

# Классы

## Улучшения в Котлин

### Содержание

1. Синглтон паттерн в Джава
2. Object и другие продвинутые виды классов в Котлин

#### 1. Синглтон паттерн в Джава

Иногда возникает ситуация в которой нам необходим 1 единственный инстанс класса – т.е. чтобы нельзя создать более 1 объекта конкретного класса. Например вы храните кеш данных в объекте А, но если из другого места кто-то другой попыбует получить его, то он не зная о существовании объекта попыбует создать новый инстанс и тем самым у нас будет 2 объекта, в первом будет кеш данных, а во втором не будет. Итак, что же нам делать в таком случае? Если мы создаем объект класса через конструктор, то как мы можем запретить этого делать другим? Помните про модификаторы доступа? Мы же до сих пор писали конструктор публик. А что если сделать его private?

```
public class Singleton {  
    private Singleton() {  
        //no one can call this outside  
    }  
}
```

И да, если я не говорил ранее – в джава есть комментарии – любая линия кода если начинается с // то компилятор будет его игнорировать. Давайте проверим что мы не можем вызвать конструктор этого класса. (комменты удобно юзать через Ctrl+ '/')

```
public static void main(String[] args) {  
    Singleton singleton = new Singleton();  
}
```

'Singleton()' has private access in 'Singleton'

Ну вот. Мы создавали объекты класса через конструктор, но и это не всегда возможно. Да, в этом была наша цель. Запретить создание. Теперь перейдем к такому вопросу – если извне создать объект нельзя, значит надо создавать его внутри класса. А как тогда дать доступ? И здесь нам поможет статик.

Для проверки мы залогируем вызов конструктора в консоль. Теперь посмотрите на это.

```

public class Singleton {

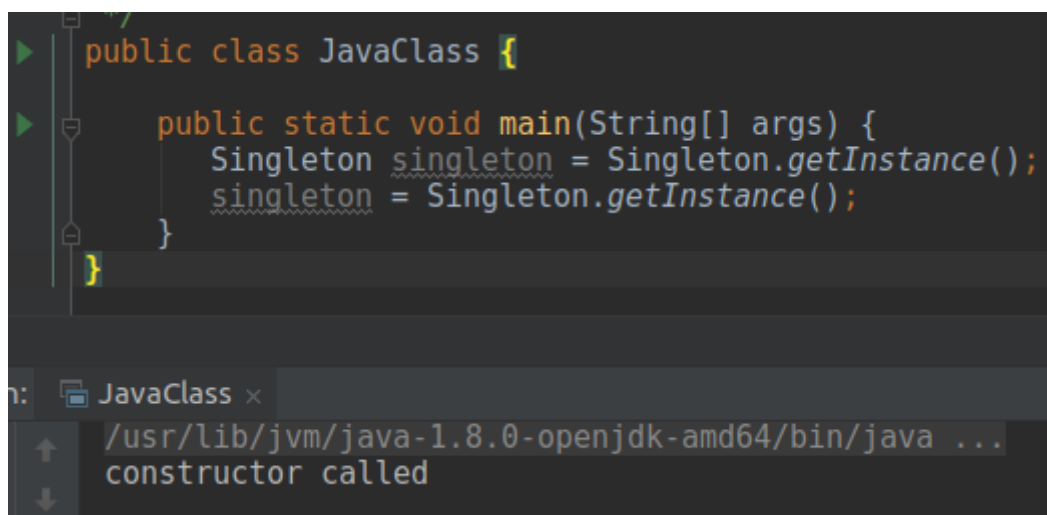
    private static Singleton sSingleton;

    public static Singleton getInstance() {
        if (sSingleton == null) {
            sSingleton = new Singleton();
        }
        return sSingleton;
    }

    private Singleton() {
        //no one can call this outside
        System.out.println("constructor called");
    }
}

```

Если в нашей статик переменной ничего нет, то мы создаем объект класса и кладем его туда. Но если переменная была уже инициализирована, то вернем что там было. Давайте проверим



```

public class JavaClass {

    public static void main(String[] args) {
        Singleton singleton = Singleton.getInstance();
        singleton = Singleton.getInstance();
    }
}

```

Output: constructor called

Мы 2 раза вызвали метод получения инстанса, но конструктор был вызван единожды. В этом и суть патерна синглтон. Конечно же оно в таком виде не будет работать с многопоточностью, но вы можете самостоятельно погуглить этот вопрос. Теперь, как же это все делается в Котлин?

## 2. Object и другие продвинутые виды классов в котлин

Но как видите вызов методов будет выглядеть как вызов публик статик метода в джава. Все потому что по сути своей мы и не создаем объекта типа Singleton в котлин. У нас будто бы хранилище данных. Но мы точно знаем что обращаясь к методам этого класса мы будем иметь дело с одними и теми же переменными внутри этого класса.

Вообще конечно синглтон это антипатерн, т.е. использовать его нужно только тогда, когда нет других решений. В Android по крайней мере вам не нужно будет создавать синглтоны. Но знать об этом патерне вам нужно, потому что многие библиотеки используют этот патерн.

```

@JvmStatic
fun main(args: Array<String>) {
    Singleton.increment()
    Singleton.increment()
    Singleton.print()
}

object Singleton {

    private var x = 0

    fun increment() {
        x++
    }

    fun print() {
        print(x)
    }
}

```

Ладно. Давайте рассмотрим обычный класс в котлин.

```

class SimpleClass {
    private val data: Int
}

```

Так, мы написали простой класс и сделали ему 1 поле. И теперь надо сгенерировать конструктор для инициализации как было в джаве, так? Давайте попробуем – Alt+Enter.

```

class SimpleClass(private val data: Int) {
}

```

Как это называется? Аргумент конструктора ушел в объявление класса? Да, это еще одно сокращение от Котлина. Ведь если у нас уже есть объявление класса, то почему бы не переиспользовать его для конструктора? Хорошо, скажет мне наш джава-любитель. А что если у меня несколько перегруженных конструкторов? Ну окей. Это можно сделать с помощью дефолтных значений.

Кстати, вы заметили? Мы больше не пишем слово new. Еще одно сокращение от Котлин. Просто имя класса и круглые скобки. Не передали аргумент – берем дефолтное, передали – юзаем его. Ок. А что если мне реально не хочется так делать и я хочу много разных конструкторов? Можно сделать так, ладно. Посмотрите.

```

@JvmStatic
fun main(args: Array<String>) {
    val simpleClass = SimpleClass()
    val other = SimpleClass( data: 1)
}

class SimpleClass(private val data: Int = 0) {
}

```

```

@JvmStatic
fun main(args: Array<String>) {
    val simpleClass = SimpleClass()
    val other = SimpleClass( number: 1)
    val new = SimpleClass( text: "text")
}

class SimpleClass {

    private val data: Int

    constructor(text: String = "") {
        data = text.length
    }

    constructor(number: Int) {
        data = number
    }
}

```

В котлин сделали ключевое слово constructor и туда можете передавать какие угодно аргументы. Но, стоп, что насчет вызова конструктора из другого?

```

class SimpleClass {

    private val data: Int

    constructor(text: String = "") : this(text.length)

    constructor(number: Int) {
        data = number
    }
}

```

Тоже можно, но не так как в джава. Здесь синтаксис немного другой. Если после этого вам нужно дописать еще кода – откройте фигурные скобки.

Предположим у вас много конструкторов, но вам нужно выполнить какой-то код в моменте инициализации объекта. В джава это можно сделать блоком кода. Давайте посмотрим.

```
public class JavaClass {  
    public JavaClass() {  
        System.out.println("constructor called");  
    }  
    {  
        System.out.println("some code done at start");  
    }  
}
```

Да, я вам не рассказывал об этом когда мы проходили джава. Но теперь вы знаете.

Посмотрите внимательно на консоль. Как думаете какой код будет сделан раньше: тот что в конструкторе или в блоке?

```
public static void main(String[] args) {  
    JavaClass javaClass = new JavaClass();  
}  
}
```

Singleton > main()

Run: Singleton x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  
some code done at start  
constructor called

Process finished with exit code 0

Неожиданно, но в блоке кода. Это может быть полезно если у вас просто нет конструктора и вы не хотите его писать или же просто нужно сделать что-то перед созданием объекта. Так как это редко используется в джаве, мы проходим это в разделе котлина. В котлин коде мы часто делаем что-то на старте. Но оно выглядит иначе. Посмотрим.

И да, вы заметили, что в котлин мы можем написать несколько классов в одном файле? Не вложенные классы, а именно подряд. В джава нужно писать 1 класс в 1 файле или же вложенные классы делать и создавать зависимости. Но сразу предупрежу – не злоупотребляйте. Мы всегда смотрим на имя файла когда ищем что-то и должны быстро найти что-то. Не делайте файлов с несвязанными между собой классами.

На самом деле вы можете никогда и не писать инит блок в котлин коде, но многие используют его и вам нужно хотя бы знать что это.

Ладно, какие еще фишки есть у Котлина, которых не было у джава?

```

object Main {
    @JvmStatic
    fun main(args: Array<String>) {
        val simpleClass = SimpleClass()
    }
}

class SimpleClass {
    constructor() {
        println("constructor call")
    }

    init {
        println("init kotlin SimpleClass")
    }
}

```

```

Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64
init kotlin SimpleClass
constructor call

Process finished with exit code 0

```

Помните я говорил про то, что не нужно делать приватные поля и давать им публик геттеры и сеттеры? Да, то же самое правило касается и котлина. Но по крайней мере в Android фреймворке очень много кода, где объекту можно задать свойства через сеттер и получать их через геттер. Джава код вам будет ясен и понятен. Но в котлин есть маленький трюк. Посмотрите на это. Кстати, я не говорил что в котлин коде можно спокойно общаться с джава кодом?

```

@JvmStatic
fun main(args: Array<String>) {
    val javaClass = JavaClass()
    javaClass.setI
}

```

Int  
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards >>

```

@JvmStatic
fun main(args: Array<String>) {
    val javaClass = JavaClass()
    javaClass.setI(3)
}

```

```
public class JavaClass {
    private int i;

    public int getI() {
        return i;
    }

    public void setI(int i) {
        this.i = i;
    }
}
```

Как видите в котлине можем обращаться к геттерам и сеттерам так же как будто поля не были приватными, а были публичными.

```
fun main(args: Array<String>) {
    val javaClass = JavaClass()
    javaClass.i = 3
}
```

Еще пара слов про котлин классы – все так же можно сделать приватный конструктор как в джава, так же как в джава можно сделать вложенные классы. Но посмотрите на это.

```
public class JavaClass {
    class NestedJava {
        private final int s;

        public NestedJava(int s) {
            this.s = s;
        }
    }
}
```

Вот мы написали вложенный класс в джаве. Но можем ли породить объект этого класса?

```
public static void main(String[] args) {
    JavaClass javaClass = new JavaClass();
    JavaClass.NestedJava nested = new JavaClass.NestedJava(4);
}
```

'JavaClass' is not an enclosing class

Нет. Нужно сделать вложенный класс статическим. А как обстоят дела в котлин?

Если вы еще не поняли – в котлине исправлено много всего что было плохо в джава. И вложенные классы в том числе. В котлин просто нет слова статик и потому статик классов быть не может. Но если вы хотите сделать невидимым этот вложенный класс то просто объявите его приватным. Кстати, я вам не говорил, но внутри интерфейса тоже можно положить классы.

```

@JvmStatic
fun main(args: Array<String>) {
    val nested = KotlinClass.NestedKotlin(5)
    nested.print()
}

class KotlinClass {
    class NestedKotlin(private val i: Int) {
        fun print() {
            print("hello from nested")
            print(i)
        }
    }
}

```

Если вы хотите чтобы вложенный класс в Kotlin был так же невидим можно объявить его `inner` т.е. внутренним. Больше деталей можете найти здесь. <https://kotlinlang.org/docs/>

И еще в тему `object`. Помните мы писали анонимный класс для `Runnable` чтобы передать в класса потока? В Kotlin так же можно сделать, но синтаксис другой

```

fun main(args: Array<String>) {
    Thread(object : Runnable {
        override fun run() {
            // ...
        }
    })
}

```

И конечно же лямбды присутствуют и здесь. Но по крайней мере здесь нет аннотации `@Override` а вместо этого ключевое слово `override` что читается проще. Но лямбды выглядят в Kotlin немного иначе. Это именно пример анонимного класса.

```

@JvmStatic
fun main(args: Array<String>) {
    Thread(Runnable { })
}

```

Лямбды в Kotlin пройдем в другой лекции.

Мы не закончили с классами, еще минимум 2 классных фишки в Kotlin мы рассмотрим в следующей лекции – дата классы и sealed классы.

А пока опять же вам задание – найдите свой старый код на Java и попробуйте переписать его на Kotlin.