Лекция #9.2 Геттеры и сеттеры

Геттеры (getter) и сеттеры (setter) (еще их называют методами доступа) позволяют управлять доступом к переменной. Их формальный синтаксис:

```
var имя_свойства[: тип_свойства] [= инициализатор_свойства]
    [getter]
    [setter]
```

Инициализатор, геттер и сеттер свойства необязательны. Указывать тип свойства также необязательно, если он может быть выведен их значения инициализатора или из возвращаемого значения геттера.

Геттеры и сеттеры необязательно определять именно для свойств внутри класса, они могут также применяться к переменным верхнего уровня.

Сеттер

Сеттер определяет логику установки значения переменной. Он определяется с помощью слова **set**. Например, у нас есть переменная age, которая хранит возраст пользователя и представляет числовое значение.

```
var age: Int = 18
```

Но теоретически мы можем установить любой возраст: 2, 6, -200, 100500. И не все эти значения будут корректными. Например, у человека не может быть отрицательного возраста. И для проверки входных значений можно использовать сеттер:

```
var age: Int = 18
    set(value){
        if((value>0) and (value <110))
            field = value
    }

fun main() {

    println(age)    // 18
    age = 45
    println(age)    // 45
    age = -345
    println(age)    // 45
}
```

Блок **set** определяется сразу после свойства, к которому оно относится - в данном случае после свойства age. При этом блок set фактически представляет собой функцию, которая принимает один параметр - value, через этот параметр передается устанавливаемое значение. Например, в выражении age = 45 число 45 и будет представлять тот объект, который будет храниться в value.

В блоке set проверяем, входит ли устанавливаемое значение в диапазон допустимых значений. Если входит, то есть если значение корректно, то передаем его объекту **field**. Если значение некорректно, то свойство просто сохраняет свое предыдущее значение.

Идентификатор **field** представляет автоматически генерируемое поле, которое непосредственно хранит значение свойства. То есть свойства фактически представляют надстройку над полями, но напрямую в классе мы не можем определять поля, мы можем работать только со свойствами. Стоит отметить, что к полю через идентификатор field можно обратиться только в геттере или в сеттере, и в каждом конкретном свойстве можно обращаться только к своему полю.

В функции main при втором обращении к сеттеру (age = -345) можно заметить, что значение свойства age не изменилось. Так как новое значение -345 не входит в диапазон от 0 до 110.

Геттер

Геттер управляет получением значения свойства и определяется с помощью ключевого слова get:

```
var age: Int = 18
    set(value){
        if((value>0) and (value <110))
            field = value
    }
    get() = field
```

Справа от выражения `get()` через знак равно указывается возвращаемое значение. В данном случае возвращается значения поля `field`, которое хранит значение свойства name. Хотя в таком геттер большого смысла нет, поскольку получить подобное значение мы можем и без геттера.

Если геттер должен содержать больше инструкций, то геттер можно оформить в блок с кодом внутри фигурных скобок:

```
var age: Int = 18
    set(value){
        println("Call setter")
        if((value>0) and (value <110))
            field = value
    }
    get(){
        println("Call getter")
        return field
    }
```

Если геттер оформлен в блок кода, то для возвращения значения необходимо использовать оператор `return`. И, таким образом, каждый раз, когда мы будем получать значение переменной `age` (например, в случае с вызовом `println(age)`), будет срабатывать геттер, когда возвращает значение. Например:

```
fun main() {

    println(age)    // срабатывает get
    age = 45        // срабатывает set
    println(age)    // срабатывает get
}
```

Консольный вывод программы:

```
Call getter
18
Call setter
Call getter
45
```

Использование геттеров и сеттеров в классах

Хотя геттеры и сеттеры могут использоваться к глобальным переменным, как правило, они применяются для опосредования доступа к свойствам класса.

Используем сеттер:

```
fun main() {

    val bob: Person = Person("Bob")
    bob.age = 25        // вызываем сеттер

    println(bob.age)   // 25
    bob.age = -8        // вызываем сеттер
    println(bob.age)   // 25
}
class Person(val name: String){

    var age: Int = 1
        set(value){
            if((value>0) and (value <110))
                field = value
        }
}
```

При втором обращении к сеттеру (`bob.age = -8`) можно заметить, что значение свойства age не изменилось. Так как новое значение -8 не входит в диапазон от 0 до 110.

Вычисляемый геттер

Геттер может возвращать вычисляемые значения, которые могут задействовать несколько свойств:

```
fun main() {
    val tom = Person("Tom", "Smith")
    println(tom.fullname)   // Tom Smith
    tom.lastname = "Simpson"
    println(tom.fullname)   // Tom Simpson
}
class Person(var firstname: String, var lastname: String){

    val fullname: String
        get() = "$firstname $lastname"
}
```

Здесь свойство fullname определяет блок get, который возвращает полное имя пользователя, созданное на основе его свойств firstname и lastname. При этом значение самого свойства `fullname` напрямую мы изменить не можем - оно определено доступно только для чтения. Однако если изменятся значения составляющих его свойств - firstname и lastname, то также изменится значение, возвращаемое из fullname.

Использование полей для хранения значений

Выше уже рассматривалось, что с помощью специального поля `field` в сеттере и геттере можно обращаться к непосредственному значению свойства, которое хранится в специальном поле. Однако мы сами можем явным образом определить подобное поле. Нередко это приватное поле:

Можно использовать одновременно и геттер, и сеттер:

```
fun main() {

    val tom = Person("Tom")
    println(tom.age)    // 1
    tom.age = 37
    println(tom.age)    // 37
    tom.age = 156
    println(tom.age)    // 37
}
class Person(val name: String){

    private var _age = 1
    var age: Int
        set(value){
            if((value > 0) and (value < 110))
                _age = value
        }
        get()=  _age
}
```

Здесь для свойства age добавлены геттер и сеттер, которые фактически являются надстройкой над полем _age, которое, собственно, хранит значение.