



Notes on iOS Programming

Antony Harfield

November 2011

CSIT, Faculty of Science, Naresuan University

Why?

- Learn something new...
 - Use Mac OS X and the Xcode IDE
 - Try a new programming language
 - Develop simple iPhone applications
 - Understand the difference between mobile and desktop apps

Part 1: Objective C

Objective C

- What do you know about Objective C?
 - Similar to C (its a superset of C)
 - It is an OOP language
 - Primary language used on Mac OS X and iOS
 - Started life in 1983

To start with...

- Let's take a look at a simple Java program...

```
public class Fraction {  
  
    private int topNumber;  
    private int bottomNumber;  
  
    public void setTopNumber(num) {  
        topNumber = num;  
    }  
  
    public void setBottomNumber(num) {  
        bottomNumber = num;  
    }  
  
    public int getTopNumber() {  
        return topNumber;  
    }  
  
    public int getBottomNumber() {  
        return bottomNumber;  
    }  
  
    public void print() {  
        System.out.println(Int.toString(topNumber) + "/"  
            + Int.toString(bottomNumber));  
    }  
}
```

```
public class Main {  
  
    public static void main(String args[]) {  
  
        Fraction frac = new Fraction();  
  
        frac.setTopNumber( 1);  
        frac.setBottomNumber( 3);  
  
        System.out.print( "The fraction is: ");  
        frac.print();  
        System.out.println();  
  
    }  
}
```

What is the output?

```

public class Main {

    public static void main(String args[]) {

        Fraction frac = new Fraction();

        frac.setTopNumber( 1);
        frac.setBottomNumber( 3);

        System.out.print( "The fraction is: ");
        frac.print();
        System.out.println();

    }
}

```

Java

Objective C

```

#import <stdio.h>
#import "Fraction.h"

int main(int argc, const char *argv[]) {

    Fraction *frac = [[Fraction alloc] init];

    [frac setTopNumber:1];
    [frac setBottomNumber:3];

    printf("The fraction is: ");
    [frac print];
    printf("\n");

    [frac release];

    return 0;

}

```

```

public class Fraction {

    private int topNumber;
    private int bottomNumber;

    public void setTopNumber(num) {
        topNumber = num;
    }

    public void setBottomNumber(num) {
        bottomNumber = num;
    }

    public int getTopNumber() {
        return topNumber;
    }

    public int getBottomNumber() {
        return bottomNumber;
    }

    public void print() {
        System.out.println(Int.toString(topNumber) + "/"
            + Int.toString(bottomNumber));
    }

}

```

Java

Objective C

```

#import "Fraction.h"
#import <stdio.h>

@implementation Fraction

- (void)setTopNumber:(int)num {
    topNumber = num;
}

- (void)setBottomNumber:(int)num {
    bottomNumber = num;
}

- (int)topNumber {
    return topNumber;
}

- (int)bottomNumber {
    return bottomNumber;
}

- (void)print {
    printf("%i/%i", topNumber, bottomNumber);
}

@end

```

```

public class Fraction {

    private int topNumber;
    private int bottomNumber;

    public void setTopNumber(num) {
        topNumber = num;
    }

    public void setBottomNumber(num) {
        bottomNumber = num;
    }

    public int getTopNumber() {
        return topNumber;
    }

    public int getBottomNumber() {
        return bottomNumber;
    }

    public void print() {
        System.out.println(Int.toString(topNumber) + "/"
            + Int.toString(bottomNumber));
    }
}

```

Java

Objective C

```

@interface Fraction : NSObject {
    int topNumber;
    int bottomNumber;
}

- (void)setTopNumber:(int)num;
- (void)setBottomNumber:(int)num;
- (int)topNumber;
- (int)bottomNumber;

- (void)print;

@end

```

Also need a header file for your class!

```

@interface Fraction : NSObject {
    int topNumber;
    int bottomNumber;
}

- (void)setTopNumber:(int)num;
- (void)setBottomNumber:(int)num;
- (int)topNumber;
- (int)bottomNumber;

- (void)print;

@end

```

Header file (.h)

In Objective C,
you need two files
for one class!

```

#import "Fraction.h"
#import <stdio.h>

@implementation Fraction

- (void)setTopNumber:(int)num {
    topNumber = num;
}

- (void)setBottomNumber:(int)num {
    bottomNumber = num;
}

- (int)topNumber {
    return topNumber;
}

- (int)bottomNumber {
    return bottomNumber;
}

- (void)print {
    printf("%i/%i", topNumber, bottomNumber);
}

@end

```

Implementation file (.m)

Basic types

- Basic types: `int`, `float`, `char`
- Insert into strings using `printf`:

```
printf("%i %f", myInt, myFloat);
```

- To convert int to float:

```
(float)number or 1.0 * number
```

They are NOT objects!

Objects

- Objects are different to the basic types:

```
int topNumber; // integer
```

```
Fraction *frac; // Fraction object
```

- Objects always use pointers:

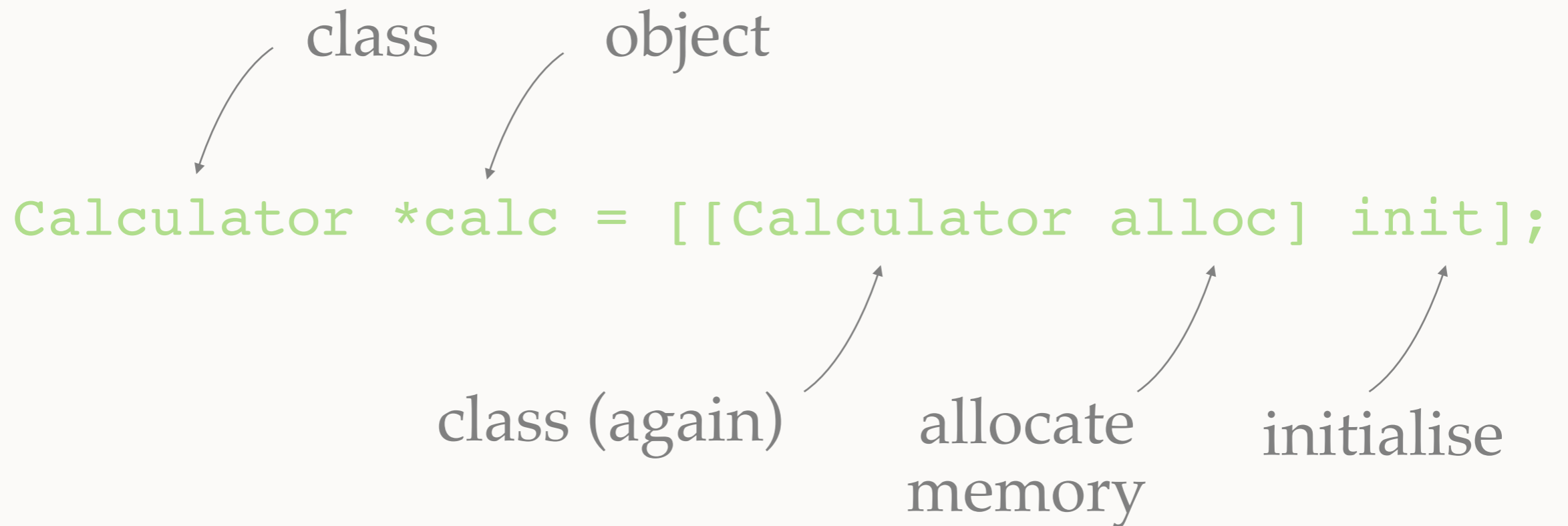
```
Fraction *frac;
```

class

object



Creating objects



In Java:

```
Calculator calc = new Calculator();
```

Calling methods

- Some methods return nothing:

```
[frac print];
```

- Some methods return a basic type:

```
float f = [frac floatValue];
```

- Some methods return an object:

```
Fraction *result = [calc add];
```

In Java:

```
frac.print();  
float f = frac.floatValue();  
Fraction result = calc.add();
```

Calling methods 2

- Some methods have no parameters:
`[frac print];`
- Some methods have one parameter:
`[frac setTopNumber:x];`
- Some methods have many parameters:
`[frac setTop:x andBottom:y];`

In Java:

```
frac.print();  
frac.setTopNumber(x);  
frac.setTopAndBottom(x,y);
```

Nested methods

- You can use the result of one method as the object or parameter of another method call:

```
[[calculator add] print];
```

or:

```
[frac setBottomNumber:[frac topNumber]];
```

In Java:

```
calculator.add().print();
```

or:

```
frac.setBottomNumber(frac.topNumber());
```

An object is...
an *instance* of a class

Creating a class

- Two files (same as C++):
 - .h \Rightarrow h header file
 - .m \Rightarrow m implementation file
- Write the header file first!
- Implement second.

Header files

```
#import ...  
  
@interface ClassName : ParentClass {  
    // Declare variables  
    int x;  
}  
  
// Declare methods  
- (void)setX:(int)num;  
  
@end
```

Implementation files

```
#import "ClassName.h"  
  
@implementation ClassName  
  
// Implement methods  
  
- (void)setX:(int)num {  
    x = num;  
  
}  
  
@end
```

Declaring methods

- A method that returns nothing:
 - `(void)print;`
- A method that returns a basic type:
 - `(float)floatValue;`
- A method that returns an object:
 - `(Fraction *)add;`

In Java:

```
public void print() {...}
public float floatValue() {...}
public Fraction add() {...}
```

Declaring methods 2

- A method with no parameters:
 - `(void)print;`
- A method with one parameter:
 - `(void)setTopNumber:(int)x;`
- A method with two parameters:
 - `(void)setTop:(int)x andBottom:(int)y;`

In Java:

```
public void print() {...}
public void setTopNumber(int x) {...}
public void setTopAndBottom(int x, int y) {...}
```

Implementing methods

- If a method is not void then it must return:
 - `(float)floatValue {
 return ((1.0*topNum)/bottomNum);
}`
- Note: a method can have the same name as a variable!

self

- To call a method in the same class:

```
- (float)floatValue {  
    return topNumber / bottomNumber;  
}  
- (void)print {  
    printf("%f", [self floatValue]);  
}
```

this object



In Java: `this.floatValue()`

Style!

- ClassNames
- variableNames and methodNames
- Getters and setters:
 - (int)topNumber;
 - (void)setTopNumber:(int)num;

In Java:

```
public int getTopNumber() {...}  
public void setTopNumber(int x) {...}
```

Common classes

- NSObject
- NSString
- NSArray
- NSMutableArray

A little bit of history

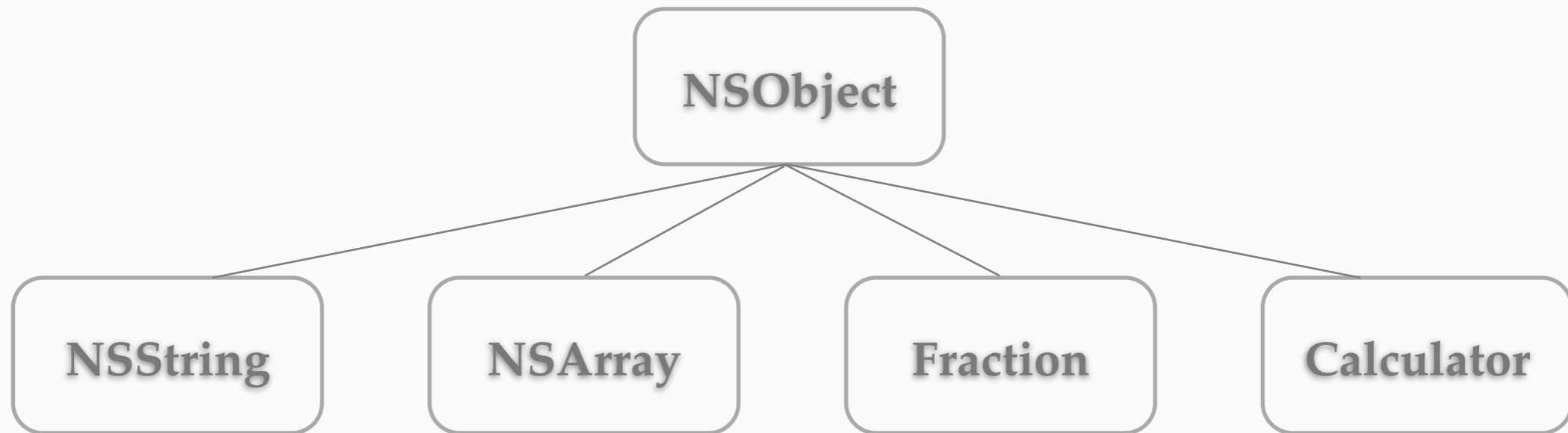
- Objective C is a general OOP language, but it is mostly used on the Mac (and recently iOS)
- When Steve Jobs founded NextStep, he chose Objective C as the programming language
- When he moved back to Apple, Objective C and the NextStep libraries were used for development on the new Mac OS X 10

A little bit of history

- Many objects in Objective C begin with the letters NS (e.g. NSString, NSDate, NSArray)
- Because they come from the NextStep (NS)
- On Mac OS X, the framework is called Cocoa
- On iOS, the framework is called Cocoa Touch (but it is very similar to Cocoa)

NSObject


- Base class
 - Every class *inherits* from NSObject, e.g.:
- more next session*



NSObject

- Fraction is a sub-class of NSObject:

```
@interface Fraction : NSObject {  
    ...  
}  
  
...  
  
@end
```



NSString

- A string is an object, of class NSString

- To create a string:

```
NSString *myString = @"Hello baby"
```

NSString

- To create a string from other strings / ints / floats, use `stringWithFormat`:

```
NSString *myString = [NSString  
stringWithFormat:@"Hello %@, pleased  
%d meet you", @"Alice", 2];
```

- Result: Hello Alice, pleased 2 meet you.

NSString

- stringByAppendingString:

```
NSString *str = @"hello";
```

```
str = [str  
      stringByAppendingString:@" you"];
```

- Result: hello you

NSString

- To find the number of characters, use length:

```
NSString *str = @"imissdiscretemaths";
```

```
int x = [str length];
```

NSString

- To test if two strings are equal:

```
NSString *str = @"hello";
```

```
if ([str isEqualToString:@"hello"]) {
```

```
    // do something
```

```
    ...
```

NSString

- To change the case:

```
NSString *str = @"hello";
```

```
// HELLO
```

```
str = [str uppercaseString];
```

```
// Hello
```

```
str = [str capitalizedString];
```

NSString

- To get a substring:

```
NSString *str1, str2;
```

```
str1 = @"the dog ate my homework";
```

```
str2 = [str1 substringWithRange:  
        NSRange(5, 3)];
```

```
// str2 = @"dog"
```

NSString + Thai

- Objective C is very good for non-English languages
- NSString is UNICODE, so you can use Thai:

```
NSString *str = @"มหาวิทยาลัยนเรศวร";
```

NSString warning

- NSString is not a C string:

```
NSString *str = @"objective c";  
char *old = "c string";
```

- If you want to use printf then you must convert NSString to C string:

```
NSString *test = @"hello";  
printf([test cString]);
```

NSString review

- stringWithFormat (to create a new string)
- stringByAppendingString (concatenate)
- length
- isEqualToString
- lowercaseString, uppercaseString
- substringWithRange

NSString reference

- See the official iOS reference for NSString:
http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/Reference/NSString.html

NSArray

- Static arrays (you cannot add or remove objects)

NSArray

- To create a new array, use `arrayWithObjects`:

```
NSArray *arr = [NSArray  
arrayWithObjects:@"cats", @"drink",  
@"milk", nil];
```

- To count the number of objects:

```
int x = [arr count];    // x = 3
```

NSArray

```
NSArray *arr = [NSArray  
arrayWithObjects:@"cats", @"drink",  
@"milk", nil];
```

- To select an object from the array:

```
NSString *str;
```

```
str = [arr objectAtIndex:0];
```

```
// str = @"cats"
```

NSArray

```
NSArray *arr = [NSArray  
arrayWithObjects:@"cats", @"drink",  
@"milk", nil];
```

- To test if an object is in an array:

```
if ([arr containsObject:@"dog"]) {  
    // do something
```

NSArray + NSString

- To split a string into an array:

```
NSString *str = @"jack,james,joe";
```

```
NSArray *arr = [str  
componentsSeparatedByString:@","];
```

NSArray looping

```
for (int i=0; i < [arr count]; i++)  
{...}
```

or

```
NSEnumerator *enumerator =  
    [arr objectEnumerator];  
while (NSString *obj =  
    [enumerator nextObject]) {...}
```

NSArray review

- arrayWithObjects
- count
- objectAtIndex
- containsObject

NSArray reference

- http://developer.apple.com/library/ios/#DOCUMENTATION/Cocoa/Reference/Foundation/Classes/NSArray_Class/NSArray.html

NSMutableArray

- Dynamic arrays (you can add and remove objects)
- NSMutableArray is a subclass of NSArray (so it has all the methods of NSArray)

NSMutableArray

- To create an empty array:

```
NSMutableArray *arr = [NSMutableArray  
                      arrayWithCapacity:16];
```

- To add an object:

```
NSString *str = @"kitten";  
[arr addObject:str];
```

NSMutableArray

```
NSMutableArray *arr = [NSMutableArray  
                      arrayWithCapacity:16];  
NSString *str = @"kitten";  
[arr addObject:str];
```

- To remove an object:

```
[arr removeObjectAtIndex:0];
```

NSMutableArray

```
NSMutableArray *arr = [NSMutableArray  
                      arrayWithCapacity:16];  
[arr addObject:x];  
[arr addObject:y];
```

- To remove all the objects:

```
[arr removeAllObjects]; // empty
```

NSMutableArray

```
NSMutableArray *arr = [NSMutableArray  
arrayWithObjects:@"dog", @"cat",  
@"fish", @"bear", @"turtle", nil];
```

- To sort an array of strings:

```
[arr sortUsingSelector:  
@selector(compare:)];
```

NSMutableArray

- Arrays can contain any object:

```
NSMutableArray *arr = [NSMutableArray  
                      arrayWithCapacity:16];  
Fraction *frac =  
            [[Fraction alloc] init];  
[arr addObject:frac];
```

NSMutableArray

- Create NSMutableArray from NSArray:

```
NSArray *arr = ...;
```

```
NSMutableArray *mutArr =  
    [arr mutableCopy];
```

NSMutableArray review

- Everything in NSArray, plus...
- arrayWithCapacity
- addObject
- removeObjectAtIndex
- removeAllObjects

NSMutableArray

- http://developer.apple.com/library/ios/#DOCUMENTATION/Cocoa/Reference/Foundation/Classes/NSMutableArray_Class/Reference/Reference.html

WHAT IS OBJECT-ORIENTED PROGRAMMING?

“OOP is dividing a program up into objects, with their own data and methods.”

Why OOP?

- Separate functionality
- Put data and methods together inside 'an object'
- Objects communicate by passing messages (e.g. calling each other's public methods)
- Code re-use

Encapsulation

- A class is the definition of an object (its public methods)
- Other objects communicate with the object only through public methods
- Data is internal to an object
- A programmer can change the internal implementation of an object (without breaking other objects!)

Message passing

- A program is a set of inter-connected objects
- Each object has a role or function in the program
- Objects talk to one another (by calling their public methods) to get jobs done

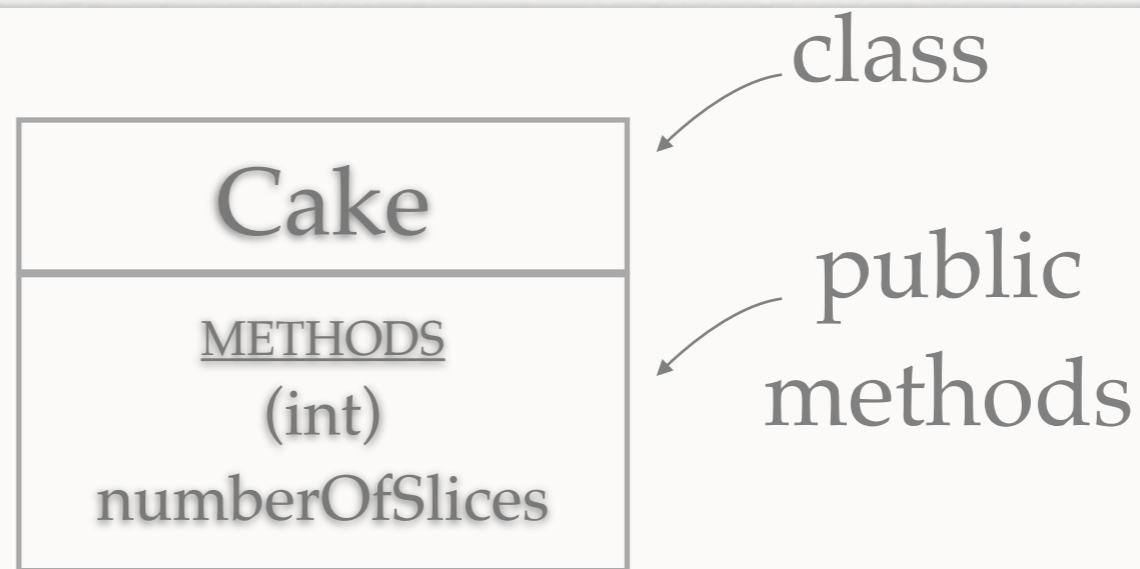
Inheritance

- Create new classes based on other classes
- If you have an existing class X and you create a new class Y as a sub-class of X , then Y *inherits* the definition and implementation of X

Example

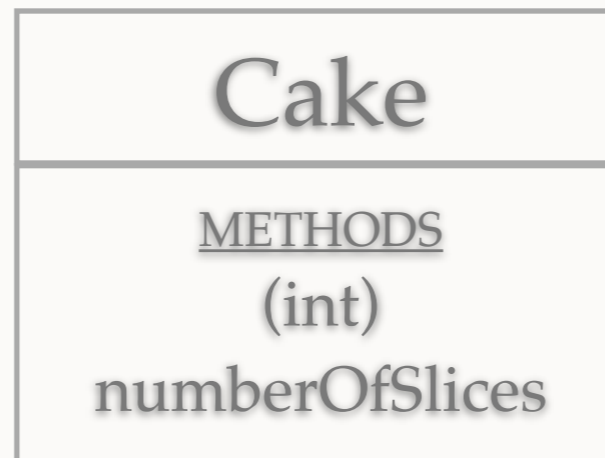
- Sabaii Bakery would like to build an application to manage its business...

Example



```
@interface Cake : ... {  
  
}  
  
- (int) numberOfSlices;  
  
- (Slice *) takeSlice;
```

Example

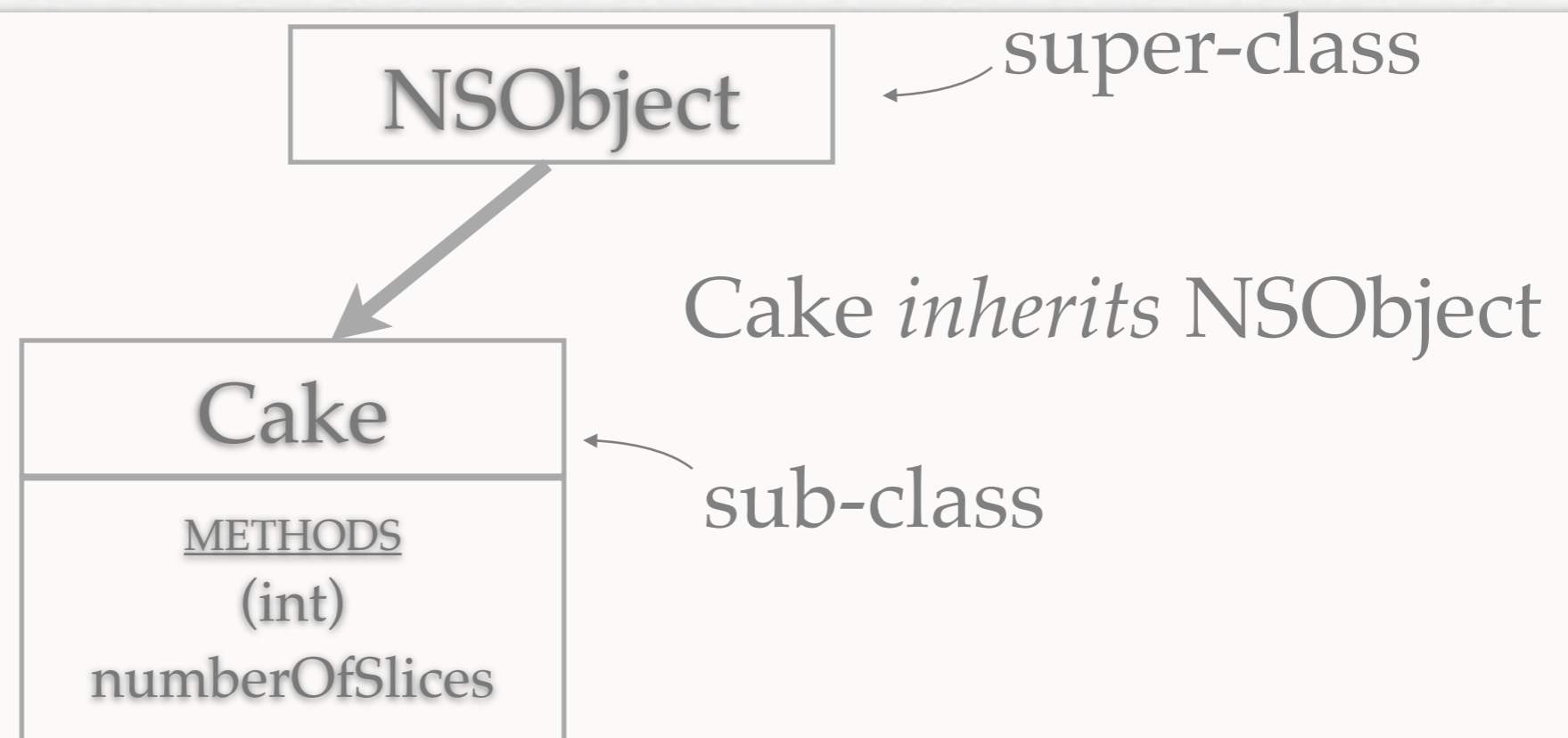


```
Cake *myBigCake = [[Cake alloc] init];
```

```
int x = [myBigCake numberOfSlices];
```

```
Slice *toEat = [myBigCake takeSlice];
```

Example



```
@interface Cake : NSObject {
}
- (int)numberOfSlices;
- (Slice *)takeSlice;
```

Example

NSObject

Cake

METHODS

(int)

numberOfSlices

SpongeCake *inherits* Cake

SpongeCake

METHODS

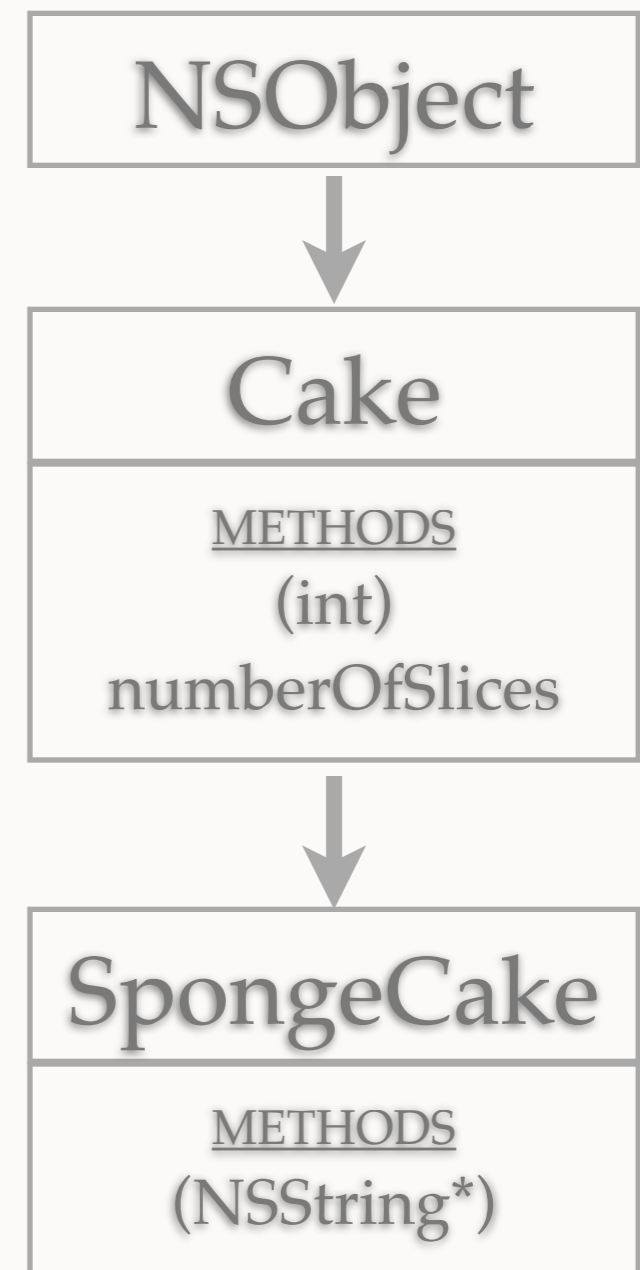
(NSString*)

```
@interface SpongeCake : Cake {  
}  
- (NSString *)jamType;
```

Example

```
SpongeCake *mySponge =  
    [[SpongeCake alloc] init];  
  
printf("%s",  
    [[mySponge jamType] cString]);  
  
Slice *toEat =  
    [mySponge takeSlice];
```

method from Cake!



Example

- If `SpongeCake` *inherits* `Cake` then:
 - `SpongeCake` has its own methods
 - And, it has all the methods on `Cake`
 - And, it has all the methods on `NSObject`
- This is inheritance!

Method overriding

- When a sub-class re-implements a method from a super-class

main.m

```
#import <Foundation/Foundation.h>
#import "Fruit.h"
#import "Orange.h"

int main(void) {

    NSAutoreleasePool *pool =
        [[NSAutoreleasePool alloc]
init];

    Fruit *fruit = [[Fruit alloc] init];
    printf("1: %s\n", [fruit name]);

    Orange *orange = [[Orange alloc] init];
    printf("2: %s\n", [orange name]);

    // Release memory
    [pool release];

    return 0;
}
```

Fruit.h

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

@interface Fruit : NSObject {
}

- (NSString *)name;

@end
```

Fruit.m

```
#import "Fruit.h"

@implementation Fruit

- (NSString *)name {
    return @"FRUIT"
}

@end
```

Orange.h

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

@interface Orange : Fruit {
}

@end
```

Orange.m

```
#import "Orange.h"

@implementation Orange

- (NSString *)name {
    return @"ORANGE"
}

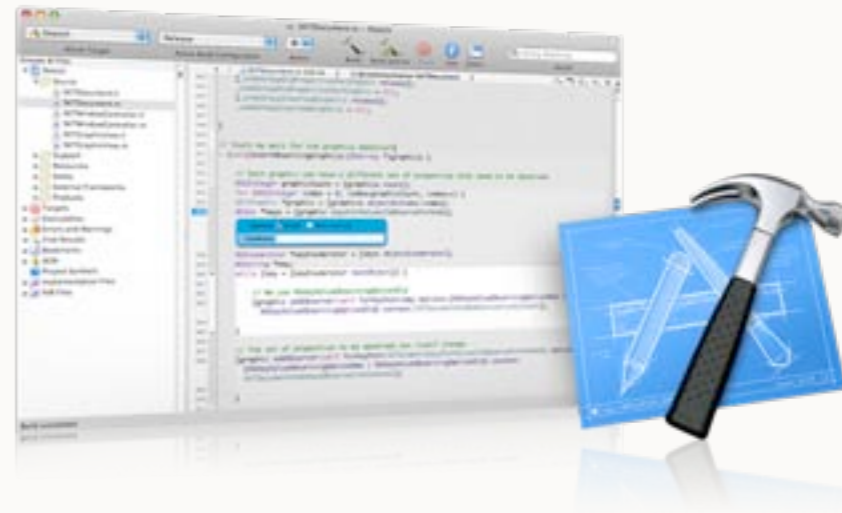
@end
```

Part 2: Xcode

Writing programs

- So far, I made you use a basic text editor and the command line to write programs.
- But there is an easier way...
- Use an IDE!

Xcode



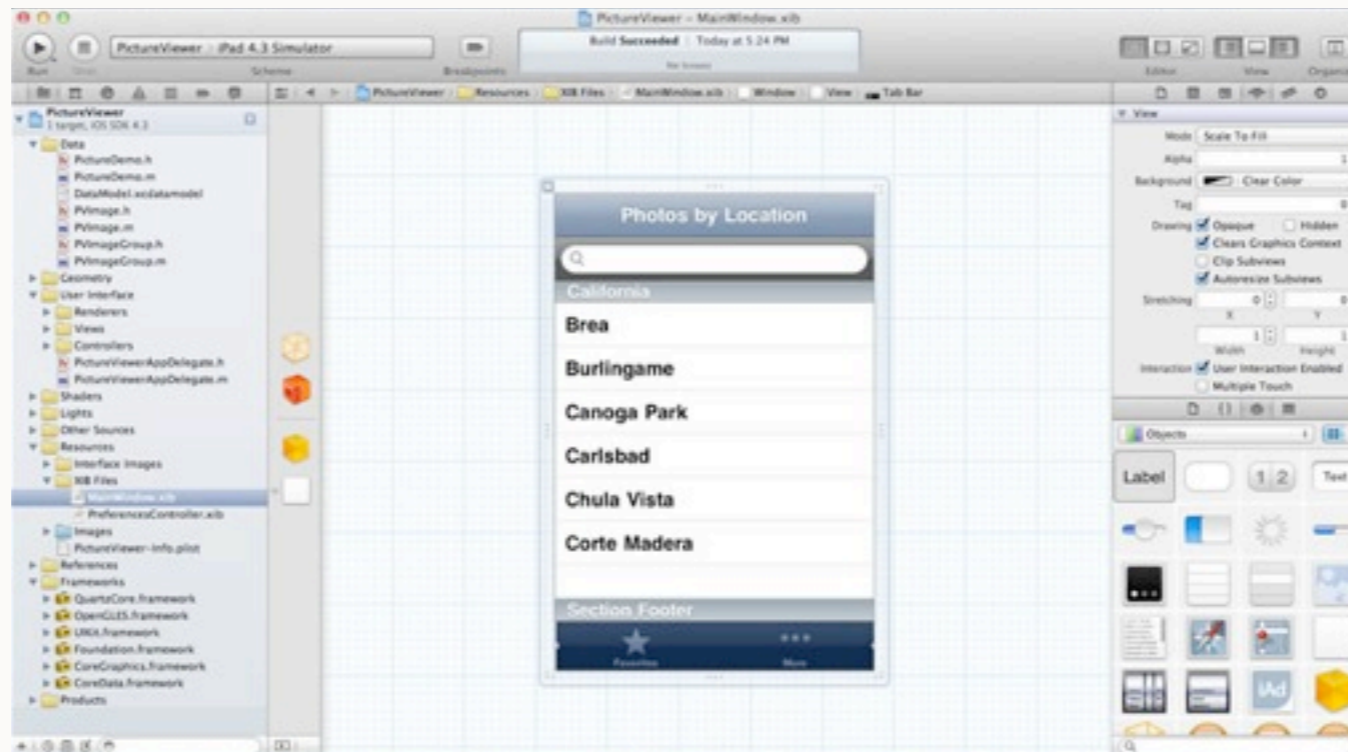
- Xcode is an IDE
(Integrated Development Environment, for writing, running & debugging Mac OS X and iOS apps)
- We are using Xcode 4, and testing our apps on the iOS Simulator

Starting Xcode



Part 3: iPhone Dev

Xcode Interface Builder



- The Xcode Interface Builder is a visual tool for creating UIs
- The UI is stored in a XIB file (sometimes also called a NIB file)

Objective C & OOP

- Objects and classes - what is the difference?
- How do we write a class?
- How do we create an object?

Classes - header files

```
#import ...  
@interface ClassName : ParentClass {  
    // Declare variables  
    int x;  
}  
// Declare methods  
- (void)setX:(int)num;  
- (int)x;  
@end
```

Classes: implementation files

```
#import "ClassName.h"  
@implementation ClassName  
// Implement methods  
- (void)setX:(int)num {  
    x = num;  
}  
- (int)x {  
    return x;  
}  
@end
```

Properties

```
#import ...  
@interface ClassName : ParentClass {  
    // Declare variables  
    int x;  
}  
// Declare methods  
@property(nonatomic,assign) int x;  
@end
```

Synthesize

```
#import "ClassName.h"  
@implementation ClassName  
// Implement methods  
@synthesize x;  
@end
```

Properties and synthesize

```
@property(...) int x;
```



```
- (void)setX:(int)num;  
- (int)x;
```

```
@synthesize x;
```



```
- (void)setX:(int)num {  
    x = num;  
}  
- (int)x {  
    return x;  
}
```

- If you have a `@property` then you [usually] also have a `@synthesize`

Cocoa Touch

- Cocoa Touch is the framework for iOS programming
- It gives you UIButtons, UILabels, UITableViews, etc
- Declare and use like other classes, e.g.

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

App Delegate

- The code where an iOS app starts!
(e.g. ContactsAppDelegate)
- Usually loads one or more UIViewControllers

UIViewController

- Your views are all subclasses of UIViewController (e.g. `ContactsViewController.h/m`)
- Usually has a `.xib` file for the UI (e.g. `ContactsViewController.xib`)
- Contains the method `viewDidLoad` which you can use to load the data into your view

UI Components

- UILabel, UIImageView (simple outlets)
- UIButton, UISlider (have actions)
- UITableView (must have methods to load the data source)

Connections - IBOutlet

- E.g. connect a label in the XIB to an object in the code
- Must have:
 - Variable declaration
 - Property and synthesize
 - Connection in XIB... 'Referencing Outlet'

Connections - IBActions

- E.g. connect a button press to a method in the code
- Must have:
 - A method declaration in the header:
 - `(IBAction)doSomething;`
 - A method implementation
 - A connection in the XIB from the event (Touch Up Inside) to the method

UINavigationController

- Allows you to load one UIViewController on top of another (like a stack)



Summary

What have we covered?

- The Objective C language
 - OOP, data types, objects like NSString, NSArray
- The Xcode IDE, Interface builder & iOS Simulator
 - Creating, writing and running projects
- Basic iOS applications
 - Buttons, labels, sliders, views, navigation, tables...

Why?

- Mobile development is very popular...
- ...and it is still new, so companies are looking for people with skills in iOS, Android, Blackberry, etc
- You should learn about these technologies in your free time!

That's it!