Лекция #9.5 Интерфейсы

Интерфейсы представляют контракт, который должен реализовать класс. Интерфейсы могут содержать объявления свойств и функций, а также их реализацию по умолчанию.

Для определения интерфейса применяется ключевое слово **interface**. Например:

```
interface Movable{
    var speed: Int  // объявление свойства
    fun move()      // определение функции без реализации
    fun stop(){     // определение функции с реализацией по умолчанию
        println("Остановка")
    }
}
```

Например, в данном случае интерфейс Movable представляет функционал транспортного средства. Он содержит две функции и одно свойство. Функция `move()` представляет абстрактный метод - она не имеет реализации. Вторая функция `stop()` имеет реализацию по умолчанию.

При определении свойств в интерфейсе им не присваиваются значения.

Мы не можем напрямую создать объект интерфейса, так как интерфейс не поддерживает конструкторы и просто представляет шаблон, которому класс должен соответствовать.

Определим два класса, которые применяют интерфейс:

```
class Car : Movable{

    override var speed = 60
    override fun move(){
        println("Машина едет со скоростью $speed км/ч")
    }
}
class Aircraft : Movable{

    override var speed = 600
    override fun move(){
        println("Самолет летит со скоростью $speed км/ч")
    }
    override fun stop(){
        println("Приземление")
    }
}
```

Для применения интерфейса после имени класса ставится двоеточие, за которым следует название интерфейса. При применении интерфейса класс должен реализовать все его абстрактные методы и свойства, а также может предоставить свою реализацию для тех свойств и методов, которые уже имеют реализацию по умолчанию. При реализации функций и свойств перед ними ставится ключевое слово **override**.

Так, класс `Car` представляет машину и применяет интерфейс `Movable`. Так как интерфейс содержит абстрактный метод `move()`, то класс `Car` обязательно должен его реализовать.

Тоже касается свойства `speed` - класс Car должен его определить. Здесь реализация свойства заключается в установке для него начального значения.

А вот функцию `stop()` класс Car может не реализовать, так как она уже содержит реализацию по умолчанию.

Класс Aircraft представляет самолет и тоже применяет интерфейс Movable. При этом класс Aircraft реализует обе функции интерфейса.

В последствии в программе мы можем рассматривать объекты классом Car и Aircraft как объекты Movable:

```
fun main() {

    val m1: Movable = Car()
    val m2: Movable = Aircraft()
    // val m3: Movable = Movable() напрямую объект интерфейса создать нельзя

    m1.move()
    m1.stop()
    m2.move()
    m2.stop()
}
```

Консольный вывод программы:

```
Машина едет со скоростью 60 км/ч
Останавливается
Самолет летит со скоростью 600 км/ч
Самолет приземляется
```

Реализация свойств

Рассмотрим еще пример. Определим интерфейс Info, который объявляет ряд свойств:

```
interface Info{
    val model: String
        get() = "Undefined"
    val number: String
}
```

Первое свойство имеет геттер, а это значит, что оно имеет реализацию по умолчанию. При применении интерфейса такое свойство необязательно реализовать. Второе свойство - number является абстрактным, оно не имеет ни геттера, ни сеттера, то есть не имеет реализации по умолчанию, поэтому классы его обязаны реализовать.

Для реализации интерфейса возьмем выше определенный класс Car:

```
class Car(override val model: String, override var number: String) : Movable, Info{

    override var speed = 60
    override fun move(){
        println("Машина едет со скоростью $speed км/ч")
    }
}
```

Теперь класс Car применяет два интерфейса. Класс может применять несколько интерфейсов, в этом случае они указываются через запятую, и все эти интерфейсы класс должен реализовать. Класс Car реализует оба свойства. При этом при реализации свойств в классе необязательно указывать геттер или сеттер. Кроме того, можно реализовать свойства в первичном конструкторе, как это сделано в случае со свойствами model и number.

Применение класса:

```
fun main() {

    val tesla: Car = Car("Tesla", "2345SDG")
    println(tesla.model)
    println(tesla.number)

    tesla.move()
    tesla.stop()
}
```

Правила переопределения

В Kotlin мы можем наследовать класс и применять интерфейсы. При этом мы можем одновременно и наследоваться от класса, и применять один или несколько интерфейсов. Однако, что если переопределяемая функция из базового класса имеет то же имя, что и функция из применяемого интерфейса:

```
open class Video {
    open fun play() { println("Play video") }
}

interface AudioPlayable {
    fun play() { println("Play audio") }
}

class MediaPlayer() : Video(), AudioPlayable {
    // Функцию play обязательно надо переопределить
    override fun play() {
        super<Video>.play()         // вызываем Video.play()
        super<AudioPlayable>.play() // вызываем AudioPlayable.play()
    }
}
```

Здесь класс Video и интерфейс AudioPlayable определяют функцию play. В этом случае класс MediaPlayer, который наследуется от Video и применяет интерфейс AudioPlayable, обязательно должен определить функцию с тем же именем, то есть play. С помощью конструкции `super<имя_типа>.имя_функции` можно обратиться к опредленной реализации либо из базового класса, либо из интерфейса.