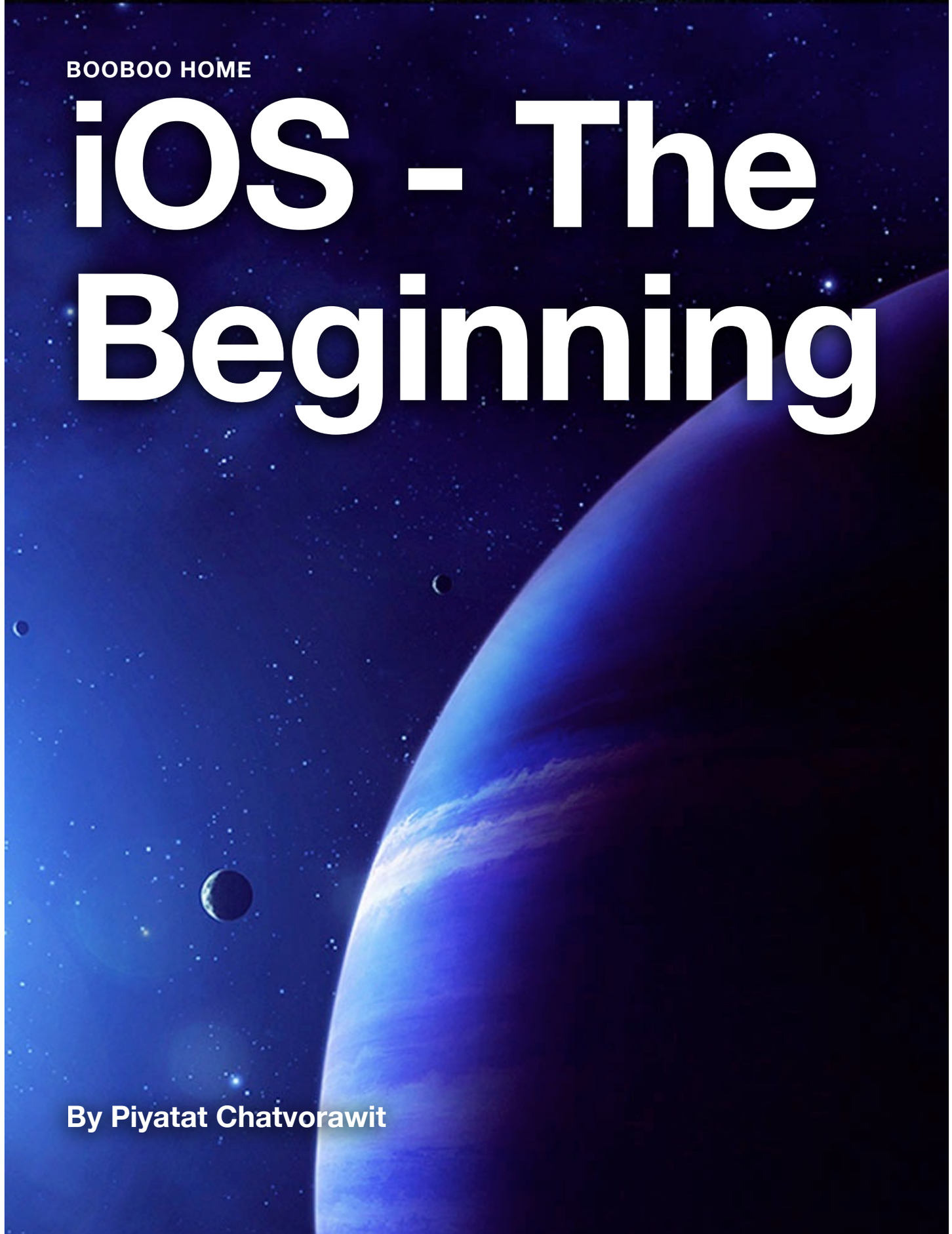
BOOBOO HOME
iOS - The Beginning
By Piyatat Chatvorawit

Chapter 5

Objective C - About Class

ในบทนี้เราจะเปลี่ยนกลับมาพูดเกี่ยวกับภาษา Objective C กันบ้าง โดยสิ่งที่เราจะพูดถึงกันในบทนี้ก็จะเป็นเรื่องเกี่ยวกับ class, โครงสร้างของ class, วิธีการสร้าง class, property, category

CLASS STRUCTURE

1. คลาสจะแบ่งออกเป็น 2 ส่วนคือ interface และ implementation
2. Interface จะประกาศไว้ในไฟล์ .h จะแบ่งออกเป็น 3 ส่วนหลักๆ ได้แก่
    1. ส่วนประกาศตัวแปร
    2. ส่วนประกาศ property
    3. ส่วนประกาศเมธอด
3. Implementation จะประกาศไว้ในไฟล์ .m ซึ่งจะเป็นส่วนที่ไว้เขียนการทำงานของเมธอดต่างๆที่เราประกาศเอาไว้

ในบทนี้เราจะพูดถึงเรื่องของคลาสในภาษา Objective C กัน ซึ่งหากเคยเขียนโปรแกรมในลักษณะของ Object-Oriented มา ก็คงจะคุ้นเคยแนวคิดเรื่องคลาสกันมาแล้ว แต่สำหรับผู้ที่ยังไม่เคยเขียนโปรแกรมในลักษณะนี้มาก่อน ก็อาจจะต้องไปลองหาหนังสือเกี่ยวกับการเขียนโปรแกรมเชิงวัตถุมาลองอ่านดูซะหน่อย เพื่อให้เข้าใจแนวคิดเกี่ยวกับการเขียนโปรแกรมเชิงวัตถุ ซึ่งโดยหลักๆแล้วการเขียนโปรแกรมในลักษณะนี้จะมองโปรแกรมเป็นวัตถุ ซึ่งประกอบขึ้นมาจากวัตถุหลายๆวัตถุ คลาสจะเป็นเหมือนกับต้นแบบที่ใช้ในการสร้างวัตถุขึ้นมาใช้ในโปรแกรมนั่นเอง

จากการที่ภาษา Objective C จะแบ่งไฟล์ของโปรแกรมออกเป็น 2 ส่วนหลักๆ ก็คือไฟล์ .h และไฟล์ .m ดังนั้นโดยทั่วไปแล้ว คลาสก็จะถูกประกาศโดยแยกออกเป็น 2 ส่วนเช่นเดียวกับคือ

1. Interface ของคลาส เก็บไว้ที่ไฟล์ .h ซึ่งส่วนนี้จะเป็นเหมือนกับส่วนประกาศลักษณะคุณสมบัติ สมาชิก และเมธอดต่างๆของคลาสเอาไว้ เพื่อให้ตัวโปรแกรมสามารถเรียกใช้คลาสนี้ได้ เนื่องจากเราจะทำการ include เฉพาะไฟล์ .h เข้ามาในโปรแกรมเท่านั้น ทำให้เราสามารถเรียกใช้งานคุณสมบัติต่างๆของคลาสได้เฉพาะที่ประกาศไว้ในนี้เท่านั้น

ตัวอย่างนี้จะเป็น interface ของคลาส HelloWorldViewController ที่เคยนำมายกตัวอย่างกันไปแล้วในบทก่อนหน้านี้

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

@interface HelloWorldViewController : UIViewControl-
ler

- (void)setUI;

@end
```

2. Implementation ของคลาส เก็บไว้ที่ไฟล์ .m ซึ่งจะเป็นส่วนโค๊ดหลักของตัวคลาส โดยหลักๆก็จะเป็นส่วนที่เขียนการทำงานของเมธอดต่างๆ ที่ประกาศในส่วนของ Interface

ตัวอย่างนี้จะเป็นบางส่วนของ implementation ของคลาส HelloWorldViewController

```
#import "HelloWorldViewController.h"

@implementation HelloWorldViewController

//...

@end
```

ในไฟล์ .h นั้น เราสามารถที่จะเขียน Interface ของคลาสลงไปได้มากกว่า 1 คลาส เช่นเดียวกับไฟล์ .m ที่เราสามารถเขียน Implementation ของคลาสลงไปได้มากกว่า 1 คลาส ดังตัวอย่างนี้

```
@interface MyClass : NSObject

//...

@end

@interface MyClass2 : NSObject

//...

@end
```

ที่นี้เราจะมาพูดถึงรายละเอียดในส่วนของ interface กัน โดยทั่วไปแล้ว interface ของคลาสจะประกอบด้วย 4 ส่วนหลักๆด้วยกัน ดังนี้

1. ชื่อคลาส และคลาสที่เป็น superclass โดยทั่วไปแล้วเรานิยมใช้การตั้งชื่อคลาสโดยใช้ตัวอักษรแรกของคำเป็นตัวอักษรใหญ่ตามด้วยตัวอักษรเล็ก อย่างเช่น MyClass เป็นต้น ซึ่งชื่อคลาสนี้ ไม่จำเป็นที่จะต้องตั้งให้เหมือนกับชื่อไฟล์ ที่เก็บ interface นี่ไว้ ส่วน superclass คือ คลาสที่เราจะทำการ subclass เข้ามาสร้างเป็นคลาสใหม่

2. ตัวแปรของคลาส โดยระบุชนิดของตัวแปร และชื่อของตัวแปร ซึ่งตัวแปรที่เหล่านี้จะมองเห็น และเข้าถึงโดยตรงได้เฉพาะภายในคลาสนี้เท่านั้น

3. property ของคลาส จะคล้ายคลึงกับตัวแปรของคลาส เพียงแต่ property จะสามารถเข้าถึงได้โดยตรงจากภายนอกคลาส ซึ่งโดยปกติเราจะใช้ควบคู่ไปกับตัวแปร โดยจะใช้ property เชื่อมกับตัวแปรเพื่อให้เราสามารถเข้าถึงตัวแปรได้จากภายนอกคลาส

4. เมธอดของคลาส โดยเราสามารถประกาศเมธอดได้ 2 ประเภทด้วยกัน คือ class method และ instance method ซึ่ง class method เราสามารถเรียกใช้ได้โดยไม่ต้องทำการสร้าง object ของคลาสนี้ขึ้นมา

ก่อน ส่วน instance method นั้นเราจะต้องเรียกผ่านทาง object ของคลาสนี้

โดยโครงสร้างของ interface จะเป็นดังนี้

```
@interface ClassName : SuperClassName
{
    // Variable Declaration
}

// Property Declaration

// Method Declaration

@end
```

ตัวอย่างต่อไปนี้จะเป็นตัวอย่างการประกาศ interface

```
@interface MyClass : NSObject
{
    int aVar;
    float bVar;

    NSMutableArray *myList;
}

@property (nonatomic, assign, readwrite) int aVar;
@property (nonatomic, assign, readonly) float bVar;
@property (nonatomic, assign, readwrite) int cVar;

- (int)methodA;
- (double)methodB;

@end
```

ในตัวอย่างนี้ จะเป็นการประกาศ interface สำหรับคลาส MyClass ซึ่งทำการ subclass มาจากคลาส NSObject

ใน interface นี่เราได้ทำการประกาศตัวแปร 3 ตัวด้วยกัน คือ aVar, bVar และ myList ซึ่งมีเป็นตัวแปรชนิด int, float และ NSMutableArray ตามลำดับ

ในส่วนต่อมาก็จะเป็นการประกาศ property ซึ่งเราได้ประกาศ property ไว้ 3 ตัวด้วยกัน ได้แก่ aVar, bVar และ cVar ซึ่งจะเห็นว่า 2 ตัวแรกนั้นเราตั้งชื่อเหมือนกับตัวแปร เพื่อใช้เชื่อมกับตัวแปรให้สามารถเข้าถึงได้จากภายนอกคลาส สำหรับค่าต่างๆที่ใช้กำหนดในการประกาศ property นั้นจะกล่าวถึงอีกทีในภายหลัง

ส่วนสุดท้ายก็จะเป็นส่วนของการประกาศเมธอด ซึ่งเราได้ประกาศเมธอดไว้ 2 เมธอด คือ methodA และ methodB

MEMBER OF CLASS

1. **Variable ตัวแปรต่างๆ ที่ใช้งานภายในคลาส ซึ่งโดยปกติจะสามารถเข้าถึงได้เฉพาะภายในคลาส หากต้องการเข้าถึงจากภายนอกคลาส โดยทั่วไปเราจะเขียนเป็นในลักษณะของ getter/setter method**
2. **Property ช่วยอำนวยความสะดวกในการเขียน getter/setter method สำหรับตัวแปร และช่วยให้เราสามารถใช้ . (dot syntax) ในการเข้าถึงตัวแปรได้อีกด้วย**
3. **Method มีด้วยกัน 2 ประเภทหลักๆ คือ class method และ instance method**

จากที่ได้กล่าวไปแล้วในหัวข้อที่แล้วว่าในคลาสนึงนั้น จะประกอบได้ด้วยสมาชิก 3 แบบหลักๆ ด้วยกัน นั้นคือ

1. Variable ตัวแปรต่างๆ โดยตัวแปรเหล่านี้จะถูกนำมาใช้งานภายในคลาสที่ประกาศเอาไว้ ซึ่งปกติแล้วเราจะไม่สามารถเข้าถึงตัวแปรเหล่านี้จากภายนอกคลาสได้ หากเราต้องการจะเข้าถึงตัวแปรต่างๆเหล่านี้ โดยปกติแล้วเราจะทำการสร้างเมธอดสำหรับใช้ในการอ่านค่าและกำหนดค่าให้กับตัวแปรเหล่านี้ โดยเมธอดในลักษณะนี้จะถูกเรียกว่า accessor method หรือ getter/setter method

ในภาษา Objective C นั้น เราจะนิยมตั้งชื่อ getter/setter method ให้เป็นชื่อเดียวกับตัวแปรที่เราจะเข้าถึง โดยเราจะตั้งชื่อ getter method เป็นชื่อเดียวกับชื่อตัวแปร และ setter method เป็น set+ชื่อตัวแปร

```
- (int)myVar;
- (void)setMyVar:(int)aMyVar;
```

ในตัวอย่างด้านบนนี้ จะกำหนด getter method ชื่อ myVar และกำหนด setter method เป็น setMyVar

โดยทั่วไปในเมธอดเหล่านี้ ก็จะทำเพียงแค่คืนค่าตัวแปรนั้นๆออกมา และกำหนดค่าให้กับตัวแปรนั้นๆเท่านั้น

```
- (int)myVar
{
    return myVar;
}

- (void)setMyVar:(int)aMyVar
{
    myVar = aMyVar;
}
```

ในตัวอย่างด้านบนนี้ ในเมธอด myVar จะทำเพียงแค่คืนค่าของตัวแปร myVar ออกมา ในขณะที่เมธอด setMyVar จะนำค่าพารามิเตอร์ที่ส่งเข้ามาไปกำหนดค่าให้กับตัวแปร myVar

ถ้าหากเราต้องการเข้าถึงตัวแปรเป็นจำนวนมาก การมานั่งเขียนเมธอดเหล่านี้อาจจะทำให้เสียเวลาไปพอสมควร ซึ่งในจุดนี้ภาษา Objective C ก็จะมี property เข้ามาเพื่อช่วยแก้ปัญหาในจุดนี้

2. Property เพื่อลดภาระในการเขียนโค๊ดที่ซ้ำซ้อนกัน ภาษา Objective C จึงได้มี property ขึ้นมาเพื่อช่วยแก้ปัญหาในส่วนนี้ โดย setter/getter method สำหรับ property นั้นตัว compiler จะทำการสร้างขึ้นมาให้โดยอัตโนมัติ ซึ่งในจุดนี้ หากเราต้องการกำหนดการทำงานโดยละเอียดก็สามารถทำได้เช่นกัน อย่างเช่น ต้องการแค่ getter method อย่างเดียว หรือใน setter method ต้องการทำการคำนวนบางอย่างก่อนกำหนดค่าลงไป เป็นต้น

การประกาศ property ขึ้นมานั้น จะมีรูปแบบดังนี้

```
@property (<Property Attributes>) <Type> <Property-
Name>;
```

ซึ่งเราสามารถกำหนดคุณสมบัติต่างๆให้กับ property ได้ดังนี้

(1) Accessor Method Name เราสามารถที่จะกำหนดชื่อของ accessor method สำหรับ property ให้เป็นชื่ออื่นๆได้ โดยใช้คำสั่งดังนี้

```
getter=<MethodName>
setter=<MethodName>
```

ซึ่งถ้าหากเราไม่กำหนดค่านี้ จะถือเป็นการใช้ชื่อ accessor method ตามลักษณะที่กล่าวมาในช่วงแรก (คือใช้เป็นชื่อเดียวกับ property)

(2) Writability สำหรับกำหนดการเข้าถึง property ตัวนี้ โดยเราสามารถกำหนดได้ 2 แบบคือ

(1) readwrite สำหรับให้สามารถอ่าน และเขียนค่าให้กับ property นี้ได้

(2) readonly สำหรับให้สามารถอ่านค่าได้เพียงอย่างเดียว ซึ่งหากกำหนดคุณสมบัตินี้ ตัว compiler ก็จะไม่ทำการสร้าง setter method ขึ้นมา

(3) Setter Semantic สำหรับกำหนดรูปแบบการกำหนดค่าให้กับ property โดยเราสามารถกำหนดค่าได้ดังนี้

(1) strong กำหนดความเป็นเจ้าของให้กับ object ที่นำมากำหนดให้กับ property ซึ่งจะ keyword สำหรับใช้กับ ARC

(2) weak กำหนดให้ไม่มีความเป็นเจ้าของให้กับ object ที่นำมากำหนดให้กับ property (นั่นคือ ถ้า object นั่นถูก deallocate ค่า property ก็จะถูกกำหนดให้เป็น nil โดยอัตโนมัติ) ซึ่งจะเป็น keyword สำหรับใช้กับ ARC

(3) copy กำหนดให้ทำการ copy object มาให้กับ property โดยถ้าหาก property มีค่าอื่นอยู่ ค่านั้นจะถูกเรียก release (โดยปกติจะใช้กับตัวแปรของคลาส NSString)

(4) assign กำหนดให้ทำการกำหนดค่าให้กับ property โดยตรง (โดยปกติจะใช้กับตัวแปรพื้นฐาน เช่น int, float เป็นต้น) โดยค่านี้จะเป็น default หากเราไม่กำหนดค่า setter semantic ใดๆ

(5) retain กำหนดให้ทำการเรียก retain ให้กับ object ที่นำมากำหนดให้กับ property โดยถ้าหาก property มีค่าอื่นอยู่ ค่านั้นจะถูกเรียก release

(4) Atomicity กำหนดความเป็น atomic ให้กับ property โดยสามารถกำหนดค่าได้ดังนี้

(1) nonatomic กำหนดให้ property นี้เป็น nonatomic

(2) ไม่กำหนดค่า โดยถ้าไม่กำหนดค่าจะถือว่าเป็น atomic ซึ่งก็คือเป็นการรับรองว่า หากมีการเข้าถึง property นี้จากหลายๆ thread พร้อมกัน ทุกๆ thread จะสามารถอ่านค่า หรือกำหนดค่าให้กับ property ได้อย่างแน่นอน แต่ไม่รับรองลำดับในการทำงานของโปรแกรม นั่นคือ หาก A อ่านค่า ในขณะที่ B ทำการกำหนดค่าให้กับ property เดียวกันนั้น A อาจจะได้ค่าเดิม หรือค่าใหม่ก็ได้

```
@interface MyClass : NSObject
{
    int myVar;
}

@property (nonatomic, assign, readwrite) int myVar;

@end
```

ในตัวอย่างนี้จะเป็นการประกาศ property ชื่อ myVar ที่มีชนิดเป็น int โดยกำหนดคุณสมบัติเป็น nonatomic, assign และ readwrite

เมื่อเราทำการประกาศ property เสร็จเรียบร้อยแล้ว เราจะต้องเขียนคำสั่งให้ compiler ทำการสร้าง property ขึ้นมาตามที่เรากำหนดไว้ โดยจะใช้คำสั่งดังนี้

```
@synthesize <PropertyName>;
```

หรือ

```
@dynamic <PropertyName>;
```

ซึ่งคำสั่ง 2 อันนี้จะแตกต่างกันตรงที่ @synthesize จะเป็นการบอกให้ compiler ทำการสร้าง getter/setter method ให้โดยอัตโนมัติ ในขณะที่ @dynamic จะเป็นการบอกว่า getter/setter method จะถูกสร้างขึ้นจากส่วนอื่น (สร้างขึ้นเอง, ถูกประกาศไว้ที่อื่น, ...) ให้ compiler ไม่ต้องทำในส่วนนี้ ซึ่งโดยปกติเราจะใช้คำสั่ง @synthesize สำหรับคำสั่ง @dynamic นั้นเราจะใช้กับคลาสที่ subclass มาจาก NSManagedObject ซึ่งจะกล่าวถึงในส่วนของ CoreData ในภายหลัง

โดยจะนำไปประกาศไว้ใน implementation ของคลาส ตัวอย่างเช่น

```
@implementation MyClass
@synthesize myVar;
@end
```

หรือ

```
@implementation MyClass
@dynamic myVar;
@end
```

จริงๆแล้วเราสามารถที่จะประกาศ property โดยไม่ต้องสร้างตัวแปรของคลาสขึ้นมารองรับก็ได้ หรืออาจจะประกาศ property เป็นชื่อนึงและนำมาผูกกับตัวแปรที่เป็นชื่ออื่นก็ได้เช่นกัน ดังตัวอย่างเช่น

```
@interface MyClass : NSObject
{
    int a;
}

@property (nonatomic, assign, readwrite) int myVar;
@property (nonatomic, assign, readwrite) int b;

@end
```

```
@implementation MyClass

@synthesize myVar;
@synthesize b = a;

@end
```

ในตัวอย่างด้านบนนี้ เราได้ประกาศ property ไว้ 2 ตัวด้วยกัน คือ myVar และ b ซึ่งเราไม่ได้ทำการประกาศตัวแปร myVar ขึ้นมาด้วย ส่วน property b นั้นเราได้ประกาศตัวแปร a ขึ้นมาเพื่อใช้เก็บค่าแทน

ซึ่งการประกาศ property โดยไม่ประกาศตัวแปรขึ้นมาไว้รองรับนั้น ใน Objective C runtime เวอร์ชันใหม่ๆ (ซึ่งถูกใช้อยู่ใน iOS เวอร์ชันปัจจุบัน) จะทำการประกาศตัวแปรขึ้นมารองรับให้โดยอัตโนมัติ หรือก็คือมันจะให้ผลเหมือนกับการประกาศ property พร้อมกับประกาศตัวแปรนั่นเอง

นอกจาก property จะช่วยประหยัดเวลาในการเขียน setter/getter method แล้ว property ยังทำให้เราสามารถเข้าถึง property ของคลาสโดยการใช้ . (dot syntax) อย่างที่เราใช้ในการเข้าถึงตัวแปรของคลาสในภาษาอื่นๆ

```
<ObjectName>.<PropertyName>
```

ซึ่งช่วยให้เราสะดวกขึ้นเป็นอย่างมากในการเข้าถึงตัวแปร และกำหนดค่าให้กับตัวแปร (เฉพาะ property ที่กำหนดให้เป็น readwrite)

```
MyClass *a = [[MyClass alloc] init];
int aVar = a.myVar;
a.myVar = 10;
```

ในตัวอย่างนี้ จะแสดงให้เห็นว่าเราสามารถเข้าถึง property โดยใช้ . ได้ ซึ่งเราสามารถใช้ได้ทั้งการอ่านค่า และกำหนดค่า

3. Method เราสามารถประกาศเมธอดได้ 2 ประเภทด้วยกัน คือ class method และ instance method โดยเราจะต้องประกาศชื่อเมธอดไว้ใน interface และเขียนการทำงานของเมธอดนั้นๆใน implementation ของคลาส โดยการประกาศเมธอดนั้นจะมีรูปแบบดังนี้

```
+ (<ReturnType>)<ClassMethodName>:(<Type>)<Param1>
...;

- (<ReturnType>)<InstanceMethodName>:(<Type>)<Param1
  > ...;
```

ซึ่งจะมีตัวอย่างการประกาศเมธอดดังนี้

```
@interface MyClass : NSObject

+ (float)calculateValueFrom:(int)input;

- (float)calculateValueFrom:(int)input;

@end
```

โดยข้อแตกต่างของ class method กับ instance method ก็คือ เราสามารถเรียกใช้งาน class method ได้โดยตรง โดยไม่จำเป็นต้องสร้าง object ของคลาสขึ้นมาก่อน ในขณะที่ instance method นั้นเราจำเป็นต้องสร้าง object ขึ้นมา และเรียกใช้งานผ่านทาง object นี้

```
float output1 = [MyClass calculateValueFrom:10];

MyClass *a = [[MyClass alloc] init];
float output2 = [a calculateValueFrom:10];
```

ในตัวอย่างนี้ เราทำการคำนวณค่า output1 จาก class method ในขณะที่ output2 เราทำการคำนวณโดยใช้ instance method

ในกรณีที่เราต้องการเรียกใช้งานเมธอดภายในคลาสเดียวกันนั้น เราจะใช้คำสั่ง self ซึ่งจะเป็นอ้างอิงไปที่ object ที่กระทำงานอยู่ (หรือก็คือ object ที่เรียกคำสั่งนี้) โดยจะมีรูปแบบการใช้งานดังนี้

```
[self <MethodName>];
```

โดยจะมีตัวอย่างการใช้งานดังนี้

```
[self calculateValueFrom:10];
```

ซึ่งนอกจากจะใช้ self ในการเรียกเมธอดในคลาสเดียวกันแล้ว เรายังสามารถใช้ self ในการเข้าถึง property ของคลาสเดียวกันได้อีกด้วย ตัวอย่างเช่น

```
self.myVar = 10;
```

ซึ่งการใช้ self ในลักษณะนี้ จะต้องระวังไว้เรื่องนึง นั่นคือ การนำไปใช้งานใน getter/setter method นั่นเอง เนื่องจากว่า การใช้ self.<propertyName> นั่นจะเป็นการเรียกไปยัง getter/setter method นั่นเอง ถ้าหากเราเรียก self.<propertyName> ภายใน getter/setter method ก็จะทำให้เกิดลูปของการเรียกเมธอดขึ้นไม่สิ้นสุด ทำให้โปรแกรมเกิด error ขึ้นได้ ดังตัวอย่างต่อไปนี้

```
- (int)myVar
{
    return self.myVar;
}
```

SUBCLASS

1. ประกาศ superclass ใน interface
2. โดยทั่วไป ถ้าไม่ต้องการ subclass จากคลาสใดๆ เลย เราจะทำการ subclass จากคลาส NSObject เพื่อลดความซับซ้อนในการจัดการกับ object
3. สามารถ overwrite เมธอดของ superclass ได้
4. สามารถเข้าถึงเมธอดของ superclass โดยใช้คำสั่ง super
5. สามารถเข้าถึงเมธอดของคลาสเดียวกันได้โดยใช้คำสั่ง self

อย่างที่เราได้กล่าวไปแล้วในเรื่องของ class structure ในการประกาศ interface ของคลาสนั้น เราจะสามารถระบุคลาสที่เป็น superclass ลงไปด้วยได้ ซึ่งการ subclass นี้เป็นคุณสมบัติสำคัญประการหนึ่งของการเขียนโปรแกรมเชิงวัตถุ โดยจะทำให้คลาสลูกได้รับคุณสมบัติต่างๆของคลาสแม่มาด้วย นั่นคือ เราสามารถเรียกใช้งาน ตัวแปร, property ต่างๆ ที่มีอยู่ในคลาสแม่ได้ทั้งหมด สำหรับเมธอดนั้น คลาสลูกจะสามารถเรียกใช้ได้เฉพาะ instance method เท่านั้น เนื่องจาก class method จะเชื่อมโยงโดยตรงกับคลาสนั้นๆ ทำให้ไม่สามารถเรียกใช้จากคลาสลูกได้

ในคลาสลูกนั้น เราสามารถที่จะทำการ overwrite เมธอดของคลาสแม่ได้ โดยเวลาที่เรียกใช้งานเมธอดของคลาสที่ object ถูกสร้างขึ้น นั้นคือ ถ้าเป็น object ของคลาสลูก ก็จะเรียกไปยังเมธอดในคลาสลูก แต่ถ้าเป็น object ของคลาสแม่ ก็จะเรียกไปยังเมธอดในคลาสแม่ แต่ในบางครั้งเราต้องการเพียงแค่เพิ่มการทำงานบางอย่างเข้าไปในเมธอดสำหรับคลาสลูกเท่านั้น (คือให้ทำงานตามเมธอดของคลาสแม่ และเพิ่มการทำงานบางอย่างเข้ามา) ซึ่งหากต้องเขียนโปรแกรมทำงานในส่วนนี้ใหม่ทั้งหมด ก็อาจจะไม่สะดวกนั้น และอาจจะมีปัญหาในการจัดการกับโค๊ดในภายหลังได้ ซึ่งตรงนี้ ก็จะมีคำสั่งให้เราสามารถเข้าถึงเมธอดต่างๆของคลาสแม่ได้จากภายในคลาสลูก โดยใช้คำสั่ง super ซึ่งจะมีรูปแบบดังนี้

```
[super <MethodName>];
```

โดยจะมีตัวอย่างการใช้งานดังนี้

```
[super calculateValueFrom:10];
```

โดยตามหลักของการเขียนโปรแกรมเชิงวัตถุนั้น เราอาจจะเห็นว่า คลาสที่เราสร้างขึ้นอาจจะไม่จำเป็นที่จะต้อง subclass จากคลาสใดๆเลยก็ได้ ซึ่งการทำเช่นนั้นก็สามารถทำได้เช่นกัน เพียงแต่ว่าการทำเช่นนี้ เราจะต้องทำการจัดการการทำงานทั้งหมดของ object ของคลาสนี้ด้วยตัวเอง อาทิเช่น การจอง/คืนหน่วยความจำ, การจัดการ retain count, การทำงานกับเมธอดอื่นๆตามขั้นตอนการทำงานของภาษา Objective C เป็นต้น ซึ่งการจัดการเรื่องต่างๆเหล่านี้จะมีความซับซ้อนเป็นอย่างมาก และอาจจะทำให้เกิดปัญหาต่างๆขึ้นมาได้ หากจัดการไม่เหมาะสม

ในภาษา Objective C ก็จะได้ทำการเตรียมคลาส NSObject ไว้ให้ใช้งาน โดยคลาส NSObject จะรับผิดชอบการทำงานพื้นฐานต่างๆที่จำเป็นสำหรับการทำงานของ object ในระบบ ซึ่งหากเราไม่ต้องการจะทำการ subclass จากคลาสใดๆเลย เราจะทำการ subclass จากคลาส NSObject ซึ่งจะถือว่าเป็น superclass ของทุกๆคลาสในภาษา Objective C

โดยปกติในการสร้างคลาสใหม่ขึ้นมานั้น จะมีเมธอดสำคัญๆ ที่เราจะต้องทำการ implement นั้นคือ เมธอดที่เกี่ยวข้องกับ life cycle ของ object นั้นเอง ซึ่งเมธอดหลักๆก็ได้แก่ init, dealloc นั้นเอง โดยเมธอดเหล่านี้จะแตกต่างจากเมธอดอื่นๆ ตรงที่เราจะต้องเรียกเมธอดเดียวกันของ superclass ให้ทำงานด้วย เพื่อให้การสร้างและทำลาย object ในระบบทำงานได้อย่างถูกต้อง

```
- (id)init
{
    self = [super init];
    if (self) {
        // ...
    }

    return self;
}
```

ในตัวอย่างด้านบนนี้ เราจะทำการเรียก init ใน superclass ก่อน และนำ object ที่ได้มาทำงานในเมธอด init ของคลาสต่อ และคืน object นี้ออกมา

```
- (void)dealloc
{
    // ...

    [super dealloc];
}
```

ในตัวอย่างด้านบนนี้ เราจะทำการคืนค่าหน่วยความจำต่างๆ ที่เราใช้งานในคลาสให้หมด จากนั้นเราก็จะทำการเรียก dealloc ใน superclass เพื่อให้คืนค่าหน่วยความจำต่างๆ ที่สืบทอดมาจากคลาสแม่

สำหรับการตรวจสอบว่า object ที่เราทำงานด้วยอยู่นั้นเป็น object ของคลาสที่เราต้องการหรือไม่ เราจะสามารถใช้เมธอดต่อไปนี้ในการทดสอบได้ ได้แก่

1. - (BOOL)isKindOfClass:(Class)aClass ใช้ในการตรวจสอบว่า object นี้เป็น object ของคลาส aClass หรือคลาสลูกของคลาส aClass หรือไม่
2. - (BOOL)isMemberOfClass:(Class)aClass ใช้ในการตรวจสอบว่า object นี้เป็น object ของคลาส aClass หรือไม่
3. + (BOOL)isSubclassOfClass:(Class)aClass ใช้ในการตรวจสอบว่า คลาส นี้เป็น subclass ของคลาส aClass หรือไม่
4. + (Class)class
5. + (Class)superclass

```
if ([anObject isKindOfClass:[MyClass class]]) {
    // ....
}
```

ในตัวอย่างด้านบนนี้ เราทำการตรวจสอบว่าตัวแปร anObject เป็น object ของคลาส MyClass หรือคลาสลูกใดๆของคลาส MyClass หรือไม่

CATEGORY

1. เป็นการเพิ่มเมธอดใหม่ หรือ **overwrite** เมธอดใดๆ ในคลาสที่มีอยู่
2. เราไม่สามารถประกาศตัวแปรเพิ่มใน **category** ได้
3. **object** เดิมของคลาส จะสามารถใช้งานเมธอดใหม่ที่เพิ่มเติม หรือ **overwrite** ผ่านทาง **category** ได้เลย โดยไม่ต้องทำอะไรเพิ่มเติม

นอกเหนือจากการ subclass ใหม่ขึ้นมาเพื่อเพิ่มเมธอดให้กับคลาสที่เราต้องการแล้ว ในภาษา Objective C นั้น เรายังสามารถทำการสร้าง category ของคลาสได้อีกด้วย ซึ่ง category จะทำให้เราสามารถเพิ่มเมธอดใหม่ลงไปในคลาสที่มีอยู่แล้วได้โดยไม่ต้องทำการ subclass เป็นคลาสใหม่ขึ้นมา ซึ่งทำให้เราไม่จำเป็นต้องแก้ไขโปรแกรมเพื่อให้ทำงานกับคลาสใหม่

การประกาศ category นั้น จะคล้ายกับการประกาศคลาสใหม่ นั้นคือจะประกอบไปด้วย interface และ implementation เพียงแต่ เราจะต้องตั้งชื่อคลาสเป็นชื่อเดียวกับคลาสที่เราจะนำมาทำ category โดยเพิ่มชื่อ category ลงไปในวงเล็บ และไม่ต้องกำหนด superclass ซึ่งจะมีรูปแบบดังนี้

```
#import “ClassName”

@interface ClassName (CategoryName)

// method declaration

@end
```

และโดยปกติเราจะตั้งชื่อไฟล์เป็น “ClassName+CategoryName.h” และ “ClassName +CategoryName.m”

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

```
@interface NSString (MyString)

// method declaration

@end
```

ในตัวอย่างด้านบนนี้ เราประกาศ category สำหรับคลาส NSString โดยใช้ชื่อ category ว่า MyString

category จะไม่สามารถเพิ่มตัวแปร หรือ property ใหม่ลงไปได้ จะสามารถทำได้เพียงเพิ่มเมธอดใหม่ลงไปเท่านั้น โดยเราจะต้องประกาศเมธอดใน interface และเขียนโค๊ดของเมธอดนั้นใน implementation เช่นเดียวกับคลาส

```
@interface NSString (MyString)

- (NSString *)computeMyString;

@end
```

```
#import "NSString+MyString.h"

@implementation NSString (MyString)

- (NSString *)computeMyString
{
    return [NSString stringWithFormat:@"My String is
%@", self];
}

@end
```

ในตัวอย่างด้านบนนี้ เราได้ทำการเพิ่มเมธอด computeMyString ลงไปในคลาส NSString โดยการใช้ category

ภายใน category นั้น เราจะสามารถเข้าถึงตัวแปรทั้งหมดของคลาสที่เราขยายมาได้ ซึ่งรวมไปถึงตัวแปรที่เราประกาศเป็น @private ด้วย

ยิ่งไปกว่านั้น เราสามารถทำการ overwrite เมธอดโดยใช้ category ได้อีกด้วย

```
#import "NSString+MyString.h"

@implementation NSString (MyString)

- (NSString *)description
{
    return [NSString stringWithFormat:@"Overwrite de-
scription of %@", self];
}

@end
```

ในตัวอย่างด้านบนนี้เราได้ทำการ overwrite เมธอด description ของคลาส NSString โดยใช้ category

การนำ category ไปใช้งานนั้น เราเพียงแค่ import category header ของเราลงในโปรแกรมก็สามารถใช้งานได้เลย

```
#import "NSString+MyString.h"

@interface MyClass : NSObject
```

```
- (NSString *)generateCustomStringFrom:(NSString
*)inputString;

@end
```

```
@implementation MyClass

- (NSString *)generateCustomStringFrom:(NSString
*)inputString
{
    return [NSString stringWithFormat:@"%@ - %@",
[inputString computeMyString], [inputString descrip-
tion]];
}

@end
```

ในตัวอย่างด้านบนนี้ เราทำการนำ category ของคลาส NSString ที่เราสร้างขึ้นมาก่อนหน้านี้มาใช้งาน ซึ่งจะเห็นว่าเราสามารถเรียกใช้งานเมธอดที่เราเพิ่มเข้ามาได้จาก object ของคลาส NSString ได้เลย ซึ่งจะสะดวกกว่าการที่เราต้องประกาศคลาสใหม่ และสร้าง object ของคลาสใหม่ และอาจจะต้องทำการแปลงข้อมูลจาก object ของคลาสเดิมมายัง object ตัวใหม่อีกด้วย

สรุปสิ่งที่กล่าวถึงในบทนี้

1. โครงสร้างของคลาส
2. ส่วนประกอบต่างๆของคลาส
3. การ subclass
4. คุณสมบัติ Category

ในบทนี้เราได้พูดถึงเรื่องพื้นฐานต่างๆเกี่ยวกับคลาสในภาษา Objective C ซึ่งพื้นฐานของคลาสก็จะเหมือนกับคลาสในภาษาเชิงวัตถุอื่นๆ แต่ต่างกันที่ตัว syntax ของภาษา โดยในภาษา Objective C นั้น จะแยกส่วนของการประกาศตัวแปรต่างๆของคลาส กับการประกาศเมธอดของคลาสออกจากกันอย่างชัดเจน

นอกจากนี้ในภาษา Objective C ยังมี property ซึ่ง property จะช่วยให้ผู้เขียนโปรแกรมสะดวกในการเข้าถึงและใช้งานตัวแปรต่างๆของคลาส โดย property จะช่วยในการสร้าง getter/setter method ให้โดยอัตโนมัติ และยังช่วยให้สามารถเข้าถึงตัวแปรโดยใช้ . (dot syntax) ได้อีกด้วย

คุณสมบัติที่สำคัญอีกอย่างหนึ่งก็คือ category ซึ่งคุณสมบัตินี้จะช่วยให้ผู้เขียนโปรแกรมสามารถเพิ่มเมธอด หรือ overwrite เมธอดของคลาสที่มีอยู่แล้วได้ ซึ่งทำให้เราสามารถทำงานร่วมกับคลาสที่มีอยู่แล้วได้โดยไม่ต้องทำการสร้างคลาสใหม่ขึ้นมา