1. **Đặc điểm chung của lập trình mobile là gì? Yếu tố nào quyết định tới ứng dụng mobile?**

- thiết kế, xây dựng các ứng dụng chạy trên các hệ điều hành mobile android, ios

- phát triển cơ sở dữ liêuj cho app.

- yếu tố quyết định tới ứng dụng mobile: hệ điều hành, cơ sở dữ liệu, kiến trúc app.

1. **Liệt kê các cách passing data trong iOS & Swift?**

<https://techmaster.vn/posts/36328/truyen-du-lieu-trong-ios-swift>

- passing data: truyền dữ liệu

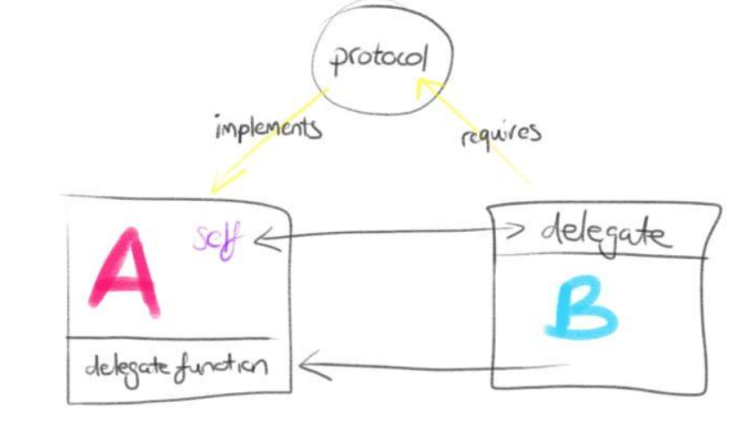
1. *passing data between viewcontrollers: bị ảnh hưởng bởi kiến trúc ứng dụng.*

+ Properties (A -> B): property là một variable of class. Bạn có thể gán giá trị cho variable và gán sang class B.



1. *Truyền data ngược lại giữa các ViewController với Properties (A <- B):*

- **Delegate**: Với delegation, một base class có thể chuyển chức năng cho một secondary class. Như vậy coder có thể implement secondary class này và phản hồi event cho base class bằng cách sử dụng protocol.



+ B1: định nghĩa một protocol bao gồm các chức năng mà chúng ta đang xử lý

+ delegate property là optional, và chúng ta sử dụng optional chaining khi gọi cookieWasBaked(\_:) function. Khi delegate là nil, thì function sẽ không được gọi.

B2: tao 1 class ke thua delegate protocol

+ UITableView class sử dụng UITableViewDelegate và UITableViewDataSource protocol.

+ CLLocationManager sử dụng CLLocationManagerDelegate để báo cáo dữ liệu liên quan đến vị trí cho ứng dụng của bạn, chặng hạn toạ độ GPS của iPhone.

+ UITextView sử dụng UITextViewDelegate để báo cho bạn về việc thay đổi trong text view.

+ Pros of delegate:

cách tiếp cận gọn nhẹ để chuyển giao các nhiệm vụ và tương tác từ class này sang class khác.

Chỉ cần một protocol để giao tiếp các yêu cầu giữa các class. Điều này làm giảm đáng kể việc ghép nối giữa các class.

Phân tách các nhiệm vụ của class tạo ra các tương tác với class phản hồi các nhiệm vụ này.

+ Một sai lầm điển hình của người mới bắt đầu là đặt ViewController làm delegate của mọi thứ. Một số lựa chọn thay thế:

- Tạo một class delegate riêng biệt, một controller, chịu trách nhiệm phản hồi các delegate function.

- Sử dụng các extension của Swift để phân tách code của bạn, tạo mỗi delegate là một extension

- Tóm tắt nhiều delegate function vào một controller (hoặc "manager") và sử dụng các closure function để phản hồi dữ liệu chi tiết mà bạn thực sự cần.

**- Closure:**

<https://fxstudio.dev/closure-trong-10-phut/#google_vignette>

Closure là một block code, có thể tách ra để tái sử dụng. Hiểu đơn giản hơn thì Closure là function, nhưng khuyết danh. Ta có thể gán Closure vào biến và sử dụng như các kiểu value khác.

+ Promise hoặc async / await là một cách tiếp cận hữu ích để quản lý nhiều async closure.

+ Sử dụng với cơ chế callback như delegate.

+ Quản lý memory khi sử dụng closure: using [weak self] or unowned.

+ Nguyen nhan cua leak memory of closure: Đó chính là khả năng Capturing Values của closure, 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.

**- NotificationCenter:**

+ It helps observing and broadcasting singles from a class to other classes.

+ 3 steps:

- observing notification: register notification with a key

- post notification: with the key

- responding to notification

+ Pros of NotificationCenter:

- passing data between classes which are not connected.

- ViewController can be not exist. It’s happened when calling Rest Api and receiving data before TableView is showed.

- Notification is many to many. Nhiều ViewController có thể phản hồi một notification, hoặc một ViewController muốn phản hồi nhiều notification.

+ Nếu bạn muốn sử dụng local traffic giữa các ViewController, thì không cần phải sử dụng NotificationCenter, đơn giản là bạn chỉ cần dùng delegate, property hoặc closure. Nhưng nếu bạn muốn liên tục và thường xuyên gửi dữ liệu từ một phần của ứng dụng này sang phần khác, NotificationCenter là một giải pháp tốt.

1. **Vì sao Swift được gọi là ngôn ngữ an toàn:**

- Swift :

+ doesn’t have undefined variables,

+ doesn’t have error of array out of bound,

+ doesn’t have integer overflow

+ handle nil, null value of variable

+ auto in manager memory.

1. **Generic là gì? Trình bày các loại cơ bản Generic trong Swift?**

<https://viblo.asia/p/generic-trong-swift-Ljy5VodkKra>

<https://fxstudio.dev/generics-trong-10-phut-swift/#google_vignette>

- Generic is data type which can represent/ replace for other data types

- **func** swapTwoValues**<**T**>(**\_ a: **inout** T, \_ b: **inout** T**)** **{}**

T is placeholder in generic.

- Generic type: type constraint,

+ type constraint:



1. **Trình bày khái niệm về Higher Order Function?**

- Higher order function has:

+ parameter can be value, object, another function.

+ return a function.

- Higher order function in swift:

+ map: convert a object to another

+ compactMap: is same map, one moreover, it handle optional element. CompactMap is used to filter nil values.

+ flatMap: = map + compactMap:

- convert value of elements

- handle element

- convert type of object

+ filter:

+ reduce: combining all elements in array to become a value.

+ sorted:

1. **Functional Programming:**

- Functional Programming: phương thức giải quyết vấn đề bằng cách chia nhỏ thành các function. Mỗi function lại có input là 1 function khác và output lại là 1 function.

1. **Trình bày về các cách mà bạn đã debug trong iOS & Swift?**

- print statements

- set breakpoints

- conditional breakpoints: when met the condition, breakpoints will be triggered.

- exception breakpoints: this can help to catch and diagnose crashes.

- visual debugging: view hierarchy of UI layout

- logging library:

- profiling tools: example instrument. Using this to analyze app performance, memory usage and energy consumption.

1. **What is the different between MTP and FTP protocol:**

|  |  |  |
| --- | --- | --- |
|  | MTP (media transfer protocol) | FTP (file transfer protocol) |
| Purpose | Using to transfer media files between devices. It’s commonly used for communication between devices: camera, phone... | Using for transfer files of all types between device and servers. |
| device | Communicating device - device | Device - server. |
| Transfer method | Usb connection | Network connection. |

1. **Manage memory:**
   1. ARC:

<https://topdev.vn/blog/moi-thu-ban-nen-biet-ve-memory-leaks-trong-iosphan-1/>

- In IOS, when a object is initialized, it will be managed by ARC.

- khi 1 object được khởi tạo, ARC sẽ cấp 1 vùng nhớ để lưu thông tin về object. Khi object không cần thiết nữa func deinit() sẽ đc gọi và ARC sẽ giải phóng vùng nhớ đó của object.

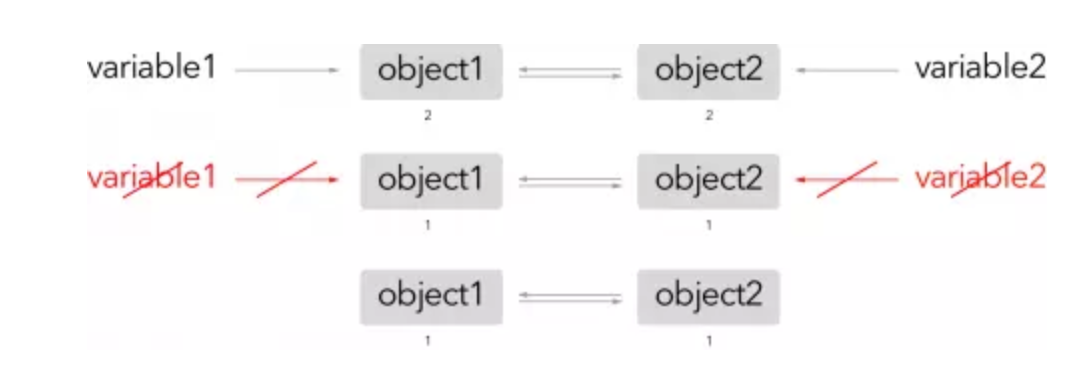
- Khi nào object được giải phóng khỏi bộ nhớ:

+ retainCount của đối tượng về 0.

+ Khi retainCount > 0 và gán object = nil.

- Reference count: is number of reference to the object A. When a object B references to the object A => reference count of A will be increased by 1. When B is deal-locate => reference count of A will be decreased by 1. A will be deal-located automatically when RC of A is 0.

- Reference cycle: là trường hợp 2 object cùng reference đến nhau => RC luôn > 0 => không thể tự giải phóng bộ nhớ => leak memory.



**- Strong reference:**

+ It keep a strong hold on an object, preventing it from being deallocted when it’s retain count still exists.

+ It can make reference cycle -> leak memory.

Scenarios when an object is considered to have a strong reference:

+ create an object with: var, let

+ assign object to a property which is declared with var in class or struct. Properties of a class, struct are declared with var -> they a strong properties.

+ passing object as fucntion parameters:

When you pass an object as a parameter to a function or method, the function's or method's parameters act as strong references within the scope of the function.

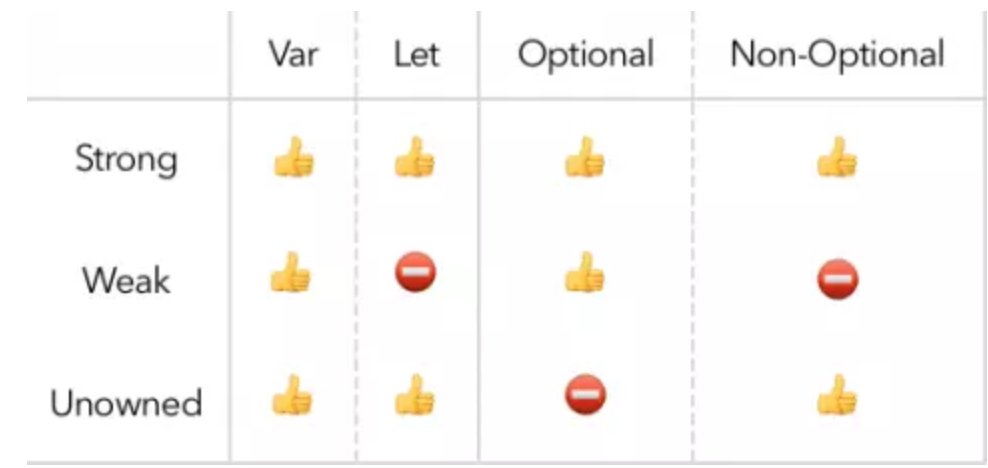
* 1. Weak reference - unowned reference:

- Weak and unowned: đều không làm tăng reference count của object được reference.

- Khác nhau:

+ weak: dùng cho các optional property

+ unowned: dùng khi các property chắc chắn khác Nil. -> accessing an unowned object after reference object is deallocated -> runtime crash.



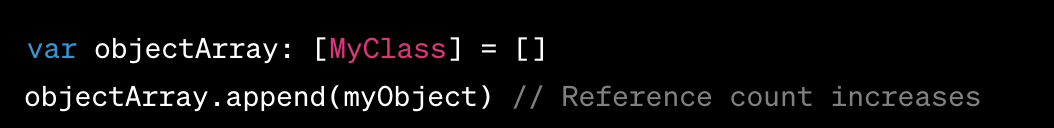
* 1. When does the reference count increase?

1. Initialization:

2. Assignment to a variable or a property:

A = B => both reference count of A and B are increase

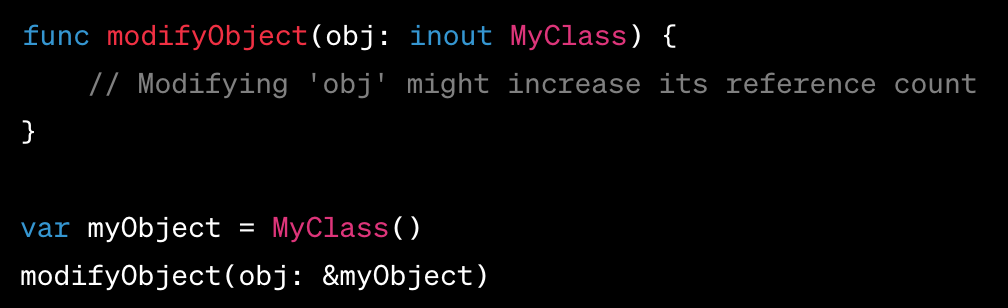
3. Adding to collection:



=> reference count of myObject +1

4. Passing a parameter with ‘inout’ keyword to function:

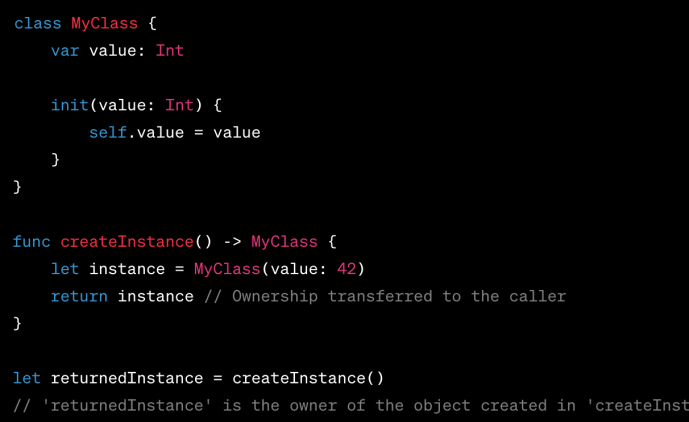
+ inout: param has inout => the changes in funtion with affect to itself in outside function.



1. Return from function with ownership:

- ownership: is a way of memory manager using ARC.

- when function return an object -> ownership of the object is transferred to caller -> the caller managers lifetime of the returned object.



1. Closure:

- There is an object. If the object is accessed in closure -> reference count +1

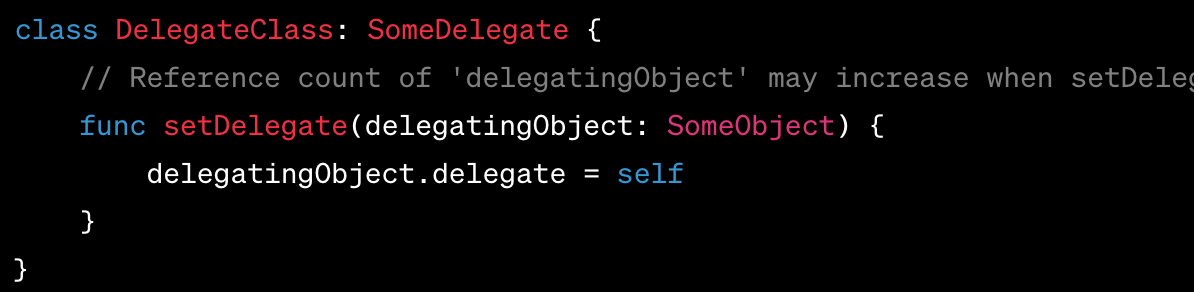
- Prevent strong reference count in closure:

+ in order to access self using: [weak self], [unowned self]

+ using local variable

1. Delegate relationship:

In delegate patterns, when an object becomes the delegate of another object, the reference count of the delegating object may increase.



* 1. Reference cycle with closure:

- Trong closure có reference tới object -> reference count của object sẽ tăng lên 1.

- object reference tới closure, closure reference tới object => reference cycle.

- Cách tránh reference count trong closure: using [weak self] or [unowned self]

1. **Property attributes in objective c:**

<https://viblo.asia/p/ios-propertyattributes-n7prv3XoRKod>

* 1. **Nhóm writability:**

- Readwrite: compiler will generate both setter and getter. Value of instance can change.

- Readonly: compiler will generate only getter. Value of instance can not be changed.

- default is readwrite.

* 1. **Setter semantics:**

- strong:

+ is newer form of retain. In bellow ios 5, we use retain.

+ will make reference cycle.

+ strong is used for parent of objects. Ex: UIviewcontroller

+ thông thường khi 1 object đc tạo ra, property của object sẽ là strong.

- retain:

+ only using in project which doesn’t use ARC.

+ is default in no ARC project.

+ creating a setter which creates a new ownership (increasing retainCount by 1)

- weak:

+ there is no retain in setter => retainCount is not created by 1, mà chỉ đơn thuần gán con trỏ trỏ đến vùng nhớ mới.

+ used to avoid retain cycle.

+ used with IBoutlet, Delegate and other objective C references.

- assign:

+ is older version of weak.

+ retainCount is not increased

+ using with primitive data types: int, float...

+ using in objective c

* 1. **Automicity:**

- relating to multithread.

- atomic:

+ is default attribute in multithread and thread safe.

+ Khi thực hiện multithread, kết quả sau khi các thread thực hiện có thể sẽ q không như mong muốn => using automic để đảm bảo sẽ trả ra 1 kết quả đúng.

+ bên trong setter/getter sẽ sử dụng lock để tránh nhiều thread truy cập đồng thời. => tốc độ xử lý sẽ chậm hơn.

- nonatomic:

+ để tránh vấn đề tốc độ chậm trên atomic

+ dùng khi không sử dụng multithread hoặc không cần thread safe.

+ it doesn’t guarantee a true result but it will always gives a result.

+ thường sẽ sử dụng nonatomic.

1. **Phân biệt viewDidLoad và viewDidAppear?**

- viewDidLoad: được gọi một lần khi lần đầu tiên đối tượng view của đối tượng UIViewController hiển thị.

- viewDidAppear: tương tự viewWillAppear, có thể được gọi nhiều lần. Method này được gọi sau khi view đã hiển thị. Nếu data là dạng tĩnh và không thay đổi nhiều, chúng ta nên load chúng ở phương thức viewDidlLoad. Tuy nhiên, nếu data của bạn là động và thay đổi thường xuyên, chúng ta nên load tại phương thức ViewDidAppear. Và lưu ý là tại 2 phương thức, data cần phải load theo dạng bất đồng bộ ở một thread khác để tránh ảnh hưởng đến UI.

1. **Thread, MultiThread in IOS:**

- Process: is a program that is running on the computer.

- Thread: is a single sequential flow of control within program.

- How do you know when the task is completed?: using closure to handle completion

- Disadvantage of multithread:

+ waste of memory: Memory is used to store the data structures and attributes related to that thread. it consumes ~1KB of memory.

+ multithread can cause deadlock.

+ priority inversion.

- In order to manage multithread, apple has:

+ manage thread manually

+ GCD (grand center dispatch)

+ NSOperationQueue.

1. ***Manage thread manually:***

- using Thread class

- Disadvantage:

+ Manage them carefully, deallocating them from memory once they have finished executing.

+ In order to performing synchronous threads needs using mutex, lock, semaphore => make more overhead to application code

+ improper thread management may cause leak memory.

1. ***GCD (Grand Center Dispatch):***

- GCD:

+ tạo queue để chứa các task. GCD sẽ sử dụng DispatchQueue.

+ đưa các block code, function, tasks vào queue và GCD sẽ quyết định luồng nào sẽ xử lý các work này.

- DispatchQueues:

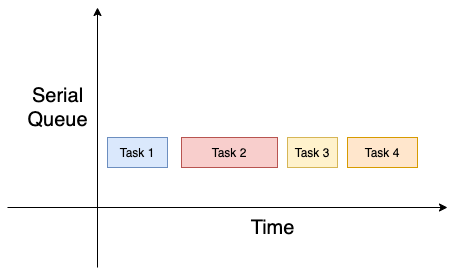
+ như 1 hàng đợi, khi cần thực hiện tasks, bạn sẽ đẩy tasks vào queue đó. Task đưa vào trước sẽ được thực hiện trước.

+ là 1 thread-safe => có thể thực hiện multithread mà không cần quan tâm sẽ gây ra race conditions.

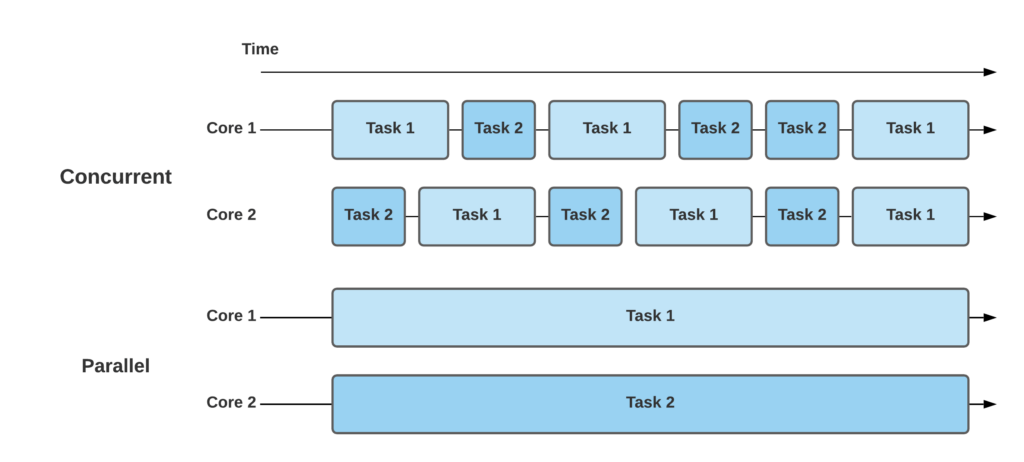
- Queue Operation: Có 2 cách vận hành queue: serial và concurrent.

<https://viblo.asia/p/ios-concurrency-phan-2-nhung-thuat-ngu-va-van-de-hay-gap-phai-trong-concurrency-4P856axWlY3>

+ Serial queue: tại 1 thời điểm nhất định chỉ có 1 task chạy. Task 1 chạy xong thì task 2 mới bắt đầu chạy. GCD sẽ quản lý thời điểm chạy của từng task.



+ Concurrent queue: các task có thể chạy đồng thời. Nhưng vẫn đảm bảo task nào vào trước sẽ đc start trước. GCD sẽ quản lý khi nào task start. Nếu 2 task cùng chạy trong 1 khoảng thời gian thì GCD sẽ phân chia task ra các luồng khác nhau. Nếu chỉ có 1 luồng thì GCD sẽ context switch để phân chia task ra.

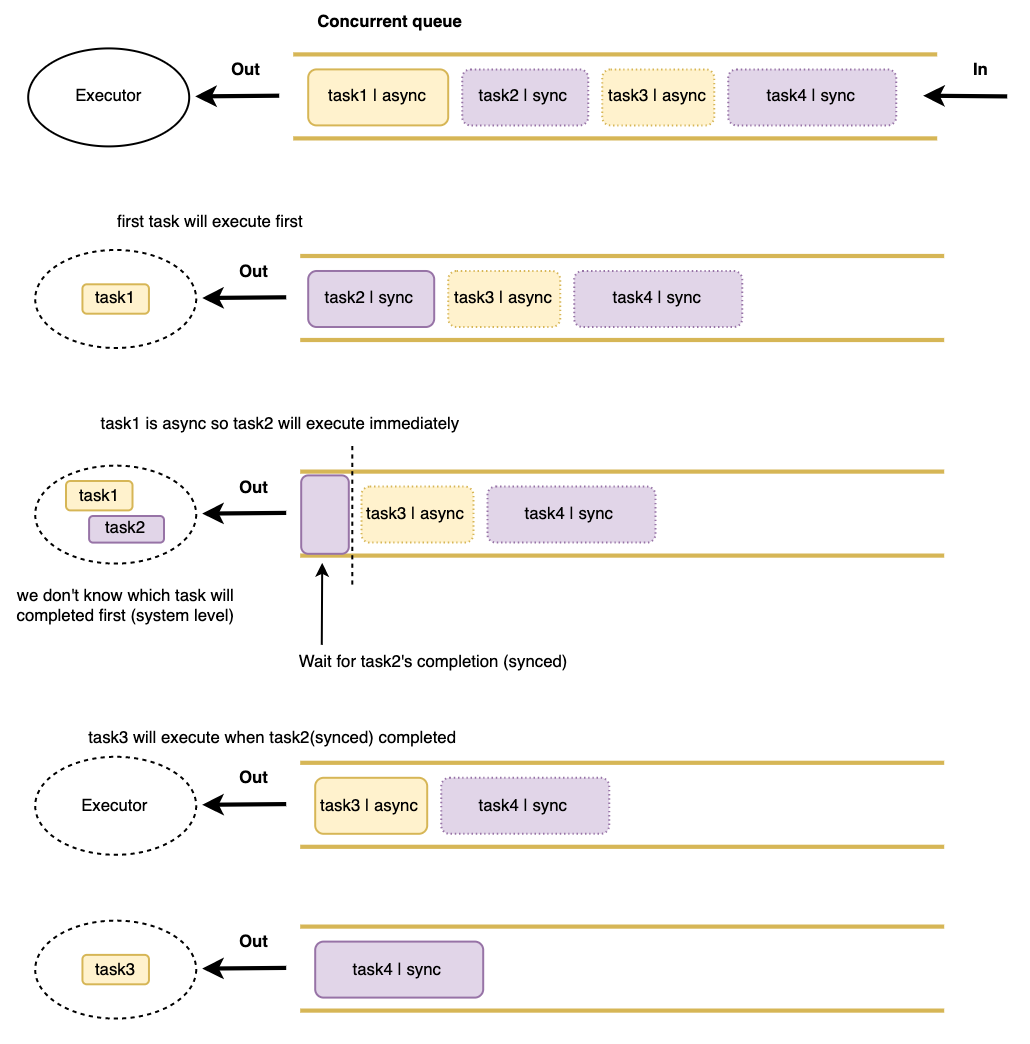


- Các task sử dụng tài nguyên dùng chung (shared resource) ko nên dùng concurrent vì sẽ gây ra sai lệch. Muốn dùng concurrent thì fai dùng thêm cơ chế đồng bộ (sync).

- Asyn and sync: dùng để mô tả cách 1 function hoạt động.

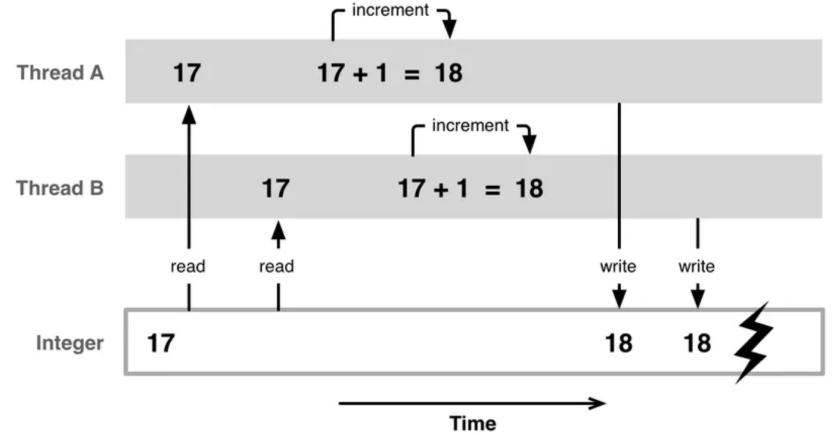
+ sync: trả về công việc sau khi function đc hoàn thành => nó sẽ block thread hiện tại đang thực thi function đó => các thread tiếp theo sẽ phải đợi function đó xong mới có thể chạy.

+ async: khi function bắt đầu được gọi, nó sẽ trả về ngay lập tức mà không cần đợi tất cả các công việc hoàn thành => async func không block thread hiện tại => các thread khác sau đó có thể chạy bình thường.

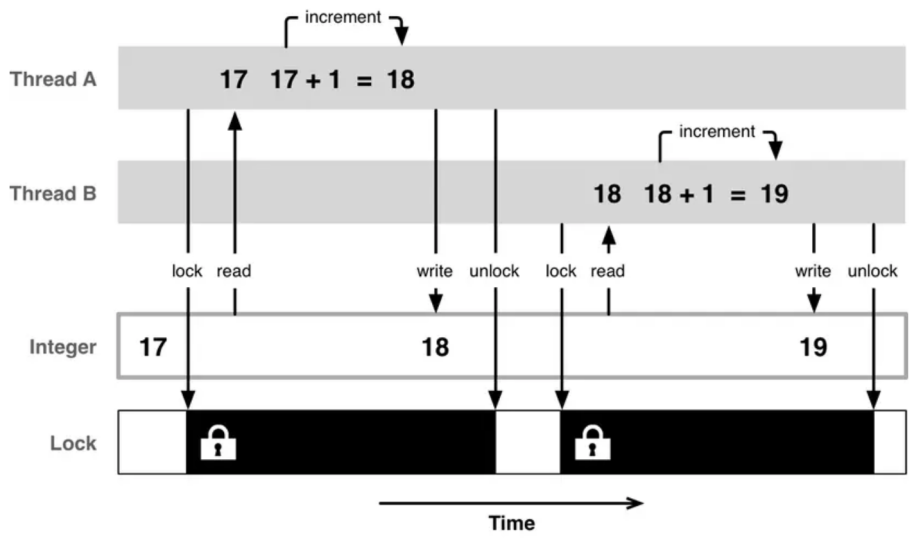


- Concurrency problems:

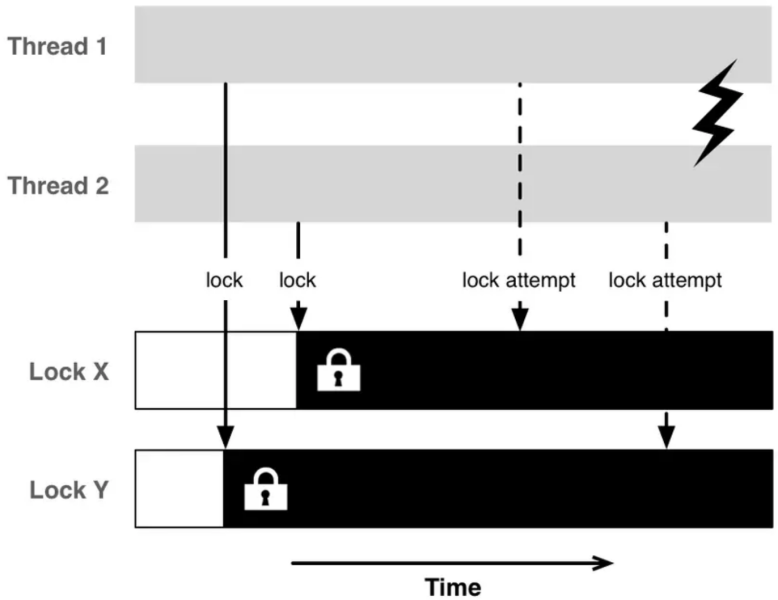
+ Race condition: là trạng thái nhiều thread đồng thời cùng truy cập shared source => gây ra conflict. Shared source có thể là: 1 thuộc tính, object, file, memory, ...



=> để giải quyết vấn đề này fai dùng 1 mutex lock. Mỗi khi 1 thread A truy suất cập đến 1 shared source, nó sẽ dùng 1 lock. Khi đó thread B sẽ fai đợi đến khi khoá đc unlock.

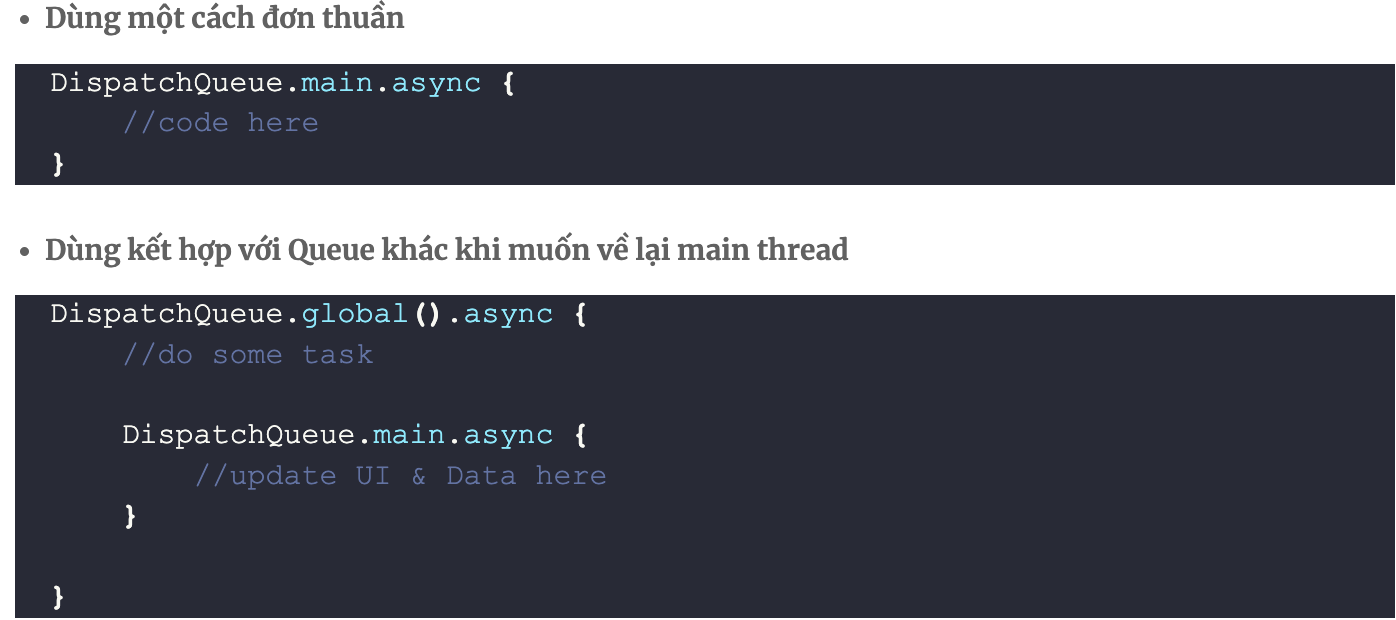


+ Deadlock: khi dùng mutex lock để giải quyết race condition, các thread khác sẽ fai đợi nhau để hoàn thành task => khi có nhiều thread đợi nhau sẽ gây ra tắc nghẽn. Đây là deadlock.



- Types of queue in GCD:

+ Main queue: là 1 serial queue, các task trong main queue sẽ đc run tại main thread. Các task liên quan tới UI thường sẽ được đưa vào main queue.



+ Global queue: là concurrent queue. Có thể thay đổi Qos. Mặc định Qos sẽ là default. Đơn giản cho việc tạo ra queue khi không muốn thay đổi nhiều tham số. Mọi thuộc tính khác và delay đều tương tự các queue custom.

DispatchQueue.global()

+ Custom queue: Do chúng ta tự tạo ra có thể là serial queue hoặc concurrent queue.

let queue = DispatchQueue(label: "test", qos: .userInitiated, attributes: .concurrent)

- Có thể custom độ ưu tiên của các task với QoS (Quality of Service). Có 5 mức độ ưu tiên:

+ UserInteractive: working vitually instantaneous. Dùng cho các task liên quan đến tương tác với user: refresh UI, performing animation. Nên được chạy trên main thread.

+ UserInitated: working nearly instantaneous, such as a few second or less. Dùng cho các task khi user chờ có kết quả ngay: opening a document, performing action of user clicking in something in UI.

+ Default

+ Utility: thường dành cho các tác vụ chạy lâu, có hiển thị chỉ số tiến trình. Ví dụ: tính toán, I/O, networking, download or importing

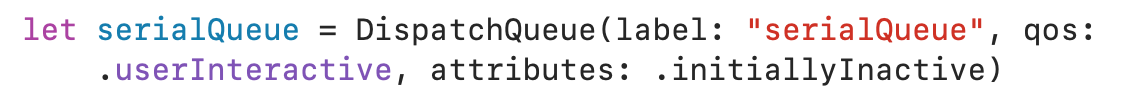
+ Background: dùng cho những task không cần kết quả ngay lập tức, tốn nhiều thời gian: prefetching data, maintenance, hoặc những task mà không bắt buộc có sự tương tác của user.

- Custome thời gian bắt đầu của queue:

Khi tạo ra 1 queue, thêm các task vào queue thì auto sẽ thực thi ngay. đôi lúc chúng ta muốn gọi queue thực thi lúc nào thì sẽ chạy lúc đó => dùng initiallyInactive or delay

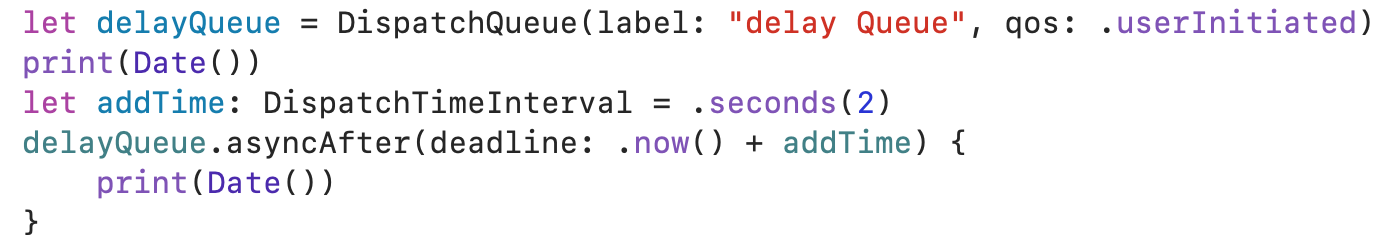
**+ InitiallyInactive**:

Queue sẽ thực thi khi gọi serialQueue.active()



**+ Delay the execution**: trì hoãn việc thực thi 1 khối lệnh (công việc) trong 1 block code sau 1 khoản thời gian đặt trước.

Sử dụng func: asyncAfter(deadline:) or syncAfter(deadline:)



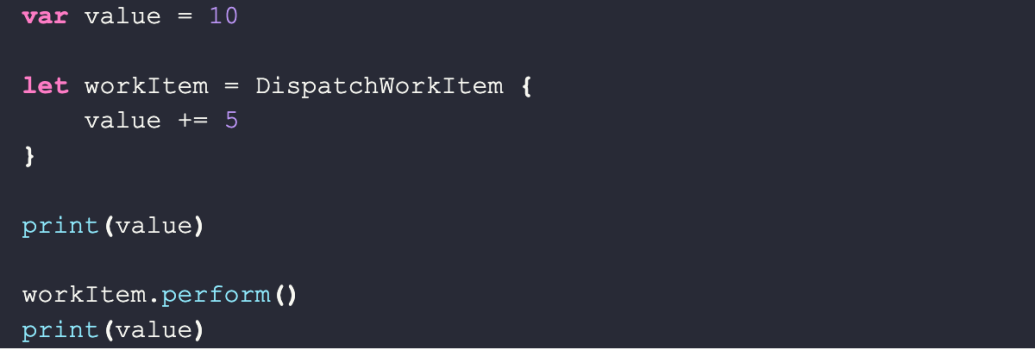
- DispatchWorkItem:

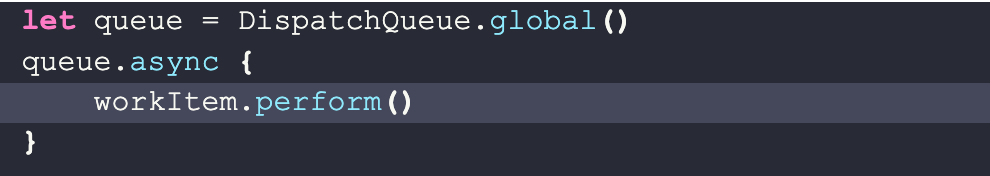
+ Để define 1 task/1 công việc/1 nhiệm vụ.

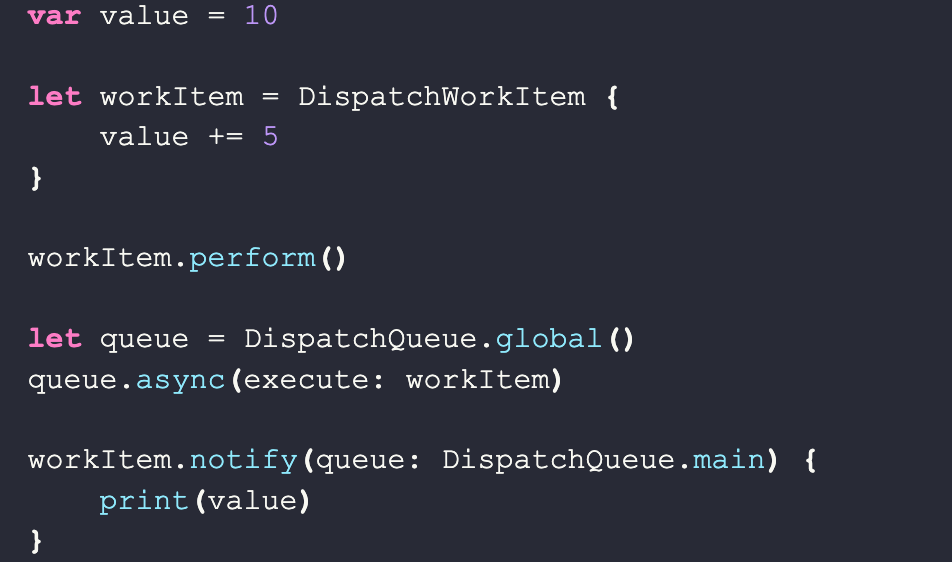
+ là 1 block code

+ Có thể được thêm vào bất kỳ queue nào

+ Khi 1 workItem được thực thi thì có thể thông báo cho queue.







1. **Dispatch Group:**

- DispatchGroup sẽ giúp chúng ta nhóm các task cần xử lý bất đồng bộ lại thành 1 nhóm và nhận được thông báo khi các task đó hoàn thành.

- Quyết định được việc chờ đợi để hoàn thành hay không.

- Đưa 1 task vào group để thực thi, sử dụng .enter() -> Thực thi task -> Thoát task ra khỏi DispatchGroup, sử dụng .leave().

- Sau khi hoàn thành tất cả các task thì Group sẽ báo 1 notify về main queue (hoặc có thể bất kỳ queue nào)

- “bao nhiêu lần .enter(), thì bấy nhiều lần .leave(). Nếu như số lần leave ít hơn enter thì DispatchGroup sẽ không bao giờ kết thúc.

- sử dụng wait() để ừng toàn bộ thread của DispatchGroup lại cho đến khi các task được thực thi xong hết.

- wait(timeout: ) để người lập trình xét thời gian cho việc hoàn thành các task.

Nếu các task chưa kịp hoàn thành thì các lệnh trong hàm wait(timeout: ) sẽ được thực thi.

1. **Runloop:**

- Mỗi NSThread có 1 NSRunloop

- NSRunloop: được tạo thành bởi 1 bộ sưu tập các input sources (keyboard, touches, ...), các observers để thông báo.

- NSRunloop được khởi tạo với 1 mode. Chỉ những sources liên kết đến mode đó mới được theo dõi và cho phép gửi các events, chỉ các observer liên kết với mode đó mới được thông báo về dữ liệu mới.

- UITrackingRunLoopMode: được thiêt lập trong lúc theo dõi các controls diễn ra. Mode này cho phép các UI event thực hiện mượt mà hơn. Ví dụ như việc drap loop hoặc các user interface tracking loops: xử lý trong mode này sẽ giới hạn các input event khác. Ví dụ khi khi bạn drag ở UITableView trong mode này, trong lúc run loop của main thread ở trong UITrackingRunLoopMode hầu hết các event background như callback của network sẽ không deliver và không có việc xử lý thêm nào cả (tức là sẽ không có lag trong lúc scrolling).

- NSRunloop khi thấy các event thuộc kiểu Timer interval -> nó sẽ gọi method mà đã được register bởi NSTimer. Điều này sẽ giúp Timer ko bị kích hoạt khi user đang scroll table or scrollView, hoặc làm bất cứ điều gì khác mà đặt runloop trong mode theo dõi các event.

1. **Compare release and drain:**
2. Release:

- is a method to decrement reference count of an object by 1.

- memory of object will be deallocated when retain count = 0.

1. Drain:
2. **Swift code revice checklist:**

- Are there any merge conflicts in the PR?

- Code is written following the coding standards/guidelines.

- No hardcoded values, use constants values.

- Prefer enum, switch over if else.

- Avoid massive view controllers, add managers, helpers, and utils instead.

- Remove empty, unused variables, functions, and imports

- If a class won’t be ever instantiated — use an enum instead.

- Group similar kinds of values under an enumeration.

- Is naming clear and consistent? The naming is important for writing self-documenting code!

- Closures should use weak self.

- Delegates should be weak.

- Check for any retain cycles.

- Check if all strings are localized.

- Instead of == nil, isEmpty should be used.

- Instead of == false, ! should be used.

- There should be no print in the code.

- Use of Any or AnyObject should be minimal. Use specific types or protocols or even better use generics.

- Heavy operation shall not be done on the main thread, the main thread is designed mainly for UI operations.

- Final classes should be used with care.

- Instead of +, string concatenation should be performed with /().

- Are there any new Xcode warnings introduced?

- Check if there’s any API provided by Apple which can make things simple.

- Static Code Analyzer (Swiftlint).

- Force unwrap should be avoided, guard let and if let should be used more often.

- Prefer static constants over computed properties.

Prefer: static let language: String = “swift”

Over: static var language: String { return “swift” }

- Use UpperCamelCase for types and protocols, and lowerCamelCase for everything else.

- Prefer composition with protocols over inheritance.

- Errors handling eg. server down, no internet connection, slow internet connection.

- Test Cases — Code Coverage of the new code.

1. **How to detect retain cycles and memory leak:**
2. Use memory graph debugger: in xcode scheme editor:

- malloc stack loggin: enable

- malloc scribble: enable

-> freed memory blocks will be filled with a known pattern. If program tries to access to ther freed memory blocks, it can detect issue.

1. Zombies intrument:

Product > Profile > Zombies

- Zombies are objects having retain count 0 still continuing to exist.

- option for ObjectiveAlloc:

+ record reference count: check

+ enable NSZombie detection: check

1. **Detect thread issues:**

<https://developer.apple.com/documentation/xcode/diagnosing-memory-thread-and-crash-issues-early>

- Thread issue is often data race problem.

- Data race: multi thread accesses to shared variables at same time without synchronization => memory corruption, crashes

1. Thread sanitizer: (TSan)

<https://medium.com/@unleased_coder/thread-sanitizer-xcode-b9c673106b77>

Edit scheme > run > diagnostics > check Thread sanitizer

- TSan: detect when multi thread accesses to the same memory without synchronization.

- TSan is only supported when running in simulator

- with data race issue: => fix: create a lockQueue using DispatchQueue => we synchronized access and ensured that only one thread at a time accessed the variable.

- Mutex: được tạo ra nhằm ngăn chặn Race condition. Dung trong C, C++

- Thread leak issue: Detects when you don’t close threads after use. => fix by using pthread\_join: pthread\_join(thread, NULL)

1. Time Profile intrument:

Product > Profile > Time Profile

1. Use Breakpoint:
2. **Singleton object:**

- Singleton object is still lives until you terminate the app.

- Singleton object is popular in objective c. It makes debugging and unit testing very hard.

- there is only one instance of singleton class.

- It provides a global point of access to its instance -> allow other objects easily obtain a reference to it.

- It’s often created as lazy initialization principle: init only when it’s first requested

1. **Next reseach:**

<https://medium.com/geekculture/threads-in-ios-gcd-nsoperation-part-1-64e460c0bdea>

<https://medium.com/@ios.deepkumar/ios-interview-questions-2023-674c7f2d15dd>

<https://kushwaha03.medium.com/top-ios-interview-question-2023-52aeeb927645>