

# Chapter 3

Basic Objective-C

# Topics

- Objective-C Class, Object
- Method, Message
- Accessor, Property
- Example

# Objective-C

- ⦿ Objective-C เป็นภาษาที่เป็น superset ของภาษา ANSI C programming language เพราะฉะนั้น syntax พื้นฐานจึงเหมือนกับภาษา C.
- ⦿ เช่นเดียวกับภาษา C โค้ดของ Objective-C จะถูกแบ่งออกเป็น header file และ source file โดยที่
  - ⦿ Case Sensitive
  - ⦿ Header file จะเป็นไฟล์ที่มีนามสกุล .h ใช้เพื่อประกาศ class, type, function หรือ constant ต่างๆ
  - ⦿ Source file จะเป็นไฟล์ที่มีนามสกุล .m สำหรับ implement code รองรับทั้งภาษา Objective-C และ C
  - ⦿ Source file นามสกุล .mm จะเป็น source files ที่มี code ภาษา C++ นอกเหนือไปจาก Objective-C และ C code. แต่ในการพัฒนา iOS App เรามักจะไม่ได้ใช้

# Header file (.h) & Implementation File (.m)

```
// Customer.h                                // Customer.m

@interface Customer : NSObject
{
    NSString * firstName;
    NSString * lastName;
}

- (void)setFirstName:(NSString *)fname;
- (void)setLastName:(NSString *)lname;

@end                                        

#import "Customer.h"

@implementation Customer
{
    NSString * firstName;
    NSString * lastName;
}

- (void)setFirstName:(NSString *)fname
{
    firstName = fname;
}

- (void)setLastName:(NSString *)lname
{
    lastName = lname;
}

@end
```

# Method vs. Function

- ◉ Objective-C มาจากภาษา C เพราะฉะนั้นจึงมีคุณสมบัติของภาษา C มาด้วย เช่น การประกาศ variable และ function
- ◉ การประกาศ method อาจจะไม่ต้องประกาศใน header file ก็ได้ แต่ class อื่นที่เรียกใช้ method จะไม่เห็นถ้า import header file ไปใช้ แต่จะเห็น method นั้นถ้า import implementation file (.m) ไปใช้
- ◉ ถ้าไม่มีการประกาศใน header แม้ว่า method นั้นจะอยู่ใน implementation file เดียวกันแต่ถูกประกาศไว้ด้านล่างของ code ที่เรียกใช้ code ที่เรียกจะมองไม่เห็น method นั้น

# Method vs. Function

## ⦿ ตัวอย่าง

```
@implementation  
-(void)a  
{  
    b(); // <- error  
}  
  
-(void)b  
{  
    // do something;  
}  
  
@end
```

```
@implementation  
-(void)b  
{  
    // do something;  
}  
  
-(void)a  
{  
    b(); // OK  
}  
  
@end
```

# Classes

- การประกาศ class ในภาษา Objective-C นั้นจะต้องประกาศ 2 ส่วน คือ interface และ implementation.
  - ในส่วนของ Interface นั้นจะเป็นส่วนของการประกาศ class และกำหนด ตัวแปรแบบ instance variables รวมไปถึง methods ของ class นั้น ปกติ แล้วจะอยู่ในไฟล์ .h (อยู่ใน .m ก็ได้)
  - ในส่วนของ implementation นั้นจะเป็นส่วนของ code ที่ถูก implement จริงใน methods ที่ประกาศไว้ใน interface ปกติแล้ว implementation code จะอยู่ในไฟล์นามสกุล .m

# Classes (2)

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

Member variable declarations

Method declarations

```
MyClass *myObject1; // Strong typing
id      myObject2; // Weak typing
```

# Declare Instance and Create Object

- การประกาศ instance

```
Customer * c;
```

- การ Allocate (จอง memory) และ initiate default value ของ object.

```
Customer * c = [[Customer alloc] init];
```

- Release (ฟรี object, ถูกยกเลิกไปเมื่อกำหนดเป็น ARC)

```
[c release];
```

- รูปแบบอื่นๆ (แต่ไม่แนะนำและไม่เป็นที่นิยม)

```
car = [Car alloc];
```

```
car = [Car init];
```

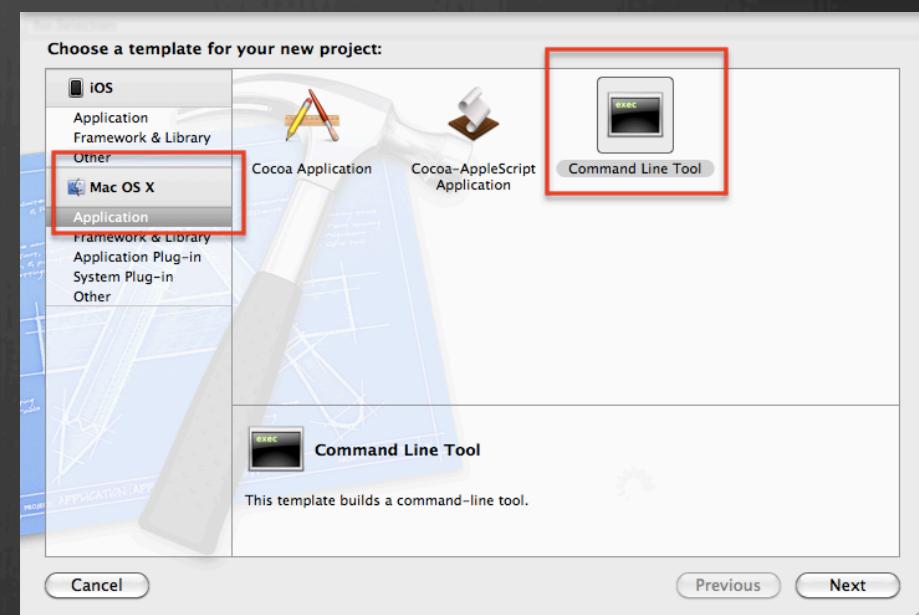
```
Car *car = [Car new];
```

# Example : Hello, My object

1. สร้าง project ใหม่

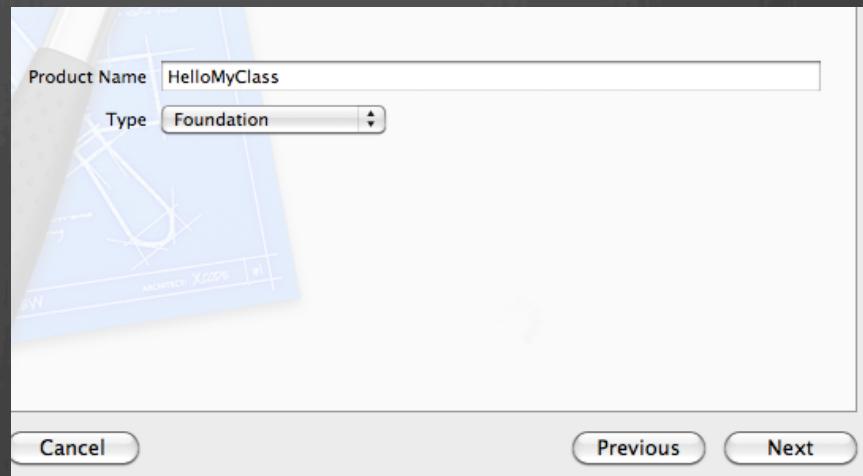


2. เลือกประเภท project เป็น  
Command Line Tool



# Example : Hello, My object (2/6)

3. ตั้งชื่อ project ว่า HelloMyClass และเลือก type เป็น Foundation



4. Ready to develop

A screenshot of the Xcode interface. The Project Navigator on the left shows a single target named 'HelloMyClass'. The Editor on the right displays the 'main.m' file with the following code:

```
// main.m
// HelloMyClass
//
// Created by Olarn U. on 4/4/11.
// Copyright 2011 __MyCompanyName__. All rights reserved.

#import <Foundation/Foundation.h>

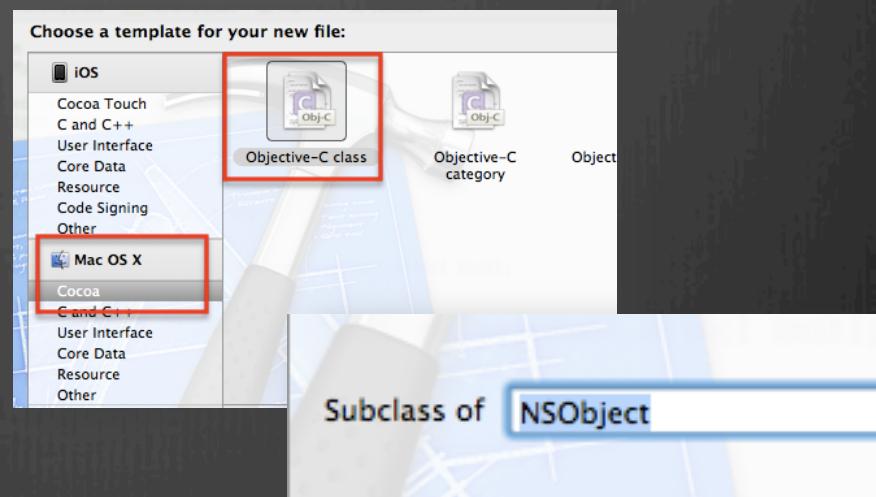
int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // insert code here...
    NSLog(@"Hello, World!");

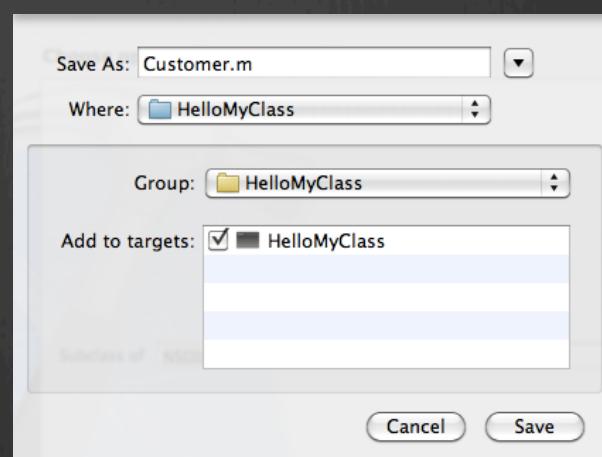
    [pool drain];
    return 0;
}
```

# Example : Hello, My object (3/6)

5. สร้าง class ใหม่จากเมนู  
File -> New -> New File...



6. ระบุ Subclass เป็น NSObject



7. ตั้งชื่อ class ว่า “Customer”  
แล้ว click “Save”

# Example : Hello, My object (4/6)

8. เขียน interface ของ class ในไฟล์ Customer.h

```
@interface Customer : NSObject
{
    NSString * firstName;
    NSString * lastName;
}

- (void)setFirstName:(NSString *)fName;
- (void)setLastName:(NSString *)lName;
- (NSString *)getFullName;

@end
```

# Example : Hello, My object (5/6)

9. เขียนโค้ด ในส่วน implementation ของ class ในไฟล์ Customer.m

```
#import "Customer.h"

@implementation Customer

- (void)setFirstName:(NSString *)fName {
    firstName = fName;
}

- (void)setLastName:(NSString *)lName {
    lastName = lName;
}

- (NSString *)getFullName {
    return [NSString stringWithFormat:@"%@ %@", firstName, lastName];
}

@end
```

# Example : Hello, My object (6/6)

## 10. Modify main.m

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

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

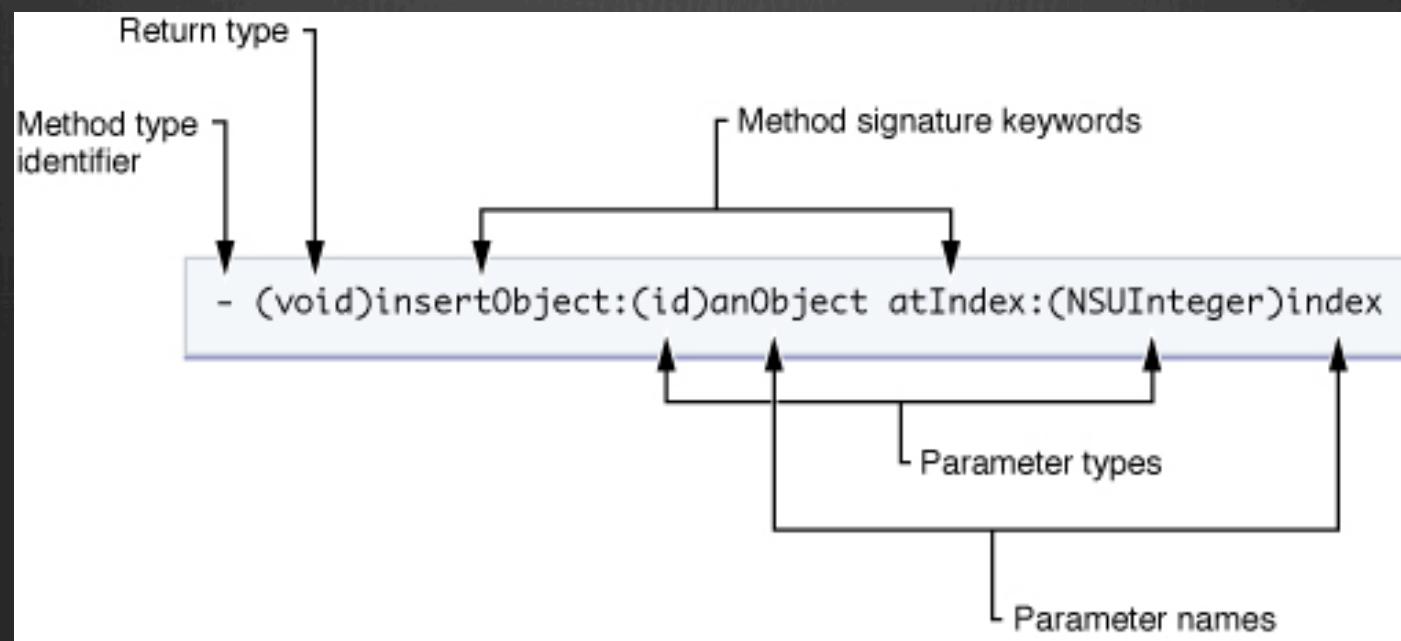
        Customer *c = [[Customer alloc] init];
        [c setFirstName:@"Harry"];
        [c setLastName:@"Potter"];

        NSLog(@"%@", [c getFullName]);
    }
    return 0;
}
```

## 11. Run เพื่อดูผลลัพธ์

# Methods and Messaging

- A class in Objective-C can declare two types of methods: instance methods and class methods.



*- instance method  
+ class method*

# Messaging an object

```
[myArray insertObject:anObject atIndex:0];
```

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

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

```
myAppObject.theArray = aNewArray;
```

# Declared Properties

- Declared properties are a convenience notation used to replace the declaration and, optionally, implementation of accessor methods.

```
@property BOOL flag;  
  
@property (copy) NSString *nameObject;  
// Copy the object during assignment.  
  
@property (readonly) UIView *rootView;  
// Declare only a getter method.  
  
@property(retain,  
          getter = firstName, setter = setFirstName,  
          nonatomic, readwrite) NSString *name;  
  
@property(assign,  
          getter = idCustomer, setter = setIDCustomer)  
NSInteger id;
```

# Example : Hello, My object (7)

12. สร้าง class ใหม่ ชื่อ Customer2

13. เพิ่ม code ใน Customer2.h

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

@interface Customer2 : NSObject

@property(nonatomic, strong)NSString * firstName;
@property(nonatomic, strong)NSString * lastName;

- (NSString *)getFullName;

@end
```

# Example : Hello, My object (8)

14. เพิ่ม implementation code ในไฟล์ Customer.m

```
#import "Customer.h"

@implementation Customer

- (NSString *)getFullName
{
    return [NSString stringWithFormat:@"%@ %@",  

            self.firstName,  

            self.lastName];
}

@end
```

# Example : Hello, My object (9)

15. แก้ไฟล์ main.m เพื่อสร้าง object Customer2 แทน

```
#import <Foundation/Foundation.h>
#import "Customer2.h"

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

        Customer2 *c = [[Customer2 alloc] init];
        c.firstName = @"Harry";
        c.lastName = @"Potter";

        NSLog(@"%@", [c getFullName]);
    }
    return 0;
}
```

# Behind the Scene!

- ใน version ก่อน การประกาศ property นั้น เราจะต้องสร้าง attribute ภายใน class และประกาศ `@synthesize` ด้วย
- แต่ตั้งแต่ Xcode 4.4 (iOS5) เป็นต้นมา เราไม่ต้องเขียนอีกแล้ว เพราะ Xcode จะสร้าง temporary code ขึ้นมาและเพิ่ม `@synthesize` ให้เราในระหว่างการ compile

```
// Customer.h

@interface Customer2 : NSObject
{
    NSString * firstName;
    NSString * lastName;
}

@property(nonatomic, strong)NSString * firstName;
@property(nonatomic, strong)NSString * lastName;

- (NSString *)getFullName;

@end
```

```
// Customer.m

#import "Customer2.h"

@implementation Customer2

@synthesize firstName;
@synthesize lastName;

@end
```

- ความสามารถอ้างถึง attribute โดยไม่ต้องระบุ self instance โดยใช้ `_` (under scroll) ขึ้นหน้าชื่อ property เช่น `_firstName`, `_lastName` เป็นต้น

# @property attributes

- Writability

Attribute	Effect
readwrite	<ul style="list-style-type: none"><li>• Indicates that the property should be treated as read/write.</li><li>• This attribute is the <b>default</b>.</li></ul>
readonly	<ul style="list-style-type: none"><li>• If you specify readonly, only a getter method is required in the @implementation block.</li><li>• If you use the @synthesize directive in the implementation block, only the getter method is synthesized.</li><li>• Moreover, if you attempt to assign a value using the dot syntax, you get a compiler error.</li></ul>

```
@property (readonly) NSString * fullName;
```

# @property attributes

- Setter Semantics (obsoleted in Xcode5)

Attribute	Effect
assign	<ul style="list-style-type: none"><li>• Specifies that the setter uses simple assignment.</li><li>• You typically use this attribute for scalar types.</li><li>• This attribute is the default.</li><li>• Not support in ARC</li></ul>
retain	<ul style="list-style-type: none"><li>• Specifies that retain should be invoked on the object upon assignment.</li><li>• The previous value is sent a release message.</li><li>• Not support in ARC</li></ul>
copy	<ul style="list-style-type: none"><li>• Specifies that a copy of the object should be used for assignment.</li><li>• The previous value is sent a release message.</li></ul>

# @property attributes

- Setter Semantics (for ARC & Xcode5 and later)

Attribute	Effect
weak	<ul style="list-style-type: none"><li>• For ARC, to replace assign</li><li>• Specifies that the setter uses simple assignment.</li><li>• You typically use this attribute for scalar types.</li></ul>
strong	<ul style="list-style-type: none"><li>• For ARC, replace retain</li><li>• Specifies that retain should be invoked on the object upon assignment.</li><li>• The previous value is sent a release message.</li></ul>

# @property attributes

- Atomicity

Attribute	Effect
nonatomic	<ul style="list-style-type: none"><li>• A synthesized accessor for an object property simply returns the value directly.</li></ul>
atomic	<ul style="list-style-type: none"><li>• For multi-thread environment.</li><li>• Provide robust access to properties in a multithreaded environment.</li><li>• The value returned from the getter or set via the setter is always fully retrieved or set regardless.</li><li>• This attribute is the <b>default</b>.</li></ul>