Урок 4

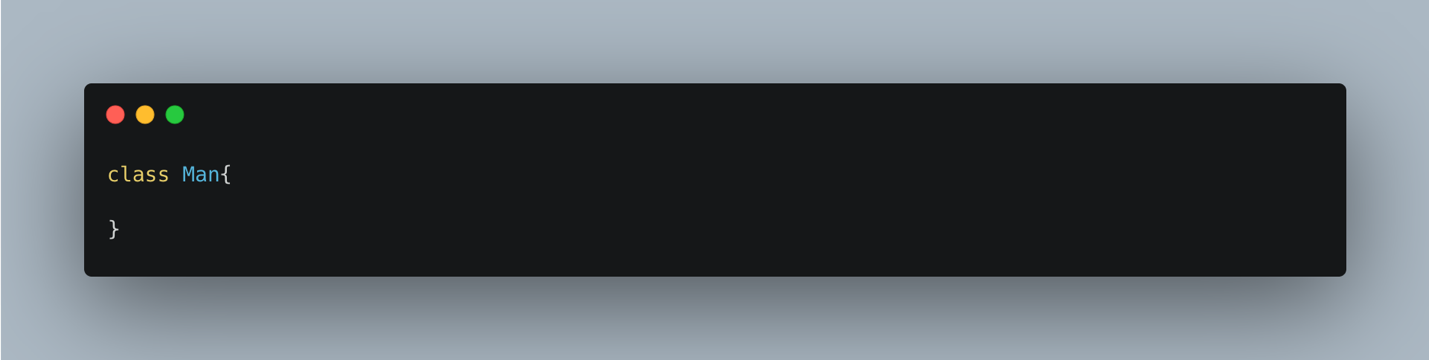
**План урока:**

* Классы и структуры и их сравнение.
* Инициализация классов.

Классы и структуры являются неотъемлемой частью языка Swift, как и любого языка ООП. Хоть они во многом и похожи друг на друга, но имеют ряд отличий и сферу применения. В swift классы и структуры являются некими строительными блоками, для кода нашей программы. Давайте рассмотрим каждый из них по отдельности.

**Классы**

В большинстве своем, каждый контроллер в нашем приложении будет классом. Что бы создать простой класс, достаточно написать слово class затем его название и открыть тело самого класс при помощи фигурных скобок.



Как правило, каждый класс мы будем создавать в отдельном файле, но, даже если мы создадим все классы в одном документе, то на работоспособность это не повлияет. Когда мы создаем класс, это регистрируется во всей системе и мы получаем к нему доступ независимо от того, в одном файле мы создали класс или в разных.

Когда мы создали класс, то соответственно должны его чем-то заполнить. Как мы сказали, классы являются важной частью парадигмы ООП, проще говоря, объекты, которыми мы оперируем это по сути и есть классы. Для примера мы можем сказать что наш класс или объект, это некий объект из нашего мира, к примеру человек, собака, машина и тд.

К примеру, давайте опишем объект Машина, и попробуем на основе описания создать класс. Проще говоря, представьте, что на автозаводе нужен шаблон, под который будут выпускаться большинство моделей, и в зависимости от параметров (скорость, количество дверей, цвет и тд) будут получаться разные модели, но в сущности своей это будет автомобиль.

Опишем наш объект, или его свойства, у него:

* Название
* Есть 4 колеса
* Есть двери (могут быть разное количество)
* Это может быть электромобиль
* И тд (для примера остановимся на этих)

А теперь опишем его функции, или методы, с которыми мы можем взаимодействовать.

* Он может ездить
* В нем можно слушать музыку
* И тд (для примера хватит два)

А теперь давайте перенесем это в код:

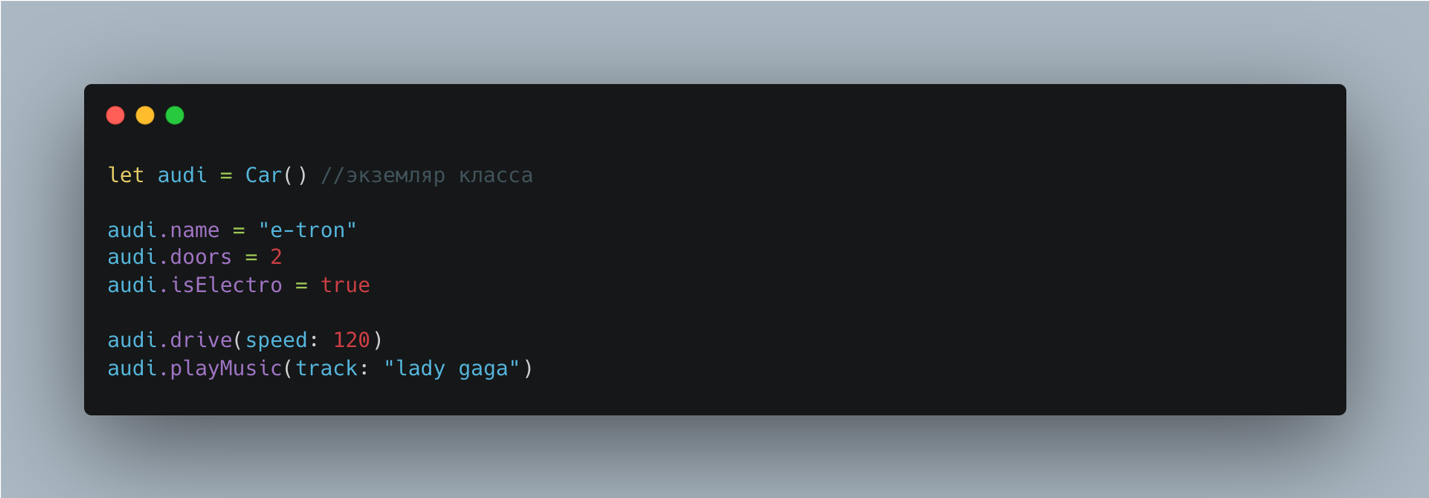


В итоге, мы взяли физическую модель из нашего мира и перенесли его в код. Теперь мы можем оперировать следующими понятиями:

* Класс – это объект
* Переменные – свойство объекта/свойство класса (равносильно)
* Функции – методы объекта/методы класса (равносильно)

Теперь, когда на заводе есть шаблон автомобиля, следовательно можно создать первый экземпляр.

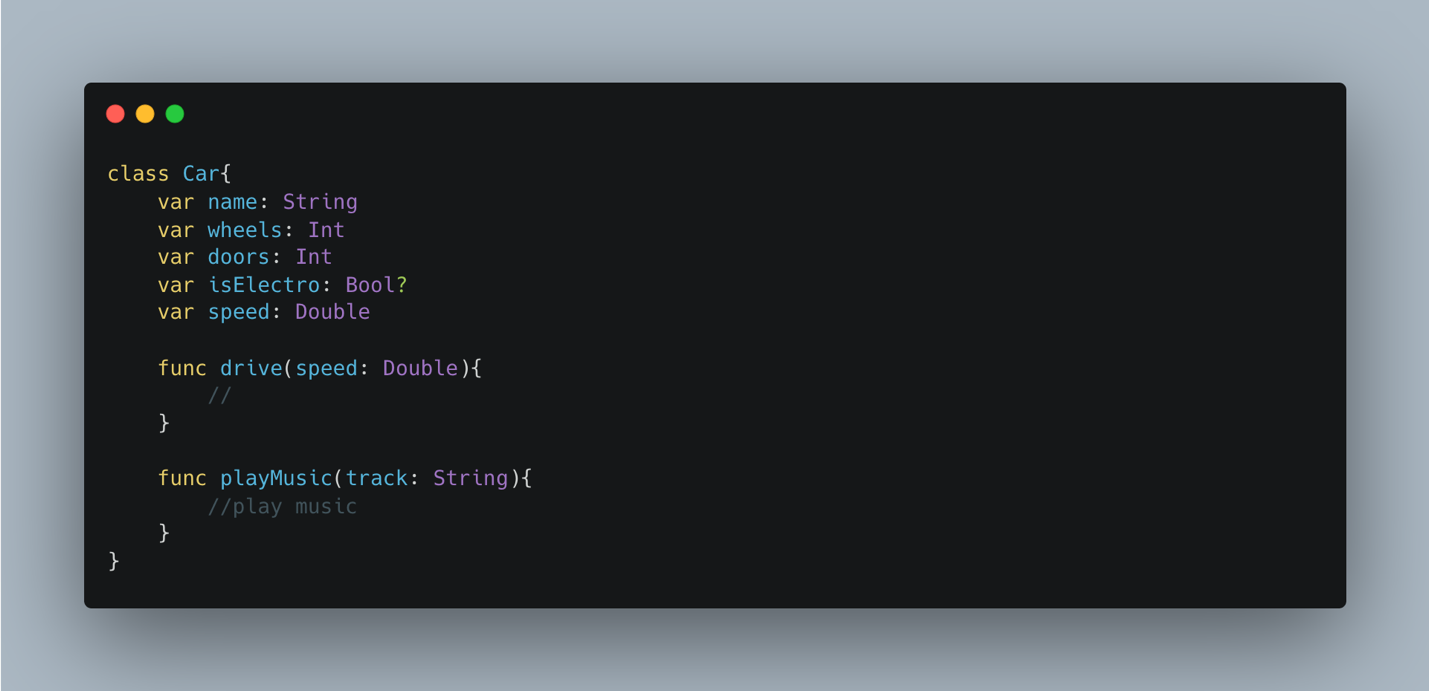
В живую это звучало бы следующим образом – создать экземпляр на основе подготовленного макета (объекта) «Автомобиль», у этого экземпляра будет 2 двери, это будет электромобиль, после того как мы его сделаем, мы ходим на нем поездить и послушать музыку. Все просто, еще проще перенести это в код.



Первым делом мы создаем экземпляр, в ооп это так и называется, создать экземпляр объекта, в данном случае мы создали экземпляр класса, создается он очень просто, пишем переменную, через которую будем обращаться к свойствам и методам класса, и присваиваем этой переменной класс. Теперь можно сказать, что у нас готов первый экземпляр, и его можно начать менять использую внутренние свойства самого класса.

Так же теперь у нас есть доступ к методам объекта.

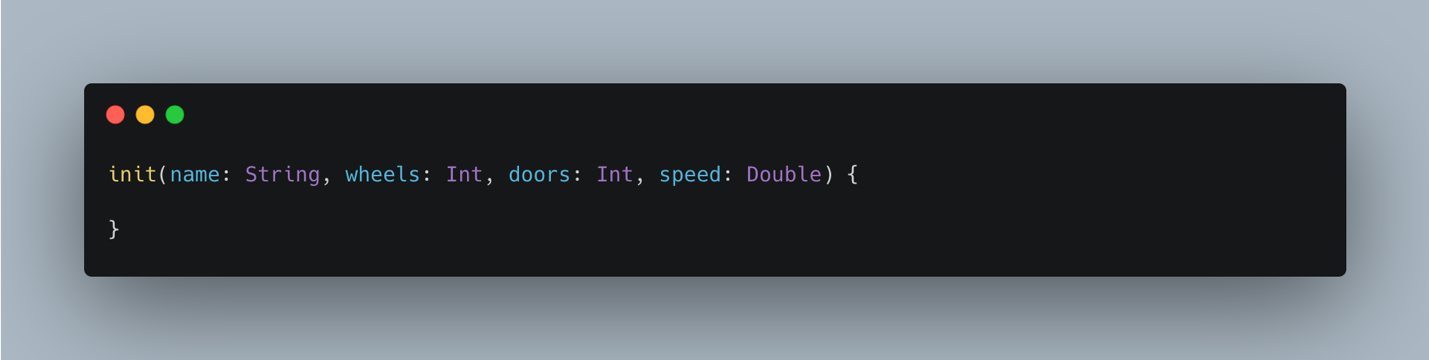
Это пример самого просто класса, с заранее инициализированными свойствами. Давай рассмотрим пример, когда свойства класса у нас заданы, но не определены.



Тут мы просто убрали значения переменных, но добавили тип данных, иначе не поймем, какое значение можем устанавливать в переменную.

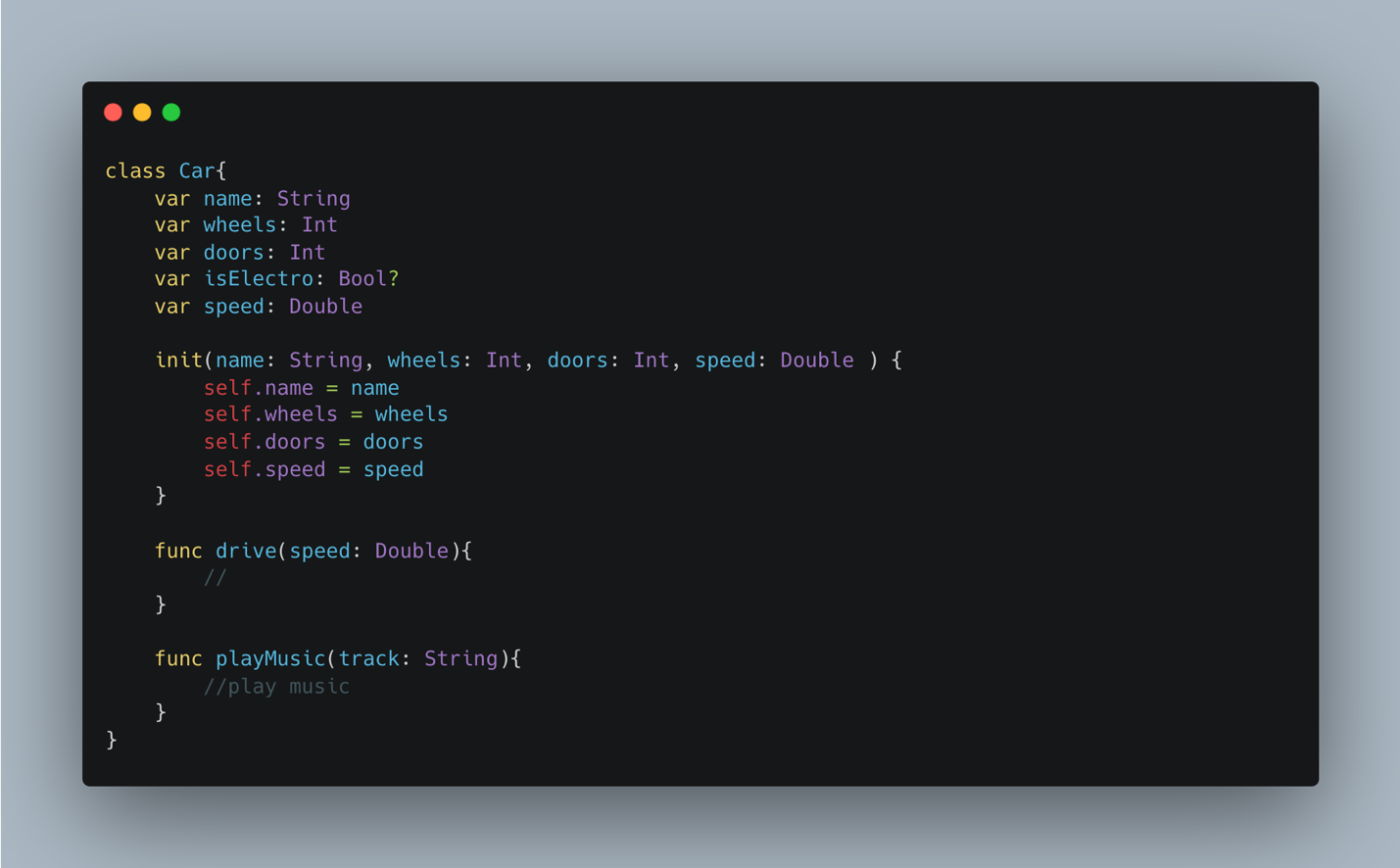
Если так оставим, то у xcode выдаст ошибку (Class 'Car' has no initializers). Он говорит, что класс у нас не проинициализирован. Это обязательное правило, переменные класса не могут быть пустыми, если тип не опциональный, если опциональный то он по умолчанию присвоит ему nil.

Хорошо, значит мы должны их проинициализировать, но что делать, если мы пока не знаем, какие данные должны быть в них? Есть два варианта, первый, как мы сделали ранее, это проставить начальные значения каждой переменной, и потом просто переопределять их, как мы уже делали, и второй, более правильный, это создать инициализатор. Проще говоря, инициализатор – это функция, которая запускается в момент создания экземпляра. Создается внутри самого класса. Давайте такой сделаем, и в качестве аргументов добавим в функцию значения, которые мы ждем при инициализации класса.



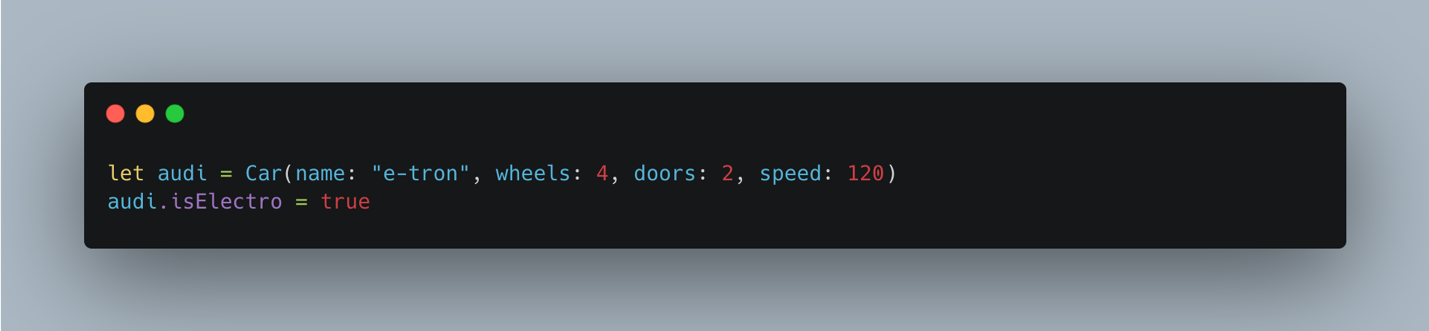
Инициализатор называется по умолчанию init. И мы пишем, что хотим принимать ряд значений в качестве аргумента, **ВАЖНО** – название НЕ обязательно должны совпадать с названиями переменных в классе, просто это такое неписаное правило, аргументы называть также, как и инициализируемые переменные.

И теперь мы можем присвоить входящие аргументы свойствам нашего класса.

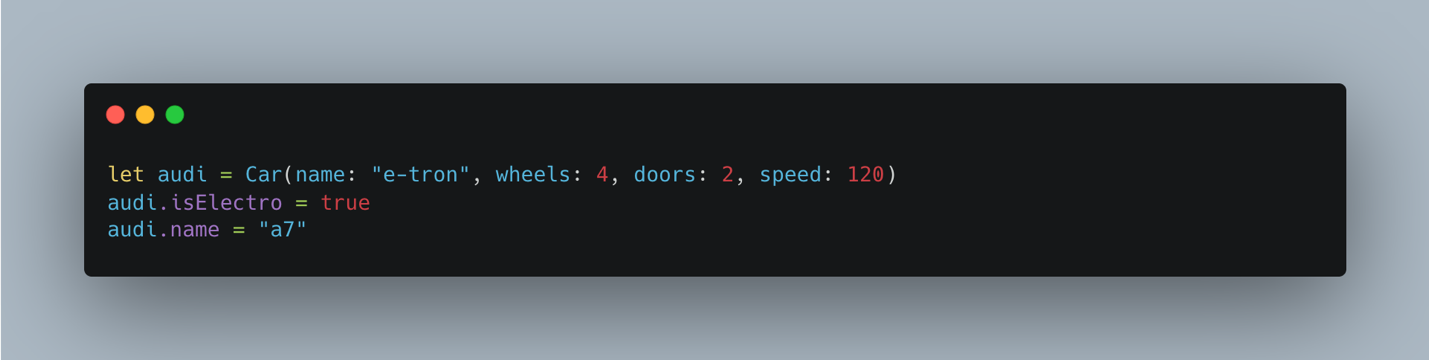


Тут вы видите новое для себя слово self, по сути, это ссылка класса на самого себя, так как мы используем одинаковые названия в аргементах инициализатора и свойствах класса, чтобы swift понимал, что мы хотим присвоить значения свойствам класса, мы обращаемся через self к самому классу и ищем у него нужную нам переменную. Как правило swift сам понимает из контекста, что мы хотим, но нередко требуется ему помочь.

Теперь, инициализатор мы написали, значит должны как то его вызвать и передать туда какие то значения, инициализатор вызывается при создании экземпляра класса. Значит изменим наш вызов класса.



Теперь мы создаем экземпляр автомобиля и сразу передаем туда параметры. Обратите внимание, что изначально у нас переменна isElectro опциональная, поэтому мы ее не добавляли в инициализатор (хотя можно, никаких ограничений), но это не мешает нам потом как обычно обратиться к экземпляру класса и определить нужное нам свойство. Так же мы можем и переопределять уже определенные свойства при инициализации.



Более детально поговорим про классы на следующем уроке.

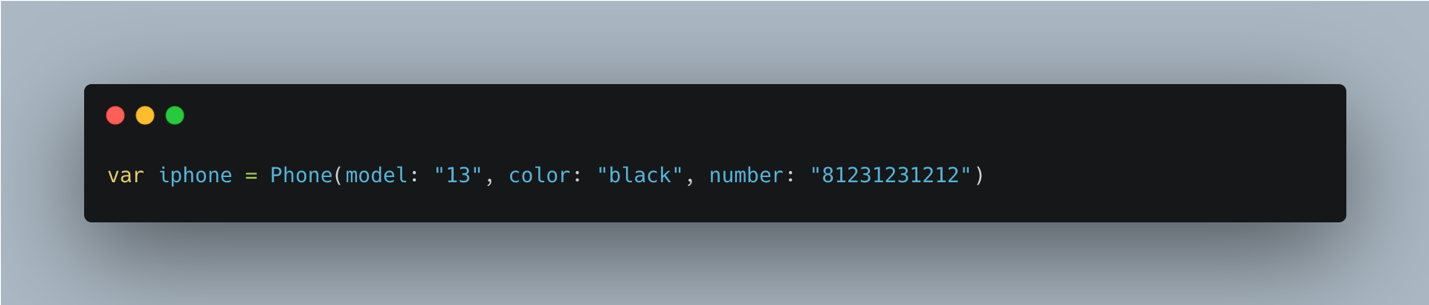
А теперь давайте поговорим про структуры, и в конечном счет сравним их.

**Структуры**

Структуры имеют схожий синтаксис и логику. Мы так же можешь объявлять структуру объекта, задавать свойства и методы.

****

Но обратите внимание, мы определили свойства, но ни как их не проинициализировали, и не добавили инициализатор, пожалуй, тут и кроется один из основных отличий от класса, это то, что для структур не нужны инициализаторы, по умолчанию он уже встроен в вызов структуры. Но при необходимости нам так же никто не мешает создать свой инициализатор.



При создании экземпляра структуры, xcode автоматически выведет инициализируемые свойства.

В Swift все типы значения деляться на два типа, reference type (ссылочный тип) и value type (хранимый тип). Мы работаем с вами с переменными, с типами данных int, float, string и тд, - так это все хранимый тип данных. Это значит что мы создаем переменную, можем записать в нее данные, можем скопировать эту переменную в другую переменную, и задавать ей новое значение и потом это будут две разные переменные. Представьте что есть клетка, она разделилась, и они теперь развиваются совершенно разными путями и это уже совершенно разные клетки, хотя изначально произошли от одного экземпляра. И новое откровение – все типы данных, которые мы использовали это **структуры.** Int, float, string и тд – это структуры, за которыми скрывается свой код и в конечном счете они работают так, как работают.

Отсюда вытекает очень просто следствие: структуры – это value type (хранимый тип). Тогда если переменные – это хранимый тип, структуры хранимый тип, то что тогда представляет ссылочный тип? Верно – это классы.

Давайте рассмотрим, как это вообще работает и на что это влияет.

Допустим у нас есть наша структура телефон, давайте также сделаем экземпляр структуры и еще ряд операций:



Мы определили переменную iphone как экземпляр структуры, и создаем новую переменную, куда записываем переменную iphone, в этот момент у нас произошло просто копирование, так как у нас структура это хранимое значение, то теперь у нас в память скопировались две совершенно не связанные структуры. Это видно на примере, мы изменили модель iphone и изменили модель xiaomi и в итоге выводим совершенно две разные структуры.

А давайте посмотрим, как в таком случае работает класс.

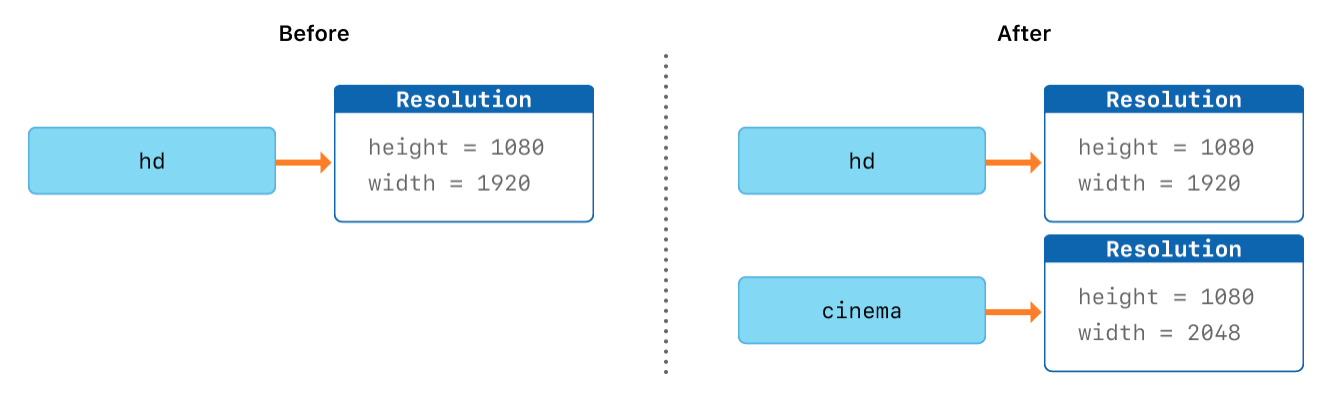


Мы так же сделали экземпляр, присвоили одну переменную другой, определили у каждой из них свои свойства. Но когда пытаемся выводить, то у нас выводится все равно одно и тоже, в данном случае последнее что мы присвоили свойству name то и выводится.

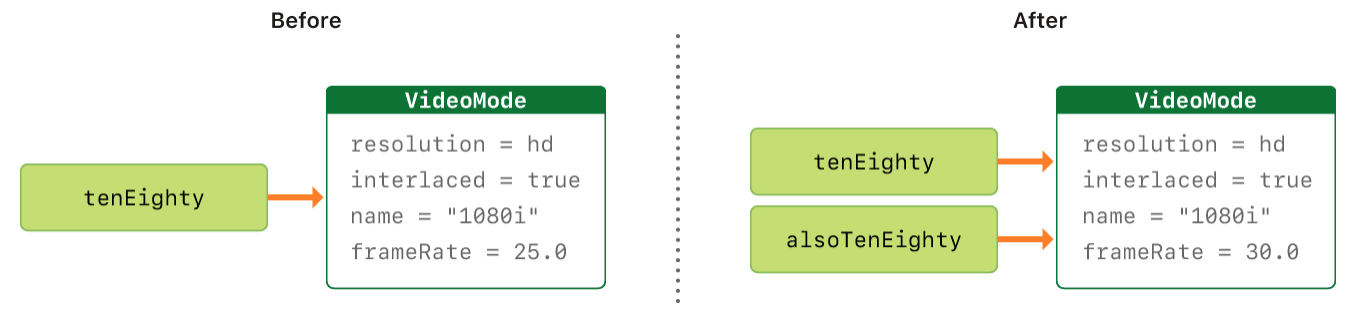
В данном случае у нас не произошло копирование, а теперь у нас просто на один экземпляр ссылаются две переменные, мы просто создали новую ссылку.

Посмотрим на наглядный пример:

Это хранимый тип или value type (Структура)



Это ссылочный тип или reference type (класс)



Благодаря тому, мы можем создать один класс и нужное количество ссылок на него, а не копировать новые экземпляры в память, классы и будем использовать в большинстве случаев. Но есть большой минус такого подхода, мы можем так увлечься, и создать кучу ненужный ссылок, лишний раз забивая этим память, но в swift существует механизм автоматического подсчета ссылок, что бы не было утечки памяти, этот момент мы механизм мы рассмотрим в следующих уроках.

В свою очередь структуры отлично подходят для модели данных, потому что с ними очень просто работать, не нужно писать инициализаторы и не нужно думать о том, нет ли лишних ссылок на экземпляр. Так же структуры используются в SwiftUI, при помощи них мы будет делать интерфейсы, а для всего остального классы.

ДЗ.

1. Создайте класс любого объекта, на ваш выбор, опишите его свойства и методы (желательно так же в методах написать простой функционал, на ваше усмотрение). Сделайте инициализацию внутри класса. Создайте экземпляр этого класса и задайте свойства и вызовите его методы.
2. Опишите структурой любой объект. Создайте несколько экземпляров (несколько объектов)

<https://swiftbook.ru/content/languageguide/classes-and-structures/>

<https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html>