**Структуры. Enum. Protocol**

* Использование структур
* Что такое enum
* Что такое и для чего нужны протоколы

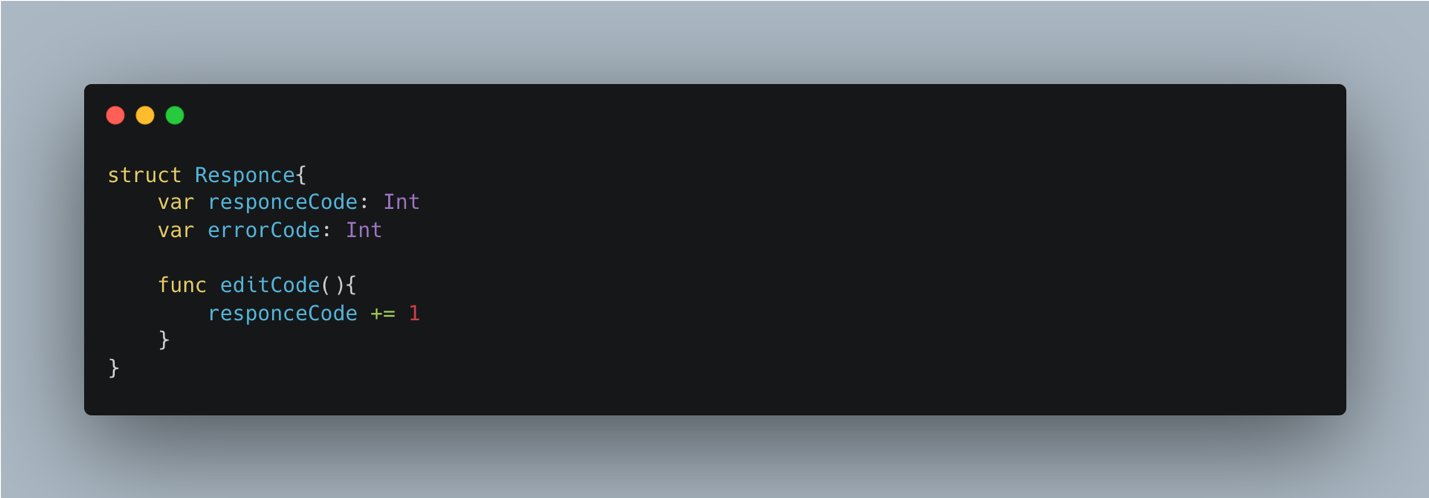
**Структуры**

На прошлом уроке мы более детально разобрали классы и их инициализаторы, давайте в этом рассмотрим структуры и как они используются на практике.

Кардинальных отличий от классов у структур два:

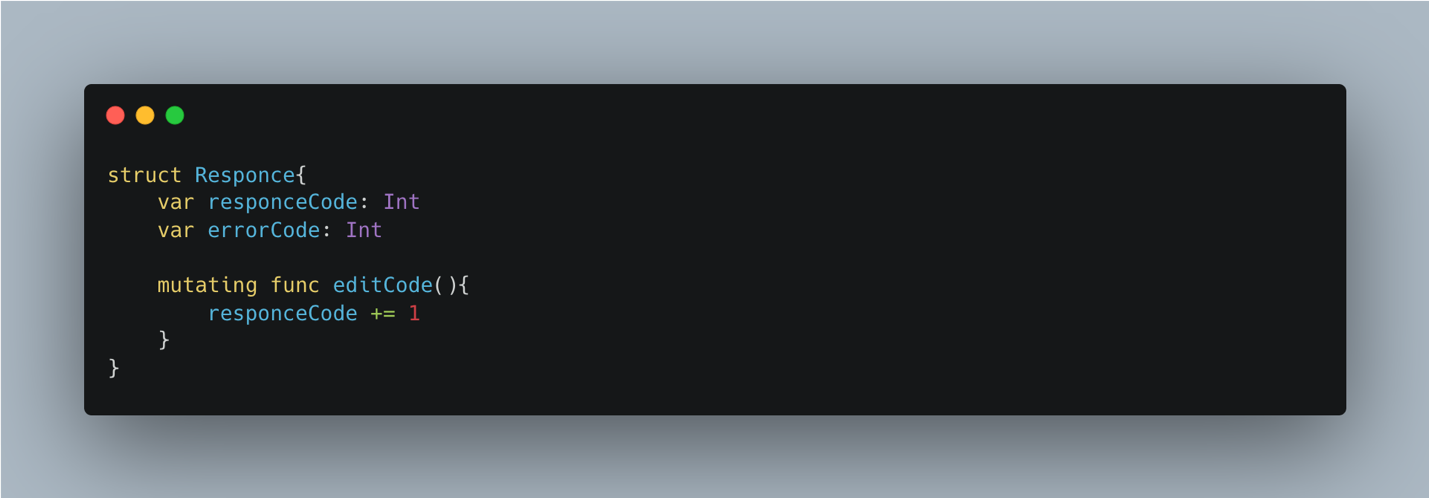
1. Структуры являются **value type**
2. Структура имеет встроенный инициализатор

Этот список можно дополнить еще тем, что нельзя менять свойства структуры внутри самой структуры как в классах, рассмотрим на примере:



Тут мы пытаемся изменить свойство responceCode, в таком случае xcode выдаст нам ошибку (Left side of mutating operator isn't mutable: 'self' is immutable).

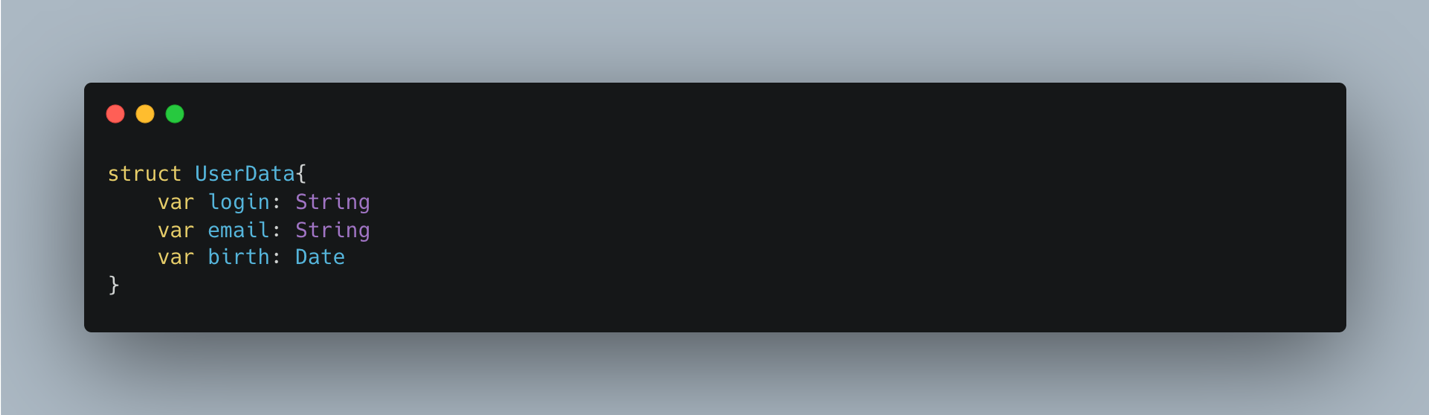
Но если мы все же хотим, что бы наша структура имела такую возможность, то до объявления функции надо добавить ключевое слово **mutating**



На практике структуры отлично подходят для создания моделей данных. При помощи структуры мы будем описывать передавать данные между классами или функциями, а также преобразовывать json файлы или ответы от сервера.

Разберем простой пример, допустим, мы отправляем запрос на сервер заранее зная, какие параметры от нас ожидают. Сервер от наш ждет имя пользователя, почту и к примеру дату рождения, в таком случае мы можем написать простой класс, внутри которого будут нужные нам методы (функции отправляющий и получающий запросы), а в качестве аргумента будем передавать структуру.

Для начала сделаем модель данных, проще говоря структуру с нужными полями (свойствами/переменными)

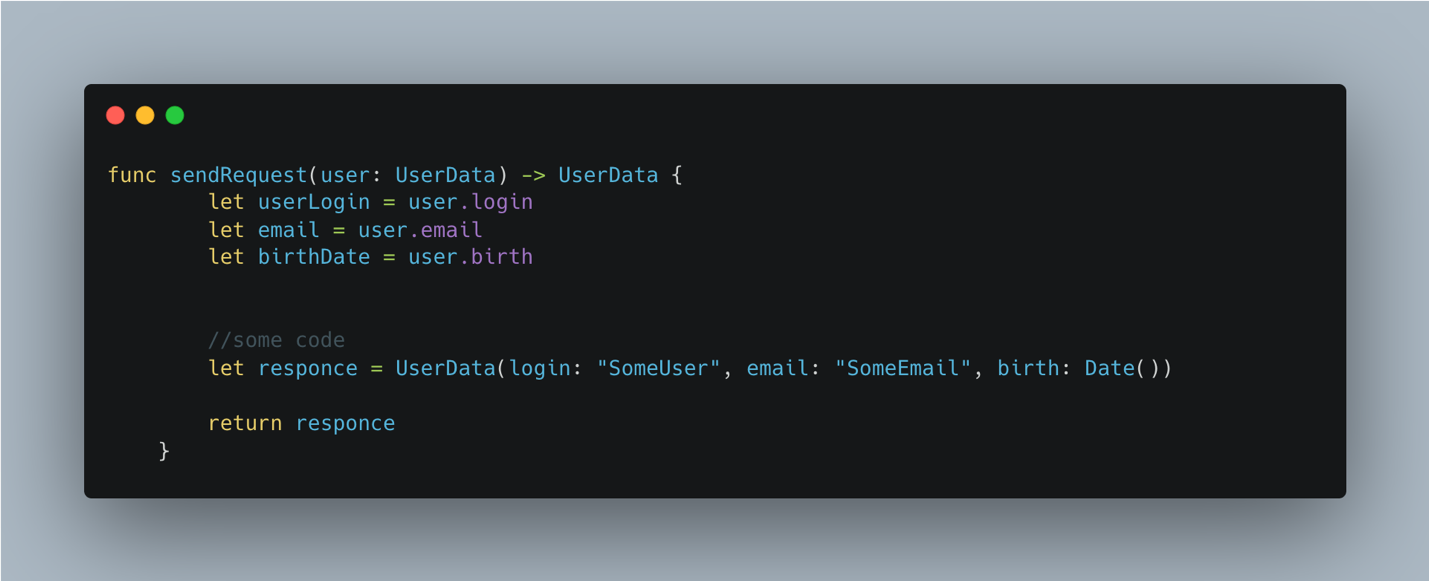


Теперь нам нужен класс, реализующий нужный нам механизм отправки запросов, давай зададим тестовый:



Внутри класса у нас один метод, который принимает в качестве аргументы тип UserData (напоминаю, любой класс или структура по умолчанию являются такими же типами данных, как все остальные).

И теперь внутри функции мы можем получить нужные нам данные, через свойство user.



Так же в качестве ответа, мы можем сформировать нужную структуру, и вернуть ее. В дальнейшем использование структур, мало чем будет отличаться от текущего примера, за исключением SwiftUI, где структуры служат для создания интерфейсов, но об этом позже.

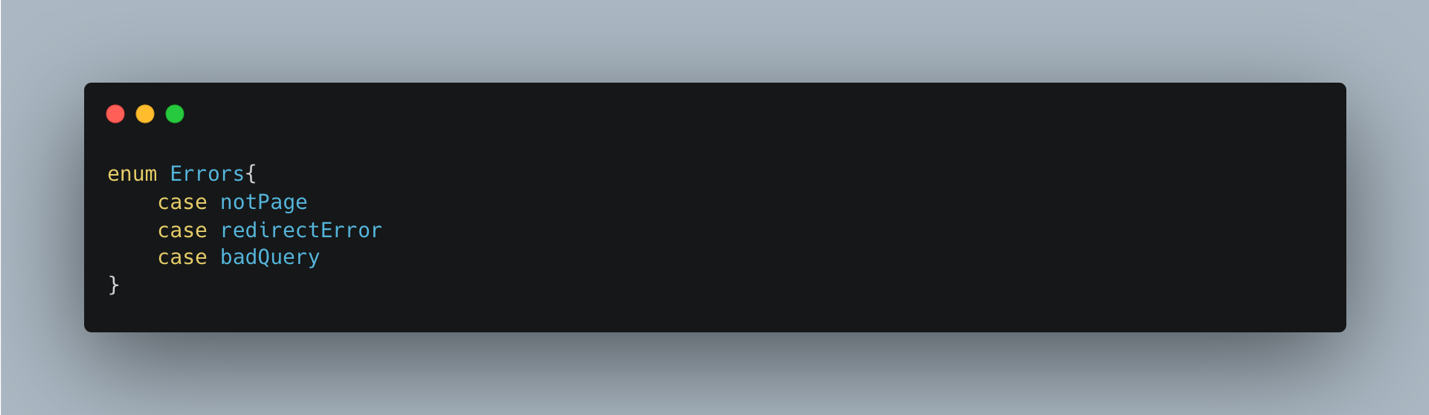
**Enum и где используются**

Есть еще тип данных это enum или перечисления. На практике enum используются как вспомогательные, но очень полезные элементы.

Такие перечисления отлично подходят для составления список данных, за рамки которых мы не сможем выйти, так как это ограниченно самим типом данных. Давайте разберем на практике. Допустим, мы знаем, что сервер может нам вернуть определенные ошибки, и мы знаем коды этих ошибок.



Здесь в качестве кода ошибки мы используем string (можно int но без разницы) и мы можем в один момент просто ошибиться в написании кода ошибки, вместо 404 написать 405 или не дописать символ, суть в том, что мы ни как не ограниченны. А теперь представьте, что мы принимаем в качестве аргумента ошибки не string, а перечисление, в котором строго ограниченный список возможных значений. Давайте для начала создадим такой список.

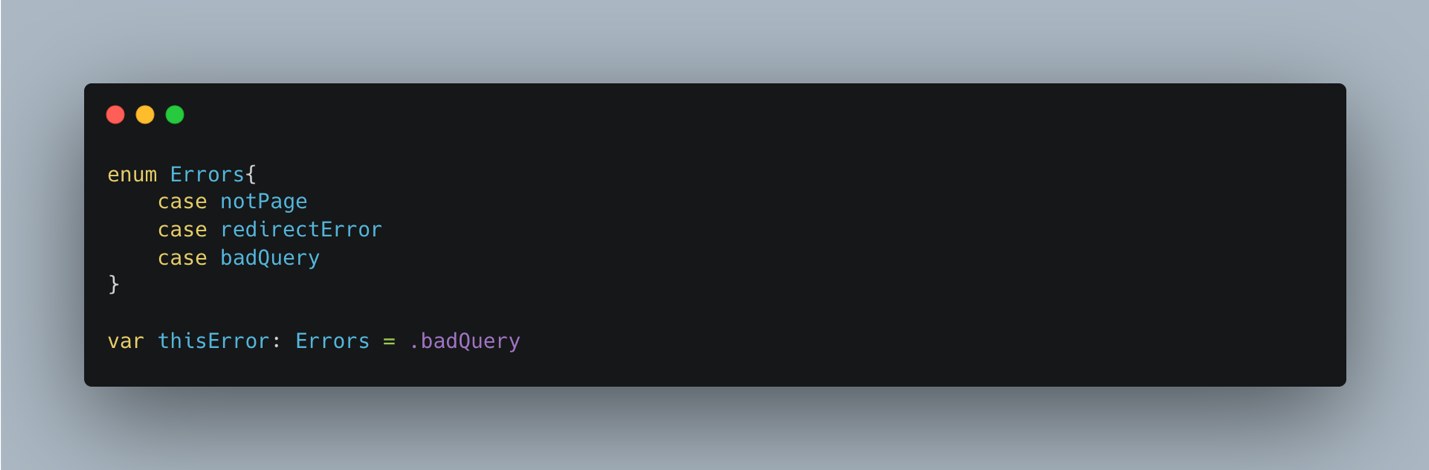


Перечисления создаются при помощи ключевого слова enum, а все возможные значния пишутся через case. Теперь у нас есть тип данных Errors который строго ограничен только теми значениями, которые мы сами установили (создавать название только из цифр или что бы начиналось с цифры нельзя, к примеру параметр 404 нельзя создать)

И теперь, мы можем передать в функцию или в любое нужное место, аргумент с типом данных Errors.



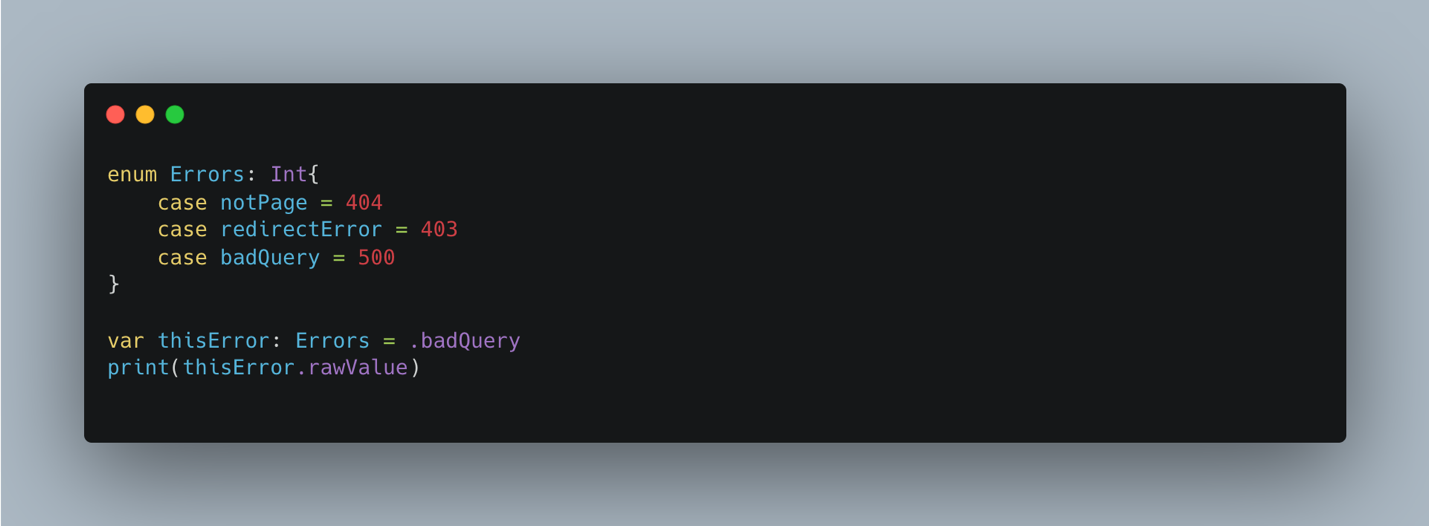
Для проверки текущего значения используем switch, так как он автоматически пропишет условия для всех состояний. Для того что бы установить значение для переменной, то можно написать следующий код:



Здесь переменной мы присвоили созданный тип, и через точку выбрали одно из доступных значений.

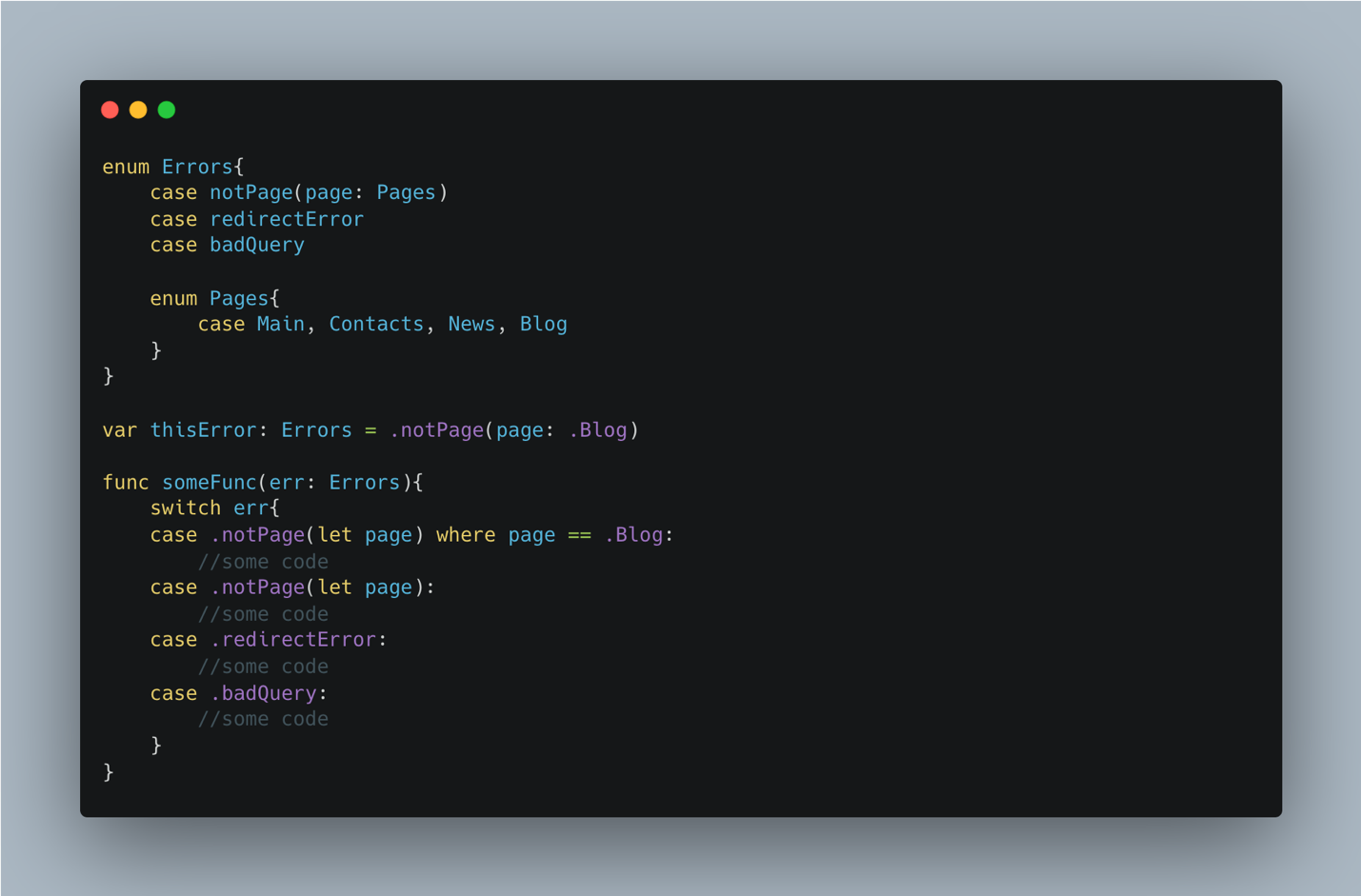
Так же нередко бывает необходимо, установить какое то значение для каждого элемента перечисления, для этого есть два способа, вернее он один, но вариативности два. Первый, если хотим, чтобы значения были такие же, как и название самих свойств, то просто наследуем наш enum от string



А что бы вывести значение используем свойство rawValue. И второй способ, если мы хотим что бы значения отличались от названия ключей, то присваиваем нужные нам значения. 

Тут мы наследуем перечисление от Int и присваиваем в качестве значния целочисленные данные, и теперь если вывести необходимый ключ, мы получим установленное у этого ключа целочисленное значение.

Так же можно перечисления делать вложенными, но на практике это используется редко, поэтому сильно не углубимся, а просто рассмотрим как это выглядит.

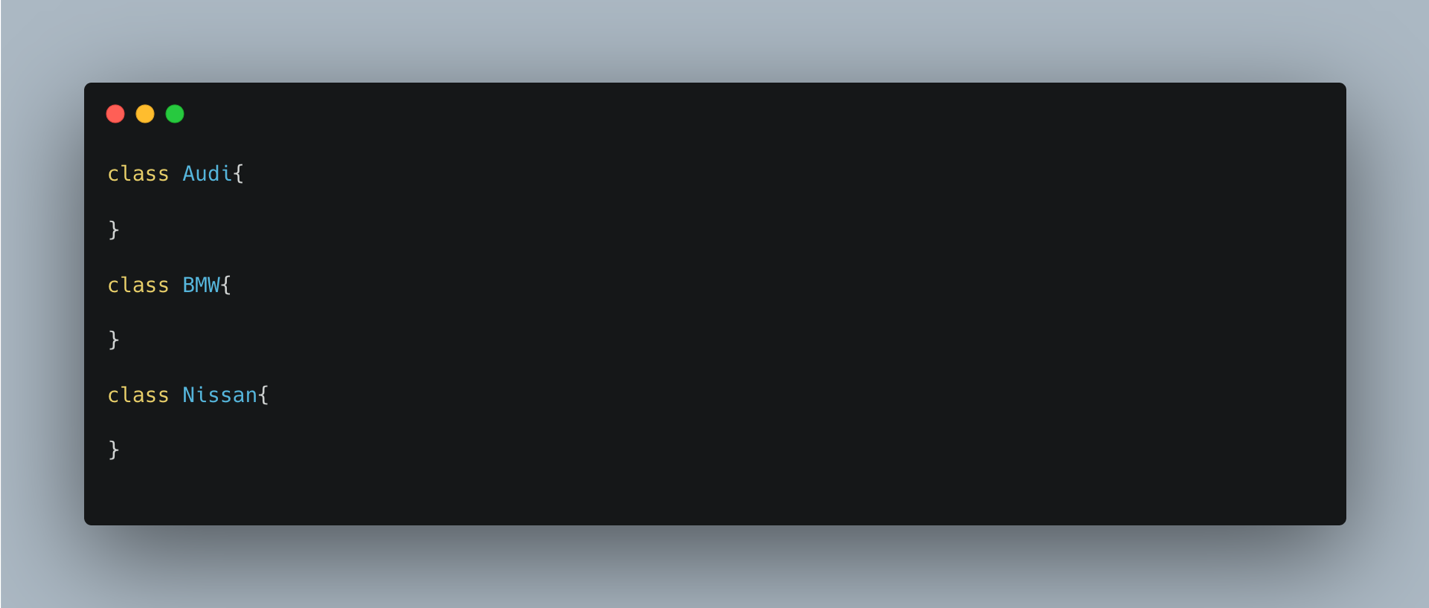


При помощи вложенных перечислений, мы можем создавать дополнительные проверки.

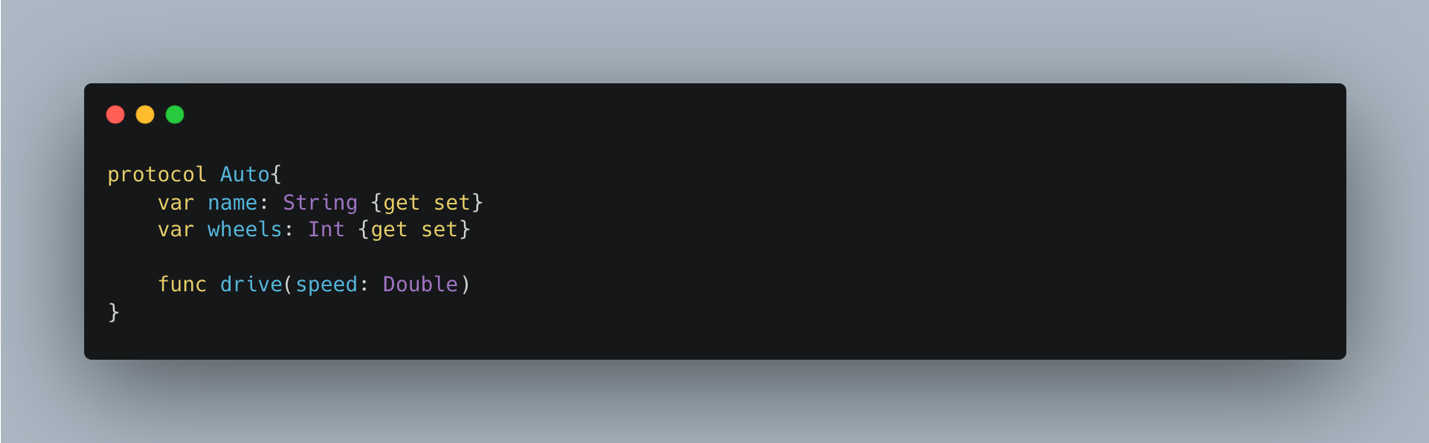
**Протоколы**

В язке swift есть еще один тип, который называется протокол, он позволяет нам выносить часть методов и свойств за пределы объекта, так же гарантирует, что у объектов будут необходимы свойства. При помощи протокола мы можем объединять разные объекты под одним типом данных, а также протоколы реализуют шаблон делегат. Про делегирование мы поговорим в будущих уроках, сейчас остановимся на его основных возможностях.

К примеру, допустим у нас есть несколько объектов, ни как между собой не связаных, и нам нужно гарантировать, что каждый из этих объектов должен иметь определенные свойства или методы, для этого мы можем создать протокол, описать в нем нужные нам данные, а затем наследовать объекты от созданного протокола. В таком случае, объекты должны реализовать данные протокола, или говоря иначе реализовать протокол. Давайте рассмотрим на примере.



Допустим нам необходимо что бы у каждого класса был метод drive и свойство name, сейчас мы можем написать это самостоятельно, но нет какой либо проверки наличия этого в классе, значит на нужны гарантии, что нужные нам данные точно будут. Тут как раз и включается протокол. Сделаем протокол с данными, которые обязательно должны быть в объектах:



Протоколе мы описали переменные, которые хотим чтобы были у всех объектов, а также метод. Заметьте, что у переменных мы пишем get и set, это значит, что мы можем как считывать, так и записывать в нее данные.

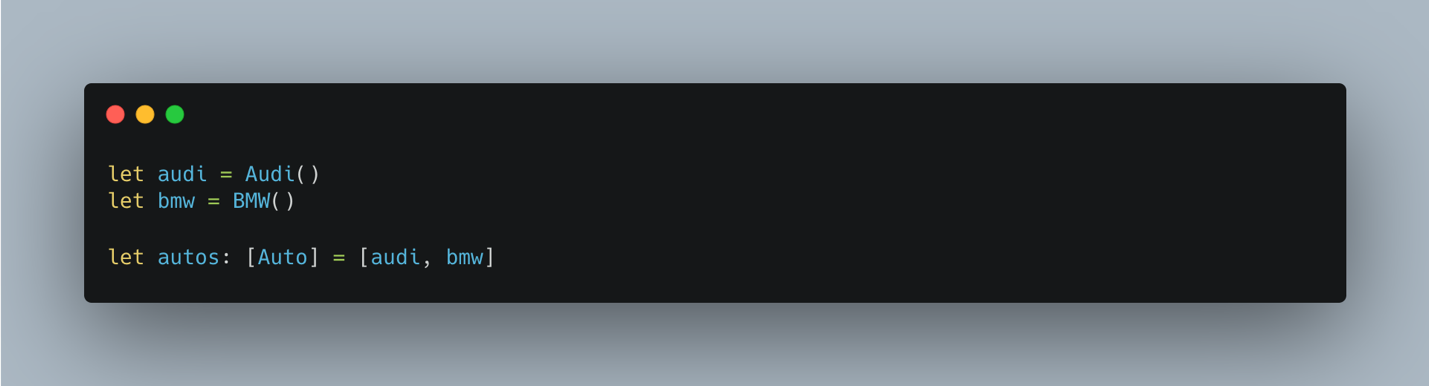
Теперь остается только наследоваться от этого протокола и классы у нас будут обязаны реализовать протокол.



Как видно класс теперь реализует протокол Auto и, следовательно, обязан реализовать все его свойства и методы. Каждый класс или структура, которая будет наследоваться от данного протокола, также буже реализовывать эти данные.



Тут мы видим два класса, которые реализуют один протокол, но в этом виде мы можем сказать, что протокол выступает тут в качестве типа данных, а раз это тип данных, мы можем делать с ними определенные операции, допустим, у нас будет массив типа [Auto], а так как наши классы наследуют протокол Auto, то можем написать следующий код.



ДЗ.

1. Создайте класс, с методом который принимает структуру данных (это может быть любые данные) и сохраните эти данные в словарь. В качестве ключа используйте заранее заданный enum
2. Создайте несколько классов, и реализуйте для них общий протокол, ограничьте протокол только классами.

Ссылки:

<https://swiftbook.ru/content/languageguide/enumerations/>

<https://swiftbook.ru/content/languageguide/protocols/>