

Лабораторные работы
по программированию для ОС Android на Kotlin с использованием Compose

Оглавление

Лабораторная работа №0.1. Введение в Kotlin.....	2
Лабораторная работа №0.2. Массивы в Kotlin.....	6
Лабораторная работа №0.3. ООП в Kotlin	8
Лабораторная работа №1. Введение в программирование для ОС Android	12
Лабораторная работа №2. Использование списков для отображения набора визуальных объектов	20
Лабораторная работа №3. Использование Image и дополнительной информации на экране	25
Лабораторная работа №4. Меню, несколько Activity (окон) в одном приложении	30
Лабораторная работа №5. Сохранение состояний. Контекстное меню и работа с внешними ресурсами.....	34
Лабораторная работа №6. Работа с БД.....	39
Лабораторная работа №7. Боковая панель навигации. Работа с графикой в Android. Локализация текстовых ресурсов.....	43

Лабораторная работа №0.1. Введение в Kotlin

Kotlin – это статически типизированный объектно-ориентированный язык, совместимый с Java и предназначенный для промышленной разработки приложений. Поддерживает функциональный стиль программирования. Работает поверх JVM. Есть онлайн площадка для написания кода и запуска программ на Kotlin по адресу <https://try.kotlinlang.org/>. Онлайн документация по языку находится по адресу <https://kotlinlang.org/docs/reference/>.

Удобнее всего работать с Kotlin в среде программирования IntelliJ IDEA. Для этого после ее запуска выбираем Create New Project (если среда запускается в первый раз или предыдущий проект был закрыт командой Close Project из меню File) или в меню File выбираем New – Project... (рис. 1).

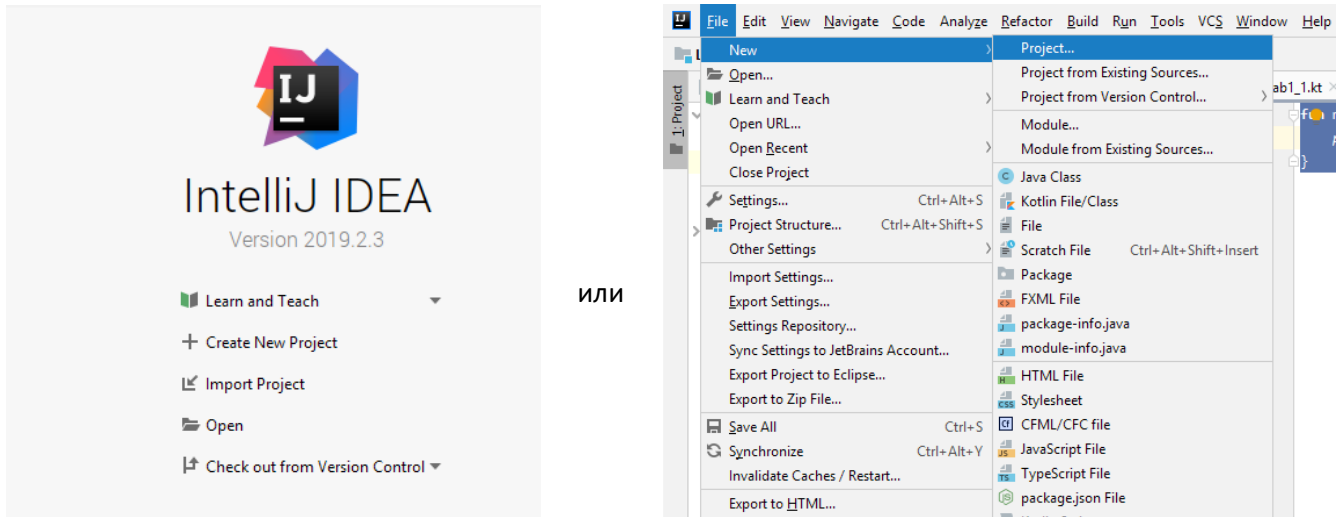
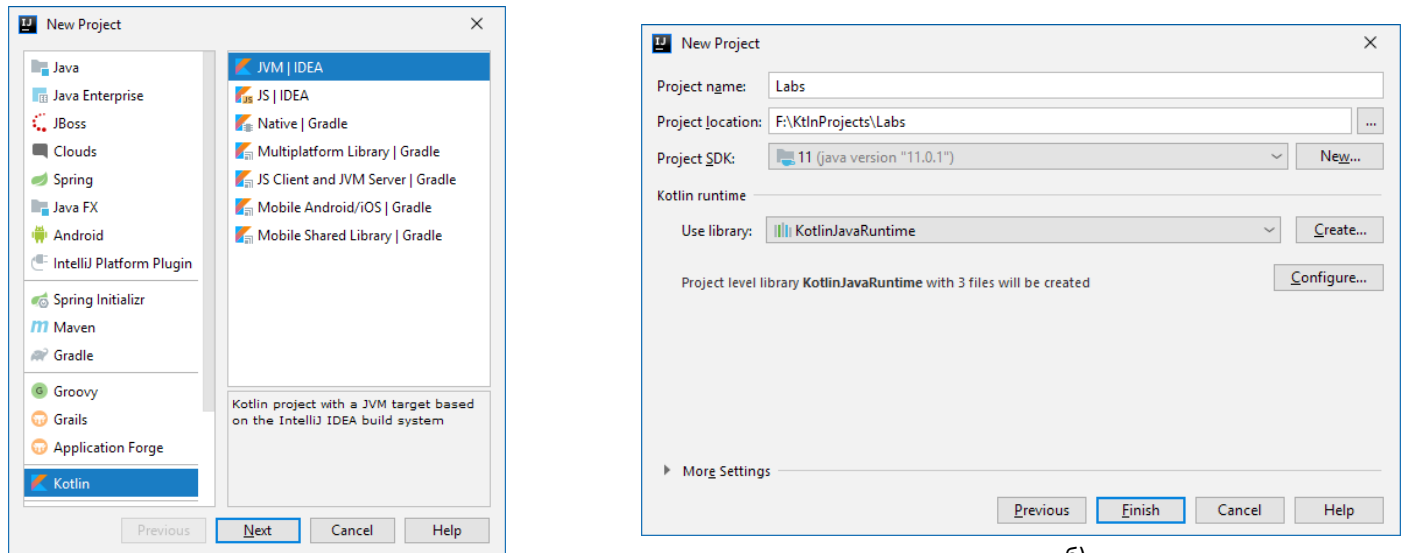


Рис. 1. Варианты создания Kotlin-проекта в IntelliJ IDEA.

Появится окно с вариантами проектов. Выбираем раздел Kotlin, в нем JVM | IDEA (рис. 2а). Нажимаем Next. Во втором окне вводим название проекта и жмем Finish (рис. 2б).



а)

б)

Рис. 2. Выбор проекта и задание его параметров.

В появившемся проекте в левой части будет структура папок и файлов проекта. Вызываем контекстное меню папки src и выбираем New → Kotlin File/Class. В появившемся окне в поле Name вводим название файла (например, lab1_1) и жмем Enter (рис. 3).

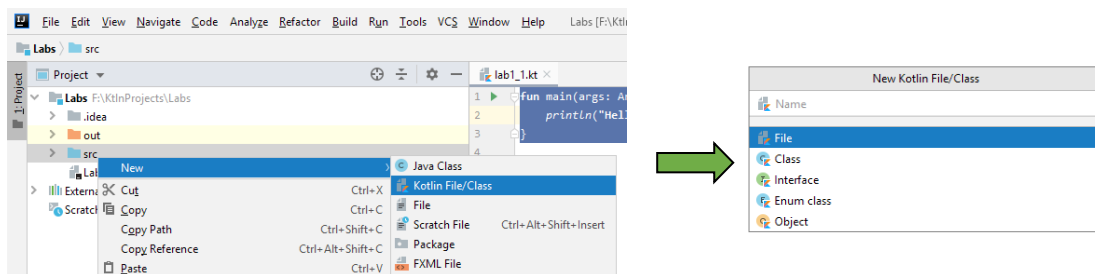


Рис. 3. Добавление файла в проект.

Созданный файл сразу появится в центральном окне.

Первая программа на Kotlin:

```
fun main() {
    println("Hello, world!")
}
```

При запуске в консоль выведется hello, world!

Разберем этот код:

- **fun** – зарезервированное слово для обозначения функции;
- **fun main()** – создание функции;
- **println("Hello, world!")** – вывод данных в консоль.

Именно функция **main** является запускной в программах на языке Kotlin.

Базовые типы данных в Kotlin:

Название типа	Размер (в битах)	Принимаемые значения
Byte	8	-128 .. 127
Short	16	-32768 .. 32767
Int	32	-2,147,483,648 (-2^{31}) .. 2,147,483,647 ($2^{31} - 1$)
Long	64	-9,223,372,036,854,775,808 (-2^{63}) .. 9,223,372,036,854,775,807 ($2^{63} - 1$)
Float	32	1.40129846432481707e-45 .. 3.40282346638528860e+38
Double	64	4.94065645841246544e-324 .. 1.79769313486231570e+308

Объявление переменных осуществляется с помощью ключевого слова **var** в любом месте программы, причем тип данных указывать не обязательно, если его можно выяснить из самого значения, но можно указать тип данных через двоеточие от переменной:

```
var x = 10; //объявление целого
println("x=$x")
var y = 6.5 //объявление вещественного
println("y=$y")
var z = 5f //объявление вещественного
println("z=$z")
var a: Int = 25 //объявление целого с явным указанием типа
println("a=$a")
```

В консоль будет выведено

```
x=10
y=6.5
z=5.0
a=25
```

В операторе **println** обозначение выводимой переменной происходит внутри кавычек после символа **\$**. Также после этого символа можно в фигурных скобках указывать небольшие выражения, результат которых будет подставлен в выводимую строку.

```
var y = 6.5
var z = 5f
println("z+y=${z+y}")
```

В консоль будет выведено

```
z+y=11.5
```

Для объявления констант можно использовать ключевое слово **val**:

```
val x = 10; //неизменяемая величина
println("x=$x")
val y = 2
println("y=$y")
```

Такие переменные нельзя изменять после первого присвоения значения.

Строки в Kotlin объявляются как обычные переменные, значения им присваиваются в двойных кавычках. Есть тип String. Примеры объявлений строк:

```
var str = "Hello, world!"
println(str)
val str2: String = "Kotlin - это как Java, только Kotlin ☺."
println(str2)
```

Дополнительно есть специальный вид строк в тройных кавычках (так называемые «необработанные строки»). Пример:

```
val s = """
    О сколько нам открытий чудных
    Готовят просвещения дух
    И опыт, сын ошибок трудных,
    И гений, парадоксов друг,
    И случай, бог изобретатель..."""
println(s)
```

В консоль будет выведено

```
О сколько нам открытий чудных
Готовят просвещения дух
И опыт, сын ошибок трудных,
И гений, парадоксов друг,
И случай, бог изобретатель...
```

Для ввода данных с консоли можно использовать функцию `readLine()`, она возвращает строку, и эту строку можно сразу перевести в нужный тип данных, при этом нужно использовать два восклицательных знака в конце этой функции. Пример ввода данных с клавиатуры:

```
print("Input a: ")
val a = readLine()!!.toInt()
print("Input b: ")
val b = readLine()!!.toInt()
println("a=$a b=$b")
```

В консоль будет выведено

```
Input a: 6
```

```
Input b: 3
```

```
a=6 b=3
```

Оператор `!!` делает возможным действия в условиях вероятного получения `null`, т.к. обычно Kotlin запрещает действия с объектом, если он может быть `null`. Это сделано для большей безопасности создаваемых приложений. В нашем случае `readLine()` может вернуть `null`, поэтому напрямую нельзя вызывать `toInt()`, но можно поставить `!!` и тогда компилятор разрешит преобразование. **Но пользоваться этим нужно осторожно, только в тех случаях, когда точно известно, что объект не null !!!**

Операция присваивания (`=`) и арифметические операции (`+`, `-`, `*`, `/`, `%`) заимствованы из языка C (C++/Java).

Оператор `if` также заимствован из C-подобных языков, но при этом еще может возвращать значение, как тернарный оператор `? : .` Например:

```
val a = 5
val b = 3
var x = if (a>b) a else b
println(x)
```

В консоль будет выведено 5.

Также в Kotlin есть оператор `when`, заменивший привычный `switch ... case`. Например, код для определения четности введенного числа:

```
val x1: Int = readLine()!!.toInt()
when (x1 % 2) {
    1 -> print("нечетное")
    0 -> print("четное")
}
```

Можно проверять вхождение в диапазон:

```
when (x1) {
    in 0..9 -> println("x - от 0 до 9")
    in 10..99 -> println("x - от 11 до 99")
    else -> println("другое")
}
```

Можно использовать без параметра и просто проверять выполнение условий:

```
when {
    x1 % 3 == 0 -> println("число делится на 3")
    x1 % 5 == 0 -> println("число делится на 5")
    else -> println("другое")
}
```

Но в данном случае нужно помнить, что при выполнении одного из условий **остальные не проверяются**.

Цикл for представлен в Kotlin в немного отличном от Java виде. Например:

```
for (i in 1..3) {
    println(i)
}
```

Данный цикл выведет в консоль числа от 1 до 3. Другой пример:

```
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

Этот цикл выведет в консоль числа от 6 до 0 с шагом 2 (6 4 2 0).

Циклы while и do ... while работают как в C/C++/Java:

```
var x = 10
while (x > 0) {
    print("x=$x ")
    x--
}
```

В консоль будет выведена строка

x=10 x=9 x=8 x=7 x=6 x=5 x=4 x=3 x=2 x=1

```
var y = 0
do {
    println("${y++}")
} while (y < 10)
```

В консоль будут выведены числа от 0 до 9 включительно.

Задания

1. Написать программу, выясняющую кол-во четных цифр во введенном пользователем числе.
2. Написать программу, выясняющую кол-во нечетных цифр во введенном пользователем числе.
3. Разложить заданное число на простые множители.
4. Число, равное сумме всех своих делителей, включая единицу, называется совершенным. Найти и напечатать все совершенные числа в интервале от 2 до x.
5. Среди всех n-значных чисел указать те, сумма цифр которых равна данному числу k (пользователь вводит n и k).
6. Среди всех n-значных чисел указать те, произведение цифр которых равна данному числу k (пользователь вводит n и k).
7. Найти наибольшую и наименьшую цифры в записи данного натурального числа n.
8. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N-1, у которых сумма всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 44. Числа: 17, 26, 35.

9. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых произведение всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 32. Числа: 6, 16, 23.
10. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых произведение всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 44. Числа: 18, 24.
11. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых сумма всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 32. Числа: 24, 15, 6.
12. Составить программу, которая выводит на экран таблицу умножения от 1 до 9 в десятичной системе счисления.
13. С клавиатуры вводится двузначное число, среди всех четырехзначных чисел вывести на экран те, которые начинаются или заканчиваются этим числом.
14. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна сумме последних двух.
15. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равна сумме последних двух.
16. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равно произведению последних двух.
17. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна произведению последних двух.
18. Пользователь вводит числа x, a, b. Из промежутка от a до b найти все числа, сумма цифр которых дает x.
19. Пользователь вводит числа x, a, b. Из промежутка от a до b найти все числа, разность цифр которых по модулю дает x.
20. Пользователь вводит x, a, b. Из промежутка от a до b найти все числа, произведение цифр которых по модулю дает x.

Лабораторная работа №0.2. Массивы в Kotlin

Для создания массивов в Kotlin есть несколько вариантов.

Для примитивных типов данных существуют специальные классы ByteArray, ShortArray, IntArray и т.д.

Например, создадим массив из 3 целых чисел:

```
val x: IntArray = intArrayOf(5, 2, 8)
x.forEach { print("$it ") } //выведет на экран 5 2 8
```

Переменная x представляет собой массив. Во второй строчке для массива вызывается цикл forEach (есть для каждого класса массивов и списков), в теле которого можно прописать действие, которое будет выполняться с каждым элементом массива, причем сам элемент будет доступен по ссылке it (на самом деле здесь используется подход, основанный на понятии лямбда, о нем в следующей лабораторной). То же самое можно написать и более стандартным способом:

```
for (i in x) {
    print("$i ")
}
```

Но предпочтительнее первый способ как менее подверженный потенциальным ошибкам.

Можно создать массив без заранее определенных элементов и задать их через генератор случайных значений:

```
var arr = IntArray(5)           //создаем массив из 5 целых чисел
for (i in arr.indices){        //делаем цикл по индексам массива
    arr[i] = Random.nextInt(10) //и присваиваем каждому элементу случайное значение
}
```

```
    print("${arr[i]} ")
}
```

Можно совместить объявление массива и присвоение каждому элементу случайного значения:

```
var mas = IntArray(5) { Random.nextInt(10) } //или var mas = IntArray(5, { Random.nextInt(10) })
mas.forEach { print("$it ") }
```

В первой строчке мы задаем размер массива и функцию, которая будет вызываться для каждого элемента массива.

В языке Kotlin есть много встроенных методов для работы с массивом:

- `reversedArray()` – переворачивает массив и возвращает новый массив, старый массив остается без изменений

```
val numbers: IntArray = intArrayOf(1, 2, 3, 4)
var numbers2 = numbers.reversedArray()
println(Arrays.toString(numbers2)) //выведет на экран [4, 3, 2, 1]
```

- `reverse()` – переворачивает массив без создания нового

```
val numbers: IntArray = intArrayOf(1, 2, 3, 4)
numbers.reverse()
println(Arrays.toString(numbers)) //выведет на экран [4, 3, 2, 1]
```

- `sort()` – сортирует массив от меньшего к большему

```
val numbers: IntArray = intArrayOf(7, 5, 8, 4, 9, 6)
numbers.sort()
println(Arrays.toString(numbers)) //выведет на экран [4, 5, 6, 7, 8, 9]
```

Сортировать можно не весь массив, а только определённый диапазон. Для этого в методе `sort()` нужно указать начало и размер диапазона. Например, для сортировки только первых трёх элементов из предыдущего примера нужно написать `numbers.sort(0, 3)`. Тогда будут отсортированы только первые 3 элемента, остальные останутся в прежнем виде.

- `sortDescending()` – сортирует массив от большего к меньшему
- `sortedArray()`, `sortedArrayDescending()` – сортируют и возвращают элементы в новый массив, старый остается без изменений.
- `contains(element)` – проверяет, содержится ли `element` в массиве, возвращает `true` или `false`

```
val numbers: IntArray = intArrayOf(7, 5, 8, 4, 9, 6)
println("Число 4 ${if (!numbers.contains(4)) "не" else ""} содержится в массиве")
```

В данном примере в консоль будет выведено «Число 4 содержится в массиве». Строка

```
println("Число 4 ${if (!numbers.contains(4)) "не" else ""} содержится в массиве")
```

использует оператор `if` для формирования строки, т.е. если не содержится элемент 4, то в строку добавляется частица «не», если содержится – то ничего не добавляется. В данном случае необходимо использовать именно полный формат `if ... else`, этого требует синтаксис встраиваемых выражений внутри `println`.

- `average()` – среднее арифметическое элементов массива

```
val array = arrayOf(2, 3, 2)
println(array.average()) //2.3333333333333335
```

Чтобы вывести число с нужным количеством знаков в дробной части, можно использовать форматирование в стиле языка Си:

```
println("%.2f".format(array.average())) // 2,33
```

- `sum()` – сумма элементов массива
- `min()` – минимальный элемент массива
- `max()` – максимальный элемент массива
- `intersect()` – пересечение массивов

```
val firstArray = intArrayOf(1, 2, 3, 4, 5)
val secondArray = intArrayOf(3, 5, 6, 7, 8)
val intersectedArray = firstArray.intersect(secondArray.asIterable())
intersectedArray.forEach{print("$it ")} // 3 5
```

- `distinct()` – возвращает набор только уникальных элементов массива

```
var ArrNums = intArrayOf(5, 4, 3, 2, 3, 5, 1)
println("Уникальные эл-ты в ArrNum: " + ArrNums.distinct()) //5, 4, 3, 2, 1
```

Задания

1. Дан массив. Удалить из него нули и после каждого числа, оканчивающего на 5, вставить 1.
2. Случайным образом генерируется массив чисел. Пользователь вводит числа a и b . Заменить элемент массива на сумму его соседей, если элемент массива четный и номер его лежит в промежутке от a до b .
3. В одномерном массиве удалить промежуток элементов от максимального до минимального.
4. Дан одномерный массив. Переставить элементы массива задом-наперед.
5. Сформировать одномерный массив случайным образом. Определить количество четных элементов массива, стоящих на четных местах.
6. Задается массив. Определить порядковые номера элементов массива, значения которых содержат последнюю цифру первого элемента массива 2 раза (т.е. в массиве должны быть не только однозначные числа).
7. Сформировать одномерный массив из целых чисел. Вывести на экран индексы тех элементов, которые кратны трем и пяти.
8. Задается массив. Написать программу, которая вычисляет, сколько раз введенная с клавиатуры цифра встречается в массиве.
9. Задается массив. Узнать, какие элементы встречаются в массиве больше одного раза.
10. Даны целые числа a_1, a_2, \dots, a_n . Вывести на печать только те числа, для которых $a_i \geq i$.
11. Дан целочисленный массив с количеством элементов n . Напечатать те его элементы, индексы которых являются степенями двойки.
12. Задана последовательность из N чисел. Определить, сколько среди них чисел меньших K , равных K и больших K .
13. Задан массив действительных чисел. Определить, сколько раз меняется знак в данной последовательности чисел, напечатать номера позиций, в которых происходит смена знака.
14. Задана последовательность N чисел. Вычислить сумму чисел, порядковые номера которых являются простыми числами.
15. Дан массив чисел. Указать те его элементы, которые принадлежат отрезку $[c, d]$.
16. Массив состоит из нулей и единиц. Поставить в начало массива нули, а затем единицы.
17. Дан массив целых положительных чисел. Найти среди них те, которые являются квадратами некоторого числа x .
18. В массиве целых чисел найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
19. Дан целочисленный массив с количеством элементов n . Сжать массив, выбросив из него каждый второй элемент.
20. Дан массив, состоящий из n натуральных чисел. Образовать новый массив, элементами которого будут элементы исходного, оканчивающиеся на цифру k .
21. Даны действительное число x и массив $A[n]$. В массиве найти два члена, среднее арифметическое которых ближе всего к x .
22. Даны два массива A и B . Найти, сколько элементов массива A совпадает с элементами массива B .

Лабораторная работа №0.3. ООП в Kotlin

Для создания класса с нужными полями и конструктором в языке Kotlin достаточно всего одной строчки:

```
open class Auto(var firm: String = "Без названия", var maxSpeed: Int = 0)
```

В данном случае создается класс Auto с двумя полями (firm и maxSpeed) и с конструктором, который по умолчанию присваивает этим полям нужные значения. Для проверки используем следующий код:

```
fun main(){
    val myAuto: Auto = Auto("Ford", 400)
    println("${myAuto.firm} ${myAuto.maxSpeed}")
    val myAuto2 = Auto()
    println("${myAuto2.firm} ${myAuto2.maxSpeed}")
    val myAuto3 = Auto(maxSpeed = 200)
    println("${myAuto3.firm} ${myAuto3.maxSpeed}")
}
```

В консоль будет выведено

```
Ford 400
Без названия 0
Без названия 200
```

Т.е. мы можем вызывать конструктор с нужными параметрами, без параметров и с частичными параметрами, указав имя параметра.

Теперь можно наследоваться от созданного класса и расширять его функциональность:

```
class Car (firm: String = "No",
    maxSpeed: Int = 0,
```



```

var model: String = "Нет",
var numDoors: Int = 4,
var fullTime: Boolean = false): Auto(firm, maxSpeed) {

    override fun toString() = "Легковая машина (фирма=$firm, максСкорость=$maxSpeed, модель=$model, " +
        "кол-во дверей=$numDoors, полноприводный=${if (fullTime) "да" else "нет"})"
}

```

В данном случае мы создали класс Car, который унаследован от класса Auto. При этом мы указываем все поля создаваемого класса, в том числе и те, которые наследуются из класса Auto, и задаем им значения по умолчанию. Те параметры, которые наследуются из класса Auto (это firm и maxSpeed), мы передаем в класс Auto.

Также в новом классе мы переопределяем метод toString (это стандартный метод, он определен для всех объектов по умолчанию, когда мы выводим объект в консоль или в строку, но информация, которую он выводит, не очень полезная, например, println(myAuto3) выведет на экран Lab3.Auto@2812cbfa). Для перегрузки (или переопределения) используется ключевое слово override перед именем перегружаемой функции. В самой функции мы просто написали формат вывода информации об объекте.

Теперь можем проверить, как выводятся на экран объекты класса Car:

```

val myCar1 = Car(firm = "Mers", numDoors = 2)
println(myCar1)

```

В консоль будет введено

Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Мы создали объект класса Car, используя только 2 параметра, остальные были взяты из конструктора класса, в котором мы указали значения по умолчанию.

Создадим еще один класс-наследник от Auto – Truck (грузовой автомобиль).

```

class Truck (firm: String = "No",
    maxSpeed: Int = 0,
    var model: String = "Нет",
    var power: Int = 0,
    var trailer: Boolean = false) : Auto(firm, maxSpeed){

    fun setAllInfo() {
        print("Введите фирму-производитель грузового авто: ")
        firm = readLine()!!
        print("Введите максимальную скорость грузового авто: ")
        maxSpeed = readLine()!!.toInt();
        print("Введите модель грузового авто: ")
        model = readLine()!!
        print("Введите мощность грузового авто: ");
        power = readLine()!!.toInt()
        print("Введите признак прицепа грузового авто (true/false): ");
        trailer = readLine()!!.toBoolean()
        println()
    }

    override fun toString() = "\n\tГрузовик"+ "\n\t" + " Фирма: "+firm+"\n\t"+
        " Максимальная скорость: " +maxSpeed+"\n\t" + " Модель: "+model+"\n\t"+
        " Мощность: "+power+"\n\t" + " Признак прицепа: " +trailer+"\n"
}

```

В этом классе добавлена функция setAllInfo, которая позволяет ввести всю нужную информацию с клавиатуры и присвоить ее полям создаваемого объекта класса Truck. Можно проверить работу с классом:

```

val myTruck = Truck()
myTruck.setAllInfo()
println(myTruck)

```

Пример ввода с консоли и вывода полной информации об объекте:

Введите фирму-производитель грузового авто: Dove
 Введите максимальную скорость грузового авто: 300
 Введите модель грузового авто: k3
 Введите мощность грузового авто: 500
 Введите признак прицепа грузового авто (true/false): true

Грузовик
 Фирма: Dove
 Максимальная скорость: 300

Модель: k3
Мощность: 500
Признак прицепа: true

Теперь можно создать класс-агрегатор (гараж) для хранения наших объектов:

```
class GarageAuto {  
    private var masAuto = ArrayList<Auto>();  
    fun addAuto (auto: Auto) { //для добавления объектов в массив  
        masAuto.add(auto)  
    }  
    fun findAuto(a: Auto) : Boolean{ //для поиска объектов в массиве  
        return masAuto.contains(a)  
    }  
    override fun toString(): String { //для вывода в консоль  
        var str = "В гараже:\n "  
        for (a in masAuto) {  
            str = str+("\t" + a+"\n")  
        }  
        return str  
    }  
}
```

Объекты в этом классе хранятся в массиве типа ArrayList. Это шаблонный класс, мы указали в угловых скобках тип объектов, которые будут в нем храниться (Auto). Т.к. мы указали базовый класс, то и все объекты классов наследников смогут храниться в этом массиве.

Для проверки работы нового класса можно использовать следующий код:

```
val myAuto: Auto = Auto("Ford", 400)  
val myCar1 = Car(firm = "Mers", numDoors = 2)  
val myTruck = Truck("Kamaz", 200, "Masters", 500, false)  
var myGarage = GarageAuto()  
myGarage.addAuto(myAuto)  
myGarage.addAuto(myCar1)  
myGarage.addAuto(myTruck)  
println(myGarage)  
if (myGarage.findAuto(myCar1)) println("Есть машина $myCar1")
```

Вывод программы:

В гараже:

Lab3.Auto@96532d6

Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Грузовик

Фирма: Kamaz

Максимальная скорость: 200

Модель: Masters

Мощность: 500

Признак прицепа: false

Есть машина Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Также в Kotlin есть специальный оператор is, который позволяет определить принадлежность объекта к определенному классу. Например, при выполнении следующего кода:

```
for (a in masAuto) {  
    if (a is Car) println("Легковая машина")  
}
```

текст **Легковая машина** будет выведен столько раз, сколько легковых машин содержится в гараже.

Лямбда-функция – это вид анонимной функции. Состоит как правило из двух частей: до знака → и после него, но можно и без первой части и знака →. В примере ниже, у объекта fields (это хэш-мап) вызывается метод forEach (это цикл Для Каждого по всем парам ключ-значение, хранящимся в хэш-мапе). И дальше в фигурных скобках идет лямбда-функция. Ей в качестве параметров даются переменные k и v (ключ и значение каждой пары соответственно), они указаны до стрелки →. После стрелки указываются действия, которые нужно выполнить с участием переменных k и v.

```

var fields = HashMap<String, Int>()
fields.set("Иванов", 25)
fields.set("Петров", 30)
fields.set("Сидоров", 35)
fields.forEach { k, v ->
    println("$k - $v")
}

```

Вывод в консоль

```

Иванов - 25
Сидоров - 35
Петров - 30

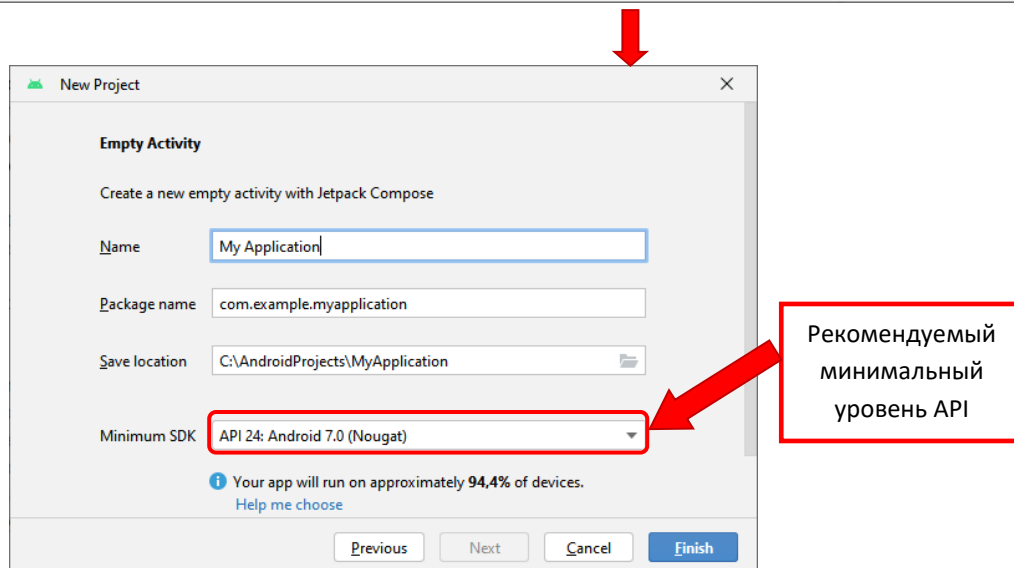
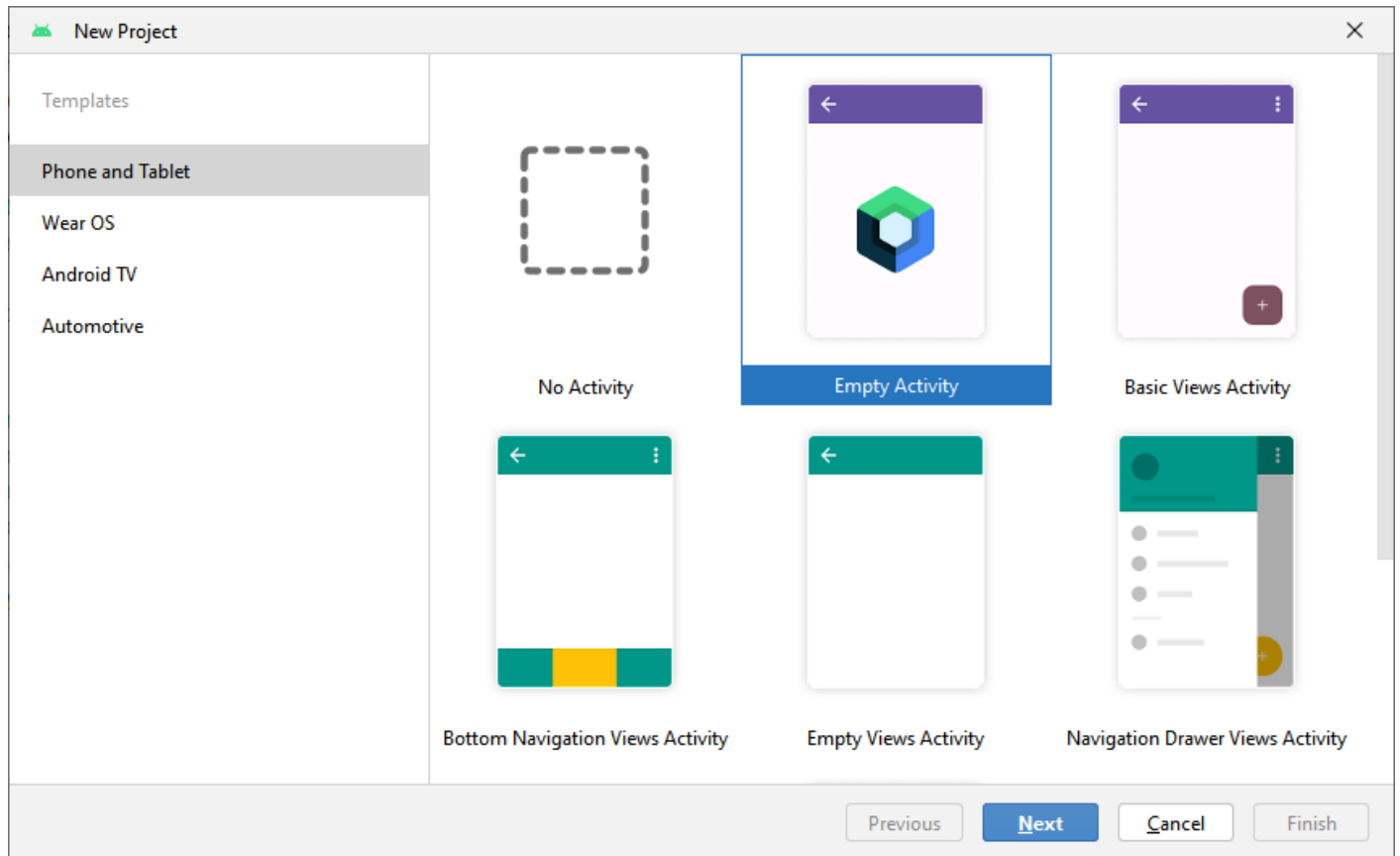
```

Задания (создать классы, в них предусмотреть различные поля классов и методы для работы):

1. Базовый класс – учащийся. Производные – школьник и студент. Создать класс Конференция, который может содержать оба вида учащихся. Предусмотреть метод подсчета участников конференции отдельно по школьникам и по студентам (использовать оператор is).
2. Базовый класс – работник. Производные – работник на почасовой оплате и на окладе. Создать класс Предприятие, который может содержать оба вида работников. Предусмотреть метод подсчета работников отдельно на почасовой оплате и на окладе (использовать оператор is).
3. Базовый класс – компьютер. Производные – ноутбук и смартфон. Создать класс РемонтСервис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно ремонтируемых ноутбуков и смартфонов (использовать оператор is).
4. Базовый класс – печатные издания. Производные – книги и журналы. Создать класс КнижныйМагазин, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно книг и журналов (использовать оператор is).
5. Базовый класс – помещения. Производные – квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор is).
6. Базовый класс – файл. Производные – звуковой файл и видео-файл. Создать класс Каталог, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно звуковых и видео-файлов (использовать оператор is).
7. Базовый класс – летательный аппарат. Производные – самолет и вертолет. Создать класс Авиакомпания, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно самолетов и вертолетов (использовать оператор is).
8. Базовый класс – соревнование. Производные – командные соревнования и личные. Создать класс Чемпионат, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно командных соревнований и личных (использовать оператор is).
9. Базовый класс – мебель. Производные – диван и шкаф. Создать класс Комната, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно диванов и шкафов (использовать оператор is).
10. Базовый класс – оружие. Производные – огнестрельное и холодное. Создать класс ОружейнаяПалата, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно огнестрельного и холодного оружия (использовать оператор is).
11. Базовый класс – оргтехника. Производные – принтер и сканер. Создать класс Офис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно принтеров и сканеров (использовать оператор is).
12. Базовый класс – СМИ. Производные – телеканал и газета. Создать класс Холдинг, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно телеканалов и газет (использовать оператор is).

Лабораторная работа №1. Введение в программирование для ОС Android

Для создания базового Android-проекта достаточно запустить Android Studio (начиная с версии 2022.2.1) или в уже запущенной IDE выбрать пункт меню File – New – New Project и последовательно настроить проект (может выглядеть немного по-разному в зависимости от версии Android Studio, главное выбрать Empty Activity):



После нажатия кнопки Finish и по прошествии некоторого времени (пока загрузится базовый шаблон проекта) появится окно с созданным проектом (рис.1).

В папке java находятся исходные тексты программного кода приложения (основной файл – MainActivity). В папке res находятся основные ресурсы приложения, например, файлы строковых ресурсов (папка values, основной файл strings.xml) и другие папки.

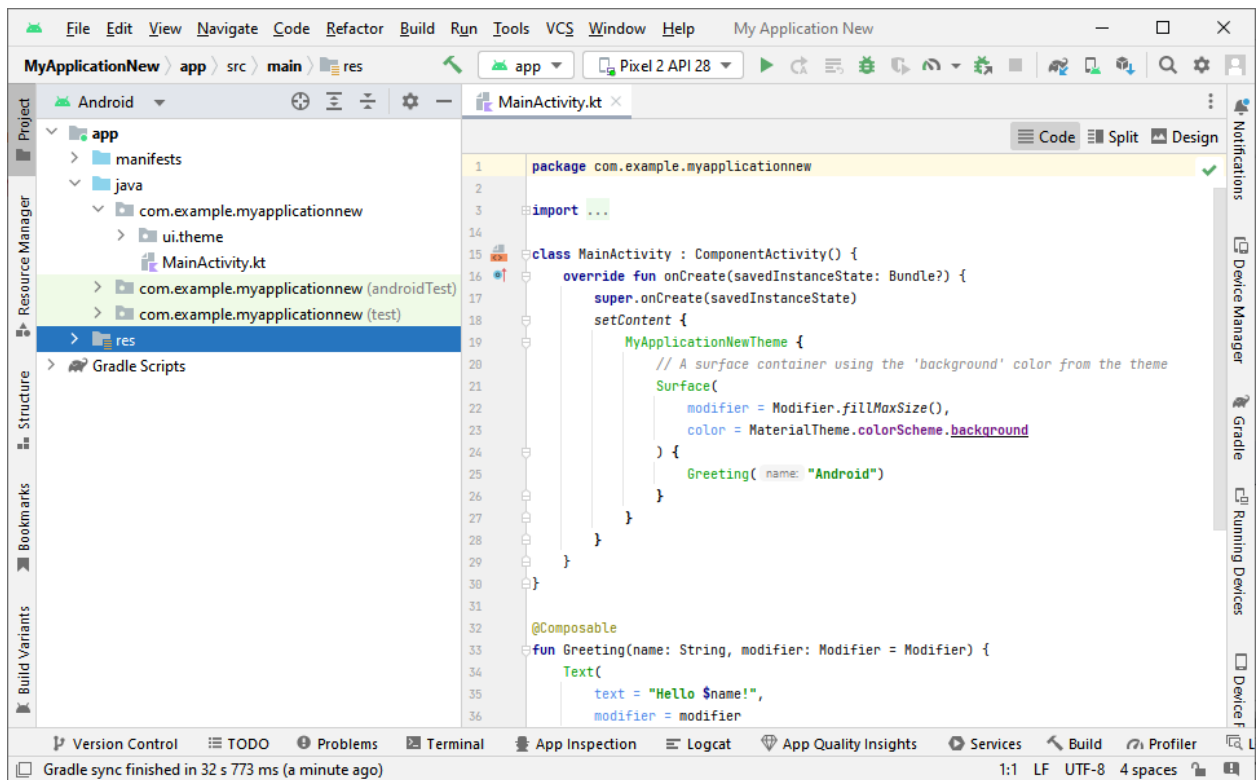


Рис. 1. Окно созданного проекта.



Рис. 2. Вызов менеджера виртуальных устройств.

Для запуска проекта необходимо создать виртуальное Android-устройство. Нажимаем кнопку менеджера виртуальных Android-устройств (рис. 2) (в правой части окна сверху), справа появится панель Device Manager, в ней нажимаем кнопку Create Device и выбираем устройство Pixel 2 (можно и другой). Нажимаем кнопку Next. Появится окно с выбором нужного образа ОС Android, который будет на устройстве (этот образ должен быть установлен, иначе его установка будет происходить в процессе настройки виртуального устройства и придется дожидаться его окончания) (рис. 3). Переходим на вкладку x86 Images и выбираем Pie x86_64 Android 9.0 (Google API). Если возле слова Pie есть значок \updownarrow , то нужно нажать на него и начнется загрузка образа ОС.

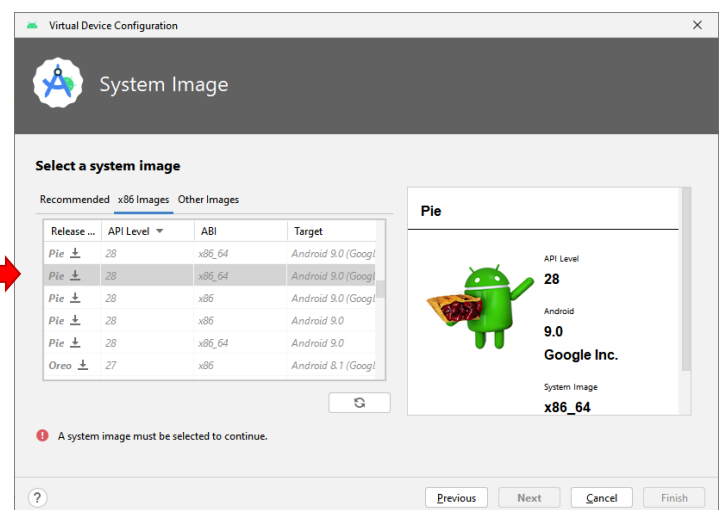
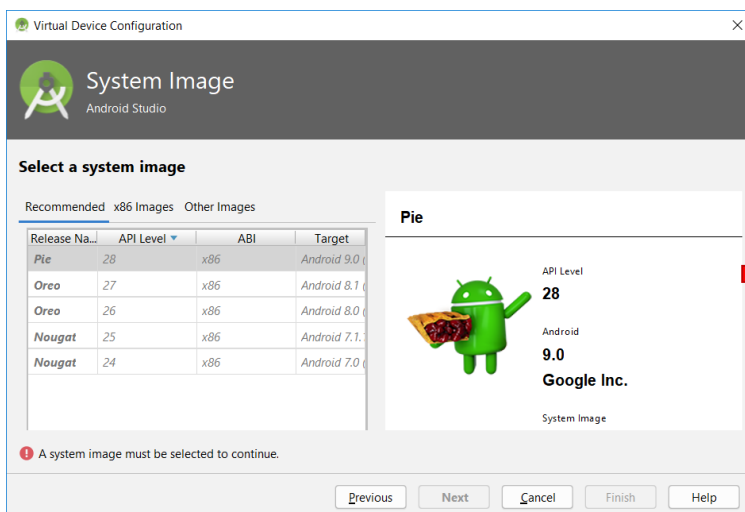


Рис. 3. Выбор начальных параметров устройства.

После загрузки образа ОС (или если образ уже был скачан) нажимаем Next. Появится окно с окончательной настройкой устройства (рис. 4). Нажимаем кнопку Show Advanced Settings и проверяем настройки по примеру на рис. 4. Обычно ничего менять не нужно, но всякое бывает.

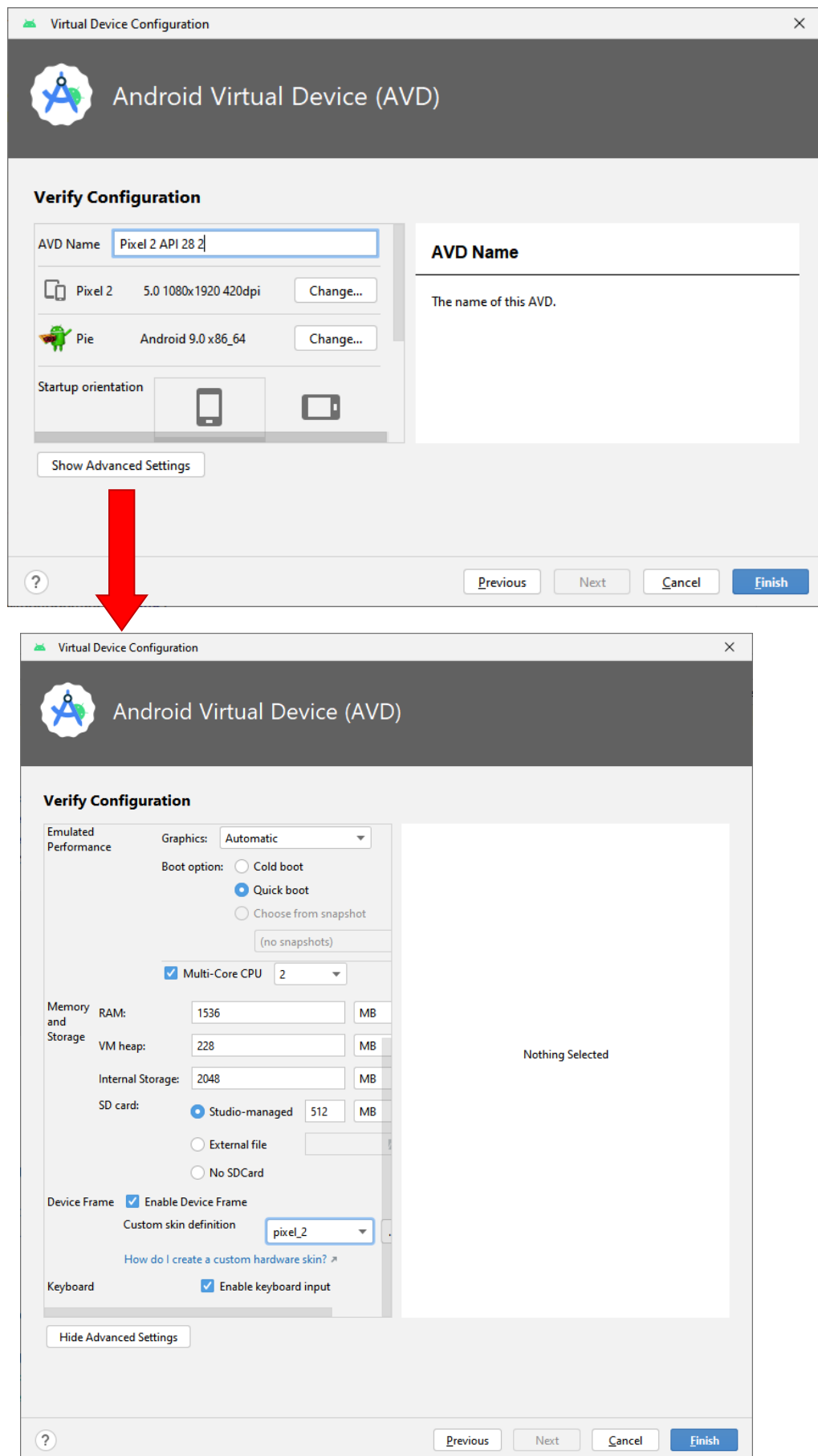


Рис. 4. Окончательная настройка устройства.

Жмем Finish. В панели Device Manager появится наше устройство. В строке с нужным устройством нажимаем запуск – символ ► (рис. 5). Начнется запуск устройства, это займет некоторое время, поэтому в дальнейшем лучше не закрывать его (рис. 6).

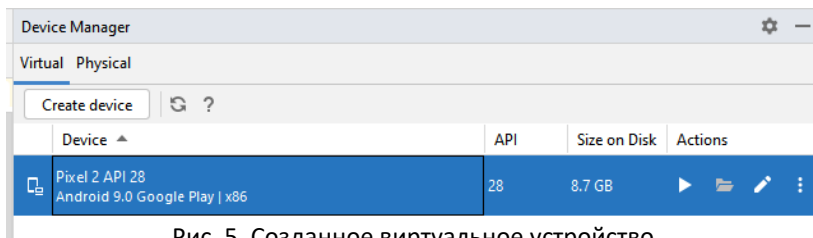


Рис. 5. Созданное виртуальное устройство.

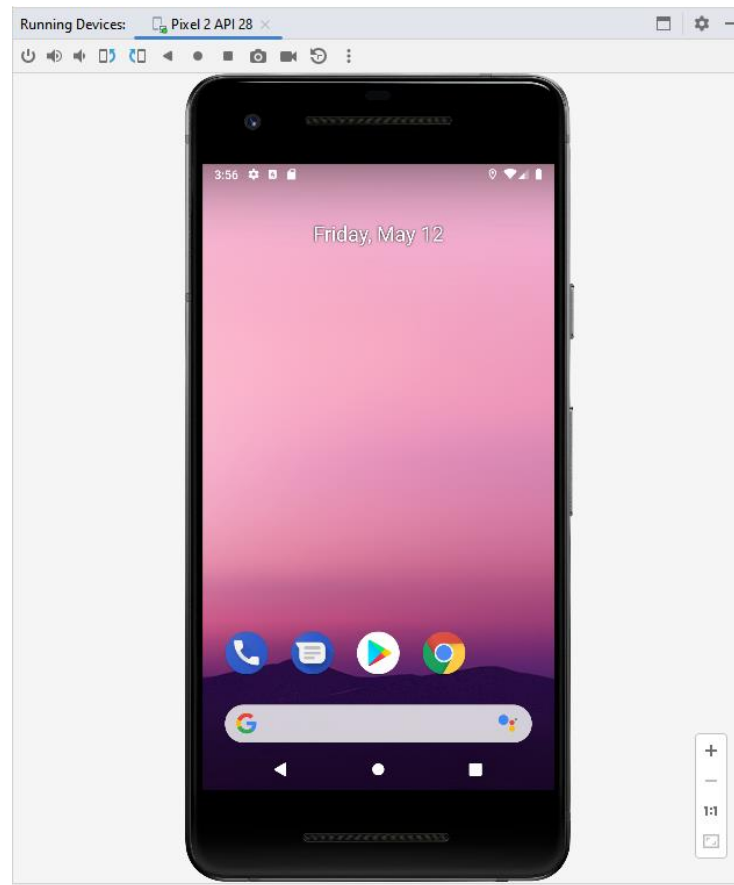


Рис. 6. Виртуальное устройство.

Можно приступить к созданию приложения. Для начала познакомимся с базовыми принципами подхода Jetpack Compose.

Построение UI (User Interface – интерфейс пользователя) происходит непосредственно в коде, используя декларативный подход (похоже на работу с веб-фреймворками вроде React или Flutter).

Единицей построения интерфейса является функция, помеченная аннотацией `@Composable`. Таким образом, построение интерфейса заключается в композиции таких функций. Проект, который создается с шаблоном Empty Activity, уже содержит в файле `MainActivity.kt` (основной файл проекта) образец такого подхода с примером для первого запуска. Рассмотрим его:

```
class MainActivity : AppCompatActivity() { //основной класс нашего проекта, он будет запускаться
    override fun onCreate(savedInstanceState: Bundle?) { //ф-ия, которая срабатывает при запуске
                                                //или перезапуске программы
        super.onCreate(savedInstanceState)
        setContent { //секция для установки содержимого главного окна приложения
            MyApplicationNewTheme { //секция с основной цветовой темой приложения
                //секция «поверхности», которая содержит параметры окна и вызывает функции
                Surface( //для формирования графического интерфейса
                    modifier = Modifier.fillMaxSize(), //параметр для заполнения максимального
                                                //свободного места на экране
                    color = MaterialTheme.colorScheme.background //для цвета фона
                ) {
                    Greeting("Android") //вызываем ф-ию с содержимым окна (описана ниже)
                }
            }
        }
    }
}
```

```

@Composable //обязательный префикс-аннотация для всех функций в стиле Compose
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!", //текст для вывода
        modifier = modifier //модификатор
    )
}
//здесь прописывается вызов нашей вышеописанной ф-ии
//для предпросмотра без запуска программы
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    MyApplicationNewTheme {
        Greeting("Android")
    }
}

```

При запуске проекта в эмуляторе появится окно программы как на рис. 7.

В данном случае мы просто выводим в окно программы текст “Hello Android”. Для этого используется компонент Text функции Greeting с нужными параметрами. В нашем примере используются параметры text и modifier, более подробно параметры описаны ниже:

- text: объект String, который представляет выводимый текст
- modifier: объект Modifier, который представляет применяемые к компоненту модификаторы
- color: объект Color, который представляет цвет текста. По умолчанию имеет значение Color.Unspecified
- fontSize: объект TextUnit, который представляет размер шрифта. По умолчанию равен TextUnit.Unspecified
- fontStyle: объект FontStyle?, который представляет стиль шрифта. По умолчанию равен null
- fontWeight: объект FontWeight?, который представляет толщину шрифта. По умолчанию равен null
- fontFamily: объект FontFamily?, который представляет тип шрифта. По умолчанию равен null
- letterSpacing: объект TextUnit, который представляет отступы между символами. По умолчанию равен TextUnit.Unspecified
- textDecoration: объект TextDecoration?, который представляет тип декораций (например, подчеркивание), применяемых к тексту. По умолчанию равен null
- textAlign: объект TextAlign?, который представляет выравнивание текста. По умолчанию равен null
- lineHeight: объект TextUnit, который представляет высоту строки текста. По умолчанию равен TextUnit.Unspecified
- overflow: объект TextOverflow, который определяет поведение текста при его выходе за границы контейнера. По умолчанию равен TextOverflow.Clip
- softWrap: объект Boolean, который определяет, должен ли текст переноситься при завершении строки. При значении false текст не переносится, как будто строка имеет бесконечную длину. По умолчанию равен true
- maxLines: объект Int, который представляет максимальное количество строк. Если текст превысил установленное количество строк, то он усекается в соответствии с параметрами overflow и softWrap. По умолчанию равен Int.MAX_VALUE
- onTextLayout: объект (TextLayoutResult) -> Unit, который представляет функцию, выполняемую при определении компоновки текста.
- style: объект TextStyle, который представляет стиль текста. Значение по умолчанию - LocalTextStyle.current

Попробуем применить некоторые параметры для преобразования выводимого текста:

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyApplicationNewTheme {
                Surface(

```

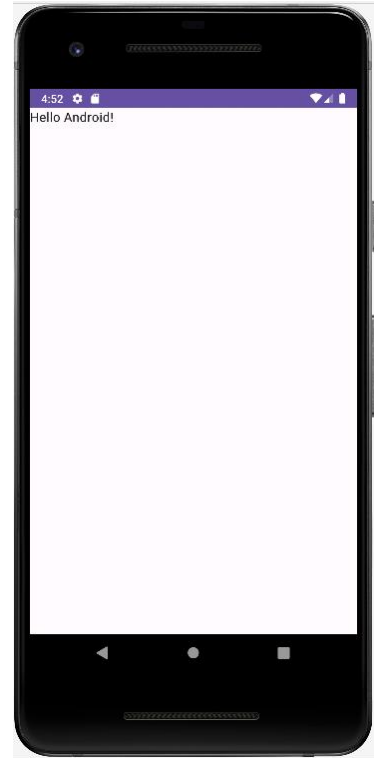


Рис. 7. Первый запуск программы.


```

        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        Column() { // вертикальная колонка для размещения объектов
            TextCenter("Android Center")
            TextGreenBig("Big Green Android")
            TextRightBold(name = "Right Bold Android")
            TextCursiveUnderlined("Cursive UnderLined Android")
        }
    }
}

@Composable
fun TextCenter(name: String) {
    Text(
        text = "Hello $name!",
        textAlign = TextAlign.Center, //выравниваем по центру
        //применяем модификатор, чтобы компонент занял все возможное
        modifier = Modifier.fillMaxWidth(1f) // пространство
    )
}

@Composable
fun TextGreenBig(name: String) {
    Text(
        text = "Hello $name!",
        color = Color.Green, //меняем цвет
        fontSize = 24.sp, //меняем размер шрифта
        modifier = Modifier.fillMaxWidth(1f)
    )
}

@Composable
fun TextRightBold(name: String) {
    Text(
        text = "Hello $name!",
        textAlign = TextAlign.Right, //выравниваем справа
        fontWeight = FontWeight.Bold, //делаем жирным
        modifier = Modifier.fillMaxWidth(1f)
    )
}

@Composable
fun TextCursiveUnderlined(name: String) {
    Text(
        text = "Hello $name!",
        fontFamily = FontFamily.Cursive, //делаем курсивом
        textDecoration = TextDecoration.Underline, //и подчеркнутым
        fontSize = 24.sp, //меняем размер шрифта
        modifier = Modifier.fillMaxWidth(1f)
    )
}

```

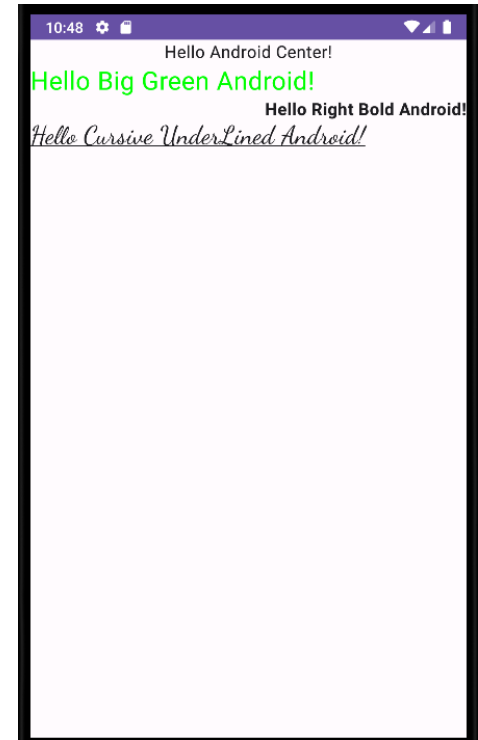


Рис. 8. Различные параметры текста.

Результат запуска программы представлен на рис. 8. Более подробно с параметрами вывода текста можно познакомиться по ссылке <https://developer.android.com/jetpack/compose/text>.

Теперь попробуем сделать ~~синхронизатор~~ что-нибудь более полезное – программу для суммирования нескольких чисел:

```

class MainActivity : ComponentActivity() {
    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyApplicationComposeTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MyUI() //вызываем функцию для построения интерфейса
                }
            }
        }
    }
}

```

```

}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MyUI() { функция для построения интерфейса
    var value1 by remember { //объект для работы с текстом, для TextField
        mutableStateOf("") //его начальное значение
    } //в функцию mutableStateOf() в качестве параметра передается отслеживаемое значение
    var value2 by remember { //объект для работы с текстом, для кнопки и суммы
        mutableStateOf("") //его начальное значение
    }
    var result = 0 //будущий результат
    val context = LocalContext.current //объект-контекст, нужен для всплывающего сообщения
    Column( //создаем колонку для вертикального размещения объектов
        modifier = Modifier.fillMaxSize(), //заполняем всё доступное пространство
        horizontalAlignment = Alignment.CenterHorizontally, //по центру горизонтально
        verticalArrangement = Arrangement.Center //и вертикально
    ) {
        TextField( //текстовое поле для ввода данных
            value = value1, //связываем текст из поля с созданным ранее объектом
            onChange = { newText -> //обработчик ввода значений в поле
                value1 = newText //все изменения сохраняем в наш объект
            },
            textStyle = TextStyle( //объект для изменения стиля текста
                fontSize = 24.sp //увеличиваем шрифт
            ),
            //меняем тип допустимых символов для ввода - только цифры
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number)
        )
        Button( //кнопка
            modifier = Modifier.padding(30.dp), //делаем отступ сверху и снизу от кнопки
            onClick = { //обработчик нажатия на кнопку
                //разделяем введенные значения через пробел и сохраняем их в виде списка
                val numbers = value1.split(" ").toList()
                //показываем всплывающее сообщение для проверки списка с числами
                Toast.makeText(context, "list = $numbers", Toast.LENGTH_LONG).show()
                //в переменную result сохраняем сумму эл-ов списка
                result = numbers.sumOf { it.toInt() }
                value2 = result.toString() //и сохраняем сумму во второй объект для текста
            }
        ) {
            Text("Ok", fontSize = 24.sp) //текстовая надпись для кнопки
        }
        Text( //текст для вывода результата с нужными параметрами
            text = "sum = $value2",
            color = Color.Green, //меняем цвет
            fontSize = 24.sp
        )
    }
}

```

Вместе с Column можно использовать Row – для горизонтального размещения объектов.

Результат работы программы показан на рис. 9.

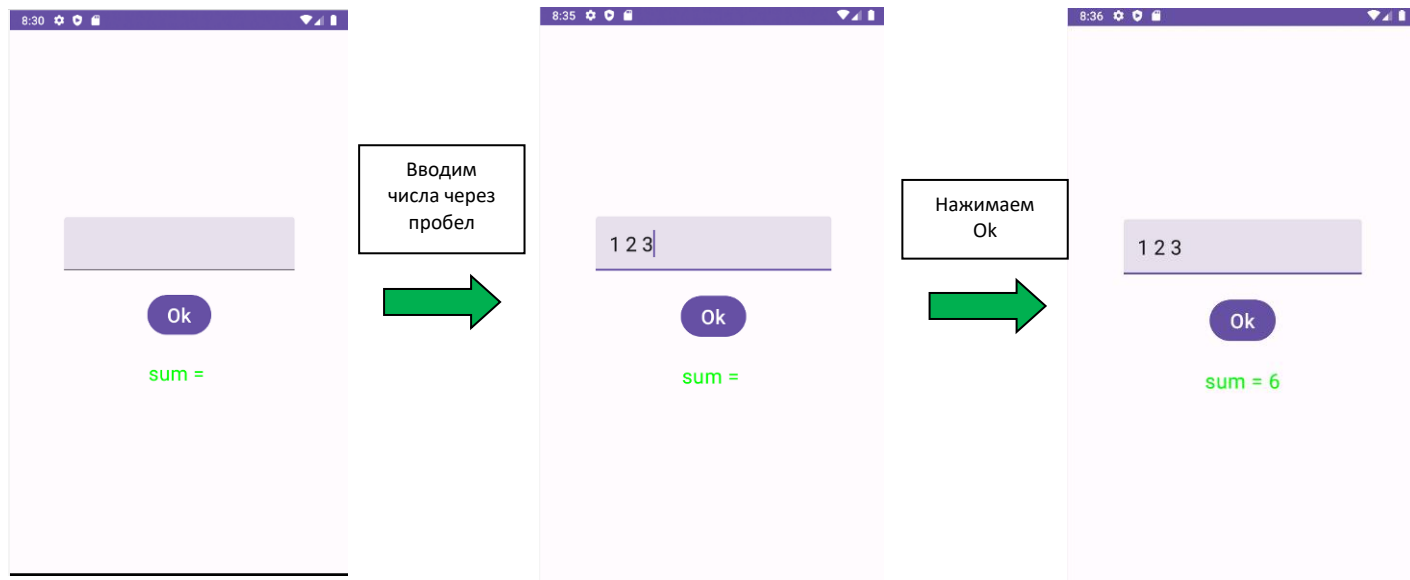


Рис. 9. Сложение чисел.

Задачи:

1. Создать программу для перемножения 3-х чисел.
2. Создать программу для вычисления среднегеометрического 3-х чисел.
3. Создать программу для вычисления среднеарифметического 3-х чисел.
4. Создать программу для вычисления гипотенузы, периметра и площади прямоугольного треугольника по введенным значениям катетов.
5. Создать программу для вычисления определителя квадратной матрицы 2x2.
6. Создать программу для вычисления суммы первых n членов арифметической прогрессии (вводятся a , d , n , выводятся все члены последовательности и их сумма).
7. Создать программу для вычисления суммы первых n членов геометрической прогрессии (вводятся a , q , n , выводятся все члены последовательности и их сумма).
8. Создать программу для вычисления произведения введенных чисел.
9. Создать программу для вычисления среднегеометрического введенных чисел.
10. Создать программу для поиска минимального, максимального и среднего значения среди введенных чисел.
11. Создать программу для вычисления среднеарифметического введенных чисел.
12. Создать программу для выяснения кол-ва положительных, отрицательных и нулевых элементов среди введенных чисел.
13. Создать программу для выяснения суммы элементов среди введенных чисел, кратных p .

Лабораторная работа №2. Использование списков для отображения набора визуальных объектов

Для отображения визуальных объектов интерфейса в виде списка можно использовать компонент LazyColumn. Простой пример его использования:

```
class MainActivity : AppCompatActivity() {
    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyApplicationComposeTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    EasyList()
                }
            }
        }
    }
}

@Composable
fun EasyList() {
    LazyColumn { //объект для представления списка
        item { // добавляем 1 элемент
            Text(text = "First item", fontSize = 24.sp)
        }

        items(5) { index -> // добавляем 5 эл-ов
            Text(text = "Item: $index", fontSize = 24.sp)
        }

        item { // добавляем 1 элемент
            Text(text = "Last item", fontSize = 24.sp)
        }
    }
}
```

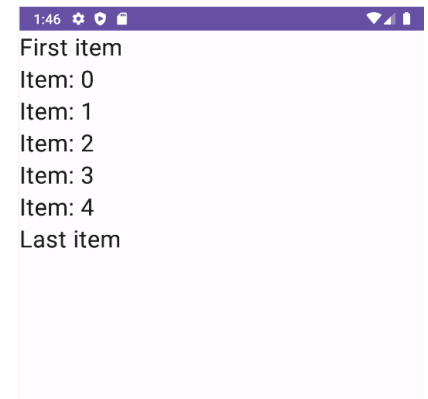


Рис. 10. Список элементов.

Получившееся приложение можно увидеть на рис. 10.

Но для более комплексного и функционального использования списка из визуальных элементов удобнее использовать отдельный класс, наследующийся от ViewModel. Создадим такой на примере списка, представляющего собой описание языков программирования. Для этого добавим к проекту файл ItemViewModel.kt со следующим содержимым:

```
//создаем дата-класс для представления языка программирования
data class ProgrLang(val name: String, val year: Int)

class ItemViewModel : ViewModel() {

    private var langList = mutableListOf( //создаем список из языков программирования
        ProgrLang("Basic", 1964),
        ProgrLang("Pascal", 1975),
        ProgrLang("C", 1972),
        ProgrLang("C++", 1983),
        ProgrLang("C#", 2000),
        ProgrLang("Java", 1995),
        ProgrLang("Python", 1991),
        ProgrLang("JavaScript", 1995),
        ProgrLang("Kotlin", 2011)
    )

    //добавляем объект, который будет отвечать за изменения в созданном списке
    private val _langListFlow = MutableStateFlow(langList)
    //и геттер для него, который его возвращает
    val langListFlow: StateFlow<List<ProgrLang>> get() = _langListFlow

    fun clearList() { //метод для очистки списка, понадобится в лаб.раб.№5
        langList.clear()
    }

    fun addLangToHead(lang: ProgrLang) { //метод для добавления нового языка в начало списка
        langList.add(0, lang)
    }

    fun addLangToEnd(lang: ProgrLang) { //метод для добавления нового языка в конец списка
```

```

        langList.add( lang)
    }

    fun removeItem(item: ProgrLang) { //метод для удаления элемента из списка
        val index = langList.indexOf(item)
        langList.remove(langList[index])
    }
}

```

Далее основной файл проекта MainActivity:

```

class MainActivity : ComponentActivity() {
    private val viewModel = ItemViewModel() //модель данных нашего списка

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView {
            ComposeExampleTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    LazyColumn( //объект для представления списка
                        //добавляем отступы между эл-ми списка
                        verticalArrangement = Arrangement.spacedBy(12.dp),
                        modifier = Modifier
                            .fillMaxSize()
                            .background(Color.White)
                    ) {
                        // !!! нужен import androidx.compose.foundation.lazy.items
                        //создаем эл-ты визуального списка из модели
                        items(
                            items = viewModel.langListFlow.value, //сами эл-ты
                            key = { lang -> lang.name}, //ключевое поле
                            itemContent = { item -> //содержимое эл-та списка
                                ListRow(item) //вызываем метод для формирования каждого эл-та списка
                            }
                        )
                    }
                }
            }
        }
    }
}

@Composable
fun ListRow(item: ProgrLang){ //ф-ия для создания ряда с данными для LazyColumn
    Row( //создаем ряд с данными
        verticalAlignment = Alignment.CenterVertically,
        modifier = Modifier
            .wrapContentSize()
            .fillMaxWidth()
            .border(BorderStroke(2.dp, Color.Blue)) //синяя граница для каждого эл-та списка
    ) {
        Text( // поле с текстом для названия языка
            text = item.name, //берем имя языка
            fontSize = 24.sp, //устанавливаем размер шрифта
            fontWeight = FontWeight.SemiBold, //делаем текст жирным
            //и добавляем отступ слева
            modifier = Modifier.padding(start = 20.dp)
        )
        Text( // поле с текстом для года создания языка
            text = item.year.toString(), //берем год и преобразуем в строку
            fontSize = 20.sp, //устанавливаем размер шрифта
            modifier = Modifier.padding(10.dp), //добавляем отступ
            fontStyle = FontStyle.Italic //и делаем шрифт курсивом
        )
    }
}

```

Basic 1964
Pascal 1975
C 1972
C++ 1983
C# 2000
Java 1995
Python 1991
JavaScript 1995
Kotlin 2011

Рис. 11. Список языков программирования.

Пример запуска приложения можно увидеть на рис. 11.

Объекты внутри ряда для LazyColumn можно располагать как угодно, например, мы можем созданные текстовые поля заключить внутрь Column, и тогда имя языка и год будут располагаться не горизонтально, а вертикально (рис. 12):

```
Column {
    Text(
        text = model.name,
        fontSize = 24.sp,
        fontWeight = FontWeight.SemiBold,
        modifier = Modifier.padding(start = 20.dp)
    )
    Text(
        text = model.year.toString(),
        fontSize = 20.sp,
        modifier = Modifier.padding(10.dp),
        fontStyle = FontStyle.Italic
    )
}
```

Basic
1964
Pascal
1975
C
1972
C++
1983
C#
2000
Java
1995
Python
1991

Рис. 12. Элементы списка с вертикальным размещением полей.

Таким образом можно комбинировать различное размещение объектов.

Теперь попробуем сделать добавление нового элемента в список.

Для этого нужно перед списком вставить поля для ввода данных и кнопку для добавления нового элемента в список. Изменяем секцию setContent в методе onCreate :

```
setContent {
    val lazyListState = rememberLazyListState() //объект для сохранения состояния списка
    ComposeExampleTheme {
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            Column(Modifier.fillMaxSize()) { //создаем колонку
                MakeInputPart(viewModel, lazyListState) //вызываем ф-ию для создания полей ввода данных
                MakeList(viewModel, lazyListState) //вызываем ф-ию для самого списка с данными
            }
        }
    }
}
```

Создаем функцию для верхней части приложения, в ней будут поля для ввода данных и кнопка:

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MakeInputPart(model: ItemViewModel, lazyListState: LazyListState) {
    var langName by remember { //объект для работы с текстом, для названия языка
        mutableStateOf("") //его начальное значение
    } //в функцию mutableStateOf() в качестве параметра передается отслеживаемое значение
    var langYear by remember { //объект для работы с текстом, для года создания языка
        mutableStateOf(0) //его начальное значение
    }
    val scope = rememberCoroutineScope() //объект для прокручивания списка при вставке нового эл-та
    Row( //ряд для расположения эл-ов
        verticalAlignment = Alignment.CenterVertically, //центрируем по вертикали
        horizontalArrangement = Arrangement.spacedBy(10.dp), //и добавляем отступы между эл-ми
    ) {
        TextField( //текстовое поле для ввода имени языка
            value = langName, //связываем текст из поля с созданным ранее объектом
            onChange = { newText -> //обработчик ввода значений в поле
                langName = newText //все изменения сохраняем в наш объект
            },
            textStyle = TextStyle( //объект для изменения стиля текста
                fontSize = 20.sp //увеличиваем шрифт
            ),
            label = { Text("Название") }, //это надпись в текстовом поле
            modifier = Modifier.weight(2f) //это вес колонки.Нужен для распределения долей в ряду.
        ) //Контейнер Row позволяет назначить вложенным компонентам ширину в соответствии с их весом.
        //Поэтому полям с данными назначаем вес 2, кнопке вес 1, получается сумма
        //всех весов будет 5, и для полей с весом 2 будет выделяться по 2/5 от всей ширины ряда, для
        //кнопки с весом 1 будет выделяться 1/5 от всей ширины ряда
        TextField( //текстовое поле для ввода года создания языка
            value = langYear.toString(), //связываем текст из поля с созданным ранее объектом
            onChange = { newText -> //обработчик ввода значений в поле
                langYear = if (newText != "") newText.toInt() else 0 //в нужный формат
            },
            textStyle = TextStyle( //объект для изменения стиля текста
                fontSize = 20.sp //увеличиваем шрифт
            ),
            //и меняем тип допустимых символов для ввода - только цифры
        )
    }
}
```

```

keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
label = { Text("Год создания") },
modifier = Modifier.weight(2f) //назначаем вес поля
)
Button( //кнопка для добавления нового языка
onClick = { //при нажатии кнопки делаем отладочный вывод
println("added $langName $langYear")
//и добавляем в начало списка новый язык с нужными параметрами
model.addLangToHead(ProgrLang(langName, langYear))
scope.launch { //прокручиваем список, чтобы был виден добавленный элемент
lazyListState.scrollToItem(0)
}
langName = "" //и очищаем поля
langYear = 0
},
modifier = Modifier.weight(1f) //назначаем вес кнопки
) {
Text("Add") //надпись для кнопки
}
}
}

```

И добавляем еще функцию для создания списка с объектами:

```

@Composable
fun MakeList(viewModel: ItemViewModel, lazyListState: LazyListState) {
    LazyColumn(
        verticalArrangement = Arrangement.spacedBy(12.dp),
        modifier = Modifier
            .fillMaxSize()
            .background(Color.White),
        state = lazyListState //состояние списка соотносим с переданным объектом
    ) {
        items(
            items = viewModel.langListFlow.value,
            key = { lang -> lang.name },
            itemContent = { item ->
                ListRow(item)
            }
        )
    }
}

```

Остальной код не меняем. В итоге после запуска приложения должен получиться функционал как на рис. 13.

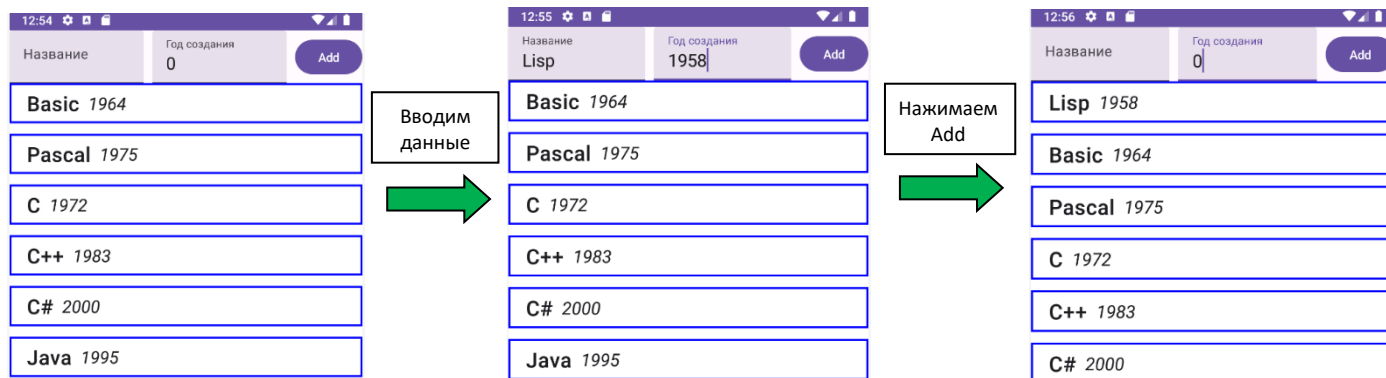


Рис. 13. Добавление нового элемента в список.

Задачи

I часть. Общее для всех задание: Добавить поле для процента популярности языка программирования (например, возле названия языка). Внести соответствующие изменения в код.

II часть. По вариантам:

1. Создать программу для заполнения списка домов нашего города. Должны быть поля для ввода названия улицы, номера дома, кол-ва квартир.
2. Создать программу для заполнения списка городов. Должны быть поля для ввода названия города, кол-ва жителей и названия области.
3. Создать программу для заполнения списка студентов вуза. Должны быть поля для ввода ФИО, номера группы, факультета.
4. Создать программу для заполнения списка преподавателей вуза. Должны быть поля для ввода ФИО, возраста, кафедры.

5. Создать программу для заполнения списка автомобилей. Должны быть поля для ввода гос. номера, объема двигателя и названия фирмы.
6. Создать программу для заполнения списка книг. Должны быть поля для ввода ФИО автора, названия книги и литературный жанр.
7. Создать программу для заполнения списка стран. Должны быть поля для ввода названия страны, ФИО главы государства и форма правления.
8. Создать программу для заполнения списка фильмов. Должны быть поля для ввода названия фильма, режиссера, жанра и кинокомпании.
9. Создать программу для заполнения списка авиарейсов. Должны быть поля для ввода номера рейса, город отправки, город прибытия, тип самолета и авиакомпания.
10. Создать программу для заполнения списка телепередач. Должны быть поля для ввода названия программы, времени показа, названия канала, ФИО ведущего.
11. Создать программу для заполнения списка оборудования. Должны быть поля для ввода инвентаризационного номера, типа оборудования, названия фирмы.

Лабораторная работа №3. Использование Image и дополнительной информации на экране

Элемент Image позволяет загружать и просматривать изображения. Можно включить такой элемент в приложение из лаб. раб. №2. Например, мы хотим вместе с названием и годом создания отображать фотографию создателя языка. Нам понадобятся файлы с изображениями создателей языков и один файл с надписью No picture (для тех, чье фото не найдется). (Можно взять с сервера – папка \Java&Android\Android\drawable, можно найти самостоятельно).

Файлы можно использовать различного формата (jpg, png и т.д.), главное, чтобы в названии не было пробелов и русских символов. Например, можно файлы назвать по имени языка (c.jpg, basic.png и т.д.). Получившиеся файлы копируем в папку *проект\app\src\main\res\drawable*. Теперь можно менять код проекта.

Сначала изменим класс ProgLanг:

```
data class ProgLanг(val name: String, val year: Int, var picture: Int = R.drawable.no_picture )
```

Мы добавили еще один параметр – для изображения – и сделали его равным по умолчанию картинке с текстом No picture. И теперь если будет добавляться объект без указания графического файла, то будет автоматом вставляться эта картинка.

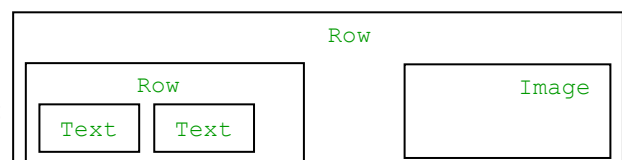
Меняем объявление нашего списка с языками программирования:

```
private var langList = mutableStateListOf(
    ProgLanг("Basic", 1964, R.drawable.basic),
    ProgLanг("Pascal", 1975, R.drawable.pascal),
    ProgLanг("C", 1972, R.drawable.c),
    ProgLanг("C++", 1983, R.drawable.cpp),
    ProgLanг("C#", 2000, R.drawable.c_sharp),
    ProgLanг("Java", 1995, R.drawable.java),
    ProgLanг("Python", 1991, R.drawable.python),
    ProgLanг("JavaScript", 1995),
    ProgLanг("Kotlin", 2011)
)
```

Теперь нужно добавить в наш визуальный список элемент для отображения картинки. Для этого функцию [ListRow](#) из лаб.раб.2 перепишем в следующем виде:

```
@Composable
fun ListRow(model: ProgLanг) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceBetween, //для правильного расположения эл-ов
        //в данном случае они будут располагаться по краям ряда
        modifier = Modifier
            .wrapContentHeight()
            .fillMaxWidth()
            .border(BorderStroke(2.dp, Color.Blue))
    ) {
        Row(verticalAlignment = Alignment.CenterVertically) { //ряд для текстовых полей
            Text(
                text = model.name,
                fontSize = 24.sp,
                fontWeight = FontWeight.SemiBold,
                modifier = Modifier.padding(start = 20.dp)
            )
            Text(
                text = model.year.toString(),
                fontSize = 20.sp,
                modifier = Modifier.padding(10.dp),
                fontStyle = FontStyle.Italic
            )
        }
        Image(//нужен import androidx.compose.foundation.Image
            painter = painterResource(id = model.picture), //указываем источник изображения
            contentDescription = "", //можно вставить описание изображения
            contentScale = ContentScale.Fit, //параметры масштабирования изображения
        )
    }
}
```

В итоге получится следующая структура расположения объектов:



Т.е. объект Image будет всегда в правой части элемента списка.

Это стало возможным из-за строки `horizontalArrangement = Arrangement.SpaceBetween` в верхнем объекте Row. Значения `horizontalArrangement` могут быть следующие:

- `Arrangement.Center`: расположение по центру
- `Arrangement.End`: расположение в конце (справа — для левосторонних языков, слева — для правосторонних языков)
- `Arrangement.Start`: расположение в начале (слева — для левосторонних языков, справа — для правосторонних языков)
- `Arrangement.SpaceAround`: компоненты равномерно распределяются по всей ширине с равномерными отступами между элементами, при этом отступы между первым и последним элементами и границами контейнера равен половине отступов между элементами
- `Arrangement.SpaceBetween`: компоненты равномерно распределяются по всей ширине с равномерными отступами между элементами, при этом первый и последний элементы прижимаются к границам контейнера
- `Arrangement.SpaceEvenly`: компоненты равномерно распределяются по всей ширине с равномерными отступами между элементами, при этом отступы между первым и последним элементами и границами контейнера равны отступам между элементами

На рис. 14 показано влияние различных параметров на расположение элементов при разных размерах экрана (там есть еще параметр `Equal Weight`, это параметр по умолчанию, если не использовать никакой другой).

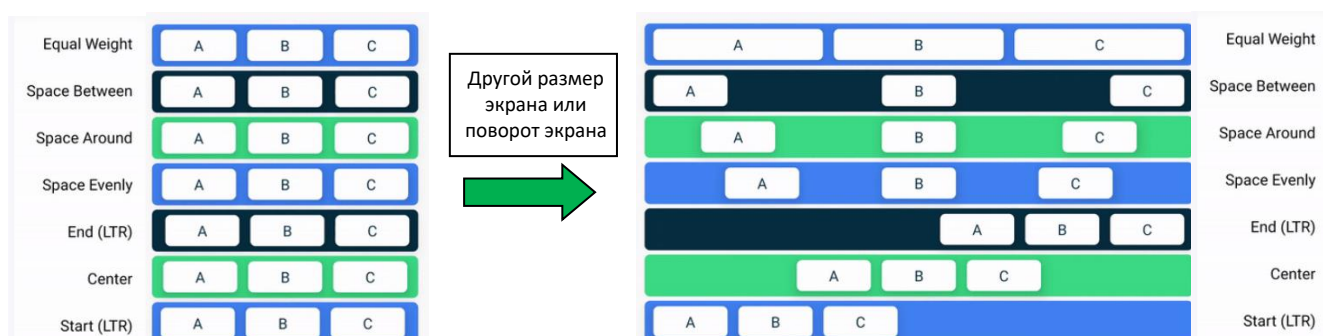


Рис. 14. Влияние параметров `horizontalArrangement` на горизонтальное размещение объектов внутри ряда.

Параметр `contentScale` для объекта Image может принимать следующие значения:

- `ContentScale.Crop`: масштабирует изображение с сохранением аспектного отношения (отношение высоты и ширины) таким образом, что ширина и высота оказываются равными или больше сторон контейнера.
- `ContentScale.FillBounds`: неравномерно масштабирует изображение для полного заполнения пространства контейнера.
- `ContentScale.FillHeight`: масштабирует изображение с сохранением аспектного отношения таким образом, что высота изображения равна высоте контейнера.
- `ContentScale.FillWidth`: масштабирует изображение с сохранением аспектного отношения таким образом, что ширина изображения равна ширине контейнера.
- `ContentScale.Fit`: масштабирует изображение с сохранением аспектного отношения (отношение высоты и ширины) таким образом, что ширина и высота оказываются равными или меньше сторон контейнера.

- `ContentScale.Inside`: масштабирует изображение с сохранением аспектного отношения (отношение высоты и ширины) таким образом, чтобы вместить изображение внутри контейнера, если ширина и(или) высота изображения больше ширины и(или) высоты контейнера.
- `ContentScale.None`: масштабирование отсутствует

В итоге при запуске программы получим вид как на рис. 15а.

Но можно заметить, что размеры рядов (по высоте) отличаются. Это произошло из-за разных изначальных размеров изображений. Чтобы это исправить, мы можем просто добавить к [параметрам объекта Image](#) строку `modifier = Modifier.size(90.dp)` (сразу после строки `contentScale = ContentScale.Fit,`). Этой строкой мы делаем размер всех изображений одинаковым (рис. 15б).

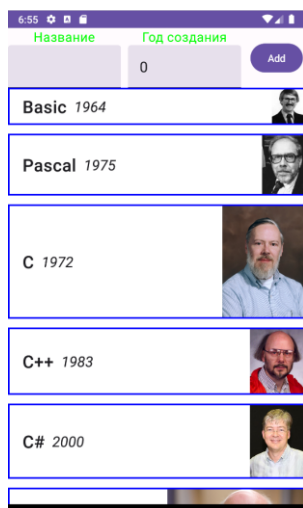


Рис. 15а. Список с изображениями.



Рис. 15б. Список с изображениями одинакового размера.



Рис. 16. Появление всплывающего сообщения при нажатии на элемент списка.

Теперь сделаем возможность вывода дополнительной информации на экран. В ОС Android есть несколько вариантов для этого.

Первый – всплывающее сообщения. Например, сделаем возможность при нажатии на элемент списка вывода более подробной информации о нужном языке.

Начнем с всплывающего сообщения, в котором выведем номер элемента списка, на который нажали.

Для начала нужно сделать обработчик нажатия на элемент списка. Немного изменим начало функции `ListRow` (новый код выделен серым):

```
fun ListRow(model: ProgLan) {
    val context = LocalContext.current //получаем текущий контекст, он нужен для создания
                                        //всплывающего сообщения
    Row(
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceBetween,
        modifier = Modifier
            .wrapContentSize()
            .fillMaxWidth()
            .border(BorderStroke(2.dp, Color.Blue))
            .combinedClickable{ //добавляем модификатор, отвечающий за обработку нажатий
                onClick = { //и прописываем действия в случае нажатия на элемент списка
                    println("item = ${model.name}") //это будет выведено в Logcat (вкладка
                                                        //внизу Android Studio)
                    // а тут делаем всплывающее сообщение, передаем ему текущий контекст, текст сообщения –
                    // названия языка, на который нажали, и время показа сообщения, после этого вызываем
                    // show() у созданного сообщения
                    Toast.makeText(context, "item = ${model.name}", Toast.LENGTH_LONG).show()
                }
            }
    )
}
```

Остальной код пока не трогаем. В результате, теперь, при нажатии на элемент списка, будет появляться всплывающее сообщение с названием языка, на который нажали (см. рис. 16).

Другой способ показать дополнительную информацию на экране – дочернее окно. Для этого в Android SDK есть класс `AlertDialog`. Дополним наш обработчик нажатия на элемент списка – сделаем вывод более подробной информации о нажатом языке. Для этого нам понадобится поместить более подробную

информацию в строковые ресурсы нашего приложения. Эти ресурсы находятся в файле strings.xml в папке res\values нашего проекта (см. рис. 17). Там уже есть одна строка с названием приложения, добавляем еще 3 строки с названиями языков, чтобы файл был примерно следующего вида:

```
<resources>
  <string name="app_name">Compose Example</string>
  <string name="Pascal">Автор - Никлаус Вирт.</string>
  <string name="C">Автор - Деннис Ритчи.</string>
  <string name="Java">Автор - Джеймс Гослинг.</string>
</resources>
```

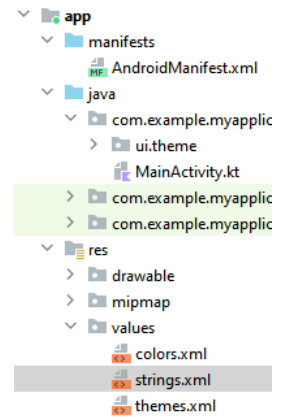


Рис. 17. Строковые ресурсы приложения.

И теперь мы сможем обращаться к содержимому нужного строкового ресурса по его имени (значение поля name). Для начала создадим еще одну функцию, перед ListRow:

```
@Composable
fun MakeAlertDialog(context: Context, dialogTitle: String, openDialog: MutableState<Boolean>) {
  //создаем переменную, в ней будет сохраняться текст, полученный из строковых ресурсов для выбранного языка
  var strValue = remember{ mutableStateOf("") } //для получения значения строки из ресурсов
  //получаем id нужной строки из ресурсов через имя в dialogTitle
  val strId = context.resources.getIdentifier(dialogTitle, "string", context.packageName)
  //секция try..catch нужна для обработки ошибки Resources.NotFoundException - отсутствие искомого ресурса
  try{ //если такой ресурс есть (т.е. его id не равен 0), то берем само значение этого ресурса
    if (strId != 0) strValue.value = context.getString(strId)
  } catch (e: Resources.NotFoundException) {
    //если произошла ошибка Resources.NotFoundException, то ничего не делаем
  }
  AlertDialog( // создаем AlertDialog
    onDismissRequest = { openDialog.value = false }, //действия при закрытии окна
    title = { Text(text = dialogTitle) }, //заголовков окна
    text = { Text(text = strValue.value, fontSize = 20.sp) }, //содержимое окна
    confirmButton = { //кнопка Ok, которая будет закрывать окно
      Button(onClick = { openDialog.value = false })
        { Text(text = "OK") }
    }
  )
}
```

Далее перепишем начало функции ListRow:

```
fun ListRow(model: ProgrLang) {
  val context = LocalContext.current
  //создаем переменную для обозначения, вызвано дочернее окно (AlertDialog) или нет
  val openDialog = remember { mutableStateOf(false) } //по умолчанию - false, т.е. окно не вызвано
  val langSelected = remember { mutableStateOf("") } // и переменная для сохранения названия языка
  if (openDialog.value) //если дочернее окно (AlertDialog) вызвано
    MakeAlertDialog(context, langSelected.value, openDialog) //то создаем его
  Row(
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceBetween,
    modifier = Modifier
      .wrapContentSize()
      .fillMaxWidth()
      .border(BorderStroke(2.dp, Color.Blue))
      .combinedClickable(
        onClick = {
          println("item = ${model.name}")
          langSelected.value = model.name //сохраняем имя языка, чтобы вставить в заголовок
          // AlertDialog
          Toast.makeText(context, "item = ${model.name}", Toast.LENGTH_LONG).show()
          openDialog.value = true //присваиваем признаку открытия дочернего окна true
        }
      )
  )
}
```

Остальной код остается без изменений.

В итоге после запуска приложения и нажатия на любой язык программирования получаем диалоговое окно с информацией о соответствующем языке (рис. 18). Если для языка не задана строка с описанием в ресурсах, то просто будет окно с пустым полем, благодаря тому, что strValue при инициализации равна "" (т.е. по умолчанию просто пустая строка, если ничего не найдем в ресурсах).

Задание

Добавить к элементам списка в решенной задаче из лаб. раб. №2 изображения и обработчик нажатия на элемент списка с показом диалогового окна с информацией (информацию разместить в ресурсах).

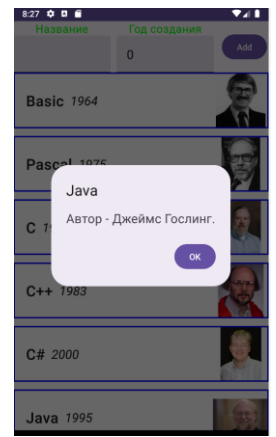


Рис. 18. Дочернее окно при нажатии на язык Java.

Лабораторная работа №4. Меню, несколько Activity (окон) в одном приложении

Удобно разбить свое приложение на несколько функциональных блоков и под каждый выделить отдельное окно (Activity). В нашем случае можно выделить блок, показывающий список, блок для ввода элементов списка и блок для показа информации при нажатии на элемент списка.

Рассмотрим добавление класса ComponentActivity, в который перенесем создание нового элемента списка, причем вызов его сделаем через меню. Сначала добавим само меню в наше приложение. Для этого можно использовать компонент `AppBar`. Само меню можно реализовать через объект `DropDownMenu`. Добавляем к нашей основной функции `setContent` в методе `onCreate` вызов новой функции, которую распишем ниже:

```
setContent {  
    val lazyListState = rememberLazyListState()  
    ComposeExampleTheme {  
        Surface(  
            modifier = Modifier.fillMaxSize(),  
            color = MaterialTheme.colorScheme.background  
        ) {  
            Column(Modifier.fillMaxSize()) {  
                MakeAppBar(viewModel, lazyListState) // вызываем новую функцию  
                //а часть с вводом данных пока комментируем  
                //MakeInputPart(viewModel, lazyListState)  
                MakeList(viewModel, lazyListState)  
            }  
        }  
    }  
}
```

И сама новая функция:

```
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun MakeAppBar(model: ItemViewModel, lazyListState: LazyListState) {  
    //создаем объект для хранения состояния меню - открыто (true) или нет (false)  
    var mDisplayMenu by remember { mutableStateOf(false) }  
    val mContext = LocalContext.current // контекст нашего приложения  
    val openDialog = remember { mutableStateOf(false) } //объект для состояния дочернего окна  
    if (openDialog.value) //если дочернее окно вызвано, то запускаем функцию для его создания  
        makeAlertDialog(context = mContext, dialogTitle = "About", openDialog = openDialog)  
    TopAppBar( //создаем верхнюю панель нашего приложения, в нем будет меню  
        title = { Text("Языки программирования") }, //заголовок в верхней панели  
        actions = { //здесь разные действия можно прописать, например, иконку для меню  
            IconButton(onClick = { mDisplayMenu = !mDisplayMenu }) { //создаем иконку  
                Icon(Icons.Default.MoreVert, null) //в виде трех вертикальных точек  
            } //в методе onClick прописано изменение объекта для хранения состояния меню  
            DropDownMenu( //создаем меню  
                expanded = mDisplayMenu, //признак, открыто оно или нет  
                onDismissRequest = { mDisplayMenu = false } //при закрытии меню устанавливаем  
                    //соответствующее значение объекту mDisplayMenu  
            ) {  
                DropDownMenuItem( //создаем пункт меню для вызова информации о программе (About)  
                    text = { Text(text = "About") }, //его текст  
                    onClick = { //и обработчик нажатия на него  
                        //всплывающее сообщение с названием пункта  
                        Toast.makeText(mContext, "About", Toast.LENGTH_SHORT).show()  
                        mDisplayMenu = !mDisplayMenu //меняем параметр, отвечающий за состояние меню,  
                        openDialog.value = true //и параметр, отвечающий за состояние дочернего окна,  
                        //в котором выводим доп. информацию  
                    }  
                )  
                //создаем второй пункт меню для вызова окна, в которое перенесли ввод нового языка  
                DropDownMenuItem(  
                    text = { Text(text = "Add lang") }, //его текст  
                    onClick = { //и обработчик нажатия на него  
                        Toast.makeText(mContext, "Add lang", Toast.LENGTH_SHORT).show()  
                        //создаем Intent - намерение, специальный объект для вызова нового окна приложения, сам класс  
                        val newAct = Intent(mContext, InputActivity::class.java) //описан ниже  
                        mContext.startActivity(newAct) //вызываем новое окно  
                        mDisplayMenu = !mDisplayMenu //и меняем признак открытия меню  
                    }  
                )  
            }  
        }  
    }  
}
```



```

    )
}

```

Вызов нового окна осуществляется в 2 этапа:

- создаем объект класса Intent (намерение), в котором указываем контекст (через `mContext = LocalContext.current`, т.е. берем контекст текущего класса) и класс созданного нового окна (`InputActivity::class.java`) – описан ниже;
- вызываем метод `startActivity` и передаем ему в качестве параметра созданное намерение `newAct`.

Для продолжения работы нам необходимо немного изменить наш базовый класс для языков программирования `ProgrLang` в файле `ItemViewModel.kt` – сделать его реализующим интерфейс `Serializable`:

```
data class ProgrLang(val name: String, val year: Int, var picture: String = R.drawable.no_picture.toString()) : Serializable
```

Это позволит передавать объекты этого класса между частями приложения.

Также нужно создать сам класс `InputActivity` (контекстное меню папки `java`, пункт `New` → `Activity` → `Gallery` → `Empty Activity`), в него переносим функцию `MakeInputPart`:

```
class InputActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ComposeExampleTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MakeInputPart() //наша функция по созданию интерфейса для ввода нового языка
                }
            }
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MakeInputPart() {
    var langName by remember { //объект для работы с текстом, для названия языка
        mutableStateOf("") //его начальное значение
    } //в функцию mutableStateOf() в качестве параметра передается отслеживаемое значение
    var langYear by remember { //объект для работы с текстом, для года создания языка
        mutableStateOf(0) //его начальное значение
    }
    Row(
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(10.dp),
    ) {
        TextField( //текстовое поле для ввода имени языка
            value = langName, //связываем текст из поля с созданным ранее объектом
            onChange = { newText -> //обработчик ввода значений в поле
                langName = newText //все изменения сохраняем в наш объект
            },
            textStyle = TextStyle( //объект для изменения стиля текста
                fontSize = 20.sp //увеличиваем шрифт
            ),
            label = { Text("Название") },
            modifier = Modifier.weight(2f)
        )
        TextField( //текстовое поле для ввода года создания языка
            value = langYear.toString(), //связываем текст из поля с созданным ранее объектом
            onChange = { newText -> //обработчик ввода значений в поле
                langYear = if (newText != "") newText.toInt() else 0 //в нужный формат
            },
            //с учетом возможной пустой строки
            textStyle = TextStyle( //объект для изменения стиля текста
                fontSize = 20.sp //увеличиваем шрифт
            ),
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
            label = { Text("Год создания") },
            modifier = Modifier.weight(2f) //назначаем вес поля
        )
        Button( //кнопка для добавления нового языка
            onClick = { //при нажатии кнопки делаем отладочный вывод
                println("added $langName $langYear")
                //создаем новый язык с введенными параметрами
                val newLang = ProgrLang(langName, langYear)
                val intent = Intent() //создаем намерение
            }
        )
    }
}
```

```

        //задаем доп.данные для намерения - наш новый объект
        intent.putExtra("newItem", newLang)
        //вставляем намерение в результат текущего окна
        setResult(RESULT_OK, intent);
        langName = "" //очищаем поля
        langYear = 0
        finish() //и закрываем текущее окно
    },
    modifier = Modifier.weight(1f)
) {
    Text("Add")
}
}
}

```

Если элементов ввода данных много, то их можно расположить вертикально, а кнопку для подтверждения ввода сделать в самом низу, так что исходим из необходимой функциональности интерфейса.

Теперь нажатие на пункт меню «Add lang» приведет к переходу в новое окно нашей программы (рис. 19).

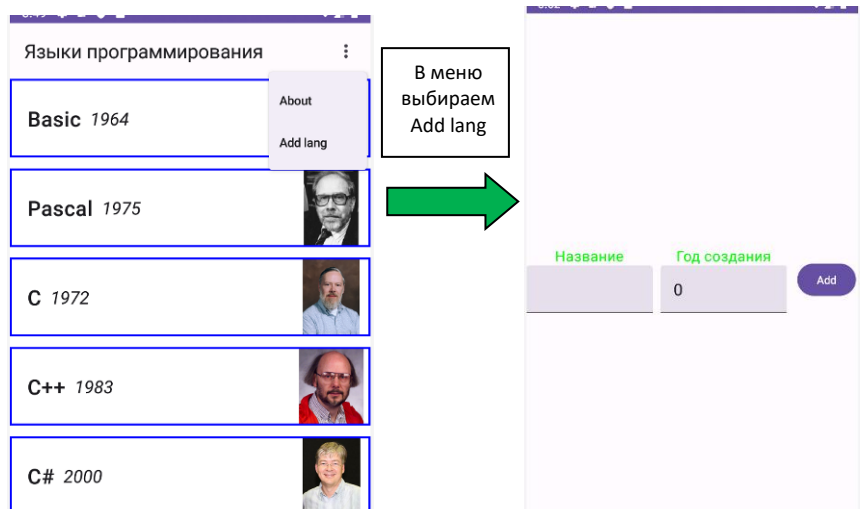


Рис. 19. Вызов нового окна.

Но если мы введем нужные данные и нажмем Add, то ничего не произойдет. Для добавления новому окну нужного функционала (т.е. чтобы по нажатию Add происходил возврат в первое окно с добавлением новых данных в список) нужно внести изменения в функцию `MakeAppBar` в классе `MainActivity`:

после строки

```
val openDialog = remember { mutableStateOf(false) } //объект для состояния дочернего окна
```

вставляем

```

val scope = rememberCoroutineScope() //объект для прокручивания списка при вставке нового эл-та
val startForResult = //переменная-объект класса ManagedActivityResultLauncher,
//ей присваиваем результат вызова метода rememberLauncherForActivityResult
rememberLauncherForActivityResult(ActivityResultContracts.StartActivityForResult()) {
    result ->
    //внутри метода смотрим результат работы запущенного активити - если закрытие с кодом RESULT_OK
    if (result.resultCode == Activity.RESULT_OK) { //то берем объект из его данных
        val newLang = result.data?.getSerializableExtra("newItem") as ProgrLang //как язык
        println("new lang name = ${newLang.name}") //вывод для отладки
        model.addLangToHead(newLang)
        scope.launch { //прокручиваем список, чтобы был виден добавленный элемент
            lazyListState.scrollToItem(0)
        }
    }
}

```

и в блоке создания меню меняем обработчик нажатия на пункт меню Add lang:

```

DropDownMenuItem(
    text = { Text(text = "Add lang") },
    onClick = {
        Toast.makeText(mContext, "Add lang", Toast.LENGTH_SHORT).show()
        val newAct = Intent(mContext, InputActivity::class.java)
        startForResult.launch(newAct) //запускаем новое окно и ждем от него данные
        mDisplayMenu = !mDisplayMenu
    }
)

```

Результат нашей деятельности можно увидеть на рис. 20 (если начинать, как на рис. 19).

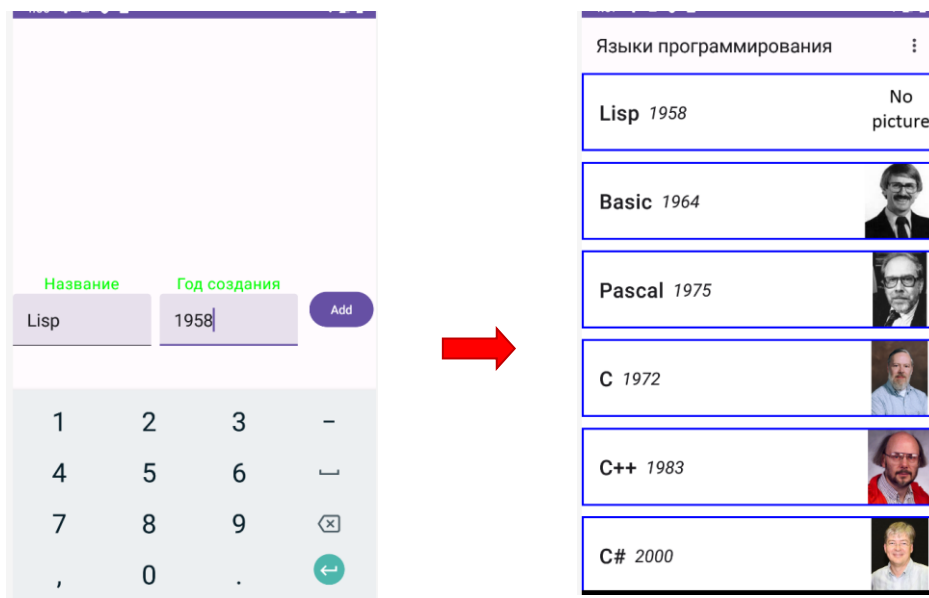


Рис. 20. Возвращение значения из нового окна.

Задание

Добавить к задаче из лаб.раб.№3 меню с обязательными пунктами «О программе» и «Добавить новый элемент» и вынести добавление нового элемента списка в отдельное активити с возвращением результата. В пункт «О программе» обязательно добавить ФИО автора лабораторной.

Лабораторная работа №5. Сохранение состояний. Контекстное меню и работа с внешними ресурсами

Если после добавления нового элемента в список повернуть устройство или выйти из приложения (нажав кнопку Home), а потом заново загрузить его, то все наши изменения в списке элементов сбросятся, т.к. они не сохранены в приложении (см. рис. 21).

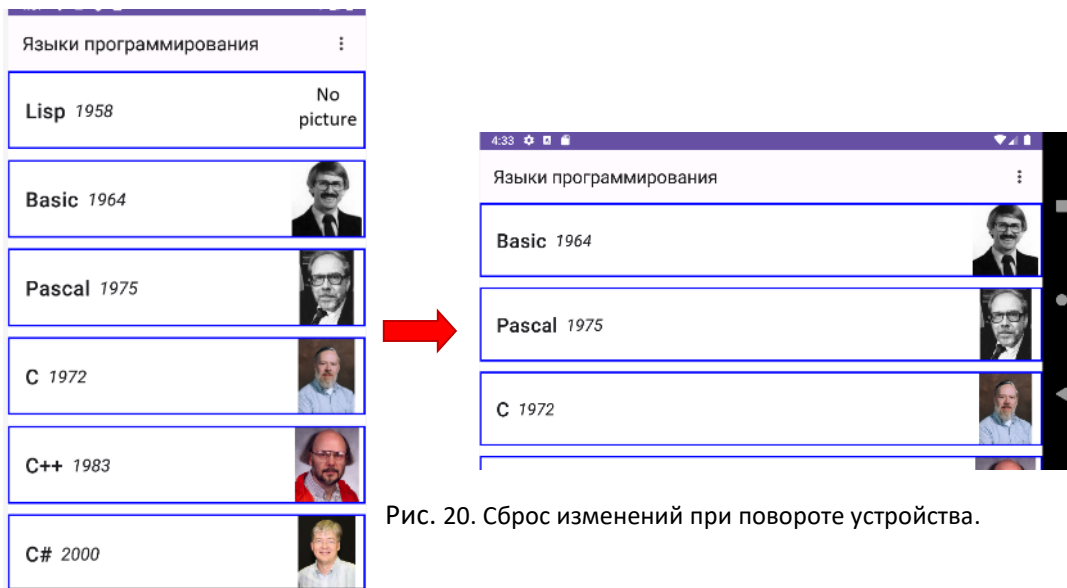


Рис. 20. Сброс изменений при повороте устройства.

Попробуем устранить этот недостаток. Используем специальный класс `Bundle`. В объект этого класса можно передавать различные значения для временного хранения в период каких-то изменений в интерфейсе или жизненном цикле приложения.

Для этого добавляем в класс `MainActivity` метод `onSaveInstanceState`, который будет вызываться при необходимости сохранить данные в переходах между состояниями в жизненном цикле приложения (например, при изменении ориентации экрана). Параметром этого метода является объект класса `Bundle`. В него можно поместить массив `langList`. Но есть только миг между прошлым и будущим один нюанс – в `Bundle` нельзя поместить объект класса `SnapshotStateList` (именно представителем такого класса является наш массив с языками программирования, который мы создали методом `mutableStateListOf`), но можно использовать обычный `ArrayList`.

Поэтому мы создадим временный массив типа `ArrayList`, перенесем в него наши объекты из `langList` и именно его и сохраним в `Bundle`.

Метод `onSaveInstanceState` в наших целях можно оформить следующим образом (вставляем его после метода `onCreate` в конце класса `MainActivity`):

```
override fun onSaveInstanceState(outState: Bundle) {
    Toast.makeText(this, "saved", Toast.LENGTH_SHORT).show() //сообщение для отслеживания
    var tempLangArray = ArrayList<ProgrLang>() //временный ArrayList для сохранения данных
    viewModel.langListFlow.value.forEach { //переносим данные из нашего основного массива
        tempLangArray.add(it)
    }
    outState.putSerializable("langs", tempLangArray) //помещаем созданный массив в хранилище
    //и даем ему метку langs, по ней потом его и найдем
    super.onSaveInstanceState(outState) //вызов метода базового класса
}
```

Метод `putSerializable` объекта `outState` позволяет сохранить в памяти любой объект, реализующий интерфейс `Serializable` (это интерфейс-маркер, информирующий о возможности сериализации объекта, т.е. его сохранения, очень многие базовые классы Kotlin и Android реализуют этот интерфейс, в том числе и `ArrayList`). Первый параметр этого метода – метка, по которой можно будет восстановить объект, второй параметр – сам сохраняемый объект (наш массив типа `ArrayList`).

И добавим работу с объектом класса `Bundle` в методе `onCreate(savedInstanceState: Bundle?)` – параметр `savedInstanceState` дает нам доступ к объекту класса `Bundle`, в который мы сохранили наш массив. Для этого перепишем начало метода :

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //если в хранилище есть наш массив с языками программирования
    if (savedInstanceState!=null && savedInstanceState.containsKey("langs")) {
        //то в нашу модель переписываем эл-ты из savedInstanceState
        val tempLangArray = savedInstanceState.getSerializable("langs") as ArrayList<ProgrLang>
        viewModel.clearList()
        tempLangArray.forEach {
            viewModel.addLangToEnd(it)
        }
        Toast.makeText(this, "From saved", Toast.LENGTH_SHORT).show()
    } else Toast.makeText(this, "From create", Toast.LENGTH_SHORT).show() //иначе просто сообщение
}

```

Далее код пока не меняем. И теперь, если добавить новый элемент в наш список, а потом повернуть экран, то список останется прежним (рис. 21).

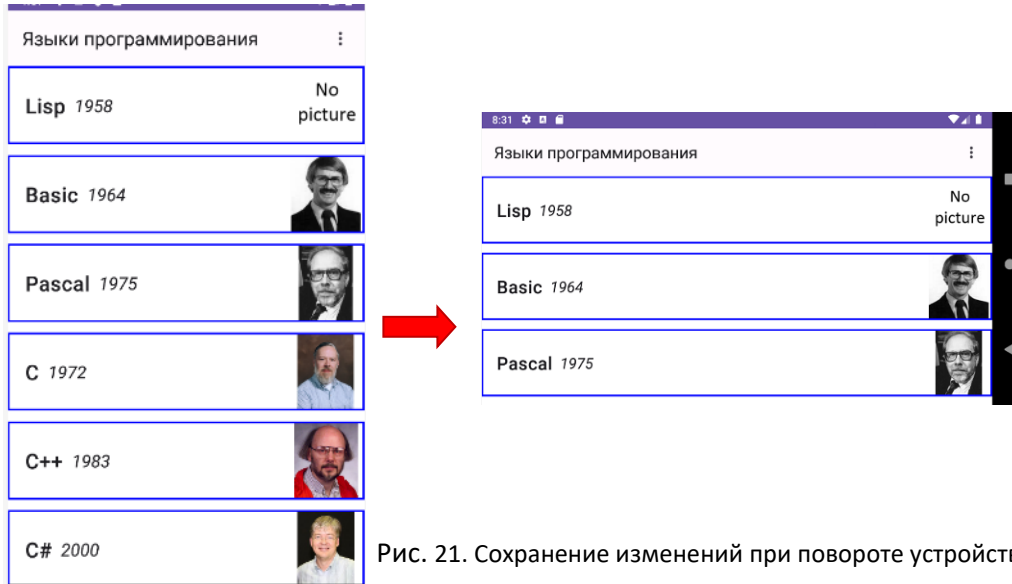


Рис. 21. Сохранение изменений при повороте устройства.

Все добавляемые нами новые элементы списка связываются с картинкой No picture. Но, допустим, мы хотим поменять всё в своей жизни эту картинку или картинку другого элемента на новую картинку. Сделать это можно через контекстное меню объекта. Для этого сначала нужно, чтобы в эмуляторе или в реальном устройстве был тот файл, который мы хотим использовать. С реальным устройством все просто (копируем на него), с эмулятором нужны следующие действия:

- в параметрах нашего созданного виртуального устройства (см. лаб. раб.№1, [рис. 4](#)) в пункте SD card в поле должен быть указан размер виртуальной карты памяти;
- после запуска виртуального устройства можно передавать ему файлы: в Android Studio пункт меню View > Tool Windows > Device File Explorer. В левой части экрана появится файловый обозреватель виртуального устройства (рис. 22).
- выбираем каталог sdcard, вызываем его контекстное меню и выбираем пункт Upload... Откроется диалоговое окно загрузки файла. Находим нужный файл (например, face.png, есть на сервере, нужно скопировать в папку проекта). Жмем Ок и файл появится в виртуальном устройстве.

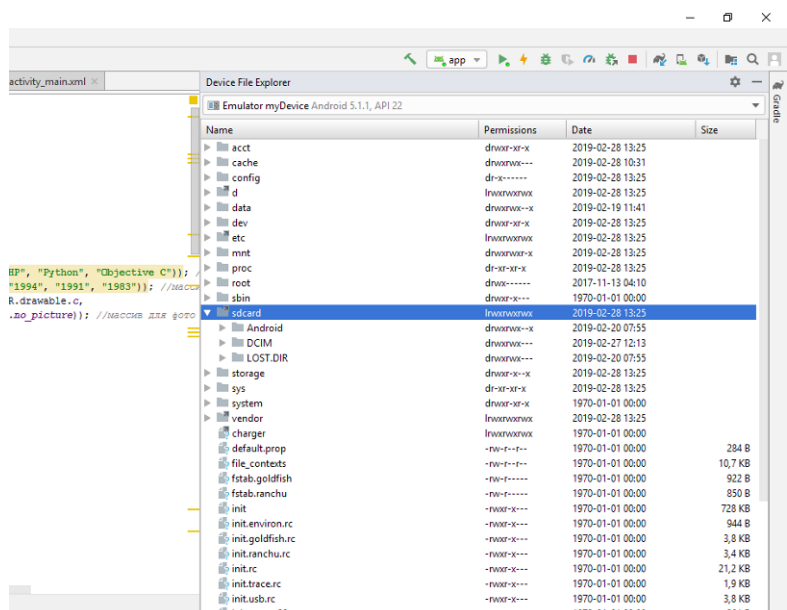


Рис. 22. Файловый обозреватель виртуального устройства.

Нужно разрешить нашему приложению выполнять чтение с хранилища телефона. Для этого в файл AndroidManifest.xml перед тегом `<application>` добавляем строку:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Далее нам понадобится внести ряд изменений в уже существующий код. Перепишем метод `MakeList` в следующем виде:

```
@Composable
fun MakeList(viewModel: ItemViewModel, lazyListState: LazyListState) {
    //создаем объект для получения данных о состоянии списка из модели
    val langListState = viewModel.langListFlow.collectAsState()
    LazyColumn(
        verticalArrangement = Arrangement.spacedBy(12.dp),
        modifier = Modifier
            .fillMaxSize()
            .background(Color.White),
        state = lazyListState
    ) {
        items(
            items = viewModel.langListFlow.value,
            key = { lang -> lang.name },
            itemContent = { item ->
                //меняем параметры вызова метода ListRow
                ListRow(item, langListState, viewModel)
            }
        )
    }
}
```

Немного изменим класс, представляющий язык программирования:

```
data class ProgrLang(val name: String, val year: Int,
    var picture: String = R.drawable.no_picture.toString()): Serializable
```

Мы изменили тип поля `picture`, теперь оно строковое, чтобы можно было хранить путь к изображению, которое будем загружать из внешних ресурсов. И поэтому теперь меняем объявление массива `langList` в классе `ItemViewModel`:

```
private var langList = mutableStateListOf(
    ProgrLang("Basic", 1964, R.drawable.basic.toString()),
    ProgrLang("Pascal", 1975, R.drawable.pascal.toString()),
    ProgrLang("C", 1972, R.drawable.c.toString()),
    ProgrLang("C++", 1983, R.drawable.cpp.toString()),
    ProgrLang("C#", 2000, R.drawable.c_sharp.toString()),
    ProgrLang("Java", 1995, R.drawable.java.toString()),
    ProgrLang("Python", 1991, R.drawable.python.toString()),
    ProgrLang("JavaScript", 1995),
    ProgrLang("Kotlin", 2011)
)
```

В класс `ItemViewModel` добавляем метод для изменения изображения в списке:

```
fun changeImage(index: Int, value: String) {
    langList[index] = langList[index].copy(picture = value)
}
```

Т.е., при вызове этого метода у нужного элемента списка будет меняться поле `picture`.

Для дальнейшего изменения кода нам нужно добавить строку

```
implementation "io.coil-kt:coil-compose:1.3.2"
```

в файл `build.gradle` в секцию `dependencies`. После этого нужно нажать кнопку `Sync now` (она появляется при внесении изменений в этот файл).

Далее вносим изменения в метод `ListRow` в `MainActivity`, чтобы привести его к следующему виду:

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun ListRow(model: ProgrLang, langListState: State<List<ProgrLang>>, viewModel: ItemViewModel) {
    val context = LocalContext.current //получаем текущий контекст
    val openDialog = remember { mutableStateOf(false) }
    var langSelected = remember { mutableStateOf("") } //переменная для сохранения названия языка
    if (openDialog.value) //если дочернее окно (AlertDialog) вызвано
        MakeAlertDialog(context, langSelected.value, openDialog) //то создаем его
    //создаем объект для обозначения появления контекстного меню
    var mDisplayMenu by remember { mutableStateOf(false) }
```

```

//объект для запуска деятельности выбора нового изображения
//запускается метод rememberLauncherForActivityResult с контрактом на получение данных
val launcher =
rememberLauncherForActivityResult(ActivityResultContracts.StartActivityForResult()) { res ->
//res - результат выполнения метода
    if (res.data?.data != null) { //если была выбрана новая картинка
        println("image uri = ${res.data?.data}") //отладочный вывод (будет в разделе Run внизу IDE)
        val imgURI = res.data?.data //берем адрес картинки
        val index = langListState.value.indexOf(model) //получаем индекс текущего объекта в списке
        viewModel.changeImage(index, imgURI.toString()) //и меняем картинку для нужного языка
    }
}
Row(
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceBetween, //для правильного расположения эл-ов
    modifier = Modifier
        .wrapContentSize()
        .fillMaxWidth()
        .border(BorderStroke(2.dp, Color.Blue))
        .combinedClickable(
            onClick = { //и прописываем действия в случае нажатия на элемент списка
                println("item = ${model.name}") //это будет выведено в Logcat
                langSelected.value = model.name //сохраняем имя языка, чтобы вставить в заголовок
                Toast
                    .makeText(context, "item = ${model.name}", Toast.LENGTH_LONG)
                    .show()
                openDialog.value = true //присваиваем признаку открытия дочернего окна true
            },
            //обработчик долгого нажатия на эл-нт списка для вызова контекстного меню
            onLongClick = { mDisplayMenu = true } //меняем значение объекта для меню
        )
) {
    Row(verticalAlignment = Alignment.CenterVertically) { //ряд для текстовых полей
        Text(
            text = model.name,
            fontSize = 24.sp,
            fontWeight = FontWeight.SemiBold,
            modifier = Modifier.padding(start = 20.dp)
        )
        Text(
            text = model.year.toString(),
            fontSize = 20.sp,
            modifier = Modifier.padding(10.dp),
            fontStyle = FontStyle.Italic
        )
    }
    DropdownMenu{//создаем контекстное меню
        expanded = mDisplayMenu, //связываем его св-во, отвечающее за показ меню, с объектом
        onDismissRequest = { mDisplayMenu = false }
    } {
        DropdownMenuItem( //вставляем нужный пункт меню
            text = { Text(text = "Поменять картинку", fontSize = 20.sp) },
            onClick = { //обрабатываем нажатие на него
                mDisplayMenu = !mDisplayMenu //меняем объект, отвечающий за открытие меню
                //получаем разрешение на чтение внешних ресурсов
                val permission: String = Manifest.permission.READ_EXTERNAL_STORAGE
                val grant = ContextCompat.checkSelfPermission(context, permission)
                if (grant != PackageManager.PERMISSION_GRANTED) {
                    val permission_list = arrayOfNulls<String>(1)
                    permission_list[0] = permission
                    ActivityCompat.requestPermissions(context as Activity, permission_list, 1)
                }
                //создаем намерение на получение внешнего объекта в виде картинки
                val intent = Intent(Intent.ACTION_OPEN_DOCUMENT,
                    MediaStore.Images.Media.EXTERNAL_CONTENT_URI).apply {
                    addCategory(Intent.CATEGORY_OPENABLE) }
                launcher.launch(intent) //стартуем объект для получения картинки
            }
        )
    }
}
Image( //немного меняем параметры объекта для показа изображения
    //нужен import coil.compose.rememberImagePainter
    //смотрим: если картинка взята из внутренних ресурсов, т.е. ф-ия pictureIsInt вернет true,
    //то painter = painterResource(model.picture.toInt()),
    //иначе картинка из внешних ресурсов и painter = rememberImagePainter(model.picture)
    painter = if (pictureIsInt(model.picture)) painterResource(model.picture.toInt())

```

```

        else rememberImagePainter(model.picture),
        contentDescription = "", //можно вставить описание изображения
        contentScale = ContentScale.Fit, //параметры масштабирования изображения
        modifier = Modifier.size(90.dp)
    )
}

fun pictureIsInt(picture: String): Boolean{ //ф-ия для проверки источника изображения
// переменной data присваиваем результат блока try ... catch
    var data = try { //попробуем перевести строку с ресурсом картинки в число, т.к. внутренние
        picture.toInt() //ресурсы приложения хранятся в виде числового id
    } catch (e:NumberFormatException){ //если строка не переводится в число, то значит это
        null //изображение из внешних ресурсов и присваиваем null
    } //в результате data будет равна либо picture.toInt(), либо null
    return data!=null //результат ф-ии зависит от значения переменной data
}

```

После всех этих долгих мучений изменений при длинном нажатии на элемент списка будет появляться контекстное меню. В итоге процесс изменения картинки для нужного элемента (например, для языка C#) должен выглядеть как на рис. 23.

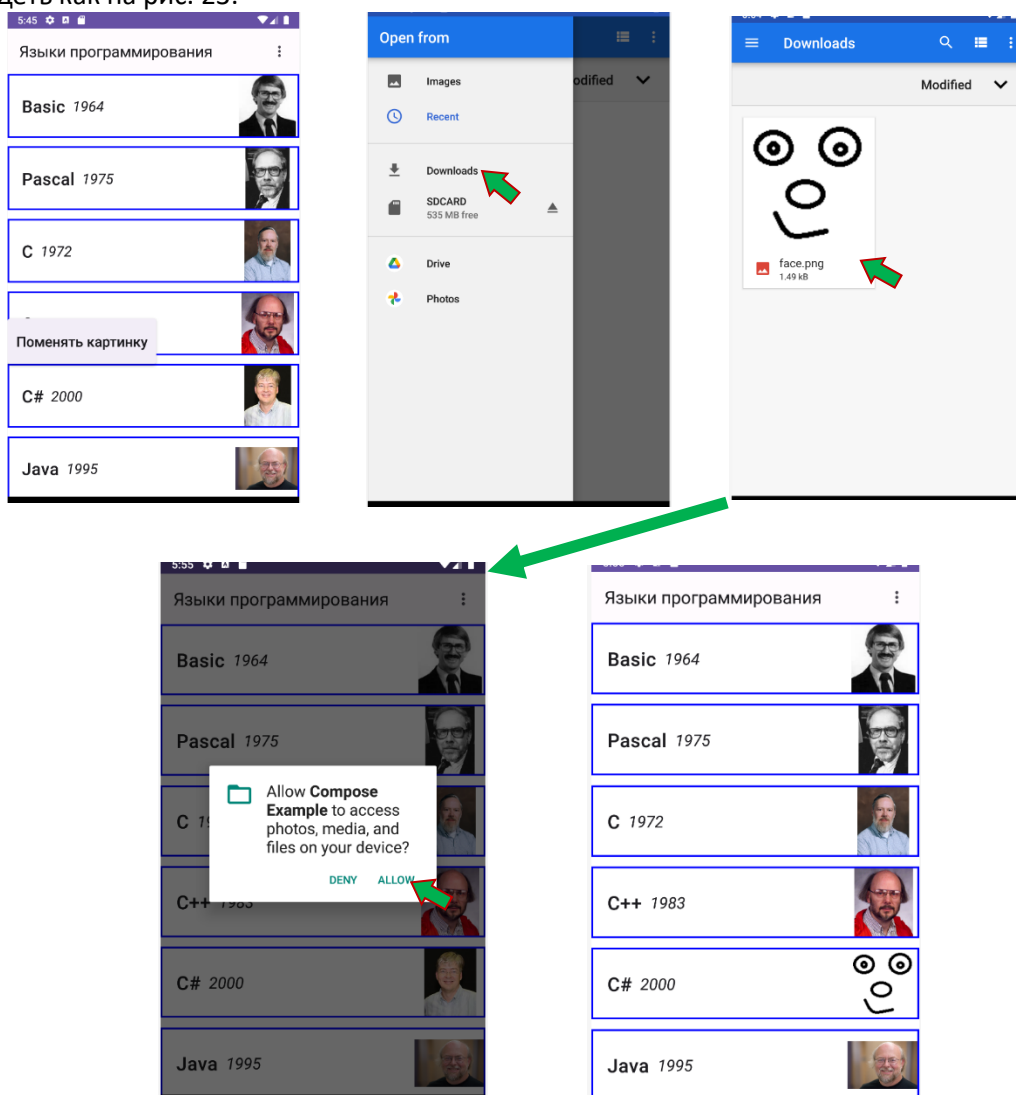


Рис. 23. Изменение картинки в элементе списка.

Задание

Добавить к своей программе контекстное меню. Реализовать как минимум пункт с изменением картинки в выбранном элементе списка, дополнительно можно попробовать реализовать удаление элемента из списка.

Лабораторная работа №6. Работа с БД

Наше приложение не будет полноценным, если не сможет сохранять данные между запусками (например, сейчас, если мы запустим программу и добавим новый язык, потом выйдем из программы, удалим ее из оперативной памяти, потом опять запустим, то новые данные не сохранятся). Для сохранения данных между запусками программы удобно использовать базу данных.

Стандартной встроенной СУБД в Android является SQLite. С помощью SQLite можно создавать для своего приложения независимые реляционные базы данных для хранения и управления сложными данными. Android хранит базы данных в каталоге `/data/data/<имя_вашего_пакета>/databases` на устройстве (или эмуляторе). По умолчанию все базы данных приватные, доступ к ним могут получить только те приложения, которые их создали.

В Android SDK уже есть готовый класс `SQLiteOpenHelper`, от которого можно наследоваться для создания класса для работы с БД. Вот пример такого класса, который будет представлять БД для хранения нашего списка языков программирования со всеми данными (в том числе и графическими):

```
class LangsDbHelper (context: Context) : //наш класс для работы с БД, наследуется от
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION){ //стандартного класса
    companion object{ // тут прописываем переменные для БД
        private val DATABASE_NAME = "LANGS" //имя БД
        private val DATABASE_VERSION = 1 // версия
        val TABLE_NAME = "langs_table" // имя таблицы
        val ID_COL = "id" // переменная для поля id
        val NAME_COL = "lang_name" // переменная для поля lang_name
        val YEAR_COL = "year" // переменная для поля year
        val PICTURE_COL = "picture" // переменная для поля picture
    }

    override fun onCreate(db: SQLiteDatabase) { //метод для создания таблицы через SQL-запрос
        val query = ("CREATE TABLE " + TABLE_NAME + " (" //конструируем запрос через
            + ID_COL + " INTEGER PRIMARY KEY autoincrement, " + //создаем выше
            NAME_COL + " TEXT," + //переменные
            YEAR_COL + " INTEGER," + PICTURE_COL + " TEXT" + ")")
        db.execSQL(query) // выполняем SQL-запрос
    }


    override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) { //метод для обновления БД
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME)
        onCreate(db)
    }

    fun getCurcor(): Cursor? { // метод для получения всех записей таблицы БД в виде курсора
        val db = this.readableDatabase // получаем ссылку на БД только для чтения
        return db.rawQuery("SELECT * FROM " + TABLE_NAME, null) //возвращаем курсор в виде
        //результата выборки всех записей из нашей таблицы
    }

    fun isEmpty(): Boolean { //метод для проверки БД на отсутствие записей
        val cursor = getCurcor() //получаем курсор таблицы БД с записями
        return !cursor!!.moveToFirst() //и возвращаем результат перехода к первой записи,
    } //инвертируя его, т.е. если нет записей, cursor!!.moveToFirst() вернет false, отрицание его
        //даст true

    fun printDB(){ //метод для печати БД в консоль
        val cursor = getCurcor() //получаем курсор БД
        if (!isEmpty()) { //если БД не пустая
            cursor!!.moveToFirst() //переходим к первой записи
            val nameColIndex = cursor.getColumnIndex(NAME_COL) //получаем индексы для колонок
            val yearColIndex = cursor.getColumnIndex(YEAR_COL) //с нужными данными
            val pictureColIndex = cursor.getColumnIndex(PICTURE_COL)
            do { //цикл по всем записям
                print("${cursor.getString(nameColIndex)} ") //печатаем данные поля с именем
                print("${cursor.getString(yearColIndex)} ") //поля с годом
                println("${cursor.getString(pictureColIndex)} ") //поля с картинкой
            } while (cursor.moveToNext()) //пока есть записи
        } else println("DB is empty") //иначе печатаем, что БД пустая
    }

    fun addArrayToDB(progLangs: ArrayList<ProgrLang>){ //метод для добавления целого массива в БД
        progLangs.forEach { //цикл по всем элементам массива
            addLang(it) //добавляем элемент массива в БД
        }
    }
}
```



```

fun addLang(lang: ProgrLang){ // метод для добавления языка в БД
    val values = ContentValues() // объект для создания значений, которые вставим в БД
    values.put(NAME_COL, lang.name) // добавляем значения в виде пары ключ-значение
    values.put(YEAR_COL, lang.year)
    values.put(PICTURE_COL, lang.picture)
    val db = this.writableDatabase //получаем ссылку для записи в БД
    db.insert(TABLE_NAME, null, values) // вставляем все значения в БД в нашу таблицу
    db.close() // закрываем БД (для записи)
}

fun changeImgForLang(name: String, img: String){ // метод для изменения картинки для языка
    val db = this.writableDatabase //получаем ссылку для записи в БД
    val values = ContentValues() // объект для изменения записи
    values.put(PICTURE_COL, img) // вставляем новую картинку
    //и делаем запрос в БД на изменение поля с нужным названием в нашей таблице
    db.update(TABLE_NAME, values, NAME_COL+" = '$name'", null)
    db.close() // закрываем БД (для записи)
}

fun getLangsArray(): ArrayList<ProgrLang>{ // метод для получения данных из таблицы в виде
//массива
    var progsArray = ArrayList<ProgrLang>() //массив, в который запишем данные
    val cursor = getCurcor() //получаем курсор таблицы БД
    if (!isEmpty()) { //если БД не пустая
        cursor!!.moveToFirst() //переходим к первой записи
        val nameColIndex = cursor.getColumnIndex(NAME_COL) //получаем индексы для колонок
        val yearColIndex = cursor.getColumnIndex(YEAR_COL) //с нужными данными
        val pictureColIndex = cursor.getColumnIndex(PICTURE_COL)
        do { //цикл по всем записям
            val name = cursor.getString(nameColIndex) //получаем данные полей
            val year = cursor.getString(yearColIndex).toInt() //и записываем их в переменные
            val picture = cursor.getString(pictureColIndex)
            progsArray.add(ProgrLang(name, year, picture)) //и создаем объект с этими данными
        } while (cursor.moveToNext()) //пока есть записи
    } else println("DB is empty") //иначе пишем, что БД пустая
    return progsArray //возвращаем созданный массив
}

```

Теперь мы можем создать объект этого класса в нашем основном классе. Для этого метод `onCreate` приводим к следующему виду:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val dbHelper = LangsDbHelper(this) //создаем объект класса LangsDbHelper
    if (savedInstanceState != null && savedInstanceState.containsKey("langs")) {
        val tempLangArray = savedInstanceState.getSerializable("langs") as ArrayList<ProgrLang>
        viewModel.clearList()
        tempLangArray.forEach {
            viewModel.addLangToEnd(it)
        }
        Toast.makeText(this, "From saved", Toast.LENGTH_SHORT).show()
    } else {
        Toast.makeText(this, "From create", Toast.LENGTH_SHORT).show()
        if (dbHelper!!.isEmpty()) { //если БД пустая
            println("DB is empty")
            var tempLangArray = ArrayList<ProgrLang>() //временный ArrayList для сохранения данных
            viewModel.langListFlow.value.forEach { //переносим данные из нашего основного массива
                tempLangArray.add(it)
            }
            dbHelper!!.addArrayToDB(tempLangArray) //заносим в БД наш массив
            dbHelper!!.printDB() //и выводим в консоль для проверки
        } else { //иначе, если в БД есть записи
            println("DB has records")
            dbHelper!!.printDB() //выводим записи в консоль для проверки
            val tempLangArray = dbHelper!!.getLangsArray() //берем записи из БД в виде массива
            viewModel.clearList() //очищаем нашу модель данных
            tempLangArray.forEach { //и в цикле по массиву переносим данные в нашу модель
                viewModel.addLangToEnd(it)
            }
        }
    }
}

setContent {
    val lazyListState = rememberLazyListState()
    ComposeExampleTheme {
        Surface(

```



```

        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        Column(Modifier.fillMaxSize()) {
            // добавляем в вызовы ф-ии объект нашей БД
            MakeAppBar(viewModel, lazyListState, dbHelper!!)
            MakeList(viewModel, lazyListState, dbHelper!!)
        }
    }
}

```

И вносим немного изменений в вызываемые ф-ии. Начало ф-ии MakeAppBar переписываем в следующем виде:

```

fun MakeAppBar(model: ItemViewModel, lazyListState: LazyListState, dbHelper: LangsDbHelper) {
    var mDisplayMenu by remember { mutableStateOf(false) }
    val mContext = LocalContext.current // контекст нашего приложения
    val openFileDialog = remember { mutableStateOf(false) } //объект для состояния дочернего окна

    val scope = rememberCoroutineScope()
    val startForResult = //переменная-объект класса ManagedActivityResultLauncher,
    //ей присваиваем результат вызова метода rememberLauncherForActivityResult
    rememberLauncherForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        result ->
            if (result.resultCode == Activity.RESULT_OK) { //то берем объект из его данных
                val newLang = result.data?.getSerializableExtra("newItem") as ProgrLang
                println("new lang name = ${newLang.name}") //вывод для отладки
                model.addLangToHead(newLang)
                dbHelper.addLang(newLang) //добавляем новый язык в БД
                scope.launch { //прокручиваем список, чтобы был виден добавленный элемент
                    lazyListState.scrollToItem(0)
                }
            }
    }
}

```

Остальной код в ф-ии MakeAppBar оставляем прежним.

Ф-ию MakeList переписываем в следующем виде:

```

@Composable
fun MakeList(viewModel: ItemViewModel, lazyListState: LazyListState, dbHelper: LangsDbHelper) {
    val langListState = viewModel.langListFlow.collectAsState()
    LazyColumn(
        verticalArrangement = Arrangement.spacedBy(12.dp),
        modifier = Modifier
            .fillMaxSize()
            .background(Color.White),
        state = lazyListState
    ) {
        items(
            items = viewModel.langListFlow.value,
            key = { lang -> lang.name },
            itemContent = { item ->
                ListRow(item, langListState, viewModel, dbHelper) //добавляем параметр dbHelper
            }
        )
    }
}

```

И начало ф-ии ListRow делаем следующим:

```

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun ListRow(model: ProgrLang, langListState: State<List<ProgrLang>>, viewModel: ItemViewModel,
    dbHelper: LangsDbHelper) {
    val context = LocalContext.current
    val openFileDialog = remember { mutableStateOf(false) }
    var langSelected = remember { mutableStateOf("") }
    if (openDialog.value)
        MakeAlertDialog(context, langSelected.value, openFileDialog)
    var mDisplayMenu by remember { mutableStateOf(false) }
    val launcher =
        rememberLauncherForActivityResult(ActivityResultContracts.StartActivityForResult()) { res ->
            if (res.data?.data != null) {
                println("image uri = ${res.data?.data}")
                val imgURI = res.data?.data
                val index = langListState.value.indexOf(model)
                viewModel.changeImage(index, imgURI.toString())
                dbHelper!!.changeImgForLang(model.name, imgURI.toString()) //меняем картинку в БД
            }
        }
}

```

Теперь, если мы добавим ~~соль, перец по вкусу~~ новые элементы в список, поменяем картинки авторов, выйдем из программы, удалим ее из оперативной памяти или перезагрузим эмулятор/устройство – все изменения в данных сохранятся.

Задание

Добавить в своей программе работу с БД (сохранение и загрузка всех данных). Проверить сохранность изменений при удалении программы из памяти, перезагрузке устройства или эмулятора.

Лабораторная работа №7. Боковая панель навигации. Работа с графикой в Android. Локализация текстовых ресурсов

Кроме обычного меню, рассмотренного в лаб.раб.№4, в Android существует еще одна возможность навигации – боковое выезжающее меню, в левой части экрана. Реализовать его можно с помощью класса `ModalNavigationDrawer`.

Добавим к нашей программе новую активность (`DrawingActivity`), в которой можно будет рисовать графические примитивы по нашему выбору – окружность, квадрат или картинку из файла. Переход к этой активности сделаем как раз из пункта, реализованного через `ModalNavigationDrawer`. Для этого меняем код функции `MakeAppBar`, после строки `if (openDialog.value)` (новый код помечен комментариями):

```
if (openDialog.value)
    MakeAlertDialog(context = mContext, dialogTitle = "About", openDialog = openDialog)
//добавляем объект для хранения состояния бокового меню
val drawerStateObj = rememberDrawerState(initialValue = DrawerValue.Closed)
TopAppBar(
    title = { Text("Языки программирования") },
    actions = {
        IconButton(onClick = { mDisplayMenu = !mDisplayMenu }) {
            Icon(Icons.Default.MoreVert, null)
        }
        DropdownMenu(
            expanded = mDisplayMenu,
            onDismissRequest = { mDisplayMenu = false }
        ) {
            DropdownMenuItem(
                text = { Text(text = "About") },
                onClick = {
                    Toast.makeText(mContext, "About", Toast.LENGTH_SHORT).show()
                    mDisplayMenu = !mDisplayMenu
                    openDialog.value = true
                }
            )
            DropdownMenuItem(
                text = { Text(text = "Add lang") },
                onClick = {
                    Toast.makeText(mContext, "Add lang", Toast.LENGTH_SHORT).show()
                    val newAct = Intent(mContext, InputActivity::class.java)
                    startForResult.launch(newAct)
                    mDisplayMenu = !mDisplayMenu
                }
            )
        }
    }
),
navigationIcon = { //описываем левую кнопку с навигацией (три горизонтальных полосы)
    IconButton( //кнопка с иконкой
        onClick = { //при нажатии на нее будет раскрываться или закрываться меню
            scope.launch {
                if (drawerStateObj.isClosed) drawerStateObj.open() //для открытия
                else drawerStateObj.close() //для закрытия
            }
        }
    ),
) {
    Icon( //для самой иконки
        Icons.Rounded.Menu, //берем изображение из системных ресурсов
        contentDescription = "" //можно добавить описание
    )
}
)
ModalNavigationDrawer( //это само боковое левое меню
    drawerState = drawerStateObj, //параметр, отвечающий за раскрытие меню, связываем с нашим объектом
    drawerContent = { //содержимое меню
        ModalDrawerSheet { //лист с меню
            Spacer(Modifier.height(12.dp)) //отступ
            NavigationDrawerItem( //пункт меню
                icon = { Icon(Icons.Default.Star, contentDescription = null) }, //иконка для него
                label = { Text("Drawing") }, //текст для него
                selected = false, //выбран или нет (актуально, когда несколько эл-ов)
                onClick = { //обработчик нажатия
                    scope.launch { drawerStateObj.close() } //закрываем меню
                    val newAct = Intent(mContext, DrawingActivity::class.java) //создаем намерение
```

```

        mContext.startActivity(newAct) //и запускаем новое активити (описано ниже)
    },
    modifier = Modifier.padding(NavigationDrawerItemDefaults.ItemPadding)
)
}
},
content = { //а здесь содержимое нашего приложения, сюда переносим вызов метода MakeList
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        MakeList(viewModel = model, lazyListState, dbHelper)
    }
}
)

```

Т.к. мы перенесли вызов функции MakeList внутрь ModalNavigationDrawer, то удаляем ее вызов в блоке setContent, он теперь будет таким:

```

setContent {
    val lazyListState = rememberLazyListState()
    ComposeExampleTheme {
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            Column(Modifier.fillMaxSize()) {
                MakeAppBar(viewModel, lazyListState, dbHelper!!) //только этот вызов оставляем
            }
        }
    }
}

```

Теперь нужно создать сам класс DrawingActivity, в котором и будет происходить работа с графикой:

```

class DrawingActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val drawingObjects = remember { mutableStateListOf("") } //список для рисования
            val buttonNames = arrayOf("Rect", "Circle", "Image") //массив с названиями кнопок
            ComposeExampleTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Column(Modifier.fillMaxSize()) { //колонка с интерфейсом
                        MakeTopButtons(buttonNames, drawingObjects)
                        Canvas(modifier = Modifier.fillMaxSize()) { //сам канвас, на котором рисуем
                            val canvasQuadrantSize = size / 2f //делим размер канваса на 2
                            val canvasWidth = size.width //получаем ширину канваса
                            drawingObjects.forEach { //проходим по элементам списка для рисования
                                if (it.contains("Rect")) //если это прямоугольник
                                    drawRect( //то рисуем его
                                        color = Color.Magenta, //цвет рисования
                                        size = canvasQuadrantSize //размер
                                    )
                                if (it.contains("Circle")) //если это круг
                                    drawCircle( //то рисуем его
                                        Color.Red, //цвет рисования
                                        radius = canvasWidth / 4f //и радиус
                                    )
                                if (it.contains("Image")) { //если это изображение
                                    val mBitmapFromSdcard = //то считываем с sd карты файл
                                    BitmapFactory.decodeFile("/mnt/sdcard/face.png").asImageBitmap()
                                    drawImage( //и выводим его на канвас
                                        image = mBitmapFromSdcard,
                                        topLeft = Offset(x = 0f, y = 0f) //координаты верхнего
                                    ) //левого угла для картинки
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}
}
//метод для создания кнопок
@Composable
fun MakeTopButtons(buttonNames: Array<String>, drawingObjects: SnapshotStateList<String>) {
    Row( //ряд для создания кнопок
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceEvenly,
        modifier = Modifier
            .wrapContentSize()
            .fillMaxWidth()
            .border(BorderStroke(2.dp, Color.Blue))
    ) {
        buttonNames.forEach { //цикл по названиям кнопок
            Button(onClick = { //создаем кнопку и описываем обработчик нажатия на нее
                //удаляем из списка для рисования объект с названием кнопки, если он там был
                drawingObjects.remove(it)
                drawingObjects.add(it) //и снова добавляем, это нужно для правильного
                //расположения объектов на холсте, в порядке нажатия
            }) {
                Text(text = it) //текст кнопки
            }
        }
    }
}

```

И теперь после запуска программы можно открывать боковое левое меню, выбирать пункт, который приведет в новое окно, в нем можно нажимать кнопки и видеть результат рисования (рис. 24).

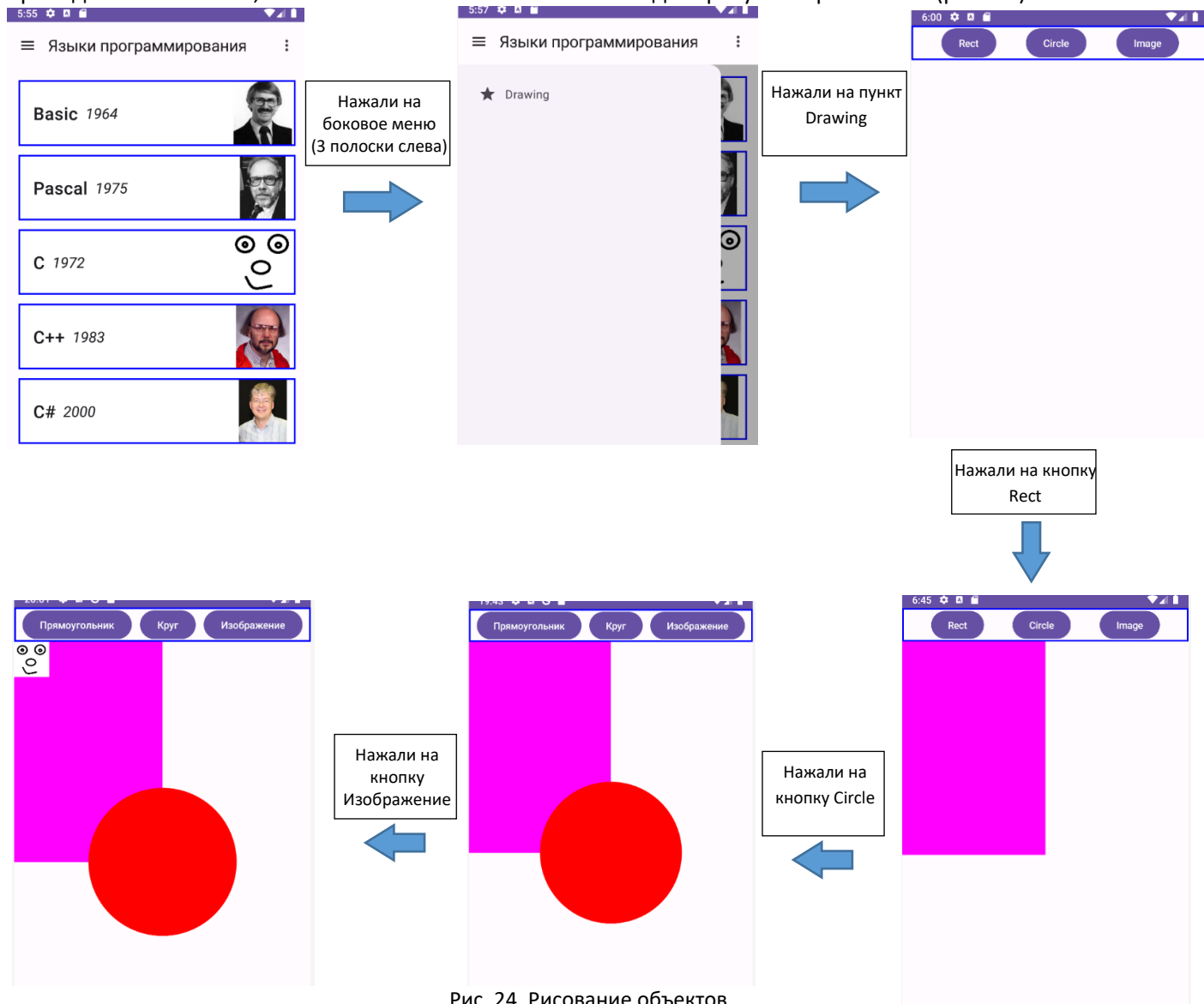


Рис. 24. Рисование объектов.

Если посмотреть внимательно на рис. 24, то можно заметить разные названия кнопок – на 4 скринах на английском, на двух – на русском. В данном случае к приложению была добавлена возможность интернационализации, т.е. в зависимости от языка интерфейса системы в программе подгружаются разные

строковые ресурсы. Для этого нужно, чтобы в папке res были отдельные папки values для каждого поддерживаемого языка (values – для английского, values-ru – для русского, values-fr – для французского и т.д.). В этих папках можно поместить файл strings.xml с описанием строковых ресурсов на нужном языке. Например, содержимое файлов strings.xml в соответствующих папках:

values \ strings.xml

```
<resources>
  <string name="app_name">Compose Example</string>
  <string name="Pascal" translatable="false">Автор - Никлаус Вирт.</string>
  <string name="C" translatable="false">Автор - Деннис Ритчи.</string>
  <string name="Java" translatable="false">Автор - Джеймс Гослинг.</string>
  <string name="title_activity_input" translatable="false">InputActivity</string>
  <string name="title_activity_drawing" translatable="false">DrawingActivity</string>
  <string name="title_activity_drawing2" translatable="false">DrawingActivity2</string>
  <string name="rect">Rect</string>
  <string name="circle">Circle</string>
  <string name="image">Image</string>
</resources>
```

values-ru \ strings.xml

```
<resources>
  <string name="app_name">Пример</string>
  <string name="rect">Прямоугольник</string>
  <string name="circle">Круг</string>
  <string name="image">Изображение</string>
</resources>
```

Далее, нужно сделать подгрузку названий кнопок из текстовых ресурсов. Для этого меняем объявление массива имен кнопок:

```
val buttonNames = arrayOf(
    stringResource(R.string.rect),
    stringResource(R.string.circle),
    stringResource(R.string.image)
)
```

И меняю сравнение

```
if (it.contains("Rect")) //если это прямоугольник
    drawRect( //то рисуем его
        color = Color.Magenta, //цвет рисования
        size = canvasQuadrantSize //размер
    )
на
```

```
if (it.contains(buttonNames[0]))
    drawRect(
        color = Color.Magenta,
        size = canvasQuadrantSize
    )
```

И аналогично для круга

```
if (it.contains(buttonNames[1]))
```

А для картинки делаем немного по-другому, с проверкой на существование картинки:

```
if (it.contains(buttonNames[2])) {
    try {
        val mBitmapFromSdcard = BitmapFactory.decodeFile("/mnt/sdcard/face.png").asImageBitmap()
        drawImage(
            image = mBitmapFromSdcard,
            topLeft = Offset(x = 0f, y = 0f)
        )
    } catch (e: NullPointerException) {
        Toast.makeText(applicationContext, "No image", Toast.LENGTH_LONG).show()
    }
}
```

И теперь, если системный язык – английский, то строковые ресурсы будут браться из values \ strings.xml, если русский, то из values-ru \ strings.xml (автоматически).

Можно добавить возможность свободного рисования к нашему проекту. Для этого в файл DrawingActivity.kt перед строкой `class DrawingActivity : ComponentActivity() {` добавляем класс

```
data class Line(
    val start: Offset,
    val end: Offset,
    val color: Color = Color.Black,
    val strokeWidth: Dp = 1.dp
)
```

Этот класс будет отвечать за объект Линия, которым мы и будем рисовать.

Далее, в блок setContent в самом начале добавляем строки

```
val lines = remember {
    mutableStateListOf<Line>()
}
```

Т.е. создаем объект для запоминания нарисованных линий. Начало блока Canvas переписываем в следующем виде:

```
Canvas(
    modifier = Modifier.
        fillMaxSize()
        .pointerInput(true) { //добавляем обработчик касаний
            detectDragGestures { change, dragAmount -> //и прописываем обработку движений
                // change - это объект, отвечающий за изменения при касании экрана,
                // dragAmount - смещение при касании экрана
                val line = Line( //создаем объект класса Line
                    start = change.position - dragAmount, // начальная позиция линии
                    end = change.position //конечная позиция линии
                )
                lines.add(line) //добавляем линию в объект для запоминания
            }
        }
)
```

И внутри блока Canvas после блока drawingObjects.forEach { ... } добавляем строки

```
lines.forEach { line -> //идем по всем добавленным линиям
    drawLine( //и рисуем реальный графический объект
        color = line.color, //цвет линии
        start = line.start, //начало линии
        end = line.end, //конец линии
        strokeWidth = line.strokeWidth.toPx(), //толщина линии
        cap = StrokeCap.Round //тип концов линии (тут закругленный)
    )
}
```

И теперь можно рисовать поверх нашего канваса (рис. 25).

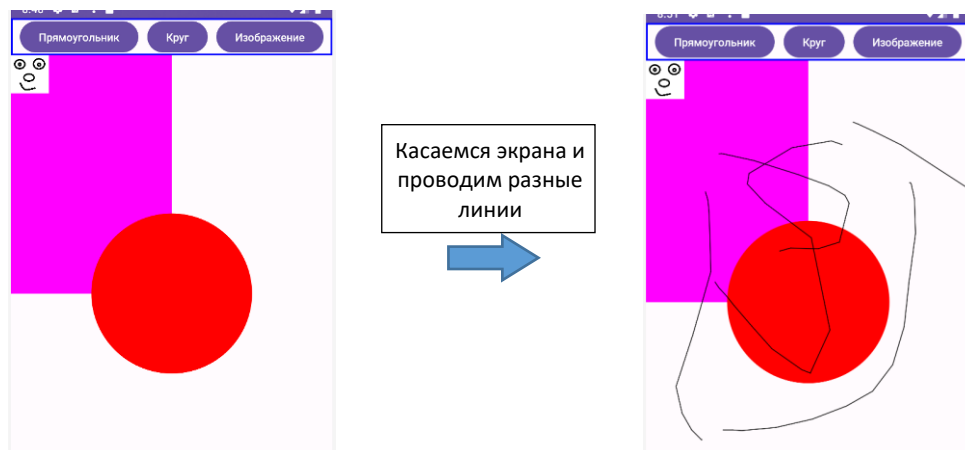


Рис. 25. Свободное рисование.

Второй способ рисования – создание собственного класса, унаследованного от класса View, и реализация в нем необходимых методов:

```
class MyGraphView(context: Context?) : View(context) {
    private lateinit var path: Path
    private var mPaint: Paint? = null //объект для параметров рисования графических примитивов
    private var mBitmapPaint: Paint? = null //объект для параметров вывода битмапа на холст
    private var mBitmap : Bitmap? = null //сам битмап
    private var mCanvas: Canvas? = null //холст
    init { //секция инициализации полей класса
        //создаем объект класса Paint для параметров вывода битмапа на холст
        mBitmapPaint = Paint(Paint.DITHER_FLAG) // Paint.DITHER_FLAG - для эффекта сглаживания
        mPaint = Paint() //создаем объект класса Paint для параметров рисования графики
    }
}
```



```

        mPaint!!.setAntiAlias(true) //устанавливаем антиалиасинг (сглаживание)
        mPaint?.setColor(Color.GREEN) //цвет рисования
        mPaint?.setStyle(Paint.Style.STROKE) //стиль рисования
// (Paint.Style.STROKE - без заполнения)
        mPaint?.setStrokeJoin(Paint.Join.ROUND) //стиль соединения линий (ROUND - скруглённый)
        mPaint?.setStrokeCap(Paint.Cap.ROUND) //стиль концов линий (ROUND - скруглённый)
        mPaint?.setStrokeWidth(12F) //толщина линии рисования
    }
    //метод onSizeChanged вызывается первый раз при создании объекта,
    //далее - при изменении размера объекта, нам он нужен для выяснения первичных размеров битмапа
    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
        super.onSizeChanged(w, h, oldw, oldh)
        //создаем битмап с высотой и шириной как у текущего объекта и с параметром
        Bitmap.Config.ARGB_8888,
        //это четырехканальный RGB (прозрачность и 3 цвета)
        mBitmap = Bitmap.createBitmap(w, h, Bitmap.Config.ARGB_8888)
        mCanvas = Canvas(mBitmap!!) //создаем канвас и связываем его с битмапом
        Toast.makeText(this.context, "onSizeChanged ", Toast.LENGTH_SHORT).show() //для отладки
    }
    //метод перерисовки объекта, он будет срабатывать каждый раз
    override fun onDraw(canvas: Canvas) { //при вызове функции invalidate() текущего объекта
        super.onDraw(canvas)
        //отрисовываем на канвасе текущего объекта (не путать с созданным нами канвасом) наш битмап
        canvas.drawBitmap(mBitmap!!, 0f, 0f, mBitmapPaint!!)
    }
    fun drawCircle() { //метод для рисования круга
        println("mCanvas = $mCanvas")
        mCanvas!!.drawCircle(100f, 100f, 50f, mPaint!!)
        invalidate() //для срабатывания метода onDraw
    }
    fun drawSquare() { //метод для рисования квадрата
        println("mCanvas = $mCanvas")
        mCanvas!!.drawRect(200f, 200f, 300f, 300f, mPaint!!)
        invalidate()
    }
    fun drawFace() { //метод для рисования картинки из файла
        //создаем временный битмап из файла
        val mBitmapFromSdcard = BitmapFactory.decodeFile("/mnt/sdcard/face.png")
        mCanvas!!.drawBitmap(mBitmapFromSdcard, 100f, 100f, mPaint) //рисует его на нашем канвасе
        invalidate()
    }

    fun onSaveClick() { // метод для сохранения нарисованного
        //получаем путь к каталогу программы на карте памяти (для этого проекта -
        // /storage/emulated/0/Android/data/com.example.composeexample/files)
        val destPath: String = context.getExternalFilesDir(null)!!.absolutePath
        var outputStream: OutputStream? = null //объявляем поток вывода
        val file = File(destPath, "my.PNG") //создаем файл с нужным путем и названием
        println("path = $destPath") //вывод в консоль для отладки
        outputStream = FileOutputStream(file) //создаем объект потока и связываем его с файлом
        //у нашего битмапа вызываем функцию для записи его с нужными параметрами (тип графического файла,
        //качество в процентах и поток для записи)
        mBitmap!!.compress(Bitmap.CompressFormat.PNG, 100, outputStream)
        outputStream.flush() //для прохождения данных вызываем функцию flush у потока
        outputStream.close() //закрываем поток
    }
    //создаем массив функций (понадобится позже)
    val funcArray = arrayOf(::drawSquare, ::drawCircle, ::drawFace, ::onSaveClick)

    //этот метод будет срабатывать при касании нашего объекта пользователем (для свободного рисования)
    override fun onTouchEvent(event: MotionEvent): Boolean { //event хранит информацию о событии
        when (event.action) { // в зависимости от события
            MotionEvent.ACTION_DOWN -> { //если пользователь только коснулся объекта
                path = Path() //создаем новый объект класса Path для записи линии рисования
                path.moveTo(event.x, event.y) //перемещаемся к месту касания
            }
        }
        //если пользователь перемещает палец по экрану или отпустил палец
        //проводим линию в объекте path до точки касания
        MotionEvent.ACTION_MOVE, MotionEvent.ACTION_UP -> path.lineTo(event.x, event.y)
    }
    if (path != null) { //если объект не нулевой
        println("mCanvas = $mCanvas")
        mCanvas!!.drawPath(path, mPaint!!) //рисует на канвасе объект path (и что с ним связано)
        invalidate() //для срабатывания метода onDraw
    }
    return true
}

```

```

    }
}

```

А файл `DrawingActivity.kt` приводим к следующему виду:

```

class DrawingActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val buttonNames = arrayOf(
                stringResource(R.string.rect),
                stringResource(R.string.circle),
                stringResource(R.string.image),
                stringResource(R.string.save) //добавляем надпись для кнопки Save
            )
            // создаем наш объект для рисования
            val myView: MyGraphView? = MyGraphView(applicationContext)
            val viewRemember = remember { //и объект для его хранения
                mutableStateOf(myView)
            }
            ComposeExampleTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Column(Modifier.fillMaxSize()) {
                        //вызываем функцию для создания кнопок, ей передаем массив с именами и объект для рисования
                        MakeTopButtons(buttonNames, viewRemember.value)
                        //и вызываем функцию с нашим объектом для рисования, где и будет всё отображаться
                        CustomView(viewRemember.value)
                    }
                }
            }
        }
    }

    @Composable
    fun MakeTopButtons(buttonNames: Array<String>, myView: MyGraphView?) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceEvenly,
            modifier = Modifier
                .wrapContentHeight()
                .fillMaxWidth()
                .border(BorderStroke(2.dp, Color.Blue))
        ) {
            buttonNames.forEach {
                Button(onClick = { //цикл по названиям кнопок
                    //и вызываем из массива функций нужный метод согласно номеру кнопки в массиве
                    myView!!.funcArray[buttonNames.lastIndexOf(it)] ()
                }) {
                    Text(text = it) //текст для каждой кнопки
                }
            }
        }
    }

    @Composable
    fun CustomView(myView: MyGraphView?) { //функция для вставки нашего объекта для рисования
        AndroidView(
            modifier = Modifier.fillMaxSize(),
            factory = { context ->
                myView!! //сам объект
            },
        )
    }
}

```

Результат запуска можно увидеть на рис. 26.

При нажатии на кнопку `Save` нарисованное сохранится в файл `my.png` в каталоге `/storage/emulated/0/Android/data/com.example.composeexample/files` (`com.example.composeexample` – это название проекта, будет меняться в зависимости от проекта, остальное - стандартное).

Если файл не сохраняется из-за отсутствия разрешения, то можно прописать в файле AndroidManifest.xml перед тегом `<application>` строку

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

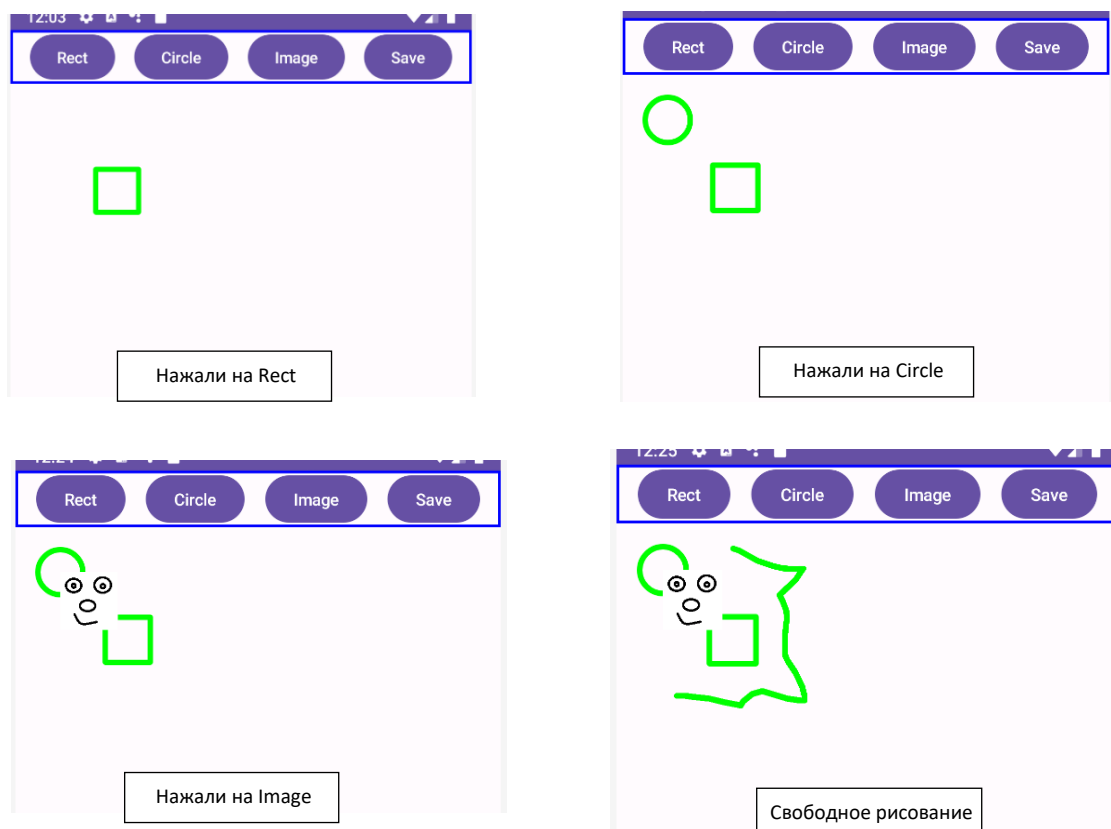


Рис. 26. Второй способ рисования

Задания

1. Добавить к проекту возможности по локализации (английский и русский варианты всех надписей в ресурсах).
2. Реализовать аналог графического редактора с базовыми возможностями (выбор цвета, толщины и стиля линий, рисование примитивов, свободное рисование, сохранение (с выбором названия файла) и загрузка изображений). Отдельной кнопкой или пунктом меню должна рисоваться фамилия автора лабораторной.