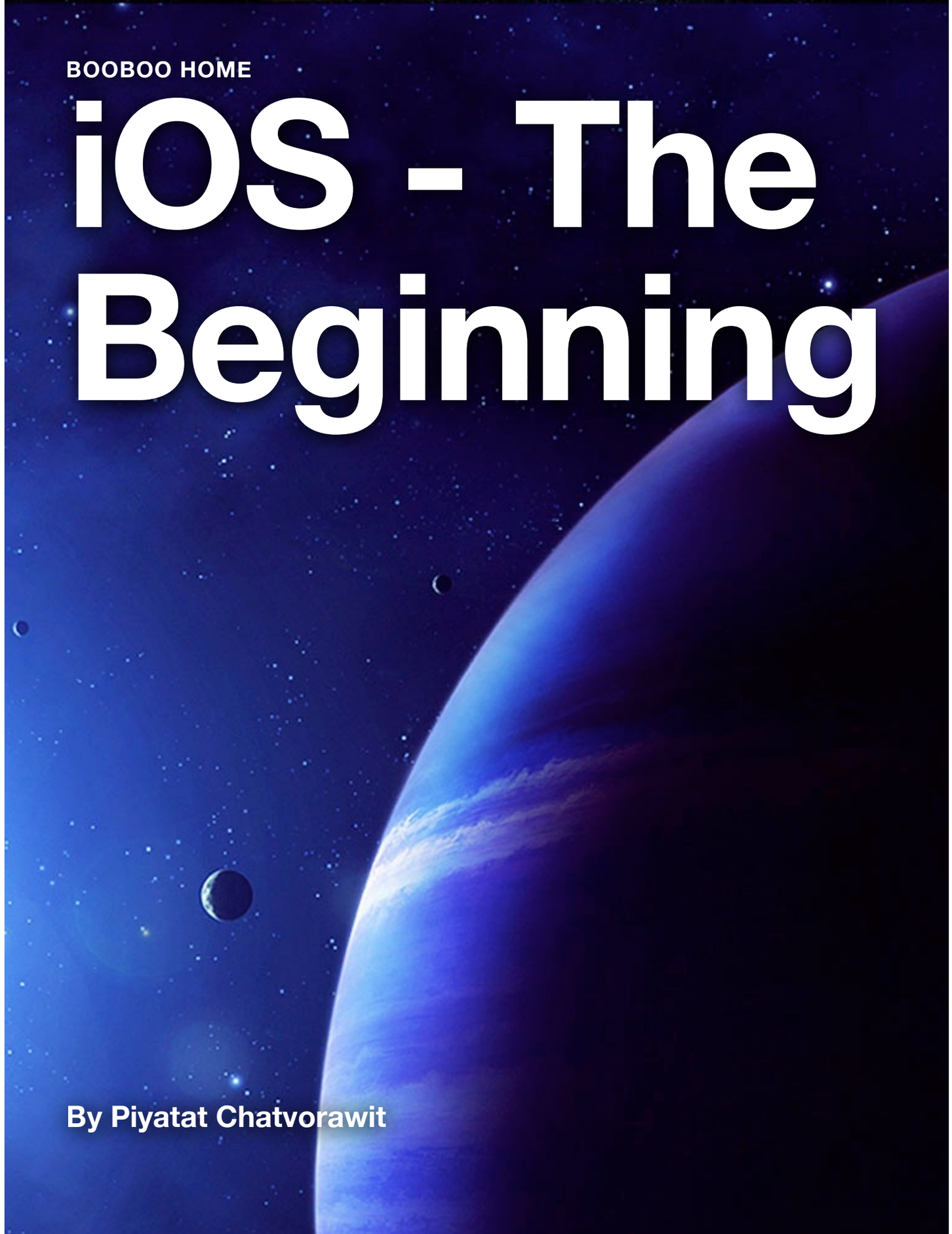

BOOBOO HOME
iOS - The Beginning
By Piyatat Chatvorawit

Chapter 7

Objective C - Protocol and Delegate

ในบทนี้เราจะกลับมาพูดถึงเรื่องเกี่ยวกับ Object ในภาษา Objective C ต่อ โดยเราจะพูดถึงเรื่องของ Protocol และ Delegate

PROTOCOL

1. กำหนดรูปแบบของเมธอดสำหรับใช้ในการติดต่อระหว่าง object
2. ประกาศไว้ในไฟล์ Interface ของคลาส
3. สามารถกำหนดให้เมธอดเป็น required หรือ optional ได้
4. คลาสที่ทำการ implement Protocol จะต้อง implement เมธอดที่เป็น required ทั้งหมด

ในการเขียนโปรแกรมนั้น มีหลายครั้งที่ในคลาสที่เราเขียนจำเป็นจะต้องติดต่อกับ object อื่นๆ ภายนอก ตัวอย่างเช่น object b ส่งข้อมูลเข้ามาให้ object a เพื่อทำการประมวลผล และเมื่อ object a ประมวลเสร็จแล้วก็จะส่งข้อมูลกลับไปให้ object b ซึ่งตรงนี้ทั้ง object a และ object b จำเป็นที่จะต้องรู้รูปแบบของเมธอดที่จะใช้ในการติดต่อกัน แต่มีบางครั้งที่เราไม่อาจจะรู้ได้ว่าเมธอดของ object ที่เรียกเข้ามานั้นมีเมธอดอะไรบ้าง หรือต้องเรียกเมธอดใดเพื่อใช้ในการติดต่อ ซึ่งกรณีเช่นนี้จะพบได้อยู่เสมอกับคลาสที่นำมาจากภายนอก เนื่องจากคลาสเหล่านี้จะถูกเขียนขึ้นมาก่อนสำหรับใช้ทำงานอย่างใดอย่างหนึ่ง ทำให้ไม่สามารถรู้ถึง object ที่จะมาเรียกใช้งานได้

Protocol เป็นข้อกำหนดของเมธอดสำหรับให้คลาสนำไปทำการ implement เพื่อให้ object จากคลาสอื่นๆ สามารถติดต่อกับ object ของคลาสนี้ผ่านเมธอดที่กำหนดไว้ใน Protocol ได้ ซึ่งทำให้เราไม่จำเป็นที่จะต้องรู้จักเมธอดต่างๆของคลาสนี้

การประกาศ Protocol นั้นจะประกาศไว้ในไฟล์ .h โดยจะมีรูปแบบดังนี้

```
@protocol ProtocolName <protocol list>

// Method Definition

@end
```

โดยเมธอดที่ประกาศไว้ใน Protocol จะมีอยู่ 2 แบบด้วยกัน

1. Required Method เป็นเมธอดที่คลาสที่ implement Protocol นี้จะต้องทำการ implement เมธอดนี้ด้วย

2. Optional Method คลาสที่ implement Protocol นี้จะทำการ implement เมธอดประเภทนี้หรือไม่ก็ได้

```
@protocol MyFirstProtocol <NSObject>
- (void)protocolDefaultMethod;
@required
- (void)protocolRequiredMethod;
@optional
- (void)protocolOptionalMethod;
@end
```

ในตัวอย่างนี้ เราได้ทำการประกาศ Protocol ขึ้นมา และประกาศเมธอดไว้ 3 เมธอดด้วยกัน โดยเมธอดที่ถูกประกาศอยู่นอก(ประกาศก่อน) คำสั่ง @required และ @optional จะถือว่าเป็น required method นั่นคือเมธอด protocolDefaultMethod จะถือว่าเป็น required method เช่นเดียวกับเมธอด protocolRequiredMethod

เราสามารถประกาศ Protocol ได้มากกว่า 1 Protocol ในไฟล์ .h

```
@protocol MyFirstProtocol <NSObject>
- (void)protocolDefaultMethod;
@required
- (void)protocolRequiredMethod;
@optional
- (void)protocolOptionalMethod;
@end
@protocol MySecondProtocol <NSObject>
- (void)mySecondProtocolMethod;
@end
```

นอกจากนี้ Protocol ยังสามารถขยายมาจาก Protocol อื่นได้อีกด้วย โดย Protocol ที่ขยายเพิ่มมานั้นจะมีเมธอดทั้งหมดที่กำหนดไว้ใน Protocol ที่นำมาขยาย สำหรับการประกาศ Protocol ที่ขยายมาจาก Protocol อื่นนั้นจะมีโครงสร้างในการประกาศดังนี้

```
@protocol MyExtendedProtocol <MyFirstProtocol>
- (void)extendedMethod;
@end
```

ในตัวอย่างนี้ คลาสที่จะทำการ implement MyExtendedProtocol จะต้องทำการ implement MyFirstProtocol ด้วย

สำหรับคลาสที่จะ implement Protocol นั้นจะมีรูปแบบดังนี้

```
@protocol MyFirstProtocol <NSObject>
- (void)protocolDefaultMethod;
@required
- (void)protocolRequiredMethod;
@optional
- (void)protocolOptionalMethod;
@end
@protocol MySecondProtocol <NSObject>
- (void)mySecondProtocolMethod;
@end
@interface MyClass : NSObject <MyFirstProtocol, My-
SecondProtocol>
@end
```

ในตัวอย่างนี้ คลาส MyClass จะต้องทำการ implement เมธอด protocolDefaultMethod, protocolRequiredMethod และ mySecondProtocolMethod ส่วนเมธอด protocolOptionalMethod นั้นจะทำการ implement หรือไม่ก็ได้ เนื่องจากเป็น optional method

สำหรับการทดสอบว่า object นี้ได้ทำการ implement Protocol ที่เราต้องการหรือไม่นั้น เราจะใช้เมธอดนี้ในการทดสอบ

```
- (BOOL)conformsToProtocol:(Protocol *)aProtocol;
```

โดยเรียกไปยัง object ที่เราต้องการทดสอบ และส่ง Protocol ที่เราต้องการเข้าไป ดังนี้

```
if ([myObject
conformsToProtocol:@protocol(MyFirstProtocol)]) {
    // Do Something ...
}
```

นอกจากนี้เรายังสามารถทำการตรวจสอบการ implement Protocol โดย compiler ได้อีกด้วย โดยใช้คำสั่งดังนี้

```
id<MyFirstProtocol> delegate;
```

โดยวิธีนี้ compiler จะทำการตรวจสอบว่า object ที่นำมากำหนดให้กับตัวแปร delegate นั้นทำการ implement MyFirstProtocol ไว้หรือไม่ ถ้าหากไม่ทำการ implement เอาไว้ ก็จะมี warning เตือนขึ้นมา

ในบางครั้งเราอาจจำเป็นที่จะต้องทำการประกาศ Protocol ที่อ้างอิงไปยัง Protocol อื่น ซึ่งอาจจะทำให้เกิดการอ้างอิงกลับไปกลับมาไม่รู้จบได้

```
//
//  MyFirstClass.h
//
#import <Foundation/Foundation.h>
#import "MySecondClass.h"

@protocol FirstProtocol <NSObject>
```

```
-
(void)firstMethodWithObject:(id<SecondProtocol>)obje
ct;

@end

@interface MyFirstClass : NSObject

@end
```

```
//
//  MySecondClass.h
//

#import <Foundation/Foundation.h>

#import "MyFirstClass.h"

@protocol SecondProtocol <NSObject>

-
(void)secondMethodWithObject:(id<FirstProtocol>)obje
ct;

@end

@interface MySecondClass : NSObject

@end
```

ในตัวอย่างนี้ เมธอดที่อยู่ใน FirstProtocol จะอ้างอิงถึง object ที่ทำการ implement SecondProtocol เอาไว้ ในขณะที่เมธอดที่ประกาศไว้ใน SecondProtocol จะทำการอ้างอิงกลับมายัง object ที่ implement FirstProtocol เอาไว้ ซึ่งทำให้เกิด loop ของการ import ไฟล์ทั้ง 2 นี้ไม่รู้จบ

ซึ่งในกรณีเช่นนี้เราจะทำการประกาศ forward reference โดยการประกาศชื่อของ Protocol ที่เราต้องการอ้างอิงถึงไว้ โดยไม่ทำการ import เข้ามาเพื่อป้องกัน loop ของการ import ไม่รู้จบ

```
//
//  MyFirstClass.h
//

#import <Foundation/Foundation.h>

@protocol SecondProtocol;

@protocol FirstProtocol <NSObject>

-
(void)firstMethodWithObject:(id<SecondProtocol>)obje
ct;

@end

@interface MyFirstClass : NSObject

@end
```

```
//
//  MySecondClass.h
//

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

```
@protocol FirstProtocol;

@protocol SecondProtocol <NSObject>

-
(void)secondMethodWithObject:(id<FirstProtocol>)obje
ct;

@end

@interface MySecondClass : NSObject

@end
```

ในตัวอย่างนี้ เราจะใช้วิธีการประกาศ forward reference ขึ้นมาแทน ทำให้ไม่เกิด loop ของการ import ขึ้น และเราจะทำการ import ไฟล์ที่ต้องการในไฟล์ .m แทน

DELEGATE & DATASOURCE

1. ใช้เรียก object ที่จะใช้ติดต่อเมื่อเกิดเหตุการณ์ที่กำหนดขึ้น
2. ทำการ implement Protocol สำหรับตอบสนองต่อเหตุการณ์ที่กำหนด

Delegate และ DataSource คือ object ที่จะถูกเรียกหรือส่ง message ให้เมื่อเกิดเหตุการณ์บางอย่างขึ้นในโปรแกรม ตัวอย่างเช่น UITableView จะส่ง message ไปยัง dataSource เพื่อขอข้อมูลจำนวน row ของ tableView ที่จะแสดง หรือ UITableView จะส่ง message ไปยัง delegate เมื่อผู้ใช้ทำการเลือก row ใดๆใน tableView เป็นต้น ซึ่งโดยทั่วไปแล้ว delegate และ dataSource ก็คือ object ของคลาสใดๆที่ทำการ implement Protocol ที่ระบุไว้นั้นเอง

Delegate กับ DataSource โดยทั่วไปแล้วจะเหมือนกัน แต่ที่เรียกชื่อต่างกันก็เพราะถูกนำมาใช้ในจุดประสงค์ที่ต่างกัน นั้นคือ delegate โดยส่วนมากจะใช้เพื่อตอบสนองต่อเหตุการณ์บางอย่างในโปรแกรม ซึ่งส่วนมากคือตอบสนองต่อ input ของผู้ใช้นั้นเอง ในขณะที่ dataSource จะถูกใช้สำหรับเรียกข้อมูลขึ้นมาแสดงในโปรแกรม หรือก็คือใช้เพื่อดึงข้อมูลขึ้นมาแสดงนั้นเอง

ใน iOS framework นั้นจะมีการนำ delegate และ dataSource มาใช้งานอยู่บ่อยครั้ง ซึ่งเราจะได้พบในบทต่อๆไป

โดยทั่วไปนั้น การนำ delegate หรือ dataSource มาใช้งานนั้น เราจะเริ่มโดยการประกาศ Protocol ที่เราต้องการไว้ในคลาสที่จะเรียกใช้งาน delegate หรือ dataSource ก่อน จากนั้นเราก็จะทำการประกาศ property delegate หรือ dataSource เอาไว้ เพื่อให้สามารถทำการกำหนด delegate object หรือ dataSource object ได้โดยง่าย

```
//
//  MyClass.h
//

#import <Foundation/Foundation.h>

@protocol MyDataSource;
@protocol MyDelegate;

@interface MyClass : NSObject
{
    id<MyDataSource> dataSource;
    id<MyDelegate> delegate;
}

@property (nonatomic, assign, readwrite) id<MyData-
Source> dataSource;
@property (nonatomic, assign, readwrite) id<MyDele-
gate> delegate;

@end

@protocol MyDataSource <NSObject>

- (int)numberOfItemForMyClass:(MyClass *)sender;
- (id)myClass:(MyClass *)sender
objectAtIndex:(int)index;

@end

@protocol MyDelegate <NSObject>

-     (void)myClass:(MyClass *)sender
selectObjectAtIndex:(int)index;

@end
```

ในตัวอย่างนี้ เราทำการประกาศ Protocol ขึ้นมา 2 อัน เพื่อใช้งานกับ DataSource และ Delegate โดยใช้วิธี forward reference เนื่องจากเราต้องทำการอ้างอิงถึงคลาสภายในเมธอดของ Protocol ในขณะที่ในคลาสก็ต้องอ้างอิงถึง Protocol เช่นกัน

โดยเมธอดที่ประกาศไว้ใน MyDataSource จะเป็นเมธอดที่ใช้ในการดึงข้อมูลเพื่อนำมาใช้ภายในคลาส ส่วนเมธอดที่อยู่ใน MyDelegate จะใช้สำหรับการตอบสนองต่อการเลือกข้อมูลของคลาส

เวลาที่เราจะทำการเรียก delegate หรือ dataSource นั้นเราจะต้องทำการทดสอบก่อนว่าเมธอดที่ต้องการจะเรียกใช้นั้น ได้ถูก implement ไว้หรือไม่ เนื่องจากเมธอดนั้นอาจจะเป็น optional เมธอดก็ได้

```
if ([myObject
respondsToSelector:@selector(myClass:selectObjectAtI
ndex:)]) {
    // Do Something ...
}
```

Project 7.1 - ImageLoader

PROJECT SUMMARY

1. สร้างคลาสสำหรับใช้ในการโหลดรูปขึ้นมา โดยจะโหลดรูปได้ใน 2 ลักษณะ คือ **synchronous** และ **asynchronous**
2. ประกาศ **Protocol** สำหรับให้เรียกทำงานเมื่อทำการโหลดรูปแบบ **asynchronous** เสร็จ
3. สร้างโปรแกรมที่ **implement Protocol** สำหรับใช้รับ **URL** จากผู้ใช้ และทำการโหลดรูปขึ้นมาแสดง

ในตัวอย่างนี้ เราจะทำการสร้างคลาสสำหรับใช้ในการโหลดรูป โดยเราจะทำการโหลดรูปใน 2 ลักษณะ คือ

1. โหลดแบบ Synchronous หรือก็คือ การโหลดโดยใช้ thread เดียวกับ thread ที่เรียกเข้ามา ซึ่งจะทำให้ thread นี้จะหยุดทำงานไปจนกว่ารูปจะถูกโหลดเสร็จ

2. โหลดแบบ Asynchronous หรือก็คือ การสร้าง thread ใหม่ขึ้นมา เพื่อโหลดรูป

เพื่อแสดงให้เห็นการใช้งาน Delegate และแสดงตัวอย่างของการทำงานแบบ multi-thread ซึ่งเราจะกล่าวถึงอีกทีในภายหลัง

ใน project นี้เราจะทำการสร้างไฟล์ขึ้นมา 4 ไฟล์ด้วยกัน คือ

```
ImageLoader.h
ImageLoader.m
MyViewController.h
MyViewController.m
```

โดยในแต่ละไฟล์จะมีโค๊ดดังนี้

```
//
//  ImageLoader.h
//
```

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

@protocol ImageLoaderDelegate;

@interface ImageLoader : NSObject
{
    NSMutableDictionary *imageDictionary;

    id<ImageLoaderDelegate> delegate;
}

@property (nonatomic, assign, readwrite) id<Image-
LoaderDelegate> delegate;

- (UIImage *)syncLoadImageAtURL:(NSURL *)imageURL;
- (void)asyncLoadImageAtURL:(NSURL *)imageURL;

@end


@protocol ImageLoaderDelegate <NSObject>

-     (void)imageLoader:(ImageLoader *)imageLoader
didFinishLoadImageAtURL:(NSURL *)imageURL
              withImage:(UIImage *)image;
-   (void)imageLoader:(ImageLoader *)imageLoader
didFailLoadImageAtURL:(NSURL *)imageURL;

@end
```

คลาส ImageLoader จะนำมาใช้ในการโหลดรูปภาพมาเก็บไว้ในหน่วยความจำ โดยจะมีรูปแบบการโหลดอยู่ 2 แบบตามที่ได้กล่าวไปแล้ว

ไฟล์ Interface ของคลาส ImageLoader จะทำการประกาศ Protocol แบบ forward reference ขึ้นมาสำหรับใช้ในการติดต่อเพื่อส่งข้อมูลกลับไป เมื่อทำการโหลดรูปเสร็จเรียบร้อยแล้ว หรือเกิด error ขึ้น

ในไฟล์นี้จะมีการประกาศตัวแปรอยู่ 2 ตัวด้วยกัน คือ

1. imageDictionary เป็นชนิด NSMutableArray ซึ่งจะใช้เก็บรูปที่โหลดมาแล้ว

2. delegate เป็นชนิด id<ImageLoaderDelegate> สำหรับใช้ในการติดต่อส่งข้อมูลกลับไปเมื่อ โหลดรูปเสร็จแล้ว หรือเกิด error ขึ้น โดยจะประกาศให้ตัวแปรนี้เป็น Property ด้วย

และจะทำการประกาศเมธอดไว้ด้วยกัน 2 เมธอด คือ

1. - (UIImage *)syncLoadImageAtURL:(NSURL *)imageURL สำหรับทำการโหลดรูปแบบ synchronous

2. - (void)asyncLoadImageAtURL:(NSURL *)imageURL สำหรับทำการโหลดรูปแบบ asynchronous

ส่วนใน Protocol ImageLoaderDelegate นั้น จะทำการประกาศเมธอดไว้ด้วยกัน 2 เมธอดคือ

1. - (void)imageLoader:(ImageLoader *)imageLoader didFinishLoadImageAtURL:(NSURL *)imageURL สำหรับส่งข้อมูลรูปกลับไปเมื่อทำการโหลดรูปเสร็จเรียบร้อยแล้ว

2. - (void)imageLoader:(ImageLoader *) didFailLoadImageAtURL:(NSURL *)imageURL สำหรับทำการเรียกเมื่อเกิด error ขึ้น

```
//
//  ImageLoader.m
//

#import "ImageLoader.h"

@implementation ImageLoader

@synthesize delegate;

- (id)init
{
    self = [super init];
    if (self) {
        imageDictionary = [[NSMutableDictionary alloc] init];
    }

    return self;
}

- (void)dealloc
{
    [imageDictionary release];

    [super dealloc];
}

#pragma mark - Instance Method

- (UIImage *)syncLoadImageAtURL:(NSURL *)imageURL
{
    UIImage *image = [imageDictionary objectForKey:[imageURL absoluteString]];

    if (image == nil) {
        image = [UIImage imageWithData:[NSData dataWithContentsOfURL:imageURL]];
        if (image) {
            [imageDictionary setValue:image
                               forKey:[imageURL absoluteString]];

        }
    }

    return image;
}

- (void)asyncLoadImageAtURL:(NSURL *)imageURL
{
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
        UIImage *image = [self syncLoadImageAtURL:imageURL];

        if (delegate) {
            if (image) {
                if ([delegate respondsToSelector:@selector(imageLoader:didFinishLoadImageAtURL:withImage:)]) {
                    [delegate imageLoader:self
                  didFinishLoadImageAtURL:imageURL
                                withImage:image];
                }
            } else {
                if ([delegate respondsToSelector:@selector(imageLoader:didFailLoadImageAtURL:)]) {
                    [delegate imageLoader:self
```

```
                    didFailLoadImageAtURL:imageURL];
                }
            }
        }
    });
}

@end
```

สำหรับไฟล์ Implementation ของคลาส ImageLoader นั้น ในเมธอด init เราจะทำการสร้างตัวแปร imageDictionary เพียงอย่างเดียว

ในเมธอด syncLoadImageAtURL: นั้นเราจะตรวจสอบก่อนว่า URL ที่รับเข้ามานั้น เคยถูกโหลดแล้วหรือไม่ ถ้าเคยโหลดแล้ว ก็จะคืนรูปที่เก็บไว้ออกมา ถ้ายังไม่เคยโหลดมาก่อน ก็จะทำการโหลดรูปเข้ามาและเก็บในตัวแปร imageDictionary และคืนรูปนั้นออกมา

ในเมธอด asyncLoadImageAtURL: เราจะทำการสร้าง thread ใหม่ขึ้นมา และเรียกเมธอด syncLoadImageAtURL: จาก thread ใหม่นี้ และถ้ามีรูปคืนกลับมา (รูปโหลดได้สำเร็จ หรือเคยถูกโหลดไว้แล้ว) ก็จะเรียกไปยังเมธอด imageLoader:didFinishLoadImageAtURL:withImage: ของ delegate object

แต่ถ้าไม่มีรูปคืนออกมา (นั้นคือ เกิด error ขึ้น) ก็จะเรียกไปยังเมธอด imageLoader:didFailLoadImageAtURL: ของ delegate object

```
//
//  MyViewController.h
//
```

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

#import "ImageLoader.h"

@interface MyViewController : UIViewController
<ImageLoaderDelegate>
{
    ImageLoader *imageLoader;

    UIImageView *imageView;
    UIActivityIndicatorView *loadingIndicator;
    UITextField *urlField;
    UIButton *syncLoadButton;
    UIButton *asyncLoadButton;
}

- (void)syncLoadImage;
- (void)asyncLoadImage;

@end
```

สำหรับคลาส MyViewController นั้น จะเป็นส่วนของ UI ที่ของโปรแกรม โดยตัวโปรแกรม จะมีส่วนประกอบหลักๆอยู่ 4 ส่วนด้วยกัน คือ

1. imageView เป็นส่วนแสดงผลรูปภาพ สำหรับใช้แสดงรูปที่โหลดเสร็จแล้ว
2. urlField สำหรับใช้พิมพ์ URL ของรูปที่จะทำการโหลดลงไป
3. syncLoadButton และ asyncLoadButton สำหรับทำการโหลดรูปในแต่ละแบบ

4. loadingIndicator สำหรับแสดง indicator ว่ากำลังทำการโหลดรูปอยู่

และจะประกาศเมธอดไว้ 2 เมธอดสำหรับสั่งโหลดรูปในแต่ละแบบ

นอกจากนี้คลาสนี้ จะยังทำการ implement Protocol

ImageLoaderDelegate ไว้ด้วย

```
//
//  MyViewController.m
//

#import "MyViewController.h"

@interface MyViewController ()

@end

@implementation MyViewController

- (id)init
{
    self = [super init];
    if (self) {
        // Create variables
        imageLoader = [[ImageLoader alloc] init];
        imageLoader.delegate = self;

        // Set view property
        self.view.backgroundColor = [UIColor white-
Color];
        self.view.autoresizesSubviews = YES;
        self.view.autoresizingMask = UIViewAutoresiz-
ingFlexibleWidth | UIViewAutoresizingFlexibleHeight;

        // Create view components
        imageView = [[UIImageView alloc] init];
        imageView.frame = CGRectMake(5, 5,
self.view.frame.size.width - 10, 160);
        imageView.autoresizesSubviews = YES;
        imageView.autoresizingMask = UIViewAutoresiz-
ingFlexibleWidth | UIViewAutoresizingFlexibleBottom-
Margin;
        imageView.contentMode = UIViewContentModeS-
caleAspectFill;
        imageView.clipsToBounds = YES;
        [self.view addSubview:imageView];

        loadingIndicator = [[UIActivityIndicatorView
alloc] init];
        loadingIndicator.frame =
CGRectMake(imageView.frame.size.width / 2 - 15,
imageView.frame.size.height / 2 - 15, 30, 30);
        loadingIndicator.autoresizingMask = UIViewAu-
toresizingFlexibleTopMargin | UIViewAutoresizing-
FlexibleBottomMargin | UIViewAutoresizingFlexible-
LeftMargin | UIViewAutoresizingFlexibleRightMargin;
        loadingIndicator.activityIndicatorViewStyle
= UIActivityIndicatorViewStyleGray;
        [imageView addSubview:loadingIndicator];

        urlField = [[UITextField alloc] init];
        urlField.frame = CGRectMake(10, 180,
self.view.frame.size.width - 20, 30);
        urlField.autoresizingMask = UIViewAutoresiz-
ingFlexibleWidth | UIViewAutoresizingFlexibleBottom-
Margin;
        urlField.borderStyle = UITextBorderSty-
leRoundedRect;
        [self.view addSubview:urlField];

        syncLoadButton = [UIButton
buttonWithType:UIButtonTypeRoundedRect];
        [syncLoadButton retain];
        syncLoadButton.frame = CGRectMake(10, 220,
self.view.frame.size.width / 2 - 15, 30);
```

```
        [syncLoadButton setTitle:@"Sync Load"

forState:UIControlStateNormal];
        [syncLoadButton addTarget:self

action:@selector(syncLoadImage)

forControlEvents:UIControlEventTouchUpInside];
        [self.view addSubview:syncLoadButton];

        asyncLoadButton = [UIButton
buttonWithType:UIButtonTypeRoundedRect];
        [asyncLoadButton retain];
        asyncLoadButton.frame =
CGRectMake(self.view.frame.size.width / 2 + 5, 220,
self.view.frame.size.width / 2 - 15, 30);
        [asyncLoadButton setTitle:@"Async Load"

forState:UIControlStateNormal];
        [asyncLoadButton addTarget:self

action:@selector(asyncLoadImage)

forControlEvents:UIControlEventTouchUpInside];
        [self.view addSubview:asyncLoadButton];
    }

    return self;
}

- (void)dealloc
{
    [imageLoader release];

    [imageView release];
    [loadingIndicator release];
    [urlField release];
    [syncLoadButton release];
    [asyncLoadButton release];
    [super dealloc];
}

#pragma mark - Instance Method

- (void)syncLoadImage
{
    [urlField resignFirstResponder];
    imageView.image = nil;
    [loadingIndicator startAnimating];

    NSURL *imageURL = [NSURL
URLWithString:urlField.text];
    UIImage *image = [imageLoader
syncLoadImageAtURL:imageURL];

    if (image) {
        imageView.image = image;
    } else {
        UIAlertView *alertView = [[UIAlertView al-
loc] initWithTitle:@"Error"

message:@"Unable to load image"

delegate:nil

cancelButtonTitle:@"OK"

otherButtonTitles:nil];
        [alertView show];
        [alertView release];
    }

    [loadingIndicator stopAnimating];
}

- (void)asyncLoadImage
{
```

```
    [urlField resignFirstResponder];
    imageView.image = nil;
    [loadingIndicator startAnimating];

    NSURL *imageURL = [NSURL
URLWithString:urlField.text];
    [imageLoader asyncLoadImageAtURL:imageURL];
}

#pragma mark - ImageLoaderDelegate

-     (void)imageLoader:(ImageLoader *)imageLoader
didFinishLoadImageAtURL:(NSURL *)imageURL
              withImage:(UIImage *)image
{
    imageView.image = image;

    [loadingIndicator stopAnimating];
}

-    (void)imageLoader:(ImageLoader *)imageLoader
didFailLoadImageAtURL:(NSURL *)imageURL
{
    UIAlertView *alertView = [[UIAlertView alloc]
initWithTitle:@"Error"

message:@"Unable to load image"

delegate:nil

cancelButtonTitle:@"OK"

otherButtonTitles:nil];
    [alertView show];
    [alertView release];

    [loadingIndicator stopAnimating];
}
```

```
@end
```

สำหรับไฟล์ Implementation ของคลาส MyViewController นั้น รายละเอียดหลักๆส่วนใหญ่ของไฟล์นี้ จะเป็นการสร้างและจัดตำแหน่งขององค์ประกอบต่างๆสำหรับแสดงผล ซึ่งจะขอไม่พูดถึงในที่นี้

สำหรับเมธอด syncLoadImage จะมีการทำงานดังนี้

1. ลบรูปที่แสดงอยู่ใน imageView เก่าออก

2. สั่งให้ loadingIndicator ทำงาน

3. อ่าน URL จากที่ผู้ใช้พิมพ์เข้ามา

4. เรียกเมธอด syncLoadImageAtURL: จาก imageLoader

5. ถ้าได้รูปกลับมา ก็นำรูปมาแสดงใน imageView แต่ถ้าไม่มีรูปกลับมา ก็จะแสดงกล่องข้อความว่าไม่สามารถโหลดรูปได้

6. สั่งให้ loadingIndicator หยุดทำงาน

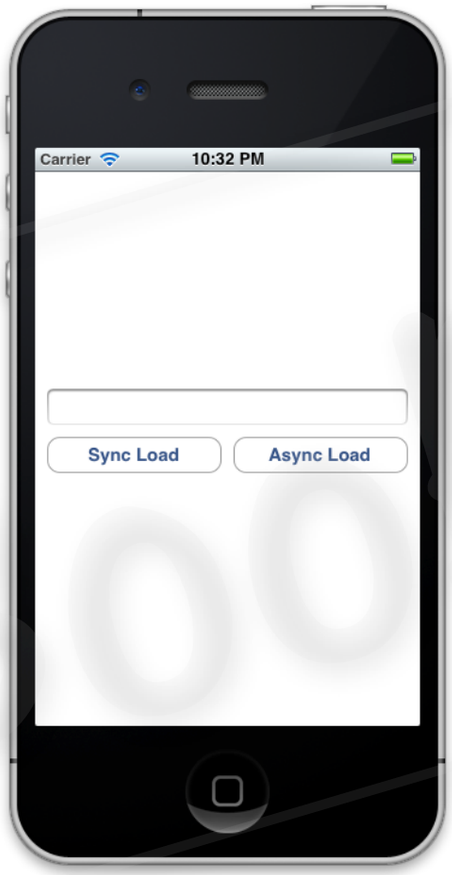
สำหรับเมธอด asyncLoadImage จะมีการทำงานเฉพาะในข้อ 1-4 (แต่จะเรียกเมธอด asyncLoadImageAtURL: แทน) เท่านั้น

ซึ่งเมื่อ imageLoader ทำงานเสร็จก็เรียกกลับมายังเมธอดของ ImageLoaderDelegate ซึ่งถ้าโหลดรูปสำเร็จ ก็จะนำรูปขึ้นมาแสดง และ

สั่งให้ loadingIndicator หยุดทำงาน (ในเมธอด imageLoader:didFinishLoadImageAtURL:withImage:)

แต่ไม่สามารถโหลดรูปได้ ก็จะแสดงกล่องข้อความ และสั่งให้ loadingIndicator หยุดทำงาน (ในเมธอด imageLoader:didFailLoadImageAtURL:)

เมื่อรันโปรแกรมนี้ จะมีหน้าตาเป็นดังรูปด้านล่างนี้



สังเกตุว่าการโหลดรูปแบบ synchronous นั้น จะทำให้ loadingIndicator มีการหยุดหมุนไปจนรูปโหลดเสร็จ ในขณะที่ asynchronous นั้น จะไม่ทำให้ loadingIndicator หยุดทำงาน

ซึ่งโดยทั่วไปแล้ว เราควรออกแบบการทำงานของโปรแกรมให้เป็นแบบ asynchronous สำหรับการทำงานใดๆ ที่ต้องใช้เวลาในการทำงาน เพื่อป้องกันไม่ให้โปรแกรมดูเหมือนหยุดทำงานไปชั่วขณะ

สรุปสิ่งที่กล่าวถึงในบทนี้

1. Protocol
2. Delegate & DataSource
3. ตัวอย่างโปรแกรม ImageLoader

ในบทนี้เราได้พูดถึงเรื่องของ Protocol, Delegate และ DataSource ซึ่งถูกนำมาใช้กันเป็นอย่างมากในหลายๆคลาส

Protocol จะใช้ระบุถึงเมธอดที่จะใช้ในการติดต่อ ซึ่งสามารถกำหนดให้เป็นเมธอดแบบ required หรือ optional ได้

Delegate และ DataSource จะเป็น object ที่ทำการ implement Protocol สำหรับใช้ในการติดต่อเมื่อเกิดเหตุการณ์ที่กำหนดไว้ขึ้น ซึ่ง Delegate จะใช้เรียก object ที่จะรองรับการทำงานเพื่อตอบสนองต่อเหตุการณ์บางอย่าง ในขณะที่ DataSource จะใช้เรียก object ที่จะใช้รองรับการเรียกข้อมูล