**Optional Phần 2**

**5. Phương thức chaining**

Swift cũng như các ngôn ngữ hiện đại khác, nó hỗ trợ các lập trình viên một cú pháp gọi là method chaining. Ví dụ:

extension Int {

//Phương thực hiện cộng chính số đó với một số khác

func addWithNumber(otherNum: Int) -> Int {

return self + otherNum

}

//Phương thức nhân chính số hiện tại với số khác

func multiply(orterNum: Int) -> Int {

return self \* orterNum

}

}

var numberInt: Int = 10

numberInt.addWithNumber(otherNum: 2).multiply(orterNum: 3)//Kết quả 36

Ở đây, ta viết thêm 2 phương thức add() và multiply() cho kiểu Int để cộng và nhân 1 số nguyên với 1 số nguyên khác, kết quả trả ra 1 số nguyên (cách làm này gọi là viết **Extension**, tôi sẽ đề cập ở 1 bài khác). Sau khi khai báo 2 hàm, với mỗi số nguyên kiểu Int, ta có thể gọi add() và mutiply() theo cú pháp number.add() hoặc number.multiply().

Điểm đặc biệt là do kết quả trả về vẫn là Int, nên với các công thức gồm nhiều phép tính, ta có thể gọi liên tiếp add() và multiply() mà không cần phải sử dụng đến 1 biến để lưu trữ các kết quả trung gian.

**6. Optional chaining.**

Bây giờ ta thêm một hàm divice() thực hiện phép chia với một số, hàm đó như sau:

extension Int {

func divice(otherNum: Int) -> Int? {

if otherNum == 0 {

return nil

}

return self / otherNum

}

}

Các bạn nhận thấy kết quả trả về của hàm trên không phải là kiểu Int mà nó là kiểu optional của Int (Int?). Kiểm tra nếu other có giá trị bằng 0 thì kế quả trả về nil, còn không thì nó trả về kết quả của phép chia.

Lúc này muốn sử dụng phương thức chaining, ta có 2 cách

**Cách 1**. Kiểm tra nó có nil hay không trước khi sử dụng

var result = 10.divice(otherNum: 5)

if result != nil {

print(result) //Optional(2)

}else {

print("Kết quả nil")

}

Như bạn thấy kết quả trả về là một Optional, điều này hoàn toàn hợp lý

**Cách 2.** Sử dụng phương thức chaining

var di = 10

di.divice(otherNum: 2)?.addWithNumber(otherNum: 5)//10

di.divice(otherNum: 0)?.addWithNumber(otherNum: 3) //nil

Nếu thực hiên phép chia cho một giá tri bằng 0 thì nó sẽ trả ngay về nil và không thực hiện tiếp phép cộng

**7. Ép kiểu**

Swift cũng như Java hay c#, PHP …, nó cũng là một ngôn ngữ hướng đối tượng. Vậy nên nó tính kế thừa, tức là 1 class có thể kế thừa từ 1 class khác. Như NSArray và NSDictionary đều kế thừa từ class NSOject

//Ép kiểu

var json: [String: NSObject?] =

[

"array" : NSArray(),

"dict" : NSDictionary()

"nil" : nil

]

Khi đó, nếu gọi json[“array”] ta sẽ nhận được một giá trị có kiểu NSOject. Ta cần phải ép nó về kiểu NSArray để có thể thực hiện được các phương thức dành riêng cho NSArray, như count …

let myArray = json[“array”] as! NSArray

Do vậy myArray bây giờ đã là một kiểu NSArray

*Tiếp theo: Câu lệnh if else*