Lex Debash

Follow 72 Followers About

Оформление и структура класса

В какой последовательности должны объявляться свойства и методы класса?

 Lex Debash Nov 28, 2019 · 4 min read

Введение

Читаемость кода является важным аспектом в программировании и поэтому существуют определенные стандарты, как в написании кода, так и в оформлении классов. Особенно это касается вью контроллеров, которые сами по себе являются довольно массивными объектами и поэтому навигация по ним должна быть предсказуемой, в не зависимости от того работаете ли вы со своим проектом или же со сторонним.

В этой не большой заметке я хочу расписать основные аспекты правильного оформления классов, что бы навигация по ним была интуитивно понятной, как для вас, так и для стороннего наблюдателя.

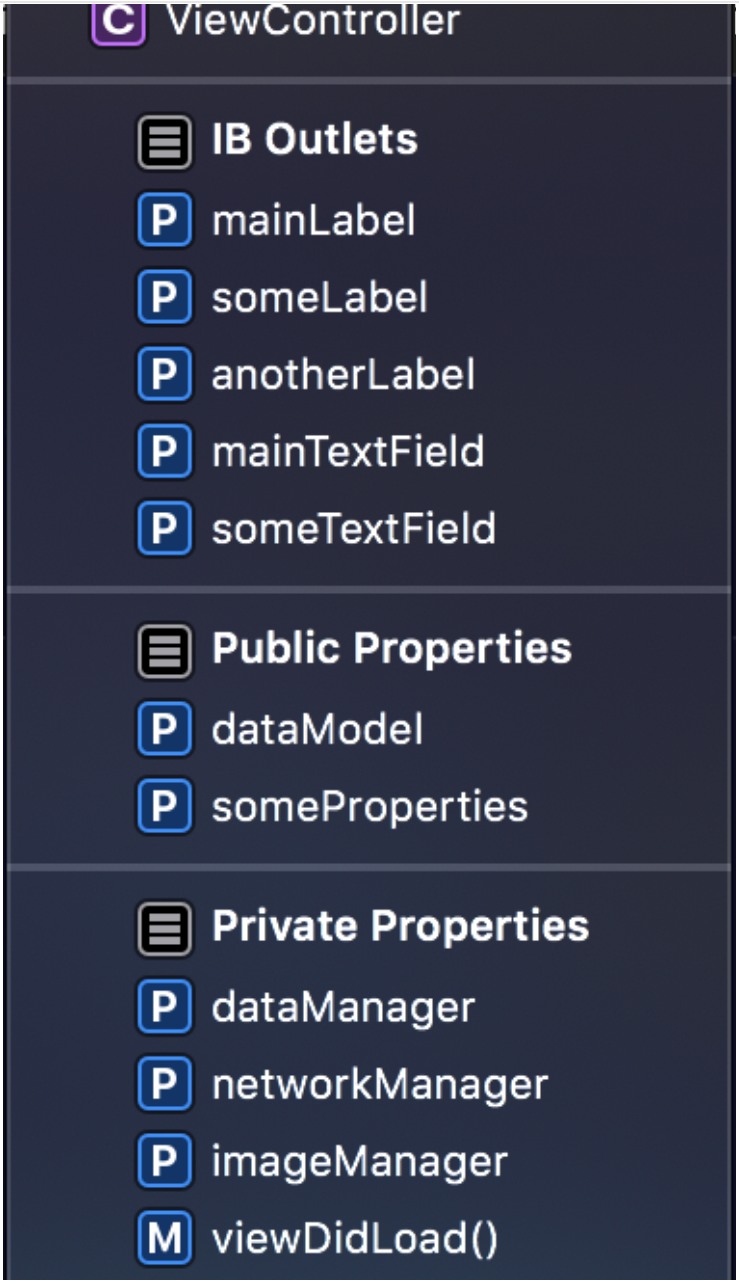
Структура класса

Класс начинается со свойств к которым относятся аутлеты, а также публичные и приватные переменные и константы:

```
    // MARK: - IB Outlets
    @IBOutlet var mainLabel: UILabel!
    @IBOutlet var someLabel: UILabel!
    @IBOutlet var anotherLabel: UILabel!

    @IBOutlet var mainTextField: UITextField!
    @IBOutlet var someTextField: UITextField!

    // MARK: - Public Properties
    var dataModel: DataModel!
    var someProperties = ""

    // MARK: - Private Properties
    private let dataManager = DataManager()
    private let networkManager = NetworkManager()
    private let imageManager = ImageManager()
```

Обратите внимание в какой последовательности объявлены свойства: сначала объявляются аутлеты, затем публичные переменные и константы, после которых уже идут приватные свойства. Так же обратите внимание на то, что аутлеты для лейблов разделяет пустая строка от аутлетов для текстовых полей. Такие разделители в виде пустых строк помогают визуально отделить логические блоки кода друг от друга, что повышает читаемость кода. Не пренебрегайте ими, но и не злоупотребляйте. Достаточно одной пустой строки, что бы отделить одно от другого.

При помощи пометки `// MARK:` класс можно поделить на разделы. Эти пометки играют роль заголовков, по которым можно быстро перемещаться. У каждого файла в проекте есть содержание, открыть которое можно, кликнув на название класса в верхней строке в которой прописан путь до класса:



В этом списке отображаются все свойства и методы класса, а заголовки позволяют легче ориентироваться по всему содержимому и перемещаться в нужное место в один клик.

В том случае, если класс не является вью контроллером, то после свойств класса объявляются инициализаторы, если в них есть необходимость:

```
    init(name: String) {
        self.name = name
    }
```

После инициализаторов объявляются переопределенные методы родительского класса. Это те методы, которые помечены ключевым словом `override`. Если же вы работаете с вью контроллером, то переопределенные методы идут сразу после свойств класса:

```
    // MARK: - Life Cycles Methods
    override func viewDidLoad() {
        super.viewDidLoad()
        setupUI()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesBegan(touches, with: event)
    }

    // MARK: - Navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    }
```

После переопределенных методов объявляются IB Actions, т.е. методы связанные с пользовательским интерфейсом:

```
    // MARK: - IB Actions
    @IBAction func someButtonPressed(_ sender: UIButton) {

    }

    @IBAction func changeColor(_ sender: UISlider) {

    }
```

вызвать из экземпляра:

```
    // MARK: - Public Methods
    func fecthData(from url: String) {

    }

    func someMethod() {

    }
```

И в самом конце класса объявляются приватные методы. Это те методы, которые используются исключительно внутри самого класса и не должны вызываться из экземпляра:

```
    // MARK: - Private Methods
    private func setupUI() {

    }

    private func setupNavigationBar() {

    }

    private func setImage() {

    }

}
```

Так же классы могу иметь расширения или `extensions` . В них можно подписывать класс под протоколы и реализовывать методы эти протоколов , что бы все они были собраны в одном месте:

```
extension ViewController: UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        code
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        code
    }
}
```

Так же можно делать расширения для отдельной группировки приватных методов или для группировки всех методов, связанных с определенной логикой класса (работа с клавиатурой, работа с уведомлениями и т.д.). Т.е. при помощи расширений можно агрегировать методы со схожими задачами. Кроме того расширения полезно использовать для распределения обязанностей между командой. В этом случае для каждого расширения создается отдельный файл, что позволяет работать над одним классом сразу нескольким участникам проекта.

В итоге структура класса должна выглядеть следующим образом:

```
// MARK: - IB Outlets

// MARK: - Public Properties

// MARK: - Private Properties

// MARK: - Override Methods

// MARK: - IB Actions

// MARK: - Public Methods

// MARK: - Private Methods
```


Общие советы

- Не храните в классе методы, которые ни как не используются. Т.е. если у вас есть метод `viewDidLoad()`, но при этом он пустой, то смело его удаляйте. Ни

- Удалите все шаблонные комментарии, которые достаются вам "из коробки" при добавлении новых классов
- Используйте в качестве разделителей логических блоков кода пустые строки, но н более одной

Имена классов должны быть емкими и отражать их суть. В проекте не должно быть классов с именем `ViewController`. Не сокращайте имена классов. В проекте не должно быть классов с именем `MainVC`.