Thứ sáu 07/01/2022

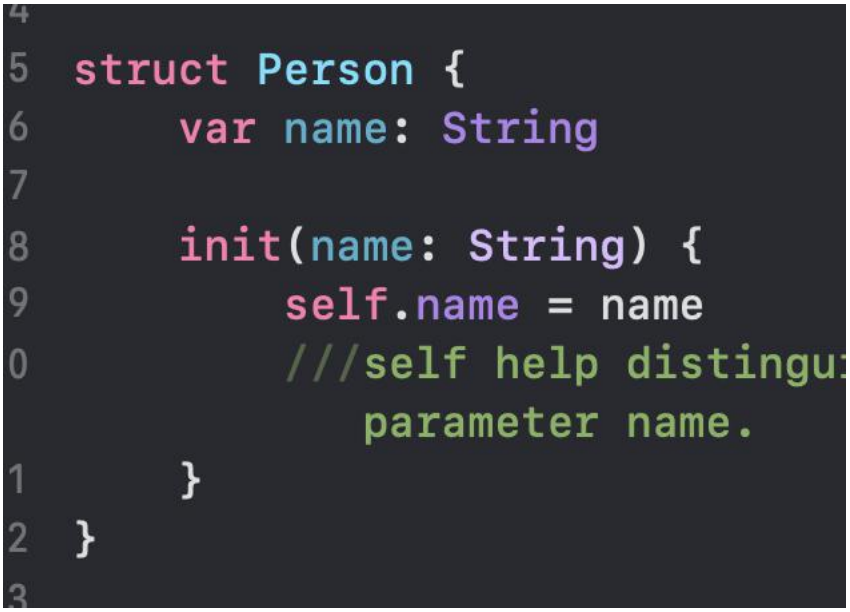
Note

I. Structs part 2

1. Initializers:

- Mọi structs mà ta tạo ra đều có sẵn 1 init mặc định cho các property mà chưa có đặt sẵn giá trị mặc định trong structs. Nếu trong structs có property nào mà có sẵn giá trị mặc định thì init mặc định sẽ cho phép ta có thể không đặt giá trị cho property đó
- Ta có thể tạo 1 init riêng cho struct để thay thế memberwise init mà swift cho. Tuy nhiên nếu ta làm thế thì swift sẽ loại bỏ init mặc định cho structs đó.
- Nếu ta muốn giữ lại init mặc định mà cũng muốn có init riêng thì ta có sử dụng extension để cung cấp cho structs init riêng mà không lo bị mất init mặc định.
- Khi khởi tạo các thành phần trong structs thì swift bắt buộc ta phải chắc chắn tất cả các property trong struct phải có giá trị trước khi mà khởi tạo kết thúc.

2. Referring to the current instances:
    - Bên trong các method của 1 structs có 1 hằng self dùng để chỉ đến thực thể structs hiện tại. Ta có thể dùng hằng này bên trong 1 init để chỉ đến property của thực thể 1 structs để cho swift phân biệt nó với 1 tham số trùng tên.
    - Ví dụ:



```
struct Person {
    var name: String

    init(name: String) {
        self.name = name
        ///self help distingu
           parameter name.
    }
}
```

    - Hằng self hầu hết dùng trong structs để cho công việc khởi tạo. Nhưng đôi lúc cũng sử dụng trong các extension để cho code ngắn gọn hơn.
3. Lazy property:
    - Lazy giống như tên gọi là 1 property mà chỉ được khởi tạo khi mà lần đầu tiên gọi nó.
    - Lazy property khác với computed property thì

thay vì tính toán xong giá trị rồi không lưu lại thì lazy property sẽ lưu lại giá trị để cho lần gọi kế tiếp.

- Lazy property trong swift bắt buộc phải sử dụng với biến do property đó bị thay đổi lúc truy cập lần đầu tiên. Do đó nên ta không sử dụng được lazy với 1 hằng struct.
- Lazy property còn có công dụng khác là giúp cho ta gọi tới self của struct. Do khi khởi tạo swift không cho phép ta gọi tới self trong lúc khởi tạo, nếu ta sử dụng lazy thì có thể chạy sau khi đã khởi tạo hết các property bên trong struct.
- Chú ý:
    + Nếu ta khởi tạo 1 biến lazy sử dụng 1 closures. Ví dụ abc: String = { “abc” }() thì ta không cần gọi self bên trong khi ta gọi các thành phần khác của struct.
    + Khởi tạo sử dụng closures không cần lo về tạo rò rỉ bộ nhớ(retain cycle) do closures không có thoát ra.

4. Static properties & methods:

- Thay vì ta tạo 1 property/method mà cho từng struct ta có thể tạo 1 property/method riêng cho tất cả các thực thể của 1 structs.
- Để gọi 1 static property/method thì ta phải gọi tên struct mà chứa nó. Ví dụ: Student.class
- Công dụng của static property/method là để làm cho các property/method mà ta hay sử dụng làm cho dễ dàng gọi ở mọi nơi mà không cần khởi tạo structs đó.

5. Access control:
    - Access Control (Quản lý truy cập) thường dùng để cho ta quản lý cách mà ta truy cập các property/method ở bên trong structs. Hầu hết thường dùng để cho khi mà có nhiều người lập trình thì có thể hiển thị property/method nào chỉ dành cho công việc bên ngoài structs và các property nào có thể read/write trực tiếp được.
    - Access Level mặc định trong swift là internal mà giới hạn mức độ truy cập ở bên trong 1 project.
    - 1 Structs chỉ có thể chứa property/methods mà có access level thấp hơn hoặc bằng structs đó.

II. Classes

1. Creating your own classes:
    - Class giống như struct mà cho phép ta tạo các kiểu riêng mà có các property và method nhưng có các điểm khác biệt. Ta sử dụng structs cho hầu hết các công việc và chỉ sử dụng class khi mà ta cần sử dụng chức năng của class.
    - Khác với struct class không có init khởi tạo mặc định do bản chất kế thừa của nó nên ta phải tạo 1 init riêng cho class.
2. Class Inheritance:
    - Class có tính năng kế thừa để 1 class có thể sử dụng để làm nền tảng xây dựng cho nhiều class khác nhau.
    - 1 class con mà kế thừa sẽ có thể sử dụng mọi method/property của class cha.
    - Khi mà ta kế thừa 1 class thì khi khởi tạo class con ta bắt buộc phải gọi khởi tạo class cha trước để đảm bảo an toàn nếu class cha làm công việc nào đó trong lúc khởi tạo.
3. Overriding methods:

- Mọi class kế thừa có thể nạp đè (override) property/method để định nghĩa lại method/property cho class con áp dụng lại theo ý muốn.
- Để thông báo nạp đè ta phải để mác override trước method/property.
- Nạp đè thường dùng để giúp ta sử dụng 1 class và sửa lại các method mà ta cần sử dụng của class đó theo ý muốn.
- Quy luật override:
    + 1 property/method chỉ nạp đè được khi mà có cùng kiểu và tên/cùng tên và cùng tham số.
    + 1 property không thể được nạp đè thành 1 stored property. Nhưng có thể được nạp đè thành 1 observed hay computed property.

4. Final classes:
    - Final class là 1 class mà không cho phép kế thừa.
    - Mác final thường dùng để làm rõ cho người khác biết không cho phép thay đổi cách áp

dụng của class này.

5. Copying objects:
    - Khác với kiểu structs mà là kiểu dữ liệu thì class là kiểu con trỏ.
    - Khi ta copy 1 struct sang 1 struct khác thì ta tạo 1 struct khác.
    - Khi ta copy 1 class thì ta đang copy con trỏ mà chỉ đến thực thể class hiện tại.
    - Do đó nếu ta mà thay đổi giá trị biến property của thực thể class hiện tại thì sẽ ảnh hưởng đến tất cả biến property của hằng/biến mà đang trỏ đến thực thể đó.
6. Deinitializers:
    - Deinit là 1 method ngược lại của init mà sử dụng trước khi 1 class bị loại bỏ.
    - Do class là kiểu con trỏ nên Swift sử dụng ARC(Automatic Reference Counting) mà thực hiện tính số hằng/biến mà đang trỏ đến thực thể class. Khi mà tất cả các con trỏ đã bị loại bỏ thì Swift sẽ gọi deinit để thực hiện các công việc khi loại bỏ class.
    - Lý do mà struct không cần deinit do struct là

kiểu dữ liệu nên nếu như thứ mà đang sở hữu struct đó bị loại bỏ thì struct sẽ bị loại bỏ theo. Còn class thì có nhiều con trỏ ở nhiều nơi khác nhau nên class cần phải còn sống trước khi tất cả các con trỏ của thực thể đó bị loại bỏ.

7. Mutability:
    - Khác với Structs thì class không có ép buộc nhiều với tính bất biến của nó.
    - Trong swift việc thay đổi 1 phần nào đó của 1 structs đồng nghĩa với tạo 1 struct mới.
    - Còn với class thì bất kỳ thành phần nào của class cũng có thể được tự do thay đổi cho dù class là 1 hằng.