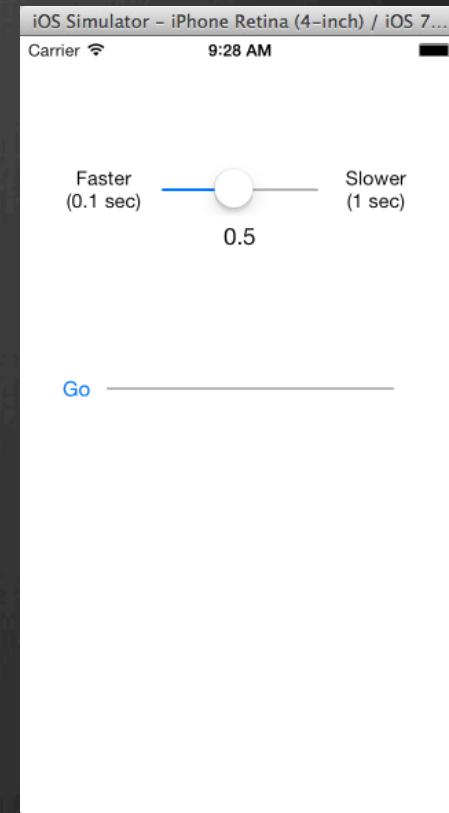


# Chapter 13

iOS Multi-Thread & Objective-C Block

# Multi-Threading

- ⦿ โดยปกติ App จะมี thread อญี่่แล้ว 1 thread คือ main thread
- ⦿ UI จะทำงานอยู่บน Main Thread
- ⦿ ถ้ามี process ที่ทำงานนานๆ process นั้นจะ block thread จนกว่าจะทำงานเสร็จ ผลกระทบคือ UI จะ freeze
- ⦿ เราสามารถเขียนโปรแกรมเพื่อสร้าง sub-thread ขึ้นมาทำงานที่ต้องใช้เวลานานๆ ได้ ด้วย static method ของ class “NSThread”
- ⦿ การ update ข้อมูลใน sub-thread จะไม่มีผลกระทบ main thread เช่น update UI จาก sub-thread แต่มี method ที่ช่วย join thread มากัง main thread ได้ เช่น performSelectorOnMainThread:



# Lab 1-3 : Single Thread (1/8)

## ◎ วัตถุประสงค์

- ◎ เพื่อให้เห็นปัญหาของ iOS App เมื่อทำงานแบบ Single Thread

## ◎ ขั้นตอน

- ◎ สร้าง project

- ◎ วางแผน control ต่างๆ เพื่อกำหนด speed ของ loop และให้เห็นว่า process กำลังทำงาน

- ◎ ผูก delegate และ action

- ◎ เขียน loop เพื่อจำลอง process ที่ใช้เวลานานๆ

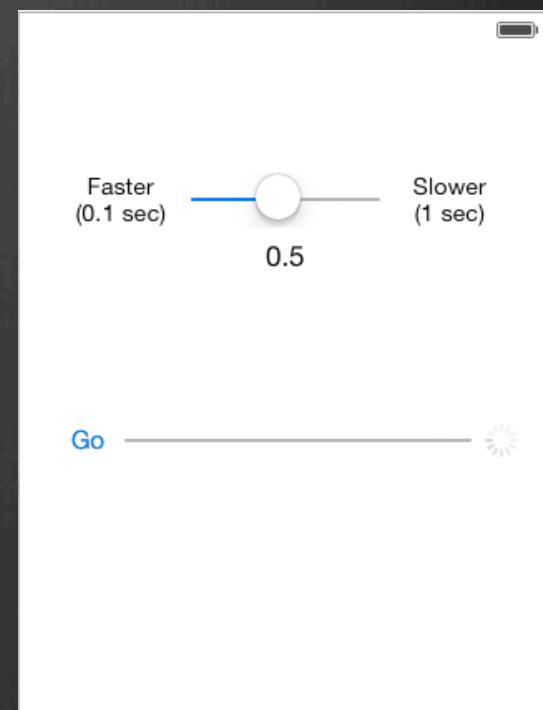
# Task : Create Project (2/8)

1. จาก Xcode สร้าง project ใหม่โดยเลือก iOS > Application > Single View Application
2. ตั้งชื่อ project ว่า “MultiThreading” และเลือก Devices เป็น iPhone
3. Click “Next” เลือก folder ที่จะ save project แล้ว click ปุ่ม “Create”

# Task : UI Design (3/8)

4. เปิด Main.storyboard เพิ่ม control ต่างๆ ดังนี้

- Label :- Text = “Faster (0.1 sec)”
- Label :- Text = “Slower (1 sec)”
- Slider :- Minimum = 0.1  
Maximum = 1  
Current = 0.5
- Label :- Text = “0.5”
- Button :- Title = “Go”
- Progress View :-  
Progress = 0
- Activity Indicator View  
Hides When Stopped = true



## Task : Binding Outlet & Action (4/8)

5. เปลี่ยน editor mode เป็น Assistant editor และผูก Outlet และ Action โดยที่

- Slider :-  
Type = Action  
Event = “Value Changed”  
Method name = “sliderValueChanged”
- Slider :-  
Type = Outlet  
Property name = “slider”
- Label :- (Text = 0.5)  
Type = Outlet  
Property name = “txtValue”
- Button :-  
Type = Action  
Method name = “btnGoTapped”
- Progress View :-  
Type = Outlet  
Property name = “progressView”
- Activity View :-  
Type = Outlet  
Property name = “activityView”

# Task : Coding – Create Worker Class (5/8)

6. สร้าง class ใหม่สำหรับจำลองการทำงานใน background thread โดย click ขวาที่ project เลือก New File... > iOS > Cocoa Touch > Objective-C Class เลือก Subtype of เป็น NSObject ตั้งชื่อว่า “Worker”
7. เปิดไฟล์ “Worker.h” เพิ่ม method สำหรับจำลองการ run process

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

@interface Worker : NSObject

@property (atomic) double sleepInterval;
- (void)doWorkForSleepInterval:(double)sleepTime;

@end
```

# Task : Coding – Create Worker Class (6/8)

8. เปิดไฟล์ “Worker.m” เพิ่ม code ดังนี้

```
#import "Worker.h"

@implementation Worker

- (void)doWorkForSleepInterval:(double)sleepTime
{
    self.sleepInterval = sleepTime;

    for (int i = 0; i <= 100; i++) {
        [NSThread sleepForTimeInterval:self.sleepInterval];
        NSLog(@"Current value = %i", i);
    }
}

@end
```

# Task : Coding – Call Worker (7/8)

9. เปิดไฟล์ “ViewController.h” เพิ่ม code ดังนี้

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

@interface ViewController : UIViewController

@property (nonatomic, strong) Worker * worker;

@property (weak, nonatomic) IBOutlet UISlider *slider;
@property (weak, nonatomic) IBOutlet UILabel *txtValue;
@property (strong, nonatomic) IBOutlet UIProgressView *progressView;
@property (weak, nonatomic) IBOutlet UIActivityIndicatorView *activityView;

- (IBAction)sliderValueChanged:(id)sender;
- (IBAction)btnGoTapped:(id)sender;

@end
```

10. เปิดไฟล์ “ViewController.m” เพิ่ม code ใน method “viewDidLoad:” ดังนี้

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.worker = [[Worker alloc] init];
}
```

# Task : Coding – Call Worker (8/8)

11. เพิ่ม code ใน method “sliderValueChanged:” ดังนี้

```
- (IBAction)sliderValueChanged:(id)sender
{
    self.txtValue.text = [NSString stringWithFormat:@"%0.1f",
                           self.slider.value];
    self.worker.sleepInterval = self.slider.value;
}
```

12. เพิ่ม code ใน method “btnGoTapped:” ดังนี้

```
- (IBAction)btnGoTapped:(id)sender
{
    [self.activityView startAnimating];
    [self.worker doWorkForSleepInterval:self.slider.value];
}
```

13. Run โปรแกรมเพื่อดูผลลัพธ์

- เมื่อ click “Go” โปรแกรมจะทำงานแต่ App จะค้าง  
ไม่สามารถเลื่อน slide bar ได้

# Lab 2-3 : Multi-Thread (1/3)

- ◎ วัตถุประสงค์
  - ◎ เพื่อให้สามารถพัฒนา iOS App ที่ทำงานแบบ Multi-Thread ได้
- ◎ ขั้นตอน
  - ◎ แก้ code ใน class Work เพื่อแตก thread ใหม่
  - ◎ ทดสอบว่า เมื่อ sub-thread ทำงานแล้ว main thread (UI) ถูก block หรือไม่

# Task : Coding – Do Background Thread (2/3)

1. เปิดไฟล์ “Worker.m” เพิ่ม code ดังนี้

```
#import "Worker.h"

@interface Worker (Private)
- (void)doBackgroundThread;
@end

@implementation Worker

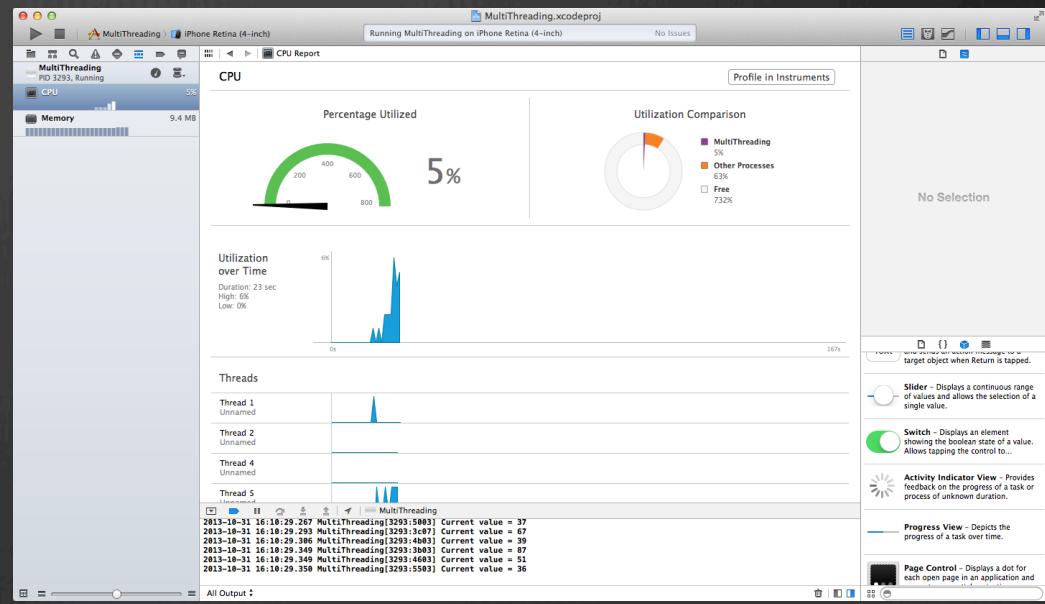
- (void)doWorkForSleepInterval:(double)sleepTime
{
    self.sleepInterval = sleepTime;
    [NSThread detachNewThreadSelector:@selector(doBackgroundThread)
                             toTarget:self
                               withObject:nil];
}

- (void)doBackgroundThread
{
    for (int i = 0; i <= 100; i++) {
        [NSThread sleepForTimeInterval:self.sleepInterval];
        NSLog(@"Current value = %i", i);
    }
}
@end
```

# Task : Coding – Run & Monitoring (3/3)

## 2. Run โปรแกรมเพื่อดูผลลัพธ์

- เมื่อ run โปรแกรม เราสามารถเลื่อน slide bar เพื่อเพิ่มลดความเร็วได้ โดยสังเกต จากความเร็วของ console ที่ write ออกมา
- ในขณะที่ run โปรแกรม เราสามารถ click ปุ่ม Go ช้าๆ ได้ ซึ่ง App จะสร้าง Thread ใหม่ สังเกตุได้จากตัวเลขใน console จะเริ่มนับอีกชุดแทรกกันไป
- เราสามารถ monitor ดู performance ได้ (CPU, Memory, Thread) โดยเปิด Debug Navigator บน Navigator Pane
- เราสามารถดูการใช้ CPU ของแต่ละ Thread ได้โดย click ที่ CPU เมื่อ Thread ทำงานจน Thread ก็จะถูก ทำลายไปเอง



# Lab 3-3 : Multi-Thread Callback with Objective-C Block (1/6)

- ◎ วัตถุประสงค์
  - ◎ เพื่อให้สามารถพัฒนา iOS App ด้วย Objective-C Block สำหรับการทำงานกับ App แบบ Multi-Thread และ Asynchronous
- ◎ ขั้นตอน
  - ◎ เพิ่ม type definition เพื่อกำหนด signature ของ Block
  - ◎ นำ Block ที่ประกาศไว้มาใช้ใน method ที่จะ callback กลับไป
  - ◎ เขียน code เพื่อ callback ผ่านทาง Block instant
  - ◎ เขียน code เพื่อเรียกใช้ Block method ที่ประกาศไว้เพื่อ update UI ตาม process ใน sub-thread

# Task : Declare Block TypeDef (2/6)

1. เปิดไฟล์ “Worker.h” เพิ่ม code การประกาศ type definition สำหรับ Objective-C Block code

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

typedef void (^onProgressBlock) (int currentProgress); // 1
typedef void (^onFinishBlock) (); // 2

@interface Worker : NSObject
{
    __strong onProgressBlock progressBlockObj;
    __strong onFinishBlock finishBlockObj;
}

@property (atomic) double sleepInterval;
- (void)doWorkForSleepInterval:(double)sleepTime
    onProgress:(onProgressBlock)progress // 1
    onFinish:(onFinishBlock)finish; // 2

@end
```

## Task : Declare Block Method (3/6)

2. เปิดไฟล์ “Worker.m” แก้ code ใน method “doWorkForSleepInterval:” เพื่อเก็บ pointer ของ block ไว้สำหรับ callback

```
- (void)doWorkForSleepInterval:(double)sleepTime  
    onProgress:(onProgressBlock)progress  
    onFinish:(onFinishBlock)finish  
{  
    progressBlockObj = progress;  
    finishBlockObj = finish;  
  
    self.sleepInterval = sleepTime;  
    [NSThread detachNewThreadSelector:@selector(doBackgroundThread)  
        toTarget:self  
        withObject:nil];  
}
```

# Task : Block Callback (4/6)

3. ที่ไฟล์ “Worker.m” เพิ่ม method “updateProgress:” และ “finish”  
เพื่อใช้เรียกกลับไปยัง main thread

```
- (void)updateProgress:(NSNumber *)currentProgress
{
    if (progressBlockObj) {
        progressBlockObj([currentProgress intValue]);
    }
}

- (void)finish
{
    if (finishBlockObj) {
        finishBlockObj();
    }
}
```



# Task : Update UI (6/6)

5. แก้ code ใน method “doBackgroundThread:” เพื่อเรียก method ที่ใช้ block กลับไปยัง main thread

```
- (IBAction)btnGoTapped:(id)sender
{
    [self.activityView startAnimating];
    [self.worker doWorkForSleepInterval:self.slider.value

        onProgress:^(int currentProgress) {
            float x = currentProgress;
            self.progressView.progress = x / 100.0f;
        }

        onFinish:^{
            self.progressView.progress = 0;
            [self.activityView stopAnimating];
        }
    ];
}
```

6. Run โปรแกรมเพื่อดูผลลัพธ์

# Objective-C Block

- ◉ Blocks เป็นส่วนขยายของภาษา C ที่ถูกเพิ่มเข้ามาใน Objective-C ตอนที่ iOS SDK4 ถูกพัฒนาขึ้นมา
- ◉ Feature ของ Block นั้นมีใช้แพร่หลาย ในหลายภาษา เช่น Ruby, Python, หรือ Lisp เป็นต้น
- ◉ แนวคิดของ Block ก็คือ เขียน function แบบ anonymous และส่ง function นั้นเป็น parameter ให้กับ method ที่ต้องการ เพื่อทำให้ method นั้นสามารถเพิ่ม (extend) functionality ได้แบบ dynamic
- ◉ Block ไม่ได้ถูกนำมาใช้แทน delegate แต่หมายความว่าสามารถใช้กับ multi-thread และ asynchronous callback

# Delegate vs. Block

## Delegate

```

1 // ViewController.m
2 // MultiThreading
3 //
4 // Created by Olarn U. on 10/31/2556 BE.
5 // Copyright (c) 2556 Olarn U. All rights reserved.
6 //
7 //
8 #import "ViewController.h"
9 @interface ViewController : NSObject
10 @end
11
12 @implementation ViewController
13
14 - (void)viewDidLoad
15 {
16     [super viewDidLoad];
17     self.worker = [[Worker alloc] init];
18 }
19
20 - (void)doSomething
21 {
22     int (^multiply)(int, int) = ^{int x, int y) {
23         return x * y;
24     };
25
26     int result = multiply(5,10);
27     NSLog(@"%@", result);
28 }
29
30 - (void)didReceiveMemoryWarning
31 {
32     [super didReceiveMemoryWarning];
33     // Dispose of any resources that can be recreated.
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88

```



Delegate  
callback

Delegate  
Handler

Single Object

```

1 // ViewController.m
2 // MultiThreading
3 //
4 // Created by Olarn U. on 10/31/2556 BE.
5 // Copyright (c) 2556 Olarn U. All rights reserved.
6 //
7 //
8 #import "ViewController.h"
9 @interface ViewController : NSObject
10 @end
11
12 @implementation ViewController
13
14 - (void)viewDidLoad
15 {
16     [super viewDidLoad];
17     self.worker = [[Worker alloc] init];
18 }
19
20 - (void)doSomething
21 {
22     int (^multiply)(int, int) = ^{int x, int y) {
23         return x * y;
24     };
25
26     int result = multiply(5,10);
27     NSLog(@"%@", result);
28 }
29
30 - (void)didReceiveMemoryWarning
31 {
32     [super didReceiveMemoryWarning];
33     // Dispose of any resources that can be recreated.
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88

```



Object object object object

Delegate  
Handler

Multiple Objects

# Delegate vs. Block

## Block

```

1 // ViewController.m
2 // MultiThreading
3 //
4 // Created by Olarn U. on 10/31/2556 BE.
5 // Copyright (c) 2556 Olarn U. All rights reserved.
6 //
7 //
8
9 #import "ViewController.h"
10 @interface ViewController : NSObject
11 @end
12
13 @implementation ViewController
14
15 - (void)viewDidLoad
16 {
17     [super viewDidLoad];
18     self.worker = [[Worker alloc] init];
19 }
20
21 - (void)doSomething
22 {
23     int (^multiply)(int, int) = ^{int x, int y) {
24         return x * y;
25     };
26
27     int result = multiply(5,10);
28     NSLog(@"%@", result);
29 }
30
31 - (void)didReceiveMemoryWarning
32 {
33     [super didReceiveMemoryWarning];
34     // Dispose of any resources that can be recreated.
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88

```



Object  
Delegate  
callback

Delegate  
Handler

Single Object

```

1 // ViewController.m
2 // MultiThreading
3 //
4 // Created by Olarn U. on 10/31/2556 BE.
5 // Copyright (c) 2556 Olarn U. All rights reserved.
6 //
7 //
8
9 #import "ViewController.h"
10 @interface ViewController : NSObject
11 @end
12
13 @implementation ViewController
14
15 - (void)viewDidLoad
16 {
17     [super viewDidLoad];
18     self.worker = [[Worker alloc] init];
19 }
20
21 - (void)doSomething
22 {
23     int (^multiply)(int, int) = ^{int x, int y) {
24         return x * y;
25     };
26
27     int result = multiply(5,10);
28     NSLog(@"%@", result);
29 }
30
31 - (void)didReceiveMemoryWarning
32 {
33     [super didReceiveMemoryWarning];
34     // Dispose of any resources that can be recreated.
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88

```



Object

Object

Multiple Objects

# Objective-C Block

- ◎ การประกาศ type ของ Block เป็นการประกาศ header ของ method ซึ่ง body ของ method จะถูก implement ภายหลัง
- ◎ Block เป็น type ที่ประกาศขึ้นมาเพื่อเป็น instant ของ function เพราะฉะนั้นเราสามารถประกาศ instant เป็น type Block
- ◎ Instant นั้นอาจจะเป็น named instant หรืออาจจะเป็น anonymous ก็ได้
- ◎ Pattern ของการประกาศ Block จะใกล้เคียงกับ C มากกว่าของ Objective-C

# Declaration with Typedef

## ❖ Block Declaration

Return type



```
typedef void (^OnProgressBlock) (int currentProgress);
```

Block Type Name



Parameters  
(ถ้าไม่มีให้空着แล้วเปลี่ยน)



## ❖ Block Instant Declaration

```
OnProgressBlock progressBlockObj;
```



Block Type Name



Instant Name

# Using Block Instant

## ❖ Block Declaration with Return Type

```
typedef int (^OnProgressBlock) (int currentProgress);  
OnProgressBlock progressBlockObj;
```

## ❖ Using Block with Return Value

```
if (progressBlockObj) {  
    int x = progressBlockObj(10);  
    . . .  
}
```

# Using Block in Method

## Method Declaration with Block

```
- (void)doWorkForSleepInterval:(double)sleepTime  
    onProgress:(OnProgressBlock)progress;
```

Block Type

## Call method with Block Handler

```
[self.worker doWorkForSleepInterval:0.1  
    onProgress:^(int currentProgress) {  
        progressView.progress = x / 100.0f;  
    }  
];
```

Anonymous Instant

Parameters passed  
from Block

Method body, implement code when callback

# Local Block

- Declare & Use Block with in the Same Method

```
- (void)doSomething  
{
```

Declare block instant

```
    ↓  
int (^multiply)(int, int) = ^(int x, int y) {  
    return x * y;  
};
```

Block code

```
// do some code ...
```

```
int result = multiply(5,10); ← Call Block  
NSLog(@"%@", result);  
}
```

# Block Type vs. Block Declaration

## ⌚ Method with Block Type

```
- (void)doWorkForSleepInterval:(double)sleepTime  
    onProgress:(OnProgressBlock)progress;
```

## ⌚ Method with Block Declaration

```
- (void)doWorkForSleepInterval:(double)sleepTime  
    onProgress:( int (^)(int) )progress;  
  
- (void)doWorkForSleepInterval:(double)sleepTime  
    onProgress:( int (^)(int) )progress  
{  
    if (progress) {  
        progress(10);  
    }  
}
```

# Writing Methods That Return Blocks

- Instant ที่เราประกาศ type เป็น block ก็คือ instance ธรรมดา เราสามารถเขียน method เพื่อ return block instant ได้ซึ่งทำให้เราสามารถ reuse block ได้แทนที่จะเป็น anonymous method เช่น

```

typedef int (^sqRootBlock) (int num1, int num2);
- (sqRootBlock) mySqRoot:(int)num1 with:(int)num2 {
    sqRootBlock block = ^(int n1, int n2){
        return n1 * n2;
    };
    return block;
}

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        BlockCaller *blockCaller = [[BlockCaller alloc] init];
        int result = [blockCaller squareRoot:2
            withBlock:[blockCaller mySqRoot:2 with:2]];
        NSLog(@"%@", result);
    }
    return 0;
}

```

return type เป็น block  
จาก typedef

เรียกใช้ block จาก  
method

# Blocks Are Closures

- เมื่อเรา assign ค่าให้กับ code ใน block และ ค่าที่อยู่ใน block จะถูก snapshot ไว้ (หรือถูก freeze ไว้) นั่นหมายความว่า ถึงเราจะแก้ค่าตัวแปรที่อยู่นอก block ไป แต่ค่าตัวแปรที่อยู่ใน block ก็ไม่เปลี่ยนตาม เพราะถูก block ไว้ลักษณะที่ตัวแปรถูก snapshot ไว้เรียกว่า Closure

```
NSDate *date = [NSDate date];  
  
void (^now)(void) = ^ {  
    NSLog(@"The date and time is %@", date);  
};  
  
now();    <- เมื่อเรียก block ค่าของ date จะถูก snapshot  
  
sleep(5);  
  
date = [NSDate date];  
  
now();    <- เมื่อเรียก now อีกครั้ง ค่าของ date จะไม่เปลี่ยนไป
```

# Blocks Are Closures (cont.)

- ในกรณีที่เราต้องการแก้ local variable ที่ถูกประกาศนอก block ใน code ของ block ตัว compiler จะไม่ยอมให้เราแก้ไข เพราะ block จะทำการ snapshot ค่า นั้นไว้เป็น read-only เช่น

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

        int value = 0; // local var
        int (^multiplier)(int, int) = ^(int num1, int num2){
            value = num1 * num2;
            return value;
        };

        multiplier(3, 3);
        NSLog(@"Value = %i", value);
    }
    return 0;
}
```

Compile Error



# Blocks Are Closures (cont.)

- ให้กำหนด type modifier ตอนประกาศ local variable ว่าเป็น “`_block`”

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

        _block int value = 0;    ← ประกาศเป็น _block (under scroll 2 อัน)
        int (^multiplier)(int, int) = ^(int num1, int num2) {
            value = num1 * num2;
            return value;
        };

        multiplier(3, 3);
        NSLog(@"Value = %i", value);
    }
    return 0;
}
```

# References

- The Pragmatic Studio – Using Blocks in iOS 4
  - <http://pragmaticstudio.com/blog/2010/7/28/ios4-blocks-1>
  - <http://pragmaticstudio.com/blog/2010/9/15/ios4-blocks-2>