Конспект 28.02.2025 по теме “Классы и объекты в Kotlin”

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.

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:

class Person

// либо можно так

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

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}

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"}}

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())}

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

fun main() { }

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"} }

Консольный вид программы:

Hello, my name is Tom

Tom goes to the shop

Name: Tom Age: 37

**Конструкторы**

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**

Для инициализации свойств класса 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}}

Также стоит отметить, что в данном случае в инициализаторе нет смысла, так как

параметры первичного конструктора можно напрямую передавать свойствам:

class Person(\_name: String){

val name: String = \_name}