Лекция #9.10 Анонимные классы и объекты

Иногда возникает необходимость создать объект некоторого класса, который больше нигде в программе не используется. То есть класс необходим только для создания только одного объекта. В этом случае мы, конечно, можем, как и обычно, определить класс и затем создать объект этого класса. Но Kotlin для таких ситуаций предоставлять возможность определить объект анонимного класса.

Анонимные классы не используют ключевое слово **class** для определения. Они не имеют имени, но, как и обычные классы, могут наслдовать другие классы или применять интерфейсы. Объекты анонимных классов называют анонимыми объктами.

Для определения анонимного объекта применяется ключевое слово **object**:

```
fun main() {

    val tom = object {
        val name = "Tom"
        var age = 37
        fun sayHello(){
            println("Hi, my name is $name")
        }
    }
    println("Name: ${tom.name}  Age: ${tom.age}")
    tom.sayHello()
}
```

После ключевого слова object идет блок кода в фигурных скобках, в которые помещается определение объекта. Как и в обычном классе, анонимный объект может содержать свойства, функции. И далее по имени переменной мы можем обращаться к свойствам и функциям этого объекта.

Наследование анонимных объектов

При наследовании после слова **object** через двоеточия указывается имя наследуемого класса или его первичный конструктор:

```
fun main() {

    val tom = object : Person("Tom"){

        val company = "JetBrains"
        override fun sayHello(){
            println("Hi, my name is $name. I work in $company")
        }
    }

    tom.sayHello()  // Hi, my name is Tom. I work in JetBrains
}
open class Person(val name: String){
    open fun sayHello(){
        println("Hi, my name is $name")
    }
}
```

Здесь класс анонимного объекта наследует класс Person и переопределяет его функцию `sayHello()`.

Анонимный объект как аргумент функции

Анонимный объект может передаваться в качестве аргумента в вызов функции:

```
fun main() {
    hello(
        object : Person("Sam"){
            val company = "JetBrains"
            override fun sayHello(){
                println("Hi, my name is $name. I work in $company")
            }
    })
}
fun hello(person: Person){
    person.sayHello()
}
open class Person(val name: String){
    open fun sayHello() = println("Hi, my name is $name")
}
```

Здесь поскольку класс анонимного объекта наследуется от класса Person, мы можем передавать этот анонимный объект параметру функции, который имеет тип Person.

Анонимный объект как результат функции

Функция может возвращать анонимный объект:

```
fun main() {
    val tom = createPerson("Tom", "JetBrains")
    tom.sayHello()
}
private fun createPerson(_name: String, _company: String) = object{
    val name = _name
    val company = _company
    fun sayHello() = println("Hi, my name is $name. I work in $company")
}
```

Однако тут есть нюансы. Чтобы мы могли обращаться к свойствам и функциям анонимного объекта, функция, которая возвращает этот объект, должна быть приватной, как в примере выше.

Если функция имеет модификатор `public` или `private inline`, то в этом случае свойства и функции анонимного класса (за исключением унаследованных) недоступны:

```
fun main() {
    val tom = createPerson("Tom", "JetBrains")
    println(tom.name)   // норм - свойство name унаследовано от Person
    println(tom.company)    // ! Ошибка - свойство недоступно
}
private inline fun createPerson(_name: String, _comp: String) = object: Person(_name){
    val company = _comp
}

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

В данном случае функция `createPerson()` имеет модификатор `private inline`, поэтому у анонимного объекта будут доступны только унаследованные свойства и функции от класса Person, но собственные свойства и функции будут не доступны.