Московский авиационный институт   
(государственный технический университет)   
  
Факультет прикладной математики   
  
Кафедра вычислительной математики и программирования

Реферат по теме

“Котлин – язык программирования”

Студент: Стрыгин Д.Д.

Группа: М8О-106Б-19, №22

Преподаватель: Дубинин А.В.

**Содержание:**

Введение……………………………………………………………………………………3

Основные возможности и преимущества…..……………………………3

Null безопасность………………………………………………………………………5

Отличие от Java………………………………………………………………..………..8

Дизайн………………………………………………………………………………………..9

Процедурный стиль программирования………………………………..10

Основная точка входа………………………………………………………….…..11

Функции расширения………………………………….……………………….….11

Распаковка аргументов с помощью

оператора распространения…………………….…………………....11

Деструктурирующие объявления……………………………………………12

Абстрактные классы открыты по умолчанию…………………………12

Основной конструктор против вторичных конструкторов…….13

Класс данных……………………………………………………………………………14

Интерактивная оболочка Kotlin……………………………………………….14

Котлин как язык сценариев……………………………………………………..15

Лямбды ………………………………………………………………………………….…15

Пример сложного "привет, мир"……………………………………………..15

Инструменты……………………………………………………………………………..16

Приложения……………………………………………………………………………...16

# Объектно-ориентированное программирование в Kotlin……….17

Принятие…………………………………………………………………………………….20

Будущее Kotlin………………………………………………………………………..…..21

**Введение**

Kotlin полностью совместим с Java, а [JVM](https://wiki2.wiki/wiki/Java_virtual_machine) версия Котлина [стандартная библиотека](https://wiki2.wiki/wiki/Standard_library) зависит от [библиотеки классов Java](https://wiki2.wiki/wiki/Java_Class_Library), но вывод типа…

позволяет [синтаксис](https://wiki2.wiki/wiki/Syntax_(programming_languages))у быть более лаконичным. Kotlin в основном нацелен на JVM, но также компилируется в [JavaScript](https://wiki2.wiki/wiki/JavaScript" \o "JavaScript) или [собственный код](https://wiki2.wiki/wiki/Machine_code) (через [LLVM](https://wiki2.wiki/wiki/LLVM)), например для родных [iOS](https://wiki2.wiki/wiki/IOS" \o "IOS) совместное использование приложений [бизнес логика](https://wiki2.wiki/wiki/Business_logic) с [Android](https://wiki2.wiki/wiki/Android_(operating_system)" \o "Android (операционная система)) Программы. Затраты на развитие языка покрываются [JetBrains](https://wiki2.wiki/wiki/JetBrains" \o "JetBrains), а Kotlin Foundation защищает товарный знак Kotlin.

7 мая 2019 года Google объявил, что язык программирования Kotlin теперь является предпочтительным языком для [Android](https://wiki2.wiki/wiki/Android_(operating_system)" \o "Android (операционная система)) разработчиков приложений. С момента выпуска [Android Studio](https://wiki2.wiki/wiki/Android_Studio" \o "Android Studio) 3.0 в октябре 2017 года, Kotlin был включен в качестве альтернативы стандартному компилятору Java. Компилятор Android

Kotlin по умолчанию нацелен на Java 6, но позволяет программисту выбрать целевую версию для Java 8–13 для оптимизации, или других функций.

В июле 2011 г. [JetBrains](https://wiki2.wiki/wiki/JetBrains" \o "JetBrains) представила Project Kotlin, новый язык для JVM, который разрабатывался в течение года. Руководитель JetBrains Дмитрий Емеров сказал, что у большинства языков нет тех функций, которые они искали, за Kotlin - компилировать так же быстро, как Java. В феврале 2012 года JetBrains открыла исходный код проекта под [Лицензия Apache 2](https://wiki2.wiki/wiki/Apache_license).

Название происходит от [Котлин остров](https://wiki2.wiki/wiki/Kotlin_Island), возле [Санкт-Петербург](https://wiki2.wiki/wiki/Saint_Petersburg). [Андрей Бреслав](https://wiki2.wiki/w/index.php?title=Andrey_Breslav&action=edit&redlink=1) упомянул, что команда решила назвать его в честь острова, так же как Ява была названа в честь индонезийского острова [Ява](https://wiki2.wiki/wiki/Java) (хотя язык программирования Java, возможно, был назван в честь кофе).

JetBrains надеется, что новый язык продвинет [IntelliJ IDEA](https://wiki2.wiki/wiki/IntelliJ_IDEA" \o "IntelliJ IDEA) продажи.

Kotlin v1.0 был выпущен 15 февраля 2016 года. Это считается первым официально стабильным выпуском, и JetBrains взяла на себя обязательство обеспечивать долгосрочную обратную совместимость, начиная с этой версии.

В [Google I / O](https://wiki2.wiki/wiki/Google_I/O" \o "Google I / O) 2017, Google объявил о первоклассной поддержке Kotlin на [Android](https://wiki2.wiki/wiki/Android_(operating_system)" \o "Android (операционная система)).

**Основные возможности и преимущества Kotlin:**

* компилируется в [байткод JVM](https://tproger.ru/translations/programming-concepts-compilation-vs-interpretation/" \t "_blank) или в JavaScript;
* программы могут использовать все существующие Java-фреймворки и библиотеки. Kotlin можно интегрировать с Maven, Gradle и другими системами сборки;
* язык очень прост для изучения;
* исходный код открыт;
* в IntelliJ доступна автоматическая конвертация Java-кода в Kotlin и наоборот;
* язык null-безопасен — надоедливые NullPointerException [остались в Java](https://tproger.ru/translations/10-ways-to-handle-null-in-java/). Вот пример кода

val name: String = null // попытка присвоить null, не скомпилируется.

fun getName() : String = null // попытка вернуть null, не скомпилируется.

* легко читаемый синтаксис не составит проблем при code review.

**Null безопасность**

При попытке присваивания или возвращения null код не скомпилируется. Тем не менее, в языке есть поддержка Nullable-типов. Задать такую переменную или функцию можно, приписав ? к названию типа:

val name: String? = null // присваивается null, код компилируется.

fun getName() : String? = null // возвращается null, код компилируется.

/\* неверно \*/

val name: String? = null

val len = name.length

/\* верно \*/

val name: String? = null

val len = name?.length

Гибкость и простота синтаксиса

Простые функции и структуры можно объявить одной строкой. Геттеры и сеттеры задаются за кулисами для интероперабельности с Java-кодом. Добавление data-аннотации к классу активирует автоматическую генерацию различных шаблонов.

Рассмотрим следующий пример:

/\* Программа на Java \*/

public class Address {

private String street;

private int streetNumber;

private String postCode;

private String city;

private Country country;

public Address(String street, int streetNumber, String postCode, String city, Country country) {

this.street = street;

this.streetNumber = streetNumber;

this.postCode = postCode;

this.city = city;

this.country = country;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Address address = (Address) o;

if (streetNumber != address.streetNumber) return false;

if (!street.equals(address.street)) return false;

if (!postCode.equals(address.postCode)) return false;

if (!city.equals(address.city)) return false;

return country == address.country;

}

@Override

public int hashCode() {

int result = street.hashCode();

result = 31 \* result + streetNumber;

result = 31 \* result + postCode.hashCode();

result = 31 \* result + city.hashCode();

result = 31 \* result + (country != null ? country.hashCode() : 0);

return result;

}

@Override

public String toString() {

return "Address{" +

"street='" + street + '\'' +

", streetNumber=" + streetNumber +

", postCode='" + postCode + '\'' +

", city='" + city + '\'' +

", country=" + country +

'}';

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public int getStreetNumber() {

return streetNumber;

}

public void setStreetNumber(int streetNumber) {

this.streetNumber = streetNumber;

}

public String getPostCode() {

return postCode;

}

public void setPostCode(String postCode) {

this.postCode = postCode;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public Country getCountry() {

return country;

}

public void setCountry(Country country) {

this.country = country;

}

}

/\* Та же программа на Kotlin \*/

data class Address(var street:String,

var streetNumber:Int,

var postCode:String,

var city:String,

var country:Country)

**Отличие от Java**

**1.Null-безопасность**

Как уже говорилось ранее, Kotlin [не допускает](http://kotlinlang.ru/docs/reference/null-safety.html) возникновения NullPointerException, выдавая ошибку компиляции.

**2.Классы данных (Data Classes)**

В Kotlin появились [специальные классы](http://kotlinlang.ru/docs/reference/data-classes.html), предназначенные специально для хранения данных. Они генерируют различные шаблоны: equals(), hashCode(), toString(), геттеры и сеттеры и т.д. Сравним код на Java:

class Book {

private String title;

private Author author;

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public Author getAuthor() {

return author;

}

public void setAuthor(Author author) {

this.author = author;

}

}

И на Kotlin:

/\* Kotlin \*/

data class Book(var title:String,var author:Author)

Легко создавать копии классов данных при помощи метода copy():

val book = Book("Kotlin", "JetBrains")

val copy = book.copy()

**3.Функции-расширения**

Kotlin позволяет расширять функциональность существующих классов, не прибегая к наследованию. Это делается при помощи [функций-расширений](https://kotlinlang.ru/docs/reference/extensions.html). Для объявления такой функции к её имени нужно приписать префикс в виде расширяемого типа. Вот так можно добавить функцию swap в MutableList:

fun MutableList<Int>.swap(index1:Int,index2:Int){

val tmp=this[index1]

this[index1]=this[index2]

this[index2]=tmp

}

[Ключевое слово this](https://kotlinlang.ru/docs/reference/this-expressions.html) внутри функции-расширения относится к объекту-получателю, который передаётся перед точкой. Теперь мы можем применить функцию swap к любому изменяемому списку:

val abc = mutableListOf(1, 2, 3)

abc.swap(0, 2)

**4.Умные приведения типов**

Компилятор Kotlin очень умён, когда речь заходит о [приведениях типов](https://kotlinlang.ru/docs/reference/typecasts.html). В большинстве случаев не требуется явно указывать операторы приведения, поскольку в языке есть оператор is, который делает за вас всю работу:

fun demo(x:Any) {

if(x is String) {

print(x.length) // x автоматически приводится к типу String

}

}

**5.Вывод типов**

В Kotlin необязательно явно указывать тип переменной:

/\* неявное определение \*/

fun main(args: Array<String>) {

val text = 10

println(text)

}

/\* явное определение \*/

fun main(args: Array<String>) {

val text: Int = 10

println(text)

}

**6.Функциональное программирование**

Важно отметить, что Kotlin заточен под [функциональное программирование](https://tproger.ru/translations/functional-programming-concepts/). Он предоставляет большое количество полезных возможностей, например, [функции высшего порядка](https://kotlinlang.ru/docs/reference/lambdas.html), [лямбда-выражения](https://kotlinlang.ru/docs/reference/lambdas.html), [перегрузку операторов](https://kotlinlang.org/docs/reference/operator-overloading.html) и ленивые вычисление логических выражений. Вот пример работы с [коллекциями](https://kotlinlang.ru/docs/reference/collections.html):

fun main(args: Array<String>) {

val numbers = arrayListOf(15, -5, 11, -39)

val nonNegativeNumbers = numbers.filter { it >= 0 }

println(nonNegativeNumbers)}// Вывод: 15, 11

Функции высшего порядка — это функции, которые принимают другие функции в качестве аргументов и возвращают функции. Рассмотрим следующий пример:

fun alphaNum(func: () -> Unit) {}

В нём func — это имя аргумента, а ( ) -> Unit — это тип функции. Мы говорим, что func будет функцией, не принимающей аргументов и ничего не возвращающей.

Лямбда-выражения, или анонимные функции — это функции, которые не объявляются, а передаются в виде выражений. Вот пример:

val sum: (Int, Int) -> Int = { x, y -> x + y }

Мы объявляем переменную sum, которая берёт два числа, складывает их и принимает значение суммы, приведённое к целому. Для вызова достаточно простого sum(2,2).

**7.Сравнение скорости Java и Kotlin**

Первая сборка Kotlin-кода занимает примерно на 15–20% больше времени, чем аналогичный процесс на Java. Однако инкрементная сборка Kotlin даже немного быстрее, чем у Java. Таким образом, языки примерно равны по скорости компиляции.

**Дизайн**

Руководитель разработки Андрей Бреслав сказал, что Котлин задуман Kotlin v1.2 был выпущен 28 ноября 2017 года. В этот выпуск была недавно добавлена ​​функция совместного использования кода между платформами JVM и JavaScript (с версии 1.4 многоплатформенное программирование является [альфа](https://wiki2.wiki/wiki/Software_release_life_cycle#Alpha) особенность модернизированный с "подопытного"). А [полный стек](https://wiki2.wiki/wiki/Full-stack) демо было сделано с новым плагином Kotlin / JS Gradle.

Kotlin v1.3 был выпущен 29 октября 2018 года, предлагая сопрограммы для асинхронного программирования.

7 мая 2019 года Google объявил, что язык программирования Kotlin теперь является предпочтительным языком для разработчиков приложений Android.

Kotlin v1.4 был выпущен в августе 2020 года, например, небольшие изменения в поддержке платформ Apple,т.е. [ЦельC](https://wiki2.wiki/wiki/Objective-C" \o "Цель-C)/[Быстрый](https://wiki2.wiki/wiki/Swift_(programming_language)) [взаимодействие](https://wiki2.wiki/wiki/Interoperability).

исключением [Scala](https://wiki2.wiki/wiki/Scala_(programming_language)" \o "Scala (язык программирования)). Однако он сослался на медленное [сборник](https://wiki2.wiki/wiki/Compiler) время Scala как недостаток. Одна из заявленных целей как промышленный объект. [объектно-ориентированный](https://wiki2.wiki/wiki/Object_oriented_programming) язык, и "лучший язык", чем [Ява](https://wiki2.wiki/wiki/Java_(programming_language)), но при этом полностью совместимы с кодом Java, что позволяет компаниям постепенно переходить с Java на Kotlin.

[Точка с запятой](https://wiki2.wiki/wiki/Semicolon) являются необязательными как [утверждение](https://wiki2.wiki/wiki/Statement_(computer_science)) [терминатор](https://wiki2.wiki/wiki/Statement_terminator#Statements); в большинстве случаев [новая линия](https://wiki2.wiki/wiki/Newline) достаточно для [компилятор](https://wiki2.wiki/wiki/Compiler) сделать вывод, что заявление закончилось.

Котлин [переменная](https://wiki2.wiki/wiki/Variable_(computer_science)) декларации и [списки параметров](https://wiki2.wiki/wiki/Parameter_(computer_programming)) иметь [тип данных](https://wiki2.wiki/wiki/Data_type) идут после имени переменной (и с [двоеточие](https://wiki2.wiki/wiki/Colon_(punctuation)) разделитель), аналогично Базовый, [Паскаль](https://wiki2.wiki/wiki/Pascal_(programming_language)) и [Машинопись](https://wiki2.wiki/wiki/TypeScript).

Переменные в Kotlin доступны только для чтения, они объявляются с vall ключевое слово или изменяемое, объявленное с var ключевое слово.

Члены класса по умолчанию являются общедоступными, а сами классы по умолчанию являются окончательными, что означает создание производного класса “отключено”, если базовый класс не объявлен с открыто ключевое слово.

В добавок к [классы](https://wiki2.wiki/wiki/Class_(computer_programming)) и [функции-члены](https://wiki2.wiki/wiki/Method_(computer_programming)) (эквивалент методов) объектно-ориентированного программирования, Kotlin также поддерживает [процедурное программирование](https://wiki2.wiki/wiki/Procedural_programming) с использованием [функции](https://wiki2.wiki/wiki/Function_(computer_science)). Поддержка функций (и конструкторов) Kotlin [аргументы по умолчанию](https://wiki2.wiki/wiki/Default_argument), [аргумент переменной длины](https://wiki2.wiki/wiki/Variadic_function) списки, [именованные аргументы](https://wiki2.wiki/wiki/Named_parameter) и перегрузка по уникальной сигнатуре. Функции-члены класса являются виртуальными, т.е. отправляются в зависимости от типа среды выполнения объекта, для которого они вызываются.

Kotlin 1.3 добавляет (экспериментальную) поддержку контрактов (вдохновленный [Эйфеля](https://wiki2.wiki/wiki/Eiffel_(programming_language)) [дизайн по контракту](https://wiki2.wiki/wiki/Design_by_contract) парадигма программирования)

**Процедурный стиль программирования**

Kotlin ослабляет ограничение Java на разрешение [статический](https://wiki2.wiki/wiki/Static_(keyword)) методы и переменные должны существовать только в теле класса. Статические объекты и функции могут быть определены на верхнем уровне пакета без необходимости в избыточном уровне класса. Для совместимости с Java Kotlin предоставляет JvmName аннотация, определяющая имя класса, используемое при просмотре пакета из проекта Java. Например, @ файл: JvmName ("JavaClassName").

**Основная точка входа**

Как и в C, C ++, C #, Java и Go, [точка входа](https://wiki2.wiki/wiki/Entry_point) в Котлин [программа](https://wiki2.wiki/wiki/Computer_program) - функция с именем "main", которой можно передать массив, содержащий любые [командная строка](https://wiki2.wiki/wiki/Command-line_interface) аргументы. (Это необязательно, так как Kotlin 1.3). [Perl](https://wiki2.wiki/wiki/Perl" \o "Perl) и [Оболочка Unix](https://wiki2.wiki/wiki/Unix_shell) стиль [строковая интерполяция](https://wiki2.wiki/wiki/String_interpolation) поддерживается. [Вывод типа](https://wiki2.wiki/wiki/Type_inference) также поддерживается.

1 *// Привет, мир! пример* 2 **весело** основной() { 3 **вал** объем = "Мир" 4 println("Привет, $ scope!") 5 } 6 7 **весело** основной(аргументы:

Множество<Строка>) { 8 **за** (аргумент **в** аргументы) { 9 println(аргумент)10 }11 }

**Функции расширения**

Как уже было сказано, подобно C #, Kotlin позволяет пользователю добавлять функции в любой класс без формальностей, связанных с созданием производного класса с новыми функциями. Вместо этого Котлин добавляет концепцию [функция расширения](https://wiki2.wiki/w/index.php?title=Extension_function&action=edit&redlink=1) что позволяет «приклеить» функцию к списку общедоступных функций любого класса без формального размещения внутри класса. Другими словами, функция расширения - это вспомогательная функция, которая имеет доступ ко всему общедоступному интерфейсу класса, который она может использовать для создания нового интерфейса функции к целевому классу, и эта функция будет выглядеть точно так же, как функция класса, появляясь как часть проверки завершения кода функций класса. Например:

1 пакет MyStringExtensions2 3 весело Строка.lastChar(): Char = получать(длина - 1)4 5 >>> println(«Котлин».lastChar())

Помещая предыдущий код на верхнем уровне пакета, класс String расширяется и включает lastChar функция, которая не была включена в исходное определение класса String

1 *// Перегрузка оператора '+' с помощью функции расширения*2 оператор весело Точка.плюс(Другой: Точка): Точка {3 вернуть Точка(Икс + Другой.Икс, у + Другой.у)4 }5 6 >>> вал p1 = Точка(10, 20)7 >>> вал p2 = Точка(30, 40)8 >>> println(p1 + p2)9 Точка(Икс=40, у=60

**Распаковка аргументов с помощью оператора распространения**

Подобно Python, оператор распространения звездочка (\*) распаковывает содержимое массива как аргументы функции, разделенные запятыми:

1 весело основной(аргументы: Множество<Строка>) { 2 вал список = Список("аргументы:", \*аргументы)3 println(список)4 }

**Деструктурирующие объявления**

*Деструктурирующие объявления* разложить объект сразу на несколько переменных, например объект 2D координат может быть *деструктурированный* на два целых числа x и y.

Например, Map.Entry объект поддерживает деструктуризацию для упрощения доступа к его полям ключа и значения:

1 за ((ключ, ценить) в карта) {2 println("$ ключ: $ значение")3 }

Kotlin позволяет объявлять локальные функции внутри других функций или методов.

1 класс Пользователь(вал я бы: Int, вал имя: Строка, вал адрес: Строка) 2 3 весело saveUserToDb(пользователь: Пользователь) { 4 весело подтверждать(пользователь: Пользователь, ценить: Строка, fieldName: Строка) { 5 требовать(ценить.isNotEmpty()) { «Не удается сохранить пользователя $ {user.id}: пустое поле $ fieldName» } 6 } 7 8 подтверждать(пользователь, пользователь.имя, "Имя") 9 подтверждать(пользователь, пользователь.адрес, "Адрес")10 *// Сохраняем пользователя в базу данных* 11 ...12

В Kotlin, чтобы получить новый класс из типа базового класса, базовый класс должен быть явно отмечен как «открытый». Это контрастирует с большинством объектно-ориентированных языков, таких как Java, где классы открыты по умолчанию.

Пример базового класса, который открыт для создания от него нового подкласса.

1 // открыть в классе означает, что этот класс разрешит производные классы 2 открыто класс MegaButton { 3 4 // запрет на открытие функции означает, что 5 // полиморфное поведение отключено, если функция переопределена в производном классе 6 весело отключить() { ... } 7 8 // открытие функции означает, что 9 // полиморфное поведение разрешено, если функция переопределена в производном классе10 открыто весело оживлять() { ... }11 }12 13 класс GigaButton: MegaButton {14 15 // Явное использование ключевого слова override, необходимого для переопределения функции в производном классе16 отменять весело оживлять() { println("Giga Click!") } 17 }

**Абстрактные классы открыты по умолчанию**

[Абстрактные классы](https://wiki2.wiki/wiki/Abstract_class) определить абстрактные или «чисто виртуальные» функции-заполнители, которые будут определены в производном классе. Абстрактные классы открыты по умолчанию.

1 // Здесь ключевое слово open не нужно, оно уже открыто по умолчанию 2 Абстрактные класс Анимированные { 3 4 // Эта виртуальная функция *уже открыта по умолчанию* 5 Абстрактные весело оживлять() 6 7 открыто весело stopAnimating() { } 8 9 весело animateTwice() { }10 }

Классы по умолчанию общедоступны

**Основной конструктор против вторичных конструкторов**

Kotlin предоставляет следующие ключевые слова для ограничения видимости для объявления верхнего уровня, такого как классы, и для членов классов:

общедоступный, внутренний, защищенный и частный.

Применительно к члену класса:

public (по умолчанию): Виден везде internal: Виден в защищенном модуле: Виден в подклассах private: Виден в классе

Применительно к объявлению верхнего уровня

общедоступный (по умолчанию): виден везде internal: виден в модуле private: виден в файле

Пример:

1 *// Класс виден только текущему модулю*2 внутренний открыто класс TalkativeButton : Фокусируемый {3 *// метод виден только текущему классу* 4 частный весело кричать() = println("Привет!")5 6 *// метод виден текущему классу и производным классам*7 защищенный весело шепот() = println("Давай поговорим!")8 }

Kotlin поддерживает спецификацию «первичного конструктора» как часть самого определения класса, состоящего из списка аргументов, следующего за именем класса. Этот список аргументов поддерживает расширенный синтаксис списков аргументов стандартных функций Kotlin, который позволяет декларировать свойства класса в основном конструкторе, включая атрибуты видимости, расширяемости и изменяемости. Кроме того, при определении подкласса свойства в супер-интерфейсах и суперклассах могут быть переопределены в основном конструкторе.

1 *// Пример класса с использованием синтаксиса основного конструктора*2 *// (Для этого класса требуется только один конструктор)*3 открыто класс Продвинутый пользователь : Пользователь (4 защищенный вал прозвище: Строка, 5 окончательный отменять вар isSubscribed: Булево = истинный) 6 {7 ...8 }

Однако в случаях, когда для класса требуется более одного конструктора, можно использовать более общий конструктор, называемый синтаксис вторичного конструктора который очень похож на синтаксис конструктора, используемый в большинстве объектно-ориентированных языков, таких как C ++, C # и Java.

1 *// Пример класса с использованием синтаксиса вторичного конструктора* 2 *// (для этого класса требуется более одного конструктора)* 3 класс MyButton : Посмотреть { 4 5 *// Конструктор №1*  6 конструктор(ctx: Контекст) : супер(ctx) { 7 *// ...*  8 } 9 10 *// Конструктор №2*11 конструктор(ctx: Контекст, attr: AttributeSet) : супер(ctx, attr) { 12 *// ...* 13 }14 }

**Класс данных**

Kotlin предоставляет классы данных для определения классов, хранящих только свойства. В программировании на Java классы, хранящие только свойства, не являются чем-то необычным, но для этой цели используются обычные классы. Kotlin предоставил возможность исключительно определять классы, которые хранят только свойства. Эти классы данных не обязаны объявлять какие-либо методы, только свойства. Класс данных обычно не содержит тела, в отличие от обычного класса. Переопределения по умолчанию для методов объекта, таких как равно () или нанизывать() генерируются во время компиляции. В данные ключевое слово используется перед класс ключевое слово для определения класса данных.

1 весело основной(аргументы: Множество) {2 *// создаем объект класса данных как любой другой объект класса*3 вар book1 = Книга(«Программирование на Котлине», 250)4 println(book1)5 *// вывод: Book (name = Kotlin Programming, цена = 250)*6 }7 8 *// класс данных с параметрами и их необязательными значениями по умолчанию*9 данные класс Книга(вал имя: Строка = "", вал цена: Int = 0)

**Интерактивная оболочка Kotlin**

$ kotlinc-jvmтип :помощь **за** Помогите; :покидать **за** выйти >>> 2 + 24>>> println("Привет, мир!")Привет, мир! >>>

**Котлин как язык сценариев**

Kotlin также можно использовать как язык сценариев. Скрипт - это исходный файл Kotlin (.kts) с исполняемым кодом верхнего уровня.

1 *// list\_folders.kts*2 импорт java.io.File3 вал папки = файл(аргументы[0]).listFiles { файл -> файл.isDirectory() }4 папки?.для каждого { папка -> println(папка) }

Скрипты можно запустить, передав -скрипт параметр и соответствующий файл сценария компилятору.

$ kotlinc -script list\_folders.kts "path\_to\_folder\_to\_inspect"

**Лямбды**

Kotlin обеспечивает поддержку [функции высшего порядка](https://wiki2.wiki/wiki/Higher_order_functions) и [анонимные функции](https://wiki2.wiki/wiki/Anonymous_functions) или лямбды.

1 *// следующая функция принимает лямбду f и выполняет f, передавая ей строку «лямбда»*2 *// обратите внимание, что (s: String) -> Unit указывает лямбда с параметром String и типом возвращаемого значения Unit*3 весело executeLambda(ж: (s: Строка) -> Единица измерения) {4 ж("лямбда")5 }

Лямбды объявляются фигурными скобками, { } . Если лямбда принимает параметры, они объявляются в фигурных скобках и сопровождаются -> оператор.

1 *// следующий оператор определяет лямбду, которая принимает единственный параметр и передает его функции println*2 вал л = { c : Любой? -> println(c) }3 *// лямбды без параметров можно просто определить с помощью {}*4 вал l2 = { Распечатать("без параметров") }

**Пример сложного "привет, мир"**

1 весело основной(аргументы: Множество<Строка>) { 2 приветствовать { 3 к.место 4 }.Распечатать() 5 } 6 7 *// Встроенные функции высшего порядка* 8 встроенное веселье приветствовать(s: () -> Строка) : Строка = приветствие и другой s() 9 10 *// Инфиксные функции, расширения, вывод типов, типы, допускающие значение NULL,* 11 *// лямбда-выражения, помеченные этим, оператор Элвиса (? :)*12 инфикс весело Строка.и другой(Другой : Любой?) = buildString() 13 { 14 добавить(это@и другой); добавить(" "); добавить(Другой ?: "") 15 }16 17 *// Неизменяемые типы, делегированные свойства, отложенная инициализация, строковые шаблоны*18 вал приветствие к ленивый { вал doubleEl: Строка = "ll"; "он $ {doubleEl} o" }19 20 *// Запечатанные классы, сопутствующие*

*объекты*21 запечатанный класс к { товарищ объект { вал место = "Мир"} }22 23 *// Расширения, блок*24 весело Строка.Распечатать() = println(это)

**Инструменты**

* [IntelliJ IDEA](https://wiki2.wiki/wiki/IntelliJ_IDEA) имеет подключаемый модуль для Kotlin. IntelliJ IDEA 15 была первой версией, которая объединила плагин Kotlin в IntelliJ Installer и предоставила поддержку Kotlin из коробки.
* JetBrains также предоставляет плагин для [Затмение](https://wiki2.wiki/wiki/Eclipse_(software)).
* Поддерживается интеграция с распространенными инструментами сборки Java, включая [Apache Maven](https://wiki2.wiki/wiki/Apache_Maven" \o "Apache Maven), [Apache Ant](https://wiki2.wiki/wiki/Apache_Ant" \o "Apache Ant) и [Gradle](https://wiki2.wiki/wiki/Gradle" \o "Gradle).
* [Android Studio](https://wiki2.wiki/wiki/Android_Studio) (на основе IntelliJ IDEA) имеет официальную поддержку Kotlin, начиная с Android Studio 3.
* [Emacs](https://wiki2.wiki/wiki/Emacs) имеет режим Kotlin в репозитории пакетов Melpa.
* [Vim](https://wiki2.wiki/wiki/Vim_(text_editor)) есть плагин, поддерживаемый на Github.
* [Json2Kotlin](https://json2kotlin.com/) генерирует [POJO](https://wiki2.wiki/wiki/Plain_old_Java_object) стиль собственного кода Kotlin для сопоставления ответов веб-службы.

**Приложения**

Kotlin широко используется на стороне сервера, и [Android](https://wiki2.wiki/wiki/Android_(operating_system)" \o "Android (операционная система)) разработка. На Android платформа на какое-то время застряла на Java 7 (с некоторыми современными языковыми функциями, доступными благодаря использованию Retrolambda или набор инструментов Jack) и Kotlin предлагает множество улучшений для программистов, таких как безопасность нулевого указателя, функции расширения и инфиксная нотация. Сопровождается полной совместимостью с Java и хорошей поддержкой IDE (*Android Studio*) он призван улучшить читаемость кода, упростить расширение классов Android SDK и ускорить разработку.

Kotlin был объявлен официальным языком разработки Android на [Google I / O](https://wiki2.wiki/wiki/Google_I/O" \o "Google I / O) 2017. Он стал третьим языком, полностью поддерживаемым для Android, помимо Java и C ++.

В [Spring Framework](https://wiki2.wiki/wiki/Spring_Framework" \o "Spring Framework) официально добавлена ​​поддержка Kotlin с версией 5 от 4 января 2017 г. **перед** анонс Android, сделанный Google на [Google I / O](https://wiki2.wiki/wiki/Google_I/O" \o "Google I / O) 2017. Часть официальной поддержки Kotlin в Spring охватывает документацию, поддержка библиотеки, и видео-презентации. Многие другие серверные библиотеки / фреймворки, такие как Ktor, и Vert.x предоставить официальную поддержку Kotlin. Google Cloud Platform официально поддерживает Kotlin, однако другие облачные платформы, такие как AWS (Amazon Web Services) и Azure, официально не предоставляют поддержку Kotlin, хотя известно, что Kotlin работает на этих платформах.

Котлин в основном используется в новых проектах[[51]](https://wiki2.wiki/wiki/Kotlin_(programming_language)#cite_note-51) (96%), и в основном используется в рабочих проектах (75%), за которыми следуют личные проекты (68%). На Kotlin разрабатываются следующие типы программного обеспечения:

* Мобильный (56%)
* Веб-сервер (47%)
* Библиотека / фреймворк (28%)
* Инструменты (20%)

Linux - главная цель используется в проектах Kotlin Native (74%), затем идут Android (48%) и MacOS (47%).

# **Объектно-ориентированное программирование в Kotlin**

Объектно-ориентированное программирование в Kotlin имеет ряд особенностей, связанных с появлением первичного конструктора, у полей – геттеров и сеттеров по-умолчанию. Есть возможность определять методы за пределами класса.

В Kotlin по-умолчанию классы и их функции имеют модификатор final. Это значит, что от таких классов нельзя наследовать, а функции нельзя переопределять в дочерних классах. Чтобы разрешить наследование и переопределение необходимо это указать явно – писать модификатор open у родительского класса и его методов. В дочернем классе переопределяемые методы должны иметь модификатор override.

В Kotlin есть абстрактные классы и интерфейсы, они являются открытыми по-умолчанию.

Дочерние классы могут наследовать от одного родительского и от множества интерфейсов.

open **class** Parent

**interface** First

**interface** Second

**class** Child: Parent(), First, Second

Поскольку в Kotlin параметры функций могут иметь значения по-умолчанию, то зачастую в классах достаточно одного конструктора, который обычно делают первичным и нередко объединяют с объявлением свойств.

fun main() {

val obj1 = MyClass()

val obj2 = MyClass(10, 5)

}

**class** MyClass(var a: Int = 0, var b: Int = 0)

Приведенное выше определение MyClass это сокращенный вариант от примерно такого:

**class** MyClass(aa: Int = 0, bb: Int = 0) {

var a = aa

var b = bb

}

Если преобразовать первичный конструктор ко вторичному, определение класса получится более характерным для других языков программирования, но зато и более длинным:

**class** MyClass {

var a: Int

var b: Int

constructor(aa: Int = 0, bb: Int = 0) {

a = aa

b = bb

}

}

В случае необходимости у класса может быть несколько конструкторов, среди которых только один первичный. При наличии первичного конструктора вторичные должны делегировать к нему непосредственно или опосредованно через другой вторичный. Делается это с помощью ключевого слова this.

fun main() {

val obj1 = MyClass()

println(obj1.d) *// true*

val obj2 = MyClass(10, 5)

println(obj2.d) *// false*

}

**class** MyClass(aa: Int, bb: Int) {

var a = aa

var b = bb

var d = **false**

constructor(): **this**(0, 0) {

d = **true**

}

}

Поскольку первичный конструктор не имеет тела, в случае необходимости какого-либо инициирующего кода, он заключается в блок init{}.

fun main() {

val obj1 = MyClass()

println(obj1.d) *// true*

val obj2 = MyClass(0, 0)

println(obj2.d) *// true*

}

**class** MyClass(aa: Int = 0, bb: Int = 0) {

var a = aa

var b = bb

var d: Boolean

init {

**if** (aa == 0 && bb == 0)

d = **true**

**else** d = **false**

}

}

Каждое поле класса имеет свои геттер и сеттер, вместе они формируют свойство. По-умолчанию геттер возвращает значение поля и сеттер присваивает полю значение без всякой обработки. Явно это выражается так:

**class** MyClass(aa: Int, bb: Int) {

var a = aa

get() {**return** field}

set(value) {field = value}

var b = bb

get() {**return** field}

set(value) {field = value}

}

Слово field обозначает текущее поле. Используется, чтобы избежать рекурсивных вызовов. Явное указание геттеров и сеттеров имеет смысл, если их логика сложнее, или надо запретить возможность изменения значения поля за пределами класса, оставив возможность получать значение.

fun main() {

val obj1 = MyClass(12, 3)

obj1.a = -5

println(obj1.a) *// 12*

*// obj1.b = 9 Error*

println(obj1.b) *// Ok 3*

}

**class** MyClass(aa: Int, bb: Int) {

var a = aa

set(value) {

**if** ((value) > 0)

field = value

}

var b = bb

**private** set

}

В Kotlin методы, определенные внутри класса, называются функциями-членами. Функции-расширения – это методы, определенные за пределами класса.

fun main() {

val obj1 = MyClass(12, 3)

obj1.member()

obj1.extension()

}

**class** MyClass(val a: Int = 0, val b: Int = 0) {

fun member() {

println("member-function")

println(a + b)

}

}

fun MyClass.extension() {

println("extension-function")

println(a - b)

}

В Kotlin многие методы встроенных классов реализованы как функции-расширения. Это связано с тем, что Kotlin во многом использует java-библиотеки, и разработчикам было проще дополнить их код за пределами классов.

**Принятие**

В 2018 году Kotlin был самым быстрорастущим языком на GitHub: количество разработчиков было в 2,6 раза больше, чем в 2017 году. Это четвертый по популярности язык программирования согласно опросу разработчиков Stack Overflow 2020.

Котлин также был награжден премией O'Reilly Open Source Software Conference Breakout Award за 2019 год.

Многие компании / организации использовали Kotlin для внутренней разработки:

* Google
* Норвежская налоговая администрация
* Gradle
* Amazon
* Квадрат
* JetBrains
* Поток
* Аллегро
* OLX
* Shazam
* Основной
* Ракетное Путешествие
* Meshcloud
* Zalando

Некоторые компании / организации использовали Kotlin для веб-разработки:

* JetBrains
* Data2viz
* Fritz2
* Банк Барклая

Ряд компаний публично заявили об использовании Kotlin:

* DripStat
* [Базовый лагерь](https://wiki2.wiki/wiki/Basecamp_(software))
* [Pinterest](https://wiki2.wiki/wiki/Pinterest)
* Coursera
* Netflix
* Убер
* Квадрат
* Trello
* Duolingo
* Corda, распределенный реестр, разработанный консорциумом известных банков (таких как [Голдман Сакс](https://wiki2.wiki/wiki/Goldman_Sachs" \o "Голдман Сакс), [Уэлс Фарго](https://wiki2.wiki/wiki/Wells_Fargo" \o "Уэлс Фарго), [Дж. П. Морган](https://wiki2.wiki/wiki/JPMorgan_Chase), [немецкий банк](https://wiki2.wiki/wiki/Deutsche_Bank), [UBS](https://wiki2.wiki/wiki/UBS), [HSBC](https://wiki2.wiki/wiki/HSBC), [BNP Paribas](https://wiki2.wiki/wiki/BNP_Paribas), [Société Générale](https://wiki2.wiki/wiki/Soci%C3%A9t%C3%A9_G%C3%A9n%C3%A9rale" \o "Société Générale) ), имеет более 90% кода Kotlin в своей кодовой базе.

**Будущее Kotlin**

Kotlin — это следующий этап развития Java, с которой он полностью совместим. Это делает его отличным инструментом для мобильных и энтерпрайз-приложений. Теперь Kotlin теперь является официальным языком Android.