Хрестоматия Swift паттернов. На всякий…

v.2.0

Дима Малеев и Сережа Суханов

**Оглавление**

Привет 3

История изменений 5

Prototype 6

# ****Привет****

Привет, Друг! Очень мило, что ты решил прочитать эту книгу. Ну, или просто скачал, чтобы посмотреть, что тут находится. А находятся здесь просто примеры реализации паттернов GoF для iOS. Так как примеры писаны на Swift, вероятнее всего их можно использовать и для Mac, но так как я круче программы, чем “Hello, World!” под Mac не писали – то утверждать не могу.

Что тебя ждет дальше? Невероятно, но паттерны! И вероятнее всего – грамматические ошибки. Хоть это и вторая версия книги, исправленная и доработанная – старые ошибки хоть и были исправлены (вряд ли все), но к ним добавились еще и новые. Поэтому я торжественно клянусь, что исправлю каждую ошибку, которую ты мне пришлешь на [ssuhanov@gmail.com](mailto:ssuhanov@gmail.com). Я очень надеюсь, что ты не граммар-наци и поможешь мне исправить ошибки. И, конечно же, оценишь те знания, которыми я пытался поделиться с тобой.

## Для кого эта книга?

Книга будет полезна всем Swift разработчикам, потому как книги просто полезно читать. Говорят, это улучшает память. Для начинающих разработчиков, можно будет прочитать про паттерны и примеры их реализации, в будущем про это вас спросят на собеседовании. Для более продвинутых ребят, книга может послужить небольшой напоминалкой про забытое описание паттернов. Книга однозначно не являет собой учебник по Swift, и прочитав ее вы вряд ли сможете написать второй Instagram, однако некоторые проблемы она вам все же решить поможет. Паттерны вообще хорошо знать, чтобы структурировать свои знания в голове. ☺

## Почему эта книга?

По моему глубокому личному убеждению – знания должны быть бесплатны. Просто представьте, как далеко было бы человечество, если бы у нас культура была направлена не на бесконечное зарабатывание денег, а на продвижение человечества к звездам. К сожалению, я не фармацевт, который придумал лекарство от рака, и не ученый, который придумал телепортацию. ☺

Поэтому, я попытаюсь поделиться знаниями, которые есть у меня. Что меня радует – я далеко не первый и не второй, и даже решение о написании этой книги пришло мне в голову из-за успеха Андрея Будая с его книжкой (<http://andriybuday.com/book>), которую я вам очень рекомендую почитать. Внимательный читатель увидит, что книги сами по себе очень похожи, разве что язык примеров другой (ну и сами примеры, кроме одного).

## Распространение книги

Книга бесплатна. ☺

Глупо писать другое, особенно если вы прочитали предыдущий пункт. Распространяться книга будет путем скачивания откуда угодно. Я не очень знаю, какие лицензии в ходу, да и в данном случае мне откровенно все равно. Потому, просто пару правил:

1. Книга не должна продаваться. Книга может только бесплатно распространяться в любом виде, но за просто так. Совсем.
2. Книгу можно перепечатывать, копировать себе в блог, отсылать голубиной почтой и даже менять имя автора. Если вспомните вашего покорного слугу – респект вам и уважуха, если нет – это тоже отлично, наверное, у вас была причина.
3. Естественно, автор не несет ответственности за те знания, которые вы тут получили. ☺ Ну серьезно – делайте добро, вы ж не политики.
4. Вы можете дописывать книгу, изменять в любом месте, но тогда будьте добры, меняйте автора.
5. Если вы поменяли контент книги, даже одну буковку – это уже ваша книга. ☺ Наслаждайтесь! Вы стали автором! В этой второй версии книги именно так и произошло. ☺
6. Кстати да, менять можно все, кроме этих простых правил.
7. После прочтения книги задумайтесь, какими знаниями можете поделиться вы. Вероятнее всего, вы можете рассказать уникальные и интересные вещи, которые поменяют что-то в этом мире! Пользуйтесь! Это ваша суперсила!

## Напутственное слово

Читайте.

# ****История изменений****

Книга же не бумажная, поэтому имеет возможность эволюционировать и меняться!

v.1.0 – базовый контент книги

v.1.1 – «подкрашен» код, немного пофиксаны ошибки, книга выложена в опен сорс

v.2.0 – исправлены ошибки в описаниях, примеры кода переписаны с Objective-C на Swift. Книга переведена на английский и украинский языки.

# ****Prototype****

Прототип – один из самых простых паттернов, который позволяет нам получить точную копию необходимого объекта. То есть – использовать как прототип для нового объекта.

**Когда использовать:**

1. У нас есть семейство схожих объектов, разница между которыми только в состоянии их полей.
2. Чтобы создать объект вам надо пройти через огонь, воду и медные трубы. Особенно если этот объект состоит из еще одной кучи объектов, многие из которых для заполнения требуют подгрузки данных из базы, веб-сервисов и подобных источников. Часто легче скопировать объект и поменять в нем несколько полей.
3. Да и в принципе, нам особо и не важно, как создается объект. Ну есть и есть.
4. Нам страшно лень писать иерархию фабрик (читай дальше), которые будут инкапсулировать всю противную работу создания объекта.

Да, и есть еще частое заблуждение (вероятнее всего из названия), что прототип – это архетип, который никогда не должен использоваться, и служит только для создания себе подобных объектов. Хотя, прототип, как и архетип – тоже достаточно популярный кейс. Собственно, нам ничего не мешает делать прототипом любой объект, который у нас в подчинении.

**Поверхностное и глубокое копирование**

Тут нет особой разницы с другими языками программирования. Есть указатель, есть значение в куче.

Поверхностное копирование – это просто создание нового на те же самые байты в куче. То есть – в результате мы можем получить два объекта, которые указывают на одно и то же значение.

К примеру создадим объект:

public class Person {

public var name: String

public var surname: String

public var age: Int

public init(name: String,

surname: String,

age: Int) {

self.name = name

self.surname = surname

self.age = age

}

}

А теперь давайте просто создадим два объекта и посмотрим что же получится:

let firstPerson = Person(name: "Adam",

surname: "Adams",

age: 32)

print("First Person:\(firstPerson.name) \(firstPerson.surname)")

let secondPerson = firstPerson

secondPerson.name = "Bobby"

print("Second Person: \(secondPerson.name) \(secondPerson.surname)")

print("First Person: \(firstPerson.name) \(firstPerson.surname)")

Как видим лог достаточно ожидаемый:

**First Person: Adam Adams**

**Second Person: Bobby Adams**

**First Person: Bobby Adams**

Заметьте, что хоть и меняли мы имя для secondPerson, но и у firstPerson имя поменялось. Просто потому что мы создали два указателя на один и тот же объект.

Для таких задач, стоит использовать глубокое копирование, которое в Swift сделано в принципе очень похоже как и в .NET:

Для этого надо реализовать протокол NSCopying, и перегрузить:

public func copy(with: NSZone? = nil) -> Any

Расширим наш класс:

extension Person: NSCopying {

public func copy(with: NSZone? = nil) -> Any {

let copy = Person(name: self.name,

surname: self.name,

age: self.age)

return copy

}

}

И немного изменим код нашего тестового приложения:

let firstPerson = Person(name: "Adam",

surname: "Adams",

age: 32)

print("First Person:\(firstPerson.name) \(firstPerson.surname)")

let secondPerson = firstPerson.copy() as! Person

secondPerson.name = "Bobby"

print("Second Person: \(secondPerson.name) \(secondPerson.surname)")

print("First Person: \(firstPerson.name) \(firstPerson.surname)")

Ну и, естественно, лог:

**First Person: Adam Adams**

**Second Person: Bobby Adam**

**First Person: Adam Adams**

Как видим, мы в результате получили два независимых объекта, один из которых сделан по подобию первого.

[Код примера](https://github.com/ssuhanov/ios_patterns_2/tree/master/PrototypePattern.playground).