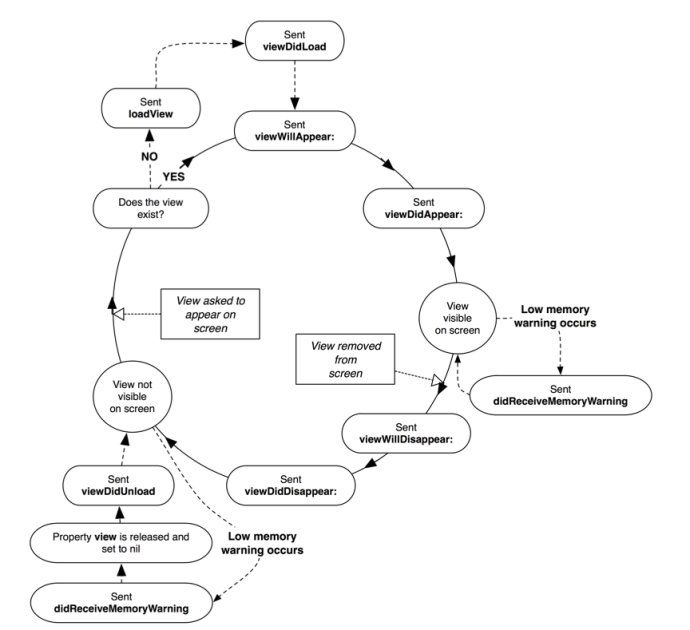
|  |  |
| --- | --- |
| **CÔNG TY …** | Họ và tên: Trần Hoài Nam  Ngày test: 23/6/2020 |

**BÀI TEST ĐẦU VÀO IOS**

**1. Liệt kê vòng đời của 1 view controller trong swift và định nghĩa cách sử dụng.**



**LoadView**

* Hàm này sẽ được gọi đầu tiên khi khởi tạo một view controller.

**viewDidLoad**

* Khi view controller đã được nạp vào bộ nhớ, thì hàm viewDidLoad được gọi.
* Chỉ được gọi một lần duy nhất trong chu kỳ sống của view đó thôi,giống hàm onCreat() trong Android
* Thường dùng để chuẩn bị data hoặc là khởi tạo các giá trị mặc định cho các object cũng như UI trên màn hình.

**viewDidUnload**

* Khi app của bạn nhận được cảnh báo từ hệ thống về trạng thái bộ nhớ đang gần hết thì hàm này sẽ được gọi
* Tại hàm này sẽ giải phóng bớt các property không cần dùng, gán nil chúng để giải phóng bộ nhớ.

**viewWillAppear**

* Hàm sẽ được gọi trước khi một view được thêm vào hệ thống view và trước animation hiển thị một view.

**viewDidAppear**

* Hàm sẽ được gọi khi một view đã được thêm vài hệ thống view và đã hiển thị lên màn hình.

**viewWillDisappear**

* Gọi khi một view đã được ẩn khỏi màn hình và amination khi ẩn view đó.

**viewDidDisappear**

* Gọi khi một view đã được ẩn khỏi màn hình.

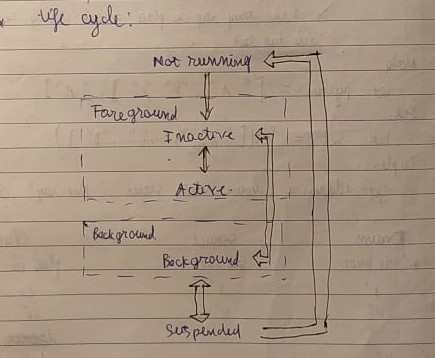
***Lưu ý khi load dữ liệu tại các hàm:***

Nếu dữ liệu là dạng tĩnh và không thay đổi nhiều, bạn nên load chúng ở phương thức viewDidlLoad.

Nếu dữ liệu là động và thay đổi thường xuyên, bạn nên load tại phương thức viewDidAppear.

Lưu ý là tại 2 phương thức, dữ liệu cần phải load theo dạng bất đồng bộ ở một thread khác để tránh ảnh hưởng đến UI.

**2. Liệt kê vòng đời của 1 ứng dụng và định nghĩa cách sử dụng.**



**Các trạng thái (Application State) :** 5 trạng thái

1. Not Running : App chưa được chạy.
2. Inactive : đã hiện lên nhưng chưa nhận bất kỳ sự kiện nào (chuyển giữa các giao diện)
3. Active : App chạy bình thường và người dùng tương tác được.
4. Background : App chạy ngầm (vd : như ấn nút home)
5. Suspended : App gần như bị đóng băng (code không chạy)

* **Not Running —**Either the application has not started yet or was running and has been terminated by the system.
* **Inactive —**An application is running in the Foreground but is not receiving any events. This could happen in case a Call or Message is received. An application could also stay in this state while in transition to a different state. In this State, we can not interact with app’s UI.
* **Active —**An application is running in the Foreground and receiving the events. This is the normal mode for the Foreground apps. The only way to go to or from the Active state is through the Inactive state. User normally interacts with UI, and can see the response/result for user actions.
* **Background —**An application is running in the background and executing the code. Freshly launching apps directly enter into In-Active state and then to Active state. Apps that are suspended, will come back to this background state, and then transition to In-Active → Active states. In addition, an application being launched directly into the background enters this state instead of the inactive state.
* **Suspended —**An application is in the background but is not executing the code. The system moves the application to this state automatically and does not notify. In case of low memory, the system may purge suspended application without notice to make free space for the foreground application. Usually after 5 secs spent in the background, apps will transition to Suspend state, but we can extend the time if app needs.

**3. Kể tên 1 số ưu điểm của ngôn ngữ lập trình Swift so với ngôn ngữ Obj-C : 6 ưu điểm.**

1. Swift ít code hơn, ngắn gọn dễ đọc.
2. Swift ít lỗi hơn.
3. Swift nhanh hơn, tốc độ thực thi đoạn mã nhanh.
4. Swift là mã nguồn mở.
5. Swift có khả năng tương tác cao, ít gây hao mòn phần cứng hơn.
6. Swift gần hơn với các platforms khác.
7. Nhu cầu tuyển dụng các lập trình viên Swift ngày càng cao.

**4. Kiểu tham chiếu và kiểu tham trị là gì? Chúng khác nhau như thế nào?**

* Kiểu tham chiếu (reference types) không được copy khi chúng được gán tới một biến hay hằng, hoặc khi chúng được truyền tới một function. Thay vì copy giá trị, một tham chiếu (reference) tới instance đang tồn tại được sử dụng.. (sử dụng trong class). === kiểm tra nếu hai các đối tượng hoàn toàn giống hệt nhau, ngay đến địa chỉ bộ nhớ lưu trữ dữ liệu. Dữ liệu được lưu trên bộ nhớ HEAP.
* Kiểu tham trị (value type) là một kiểu mà giá trị của nó được copy khi nó được gán tới một biến hay một hằng, hoặc khi nó được truyền như thể là một parameter tới function. (sử dụng trong struct). Toán tử so sánh **== .** Dữ liệu được lưu trên bộ nhớ STACK.

VD :

var a = trues

var b = a

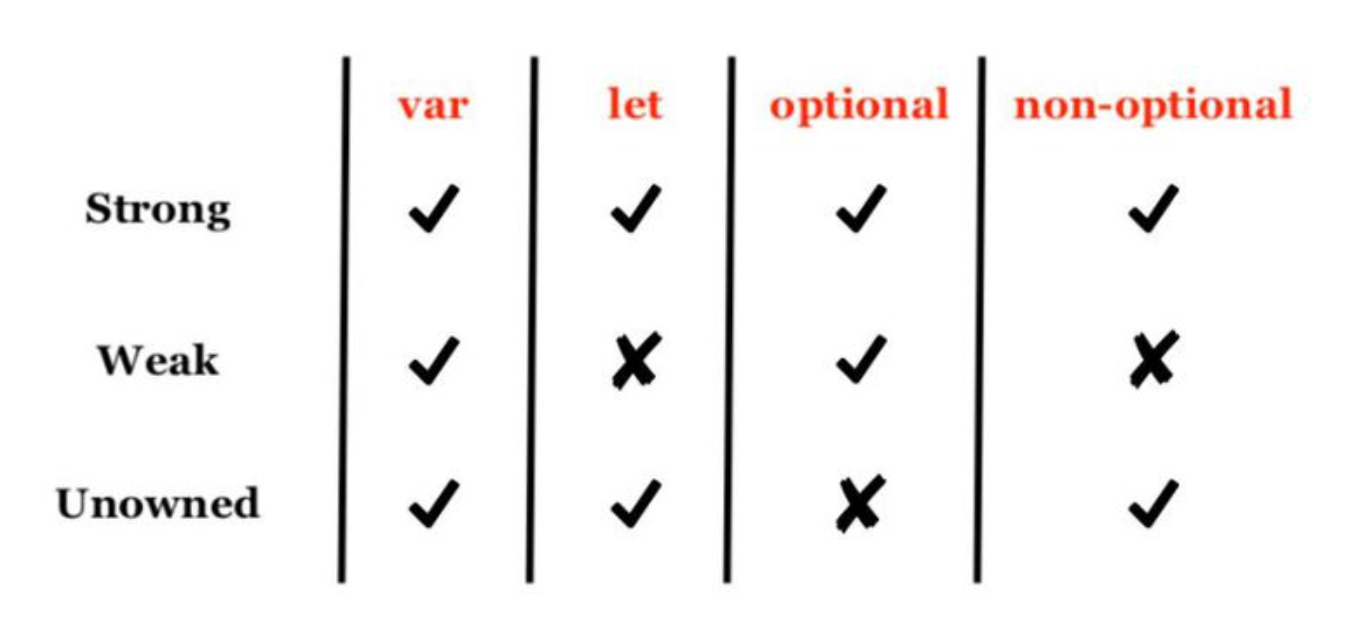
b = false

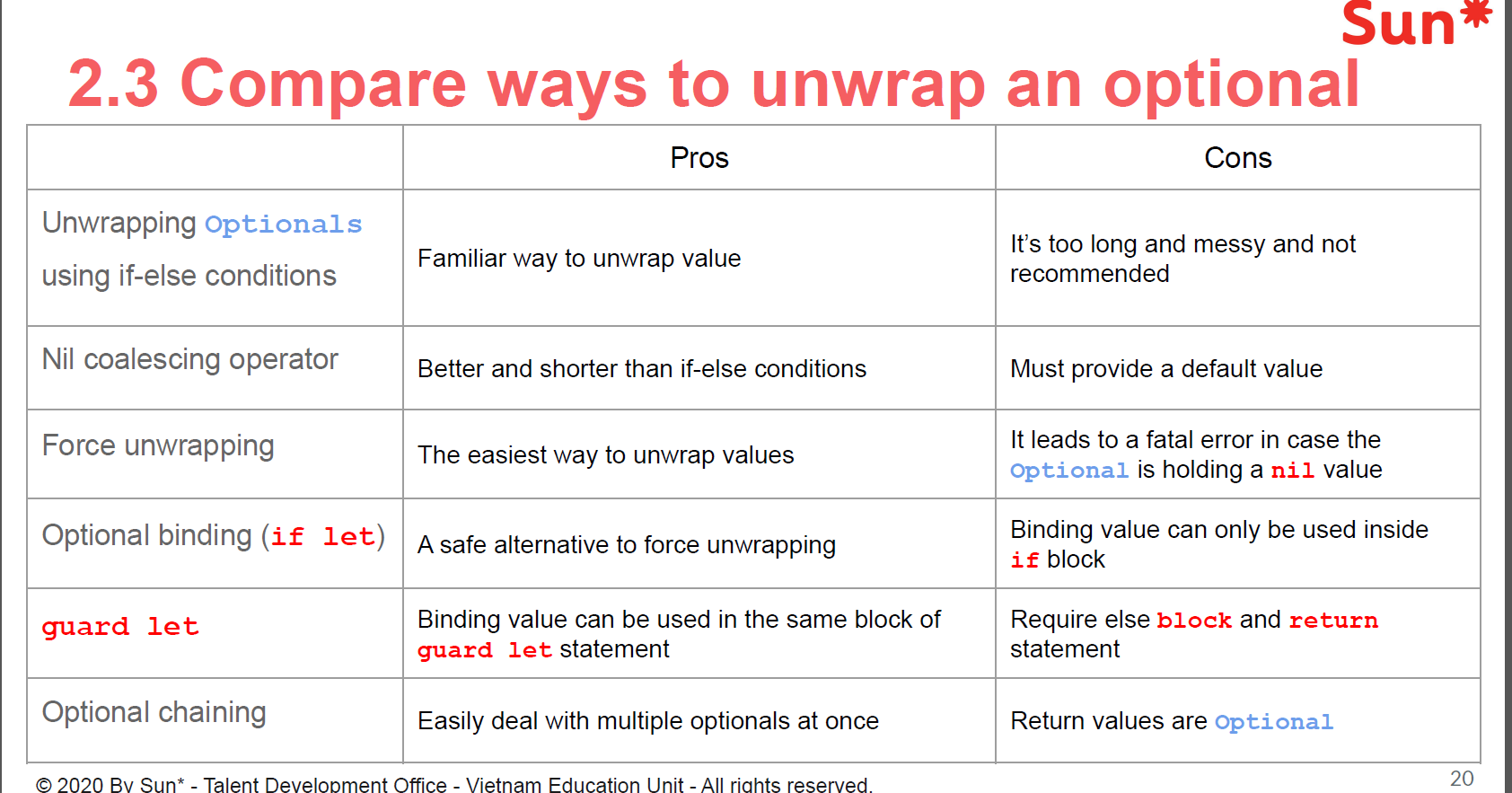
print(\(a))

* Nếu kết quả giá trị a = false thì đó gọi là tham chiếu.
* Nếu kết quả mà a = true thì đó gọi là tham trị.

Giá trị ban đầu

Bị thay đổi





**5. Struct và class khác nhau như thế nào? Khi nào dùng struct và khi nào dùng class?**

* Sự khác nhau giữa Struct và class

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Struct | Class |
| Init | | Không bắt buộc | Bắt buộc phải có |
| Kế thừa | | Không | Có |
| Viết đè (override) | | Không | Có |
| Hàm hủy | | Không | deinit |
| Thay đổi biến | let | Không | Không |
| var | Có (mutating) | Có (luôn đổi được) |
|  | | Kiểu tham trị | Kiểu tham chiếu |

* Khi nào sử dụng Struct, khi nào sử dụng class:
* Struct :
* Sử dụng Struct khi tạo đối tượng với nhiều thuộc tính có kiểu dữ liệu đơn giản, thường ở kiểu giá trị (như Int, Float, String, ...)
* Khi làm việc đa luồng. Ví dụ kết nối database được thực hiện trên một luồng song với luồng Main, việc sử dụng Struct an toàn hơn do nó có thể copy giá trị từ luồng này sang luồng khác.
* Sử dụng Struct sẽ đảm bảo không có phần nào trong code có được tham chiếu tới đối tượng của chúng ta trừ khi ta truy xuất thẳng tới chúng. Do đó, dễ quản lý, việc kiểm soát giá trị đối tượng khi bị thay đổi cũng trở nên đơn giản hơn.
* Class :
* Class được sử dụng trong trường hợp chúng ta cần những tính chất đặc biệt của nó mà Struct không có (như đã nêu ở trên)
* Khi muốn làm việc với cả trình biên dịch của Swift và Objective - C, bạn bắt buộc phải sử dụng Class. Ví dụ khi xây dựng Realm Model, phải khai báo từ khoá @objc hay và dynamic, do đó, bắt buộc phải sử dụng Class.
* Việc copy các instance là không hợp lý hoặc không cần thiết.  
  Ví dụ window hay UIViewController chỉ có 1 đối tượng active tại một thời điểm. Chúng ta sẽ luôn tạo mới một đối tượng controller để sử dụng thay vì copy giá trị chúng.

|  |  |  |
| --- | --- | --- |
|  | Stack | HEAP |
| Loại cấu trúc dữ liệu | Ngăn xếp là một cấu trúc dữ liệu tuyến tính. | Heap là một cấu trúc dữ liệu phân cấp. |
| Tốc độ truy cập | Truy cập tốc độ cao | Chậm hơn so với ngăn xếp |
| Quản lý không gian | Không gian được quản lý hiệu quả bởi hệ điều hành để bộ nhớ sẽ không bao giờ bị phân mảnh. | Heap Space không được sử dụng hiệu quả. Trí nhớ có thể trở thành bị phân mảnh như các khối bộ nhớ được cấp phát đầu tiên và sau đó được giải phóng. |
| Truy cập | Chỉ các biến cục bộ | Nó cho phép bạn truy cập các biến trên toàn cầu. |
| Giới hạn kích thước không gian | Giới hạn về kích thước ngăn xếp phụ thuộc vào hệ điều hành. | Không có giới hạn cụ thể về kích thước bộ nhớ. |
| Thay đổi kích thước | Không thể thay đổi kích thước các biến | Các biến có thể được thay đổi kích thước. |
| Cấp phát bộ nhớ | Bộ nhớ được cấp phát trong một khối liền kề. | Bộ nhớ được cấp phát theo bất kỳ thứ tự ngẫu nhiên nào. |
| Phân bổ và Giao dịch | Tự động thực hiện bằng hướng dẫn của trình biên dịch. | Nó được thực hiện thủ công bởi lập trình viên. |
| Sự phân bổ | Không yêu cầu bỏ cấp phát biến. | Phân bổ rõ ràng là cần thiết. |
| Chi phí | Ít hơn | Nhiều hơn |
| Thực hiện | Một ngăn xếp có thể được triển khai theo 3 cách đơn giản dựa trên mảng, sử dụng bộ nhớ động và dựa trên danh sách được liên kết. | Heap có thể được thực hiện bằng cách sử dụng mảng và cây. |
| Vấn đề về Main | Thiếu bộ nhớ | Phân mảnh bộ nhớ |
| Vị trí của tham chiếu | Hướng dẫn thời gian biên dịch tự động. | Thích hợp |
| Tính linh hoạt | Kích thước cố định | Có thể thay đổi kích thước |
| Thời gian truy cập | nhanh hơn | Chậm hơn |

|  |  |
| --- | --- |
| VALUE TYPE | Reference TYPE |
| -Struct  -Enum  - Tuple | -class |

**6. Hiện tượng memory leak là gì? Kể tên một vài nguyên nhân gây ra memory leak.**

* Memory leak là một phần bộ nhớ bị chiếm vĩnh viễn và không thể sử dụng lại được nữa. Chúng là rác, chúng chiếm lĩnh các khoảng trống và chúng gây ra rất nhiều vẫn đề như làm tăng dung lương bộ nhớ của ứng dụng, gây nên các hiệu ứng phụ và crashe.
* Nguyên nhân gây ra memory leak :

+ Memory leak có thể đến từ SDK hoặc framework của bên thứ 3.

+ Có thể đến từ chính Apple như CALayer hay UILabel.

+ Do quên hoặc bỏ lỡ việc giải phóng bộ nhớ.

**7. Autolayout được sử dụng để làm gì?** **Kể tên một vài phương thức autolayout bằng giao diện và bằng code**.

Auto Layout là cơ chế tự động tính toán size và vị trí của tất các view nằm trong ứng dụng của bạn, nó dựa trên một hệ thống điều kiện gọi là constraint đặt trên từng view.

#### 10. Khi sử dụng 1 closure cần phải chú ý đến điều gì?

Những vấn đề mà chúng ta cần phải cẩn thận và chú ý nhất khi sử dụng closure đó chính là vấn đề memory leak. Mặc định nếu chúng ta không sử dụng weak hoăc unowned cho biến số bên ngoài khối closure thì closure sẽ tạo một strong reference đến biến số đó và chỉ giải phóng khi closure kết thúc.

#### 11. Kể tên 5 phương thức dùng để truyền dữ liệu giữa 2 màn hình?

1. Truyền dữ liệu sử dụng Segue : kỹ thuật phổ biến nhất và thường xuyên được sử dụng để truyền tải dữ liệu thông qua 1 segue nối giữa 2 view controllers trên storyboard.
2. Truyền dữ liệu không sử dụng Segue : sử dụng trong các trường hợp :

* 2 màn hình nằm ở 2 storyboard khác nhau
* chuyển tới màn hình tạo bởi file nib
* các màn hình được khởi tạo hoàn toàn bằng code

1. Truyền dữ liệu trở lại ViewController trước đó thông qua unwind segue
2. Truyền dữ liệu thông qua Delegation.
3. Truyền dữ liệu bằng cách sử dụng Closures.

#### 12. Định nghĩa atomic và nonatomic.

Khi bạn có nhiều hơn một thread (luồng), nó có thể cho các setter và getter được gọi cùng một lúc. Điều này có nghĩa rằng getter / setter có thể bị gián đoạn bởi hoạt động khác, có thể dẫn đến dữ liệu bị hỏng.

Thuộc tính Atomic sẽ ngăn chặn điều này xảy ra, đảm bảo rằng các hoạt động được, hoặc thiết lập đang làm việc với một giá trị đầy đủ. Tuy nhiên, điều quan trọng là phải hiểu rằng đây chỉ là một khía cạnh của thread-safe-using (luồng an thoàn) và thuộc tính Atomic không đảm bảo là mã của bạn là thread-safe.

#### 13.Phân biệt giữa frame và bounds?

**+ Frame** là tọa độ và kích thước của View trong View cha của nó.

**+ Bounds** là tọa độ và kích thước của View trong hệ tọa độ của chính nó.

**14. Phân biệt sự khác nhau giữa weak và strong?**

* Weak : Khi set weak capturing, value sẽ không được giữ lại bởi closure, nó sẽ bị huỷ và set về nil. Vì vậy, những giá trị sử dụng weak capturing luôn là optional trong Swift.
* Strong : Swift sẽ luôn sử dụng giá trị mặc định là strong capturing. Điều này đồng nghĩa với việc closure sẽ được capture với bất cứ external value nào được sử dụng bên trong thân của closure, và đảm bảo rằng chúng sẽ không bao giờ bị phá huỷ.

**15. ARC là gì?**

Swift sử dụng Đếm tham chiếu tự động (ARC) để theo dõi và quản lý việc sử dụng bộ nhớ của ứng dụng. ARC tự động giải phóng bộ nhớ được sử dụng bởi các thể hiện của lớp khi các thể hiện đó không còn cần thiết nữa.

Vd :

class Person {

let name: String

init(name: String) {

self.name = name

print("\(name) is being initialized")

}

deinit {

print("\(name) is being deinitialized")

}

}

var reference1: Person?

var reference2: Person?

var reference3: Person?

reference1 = Person(name: "John Appleseed")

// Prints "John Appleseed is being initialized"

reference2 = reference1

reference3 = reference1

reference1 = nil

reference2 = nil

reference3 = nil

// Prints "John Appleseed is being deinitialized"

**16. Delegate là gì? Cơ chế hoạt động của delegate như thế nào?**

* **Delegates** là "design patterns" cho phép đối tượng gửi "message" đến đối tượng khác khi có một sự kiện đặc biệt xảy ra. Ví dụ khi chúng ta có hai object A và B để thực hiện một hành động. Một khi hành động hoàn tất, object A biết được object B đã thực hiện xong hành động và nó sẽ thực hiện hành động tương ứng tiếp theo từ kết quả của hành động trong object B, điều này có thể giải quyết bằng việc sử dụng delegates.
* **Cơ chế hoạt động : gửi dữ liệu từ VC2 -> về VC1 (Truyền ngược)**

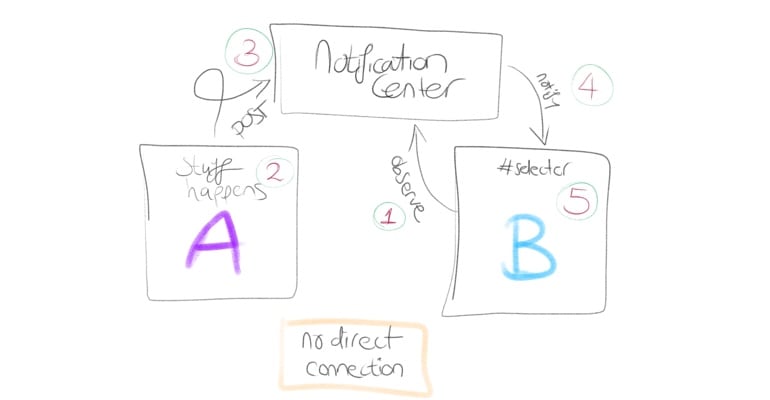
+ Bước 1: cần là tạo 1 delegate protocol cho VC 2 - nơi data được truyền đi. Khi VC 2 cần truyền dữ liệu trở lại cho VC 1, nó sẽ truyền thông qua delegate. Delegate sẽ đóng vai trò là trung gian, đứng ra nhận và truyền dữ liệu giữa 2 View Controllers.

**+** Bước 2 : Khai báo thuộc tính delegate (thuộc tính delegate cần được khai báo tham chiếu weak để tránh vòng tham chiếu mạnh, gây leak memory).

+Bước 3 : Cuối cùng, để nhận được data, VC 1 phải conform chính protocol đó.

**17. NSNotificationCenter là gì? Cơ chế hoạt động ra sao?**

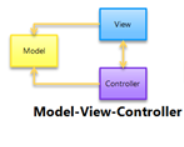
* **NSNotificationCenter**  là một cơ chế nghe-gọi mà Apple phát triển từ Observer Pattern trong thư viện Cocoa . Ý tưởng ở đây là Bên A từ một thao tác sẽ gửi thông điệp tới bên B qua một protocol, từ đố dữ liệu sẽ được truyền qua thông điệp tới bên B và bên B sẽ xử lý.
* **Cơ chế hoạt động :**



Đây là những gì sẽ xảy ra:

1. Thành phần B nói với NotificationCenter rằng họ muốn quan sát một thông báo với addObserver(\_:selector:name:object:)
2. Một cái gì đó xảy ra trong Thành phần A, như dữ liệu đến hoặc một nhiệm vụ hoàn thành
3. Thành phần A tạo thông báo và gửi thông báo đến Trung tâm thông báo
4. Trung tâm thông báo thông báo cho Thành phần B rằng một thông báo mà nó quan sát được đã được đăng bằng cách gọi một bộ chọn trên Thành phần B
5. Bộ chọn được gọi, thực hiện hiệu quả chức năng của Thành phần B

**18. Mô hình MVC là gì? Ưu và nhược điểm của mô hình MVC trong lập trình iOS?**



**MVC** là viết tắt của Model – View – Controller. Là một kiến trúc phần mềm hay mô hình thiết kế được sử dụng trong kỹ thuật phần mềm. Nói cho dễ hiểu, nó là mô hình phân bố source code thành 3 phần, gồm Model-View-Controller, mỗi thành phần có một nhiệm vụ riêng biệt và độc lập với các thành phần khác.

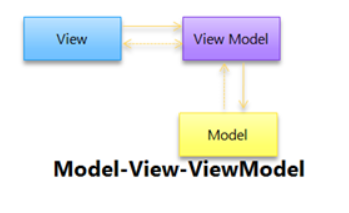
* Model : thành phần chứa tất cả các nghiệp vụ logic, phương thức xử lý, truy xuất database, đối tượng mô tả dữ liệu như các Class, hàm xử lý…
* View : Đảm nhận việc hiển thị thông tin, tương tác với người dùng, nơi chứa tất cả các đối tượng GUI như textbox, images… Hiểu một cách đơn giản, nó là tập hợp các form hoặc các file HTML.
* Controller : Giữ nhiệm vụ nhận điều hướng các yêu cầu từ người dùng và gọi đúng những phương thức xử lý chúng… Chẳng hạn thành phần này sẽ nhận request từ url và form để thao tác trực tiếp với Model.

**Ưu điểm và nhược điểm của mô hình MVC :**

**–** Ưu điểm của MVC **:** Cho thấy sự chuyên nghiệp trong lập trình và phân tích đối tượng, vì được chia các thành phần riêng biệt nên hoạt động độc lập tách biệt giúp phát triển ứng dụng nhanh hơn, đơn giản hơn và dễ nâng cấp, bảo trì hơn.

**–** Nhược điểm của MVC **:** Là mô hình làm việc rất bài bản nên với những ứng dụng nhỏ, sử dụng MVC rất tốn nhiều thời gian và gây ra nhiều phức tạp.

**19.** **Mô hình MVVM là gì? Ưu và nhược điểm của mô hình MVVM trong lập trình iOS?**



### **View Class**

1. Là nơi chứa các visual element, như window, page, user control,..
2. View tham chiếu đến View Model thông qua **DataContext** property.
3. Bạn có thể sử dụng converters để custom lại giá trị được binding từ View Model lên View. code-behind của view nên được dùng để đinh nghĩa các code login liên quan đến giao diện hoặc các chức năng liên quan đến giao diện mà bạn khó thực hiện bằng xaml

### **View Model Class**

1. implement các properties và command mà view có thể data bind.Nó thông báo cho view bất cứ state changes thông qua notification events thông qua **INotifyPropertyChanged** và  **INotifyCollectionChanged** interfaces.
2. Nó cũng có thể đùng để implement data validation thông qua **IDataErrorInfo** hoặc  **INotifyDataErrorInfo** interfaces.
3. View model có thể định nghĩa các trạng thái logic mà view thể hiện trực quan cho người dùng.

### **Model Class**

* Là các lớp có UI, đóng gói dữ liệu và logic nghiệp vụ của ứng dụng. chúng  trách nhiệm quản lý dữ liệu của ứng dụng và đảm bảo tính nhất quán và hợp lệ của nó bằng cách đóng gói các quy tắc nghiệp vụ bắt buộc và logic xác thực dữ liệu.
* model classes không reference trực tiếp đến view hoặc view model classes và không phục thuộc vào implement của view hoặc view model classes.
* model classes thường tạo tra property và collection change notification events thông qua INotifyPropertyChanged và INotifyCollectionChanged interfaces. Việc này cho phép chúng dễ dàng bị ràng buộc dữ liệu trong View. Các model classes đại diện cho các bộ sưu tập các đối tượng thường xuất phát từ **ObservableCollection<T>** class
* Các lớp mô hình thường cung cấp xác thực dữ liệu và báo cáo lỗi thông qua **IDataErrorInfo** hoặc **INotifyDataErrorInfo** interfaces.

|  |  |
| --- | --- |
| View | Kích thước nội dung bên trong |
| UIView and NSView | Không có kích thước nội dung nội tại. |
| Sliders | Chỉ xác định chiều rộng (iOS).  Xác định chiều rộng, chiều cao hoặc cả hai — tùy thuộc vào loại của thanh trượt (OS X). |
| Labels, buttons, switches, and text fields | Xác định cả chiều cao và chiều rộng. |
| Text views and image views | Kích thước nội dung bên trong có thể khác nhau. |

|  |  |  |
| --- | --- | --- |
| Auto Layout Attributes | Value | Notes |
| + Height  + Width | The size of the view. | Các thuộc tính này có thể được chỉ định các giá trị không đổi hoặc kết hợp với các giá trị khác Thuộc tính Chiều cao và Chiều rộng. Này giá trị không được âm. |
| + Top  + Bottom  + Baseline | Các giá trị tăng lên khi bạn di chuyển xuống màn hình. | These attributes can be combined only with Center Y, Top, Bottom, and Baseline attributes. |
| + Leading  + Trailing | Các giá trị tăng lên khi bạn hướng tới cạnh sau. Để có bố cục từ trái sang phải hướng, các giá trị tăng lên khi bạn di chuyển sang phải. Đối với hướng bố cục từ phải sang trái, các giá trị sẽ tăng khi bạn di chuyển sang trái. | These attributes can be combined only  with Leading, Trailing, or Center X attributes |
| + left  + right | The values increase as you move to the right. | These attributes can be combined only with Left, Right, and Center X attributes. Avoid using Left and Right attributes. Use Leading and Trailing instead. Điều này cho phép bố cục thích ứng với cách đọc của chế độ xem  hướng đi. |
| + center X  + center Y | Sự giải thích dựa trên cái khác thuộc tính trong phương trình. | Center X can be combined with Center X, Leading, Trailing, Right, and Left attributes. Center Y can be combined with Center Y, Top, Bottom, and Baseline attributes. |