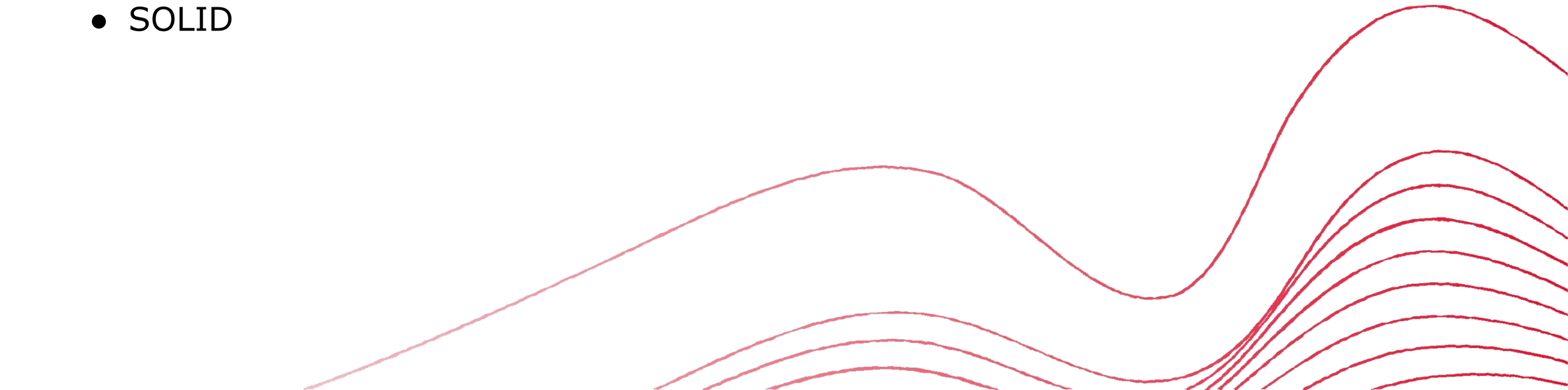
Лекция 14

Agenda

- Способы передачи данных между классами
  - Delegate
  - Closure
- Принципы написания кода: DRY, YAGNI, KISS, Бритва Оккама и другие
- Code Refactoring
- SOLID



Передача данных через delegate:

Создаем протокол, лучше всего его назвать как и класс, который его использует, только со словом Delegate в конце.
Пример:
protocol NetworkManagerADelegate: AnyObject { }

Добавляем метод, в протокол который мы хотим вызывать.

В класс из которого мы хотим передать данные, добавляем проперти delegate. Чтобы избежать утечек памяти, объявляем его как weak и optional, добавив вопросительный знак.

В методе из которого мы хотим передать данные вызываем проперти delegate и вызываем объявленный в протоколе метод.

```
protocol NetworkManagerADelegate: AnyObject {
    func dataLoaded(data: String)
}

struct NetworkManagerA {

    weak var delegate: NetworkManagerADelegate?

    func performRequest() {

        let resonce = "Some data from NetworkManagerA"
        self.delegate?.dataLoaded(data: resonce)
    }
}
```

Подробнее о делегатах и замыканиях в swift для передачи данных:
https://habr.com/ru/post/510882/

Передача данных через delegate:

В классе, в который мы хотим передать данные, необходимо указать, что этот класс реализует метод(ы) делегата, *NetworkManagerADelegate:*

```
class MediaViewController: UIViewController, NetworkManagerADelegate {

    func dataLoaded(data: String) {
        print(data)
    }
```

И в том месте, где мы уже имеем доступ к классу, из которого необходимо передать данные, мы должны указать delegate = self. Что значит, что принимающий данные класс обязуется реализовать методы этого делегата:

```
var networkManagerA = NetworkManagerA()
networkManagerA.delegate = self
networkManagerA.performRequest()
```

После чего нужно только не забыть вызвать метод, который будет отправлять данные, на примере: .performRequest.

Передача данных через closure:

Создаем в NetworkManager метод с параметром, который является замыканием.

Пример: … *completion: @escaping((String) -> ())*

```
struct NetworkManagerB {

    static let shared = NetworkManagerB()

    func performRequest(url: String, comletion: @escaping((String) -> ())) {
        let responce = "Some data from NetworkManagerB"
        comletion(responce)
    }
}
```

Во ViewController мы вызываем метод performRequest и сразу описываем в замыкании, что мы хотим сделать. Пример: *.performRequest(url: "") { responce in … }*

```
NetworkManagerB.shared.performRequest(url: "") { responce in
    print(responce)
}
```

DRY, YAGNI

DRY (Don’t Repeat Yourself / Не повторяйтесь) - не копируйте код там где это не нужно. Скажем 2 одинаковых вью контроллера с разницей в пару методов. Лучше найти способ переиспользования кода (элегантный способ). В противном случае нам придется потом вносить изменения одновременно в двух местах. И тестировать сразу два контроллера.

YAGNI (You Aren’t Gonna Need It / Вам это не понадобится) - не пишите кода который вам не нужен и не оставляйте в проекте код, который уже не нужен. Если метод выглядит как такой что не нужен, но может пригодится на будущее, его лучше удалить. Или сепарировать в каком-то файле, если речь идет о “полезных” расширениях стандартных классов.

KISS, Бритва Оккама

KISS (Keep It Simple, Stupid / Делай это проще) - старый подход, пришедший из 20 века. “Не изобретай колесо”, подразумевает что чем проще написан код тем он будет лучше работать. Тот же принцип применен в проектировке Автомате Калашникова.

Бритва Оккама - похоже на YAGNI, но более старый принцип, пришедший в том числе в программирование. Его формулировка гласит: “Не создавайте ненужных сущностей без необходимости”.

Обо всех правилах кратко в одной статье:
https://habr.com/ru/company/itelma/blog/546372/
DRY, KISS, YAGNI краткое объяснение:
https://www.youtube.com/watch?v=dz6ZsIcxoo0&ab_channel=SwiftBook.ru

Big Design Up Front, Avoid Premature Optimization

Big Design Up Front (Глобальное проектирование прежде всего) - старое как мир правило, сначала нужно спроектировать, потом делать. Можно провести аналогию с поговоркой “Семь раз отмерь...”, чем лучше спроектировано приложение, тем меньше кода придется написать.

Avoid Premature Optimization (избегайте преждевременно оптимизации) - не стоит приступать к оптимизации кода, которая предположительно пригодится. Старайтесь отталкиваться от того что в данный момент необходимо сделать.

Code refactoring

- Вынести из ViewController класса все, чему там не место: работа с сетью и работа с данными.
- Прочитать названия всех методов и убедиться, что они названы правильно: название должно отвечать на вопрос, что происходит в этом методе (в процессе скорее всего возникнут идеи, как разделить какие-то методы на 2 части, или даже объединить их).
- Все переменные, экземпляры классов и проперти тоже должны быть названы так, чтобы с первого взгляда было понятно, что это.
- Все, что можно вынести в константы, нужно вынести: все строки, а также цифровые значения. Как вариант, цифровые значения можно создавать в начале метода и в названии переменной указывать, что это за цифра.
- В конце концов, проверить, соблюдаются ли принципы написания красивого кода в вашем приложении. Речь об описанных ранее принципах и принципах SOLID.

О встроенном логгере для swift:
https://www.scalyr.com/blog/getting-started-ios-logging/
Как рефакторить swift код:
https://www.hackingwithswift.com/articles/34/before-and-after-how-to-refactor-your-swift-code

SOLID

SRP - Принцип единственной ответственности (single responsibility principle)

OCP - Принцип открытости/закрытости (open–closed principle)

LSP - Принцип подстановки Лисков (Liskov substitution principle)

ISP - Принцип разделения интерфейса (interface segregation principle)

DIP - Принцип инверсии зависимостей (dependency inversion principle)

Золотая статья, где подробно разобраны все принципы SOLID:
https://codeburst.io/solid-design-principle-using-swift-fa67443672b8
Свежая статья от Рэя о SOLID:
https://www.raywenderlich.com/21503974-solid-principles-for-ios-apps

SOLID

SRP - Single Responsibility Principle

“Каждый класс должен иметь одну и только одну причину для изменений.”

Не добавляй в класс не нужной ему функциональности.

OCP - Open and Closed Principle

“Программные сущности … должны быть открыты для расширения, но закрыты для модификации.”

Пример: Создай протокол оплаты, реализация пусть будет в разных классах. Тогда изменения каждого класса не будут в ненужном месте.

Так же: не вноси изменения в класс вью контроллер, но наследуй его и расширяй.

SOLID

LSP - Liskov Substitution Principle

“Наследующий класс должен дополнять, а не изменять базовый.”

Не изменяй класс, наследуя его. А лучше используй композицию, вместо наследования.

Пример: Не используем протокол птица с методами полета. Так как есть исключения, где он работает неполноценно. Пишем протоколы (птица, полет). Создаем протокол Летающая птица. Класс пингвин - не умеет летать, подписан только на птица. Орел подписан на летающая птица.

SOLID

ISP - Interface Segregation Principle

In simple words, Clients should not be forced to depend upon interfaces that they don't use.

«Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения»

Пиши два протокола, если не уверен, что все методы одного пригодятся конкретному классу. Нам не нужны лишние методы.

SOLID

DIP. Dependency Inversion Principle

«Зависимость на Абстракциях. Нет зависимости на что-то конкретное.»

Пусть родительский класс принимает любой объект, соответствующий протоколу, а не конкретный объект.

Все ссылки:

Подробнее о делегатах и замыканиях в swift для передачи данных:
https://habr.com/ru/post/510882/
Обо всех правилах кратко в одной статье:
https://habr.com/ru/company/itelma/blog/546372/
DRY, KISS, YAGNI краткое объяснение:
https://www.youtube.com/watch?v=dz6ZsIcxoo0&ab_channel=SwiftBook.ru
О встроенном логгере для swift:
https://www.scalyr.com/blog/getting-started-ios-logging/
Как рефакторить swift код:
https://www.hackingwithswift.com/articles/34/before-and-after-how-to-refactor-your-swift-code
Золотая статья, где подробно разобраны все принципы SOLID:
https://codeburst.io/solid-design-principle-using-swift-fa67443672b8
Свежая статья от Рэя о SOLID:
https://www.raywenderlich.com/21503974-solid-principles-for-ios-apps

Web Academy

CREATE YOUR IT FUTURE.

WITH WEB ACADEMY