



# LẬP TRÌNH iOS

## Module 2

☞ Click vào phụ lục để chuyển tới bài cần đọc

### Phụ lục

Bài 1 Tính kế thừa, Protocol & Delegate .....	2
Bài 2 Các kiểu dữ liệu cơ sở trong Objective-C.....	13
Bài 3 Vấn đề cấp phát và thu hồi bộ nhớ .....	27
Bài 4 Mô hình MVC.....	39
Bài 5 Các điều khiển lựa chọn và điều hướng .....	51
Bài 6 Dữ liệu dạng tập hợp nâng cao .....	70
Bài 7 Tạo và đọc ghi trên thư mục dự án .....	90
Bài 8 Xây dựng giao diện mới với các controller .....	112
Bài 9 Các điều khiển trạng thái và điều chỉnh .....	132



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 1. *Tính kế thừa, Protocol & Delegate*

Ngành Mạng & Lập trình thiết bị di động



2014

2014



## Nội dung



1. Tính kế thừa
2. Protocol
3. Delegate



### 1.1 Tính kế thừa



- **Đặt vấn đề:**
  - Xây dựng các lớp sau:
    - Lớp hình chữ nhật
    - Lớp tam giác
  - Trong mỗi lớp yêu cầu cần có phương thức sau :
    - Phương thức tính diện tích
      - Tam giác:  $S = (a*h)/2$  (h: chiều cao, a: chiều dài cạnh đối đường cao)
      - Hình chữ nhật:  $S = a*b$  (a: chiều dài, b: chiều rộng)



## 1.1 Tính kế thừa



### □ Giải quyết vấn đề:

- Xây dựng các lớp `HinhChuNhat`:

```
#import <Foundation/Foundation.h> // HinhChuNhat.m  
  
// HinhChuNhat.h  
  
@interface HinhChuNhat : NSObject {  
}  
  
@property int chieuDai;  
@property int chieuRong;  
  
- (int) tinhDienTich:  
(int)chieuDai :(int)chieuRong;  
  
@end  
  
@end
```



## 1.1 Tính kế thừa



### □ Giải quyết vấn đề:

- Xây dựng các lớp `HinhTamGiac`:

```
// HinhTamGiac.h // HinhTamGiac.m  
  
@interface HinhTamGiac : NSObject #import "HinhTamGiac.h"  
  
@property int chieuCao;  
@property int chieuDaiCanhDoi;  
  
- (int) tinhDienTich:  
(int)chieuCao :(int)chieuDaiCanhDoi {  
    return (chieuDaiCanhDoi*chieuCao)/2;  
}  
  
@end @end
```



## 1.1 Tính kế thừa



### ❑ Nhận xét cách vấn đề và cách giải quyết vấn đề:

- Lớp hình chữ nhật và hình tam giác có số lượng thuộc tính giống nhau.
- Cả hai lớp đều là đa giác.
- Phải định nghĩa 2 lần cho hai phương thức tính diện tích ở hai lớp.
- Phải viết lại nếu mở rộng trên các đa giác khác.



## 1.1 Tính kế thừa



### ❑ Tính kế thừa:

- Tạo lớp với những thuộc tính nhất định, sau đó dùng các lớp khác dẫn xuất lại tất cả thuộc tính.
- Lớp kế thừa có thể mở rộng thêm thuộc tính nếu cần thiết.
- Lớp kế thừa có thể tiếp tục được kế thừa, khi đó hình cây kế thừa.



- Trong Objective-C, không có tính chất đa kế thừa, NSObject là lớp gốc.





## 1.1 Tính kế thừa

### □ Giải quyết vấn đề áp dụng tính kế thừa:

- Xây dựng lớp `HinhDaGiac`, trong lớp này xây dựng thuộc tính cạnh a, cạnh b và phương thức tính diện tích.

```
// HinhDaGiac.h                                // HinhDaGiac.m
#import <Foundation/Foundation.h>              #import "HinhDaGiac.h"
@interface HinhDaGiac : NSObject                @implementation HinhDaGiac
@property float a;                             - (float)tinhDienTich {
@property float b;                           return self.a + self.b;
}                                             }
- (float)tinhDienTich;                      @end
@end
```



## 1.1 Tính kế thừa

### □ Giải quyết vấn đề áp dụng tính kế thừa:

- Xây dựng lớp `HinhChuNhat`, `HinhTamGiac` kế thừa từ lớp `HinhDaGiac` và thực thi phương thức tính diện tích.

```
// HinhChuNhat.m                                // HinhTamGiac.m
#import "HinhChuNhat.h"                         #import "HinhTamGiac.h"
@implementation HinhChuNhat                      @implementation HinhTamGiac
- (float)tinhDienTich {                         - (float)tinhDienTich {
    return self.a * self.b;                     return (self.a * self.b)/2;
}                                              }
@end
```



## Nội dung



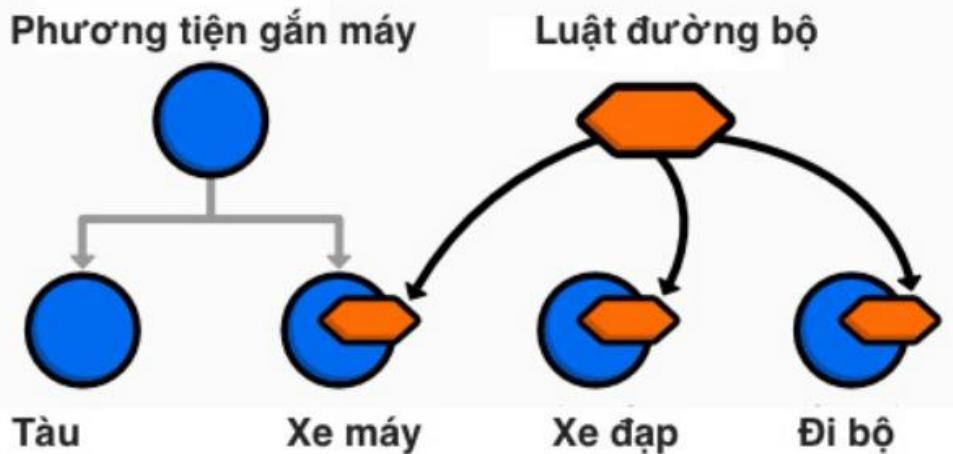
1. Tính kế thừa
2. Protocol
  - Khái niệm
  - Sử dụng Protocol trong lớp
  - Kiểm tra lớp sử dụng
3. Delegate



### 2.1 Khái niệm



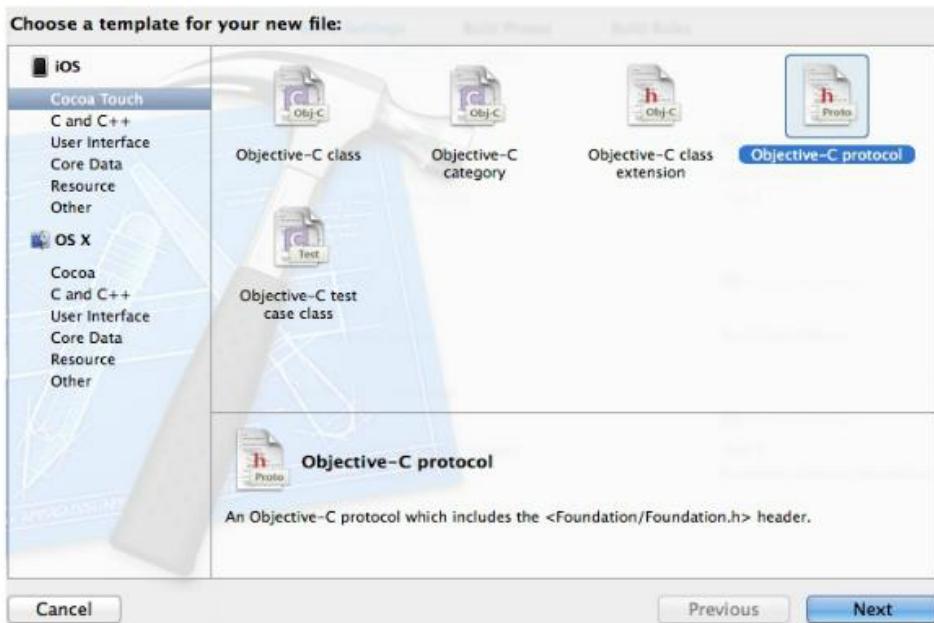
- ❑ Protocol bao gồm một tập các thuộc tính và phương thức cho phép xây dựng các lớp sử dụng có cùng cách thức hoạt động:
  - Ví dụ thực tế:



## 2.2 Sử dụng Protocol trong lớp



### ❑ Tạo Protocol:



## 2.2 Sử dụng Protocol trong lớp



### ❑ Khai báo Protocol:

```
// LuatDuongBo.h

#import <Foundation/Foundation.h>

@protocol LuatDuongBo <NSObject>

- (void) tinHieuDung;
- (void) tinHieuReTrai;
- (void) tinHieuRePhai;

@end
```





## 2.2 Sử dụng Protocol trong lớp

- ❑ Khai báo lớp XeDap sử dụng protocol LuatDuongBo:

```
// XeDap.h                                     // XeDap.m
#import "LuatDuongBo.h"                      #import "XeDap.h"
@interface XeDap : NSObject <LuatDuongBo>      @implementation XeDap
@end                                         - (void)tinHieuDung {
                                              NSLog(@"Tín hiệu dừng");
}
- (void)tinHieuRePhai {
                                              NSLog(@"Tín hiệu được rẽ phải");
}
- (void)tinHieuReTrai {
                                              NSLog(@"Tín hiệu được rẽ trái");
}
@end
```



## 2.3 Kiểm tra lớp sử dụng

- ❑ Có thể kiểm tra một lớp có sử dụng protocol hay không thông qua phương `conformsToProtocol`.

- Ví dụ: kiểm tra đối tượng phuongTien trước khi thực thi phương thức trong protocol.

```
id phuongTien = [[PhuongTien alloc] init];
[phuongTien tinHieuReTrai];
phuongTien = [[XeDap alloc] init];
if ([phuongTien conformsToProtocol:@protocol(LuatDuongBo) ]) {
    [phuongTien tinHieuDung];
    [phuongTien tinHieuRePhai];
}
```



## Nội dung



1. Tính kế thừa
2. Protocol
3. Delegate
  - Khái niệm
  - Sử dụng Delegate kết hợp Protocol



### 3.1 Khái niệm Delegate



- ❑Delegate có sử dụng như một cách thức để một đối tượng có thể gọi đến đối tượng khác thực thi một hành động nào đó và nhận kết quả trả về.
- ❑Có thể minh họa Delegate qua các bước sau:
  - A uỷ thác đối tượng cho B.
  - B thực hiện tạo tham chiếu đến A.
  - A thực hiện phương thức uỷ thác khai báo trong B.
  - B thông báo kết quả thông qua phương thức uỷ thác





## 3.2 Sử dụng Delegate kết hợp Protocol

- Khai báo Delegate: sử dụng Protocol xây dựng hàm thông báo.

```
// SampleProtocolDelegate.h
#import <Foundation/Foundation.h>
@protocol SampleProtocolDelegate <NSObject>
@required
- (void) processCompleted;
@end
@interface SampleProtocolDelegate : NSObject {
    id <SampleProtocolDelegate> _delegate;
}
@property (nonatomic, strong) id delegate;
- (void) startSampleProcess;
@end

// SampleProtocolDelegate.m
#import "SampleProtocolDelegate.h"
@implementation SampleProtocolDelegate
- (void)startSampleProcess {
    [NSTimer scheduledTimerWithTimeInterval:3.0
        target:self.delegate
        selector:@selector(processCompleted)
        userInfo:nil
        repeats:NO];
}
@end
```



## 3.2 Sử dụng Delegate kết hợp Protocol

- Sử dụng Delegate trong ViewController

```
@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    SampleProtocolDelegate *samplePD = [[SampleProtocolDelegate alloc] init];
    samplePD.delegate = self;
    [labelProcess setText:@"Processing..."];
    [samplePD startSampleProcess];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

#pragma mark - Sample protocol delegate
- (void)processCompleted {
    [labelProcess setText:@"Process Complete"];
}
```



## Thảo luận





Trường ĐH Khoa Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 2. Các kiểu dữ liệu cơ sở Objective-C

Ngành Mạng & Thiết bị di động

2014



5014



## Nội dung



1. Tổng quan
2. Một số kiểu dữ liệu cơ sở



## 1 Tổng quan



- q **Objective-C kế thừa tất cả kiểu dữ liệu của C như int, float, double**  
...
- q **Objective-C tự định nghĩa nhiều lớp cung cấp cấu trúc dữ liệu hướng đối tượng như Strings, Dictionaries, Dates ...**



## Nội dung



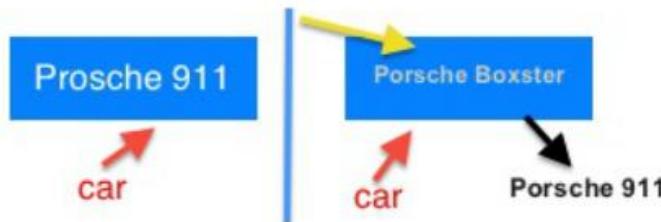
1. Tổng quan
2. Một số kiểu dữ liệu cơ sở
  - NSMutableString
    - § Khởi tạo chuỗi.
    - § Mở rộng chuỗi.
    - § Thay đổi/xoá chuỗi con trong chuỗi.
  - NSNumber
  - NSDecimalNumber
  - NSDate



### 2.1 NSMutableString



- q **NSMutableString** là lớp kế thừa từ lớp **NSString**.
- q Không giống với **NSString** nó có thể thay đổi nội dung mà không cần tạo ra đối tượng mới. Điều này làm cho người lập trình dễ dàng chỉnh sửa những thao tác nhỏ trên chuỗi hơn.





## 2.1.1 Khởi tạo chuỗi

- q Các cách tạo một chuỗi kiểu `NSMutableString`.

- Tạo chuỗi với phương thức `stringWithString`.
- Tạo giá trị mới cho chuỗi với phương thức `setString`.

```
NSMutableString *car = [NSMutableString
stringWithString:@"Porsche 911"];
NSLog(@"%@",car);
// Porsche 911
[car setString:@"Porsche Boxster"];
NSLog(@"%@",car);
// Porsche Boxster
```



## 2.1.2 Mở rộng chuỗi

- q Trong `NSMutableString` ta có thể thêm chuỗi vào đối tượng.

- Dùng `appendString`, `appendStringFor` `mat` để mở rộng chuỗi.
- Dùng `insertString` để chèn một chuỗi vào chuỗi.

```
NSMutableString *car = [NSMutableString
stringWithCapacity:20];
NSString *model = @"458 Spider";
[car setString:@"Ferrari"];
[car appendString:model];
NSLog(@"%@", car); // Ferrari458 Spider
[car setString:@"Ferrari"];
[car appendFormat:@"%@ %@", model];
NSLog(@"%@", car); // Ferrari 458 Spider
[car setString:@"Ferrari Spider"];
[car insertString:@"458 " atIndex:8];
NSLog(@"%@", car); // Ferrari 458 Spider
```



## 2.1.3 Thay đổi/xoá chuỗi con trong chuỗi



- q Ta có thể thay đổi nội dung của chuỗi mà không cần tạo đối tượng mới.

- **replaceCharactersInRange** dùng để thay đổi chuỗi con.
- **deleteCharactersInRange** dùng để xoá chuỗi con.

```
NSMutableString *car = [NSMutableString  
stringWithCapacity:20];  
  
[car setString:@"Lotus Elise"];  
  
[car replaceCharactersInRange:NSMakeRange(6, 5)  
withString:@"Exige"];  
  
 NSLog(@"%@", car);  
  
// Lotus Exige  
  
[car deleteCharactersInRange:NSMakeRange(5, 6)];  
  
 NSLog(@"%@", car);  
  
// Lotus
```



## Nội dung



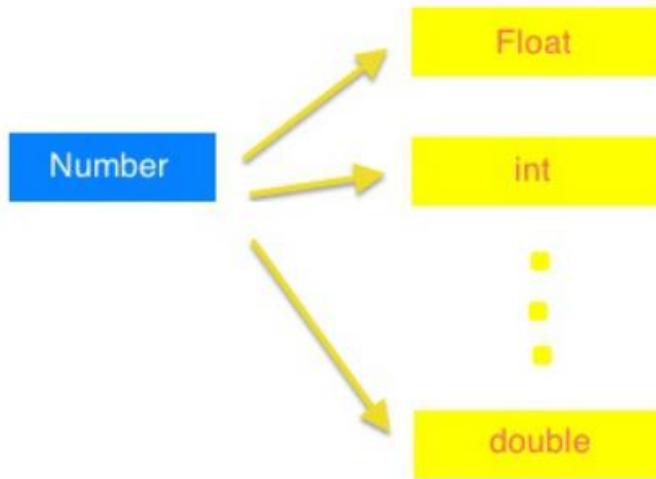
1. Tổng quan
2. Một số kiểu dữ liệu cơ sở
  - NSMutableString
  - NSNumber
    - § Khởi tạo number
    - § Đổi sang kiểu cơ bản/chuỗi
    - § So sánh number
  - NSDecimalNumber
  - NSDate





## 2.2 NSNumber

- NSNumber là một đối tượng chứa các kiểu dữ liệu số trong C như int, float, double, ...



### 2.2.1 Khởi tạo number

- NSNumber cung cấp một số cách để tạo các giá trị thuộc kiểu dữ liệu cơ sở.

<ul style="list-style-type: none"> <li>Tạo number với các phương thức.</li> <li>Tạo bằng cách gán giá trị cho number.</li> </ul>	<pre> NSNumber *aChar = [NSNumber numberWithChar:'z']; NSNumber *anInt = [NSNumber numberWithInt:2147483647]; NSNumber *aFloat = [NSNumber numberWithFloat:26.99f];  NSNumber *aChar = @'z'; NSNumber *anInt = @2147483647; NSNumber *aFloat = @26.99F;  // 122 - 2147483647 - 26.99   </pre>
--	---



## 2.2.2 Đổi sang kiểu cơ bản/chuỗi



- q Đổi kiểu NSNumber sang kiểu cơ bản hoặc kiểu chuỗi.

- Dùng các phương thức để đổi sang kiểu cơ bản/chuỗi
- NSNumber còn cho phép dùng %@ để hiển thị.

```
NSNumber *anIntTemp = [NSNumber numberWithInt:42];
float asFloat = [anIntTemp floatValue];
NSLog(@"%@", asFloat);
NSString *asString = [anIntTemp stringValue];
NSLog(@"%@", asString);
NSLog(@"%@", anIntTemp);

// 42.00 - 42 - 42
```



## 2.2.3 So sánh number



- q Sử dụng phương thức compare để so sánh.

```
NSNumber *anInt = @27;
NSNumber *anotherInt = @42;
NSComparisonResult result = [anInt compare:anotherInt];
if (result == NSOrderedAscending) {
    NSLog(@"anInt < anotherInt");
} else if (result == NSOrderedSame) {
    NSLog(@"anInt == anotherInt ");
} else if (result == NSOrderedDescending) {
    NSLog(@"anInt > anotherInt ");
}
// anInt < anotherInt
```



## Nội dung



1. Tổng quan
2. Một số kiểu dữ liệu cơ sở
  - NSMutableString
  - NSNumber
  - NSDecimalNumber
    - § Khởi tạo NSDecimalNumber
    - § Các phép tính
    - § So sánh NSDecimalNumber
  - NSDate



## 2.3 NSDecimalNumber



- q NSDecimalNumber cung cấp dấu chấm tĩnh, được thiết kế dựa trên hệ thập phân. Có thể đại diện cho bất kỳ số nào được hiển thị như  $x * 10^y$ .

$$15.99 = + \boxed{1} \boxed{5} \boxed{9} \boxed{9} \times 10^{\boxed{-2}}$$

Dấu      Phần định trị      cơ số





## 2.3.1 Khởi tạo NSDecimalNumber

q NSDecimalNumber có các phương thức khởi tạo sau.

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>Dùng phương thức để tạo đối tượng NSDecimalNumber với phần trị số, phần cơ số, phần dấu.</li> <li>NSDecimalNumber còn dùng hàm <b>decimalNumberWithString</b> để khởi tạo bằng chuỗi.</li> </ul> | <pre>NSDecimalNumber *a = [NSDecimalNumber decimalNumberWithMantissa:1299 exponent:-2 isNegative:NO]; NSLog(@"%@", a); // 12.99 = 1299 * 10^-2 * 1</pre><br><pre>NSDecimalNumber *b = [NSDecimalNumber decimalNumberWithString:@"33.99"]; NSLog(@"%@", b); // 33.99</pre> |
|---|---|



## 2.3.2 Các phép tính

q Ta có thể tính toán với những phương thức được lớp NSDecimalNumber hỗ trợ.

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>Sử dụng một số phép tính trong NSDecimalNumber.</li> </ul> | <pre>NSDecimalNumber *cong = [a decimalNumberByAdding:b]; NSDecimalNumber *tru = [a decimalNumberBySubtracting:b]; NSDecimalNumber *nhan = [a decimalNumberByMultiplyingBy:b];  NSLog(@"a + b = %@", cong); // a + b = 46.98 NSLog(@"a - b = %@", tru); // a - b = -21 NSLog(@"a * b = %@", nhan); // a * b = 441.5301</pre> |
|---|--|





## 2.2.3 So sánh NSDecimalNumber

- q Sử dụng phương thức compare để so sánh.

```
NSComparisonResult result = [a compare:b];  
if (result == NSOrderedAscending) {  
    NSLog(@"a < b");  
} else if (result == NSOrderedSame) {  
    NSLog(@"a == b");  
} else if (result == NSOrderedDescending) {  
    NSLog(@"a > b");  
}  
// Kết quả: a < b
```



## Nội dung

1. Tổng quan
2. Một số kiểu dữ liệu cơ sở
  - NSMutableString
  - NSNumber
  - NSDecimalNumber
  - NSDate
    - § Khởi tạo NSDate
    - § Một số lớp thao tác với NSDate
    - § So sánh NSDate
    - § Một số phương thức khác





## 2.4 NSDate

- q NSDate là lớp đối tượng dùng để lưu trữ ngày, giờ.
- q Để thao tác tốt hơn với NSDate ta sử dụng một số lớp sau:
  - NSDateComponents
  - NSDateFormatter
  - NSCalendar



### 2.4.1 Khởi tạo NSDate

- q Lớp NSDate chỉ cung cấp vài phương thức ở mức độ thấp để tạo ngày.

- Tạo NSDate với phương thức **date** để lấy ngày giờ hiện tại.
- Tạo với phương thức **dateWithTimeInterval:sinceDate:** để tạo một ngày cách ngày hiện tại một khoảng thời gian tính bằng giây.

```

NSDate *hienTai = [NSDate date];
NSTimeInterval motTuan = 7 * 24 * 60 * 60;
NSDate *tuanTruoc = [NSDate dateWithTimeInterval:
-motTuan sinceDate:hienTai];
NSDate *tuanSau = [NSDate
dateWithTimeInterval:motTuan sinceDate:hienTai];
NSLog(@"%@", "Tuần trước: %@", tuanTruoc);
// 2014-03-12 02:55:09 +0000
NSLog(@"%@", "Hiện tại: %@", hienTai);
// 2014-03-19 02:55:09 +0000
NSLog(@"%@", "Tuần sau: %@", tuanSau);
// 2014-03-26 02:55:09 +0000

```



## 2.4.2 Một số lớp thao tác với NSDate



- q Lớp NSDateComponents, NSCalendar.

- Sử dụng **NSCalendar**, **NSDateComponents**, để lấy ngày tháng năm trong NSDate

```
NSDate *now = [NSDate date];
NSCalendar *calendar = [[NSCalendar alloc]
initWithCalendarIdentifier:NSGregorianCalendar];
NSCalendarUnit units = NSYearCalendarUnit | NSMonthCalendarUnit | NSDayCalendarUnit;
NSDateComponents *components = [calendar
components:units fromDate:now];
NSLog(@"Day: %ld", (long)[components day]);
NSLog(@"Month: %ld", (long)[components month]);
NSLog(@"Year: %ld", (long)[components year]);
// Day: 19 // Month: 03 // Year: 2014
```



## 2.4.2 Một số lớp thao tác với NSDate



- q Lớp NSDateFormatter.

- Sử dụng **NSDateFormatter** để định dạng lại cấu trúc của NSDate sau khi chuyển qua chuỗi.

```
NSDateFormatter *formatter = [[NSDateFormatter alloc]
init];
[formatter setDateFormat:@"M.d.y"];
NSString *afterFormat = [formatter
stringFromDate:now];
NSLog(@"%@", afterFormat);
// 3.19.2014
```





## 2.4.3 So sánh NSDate

- q Sử dụng phương thức compare để so sánh NSDate.

```
NSComparisonResult result = [hienTai compare:tuanSau];
if (result == NSOrderedAscending) {
    NSLog(@"hienTai < tuanSau");
} else if (result == NSOrderedSame) {
    NSLog(@"hienTai == tuanSau");
} else if (result == NSOrderedDescending) {
    NSLog(@"hienTai > tuanSau");
}
// hienTai < tuanSau
```



## 2.4.4 Một số phương thức khác

- q Dùng phương thức earlierDate để lấy v ề ngày nhỏ hơn giữa hai ngày, và phương thức laterDate để lấy v ề ngày lớn hơn.

```
NSDate *ngayNhoHon = [hienTai earlierDate:tuanSau];
NSDate *ngayLonHon = [hienTai laterDate:tuanSau];
NSLog(@"%@", @" nhở hơn %@", ngayNhoHon, ngayLonHon);

// 2014-03-19 03:39:01 +0000 nhở hơn 2014-03-26 03:39:01 +0000
```



## Thảo luận





Trường ĐH Khoa Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 3. Vấn đề cấp phát và thu hồi bộ nhớ

Ngành Mạng & Thiết bị di động

2014



5014



## Nội dung



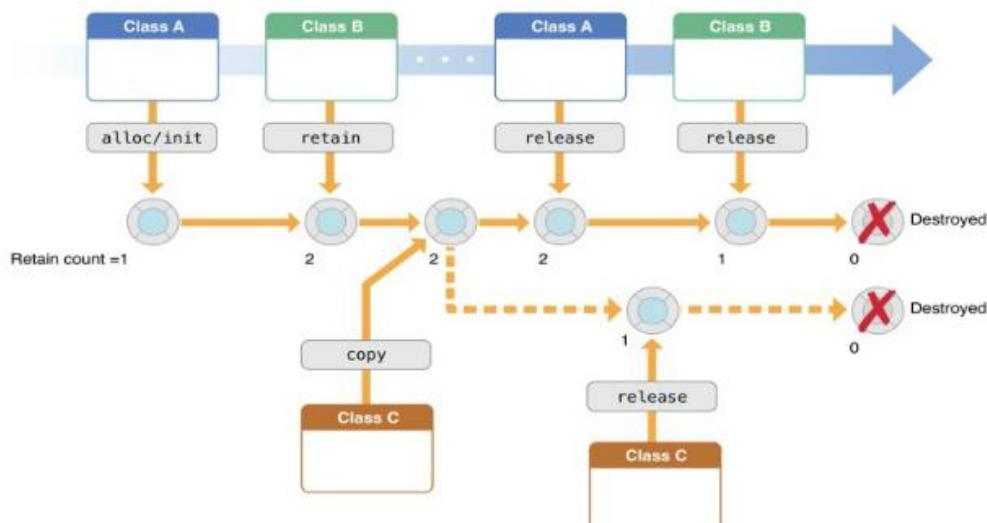
1. Tổng quan quản lý bộ nhớ
2. Các quy tắc trong quản lý bộ nhớ
3. Cách quản lý bộ nhớ hiệu quả
4. Sử dụng AutoRelease Pool Block



## 1 Tổng quan quản lý bộ nhớ



- q Quản lý bộ nhớ là quá trình cấp phát bộ nhớ khi sử dụng chương trình và giải phóng bộ nhớ khi không dùng nữa.



# 1 Tổng quan quản lý bộ nhớ



- q Trong Objective-C, có hai cách để quản lý bộ nhớ của ứng dụng:
  - Manual Retain-Release (MRR), ta quản lý bằng cách theo dõi các đối tượng đã được tạo ra.
  - Automatic Reference Counting (ARC), hệ thống sử dụng cùng “reference counting” như MRR, nhưng nó có kèm theo các phương thức quản lý bộ nhớ vào lúc biên dịch.
- q Có hai vấn đề chính trong việc quản lý bộ nhớ không đúng:
  - Giải phóng hoặc ghi đè dữ liệu trong khi đang sử dụng. Việc này làm cho ứng dụng bị lỗi, hoặc tệ hơn là mất dữ liệu.
  - Không giải phóng dữ liệu khi không cần thiết, gây ra việc rò rỉ bộ nhớ. Việc rò rỉ làm cho bộ nhớ của ứng dụng ngày càng tăng, làm cho hiệu suất hệ thống giảm hoặc ứng dụng bị lỗi.



## Nội dung



1. Tổng quan quản lý bộ nhớ
2. Các quy tắc trong quản lý bộ nhớ
3. Cách quản lý bộ nhớ hiệu quả
4. Sử dụng AutoRelease Pool Block





## 2 Các qui tắc trong quản lý bộ nhớ

q **Mô hình quản lý bộ nhớ dựa trên quy tắc sở hữu đối tượng.** Một đối tượng bất kỳ có thể có một hoặc nhiều chủ sở hữu. Nếu đối tượng không có chủ sở hữu, khi chạy hệ thống sẽ tự động hủy đối tượng đó. Cocoa có đặt vài quy tắc sau:

- Ta sở hữu bất kỳ đối tượng nào do ta tạo ra. .
- Lấy quy tắc sở hữu của đối tượng bằng cách dùng **retain** .
- Khi không cần sử dụng nữa, ta phải từ bỏ quy tắc sở hữu đối tượng.
- Ta không được phép từ bỏ quy tắc sở hữu đối tượng khi không làm chủ.
- Ta không sở hữu đối tượng được trả về bằng tham chiếu.



## 2 Các qui tắc trong quản lý bộ nhớ

q **Ví dụ:**

Ta có lớp Person như sau:

```
@interface Person : NSObject  
  
@property (retain) NSString *firstName;  
  
@property (retain) NSString *lastName;  
  
@property (assign, readonly) NSString *fullName;  
  
@end
```

Ta sẽ sử dụng lớp Person như sau:

```
Person *aPerson = [[Person alloc] init];  
// ...  
  
NSString *name = aPerson.fullName;  
[aPerson release];  
  
// ta tạo aPerson bằng phương thức alloc nên ta phải release khi  
// không dùng nữa.
```



## 2 Các qui tắc trong quản lý bộ nhớ



q Ví dụ sử dụng autorelease:

```
- (NSString *)fullName {
    NSString *string = [[[NSString alloc] initWithFormat:@"%@%@",
   (@"%@", self.firstName, self.lastName] autorelease];
    return string;
} // ta gọi autorelease thì bộ nhớ sẽ bị huỷ sau khi giá trị của hàm được trả về.

- (NSString *)fullName {
    NSString *string = [NSString stringWithFormat:@"%@ %@", self.firstName,
    self.lastName];
    return string;
} // ta không gọi bất kỳ phương thức lấy quyền sở hữu nào cho biến string nên ta không cần release
```

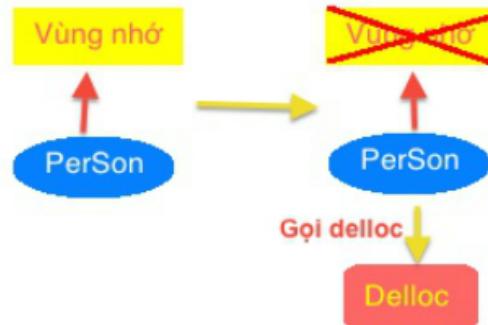


## 2 Các qui tắc trong quản lý bộ nhớ



q Lớp NSObject định nghĩa phương thức dealloc, được tự động gọi khi một đối tượng không có chủ sở hữu và bộ nhớ của nó thì bị thu hồi. Nhiệm vụ của phương thức dealloc là làm trống bộ nhớ của đối tượng và hủy bỏ tất cả tài nguyên mà nó đang giữ :

```
- (void)dealloc
{
    [_firstName release];
    [_lastName release];
    [super dealloc];
}
```



// phương thức dealloc sẽ hủy bỏ tất cả tài nguyên của lớp Person.



## Nội dung



1. Tổng quan quản lý bộ nhớ
2. Các quy tắc trong quản lý bộ nhớ
3. **Cách quản lý bộ nhớ hiệu quả**
4. Sử dụng AutoRelease Pool Block



## 3 Cách quản lý bộ nhớ hiệu quả



- q **Sử dụng các phương thức truy xuất làm cho việc quản lý bộ nhớ dễ dàng hơn.**
- q **Sử dụng các phương thức truy xuất để cài đặt giá trị cho thuộc tính.**
- q **Không sử dụng các phương thức truy xuất trong phương thức khởi tạo và dealloc.**
- q **Sử dụng tham chiếu yếu để tránh vòng chỉ m giữ.**





### 3 Cách quản lý bộ nhớ hiệu quả

- q Sử dụng các phương thức truy xuất làm cho việc quản lý bộ nhớ dễ dàng hơn.

```
- (NSString *)firstName{  
    return _firstName;  
}  
  
- (void)setFirstName:(NSString *)newFirstName{  
    [newFirstName retain];  
    [_firstName release];  
    // Make the new assignment.  
    _firstName= newFirstName ;  
}
```



### 3 Cách quản lý bộ nhớ hiệu quả

- q Sử dụng các phương thức truy xuất để cài đặt giá trị cho thuộc tính.

- Tạo một NSNumber với phương thức alloc, và sẽ release nó.

```
- (void)reset {  
    NSNumber *zero = [[NSNumber alloc] initWithInteger:0];  
    [self setCount:zero];  
    [zero release];  
}
```

- Sử dụng hàm khởi tạo để tạo đối tượng NSNumber mới. Ta sẽ không cần retain và release đối tượng.

```
- (void)reset {  
    NSNumber *zero = [NSNumber numberWithInt:0];  
    [self setCount:zero];  
}
```



### 3 Cách quản lý bộ nhớ hiệu quả



q Không sử dụng các phương thức truy xuất trong phương thức khởi tạo và dealloc.

- Trong phương thức khởi tạo sẽ có một số thuộc tính được tạo sau hàm khởi tạo nếu ta truy xuất có thể dẫn đến lỗi và không tạo được đối tượng.
- Dealloc là phương thức dùng để dọn dẹp vùng nhớ của đối tượng nếu ta dùng các phương thức truy xuất trong hàm này nó có thể phát sinh vùng nhớ khi đối tượng bị huỷ và đây là mối nguy hiểm không nên có.

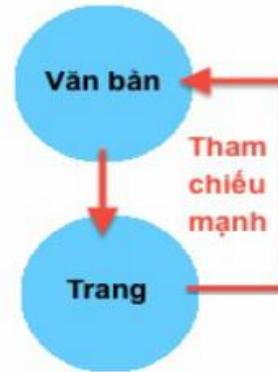


### 3 Cách quản lý bộ nhớ hiệu quả



q Sử dụng tham chiếu yếu để tránh vòng참 giữ.

- Chiếm giữ một đối tượng tạo nên một tham chiếu mạnh. Một đối tượng không thể hủy cho tới khi tất cả tham chiếu mạnh đều được giải phóng.
- **Retain Cycle** (vòng chiế m giữ) xảy ra nếu có hai đối tượng tham chiếu mạnh tới nhau

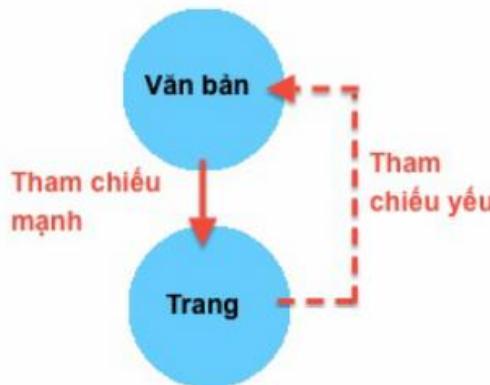


### 3 Cách quản lý bộ nhớ hiệu quả



#### q Sử dụng tham chiếu yếu để tránh vòng참 m giữ.

- Để giải quyết vấn đề cho vòng참 m giữ, ta dùng tham chiếu yếu.  
Tham chiếu yếu sẽ không sở hữu đối tượng.
- Để giữ sơ đồ đối tượng, ta cần phải có tham chiếu mạnh (nếu chỉ có tham chiếu yếu, các đối tượng sẽ không có chủ sở hữu sẽ bị hủy).



## Nội dung



- Tổng quan quản lý bộ nhớ
- Các quy tắc trong quản lý bộ nhớ
- Cách quản lý bộ nhớ hiệu quả
- Sử dụng AutoRelease Pool Block



## 4 Sử dụng AutoRelease Pool Block



- q Autorelease pool block cung cấp cơ chế mà ở đó ta có thể từ bỏ quyền sở hữu đối tượng và tránh được việc nó bị hủy ngay lập tức. Thông thường ta không cần phải tạo Autorelease pool block, nhưng trong một số tình huống ta phải tạo hoặc do nó có lợi.
- q Một autorelease pool block được đánh dấu bởi **@autoreleasepool**, ví dụ:

```
@autoreleasepool {  
    // Lệnh để tạo các đối tượng autorelease.  
}
```



## 4 Sử dụng AutoRelease Pool Block



- q Cuối autorelease pool block, các đối tượng đã nhận một thông điệp autorelease đều được gửi thông điệp release.
- q Autorelease pool block có thể lồng vào nhau.

```
@autoreleasepool {  
    // ...  
@autoreleasepool {  
    // ...  
}  
// ...  
}
```



## 4 Sử dụng AutoRelease Pool Block



- q Cocoa luôn muốn lệnh được thực thi bên trong một autorelease pool block, nếu không các đối tượng được autorelease sẽ không được giải phóng và làm cho ứng dụng bị rò rỉ bộ nhớ.
- q Framework AppKit và UIKit xử lý mỗi sự kiện bên trong một autorelease pool block, do đó thông thường ta không cần phải tạo autorelease pool block hay thấy được lệnh sử dụng nó.



## 4 Sử dụng AutoRelease Pool Block



- q Có ba trường hợp ta có thể sử dụng autorelease pool block:
  - Khi đang viết một chương trình không dựa trên framework UI như công cụ command-line.
  - Viết một vòng lặp để tạo nhiều đối tượng tạm thời.
  - Nếu ta tạo ra một tiến trình thứ hai



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 4. Mô hình MVC

Ngành Mạng & Thiết bị di động



2014

2014



## Nội dung



1. Giới thiệu mô hình MVC
2. Cấu trúc mô hình MVC
3. Cách hoạt động của mô hình MVC



## 1 Giới thiệu mô hình MVC



- ❑ *Model-View-Controller (MVC)* là một **mẫu kiến trúc phần mềm** trong kỹ thuật phần mềm.
- ❑ Khi sử dụng cách, **mẫu MVC** giúp cho người phát triển phần mềm cô lập các nguyên tắc nghiệp vụ và giao diện người dùng một cách rõ ràng hơn.
- ❑ Phần mềm phát triển theo **mẫu MVC** tạo nhiều thuận lợi cho việc bảo trì và nâng cấp vì giao diện ít liên quan với nhau.



## Nội dung



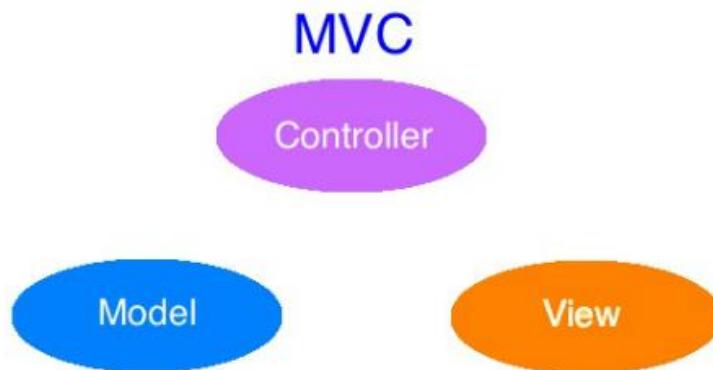
1. Giới thiệu mô hình MVC
2. Cấu trúc mô hình MVC
3. Cách hoạt động của mô hình MVC



## 2 Cấu trúc mô hình MVC



- ❑ **Mô hình MVC gồm có 3 thành phần chính:**
  - Model: phần dữ liệu có trong chương trình.
  - View: phần giao diện của người dùng.
  - Controller: quản lý sự trao đổi dữ liệu, trình bày giao diện cho người dùng.



## Nội dung



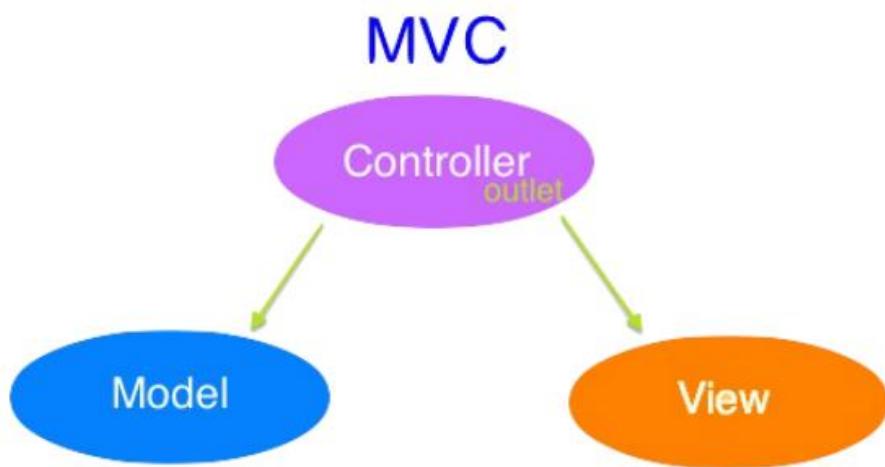
1. Giới thiệu mô hình MVC
2. Cấu trúc mô hình MVC
3. Cách hoạt động của mô hình MVC



## 2 Cách hoạt động của mô hình MVC



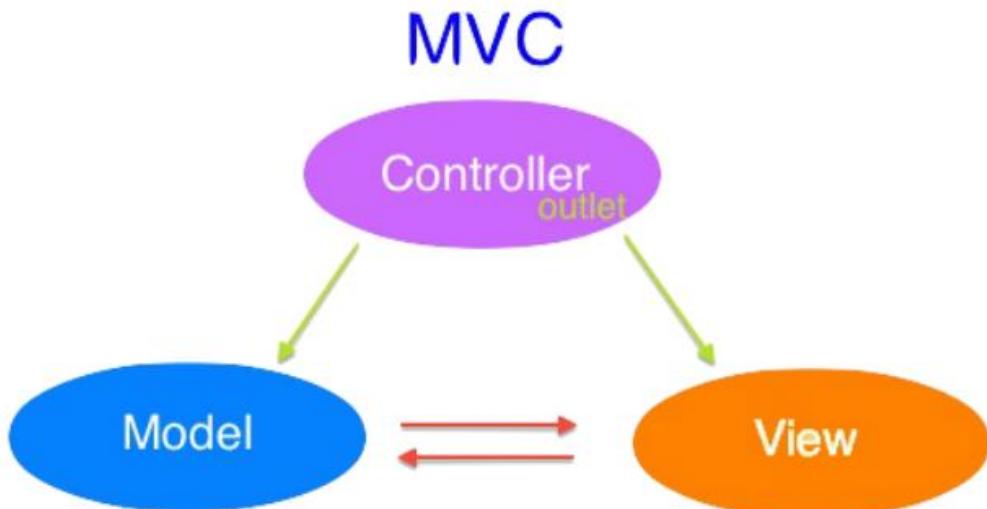
- ❑ Liên kết giữa các phần trong MVC:
  - Controller thì có thể nói chuyện với Model của nó.
  - Controller cũng có thể nói chuyện với View của nó.



## 2 Cách hoạt động của mô hình MVC



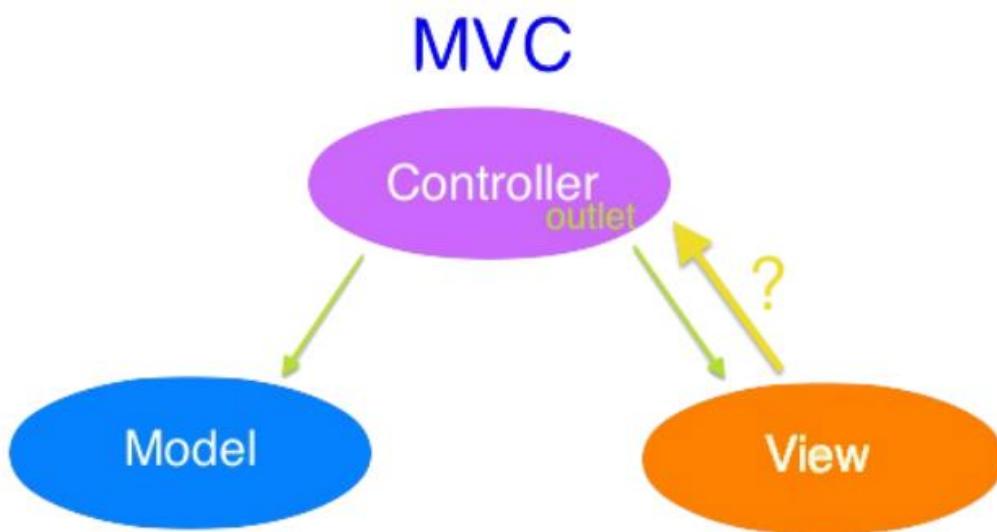
- ❑ Liên kết giữa các phần trong MVC:
  - Model và View thì không thể nói chuyện với nhau.



## 2 Cách hoạt động của mô hình MVC

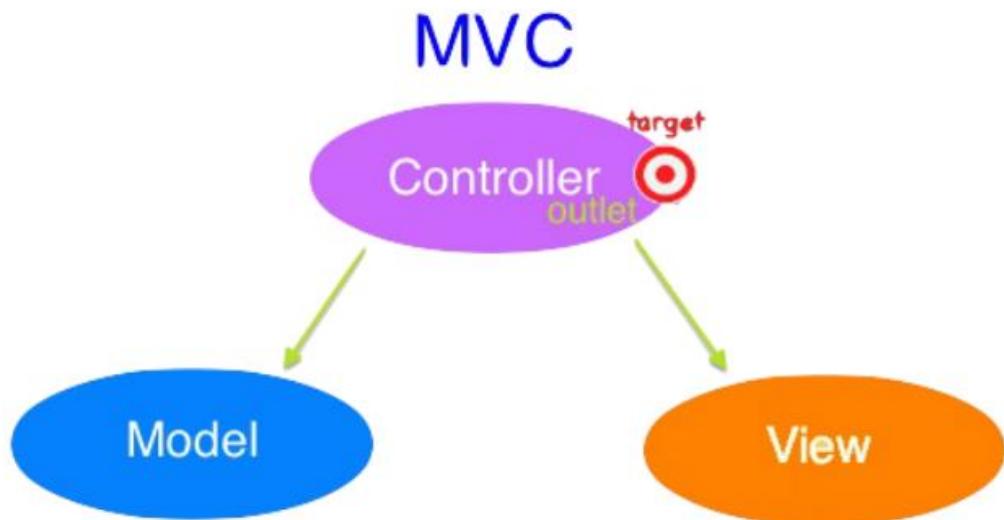


- ❑ Vậy thì làm sao để View nói chuyện với Controller?



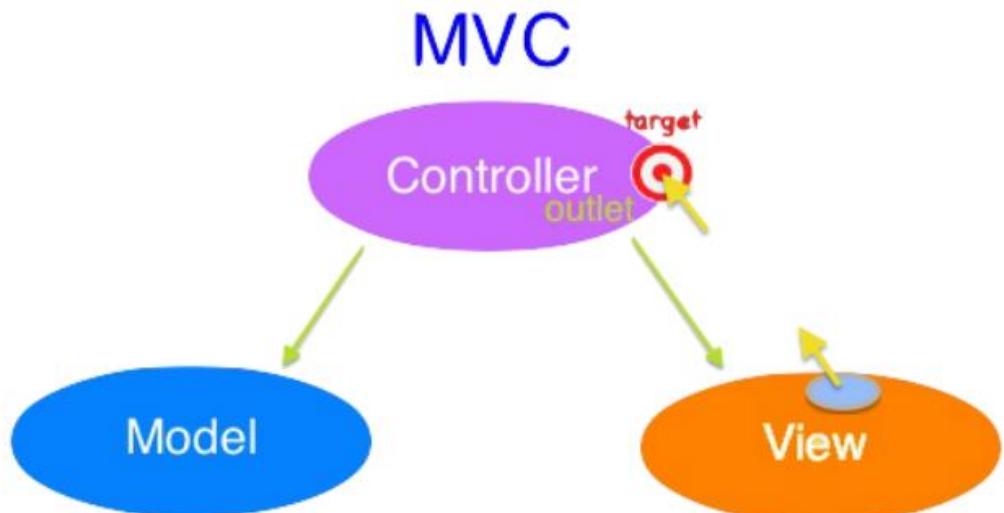
## 2 Cách hoạt động của mô hình MVC

- Controller sẽ thả một Taget(mục tiêu) trên chính nó để nhận lại Action(hành động) xảy ra trên View.



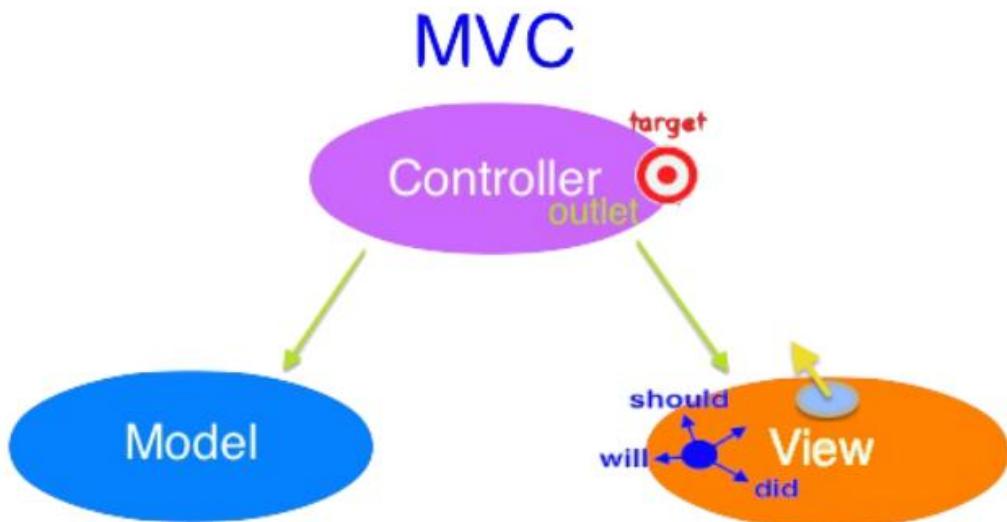
## 2 Cách hoạt động của mô hình MVC

- Sau đó:
  - Khi Action xảy ra trên View.
  - View sẽ gửi Action xảy ra trong giao diện người dùng cho Taget.



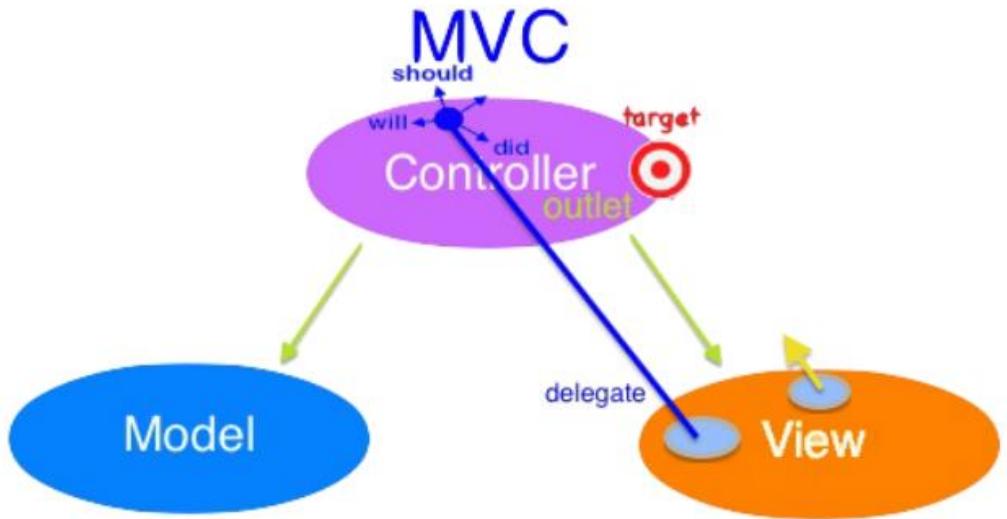
## 2 Cách hoạt động của mô hình MVC

- ❑ Một số trường hợp View cần động bộ hoá với controller.



## 2 Cách hoạt động của mô hình MVC

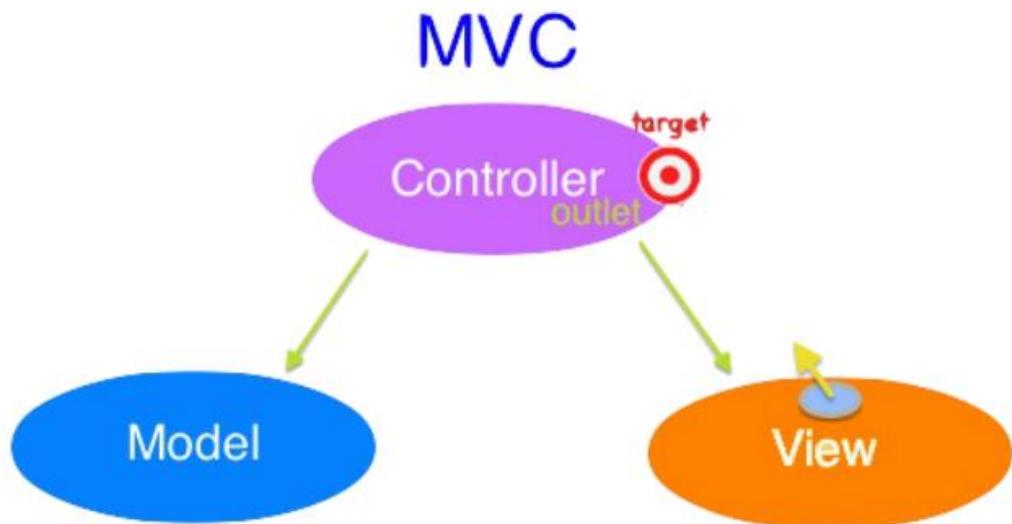
- ❑ Controller sẽ thiết lập chính nó làm đại diện(delegate) cho View.  
Delegate sẽ được thiết lập thông qua Protocol(giao thức).





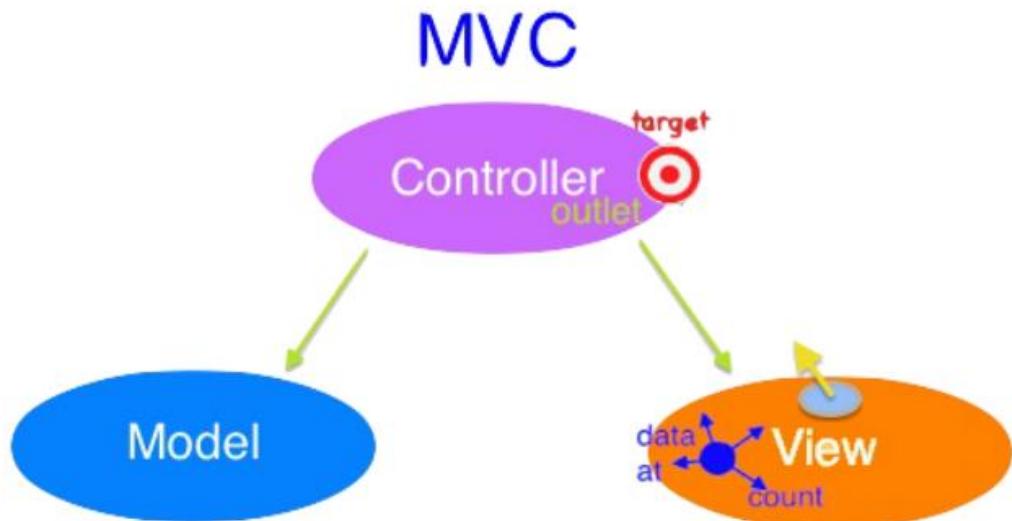
## 2 Cách hoạt động của mô hình MVC

- View không sở hữu dữ liệu mà chúng hiển thị.



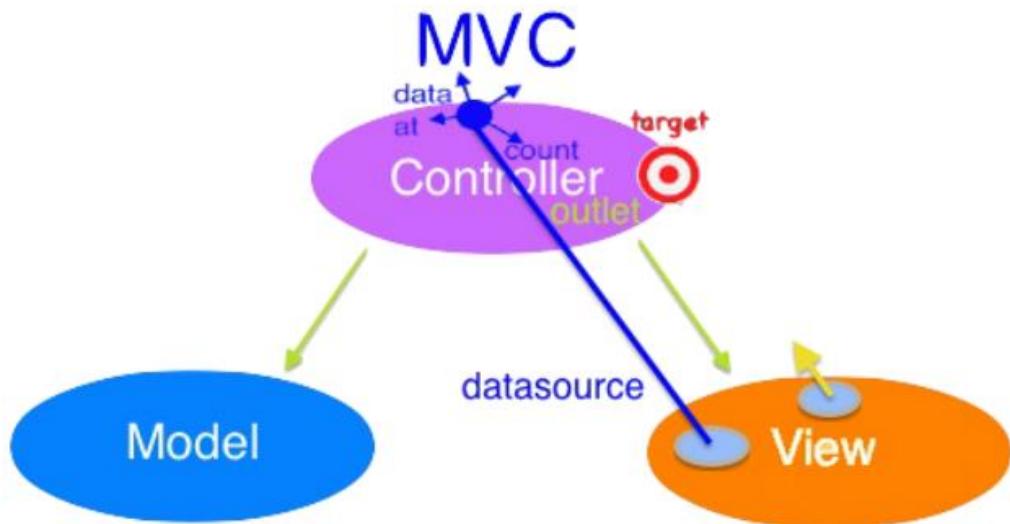
## 2 Cách hoạt động của mô hình MVC

- Vì thế nếu cần thiết view có một giao thức để có được dữ liệu.



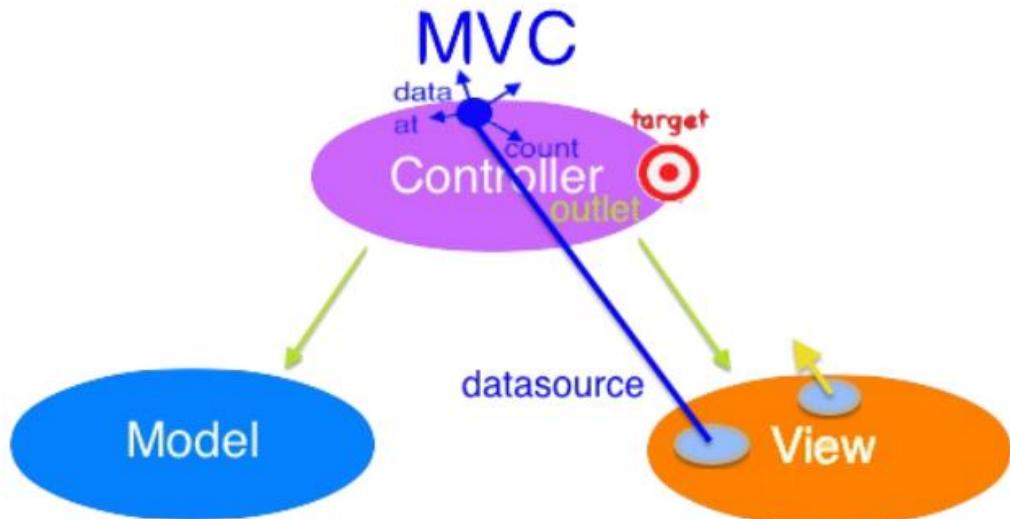
## 2 Cách hoạt động của mô hình MVC

- Controller gần như, luôn luôn là nguồn dữ liệu(không phải Model).



## 2 Cách hoạt động của mô hình MVC

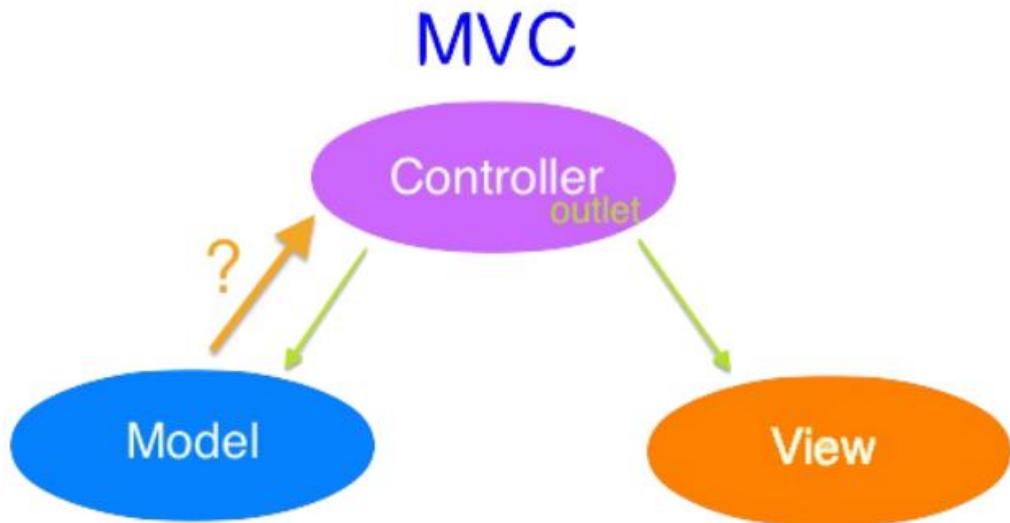
- Controller sẽ đọc và định dạng lại dữ liệu từ Model cho View.



## 2 Cách hoạt động của mô hình MVC



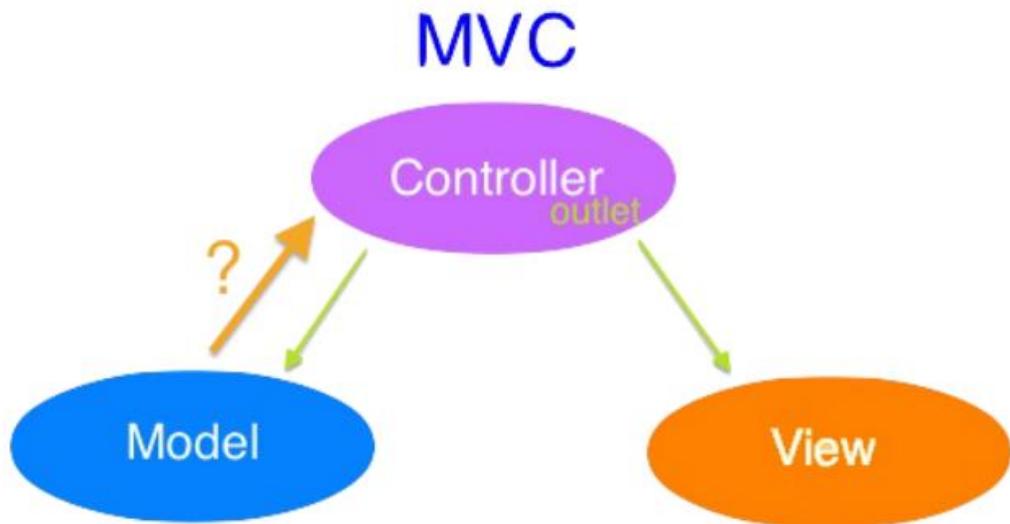
- Model có thể nói chuyện trực tiếp với Controller?



## 2 Cách hoạt động của mô hình MVC



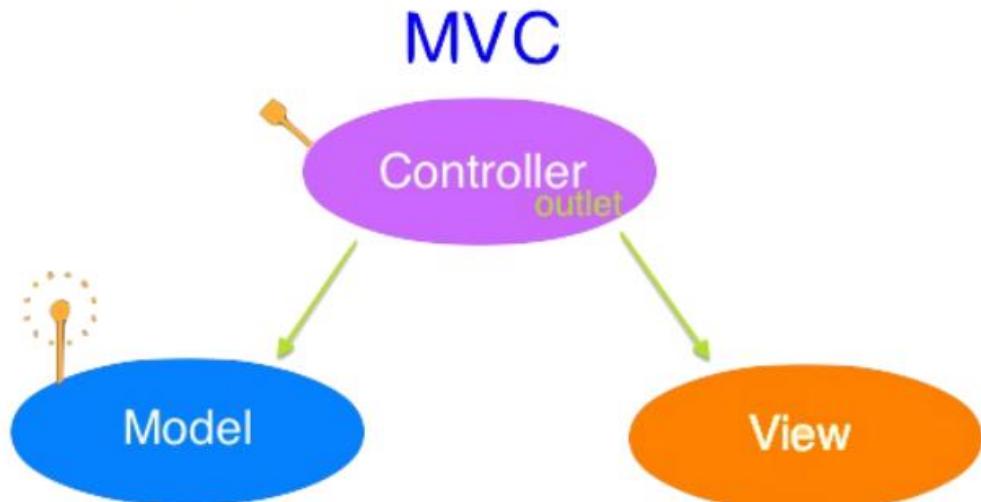
- Không. Model nên là giao diện người dùng độc lập.
- Vậy nếu có thông tin cần cập nhật thì phải làm như thế nào?



## 2 Cách hoạt động của mô hình MVC



- Model sử dụng một “radio station”(đài phát thanh) giống như cơ chế phát sóng.
- Controller hoặc những Model khác sẽ điều chỉnh ở bên trong và View sẽ được điều chỉnh lại.



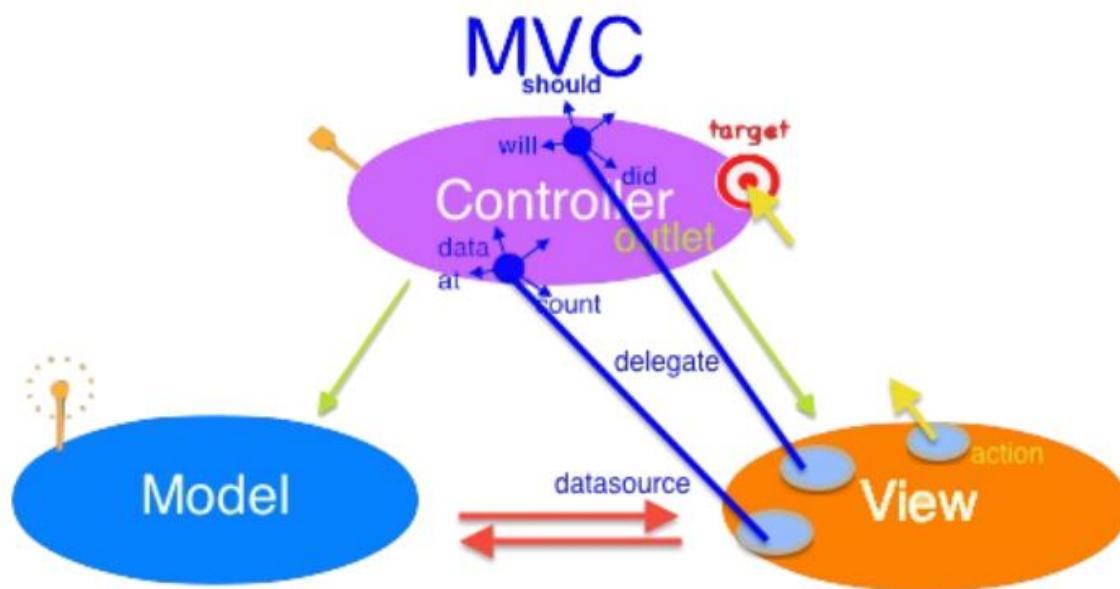
Lập trình iOS (2014) – Bài 4. Mô hình MVC

20

## 2 Cách hoạt động của mô hình MVC



- Bây giờ ta kết hợp MVC lại và có mô hình như sau.



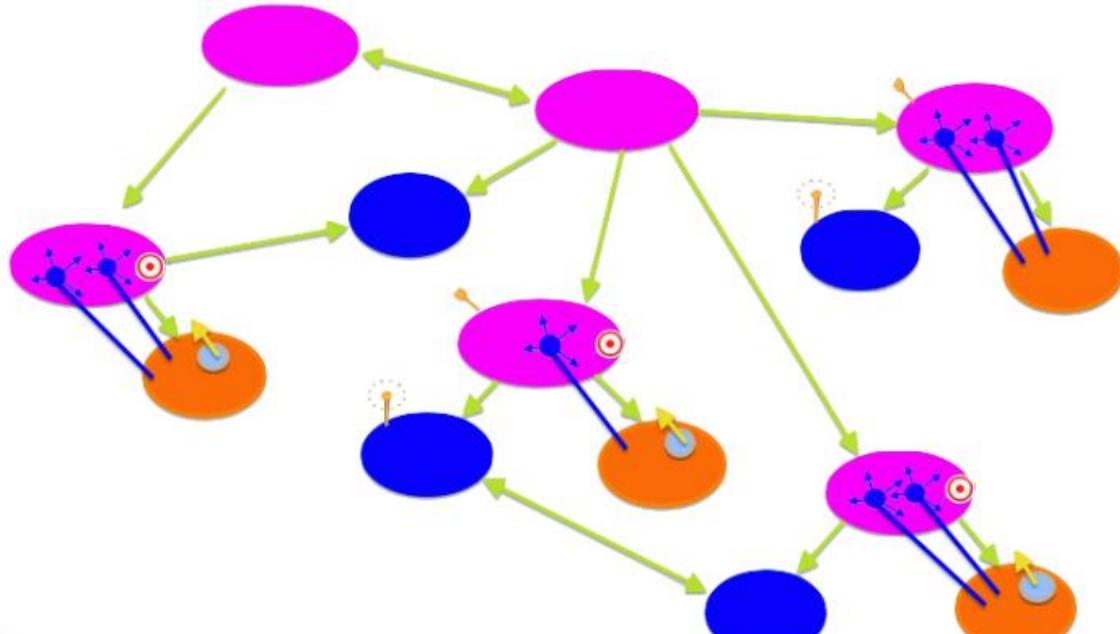
Lập trình iOS (2014) – Bài 4. Mô hình MVC

21

## 2 Cách hoạt động của mô hình MVC



### ❑ Mô hình MVC hoạt động cùng nhau



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 5. Các điều khiển lựa chọn và điều hướng

Ngành Mạng & Thiết bị di động



2014

5014



## Nội dung



### 1. PickerView

- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp UIPickerView
- Xây dựng PickerView
- Các tác vụ trong PickerView

### 2. ActionSheet

### 3. Tab Bar

### 4. NavigationBar



## 1.1 Mục đích sử dụng trong ứng dụng



- ❑ Lớp PickerView sử dụng một vòng quay bánh xe, hoặc một máy tự động để hiển thị một hoặc nhiều bộ giá trị. Người dùng chọn giá trị bằng cách xoay bánh xe để giá trị cần chọn trùng với selection indicator.
- ❑ Dùng để hiển thị nội dung của một đối tượng ở mức độ chi tiết.

Galaxy	3	7h30
Megastart	4	9h
Lotte	5	10h30
<b>Hùng Vương</b>	<b>6</b>	<b>11h</b>
Trống Đồng	7	15h
Nguyễn Trãi	8	17h
Nguyễn Du	9	19h

IOS7



IOS6





## 1.2 Khảo sát lớp UIPickerView

### ❑ Tuỳ chỉnh kích thước của PickerView.

- [numberOfComponents property](#)
- [numberOfRowsInComponent:](#)
- [rowSizeForComponent:](#)

### ❑ Gọi hàm reload lại dữ liệu

- [reloadAllComponents](#)
- [reloadComponent:](#)



## 1.2 Khảo sát lớp UIPickerView

### ❑ Tuỳ chỉnh chọn dòng trên PickerView

- [selectRow:inComponent:animated:](#)
- [selectedRowInComponent](#)

### ❑ Trả lại một cột, dòng trên PickerView

- [viewForRow:forComponent](#)



## 1.3 Xây dựng PickerView



### ❑ Xây dựng PickerView từ Interface Builder.

- Thực hiện kéo đối tượng UIPickerView từ Object Library vào xib, sau đó thực hiện tạo outlet cho PickerView đến File's Owner.

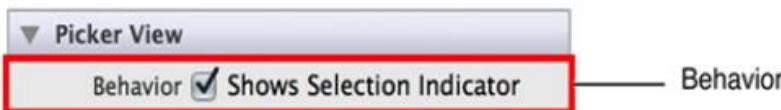
Lập trình iOS (2014) – Bài 5. Các điều khiển lựa chọn và điều hướng 6

## 1.3 Xây dựng PickerView



### ❑ Thuộc tính cần lưu ý của PickerView là Selection Indicator:

- Selection Indicator xác định xem dòng nào đang được chọn.



### ❑ iOS7 mặc định hiển thị dù ta có thiết lập thuộc tính như thế nào.

Galaxy	3	7h30
Megastart	4	9h
Lotte	5	10h30
<b>Hùng Vương</b>	<b>6</b>	<b>11h</b>
Trống Đồng	7	15h
Nguyễn Trãi	8	17h
Nguyễn Du	9	19h



## 1.3 Xây dựng PickerView



- Để sử dụng PickerView ta phải tạo dữ liệu và sử dụng một số phương thức của PickerView.

- Khai báo PickerView sử dụng **DataSource** và **Delegate**.

```
@interface ViewController : UIViewController  
<UIPickerViewDataSource,  
UIPickerViewDelegate>
```

```
self.pickerView.delegate = self;  
self.pickerView.dataSource = self;
```



## 1.3 Xây dựng PickerView



- Sử dụng protocol của PickerView để thiết lập dữ liệu cho UIPickerView

- Khai báo số dòng trong một cột của PickerView.

```
- (NSInteger)pickerView:(UIPickerView *)pickerView  
numberOfRowsInComponent:(NSInteger)component {  
    return 8;  
}
```

- Khai báo số cột của PickerView.

```
- (NSInteger)numberOfComponentsInPickerView:(  
UIPickerView *)pickerView{  
    return 4;  
}
```





## 1.4 Các tác vụ trong PickerView

### ❑ Tuỳ chỉnh kích thước PickerView

- [pickerView:rowHeightForComponent:](#)
- [pickerView:widthForComponent:](#)

### ❑ Trả về dòng đang được chọn

- [pickerView:didSelectRow:inComponent:](#)



## 1.4 Các tác vụ trong PickerView

### ❑ Tuỳ chỉnh nội dung dòng của cột trên PickerView

- [pickerView:titleForRow:forComponent:](#)
- [pickerView:attributedTitleForRow:forComponent:](#)
- [pickerView:viewForRow:forComponent:reusingView:](#)



## Nội dung



1. UIPickerView
2. ActionSheet
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UIActionSheet
  - Xây dựng ActionSheet
  - Các tác vụ trong ActionSheet
3. Tab Bar
4. NavigationBar



## 2.1 Mục đích sử dụng trong ứng dụng



- Lớp Action Sheet để trình bày cho người dùng một danh sách các lựa chọn. Action Sheet có tiêu đề và nhiều nút, mỗi nút có một chức năng khác nhau.





## 2.2 Khảo sát lớp UIActionSheet

### ❑ Tạo một ActionSheet.

- [initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:](#)

### ❑ Tuỳ chỉnh ActionSheet.

- [delegate property](#)
- [title property](#)
- [visible property](#)
- [actionSheetStyle property](#)



## 2.2 Khảo sát lớp UIActionSheet

### ❑ Tuỳ chỉnh cấu hình nút của ActionSheet

- [addButtonWithTitle:](#)
- [numberOfButtons property](#)
- [buttonTitleAtIndex:](#)
- [cancelButtonIndex property](#)
- [destructiveButtonIndex property](#)
- [firstOtherButtonIndex property](#)





## 2.2 Khảo sát lớp UIActionSheet

### ❑ Tuỳ chỉnh hiển thị ActionSheet

- [showFromTabBar:](#)
- [showFromToolbar:](#)
- [showInView:](#)
- [showFromBarButtonItem:animated:](#)
- [showFromRect:inView:animated:](#)

### ❑ Tuỳ chỉnh loại bỏ ActionSheet

- [dismissWithClickedButtonIndex:animated:](#)



## 2.3 Xây dựng ActionSheet

### ❑ Sử dụng phương thức để khởi tạo một ActionSheet.

- Tạo một ActionSheet với tiêu đề, delegate, nút cancel, nút destructive, các nút khác.

```
UIActionSheet *actionSheet =
[[UIActionSheet alloc]
initWithTitle:@"ActionSheet" delegate:self
cancelButtonTitle:@"Cancel"
destructiveButtonTitle:@"destructiveButton"
"
otherButtonTitles:@"otherButton",@"otherButton1",@"otherButton2", nil];
```





## 2.3 Xây dựng ActionSheet

- ☐ Số thứ tự của các nút trong ActionSheet được đánh theo chiều từ trên xuống và bắt đầu từ 0.



## 2.4 Các tác vụ trong ActionSheet

- ☐ Nhận sự kiện chọn nút trên ActionSheet.
  - [actionSheet:clickedButtonAtIndex:](#)
- ☐ Tuỳ chỉnh hành vi trên ActionSheet.
  - [willPresentActionSheet:](#)
  - [didPresentActionSheet:](#)
  - [actionSheet:willDismissWithButtonIndex:](#)
  - [actionSheet:didDismissWithButtonIndex:](#)





## 2.4 Các tác vụ trong ActionSheet

### ❑ Tuỳ chỉnh khi tắt ActionSheet.

- [actionSheetCancel:](#)



## Nội dung

### 1. UIPickerView

### 2. ActionSheet

### 3. Tabbar

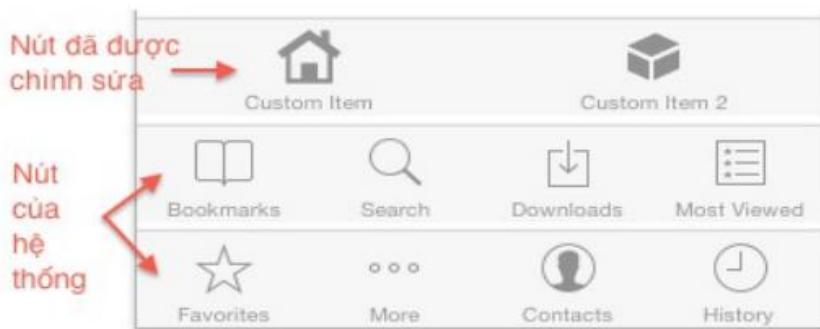
- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp UITabbar
- Xây dựng Tabbar
- Các tác vụ trong Tabbar

### 4. NavigationBar



### 3.1 Mục đích sử dụng Tabbar

- ❑ Tabbar là một điều khiển, thường nằm dưới màn hình trong một tab bar controller, người dùng có thể thay đổi lựa chọn bằng cách chạm vào đối tượng trên tabbar.
- ❑ Mỗi một nút trong tab bar được gọi là tabbar item và là một thẻ hiện của lớp UITabBarItem.



### 3.2 Khảo sát lớp UITabbar

- ❑ **Tùy chỉnh cấu hình Tabbar Item.**
  - [initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:](#)
- ❑ **Hỗ trợ tùy chỉnh Tabbar**
  - [beginCustomizingItems:](#)
  - [endCustomizingAnimated:](#)
  - [isCustomizing](#)



## 3.2 Khảo sát lớp UITabbar



### ❑ Tuỳ chỉnh giao diện bên ngoài của Tabbar

- [barTintColor property](#)
- [itemPositioning property](#)
- [itemSpacing property](#)
- [itemWidth property](#)
- [tintColor property](#)
- [selectedImageTintColor property](#)
- [translucent property](#)
- [backgroundImage property](#)
- [shadowImage property](#)
- [selectionIndicatorImage property](#)

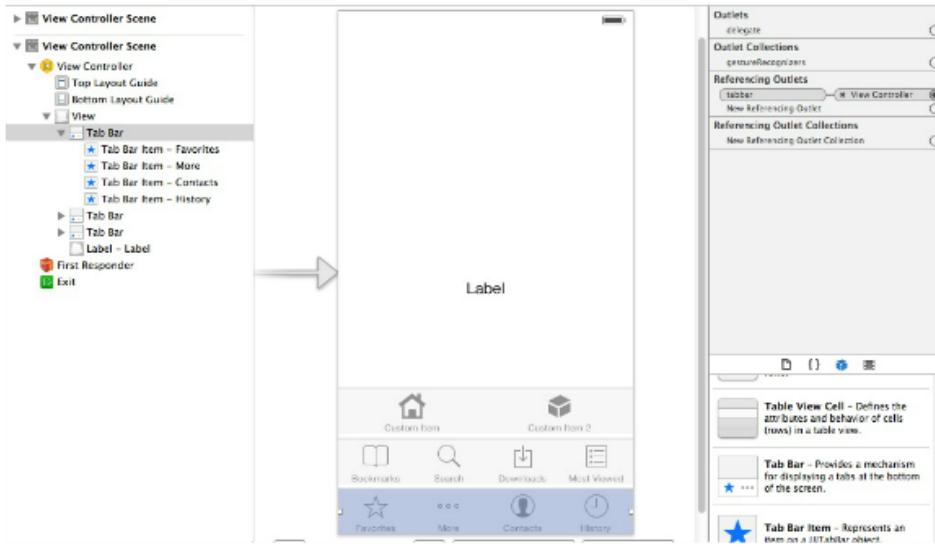


## 3.3 Xây dựng Tabbar



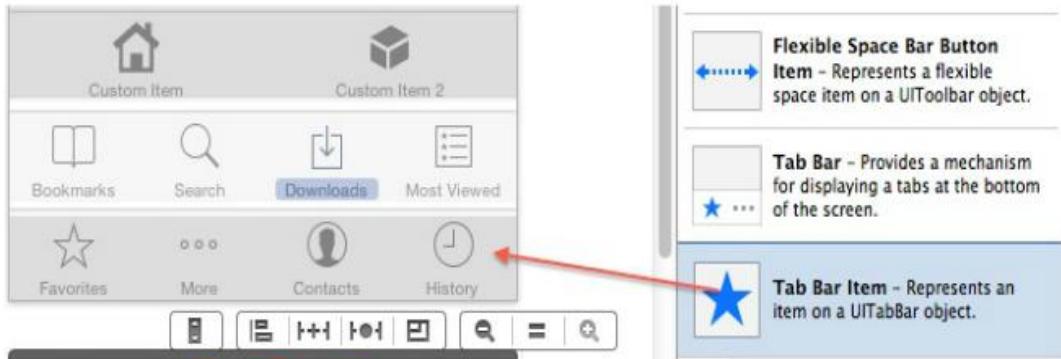
### ❑ Xây dựng Tabbar từ Interface Builder.

- Thực hiện kéo đổi tượng UITabbar từ Object Library vào xib, sau đó thực hiện tạo outlet cho UITabbar đến File's Owner.



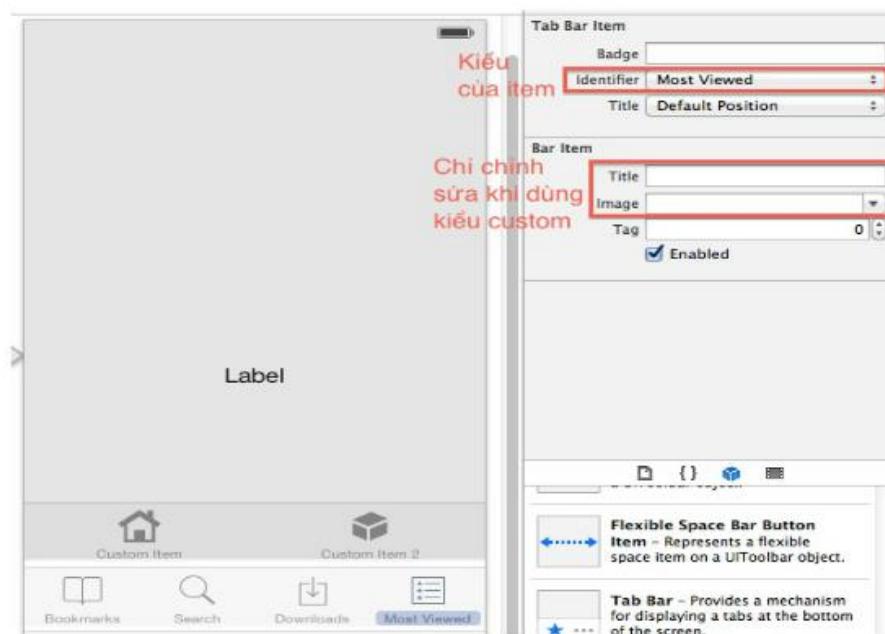
### 3.3 Xây dựng Tabbar

- Để thêm item cho tabbar ta kéo TabBarItem vào trong tabbar.



### 3.3 Xây dựng Tabbar

- Chỉnh sửa item trong tabbar.





## 3.4 Các tác vụ trong Tabbar

### ❑ Tuỳ chỉnh Tabbar.

- [tabBar:willBeginCustomizingItems:](#)
- [tabBar:didBeginCustomizingItems:](#)
- [tabBar:willEndCustomizingItems:changed:](#)
- [tabBar:didEndCustomizingItems:changed:](#)
- [tabBar:didSelectItem:](#) *required method*



## Nội dung

1. UIPickerView
2. ActionSheet
3. Tabbar
4. NavigationBar
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UINavigationBar
  - Xây dựng NavigationBar
  - Các tác vụ trong NavigationBar





## 4.1 Mục đích sử dụng UINavigationBar

- ❑ NavigationBar là một điều khiển, thường nằm phía trên màn hình trong một Navigation controller, dùng để hiển thị tiêu đề của view.
- ❑ Trên NavigationBar thường có những nút chuyển view nếu nó được quản lý bởi UINavigationController.



## 4.2 Khảo sát lớp UINavigationBar

- ❑ Thêm, bớt item vào NavigationBar.
  - [pushNavigationItem:animated:](#)
  - [popNavigationItemAnimated:](#)
  - [setItems:animated:](#)
  - [items property](#)
  - [topItem property](#)
  - [backItem property](#)



## 4.2 Khảo sát lớp UINavigationBar



### □ Tuỳ chỉnh giao diện bên ngoài của NavigationBar.

- [pushNavigationItem:animated:](#)
- [backIndicatorImage property](#)
- [backIndicatorTransitionMaskImage property](#)
- [barStyle property](#)
- [barTintColor property](#)
- [shadowImage property](#)
- [tintColor property](#)
- [translucent property](#)
- [backgroundImageForBarMetrics:](#)
- [setBackgroundImage:forBarMetrics:](#)
- [backgroundImageForBarPosition:barMetrics:](#)
- [arMetrics:](#)
- [setBackgroundImage:forBarPosition:barMetrics:](#)
- [titleVerticalPositionAdjustmentForBarMetrics:](#)
- [setTitleVerticalPositionAdjustment:forBarMetrics:](#)
- [titleTextAttributes property](#)

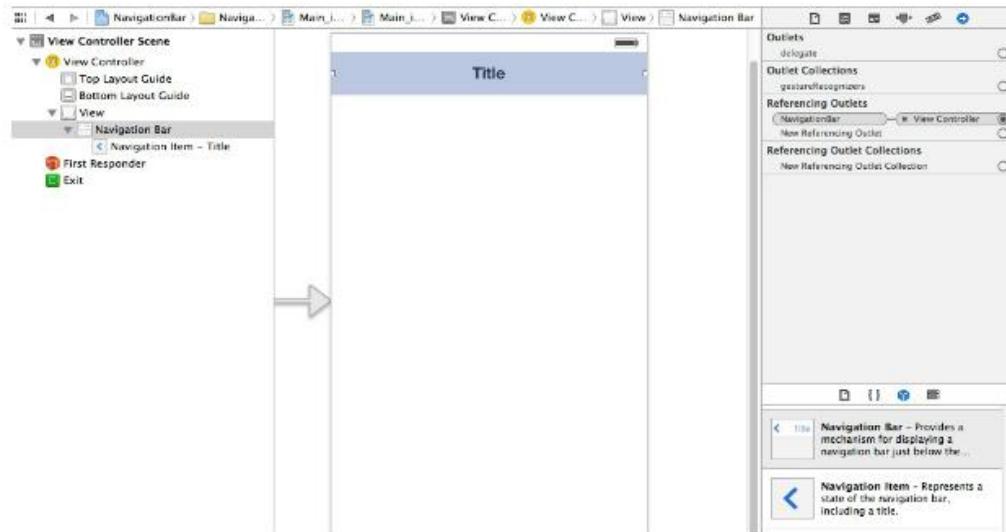


## 4.3 Xây dựng NavigationBar



### □ Xây dựng NavigationBar từ Interface Builder.

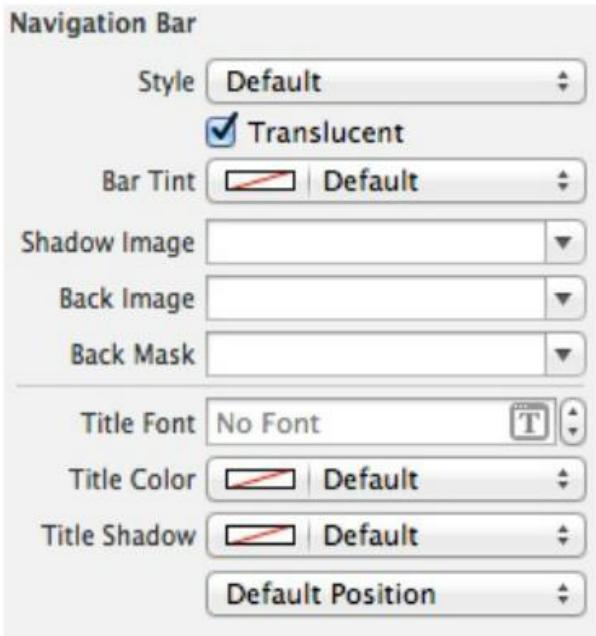
- Thực hiện kéo đối tượng UINavigationBar từ Object Library vào xib, sau đó thực hiện tạo outlet cho UINavigationBar đến File's Owner.





## 4.3 Xây dựng NavigationBar

- ❑ **Chỉnh sửa NavigationBar.**



## 4.4 Các tác vụ trong NavigationBar

- ❑ **Tùy chỉnh Thêm item NavigationBar.**

- [navigationBar:shouldPushItem:](#)
- [navigationBar:didPushItem:](#)





## 4.4 Các tác vụ trong NavigationBar

### ❑ Tuỳ chỉnh bót item NavigationBar.

- [navigationBar:shouldPopItem:](#)
- [navigationBar:didPopItem:](#)



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 6. *Dữ liệu dạng tập hợp nâng cao*

Ngành Mạng & Thiết bị di động



2014

2014





## Nội dung

### 1. NSSet

- NSSet là gì?
- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp NSSet
- Sử dụng NSSet

### 2. NSDictionary

### 3. NSEnumerator

### 4. NSHashTable



## 1.1 NSSet là gì?

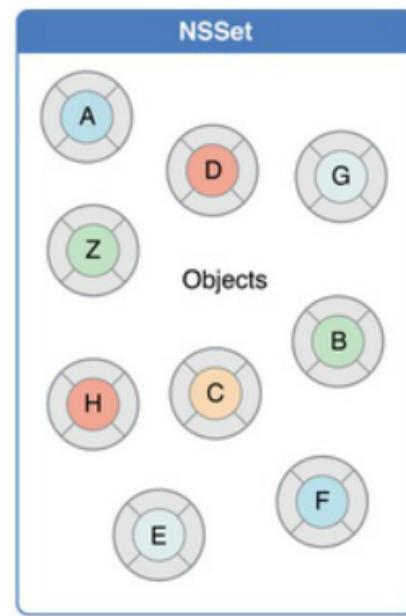
- ❑ **Lớp NSSet và NSMutableSet là một tập hợp của các đối tượng không được sắp xếp.**
- ❑ **NSSet định nghĩa một tập hợp các đối tượng duy nhất và là bất biến đổi.**
- ❑ **NSMutableSet định nghĩa một tập hợp các đối tượng duy nhất và là biến đổi.**



## 1.2 Mục đích sử dụng trong ứng dụng

### □ Ta sử dụng NSSet khi cần tạo một mảng các đối tượng mà:

- Thứ tự không quan trọng
- Các đối tượng là duy nhất
- Có hiệu xuất tốt cho việc tìm kiếm



## 1.3 Khảo sát lớp NSSet

### □ Tạo một đối tượng NSSet

- [set](#)
- [setWithArray:](#)
- [setWithObject:](#)
- [setWithObjects:](#)
- [setWithObjects:count:](#)
- [setWithSet:](#)
- [setByAddingObject:](#)
- [setByAddingObjectsFromSet:](#)
- [setByAddingObjectsFromArray:](#)





## 1.3 Khảo sát lớp NSSet

### ❑ Khởi tạo một đối tượng NSSet với init

- [initWithArray:](#)
- [initWithObjects:](#)
- [initWithObjects:count:](#)
- [initWithSet:](#)
- [initWithSet:copyItems:](#)
- [Init](#)

### ❑ Đếm phần tử trong NSSet

- [count](#)



## 1.3 Khảo sát lớp NSSet

### ❑ Truy xuất phần tử trong NSSet

- [allObjects](#)
- [anyObject](#)
- [containsObject:](#)
- [filteredSetUsingPredicate:](#)
- [makeObjectsPerformSelector:](#)
- [makeObjectsPerformSelector:withObject:](#)
- [member:](#)
- [objectEnumerator](#)
- [enumerateObjectsUsingBlock:](#)
- [enumerateObjectsWithOptions:usingBlock:](#)
- [objectsPassingTest:](#)
- [objectsWithOptions:passingTest:](#)





## 1.3 Khảo sát lớp NSSet

### ❑ So sánh NSSet

- [isSubsetOfSet:](#)
- [intersectsSet:](#)
- [isEqualToString:](#)
- [valueForKey:](#)
- [setValue:forKey:](#)

### ❑ Tạo một mảng được sắp xếp

- [sortedArrayUsingDescriptors:](#)



## 1.3 Khảo sát lớp NSSet

### ❑ Một số phương thức khác

- [sortedArrayUsingDescriptors:](#)
- [addObserver:ForKeyPath:options:context:](#)
- [removeObserver:ForKeyPath:context:](#)
- [removeObserver:ForKeyPath:](#)
- [description](#)
- [descriptionWithLocale:](#)



## 1.4 Sử dụng NSSet



### □ Tạo đối tượng NSSet.

- Sử dụng **initWithObjects** để tạo mảng NSSet.

Tạo đối tượng:

```
NSSet *array = [[NSSet alloc]  
initWithObjects:@"1", @"3",@"4",nil];
```

```
NSMutableSet *mutableArray =  
[[NSMutableSet alloc]  
initWithObjects:@"1", @"3",@"4",nil];
```



## 1.4 Sử dụng NSSet



### □ Truy xuất mảng NSSet.

- Đếm các phần tử trong mảng.
- Kiểm tra sự tồn tại của đối tượng trong mảng với **containsObject**.
- Gán cho kiểu mảng khác.

**array.count**

**mutableArray.count**

```
[array containsObject:@"1"]
```

```
NSArray *a = [[NSArray alloc]  
initWithArray:[array allObjects]];
```





## 1.4 Sử dụng NSSet

### ❑ Thêm/xoá phần tử trong mảng NSMutableSet.

- Thêm phần tử trong mảng.

```
[mutableArray addObject:@"2"];
```

- Xoá phần tử trong mảng.

```
[mutableArray removeObject:@"2"];
```



## Nội dung

### 1. NSSet

### 2. NSDictionary

- NSDictionary là gì?
- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp NSDictionary
- Sử dụng NSDictionary

### 3. NSEnumerator

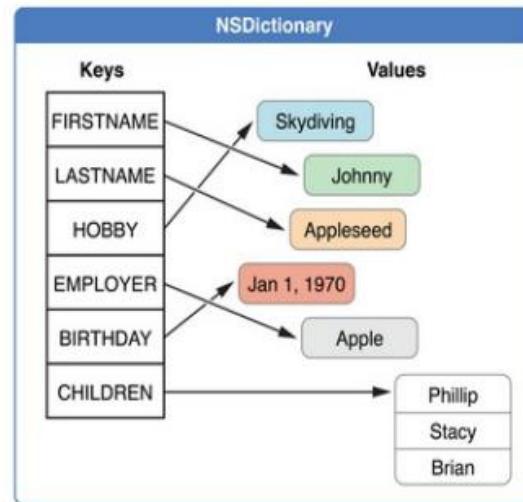
### 4. NSHashTable





## 2.1 NSDictionary là gì?

- ❑ Lớp NSDictionary là một tập hợp bất biến các đối tượng và quản lý các đối tượng dựa vào mối liên kết giữa khóa và giá trị.
- ❑ NSDictionary là tập hợp bất biến, còn NSMutableDictionary là tập hợp biến đổi các đối tượng.



## 2.2 Mục đích sử dụng trong ứng dụng

- ❑ Ta sử dụng NSDictionary với những dữ liệu tương tự từ điển.
- ❑ Trong một số trường hợp việc truy xuất đến một phần tử kiểu NSDictionary nhanh hơn so với NSArray.





## 2.3 Khảo sát lớp NSDictionary

### ❑ Tạo một đối tượng NSDictionary

- [dictionary](#)
- [dictionaryWithContentsOfFile:](#)
- [dictionaryWithContentsOfURL:](#)
- [dictionaryWithDictionary:](#)
- [dictionaryWithObject:forKey:](#)
- [dictionaryWithObjects:forKeys:](#)
- [dictionaryWithObjects:forKeys:count:](#)
- [dictionaryWithObjectsAndKeys:](#)



## 2.3 Khảo sát lớp NSDictionary

### ❑ Khởi tạo một đối tượng NSDictionary với init

- [initWithArray:](#)
- [initWithObjects:](#)
- [initWithObjects:count:](#)
- [initWithSet:](#)
- [initWithSet:copyItems:](#)
- [Init](#)



## 2.3 Khảo sát lớp NSDictionary



### ❑ Truy xuất phần tử trong NSDictionary

- [allKeys](#)
- [allKeysForObject:](#)
- [allValues](#)
- [getObjects:andKeys:](#)
- [objectForKey:](#)
- [objectForKeyedSubscript:](#)
- [objectsForKeys:notFoundMarker:](#)
- [valueForKey:](#)
- [count](#)



## 2.3 Khảo sát lớp NSDictionary



### ❑ Xuất ra kiểu Enumertator từ NSDictionary

- [isEqualToString:](#)
- [keyEnumerator](#)
- [objectEnumerator](#)
- [enumerateKeysAndObjectsUsingBlock:](#)
- [enumerateKeysAndObjectsWithOptions:usingBlock:](#)





## 2.3 Khảo sát lớp NSDictionary

### ❑ Sắp xếp NSDictionary

- [keysSortedByValueUsingSelector:](#)
- [keysSortedByValueUsingComparator:](#)
- [keysSortedByValueWithOptions:usingComparator:](#)

### ❑ Lọc NSDictionary

- [keysOfEntriesPassingTest:](#)
- [keysOfEntriesWithOptions:passingTest:](#)



## 2.3 Khảo sát lớp NSDictionary

### ❑ Tạo Description

- [description](#)
- [descriptionInStringsFileFormat](#)
- [descriptionWithLocale:](#)
- [descriptionWithLocale:indent:](#)





## 2.3 Khảo sát lớp NSDictionary

### ❑ Truy xuất tập tin

- [fileSize](#)
- [fileCreationDate](#)
- [fileExtensionHidden](#)
- [fileGroupOwnerAccountID](#)
- [fileGroupOwnerAccountName](#)
- [fileHFSCreatorCode](#)
- [fileHFSTypeCode](#)
- [fileIsAppendOnly](#)
- [fileIsImmutable](#)
- [fileModificationDate](#)
- [fileOwnerAccountID](#)
- [fileOwnerAccountName](#)
- [filePosixPermissions](#)



## 2.3 Khảo sát lớp NSDictionary

### ❑ Một số phương thức khác

- [writeToFile:atomically:](#)
- [writeToURL:atomically:](#)
- [keysOfEntriesPassingTest:](#)
- [isEqualToString:](#)
- [sharedKeySetForKeys:](#)





## 2.4 Sử dụng NSDictionary

### ❑ Tạo đối tượng NSDictionary.

- Sử dụng **initWithObjects-AndKeys** để tạo mảng NSDictionary.

```
NSDictionary *dictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"giá trị 1",@"khóa 1",@"giá trị 2",@"khóa 2", nil];
```

```
NSMutableDictionary *mutableDictionary = [[NSMutableDictionary alloc] initWithObjectsAndKeys:@"giá trị 1",@"khóa 1",@"giá trị 2",@"khóa 2", nil];
```



## 2.4 Sử dụng NSDictionary

### ❑ Truy xuất đối tượng NSDictionary.

- Sử dụng **initWithObjects-AndKeys** để tạo mảng NSDictionary.
- Truy xuất phần tử trong mảng với **objectForKey**

```
dictionary.count
```

```
mutableDictionary.count
```

```
[dictionary objectForKey:@"khóa 1"]
```

```
[mutableDictionary objectForKey:@"khóa 1"]
```



## 2.4 Sử dụng NSDictionary



### □ Thêm/xoá phần tử trong NSDictionary.

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>Thêm phần tử NSDictionary với <b>setValue</b>, <b>setObject</b>.</li><li>Xoá phần tử trong NSDictionary với <b>removeObjectForKey</b>.</li></ul> | <pre>[mutableDictionary setValue:@"giá trị 3"<br/>forKey:@"khóa 3"];<br/><br/>[mutableDictionary setObject:@"giá trị 3"<br/>forKey:@"khóa 3"];<br/><br/>[mutableDictionary<br/>removeObjectForKey:@"khóa"]]</pre> |
|--|---|



## Nội dung



1. NSSet
2. NSDictionary
3. NSEnumerator
  - NSEnumerator là gì?
  - Khảo sát lớp NSEnumerator
  - Sử dụng NSEnumerator
4. NSHashTable





### 3.1 NSEnumerator là gì?

- Lớp NSEnumerator là một lớp trừu tượng, mà lớp con liệt kê các tập hợp của các đối tượng khác như mảng.
- Tất cả phương thức khởi tạo được định nghĩa trong các lớp tập hợp như NSArray, NSSet và NSDictionary.
- Ta không thể sử dụng lại đối tượng NSEnumerator khi đã dùng rồi.



### 3.3 Khảo sát lớp NSEnumerator

- **Lấy phần tử trong NSEnumerator**
  - [allObjects](#)
  - [nextObject](#)





## 3.4 Sử dụng NSEnumerator

### ❑ Tạo đối tượng NSEnumerator.

- Tạo một NSEnumerator từ NSArray sử dụng **objectEnumerator**.

```
NSArray *aArray = [NSArray
arrayWithObjects: @"John", @"Bob",
@"Jane", nil];

NSEnumerator *enumerator = [aArray
objectEnumerator];
```



## 3.4 Sử dụng NSEnumerator

### ❑ Lấy phần tử trong NSEnumerator.

- Lấy phần tử trong NSEnumerator sử dụng **nextObject**.

```
id obj;
while(obj = [enumerator nextObject]){
    NSLog(@"%@",obj);
}
```



## Nội dung



1. NSSet
2. NSDictionary
3. NSEnumerator
4. **NSHashTable**
  - NSHashTable là gì?
  - Khảo sát lớp NSHashTable
  - Sử dụng NSHashTable



### 4.1 NSHashTable là gì?



- ❑ Lớp NSHashTable được mô hình hóa sau NSSet nhưng cung cấp các lựa chọn khác nhau, đặc biệt là có hỗ trợ tham chiếu yếu.
- ❑ Các phần tử có thể được sao chép ở đầu vào hoặc sử dụng con trỏ định danh cho việc so sánh và băm.
- ❑ Nó có thể chứa con trỏ tùy (các phần tử không cần phải là các đối tượng).





## 4.2 Khảo sát lớp NSHashTable

### ❑ Khởi tạo NSHashTable

- [initWithOptions:capacity:](#)
- [initWithPointerFunctions:capacity:](#)

### ❑ Thao tác với phần tử trong NSHashTable

- [addObject:](#)
- [removeAllObjects](#)
- [removeObject:](#)



## 4.2 Khảo sát lớp NSHashTable

### ❑ Truy xuất phần tử trong NSHashTable

- [allObjects](#)
- [anyObject](#)
- [containsObject:](#)
- [count](#)
- [member:](#)
- [objectEnumerator](#)
- [setRepresentation](#)





## 4.2 Khảo sát lớp NSHashTable

### ❑ So sánh NSHashTable

- [intersectsHashTable:](#)
- [intersectHashTable:](#)
- [isEqualToStringTable:](#)
- [isSubsetOfHashTable:](#)



## 4.2 Khảo sát lớp NSHashTable

### ❑ Một số phương thức khác

- [weakObjectsHashTable](#)
- [hashTableWithOptions:](#)
- [minusHashTable:](#)
- [unionHashTable:](#)
- [pointerFunctions](#)
- [hashTableWithWeakObjects](#) Deprecated in OS X v10.8





## 4.3 Sử dụng NSHashTable

### ❑ Sử dụng NSHashTable.

- Tạo một **NSHashTable** sử dụng **hashTable-WithOptions**.
- Thêm một phần tử bằng  **addObject**.
- Xoá phần tử bằng  **removeObject**.

```
NSHashTable *hashTable =  
[NSHashTable  
hashTableWithOptions:NSHashTableW  
eakMemory];  
  
[hashTable  addObject:@"foo"];  
  
[hashTable  addObject:@"bar"];  
  
[hashTable  addObject:@42];  
  
[hashTable  removeObject:@"bar"];
```



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 7. Tạo và đọc ghi trên thư mục dự án

Ngành Mạng & Thiết bị di động



2014

2014



## Nội dung



1. Giới thiệu về BUNDLES
2. Sơ lược về BUNDLES
3. Những cấu trúc BUNDLE
4. Truy xuất nội dung của BUNDLE



## 1 Giới thiệu về BUNDLES



- ❑ Bundles là một công nghệ cơ bản trong OS X và iOS, được sử dụng để đóng gói mã nguồn và tài nguyên.
- ❑ Bundles sử dụng các thư mục và các tập tin để sắp xếp theo một cách tự nhiên hơn, để có thể được sửa đổi một cách dễ dàng cả trong và sau khi phát triển.
- ❑ Để hỗ trợ bundles, cả Cocoa và Core Foundation cung cấp giao diện lập trình để truy xuất nội dung của bundles.



## Nội dung



1. Giới thiệu về BUNDLES
2. Sơ lược về BUNDLES
  - Bundles và Packages
    - Packages
    - Bundles
    - Cách mà hệ thống xác định Bundles và Packages
  - Ưu điểm của Bundle
3. Những cấu trúc BUNDLE
4. Truy xuất nội dung của BUNDLE



## 2 Sơ lược về BUNDLES



- ❑ Bundles là cách đơn giản để cung cấp phần mềm trong OS X và iOS. Bundles cung cấp một giao diện đơn giản cho các người dùng đầu cuối và cùng lúc cung cấp hỗ trợ cho nhà phát triển.





## 2.1 Bundles và Packages

- Một package là bất kì thư mục nào mà Finder thể hiện cho người sử dụng như thể nó là một tập tin đơn lẻ.
- Một bundle là một thư mục với một cấu trúc phân cấp được tiêu chuẩn hóa chứa mã thực thi và các tài nguyên được sử dụng bởi các mã thực thi này.



### 2.1.1 Packages

- Packages cung cấp một trong các khái niệm trừu tượng cơ bản để làm cho OS X dễ dàng sử dụng. Nếu ta thấy một ứng dụng hoặc một plug-in trên máy tính, cái ta thực sự thấy là thư mục.
- Bên trong thư mục package là các tập tin mã lệnh và tài nguyên cần thiết để giúp cho ứng dụng hoặc plug-in chạy. Khi ta tương tác với thư mục package, Finder sẽ xem nó như là một tập tin đơn.



## 2.1.1 Packages

- ❑ Các ứng dụng nằm trong Application là các packages.



Lập trình iOS (2014) – Bài 7. Tạo và đọc ghi trên thư mục trên dự án

8

## 2.1.2 Bundles

- ❑ Bundles được hướng tới việc giúp đỡ các nhà phát triển đóng gói mã lệnh của họ và giúp hệ điều hành truy xuất mã lệnh đó.
- ❑ Bundles định nghĩa cấu trúc cơ bản cho việc tổ chức mã lệnh và tài nguyên liên quan đến phần mềm. Cấu trúc chính xác của một bundle phụ thuộc vào việc tạo ra ứng dụng, framework, hay là plugin.
- ❑ Bundles và packages đôi khi có thể hoán đổi cho nhau. Tuy nhiên, không phải tất cả các bundles đều là package và ngược lại.



Lập trình iOS (2014) – Bài 7. Tạo và đọc ghi trên thư mục trên dự án

9

## 2.1.3 Cách mà hệ thống xác định Bundles và Packages



- ❑ Finder xem một thư mục như là package nếu các điều kiện sau là đúng:
  - Thư mục có phần mở rộng trong tên là: .app, .bundle, .framework, .plugin, .kext, ...
  - Thư mục có phần mở rộng mà một số ứng dụng khác hiển thị một loại package.
  - Thư mục được tạo nên từ gói bit.



## 2.2 Ưu điểm của Bundles



- ❑ Vì bundles là hệ thống phân cấp thư mục trong tập tin hệ thống, một bundle chỉ chứa các tập tin. Do đó, ta có thể sử dụng những giao diện dựa trên tập tin tương tự để mở tài nguyên của bundle khi mở tập tin thuộc loại khác.
- ❑ Cấu trúc thư mục bundle dễ dàng hỗ trợ đa cục bộ. Ta có thể dễ dàng thêm các tài nguyên cục bộ mới hoặc loại bỏ những cái không cần.
- ❑ Bundles có thể giới hạn dung lượng của nhiều định dạng khác nhau.





## 2.2 Ưu điểm của Bundles

- Người dùng có thể cài đặt, di chuyển, loại bỏ các bundle một cách đơn giản là kéo chúng quanh Finder.
- Bundles cũng là packages, do đó chúng được xem như các tập tin không rõ ràng, người dùng sẽ không thể vô tình mà thay đổi chúng.
- Bundle có thể hỗ trợ nhiều kiến trúc chip (PowerPC, Intel) và các yêu cầu về không gian địa chỉ (32 bit/64 bit). Nó cũng có thể hỗ trợ các tập tin thực thi chuyên biệt.



## Nội dung

1. Giới thiệu về BUNDLES
2. Sơ lược về BUNDLES
3. Những cấu trúc BUNDLE
  - Bundles ứng dụng
    - Các loại tập tin trong Bundle ứng dụng
    - Cấu tạo của một Bundle ứng dụng iOS
      - Cấu trúc bundle ứng dụng iOS
      - Tập tin danh sách thông tin thuộc tính
      - Tài nguyên trong ứng dụng iOS
  - Bundle Framework
4. Truy xuất nội dung của BUNDLE





## 3.1 Bundles ứng dụng

- ❑ Bundles ứng dụng là một trong các loại thông thường được tạo bởi các nhà phát triển.
- ❑ Bundle ứng dụng lưu trữ mọi thứ mà ứng dụng yêu cầu để có thể chạy được.
- ❑ Mặc dù cấu trúc của bundle ứng dụng phụ thuộc vào nền tảng mà ta đang phát triển, cách sử dụng bundle tương tự trên các nền tảng.



### 3.1.1 Các loại tập tin trong Bundles ứng dụng

Tập tin info.plist	(Bắt buộc) Tập tin danh sách thông tin thuộc tính (information property list) là một tập tin được cấu trúc mà chứa thông tin cấu trúc của ứng dụng. Hệ thống dựa trên tập tin này để xác định thông tin liên quan tới ứng dụng và các tập tin liên quan.
Thực thi	(Bắt buộc) Mọi ứng dụng phải có một tập tin thực thi. Tập tin này chứa đầu vào chính của ứng dụng và bất kỳ mã lệnh nào đã được liên kết tĩnh tới ứng dụng.
Các tập tin tài nguyên	Tài nguyên là các tập tin dữ liệu bên ngoài tập tin thực thi của ứng dụng. Tài nguyên thường bao gồm hình ảnh, hình icon, âm thanh, các tập tin nib, các tập tin chuỗi, các tập tin cấu hình và các tập tin dữ liệu. Các tập tin tài nguyên có thể được cục bộ hóa cho một ngôn ngữ hoặc khu vực cụ thể. Vị trí của các tập tin tài nguyên trong cấu trúc thư mục bundle phục thuộc vào việc phát triển một ứng dụng trên iOS hay Mac.
Các tập tin hỗ trợ khác	Các ứng dụng Mac có thể nhúng các tài nguyên cấp cao như các framework riêng, các plug-in, các mẫu tài liệu, và tài nguyên dữ liệu tùy chỉnh khác mà không thể thiếu cho các ứng dụng. Mặc dù có thể chứa tài nguyên dữ liệu tùy chỉnh trong các bundle ứng dụng iOS, không thể chứa các framework tùy chỉnh hoặc các plug-in.



### 3.1.2 Cấu tạo của một Bundles ứng dụng iOS

- ❑ Cấu trúc bundle của các ứng dụng iOS hướng nhiều hơn đến nhu cầu của thiết bị di động.
- ❑ Nó sử dụng một cấu trúc tương đối phẳng với các thư mục không liên quan trong việc tiết kiệm không gian đĩa và đơn giản hóa truy cập các tập tin.



#### 3.1.2.1 Cấu trúc Bundles ứng dụng iOS

- ❑ Một bundle ứng dụng iOS đặc trưng chứa ứng dụng có thể thực thi và bất kỳ tài nguyên nào được sử dụng bởi ứng dụng ở mức độ trên cùng của thư mục bundle.
- ❑ Cấu trúc của một ứng dụng iPhone cơ bản tên là **MyApp**.

```

MyApp.app
MyApp
MyAppIcon.png
MySearchIcon.png
Info.plist
Default.png
MainWindow.nib
Settings.bundle
MySettingsIcon.png
iTunesArtwork
en.lproj
    MyImage.png
fr.lproj
    MyImage.png

```





### 3.1.2.1 Cấu trúc Bundles ứng dụng iOS

- ❑ Một ứng dụng iOS nên được quốc tế hóa và có một thư mục `language.lproj` cho mỗi ngôn ngữ mà nó hỗ trợ.
- ❑ Ta cũng có thể cục bộ hóa các biểu tượng của ứng dụng và hình ảnh khi chạy bằng cách đặt các tập tin cùng tên trong thư mục dành riêng cho ngôn ngữ.



### 3.1.2.2 Tập tin danh sách thông tin thuộc tính

- ❑ Mọi ứng dụng iOS phải có một tập tin danh sách thông tin thuộc tính (`Info.plist`) chứa thông tin cấu hình của ứng dụng.
- ❑ Khi ta tạo một dự án ứng dụng iOS mới, Xcode tự động tạo ra tập tin này và thiết lập giá trị của một số khóa thuộc tính. Những khóa thuộc tính cần thiết cho tập tin `Info.plist`.
  - `CFBundleDisplayName` (Tên Hiển thị Bundle)
  - `CFBundleIdentifier` (Định danh Bundle)
  - `CFBundleVersion` (Phiên bản bundle)
  - `CFBundleIconFiles`
  - `LSRequiresiPhoneOS` (Ứng dụng yêu cầu môi trường iOS)
  - `UIRequiredDeviceCapabilities`





### 3.1.2.3 Tài nguyên trong ứng dụng iOS

- ❑ Trong một ứng dụng iOS, tài nguyên không được cục bộ hóa được đặt ở mức độ trên cùng của thư mục bundle.
- ❑ Tài nguyên không cục bộ hóa gồm: Hand.png, MainWindow.nib, MyAppViewController.nib...
- ❑ Tài nguyên cục bộ hóa bao gồm mọi thứ trong en.lproj và jp.lproj.

```
MyApp.app/
Info.plist
MyApp
Default.png
Icon.png
Hand.png
MainWindow.nib
MyAppViewController.nib
WaterSounds/
    Water1.aiff
    Water2.aiff
en.lproj/
    CustomView.nib
    bird.png
    Bye.txt
    Localizable.strings
jp.lproj/
    CustomView.nib
    bird.png
    Bye.txt
    Localizable.strings
```



### 3.2 Bundle Framework

- ❑ Framework là một thư mục phân cấp đóng gói một thư viện được chia sẻ động và các tập tin tài nguyên cần để hỗ trợ cho thư viện.
- ❑ Framework cung cấp lợi ích còn hơn so với thư viện được chia sẻ động trong việc cung cấp một vị trí duy nhất cho tất cả tài nguyên liên quan của framework.
- ❑ Giống như một thư viện được chia sẻ động, framework cung cấp một cách để sử dụng như một mã lệnh thông thường và có thể được chia sẻ cho nhiều ứng dụng.



## 3.2 Bundle Framework



- ❑ Chỉ cần sao chép một lần tài nguyên và mã lệnh của framework nằm trong bộ nhớ, bất kỳ lúc nào, bất kể có bao nhiêu tiến trình đang sử dụng những nguồn tài nguyên đó.
- ❑ Các ứng dụng liên kết với framework sau đó chia sẻ bộ nhớ chứa framework. Điều này giúp giảm bộ nhớ của hệ thống và giúp cải thiện hiệu suất.



## Nội dung



1. Giới thiệu về BUNDLES
2. Sơ lược về BUNDLES
3. Những cấu trúc BUNDLE
4. Truy xuất nội dung của BUNDLE
  - Định vị và mở Bundle
  - Tham chiếu tới tài nguyên Bundle
  - Tìm các tập tin khác trong Bundle
  - Lấy dữ liệu Info.plist của Bundle





## 4 Truy xuất nội dung của Bundle

- ❑ Khi viết mã lệnh dựa trên bundle, ta không bao giờ sử dụng chuỗi bất biến trả tới vị trí của các tập tin trong bundle. Thay vì thế, ta sử dụng lớp NSBundle hoặc loại CFBundleRef(không rõ ràng) để nhận đường dẫn tới tập tin ta muốn.



### 4.1 Định vị và mở Bundle

#### ❑ Lấy main bundle

- Main Bundle là bundle mà chứa mã lệnh và tài nguyên cho việc chạy ứng dụng.
- Để lấy main bunlde trong ứng dụng Cocoa, gọi **mainBundle** phương thức của lớp NSBundle:

```
NSBundle *mainBundle = [NSBundle mainBundle];
```

- Nếu ta đang viết ứng dụng dựa trên nền C, ta có thể dùng hàm **CFBundleGetMainBundle** để lấy main bundle:

```
CFBundleRef mainBundle = CFBundleGetMainBundle();
```





## 4.1 Định vị và mở Bundle

### ❑ Lấy bundle theo đường dẫn

- Nếu muốn truy xuất một bundle khác không phải là main bundle, ta có thể tạo một đối tượng bundle thích hợp nếu ta biết đường dẫn tới thư mục bundle.
- Để nhận bundle theo đường dẫn bằng cách dùng Cocoa, gọi phương thức **bundleWithPath:** của lớp NSBundle.

```
NSBundle *mainBundle = [NSBundle  
bundleWithPath:@"/Library/MyBundle.bundle"];
```



## 4.1 Định vị và mở Bundle

### ❑ Lấy bundle theo định danh

- Một định danh bundle là chuỗi được gán vào khóa **CFBundleIdentifier** trong tập tin **Info.plist** của bundle.
- Lấy một bundle bằng cách sử dụng định danh bundle trong Cocoa, gọi phương thức **bundleWithIdentifier:** của lớp NSBundle:

```
NSBundle *mainBundle = [NSBundle  
bundleWithPath:@"/Library/MyBundle.bundle"];
```

- Định vị một bundle bằng cách sử dụng định danh trong Core Foundation:

```
CFBundleRef requestedBundle = CFBundleGetBundleWithIdentifier  
(CFSTR("com.apple.Finder.MyGetInfoPlugIn")) ;
```





## 4.2 Tham chiếu tới tài nguyên Bundle

- ❑ Nếu ta tham chiếu tới một đối tượng bundle, ta có thể sử dụng đối tượng đó để xác định vị trí của tài nguyên bên trong bundle.
- ❑ Cả Cocoa và Core Foundation đều cung cấp nhiều cách khác nhau trong việc định vị tài nguyên bên trong bundle.



## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Bundle Search Pattern

- Khi sử dụng một đối tượng NSBundle hoặc CFBundleRef để định vị tài nguyên, mã lệnh bundle không bao giờ quan tâm tới cách mà nó nhận tài nguyên từ một bundle. Cả NSBundle và CFBundleRef tự động lấy dữ liệu ngôn ngữ cụ thể thích hợp dựa trên cài đặt sẵn của người dùng và cấu hình bundle. Tuy nhiên, vẫn phải đặt tất cả dữ liệu ngôn ngữ vào trong bundle, vì vậy biết được cách mà lấy dữ liệu rất quan trọng.





## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Bundle Search Pattern

- Danh sách cho thấy thứ tự tìm kiếm dữ liệu:
  - Tài nguyên toàn cục.(không cục bộ hóa)
  - Khu vực – tài nguyên cụ thể (dựa trên ưu tiên khu vực của người dùng)
  - Dữ liệu ngôn ngữ cụ thể (dựa trên ưu tiên ngôn ngữ của người dùng)
  - Ngôn ngữ phát triển của bundle (được cụ thể bởi CFBundleDevelopmentRegion trong tập tin Info.plist của bundle).



## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Tài nguyên thiết bị cụ thể trong iOS

- Từ iOS 4.0 trở đi, có thể đánh dấu tài nguyên cá nhân khi chỉ có thể sử dụng một loại thiết bị cụ thể. Việc tạo đường dẫn mã lệnh riêng biệt để lấy một phiên bản của tập tin tài nguyên cho iPhone và tập tin phiên bản khác cho iPad.
- Tất cả những gì cần làm chỉ là đặt tên cho tập tin tài nguyên một cách thích hợp.





## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Tài nguyên thiết bị cụ thể trong iOS

- Để kết hợp một tập tin tài nguyên với một thiết bị cụ thể, ta sẽ sửa đổi tên của tập tin theo định dạng:  
*<tên cơ bản><thiết bị>. <phần mở rộng tên tập tin>*
- *<tên cơ bản>* cho biết tên gốc của tập tin.
- *<phần mở rộng tên tập tin>* là phần mở rộng tên tập tin
- *<thiết bị>* chỉ có thể là một trong các giá trị sau:
  - *~ipad* – Tài nguyên chỉ được load trên thiết bị iPad.
  - *~iphone* – Tài nguyên chỉ được load trên iPhone hoặc iPod touch.



## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Tài nguyên thiết bị cụ thể trong iOS

- Ví dụ:
  - Một tấm hình được đặt tên *MyImage.png*. Để phân biệt các phiên bản khác nhau của iPad và iPhone, nên tạo tập tin tài nguyên với tên *MyImage~ipad.png* và *MyImage~iphone.png* và thêm chúng vào bundle. Để load hình, ta tiếp tục trỏ tới *MyImage.png* và để cho hệ thống chọn lựa phiên bản thích hợp.

```
UIImage* anImage = [UIImage imageNamed:@"MyImage.png"];
```

- Trên thiết bị iPhone hoặc iPod touch, hệ thống sẽ dùng tập tin *MyImage~iphone.png*, còn trên iPad thì dùng tập tin *MyImage~ipad.png*





## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Lấy đường dẫn tới tài nguyên

- Để xác định vị trí của tập tin trong bundle, ta cần biết tên của tập tin, loại của nó, hoặc là cả hai. Phần mở rộng tập tin được sử dụng để định danh loại của tập tin; do đó, điều quan trọng là tập tin tài nguyên phải có phần mở rộng thích hợp.
- Thậm chí nếu không có đối tượng bundle, vẫn có thể tiếp tục tìm kiếm tài nguyên trong đường dẫn các thư mục mà ta biết. Cả Core Foundation và Cocoa cung cấp API để tìm kiếm tập tin mà chỉ sử dụng đường dẫn cơ bản.



## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Lấy đường dẫn tới tài nguyên

- Sử dụng Cocoa để tìm tài nguyên
  - `pathForResource:ofType:`
  - `pathForResource:ofType:inDirectory:`
  - `pathForResource:ofType:inDirectory:forLocalization:`
  - `pathsForResourcesOfType:inDirectory:`
  - `pathsForResourcesOfType:inDirectory:forLocalization:`

- Ví dụ:

```
NSBundle* myBundle = [NSBundle mainBundle];
NSString* myImage = [myBundle pathForResource:@"Seagull"
ofType:@"jpg"];
```





## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Lấy đường dẫn tới tài nguyên

- Sử dụng Core Foundation để tìm tài nguyên
  - `CFBundleCopyResourceURL`
  - `CFBundleCopyResourceURLInDirectory`
  - `CFBundleCopyResourceURLsOfType`
  - `CFBundleCopyResourceURLsOfTypeInDirectory`
  - `CFBundleCopyResourceURLsOfTypeForLocalization`
- Ví dụ:

```
CFURLRef seagullURL = CFBundleCopyResourceURL( mainBundle,  
                                              CFSTR("Seagull"),  
                                              CFSTR("jpg"),  
                                              NULL );
```



## 4.2 Tham chiếu tới tài nguyên Bundle

### ❑ Mở và sử dụng tập tin tài nguyên

- Một khi tham chiếu tới tập tin tài nguyên, ta có thể hiển thị nội dung và sử dụng nó trong ứng dụng. Các bước phải làm để hiển thị và sử dụng tập tin tài nguyên phụ thuộc vào loại tài nguyên.





## 4.3 Tìm các tập tin khác trong Bundle

- ❑ Với một đối tượng bundle hợp lệ, ta có thể lấy đường dẫn tới thư mục bundle ở mức cao nhất cũng như đường dẫn tới các thư mục con.
- ❑ Nó cũng cho phép sử dụng cùng mã lệnh trên nhiều nền tảng khác nhau.
- ❑ Ta cũng có thể sử dụng phương thức `builtInPlugInPath`, `resourcePath`, `sharedFrameworksPath`, và `sharedSupportPath` để lấy đường dẫn cho thư mục con của bundle.



## 4.3 Tìm các tập tin khác trong Bundle

- ❑ Core Foundation cũng định nghĩa các hàm để nhận nhiều thư mục bundle nội bộ khác.
- ❑ Để lấy đường dẫn của chính bundle, ta có thể sử dụng phương thức:
  - `CFBundleCopyBundleURL`.
  - `CFBundleCopyBuiltInPlugInsURL`,
  - `CFBundleCopyResourcesDirectoryURL`,
  - `CFBundleCopySharedFrameworksURL`,
  - `CFBundleCopySupportFilesDirectoryURL`
- ❑ Core Foundation luôn trả về đường dẫn bundle là một loại `CFURLRef`. Ta có thể sử dụng loại này để trích xuất một loại `CFStringRef` mà ta có thể truyền tới Core Foundation khác.





## 4.4 Lấy dữ liệu Info.plist của Bundle

- ❑ Info.plist là một tập tin văn bản XML chứa các cặp khóa – giá trị.
- ❑ Những cặp khóa – giá trị xác định thông tin về bundle, như chuỗi ID của nó, số phiên bản, khu vực phát triển, chủng loại, và tính chất quan trọng khác.
- ❑ Bundle cũng có thể bao gồm các loại khác của dữ liệu cấu hình, hầu như được sắp xếp trong danh sách thuộc tính XML.
- ❑ Lớp NSBundle cung cấp phương thức `objectForInfoDictionaryKey:` và `infoDictionary` để nhận thông tin từ tập tin Info.plist.



## 4.4 Lấy dữ liệu Info.plist của Bundle

- ❑ Core Foundation cũng cung cấp các hàm để lấy các phần dữ liệu từ tập tin danh sách thuộc tính của bundle, bao gồm chuỗi ID của bundle, phiên bản, và khu vực phát triển.
- ❑ Ta có thể lấy giá trị cục bộ cho khóa bằng cách dùng phương thức
  - `CFBundleGetValueForInfoDictionaryKey`.
  - `CFBundleGetValueForInfoDictionary`.



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 8. Xây dựng giao diện với các controller

Ngành Mạng & Thiết bị di động



2014

5014



## Nội dung



### 1. NavigationController

- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp UINavigationController
- Xây dựng NavigationController
- Các tác vụ trong NavigationController

### 2. PageViewController

### 3. PopoverController

### 4. SplitViewController

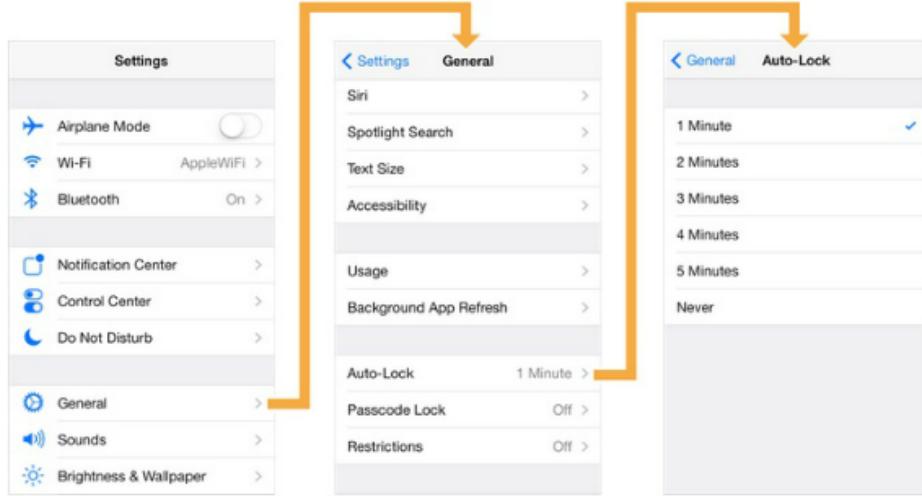
### 5. TabBarController



## 1.1 Mục đích sử dụng trong ứng dụng



- ❑ NavigationController nằm trên cùng của màn hình ứng dụng, giúp ta chuyển hướng từ ViewController này sang ViewController khác, rất thường được sử dụng trong các ứng dụng iOS.



## 1.2 Khảo sát lớp UINavigationController



- **Tạo UINavigationController.**
  - [initWithRootViewController:](#)
  - [initWithNavigationBarClass:toolbarClass:](#)
  
- **Truy cập vào ViewController trong Navigation stack**
  - [topViewController \*property\*](#)
  - [visibleViewController \*property\*](#)
  - [viewControllers \*property\*](#)
  - [setViewControllers:animated:](#)



## 1.2 Khảo sát lớp UINavigationController



- **Thêm/bớt ViewController trong Navigation stack.**
  - [pushViewController:animated:](#)
  - [popViewControllerAnimated:](#)
  - [popToRootViewControllerAnimated:](#)
  - [popToViewController:animated:](#)
  - [interactivePopGestureRecognizer \*property\*](#)





## 1.2 Khảo sát lớp UINavigationController

- **Tùy chỉnh Navigation bar.**
  - [navigationBar property](#)
  - [navigationBarHidden property](#)
  - [setNavigationBarHidden:animated:](#)
  
- **Tùy chỉnh cấu hình Toolbar**
  - [toolbar property](#)
  - [setToolbarHidden:animated:](#)
  - [toolbarHidden property](#)



## 1.3 Xây dựng NavController

- **Xây dựng Navigation Controller từ mã nguồn.**
  - Trong **AppDelegate.m** ta tạo một Navigation Controller và gán rootView cho nó với ViewController mà ta đã tạo từ trước.

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions  
{  
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen  
        mainScreen] bounds]];  
    // Override point for customization after application launch.  
  
    MainViewController *view = [[MainViewController alloc] init];  
    UINavigationController *navigation = [[UINavigationController  
        alloc] initWithRootViewController:view];  
    [self.window setRootViewController:navigation];  
  
    self.window.backgroundColor = [UIColor whiteColor];  
    [self.window makeKeyAndVisible];  
    return YES;  
}
```





## 1.3 Xây dựng NavigationController

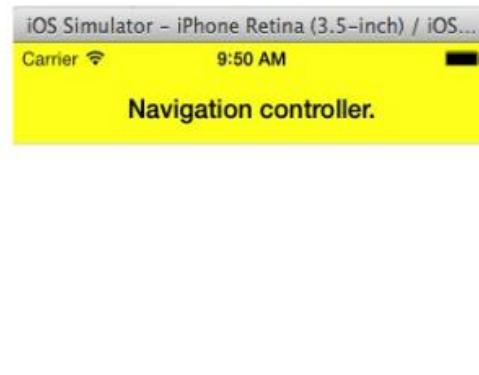
### □ Xây dựng Navigation Controller từ mã nguồn.

- Sau khi gán rootView cho navigation ta sẽ gán navigation cho rootView của window.

```
[self.window setRootViewController:navigation];
```

- Ta thay đổi màu cho Navigation.

```
self.title = @"Navigation controller.";  
[self.navigationController.navigationBar  
    setBarTintColor:[UIColor yellowColor]]  
];
```



## 1.4 Các tác vụ trong NavigationController

### □ Tác vụ hiển thị ViewController.

- [navigationController:willShowViewController:animated:](#)
- [navigationController:didShowViewController:animated:](#)





## 1.4 Các tác vụ trong NavigationController

### ❑ Hỗ trợ tùy chỉnh animation

- [navigationController:animationControllerForOperation:fromViewController:toViewController:](#)
- [navigationController:interactionControllerForAnimationController:](#)
- [navigationControllerPreferredInterfaceOrientationForPresentation:](#)
- [navigationControllerSupportedInterfaceOrientations:](#)



## Nội dung

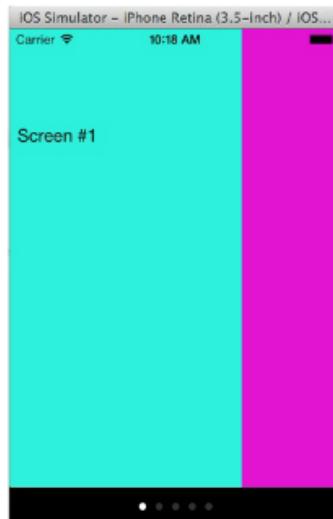
1. NavigationController
2. PageViewController
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UIPageViewController
  - Xây dựng PageViewController
  - Các tác vụ trong PageViewController
3. PopoverController
4. SplitViewController
5. TabBarController





## 2.1 Mục đích sử dụng trong ứng dụng

- PageViewController giúp ta chuyển hướng từ ViewController này sang ViewController nhưng mang tính trình diễn nhiều hơn, như khi ta cho hiển thị các hình ảnh, các trang của một tập tin pdf, word.



## 2.2 Khảo sát lớp UIPageViewController

- Tạo PageViewController.
  - `initWithTransitionStyle:navigationOrientation:options:`
  - `dataSource property`
  - `delegate property`





## 2.2 Khảo sát lớp UIPageViewController

### ❑ Cung cấp nội dung cho PageViewController.

- [setViewControllers:direction:animated:completion:](#)
- [viewControllers \*property\*](#)
- [gestureRecognizers \*property\*](#)



## 2.2 Khảo sát lớp UIPageViewController

### ❑ Tuỳ chỉnh hiển thị PageViewController.

- [navigationOrientation \*property\*](#)
- [spineLocation \*property\*](#)
- [transitionStyle \*property\*](#)
- [doubleSided \*property\*](#)





## 2.3 Xây dựng PageViewController

- Xây dựng Page View Controller từ mã nguồn.
  - Ta tạo ra 2 lớp kế thừa từ **ViewController** là **MainViewController** để chứa **PageViewController** và **ChildViewController** để chứa nội dung cho từng page.
  - Sau đó ta thiết lập cho **MainViewController** làm **rootViewController**.
  - Cung cấp thông tin về số trang, trang hiển thị, nội dung trang ... cho PageViewController thông qua các tác vụ.



## 2.4 Các tác vụ trong PageViewController

- Lấy ViewController cho PageViewController.
  - [pageViewController:viewControllerBeforeViewController:](#)
  - [pageViewController:viewControllerAfterViewController:](#)
- Hỗ trợ lấy số trang cho PageViewController
  - [presentationCountForPageViewController:](#)
  - [presentationIndexForPageViewController:](#)





## 2.4 Các tác vụ trong PageViewController

### ❑ Các tác vụ sự kiện PageViewController.

- [pageViewController:willTransitionToViewControllers:](#)
- [pageViewController:didFinishAnimating:previousViewControllers:transitionCompleted:](#)
- [pageViewController:spineLocationForInterfaceOrientation:](#)

### ❑ Một số tác vụ khác

- [pageViewControllerSupportedInterfaceOrientations:](#)
- [pageViewControllerPreferredInterfaceOrientationForPresentation:](#)



## Nội dung

1. [NavigationController](#)
2. [PageViewController](#)
3. [PopoverController](#)
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UIPopoverController
  - Xây dựng PopoverController
  - Các tác vụ trong PopoverController
4. [SplitViewController](#)
5. [TabBarController](#)





### 3.1 Mục đích sử dụng trong ứng dụng

- ❑ **PopoverController** giúp ta hiển thị một **ViewController** trên **ViewController** khác.



### 3.2 Khảo sát lớp UIPopoverController

- ❑ **Tạo Popover**
  - [initWithContentViewController:](#)
- ❑ **Hiển thị/loại bỏ Popover**
  - [presentPopoverFromRect:inView:permittedArrowDirections:animated:](#)
  - [presentPopoverFromBarButtonItem:permittedArrowDirections:animated:](#)
  - [dismissPopoverAnimated:](#)





## 3.2 Khảo sát lớp UIPopoverController

### ❑ Cấu hình Popover

- [contentViewController](#) *property*
- [setContentViewController:animated:](#)
- [popoverContentSize](#) *property*
- [setPopoverContentSize:animated:](#)
- [passthroughViews](#) *property*



## 3.2 Khảo sát lớp UIPopoverController

### ❑ Một số thuộc tính Popover

- [popoverVisible](#) *property*
- [popoverArrowDirection](#) *property*

### ❑ Tuỳ chỉnh giao diện Popover

- [popoverLayoutMargins](#) *property*
- [popoverBackgroundViewClass](#) *property*
- [backgroundColor](#) *property*





### 3.3 Xây dựng PopoverController

#### □ Xây dựng Popover Controller từ mã nguồn.

- Vì Popover Controller không hỗ trợ **iphone** nên khi khởi tạo ta phải chọn **iPad** hoặc **Universal**.
- Ta tạo một Popover trong tập tin **MainViewController.h** và sử dụng đoạn mã sau để gọi **ChildViewController** khi cần.

```
// Tạo một ChildViewController
ChildViewController *childViewController =
    [[ChildViewController alloc] init];

// Tạo popOverController chứa childViewController vừa tạo
self.popOverController = [[UIPopoverController alloc]
    initWithContentViewController:childViewController];

// Hiện PopOverController
[self.popOverController presentPopoverFromRect:CGRectMake(0, 0,
    35, 35) inView:sender permittedArrowDirections:
    UIPopoverArrowDirectionUp animated:YES];
```



### 3.4 Các tác vụ trong PopoverController

#### □ Một số tác vụ trong PopoverController.

- popoverController:willRepositionPopoverToRect:inView:
- popoverControllerShouldDismissPopover:
- popoverControllerDidDismissPopover:



## Nội dung



1. **NavigationController**
2. **PageViewController**
3. **PopoverController**
4. **SplitViewController**
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UISplitViewController
  - Xây dựng SplitViewController
  - Các tác vụ trong SplitViewController
5. **TabBarController**



### 4.1 Mục đích sử dụng trong ứng dụng



- ❑ **Lớp SplitViewController chứa ViewController quản lý hiển thị của hai ViewController.** Ta có thể dùng SplitViewController để hiển thị một danh sách các lựa chọn bên trái, và bên phải là chi tiết của lựa chọn đó.
- ❑ **Chú ý: SplitViewController cũng như PopoverController là chỉ có thể dùng trên iPad do hai ViewController này đòi hỏi khá nhiều không gian.**



## 4.2 Khảo sát lớp UISplitViewController



- Quản lý Child View Controller
  - `viewControllers` *property*
  - `presentsWithGesture` *property*
- Truy cập delegate cho SplitViewController
  - `delegate` *property*



## 4.3 Xây dựng SplitViewController



- Xây dựng SplitViewController từ mã nguồn.
  - Ta sẽ tạo ra 2 **ViewController** là **MasterViewController** và **DetailViewController**:
    - **MasterViewController** dùng để hiển thị danh sách nên ta có thể kế thừa từ **UITableViewController**.
    - **DetailViewController** dùng để hiển thị nội dung.
  - Ta vào **AppDelegate.m** tạo **SplitViewController** và gán 2 ViewController vừa tạo ở trên vào.
  - Gán **SplitViewController** cho **rootViewController** của window.





## 4.4 Các tác vụ trong SplitViewController

### ❑ Ân/hiện ViewController

- [splitViewController:shouldHideViewController:inOrientation:](#)
- [splitViewController:willHideViewController:withBarButtonItem:forPopoverController:](#)
- [splitViewController:willShowViewController:invalidatingBarButtonItem:](#)
- [splitViewController:popoverController:willPresentViewController:](#)



## 4.4 Các tác vụ trong SplitViewController

### ❑ Một số tác vụ khác

- [splitViewControllerSupportedInterfaceOrientations:](#)
- [splitViewControllerPreferredInterfaceOrientationForPresentation:](#)



## Nội dung



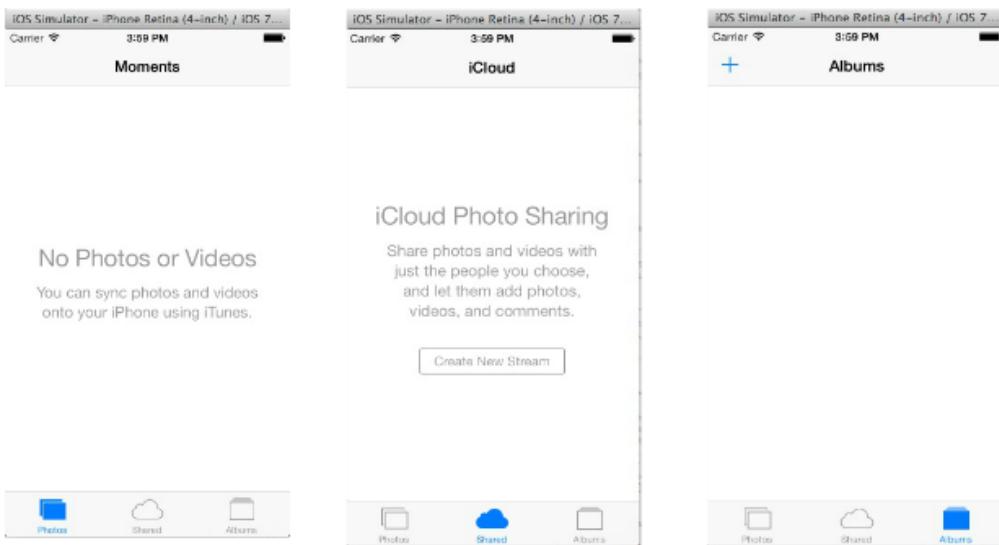
1. **NavigationController**
2. **PageViewController**
3. **PopoverController**
4. **SplitViewController**
5. **TabBarController**
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UISplitViewController
  - Xây dựng SplitViewController
  - Các tác vụ trong SplitViewController



## 5.1 Mục đích sử dụng trong ứng dụng



- ❑ **UITabBarController** có thể giúp ta chuyển hướng một cách tùy ý. Ví dụ, từ ViewController 1, ta có thể chuyển sang ViewController 2 hay ViewController 3 chỉ với một lần chạm.





## 5.2 Khảo sát lớp UITabBarController

### Truy cập vào thuộc tính điều khiển của Tabbar

- [delegate property](#)
- [tabBar property](#)

### Quản lý tab được chọn

- [selectedViewController property](#)
- [selectedIndex property](#)



## 5.2 Khảo sát lớp UITabBarController

### Quản lý ViewController

- [viewControllers property](#)
- [setViewControllers:animated:](#)
- [customizableViewControllers property](#)
- [moreINavigationController property](#)





## 5.3 Xây dựng UITabbarController

- **Xây dựng UITabbarController từ mã nguồn.**
  - Ta sẽ tạo ra 2 ViewController là **FirstViewController** và **SecondViewController** để có thể chuyển đổi.
  - Ta vào **AppDelegate.m** tạo **TabbarController** và gán 2 ViewController vừa tạo ở trên vào.
  - Gán **TabbarController** cho **rootViewController** của window.



## 5.4 Các tác vụ trong TabbarController

- **Quản lý tab được chọn**
  - [tabBarController:shouldSelectViewController:](#)
  - [tabBarController:didSelectViewController:](#)
- **Quản lý tuỳ chỉnh tabbar**
  - [tabBarController:willBeginCustomizingViewControllers:](#)
  - [tabBarController:willEndCustomizingViewControllers:changed:](#)
  - [tabBarController:didEndCustomizingViewControllers:changed:](#)





## 5.4 Các tác vụ trong TabBarController

### ❑ Một số tùy chọn khác

- [tabBarControllerSupportedInterfaceOrientations:](#)
- [tabBarControllerPreferredInterfaceOrientationForPresentation:](#)
- [tabBarController:animationControllerForTransitionFromViewController:toViewController:](#)
- [tabBarController:interactionControllerForAnimationController:](#)



## Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## Lập trình iOS

### Bài 9. Các điều khiển trạng thái và điều chỉnh

Ngành Mạng & Thiết bị di động



2014

5014



## Nội dung



### 1. Activity Indicator

- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp UIActivityIndicatorView
- Xây dựng ActivityIndicator

### 2. AlertView

### 3. ProgressView



### 1.1 Mục đích sử dụng trong ứng dụng



- **ActivityIndicator** dùng để hiển thị cho người dùng biết ứng dụng đang xử lý, người dùng sẽ phải đợi quá trình xử lý hoàn tất .





## 1.2 Khảo sát lớp UIActivityIndicatorView

- **Tạo ActivityIndicator.**
  - [initWithActivityIndicatorStyle:](#)
- **Cấu hình giao diện bên ngoài của ActivityIndicator**
  - [activityIndicatorViewStyle \*property\*](#)
  - [color \*property\*](#)



## 1.2 Khảo sát lớp UIActivityIndicatorView

- **Quản lý ActivityIndicator.**
  - [startAnimating](#)
  - [stopAnimating](#)
  - [isAnimating](#)
  - [hidesWhenStopped \*property\*](#)

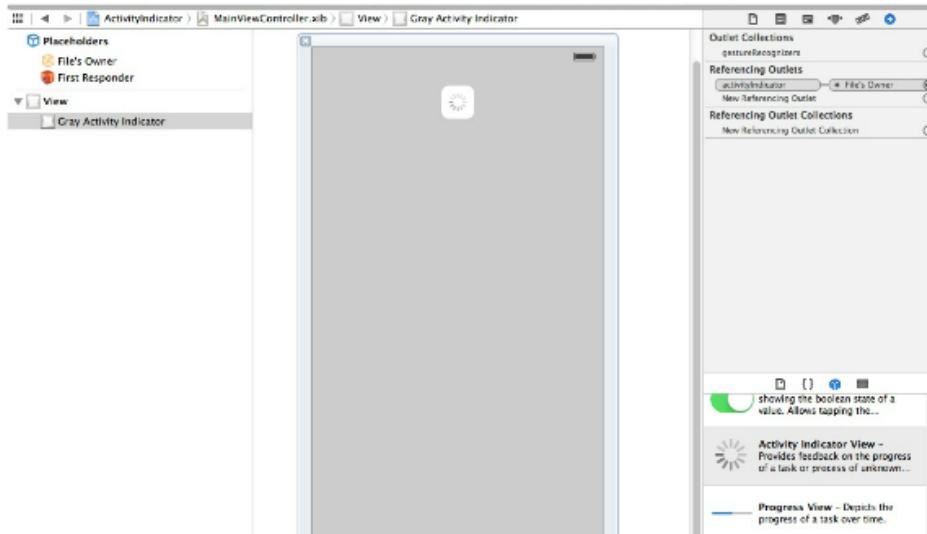


## 1.3 Xây dựng ActivityIndicator



### □ Xây dựng ActivityIndicator từ User Interface.

- Thực hiện kéo đổi tượng **ActivityIndicator** từ Object Library vào xib, sau đó thực hiện tạo outlet cho **ActivityIndicator** đến File's Owner.



Lập trình iOS (2014) – Bài 9. Các điều khiển trạng thái và điều chỉnh

6

## 1.3 Xây dựng ActivityIndicator



### □ Xây dựng ActivityIndicator từ mã nguồn.

- Ta thực hiện chạy ActivityIndicator với phương thức **StartAnimating**.

```
[self.activityIndicator startAnimating];
```

- Tắt ActivityIndicator với phương thức **StopAnimation**.

```
[self.activityIndicator stopAnimating];
```

Lập trình iOS (2014) – Bài 9. Các điều khiển trạng thái và điều chỉnh

7

## Nội dung



### 1. Activity Indicator

### 2. AlertView

- Mục đích sử dụng trong ứng dụng
- Khảo sát lớp UIAlertView
- Xây dựng AlertView
- Các tác vụ trong AlertView

### 3. ProgressView



## 2.1 Mục đích sử dụng trong ứng dụng



- Alert View dùng để hiện thông điệp cảnh báo cho người dùng một vấn đề nào đó, có thể là cảnh báo hết pin, cảnh báo khi xóa tập tin. Hoặc chỉ để hiện thông tin báo cho người dùng biết như thông báo khi tập tin đã được tải xong, đăng ký thành công ...

**Thông báo**

Hiển thị thông báo bằng AlertView

**Huỷ**





## 2.2 Khảo sát lớp UIAlertview

### ❑ Tạo AlerView.

- [initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:](#)

### ❑ Hiển thị AlerView

- [Show](#)

### ❑ Loại bỏ AlerView

- [dismissWithClickedButtonIndex:animated:](#)



## 2.2 Khảo sát lớp UIAlertview

### ❑ Chỉnh sửa thuộc tính của AlerView

- [delegate property](#)
- [alertViewStyle property](#)
- [title property](#)
- [message property](#)
- [visible property](#)





## 2.2 Khảo sát lớp UIAlertview

### □ Cấu hình nút của AlertView

- [addButtonWithTitle:](#)
- [numberOfButtons property](#)
- [buttonTitleAtIndex:](#)
- [textFieldAtIndex:](#)
- [cancelButtonIndex property](#)
- [firstOtherButtonIndex property](#)



## 2.3 Xây dựng AlertView

### □ Xây dựng AlertView từ Mã nguồn.

- Để hiển thị AlertView ta phải có đối tượng gọi đến AlertView thường là các sự kiện click chọn nút.
- Ta tạo một đối tượng AlertView với phương thức **initWithTitle: message:delegate:cancelButtonTitle:otherButtonTitles:**

```
UIAlertView *alerView = [[UIAlertView alloc]
    initWithTitle:@"Thông báo" message:@"Hiển thị thông báo
    bằng AlertView" delegate:nil cancelButtonTitle:@"Huỷ"
    otherButtonTitles:nil, nil];
```

- Sau đó dùng phương thức **show** để hiển thị AlertView.

```
[alerView show];
```





## 2.4 Các tác vụ trong AlerView

### Tuỳ chỉnh hành vi trên AlerView.

- [alertViewShouldEnableFirstOtherButton:](#)
- [willPresentAlertView:](#)
- [didPresentAlertView:](#)
- [alertView:willDismissWithButtonIndex:](#)
- [alertView:didDismissWithButtonIndex:](#)



## 2.4 Các tác vụ trong AlerView

### Một số tác vụ khác.

- [alertView:clickedButtonAtIndex:](#)
- [alertViewCancel:](#)



## Nội dung



1. Activity Indicator
2. AlertView
3. ProgressView
  - Mục đích sử dụng trong ứng dụng
  - Khảo sát lớp UIProgressView
  - Xây dựng ProgressView



### 3.1 Mục đích sử dụng trong ứng dụng



- Progress View sẽ hiển thị cho người dùng biết quá trình đã xử lý được bao nhiêu. Ví dụ như khi ta tải một tập tin từ trên mạng, khi dùng Progress View người dùng sẽ có thể biết được đã tải được bao nhiêu phần của tập tin.





## 3.2 Khảo sát lớp UIProgressView

- **Tạo ProgressView.**
  - [initWithProgressViewStyle:](#)
  
- **Quản lý thanh Progress**
  - [progress property](#)
  - [setProgress:animated:](#)



## 3.2 Khảo sát lớp UIProgressView

- **Cấu hình thanh Progress**
  - [progressViewStyle property](#)
  - [progressTintColor property](#)
  - [progressImage property](#)
  - [trackTintColor property](#)
  - [trackImage property](#)

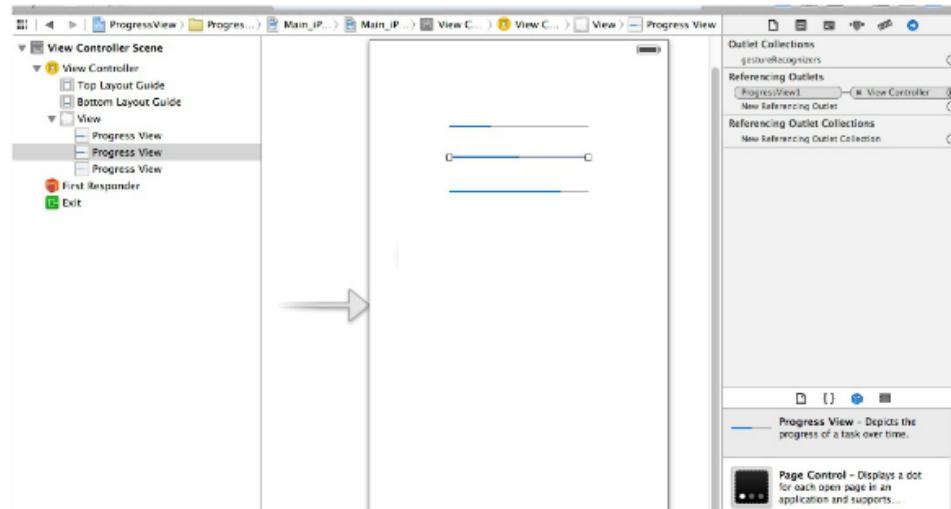




### 3.3 Xây dựng ProgressView

#### □ Xây dựng ProgressView từ User Interface.

- Thực hiện kéo đổi tượng **ProgressView** từ Object Library vào xib, sau đó thực hiện tạo outlet cho **ProgressView** đến File's Owner.



### 3.3 Xây dựng ProgressView

#### □ Xây dựng ProgressView từ mã nguồn.

- Ta có thể gán giá trị cho thanh progress với phương thức sau.

```
self.ProgressView.progress = 0.2;
```

- Ta còn có thể thực hiện animated khi gán giá trị cho progress.

```
[self.ProgressView setProgress:0.2 animated:YES];
```



## Thảo luận

