Intro to Objective-C

+

Exercises

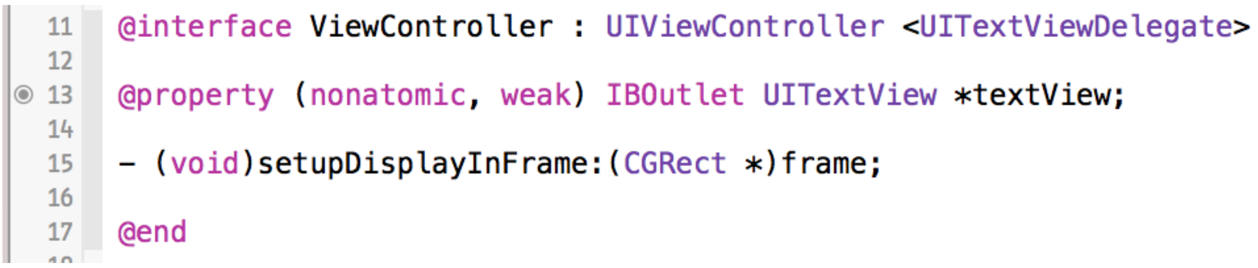
Designed by Alex Benevento & Valery Shorinov

Table of Contents

Concepts

Classes

Classes in Objective-C are considered as **Templates** for objects. They define variables and methods which represent an entire object. e.g: A human is a class, an animal is a class, a bird is a class – they are broad idea's of objects.



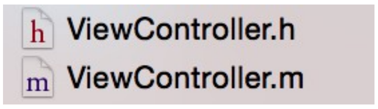
```
@interface ViewController : UIViewController <UITextViewDelegate>

@property (nonatomic, weak) IBOutlet UITextView *textView;

- (void)setupDisplayInFrame:(CGRect *)frame;

@end
```

Classes usually have header and implementation files; .h and .m

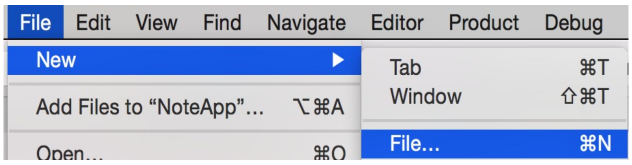


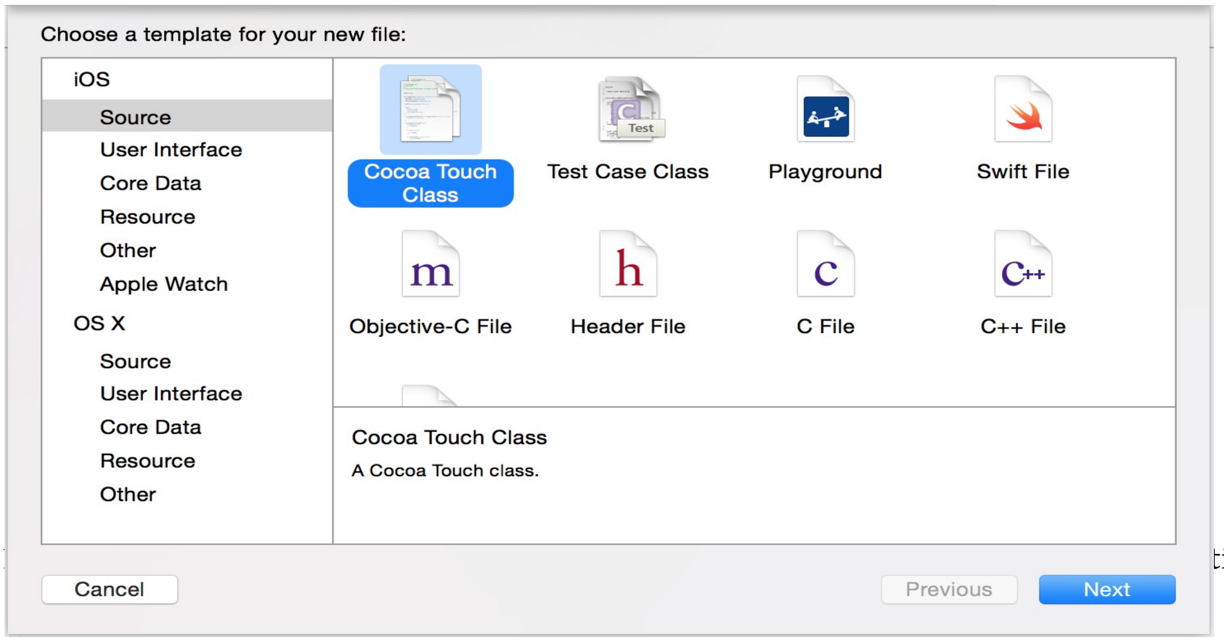
ViewController.h
ViewController.m

Header files define the public access that a class has, I.e: which other objects can see about that class/object.

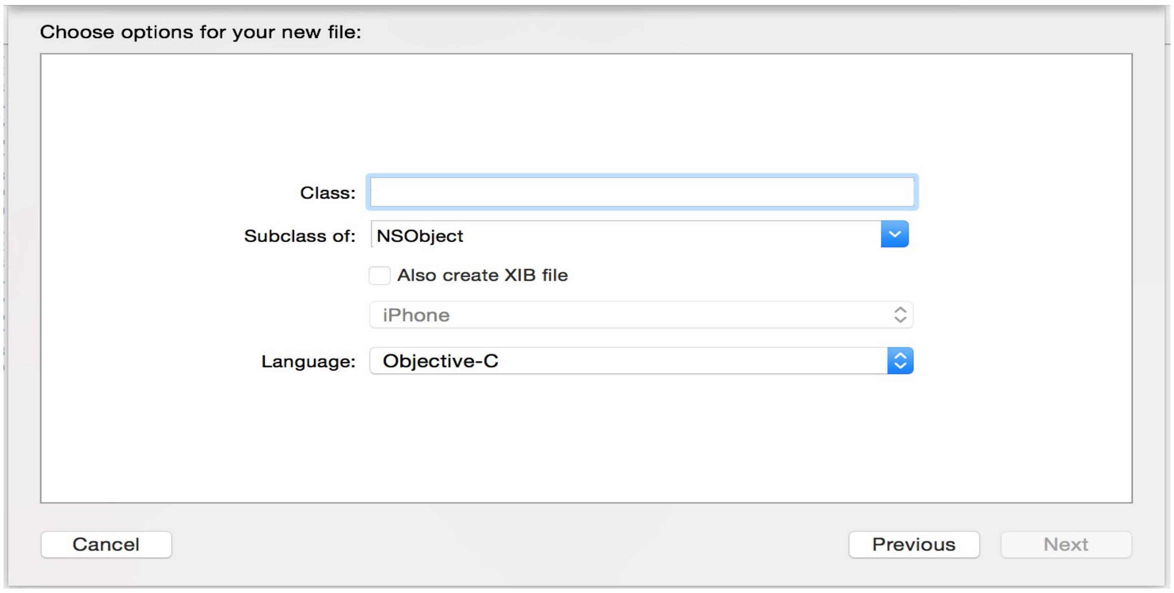
Implementation files contain all the code that make methods perform operations.

To create a new class:





Cocoa Touch Class
A Cocoa Touch class.


Choose options for your new file:

Class:

Subclass of: NSObject

Also create XIB file

iPhone

Language: Objective-C

This will create a .h & .m file. The structure of making a class is:

```
@interface Calculator : NSObject
```

@interface = the syntax for starting a class

Calculator = The class name

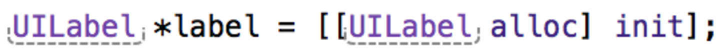
NSObject = The class that this class inherits from (its root class). All object must atleast inherit from NSObject.

Remember to end all classes with @end

Objects

Objects are instantiated classes, I.e: classes that have values in them. e.g: You are an object of class human, a cat is an object of class animal, a penguin is an object of class bird.

How to create an object:



```
UILabel *label = [[UILabel alloc] init];
```

alloc = Allocation (prepares the class to be used, allocates memory, etc)

init = Initialise (creates the object with default values)

Once created, you can call methods and access variables on that object.

Variables

A variable is a container which stores data and can be described as attributes. e.g: Your height is a variable, your name is a variable, your age is a variable.

There are many different types of variables:

```
int wholeNumber = 42; // Scaler value
float floatingPointNumber = 42.4251157; // Scaler value
double accurateNumber = 432871.436718321; // Scaler value
BOOL trueOrFalse = YES; // Scaler value - Represented in 0 (false/NO) or 1 (true/YES)
NSString *text = @"Pizza"; // Object which represents text
NSArray *containerOfObjects = @[text, text]; // Object represents a collection of objects
NSDictionary *containerOfObjectsRelatedByKey = @{@"FirstKey":text,@"SecondKey":text}; // represents a collection
```

*Note: Objects are variables, complex variables...

You can define a class variable as a property, which can be accessed by other classes.

```
@property (nonatomic, weak) IBOutlet UITextView *textView;
```

To access a class variable you need to call it on that class (or self if you're trying to access your own variable)

```
label.text = @"Setting Text"; // Label is an object, and text is a class variable
self.textView = [[UITextView alloc] init]; // We have a class object of textView in this class
```

To compare two variables you need to use == (if its scaler variables), if they are objects, you need to use methods.

```
if (wholeNumber == otherNumber) // Comparing two ints
if ([text isEqualToString:otherText]) // Comparing to see if two strings are the same
```

Arrays are a great way of storing multiple objects in one container to be used;

```
NSArray *students = [[NSArray alloc] init]; // Creates empty array
   students = [NSArray array]; // Creates empty array quickly
   students = [NSArray arrayWithObject:@"String"]; // Creates array with one object
                                       // Which is a NSString
   students = [NSArray arrayWithObjects:@"String1", @"String2", nil];
   // Creates array with 2 object which are strings
```

There are limitations with Arrays (and even strings), once they're instantiated, they can not be modified. They are considered immutable (not mutable). But you can set it as another array... There is also NSMutableString and NSMutableArray, which you can add and remove things from:

```
NSMutableArray *students = [NSMutableArray array]; // Creates empty array
   [students addObject:@"NewStudent"]; // Adds a new NSSting to students
   [students addObject:@"Seconds Student"]; // Adds a new NSString to students
   [students removeObject:@"NewStudent"]; // Removes the NSString @"NewStudent"
```

```
NSMutableString *mutableString = [NSMutableString string] // Creates empty string
[mutableString appendString:@"Added this text"]; // Adds the string to the end
[mutableString appendString:@"More text"]; // Adds @"More text" to the end
```

Methods

Methods are blocks of code which perform operations, simply put they are behaviours of a class. e.g: Humans have a method to breath, birds have a method to fly.

There are two types of methods;
Instance methods – Methods which can only be called once a class has been created.

```
- (void)performActionOnObjectWithText:(NSString *)text;
```

How to use an instance method:

```
ViewController *newController = [[ViewController alloc] init];
[newController performActionOnObjectWithText:@"SomeText"];
```

Class methods – Methods which can be used from the class itself, no need to instantiate the class first.

```
+ (void)performOperationWithText:(NSString *)text;
```

How to use a class method:

```
[ViewController performOperationWithText:@"OtherText"];
```

Structure of a method:

```
- (NSString *)combineString:(NSString *)stringOne andString:(NSString *)stringTwo;
```

- Instance or Class method syntax/declaration

(NSString *) - The return type. What this method returns.

combineString – The method name. What you use to call it.

:(NSString *) - The variable type of the first variable that is being passed to this method.

stringOne – The name of the first variable.

andString: - Continuation of the name.

:(NSString *) - The variable type of the second variable that is being passed to this method.

stringTwo – The name of the second variable.

Structure of calling a method:

```
NSString *returnString = [viewController combineString:@"I" andString:@"am"];
```

NSString *returnString – The value that is being returned will be saved to this variable.

[- Open the method

viewController – name of the object

combineString: - name of the method

@"I" – The first variable being past

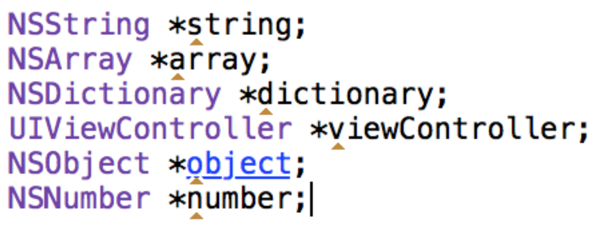
andString: - name continuation

@"am" – The Second variable being past

Pointers:

A pointer is represented by *. A pointer is used for objects, as it represents a memory location of where the actual object is stored in memory.

Example of pointers:



```
NSString *string;
NSArray *array;
NSDictionary *dictionary;
UIViewController *viewController;
NSObject *object;
NSNumber *number;
```

# Imports:

A #import is a way of saying which files to include into that class. If you have a class calculator and you wish to use it, you first need to import that header file into the class where you wish to use it.

```
#import "Calculator.h"
```

If statements:

'If' is used to compare values, a way to split operations depending on a condition. e.i; if will perform any code inside the {} braces if the condition inside the () bracket is true. Eg:

```
if (canFly)
   {
      // This only happens if canFly is equal to true
   }
```

You can compare values in the If statement

```
if (numberOfLegs == 4)
   {
      // This only happens if numberOfLegs is equal to 4
   }
```

Objects need specific methods to return a Bool to compare

```
if ([string isEqualTo:otherString])
   {
      // This only happens if both strings are equal
   }
```

You can have multiple compares in one if statement.. Use the && operator to say that all conditions need to be meet, and the || if any of the conditions need to be met.

```
if (numberOfLegs == 2 || numberOfLegs == 4)
   {
      // This only happens if number of legs is either 2 or 4
   }

if (canFly && hasWings)
   {
      // Only does this if canFly and hasWings are both true
   }
```

You can mix them too

```
if ((numberOfLegs == 4 && canFly) || hasJetPack)
   {
      // Only does this if numberOfLegs is 4 and canFly, Or if it hasJetPack
   }
```

There is a 'not' operator which reverses the outcome, I.e; if will now check the false outcome:

```
if (!canFly)
   {
      // Only does this if canFly is false
   }
```

If statements also have an else & else if conditions... That is to say, when writing an else if, it means, if you did not adhere to the first condition, will you adhere to this one? Else simply means if no condition has been met, do this.

```
if (canFly)
   {
      // This gets called if canFly is true
   }
   else if (hasJetPack)
   {
      // This gets called if canFly is false
      // But hasJetPack is true
   }
   else
   {
      // This is called if all previous conditions are not met
   }
```

For Loops:

For loops are recursive block of code that will continue until an end point is reached.

```
int total = 0;
   for (int i = 0; i < 10; i++)
   {
      total += i;
      // This will run the loop 10 times before ending
      // It will add i (which increases by 1 every time)
      // to the total, resulting in 45
   }
```

- First we need a variable to increment and tell us how many times to do the loop (int i = 0), then we give it a condition of when to finish (i < 10 (i must be less then 10, otherwise stop), finally we tell i by how much to increment each cycle i++.

You can use loops to iterate through all the objects in an array.

```
for (Student *student in students)
   {
      // This will iterate through every student in students
      [student sendHome];
      // And send each one home
   }
```

Student here is a class, making *student the object that is pulled out of the array students each cycle.

Switch statements:

Switch statements are used to quickly compare a value with multiples conveniently. A switch statement can technically be written as a long if check, but more convenient with a switch.

If statement:

```
if (numberOfLegs == 2)
   {
      // Gets called if numberOfLegs is 2
   }
   else if (numberOfLegs == 4)
   {
      // Gets called if numberOfLegs is 4
   }
   else if (numberOfLegs == 5)
   {
      // Gets called if numberOfLegs is 5
   }
   else if (numberOfLegs == 3)
   {
      // Gets called if numberOfLegs is 3
   }
   else if (numberOfLegs == 18)
   {
      // Gets called if numberOfLegs is 18
   }
   else
   {
      // All other possibilities
   }
```

Switch statement:

```
switch (numberOfLegs) {
      case 2:
         // If numberOfLegs is 2
         break;
      case 4:
         // If numberOfLegs is 4
         break;
      case 5:
         // If numberOfLegs is 5
         break;
      case 3:
         // If numberOfLegs is 3
         break;
```

```
    case 18:
        // If numberOfLegs is 18
        break;

    default:
        // If no other condition has been met
        break;
}
```

Examples

Classes

Example 1: Setup a human class which extends NSObject, having two properties of Age and Name & a method of movement. Then create two humans.

```
// human.h

@interface human : NSObject;

@property (nonatomic) int age;
@property (nonatomic, strong) NSString *name;

- (void)move;

@end

// human.m

@implementation human

- (void)move
{
    // Do movement code here
}

@end

// How to call

    human *humanOne = [[human alloc] init];
    humanOne.name = @"Bob";
    humanOne.age = 22;

    human *humanTwo = [[human alloc] init];
    humanTwo = @"Sam";
    humanTwo.age = 18;

    [humanOne move];
    [humanTwo move];
```

Example 2: Setup three classes; Animal (with a property of name & method of move which subclasses NSObject). LandCreature (with an additional property of numberOfLegs, overwrites the move method, method to jump which subclasses Animal). Bird (additional property of canFly, overwrites move method, additional fly method, subclasses Animal). Then create an example of each filling in all the properties.

```
// Animal.h

@interface Animal : NSObject

@property (nonatomic, strong) NSString *name;

- (void)move;

@end

// Landcreature.h

@interface Landcreature : Animal

@property (nonatomic) int numberOfLegs;

- (void)jump;

@end

// Bird.h

@interface Bird : Animal

@property (nonatomic) BOOL canFly;

- (void)fly;

@end
```

```
// Landcreature.m

@implementation Landcreature

- (void)move // This overWrites the move from the super class of Animal
{
   // Do land creature move here
}

___________________________  // Jump method implementation here

__
      // Do jump method here

__


@end


// Bird.m

______________________________ // Write the Bird Implementation

__________________________ // Over-Write the move method

__

      // Move code
__

- (void)fly
{
   if (!self.canFly) // If bird can't fly, finish this operation
   {
      return;
   }
      ...
}

@end
```

```
// How to call

    Animal *firstAnimal = [[Animal alloc] init];
    firstAnimal.name = ______________________ // Give the animal a name
    [firstAnimal move];

    Landcreature *firstCreature = [[Landcreature alloc] init];
    firstCreature.name = @"Tiger";
    firstCreature.numberOfLegs = 4;
    [________________________]; // Call the move method on firstCreature
    [firstCreature jump];

    Bird *firstBird = [[Bird alloc] init];
    ____________________________________  // Name the bird
    ____________________________________ // Allow this bird to fly
    ____________________________________ // Call the move method
    ____________________________________ // Call the fly method
```

In the blank space create some more animals, creatures and birds.

Example 3: Setup a class Calculator, with no properties, with an AddValue: to: class method (returns a float, and takes two values), a subtractValue: from: class method (returns float, and takes two values), a Sum class method (which returns a float and takes an array of numbers, which it adds and returns). Implement the proper code for all these methods. Write one more method on calculator which draws it on screen inside a Frame.

```
// Calculator.h
____________________________________ // Implement the  interface

+ (float)addValue:(float)firstValue to:(float)secondValue;
_____________________________________________________ // Implement Subtract

+ (float)sum:(NSArray *)arrayOfNumbers;

- (void)drawInRect:(CGRect)frame;

@end
// Calculator.m

@implementation Calculator

+ (float)addValue:(float)firstValue to:(float)secondValue
{
   float total = firstValue + secondValue;
   return total;
}
// Write the subtract Method
______________________________________________________________________
___
   _________________________________________________

   _________________________
___

+ (float)Sum:(NSArray *)arrayOfNumbers
{
   float total = 0.0;

   for (NSNumber *value in arrayOfNumbers)
   {
      total = [Calculator addValue:total to:[value floatValue]];
   }

   return total;
}
```

```
// Implement drawInRect method
______________________________________________

___
        // Drawing code goes here
___

@end
```

Now write some examples of using this calculator class:

```
float total = 0.0;                    // total will equal 0
    total = [Calculator addValue:43 to:10]; // total will equal 53
//    ________________________________________  // use add to add 12 and 13 (total will be 25)
    total = [Calculator subtractValue:10 from:21]; // total will equal 11
//    ________________________________________  // use subtract to sub 16 from 30 (total will be 14)

    // **Arrays can only hold objects, we first need to make NSNumber object from numbers
    // Which is done [NSNumber numberWithFloat:21]... or short hand @21;
    total = [Calculator sum:[NSArray arrayWithObjects:@12.1, @54, @5, @12, nil]]; // total = 83.1

    // Write your own sum use
    ________________________________________________________________________________

    ________________________________________________________________________________

    // Write the use of drawInRect for calculator - hint, pass CGRectZero as the value
    _______________________________________________

    _______________________________________________
```

Now do some of your own examples of using Add, Subtract and Sum....

Now in your Calculator class, add a new class method which multiplies two values, and then another call method which multiplies an array of numbers.

```
// Calculator.h
```

```
// Calculator.m
```

Now write some examples of using this new multiply method.

Example Condition statements:

If:

Write an if statement which compares two numbers, if the same print the same other print different (assume both numbers are int):

```
if (firstNumber == secondNumber)
   {
      NSLog(@"Same"); // Prints to console
   }
   else
   {
      NSLog(@"Different"); // Prints to console
   }
```

Write an if statement that compares two Strings:

```
if ([firstString isEqualToString:_________________])
   {
      NSLog(@"Same");
   }
   else
   {
      NSLog(@"Different");
   }
```

Write an if statement that returns “Both Same” if both string are the same and both numbers are the same, if only string is the same “Only Strings are same”, if only numbers are the same “Only numbers are same”, else “All Different”; **Hint, there are multiple ways of checking this:

```
   if (______________ == secondNumber && [firstString __________________:secondString])
   {
      NSLog(@"Both Same");
   }
   else if ([_______________________________________])
   {
      NSLog(@"Only strings are same");
   }
   else if (_________________________________________)
   {
      NSLog(@"Only numbers are same");
   }
   else
   {
      NSLog(@"All Different");
   }
```

OR

```
if (firstNumber == secondNumber)
{
   if ([firstString ________________:secondString])
   {
      NSLog(@"Both Same");
   }
   ________
   {
      NSLog(@"Only numbers are same");
   }
}
_____________([_____________ isEqualToString:________________])
{
   NSLog(@"Only Strings are same");
}
else
{
   NSLog(@"All Different");
}
```

Write some of your own statements:

If a bird can fly, let it Fly().

If an animal canMove AND isAwake, let it Move().

If a student does not have properUniform (use !), and is being rude (isBeingRude()) send to principles office, if student is just out of uniform giveWarning(), if student being rude smackBackOfHead(), else complimentStudent().

For:

Write a loop that iterates 15 times multiples all numbers

```
double total = 0;
   for (int i = 1; i <= 15; i++)
   {
      total *= i;
   }
```

Write a loop that will multiply all EVEN numbers up to (and including 20)

```
double total = 0;
   for (_________; i <= _____; i+=2) // This will increment by two every time
   {
      total *= i;
   }
```

Write a loop that subtracts all number between 60 and 40 from total

```
double total = 0;
   for (int i = _____; i ___ 40; i--) // This will decrement by one every loop
   {
      _____________
   }
```

Write a loop that prints every third number to console (NSLog()), between54 and 156

Write a loop that checks every Student in the class if they have blonde hair, if so, print their name

```
for (Student *currentStudent in allStudentsInClass)
   {
      if ([currentStudent.hairColour isEqualToString:@"Blonde"])
      {
         NSLog(@"Student:%@ has blonde hair", currentStudent.name);
      }
   }
```

Write a loop that checks all Teachers in the School and prints their subject and name to console

Switch:

Write a switch that checks numberOfWheels; If 0 “Plane”, if 1 “Unicycle”, if 2 “Bike”, if 4 “car”, default “Unidentified”

```
switch (numberOfWheels) {
      case 0:
         NSLog(@"Plane");
         break;

      ________
         NSLog(@"Unicycle");
         break;
      case 2:
         NSLog(@"Bike");

         __________

      _________
         _________@"Car");
         _________

      default:
         NSLog(@"Unidentified");
         break;
   }
```

Write a switch that checks numberOfSides on an object, and prints its shape name to the console eg: 1 side = “circle”, 2 sides = “line”, etc... Default is “some complex shape”

Practice project:

Open up PracticeExercises project. If you build you will notice heaps of warnings.. Read them! They give you TODO activities that will help you understand objective-c programming. Good luck! And you are always more than welcome to expand on them.