**Модуль1. Урок 5**

**Классы**

* Наследование классов
* Инициализаторы

Мы уже научились создавать простые классы и структуры. Так же мы знаем, что у класса обязательно должны быть проинициализированные все его свойства, а у структуры инициализатор создается автоматически, если мы не создали свой, в момент создания экземпляра.

Про структуры мы будем подробно говорить в следующем уроке, в этом давайте разберем классы более подробно, а также посмотрим, какие еще инициализаторы бывают и как с ними работать.

**Наследование**

Для начала рассмотрим наследование классов и объектов – это один из принципов ООП.

Это значит, что один объект, в нашем случае класс, может базироваться на другом классе, или проще говоря, наследовать методы и свойства. В языке SWIFT мы так же можем классы (как и структуры) наследовать друг от друга.

Давайте разберем небольшой пример, представим, что у нас есть класс Животных, это общая реализация всех видов, и есть классы отдельных животных, это частная реализация. В таком случае, мы можем сказать, что у всех животных по 4 лапы, и так же они могут принадлежать к хищниками или травоядным, и, к примеру есть название животного, для примера этих данных хватит.

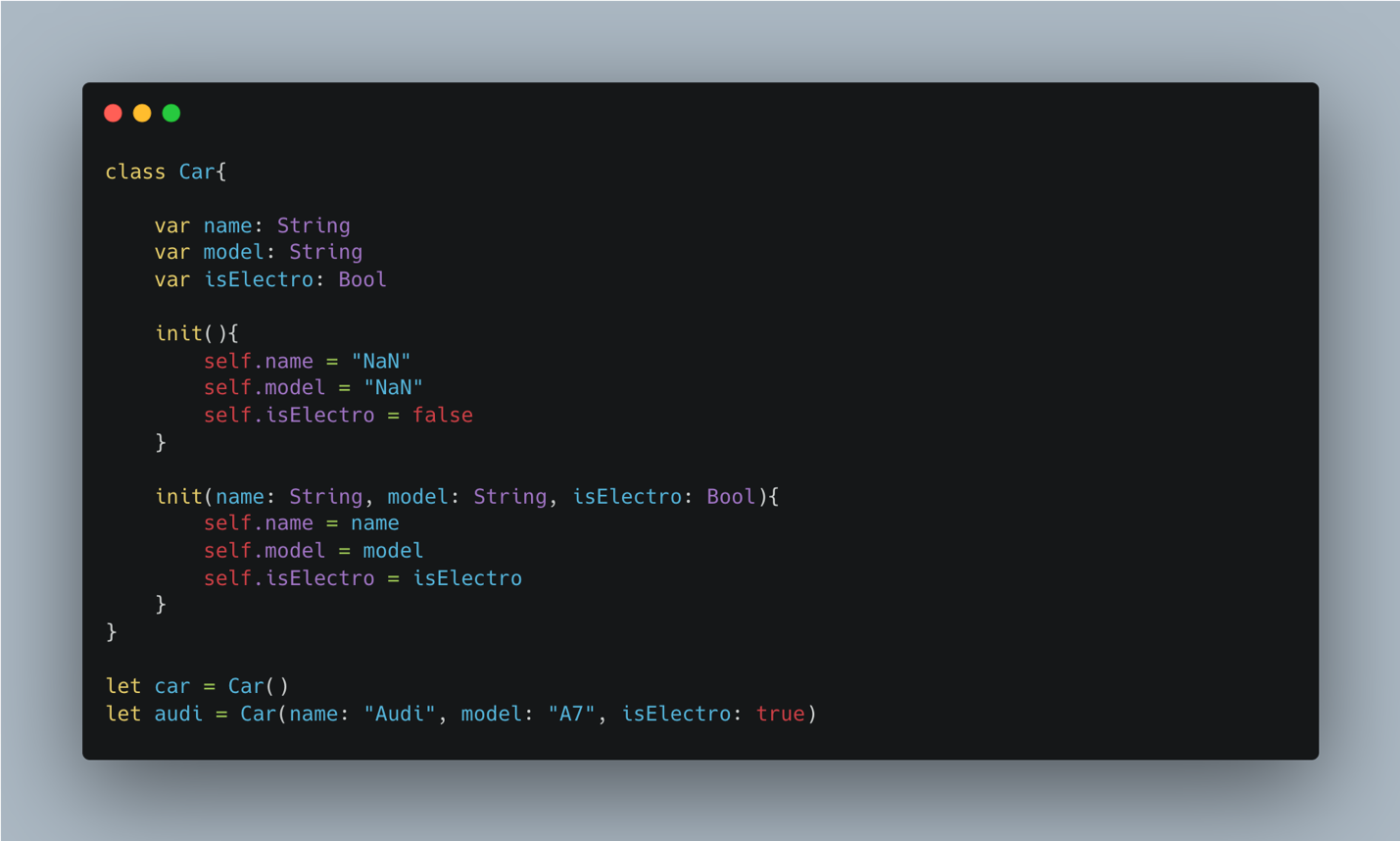
Тогда, мы можем создать родительский класс, у которого будут все эти свойства, а когда будем создавать классы отдельных животных, просто брать или **наследовать** от родительского класса нужные нас свойства и методы. Посмотрим в коде.



Теперь мы можем использовать свойства родительского класса Animal в дочернем классе Cat. Тут мы задали свойства класса Animal по умолчанию, без инициализаторы, давайте теперь более подробно разберем тему с инициализаторами и какие можно создавать их для класса.

**Инициализаторы класса**

У класса может быть несколько инициализаторов, к примеру, мы можем написать два разных инициализатора, где один принимает одни параметры, другой другие.



Мы создали у одного класса два инициализатора, и их может быть еще больше, в зависимости от необходимости, а при создании экземпляра можно выбрать, какой инициализатор нужен в данном случае.

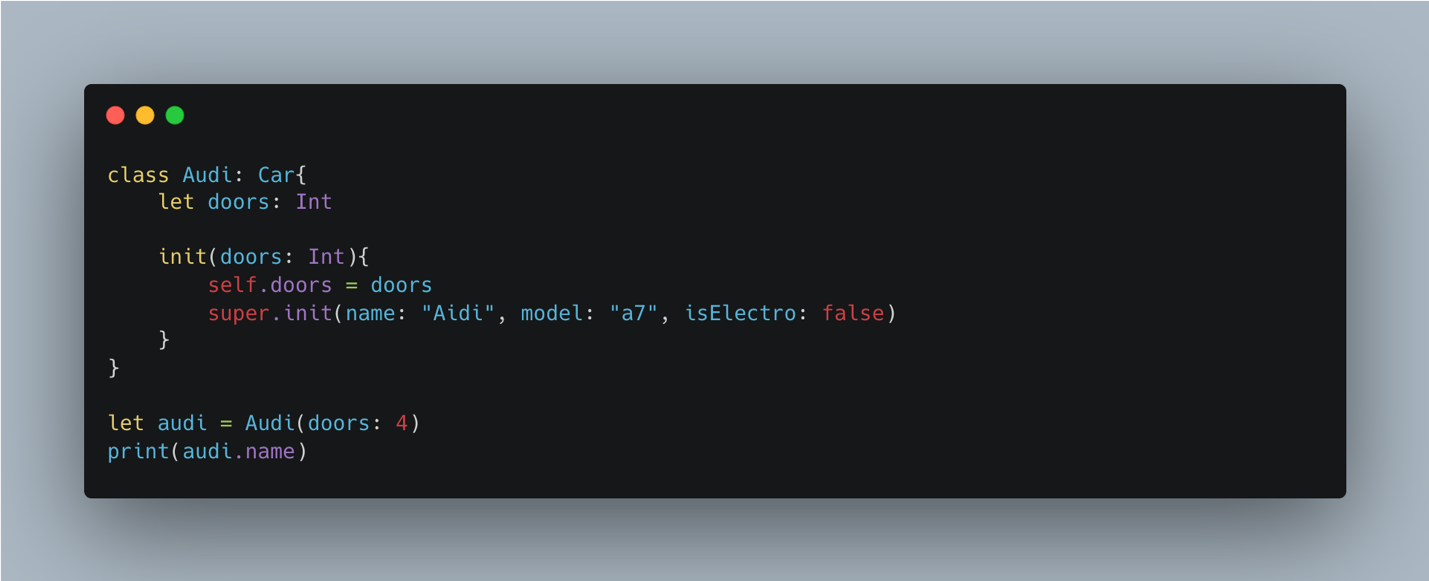
Так же можно создавать дополнительные или вспомогательные инициализаторы. Инициализаторы, которые мы делали называются назначенные инициализаторы или **designated**, а вспомогательные инициализаторы **convenience**. По сути **convenience** мы можем делать более тонкую настройку.

К примеру, через **designated** инициализатор мы можем присвоить все свойства класса, а через **convenience** проверить дополнительные условия и переопределить свойства, рассмотрим на примере.



Мы создали **designated** инициализатор, и в нем определили все свойства, так же создали вспомогательный **convenience** инициализатор, где проверяем условие, а только затем вызываем главный инициализатор. Как правило на практике **convenience** используется редко, так как все операции можно делать в основном инициализаторе, но сам механизм создание дополнительных инициализаторов существует и про них нужно знать.

Мы уже разобрали, что классы могут наследоваться друг от друга, теперь давайте посмотрим, как такие классы можно инициализировать.



Теперь, при инициализации класса Audi мы так же должны обязательно инициализировать все методы родительского класса, для этого в инициализаторе дочернего класса, вызываем designated инициализатор родительского класса. При работе с наследованием важно помнить два правила:

* Назначенные инициализаторы должны делегировать наверх
* Вспомогательные инициализаторы должны делегировать по своему уровню (классу).

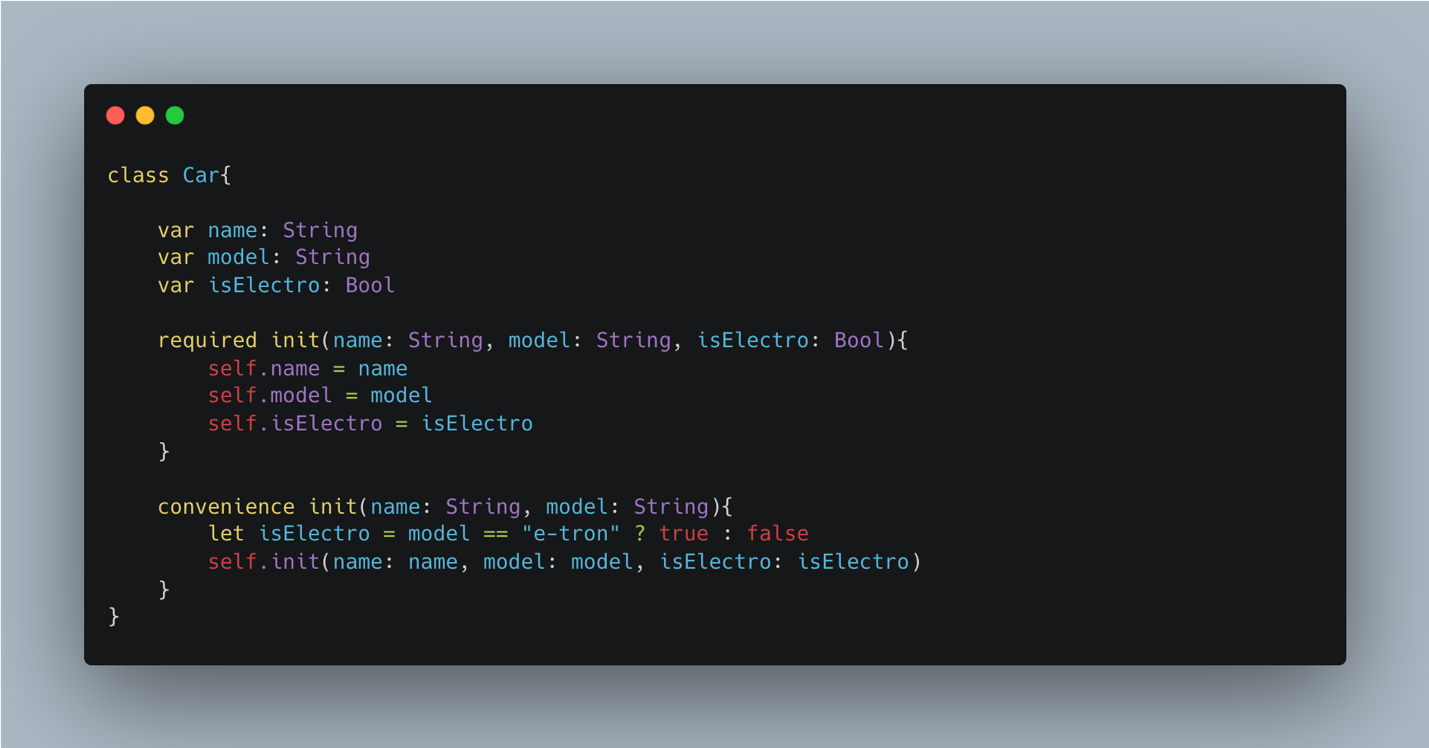
Такой механизм называется двухфазной инициализацией: сначала мы инициализируем свойства дочернего класса, затем инициализируем свойства родительского класса. Про этот механизм мы поговорим в отдельном уроке.

Рассмотрим еще один инициализатор, который называется **failable** или проваливающий инициализатор. Представим, что нам нужен экземпляр класса, только в определенном случае, и если некое условие не выполняется, то соответственно вернем nil. На примере это будет выглядеть следующим образом:

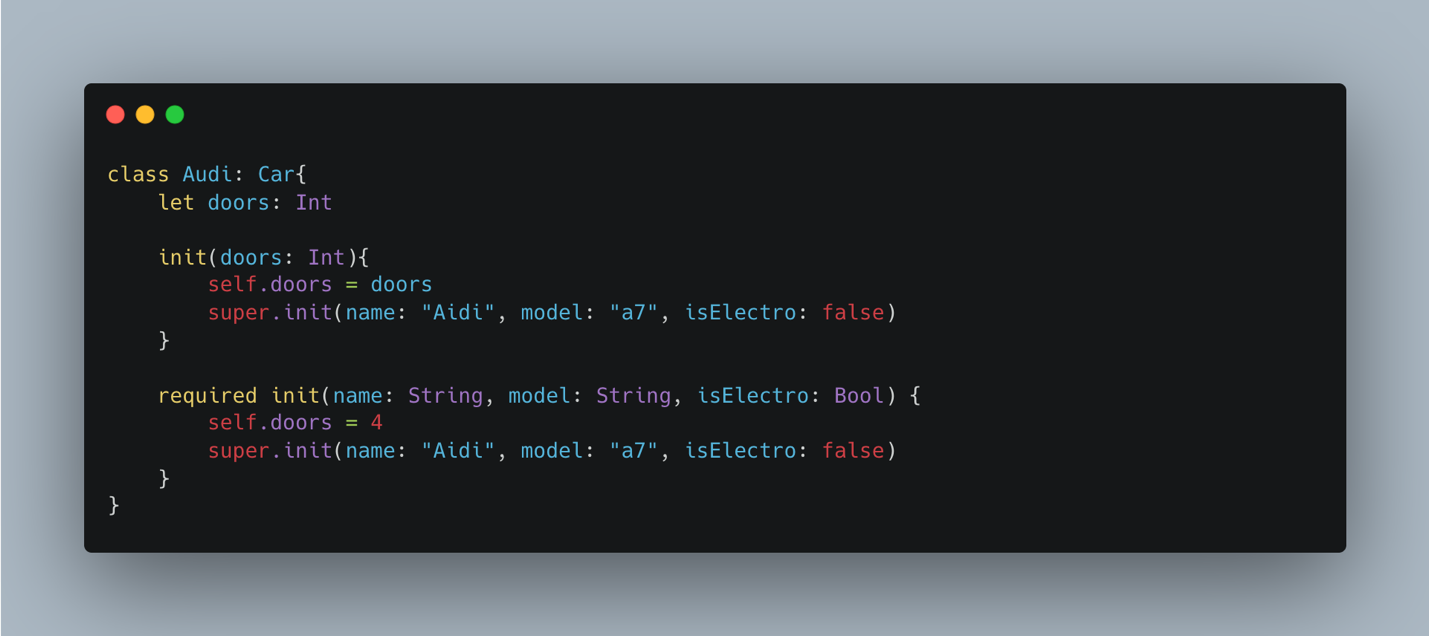


При создании инициализатора, мы просто поставим знак **?** и теперь, если условие, которые мы написали не выполниться, то экземпляр у нас будет nil, в противном случае у нас создастся экземпляр, но только он будет уже опциональным.

И напоследок давайте разберем инициализатор, который называется требуемый, или **required**. Мы используем такой инициализатор в том случае, когда каждый дочерний класс родительского класса обязан реализовать этот инициализатор:



Таком случае мы добавляем ключевое слово **required** перед инициализатором, и во всех дочерних классах появится сообщение об ошибке, что мы не реализовали **required** инициализатор. В таком случае родительский инициализатор будет иметь следующий код.



Тут мы реализовали обязательный инициализатор, и так же обязательно пишем **super.init**, так как должны инициализировать еще и все родительские свойства.

Ссылки

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