Kotlin: ООП, классы

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели модуля

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Структура модуля

- ООП, классы, объекты
- Наследование, интерфейсы
- Методы класса Any, классы данных
- Делегированные свойства и lateinit
- Расширения для классов
- Объявления объектов
- Перечисления, sealed-классы
- Перегрузка операторов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели модуля

По окончании модуля вы будете знакомы с ООП парадигмой в котлине и научитесь писать программы, которые работают с классами, интерфейсами, перечислениями, объектами. Научитесь строить иерархии наследования и расширять возможности классов.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

ООП, классы, объекты

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке мы рассмотрим, что такое ООП, классы, объекты, методы, поля. Научимся на практике работать с этими понятиями, рассмотрим области видимости, геттеры и сеттеры.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Объектно-ориентированное программирование

Kotlin - ООП язык

ООП использует:

- классы
- объекты
- иерархии

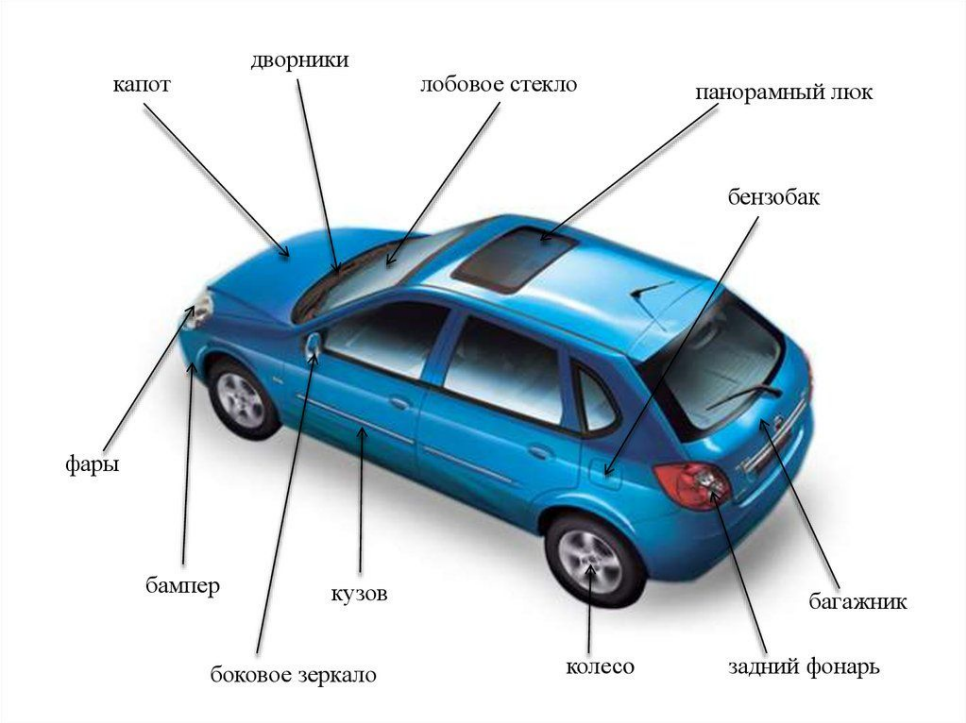
Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Класс

Название: Автомобиль

Характеристики:

- количество колес
- максимальная скорость
- мощность двигателя
- количество дверей
- размеры кузова



Действия:

- разгон (двигатель крутит колеса)
- торможение (сжатие тормозных дисков)
- поворот (руль поворачивает колеса)
- открытие дверей (механизм ручки открывает замок)

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Объект

Класс: Автомобиль

Название: Tesla Cybertruck

Характеристики:

- количество колес - 4 шт.
- максимальная скорость - 209 км/ч
- мощность двигателя - 775 л.с.
- количество дверей - 2 шт.
- размеры кузова(Д-Ш-В) - 5885-2027-1905мм

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Поля и методы

Поля:

- количество колес
- текущая скорость скорость
- мощность двигателя
- количество дверей
- размеры кузова

Методы:

- разогнаться
  - повысить подачу топлива
  - увеличить количество оборотов двигателя и колес
- затормозить
- повернуть
- открыть двери

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Принципы ООП

- Инкапсуляция
- Наследование
- Полиморфизм
- Абстракция

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Модификаторы доступа

Могут применяться к полям, методам, классам.

- public(по умолчанию) - доступность отовсюду
- protected - доступность внутри класса и внутри потомков этого класса
- internal - доступность внутри модуля где объявлен класс
- private - доступность только внутри класса

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели, на чем основывается ООП, на практике поработали с:

- классами
- конструкциями импортов
- объектами
- методами и полями
- геттерами и сеттерами
- модификаторами доступа

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Наследование, интерфейсы

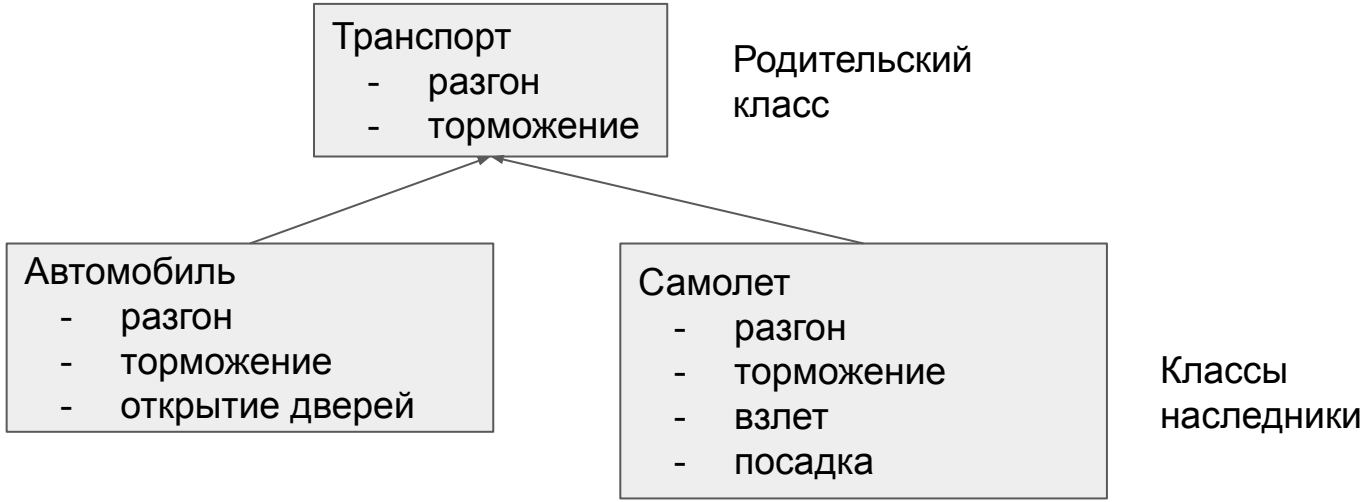
Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке мы рассмотрим наследование, интерфейсы, абстрактные и анонимные классы.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Наследование

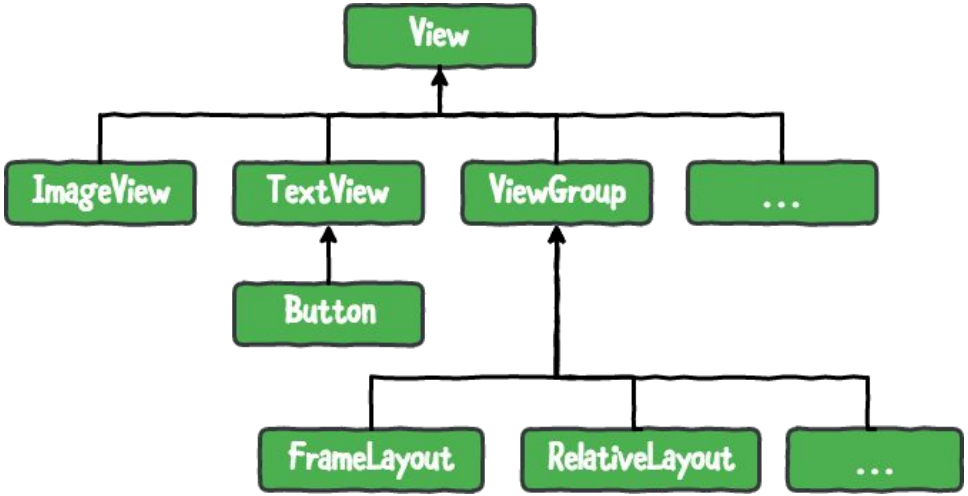


Автомобиль **является** транспортом

Самолет **является** транспортом

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Иерархия классов



Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Наследование в Kotlin

- Отсутствует множественное наследование
- На вершине иерархии наследования находится класс Any
- Необходимы модификаторы класса - open или abstract
- Дочерний класс наследует не-private члены родительского класса

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Абстрактный класс

- Предназначен для наследования
- Невозможно создать экземпляр абстрактного класса
- Может иметь абстрактные методы и свойства
- Может содержать обычные методы и свойства и хранить состояние

```
abstract class AbstractShape(
    val x: Int,
    val y: Int
) {
    abstract val name: String
    abstract fun calculateArea(): Double
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Интерфейс

- Может иметь поля и методы
- Все поля и методы будут доступны пользователю интерфейса
- Нет внутренней реализации методов и полей
- Нельзя создать объект интерфейса
- Нет внутреннего состояния
- Реализуются классами, возможна множественная реализация

```
interface Vehicle {
    val currentSpeed: Int

    fun accelerate(speed: Int)
    fun decelerate(speed: Int)
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Анонимные классы

- Не имеют имени
- Используются для создания объекта один раз
- С помощью них можно создать объект интерфейса, абстрактного класса либо наследовать существующий класс

```
val shapeInstance = object : Shape {
    override val name: String = "Anonymous"

    override fun calculateArea(): Double = 0.0
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели наследование, научились работать с классами-наследниками, интерфейсами, абстрактными и анонимными классами.
Рассмотрели интерфейс Comparable, метод toString() и scoped-функцию with

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Методы класса Any, классы данных

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

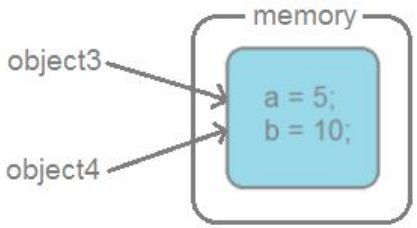
Цели урока

На этом уроке мы поговорим о сравнении объектов, рассмотрим методы класса Any - equals, hashcode, toString.
Познакомимся с классами данных и деструктивными определениями.

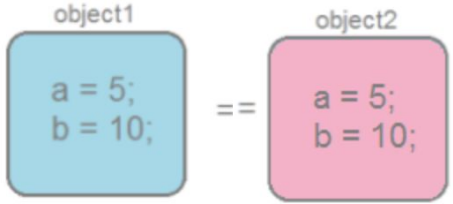
Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Сравнение

Равенство по ссылке (object3 === object4)



Равенство по значению (object3 == object4)



Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

hashCode()

Хеш-код - это целочисленное значение, которое возвращается хеш-функцией для объекта.
Функция hashCode является хеш-функцией. Она должна вычислить число быстро и с малой вероятностью коллизий.
Коллизия - это такой случай, когда для двух объектов которые являются разными хешфункция возвращает одинаковый результат.
Реализации по умолчанию hashCode зависят от версии JVM.
Самым ярким примером использования функции hashCode является структура данных хештаблица.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Контракт между equals и hashCode

- Объекты одинаковы - хешкоды одинаковы
- Хешкоды одинаковы - объекты **необязательно** одинаковы (коллизия)
- Хешкоды разные - объекты разные

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

toString()

Возвращает текстовое представление объекта.

По умолчанию возвращает название класса и хешкод в шестнадцатеричном виде:

com.skillbox.skillboxkotlin.Car@b026219

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Data class

Генерирует методы неявно:

- equals()/hashCode()
- toString()
- componentN()
- copy()

```
data class User(
    val firstName: String,
    val secondName: String
)
```

Ограничения:

- Должен быть как минимум 1 аргумент в основном конструкторе
- Все аргументы основного конструктора должны быть помечены как val/var
- Не может быть open, abstract, inner или sealed

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели ссылочное сравнение объектов и сравнение по значению. Изучили методы класса Any: equals, hashCode, toString. Рассмотрели data-классы и деструктивные определения.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Делегированные свойства и lateinit

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке мы с вами рассмотрим модификатор свойств lateinit a также делегированные свойства

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

lateinit

Позволяет произвести отложенную инициализацию свойства класса.

```
lateinit var router: Router
```

Для того, чтобы использовать модификатор lateinit необходимо:

- свойство должно быть изменяемым
- свойство не должно объявляться в основном конструкторе
- свойство не должно иметь геттеров и сеттеров
- тип свойства должен быть ненуллабельным и не примитивным

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Delegated properties

Позволяют переиспользовать код для инициализации свойств и добавить логику при обращении к полю и при установке значения

Вы можете создавать собственные и использовать существующие:

- lazy - позволяет инициализировать поле при первом обращении
- Delegates.notNull - позволяет не инициализировать поле значением при создании, но при этом избежать нуллабельности
- Delegates.observable - позволяет вызывать необходимый блок кода при изменении значения свойства

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели модификатор свойства lateinit, делегированные свойства, научились создавать свои делегированные свойства и использовать встроенное - lazy.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Расширения для классов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке мы узнаем, что такое расширения для классов, и как с ними работать.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Extensions

Предназначены для добавления нового функционала в классы без наследования от них.

Виды:

- extension function

```
fun CharSequence.isNotEmpty(): Boolean = length > 0
```

- extension property

```
val CharSequence.lastIndex: Int
    get() = this.length - 1
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели расширения, научились их создавать и использовать

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Объявления объектов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке рассмотрим такие объявления объектов, как объект-синглтон, объект-компаньон.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Singleton object

Синглтон - объект класса, который существует в единственном экземпляре во время всей работы приложения.

- объявляется с помощью ключевого слова **object**
- объявление объекта не является выражением
- у объекта есть имя
- может быть 3 области видимости - public, private, internal
- объект может наследоваться от класса или реализовывать интерфейс
- может содержать свойства и методы

```
object Cars {
    val toyota = Car(wheelCount = 4, maxSpeed = 200)
    val nissan = Car(wheelCount = 4, maxSpeed = 250)
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Companion object

- находится внутри класса
- объявляется с помощью ключевых слов **companion object**
- может быть и не быть имени
- может быть 4 области видимости - public, private, internal, protected
- компаньон может наследоваться или реализовывать интерфейс
- может содержать свойства и методы

```
class Car {
    companion object {
        val default = Car(wheelCount = 4, maxSpeed = 200)
        fun createWithDefaultWheels(maxSpeed: Int): Car {
            return Car(wheelCount = 4, maxSpeed = 200)
        }
    }
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке рассмотрели объявления объектов - объект-синглтон, объект-компаньон. Научились работать с ними на практике.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Перечисления, sealed-классы

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке рассмотрим перечисления и sealed классы, посмотрим, чем они отличаются, как с ними работать на практике.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Enums

Классы, которые могут содержать только объявленные объекты.

- Нельзя создать объект этого класса во время выполнения
- Объявляются с помощью ключевого слова enum
- Объекты описываются внутри тела класса
- Перечисления являются обычными классами

```
enum class Color(
    val hex: String
) {
    WHITE("#fff"),
    BLACK("#000"),
    RED("#f00"),
    BLUE("#00f")
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Sealed-class

Класс, наследники которого известны на момент компиляции.

- Похож на перечисления
- Наследниками могут выступать объекты-синглтоны и классы
- Нельзя создавать объекты sealed-класса

```
sealed class SealedColor(val hex: String) {
    object White: SealedColor("#fff")
    object Black: SealedColor("#000")
    object Red: SealedColor("#f00")
    object Blue: SealedColor("#00f")

    class CustomColor(hex: String): SealedColor(hex)
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы рассмотрели перечисления и sealed классы, узнали их возможности и отличия на практике.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Перегрузка операторов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Цели урока

На этом уроке мы с вами поговорим о перегрузке операторов для классов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Перегрузка операторов

```
data class Point(val x: Int, val y: Int)

operator fun Point.unaryMinus() = Point(-x, -y)

val point = Point(10, 20)

fun main() {
    println(-point)
}
```

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Выводы

На этом уроке мы узнали, как можно перегружать работу операторов для собственных классов

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Домашнее задание

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox

Домашнее задание

В домашнем задании этого модуля вы должны будете написать программу, которая симулирует битву между двумя сторонами и определяет сторону-победителя.
Для этого вам необходимо будет создать иерархию классов, поработать с интерфейсами, абстрактными классами, перечислениями.

Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox Skillbox