**Optional trong Swift Phần 1**

**1. Kiểu Optional trong Swift**

Giống với các ngôn ngữ lập trình khác (Java, C/C++, Objective C…), Swift cũng có một giá trị đặc biệt gọi là nil (tương đương như NULL, null, NIL), nó mang ý nghĩa là không có gì. Các bạn chú ý không có gì khác với “”, 0, 0.0, [], [:] …

* nil có nghĩa là không có giá trị, còn lại là các giá trị xác định
* nil thì không chiếm dụng bộ nhớ, các giá trị khác đều chiếm bộ nhớ cho dù “” hay 0, [] …

Điều gì sẽ xảy ra khi chúng ta sử dụng các giá trị nil đó trong Swift? Swift rất nhạy cảm đối với các giá trị nil, ví dụ như thực hiện các phép toán giữa các giá trị nil với nhau có thể gây ra lỗi. Chính lý do đó, kiểu Optional ra đời.

Về bản chất , Optional vẫn là một kiểu dữ liệu bình thường, nhưng nó gắn liền với các kiểu dữ liệu có sẵn và kèm theo dấu ?. Ví dụ Int?, Float?, String? …

**var** originalInt: **Int** **=** **nil** *// Nil cannot initialize specified type 'Int'*

**var** optionalInt: **Int**? **=** **nil**

originalInt **=** 14

optionalInt **=** 15

* Dòng thứ nhất xảy ra một lỗi *// Nil cannot initialize specified type 'Int' .*

Việc chúng ta xác định biến originalInt có kiểu là Int sau đó chúng ta lại gán giá trị nil cho nó gây bất bình đẳng về kiểu. Điều này cũng giống với việc

var valueInt: Int = 6.9 //Lỗi

* Dòng thứ 2 hoàn toàn hợp lệ, biến optinalInt có kiểu là optional Int?
* Ngoài 2 điểm này thì 2 biến hoàn toàn bỉnh đăng trong các phép gán giá trị Int bình thường

Chắc hẳn đến đây các bạn đang tự đặt ra câu hỏi tại sao biến optionalInt có kiểu là optional Int? , sau đó lại có thể gán giá trị có kiểu Int? Các bạn hãy cứ kiểu kiểu Optional nó như một cái hộp, nó có thể không chứa giá trị hoặc chứa giá trị mà nó xác định trong kiểu Optional. Ví dụ kiểu Optional là Int? thì có thể chứ nil (Không chứa gì cả), hoặc giá trị Int(số nguyên).

**2. Wrap và Unwrap 1 giá trị có kiểu Optional.**

Tuy nhiên, do Optional là một kiểu đặc biệt vậy nên nó cũng được xử lý một cách đặc biêt. Đầu tiên là trong việc in ra hoặc chèn 1 biến Optional vào 1 chuỗi String.

**var** optionalInt: **Int**? **=** **nil**

**print**(optionalInt) *// nil*

"optionalInt = \(optionalInt)" *// "optionalInt = nil"*

optionalInt **=** 10

**print**(optionalInt) *// Optional(10)*

"optionalInt = \(optionalInt)" *// "optionalInt = Optional(10)"*

Như mình đã nói ở trên kiểu Optional các bạn hãy liên tưởng đến cái hộp. Mọi giá trị khác nil được đặt trong đối tượng Optional(…) – giống như việc đặt chúng trong hộp. Swift gọi việc này là wrap – hay gọi là đóng gói. Tức là không biết 1 giá trị Optional có giá trị hay không ta cứ gói chúng lại cho chắc chắn.

Nếu chúng ta muốn lấy các gí trị thực (trong trường hợp khác nil) thì Swift lại xinh ra một khái niệm unwrap, bằng cách sử dụng dấu ! đằng sau biến. Ví dụ

**var** optionalInt: **Int**? **=** **nil**

**print**(optionalInt**!**) *// fatal error: unexpectedly found nil while unwrapping an Optional value*

optionalInt **=** 10

**print**(optionalInt**!**) *// 10*

"optionalInt = \(optionalInt)" *// "optionalInt = 10"*

Nếu unwrap 1 giá trị nil sẽ xảy ra lỗi.

Các bạn hãy nhớ wrap đi với dấu ?, còn unwrap sẽ đi với dấu !

**3. Unwrap tự động**

Trong các trường hợp vẫn muốn dùng **Optional** nhưng không muốn lúc nào cũng phải **unwrap**, ta có thể dùng dấu ! trong khai báo kiểu dữ liệu, thay cho ?, việc này gọi là **unwrap tự động**, ví dụ:

**var** autoUnwrap: **Int!** **=** **nil**

**print**(autoUnwrap) *// fatal error: unexpectedly found nil while unwrapping an Optional value*

autoUnwrap **=** 10

**print**(autoUnwrap) *// 10*

"optionalInt = \(autoUnwrap)"

Tất nhiên, khi ta **unwrap tự động** 1 biến nil thì vẫn nhận được thông báo lỗi fatal error: unexpectedly found nil while unwrapping an Optional valuenhư thường, vì bản chất việc này là ẩn dấu ! mỗi lần sử dụng biến mà thôi

**4. Optional binding**

Binding là việc gán hay đọc giá trị của một biến tại một thời điểm nhất định, trong 1 hoàn cảnh cụ thể. Cú pháp đơn giản nhất của binding trong Swift là dung câu lệnh if let như sau.

optionalInt **=** **nil**

**if** **let** checkedInt **=** optionalInt {

**print**(checkedInt)

"checkedInt = \(checkedInt)"

} **else** {

**print**("nil")

"no value for checkedInt"

}

Do biến optionalInt có giá trị là nil, sau đó ta tiến hành binding biến đó vào một biến tạm checkedInt, lệnh binding sẽ trả về một giá trị false 🡺 thực hiện khối lệnh else. Ngược lại nếu nó là true(có giá trị) thì ta không cần unwrap mà có thể sử dụng luôn.

Tiếp theo : Optional phần 2