Objective C 語言的基礎

王昱景　Brian Wang
brian.wang.frontline@gmail.com

Objective C 歷史

- 1980 年代，由 Brad Cox 和 Tom Love 創立 StepStone 公司產生 Objective C 雛形

- Objective C 構想來自物件導向語言老祖宗 — Smalltalk (Virtual Machine 概念)

- Cox 在 C 編譯器改造 Smalltalk 誕生 OOPC (Object-Oriented Programming in C)

- 1986年出版 Object-Oriented Programming, An Evolutionary Approach 一書中正式揭開 Objective C 面紗

- 1988 年由 NeXT 公司 (離開 Apple 公司的 Steve Jobs 創辦) 向 StpStone 取得 Objective C 的授權，以作為 NeXTSTEP 作業系統的基礎 (當時稱 OpenStep API)

- 1966 年 Apple 買下 NeXT 使用 OpenStep 作業系統，也就是 Mac OS

- 其中包括開發工具 Project Builder (現在的 Xcode) 和畫面設計工具 Interface Builder
- 跟 C 有緊密關係，因此 C 的語法在 Objective C 都可以使用也可以編譯
- 前身是 NeXTSTEP 所以在 Objective C 裡可以看到 NS 開頭的類別

可可初體驗

- Cocoa (Cocoa Framework) 使用於 Mac OS X 的一種 API
- 在 Cocoa Framework 裡可以使用 Objective C、Objective C++
- iOS 量身定做的框架稱為 Cocoa Touch Framework，可以使用 Objective C、Objective C++、C 或是 C++ 來開發

Objective C

C language

Carbon

- 在 Mac OS X 推出時為了協助開發者把 OS 9 的程式移植到 OS X 使用而提出一個名詞叫碳化 (Carbonize)
- 套上 Carbon 的 API 再重新編譯後就可以成為 OS X 的程式
- Carbon 大多數屬於比較低階的，且是非物件導向的 API，速度較快
- Carbon 和 Cocoa 的發展是並行的

```
// 宣告一個 C 語言的字串變數
char *str = “HelloWorld iPhone!”;

// 宣告一個 Cocoa 的字串指標變數，其初始值由上述的字串引入
NSString *cocoaString  = [[NSString alloc] initWithCString:str];

// 釋放該變數
[cocoaString release];
```

```
// 方法宣告
-  (void) setAge: (int)age {
…
}

- setAge: (int)age {
…
}

- (id)setAge: (int)age {
…
}
```

從 C 到 Objective C

- 物件導向 - C 語言沒有物件導向的概念，有關物件導向的部分都是 Objective C 擴充而來的
- 方法與函數 - 函數主要使用於 C 語言，而方法是物件導向語言才有的產物，因為方法是依賴類別而存在，沒有類別就不會有方法的存在

```
// C 語言
int getAge(char *name) {
…
}
```

```
// Objective C 語言
- (int) ageForName:(NSString *)name {
…
}
```

- 任意資料型態 - 在 C 語言資料型態是規定很嚴格的，但 Objective C 打破規則，創造 id 這個資料類型用來代表任意資料型態
- for 的用法 - Objective C 除了 C 語言 for 迴圈外也支援像 foreach 的用法

```
for (id key in dict) {
    NSLog(@“key=%@, value=%@“, key, [dict objectForKey: key]);
}
```

- #import - #import 與 #include 的作用大致上一樣，使用 #import 可以避免重覆定義 (Duplicate definition) 的錯誤

- 布林值 - C 語言沒有布林的資料類型，卻以 0 代表 false，非 0 則代表 true，Objective C 則多布林 (BOOL) 的資料類型，並且以 YES 代表 true 而 NO 則代表 false

- 空值 - 在 C 或 C++ 是以 NULL 作為空值，而 Objective C 則是以 nil 替代 NULL，比較特殊的是可以在 Objective C 傳遞訊息給 nil ([nil message])

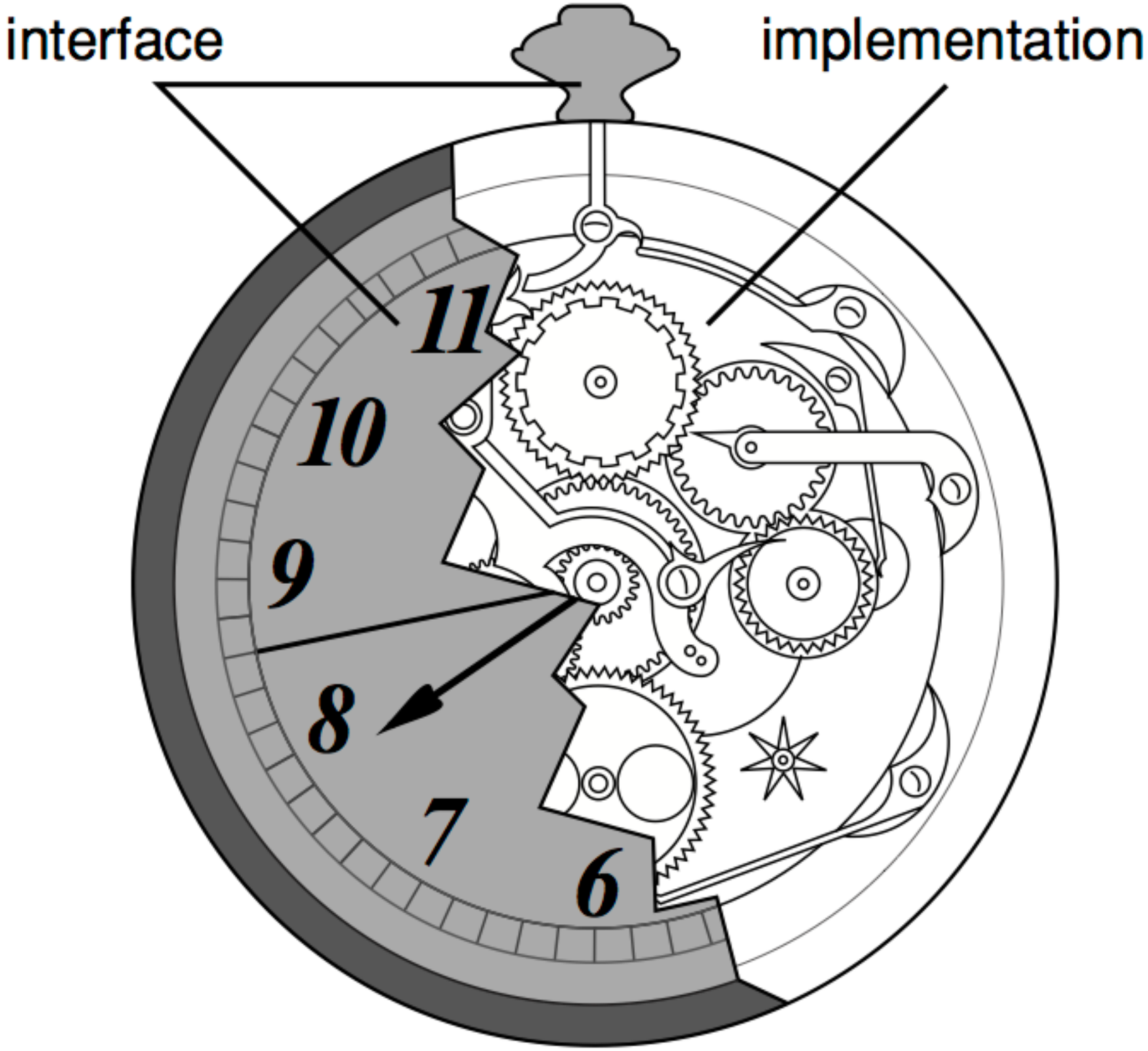
- 不支援運算子的重載 (overload)
- 不支援多重繼承 - Objective C 只支援單一繼承，所有的類別都是繼承自 NSObject 類別

Root Class

有趣的 Objective C

- Objective C 是 ANSI C 的一個超集合，因此其基本語法與 C 是一樣的

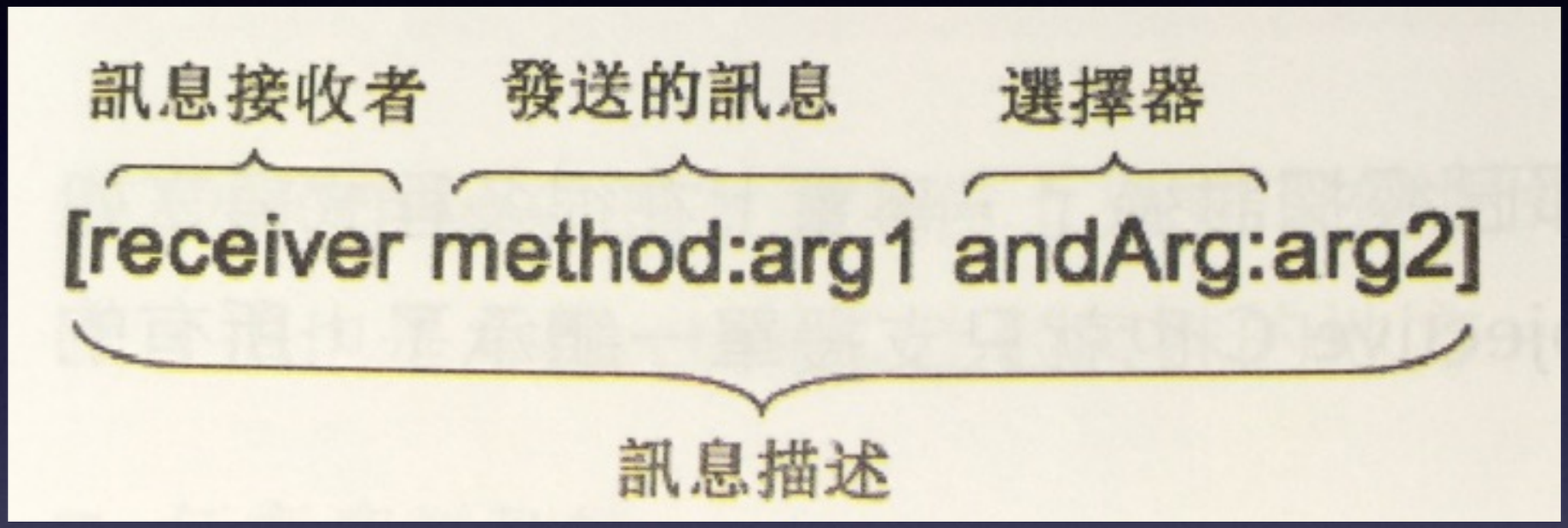
- 如同 C 一樣，Objective C 也將標題檔與原始檔分開來儲存


interface
implementation
11
10
9
8
7
6

副檔名	說明
.h	程式的標題檔，內容包含了類別、方法的宣告。通常我們也會將一些常數、巨集定義在這個檔案內。
.m	原始程式檔，內容為類別及其方法的實作，且可以包含 C 和 Objective C 的語法。
.mm	原始程式檔。只要在程式碼內參照了 C++ 的類別，那麼你應該把原始程式的副檔名更改為 .mm。
.xib	XML 格式的檔案，用來儲存使用者界面 (User Interface)，並且可以動態載入到程式內。
.plist	一般是作為設定檔使用，例如 iOS 應用程式的設定檔就是。
.strings	作為多國語系的語言檔。
.pch	用來作為所有原始檔的標題檔 (Prefix header)，如果你需要在所有的原始檔裡面加入同樣的定義或標題檔的引用，那麼你可以在這個檔案定義。
.app	編譯後而得的執行檔。事實上它是一個資料夾。

- 如果要載入標題檔，可以使用 #include 指令
- 但 Objective C 提供更好的方式 #import，可以防止標題檔不會被載入兩次，同時也會避免兩個原始檔相互載入的情況發生

- 選擇器與訊息



- 輸出到主控台 - 可以使用 NSLog() 或是 CFShow()，也可以使用 C 語言提供的 print() 與 printf()

- 類別

```objc
#import "Circle.h"  // 用來載入標題檔(*.h)

@implementation Circle  // 表示類別的實作開始

// 預設建構子(可省略)
- (id)init {
    self = [super init];
    return self;
}

// 自訂的建構子，以init開頭，必須傳回self，代表這個類別產生的實例(instance)
- (id)initWithRadiusX:(float)x andRadiusY:(float)y {
    rx = x;
    ry = y;
    return [self init];
}

// 計算圖形的面積，減號代表是instance level的方法
- (float)area {
    return M_PI * rx * ry;
}

// 訊息輸出，為靜態方法(static)
+ (void)log:(NSString *)msg {
    NSLog(@"%@", msg);
}

@end  // 類別的結束
```

```objc
#import <Foundation/Foundation.h>
#import "Circle.h"

int main(int argc, const char * argv[]) {  // C語言的入口函數
    @autoreleasepool {
        // 產生類別的物件，透過alloc配置所需的記憶體空間，再呼叫建構子就能產生物件
        Circle *circle = [[Circle alloc] initWithRadiusX:5.5f andRadiusY:16.0f];

        // 計算面積
        float area = [circle area];

        // 產生一個訊息物件，顯示計算結果
        NSString *msg = [[NSString alloc] initWithFormat:@"Area = %f", area];
        [Circle log:msg];
    }

    return 0;
}

```

- #import 與 @class - #import 會將 A 類別的成員變數與方法全部引用到 B 類別裡面，@class 只負責類別名稱的聲明而已

```objc
// A.h
@class B;

@interface A {
    B *b;
}

@end


// A.m
#import "A.h"
#import "B.h"
```

- 變數的存取權限
  - @public - 公開的成員變數，在任一個類別裡都可以被存取
  - @protected - 保護級的成員變數，也是預設的存取權限，只有該類別或是其子類別才可以存取
  - @private - 私有的成員變數，除該成員變數所在的類別外，其他的類別都不可存取

- @package - 函式庫級別的成員變數
  - 64 bit OS - 該成員變數只能在類別所在的函式庫 (Framework) 內被存取 (@public)，而在函式庫外就無法存取 (@private)
  - 32 bit OS - @package 則與 @public 一樣

```
@interface Configuration: NSObject {
    @public
        NSDictionary *data;
    @private
        NSString *filename;
}
```

- 透過 “->” 存取 data

```
NSDictionary *dict = config -> data;
```

- 在 .m 檔裡宣告成員變數 - 避免曝露給其他程式

```
// .h 檔案
@interface MyClass
- (void) myMethod;
@end

// .m 檔案
@implementation MyClass {
    NSString *myVariable;
}
- (void) myMethod {
    // 該方法的實作
}
@end
```

- 使用分類 (Category) - 成員變數、方法皆為私有的

```
// .h 檔案
@interface MyClass
@end

// .m 檔案
@interface MyClass {
    NSString *myVariable;
}
- (void) myMethod {
@end
@implementation MyClass
- (void) myMethod {
    // 該方法的實作
}
@end
```

- 建構子與解構子
  - alloc、(id)init、(id)initXXX
  - release、dealloc

- 點運算子 - Getter 與 Setter

```
employee.name = @“Eric Lin”;
NSString *name = employee.name;
```

```
[employee setName:@“Eric Lin”];
NSString *name = [employee getName];

[[employee getBoss] setName:@“Jack” andAge:45];
[employee.boss setName:@“Jack” andAge:45];
```

- 點運算子必須搭配 @property 和 @synthesize (Xcode 4.5 以後省略) 這兩個前置指令 (directive) 使用

```
@interface UserProfile:NSObject {
    NSString *name;
    NSString *address;
}
@property (copy, nonatomic, getter=getUserName) NSString *name;
@property (copy, nonatomic) NSString *address;
@end
```

```
@implementation UserProfile
    @synthesize name;
    @synthesize address;
@end
```

- @property (attributes, …) type varName;
- attributes 用來設定 Getter 與 Setter 的各項屬性
  - 設定 Getter 與 Setter 名稱 (setter=setName:)
  - 可讀寫設定 (只能其一)
    - readwriter - 為預設值，表示這變數可以讀寫 (有 Getter 和 Setter)
    - readonly - 唯讀，只有 Getter 沒有 Setter

- Setter 設定
  - assign - 為預設值，會將等號右邊的數值與位址賦予給等號的左邊
  - retain - 在賦予值的時候，會自動呼叫 retain 方法 (限Objective C 的物件類型)，先前的數值則會被呼叫 release 釋放
  - copy - 在賦予值得時候，會自動呼叫 copy 方法 (此物件類型必須實作 NSCopying) 來賦予值，先前的數值則會被呼叫 release 釋放

- Getter 設定
  - atomic - 預設值，可確保數值是執行緒安全 (thread safe) 的，但會犧牲部分效率
  - nonatomic - 大部份使用

- 自動參照計數 (ARC) 的設定 (iOS 5後)
  - strong (retain) - 強參照型物件
  - weak (assign) - 弱參照型物件
  - unsafe_unretained - 如同弱參照型物件，但最終該位址內含值不會被設定為 nil
  - 強參照類型與弱參照型的差別在於如果後者所參照的物件被釋放掉之後，那麼弱參照物件也會被釋放掉，同時會被設定為 nil

```
@interface UserProfile:NSObject
@property (copy, nonatomic, getter=getUserName) NSString *name;
@property (copy, nonatomic) NSString *address;
@end
```

```
@implementation UserProfile
    @synthesize name = p_name;
    @synthesize address;
@end
```

- 協定 Protocol

```
// WindowEvent.h
@protocol WindowEvent
    - (void)close;
    - (void)open;
@end

// MyClass.h
@interface MyClass:NSObject < WindowEvent>
@end

// MyClass.m
@implementation MyClass {
    - (void)close {
    }
    - (void)open {
    }
}
@end
```

- 使用多個協定時可以用逗點隔開，例如 <Protocol1, Protocol2>，類別實作必須包含 Protocol1 與 Protocol2 的方法
- 協定在 iOS 的開發主要是用於代理程式 (Delegate) 的事件處理上
- 在定義協定的時候，如果希望有些方法可以不需要實作，可以在方法的前面加上 @optional，反之則可以加上 @required (預設值)

```
@protocol WindowEvent
    @required
        - (void)close;
        - (void)open;
    @optional
        - (void)detectTouch;
@end
```

- 分類 Category
  - 如果要在一個類別 (Class) 加上一個新的方法時，通常就是透過繼承的方式來加入新的方法
  - 在既有的類別加入新的功能的時候，Objective C 提供一個名為 Category 的方式來擴充類別原有的功能

```
@interface NSString (CheckEmpty)
    - (BOOL)isEmpty;
@end
@implementation NSString (CheckEmpty)
    - (BOOL)isEmpty {
        return self.count == 0 ? YES : NO;
    }
@end
```

- Block 物件
  - iOS 4.0 以後才支援
- iOS 應用程式內大量使用了 Target-Action 的機制，所以可能需要在同一個類別裡面撰寫很多回呼函數

```
- (void)testBlock {
    void (^swap)(int *, int *) = ^(int *x, int *y) {
        int z = *x;
        *x = *y;
        *y = z;
    };
    int x = 100, y = 200;
    printf("Before: x=%d, y=%d\n", x, y);
    swap(&x, &y);
    printf("After: x=%d, y=%d\n", x, y);
}
```

```
[UIView beginAnimations:nil context:nil];
[UIView setAnimationDuration:2.0f];
self.animView.alpha = 1.0;
self. animView.transform = CGAffineTransformMakeRotation(M_PI);
[UIView commitAnimations];
```

精簡

```
[UIView animateWithDuration:2.0f
    animations:^ {
        self.animView.alpha = 1.0;
        self. animView.transform = CGAffineTransformMakeRotation(M_PI);
    }
];
```

- 例外捕捉
  - @try - 用來包含可能會產生錯誤的程式碼
  - @catch - 用來捕捉錯誤
  - @finally - 在離開 @try 區塊前一定會被執行到的程式碼

```
Mistake *mistake = [[Mistake alloc]init];
@try{
    [mistake make];
}@catch(NSException *e1){
    NSlog(@"例外名稱:%@, 理由:%@",[e1 name],[e1 reason]);
}@catch(id e2){
    NSLog(@"其他的不處理");
}@finally{
    [mistake release];
}
```

- 例外拋出 - 拋出例外的方式是透過 @throw 這個指令

```
-(void) handleFile:(NSString *) file {
    if( [[NSFileManager defaultManager] fileExistsAtPath:file] ){
          NSException *exception = [NSException exceptionWithName: @"FileException"
                                                           reason: @"錯誤理由：檔案不存在！"
                                                           userInfo: nil];
          @throw exception;
    }
    // 略 …
}
```

- NSAssert - 使用 NSAssert 進行檢查的動作，只有在 Debug 模式下會作用
  - NSAssert(條件式, 錯誤提示);
  - NSAssert([self myMoney] < 10000, @“金額錯誤!”);
  - 當條件式不成立 (NO) 的情況下，應用程式會退出，並且顯示錯誤提示

資料類型

- 原始資料類型

Basic Data Types

Type	*What It Is*	*Example*	*Size*
`Char`	A character	N or g	1 byte
`Int`	An integer — a whole number	42, -42, 1234	4 bytes
`Float`	Single precision floating point number	1.99999	4 bytes
`Double`	Double precision floating point number	1.9999999999	8 bytes

Additional Types Based on int

Type	*What It Is*	*Example*	*Size*
`Short`	A short integer	42, -42 1234	2 bytes
`Long`	A double short	42, -42, 1234	4 bytes
`long long`	A double long	1.99999	8 bytes

類型	格式化符號	說明
char	%c	字元
int	%i(10 進位)、%o(8 進位)、%x(16 進位)	整數
float	%f	單精度浮點數，後面可加 f 或 F 來表示
double	%f、%e、%g	雙精度浮點數

類型	格式化符號	說明
unsigned char	同 char	無記號字元
unsigned int	%u(10 進位)、%o(8 進位)、%x(16 進位)	無記號整數，後面可加 u 或 U 表示
unsigned short int	%hu(10 進位)、%ho(8 進位)、%hx(16 進位)	無記號短整數
unsigned long	%lu(10 進位)、%lo(8 進位)、%lx(16 進位)	無記號長整數 , 後面可加 ul 或 UL 表示
short int	%hi(10 進位)、%ho(8 進位)、%hx(16 進位)	短整數
long int	%li(10 進位)、%lo(8 進位)、%lx(16 進位)	長整數，後面可加 l 或 L 表示
long double	%Lf、%Le、%Lg	雙精度浮點數，後面可加 l 或 L 表示
long long int	%lli(10 進位)、%llo(8 進位)、%llx(16 進位)	8 bytes 以上精度的整數

- 擴充的資料類型
  - BOOL - 布林值，實際上是個整數值，其數值只有 YES 或 NO 兩種
  - id - 通用類型，弱型態變數，由於本身已經是個指標，所以不需要再加上星號

- IMP - 指向 Objective C 方法的指標，可以透過發送 methodFor: 訊息給一個 NSObject 物件來取得 IMP
  - IMP myImp = [myObj methodFor:selObj];
- nil - 和 C 語言的 NULL 相同，代表這個物件指向空值
- Nil - 和小寫的 nil 不一樣，代表一個指向空的類別而非物件

- SEL - Selector 類型，用以表示一個方法的名字
  - SEL myMethod = @selector(sayHello);
  - SEL myMethod = NSSelectorFromString(@“sayHello:”);
  - [myObj perform:myMethod withObject:@“Hello World!”];

- STR - 即 char * 類型，用以代表 C 語言的字串
- IBOutlet / IBAction - 是巨集的定義，與 void 是相同的意思，只是用來作為對 Xcode 提示用
- IBOutCollection - 代表 IBOutlet 的集合，以陣列的型態存在

- 動態識別
  - Objective C 是一套動態識別的語言，資料類型可不需再編譯時期決定，而是在執行時期決定的
  - 由於 NSObject 是所有類別的父類別，因此也很容易做到動態識別
  - - (BOOL)isKindOfClass:classObj
  - - (BOOL)isMemberOfClass:classObj

- - (BOOL)respondsToSelector:selector
- - (Class)class

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *hello = @"Hello World!";

        if ([hello isKindOfClass:[NSString class]] == YES) {
            printf("Return YES");
        } else {
            printf("Return NO");
        }
    }
    return 0;
}

```

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        Class ios5Class = NSClassFromString(@"NSString");

        if (nil != ios5Class) {
            NSLog(@"這是 iOS 5");
        } else {
            NSLog(@"這不是 iOS 5");
        }

        // iOS 4.2
        if ([NSString class]) {
            NSLog(@"這是 iOS 5");
        } else {
            NSLog(@"這不是 iOS 5");
        }
    }
    return 0;
}

```

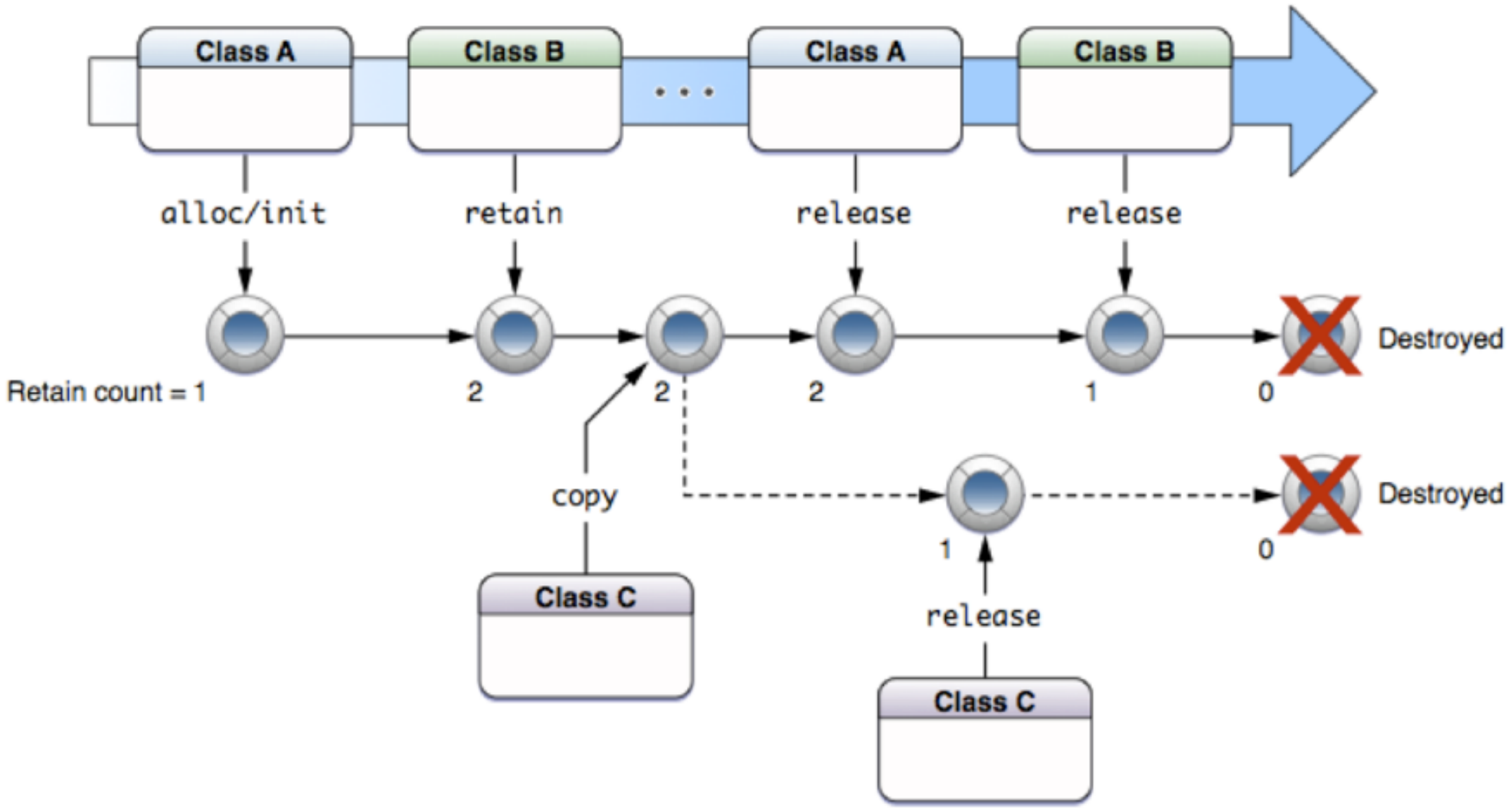
記憶體管理

- 沒有 Garbage Collection 機制
- 當使用 new、alloc-init、retain、copy 而擁有物件時，就必須負責管理回收該物件
- 管理不良時會發生 dangling pointer 或 memory leak 的問題

- 記憶體管理模式
  - 採用參照計數 (reference counting) 的方式

  - 要觀察計數器的值，可以使用 NSObject 的 retainCount 方法
    - unsigned *rc = [myObject retainCount];

- 凡是用 new、alloc-init、retain、copy 就一定得呼叫 release
- 只有當計數器的數值為 0 時，資源才會真正被釋放
- 同一個物件 alloc(new) + retain + copy = release + autorelease


Class A
Class B
Class A
Class B
alloc/init
retain
release
release
Retain count = 1
2
2
2
1
0
Destroyed
copy
Class C
1
release
Class C
0
Destroyed

- 自動釋放記憶體
  - NSAutoreleasePool 機制是固定期清掃垃圾的池子 (pool)
- 要使用這個機制，必須先啟用 NSAutoreleasePool

- 自動參考計數 ARC
  - 只能用於 Objective C 的物件上，對於 CoreFoundation (CF 開頭的類別) 以及 CoreGraphics (CG 開頭的類別) 都沒有辦法應用
- 不要在程式碼內呼叫 retain、release、autorelease、dealloc 等方法
- 不要在 C 的結構 (stuct) 裡面儲存物件的指標

- 不要直接在物件與非物件間直接進行強制轉型
- 不要再使用 NSAutoreleasePool 物件，改用 @autoreleasepool 新的指令來代替
- 不再需要 NSZone (用來複製或產生某個記憶體區塊內容的 C 結構)
- 屬性 (property) 前面不得以 new 開頭
- 適當的使用變數的所有權修飾子

命名風格

- 駝峰式大小寫規範 (Camel Case)
- 類別的命名 - Cocoa Framework 的類別都是以 NS 開頭

- 變數的命名
  - 以小寫開頭
  - 不使用底線開頭 - 底線開頭的變數名稱通常作為 Objective C 內部使用

```
NSString *_name;
```

- 與資料類型有關的命名

```
NSArray *names;
NSImage *logoImage;
```

- Selector 的命名
  - 不使用底線開頭的名稱
  - 考慮 selector 後面參數的意義

[fileWrapper writerToFile:filename append:YES];

- 讓訊息名稱看起來像是一段句子

[user setName:@“Tom” withAge:21 andAddress:@“Taipei”];

- Getter 與 Setter 的命名
  - Getter 的方法名稱前面不需加上 get，若回傳值為 BOOL 則以 is 開頭
  - Setter 則一律以 set 開頭

```
- (NSString *)color;
- (BOOL)isSelectable;
```

- 若訊息 (Message) 有回傳值時，通常會以該回傳值開頭

```
[NSString stringWithFormat:@“Total:%.2f”, 10000.25f];
```

- 靜態方法的命名 - 如果是透過靜態方法來產生物件，通常會以 new 開頭，並以 id 為回傳值的資料型態

```
+ (id)newMagic;
```

框架庫

- 框架庫 (Framework)
- Foundation.framework - 類別的原型宣告，不包含無法編譯成功
- UIKit.framework - 視覺元件
- QuartzCore.framework - 動畫特效
- AudioToolbox.framework - 聲音

Objective-C Style Guide

- https://github.com/NYTimes/objective-c-style-guide