Лекция #9.3 Переопределение методов и свойств

Kotlin позволяет переопределять в производном классе функции и свойства, которые определенны в базовом классе. Чтобы функции и свойства базового класа можно было переопределить, к ним применяется аннотация **open**. При переопределении в производном классе к этим функциям применяется аннотация **override**.

Переопределение свойств

Чтобы указать, что свойство можно переопределить в производном классе, перед его определением указывается ключевое слово **open**:

```
open class Person(val name: String){
    open var age: Int = 1
}
```

В данном случае свойство age доступно для переопределения.

Если свойство определяется через первичный конструктор, то также перед его определением ставится аннотация **open**:

```
open class Person(val name: String, open var age: Int = 1){
}
```

В производном классе для переопределения свойства перед ним указывается аннотация **override**.

```
open class Person(val name: String, open var age: Int = 1){
}

open class Employee(name: String): Person(name){

    override var age: Int = 18
}
```

Здесь переопределение заключается в изменении начального значения для свойства age.

Также переопределить свойство можно сразу в первичном конструкторе:

```
open class Person(val name: String, open var age: Int = 1){
}

open class Employee(name: String, override var age: Int = 18): Person(name, age){}
```

Применение:

```
fun main() {

    val tom = Person("Tom")
    println("Name: ${tom.name}  Age: ${tom.age}")

    val bob = Employee("Bob")
    println("Name: ${bob.name}  Age: ${bob.age}")
}
open class Person(val name: String, open var age: Int = 1)

open class Employee(name: String, override var age: Int = 18): Person(name, age)
```

Консольный вывод:

```
Name: Tom  Age: 1
Name: Bob  Age: 18
```

Переопределение геттеров и сеттеров

Также можно переопределять геттеры и сеттеры свойств:

```
open class Person(val name: String){

    open val fullInfo: String
        get() = "Person $name - $age"

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

```
open class Employee(name: String): Person(name){

    override val fullInfo: String
        get() = "Employee $name - $age"

    override var age: Int = 18
        set(value){
            if(value > 17 && value < 110)
                field = value
        }
}

fun main() {

    val tom = Person("Tom")
    tom.age = 14
    println(tom.fullInfo)

    val bob = Employee("Bob")
    bob.age = 14
    println(bob.fullInfo)
}
```

Здесь класс Employee переопределяет геттер свойства fullInfo и сеттер свойства age

Переопределение методов

Чтобы функции базового класа можно было переопределить, к ним применяется аннотация open. При переопределении в производном классе к этим функциям применяется аннотация override:

```
open class Person(val name: String){
    open fun display(){
        println("Name: $name")
    }
}
class Employee(name: String, val company: String): Person(name){

    override fun display() {
        println("Name: $name    Company: $company")
    }
}
fun main() {

    val tom = Person("Tom")
    tom.display()       // Name: Tom

    val bob = Employee("Bob", "JetBrains")
    bob.display()       // Name: Bob  Company: JetBrains
}

```

Функция display определена в классе Person с аннотацией **open**, поэтому в производных классах его можно переопределить. В классе Employee эта функция переопределена с применением аннотации **override**.

Переопределение в иерархии наследования классов

Стоит учитывать, что переопределить функции можно по всей иерархии наследования. Например, у нас может быть класс Manager, унаследованный от Employee:

```
open class Person(val name: String){
    open fun display(){
        println("Name: $name")
    }
}
open class Employee(name: String, val company: String): Person(name){

    override fun display() {
        println("Name: $name    Company: $company")
    }
}
class Manager(name: String, company: String):Employee(name, company){
    override fun display() {
        println("Name: $name Company: $company  Position: Manager")
    }
}
```

В данном случае класс Manager переопределяет функцию display, поскольку среди его базовых классов есть класс Person, который определяет эту функцию с ключевым словом **open**.

Запрет переопределения

В это же время иногда бывает необходимо запретить дальнейшее переопределение функции в классах-наследниках. Для этого применяется ключевое слово **final**:

```
open class Person(val name: String){
    open fun display(){
        println("Name: $name")
    }
}
open class Employee(name: String, val company: String): Person(name){

    final override fun display() {
        println("Name: $name    Company: $company")
    }
}
class Manager(name: String, company: String):Employee(name, company){
    // теперь функцию нельзя переопределить
    /*override fun display() {
        println("Name: $name Company: $company  Position: Manager")
    }*/
}
```

Обращение к реализации из базового класса

С помощью ключевого слова **super** в производном классе можно обращаться к реализации из базового класса.

```
open class Person(val name: String){

    open val fullInfo: String
        get() = "Name: $name"

    open fun display(){
        println("Name: $name")
    }
}
open class Employee(name: String, val company: String): Person(name){

    override val fullInfo: String
        get() = "${super.fullInfo} Company: $company"

    final override fun display() {
        super.display()
        println("Company: $company")
    }
}
```

В данном случае производный класс Employee при переопределении свойства и функции применяет реализацию из базового класса Person. Например, через `super.fullInfo` возвращается значение свойства из базового класса (то есть значение свойства name), а с помощью вызова `super.display()` вызывается реализация функции display из класса Person.