ЛАБОРАТОРНА РОБОТА №1
НАЛАШТУВАННЯ СЕРЕДОВИЩА РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ ОС ANDROID

1.1 Мета роботи

Налаштувати середовище розробки додатків для операційної системи Android. Ознайомитись з типовою структурою Android-проекту. Ознайомитись з системою автоматичного збирання проектів Gradle. Навчитись розробляти прості Android-додатки з використанням стандартної бібліотеки візуальних компонентів.

1.2 Короткі теоретичні відомості

Станом на 2019 рік основним інструментом розробки додатків для операційної системи Android є IDE Android Studio – офіційний продукт компанії Google Inc., розроблений на базі платформи IntelliJ (IntelliJ Platform), який є безкоштовним та вільним у використанні.

IDE Android Studio потіснила раніше популярний плагін ADT для Eclipse, останній нині майже не використовується при розробці проектів. Слід зазначити, що використання Android Studio не є обов'язковим. Для розробки Android-додатків достатньо мати встановлені Android SDK та інструменти командного рядка Android SDK Tools. Проекти можна генерувати за допомогою утиліти `tools/android` або зконфігурувати вручну за допомогою системи збирання проектів Gradle, а вихідні коди теоретично можливо писати навіть в блокноті.

Нативні Android-додатки пишуться на мовах Kotlin, Java (та опціонально іноді на C++). До недавніх пір (до 2017 року) єдиною офіційною мовою програмування для Android-проектів, яку підтримувала компанія Google, була Java. Починаючи з 2017 року, з випуском Android Studio 3.0, мова програмування Kotlin прийшла на заміну Java. Зараз підтримуються обидві мови програмування, але з плином часу все більше проектів переводиться на мову Kotlin.

В даних методичних вказівках всі приклади в підрозділах теоретичних відомостей наведені на обох мовах Java та Kotlin. У підрозділах виконання робіт приклади наведені на мові Java, однак за бажанням їх можна конвертувати у мову Kotlin, використовуючи вбудований механізм конвертації Android Studio.

Студент має право виконувати лабораторні роботи на будь-якій мові (Java/Kotlin) за власним бажанням.

1.2.1 Мова програмування Kotlin

Курс даної дисципліни базується на тому факті, що студенти вже проходили курс ООП та Java на попередніх курсах. Тому опис основ об'єктно-орієнтованого програмування та мови Java в даних методичних вказівках не наводиться. Однак це не стосується відносно нової мови програмування Kotlin, яка не включена до переліку обов'язкового вивчення на попередніх курсах. Тому в даному розділі наводиться її короткий опис. Більш детальну інформацію можна знайти на офіційному сайті за посиланням [1].

Особливості Kotlin:

- Повна сумісність с Java (JDK 6). Додатки, написані на Kotlin гарантовано будуть працювати без будь-яких проблем на всіх пристроях під управлінням ОС Android. Бібліотеки, написані на мові Java, можна вільно використовувати в вихідних кодах на Kotlin, більш того, в одному проекті частина коду може бути написана на Java, частина – на Kotlin. Всі класи, методи, написані на Java, вільно доступні з коду Kotlin;
- Ефективний процес збирання проектів – підтримується інкрементальна компіляція, яка зменшує час, необхідний для внесення змін у проект та для його запуску;
- Компактність стандартної бібліотеки – менше 100 Кб за розміром;
- Швидкість роботи – додаток, написаний на мові Kotlin працює з такою ж швидкістю, як і додаток, написаний на мові Java.

Далі наведено базовий синтаксис мови Kotlin.

1) Пакети

Як і Java, Kotlin підтримує групування класів по пакетам. Однак є дві важливі відмінності:

- в Kotlin пакет не обов'язково має співпадати зі структурою каталогів проекту. Наприклад, в Java деякий клас MyClass, описаний в файлі $SRC/ua/cn/stu/MyClass.java, обов'язково буде відноситись до пакету ua.cn.stu. У випадку ж з Kotlin, даний клас може відноситись до будь-якого пакету. Пакет класу задається на початку файлу наступним чином:

```
package <package_name>
package ua.cn.stu // example
```

- друга відмінність полягає у тому, що на верхньому рівні ієрархії пакету Kotlin можуть міститись не тільки класи, а й методи, змінні та константи. Тобто цілком реально написати файл hello.kt з наступним вмістом:

```
package ua.cn.stu

const val number = 10
```

```
var name = ""

fun hello() {
  print("Hello, world!")
}

hello()
```

2) Імпорти аналогічні Java, з невеликими відмінностями

```
// single class
import ua.cn.stu.AnotherOneClass
// all things from the package
import ua.cn.stu.utils.*
// import alias
import ua.cn.stu.accounts.entities.AccountRecord as Account
```

3) Основні типи в Kotlin та їх аналоги в Java:

Kotlin Type	Value description	Java Type
Byte	[-128;127]	byte
Short	[-32768;32767]	short
Int	$[-2^{31};2^{31}-1]$	int
Long	$[-2^{63};2^{63}-1]$	long
Char	single character	char
Boolean	true/false	boolean
Float	32bit floating point number (IEEE 754)	float
Double	64bit floating point number (IEEE 754)	double

4) Створення змінних відбувається за допомогою ключових слів val або var

```
val intNumber = 1                    // Int
var longNumber = 1000L               // Long
val longNumber2 = 100000000          // Long
var floatNumber = 1.0f               // Float
val doubleNumber = 1.0               // Double
var flag = true                      // Boolean
val character = 'a'                  // Char
var name = 'Ivan'                    // String
```

Як видно із прикладів вище, на відміну від Java, в Kotlin не обов'язково вказувати тип змінної, якщо він може бути визначений автоматично з літералів.

Відмінність val від var полягає у тому, що значення змінної, позначеної ключовим словом val не може бути змінено, тобто по суті val є деяким аналогом ключового слова final в Java:

```
val age = 10
age = 12 // error, can't be reassigned
```

```
var editableAge = 10
editableAge = 12 // all ok
```

Також підтримується відкладена ініціалізація для val:

```
val age: Int
…
age = 20
```

Очевидно, що доступ до val-змінної має відбуватись тільки після ініціалізації. Наприклад, тут буде помилка:

```
val age: Int
print(age)      // error
age = 20
```

При відкладеній ініціалізації обов'язково необхідно вказувати тип змінної.

Тип вказується після назви через двокрапку:

```
val studentName: String = "Ivanov"
val mark: Int = 60
```

Ковертація типів відбувається за допомогою методів toByte(), toShort(), toInt(), toLong(), toFloat(), toDouble(), toChar(), toString():

```
val input = "25" // String
val age = input.toInt() // Int
val longAge = age.toLong() // Long
```

Для типу Boolean підтримуються наступні операції: && (and), || (or), ! (not), тут все аналогічно Java:

```
val confirmed = isConfirmed()
val paid = isPaid()
val hasPromoCode = hasPromoCode()
val ordered = confirmed && (paid || hasPromoCode) && !isBanned()
```

Для бітових операцій, у свою чергу, слід використовувати методи:

- and(bits)
- or(bits)
- xor(bits)
- inv()
- shl(bits)
- shr(bits)
- ushr(bits)

Починаючи з версії Kotlin 1.3, підтримуються також без знакові типи (у статусі experimental): UByte, UInt, UShort, ULong.

5) Текстові рядки

Текстові рядки (тип String) можна ініціалізувати за допомогою багаторядкових літералів наступним чином:

```
val textMessage = """
            Line1
            Line2
            Line3
        """.trimIndent()
```

Останній метод trimIndent() видаляє пусті рядки на початку та в кінці тексту, а також табуляцію та пробіли на початку кожного рядка.

Kotlin має також доволі зручну функцію з назвою string templates, яка надає можливість підставляти в рядки значення змінних або виразів:

```
val age = 19
print("User age: $age")
print("Is user adult: ${age >= 18}")
```

6) Оператори в Kotlin:

- Умовний оператор IF

```
val number = getNumber()
if (number > 0)
  print("Positive")
else {
  print("Non Positive")
}
```

На відміну від Java, оператор IF в Kotlin може повертати значення:

```
val number = getNumber()
val message = if (number > 0) "Positive" else "Non positive"
print(messgae)
```

- Умовний оператор WHEN

Даний оператор є деяким аналогом switch в Java:

```
val mark = getMark()
val message = when (mark) {
  2 -> "Bad"
  3 -> "Not bad"
  4 -> "Good"
  5 -> "Excellent"
  else -> "Unknown mark"
}
print(message)
```

- Оператор циклу FOR

Дозволяє послідовно проходити по елементам колекцій та діапазонах (ranges):

```
val users: List<User> = getUsers()
for (user in users) {
    print("User: ${user.name}")
}

var sum = 0
for (i in 1..10) {
    sum += i
}
println("Sum from 1 to 10 is: $sum")
```

Для діапазонів можна задати напрямок та крок:

```
println("Even numbers from 10 to 1:")
for (i in 10 downTo 1 step 2) {
    println(i)
}
```

Результат:

```
Even numbers from 10 to 1:
10
8
6
4
2
```

- Умовний оператор циклу WHILE

Має два варіанти запису:

1) while(<*expression*>)

2) do { … } while *(*<*expression*>)

Приклад While:

```
val commands: Queue<Runnable> = getCommands()
while (commands.isNotEmpty()) {
    commands.poll().run()
}
```

Приклад Do…While:

```
do {
    val command: String = askForCommand()
    parseAndExecute(command)
} while ("exit" != command)
```

7) Smart Cast

В Kotlin підтримується також функція **Smart Cast**, тобто тип змінної може бути визначено в умовних виразах з автоматичним приведенням типу. Наприклад, порівняйте код в Java та Kotlin:

Kotlin:

```
val object: Any = getVariable();
val length = if (object is String) {
  object.length // тип визначено як String
} else -1
```

Java:

```
Object object = getVariable();
final int length;
if (object instanceof String) {
  // необхідно вручну привести тип для доступу до методу length()
  length = ((String) object).length();
} else {
  length = -1
}
```

8) Nullable змінні

На відміну від Java, за замовчуванням всі типи в Kotlin не можуть приймати значення NULL:

```
val student: Student = null; // помилка
```

Більш того, рекомендується взагалі не використовувати NULL в Kotlin без явної потреби, оскільки це зменшує безпеку виконання коду. Якщо ж все-таки необхідно надати можливість деякій змінній приймати значення NULL, то для цього треба позначити її відповідним Nullable-типом. Nullable-тип – це такий тип, який окрім звичайних значень може також приймати значення NULL. Nullable-тип можна отримати з будь-якого іншого типу в Kotlin шляхом додавання знаку «?» до назви типу. Наприклад:

```
var name: String? = null
var age: Int? = null
```

Варто зазначити, що доступ до полів та методів Nullable-змінних має відбуватись або за допомогою Smart Cast (див. п.п. 7), або за допомогою операторів safe call («?.»), null assertion («!!.»):

```
var name: String? = null
println(name.length) // помилка компіляції
if (name != null) println(name.length) // smart cast, ok
println(name?.length) // output: null, ok
println(name?.length ?: 0) // output: 0, ok
println(name!!.length) // NullPointerException, помилка
```

Оператор безпечного виклику (safe call «?.») повертає значення NULL та перериває виконання виразу у разі, якщо хоча б один із операндів є NULL. Наприклад:

```
val node: Node? = getNode()
```

```
val parentName = node?.getParent()?.getName()
```

Змінна parentName буде мати тип «String?» та значення NULL, якщо або змінна node=null, або метод getParent() повернув null, або метод getName() повернув null. Подібний код в Java викинув би NullPointerException та програма аварійно завершилася б, якщо node або getParent() повертали значення null:

```
Node node = getNullNode()
String name = node.getParent().getName() // NullPointerException
```

В Kotlin, при великому бажанні, також можна досягти подібного ефекту за допомогою оператору Null Assertion:

```
val node: Node? = getNode()
val name = node!!.getParent()!!.getName()
```

9) Lazy initialization

Відкладена ініціалізація дозволяє не ініціалізувати змінну значенням відразу, а зробити це пізніше. В розробці Android-додатків ця функція використовується доволі часто, оскільки ініціалізація основних компонентів відбувається не в конструкторах, а в методах життєвого циклу (наприклад onCreate). Об'єкти компонентів створюються системою, а не користувачем. Наприклад, ініціалізація компоненту Activity, завдання якого відображати графічний інтерфейс користувача, відбувається наступним чином:

```
lateinit var signInCredentials: Credentials

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_login)

    signInCredentials = savedInstanceState
        ?.getParcelable(KEY_CREDENTIALS)
        ?: Credentials()
}
```

Забігаючи наперед, тут при створенні компоненту Activity відбувається перевірка на те, чи є закешовані дані (наприклад, користувач почав вводити дані, але потів вирішив відволіктись на вхідний дзвінок, і лише через деякий час знову повернувся до додатку). Якщо такі дані існують, то вони заносяться в змінну signInCredentials, інакше – змінна ініціалізується новим пустим об'єктом типу Credentials.

10) Об'єктно-орієнтовне програмування в Kotlin

Перше, що необхідно запам'ятати, в Kotlin відсутні static змінні та методи. Замість них використовуються або глобальне поле, або об'єкти-компаньйони. Наприклад:

```
// global declaration
const val KEY_CREDENTIALS = "CREDENTIALS"
class LoginActivity : BaseActivity() {
    …
    credentials = getByKey(KEY_CREDENTIALS)
    …
}

// companion object
class LoginActivity : BaseActivity() {
    companion object {
        const val KEY_CREDENTIALS = "CREDENTIALS"
    }
    …
    credentials = getByKey(LoginActivity.KEY_CREDENTIALS)
}
```

Класи мають два види конструкторів: один основний (primary) та опціональні додаткові (secondary), а також блоки ініціалізації:

```
class Citizen( // primary constructor
    val name: String,
    val surname: String,
    val age: Int = 0,
    internalId: Long = -1
) : Serializable {

    // initialization block
    init {
        println("Object has been created")
    }

    // secondary constructor: calls primary constructor
    // via this keyword
    constructor(parent: Citizen, internalId: Long) : this(
        parent.name,
        parent.surname,
        parent.age,
        internalId
    ) {
        // some additional initialization here
    }

}
…
val citizen1 = Citizen("Ivan", "Ivanov", 20, 1)
val citizen2 = Citizen("Petro", "Petrov", 22)
val citizen3 = Citizen(citizen1)
val citizen4 = Citizen(citizen3, 42)
val luke = Citizen(surname = "Skywalker", age = 5, name = "Luke")
```

Якщо не вказувати основний конструктор, то буде створений автоматично пустий конструктор. Додаткові конструктори обов’язково мають викликати основний конструктор або інший додатковий конструктор за допомогою ключового слова **this**. Виключення – цього можна не робити,

якщо основний конструктор є пустим. В даному випадку від буде викликатись автоматично.

Як видно з прикладу, конструктори підтримують значення за замовчуванням, а також довільний порядок аргументів. Це твердження справедливе не тільки для конструкторів, а й для звичайних методів.

Порядок ініціалізації об'єкту наступний:

1) Основний конструктор
2) Додаткові конструктори
3) Блок ініціалізації

Якщо необхідно наслідувати деякий клас, або реалізувати інтерфейси, то необхідно вказати їх після основного конструктора через двокрапку:

```
class Student(
    val university : University,
    name: String,
    surname: String,
    age: Int
) : Citizen(name, surname, age), Educable { … }
```

В даному випадку клас Student наслідує клас Citizen та реалізує інтерфейс Educable.

Слід зазначити, що на відміну від Java, за замовчуванням всі класи є фінальними (final), а отже їх не можна наслідувати. Для того, щоб дозволити наслідування для певного класу, необхідно зробити його відкритим, позначивши ключовим словом open:

```
open class Citizen(
    val name: String,
    val surname: String,
    val age: Int = 0,
    internalId: Long = -1
) { … }
```

В Kotlin присутні наступні модифікатори доступу:

1) Публічний (public) – доступ з будь-якого модулю, використовується за замовчуванням, якщо не вказано інший модифікатор доступу
2) Внутрішній (internal) – доступ лише всередині модулю.
3) Захищений (protected) – доступ для підкласів, не можна використовувати в глобальній області
4) Приватний (private) – доступ всередині класу (або всередині файлу для глобальної області).

Модифікатори доступу вказуються перед методом/класом/змінною:

```
internal open class Citizen {
    internal var age = 10
    protected fun log() {
        println(this)
    }
}
```

Ще однією відмінністю від Java є наявність властивостей (properties). В Java механізм властивостей реалізовується за допомогою геттерів та сеттерів (getters, setters): геттери надають доступ до властивості, сетери дозволяють її змінювати. Відмінність властивості від звичайної змінної полягає у наявності контролю доступу. В Kotlin є вбудований механізм для таких цілей.

Фактично, коли всередині класу вказується змінна з ключовим словом val або var, то неявно створюється властивість з такою назвою та з геттером і сеттером за замовчуванням (або тільки з геттером за замовчуванням для ключового слова val). Геттери і сеттери можна вказати самостійно:

```
class Citizen(name: String, birthday: String) {

    // custom backing field
    private var date: Date = toDate(birthday)

    // property with custom backing field
    var birthday: String
        get() = DateFormat.getDateInstance().format(date)
        set(value) {
            date = toDate(value)
        }

    // read-only property without backing field
    val age: Int
        get() = calculateAge(date)

    // property with default backing field
    var name: String = name
        set(value) {
            if (value.isNotEmpty()) field = value
        }

    private fun calculateAge(date: Date): Int { ... }

    private fun toDate(s: String): Date
            = DateFormat.getDateInstance().parse(s)
}
```

На самостійне опрацювання пропонується розглянути наступні особливості Kotlin:

1) Data-класи
2) Sealed-класи
3) Extension-методи
4) Generics, reified types
5) Type aliases
6) Delegation pattern
7) Lambdas, inline functions, functional style

Деталі цих та інших особливостей можна знайти за посиланням [1].

1.2.2 Середовище розробника Android-додатків

Для комфортної розробки Android-додатків необхідно встановити на ПК:

1) Open JDK 1.8. Можна використовувати більш нові версії, але рекомендується саме восьма версія. Встановлювати самостійно не обов'язково, комплект Open JDK поставляється разом з Android Studio. Однак для використання інструментів командного рядка необхідно або встановити Open JDK в системі, або зконфігурувати систему на використання JDK саме з поставки Android Studio;

2) Android SDK. Комплект утиліт та бібліотек для розробки Android-додатків. Можна завантажити окремо без встановлення Android Studio IDE;

3) Android Studio IDE. Офіційний інструмент від Google на базі Intellij Platform для розробки Android-додатків.

Також необхідно налаштувати:

1) Віртуальні пристрої Android (AVD);

2) Якщо є реальні пристрої, то ввімкнути на них режим розробника та дозволити відлагодження додатків через USB;

3) Використовуючи SDK Manager з поставки Android SDK, встановити додаткові залежності.

Android SDK містить інструменти командного рядка, наведені нижче:

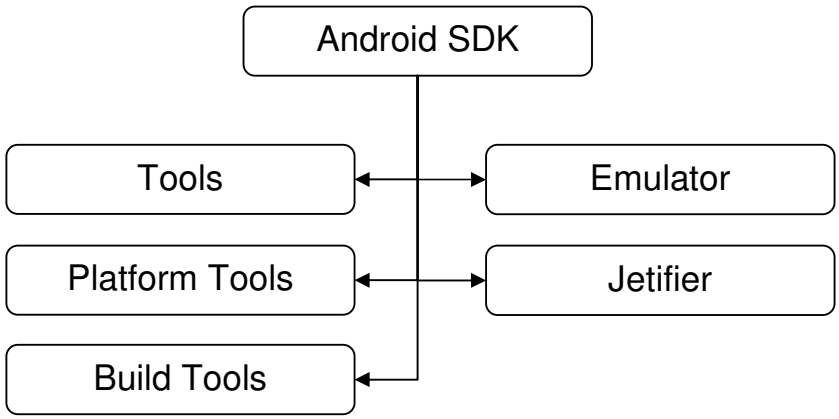

Рисунок 1.1 – Категорії інструментів командного рядка Android SDK

Android SDK Tools – це набір платформо-незалежних інструментів, тобто вони не залежать від версії Android, для якої ви розробляєте додатки. До даного набору належать:

- APK Analyzer – для аналізу результатів збирання проектів;
- AVD Manager – для управління віртуальними пристроями (AVD);
- SDK Manager – для управління пакетами з комплекту Android SDK;
- Lint – для аналізу коду проектів;
- Monkey Runner – для автоматизації встановлення і тестування додатків.

Android SDK Build Tools – це набір інструментів, які виконують збирання проекту з коду. Зазвичай вони не використовуються програмістом, за виключенням AAPT2 (компілює ресурси), Apk Signer (підписує та верифікує APK-файли) та Zip Aligner (проводить оптимізацію APK-файлів).

APK-файл – це пакет зкомпільованого та зібраного проекту Android. Кінцевою ціллю збирання Android-проекту є саме APK-файл. Маючи APK-файл додатку, можна встановити його як на віртуальний, так і на реальний пристрій.

Варто зазначити, що в останній час Google рекомендує для поширення та публікації додатків використовувати інший формат – Android App Bundle. Більш детальніше про Android App Bundle ви можете прочитати за посиланням [2].

Android SDK Platform Tools – набір інструментів, які оновлюються з кожною наступною версією платформи Android. Необхідно обов'язково знати наступні інструменти з даного набору:

- Android Debug Bridge (adb)
- Log Cat (logcat)

Android Debug Bridge дозволяє підключатися до віртуальних та реальних пристроїв, та проводити відлагодження додатків в реальному часі. Також він надає цілий набір різноманітних функцій для управління станом пристрою.

Log Cat – це інструмент перегляду системних та програмних логів, що спрощує відлагодження Android-додатків.

Android Emulator – набір інструментів для управління віртуальними пристроями. Складається з двох програм:

- emulator – дозволяє створювати, видаляти, запускати віртуальні пристрої Android;
- mksdcard – дозволяє створювати образи SD-карт, які розглядаються системою Android як зовнішнє сховище даних.

Jetifier – новий інструмент, який необхідний для міграції старих Android-проектів на AndroidX. AndroidX – це нова система найменування пакетів класів з комплекту Android SDK, розроблена та запропонована компанією Google в 2018 році.

1.2.3 Структура Android-проекту

Проект Android-додатку складається з одного або декількох модулів. Кожен модуль має свій підкаталог в каталозі проекту, назва каталогу співпадає з назвою модуля. Існує декілька типів модулів, основні з яких: Android Application, Android Library, Java Library. Хоча останній тип модулів і називається «Java Library», фактично він може бути написаний на мові Kotlin, оскільки Kotlin повністю сумісний з Java.

Насправді, різниця між типами модулів досить умовна та задається конфігурацією збирання в системі Gradle. Цілком можливо створити модуль

зі своєю власною конфігурацією, змінивши відповідним чином файл конфігурації Gradle.

Отже, типова структура проекту:

```
module1/
  src/
    main/
      java/           # source code
      res/            # app resources
      AndroidManifest.xml        # top level app structure
    test/             # unit tests
    androidTest/      # instrumented tests
  build.gradle        # build configuration of the module
  proguard-rules.pro  # ProGuard config
module2/
  …
gradle/
  …
gradle.properties     # build environment
local.properties      # properties valid only for current env
build.gradle          # top level build configuration
gradlew               # Gradle Wrapper for building app
settings.gradle       # the list of project modules
```

За замовчуванням проект складається з одного модуля з назвою app типу Android Application. Тип Android Application означає, що результатом збирання даного модуля буде APK-файл, який можна встановити на пристрій. Це – найбільш поширений тип модулів.

Далі пройдемося по файлам проекту:

1) gradle.properties – містить перелік параметрів збирання проекту, які зчитуються системою Gradle. Це можуть бути, наприклад, ліміти пам'яті, конфігурація багато поточності для більш швидкої компіляції, логування процесу збирання проекту і т.д. Також тут можна вказати і свої змінні, які потім можуть зчитуватись конфігурацією, заданою в build.gradle.

2) local.properties – містить параметри збирання проекту, актуальні лише для даного користувача даного ПК. Створюється автоматично Android Studio, яка зберігає до цього файлу шлях до Android SDK та Android NDK. Даний файл не є частиною проекту та не повинен зберігатися в системах контролю версій.

3) build.gradle – містить інформацію, яка актуальна для всіх модулів проекту, наприклад – шляхи до репозиторіїв залежностей, версії бібліотек та ін.

4) gradlew – Gradle Wrapper, скрипт (формату Bash для Linux, BAT для Windows), інструмент командного рядка, який дозволяє збирати проект вручну. Також надає можливість запускати тести, перевіряти код проекту [3].

5) settings.gradle – містить перелік модулів проекту.

6) <module>/build.gradle – конфігурація збирання модуля. Один з найважливіших файлів. Саме в ньому містить вся основна інформація та алгоритм збирання проекту. Тип модуля задається саме цим файлом. Як і всі

Gradle-файли, написаний на мові Groovy. Більш детально конфігурацію Gradle та саму систему збирання проектів Gradle розглянемо в підрозділі 1.2.4.

7) <module>/src/test – unit-тести проекту, запускаються на машині розробника, а не на пристрої Android.

8) <module>/src/androidTest – інструментальні тести, виконується або на реальному, або на віртуальному пристрої.

9) <module>/src/main – ресурси та вихідні коди проекту.

1.2.4 Система збирання проектів Gradle

Для збирання проектів Android використовуються система Gradle (аналог Maven). На відміну від Maven, конфігураційні скрипти пишуться на мові Groovy. Важливою особливістю Gradle є те, що в якості репозиторіїв залежностей Gradle використовує ті ж самі репозиторії, що і Maven. Це означає, що, наприклад, бібліотеки з Maven Central [4] можна вільно та просто підключати до проекту під управлінням Gradle.

Розглянемо типовий скрипт збирання Android-проекту:

Root build.gradle містить конфігурацію для всіх модулів:

```
buildscript {
    ext.kotlin_version = '1.3.50' // версія Kotlin
    // аналогічно тут можна задати свої змінні версій для інших
    // бібліотек:
    ext.mylib_version = '1.0.1'

    // список репозиторіїв, в яких буде відбуватись пошук
    // залежностей для підключення плагінів Gradle
    repositories {
        google()  // https://dl.google.com/dl/android/maven2
        jcenter() // https://jcenter.bintray.com
    }
    // список залежностей з репозиторіїв вище,
    // конкретно тут підключаються Android-плагін для
    // збирання проекту для ОС Android та плагін для підтримки
    // мови Kotlin
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.3'
        classpath "org.jetbrains.kotlin:
                    kotlin-gradle-plugin:$kotlin_version"
    }
}

allprojects {
    // список репозиторіїв для всіх модулів, в яких
    // проводиться пошук залежностей
    repositories {
        google()
        jcenter()
    }
}
```

```
// при виконанні задачі clean видаляємо каталог build:
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Кожен модуль містить свій скрипт build.gradle, який безпосередньо використовується для збирання модуля. Це – основний файл конфігурації збирання, який найчастіше модифікується при розробці Android-додатків серед всіх інших Gradle-файлів:

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
  compileSdkVersion 28
  // конфігурація за замовчуванням для всіх варіантів
  // збирання проекту
  defaultConfig {
    applicationId "stu.cn.ua.hello"
    minSdkVersion 21
    targetSdkVersion 28
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner
             "androidx.test.runner.AndroidJUnitRunner"
  }
  buildTypes {
    // додаткова конфігурація для конкретних варіантів збирання;
    // перезаписує/розширює конфігурацію за замовчуванням
    release {
      minifyEnabled false
      proguardFiles getDefaultProguardFile
        ('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
  }
}

// перелік залежностей проекту (бібліотеки, інші модулі)
dependencies {
  implementation fileTree(dir: 'libs', include: ['*.jar'])
  implementation "org.jetbrains.kotlin:
                  kotlin-stdlib-jdk7:$kotlin_version"
  implementation 'androidx.appcompat:appcompat:1.1.0'
  implementation 'androidx.core:core-ktx:1.1.0'
  implementation 'androidx.constraintlayout:constraintlayout:
                  1.1.3'
  testImplementation 'junit:junit:4.12'
  androidTestImplementation 'androidx.test.ext:junit:1.1.1'
  androidTestImplementation 'androidx.test.espresso:
                             espresso-core:3.2.0'
}
```

За замовчуванням існує два варіанти збирання проекту Android (за бажанням можна додати свої варіанти): debug та release. Перший варіант

використовується при розробці для від лагодження проекту. Другий варіант збирає оптимізовану версію додатку, який буде готовим до публікації.

Якщо на локальній машині встановлена Java (або налаштування змінені таким чином, щоб використовувати JDK з поставки Android Studio), то збирати проект можна за допомогою Gradle Wrapper – спеціального інструменту командного рядка (gradlew). Наприклад:

```
./gradlew assembleDebug   // зібрати debug-версію
./gradlew assembleRelease // зібрати release-версію
./gradlew test            // виконати unit-тести
```

Результуючий файл має бути підписаний ключем розробника. Тому перед виконанням команди assembleRelease необхідно налаштувати конфігурацію для автоматичного підпису. Процес підпису детально пояснюється на сторінці офіційної документації [5].

1.2.5 Проектування інтерфейсу користувача

Інтерфейс користувача можна реалізовувати як програмно (за допомогою коду), так і за допомогою декларативних XML-файлів (Layouts). Другий спосіб, очевидно, більш зручний. За відображення інтерфейсу відповідає програмний компонент Activity, якому можна призначити макет (layout), який необхідно відобразити користувачеві.

Android SDK містить набір стандартних UI-компонентів, таких як кнопки, зображення, списки та ін. Також окрім UI-компонентів, для полегшення розробки, програмістові надаються компонувальники інтерфейсу, за допомогою яких задається політика розміщення елементів на екрані.

Для кожного компонента інтерфейсу можна задавати безліч атрибутів. Варто мати на увазі, що умовно всі атрибути можна розділити на дві частини: атрибути компонента та атрибути компонувальника. Розрізнити їх дуже просто: всі атрибути компонувальника починаються з префіксу «layout_». Різниця між цими видами атрибутів полягає у тому, що атрибути компонента задають поведінку самого компонента, а атрибути компонувальника використовуються, очевидно, компонувальником для позиціонування компонента на екрані.

Базові компонувальники інтерфейсу:

- FrameLayout
- LinearLayout
- RelativeLayout
- ConstraintLayout
- TableLayout

Розглянемо кожен к них більш детально:

FrameLayout – найпростіший компонувальник. Він може розміщувати в собі елементи інтерфейсу по центру або з деякого краю. Де саме розміщуються елементи, визначає атрибут «gravity»:

```
<FrameLayout
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:padding="16dp">

    <View
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:background="#ff0000"
        android:layout_gravity="start|top"/>

    <View
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:background="#00ff00"
        android:layout_gravity="center"/>

    <View
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:background="#0000ff"
        android:layout_gravity="end|bottom"/>

</FrameLayout>
```

Результат компонування за допомогою FrameLayout наведеного прикладу:

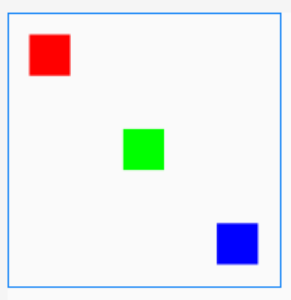

Рисунок 1.2 - FrameLayout

В наведеному прикладі атрибут android:background – це атрибут самого компонента, він задає його колір. Компонент сам зчитує цей атрибут та рисує на екрані відповідний колір. Атрибути ж android:layout_weight, android:layout_height, android:layout_gravity – це атрибути компонувальника (починаються з префіксу «layout_»). Вони використовуються компонувальником, в якому знаходяться компоненти, тобто в даному випадку – FrameLayout, зчитує ці атрибути, визначає місце та розмір компонентів на екрані.

Компоненти всередині FrameLayout можуть накладатися один на одного. Компонент, який об'явлений нижче за текстом макету, буде знаходитись над компонентами, які об'явлені вверху макету.

LinearLayout – дозволяє розміщувати компоненти в лінію, по вертикалі або по горизонталі. За замовчуванням використовується розміщення по горизонталі. Поряд з цим, він також надає можливість задати «вагу» кожного компоненту відносно інших. У разі, якщо компоненти не влазять в контейнер, то вони обрізуються краями компонувальники.

Для кожного компоненту, як і в випадку з FrameLayout, можна задати атрибут «layout_gravity», який визначає зміщення компоненту по осі, перпендикулярній напряму компонувальника.

Наприклад:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="70dp"
    android:padding="16dp"
    android:orientation="horizontal">

    <View
        android:layout_width="50dp"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="#ff0000" />

    <View
        android:layout_width="50dp"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="#00ff00"/>

    <View
        android:layout_width="50dp"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="#0000ff"/>

    <View
        android:layout_width="match_parent"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="#00ffff"/>

</LinearLayout>
```

Результат:

Рисунок 1.3 - LinearLayout

RelativeLayout – дозволяє позиціонувати компоненти один відносно одного, надає безліч атрибутів для цих цілей:

- layout_toStartOf, layout_toEndOf, layout_above, layout_below – розмістити компонент поряд зі вказаним в атрибуті (без накладення);
- layout_alignStart, layout_alignEnd, layout_alignTop, layout_alignBottom – вирівняти компонент по відповідному краю іншого компонента;
- layout_centerHorizontal, layout_centerVertical, layout_centerInParent – вирівняти компонент по центру компонувальника;
- layoutAlignParentXxx – вирівняти по відповідному краю компонувальника.

В якості значень наведених вище атрибутів приймаються ідентифікатори компонентів. Ідентифікатори задаються за допомогою атрибуту «id».

Даний компонувальник є доволі зручним при побудові складних інтерфейсів і дозволяє зменшити кількість рівнів компонування, що в свою чергу веде до більш швидкої прорисовки інтерфейсу.

Наприклад:

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="horizontal"
    android:stretchColumns="*">

    <View
        android:id="@+id/red_square"
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:background="#ff0000"
        android:layout_centerInParent="true"/>

    <View
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:background="#880000"
        android:layout_marginEnd="4dp"
        android:layout_toStartOf="@id/red_square"
        android:layout_centerVertical="true"/>

    <View
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:background="#880000"
        android:layout_marginStart="4dp"
        android:layout_toEndOf="@id/red_square"
        android:layout_centerVertical="true"/>

    <View
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:background="#880000"
        android:layout_above="@id/red_square"
```

```
        android:layout_marginBottom="4dp"
        android:layout_centerHorizontal="true"/>

    <View
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:background="#880000"
        android:layout_below="@id/red_square"
        android:layout_marginTop="4dp"
        android:layout_centerHorizontal="true"/>

</RelativeLayout>
```

Результат:

Рисунок 1.4 - RelativeLayout

TableLayout – дозволяє розміщувати компоненти в табличному вигляді: рядками та колонками, при чому клітинки можуть займати декілька колонок, або бути пустими.

Рядки задаються вкладеним компонувальником з назвою TableRow.

Наприклад:

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:stretchColumns="*">

    <TableRow>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#ff0000"
            android:layout_margin="4dp"/>

        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#ff0000"
            android:layout_margin="4dp"
            android:layout_span="2"/>
    </TableRow>
```

```
    <TableRow>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#00ff00"
            android:layout_margin="4dp" />
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#00ff00"
            android:layout_margin="4dp"/>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#00ff00"
            android:layout_margin="4dp"/>
    </TableRow>

    <TableRow>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#0000ff"
            android:layout_margin="4dp"
            android:layout_column="0"/>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#0000ff"
            android:layout_margin="4dp"
            android:layout_column="2"/>
    </TableRow>
    <TableRow>
        <View
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:background="#ff00ff"
            android:layout_margin="4dp"
            android:layout_span="3"/>

    </TableRow>

</TableLayout>
```

Результат:

Рисунок 1.5 - TableLayout

ConstraintLayout – дуже гнучкий компонувальник, який є продвинутим аналогом RelativeLayout та в принципі може замінювати собою всі інші компонувальники. Рекомендується по можливості використовувати саме цей компонувальник, оскільки він дозволяє реалізовувати складні макети без використання вкладених компонувальників, що в свою чергу зменшує час відрисовки.

Даний компонувальник є доволі складним на перший погляд із-за великої кількості атрибутів. Його можливості:

- розташування компонентів один відносно іншого;
- створення динамічних бар'єрів;
- використання відносних розмірів (у відсотках);
- пропорційне позиціонування;
- створення ланцюжків компонентів та ін.

Розташування компонентів один відносно іншого відбувається за допомогою атрибутів:

- layout_constraintStart_toStartOf
- layout_constraintStart_toEndOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottomToBottomOf

Всі ці атрибути окрім ідентифікатора компоненту можуть приймати ключову слово «parent», яке є посиланням на область компонувальника (контейнер). Якщо вказати два протилежні атрибути, наприклад constraintStart_toStartOf та constraintEnd_toEndOf, то є два випадки:

1) Розмір компонента заданий як рівний нулю (в даному випадку layout_width = «0dp») – тоді компонент буде розтягнутий на всю ширину поміж компонентами, вказаними в constraintStart_toStartOf та constraintEnd_toEndOf)

2) Розмір компонента вказаний як деяке числове значення відмінне від нуля, або значення «wrap_content» – тоді компонент буде відцентрований поміж двома вказаними компонентами. Положення можна регулювати за допомогою параметру bias (constraintVertical_bias, constraintHorizontal_bias), який приймає значення від 0 до 1. За замовчуванням приймає значення 0.5, що і відповідає центру. Вказавши значення 0, компонент буде зміщений до крайнього початкового положення; а значення 1 – навпаки, означає зміщення до крайнього кінцевого положення.

Для того, щоб розмістити компонент по центру, необхідно вказати 4 атрибути:

- layout_constraintStart_toStartOf="parent"
- layout_constraintEnd_toEndOf="parent"
- layout_constraintTop_toTopOf="parent"
- layout_constraintBottom_toBottomOf="parent"

Оскільки параметри verticalBias та horizontalBias за замовчуванням рівні 0.5, то компонент буде розміщений по центру відносно контейнера по вертикалі та по горизонталі.

Далі розглянемо таке поняття як **бар'єри**. Бар'єри зазвичай використовують у формах введення даних з декількома полями у табличному вигляді. Бар'єри дозволяють вирівняти компоненти по розміру найбільшого з них, досягаючи таким чином вирівнювання компонентів, аналогічному TableLayout. Отже, бар'єр – це особливий невидимий тип компоненту, який використовується виключно всередині ConstraintLayout, та дозволяє позиціонувати компоненти відносно деякого набору інших компонентів, коли початковий розмір компонентів з набору невідомий.

Наочно це можна зобразити так:

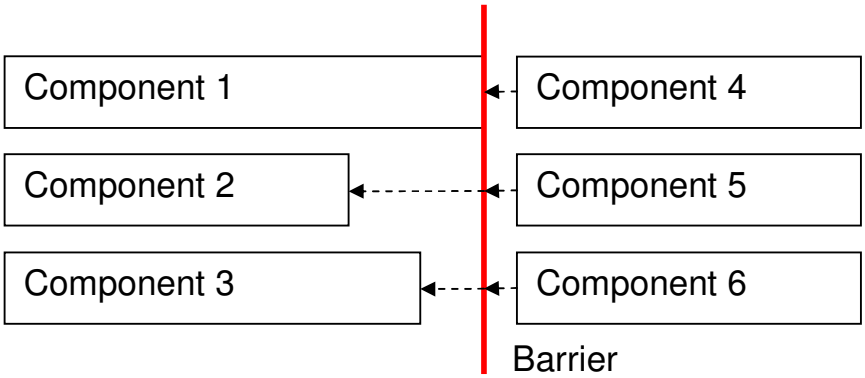

Рисунок 1.6 – Бар'єр в ConstraintLayout

Бар'єр автоматично зміщується до ширині/висоті найбільшого компонента, тобто якщо, наприклад, «Component 2» по ширині стане більший за «Component 1», то бар'єр автоматично зміститься вправо, і в результаті всі компоненти 4,5 та 6, які вирівняні по бар'єру, також змістяться вправо.

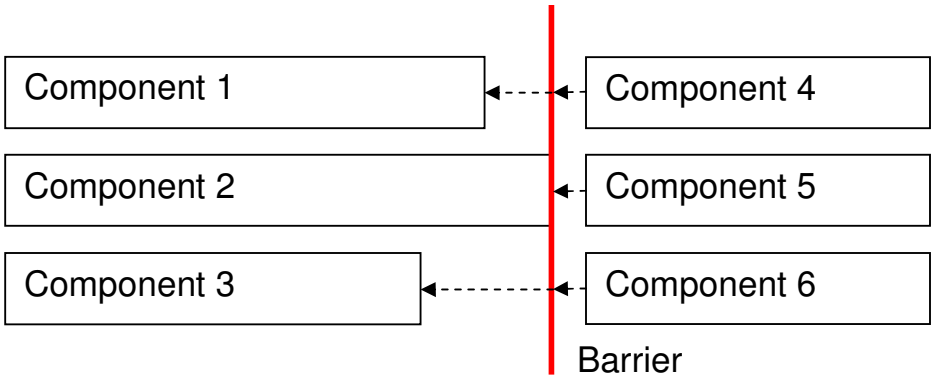

Рисунок 1.7 – Принцип роботи бар'єру

Компонентам також можна задавати відносні розміри у відсотках. Відсоток розраховується відносно розміру компонувальника, в якому знаходяться компоненти. За відносні розміри відповідають атрибути:

constraintWidth_percent, constraintHeight_percent; якщо ці атрибути вказані, то стандартні атрибути weight та height ігноруються.

Майте на увазі, що екрани смартфонів та планшетів не квадратні. Тому задаючи однакові значення для constraintWidth_percent та constraintHeight_percent в результаті маємо не однакові висоту та ширину компоненту:

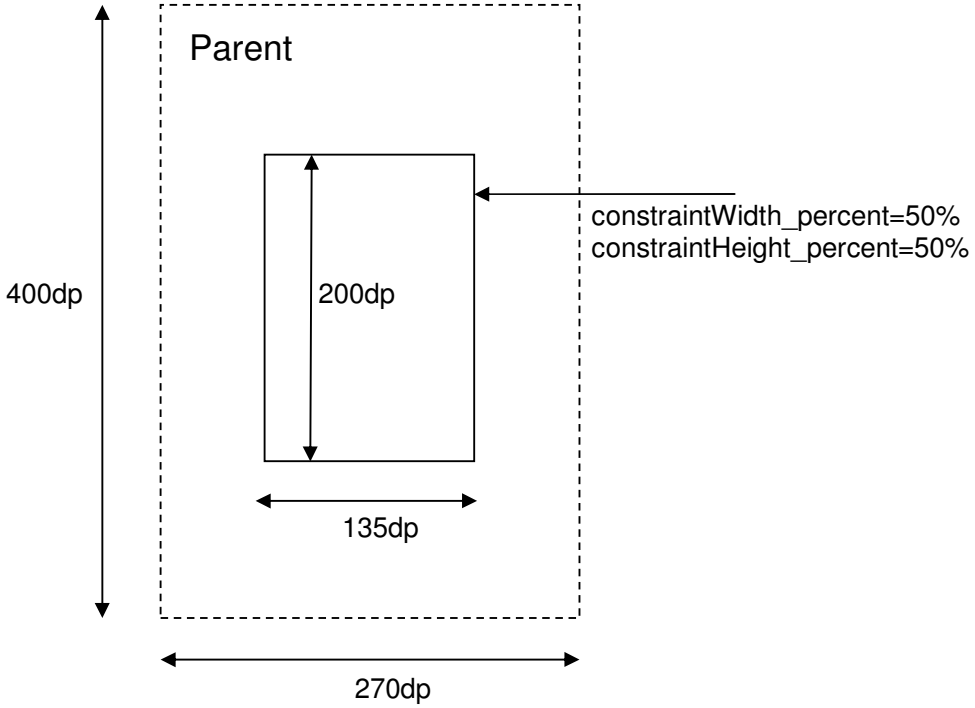

Рисунок 1.8 – Встановлення відносних розмірів

Для того, щоб обійти цю проблему, компоненту можна вказати інші атрибути: constraintDimensionRatio. Даний атрибут може приймати значення в двох форматах:

1) Числове значення, яке задає відношення широти до висоти. Для того, щоб зробити ширину та висоту однаковою,необхідно задати значення «1»;

2) Пропорція у вигляді «X:Y», де X – ширина, Y – висота. Для пропорції можна через кому вказати префікс «H» або «W», наприклад: «H,1:2», «W,2:1». Префікс задає базу пропорції, тобто, в якості бази буде використовуватись ширина (W) або висота (H).

Ще однією особливістю ConstaintLayout є **ланцюжки** компонентів. Ланцюжки дають можливість вирівнювати декілька компоненти за трьома вбудованими політиками: «packed» (компоненти «прилипають» один до одного), «spread» (компоненти рівномірно заповнюють простір з однаковими проміжками між собою), «spread_inside» (компоненти також рівномірно заповнюють простір, але без пустих країв). Ланцюжок компонентів можна в

наближеному варіанті розглядати як окремий над компонент, з яким теж можна працювати.

Отже, для того, щоб створити ланцюжок, необхідно просто вказати перехресні посилання між компонентами, які входять в ланцюжок. Перший компонент в ланцюжку задає політику ланцюжка:

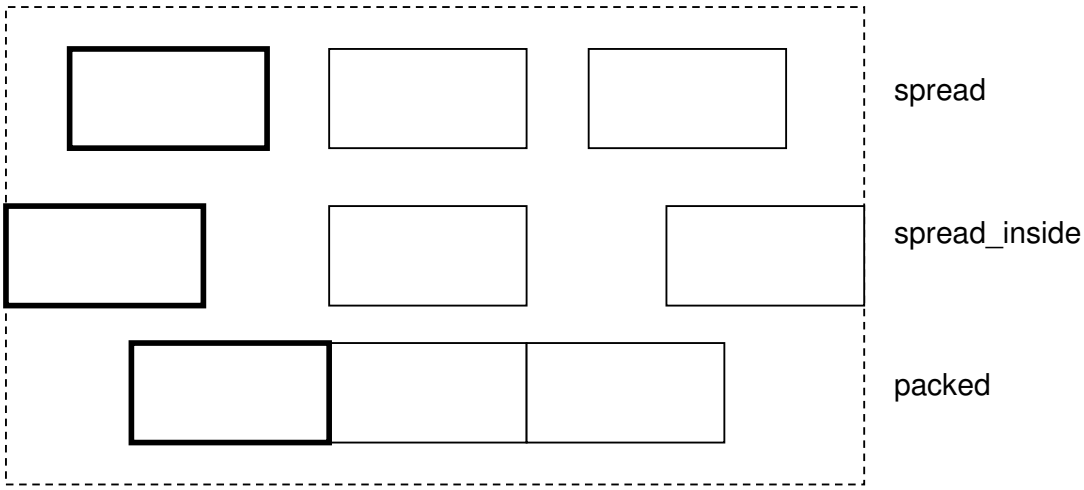

Рисунок 1.9 – Ланцюжки компонентів

В наведенному прикладі перший компонент задає атрибути:
- constraintStart_toStartOf="parent"
- constraintEnd_toStartOf="<component2_id>"

Другий компонент:
- constraintStart_toEndOf="<component1_id>"
- constraintEnd_toStartOf="<component3_id>"

Третій компонент:
- constraintStart_toEndOf="<component2_id>"
- constraintEnd_toEndOf="parent"

Приклад використання ConstraintLayout:

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:stretchColumns="*">

    <!-- Relative attributes -->

    <View
        android:id="@+id/red_square"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:background="#ff0000"
        app:layout_constraintWidth_percent="0.33"
        app:layout_constraintHeight_percent="0.2"
```

```
    app:layout_constraintDimensionRatio=""
    app:layout_constraintVertical_bias="0.3"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>

<View
    android:layout_width="64dp"
    android:layout_height="48dp"
    android:background="#880000"
    android:layout_marginEnd="4dp"
    app:layout_constraintTop_toTopOf="@id/red_square"
    app:layout_constraintBottom_toBottomOf="@id/red_square"
    app:layout_constraintEnd_toStartOf="@id/red_square"/>

<View
    android:layout_width="64dp"
    android:layout_height="48dp"
    android:background="#880000"
    android:layout_marginStart="4dp"
    app:layout_constraintTop_toTopOf="@id/red_square"
    app:layout_constraintBottom_toBottomOf="@id/red_square"
    app:layout_constraintStart_toEndOf="@id/red_square"/>

<!-- Guidelines -->

<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline_start"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintGuide_begin="24dp"/>

<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline_end"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintGuide_end="24dp"/>

<!-- Chain -->

<TextView
    android:id="@+id/tv_login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Login"
    android:layout_marginTop="32dp"
    app:layout_constraintStart_toStartOf=
              "@id/guideline_start"
    app:layout_constraintTop_toBottomOf="@id/red_square"
    app:layout_constraintBottom_toTopOf="@id/tv_password"/>
```

```
    <TextView
        android:id="@+id/tv_password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Password"
        app:layout_constraintStart_toStartOf=
                  "@id/guideline_start"
        app:layout_constraintTop_toBottomOf="@id/tv_login"
        app:layout_constraintBottom_toTopOf=
                  "@id/tv_restore_password"/>

    <TextView
        android:id="@+id/tv_restore_password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#00aaff"
        android:layout_marginBottom="32dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/tv_password"
        app:layout_constraintBottom_toBottomOf="parent"
        android:text="Forgot Password?"/>

    <!-- Barrier -->

    <androidx.constraintlayout.widget.Barrier
        android:id="@+id/barrier"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:barrierDirection="end"
        app:constraint_referenced_ids="tv_login,tv_password"/>

    <EditText
        android:id="@+id/et_login"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Enter login"
        android:layout_marginStart="12dp"
        app:layout_constraintStart_toStartOf="@id/barrier"
        app:layout_constraintEnd_toEndOf="@id/guideline_end"
        app:layout_constraintTop_toTopOf="@id/tv_login"
        app:layout_constraintBottom_toBottomOf="@id/tv_login"/>

    <EditText
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:hint="Enter password"
        android:layout_marginStart="12dp"
        app:layout_constraintTop_toTopOf="@id/tv_password"
        app:layout_constraintBottom_toBottomOf="@id/tv_password"
        app:layout_constraintStart_toStartOf="@id/barrier"
        app:layout_constraintEnd_toEndOf="@id/guideline_end"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Результат:

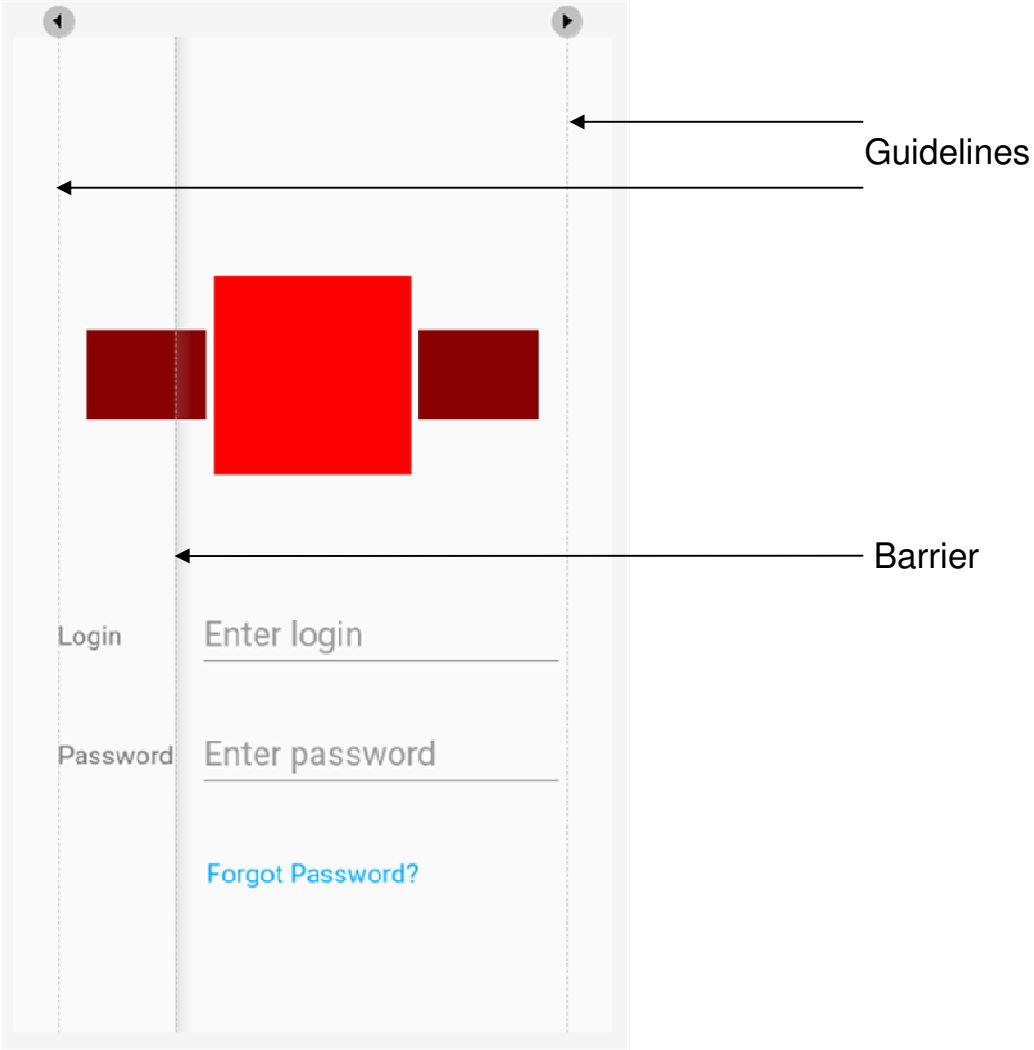

Рисунок 1.10 – ConstraintLayout

1.2.6 Android Manifest

В кожному Android-проекті є файл опису майбутнього додатку з назвою AndroidManifest.xml. Це важливий файл конфігурації, в якому описуються:

- дозволи (permissions), наприклад: доступ до камери, до книги контактів, до зовнішнього пристрою зберігання даних і т.д.
- особливості пристрою (features), які потрібні додатку для коректної роботи. Наприклад, сканеру штрихкодів, очевидно, необхідно щоб пристрій мав камеру. Без камери такий додаток просто не зможе працювати
- список та конфігурація програмних компонентів (application components). Найчастіше всього це список активностей (Activity) – компонентів, які відповідають за відображення інтерфейсу користувача (див.

наступний підрозділ 1.2.6). Рідше – сервіси, контент провайдери та Broadcast Receiver'и.

- метадані додатку: назва, іконка, тема та ін.

Файл маніфесту включається в результуючий APK-архів та потім зчитується системою Android. При створенні нового проекту, AndroidManifest.xml генерується автоматично та має приблизно наступний вигляд:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="stu.cn.ua.hello" >

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >

    <activity android:name=".MainActivity" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
                    "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>

</manifest>
```

Отже, в наведеному прикладі:

- задається пакет додатку (атрибут package);
- задається конфігурація додатку (тег application):
  - allowBackup: вказує системі, чи необхідно зберігати дані додатку після його видалення;
  - icon, roundIcon: іконка додатку;
  - label: назва додатку;
  - supportRtl: чи підтримується локалізація на мови з оберненим порядком слів (наприклад, араби читають справа наліво);
  - theme: базова UI-тема додатку;
- вказується, що додаток має лише одну Activity і ця Activity є основною (тег intent-filter з підтегами action та category), тобто буде показуватися користувачеві при запуску додатку.

1.2.6 Програмний компонент Activity

Activity (активність) – це один з чотирьох програмних компонентів Android, який відповідає за відображення інтерфейсу користувача. За задумом, активність мала надавати можливість користувачеві виконувати певну одну дію (action). В принципі, так і було до третьої версії Android, де в якості основної одиниці інтерфейсу замість активності став виступати фрагмент (Fragment). Нині активність зазвичай виступає контейнером для фрагментів, але якщо додаток є простим або спеціально не проектується для використання на планшетах, то активність і зараз може виступати основою для побудови інтерфейсу користувача.

В більш широкому задумі, активність – це не лише компонент для відображення інтерфейсу, а ще й окрема частина додатку, яка може виступати в якості елемента міжпроцесорної комунікації. Наприклад, якщо ви хочете зробити фото з камери, то рекомендується використовувати стандартну активність (вбудований додаток Camera), яка надається системою Android, замість ручної реалізації.

В більшості випадків, кожен Android-додаток має як мінімум одну або більше активностей. Одна з активностей додатку є основною – тобто, є точкою входу в додаток, її іконка та назва відображається в списку додатків пристрою. Варто зазначити, що основних активностей може бути і більше. Тоді після встановлення додатку на пристрій, в лаунчері з'явиться не одна іконка, а більше, кількість нових іконок в лаунчері рівна кількості основних активностей додатку.

Створення нової Activity. Для створення нової активності необхідно:

1) Створити макетний файл (каталог <module>/src/main/res/layout), де задати інтерфейс майбутньої активності. Назву макетного файлу зазвичай задають у форматі activity_<name>.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="New Activity"/>

</FrameLayout>
```

2) Створити клас активності, який має в своїй ієрархії наслідування містити клас Activity. Зазвичай клас активності наслідують не напряму від класу Activity, а від одного з існуючих підкласів, наприклад AppCompatActivity:

Java:

```
import stu.cn.ua.hello.R;
public class NewActivity extends AppCompatActivity {
   @Override
   protected void onCreate(@Nullable Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_new);
   }
}
```

Kotlin:

```
import stu.cn.ua.hello.R
class NewActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_new)
   }
}
```

В класі активності необхідно як мінімум перевизначити метод життєвого циклу onCreate, де потрібно вказати макетний файл, з яким зв'язується активність, за допомогою методу setContentView(…).

3) Додати опис активності в файл AndroidManifest.xml (всередині тегу application):

```
<application android:theme="@style/AppTheme" … >
    …
    <activity android:name=".NewActivity" />
</application>
```

В атрибуті android:name вказується клас активності відносно пакету додатку. Пакет додатку, як вже було згадано в п.п. 1.2.6, задається також в файлі маніфесту (тег manifest верхнього рівня). В даному випадку клас активності знаходиться безпосередньо в пакеті додатку, тому вказується лише назва класу, починаючи з точки. Можна вказати і повний шлях до класу:

```
<application android:theme="@style/AppTheme" … >
    …
    <activity android:name="stu.cn.ua.hello.NewActivity" />
</application>
```

Запуск Activity. Останній етап – це використати новостворену активність у своєму додатку: або зробити її основною, або запустити з іншої активності. В першому випадку необхідно до тегу activity додати підтег intent-filter наступного змісту:

```
<activity android:name=".NewActivity" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Також рекомендується задати іконку на назву для активності (атрибути android:icon, android:roundIcon та android:label), щоб розрізняти її з поміж інших активностей.

В другому випадку необхідно запустити активність з іншої активності. При створенні нового проекту зазвичай вже є як мінімум одна активність з назвою MainActivity. Наприклад, в макетний файл activity_main.xml додаємо кнопку:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/tv_main_hint"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a MainActivity"
        app:layout_constraintVertical_chainStyle="packed"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf=
                    "@+id/bt_launch_new_activity"/>

    <Button
        android:id="@+id/bt_launch_new_activity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Launch NewActivity"
        android:layout_marginTop="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/tv_main_hint"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

В класі MainActivity додаємо обробник на натискання кнопки:
Java:

```
import stu.cn.ua.hello.R;

public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        findViewById(R.id.btLaunchNewActivity)
          .setOnClickListener(button -> {
            Intent intent = new Intent(this, NewActivity.class);
            startActivity(intent);
          });
    }
}
```

Kotlin:

```
import kotlinx.android.synthetic.main.activity_main.*
import stu.cn.ua.hello.R

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btLaunchNewActivity.setOnClickListener {
            val intent = Intent(this, NewActivity::class.java)
            startActivity(intent)
        }
    }
}
```

Тут і далі використовуються лямбда-вирази з Java 8. Щоб їх підключити, необхідно в build.gradle в секцію android додати наступні рядки:

```
android {
    …
    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }
}
```

У випадку з Kotlin тут і далі використовується плагін Android Kotlin Extensions. Щоб його підключити, треба в тому ж файлі build.gradle зверху додати рядок:

```
apply plugin: 'kotlin-android-extensions'
```

В даному випадку був використаний явний виклик активності. Окрім явного запуску, є ще й неявний. Відмінність полягає у тому, що при явному виклику активності, програміст прямо зазначає ім’я активності (її клас), яку необхідно запустити. При неявному ж виклику активності програміст задає

бажану дію, а система Android, базуючись на параметрах виклику, сама підбирає необхідну активність. Наприклад, неявний виклик використовується для запуску камери, оскільки на різних пристроях від різних виробників, активність камери може називатись по різному. При неявному виклику задаються аргументи об'єкту **Intent**.

Intent – один з важливих класів в Android SDK, який представляє намір на виконання певної дії. При явному запуску в об'єкті Intent вказувалась назва активності, тим самим декларувався намір запустити конкретну активність. При неявному запуску, задаються такі аргументи як: категорія (category), дія (action), дані (extra data), URI. Приклад запуску системної камери:

```
Intent takePictureIntent = new Intent(
    MediaStore.ACTION_IMAGE_CAPTURE);
takePictureIntent.putExtra(
    MediaStore.EXTRA_OUTPUT,
    targetFileUri
);
startActivityForResult(takePictureIntent, REQUEST_CODE);
```

В даному випадку назва активності не задається. Замість цього вказуються:

- дія ACTION_IMAGE_CAPTURE, яка заявляє системі, що додаток хоче зробити фото за допомогою камери
- додаткові дані у вигляді пари ключ+значення з ключем EXTRA_OUTPUT, який задає шлях, куди необхідно зберегти фотографію.

Поряд з об'єктом Intent нерозривно зв'язаний тег intent-filter, який ви вже бачили в прикладі файлу AndroidManifest.xml. Система виконує неявний виклик активності, опираючись саме на intent-filter при пошуку активності. Тобто, якщо ви хочете створити свій власний додаток, який працює з камерою пристрою, і хочете зробити його альтернативою системній камері, то необхідно в вашій активності, яка відповідає за роботу з камерою, додати відповідний intent-filter, де вказати дію аналогічну значенню константи ACTION_IMAGE_CAPTURE («android.media.action.IMAGE_CAPTURE»):

```
<activity android:name="stu.cn.ua.hello.MyCameraActivity">
  <intent-filter>
    <action android:name="android.media.action.IMAGE_CAPTURE" />
  </intent-filter>
</activity>
```

Після встановлення такого додатку на пристрій, якщо ви спробуєте відкрити камеру, наприклад, з браузеру, то система знайде не одну, а дві активності, які можуть виконати дію ACTION_IMAGE_CAPTURE, і запропонує користувачеві вибрати, яку саме активність використати для фотографування.

Передача даних між різними Activity. Очевидно, що якщо активності можуть запускати одна одну, то вони повинні мати механізм спілкування між собою. І дійсно, такий механізм передбачений.

По-перше, в об'єкті класу Intent передаються дані від активності, котра запускає, до активності, яка запускається. До цих даних відносяться, як вже було сказано раніше: URI, action, category, extra data. Розглянемо приклад та поясними кожен з вищезазначених термінів:

Java:

```
Intent intent = new Intent(this, SettingsActivity.class);
// action
intent.setAction("stu.cn.ua.hello.ACTION_UPDATE");
// URI
intent.setData(Uri.parse("customprotocol://user/id/156"));
// Extra data
intent.putExtra(KEY_BITMAP, bitmap);
startActivity(intent);
```

Kotlin:

```
val intent = Intent(this, SettingsActivity::class.java).apply {
    // action
    action = "stu.cn.ua.hello.ACTION_UPDATE"
    // URI
    data = Uri.parse("customprotocol://user/id/156")
    // Extra data
    putExtra(KEY_BITMAP, bitmap)
}
startActivity(intent)
```

Крім вище перерахованих даних, є ще Component Name – у всіх наведених прикладах це клас Activity (SettingsActivity.class), який необхідно запустити. *Наявність Component Name визначає, яким буде виклик: явним або неявним.*

Категорія задається за допомогою методу класу Intent addCategory(). Категорія визначає тип наміру, вказує на різновид активності. Дві стандартні категорії, які слід запам'ятати:

- CATEGORY_LAUNCHER – означає, що активність є основною та може бути викликана з лаунчера додатків (її іконка відображається в списку на робочому столі);
- CATEGORY_BROWSABLE – означає, що на Activity можна посилатися за допомогою гіпертекстових посилань (наприклад, з вб-сторінки у браузері).

При неявному виклику активності, система шукає підходящу Activity на основі наступних даних в об'єкті Intent: URI, action та category. Extra data використовується вже безпосередньо тією активністю, яка буде викликана.

Отже, як передавати дані до активності, яка викликається, розібрались. Тепер розглянемо, як передавати дані в зворотному порядку. Наприклад, перша активність запустила другу активність з деякими даними для обробки,

а в результаті у відповідь необхідно дізнатись результат обробки цих даних. Щоб досягти поставленої цілі, необхідно використати зв'язку методів:

- метод startActivityForResult(…)
- метод setResult(…)
- метод зворотного виклику onActivityResult(…)

Метод startActivityForResult, на відміну від вже відомого методу startActivity, приймає в якості аргументу додатковий параметр requestCode типу Integer. Request code може бути довільним, зазвичай його зберігають у деякій константі, проте краще взяти за правило, щоб це було число в діапазоні від 0 до 0xFFFF, оскільки система Android може додавати маску до бітів зліва.

Request Code потім використовується в методі зворотного виклику для ідентифікації відповіді. Даний аргумент особливо важливий, якщо активність викликає не одну, а більше активностей різного типу: за допомогою request code можна розрізняти, з якої активності були отримані дані. Нижче наведений код активності, яка виконує запуск іншої активності:

Java:

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG =
        MainActivity.class.getSimpleName();

    private static final int RQ_CODE = 1;

    …

    private void processData() {
        Intent intent =
            new Intent(this, ProcessDataActivity.class);
        startActivityForResult(intent, RQ_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode,
                    int resultCode, @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == RQ_CODE
                && resultCode == RESULT_OK
                && data != null) {
            Log.d(TAG, "Data has been processed");
            Log.d(TAG, "Results: " + data.toString());
        }
    }
}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {

    …
```

```
    private fun processData() {
        val intent =
            Intent(this, ProcessDataActivity::class.java)
        startActivityForResult(intent, RQ_CODE)
    }

    override fun onActivityResult(requestCode: Int,
            resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == RQ_CODE
            && resultCode == RESULT_OK
            && data != null) {

            Log.d(TAG, "Data has been processed")
            Log.d(TAG, "Results: $data")
        }

    }

    // Kotlin doesn't support static variables
    companion object {

        private const val RQ_CODE = 1
        @JvmField val TAG = MainActivity::class.java.simpleName

    }

}
```

Зверніть увагу на рядки, які починаються з Log.d("…"). Клас Log дозволяє записувати будь-яку інформацію про стан програми в системний лог, який можна переглянути потім або слідкувати за ним в реальному часі за допомогою інструменту LogCat (або за допомогою відповідної вкладки в Android Studio). Більш детально логування стану та подій розглядається в п.п. 1.2.8.

З боку Activity, з якої небхідно передати дані назад, виконуються наступні дії:

1) Викликається метод setResult(…), зі статусом виконання та даними.
2) Викликається метод finish(), щоб завершити виконання активності та перейти до попередньої активності.

Java:

```
public class ProcessDataActivity extends AppCompatActivity {

    public static final String EXTRA_RESULT = "RESULT";

    public void onDataProcessed(String result) {
        Intent intent = new Intent();
        intent.putExtra(EXTRA_RESULT, result);
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

Kotlin:

```
class ProcessDataActivity : AppCompatActivity() {
    fun onDataProcessed(result: String) {
       val i = Intent().apply { putExtra(EXTRA_RESULT, result) }
       setResult(RESULT_OK, intent)
       finish()
    }
    companion object {
        const val EXTRA_RESULT = "RESULT"
    }
}
```

Життєвий цикл процесу та Activity. Activity з моменту запуску і до моменту завершення проходить через декілька станів, які називаються станами життєвого циклу, які відповідають моментам створення, старту, продовження, паузи, зупинки та знищення активності. Всі ці операції виконуються системою, напряму не залежать від дій програміста, та відмінні від життєвого циклу Java-класу активності. *Життєвий цикл необхідно знати* для того, щоб вміти коректно написати Android-додаток, без помилок та багів, які тяжко виявити. Знання життєвого циклу є теоретичним знанням, але без якого майже неможливо написати навіть примітивні Android-додатки.

На відміну від десктопних операційних систем, в ОС Android програміст не управляє життєвим циклом процесу. Система сама вирішує, коли процес треба відновити, коли зупинити, і коли знищити, беручи до уваги наявні ресурси, такі як доступна пам'ять та заряд батареї. Отже, розглянемо життєвий цикл процесу:

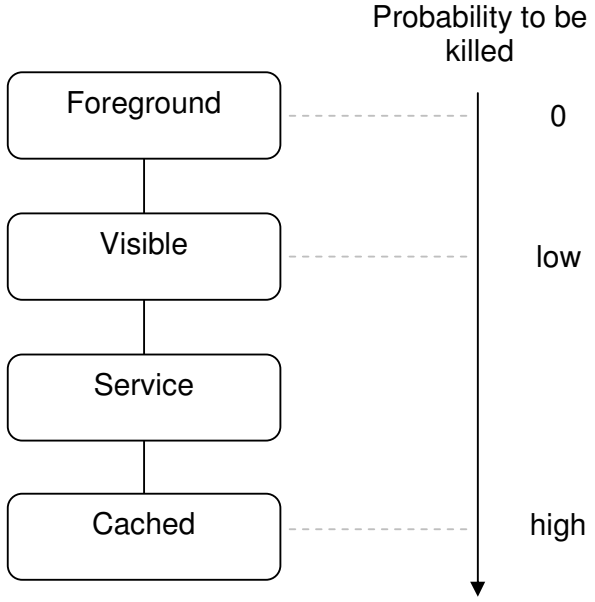

Рисунок 1.11 – Життєвий цикл процесу

Процес може перебувати в 4 станах: поточний процес, видимий процес, процес з активною службою, пустий процес.

Поточний процес – це процес з яким зараз, в поточний момент часу, безпосередньо працює користувач. Формально, до поточних процесів відноситься будь-який процес, який відповідає хоча б одній з трьох наступних вимог:

1) Процес має активну Activity, яка є запущеною, видимою користувачеві, знаходиться в стані «resumed» на передньому плані. Цей стан може бути відстежений розробником за допомогою методу зворотного виклику Activity.onResume().

2) Процес має активний програмний компонент Broadcast Receiver, який в даний момент часу виконує обробку деякої події (момент виконання методу зворотного виклику BroadcastReceiver.onReceive()).

3) Процес має активний програмний компонент Service (служба), яка в даний момент часу виконує один із наступних методів зворотного виклику: Service.onCreate(), Service.onStart(), Service.onDestroy().

Видимий процес – це процес, який в даний момент часу, як не дивно, в той чи інший спосіб, є видимим користувачеві. Процес є видимим, якщо він задовольняє одну з наступних вимог:

1) Процес має Activity, яка є видимою для користувача, але при цьому не є активною, тобто існує інша Activity, яка частково перекриває дану Activity.

2) Процес має активну службу переднього плану (Foreground Service). Служба переднього плану – це служба, до якої прив’язане видиме повідомлення (notification) в панелі статусу. Приклад подібного процесу: медіа-плеєр.

3) Додаток має службу, яка використовується системою для певних цілей, про які користувач тим чи іншим чином має уявлення. Наприклад: живі обкладинки робочого столу, програмний метод введення даних та ін.

Процес з активною службою – це процес, в якому запущена хоча б одна служба.

Життєвий цикл Activity трохи складніший, він зображений на рисунку нижче разом з методами зворотного виклику: onCreate, onStart, onResume, onPause, onStop, onDestroy. Окрім вище перелічених методів є й інші, найважливіший з яких є onSaveInstanceState.

Метод onSaveInstanceState не зображений на діаграмі життєвого циклу, оскільки момент його виклику не регламентується (він може бути викликаний як до onPause, так і після нього). Даний метод надає можливість розробнику зберегти поточний стан активності в екземплярі класу Bundle. Ці дані потім передаються в якості аргументу методу зворотного виклику onCreate(Bundle savedState), надаючи можливість відновити стан Activity при повторному створенні.

Зберігати та відновлювати стан Activity необхідно вручну при будь-якій зміні конфігурації в системі: орієнтація екрану, мова, дата та ін.

Виключення становлять деякі компоненти інтерфейсу, наприклад: EditText, який автоматично запам’ятовує останнє введене користувачем значення.

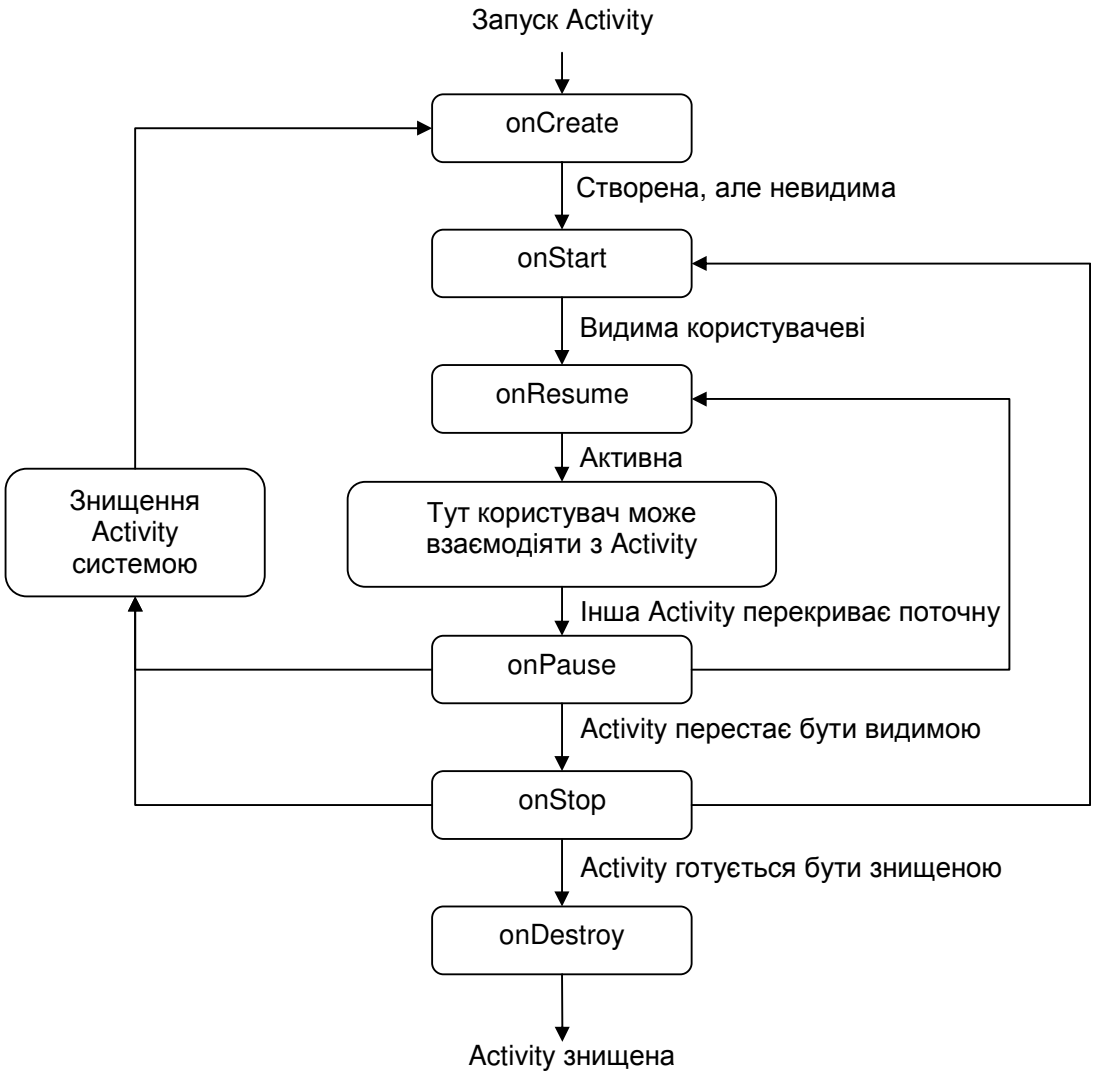

Рисунок 1.12 – Життєвий цикл активності

В прикладі нижче наведено код активності з логуванням методів життєвого циклу та коректним зберігання/відновлення стану активності при зміні конфігурації:

Java:

```
public class MainActivity extends AppCompatActivity {
    private static final String TAG =
              MainActivity.class.getSimpleName();
    private static final String KEY_COUNTER = "COUNTER";

    private TextView counterTextView;
    private int counter;

    @Override
```

```
protected void onCreate(@Nullable Bundle savedState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Log.d(TAG, "onCreate");

    if (savedState == null) {
        counter = 0;
    } else {
        counter = savedState.getInt(KEY_COUNTER);
    }

    updateView();

    counterTextView = findViewById(R.id.counterTextView);
    findViewById(R.id.incrementButton)
        .setOnClickListener(v -> {
            counter++;
            updateView();
        });
}

private void updateView() {
    counterTextView.setText("Counter value: " + counter);
}

@Override
protected void onSaveInstanceState(@NonNull Bundle outState){
    super.onSaveInstanceState(outState);
    outState.putInt(KEY_COUNTER, counter);
}

@Override
protected void onStart() {
    super.onStart();
    Log.d(TAG, "onStart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "onResume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(TAG, "onPause");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
}

@Override
protected void onDestroy() {
```

```
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {

    companion object {
        @JvmField val TAG = MainActivity::class.java.simpleName
        const val KEY_COUNTER = "COUNTER"
    }

    private var counter = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Log.d(TAG, "onCreate")

        counter = savedInstanceState?.getInt(KEY_COUNTER) ?: 0
        updateView()

        incrementButton.setOnClickListener {
            counter++
            updateView()
        }
    }

    private fun updateView() {
        counterTextView.setText("Counter value: $counter")
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putInt(KEY_COUNTER, counter)
    }

    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
```

```
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

}
```

В даному прикладі Activity має кнопку та текстову мітку, в якій відображається кількість натискань на кнопку. Оскільки тут реалізовано зберігання та відновлення стану за допомогою onSaveInstanceState/onCreate, то при повороті екрану пристрою в альбомну орієнтацію, стан активності відновлюється коректно.

1.2.7 Фрагменти (Android Fragments)

Фрагмент – це частина інтерфейсу користувача. Перевага використання фрагментів полягає у тому, що одна активність може містити кілька фрагментів. Фрагменти також можна повторно використовувати в декількох різних активностях. Загалом, фрагменти можна розглядати як деякий модульний компонент з власним життєвим циклом. Фрагменти можна в будь-який момент додавати, змінювати та видаляти з активності. Варто зазначити, що фрагменти не можуть існувати самі по собі. Для роботи їх необхідний хост – Activity. Життєвий цикл активності безпосередньо впливає на життєвий цикл фрагменту. Наприклад, коли Activity призупинена, то всі фрагменти в ній також зупиняються.

Управління фрагментами здійснюється за допомогою транзакцій. Транзакції дозволяють не тільки видаляти/додавати фрагменти, а й задавати анімаційні переходи, а також реалізовувати стекову навігацію, аналогічну стековій навігації активностей. В більшості випадків фрагменти відображають деяку частину інтерфейсу користувача. Для того, щоб відобразити компоненти інтерфейсу, фрагменту призначається компонувальник, який буде містити даний фрагмент.

Існує два способи створення фрагменту: статичний та динамічний. У першому випадку в макетний файл додається тег з назвою <fragment> та зазначається клас фрагменту, відповідальний за його поведінку:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <fragment android:name="ua.cn.stu.ListFragment"
```

```
        android:id="@+id/listContainer"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2" />
    <fragment
        android:name=" ua.cn.stu.DetailsFragment"
        android:id="@+id/contentContainer"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

У наведенному прикладі до макету Activity додано два фрагменти. Перший відображає список елементів, другий – детальну інформацію про елемент списку. Класи фрагментів, аналогічно до класів Activity, відповідають за логіку роботи фрагменту. Вони можуть мати обробники життєвого циклу, задавати обробники подій, реагувати на дії користувача та ін.

Другий спосіб створення фрагменту – за допомогою явних транзакцій у процесі виконання програми. Цей спосіб використовується для реалізації динамічного інтерфейсу та навігації на базі фрагментів. Приклад, аналогічний попередньому, але на базі явних транзакцій матиме наступний вигляд:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <FrameLayout
       android:id="@+id/listContainer"
       android:layout_width="0dp"
       android:layout_height="match_parent"
       android:layout_weight="2" />
   <FrameLayout
       android:id="@+id/contentContainer"
       android:layout_width="0dp"
       android:layout_height="match_parent"
       android:layout_weight="1" />

</LinearLayout>
```

Java:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedState) {
      super.onCreate(savedState);
      setContentView(R.layout.activity_main);
      if (savedState == null) {
        ListFragment listFragment = new ListFragment();
        ContentFragment contentFragment = new ContentFragment();
```

```
            getSupportFragmentManager()
              .beginTransaction()
              .add(R.id.listContainer, listFragment)
              .add(R.id.contentContainer, contentFragment)
              .commit();
        }
    }
}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedState: Bundle?) {
        super.onCreate(savedState)
        setContentView(R.layout.activity_main)
        if (savedState == null) {
            val listFragment = ListFragment()
            val contentFragment = ContentFragment()
            supportFragmentManager.beginTransaction()
                .add(R.id.listContainer, listFragment)
                .add(R.id.contentContainer, contentFragment)
                .commit()
        }
    }
}
```

З наведених прикладів можна зробити висновок, що за допомогою однієї транзакції можна виконати одразу декілька операцій над різними фрагментами. Транзакції мають в своєму арсеналі такі методи як add(), replace(), remove(), addToBackStack() та ін. Розглянемо їх більш детально:

- FragmentTransaction.add(layoutId, fragment, tag) – додає фрагмент до активності. Якщо в зазначеному контейнері вже існує фрагмент, то новий фрагмент ніяким чином не зачіпає старий фрагмент. Тег – це опціональний текстовий аргумент, його можна використовувати для пошуку фрагментів в активності за допомогою методу FragmentManager.findFragmentByTag(). Існує також варіант методу add() без зазначення ідентифікатора контейнеру. Його використовуються для фрагментів без інтерфейсу користувача.

- FragmentTransaction.replace(layoutId, fragment, tag) – заміщує існуючий фрагмент новим фрагментом. Старий фрагмент знищується та видаляється з активності.

- FragmentTransaction.remove(fragment) – видаляє зазначений фрагмент з активності. Аргумент даного методу – об’єкт класу фрагменту, а не його тег або ідентифікатор. Отримати фрагмент за допомогою його тегу або за допомогою ідентифікатору контейнеру можна використовуючи методи:

  - getSupportFragmentManager().findFragmentById()
  - getSupportFragmentManager().findFragmentByTag()

Пошук за ідентифікатором контейнеру завжди повертає останній активний фрагмент з контейнеру, якщо таких фрагментів більше одного.

- FragmentTransaction.addToBackStack(backStackName) – додає транзакцію до стеку операцій. Це дозволяє згодом відмінити всі операції у

зворотному порядку, повернувшись до попереднього стану. Наприклад, якщо транзакція додавала деякий фрагмент до активності, то при натисканні кнопки «Назад», доданий фрагмент буде видалений з активності. Стек транзакцій використовується для реалізації стекової навігації на базі фрагментів.

Життєвий цикл фрагмент схожий на життєвий цикл активності. На рисунку нижче зображено методи життєвого циклу фрагментів та послідовність їх виконання:

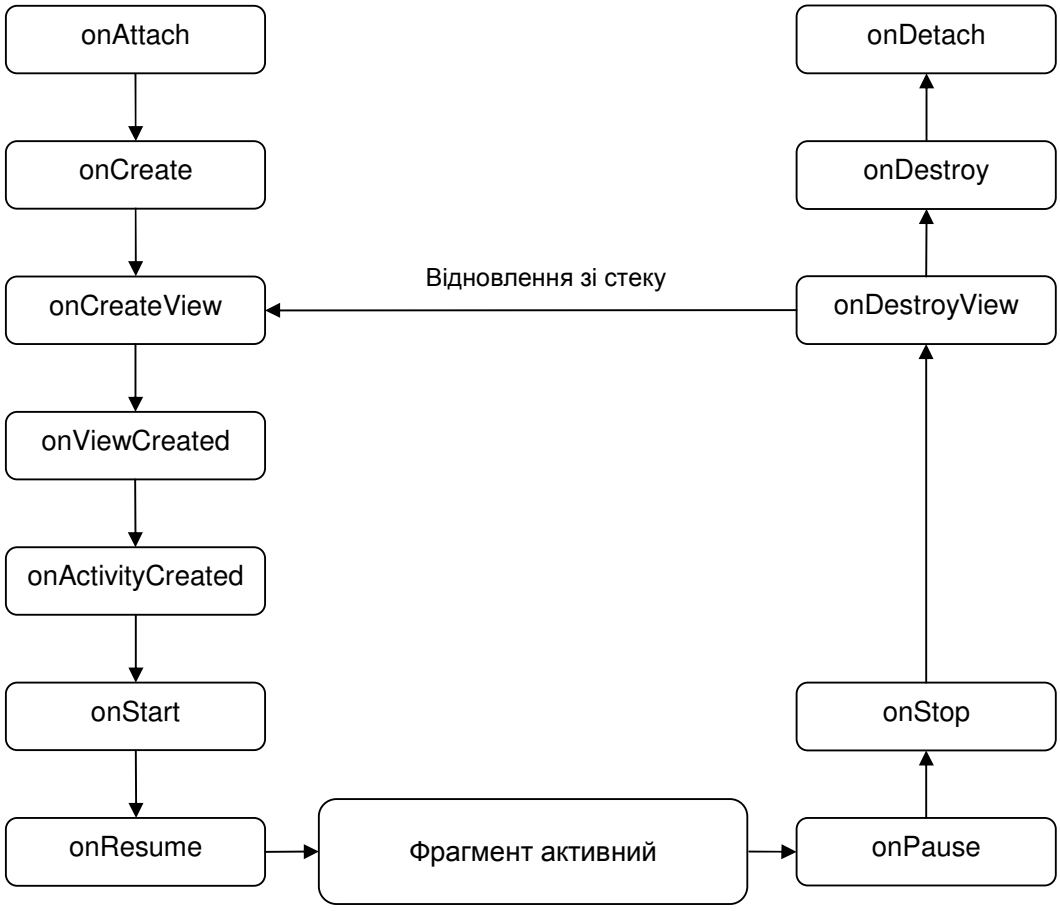

Рисунок 1.13 – Життєвий цикл фрагменту

Метод onCreate використовується для ініціалізації даних та ресурсів, які мають виживати після паузи та зупинки фрагменту. Також в даному методі ініціалізують за допомогою методу setRetainInstance(Boolean) так-звані Retain-фрагменти – тобто, фрагменти, які виживають при змінах конфігурації (наприклад, після повороту екрану).

Метод onCreateView повертає кореневу View, яка буде додана до контейнеру фрагменту. Саме даний метод задає інтерфейс фрагменту. Саме в даному методі (або, за вибором, у методі onViewCreated) ініціалізуються компоненти інтерфейсу користувача. Варто зазначити, що на відміну від

Activity, фрагменти не мають методу findViewById. Натомість слід використовувати метод з аналогічною назвою findViewById об'єкту View:

Java:

```
@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater,
                         @Nullable ViewGroup container,
                         @Nullable Bundle savedState) {
    // layout fragment_content.xml will be displayed
    return inflater.inflate(
        R.layout.fragment_content,
        container,
        false
    );
}

@Override
public void onViewCreated(@NonNull View view,
                          @Nullable Bundle savedState) {
    super.onViewCreated(view, savedState);
    Button testButton = view.findViewById(R.id.testButton);
    testButton.setVisibility(View.VISIBLE);
    testButton.setOnClickListener(v -> {
        Toast.makeText(
            getContext(),
            R.string.hello_world,
            Toast.LENGTH_SHORT
        ).show();
    }
}
```

Kotlin:

```
override fun onCreateView(inflater: LayoutInflater,
                          container: ViewGroup?,
                          savedState: Bundle?): View? {
    // layout fragment_menu.xml will be displayed
    return inflater.inflate(
        R.layout.fragment_menu,
        container,
        false
    )
}

override fun onViewCreated(view: View, savedState: Bundle?) {
    super.onViewCreated(view, savedState)
    val testButton = view.findViewById<View>(R.id.testButton)
    testButton.apply {
        visibility = View.VISIBLE
        setOnClickListener {
            Toast.makeText(
                context,
                R.string.hello_world,
                Toast.LENGTH_SHORT
            ).show()
        }
```

```
        }
    }
```

1.2.8 Використання LogCat

Для спрощення відлагодження додатків в системі Android існує системний лог, який працює за принципом черги FIFO, тобто нові записи витісняють більш старі записи з логу. Розмір буферу логів можна задати в меню налаштувань, попередньо увімкнувши режим розробника.

Переглядати логи можна за допомогою інструменту командного рядка Android Debug Bridge (команда adb logcat) або за допомогою панелі LogCat в AndroidStudio:

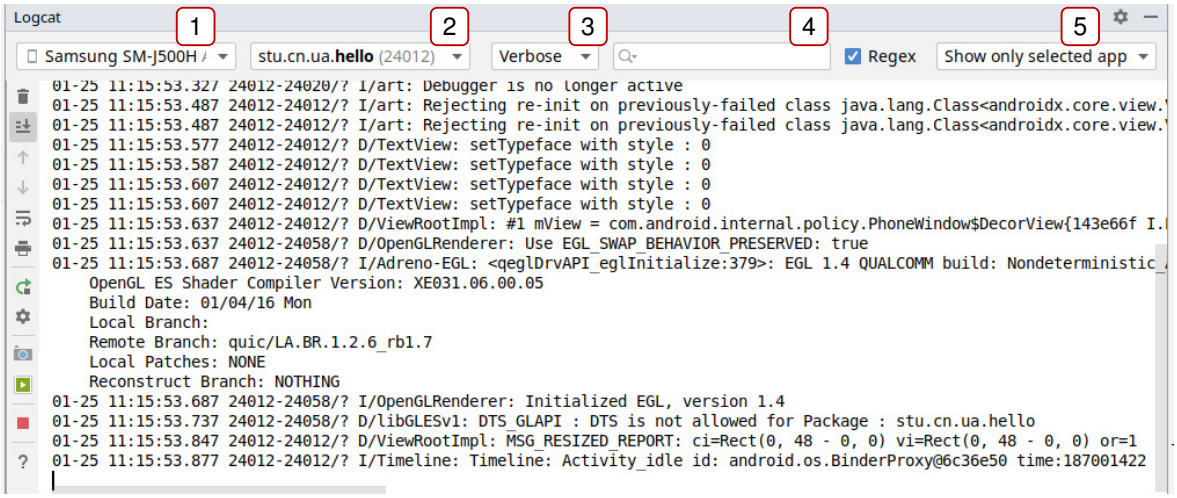

Рисунок 1.14 – Вікно LogCat

1) Пристрій/емулятор, з якого відображаються логи;

2) Показувати логи, записані вказаним процесом (при умові, що в меню 5 ввімкнена опція “Show Only Selected App”);

3) Фільтрація логів за пріоритетом;

4) Фільтрація за ключовими словами або RegExp-виразом;

5) Фільтрація за джерелом логів.

У панелі LogCat відображаються системні логи, які записуються Android-додатками за допомогою класу Log. LogCat відображає логи в режимі реального часу та зберігає історію. Щоб відобразити лише цікаву інформацію, можна створювати фільтри, змінювати кількість інформації, яка відображається в повідомленнях, встановлювати пріоритети, відображати логи, записані лише конкретним процесом та здійснювати пошук у журналі. Якщо додаток завершується помилкою, LogCat показує повний лог виключення зі стеком викликів, які спричинили помилку.

Клас Log дозволяє записувати логи до системного журналу, які потім відображаються в панелі LogCat. Як правило, використовуються наступні методи, перелічені в порядку від найвищого до найнижчого пріоритету:

```
Log.e(String tag, String message) // error
Log.w(String tag, String message) // warning
Log.i(String tag, String message) // information
Log.d(String tag, String message) // debug
Log.v(String tag, String message) // verbose
```

Логування помилок (error), попереджень (warning) та інформаційних логів (information) завжди зберігаються, навіть в релізних версіях додатку. Логування типу verbose не слід включати в результуючий APK-файл.

Перший параметр кожного з методів легування – тег (tag). Тег логу системного журналу – це короткий рядок, в якому зазвичай вказується компонент, з якого був записаний лог (наприклад, назва Activity або класу, який записує даний лог). Загалом же, тегом може бути будь-який рядок. Зазвичай тег задається константою на початку класу:

Java:

```
public class MainActivity extends AppCompatActivity {
    private static final String TAG =
                MainActivity.class.getSimpleName();

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }
}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d(TAG, "onCreate")
    }

    companion object {
        @JvmField val TAG: String =
                MainActivity::class.java.simpleName
    }
}
```

1.2.9 Основні візуальні компоненти

Android SDK разом з бібліотеками підтримки (раніше support library, зараз AndroidX) надають розробнику велику кількість найрізноманітніших

візуальних компонентів, з яких можна швидко побудувати інтерфейс користувача різної складності. Розглянемо основні з них:

- Button – звичайна кнопка. Використовується для ініціювання деякої дії. Компонент аналогічних кнопкам в звичайних десктопних системах. Основний атрибут: android:text:

```
<Button
    android:id="@+id/showMessageButton"
    style="@style/ButtonStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/show_message" />
```

У всіх прикладах до цього моменту, текст був заданий напряму в атрибуті android:text. Тут і далі натомість будемо використовувати текстові ресурси (в даному випадку посилаємось на текстовий ресурс з іменем show_message). Текстові ресурси задаються в спеціальних XML-файлах (зазвичай з назвою string.xml), які містяться в каталозі app/src/main/res/values. Приклад файлу strings.xml:

```
<resources>
    <string name="app_name">Get Variant App</string>
    <string name="show_message">Show Message</string>
    <string name="hi">Hello world</string>
</resources>
```

Додати обробник на натискання кнопки (до речі, як і на будь-який інший компонент) можна за допомогою методу setOnClickListener():

Java:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(@Nullable Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      findViewById(R.id.showMessageButton)
         .setOnClickListener(v -> {
            Toast.makeText(this, R.string.hi, Toast.LENGTH_SHORT)
                  .show();
         });
    }
}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      showMessageButton.setOnClickListener {
         Toast.makeText(this, R.string.hi, Toast.LENGTH_SHORT)
               .show()
      }
```

```
    }
}
```

- TextView – текстова мітка, відображає будь-яку текстові інформацію. Текст задається за допомогою атрибуту android:text. Також є можливість змінити текст програмно (інші компоненти, такі як Button, EditText також використовують ті ж самі методи):

Java:

```
hintTextView.setText("Some text");
hintTextView.setText(R.string.some_text_resource);
```

Kotlin:

```
hintTextView.text = "Some text"
hintTextView.setText(R.string.some_text_resource)
```

- EditText – дозволяє не тільки відображати, а й редагувати текстову інформацію. Підтримує всі вище зазначені атрибути та методи, що й компоненти Button та TextView. Для отримання введеного значення використовується метод getText():

Java:

```
EditText nameEditText = findViewById(R.id.nameEditText);
String enteredName = nameEditText.toString();
String message = getString(R.string.greeting, enteredName);
Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
```

Kotlin:

```
val enteredName = nameEditText.text.toString()
val message = getString(R.string.greeting, enteredName)
Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
```

Тут використовуються текстові ресурси з плейхолдерами. Приклад ресурсу з плейсхолдерами:

```
<resources>
    <string name="greeting">Hello, %1$s</string>
</resources>
```

- ImageView – відображає графічну інформацію. Основний атрибут android:src (або app:srcCompat, який також підтримує векторні зображення).

```
    <ImageView
        android:id="@+id/logoImageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_arrow"/>
```

Графічні ресурси зберігаються в каталозі app/src/main/res/drawable. Зазвичай невекторні ресурси для Android-додатків створюють у декількох

варіантах для екранів різного розміру, які розміщують в каталогах (від меншого до більшого) drawable-ldpi, drawable-mdpi, drawable-hdpi, drawable-xxhdpi, drawable-xxxhdpi. Більш детально про підтримку екранів різного розміру можна прочитати в офіційній документації [6].

- CheckBox / Switch – два компоненти вимикачі. Мають два стани: ввімкнений та вимкнений, який задається/зчитується атрибутом android:checked та методами setChecked, isChecked.

```
<CheckBox
    android:id="@+id/toggleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/toggle"
    android:checked="true"/>
```

- RadioButton / RadioGroup – аналог CheckBox, але з іншим механізмомо роботи: використовується для вибору лише одного елементу з групи. Підтримують ті ж самі атрибути та методи, що й CheckBox:

Java:

```
View.OnClickListener listener = v -> {
    switch (v.getId()) {
        case R.id.cashRadio:
            // user has chosen cash payment method
            break;
        case R.id.creditCardRadio:
            // user has chosen credit card payment method
            break;
        default:
            throw new RuntimeException("Unknown choice");
    }
};
findViewById(R.id.creditCardRadio).setOnClickListener(listener);
findViewById(R.id.cashRadio).setOnClickListener(listener);
```

Kotlin:

```
val listener = View.OnClickListener {
    when (it.id) {
        R.id.cashRadio -> {
            // user has chosen cash payment method
        }
        R.id.creditCardRadio -> {
            // user has chosen credit card payment method
        }
        else -> throw RuntimeException("Unknown choice")
    }
}
findViewById<View>(R.id.creditCardRadio)
    .setOnClickListener(listener)
findViewById<View>(R.id.cashRadio)
    .setOnClickListener(listener)
```

Для гарантії того, що в кожен момент часу буде активним лише один з компонентів RadioButton, їх додають до компонувальника RadioGroup:

```
<RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton
        android:id="@+id/creditCardRadio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/credit_card" />
    <RadioButton
        android:id="@+id/cashRadio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cash" />

</RadioGroup>
```

- ProgressBar – відображення прогресу виконання деякої довготривалої операції. Має два режими: indeterminate/determinate, для операцій з невідомим часом виконання та відомим відповідно. Найчастіше використовуються саме режим indeterminate. Приклади:

```
<ProgressBar
     android:id="@+id/determinateProgress"
     style="@style/Widget.AppCompat.ProgressBar.Horizontal"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:indeterminate="false"/>

<ProgressBar
     android:id="@+id/indeterminateProgress"
     style="@style/Widget.AppCompat.ProgressBar"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:indeterminate="true"/>
```

- ScrollView – компонувальник, який може містити лише один елемент (зазвичай таким елементом виступає інший компонувальник). Використовується у випадках, якщо макет за розміром більший, аніж розмір екрану, дозволяючи користовучеві жестами прокручувати контент.

Усі вище зазначені компоненти мають набір стандартних атрибутів та методів, наприклад android:enabled, android:visibility, android:onClick та ін. Більш з них очевидні за призначенням та використанням. Розглянемо лише атрибут android:visibilityта відповідні методи: setVisibility, getVisibility. Дані атрибути/методи працюють з типом int, а не boolean, оскільки існує три типи видимості компонентів: View.VISIBLE, View.INVISIBLE, View.GONE.

Видимість типу View.INVISIBLE відрізняється від View.GONE тим, що в першому випадку хоч компонент і невидимий, але все одно він має фізичні розміри, які впливають на розташування інших компонентів. У випадку ж з View.GONE, компонент згортається до нульової довжини та ширини.

1.2.10 Побудова списків в ОС Android

До спискових візуальних компонентів відносять:
- ListView
- RecyclerView
- Spinner
- AutoCompleteTextView

В Android сповідується принцип розділення даних та інтерфейсу. Тому всі спискові компоненти не працюють з даними напряму. Замість цього використовується патерн «адаптер». Кожен з вище перелічених спискових компонентів має метод setAdapter(), який дозволяє встановити адаптер даних для списку. Адаптер відповідає за перетворення масивів даних в елементи інтерфейсу списку.

Роль сискових компонентів в теорії може відігравати й ScrollView, в якому міститься інший компонувальник, наприклад, LinearLayout, до якого динамічно з програмного коду, як і до будь-якого іншого компонувальника, можна додавати компоненти інтерфейсу за допомогою методу ViewGroup.addView(), наприклад:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:id="@+id/listContainer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <!-- List items will be added here -->

    </LinearLayout>

</ScrollView>
```

Java:

```
Button button = new Button(this);
button.setText(R.string.doneButton);

TextView textView = new TextView(this);
textView.setText(R.string.hint);
textView.setPadding(12, 12, 12,12);
```

```
ViewGroup container = findViewById(R.id.listContainer);
container.addView(button);
container.addView(textView);
```

Kotlin:

```
val button = Button(this)
button.setText(R.string.doneButton)

val textView = TextView(this)
textView.setText(R.string.hint)
textView.setPadding(12, 12, 12, 12)

val container = findViewById<ViewGroup>(R.id.listContainer)
container.addView(button)
container.addView(textView)
```

Однак, використання ScrollView для відображення списків *не рекомендується*. У разі великої кількості елементів у списку, додаток буде працювати не ефективно, витрачаючи надто великі ресурси пам'яті та процесору. Також операції по створення та додаванню великої кількості компонентів інтерфейсу займають великі проміжки часу, і це навіть помітно користувачеві додатку, оскільки з'являються лаги та затримки в обробці подій.

Компоненти ListView, Spinner, AutoCompleteTextView та особливо RecyclerView – оптимізовані для роботи з великими списками та не мають вище перелічених недоліків. Дані спискові компоненти відображають тільки ті елементи списку, які безпосередньо видимі користувачеві. RecyclerView, в свою чергу, має ще й додаткову оптимізацію: даний компонент виконує кешування елементів списку, що дозволяє не створювати нові елементи при скролі списку, а просто переміщувати макети елементів, які вийшли за межі екрану, знову, підставляючи відповідні дані.

Далі розглянемо більш детально списків компонент ListView.

ListView – базовий списковий компонент інтерфейсу, який став частиною Android SDK ще з перших версій операційної системи Android. Простий у налаштуванні, особливо при використанні стандартних адаптерів. Приклад використання (тут і далі використовуємо бібліотеку Faker для генерації спискових даних) наведено нижче.

Макет інтерфейсу Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/students_list"
```

```
        android:layout_marginStart="@dimen/screenPadding"
        android:layout_marginTop="@dimen/screenPadding"/>

    <ListView
        android:id="@+id/studentsListView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:dividerHeight="1dp"
        android:divider="@color/lightDivider"
        android:overScrollMode="never"
        android:layout_marginTop="@dimen/list_margin" />

</LinearLayout>
```

Клас MainActivity:
Java:

```
public class MainActivity extends AppCompatActivity {

    private Faker faker = Faker.instance(new Random(1));

    @Override
    protected void onCreate(@Nullable Bundle savedState) {
        super.onCreate(savedState);
        setContentView(R.layout.activity_main);

        List<String> students = generateRandomStudents(100);
        ListView studentListView =
                    findViewById(R.id.studentsListView);
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
            this,
            android.R.layout.simple_list_item_1,
            students
        );
        studentListView.setAdapter(adapter);
    }

    private List<String> generateRandomStudents(int count) {
        List<String> students = new ArrayList<>(count);
        for (int i = 0; i < count; i++) {
            students.add(faker.name().fullName());
        }
        return students;
    }

}
```

Kotlin:

```
class MainActivity : AppCompatActivity() {
    private val faker = Faker.instance(Random(1))

    override fun onCreate(savedState: Bundle?) {
        super.onCreate(savedState)

        setContentView(R.layout.activity_main)
```

```
        val students = generateRandomStudents(100)
        val studentListView =
               findViewById<ListView>(R.id.studentsListView)
        val adapter = ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            students
        )
        studentListView.adapter = adapter

    }

    private fun generateRandomStudents(count:Int): List<String> {
        return (1..count).map {
            faker.name().fullName()
        }
    }
}
```

Результат:

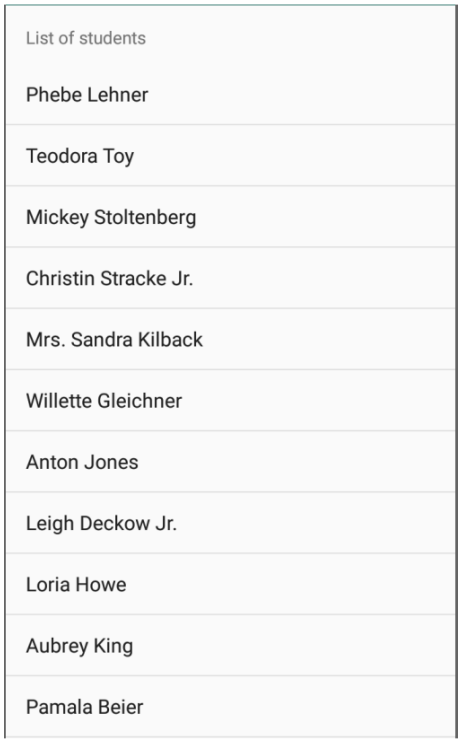

Рисунок 1.15 – Списковий компонент ListView

ArrayAdapter – найпростіший адаптер. У якості параметрів він приймає макет елементу списку (у наведеному прикладі використовується макет з поставки Android SDK: android.R.layout.simple_list_item_1), список даних або масив для відображення. Варто зазначити, що це не обов'язково має бути масив текстових рядків. Можна передавати масиви та списки будь-якого типу. Для отримання текстового значення буде використовуватись метод toString() на кожному об'єкті.

Якщо необхідно відобразити більш складні елементи списку, аніж просто текстові рядки, або якщо постачальником даних виступає не масив та не колекція List, тоді слід використовувати адаптер на базі BaseAdapter. BaseAdapter – це абстрактний клас, при його використанні необхідно реалізувати наступні методи:

- getCount() – повертає кількість елементів списку;
- getItem(int position) – повертає елемент заданої позиції списку;
- getItemId(int position) – повертає ідентифікатор елементу в заданій позиції. Використовується для оптимізації швидкості роботи списку;
- getView(int position, View convertView, ViewGroup parent) – повертає інтерфейсне відображення елементу списку.

Наприклад, відобразимо список студентів з фотографією, іменем на групою.

Створимо макет елементу списку layout/item_student:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingTop="@dimen/item_vertical_padding"
    android:paddingBottom="@dimen/item_vertical_padding"
    android:paddingStart="@dimen/item_horizontal_padding"
    android:paddingEnd="@dimen/item_horizontal_padding">

    <ImageView
        android:id="@+id/avatarImageView"
        android:layout_width="@dimen/avatar_size"
        android:layout_height="@dimen/avatar_size"
        tools:src="@mipmap/ic_launcher"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>

    <TextView
        android:id="@+id/fullNameTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="@dimen/text_size"
        android:textColor="@color/primaryText"
        android:layout_marginStart="@dimen/text_margin"
        tools:text="John Smith"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/groupTextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf=
              "@id/avatarImageView"/>

    <TextView
        android:id="@+id/groupTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:textColor="@color/secondaryText"
        android:textSize="@dimen/secondary_text_size"
        tools:text="Group: KI-101"
        app:layout_constraintStart_toStartOf=
               "@id/fullNameTextView"
        app:layout_constraintTop_toBottomOf=
               "@id/fullNameTextView"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Додамо клас Student, який міститиме інформацію про студента:
Java:

```
public class Student {
    private long id;
    private String fullName;
    private String group;
    private String imageUrl;

    public Student(long id, String fullName, String group,
                   String imageUrl) {
        this.id = id;
        this.fullName = fullName;
        this.group = group;
        this.imageUrl = imageUrl;
    }

    public long getId() {
        return id;
    }

    public String getFullName() {
        return fullName;
    }

    public String getGroup() {
        return group;
    }

    public String getImageUrl() {
        return imageUrl;
    }
}
```

Kotlin:

```
data class Student(
    val id: Long,
    val fullName: String,
    val group: String,
    val imageUrl: String
)
```

Створимо адаптер для списку студентів на основі BaseAdapter:
Java:

```
public class StudentsAdapter extends BaseAdapter {

    private Context context;
    private List<Student> students;

    public StudentsAdapter(Context context,
                           List<Student> students) {
        this.context = context;
        this.students = students;
    }

    @Override
    public int getCount() {
        return students.size();
    }

    @Override
    public Student getItem(int position) {
        return students.get(position);
    }

    @Override
    public long getItemId(int position) {
        return getItem(position).id;
    }

    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
        ViewHolder holder;
        if (convertView == null) {
            // UI елементу списку не ініціалізований
            convertView = LayoutInflater.from(context)
                .inflate(R.layout.item_student, parent, false);
            holder = new ViewHolder();
            holder.avatarImageView = convertView
                .findViewById(R.id.avatarImageView);
            holder.fullNameTextView = convertView
                .findViewById(R.id.fullNameTextView);
            holder.groupTextView = convertView
                .findViewById(R.id.groupTextView);
            convertView.setTag(holder);
        } else {
            // UI елементу списку вже був ініціалізований
            // раніше
            holder = (ViewHolder) convertView.getTag();
        }

        Student student = getItem(position);
        String fullName = student.getFullName();
        String group = context
            .getString(R.string.student_group, student.group);

        holder.fullNameTextView.setText(fullName);
        holder.groupTextView.setText(group);

        // для завантаження зображень з мережі Інтернет
```

```
        // використовуваємо бібліотеку Glide:
        Glide.with(context)
            .load(student.getImageUrl())
            .apply(RequestOptions.circleCropTransform())
            .into(holder.avatarImageView);

        return convertView;
    }

    static class ViewHolder {
        ImageView avatarImageView;
        TextView fullNameTextView;
        TextView groupTextView;
    }
}
```

Kotlin:

```
class StudentsAdapter(
    private val context: Context,
    private val students: List<Student>
) : BaseAdapter() {


    override fun getCount() = students.size

    override fun getItem(position: Int) = students[position]

    override fun getItemId(position: Int) =
        getItem(position).id

    override fun getView(position: Int, convertView: View?,
                         parent: ViewGroup): View {
        // використовуєму існуючий елемент, якщо він існує,
        // або створюємо новий
        val view = convertView ?: LayoutInflater.from(context)
            .inflate(R.layout.item_student, parent, false)

        // використовуємо закешований ViewHolder, якщо такий
        // існує, або створюємо новий ViewHolder
        val holder = view.tag as? ViewHolder ?:
            ViewHolder().apply {
              avatarImageView = view
                .findViewById(R.id.avatarImageView)
              fullNameTextView = view
                .findViewById(R.id.fullNameTextView)
              groupTextView = view
                .findViewById(R.id.groupTextView)
              view.tag = this
            }

        val student = getItem(position)
        val fullName = student.fullName
        val group = context
             .getString(R.string.student_group, student.group)

        holder.fullNameTextView.text = fullName
        holder.groupTextView.text = group
```

```
            // Бібліотека Glide використовується для завантаження
            // зображень
            Glide.with(context)
                .load(student.imageUrl)
                .apply(RequestOptions.circleCropTransform())
                .into(holder.avatarImageView)

            return view
        }

        class ViewHolder {
            lateinit var avatarImageView: ImageView
            lateinit var fullNameTextView: TextView
            lateinit var groupTextView: TextView
        }
    }
```

Згенеруємо список даних та передамо їх до адаптеру в методі активності onCreate:

Java:

```
    public class MainActivity extends AppCompatActivity {

        private Faker faker = Faker.instance(new Random(1));

        @Override
        protected void onCreate(@Nullable Bundle savedState) {
            super.onCreate(savedState);
            setContentView(R.layout.activity_main);

            List<Student> students = generateRandomStudents(100);
            ListView studentListView =
                        findViewById(R.id.studentsListView);
            ListAdapter adapter =
                       new StudentsAdapter(this, students);
            studentListView.setAdapter(adapter);
        }

        private List<Student> generateRandomStudents(int count) {
            List<Student> students = new ArrayList<>(count);
            for (int i = 1; i <= count; i++) {
                Student student = new Student(
                    i,
                    faker.name().fullName(),
                    faker.numerify("KI-10#"),
                    faker.avatar().image()
                );
                students.add(student);
            }
            return students;
        }
    }
```

Kotlin:

```
class MainActivity : AppCompatActivity() {

    private val faker = Faker.instance(Random(1))

    override fun onCreate(savedState: Bundle?) {
        super.onCreate(savedState)
        setContentView(R.layout.activity_test)

        val students = generateRandomStudents(100)
        val studentListView =
              findViewById<ListView>(R.id.studentsListView)
        val adapter = StudentsAdapter(this, students)

        studentListView.adapter = adapter
    }

    private fun generateRandomStudents(count:Int): List<Student>{

        return (1..count).map { Student(
            it.toLong(),
            faker.name().fullName(),
            faker.numerify("KI-10#"),
            faker.avatar().image()
        )}

    }
}
```

На рисунку нижче зображено результуючий інтерфейс списку:

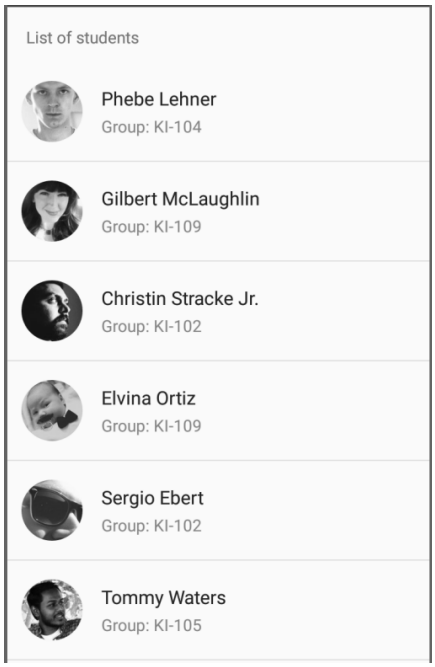

Рисунок 1.16 – Список на основі BaseAdapter

1.3 Хід виконання роботи

В рамках даної лабораторної роботи необхідно:

1) Встановити середовище розробника:
   - Android SDK
   - емулятор (Android Virtual Device – AVD)
   - IDE Android Studio
2) Створити Hello World проект, ознайомитись з типовою структурою проекту та системою збирання проектів Gradle, запустити проект на віртуальному та реальному пристроях;
3) Ознайомитись з основними поняттями, компонентом Activity, візуальними компонентами, обробкою подій, життєвим циклом Activity та Fragment, навігацією, мовою програмування Kotlin (або Java);
4) Зібрати та запустити Android-додаток для генерації номеру варіанту завдання;
5) Розробити Android-додаток згідно варіанту завдання, отриманого в результаті виконання п.п. 4.

1.3.1 Встановлення середовища розробника

Тут і далі всі кроки виконуються на операційні системі Ubuntu 18.04 (x64). Для виконання всіх лабораторних робіт рекомендується використовувати саме Linux-based операційні системи.

Є два способи підготовки середовища розробника. Перший спосіб – встановити Android Studio IDE, яка в процесі свого встановлення сама завантажить та встановить необхідні компоненти середовища розробника. Другий спосіб – встановити компоненти самостійно. Перший спосіб не є нетривіальним, достатньо дотримуватись інструкцій. Тому розглянемо саме другий спосіб, оскільки він також надає більше розуміння того, що собою представляє середовище розробника.

1) Встановлюємо JDK 1.8 (не обов'язково, але рекомендується):

```
$ sudo apt -y install openjdk-8-jdk
```

Перевірка коректності встановлення:

```
$ java -version
openjdk version "1.8.0_232"
OpenJDK Runtime Environment (build 1.8.0_232-8u232-b09-
0ubuntu1~18.04.1-b09)
OpenJDK 64-Bit Server VM (build 25.232-b09, mixed mode)
```

2) Встановлюємо Android SDK Tools.

Можна завантажити архів з сайту https://developer.android.com/studio та розпакувати його вручну:

Platform	SDK tools package	Size	SHA-256 checksum
Windows	sdk-tools-windows-4333796.zip	148 MB	7e81d69c303e47a4f0e748a6352d85cd0c8fd90a5a95ae4e076b5e5f960d3c7a
Mac	sdk-tools-darwin-4333796.zip	98 MB	ecb29358bc0f13d7c2fa0f9290135a5b608e38434aad9bf7067d0252c160853e
Linux	sdk-tools-linux-4333796.zip	147 MB	92ffee5a1d98d856634e8b71132e8a95d96c83a63fde1099be3d86df3106def9

Рисунок 1.17 – Встановлення SDK Tools

Інший варіант – завантажити та розпакувати за допомогою команд:

```
$ export ANDROID_HOME=$HOME/android-sdk
$ mkdir $ANDROID_HOME && cd $_
$ wget -nc -O sdk-tools.zip \
    https://dl.google.com/android/repository/sdk-tools-linux-
4333796.zip
$ unzip sdk-tools.zip -d .
```

В наведеному прикладі SDK Tools буде встановлено до каталогу користувача: /home/<username>/android-sdk/tools.

Далі необхідно запустити sdk-manager та встановити необхідні компоненти. SDK Manager знаходиться за адресою:

```
/home/<username>/android-sdk/tools/bin/sdk-manager
```

Мінімальний набір компонентів для встановлення станом на початок 2020 року:

```
build-tools;29.0.2
add-ons;addon-google_apis-google-24
add-ons;addon-google_gdk-google-19
tools
sources;android-29
platforms;android-29
extras;android;m2repository
extras;m2repository;com;android;support;constraint;constraint-
    layout;1.0.2
extras;m2repository;com;android;support;constraint;constraint-
    layout-solver;1.0.2
lldb;3.1
patcher;v4
platform-tools
platforms;android-29
extras;google;google_play_services
extras;google;m2repository
extras;google;instantapps
extras;android;gapid;3
extras;google;auto
extras;google;market_apk_expansion
extras;google;market_licensing
extras;google;play_billing
extras;google;simulators
extras;google;webdriver
```

Курсивом виділено пакети, які не є необхідними для виконання циклу лабораторних дисциплін.

Також, якщо ви плануєте використовувати віртуальні пристрої, то необхідно встановити як мінімум один з наступних пакетів, в залежності від розрядності вашої операційної системи та бажаних функцій. Для виконання лабораторних робіт достатньо встановлення одного з двох перших пакетів:

```
system-images;android-29;google_apis;x86
system-images;android-29;google_apis;x86_64
system-images;android-29;google_apis_playstore;x86
system-images;android-29;google_apis_playstore;x86_64
```

Версії можуть змінюватись. Щоб передивитись список актуальних пакетів, необхідно виконати команду:

```
$ sdkmanager --list
```

Увага, для того щоб успішно запустити SDK Manager, необхідно або перейти до каталогу, де він знаходиться, або додати каталог до змінної PATH.

В першому випадку:

```
$ cd $ANDROID_HOME/tools/bin
$ ./sdkmanager --list
```

В другому випадку:

```
$ export PATH=$PATH:$ANDROID_HOME/tools/bin
$ sdkmanager --list
```

Тут і далі використовується саме другий спосіб.

Перед встановленням необхідно прийняти ліцензії за допомогою команди (натиснувши клавішу Y на всі запитання):

```
$ sdkmanager --licenses
```

В результаті виконання даної команди буде створено каталог $ANDROID_HOME/licenses з файлами, які містять хеші ліцензій.

Далі створюємо файл install.sh зі списком необхідних пакетів, наприклад:

```
#!/bin/bash
packages=("build-tools;29.0.2"
  "add-ons;addon-google_apis-google-24"
  "add-ons;addon-google_gdk-google-19"
  "tools"
  "sources;android-29"
  "platforms;android-29"
  "extras;android;m2repository"
```

```
        "extras;m2repository;com;android;support;constraint;constraint-
layout;1.0.2"
        "extras;m2repository;com;android;support;constraint;constraint-
layout-solver;1.0.2"
        "lldb;3.1"
        "patcher;v4"
        "platform-tools"
        "platforms;android-29"
        "system-images;android-29;google_apis;x86_64")

      for package in "${packages[@]}"; do
        sdkmanager "$package"
      done
```

Встановлюємо пакети:

```
$ chmox +x install.sh
$ ./install.sh
```

3) Встановлюємо Android Studio IDE, шляхом завантаження з сайту https://developer.android.com/studio або за допомогою команд:

```
$ cd $HOME
$ wget -nc -O android-studio.tar.gz \
   https://dl.google.com/dl/android/studio/ide-
zips/3.5.3.0/android-studio-ide-191.6010548-linux.tar.gz
$ tar -xvzf android-studio.tar.gz
```

В результаті Android Studio IDE буде знаходитись в каталозі: /home/<username>/android-studio

Для запуску Android Studio необхідно виконати bash-скрипт studio.sh в каталозі /home/<username>/android-studio/bin/

```
$ cd $HOME/android-studio/bin
$ ./studio.sh
```

Перший раз Android Studio запитає, чи необхідно імпортувати існуючі налаштування (виберіть опцію «Do not import settings», якщо Android Studio встановлена вперше):

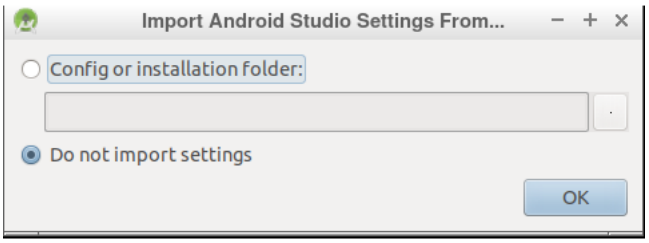

Рисунок 1.18 – Імпортування налаштувань

На наступному кроці, ви можете дозволити або заборонити відправляти анонімну статистику щодо використання вами IDE Android Studio:

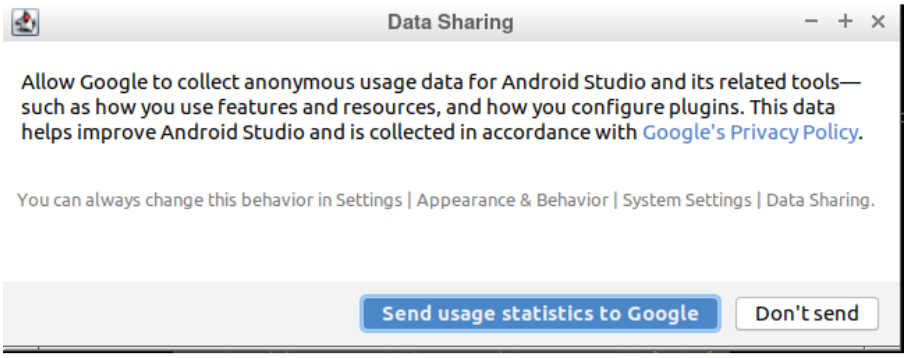

Рисунок 1.19 – Дозвіл на відправку статистики користування Android Studio

Далі запускається Setup Wizard, де на етапі вибору типу установки необхідно вибрати «Custom» та вказати шлях до Android SDK:

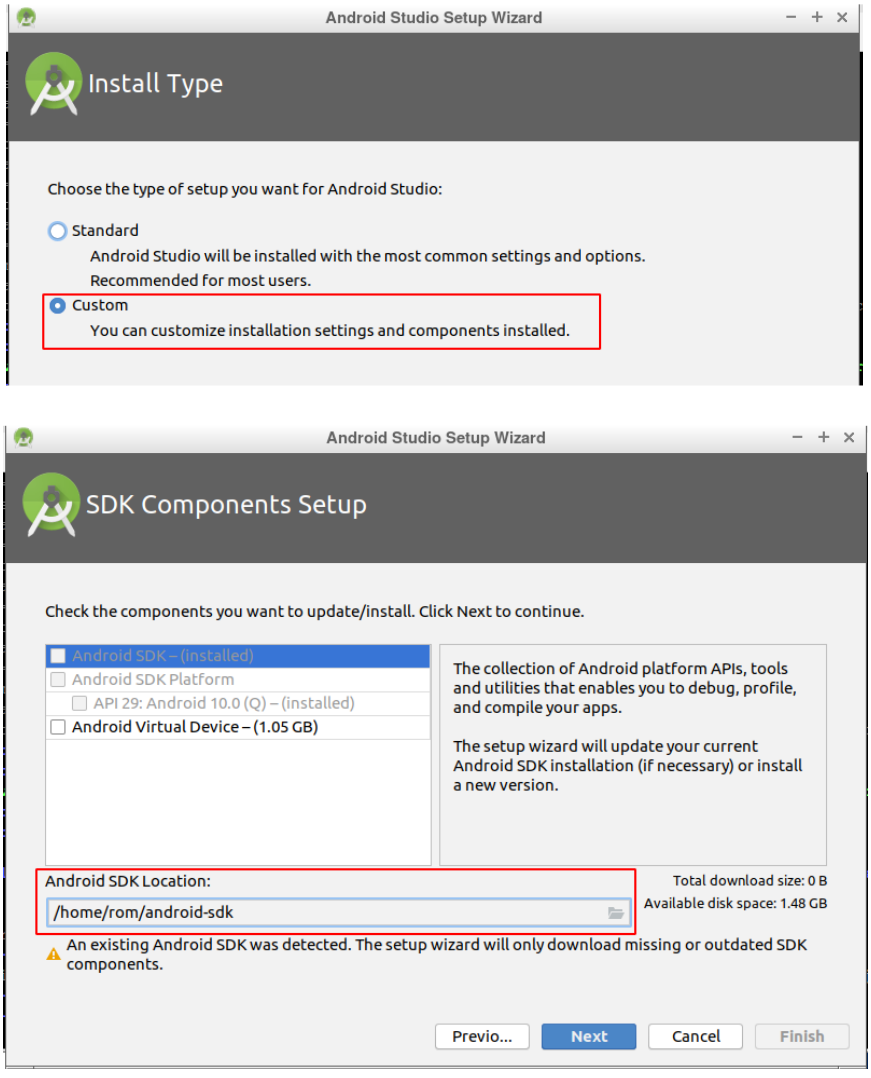

Рисунок 1.20 – Налаштування Android Studio

В результаті виконання всіх попередніх інструкцій ви повинні побачити стартове вікно Android Studio IDE:

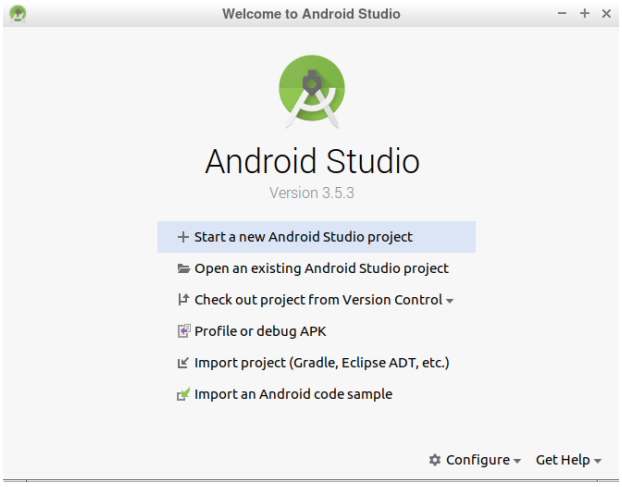

Рисунок 1.21 – Успішний запуск Android Studio

1.3.2 Створення Android-проекту

Для створення нового проекту виберіть пункт «Start a new Android Studio project». На наступному етапі можна відразу додати до проекту одну активність (Activity), виберіть «Empty Activity» - найпростішу активність.

Activity – це один з чотирьох програмних компонентів Android (більш детально програмні компоненти розглядаються в наступній лабораторній роботі), який відповідає за відображення інтерфейсу користувача. В деякому наближенні, активності можна вважати аналогом вікон в десктопних операційних системах.

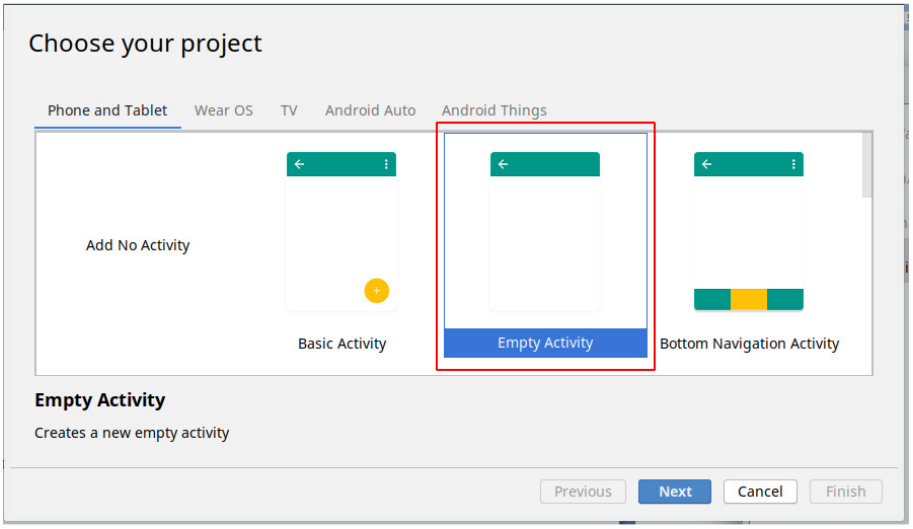

Рисунок 1.22 – Створення активності (Activity)

На наступному етапі треба вказати дані проекту: його назву, пакет (унікальний ідентифікатор), каталог проекту, мова програмування та мінімальна підтримувана версія Android.

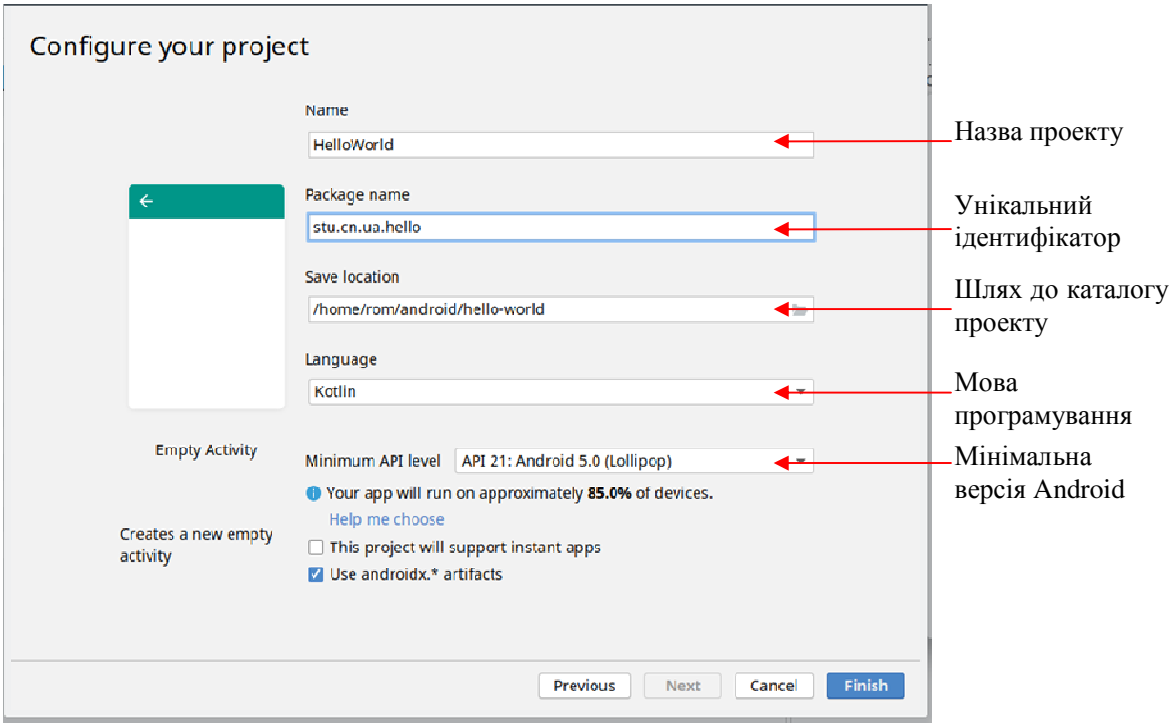

Рисунок 1.23 – Конфігурація нового проекту

Опціонально можна ввімкнути підтримку функції Instant Apps (додатки, які не потребують обов'язкового встановлення на пристрої, а можуть бути запущені відразу) та використання артефактів AndroidX – нової системи іменувань, яка прийшла на заміну бібліотек підтримки Android Support Library.

Після натискання на кнопку «Finish» проект деякий час буде ініціалізуватись (цей процес може зайняти від 1 до 10 хвилин). Проект складатиметься з:

1) Модулю Android Application з назвою «app»;

2) Файлів конфігурації збирання проекту: *.gradle

3) Файл .gitignore, в якому міститься спиок каталогів та файлів, які не повинні бути включені до системи контролю версій;

4) Файл конфігурації обфускатора та мініфікатора ProGuard;

5) Файл локальних налаштувань local.properties;

6) Скрипт Gradle Wrapper: gradlew;

В результаті ви побачите основну робочу область Android Studio:

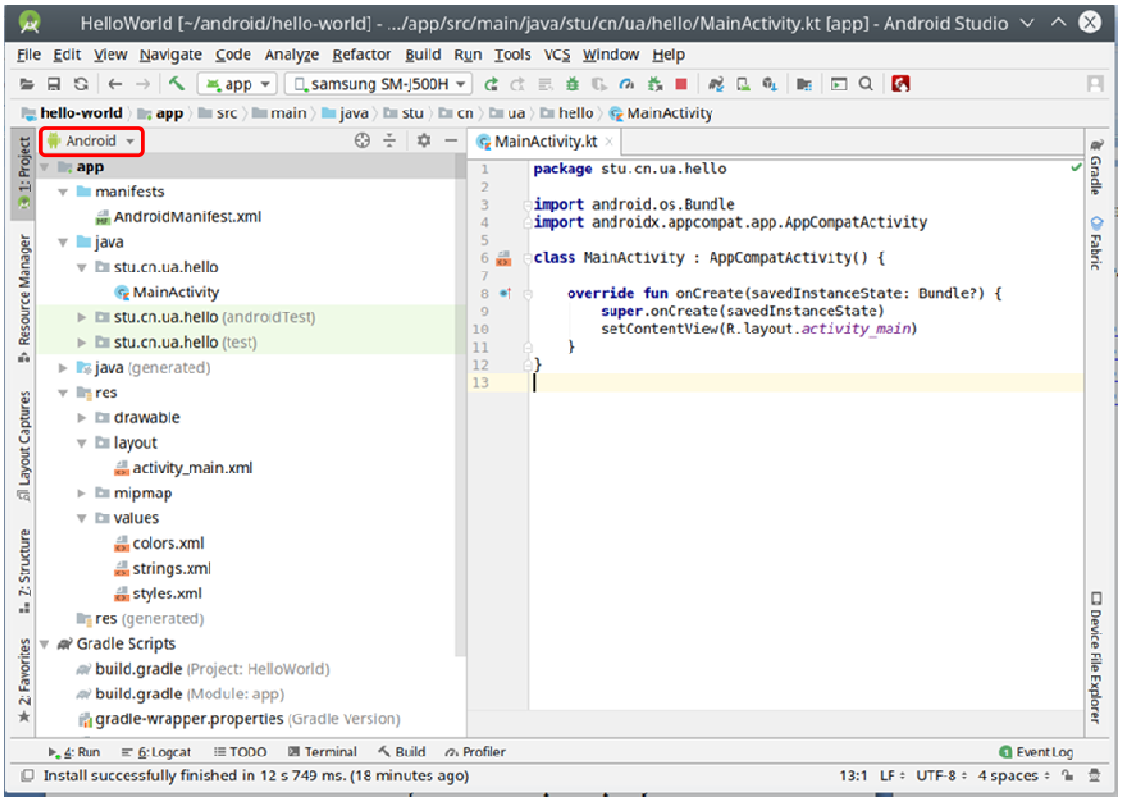

Рисунок 1.24 – Android Studio

Зліва відображається структура проекту (бажано вибрати вкладку Android).

1.3.3 Розробка першого додатку

Створимо додаток, який складається з трьох екранів та надає можливість ввести інформацію про студента, на базі якої генерується варіант для виконання лабораторних робіт.

Навігацію побудуємо на базі фрагментів. Екрани наступні:

1) Екран меню: містить назву додатку, логотип, версію та три кнопки: «Get Variant» - отримати варіант, «Options» - перейти до екрану введення даних про студента, «Quit» - вихід з додатку. Перша кнопка «Get Variant» після старту неактивна, оскільки для того, щоб отримати варіант, необхідно ввести спочатку дані на екрані Options. Макет даного екрану побудуємо на базі LinearLayout.

2) Екран введення інформації про студента «Options»: містить поля для вводу імені та прізвища, а також випадаюче меню вибору групи. Макет екрану побудуємо на базі ConstraintLayout.

3) Екран результатів: відображає інформацію, введену на екрані «Options», а також номер згенерованого варіанту. Макет екрану побудуємо на базі RelativeLayout та TableLayout.

Приблизний вигляд екранів має бути таким:

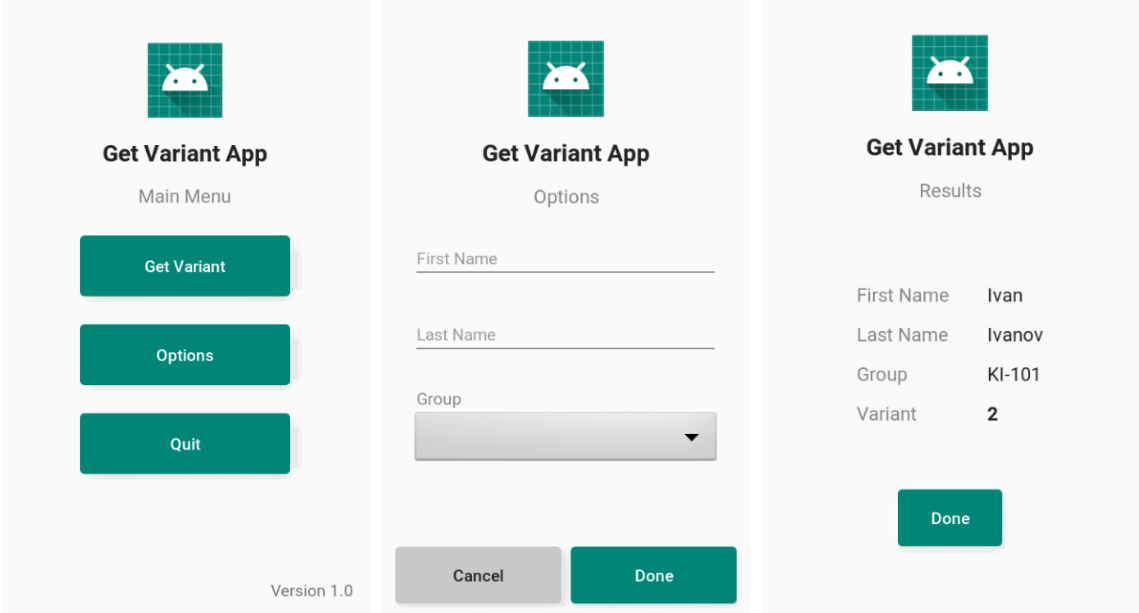

Рисунок 1.25 – Інтерфейс користувача

Ініціалізація залежностей. До кореневого файлу build.gradle (який лежить в каталозі проекту) додаємо репозиторій, в якому зберігається бібліотека генерування варіантів. Для цього в блок `allprojects.repositories` додаємо наступні рядки:

```
maven {
    url "https://mvn.elveum.com/repository/cntu"
}
```

Далі, у файлі build.gradle вашого модулю проекту (зазвичай модуль має назву «app») до блоку `dependencies` додаємо бібліотеку генерування варіантів:

```
implementation "ua.cn.stu.getvariant:getvariant:0.1"
```

В цьому ж файлі до блоку `android` додамо рядки конфігурації для включення підтримки Java 8:

```
compileOptions {
    sourceCompatibility = 1.8
    targetCompatibility = 1.8
}
```

Створення ресурсних файлів. При створенні проекту автоматично до нього додаються наступні ресурсні файли: strings.xml, colors.xml, styles.xml.

Додамо ще один додатковий файл dimens.xml, в якому будемо зберігати розміри та відступи візуальних компонентів: правою кнопкою натискаємо на назву модуля app, вибираємо пункт New → Android Resource File:

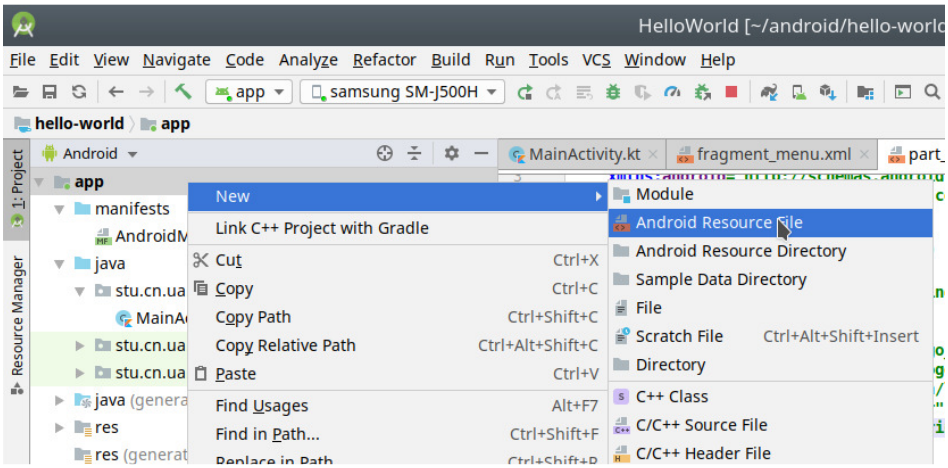

Рисунок 1.26 – Створення нового файлу ресурсів

Далі у вікні вводимо назву dimens, вибираємо тип ресурсу «Values»:

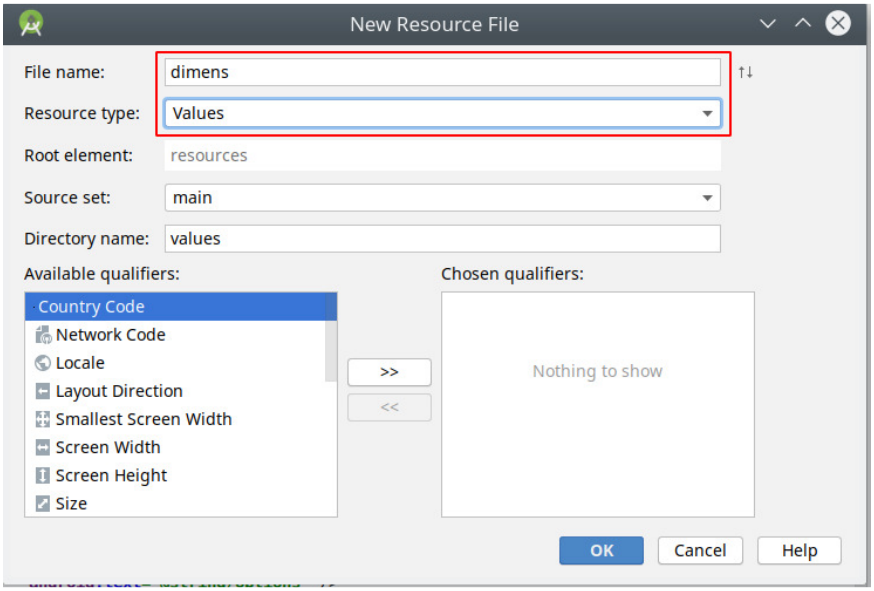

Рисунок 1.27 – Створення нового файлу ресурсів

До новоствореного файлу dimens.xml додаємо значення розмірів та відступів візуальних компонентів, які будемо використовувати у проекті:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="screenPadding">16dp</dimen>
    <dimen name="buttonRadius">4dp</dimen>
    <dimen name="menuButtonWidth">180dp</dimen>
    <dimen name="inputWidth">260dp</dimen>
```

```
    <dimen name="inputMargin">24dp</dimen>
    <dimen name="hintMargin">4dp</dimen>
    <dimen name="appNameTextSize">20sp</dimen>
    <dimen name="logoSize">64dp</dimen>
    <dimen name="logoMargin">24dp</dimen>
    <dimen name="appTitleMargin">16dp</dimen>
    <dimen name="menuItemSpace">24dp</dimen>
    <dimen name="menuItemHeight">52dp</dimen>
    <dimen name="buttonMargin">8dp</dimen>
    <dimen name="bottomButtonsSpace">4dp</dimen>
</resources>
```

До файлу strings.xml додаємо всі текстові рядки, які будуть використовуватись додатком:

```
<resources>
    <string name="app_name">Get Variant App</string>
    <string name="app_logo">App Logo</string>
    <string name="app_version">App Version: %1$s</string>
    <string name="menu">Main Menu</string>
    <string name="get_variant">Get Variant</string>
    <string name="results">Results</string>
    <string name="options">Options</string>
    <string name="quit">Quit</string>
    <string name="first_name">First Name</string>
    <string name="last_name">Last Name</string>
    <string name="group">Group</string>
    <string name="variant">Variant</string>
    <string name="done">Done</string>
    <string name="try_again">Try Again</string>
    <string name="cancel">Cancel</string>
    <string name="empty_fields_error">Some fields are
empty</string>
    <string name="error">Error! Please check your internet
connection and try again</string>
</resources>
```

До файлу colors.xml додаємо кольори, які також будуть використовуватись нашим додатком:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="primaryText">#232323</color>
    <color name="secondaryText">#868686</color>
    <color name="button">@color/colorPrimary</color>
    <color name="buttonPressed">@color/colorPrimaryDark</color>
    <color name="buttonDisabled">#c0c0c0</color>
    <color name="secondaryButton">#c8c8c8</color>
    <color name="secondaryButtonPressed">#a8a8a8</color>
    <color name="secondaryButtonDisabled">#c8c8c8</color>
    <color name="buttonText">@android:color/white</color>
    <color name="buttonTextDisabled">#989898</color>
    <color name="secondaryButtonText">@color/primaryText</color>
```

```
    <color name="secondaryButtonTextDisabled">#a0a0a0</color>
    <color name="error">#ef0022</color>
</resources>
```

Створимо свої ресурси для відображення кнопок у різних станах. За допомогою тієї ж самої команди New → Android Resource File: вибираємо тип ресурсу (Resource Type) = «Drawable» та створюємо файли: button_background.xml, button_text.xml, secondary_button_background.xml та secondary_button_text.xml.

Текст button_background.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="false">
        <shape android:shape="rectangle">
            <solid android:color="@color/buttonDisabled" />
            <corners android:radius="@dimen/buttonRadius" />
        </shape>
    </item>
    <item android:state_pressed="true">
        <shape android:shape="rectangle">
            <solid android:color="@color/buttonPressed" />
            <corners android:radius="@dimen/buttonRadius" />
        </shape>
    </item>
    <item>
        <shape android:shape="rectangle">
            <solid android:color="@color/button" />
            <corners android:radius="@dimen/buttonRadius" />
        </shape>
    </item>
</selector>
```

Текст button_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_enabled="false"
        android:color="@color/buttonTextDisabled" />
    <item
        android:color="@color/buttonText" />
</selector>
```

Текст secondary_button_background.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="false">
        <shape android:shape="rectangle">
            <solid
                android:color="@color/secondaryButtonDisabled" />
```

```
                <corners android:radius="@dimen/buttonRadius" />
            </shape>
        </item>
        <item android:state_pressed="true">
            <shape android:shape="rectangle">
                <solid
                    android:color="@color/secondaryButtonPressed" />
                <corners android:radius="@dimen/buttonRadius" />
            </shape>
        </item>
        <item>
            <shape android:shape="rectangle">
                <solid android:color="@color/secondaryButton" />
                <corners android:radius="@dimen/buttonRadius" />
            </shape>
        </item>
    </selector>
```

Текст secondary_button_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_enabled="false"
        android:color="@color/secondaryButtonTextDisabled" />
    <item
        android:color="@color/secondaryButtonText" />
</selector>
```

У файлі styles.xml реєструємо загальні стилі візуальних компонентів нашого додатку:

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="PrimaryButton">
        <item name="android:textAllCaps">false</item>
        <item name="android:background">@drawable/button_background</item>
        <item name="android:textColor">@drawable/button_text</item>
    </style>

    <style name="SecondaryButton">
        <item name="android:textAllCaps">false</item>
        <item
name="android:background">@drawable/secondary_button_background</item>
        <item name="android:textColor">@color/secondaryButtonText</item>
    </style>

    <style name="MenuButton" parent="PrimaryButton">
        <item name="android:minWidth">@dimen/menuButtonWidth</item>
        <item name="android:minHeight">@dimen/menuItemHeight</item>
        <item name="android:layout_marginTop">@dimen/menuItemSpace</item>
    </style>
```

```
    <style name="ScreenTitle">
        <item name="android:textColor">@color/secondaryText</item>
        <item name="android:textSize">16sp</item>
        <item name="android:layout_marginTop">12dp</item>
    </style>

    <style name="TextHint">
        <item name="android:textColor">@color/secondaryText</item>
        <item name="android:textSize">14sp</item>
    </style>

    <style name="Input">
        <item name="android:textSize">14sp</item>
        <item name="android:lines">1</item>
        <item name="android:inputType">textPersonName</item>
    </style>

    <style name="TextHint.Results">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:layout_marginTop">12dp</item>
        <item name="android:textSize">16sp</item>
    </style>

    <style name="TextHint.Results.Value">
        <item name="android:textColor">@color/primaryText</item>
        <item name="android:layout_marginStart">32dp</item>
    </style>

</resources>
```

Створення макетів. За допомогою цієї ж самої команди New → Android Resource File створюємо макетні файли з назвами: fragment_menu, fragment_options, fragment_results, part_logo:

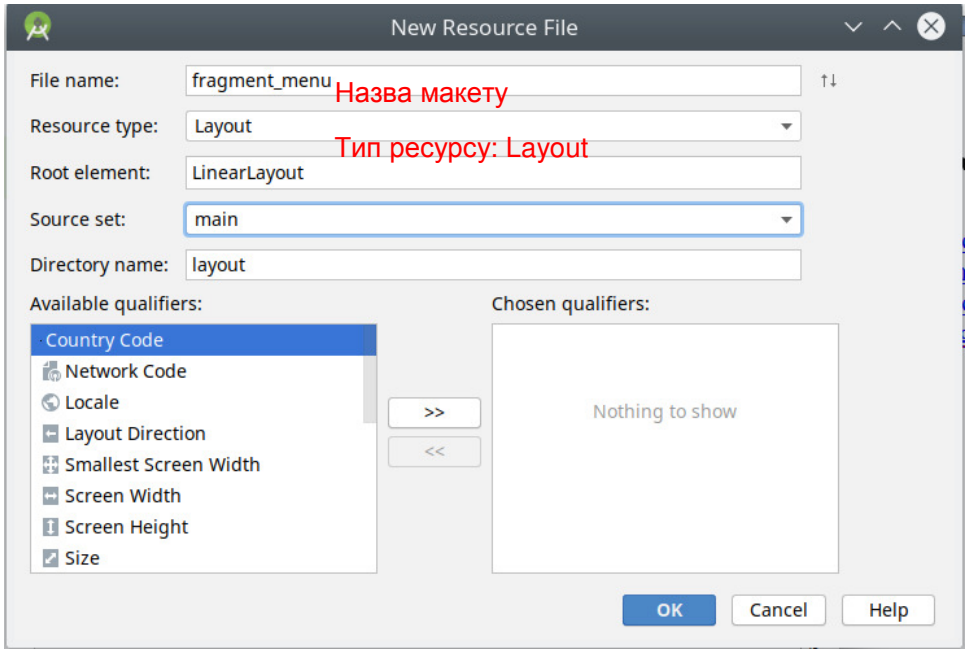

Рисунок 1.28 – Створення нового макету

Макет part_logo містить логотип та назву додатку. Винесений в окремий файл, оскільки він буде повторно використовуватись на всіх екранах:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">

    <ImageView
        android:layout_width="@dimen/logoSize"
        android:layout_height="@dimen/logoSize"
        android:layout_marginTop="@dimen/logoMargin"
        android:src="@mipmap/ic_launcher"
        android:contentDescription="@string/app_logo" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/appTitleMargin"
        android:text="@string/app_name"
        android:textStyle="bold"
        android:textSize="@dimen/appNameTextSize"
        android:textColor="@color/primaryText"/>

</LinearLayout>
```

Файл fragment_menu.xml містить макет екрану меню:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:padding="@dimen/screenPadding">

    <include layout="@layout/part_logo" />

    <TextView
        android:id="@+id/titleTextView"
        style="@style/ScreenTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/menu" />

    <Button
        android:id="@+id/getVariantButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/get_variant" />
```

```
    <Button
        android:id="@+id/optionsButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/options" />

    <Button
        android:id="@+id/quitButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/quit" />

    <TextView
        android:id="@+id/versionTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="end|bottom"
        tools:text="Version 1.0"/>

</LinearLayout>
```

Файл fragment_options.xml містить макет екрану налаштувань:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/screenPadding">

    <include
        android:id="@+id/logo"
        layout="@layout/part_logo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/titleTextView"
        style="@style/ScreenTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/options"
        app:layout_constraintTop_toBottomOf="@id/logo"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <EditText
```

```
        android:id="@+id/firstNameEditText"
        style="@style/Input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/first_name"
        android:imeOptions="actionNext"
        app:layout_constraintVertical_bias="0.25"
        app:layout_constraintVertical_chainStyle="packed"
        app:layout_constraintTop_toBottomOf="@id/titleTextView"
        app:layout_constraintWidth_max="@dimen/inputWidth"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toTopOf=
              "@+id/lastNameEditText"/>

    <EditText
        android:id="@+id/lastNameEditText"
        style="@style/Input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/inputMargin"
        android:hint="@string/last_name"
        android:imeOptions="actionDone"
        app:layout_constraintTop_toBottomOf=
              "@id/firstNameEditText"
        app:layout_constraintBottom_toTopOf=
              "@id/groupHintTextView"
        app:layout_constraintWidth_max="@dimen/inputWidth"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <TextView
        android:id="@+id/groupHintTextView"
        style="@style/TextHint"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/group"
        android:layout_marginTop="@dimen/inputMargin"
        android:layout_marginStart="@dimen/hintMargin"
        app:layout_constraintTop_toBottomOf=
              "@id/lastNameEditText"
        app:layout_constraintStart_toStartOf=
              "@id/lastNameEditText"
        app:layout_constraintBottom_toTopOf="@id/groupSpinner"/>

    <Spinner
        android:id="@+id/groupSpinner"
        style="@style/Widget.AppCompat.Spinner.Underlined"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="@android:drawable/btn_dropdown"
        app:layout_constraintTop_toBottomOf=
               "@id/groupHintTextView"
        app:layout_constraintStart_toStartOf=
               "@id/lastNameEditText"
        app:layout_constraintEnd_toEndOf="@id/lastNameEditText"
        app:layout_constraintBottom_toTopOf="@id/doneButton"/>
```

```
    <Button
        android:id="@+id/doneButton"
        style="@style/PrimaryButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="@string/done"
        android:layout_marginBottom="@dimen/bottomButtonsSpace"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/buttonsSpace"/>

    <Space
        android:id="@+id/buttonsSpace"
        android:layout_width="@dimen/buttonMargin"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@id/doneButton"
        app:layout_constraintStart_toEndOf="@id/cancelButton"/>

    <Button
        android:id="@+id/cancelButton"
        style="@style/SecondaryButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="@string/cancel"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/buttonsSpace"
        app:layout_constraintBottom_toBottomOf="@id/doneButton"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Файл fragment_results.xml містить макет екрану результатів:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/screenPadding">

    <include
        android:id="@+id/logo"
        layout="@layout/part_logo" />

    <TextView
        android:id="@+id/titleTextView"
        style="@style/ScreenTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/results"
        app:layout_constraintTop_toBottomOf="@id/logo"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
```

```
<TableLayout
    android:id="@+id/resultsTable"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="8dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/titleTextView"
    app:layout_constraintBottom_toTopOf="@+id/doneButton">

    <TableRow>
        <TextView
            style="@style/TextHint.Results"
            android:text="@string/first_name" />
        <TextView
            android:id="@+id/firstNameTextView"
            style="@style/TextHint.Results.Value"
            tools:text="Ivan" />
    </TableRow>

    <TableRow>
        <TextView
            style="@style/TextHint.Results"
            android:text="@string/last_name" />
        <TextView
            android:id="@+id/lastNameTextView"
            style="@style/TextHint.Results.Value"
            tools:text="Ivanov" />
    </TableRow>

    <TableRow>
        <TextView
            style="@style/TextHint.Results"
            android:text="@string/group" />
        <TextView
            android:id="@+id/groupTextView"
            style="@style/TextHint.Results.Value"
            tools:text="KI-101" />
    </TableRow>

    <TableRow>
        <TextView
            style="@style/TextHint.Results"
            android:text="@string/variant" />
        <TextView
            android:id="@+id/variantTextView"
            style="@style/TextHint.Results.Value"
            android:textStyle="bold"
            tools:text="2" />
    </TableRow>

</TableLayout>

<TextView
    android:id="@+id/errorTextView"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:maxWidth="200dp"
```

```
            android:textSize="16sp"
            android:gravity="center"
            android:textColor="@color/error"
            tools:text="Error"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf=
                  "@id/resultsTable"/>

        <Button
            android:id="@+id/doneButton"
            style="@style/PrimaryButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/done"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf="parent"/>

        <Button
            android:id="@+id/tryAgainButton"
            style="@style/PrimaryButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/try_again"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf="parent"/>

        <ProgressBar
            android:id="@+id/progress"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintTop_toBottomOf="@id/titleTextView"/>

    </androidx.constraintlayout.widget.ConstraintLayout>
```

Оскільки навігація в даному випадку реалізовується на базі фрагментів, то файл макету активності activity_main.xml максимально простий:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/fragmentContainer"
    tools:context=".MainActivity" />
```

Також додамо макет елементу випадаючого списку з вибором навчальної групи для екрану налаштувань (item_group.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/primaryText"
    android:textSize="14sp"
    android:gravity="center_vertical" />
```

Далі створимо структуру класів, зображену на рисунку нижче:

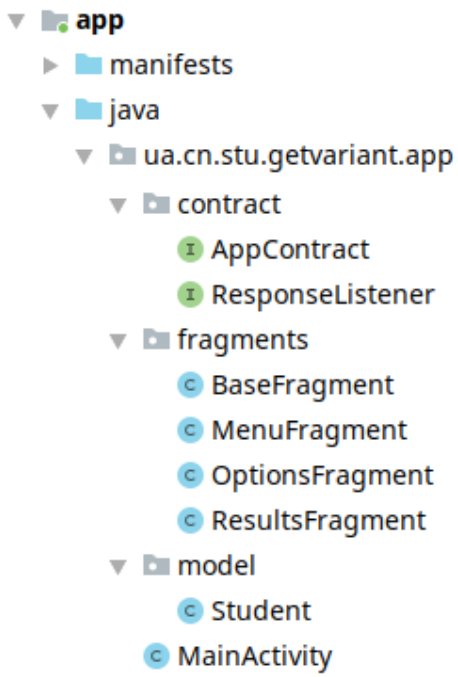

Рисунок 1.29 – Структура класів проекту

AppContract – інтерфейс, який відповідає за навігацію між екранами та передачу даних між ними. Фрагменти працюють з цим інтерфейсом і, таким чином, абстрагуються від конкретної реалізації навігації:

```
public interface AppContract {

    /**
     * Launch options screen
     * @param target fragment that launches options screen
     * @param student data about the student to be displayed
     */
    void toOptionsScreen(Fragment target,
                         @Nullable Student student);

    /**
     * Launch results screen
     * @param target fragment that launches results screen
     * @param student data used for calculating variant
```

```
         */
        void toResultsScreen(Fragment target, Student student);

        /**
         * Exit from the current screen
         */
        void cancel();

        /**
         * Publish results to the target screen
         */
        <T> void publish(T data);

        /**
         * Listen for results from other screens
         */
        <T> void registerListener(Fragment fragment, Class<T> clazz,
                                  ResponseListener<T> listener);

        /**
         * Stop listening for results from other screens
         */
        void unregisterListeners(Fragment fragment);

    }
```

ResponseListener – функціональний інтерфейс, який слугує для прослуховування відповідей від інших екранів. Фрагмент, який очікує результат від іншого фрагменту, реалізує даний інтерфейс та передає його екземпляр до методу AppContract.registerListener():

```
    public interface ResponseListener<T> {
        void onResults(T results);
    }
```

Student – DTO-клас (клас даних), який містить інформацію про студента, для якого буде розраховуватись варіант:

```
    public class Student implements Parcelable {

        public static final List<String> GROUPS = Arrays.asList(
            "КІ-161", "КІ-162", "КІ-163", "КІт-181", "МРАп-191"
        );
        public static final int MAX_VARIANT = 20;

        private String firstName;
        private String lastName;
        private String group;

        public Student(String firstName, String lastName,
                       String group) {
            this.firstName = firstName;
            this.lastName = lastName;
            this.group = group;
        }
```

```
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getGroup() {
        return group;
    }

    public boolean isValid() {
        return !isEmpty(firstName)
               && !isEmpty(lastName)
               && !isEmpty(group);
    }

    private boolean isEmpty(String str) {
        return str == null || str.trim().length() == 0;
    }

    // --- auto-generated

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(this.firstName);
        dest.writeString(this.lastName);
        dest.writeString(this.group);
    }

    protected Student(Parcel in) {
        this.firstName = in.readString();
        this.lastName = in.readString();
        this.group = in.readString();
    }

    public static final Creator<Student> CREATOR =
                          new Creator<Student>() {
        @Override
        public Student createFromParcel(Parcel source) {
            return new Student(source);
        }

        @Override
        public Student[] newArray(int size) {
            return new Student[size];
        }
    };
}
```

BaseFragment – базовий клас для всіх фрагментів додатку. Містить загальний функціонал, який використовується всіма іншими фрагментами:

```
public class BaseFragment extends Fragment {
    private AppContract appContract;

    @Override
    public void onAttach(@NonNull Context context) {
        super.onAttach(context);
        this.appContract = (AppContract) context;
    }

    @Override
    public void onDetach() {
        super.onDetach();
        this.appContract.unregisterListeners(this);
        this.appContract = null;
    }

    final AppContract getAppContract() {
        return appContract;
    }

    final <T> void registerListener(Class<T> clazz,
                                    ResponseListener<T> listener) {
        getAppContract().registerListener(this, clazz, listener);
    }

}
```

MainActivity – клас єдиної активності в додатку. Активність реалізує інтерфейс AppContract, тобто вона є відповідальною за реалізацію навігації та передачі даних між фрагментами:

```
public class MainActivity extends AppCompatActivity
                          implements AppContract {

    public static final String TAG =
                MainActivity.class.getSimpleName();

    private Map<String, List<ListenerInfo<?>>> listeners =
                new HashMap<>();

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            launchFragment(null, new MenuFragment());
        }
    }

    @Override
    public void onBackPressed() {
        cancel();
    }
```

```
        @Override
        public void toOptionsScreen(Fragment target,
                                    Student student) {
            launchFragment(target,
                           OptionsFragment.newInstance(student));
        }

        @Override
        public void toResultsScreen(Fragment target,
                                    Student student) {
            launchFragment(target,
                           ResultsFragment.newInstance(student));
        }

        @Override
        public void cancel() {
            int count = getSupportFragmentManager()
                              .getBackStackEntryCount();
            if (count <= 1) {
                finish();
            } else {
                getSupportFragmentManager().popBackStack();
            }
        }

        @Override
        public <T> void publish(T results) {
            Fragment currentFragment = getCurrentFragment();
            if (currentFragment == null) {
                Log.e(TAG, "Can't find the current fragment");
                return;
            }
            Fragment targetFragment =
                        currentFragment.getTargetFragment();
            if (targetFragment == null) {
                Log.e(TAG, "Fragment " + currentFragment +
                           " doesn't have a target");
                return;
            }

            String tag = targetFragment.getTag();
            if (tag == null) {
                Log.e(TAG, "Target fragment exists but doesn't have a
tag: " + targetFragment);
                return;
            }
            List<ListenerInfo<?>> listeners =
                                        this.listeners.get(tag);
            if (listeners != null) {
                Iterator<ListenerInfo<?>> it = listeners.iterator();
                while (it.hasNext() &&
                       !it.next().tryPublish(results));
            }
        }

        @Override
        public <T> void registerListener(Fragment fragment,
```

```
                            Class<T> clazz,
                            ResponseListener<T> listener) {
        if (fragment.getTag() == null) {
            Log.e(TAG, "Fragment '" + fragment +
                    "' doesn't have a tag");
            return;
        }
        List<ListenerInfo<?>> listeners =
                this.listeners.get(fragment.getTag());
        if (listeners == null) {
            listeners = new ArrayList<>();
            this.listeners.put(fragment.getTag(), listeners);
        }
        listeners.add(new ListenerInfo<>(clazz, listener));
    }

    @Override
    public void unregisterListeners(Fragment fragment) {
        if (fragment.getTag() == null) {
            Log.e(TAG, "Fragment '" + fragment +
                    "' doesn't have a tag");
            return;
        }
        this.listeners.remove(fragment.getTag());
    }

    private void launchFragment(@Nullable Fragment target,
                                Fragment fragment) {
        if (target != null) {
            fragment.setTargetFragment(target, 0);
        }
        String tag = UUID.randomUUID().toString();
        getSupportFragmentManager().beginTransaction()
            .addToBackStack(null)
            .replace(R.id.fragmentContainer, fragment, tag)
            .commit();
    }

    private Fragment getCurrentFragment() {
        return getSupportFragmentManager()
            .findFragmentById(R.id.fragmentContainer);
    }

    private static class ListenerInfo<T> {
        Class<T> clazz;
        ResponseListener<T> listener;

        private ListenerInfo(Class<T> clazz,
                             ResponseListener<T> listener) {
            this.clazz = clazz;
            this.listener = listener;
        }

        boolean tryPublish(Object result) {
            if (result.getClass().equals(clazz)) {
                listener.onResults((T) result);
                return true;
            }
```

```
            return false;
        }
    }

}
```

MenuFragment – клас екрану меню. Має кнопки для переходу на інші екрани. Прослуховує результати роботи екрану налаштувань та передає їх до екрану відображення результатів:

```
public class MenuFragment extends BaseFragment {
    private static final String KEY_STUDENT = "STUDENT";
    private Button getVariantButton;
    private Student student;

    @Override
    public void onAttach(@NonNull Context context) {
        super.onAttach(context);
        registerListener(Student.class, listener);
    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            student = savedInstanceState
                        .getParcelable(KEY_STUDENT);
        }
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                         @Nullable ViewGroup container,
                         @Nullable Bundle savedInstanceState) {
        return inflater.inflate(
                 R.layout.fragment_menu,
                 container,
                 false
        );
    }

    @Override
    public void onViewCreated(@NonNull View view,
                       @Nullable Bundle savedInstanceState) {

        super.onViewCreated(view, savedInstanceState);
        view.findViewById(R.id.optionsButton)
            .setOnClickListener(v -> {
              getAppContract().toOptionsScreen(this, student);
            });

        view.findViewById(R.id.quitButton)
            .setOnClickListener(v -> {
              getAppContract().cancel();
            });
```

```
        getVariantButton = view
                .findViewById(R.id.getVariantButton);
        getVariantButton.setOnClickListener(v -> {
            getAppContract().toResultsScreen(this, student);
        });
        TextView versionEditText =  view
                .findViewById(R.id.versionTextView);

        if (versionEditText != null) {
            versionEditText.setText(getString(
                R.string.app_version,
                BuildConfig.VERSION_NAME
            ));
        }
        updateView();
    }

    @Override
    public void onSaveInstanceState(@NonNull Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putParcelable(KEY_STUDENT, student);
    }

    private void updateView() {
        getVariantButton.setEnabled(student != null
              && student.isValid());
    }

    private ResponseListener<Student> listener = student -> {
        this.student = student;
    };

}
```

OptionsFragment – клас редагування даних про студента, на основі яких буде розраховуватись варіант:

```
public class OptionsFragment extends BaseFragment {
    private static final String ARG_STUDENT = "STUDENT";
    private static final String KEY_GROUP = "GROUP";

    private EditText firstNameEditText;
    private EditText lastNameEditText;
    private Spinner groupsSpinner;

    public static OptionsFragment newInstance(
            @Nullable Student student) {
        Bundle args = new Bundle();
        args.putParcelable(ARG_STUDENT, student);
        OptionsFragment fragment = new OptionsFragment();
        fragment.setArguments(args);
        return fragment;
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
```

```
                        @Nullable ViewGroup container,
                        @Nullable Bundle savedInstanceState) {
    return inflater.inflate(
            R.layout.fragment_options,
            container, false);
}

@Override
public void onViewCreated(@NonNull View view,
                      @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    firstNameEditText = view
            .findViewById(R.id.firstNameEditText);
    lastNameEditText = view
            .findViewById(R.id.lastNameEditText);
    groupsSpinner = view
            .findViewById(R.id.groupSpinner);
    setupButtons(view);

    String selectedGroup = null;
    if (savedInstanceState != null) {
        // EditText fields save/restore input automatically
        // so need to restore only Spinner data
        selectedGroup = savedInstanceState
                        .getString(KEY_GROUP);
    } else {
        Student student = getStudentArg();
        if (student != null) {
            firstNameEditText.setText(student
                 .getFirstName());
            lastNameEditText.setText(student.getLastName());
            selectedGroup = student.getGroup();
        }
    }
    setupGroupsSpinner(selectedGroup);
}
private void setupButtons(View view) {
    view.findViewById(R.id.cancelButton)
        .setOnClickListener(v -> {
            getAppContract().cancel();
        });
    view.findViewById(R.id.doneButton)
        .setOnClickListener(v -> {
            Student student = new Student(
                firstNameEditText.getText().toString(),
                lastNameEditText.getText().toString(),
                groupsSpinner.getSelectedItem().toString()
            );
            if (!student.isValid()) {
                Toast.makeText(
                    getContext(),
                    R.string.empty_fields_error,
                    Toast.LENGTH_SHORT
                ).show();
                return;
            }
            getAppContract().publish(student);
```

```
                getAppContract().cancel();
            });
    }

    private void setupGroupsSpinner(
                    @Nullable String selectedGroup) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                getContext(),
                R.layout.item_group,
                Student.GROUPS
        );
        adapter.setDropDownViewResource(
            android.R.layout.simple_dropdown_item_1line);
        groupsSpinner.setAdapter(adapter);
        if (selectedGroup != null) {
            int index = Student.GROUPS.indexOf(selectedGroup);
            if (index != -1) groupsSpinner.setSelection(index);
        }
    }

    private Student getStudentArg() {
        return getArguments().getParcelable(ARG_STUDENT);
    }

}
```

ResultsFragment – клас екрану результатів. Проводить розрахунок варіанту на основі вхідних даних та відображає їх на екрані:

```
public class ResultsFragment extends BaseFragment {
    private static final String TAG =
            ResultsFragment.class.getSimpleName();
    private static final String ARG_STUDENT = "STUDENT";
    private static final String KEY_TASK = "GET_VARIANT_TASK";

    private ViewGroup resultsTable;
    private Button doneButton;
    private Button tryAgainButton;
    private ProgressBar progress;
    private TextView variantTextView;
    private TextView errorTextView;

    private RetainManager retainManager;
    private Task<Integer> getVariantTask;

    public static ResultsFragment newInstance(Student student) {
        Bundle args = new Bundle();
        args.putParcelable(ARG_STUDENT, student);
        ResultsFragment fragment = new ResultsFragment();
        fragment.setArguments(args);
        return fragment;
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container,
```

```
                    @Nullable Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_results,
              container, false);
}

@Override
public void onViewCreated(@NonNull View view,
                    @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    resultsTable = view
        .findViewById(R.id.resultsTable);
    doneButton = view
        .findViewById(R.id.doneButton);
    tryAgainButton = view
        .findViewById(R.id.tryAgainButton);
    progress = view
        .findViewById(R.id.progress);
    variantTextView = view
        .findViewById(R.id.variantTextView);
    errorTextView = view
        .findViewById(R.id.errorTextView);
    TextView firstNameTextView = view
        .findViewById(R.id.firstNameTextView);
    TextView lastNameTextView = view
        .findViewById(R.id.lastNameTextView);
    TextView groupTextView = view
        .findViewById(R.id.groupTextView);

    Student student = getStudent();
    firstNameTextView.setText(student.getFirstName());
    lastNameTextView.setText(student.getLastName());
    groupTextView.setText(student.getGroup());

    doneButton.setOnClickListener(v ->
         getAppContract().cancel());
    tryAgainButton.setOnClickListener(v ->
         fetchVariant(true));
    fetchVariant(false);
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    getVariantTask.setListener(null);
    if (isRemoving()) {
        retainManager.destroy();
    }
}

private void fetchVariant(boolean create) {
    toPendingState();
    retainManager = new RetainManager(this);
    if (create) retainManager.delete(KEY_TASK);
    getVariantTask = retainManager.getOrCreate(KEY_TASK,
        () -> createGetVariantTask().execute());

    getVariantTask.setListener(new TaskListener<Integer>() {
```

```
        @Override
        public void onSuccess(Integer variant) {
            toSuccessState(variant);
        }

        @Override
        public void onError(Throwable error) {
            toErrorState(error);
        }
    });
}

private Task<Integer> createGetVariantTask() {
    Student student = getStudent();
    return GetVariantTasks.createGetVariantTask(
            this,
            student.getFirstName(),
            student.getLastName(),
            student.getGroup(),
            Student.MAX_VARIANT
    );
}

private void toPendingState() {
    progress.setVisibility(View.VISIBLE);
    resultsTable.setVisibility(View.INVISIBLE);
    doneButton.setVisibility(View.INVISIBLE);
    tryAgainButton.setVisibility(View.GONE);
    errorTextView.setVisibility(View.GONE);
}

private void toSuccessState(int variant) {
    doneButton.setVisibility(View.VISIBLE);
    tryAgainButton.setVisibility(View.GONE);
    resultsTable.setVisibility(View.VISIBLE);
    progress.setVisibility(View.GONE);
    errorTextView.setVisibility(View.GONE);
    variantTextView.setText(String.valueOf(variant));
}

private void toErrorState(Throwable error) {
    Log.e(TAG, "Error!", error);
    doneButton.setVisibility(View.INVISIBLE);
    tryAgainButton.setVisibility(View.VISIBLE);
    resultsTable.setVisibility(View.INVISIBLE);
    progress.setVisibility(View.GONE);
    errorTextView.setVisibility(View.VISIBLE);
    if (error instanceof IOException) {
        errorTextView.setText(R.string.error);
    } else {
        errorTextView.setText(error.getMessage());
    }
}

private Student getStudent() {
    return getArguments().getParcelable(ARG_STUDENT);
}
```

```
}
```

Додамо підтримку альбомної орієнтації на рівні ресурсів додатку. Для кожного екрану створимо окремий макет, який буде відображатись в ландшафтному режимі. Для цього виконаємо команду створення нового ресурсного файлу (натискаємо правою кнопкою на назву модуля app, вибираємо пункт New → Android Resource File). В полі File Name зазначимо те ж саме ім'я макетного файлу, для якого створюється альтернативний макет. Тобто, якщо у нас є файл макету для екрану меню, який називається fragment_menu.xml, то для створення макету для альбомної орієнтації створюємо файл з такою ж назвою: fragment_menu.xml, але при цьому в списку «Availabe qualifiers» вибираємо пункт «orientation» та натискаємо на кнопку зі стрілочками «>>». Далі у випадаючому списку вибираємо пункт «Landscape» (див. рисунок нижче):

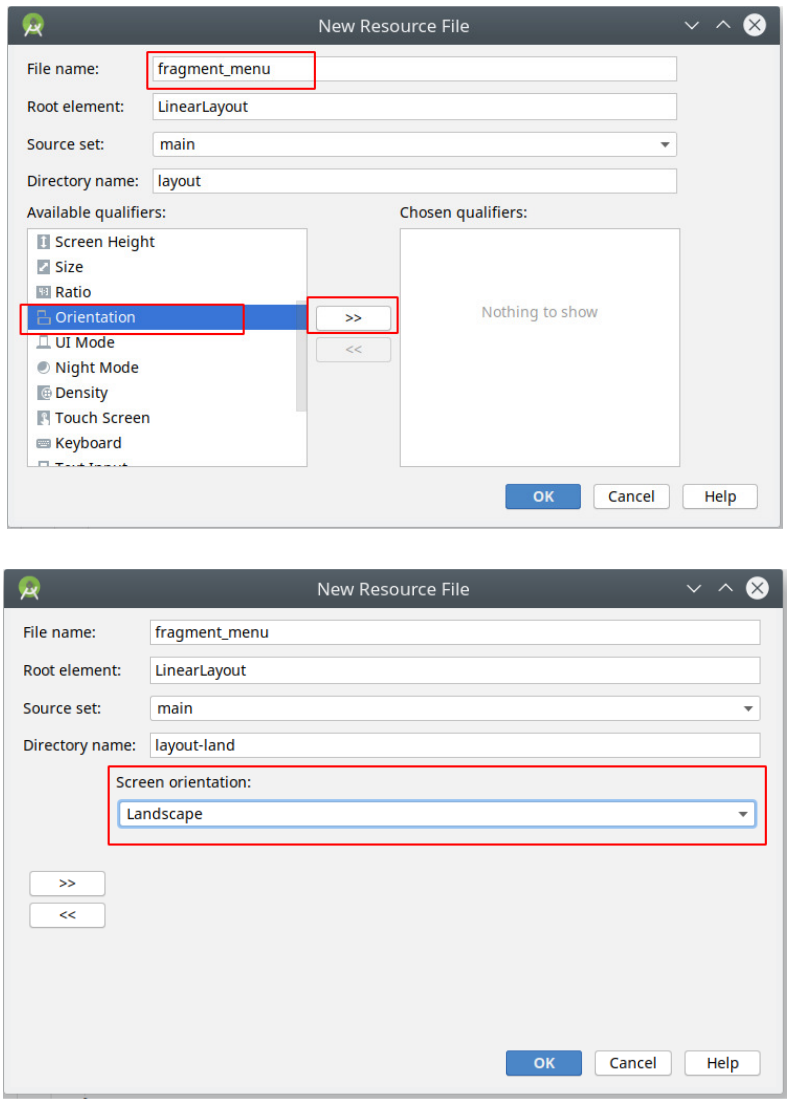

Рисунок 1.30 – Створення макету для альбомної орієнтації екрану

Аналогічно створюємо альтернативні файли для альбомної орієнтації: fragment_results.xml, fragment_options.xml, dimens.xml.

Макетний файл fragment_menu.xml (landscape):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:padding="@dimen/screenPadding">

    <include layout="@layout/part_logo" />

    <Button
        android:id="@+id/getVariantButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/get_variant" />

    <Button
        android:id="@+id/optionsButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/options" />

    <Button
        android:id="@+id/quitButton"
        style="@style/MenuButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/quit" />

</LinearLayout>
```

Макетний файл fragment_options.xml (landscape):

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/screenPadding">

    <TextView
        android:id="@+id/titleTextView"
        style="@style/ScreenTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/options"
```

```
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <EditText
        android:id="@+id/firstNameEditText"
        style="@style/Input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/first_name"
        android:imeOptions="actionNext"
        app:layout_constraintVertical_bias="0.25"
        app:layout_constraintVertical_chainStyle="packed"
        app:layout_constraintTop_toBottomOf="@id/titleTextView"
        app:layout_constraintWidth_max="@dimen/inputWidth"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toTopOf=
                "@+id/lastNameEditText"/>

    <EditText
        android:id="@+id/lastNameEditText"
        style="@style/Input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/inputMargin"
        android:hint="@string/last_name"
        android:imeOptions="actionDone"
        app:layout_constraintTop_toBottomOf=
                "@id/firstNameEditText"
        app:layout_constraintBottom_toTopOf=
                "@id/groupHintTextView"
        app:layout_constraintWidth_max="@dimen/inputWidth"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <TextView
        android:id="@+id/groupHintTextView"
        style="@style/TextHint"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/group"
        android:layout_marginTop="@dimen/inputMargin"
        android:layout_marginStart="@dimen/hintMargin"
        app:layout_constraintTop_toBottomOf=
                "@id/lastNameEditText"
        app:layout_constraintStart_toStartOf=
                "@id/lastNameEditText"
        app:layout_constraintBottom_toTopOf="@id/groupSpinner"/>

    <Spinner
        android:id="@+id/groupSpinner"
        style="@style/Widget.AppCompat.Spinner.Underlined"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="@android:drawable/btn_dropdown"
        app:layout_constraintTop_toBottomOf=
                "@id/groupHintTextView"
```

```
            app:layout_constraintStart_toStartOf=
                    "@id/lastNameEditText"
            app:layout_constraintEnd_toEndOf="@id/lastNameEditText"
            app:layout_constraintBottom_toTopOf="@id/doneButton"/>

        <Button
            android:id="@+id/doneButton"
            style="@style/PrimaryButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="@string/done"
            android:layout_marginBottom="@dimen/bottomButtonsSpace"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="@id/lastNameEditText"
            app:layout_constraintStart_toEndOf="@+id/buttonsSpace"/>

        <Space
            android:id="@+id/buttonsSpace"
            android:layout_width="@dimen/buttonMargin"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toStartOf="@id/doneButton"
            app:layout_constraintStart_toEndOf="@id/cancelButton"/>

        <Button
            android:id="@+id/cancelButton"
            style="@style/SecondaryButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="@string/cancel"
            app:layout_constraintStart_toStartOf=
                    "@id/lastNameEditText"
            app:layout_constraintEnd_toStartOf="@id/buttonsSpace"
            app:layout_constraintBottom_toBottomOf="@id/doneButton"/>

    </androidx.constraintlayout.widget.ConstraintLayout>
```

Макетний файл fragment_results.xml (landscape):

```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/screenPadding">

        <TextView
            android:id="@+id/titleTextView"
            style="@style/ScreenTitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/results"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintEnd_toEndOf="parent" />

    <TableLayout
        android:id="@+id/resultsTable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/titleTextView"
        app:layout_constraintBottom_toTopOf="@+id/doneButton">

        <TableRow>
            <TextView
                style="@style/TextHint.Results"
                android:text="@string/first_name" />
            <TextView
                android:id="@+id/firstNameTextView"
                style="@style/TextHint.Results.Value"
                tools:text="Ivan" />
        </TableRow>

        <TableRow>
            <TextView
                style="@style/TextHint.Results"
                android:text="@string/last_name" />
            <TextView
                android:id="@+id/lastNameTextView"
                style="@style/TextHint.Results.Value"
                tools:text="Ivanov" />
        </TableRow>

        <TableRow>
            <TextView
                style="@style/TextHint.Results"
                android:text="@string/group" />
            <TextView
                android:id="@+id/groupTextView"
                style="@style/TextHint.Results.Value"
                tools:text="KI-101" />
        </TableRow>

        <TableRow>
            <TextView
                style="@style/TextHint.Results"
                android:text="@string/variant" />
            <TextView
                android:id="@+id/variantTextView"
                style="@style/TextHint.Results.Value"
                android:textStyle="bold"
                tools:text="2" />
        </TableRow>

    </TableLayout>

    <TextView
        android:id="@+id/errorTextView"
        android:layout_width="0dp"
```

```
            android:layout_height="wrap_content"
            android:maxWidth="200dp"
            android:textSize="16sp"
            android:gravity="center"
            android:textColor="@color/error"
            tools:text="Error"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf=
                    "@id/resultsTable"/>

        <Button
            android:id="@+id/doneButton"
            style="@style/PrimaryButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/done"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf="parent"/>

        <Button
            android:id="@+id/tryAgainButton"
            style="@style/PrimaryButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/try_again"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/resultsTable"
            app:layout_constraintBottom_toBottomOf="parent"/>

        <ProgressBar
            android:id="@+id/progress"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintTop_toBottomOf="@id/titleTextView"/>


</androidx.constraintlayout.widget.ConstraintLayout>
```

Файл dimens.xml (landscape):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="menuItemSpace">12dp</dimen>
    <dimen name="menuItemHeight">42dp</dimen>
    <dimen name="logoMargin">12dp</dimen>
    <dimen name="logoSize">54dp</dimen>
</resources>
```

Тепер все готово для компіляції та запуску проекту, за винятком конфігурації пристрою, на якому буде запускатись додаток.

Перевірити роботу створеного додатку можна або на емуляторі, або на реальному пристрої.

Для того, щоб мати можливість запускати та підлагоджувати додаток на реальному пристрої необхідно спочатку ввімкнути режим розробника на ньому. На більшості смартфонах досягти даного результату можна натиснувши 7 разів на пункті «Build Number» в налаштуваннях телефону. Даний пункт знаходиться в різних місцях, залежно від версії ОС Android:

- Android 9: Settings → About → Build Number
- Android 8 та Android 8.1: Settings → System → About > Build Number
- Інші версії: Settings → About → Build Number

Після виконання вище перелічених дій в налаштування з'явиться новий пункт «Developer Options». Зайшовши до нього, необхідно ввімкнути верхній пункт «Developer Mode On» та увімкнути опцію «USB Debugging».

Далі підключаємо пристрій за допомогою USB-кабелю до комп'ютера в режимі «Тільки зарядка пристрою» або «Передача даних», підтверджуємо відбиток пристрою, натиснувши кнопку «Allow».

Якщо все виконано вірно, то у випадаючому списку в панелі інструментів з'явиться список підключених та готових до роботи пристроїв:

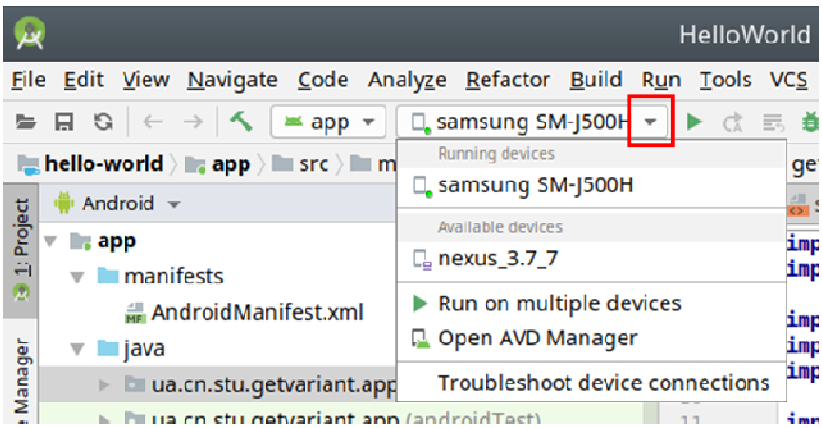

Рисунок 1.31 – Список підключених пристроїв

В цьому ж списку відображаються й доступні емулятори.

Запустити додаток можна за допомогою команди Run → Run App.

Запуск додатку з підключеним відладчиком відбувається за допомогою команди Run → Debug App.

Якщо додаток вже запущений, то до нього можна підключити відгадчик не перезапускаючи процес, за допомогою команди Run → Attach Debugger ro Android Process.

Якщо немає можливості перевіряти роботу додатку на реальному пристрої, то можна створити емулятор пристрою (Android Vurtual Device – AVD). Для цього виберіть пункт Tools → AVD Manager.

Майте на увазі, що для коректної роботи віртуальних пристроїв бажано в налаштуваннях BIOS ввімкнути підтримку апаратної віртуалізації.

Рисунок 1.32 – AVD Manager

Натискаємо кнопку «Create Virtual Device», далі вибираємо бажаний пристрій:

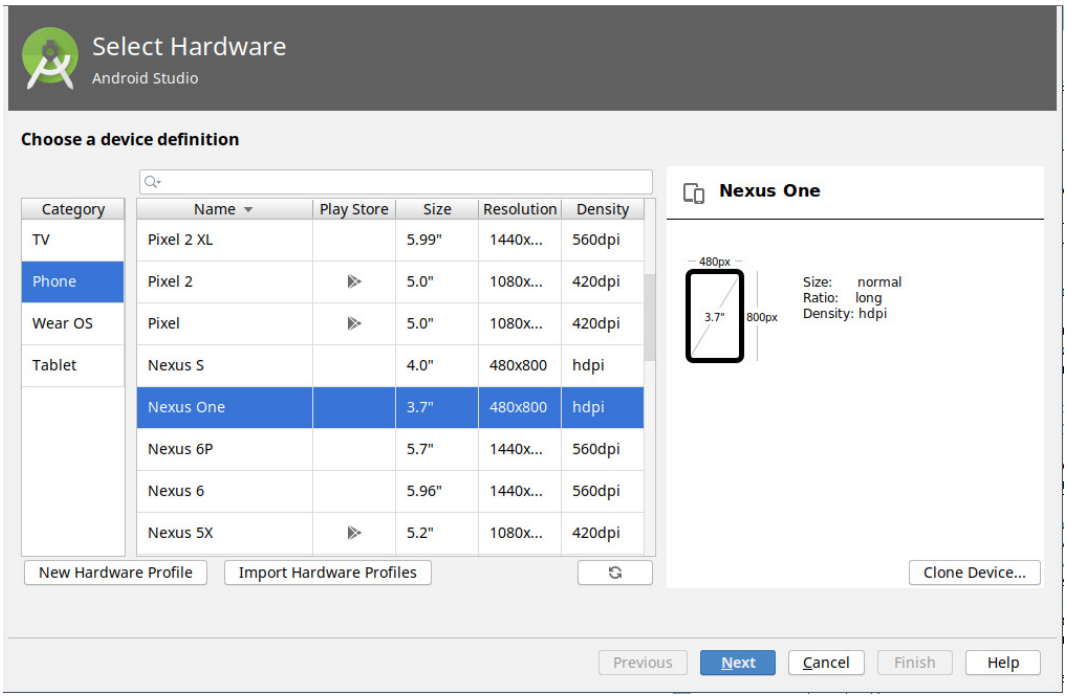

Рисунок 1.33 – Вибір пристрою

На наступному екрані вибираємо версію операційної системи Android. Якщо жодного образу ще не було встановлено, то необхідно спочатку завантажити відповідний образ системи, натиснувши кнопку «Download». При виборі образу, краще віддавати перевагу образам з архітектурою x86 та підтримкою сервісів Google Play:

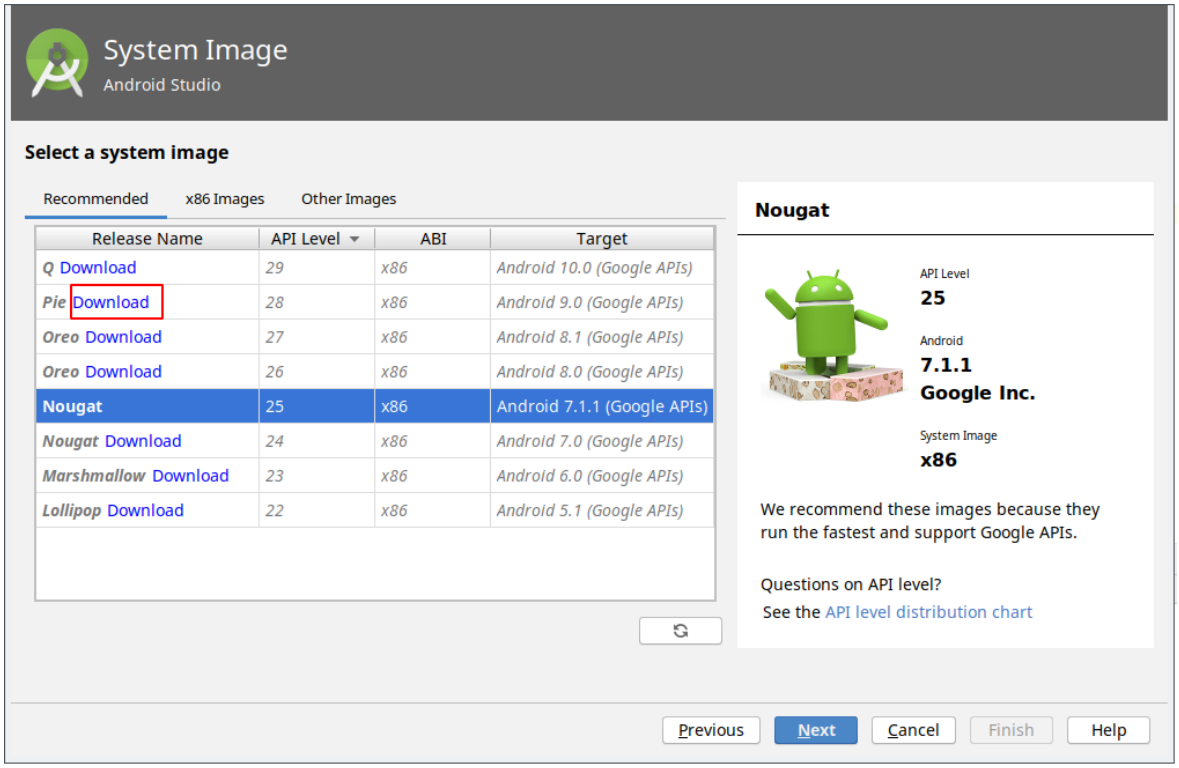

Рисунок 1.34 – Вибір образу ОС Android

На останньому екрані задається конфігурація віртуального пристрою:

1) AVD Name – назва пристрою
2) Startup Orientation – положення пристрою при старті
3) Camera – способи емуляції камери віртуального пристрою
4) Network – налаштування швидкості мережі. Даний пункт може бути корисними для тестування коректності роботи додатку в мережах зі слабким зв’язком.
5) Emulated Pefrormance – спосіб емуляції: апаратний або програмний, а також спосіб завантаження операційної системи. Cold Boot – завантаження з нуля. Quick Boot – завантаження з останнього знімку стану системи.
6) Memory And Storage – конфігурація об’єму памяті віртуального пристрою.
7) Device Frame – зовнішній вигляд пристрою
8) Keyboard – налаштування клавіатури. Якщо увімкнути пункт «Enable Keyboard Input», то роль програмної клавіатури Android буде виконувати реальна клавіатура вашого ПК.

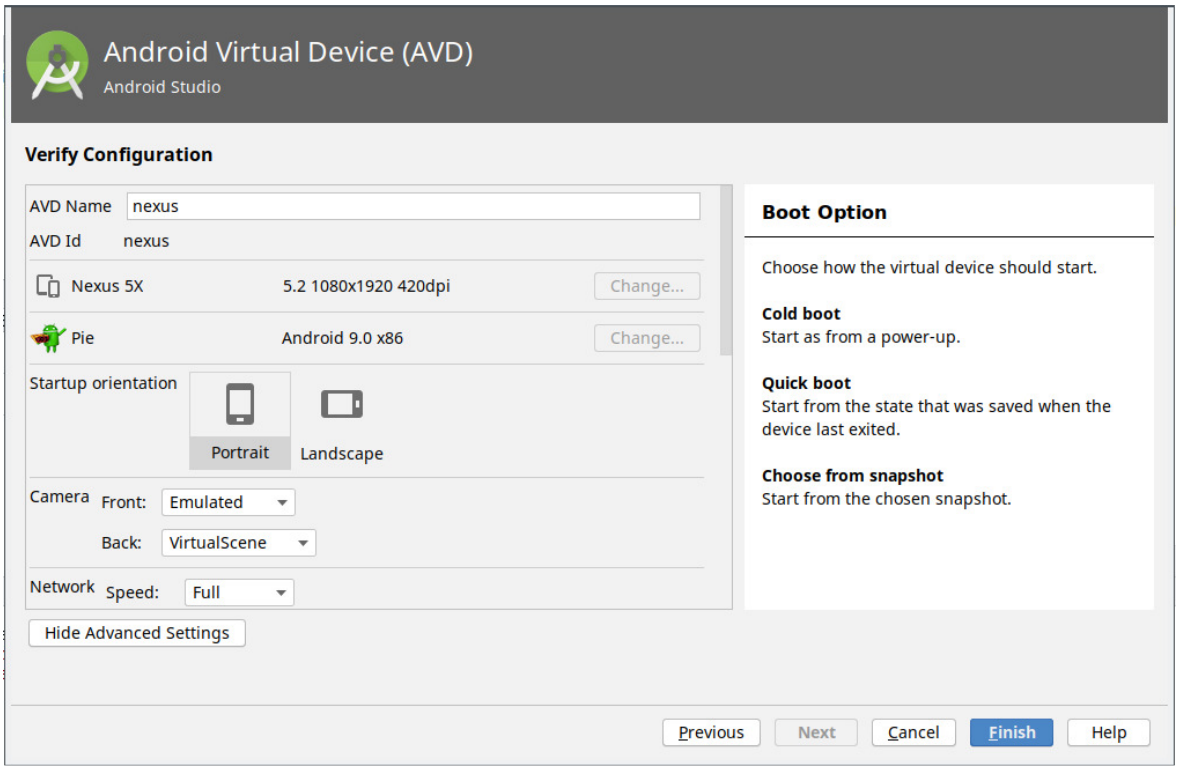

Рисунок 1.35 – Конфігурація віртуального пристрою

Якщо все виконано вірно, то віртуальний пристрій з'явиться в списку менеджера віртуальних пристроїв. Для запуску необхідно натиснути кнопку «Start»:

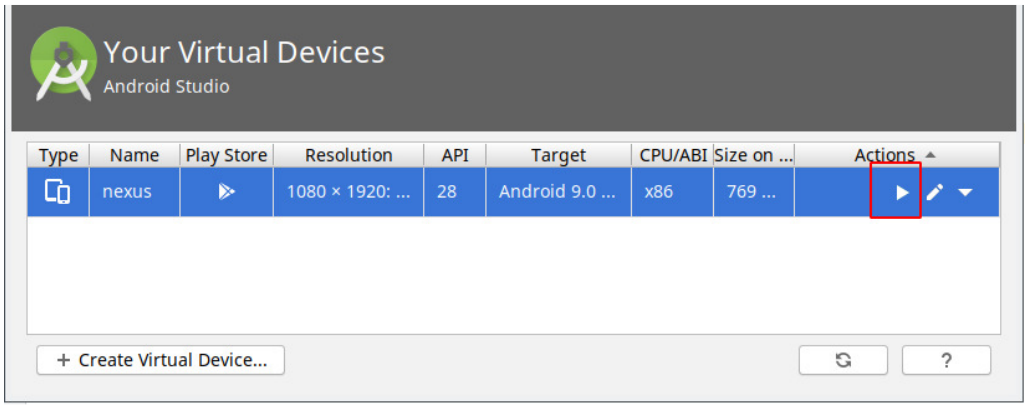

Рисунок 1.36 – Список доступних віртуальних пристроїв

Віртуальний пристрій не обов'язково запускати вручну. Якщо вибрати новостворений віртуальний пристрій в панелі інструментів Android Studio, то він буде автоматично запущений по команді Run -> Run App.

На рисунку нижче зображено вигляд віртуального пристрою з розробленим додатком для генерування номеру варіанту завдання:

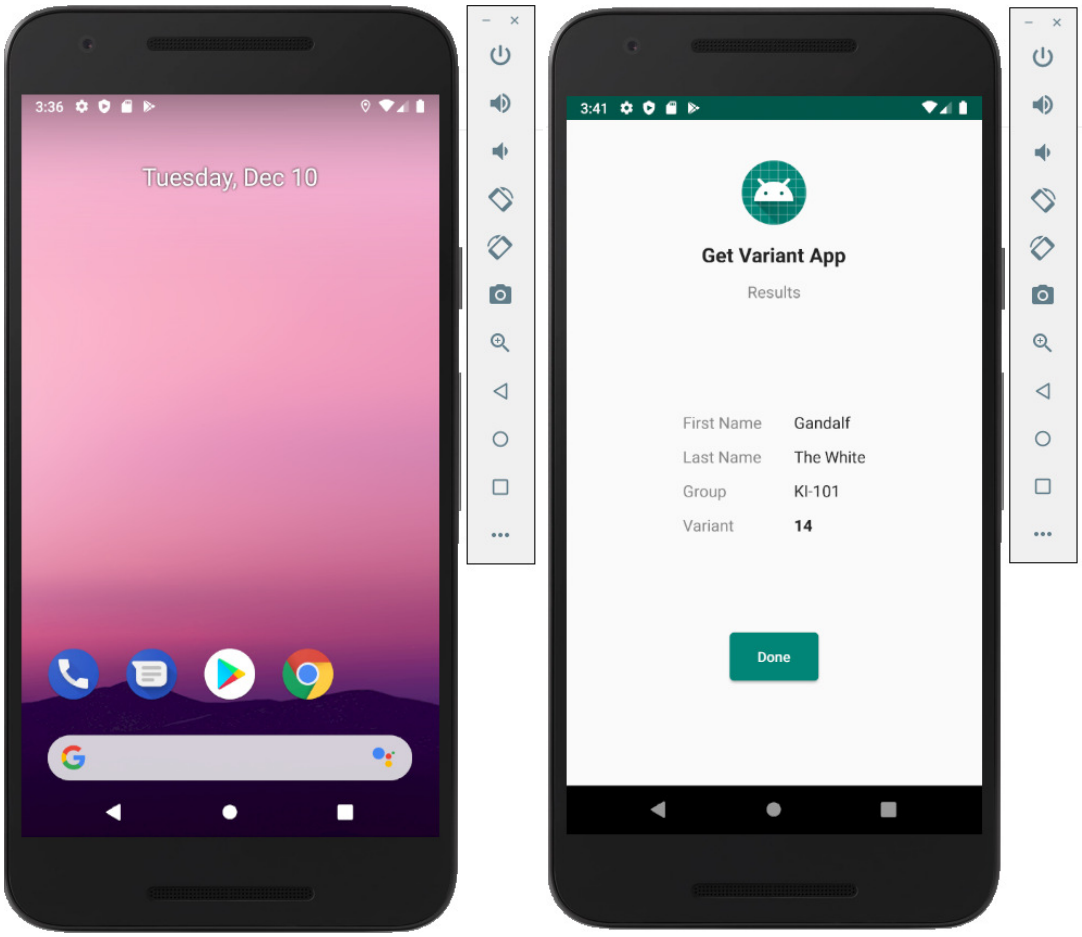

Рисунок 1.37 – Віртуальний пристрій

1.4 Завдання на самостійну роботу

На самостійну роботу пропонується створити простий Android-додаток з використанням базових компонентів інтерфейсу, який складається з декількох екранів. У кожному варіанті вказуються компонувальники, які необхідно обов'язково використати при проектуванні макетів інтерфейсу, а також тип навігації: на базі Activity, або на базі фрагментів.

1.4.1 Загальні вимоги

- Тексти інтерфейсу мають бути задані в ресурсах (strings.xml).
- Стилізація компонентів також задається в ресурсах (каталог drawable, файли dimens.xml, colors.xml, styles.xml).

- Мова програмування: Java або Kotlin (на вибір).
- Для передачі даних між екранами слід використовувати механізм Bundles (для Activity: setResult + onActivityResult, для фрагметів: newInstance + Callback Interface).
- Для тимчасового зберігання даних також слід використовувати механізм Bundles (для Activity: onSaveInstanceState + onCreate, для фрагментів: onSaveInstanceState + onViewCreated).
- Додаток має підтримувати альбомну орієнтацію екрану.
- Дані та стан інтерфейсу повинні зберігатись при повороті екрану.

1.4.2 Варіанти завдань

Номер варіанту завдання можна отримати, успішно зкомпілювавши та запустивши Android-додаток з п.п. 1.3.3.

Варіанти завдань на самостійну роботу:

№	Завдання
1	Створити Android-додаток для гри в хрестики-нолики, який складається з наступних екранів: - Splash-скрін з логотипом та назвою додатку (показується перші 2 секунди після запуску) - Екран авторизації (логін/пароль) - Екран з грою хрестики-нолики (поле 3x3), де можна зіграти проти штучного інтелекту. Екран складається з: імені гравця (логін), поля 3x3 та кнопок: «Вийти», «Почати заново». Якщо гра закінчується - необхідно вивести діалог з відповідним текстом та кнопками «Ок» (вихід з програми), «Заново» (розпочати нову гру). Компонування елементів інтерфейсу виконати на базі: LinearLayout, FrameLayout. Навігація має бути побудована на базі Activity.
2	Створити Android-додаток для гри «Відгадай число (Bulls & Cows)», який складається з наступних екранів: - Головне меню з кнопками: «Почати гру», «Про розробника», «Вийти» та випадним списком з вибором складності (легка, середня, важка) - Екран гри, на якому гравець має вгадати секретне N-значне число, згенероване програмою (легка складність N=4, середня N=5, складна N=6): - Число складається з цифр від 0 до 9; - Жодна цифра не може повторюватись; Алгоритм: - Гравець вводить число в текстове поле та натискає кнопку «Перевірити» - Програма перевіряє співпадіння цифр. Для кожної цифри: - якщо цифра існує та стоїть на правильному місці, нараховується 1 bull - якщо цифра існує, але стоїть не на тому місці, нараховується 1 cow - загальний результат відображається гравцеві. - гра закінчується тоді, коли гравець повністю вгадав число. Наприклад, програма загадала число 1643. Тоді: - якщо гравець вводить 1532, результат буде 1 bull 1 cow - якщо гравець вводить 2640, результат буде 2 bulls 0 cows

	- якщо гравець вводить 1532, результат буде 1 bull 1 cow -Додаток має на екрані гри відображати історію спроб в наступному форматі: «№ спроби - введене число - результат" Навігація має бути побудована на базі фрагментів (Fragments) Компонування інтерфейсу виконати на базі: ConstraintLayout
3	Створити Android-додаток «Оракул». Екрани: - Головний екран з кнопками: «Задати питання», «Налаштування», «Вихід» - Екран налаштувань, який дозволяє редагувати інформацію користувача: - ім'я (текстове поле) - прізвище (текстове поле) - дата народження (відображення текстом, зміна дати за допомогою DatePickerDialog) - стать (RadioButton) - Екран для запитань: - користувач вводить запитання та натискає кнопку «Запитати» - програма у відповідь пише один із наступних варіантів «Так», «Ні», «Можливо», «Цілком вірогідно», «Малоймовірно», «Не знаю». - відповідь генерується на базі наступних даних: введене запитання, інформація про користувача, поточна дата. - для одних і тих же вхідних даних відповідь має бути однакова. Навігація має бути побудована на базі Activity. Компонування інтерфейсу виконати на базі RelativeLayout.
4	Створити Android-додаток для розрахунку заощаджень в іноземній валюті. Додаток має містити в собі дані курсів євро та долару на початок року (C_{START}) та кінець року (C_{END}). Екрани: - Початковий екран з інформацією про додаток та кнопками «Почати розрахунок», «Вийти» - 1-й екран розрахунку: користувач вводить свій дохід за місяць у гривні (M) та відсоток доходу (p: $0 < p < 1$) який конвертується в іноземну валюту кожного місяця та натискає кнопку «Далі» - 2-й екран розрахунку: користувач вводить бажану валюту та натискає кнопку «Розрахувати» - 3-й екран показує результати розрахунку. Додаток розраховує: 1) Гіпотетичний повний річний дохід у гривні (S_Y) без обміну валюти: $S_Y = 12 \cdot M$, тут M – місячний дохід, грн 2) Кількість гривень (Sc), витрачених на обмін валюти: $S_C = p \cdot S_Y$ тут p – відсоток від місячного доходу ($0 < p < 1$) 3) Кількість валюти (W), придбаної за рік: $W = \sum_{i=1}^{12} \frac{p \cdot M}{C_i}$ тут Ci – інтерпольований курс за i-тий місяць У якості інтерпольованого курсу за i-тий місяць береться значення: $C_i = C_{START} + i \cdot \frac{C_{END} - C_{START}}{12}$ 4) Гривнева вартість придбаної валюти (S_H) на кінець року: $S_H = W \cdot C_{END}$ 5) Гривневий залишок (S_L): $S_L = S_Y - S_c$ 6) Сума гривневого залишку S_H та гривневої вартості валюти S_L на кінець року: $H = S_H + S_L$ 7) Сума заощадження (R): різниця між сумою H з п.п. 6 та гіпотетичним річним доходом (S_Y):

	$R = H - S_Y$ На 3-му екрані також має бути кнопка «Ок», яка повертає користувача на початковий екран. Навігація має бути побудована на базі Activity Компонування інтерфейсу виконати на базі ConstraintLayout
5	Створити Android-додаток для гри «Who Wants to Be a Millionaire». Додаток складається з екрану меню, екрану налаштувань та екрану гри. - Екран меню містить кнопки: «Налаштування», «Почати гру». - Екран налаштувань дозволяє задати кількість питань (від 5 до 10, за допомогою випадаючого списку) та можливість ввімкнути підказку 50x50 (checkbox). В межах однієї гри підказка може бути використана лише один раз. - Екран гри показує гравцеві випадкові питання з 4 варіантами відповіді, серед яких лише один варіант є правильним. Підказка 50x50 викреслює 2 неправильні відповіді. Кожне питання оцінюється в деяку суму (на розсуд студента), причому кожне наступне питання дорожче за попереднє. Гра завершується, якщо гравець дав неправильну відповідь або правильно відповів на всі запитання. На екрані гри також показується сума, яку на даний момент виграв гравець. Правильні відповіді на всі запитання дають в сумі 1 000 000$. На екрані присутня кнопку «Забрати суму», яка завершує гру. При завершенні гри додаток показує діалог з текстом «Вітаємо, ви виграли N$» (або «Ви програли» у разі неправильної відповіді) та повертає користувача на екран меню. Питання та відповіді мають перемішуватись у випадковому порядку. В межах однієї гри питання не можуть повторюватись. Навігація має бути побудована на базі Activity Компонування інтерфейсу виконати на базі RelativeLayout, LinearLayout.
6	Створити Android-додаток для перевірки реакції користувача. Додаток складається зі Splash-екрану, екрану налаштувань та екрану гри. - Splash-скрін показується першим, містить назву, логотип додатку та пропадає через 3 секунди після запуску або після натискання в будь-якому місці екрану. - Екран налаштувань, який показується після Splash-скріну, дозволяє налаштувати розмірність сітки (3x3, 4x4, 5x5), таймер (15 сек, 30 сек, 45 сек) та діапазон швидкості зміни кадрів (1-1.5 сек, 0.7-1сек, 0.5-0.7сек, 0.3-0.6сек). Всі ці параметри задаються за допомогою випадаючих списків. Екран містить кнопку «Розпочати», який запускає гру - Екран гри. Спочатку екран містить зворотний відлік таймеру та кнопку «Завершити». Таймер відраховує 3 секунди, після чого гра розпочинається: користувачеві показується поле з квадратів заданої розмірності, відлік часу та результат. Поле складається з сірих квадратів та одного кольорового квадрату. Поле змінюється випадковим чином (змінюється положення кольорового квадрату) через випадкові проміжки часу в межах, заданих на екрані налаштувань. Завдання користувача – натиснути на кольоровий квадрат. За кожне успішне попадання нараховується 2 бали. За промах знімається 1 бал. Після завершення таймеру користувачеві показується діалог з результатом, потім додаток переходить назад на екран налаштувань. Навігація має бути побудована на базі фрагментів (Fragments) Компонування інтерфейсу виконати на базі RelativeLayout, TableLayout.
7	Створити Android-додаток для перевірки словникового запасу іноземної мови. Додаток складається з екрану налаштувань, екрану перевірки з варіантами відповіді та екрану перевірки без варіантів відповіді.

	- Екран налаштувань. Показується відразу після запуску, дозволяє вибрати мову (англійська або німецька), кількість питань (від 5 до 10), значення таймеру для одного запитання (20 сек, 30 сек, 1 хв). Всі налаштування задаються за допомогою компонентів RadioButton. Екран містить також дві кнопки «Тест з варіантами», «Тест з введенням відповіді». - Екран тесту з варіантами. Користувачеві показується випадкове слово на заданій мові та 3-5 варіантів відповідей (лише одна відповідь є правильною). Час обмежений таймером, зворотній відлік показується на екрані. Після вибору варіанту, він відсвічується зеленим або червоним кольором в залежності від правильності вибору. Якщо вибрано неправильний варіант, то користувачеві підсвічується також і правильний варіант. Після відповіді таймер зупиняється та з'являється кнопка «Далі» для переходу до наступного питання або кнопка «Завершити», якщо питання було останнім. При натисканні на кнопку «Завершити» показується діалог з результатами: кількість правильних та неправильних відповідей, потім відбувається перехід на екран налаштувань. Якщо час було вичерпано до надання відповіді, то вважається, що користувач не вгадав слово. - Екран тесту без варіантів. Аналогічний попередньому екрану, але користувач має ввести переклад слова вручну в текстове поле. В межах тесту слова не мають повторюватись. Кожен тест складається з 10 випадкових слів. Слова повинні показуватись у випадковому порядку. Навігація має бути побудована на базі фрагментів (Fragments) Компонування інтерфейсу виконати на базі LinearLayout, FrameLayout.

8	Створити Android-додаток «Віднови слово». Додаток складається з екранів головного меню, налаштувань та екрану гри.
	- Головне меню містить інформацію про розробника, кнопки «Налаштування», «Розпочати гру».
	- Екран налаштувань дозволяє задати мінімальний (4,5,6) та максимальний (7,8,9) розмір слів у символах та час зворотного відліку (1 хв, 2 хв, 5 хв).
	- Екран гри показує час, який залишився до закінчення гри, кількість вгаданих слів. Гравцеві показується слово з перемішаними випадковим чином буквами. Гравець може натискати на букви, складаючи з них слово. Наприклад, програма показує букви «ідроа». Натискаючи послідовно на букви «р», «а», «д», «і», «о» гравець складає слово «радіо»:
	
	`і д р о а  і д – о а  і д – о –  і – – о –  – – – о –  – – – – –`
	`– – – – –  р – – – –  р а – – –  р а д – –  р а д і –  р а д і о`
	
	У випадку помилки, гравець може вернути букву назад вгору, натиснувши на неї. Слово заповнюється послідовно, зліва направо.
	Якщо гравець вгадав слово, програма показує Toast-повідомлення «Вгадано» та генерує наступне слово. Гра закінчується, коли час вичерпано. Гравцеві показується діалог з результатами: кількість вгаданих слів.
	
	Навігація має бути побудована на базі Activity.
	Компонування інтерфейсу виконати на базі ConstraintLayout, LinearLayout.

9	Створити Android-додаток «Конвертер». Додаток складається з екранів: Головне меню, конвертер довжин, конвертер ваги та конвертер температур. Головне меню містить інформацію про автора, а також кнопки, які ведуть до потрібного конвертера. Кожен екран конвертеру містить два випадних списки, які задають: з якої одиниці в яку має відбуватись конвертація, числове поле введення значення та

	поле відображення результату. Результат відображається відразу після введення будь-якого символу. Список одиниць: - довжина: сантиметр, метр, кілометр, дюйм, миля, ярд, фут - вага: грам, кілограм, тона, карат, фунт, пуд - температура: кельвін, цельсій, фаренгейт Навігація має бути побудована на базі фрагментів (Fragments). Компонування інтерфейсу виконати на базі RelativeLayout, FrameLayout.
10	Створити Android-додаток «Годинник». Додаток складається з екранів: Splash-скрін, екран годинника, екран налаштувань, екран таймера. - Splash-скрін показується при запуску протягом 2 секунд, потім стартує екран годинника. - Екран годинника показує поточні рік, дату, час та часовий пояс. Також на екрані годинника розміщується кнопки для переходу на екран таймеру та на екран налаштувань. - Екран налаштувань дозволяє змінити часовий пояс за допомогою випадного списку. - Екран таймера дозволяє вимірювати проміжки часу, містить кнопки «Старт», «Стоп», «Коло», «Пауза», відлік часу, список виміряних кіл. Кнопка «Старт» запускає таймер. Кнопка «Стоп» зупиняє таймер, при наступному запуску таймера, всі попередні результати видаляються. Кнопка «Пауза» лише призупиняє таймер, причому у випадку зупинки вона змінюється на кнопку «Продовжити», яка відповідно продовжує роботу таймеру. Кнопка «Коло» не зупиняє таймер, але зберігає його поточне значення, додаючи до списку кіл внизу екрану. Відлік кожного наступного кола починається від попереднього: тобто, якщо наприклад кнопка «Коло» була натиснута на 10-й та 30-й секунді роботи таймера, то в списку будуть значення 10 сек. та 20 сек. Навігація має бути побудована на базі Activity. Компонування інтерфейсу виконати на базі FrameLayout, LinearLayout.
11	Створити Android-додаток для гри в хрестики-нолики, який складається з наступних екранів: - Splash-скрін з логотипом та назвою додатку (показується перші 2 секунди після запуску) - Екран авторизації (логін/пароль) - Екран з грою крестики-нолики (поле 3x3), де можна зіграти проти штучного інтелекту. Екран складається з: імені гравця (логін), поля 3x3 та кнопок: «Вийти», «Почати заново». Якщо гра закінчується - необхідно вивести діалог з відповідним текстом та кнопками «Ок» (вихід з програми), «Заново» (розпочати нову гру). Компонування елементів інтерфейсу виконати на базі: ConstraintLayout. Навігація має бути побудована на базі фрагментів (Fragments).
12	Створити Android-додаток для гри «Відгадай число (Bulls & Cows)», який складається з наступних екранів: - Головне меню з кнопками: «Почати гру», «Про розробника», «Вийти» та випадним списком з вибором складності (легка, середня, важка) - Екран гри, на якому гравець має вгадати секретне N-значне число, згенероване програмою (легка складність N=4, середня N=5, складна N=6): - Число складається з цифр від 0 до 9; - Жодна цифра не може повторюватись; Алгоритм:

	- Гравець вводить число в текстове поле та натискає кнопку «Перевірити» - Програма перевіряє співпадіння цифр. Для кожної цифри: - якщо цифра існує та стоїть на правильному місці, нараховується 1 bull - якщо цифра існує, але стоїть не на тому місці, нараховується 1 cow - загальний результат відображається гравцеві. - гра закінчується тоді, коли гравець повністю вгадав число. Наприклад, програма загадала число 1643. Тоді: - якщо гравець вводить 1532, результат буде 1 bull 1 cow - якщо гравець вводить 2640, результат буде 2 bulls 0 cows - якщо гравець вводить 1532, результат буде 1 bull 1 cow -Додаток має на екрані гри відображати історію спроб в наступному форматі: «№ спроби - введене число - результат" Навігація має бути побудована на базі Activity Компонування інтерфейсу виконати на базі: LinearLayout, RelativeLayout
13	Створити Android-додаток «Оракул». Екрани: - Головний екран з кнопками: «Задати питання», «Налаштування», «Вихід» - Екран налаштувань, який дозволяє редагувати інформацію користувача: - ім'я (текстове поле) - прізвище (текстове поле) - дата народження (відображення текстом, зміна дати за допомогою DatePickerDialog) - стать (RadioButton) - Екран для запитань: - користувач вводить запитання та натискає кнопку «Запитати» - програма у відповідь пише один із наступних варіантів «Так», «Ні», «Можливо», «Цілком вірогідно», «Малоймовірно», «Не знаю». - відповідь генерується на базі наступних даних: введене запитання, інформація про користувача, поточна дата. - для одних і тих же вхідних даних відповідь має бути однакова. Навігація має бути побудована на базі фрагментів (Fragments). Компонування інтерфейсу виконати на базі ConstraintLayout.
14	Створити Android-додаток для розрахунку заощаджень в іноземній валюті. Додаток має містити в собі дані курсів євро та долару на початок року (C_{START}) та кінець року (C_{END}). Екрани: - Початковий екран з інформацією про додаток та кнопками «Почати розрахунок», «Вийти» - 1-й екран розрахунку: користувач вводить свій дохід за місяць у гривні (M) та відсоток доходу (p: $0 < p < 1$) який конвертується в іноземну валюту кожного місяця та натискає кнопку «Далі» - 2-й екран розрахунку: користувач вводить бажану валюту та натискає кнопку «Розрахувати» - 3-й екран показує результати розрахунку. Додаток розраховує: 1) Гіпотетичний повний річний дохід у гривні (S_Y) без обміну валюти: $S_Y = 12 \cdot M$, тут M – місячний дохід, грн 2) Кількість гривень (Sc), витрачених на обмін валюти: $S_C = p \cdot S_Y$ тут p – відсоток від місячного доходу ($0 < p < 1$) 3) Кількість валюти (W), придбаної за рік: $W = \sum_{i=1}^{12} \frac{p \cdot M}{C_i}$ тут Ci – інтерпольований курс за i-тий місяць У якості інтерпольованого курсу за i-тий місяць береться значення:

	$C_i = C_{START} + i \cdot \frac{C_{END} - C_{START}}{12}$ 4) Гривнева вартість придбаної валюти (S_H) на кінець року: $S_H = W \cdot C_{END}$ 5) Гривневий залишок (S_L): $S_L = S_Y - S_c$ 6) Сума гривневого залишку S_H та гривневої вартості валюти S_L на кінець року: $H = S_H + S_L$ 7) Сума заощадження (R): різниця між сумою H з п.п. 6 та гіпотетичним річним доходом (S_Y): $R = H - S_Y$ На 3-му екрані також має бути кнопка «Ок», яка повертає користувача на початковий екран. Навігація має бути побудована на базі фрагментів (Fragments) Компонування інтерфейсу виконати на базі LinearLayout, TableLayout
15	Створити Android-додаток для гри «Who Wants to Be a Millionaire». Додаток складається з екрану меню, екрану налаштувань та екрану гри. - Екран меню містить кнопки: «Налаштування», «Почати гру». - Екран налаштувань дозволяє задати кількість питань (від 5 до 10, за допомогою випадаючого списку) та можливість ввімкнути підказку 50x50 (checkbox). В межах однієї гри підказка може бути використана лише один раз. - Екран гри показує гравцеві випадкові питання з 4 варіантами відповіді, серед яких лише один варіант є правильним. Підказка 50x50 викреслює 2 неправильні відповіді. Кожне питання оцінюється в деяку суму (на розсуд студента), причому кожне наступне питання дорожче за попереднє. Гра завершується, якщо гравець дав неправильну відповідь або правильно відповів на всі запитання. На екрані гри також показується сума, яку на даний момент виграв гравець. Правильні відповіді на всі запитання дають в сумі 1 000 000$. На екрані присутня кнопку «Забрати суму», яка завершує гру. При завершенні гри додаток показує діалог з текстом «Вітаємо, ви виграли N$» (або «Ви програли» у разі неправильної відповіді) та повертає користувача на екран меню. Питання та відповіді мають перемішуватись у випадковому порядку. В межах однієї гри питання не можуть повторюватись. Навігація має бути побудована на базі фрагментів (Fragments) Компонування інтерфейсу виконати на базі ConstraintLayout.
16	Створити Android-додаток для перевірки реакції користувача. Додаток складається зі Splash-екрану, екрану налаштувань та екрану гри. - Splash-скрін показується першим, містить назву, логотип додатку та пропадає через 3 секунди після запуску або після натискання в будь-якому місці екрану. - Екран налаштувань, який показується після Splash-скріну, дозволяє налаштувати розмірність сітки (3x3, 4x4, 5x5), таймер (15 сек, 30 сек, 45 сек) та діапазон швидкості зміни кадрів (1-1.5 сек, 0.7-1сек, 0.5-0.7сек, 0.3-0.6сек). Всі ці параметри задаються за допомогою випадаючих списків. Екран містить кнопку «Розпочати», який запускає гру - Екран гри. Спочатку екран містить зворотний відлік таймеру та кнопку «Завершити». Таймер відраховує 3 секунди, після чого гра розпочинається: користувачеві показується поле з квадратів заданої розмірності, відлік часу та результат. Поле складається з сірих квадратів та одного кольорового квадрату. Поле змінюється випадковим чином (змінюється положення кольорового квадрату) через випадкові проміжки часу в межах, заданих на екрані налаштувань. Завдання користувача – натиснути на кольоровий квадрат. За кожне успішне попадання

	нараховується 2 бали. За промах знімається 1 бал. Після завершення таймеру користувачеві показується діалог з результатом, потім додаток переходить назад на екран налаштувань. Навігація має бути побудована на базі Activity Компонування інтерфейсу виконати на базі FrameLayout, TableLayout.
17	Створити Android-додаток для перевірки словникового запасу іноземної мови. Додаток складається з екрану налаштувань, екрану перевірки з варіантами відповіді та екрану перевірки без варіантів відповіді. - Екран налаштувань. Показується відразу після запуску, дозволяє вибрати мову (англійська або німецька), кількість питань (від 5 до 10), значення таймеру для одного запитання (20 сек, 30 сек, 1 хв). Всі налаштування задаються за допомогою компонентів RadioButton. Екран містить також дві кнопки «Тест з варіантами», «Тест з введенням відповіді». - Екран тесту з варіантами. Користувачеві показується випадкове слово на заданій мові та 3-5 варіантів відповідей (лише одна відповідь є правильною). Час обмежений таймером, зворотній відлік показується на екрані. Після вибору варіанту, він відсвічується зеленим або червоним кольором в залежності від правильності вибору. Якщо вибрано неправильний варіант, то користувачеві підсвічується також і правильний варіант. Після відповіді таймер зупиняється та з'являється кнопка «Далі» для переходу до наступного питання або кнопка «Завершити», якщо питання було останнім. При натисканні на кнопку «Завершити» показується діалог з результатами: кількість правильних та неправильних відповідей, потім відбувається перехід на екран налаштувань. Якщо час було вичерпано до надання відповіді, то вважається, що користувач не вгадав слово. - Екран тесту без варіантів. Аналогічний попередньому екрану, але користувач має ввести переклад слова вручну в текстове поле. В межах тесту слова не мають повторюватись. Кожен тест складається з 10 випадкових слів. Слова повинні показуватись у випадковому порядку. Навігація має бути побудована на базі Activity Компонування інтерфейсу виконати на базі ConstraintLayout.

18	Створити Android-додаток «Віднови слово». Додаток складається з екранів головного меню, налаштувань та екрану гри.
	- Головне меню містить інформацію про розробника, кнопки «Налаштування», «Розпочати гру».
	- Екран налаштувань дозволяє задати мінімальний (4,5,6) та максимальний (7,8,9) розмір слів у символах та час зворотного відліку (1 хв, 2 хв, 5 хв).
	- Екран гри показує час, який залишився до закінчення гри, кількість вгаданих слів. Гравцеві показується слово з перемішаними випадковим чином буквами. Гравець може натискати на букви, складаючи з них слово. Наприклад, програма показує букви «ідроа». Натискаючи послідовно на букви «р», «а», «д», «і», «о» гравець складає слово «радіо»:
	
	`і д р о а  і д - о а  і д - о -  і - - о -  - - - о -  - - - - -`
	`- - - - -  р - - - -  р а - - -  р а д - -  р а д і -  р а д і о`
	
	У випадку помилки, гравець може вернути букву назад вгору, натиснувши на неї. Слово заповнюється послідовно, зліва направо.
	Якщо гравець вгадав слово, програма показує Toast-повідомлення «Вгадано» та генерує наступне слово. Гра закінчується, коли час вичерпано. Гравцеві

	показується діалог з результатами: кількість вгаданих слів. Навігація має бути побудована на базі фрагментів (Fragments). Компонування інтерфейсу виконати на базі RelativeLayout, FrameLayout.
19	Створити Android-додаток «Конвертер». Додаток складається з екранів: Головне меню, конвертер довжин, конвертер ваги та конвертер температур. Головне меню містить інформацію про автора, а також кнопки, які ведуть до потрібного конвертера. Кожен екран конвертеру містить два випадних списки, які задають: з якої одиниці в яку має відбуватись конвертація, числове поле введення значення та поле відображення результату. Результат відображається відразу після введення будь-якого символу. Список одиниць: - довжина: сантиметр, метр, кілометр, дюйм, миля, ярд, фут - вага: грам, кілограм, тона, карат, фунт, пуд - температура: кельвін, цельсій, фаренгейт Навігація має бути побудована на базі Activity. Компонування інтерфейсу виконати на базі ConstraintLayout.
20	Створити Android-додаток «Годинник». Додаток складається з екранів: Splash-скрін, екран годинника, екран налаштувань, екран таймера. - Splash-скрін показується при запуску протягом 2 секунд, потім стартує екран годинника. - Екран годинника показує поточні рік, дату, час та часовий пояс. Також на екрані годинника розміщується кнопки для переходу на екран таймеру та на екран налаштувань. - Екран налаштувань дозволяє змінити часовий пояс за допомогою випадного списку. - Екран таймера дозволяє вимірювати проміжки часу, містить кнопки «Старт», «Стоп», «Коло», «Пауза», відлік часу, список виміряних кіл. Кнопка «Старт» запускає таймер. Кнопка «Стоп» зупиняє таймер, при наступному запуску таймера, всі попередні результати видаляються. Кнопка «Пауза» лише призупиняє таймер, причому у випадку зупинки вона змінюється на кнопку «Продовжити», яка відповідно продовжує роботу таймеру. Кнопка «Коло» не зупиняє таймер, але зберігає його поточне значення, додаючи до списку кіл внизу екрану. Відлік кожного наступного кола починається від попереднього: тобто, якщо наприклад кнопка «Коло» була натиснута на 10-й та 30-й секунді роботи таймера, то в списку будуть значення 10 сек. та 20 сек. Навігація має бути побудована на базі фрагментів (Fragments). Компонування інтерфейсу виконати на базі RelativeLayout.

1.5 Звітність

Звіт з лабораторної роботи має містити:

1) Номер та назву роботи;
2) Мету роботи;
3) Хід виконання роботи:
   a. Результати виконання п.п. 1.3.3 зі скріншотом екрану результатів розрахунку варіанту завдання;

b. Тексти класів та макетів інтерфейсу додатку згідно варіанту завдання;
c. Скріншоти додатку;
d. Опис виявлених проблем при розробці додатку та шляхи їх вирішення;

4) Висновки

1.6 Контрольні запитання

1) Структура Android-проекту
2) Структура та призначення файлу AndroidManifest.xml
3) Система збирання проектів Gradle: додавання залежностей до проекту.
4) Конфігурація Android-додатку. Встановлення та значення minSdkVersion, targetSdkVersion, compileSdkVersion, app package.
5) Типи модулів проекту: Android Application, Android Library, Java Library.
6) Програмний компонент Activity. Призначення програмного компоненту Activity.
7) Додавання нової Activity до проекту.
8) Життєвий цикл Activity.
9) Обробка зміни конфігурації при повороті екрану.
10) Об’єкт Intent. Поняття Intent Filter.
11) Явний та неявний виклик Activity.
12) Передача даних між різними Activity.
13) Поняття фрагментів. Призначення та можливості фрагментів. Переваги використання фрагментів.
14) Життєвий цикл фрагменту.
15) Методи додавання фрагментів до Activity.
16) Видалення фрагментів з Activity. Поняття транзакції фрагментів.
17) Використання зворотного стеку (Back Stack) фрагментів. Навігація на базі фрагментів.
18) Основні компонувальники інтерфейсу: LinearLayout, FrameLayout, TableLayout, RelativeLayout, ConstraintLayout.
19) Компоненти інтерфейсу Android: Button, TextView, EditText, Switch, CheckBox, RadioButton, RadioGroup, ImageView.
20) Спискові компоненти інтерфейсу: ListView, RecyclerView, Spinner, AutoCompleteTextView.
21) Принципи реалізації списків. Поняття адаптеру. Реалізація адаптерів на базі ArrayAdapter та BaseAdapter.
22) Перевірка роботи додатку та відлагодження на реальних та віртуальних пристроях

1.7 Довідкова література

1) https://kotlinlang.org/docs/reference
2) https://developer.android.com/guide/app-bundle
3) https://docs.gradle.org/current/userguide/gradle_wrapper.html
4) https://search.maven.org
5) https://developer.android.com/studio/publish/app-signing
6) https://developer.android.com/training/multiscreen/screendensities