Объектно-ориентированное программирование в Kotlin

Лекция

Александр Глускер

Курс «Мобильная разработка»

3 апреля 2026 г.

Навигация по лекции

Классы и объекты

```
1 // Определение класса
2 class Person(val name: String, var age: Int)
3
4 // Создание объекта
5 val person = Person("Alice", 30)
6 println(person.name) // Alice
7 person.age = 31
8
```

- Класс — шаблон для создания объектов.
- `val` — свойство только для чтения, `var` — изменяемое.
- Первичный конструктор объявляется в заголовке класса.

Первичный и вторичный конструкторы

```
1  class Person(val name: String) {
2    var age: Int = 0
3
4    // Вторичный конструктор
5    constructor(name: String, age: Int) : this(name) {
6      this.age = age
7    }
8  }
9
10 val p1 = Person("Bob")           // первичный
11 val p2 = Person("Bob", 25)       // вторичный
12
```

- Первичный конструктор — часть заголовка класса.
- Вторичные конструкторы должны делегировать вызов первичному (`this(...)`).

Навигация по лекции

Свойства с кастомными get/set

```
1  class Temperature {
2    var celsius: Double = 0.0
3      set(value) {
4        field = if (value < -273.15) -273.15 else value
5      }
6      get() {
7        println("Доступ к температуре")
8        return field
9      }
10 }
11
```

- `field` — резервное поле, автоматически создаётся при наличии кастомного геттера/сеттера.
- Геттер/сеттер вызываются при каждом доступе к свойству.

Навигация по лекции

Nested и Inner классы

```
1  class Outer {
2    private val bar = 1
3
4    class Nested {  // nested статический( по умолчанию)
5      fun foo() = 2
6    }
7
8    inner class Inner {  // inner – имеет доступ к внешнему классу
9      fun foo() = bar
10   }
11 }
12
13 val nested = Outer.Nested().foo() // OK
14 val inner = Outer().Inner().foo() // OK
15
```

Навигация по лекции

Абстрактные классы и наследование

```
1  abstract class Animal {
2    abstract fun makeSound()
3    open fun sleep() = println("Zzz")
4  }
5
6  class Dog : Animal() {
7    override fun makeSound() = println("Woof!")
8    override fun sleep() = println("Dog is sleeping")
9  }
10
```

- По умолчанию классы и методы `final` — нельзя наследовать/переопределять.
- Чтобы разрешить — использовать `open` или `abstract`.
- `abstract` методы не имеют реализации.

Навигация по лекции

Companion objects

```
1  class MathUtils {
2    companion object {
3      const val PI = 3.14159
4      fun double(x: Int) = x * 2
5    }
6  }
7
8  // Использование как статические члены
9  println(MathUtils.PI)
10 println(MathUtils.double(5))
11
```

- `companion object` — замена статическим методам/полям.
- `const val` — compile-time константа (только примитивы и строки).

Навигация по лекции

Интерфейсы

```
1  interface Flyable {
2    fun fly()
3    fun land() = println("Landed") // default implementation
4  }
5
6  class Bird : Flyable {
7    override fun fly() = println("Flying!")
8  }
9
10 val bird = Bird()
11 bird.fly()
12 bird.land()
13
```

- Интерфейсы могут содержать реализацию по умолчанию.
- Нет множественного наследования классов, но можно реализовать несколько интерфейсов.

Навигация по лекции

Области видимости

- `private` — только в этом классе.
- `protected` — в этом классе и подклассах.
- `internal` — в пределах модуля (аналог package-private в Java).
- `public` — по умолчанию, видно везде.

```
1 class Example {
2   private fun secret() = "hidden"
3   internal fun moduleOnly() = "visible in module"
4 }
5
```

Навигация по лекции

Lateinit

```
1  class MyActivity {
2    lateinit var adapter: RecyclerView.Adapter<*>
3
4    fun onCreate() {
5      adapter = MyAdapter()
6    }
7
8    fun useAdapter() {
9      if (::adapter.isInitialized) {
10       // безопасное использование
11     }
12   }
13 }
14
```

- `lateinit` — откладывает инициализацию var-свойства.
- Только для var, не-nullable типов, не примитивов.
- Проверка инициализации: `::prop.isInitialized`.

Делегированные свойства

```
1  import kotlin.properties.Delegates
2
3  class Example {
4    val lazyValue: String by lazy {
5      println("Вычисляется один раз!")
6      "Hello"
7    }
8
9    var observableValue: Int by Delegates.observable(0) { _, old, new ->
10     println("Изменено с $old на $new")
11   }
12 }
13
14 val e = Example()
15 println(e.lazyValue) // Вычисляется" один раз!" → "Hello"
16 e.observableValue = 42 // Изменено" с 0 на 42"
17
```

Навигация по лекции

SAM-интерфейсы

```
1 fun interface OnClickListener {
2   fun onClick()
3 }
4
5 // Использование лямбды вместо анонимного класса
6 button.setOnClickListener { println("Clicked!") }
7
```

- SAM (Single Abstract Method) — интерфейс с одним абстрактным методом.
- Позволяет передавать лямбду вместо реализации.
- Объявляется с `fun interface`.

Extension-функции и свойства

```kotlin
fun String.lastChar(): Char = this[this.length - 1]

val str = "Kotlin"
println(str.lastChar()) // 'n'

// Extensionсвойство-
val String.isLong: Boolean
  get() = length > 10

```

- Расширяют функциональность существующих типов без наследования.
- Компилируются в статические утилитарные методы.

Навигация по лекции

Data классы

```
1 data class User(val name: String, val id: Int)
2
3 val user1 = User("Alice", 1)
4 val user2 = user1.copy(name = "Bob")
5 println(user1 == user2) // false
6 println(user1) // User(name=Alice, id=1)
7
```

- Автоматически генерируют `equals()`, `hashCode()`, `toString()`, `copy()`.
- Все свойства в первичном конструкторе участвуют в сравнении.

Sealed классы

```
1 sealed class Result
2 data class Success(val data: String) : Result()
3 data class Error(val message: String) : Result()
4
5 fun handle(result: Result) = when(result) {
6   is Success -> println("Data: ${result.data}")
7   is Error -> println("Error: ${result.message}")
8 }
9
```

- Ограниченное наследование: подклассы должны быть в том же файле.
- Идеальны для моделирования закрытых множеств состояний.
- Безопасны в when — компилятор проверяет полноту.

Enum классы

```
1  enum class Color(val rgb: Int) {
2    RED(0xFF0000),
3    GREEN(0x00FF00),
4    BLUE(0x0000FF);
5
6    fun printInfo() = println("$name = $rgb")
7  }
8
9  Color.RED.printInfo() // RED = 16711680
10 for (c in Color.values()) println(c)
11
```

- Каждая константа — объект enum-класса.
- Можно добавлять свойства и методы.

Спасибо за внимание!

Вопросы?