Лекция #8. Классы и объекты

Поговорим о более серьёзных вещах. **Kotlin** поддерживает объектно-ориентированную парадигму программирования, а это значит, что программу на данном языке можно представить в виде взаимодействующих между собой объектов.

Представлением объекта является класс. Класс фактически представляет определение объекта. А объект является конкретным воплощением класса. Например, у всех есть некоторое представление о машине, например, кузов, четыре колеса, руль и т.д. - некоторый общий набор характеристик, присущих каждой машине. Это представление фактически и является классом. При этом есть разные машины, у которых отличается форма кузова, какие-то другие детали, то есть есть конкретные воплощения этого класса, конкретные объекты или экземпляры класса.

Для определения класса применяется ключевое слово **class**, после которого идет имя класса. А после имени класса в фигурных скобках определяется тело класса. Если класс не имеет тела, то фигурные скобки можно опустить. Например, определим класс, который представляет человека:

```
1  class Person
2
3  // либо можно так
4  class Person { }
```

Класс фактически представляет новый тип данных, поэтому мы можем определять переменные этого типа:

```
fun main() {

    val tom: Person
    val bob: Person
    val alice: Person
}

class Person
```

В функции **main** определены три переменных типа **Person**. Стоит также отметить, что в отличие от других объектно-ориентированных языков (как **C#** или **Java**), функция **main** в **Kotlin** не помещается в отдельных класс, а всегда определяется **вне какого-либо класса.**

Для создания объекта класса необходимо вызвать конструктор данного класса. Конструктор фактически представляет функцию, которая называется по имени класса и которая выполняет инициализацию объекта. По умолчанию для класса компилятор генерирует пустой конструктор, который мы можем использовать:

```
val tom: Person = Person()
```

Часть кода после знака равно **Person()** как раз и представляет вызов конструктора, который создает объект класса **Person**. До вызова конструктора переменная класса не указывает ни на какой объект.

Например, создадим три объекта класса **Person**:

```
fun main() {

    val tom: Person = Person()
    val bob: Person = Person()
    val alice: Person = Person()
}

class Person
```

Свойства

Каждый класс может хранить некоторые данные или состояние в виде свойств. Свойства представляют переменные, определенные на уровне класса с ключевыми словами **val** и **var**. Если свойство определено с помощью **val**, то значение такого свойства можно установить только один раз, то есть оно **immutable**. Если свойство определено с помощью **var**, то значение этого свойства можно многократно изменять.

Свойство должно быть инициализировано, то есть обязательно должно иметь начальное значение. Например, определим пару свойств:

```
class Person{
    var name: String = "Undefined"
    var age: Int = 18
}
```

В данном случае в классе **Person**, который представляет человека, определены свойства **name** (имя человека) и **age** (возраст человека). И эти свойства инициализированы начальными значениями.

Поскольку эти свойства определены с **var**, то мы можем изменить их начальные значения:

```
fun main() {

    val bob: Person = Person()  // создаем объект
    println(bob.name)       // Undefined
    println(bob.age)        // 18

    bob.name = "Bob"
    bob.age = 25

    println(bob.name)       // Bob
    println(bob.age)        // 25
}

class Person{
    var name: String = "Undefined"
    var age: Int = 18
}
```

Для обращения к свойствам используется имя переменной, которая предствляет объект, и после точки указывается имя свойства. Например, получение значения свойства:

```
val personName : String = bob.name
```

Установка значения свойства:

```
bob.name = "Bob"
```

Функции класса

Класс также может содержать функции. Функции определяют поведение объектов данного класса. Такие функции еще называют **member functions** или функции-члены класса. Например, определим класс с функциями:

```
class Person{
    var name: String = "Undefined"
    var age: Int = 18

    fun sayHello(){
        println("Hello, my name is $name")
    }

    fun go(location: String){
        println("$name goes to $location")
    }

    fun personToString() : String{
        return "Name: $name  Age: $age"
    }
}
```

Функции класса определяется также как и обычные функции. В частности, здесь в классе **Person** определена функция **sayHello()**, которая выводит на консоль строку "**Hello**" и эмулирует приветствие объекта **Person**. Вторая функция - **go()** эмулирует движение объекта **Person** к определенному местоположению. Местоположение передается через параметр **location**. И третья функция **personToString()** возвращает информацию о текущем объекте в виде строки.

В функциях, которые определены внутри класса, доступны свойства этого класса. Так, в данном случае в функциях можно обратиться к свойствам **name** и **age**, которые определены в классе **Person**.

Для обращения к функциям класса необходимо использовать имя объекта, после которого идет название функции и в скобках значения для параметров этой функции:

```
fun main() {

    val tom = Person()
    tom.name = "Tom"
    tom.age = 37

    tom.sayHello()
    tom.go("the shop")
    println(tom.personToString())

}
```

Конструкторы

Для создания объекта необходимо вызвать конструктор класса. По умолчанию компилятор создает конструктор, который не принимает параметров и который мы можем использовать. Но также мы можем определять свои собственные конструкторы. Для определения конструкторов применяется ключевое слово **constructor**.

Классы в **Kotlin** могут иметь один **первичный конструктор** (primary constructor) и один или несколько **вторичных конструкторов** (secondary constructor).

Первичный конструктор

Первичный конструктор является частью заголовка класса и определяется сразу после имени класса:

```
class Person constructor(_name: String){

}
```

Конструкторы, как и обычные функции, могут иметь параметры. Так, в данном случае конструктор имеет параметр **_name**, который представляет тип **String**. Через параметры конструктора мы можем передать извне данные и использовать их для инициализации объекта. При этом первичный конструктор в отличие от функций не определяет никаких действий, он только может принимать данные извне через параметры.

Если первичный конструктор не имеет никаких аннотаций или модификаторов доступа, как в данном случае, то ключевое слово **constructor** можно опустить:

```
class Person(_name: String){

}
```

Инициализатор

Что делать с полученными через конструктор данными? Мы их можем использовать для инициализации свойств класса. Для этого применяются блоки инициализаторов:

```
class Person(_name: String){
    val name: String
    init{
        name = _name
    }
}
```

В классе **Person** определено свойство **name**, которое хранит имя человека. Чтобы передать эту свойству значение параметра **_name** из первичного конструктора, применяется блок инициализатора. Блок инициализатора определяется после ключевого слова **init**.

Цель инициализатора состоит в инициализации объекта при его создании. Стоит отметить, что здесь свойству **name** не задается начальное значение, потому это свойство в любом случае будет инициализировано в блоке инициализатора, и при создании объекта оно в любом случае получит значение.

Теперь мы можем использовать первичный конструктор класса для создания объекта:

```
fun main() {
    val tom = Person("Tom")
    val bob = Person("Bob")
    val alice = Person("Alice")

    println(tom.name)   // Tom
    println(bob.name)   // Bob
    println(alice.name) // Alice
}

class Person(_name: String){
    val name: String
    init{
        name = _name
    }
}
```

Важно учитывать, что если мы определили первичный конструктор, то мы не можем использовать конструктор по умолчанию, который генерируется компилятором. Для создания объекта обязательно надо использовать первичный конструктор, если он определен в классе.

Стоит отметить, что в классе может быть определено одновременно несколько блоков инициализатора.

Также стоит отметить, что в данном случае в инициализаторе нет смысла, так как параметры первичного конструктора можно нарямую передавать свойствам:

```
class Person(_name: String){

    val name: String = _name
}
```

Первичный конструктор и свойства

Первичный конструктор также может использоваться для определения свойств:

```
fun main() {

    val bob: Person = Person("Bob", 23)

    println("Name: ${bob.name}  Age: ${bob.age}")
}

class Person(val name: String, var age: Int){

}
```

Свойства определяются как и параметры, при этом их определение начинается с ключевого слова **val** (если их не планируется изменять) и **var** (если свойства должны быть изменяемыми). И в этом случае нам уже необязательно явным образом определять эти свойства в теле класса, так как их уже определяет конструктор. И при вызове конструктора этим свойствам автоматически передаются значения: **Person("Bob", 23)**

Вторичные конструкторы

Класс также может определять вторичные конструкторы. Они применяются в основном, чтобы определить дополнительные параметры, через которые можно передавать данные для инициализации объекта.

Вторичные конструкторы определяются в теле класса. Если для класса определен первичный конструктор, то вторичный конструктор должен вызывать первичный с помощью ключевого слова **this**:

```
class Person(_name: String){
    val name: String = _name
    var age: Int = 0

    constructor(_name: String, _age: Int) : this(_name){
        age = _age
    }
}
```

Здесь в классе **Person** определен первичный конструктор, который принимает значение для установки свойства **name**:

```
class Person(_name: String)
```

И также добавлен вторичный конструктор. Он принимает два параметра: **_name** и **_age**. С помощью ключевого слова **this** вызывается первичный

конструктор, поэтому через этот вызов необходимо передать значения для параметров первичного конструктора. В частности, в первичный конструктор передается значение параметра **_name**. В самом вторичном конструкторе устанавливается значение свойства **age**.

```
constructor(_name: String, _age: Int) : this(_name){
    age = _age
}
```

Таким образом, при вызове вторичного конструктора вначале вызывается первичный конструктор, срабатывает блок инициализатора, который устанавливает свойство **name**. Затем выполняются собственно действия вторичного конструктора, который устанавливает свойство **age**.

Используем данную модификацию класса **Person**:

```
fun main() {

    val tom: Person = Person("Tom")
    val bob: Person = Person("Bob", 45)

    println("Name: ${tom.name}  Age: ${tom.age}")
    println("Name: ${bob.name}  Age: ${bob.age}")
}

class Person(_name: String){
    val name: String = _name
    var age: Int = 0

    constructor(_name: String, _age: Int) : this(_name){
        age = _age
    }
}
```

В функции **main** создаются два объекта **Person**. Для создания объекта tom применяется первичный конструктор, который принимает один параметр. Для создания объекта **bob** применяется вторичный конструктор с двумя параметрами.