АКТ(ф) СПбГУТ

**ОТЧЕТ**

по лабораторным и практическим работам

Разработка мобильных приложений

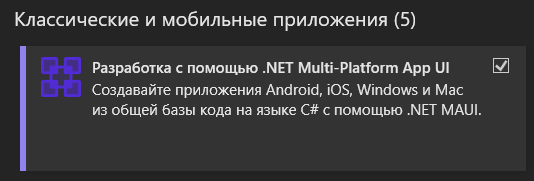
Студент ИСПП-21 21.06.2025 С.А.Миклякова

Преподаватель 21.06.2025 Р.В.Садовский

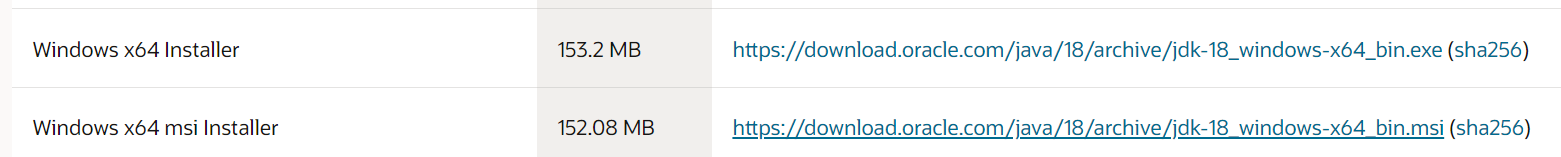
Архангельск 2025

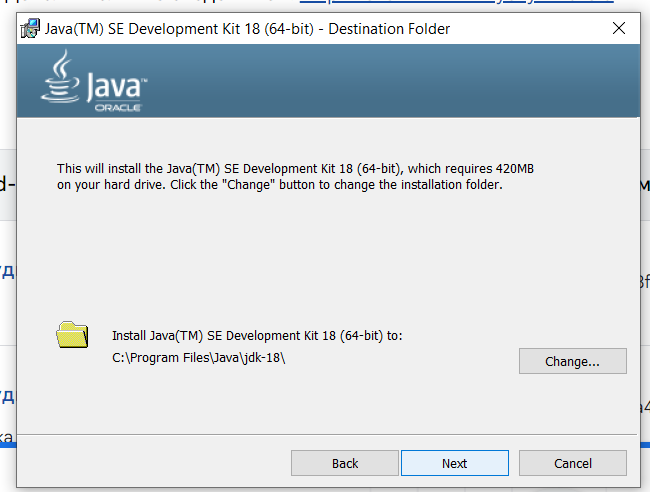
# Лабораторная работа №1 Установка инструментария и настройка среды для разработки мобильных приложений

1. **Цель работы**
   1. Изучить процесс установки среды для разработки мобильных приложений.
2. **Ход работы**
   1. Установили в Visual Studio средства разработки MAUI

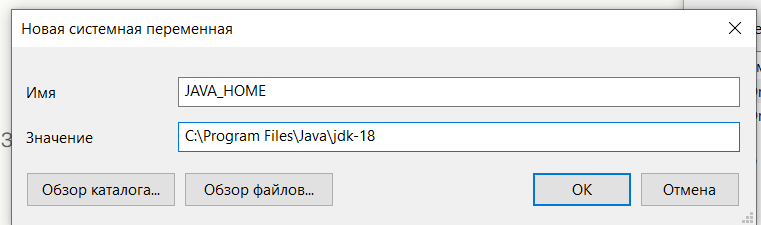


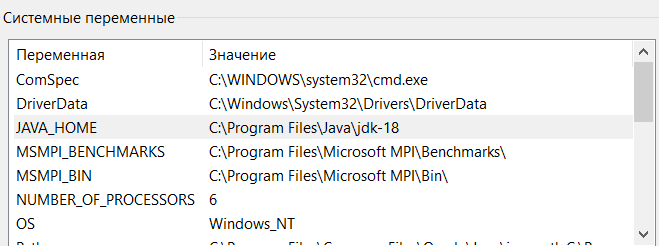
* 1. Установили JDK
     1. Скачали установочный файл Java JDK (Java SE Development Kit 18

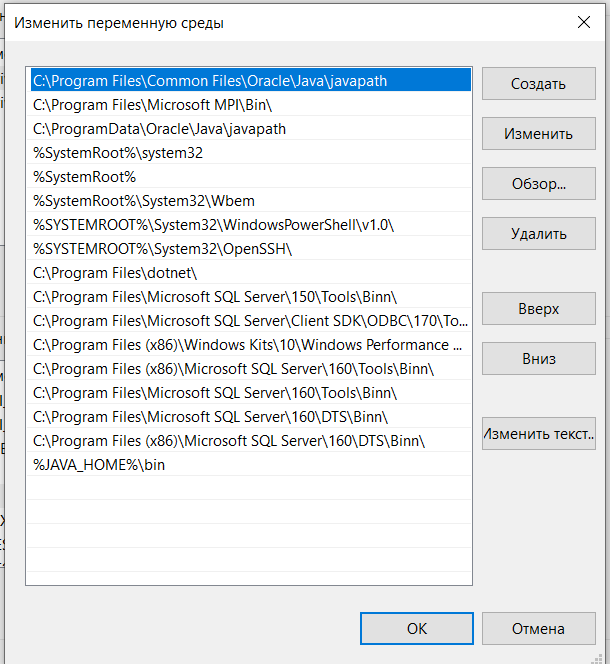


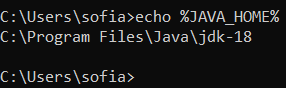


* + 1. Добавили в настройках ОС переменную среды JAVA\_HOME, в которой указать путь к JDK.



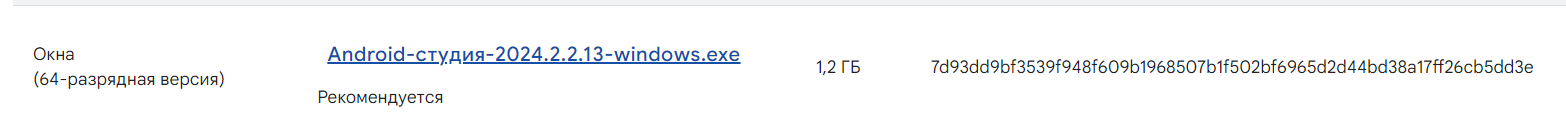




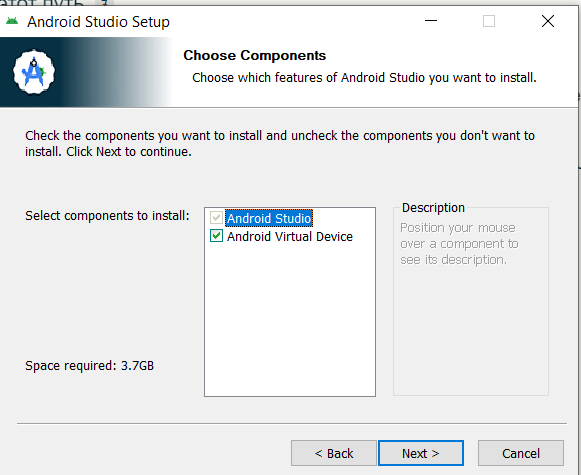




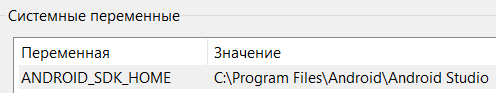
* 1. Установили Android Studio
     1. Скачали установочный файл android-studio-2024.2.2.13-windows.exe



* + 1. Отметить флажком Android Virtual Device



* + 1. Измененили расположение файла подкачки (папки .android)



* 1. Создали проект и установили SDK

1. **Ответы на контрольные вопросы**
   1. Среды разработки, поддерживающие создание мобильных приложений: Android Studio, IntelliJ IDEA, Visual Studio, Xcode, Xamarin, React Native, Flutter.
   2. Языки программирования для создания мобильных приложений: Java, Kotlin, C#, JavaScript, C/C++.
   3. Языки программирования, доступные в Android Studio: Kotlin, Java.
   4. Java – основной язык программирования для разработки приложений на Android. Он поддерживается компанией Google, и большинство приложений в Google Play построены именно на нём.
   5. Для работы Android Studio необходимо установить Java Development Kit (JDK).
   6. Java Development Kit (JDK) – это набор инструментов разработки программного обеспечения, используемых для разработки приложений на Java. В его состав входят: компилятор java (javac), инструменты для упаковки и развертывания приложений (jar, javadoc), Java Runtime Environment (JRE), необходимая для запуска Java-приложений.
2. **Вывод**
   1. В ходе лабораторной работы был изучен процесс установки среды для разработки мобильных приложений.

# Лабораторная работа №2

**Установка среды разработки мобильных приложений с применением виртуальной машины**

1. **Цель работы**
   1. Изучить процесс установки среды для разработки мобильных приложений с применением виртуальной машины.
2. **Ход работы**
   1. Создали в VirtualBox виртуальную машину Ubuntu 24.04.
   2. Установили на виртуальной машине среду разработки Intellij IDEA.
   3. Установили новую версию плагина Kotlin.
   4. Установили на виртуальной машине среду разработки Android Studio.
   5. Создали в VirtualBox виртуальную машину Android.
3. **Ответы на контрольные вопросы**
   1. Виртуальная машина – это абстрактная вычислительная среда, созданная программным способом на физическом оборудовании, имитирующая компьютер или сервер с полноценной операционной системой и аппаратным обеспечением.
   2. IDE для создания приложений для Android: Android Studio, IntelliJ IDEA, Visual Studio.
   3. Языки программирования, доступные в IntelliJ IDEA: Java, JavaScript, CoffeeScript, Python, Ruby, PHP, Kotlin, Rust и многие другие.
   4. Языки программирования, доступные в Android Studio: Kotlin, Java.
   5. Языки программирования для разработки нативных приложений для Android: Java, Kotlin.
4. **Вывод**
   1. В ходе лабораторной работы был изучен процесс установки среды для разработки мобильных приложений с применением виртуальной машины.

# Лабораторная работа №3

**Разработка линейных алгоритмов**

1. **Цель работы**
   1. Изучить процесс разработки линейных алгоритмов в приложениях на языке Kotlin;
   2. Изучить процесс ввода и вывода данных в приложениях на языке Kotlin.
2. **Ход работы**
   1. Запросили у пользователя ввод двух целых чисел a и b. Реализовали вывод результата выполнения арифметических операций (сложение, вычитание, умножение, деление, остаток от деления).
   2. Запросили у пользователя имя (строку), рост (вещественное число), массу тела (целое число). Вывели на экран сообщение: «Имя, ваш ИМТ=значение».
   3. Запросили у пользователя ввод целого числа n (количество секунд, прошедшее с начала суток). Определили, сколько часов, минут и секунд будет показано на табло электронных часов, результат вывели в формате чч:мм:сс.
   4. Запросили у пользователя ввод года, реализовали вывод true или false в зависимости от того, високосный год введен или нет.
   5. Запросили у пользователя ввод двух чисел a и b. Реализовали вывод случайного целого числа в диапазоне от a до b и случайного вещественного в диапазоне от a до b.
   6. Запросили у пользователя ввод внешнего и внутреннего радиусов и найти площадь кольца на основе значений, введенных пользователем. Результат вывели с тремя знаками после запятой.
3. **Контрольные вопросы**
   1. Объявление переменной в Kotlin:

val|var имя\_переменной: Тип = значение

* 1. Ввод данных на Kotlin:

print("Введите имя: ")

val name = readLine()

* 1. Вывод данных на Kotlin

println(переменная)

println("строка")

* 1. Для того, чтобы преобразовать значение из строкового в числовой, можно использовать следующие функции:
* toInt()/toIntOrNull(): преобразует строку в целое число;

toLong() / toLongOrNull(): преобразует строку в значение типа Long;

toFloat()/toFloatOrNull(): преобразует строку в значение типа Float;

toDouble()/toDoubleOrNull(): преобразует строку в значение типа Double;

toShort()/toShortOrNull(): преобразует строку в значение типа Short.

* 1. Чтобы округлить данные на Kotlin, можно использовать функцию String.format("%.nf", number).
  2. Чтобы сгенерировать случайное число на Kotlin, можно использовать класс kotlin.random.Random:

val randomNumber = Random.nextInt(0, 100)

* 1. Отличие между ключевыми словами var и val в Kotlin:
* var: используется для объявления изменяемых переменных, значения которых можно менять после инициализации;

val: используется для объявления неизменяемых переменных, которым значение присваивается только один раз. После присвоения значения val переменной, его нельзя изменить.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки линейных алгоритмов в приложениях на языке Kotlin;
   2. В ходе лабораторной работы был изучен процесс ввода и вывода данных в приложениях на языке Kotlin.

# Лабораторная работа №4

**Разработка разветвляющихся алгоритмов**

1. **Цель работы**
   1. Изучить процесс разработки разветвляющихся алгоритмов в приложениях на Kotlin.
2. **Ход работы**
   1. Написали программу, определяющую по введенному пользователем номеру месяца сезон.
   2. Написали программу, определяющую по введенному пользователем году и номеру месяца, является ли введенный пользователем год високосным и количество дней в месяце.
   3. Написали приложение «Конвертер валют».
   4. Написали программу, вычисляющую значение функции у(x).
   5. Написли программу, запрашивающую сумму покупки и внесенную покупателем сумму. На экране вывели сумму к оплате с учетом скидки.
3. **Ответы на контрольные вопросы**
   1. Синтаксис условного оператора на Kotlin:

if (условие) {

// Код, который выполняется, если условие истинно

} else {

// Код, который выполняется, если условие ложно

}

* 1. Синтаксис оператора множественного выбора на Kotlin

when (переменная) {

значение1 -> {

// Код для выполнения, если переменная равна значению1

}

значение2 -> {

// Код для выполнения, если переменная равна значению2

}

else -> {

// Код для выполнения, если ни одно из вышеперечисленных значений не совпадает

}

}

* 1. Синтаксис тернарного оператора на Kotlin:

val переменная = if (условие) значение\_если\_истина else значение\_если\_ложь

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки разветвляющихся алгоритмов в приложениях на Kotlin.

# Лабораторная работа №5

**Разработка циклических алгоритмов**

1. **Цель работы** 
   1. Изучить процесс разработки циклических алгоритмов в приложениях на Kotlin.
2. **Ответы на контрольные вопросы**
   1. Синтаксис цикла for в Kotlin:

for (element in collection) {

// Тело цикла

}

* 1. Синтаксис цикла while в Kotlin:

text

while (условие) {

// Тело цикла

}

* 1. Синтаксис цикла do-while в Kotlin:

text

do {

// Тело цикла

} while (условие)

* 1. Kotlin для досрочного выхода из цикла используются операторы break, continue, return.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки циклических алгоритмов в приложениях на Kotlin.

# Лабораторная работа №6

**Разработка и вызов функций**

1. **Цель работы**
   1. Изучить процесс разработки функций в приложениях на Kotlin.
2. **Ответы на контрольные вопросы** 
   1. Синтаксис создания функций на Kotlin:

fun имяФункции(параметр1: Тип1, параметр2: Тип2): ВозвращаемыйТип {

// тело функции

}

* 1. Можно задавать значения по умолчанию для параметров функции прямо при их объявлении:

fun greet(name: String = "Sonya") {

println("Hello, $name")

}

* 1. Для параметров с переменным количеством аргументов используется ключевое слово vararg:

fun printAll(vararg messages: String) {

for (m in messages) println(m)

}

* 1. Функции высокого порядка – это функции, которые принимают в качестве параметров другие функции или возвращают функции.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки функций в приложениях на Kotlin.

# Лабораторная работа №7

**Разработка классов**

1. **Цель работы**
   1. Изучить процесс разработки и применения классов на языке Kotlin.
2. **Ответы на контрольные вопросы** 
   1. Класс объявляется с помощью ключевого слова class.
   2. Свойства класса объявляются внутри тела класса или в основном конструкторе с помощью ключевых слов val или var.
   3. Синтаксис объявления геттера и сеттера свойства в Kotlin:

var name: String = "Default"

get() = field.toUpperCase()

set(value) {

field = value.trim()

}

* 1. Конструктор объявляется с помощью слова constructor.
  2. Объект создаётся вызовом конструктора:

val person = Person("John", 30)

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки и применения классов на языке Kotlin.

# Лабораторная работа №8

**Наследие классов**

1. **Цель работы**
   1. Изучить процесс разработки дочерних классов в приложениях на Kotlin.
2. **Ответы на контрольные вопросы** 
   1. Наследование – это механизм объектно-ориентированного программирования, который позволяет создавать новый класс (наследник), расширяющий или изменяющий поведение уже существующего класса (родителя).
   2. В Kotlin класс может наследоваться только от одного родительского класса.
   3. Класс в Kotlin может реализовывать любое количество интерфейсов.
   4. Родительский класс указывается после имени класса-наследника через двоеточие.
   5. Чтобы переопределить метод родительского класса, он должен быть объявлен с модификатором open в родителе, а в дочернем классе для переопределения используется ключевое слово override.
   6. Для вызова метода родительского класса внутри переопределённого метода используется ключевое слово super.
   7. Чтобы разрешить наследование, класс нужно объявить с модификатором open.
   8. Чтобы запретить наследование от класса, его объявляют с модификатором final (по умолчанию все классы final).
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки дочерних классов в приложениях на Kotlin.

# Лабораторная работа №9

**Обработка коллекций**

1. **Цель работы**
   1. Изучить процесс разработки дочерних классов в приложениях на Kotlin.
2. **Ответы на контрольные вопросы** 
   1. В Kotlin есть три основных стандартных типа коллекций: List, Set, Map.
   2. Объявление коллекций обычно происходит с помощью специальных функций:

val list = listOf(1, 2, 3)

val set = setOf("a", "b", "c")

val map = mapOf("key1" to 1, "key2" to 2)

* 1. Для добавления элемента в список используется метод add:

val list = mutableListOf(1, 2)

list.add(3)

* 1. Для добавления элемента в словарь используется метод put:

kotlin

val map = mutableMapOf<String, Int>()

map.put("key2", 100)

* 1. Для перебора элементов коллекции используется цикл for

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс разработки дочерних классов в приложениях на Kotlin.

# Лабораторная работа №10

**Использование корутин**

1. **Цель работы**
   1. Изучить процесс создания и использования сопрограмм в приложениях на Kotlin.
2. **Ответы на контрольные вопросы** 
   1. Корутина – это легковесный поток или паттерн проектирования для написания асинхронных программ, позволяющий выполнять несколько задач одновременно без создания множества системных потоков.
   2. launch – это корутин-билдер, который запускает новую корутину в заданном контексте и не блокирует текущий поток. Он используется для запуска асинхронных задач, когда не требуется возвращать результат.
   3. suspend – модификатор функций, которые могут приостанавливать своё выполнение без блокировки потока. Такие функции могут вызываться только из других suspend-функций или внутри корутин. Они позволяют писать асинхронный код в последовательном стиле, используя точки приостановки, при этом освобождая поток для других задач.
   4. Выполнение корутины приостанавливается с помощью специальных suspend-функций, например, delay().
   5. Отмена корутины происходит через вызов метода cancel() у объекта Job, который возвращается функцией launch или другими корутин-билдерами.
   6. Для работы с корутинами в Kotlin необходимо подключить библиотеку kotlinx.coroutines. В Gradle это делается добавлением зависимости implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:версия").
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс создания и использования сопрограмм в приложениях на Kotlin.

# Лабораторная работа №11

**Создание эмуляторов и подключение устройств**

1. **Цель работы**
   1. Изучить процесс создания эмуляторов и подключения устройств для мобильной разработки.
2. **Ответы на контрольные вопросы** 
   1. AVD (Android Virtual Device) – это виртуальное Android-устройство, эмулятор смартфона или планшета, на котором можно запускать и тестировать Android-приложения без необходимости использовать реальное устройство.
   2. Эмулятор – это программный инструмент, который имитирует работу физического устройства Android на компьютере, позволяя запускать и отлаживать приложения в виртуальной среде, повторяющей характеристики реального устройства.
   3. Профиль устройства определяет аппаратные характеристики виртуального устройства – размер экрана, объём оперативной памяти, состояние датчиков, разрешение, версию Android и другие параметры, которые влияют на поведение и внешний вид эмулируемого устройства.
   4. Для создания нового эмулятора в Android Studio нужно открыть AVD Manager (через меню Tools или Device Manager), нажать кнопку создания нового виртуального устройства, выбрать шаблон устройства (например, Pixel XL), затем выбрать образ операционной системы Android и настроить параметры.
   5. Расширенные настройки эмулятора задаются в AVD Manager при создании или редактировании виртуального устройства.
   6. Вместо стандартного эмулятора Android Studio можно использовать сторонние эмуляторы, например, Genymotion, который часто работает быстрее и предлагает дополнительные возможности для тестирования Android-приложений.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс создания эмуляторов и подключения устройств для мобильной разработки.

# Лабораторная работа №12

**Создание эмуляторов и подключение устройств**

1. **Цель работы**
   1. Изучить процесс использования терминала при разработке мобильных приложений Android.
2. **Ответы на контрольные вопросы** 
   1. adb (Android Debug Bridge) — это инструмент командной строки для отладки и управления устройствами на базе Android.
   2. adb входит в состав пакета Android SDK Platform Tools. Обычно путь к папке с adb находится в каталоге: <путь\_к\_Android\_SDK>/platform-tools/
   3. adb запускается из командной строки (терминала). Для этого нужно открыть терминал, перейти в папку с adb (например, platform-tools) и ввести команду: adb.
   4. Для просмотра списка доступных команд adb достаточно ввести в терминале: adb help.
   5. Для получения доступа к командной оболочке (shell) Android-устройства из adb используется команда: adb shell.
   6. Для снятия скриншота используется команда:

adb shell screencap /sdcard/screenshot.png

adb pull /sdcard/screenshot.png <путь\_на\_компьютере>

Для записи видео экрана:

adb shell screenrecord /sdcard/demo.mp4

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс использования терминала при разработке мобильных приложений Android.

# Лабораторная работа №13

**Создание нового проекта**

1. **Цель работы**
   1. Изучить процесс создания приложения в Android Studio.
2. **Ответы на контрольные вопросы** 
   1. Jetpack Compose – это современный набор инструментов от Google для создания пользовательских интерфейсов в Android-приложениях с использованием декларативного подхода.
   2. Для создания проекта с Jetpack Compose в Android Studio рекомендуется выбрать шаблон «Empty Compose Activity»
   3. Activity – это один из основных компонентов Android-приложения, представляющий экран с пользовательским интерфейсом.
   4. MainActivity – это обычно главная (стартовая) Activity приложения, с которой начинается работа пользователя. В ней обычно инициализируется UI и запускается логика приложения.
   5. Функция onCreate – это метод жизненного цикла Activity, который вызывается при создании Activity. В ней происходит инициализация компонентов, установка интерфейса и начальная настройка.
   6. В Jetpack Compose функция setContent используется внутри Activity для определения содержимого экрана с помощью Composable-функций. Она заменяет традиционную установку XML-разметки и позволяет напрямую задавать UI через Kotlin-код.
   7. Аннотация @Composable помечает функцию как composable — то есть функцию, которая описывает часть пользовательского интерфейса в Jetpack Compose. Такие функции могут вызываться внутри других composable-функций для построения UI декларативно.
   8. Аннотация @Preview используется для отображения превью composable-функции прямо в редакторе Android Studio без запуска приложения на устройстве или эмуляторе. Это позволяет быстро видеть, как будет выглядеть UI-компонент.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс использования терминала при разработке мобильных приложений Android.

# Лабораторная работа №14

**Изучение и комментирование кода**

1. **Цель работы**
   1. Изучить процесс документирования и комментирования приложения Android.
2. **Ответы на контрольные вопросы** 
   1. Комментарии в стиле KDoc оформляются как многострочные комментарии, начинающиеся с /\*\* и заканчивающиеся \*/. Каждая строка внутри может начинаться со звёздочки \*, которая не входит в содержимое комментария.
   2. Общая форма KDoc-комментария:

/\*\*

\* Краткое описание элемента.

\*

\* Подробное описание, если необходимо.

\*

\* @тег описание

\*/

* 1. Для описания свойств класса используется тег @property с указанием имени свойства и его описанием.
  2. Для параметров функции используется тег @param, после которого указывается имя параметра и его описание.
  3. Для описания возвращаемого значения используется тег @return с описанием результата.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс документирования и комментирования приложения Android.

# Лабораторная работа №15

**Изучение и комментирование кода**

1. **Цель работы**
   1. Изучить процесс изменения элементов дизайна приложения в Android Studio.
2. **Ответы на контрольные вопросы** 
   1. Модификаторы в Jetpack Compose применяются через параметр modifier у composable-функций и записываются цепочкой вызовов функций расширения Modifier.
   2. Для задания фона используется модификатор background, которому передаётся цвет или Brush.
   3. Размер элемента задаётся с помощью модификаторов size, width, height, fillMaxWidth(), fillMaxHeight(), fillMaxSize().
   4. Отступы задаются с помощью модификатора padding.
   5. Для настройки текста в Compose используются параметры компонента Text, например fontSize, color, fontWeight, fontStyle, textAlign.
   6. Для отображения изображения из ресурсов используется компонент Image с параметром painter.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс изменения элементов дизайна приложения в Android Studio.

# Лабораторная работа №16

**Изучение и комментирование кода**

1. **Цель работы**
   1. Изучить процесс добавления и применения ресурсов при разработке мобильных приложений Android.
2. **Ответы на контрольные вопросы** 
   1. В папке res хранятся все ресурсы Android-приложения, такие как изображения, файлы разметки, строки, цвета, стили и другие некодовые данные, используемые в приложении.
   2. В папке res/values хранятся ресурсы в виде XML-файлов с различными константами и значениями, такими как строки (strings.xml), цвета (colors.xml), размеры (dimens.xml), стили (styles.xml) и другие параметры.
   3. В папке res/drawable хранятся графические ресурсы приложения – растровые изображения (PNG, JPG, GIF), а также XML-файлы с описанием векторных изображений или анимаций.
   4. Resource Manager – это инструмент в Android Studio, который позволяет удобно управлять всеми ресурсами проекта: просматривать, добавлять, редактировать изображения, строки, цвета и другие ресурсы. Он обеспечивает визуальный интерфейс для работы с ресурсами и упрощает их организацию и использование.
   5. Для использования строки в коде или разметке применяется ссылка через класс R.string.
   6. Изображения из папки drawable используются в коде и разметке через идентификаторы R.drawable.
   7. Для доступа к массивам используется класс R.array.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс добавления и применения ресурсов при разработке мобильных приложений Android.

# Лабораторная работа №17

**Настройка элементов управления для ввода данных**

1. **Цель работы**
   1. Изучить процесс настройки и использования элементов управления для ввода данных приложения Android.
2. **Ответы на контрольные вопросы** 
   1. Для ввода текстовых значений в интерфейсе обычно используются элементы управления, такие как TextField и OutlinedTextField.
   2. Тип клавиатуры на экране можно настроить с помощью параметра keyboardOptions у элемента TextField.
   3. В Jetpack Compose для связывания значения поля ввода с переменной используется паттерн state hoisting — значение хранится в переменной состояния, а изменение значения передаётся через колбэк onValueChange.
   4. Combobox — это элемент управления, позволяющий выбрать одно значение из выпадающего списка.
   5. RadioButton используется для выбора одного варианта из группы взаимно исключающих опций.
   6. Slider — элемент управления для выбора значения из диапазона с помощью ползунка.
   7. Switch — переключатель, представляющий булевое состояние.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки и использования элементов управления для ввода данных приложения Android.

# Лабораторная работа №18

**Настройка кнопок**

1. **Цель работы**
   1. Изучить процесс настройки и использования элементов управления кнопок в приложениях Android.
2. **Ответы на контрольные вопросы** 
   1. В Jetpack Compose кнопка создаётся с помощью компонента Button.
   2. IconButton — это кнопка, содержащая иконку вместо текста. Используется для компактных кнопок с графическим содержимым.
   3. IconToggleButton — это кнопка с иконкой, которая может переключаться между состояниями включено/выключено. Используется для выбора или отмены выбора с визуальным индикатором.
   4. FloatingActionButton (FAB) — плавающая кнопка действия, обычно круглая, выступающая над основным содержимым и используемая для выполнения основного действия на экране.
   5. ExtendedFloatingActionButton — расширенная версия FAB, которая содержит не только иконку, но и текст. Используется для более информативных кнопок действия.
   6. При создании Icon указываются imageVector или painter contentDescription, modifier, tint.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки и использования элементов управления кнопок в приложениях Android.

# Лабораторная работа №19

**Отображение списков и таблиц**

1. **Цель работы**
   1. Изучить процесс настройки и использования элементов управления для отображения списков в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. LazyColumn — это компонент Jetpack Compose для создания вертикального списка с прокруткой
   2. LazyRow — аналог LazyColumn, но для горизонтального списка с прокруткой.
   3. LazyVerticalGrid — это компонент для создания сетки с вертикальной прокруткой, где элементы располагаются в виде ячеек по строкам и столбцам.
   4. LazyHorizontalGrid — компонент для создания сетки с горизонтальной прокруткой, где элементы располагаются в ячейках по столбцам и строкам, прокручиваясь по горизонтали.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки и использования элементов управления для отображения списков в приложении Android.

# Лабораторная работа №20

**Обработка событий: переключение между экранами**

1. **Цель работы**
   1. Изучить процесс настройки и реализации перехода между экранами в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. Для поддержки навигации в Jetpack Compose нужно добавить в файл build.gradle модуля зависимость implementation("androidx.navigation:navigation-compose:2.9.0")
   2. NavHost – это контейнер, который отображает навигационный граф и управляет отображением соответствующих composable-экранов в зависимости от текущего маршрута.
   3. NavController – это объект, управляющий навигацией в приложении: он контролирует стек экранов (backstack), обеспечивает переходы вперёд и назад, а также управление состоянием навигации.
   4. В навигационном графе (NavGraph) указываются маршруты экранов, их composable-функции и связи между ними. Также задаётся стартовый экран.
   5. sealed class используется для описания маршрутов, когда требуется типобезопасное иерархическое представление экранов с возможностью хранения параметров и логики маршрутизации в одном месте.
   6. enum применяется для простых списков маршрутов без необходимости хранения параметров или сложной логики.
   7. Параметры, переданные в навигации, считываются в целевом composable через объект NavBackStackEntry, доступный в лямбде composable.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки и реализации перехода между экранами в приложении Android.

# Лабораторная работа №21

**Обработка событий: подсказки**

1. **Цель работы**
   1. Изучить процесс настройки и использования элементов управления для отображения уведомлений и подсказок.
2. **Ответы на контрольные вопросы** 
   1. Для создания диалогового окна в Jetpack Compose используется класс-компонент AlertDialog.
   2. Общая форма использования AlertDialog:

AlertDialog(

onDismissRequest = { /\* действие при закрытии диалога \*/ },

title = { Text("Заголовок") },

text = { Text("Текст сообщения") },

confirmButton = {

TextButton(onClick = { /\* подтверждение \*/ }) {

Text("ОК")

}

},

dismissButton = {

TextButton(onClick = { /\* отмена \*/ }) {

Text("Отмена")

}

}

)

* 1. Для отображения всплывающих сообщений используется класс SnackbarHost.
  2. Функция onDismissRequest вызывается, когда пользователь пытается закрыть диалоговое окно, например, нажатием вне его области или кнопкой "Назад". В ней обычно реализуют логику скрытия диалога, например, изменение состояния, управляющего видимостью.
  3. SnackbarHost служит контейнером для отображения snackbar-сообщений – кратких уведомлений, которые появляются поверх интерфейса и автоматически исчезают.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки и использования элементов управления для отображения уведомлений и подсказок.

# Лабораторная работа №22

**Обработка событий: индикация**

1. **Цель работы**
   1. Изучить процесс изменения интерфейса для информирования пользователей в приложениях Android.
2. **Ответы на контрольные вопросы** 
   1. В Android обычно используются классы индикаторов прогресса ProgressBar, LinearProgressIndicator и CircularProgressIndicator.
   2. В Jetpack Compose значение прогресса передаётся параметром progress типа Float от 0.0 до 1.0, и шаг задаётся изменением этого значения.
   3. Для имитации задержки или анимации прогресса можно использовать функции задержки, например, delay() в Kotlin Coroutines, чтобы обновлять значение прогресса с интервалом. В классическом Android можно использовать Handler или таймеры для периодического обновления прогресса.
   4. Цвет ProgressBar можно изменить, применяя цветовые фильтры к drawable или через стили. В Jetpack Compose цвет задаётся параметром color у LinearProgressIndicator или CircularProgressIndicator.
   5. BadgedBox – это компонент Jetpack Compose, который позволяет добавить бейдж поверх другого UI-элемента, например, иконки.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс изменения интерфейса для информирования пользователей в приложениях Android.

# Лабораторная работа №23

**Настройка тем**

1. **Цель работы**
   1. Изучить процесс изменения интерфейса для информирования пользователей в приложениях Android.
2. **Ответы на контрольные вопросы** 
   1. MaterialTheme – это центральный компонент в Jetpack Compose, который задаёт визуальную тему приложения в стиле Material Design.
   2. Палитра цветов настраивается созданием объектов ColorPalette, в которых задаются основные цвета приложения.
   3. Типографика настраивается через объект Typography, в котором задаются стили текста – шрифты, размеры, веса и другие параметры для различных текстовых элементов.
   4. Для переключения между светлой и тёмной темами создаются отдельные цветовые палитры. В функции темы Theme.kt в зависимости от параметра darkTheme выбирается соответствующая палитра и передаётся в MaterialTheme.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс изменения интерфейса для информирования пользователей в приложениях Android.

# Лабораторная работа №24

**Настройка анимации**

1. **Цель работы**
   1. Изучить процесс настройки анимации интерфейса приложений Android.
2. **Ответы на контрольные вопросы** 
   1. Анимация в приложениях используется для улучшения пользовательского опыта – она делает интерфейс более живым, помогает визуально подчеркнуть изменения состояния, направляет внимание пользователя и делает взаимодействие с приложением более интуитивным и приятным.
   2. В Jetpack Compose анимация делится на высокоуровневые анимации и низкоуровневые.
   3. Функции animate\*AsState — это простейшие API для анимации одного значения. Вы задаёте конечное (целевое) значение, и анимация плавно изменяет текущее значение до указанного.
   4. updateTransition используется для создания анимаций перехода между состояниями. Он позволяет одновременно анимировать несколько свойств, связанных с изменением состояния, и управлять их синхронизацией.
   5. InfiniteTransition используется для бесконечных повторяющихся анимаций, например, для индикаторов загрузки или вращающихся элементов.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс настройки анимации интерфейса приложений Android.

# Лабораторная работа №25

**Работа с файловой системой**

1. **Цель работы**
   1. Изучить процесс работы с файлами в приложениях Android.
2. **Ответы на контрольные вопросы** 
   1. Чтобы создать папку assets нужно в панели проекта нажать правой кнопкой на каталог app, выбрать New → Directory или New → Folder → Assets Folder.
   2. Для открытия файла из папки assets используется метод getAssets().open(filename), который возвращает поток InputStream.
   3. Путь к папке приложения можно получить через контекст.
   4. Данные из файла можно считать, использовав BufferedReader для построчного чтения.
   5. Для записи данных в файл используется внутренний каталог приложения (filesDir).
   6. Для выбора файла обычно используется стандартный Intent с действием ACTION\_OPEN\_DOCUMENT.
   7. Для сохранения файла можно использовать Intent с действием ACTION\_CREATE\_DOCUMENT.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс работы с файлами в приложениях Android.

# Лабораторная работа №26

**Подготовка стандартных модулей**

1. **Цель работы**
   1. Изучить процесс создания стандартных модулей в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. Модульный подход применяется для разбивки большого приложения на отдельные, слабо связанные модули.
   2. data class – это специальный класс в Kotlin, предназначенный для хранения данных.
   3. MVVM (Model-View-ViewModel) – это архитектурный паттерн, разделяющий логику приложения на три компонента:

* Model – слой данных и бизнес-логики;
* View – пользовательский интерфейс, отображающий данные;
* ViewModel – посредник между Model и View, который предоставляет данные для отображения и обрабатывает пользовательские действия.

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс создания стандартных модулей в приложении Android.

# Лабораторная работа №27

**Передача данных между модулями**

1. **Цель работы**
   1. Изучить процесс передачи данных между модулями в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. Параметры между модулями передаются через простые типы или идентификаторы.
   2. Для передачи простого значения (например, строки или числа) через навигацию в Compose Navigation используют параметры маршрута.
   3. Чтобы передать объект через навигацию в Jetpack Compose, можно сериализовать объект в строку и передать эту строку как параметр маршрута.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс передачи данных между модулями в приложении Android.

# Лабораторная работа №28

**Работа с БД**

1. **Цель работы**
   1. Изучить процесс работы с БД в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. Для создания базы данных SQLite в приложении Android используется класс SQLiteOpenHelper. Необходимо создать собственный класс, наследующий SQLiteOpenHelper, и переопределить в нём метод onCreate() . Внутри onCreate() выполняется SQL-запрос CREATE TABLE для создания нужных таблиц .
   2. Для получения списка строк из таблицы используются методы query() или rawQuery() класса SQLiteDatabase.
   3. Для добавления новой строки в таблицу БД используется операция insert() . Для этого необходимо создать объект ContentValues, в который помещаются пары "ключ-значение", где ключ – это имя столбца, а значение – данные для этого столбца.
   4. Для изменения существующих записей в таблице БД SQLite используется операция update().
   5. Для удаления строки из таблицы БД SQLite используется операция delete().
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс работы с БД в приложении Android.

# Лабораторная работа №29

**Работа с API**

1. **Цель работы**
   1. Изучить процесс работы с API в приложении Android.
2. **Ответы на контрольные вопросы** 
   1. REST (Representational State Transfer) – это архитектурный стиль взаимодействия между клиентом и сервером через HTTP.
   2. Стандартные HTTP-методы, используемые в REST, соответствуют операциям CRUD: GET, POST, PUT, DELETE, PATCH.
   3. Данные, возвращаемые REST API, могут быть в различных форматах. Наиболее распространённые форматы – это JSON, XML и HTML.
   4. Библиотека Retrofit2 – это мощный HTTP-клиент для Android, который упрощает взаимодействие с REST API.
   5. Для разрешения доступа к интернету в Android-приложении в файле AndroidManifest.xml нужно добавить следующую строку внутри тега <manifest>:

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

1. **Вывод**
   1. В ходе лабораторной работы был изучен процесс работы с API в приложении Android.

# Лабораторная работа №30

**Тестирование и оптимизация мобильного приложения**

1. **Цель работы**
   1. Изучить процесс тестирования и оптимизации приложений Android.
2. **Ответы на контрольные вопросы** 
   1. Модульное тестирование – это тестирование отдельных, изолированных частей кода на корректность их работы в искусственно созданной среде.
   2. Модульные тесты в Android-проектах обычно размещаются в папке src/test/java/.
   3. Точку останова можно добавить в среде разработки Android Studio, кликнув левой кнопкой мыши слева от номера строки кода.
   4. Для запуска приложения в режиме отладки в Android Studio нужно нажать кнопку Debug вместо обычного запуска.
3. **Вывод**
   1. В ходе лабораторной работы был изучен процесс тестирования и оптимизации приложений Android.