Лекция #9.6 Вложенные классы и интерфейсы

В Kotlin классы и интерфейсы могут быть определены в других классах и интерфейсах. Такие классы (вложенные классы или **nested classes**) обычно выполняют какую-то вспомогательную роль, а определение их внутри класса или интерфейса позволяет разместить их как можно ближе к тому месту, где они непосредственно используются.

Например, в следующем случае определяется вложенный класс:

```
class Person{
    class Account(val username: String, val password: String){

        fun showDetails(){
            println("UserName: $username  Password: $password")
        }
    }
}
```

В данном случае класс Account является вложенным, а класс Person - внешним.

По умолчанию вложенные классы имеют модификатор видимости public, то есть они видимы в любой части программы. Но для обращения к вложенному классу надо использовать имя внешнего класса. Например, создание объекта вложенного класса:

```
fun main() {

    val userAcc = Person.Account("qwerty", "123456");
    userAcc.showDetails()
}
```

Если необходимо ограничить область применения вложенного класса только внешним классом, то следует определить вложенный класс с модификатором private:

```
class Person(username: String, password: String){

    private val account: Account = Account(username, password)

    private class Account(val username: String, val password: String)

    fun showAccountDetails(){
        println("UserName: ${account.username}  Password: $account.password")
    }
}
fun main() {

    val tom = Person("qwerty", "123456");
    tom.showAccountDetails()
}
```

Классы также могут содержать вложенные интерфейсы. Кроме того, интерфейсы тоже могут содержать вложенные классы и интерфейсы:

```
interface SomeInterface {
    class NestedClass
    interface NestedInterface
}

class SomeClass {
    class NestedClass
    interface NestedInterface
}
```

Внутренние (inner) классы

Стоит учитывать, что вложенный (nested) класс по умолчанию не имеет доступа к свойствам и функциям внешнего класса. Например, в следующем случае при попытке обратиться к свойству внешнего класса мы получим ошибку:

```
class BankAccount(private var sum: Int){

    fun display(){
        println("sum = $sum")
    }

    class Transaction{
        fun pay(s: Int){
            sum -= s
            display()
        }
    }
}
```

В данном случае у нас определен класс банковского счета BankAccount, который определяет свойство `sum` - сумма на счете и функцию `display()` для вывода информации о счете.

Кроме того, в классе BankAccount определен вложенный класс Transaction, который представляет операцию по счету. В данном случае класс Transaction определяет функцию `pay()` для оплаты со счета. Однако в нем мы не можем обратиться к свойствам и функциям внешнего класса BankAccount.

Чтобы вложенный класс мог иметь доступ к свойствам и функциям внешнего класса, необходимо определить вложенный класс с ключевым словом **inner**. Такой класс еще называют внутренним классом (inner class), чтобы отличать от обычных вложенных классов. Например:

```
fun main() {

    val acc = BankAccount(3400);
    acc.Transaction().pay(2500)
}
class BankAccount(private var sum: Int){

    fun display(){
        println("sum = $sum")
    }

    inner class Transaction{
        fun pay(s: Int){
            sum -= s
            display()
        }
    }
}
```

Теперь класс Transaction определен с ключевым словом inner, поэтому имеет полный доступ к свойствам и функциям внешнего класса BankAccount. Но теперь если мы хотим использовать объект подобного вложенного класса, то необходимо создать объект внешнего класса:

```
val acc = BankAccount(3400);
    acc.Transaction().pay(2500)
```

Совпадение имен

Но что, если свойства и функции внутреннего класса называются также, как и свойства и функции внешнего класса? В этом случае внутренний класс может обратиться к свойствам и функциям внешнего через конструкцию `this@название_класса.имя_свойства_или_функции`:

```
class A{
    private val n: Int = 1
    inner class B{
        private val n: Int = 1
        fun action(){
            println(n)          // n из класса B
            println(this.n)     // n из класса B
            println(this@B.n)   // n из класса B
            println(this@A.n)   // n из класса A
        }
    }
}
```

Например, перепишем случай выше с классами Account и Transaction следующим образом:

```
fun main() {

    val acc = BankAccount(3400);
    acc.Transaction(2400).pay()
}
class BankAccount(private var sum: Int){

    fun display(){
        println("sum = $sum")
    }

    inner class Transaction(private var sum: Int){
        fun pay(){
            this@BankAccount.sum -= this@Transaction.sum
            display()
        }
    }
}
```