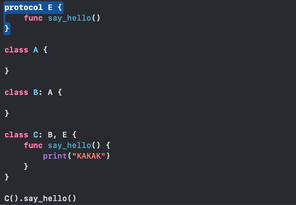
1. **Kế thừa trong Swift là đơn kế thừa hay đa thừa kế? Muốn đa thừa kế thì phải làm thế nào? Swift có cho phép kế thừa nhiều mức không ?**

Trong swift, 1 class chỉ có thể kế thừa 1 super class, nên nó là đơn kế thừa. Nhưng ta có thể sử dụng **protocol** để có thể đạt được cơ chế như đa kế thừa(**multiple inheritance**).



Như ở VD trên, ta cũng thấy được trong swift cho phép **multiple level inheritance.** Cái này hiển nhiên thoi vì với TableView và CollectionView, thì 2 View đó kế thừa UIScrollView, sau đó ta tự tạo 1 class khác để kế thừa 2 View đó.

1. **Hướng đối tượng trong Swift có tính trừu tượng hay không ?**

Hướng đối tượng trong Swift có tính trừu tượng.

1. **So sánh Class và Struct.**

Giống nhau:

* Cả Class và Struct đều có thể khai báo function để thực thi các task vụ, chúng cũng đều có thể triển khai các **protocols, và chúng cũng đều có thể tạo các extensions. Cuối cùng chúng có thể tạo các hàm init.**

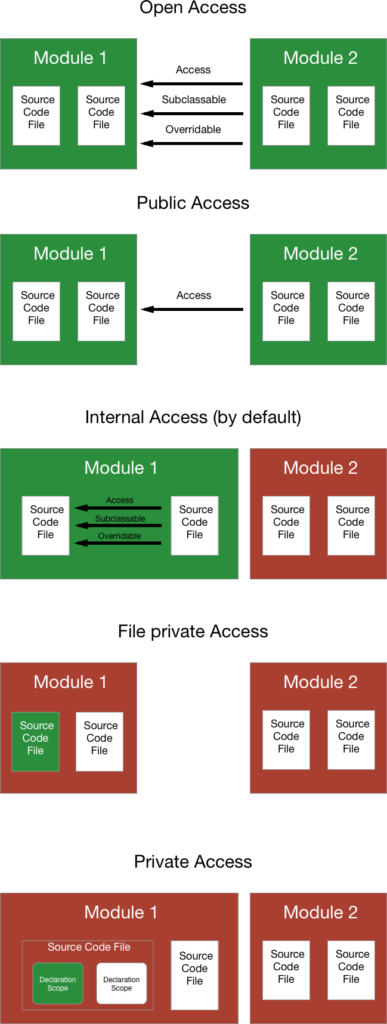
**Khác nhau:**

* Điểm khác biệt lớn nhất là **Class là reference Type còn Struct và Enum là Value Type.**
* Điểm khác biệt thứ 2 là **Class cho phép kế thừa còn Struct thì không.**

1. **Có bao nhiêu mức Access Control trong Swift? Hãy liệt kê chúng theo thứ tự giảm dần**

Trong swift có 5 mức Access Control đó là: **Open, Public, Internal, fileprivate, private**

* Với **Open Access:** Thì class đó có thể được truy cập và kế thừa từ mọi source file mà import module này, các phương thức hoặc thuộc tính của class Open có thể được **Override từ các source file đó.**
* Với **Public Access: Thì class Public đó chỉ có thể được truy cập từ các source file mà import module này, chứ ko thể kế thừa hoặc override các thuộc tính các class public.**
* Với **Internal Access:** Chỉ các entity trong Module đó mới có quyền truy cập vào class Internal
* Với **fileprivate Access:** Chỉ các entity trong file đó mới có quyền truy cập vào fileprivate class
* Với **Private Access:** Chỉ có thể được truy cập ở bên trong class khai báo nó.



1. **Quản lý bộ nhớ trong IOS thế nào**

Swift sử dụng ARC để theo dõi và quản lý bộ nhớ. ARC sẽ tự động thu hồi vùng nhớ khi các instance của class ko còn truy cập tới vùng nhớ này nữa.

1. **Property wrapper:**
2. **ARC và Non-ARC**

Khi sử dụng các project Non-ARC, người dùng sẽ phải tự mình quản lý vòng đời của bộ nhớ.

**ARC là Automatic Reference Counting**. Swift sử dụng ARC để theo dõi và quản lý bộ nhớ của ứng dụng. ARC sẽ tự động giải phóng bộ nhớ được sử dụng bởi các instance của class khi các instance này không còn được truy cập nữa. ARC là 1 thuộc tính của trình biên dịch, thay vì người dùng phải tự mình thu hồi các vùng nhớ ko sử dụng thì compiler sẽ làm nó thay ta

Ta có let a = Person(). Thì Person() là Instance còn a là biến tham chiếu đến Instance này.

ARC theo dõi số lượng tham chiếu đến vùng nhớ của Class. Khi số lượng tham chiếu bằng 0, ARC sẽ tự động thu hồi vùng nhớ này.

1. **Strong và weak là gì?**

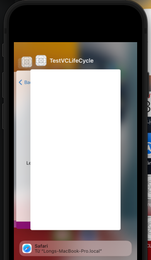
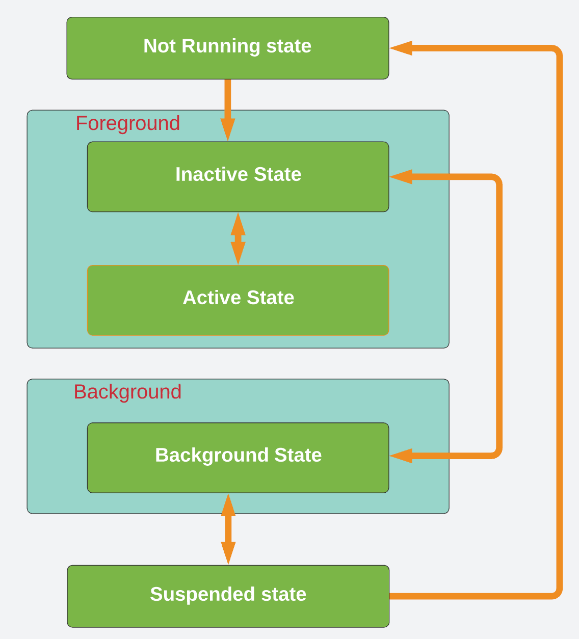
Strong Reference là 1 kiểu tham chiếu được sử dụng để giữ đối tượng trong bộ nhớ. Weak Reference là 1 kiểu tham chiếu được sử dụng khi ta không muốn giữ đối tượng trong bộ nhớ. Weak thường được sử dụng để tránh retain cycles

Cái này tự chém thêm 😊

1. **IOS Application Life Cycle**

Các trạng thái của 1 IOS Application là: **Not Running,** **InActive, Active, Background, Suspended.**

* **Not Running:** Khi chúng ta mở máy lên thì chưa có ứng dụng nào được chạy, và tất cả các App ở trạng thái **Not Running**
* **Active:** Sau khi ta nhấn vào biểu tượng App trên điện thoại chúng ta, thì App chúng ta vào trạng **thái InActive(Kiểu lúc nó mới Lauch thì ra màn hình trắng, và ko thể tương tác với UI của App vì nó chưa được load xong UI 😊)) rồi về trạng thái** **Active và chạy trên Foregroud,** lúc này người dùng có thể tương tác với giao diện của App. Khi App đã được mở sẵn nhưng chạy trên **Backgroud,** thì khi người dùng mở lại App thì App đang từ trạng Background sẽ về trạng **InActive rồi về Active**
* **In-Active:** Khi người dùng làm như ảnh bên dưới hoặc trong các trường hợp như có các cuộc gọi hoặc tin nhắn tới, thì lúc này App rơi vào trạng thái **In-Active. Lúc này App** vẫn chạy trên ForeGround, nhưng không nhận bất kì sự kiện tương tác nào của người dùng.

* **Background:** Khi người dùng nhấn home để về màn hình chính hoặc sử dụng 1 App khác thìApp hiện tại sẽ chạy dưới background nhưng vẫn thực thi code. **Khi người dùng nhấn home, App từ trạng thái Active sẽ về In-Active rồi về trạng thái Backgroud**
* **Suspended:** App chạy dưới Backgroud nhưng không thực thi code.
* **Cái này nói về trong Code:** Ta có 1 hàm sceneWillResignActive(). Hàm này sẽ được gọi khi App đang từ trạng thái Active về trạng thái In-Active

***Khi nào App chạy trên background nhưng thực thi code, và khi nào ko thực thi code ?***

* Thông thường app thực thi code và chạy trên background trong các trường hợp dưới đây: Các App Audio, Location Updates,..
* Tuy nhiên khi App không thực hiện các TH trên, nó sẽ rơi vào trạng thái suspended bởi hệ thống. Cần lưu ý rằng có những hạn chế bổ sung về những gì ứng dụng có thể thực hiện trong backgroud, tùy thuộc vào cài đặt thiết bị của người dùng, thời lượng pin và các yếu tố khác.

1. **Sự khác nhau của AppDelegate và SceneDelegate**

**Tạm bỏ**

1. **Trình bày mô hình MVC và MVVM trong swift.**

Mô hình MVC gồm 3 thành phần đó là Model, View, Controller::

* Model: Đại diện cho dữ liệu, chứa thông tin và loginc của ứng dụng
* View: Đại điện cho giao diện người dùng của ứng dụng, hiển thị dữ liệu từ model, cho phép người dùng tương tác với ứng dụng
* Controller: Đại diện cho login điều khiển của ứng dụng, là cấu nối giữa model và view, xử lý các yêu cầu từ view sau đó update view và model.

Mô hình MVVM gồm 3 thành phần đó là Model, View và ViewModel

* ViewController: Chỉ thực hiện những công việc liên quan đến giao diện: Như là lấy và hiển thị thông tin
* View Model: Nhận thông tin từ ViewController, xử lý tất cả các thông tin này, sau đó gửi lại cho VC
* Model: Đại diện cho dữ liệu của chúng ta

Flow dữ liệu của MVVM:

* Đầu tiên 1 ViewController sẽ có 1 tham chiếu tới ViewModel
* ViewController sẽ nhận 1 vài hành động của user, và sau đó sẽ gọi tới viewModel
* viewModel sẽ request APIService, sau đó trả response ngược lại cho viewModel
* Khi viewModel nhận response, ViewModel sẽ thông báo cho ViewController thông qua Binding
* View sẽ update UI với Data vừa nhận được.

1. Trình bày sự kết nối giữa các thành phần MVC và MVVM
2. Model được hiểu là như thế nào
3. Trình bày các ưu điểm mà MVVM tối ưu hơn MVC? Trong đó điểm nào quan trọng nhất
4. Định nghĩa các mẫu Design Pattern