothing*. The reason for this goes back to a common NSHipster refrain, of how Objective-C bridges the procedural paradigm of C with Smalltalk-inspired object-oriented paradigm.

C represents *nothing* as `0` for primitive values, and `NULL` for pointers (which is equivalent to `0` in a pointer context).

OBJECTIVE-C TYPES

- BOOL types to encode 'truth'
  - typedef of a signed char, with the macros YES and NO
  - YES = #define 1
  - NO = #define 0

```
// Test for true
if (myObject.value == YES) {...}
if (myObject.value) {...}

// Test for false
if (myObject.value == NO) {...}
if (!myObject.value) {...}

// Set BOOL type variable
myObject.updated = NO;
myObject.updated = YES;
```

OBJECTIVE-C TYPES

- Truth type

Name	Typedef	Header	True Value	False Value
BOOL	signed char	objc.h	YES	NO
bool	_Bool(int)	stdbool.h	true	false
Boolean	unsigned char	MacTypes.h	TRUE	FALSE
NSNumber	__NSCFBoolean	Foundation.h	@(YES)	@(NO)
CFBooleanRef	struct	CoreFoundation.h	kCFBooleanTrue	kCFBooleanFalse

OBJECTIVE-C
OBJECT MESSAGING

OBJECTIVE-C OBJECT MESSAGING

- Send a message (method) to an object (receiver)

```
[receiver message];

[Car start];
[Car alloc] init];
```

- When a message is sent, the runtime system selects method from the receiver
  - Method names in message are “selectors”

OBJECTIVE-C OBJECT MESSAGING

message expression

message

```
[receiver method:argument];
```

selector

OBJECTIVE-C OBJECT MESSAGING

- Method parameters (arguments)
  - Not optional
  - Order must be preserved

```
[myObject setName:@"Jane"];
```

```
[myObject setName:@"Jane"
          andDate:[NSDate now]
         andEmail:jd@mail.com];
```

OBJECTIVE-C OBJECT MESSAGING

- Polymorphism
  - Objects can respond differently to the same message
  - Example: send a `description` message to an `id` variable

```
[personObject description];
```

```
[carObject description];
```

NSObject provides 'description' to convert object to string. It varies depending on object. Good method to override in custom classes.

OBJECTIVE-C OBJECT MESSAGING

- Dot Syntax
  - Alternative t
  - Compiler tra

```
myInstance.value = 10;
[myInstance setValue:10];


self.age = 10      // Access object's ivar
                   // using accessor methods
[self setAge:10]; // Equivalent to above
age = 10;          // Access ivar directly
```

OBJECTIVE-C OBJECT MESSAGING

- Method types:
  - Class metho
  - Instance me

```
[[myClass alloc] init];
```

class method

instance method

OBJECTIVE-C OBJECT MESSAGING

- Method de

image via Apple, Inc.

OBJECTIVE-C OBJECT MESSAGING

- Class methods
  - Used f
  - No ac

```
// Allocate memory for the object (always pair w/init)
+ (id)alloc;

// returns the one and only, shared (singleton) instance
+ (id)sharedObject;
```

OBJECTIVE-C OBJECT MESSAGING

- Instance methods
  - Most common methods to implement
  - Access variables as if they were local
  - Can call met

```
// Setter method
-(void)setString:(NSString *)newString
{
...
}
```

OBJECTIVE-C CLASSES

OBJECTIVE-C CLASSES

- Classes declare state and behavior
  - State is maintained using instance variables
  - Behavior is implemented using methods

- Instance variables should only be accessed using getter/setter methods

OBJECTIVE-C CLASSES

- Instance variables should only be accessed using getter/setter methods
  - Prefer use of @properties

instance variables

OBJECTIVE-C CLASSES

- Inheritance
  - Class definitions are additive
  - Methods and instance variables
  - Single inheritance

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

@interface MyViewController : UIViewController {

}
@end
```

- Overwrite existing methods
  - Custom ‘description’ method

OBJECTIVE-C CLASSES

- Every Foundation framework class inherits from `NSObject`
  - Memory management
  - Runtime cooperation

image vis Apple Inc.

OBJECTIVE-C CLASSES

- Creating a class instance by sending `alloc`
  - Dynamically allocates memory

```
//  Alloc and init
Class *myClass = [[myClass alloc] init];

//  Init with parameters
Class* myClass = [[myClass alloc] initWithInt:1];
```

- Initialize a class instance's instance variables by sending `init`
  - Can customize or override `init` method
  - Must call `[super init]`

OBJECTIVE-C CLASSES

- All NSObjects implement the method `-description`

```
[NSString stringWithFormat:@”This is my class:%@”, anObject];
```

- Called in format string using %@

```
[NSString stringWithFormat:@”This is my class:%@”, anObject];
```

- Called in format string using %@

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

- Common to overwrite it for custom classes

```
-(NSString *)description
{
  return [NSString stringWithFormat:@"<GameObject: %@, Position: %f, %f>",
                  [self objectID], [self position].x, [self position].y];
}
```

OBJECTIVE-C CLASSES

- Convenience methods to create a class
  - Automatically alloc, init and autorelease (pre-ARC) an object
  - Pro: Less c
  - Cons: Less

```
//  Convenience method
NSString* myString = [NSString string];

//  Manual
NSString* myString = [[NSString alloc] init];
```

OBJECTIVE-C CLASSES

- Destroying a class by sending message `release`

```
//  Pair alloc-init with a release
Class *myClass = [[myClass alloc] init];
...
[myClass release];

//  Best practices
[myClass release], myClass = nil;
```

- Setting the variable to `nil` prevents bad access later
  - Sending a message to `nil` does nothing

Pre-ARC

OBJECTIVE-C CLASSES

- Class equalities of pointer values (e.g. class instances are identical)

```
if ([objectA class] == [objectB class]) { //...
```

- Class instance values are equivalent

```
if ([object1 isEqual: object2]) { //...
```

OBJECTIVE-C MEMORY MANAGEMENT

For historical purposes

OBJECTIVE-C MEMORY MANAGEMENT

- Reference counting - keep track of all references to an object
  - Typical pattern: alloc, retain, release

image via http://cocoadevcenttral.com

- If you create an object with `alloc` or `copy` you have to send it `release` or `autorelease`
  - Calls must be balanced or will leak memory or crash

OBJECTIVE-C MEMORY MANAGEMENT

- Reference counting part of NSObject
  - Objects exist in memory if count > 0
- Object lifecycle
  - `+alloc` and `-copy` create objects with count = 1
  - `-retain` increments retain count
  - `-release` decrements retain count
- When retain count == 0
  - Object is destroyed and memory release
  - `-dealloc` method invoked automatically; do not call

OBJECTIVE-C MEMORY MANAGEMENT

- Query the reta

```
NSString *myString = [[NSString alloc] init];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString retain];
  // NSLog(@"Count:%@",[myString count]);
  // 2

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 0 (Dealloced)
```

OBJECTIVE-C MEMORY MANAGEMENT

- Query the reta

```
NSString *myString = [[NSString alloc] init];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString retain];
  // NSLog(@"Count:%@",[myString count]);
  // 2

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 1

  [myString release];
  // NSLog(@"Count:%@",[myString count]);
  // 0 (Dealloced)

  [myString release];
  // Crash
```

OBJECTIVE-C MEMORY MANAGEMENT

- Balance retain an

```
@interface MyAppDelegate_iPhone :
MyAppDelegate {
    UILabel *theLabel;
}
@property (nonatomic, retain) IBOutlet
UILabel *theLabel;
```

```
- (void)dealloc
{
    [theLabel release];
    [super dealloc];
}
```

OBJECTIVE-C MEMORY MANAGEMENT

- Creating and i

```
- initWithCharacters:length:
- initWithString:
- initWithCString:encoding:
- initWithUTF8String:
...
+ stringWithFormat:
+ localizedStringWithFormat:
+ stringWithCharacters:length:
+ stringWithString:
+ stringWithCString:encoding:
```

Convenience methods

OBJECTIVE-C
MEMORY MANAGEMENT

- Using `autorelease`
  - Objects are added to the autorelease pool
  - The pool is “drained” after each event
- Call autorelease on object you alloc-init
- Autorelease is built-in to convenience methods

OBJECTIVE-C MEMORY MANAGEMENT

- Methods with alloc, copy or new in name need to be released or autorelease

```
[[NSString alloc] initWithString:@"hi"] autorelease];
```

- Convenience methods
  - Cover ent

```
[NSString stringWithString:@"Hi"];
```

OBJECTIVE-C MEMORY MANAGEMENT

- Each application creates an NSAutoreleasePool in main.m
  - The pool sends a release message (e.g. drains) at the end of every event loop c
- Create lo es

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

int main(int argc, char *argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

OBJECTIVE-C AUTOMATIC REFERENCE COUNTING

OBJECTIVE-C ARC

- Automatic Reference Counting
  - Introduced in iOS 5
  - Implements automatic memory management for Objective-C objects and blocks
  - Frees the programmer from the need explicitly insert retains and releases

OBJECTIVE-C ARC

- ARC is a compiler feature that provides automatic memory management of objective-c objects

OBJECTIVE-C ARC

- Follows the same memory management convention but “adds code at compile time”
- Evaluates the lifetime of an object
  - Inserts memory management into your compiled code
  - Generates dealloc methods
- How it works
  - Pointers keep objects in memory
  - If pointer gets a new value or is “zeroed” (nil) it is released

OBJECTIVE-C ARC

- Property attributes (pointer types)
  - strong
    - Keeps objects alive
  - weak
    - Point to object, but do not keep it around because another object is managing it
    - IBOutlets are weak because they are managed by parent (e.g. view controller's view)
    - Pare

```
@property (weak, nonatomic) IBOutlet UILabel *sliderValue;
@property (strong, nonatomic) NSString *name;
```

OBJECTIVE-C ARC

- New Rules
  - Cannot explicitly invoke retain, release, dealloc, retainCount, autorelease
  - Cannot use object pointers in C structs
  - No custom autorelease pools
  - Cannot prefix a property accessor with “new” (e.g. newLabel)

OBJECTIVE-C
DECLARED PROPERTIES

OBJECTIVE-C DECLARED PROPERTIES

- @property is data encapsulated or stored by an object

```
@property (strong, nonatomic) UIView *myView;
```

  - Attributes of an object (e.g. label.text, view.backgroundColor)
- Declaring a property

OBJECTIVE-C DECLARED PROPERTIES

- Properties simplify classes
  - Expose variable and methods through *accessor methods (e.g. getters and setters)*
  - Compiler automatically generates getters and setters
- Simplified memory management if using old method
  - Redundant if using ARC (but still common practice and seen in large frame

```
@property (strong, nonatomic) UIView *myView;
```

OBJECTIVE-C DECLARED PROPERTIES

```
@property (strong, nonatomic) UIColor *color;
```

- Class of the object defines an interface to set and get the values of the property
  - Property Name is color
  - Setter created named ‘setColor’
  - Getter created named ‘color’

Naming conventions

OBJECTIVE-C DECLARED PROPERTIES

- Creatin
  - reac
  - reac

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}

// Declare accessor methods
- (void) name;
- (void) setName;

@end
```

```
//
// Leaf.m
//

@implementation Leaf

// Setter
- (void) setName:(NSString *)newName {
    if (_name != newString) {
        [_name release];
        _name = [newName copy];
    }
}

// Getter
- (NSString*)name {
    return _name;
}
@end
```

- Class creation in 2005

OBJECTIVE-C DECLARED PROPERTIES

- Creatin
  - reac
  - reac

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}
@property(nonatomic, retain)
                NSString *name;

// Declare accessor methods
- (void) name;
- (void) setName;

@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;

// Setter
- (void) setName:(NSString *)newName {
    if (_name != newString) {
        [_name release];
        _name = [newName copy];
    }
}

// Getter
- (NSString*)name {
    return _name;
}
@end
```

- Class creation in 2010 (Objective-C 2.0)

OBJECTIVE-C DECLARED PROPERTIES

- Creatin
  - reac et
  - reac

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}
@property(nonatomic,retain)
                NSString *name;



@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;
@end
```

- Class creation in 2010 (Objective-C 2.0)

OBJECTIVE-C DECLARED PROPERTIES

- Creatin
  - reac
  - reac

```
//
// Leaf.h
//

@interface Leaf : NSObject {
    NSString *_name; // ivar
}
@property(strong)NSString *name;

@end
```

```
//
// Leaf.m
//

@implementation Leaf
@synthesize name = _name;
@end
```

- Class creation in 2012

OBJECTIVE-C DECLARED PROPERTIES

- Synthes
  - ivar n
  - prope
  - setter = setName
  - getter = name

```
//
// Leaf.h
//

@interface Leaf : NSObject
@property(strong)NSString *name;
@end
```

```
//
// Leaf.m
//

@implementation Leaf
@end
```

```
// @synthesize name = _name
// done by default
```

OBJECTIVE-C DECLARED PROPERTIES

```
@property (nonatomic, weak) IBOutlet UILabel *theLabel;
```

- Atomicity
  - Multi-threading option
  - ALWAYS set to (`nonatomic`) unless you mean it; atomic is default
- Writeabilty
  - `readwrite` - default behavior; creates getters and setters
  - `readonly` - creates getter only
- Memory management:
  - Applies to setters only
  - Options: `strong`,`weak`, `copy`

OBJECTIVE-C DECLARED PROPERTIES

```
// Generate getters/setters and retain the incoming object
@property (nonatomic, strong) NSString *name;

// Same
@property (nonatomic, strong) NSString *title;

// Generate getters/setters and assigns the incoming value
@property (nonatomic) int age;

// Generate getters/setters and assigns the incoming value
@property int age;

// Generate the getters only
@property (nonatomic, readonly) NSDate *created;

// Generate the getters/setters and create a new instance
// with the same values as the original
@property (nonatomic, copy) NSString *backupData;
```

OBJECTIVE-C DECLARED PROPERTIES

- Writeabilty
  - readwri
  - readonl
- Memory m
  - Applies
  - Options

```
// Generate getters/setters and retain th
@property (nonatomic, strong) NSString *n

// Same
@property (nonatomic, strong) NSString *t

// Generate getters/setters and assigns the incoming value
@property (nonatomic) int age;

// Generate getters/setters and assigns the incoming value
@property int age;

// Generate the getters only
@property (nonatomic, readonly) NSDate *created;

// Generate the getters/setters and create a new instance
// with the same values as the original
@property (nonatomic, copy) NSString *backupData;
```

copy is required when the object is mutable. Use this if you need the value of the object as it is at this moment, and you don't want that value to reflect any changes made by other owners of the object.

OBJECTIVE-C DECLARED PROPERTIES

- A class' `self`
  - Implicit local variable
  - Points to the object that was sent the message
  - Allows an object to send a message `[self addDate]`
- `self` uses the a ables

```
// Getter for name property
[self name];

// Call method "doSomething" in current class
[self doSomething];

// Self as parameter to method
[someClass doWorkOn:self];
```

OBJECTIVE-C DECLARED PROPERTIES

- Allows use of do
  - Different no ame

```
// Create property
@property int year;
@synthesize year;

// Bracket notation
[myCar year];           // Getter
[myCar setYear: 1962]; // Setter

// Dot notation
myCar.year = 1962;          // Setter
NSLog(@"%d", myCar.year);  // Getter
```

OBJECTIVE-C DECLARED PROPERTIES

- Invoking accessor methods (i.e. getters and setters)

```
[[object theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
[object.theArray insertObject:myAppObject.objectToInsert atIndex:0];
```

dot

dot

- Use for a

```
myAppObject.theArray = aNewArray;
[myAppObject setTheArray:aNewArray];
```

OBJECTIVE-C BLOCKS

BLOCKS

- Objects th
  - Anonyr
  - Passed

```
int result = myBlock(4); // result is 28
```

BLOCKS

- Simplify execution of a single task versus writing a collection of methods
  - Scoped locally and variables can be read-only or read-write
- Used throughout UIKit
  - Callbacks
  - Animations
  - Multi-threading
  - Param

```
- (id)addObserverForName:(NSString *)name
                  object:(id)obj
                   queue:(NSOperationQueue *)queue
              usingBlock:(void (^)(NSNotification *note))block
```

BLOCKS

- We will di

We will discuss this in more depth later.

OBJECTIVE-C CATEGORIES

CATEGORIES

- Allow you to add methods to an existing class
- Why not subclass?
  - Would not have to change existing codebase
  - Can be applied to other methods that return NS and UI classes

CATEGORIES

Choose options for your new file:
Class
NSArray+Shuffle
Subclass of
NSObject
Cancel
Previous
Next

CATEGORIES

- Declarin

```
//
//  NSArray+Shuffle.h
//  MapKitDemonstration
//
#import <Foundation/Foundation.h>

@interface NSArray (Shuffle)
- (NSArray*) shuffle;
@end
```

CATEGORIES

- Implem

```
//
//  NSArray+Shuffle.m
//  MapKitDemonstration

#import "NSArray+Shuffle.h"

@implementation NSArray (Shuffle)
- (NSArray*)shuffle
{
    // create temporary mutable array
    NSMutableArray *tmpArray = [NSMutableArray arrayWithCapacity:[self count]];

    for (id anObject in self) {
       NSUInteger randomPos = arc4random()%([tmpArray count]+1);
       [tmpArray insertObject:anObject atIndex:randomPos];
    }

    return [NSArray arrayWithArray:tmpArray];  // non-mutable autoreleased copy
}

@end
```

CATEGORIES

```
// Test NSSArray+Shuffle Category
   NSArray *places = [NSArray arrayWithObjects:@"Universal Studios",
                                               @"Disney World", @"Sea World", nil];
   NSLog(@"Places (before): %@",places);
   NSLog(@"Places (after): %@",[places shuffle]);
```

- Using a category

CATEGORIES

- Tips & Tricks
  - Categories are inherited (i.e. categories on NSObject are available to every class)
  - The names must be unique across all namespaces
- A couple useful categories
  - coredata-easyfetch
    - https://github.com/halostatue/coredata-easyfetch
  - Image utilities
    - http://mattgemmell.com/2010/07/05/mgimageutilities/

OBJECTIVE-C PROTOCOLS

PROTOCOLS

- Declare methods that can be implemented by any class
- Not a class; define an interface that other objects can implement
  - "Conforming to a protocol"
  - Impl

```
@protocol MyProtocol

@required
- (void)myProtocolMethod;
@option
- (void)anotherMethod;

@end
```

PROTOCOLS

- Classes that wish to implement a protocol declare it in their interface

```
@interface MyClass : NSObject <AProtocol> {}
...
@end
```

- All non-optional methods must be implemented.
- Used to specify the interface for a delegate object
  - Can be done programmatically or in Storyboard

PROTOCOLS

Main.storyboard
Images.xcassets
M
LaunchScreen.xib
Supporting Files
Info.plist
main.m
It's A ZooTests
Products
UIScrollView
Label
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Table View Controller - A controller that manages a table view.

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Protocol definition

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Declare a delegate that adopts a protocol

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Conform to a protocol

PROTOCOLS

```
@protocol UIScrollViewDelegate
@optional
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
                  willDecelerate:(BOOL)decelerate
...
@end

@interface UIScrollView : UIView
@property (assign) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate> {}
@end

MyViewController *myVC = [[MyViewController alloc] init];
UIScrollView *scrollView = ...;
scrollView.delegate=myViewController;
```

Set delegate when methods are implemented

PROTOCOLS

```
@interface MyViewController : UIViewController <UIScrollViewDelegate> {}

- (void)loadView {
...
  // Add the scrollview
  scroll = [[UIScrollView alloc] initWithFrame:CGRectMake(0,30,320,400)];
}


- (void)scrollViewDidScroll:(UIScrollView *)scrollView {
    static NSInteger previousPage = 0;
    CGFloat pageWidth = scrollView.frame.size.width;
    float fractionalPage = scrollView.contentOffset.x / pageWidth;
    NSInteger page = lround(fractionalPage);
    if (previousPage != page) {
        self.currentPage = page;
        previousPage = page;
        NSLog(@"PAGE %d",self.currentPage);
        // Do something with page
    }
}
```

Method declared in protocol. Sends messages every time the scrollview scrolls

PROTOCOLS

```
///-----------------------------------------------------------------------------
-(void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView
{

    CGFloat pageWidth = self.animalScrollView.frame.size.width;
    CGFloat fractionalPage = self.animalScrollView.contentOffset.x / pageWidth;
    NSInteger page = lround(fractionalPage);

    // Test if page is new
}


- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    [self loadScrollViewData];

}

- (void) loadScrollViewData
{
    NSLog(@"In ViewController calling loadScrollViewData");
    int numberOfAnimals = [self.animals count];

    // Set the content size of the scroll view
    self.animalScrollView.contentSize = CGSizeMake(numberOfAnimals * self.animalScrollView.frame.size.width,
                                                   self.animalScrollView.frame.size.height);

```

PROTOCOLS

Main.storyboard
Images.xcassets
M
LaunchScreen.xib
Supporting Files
Info.plist
main.m
It's A ZooTests
Products
@interface ViewController : UIViewController <UIScrollViewDelegate>
@property (weak, nonatomic) IBOutlet UIScrollView *animalScrollView;
@property (weak, nonatomic) IBOutlet UILabel *animalLabel;
@property (strong, nonatomic) NSMutableArray *animals;
@end

IOS DESIGN PATTERNS

IOS DESIGN PATTERNS

- iOS is epitome of object oriented programming
  - "Abstract designs that solve recurring problems in a particular context", Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides

IOS DESIGN PATTERNS

- Adopts many well known design patterns
  - Model-View-Controller (MVC)
  - Object Modeling (Core Data)
  - Target-Action
  - Key-value observing

MODEL-VIEW-CONTROLLER

MODEL-VIEW-CONTROLLER PATTERN

SUBTITLE

- Every object you create in iOS is either:
  - Model
  - View
  - Controller

MODEL-VIEW-CONTROLLER PATTERN

SUBTITLE

- Model objects encapsulate data and basic behaviors
  - Your data and logic
  - NSArray, NSDictionary, NSSet, CoreData

MODEL-VIEW-CONTROLLER PATTERN

SUBTITLE

- View objects present information to the user
- You see and interact with views
- Display date from the model objects
  - UILabels, UIButtons, UISlider, UIView

MODEL-VIEW-CONTROLLER PATTERN

SUBTITLE

- Controller objects tie the model to the view
- Allows views access to model
- Allow views to learn about model changes
- Manage lifecycle of objects
- Examples: UIViewController, AppDelegate

MODEL-VIEW-CONTROLLER PATTERN

Image: iOS Programming Big Nerd Ranch Guide - Joe Conway & Aaron Hillegrass

MODEL-VIEW-CONTROLLER PATTERN

SUBTITLE

- Simplifies universal application development
  - Shared models and controllers between iPad and iPhone

TARGET-ACTION MECHANISM

TARGET-ACTION MECHANISM

- Enables view objects to communicate user input and choices
- Design pattern
  - “Object holds information necessary to send a message to another object when an even occurs”

TARGET-ACTION MECHANISM

SUBTITLE

- Control
  - The object that sends the message (e.g. button, slider)
- Target
  - Object that receives the action
  - Typically a view controller
- Action message
  - The message sent when the event occurs
  - Typically sent to a controller object (e.g. tap a UIButton, message sent to UIViewController)

TARGET-ACTION MECHANISM

- Enables view objects to send a message to another object when an event occurs

image via Apple, Inc.

DELEGATION

DELEGATION

SUBTITLE

- One object acts on behalf or in coordinate of another object
- Delegate keeps a reference to an object and sends messages to it
  - Allows objects to communicate with each other

image via Apple, Inc.

DELEGATION

- Delegation important part of many frameworks
- You can write your own
  - A parent view controller communication with a child view controller

image via Apple, Inc.

OTHER NOTABLE PATTERNS

NOTIFICATION CENTER

SUBTITLE

- Broadcasts a message to all objects that are registered to observe the event
- iOS notifications (e.g. UIKeyboardWillShowNotifications)
- Custom notification (e.g. downloading complete)

image via Apple, Inc.

KEY-VALUE OBSERVING

SUBTITLE

- Objects observe a property of another object
- Notified when the value changes

image via Apple, Inc.

PROTOCOL

SUBTITLE

- Declaration of programmatic interface whose methods any class can adopt and implement
- Many Apple frameworks have protocols for interacting with objects (e.g. tableView:heightForRowAtIndexPath:)

image via Apple, Inc.