Факультет комп’ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІД МОБІЛЬНІ ПЛАТФОРМИ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання програмного проєкту

Київ 2026

УДК 004.4:004.6

Рецензенти:

Панченко Т.В., канд. ф.-м. н., доцент, завідувач кафедри теорії та технології програмування Київського національного університету імені Тараса Шевченка

Шкільняк О.С., канд. ф.-м. н., доцент, доцент кафедри інтелектуальних програмних систем Київського національного університету імені Тараса Шевченка

Рекомендовано до друку вченою радою факультету комп'ютерних наук та кібернетики (протокол №12 від 24 березня 2026 року)

Ухвалено науково-методичною комісією факультету комп'ютерних наук та кібернетики (протокол №9 від 16 березня 2026 року)

к.т.н., доцент Ткаченко Олексій Миколайович

асистент Галавай Олександр Миколайович

Ткаченко О.М., Галавай О.М. Розробка програмного забезпечення під мобільні платформи: Методичні вказівки до виконання програмного проєкту (електронне видання). – К.: КНУ імені Тараса Шевченка, 2026. – 78 с.

Викладено матеріали для вивчення технології розробки для ОС Android. Охоплено тематику реалізації в мобільному застосунку основних UI-елементів для введення і виведення даних, роботи з файлами і БД, побудови графіків і діаграм, імплементацію геопросторового модуля. У демонстраційних прикладах використано мову програмування Java. Виконання практичної частини курсу передбачає виконання наскрізного семестрового проєкту зі створення мобільного застосунку

Для для студентів 4 курсу ОПП "Інформатика" спеціальності F3 "Комп'ютерні науки" факультету комп'ютерних наук та кібернетики, а також для тих, хто цікавиться програмуванням під ОС Android.

Зміст

Вступ 4
1. Початок роботи, перша проста програма 5
2. Введення і виведення даних 15
3. Робота з файлами 22
4. Взаємодія з БД 33
5. Побудова графіків і діаграм 40
6. Визначення геопросторових координат 46
7. Геопросторові об’єкти на мапі 52
8. Геокодування, побудова маршруту 61
9. Поняття кросплатформної розробки 65
Практичні завдання 66
Список використаних джерел 70
Додатки 71
Додаток 1. Файл MainActivity.kt з кодом першої програми 71
Додаток 2. Файл activity_main.xml для програми розв'язування квадратного рівняння 72
Додаток 3. Файл MainActivity.java з кодом програми для побудови маршрутів 75

Вступ

У глобальному звіті Wearesocial [1] зазначено, що на початку 2025 р. кількість інтернет-користувачів у світі досягла 5,56 млрд., що є двома третинами населення планети. При цьому мобільний трафік становить майже 57%, зберігаючи тенденцію зростання як обсягу переданих даних, так і частки. Мобільні пристрої на Android становлять 73,5% від усіх активних. Це обумовлює зростаючий попит на розробників програмних рішень для мобільних платформ, насамперед, зокрема, Android.

На сьогодні розробники для Android можуть використовувати різний інструментарій. Найпопулярнішим середовищем розробки є Android Studio [2], а мовами програмування – Kotlin, Java, C, Dart, C#, JavaScript/TypeScript. У кросплатформній розробці найчастіше використовують фреймворки Flutter [3] та React Native [4].

Java [5] – універсальна мова високого рівня, яка історично є однією з перших мов розробників для Android, левова частка застосунків для Android створена саме на Java. Мова Kotlin [6] – це мова високого рівня зі статичною типізацією, веде свою історію з 2011 р. з компанії JetBrains. Програми на Kotlin компілюються в байт-код, який працює на JVM (Java Virtual Machine). З 2019 року Google рекомендує Kotlin розробникам застосунків для ОС Android. З документацію для розробників на Kotlin і Java для Android можна ознайомитися в [7]. У демонстраційних прикладах основного матеріалу цих вказівок використано мову Java.

Ці методичні вказівки призначенні для допомоги студентам при виконанні практичної частини завдань курсу з розробки ПЗ для мобільних платформ, а також для тих, хто цікавиться мобільною розрбкою.

1. Початок роботи, перша проста програма

Мета: навчитися створювати прості мобільні застосунки в середовищі Android Studio.

Теоретичні відомості

Android Studio – офіційне інтегроване середовище розробки (IDE) для створення мобільних застосунків для платформи Android. Середовище розроблене компанією Google на базі IDE IntelliJ IDEA.

Основні можливості та складові Android Studio:

- Редактор коду з підтримкою мов Kotlin, Java та C++, у тому числі можливістю автоматичного доповнення і рефакторингу, а також інтелектуальних функцій;
- Візуальний редактор макетів для швидкого проєктуваня UI методом перетягування та коду XML з можливістю попереднього перегляду;
- Емулятор (віртуальний пристрій), який дозволяє тестувати код для смартфонів, планшетів, а також Android TV та гаджетів на Wear OS;
- Gradle – система збірки проєкту, у тому числі з варіантами збірок;
- Інструменти оптимізації ресурсів (пам'яті, CPU, мережі);
- Gemini – вбудований ШІ-помічник в сучасних версіях IDE, який допомагає правити і генерувати код.

Приклад

Завантажте [2] і встановіть Android Studio на ваш комп'ютер (рис. 1).

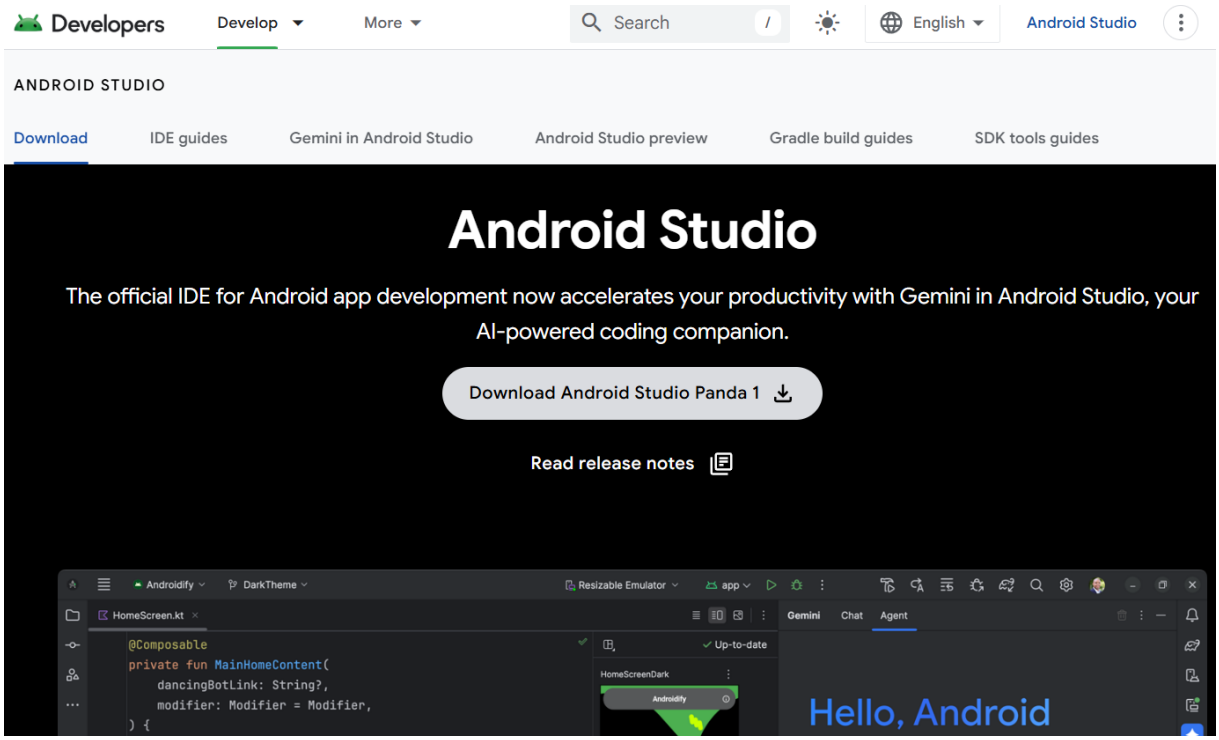


Рисунок 1 – Сторінка завантаження Android Studio

Запістіть IDE, у вас відкриється можливість створити чи відкрити проєкт, налаштувати IDE, переглянути і довстановити плагіни, ознайомитися з довідкою і новинами (рис. 2).

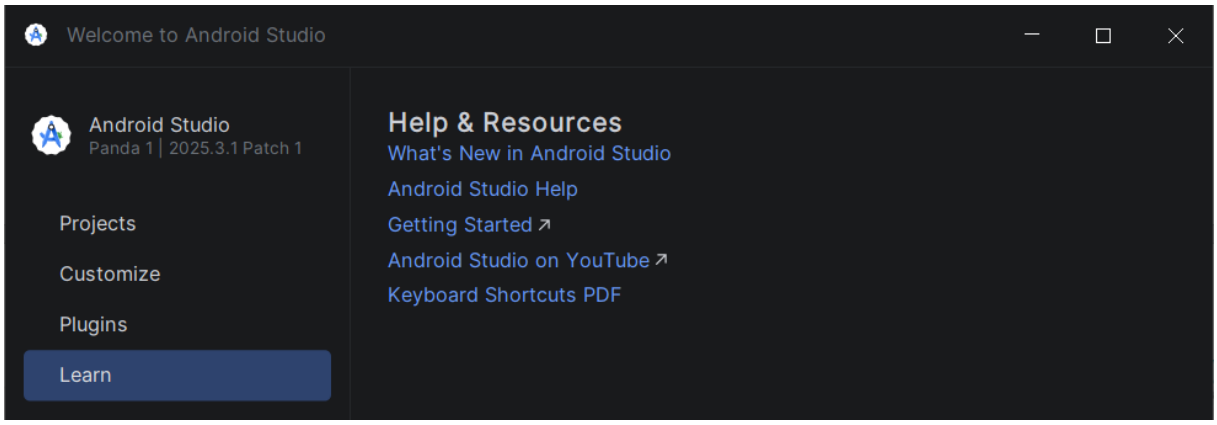


Рисунок 2 – Стартове вікно Android Studio

Для прикладу змінимо тему IDE та редактора на світлу (рис. 3):

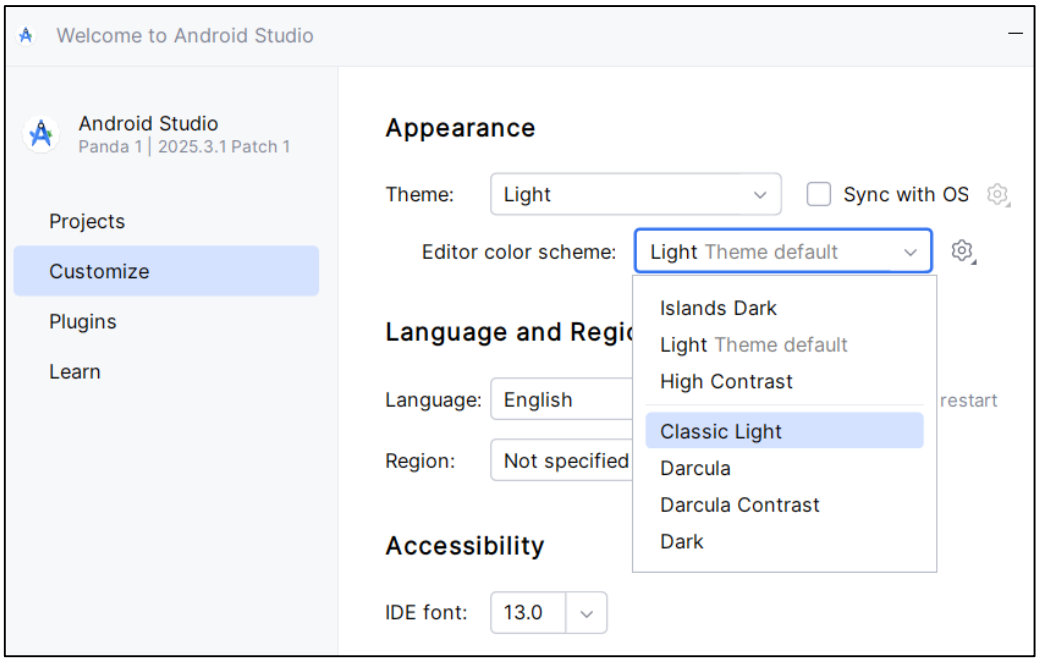


Рисунок 3 – Зміна налаштувань редактора IDE (колір)

Вибір нового проєкту як **Empty Activity** (рис. 4) орієнтований на розробку проєкта на мові Kotlin (рис. 5).

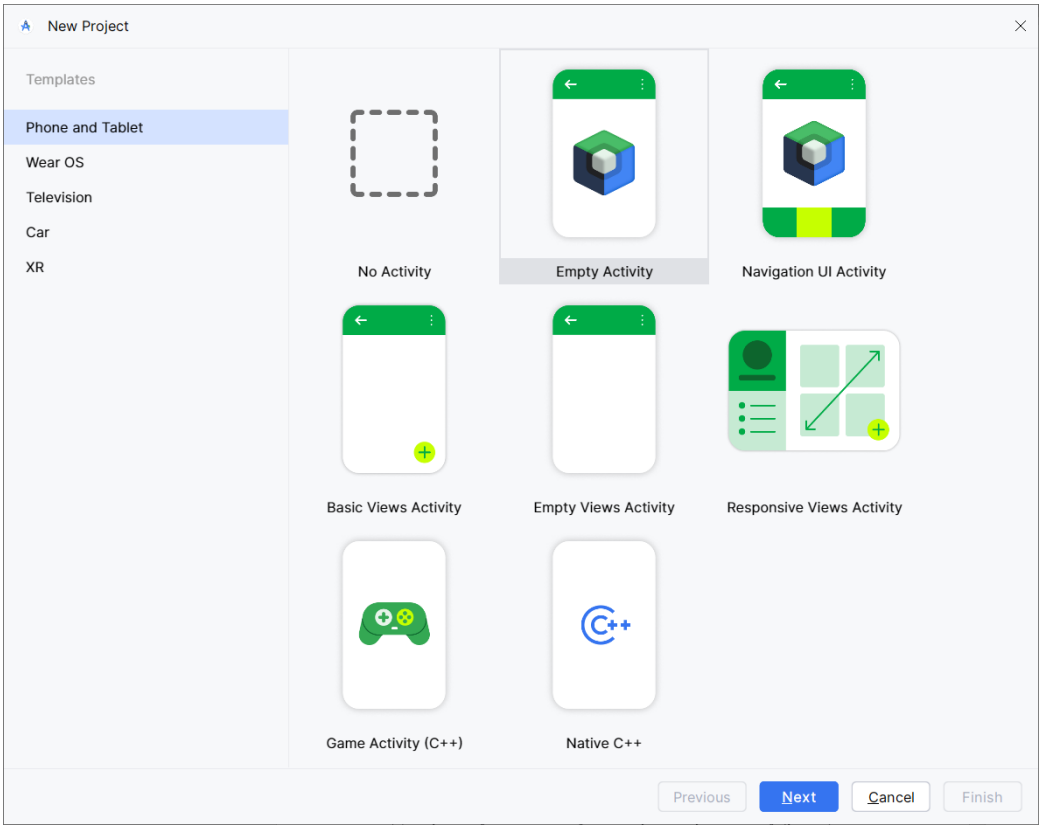


Рисунок 4 – Створення нового мобільного проєкту

Вказуємо ім’я, шлях до розміщення проєкту, версію SDK (рис. 5).

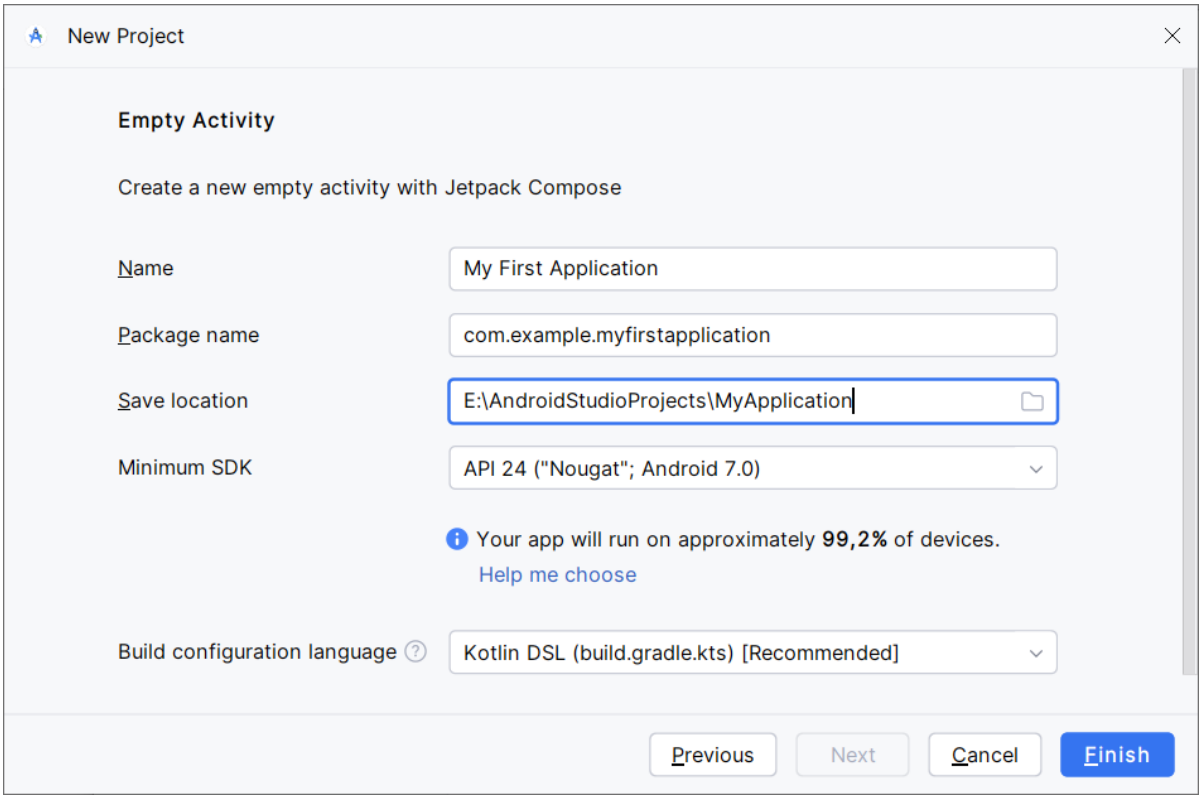


Рисунок 5 – Параметри нового проєкту

Відкривається IDE з редактором коду, навігатором проєкту, вікном емулятора пристрою (тут – Medium Phone API 36.1) та ін. (рис. 6.)

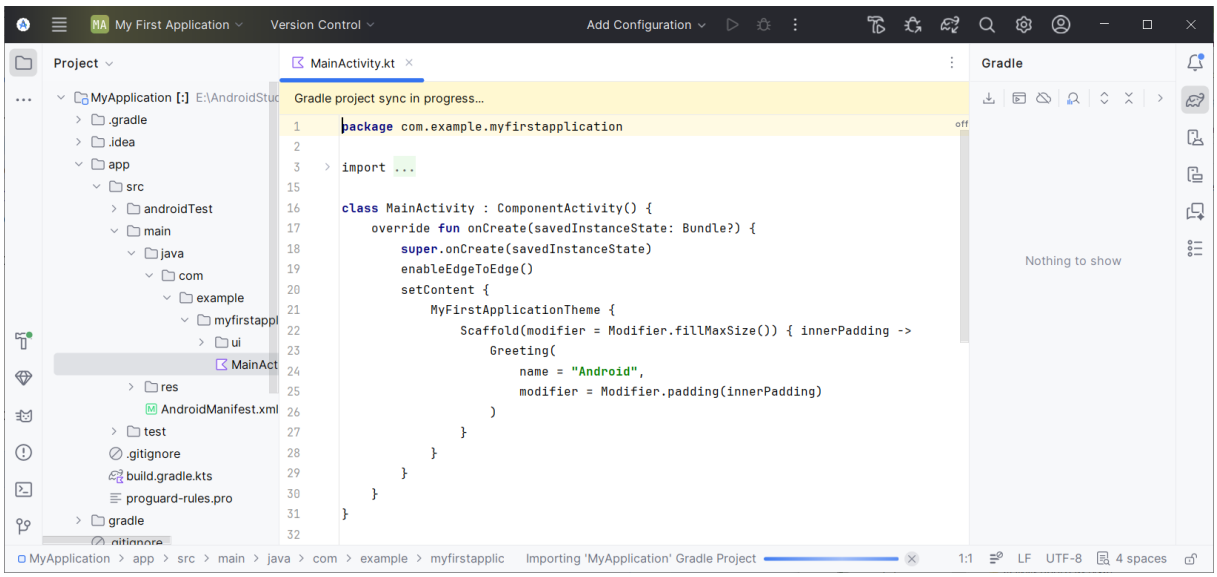
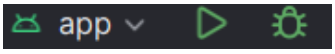


Рисунок 6 – Заготовка нового застосунку

Для запуску збірки і виконання цього стартового демопроєкту натисніть кнопку **Run** (зелений трикутник)  або або **Run → Run 'app'**. Через деякий час (кілька секунд або хвилин, залежить від швидкості комп'ютера) буде зібрано проєкт та відображено у вікні емулятора. У нашому випадку це буде зображення світлого екрану смартфона з системним часом і написом "Hello Android", який заданий у файлі **MainActivity.kt** (рис. 7).

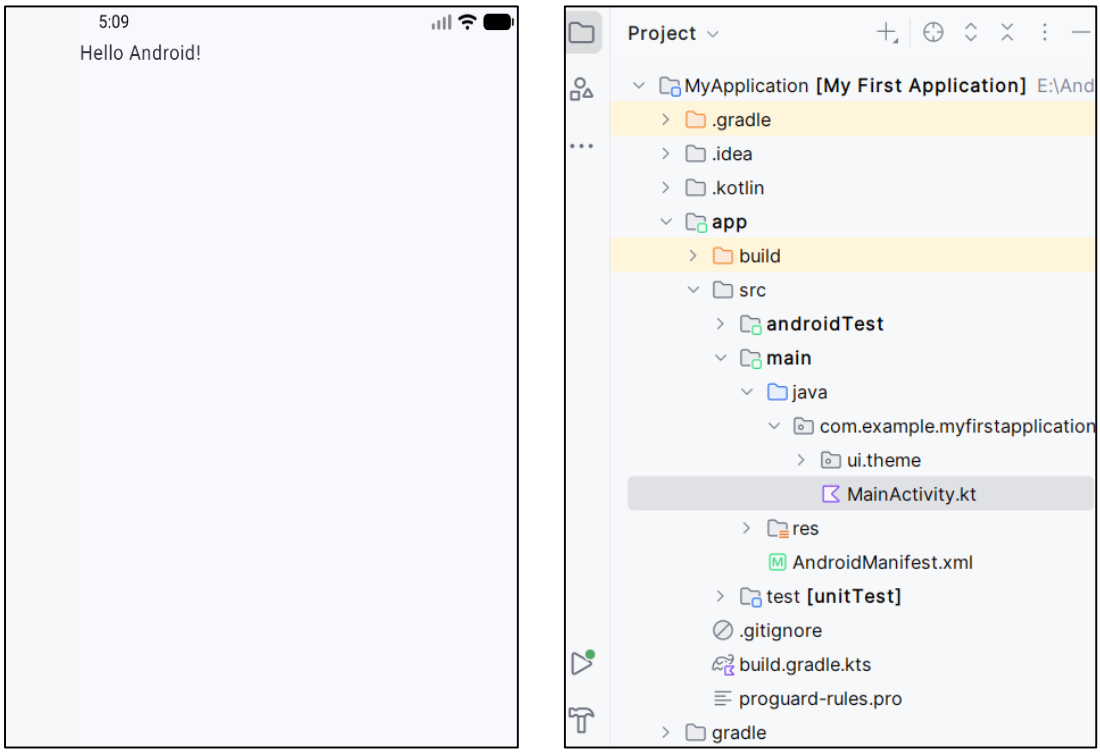


Рисунок 7 – Збірка застосунку за замовченням

Ви можете додати нові віртуальні пристрої для емуляції тестування ваших програм на них: **Tools → Device Manager →Add a new device**.

Файл **MainActivity.kt** з кодом цієї програми подано в Додатку 1.

Візуальна частина UI розміщена у файлі **activity.main.xml**. У файлі **MyApplication/app/src/main/AndroidManifest.xml** міститься опис загальних параметрів проєкту.

Для перемикання між режимами (код, візуальний редактор) скористайтесь кнопками  (рис. 8).

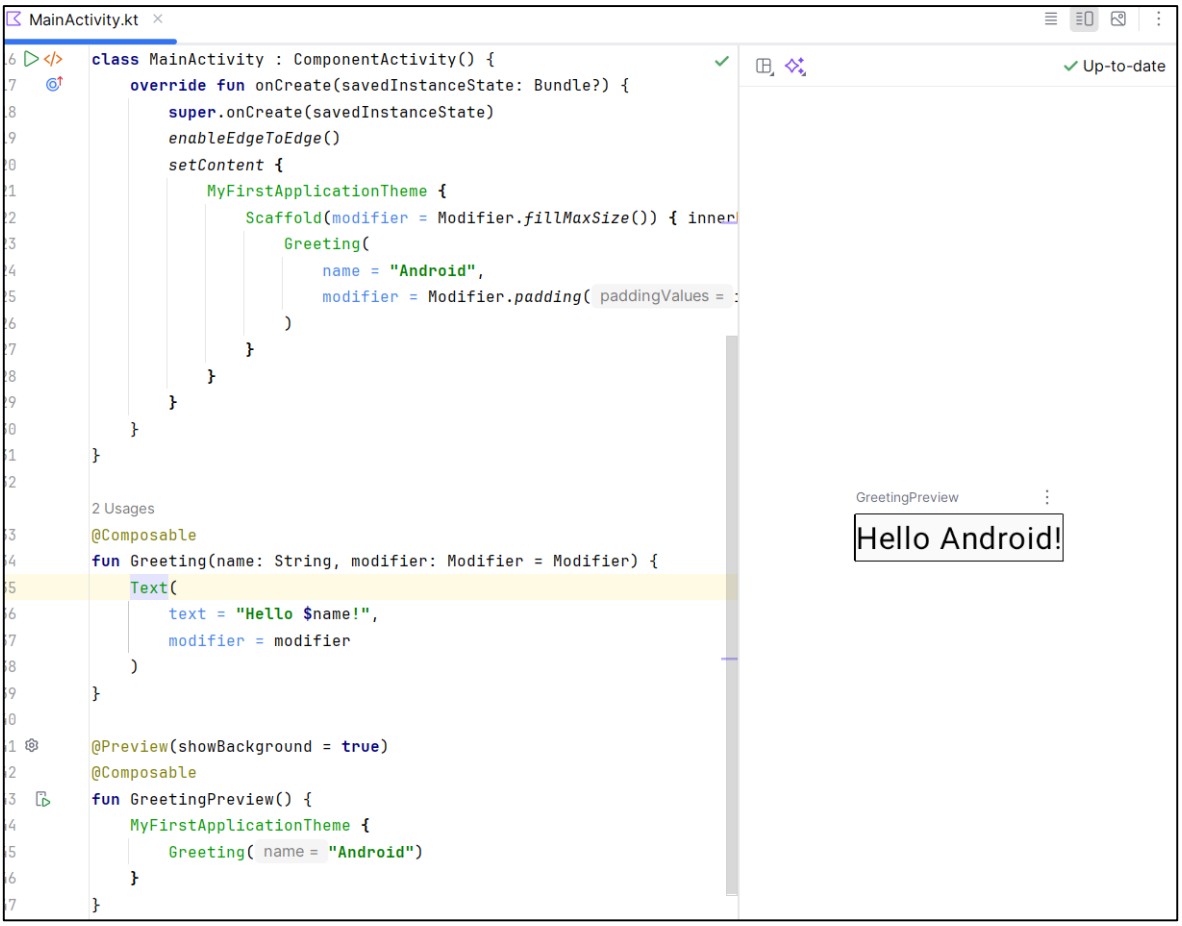


Рисунок 8 – Режим поєднання коду і дизайну

Функція **onCreate()** є точкою входу до цього застосунку та викликає інші функції для реалізації UI. У програмах Kotlin функція **main()** є точкою входу/початковою точкою виконання. У Android-застосунках цю роль виконує **onCreate(…)**.

Функція **setContent{…}** всередині **onCreate(…)** використовується для визначення UI-макета за допомогою компонованих функцій. Усі функції, позначені анотацією **@Composable**, можна викликати з **setContent{…}** або з інших компонованих функцій. Анотація повідомляє компілятору Kotlin, що ця функція використовується Jetpack Compose для створення UI.

Компонована функція **Greeting(…)** приймає певні вхідні дані та генерує те, що відображається на екрані:

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

Змінимо значення **text** на *"Привіт, мене звати $name!"*. Маємо:

Змінимо тло цього напису, наприклад, на зелений.

Щоб встановити інший колір фону для тексту, вам потрібно обрамити текст тегом **Surface –** це контейнером з UI для зміни зовнішнього виду. Щоб обрамити текст тегом **Surface**, виділіть рядок тексту, натисніть (***Alt+Enter*** для Windows або ***Option+Enter*** Mac), виберіть "Оточити віджетом" (Surround with widget).

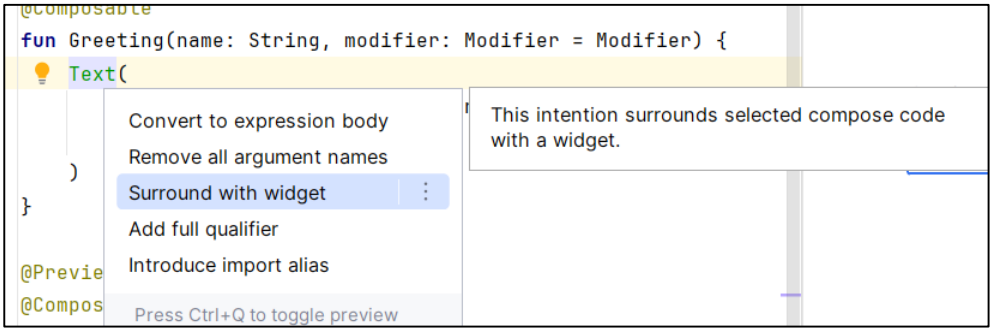


Рисунок 9 – вкладення в тег Surface

Обираємо оточення контейнером. За замовченням це буде **Box**. Замініть **Box** на **Surface(color = Color)**. Додаємо в список імпортованих:

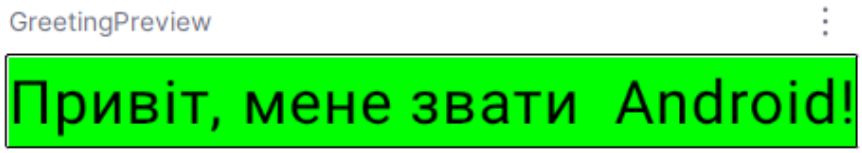
```
import androidx.compose.ui.graphics.Color
import androidx.compose.material3.Surface
```

та оптимізуємо послідовність імпортування за допомогою **Code → Optimize imports**, що усуне потенційні помилки послідовності пошуку в модулях.

Додамо зелений колір в параметр **Color**

```
Surface(color = Color.Green)
```

і отримаємо результат попереднього перегляду:



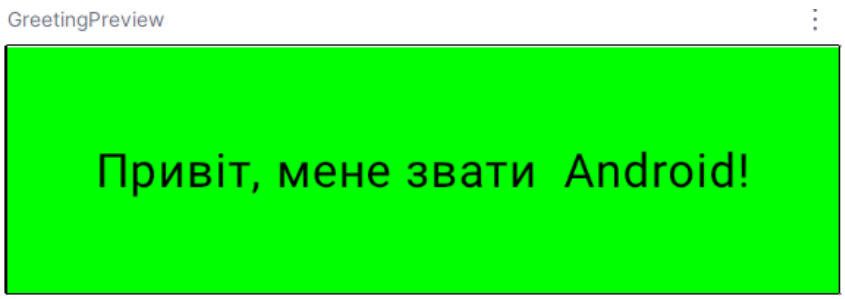
Можна розширити висоту фонового прямокутника. Для цього імпортуємо ще два модулі

```
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.layout.padding
```

знову оптимізуємо імпорти і задамо цю висоту:

```
Text(
    text = "Привіт, мене звати  $name!",
    modifier = modifier.padding(30.dp)
)
```

Отримаємо:



Про інші можливості UI засобами *Material design 3* можна дізнатися тут: https://m3.material.io/get-started

Для роботи в режимі дизайну з палітрою візуальних компонентів UI треба скористатися одним з двох способів:

1. Створювати новий проєкт за шаблоном "Empty Views Activity" замість "Empty Activity", тоді цей шаблон автоматично генерує папку макета і файл **activity_main.xml**, необхідний для роботи з палітрою.
2. Створити файл **activity_main.xml** вручну. Для цього перейдіть до **app** → src → **main** → **res** у навігаторі проєкту, клікніть правою кнопкою миші на папці **res** і виберіть **New** → **Android Resource Directory**, назвіть папку **layout** і оберіть тип ресурсу теж **layout**. Потім клацніть правою кнопкою миші на новій папці **layout** і виберіть New → **Layout Resource File**, naming it **activity_main**. Режим дизайну з палітрою візуальних компонентів UI стане доступним (рис. 10).

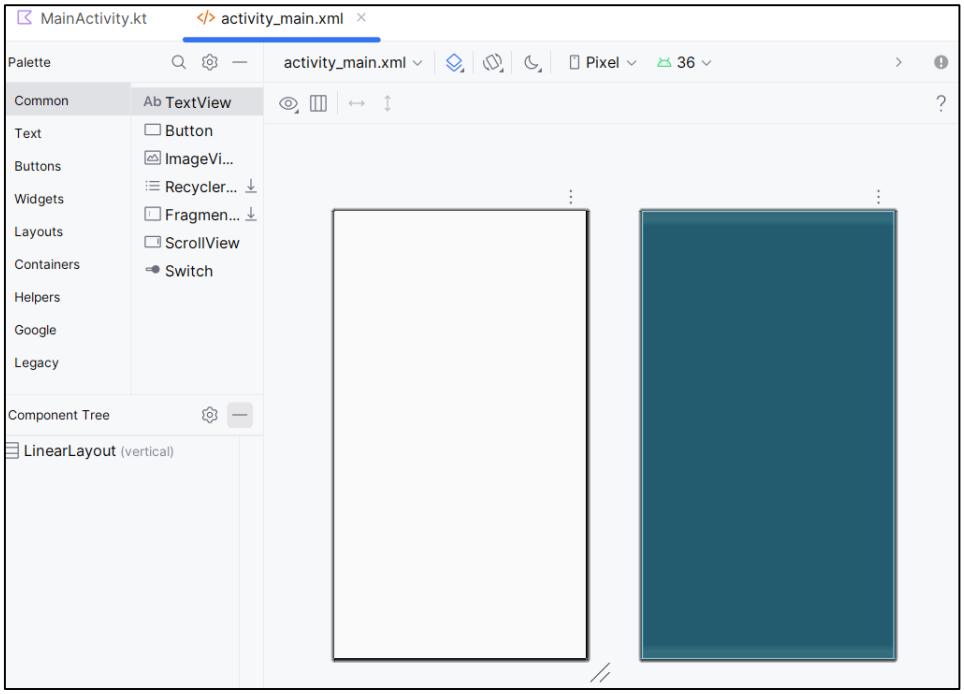


Рисунок 10 – Режим дизайну з палітрою UI-компонентів

Використання способу створення нового проєкту через **Empty Views Activity** спрощує роботу в режимі дизайнера (оскільки автоматично створюється файл **activity_main.xml** та необхідні налаштування), а також розробку для тих, хто використовує мову Java (рис. 11).

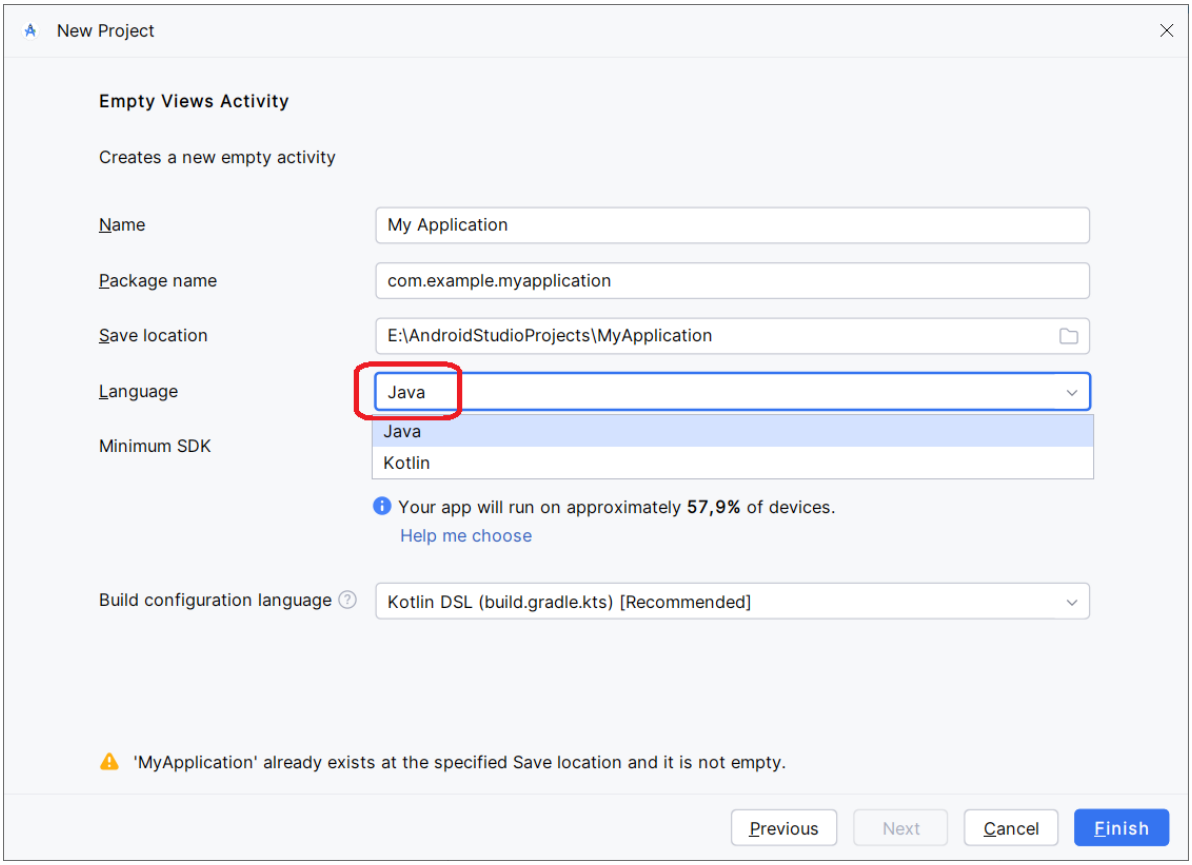


Рисунок 11 – Створення нового проєкту на мові Java

2. Введення і виведення даних

Мета: навчитися використовувати різні UI-елементи для організації введення і виведення інформації в мобільному застосунку.

Теоретичні відомості

Сучасні підходи проєктування програмних застосунків, у т.ч. для мобільних платформ, передбачає виокремлення вигляду в окремий XML-файл (тут: **activity_main.xml**) та логіки.

Розміщення елементів упорядковується такими компонентами як layouts, кожен з яких регламентує спосіб та послідовність елементів UI. Android Studio пропонує такі layouts:

- ***ConstraintLayout*** – контейнер розміщення, який дозволяє позиціонувати елементи відносно батьківського контейнера (parent) або інших *view* за допомогою обмежень (constraints) – горизонтальних та вертикальних ліній, що покращує продуктивність та полегшує верстку; нині є одним з найпопулярніших;
- ***LinearLayout*** – один із базових контейнерів (макетів) в Android Studio, який вирівнює всі дочірні елементи (кнопки, текст тощо) в одному напрямку: вертикально або горизонтально;
- ***FrameLayout*** – найпростіший контейнер (макет), призначений для відображення одного елемента або накладання кількох (View/ViewGroup); за замовчуванням дочірні елементи прикріплюються до верхнього лівого кута; використовується для ефектів накладання або як контейнер фрагментів;
- ***TableLayout*** використовується для групування елементів інтерфейсу у вигляді таблиці, що є зручно для створення форм або структурованих списків параметрів;

- ***TableRow*** – контейнер всередині *TableLayout* для групування дочірніх елементів (наприклад, *TextView* або *Button*) в один горизонтальний ряд;
- ***Space*** – легкий графічний компонент (*View*) для створення порожніх проміжків між іншими елементами UI.

Палітра надає необхідний набір елементів UI – кнопки, поля введення, віджети, елементи для Google Maps та ін. (рис. 12).

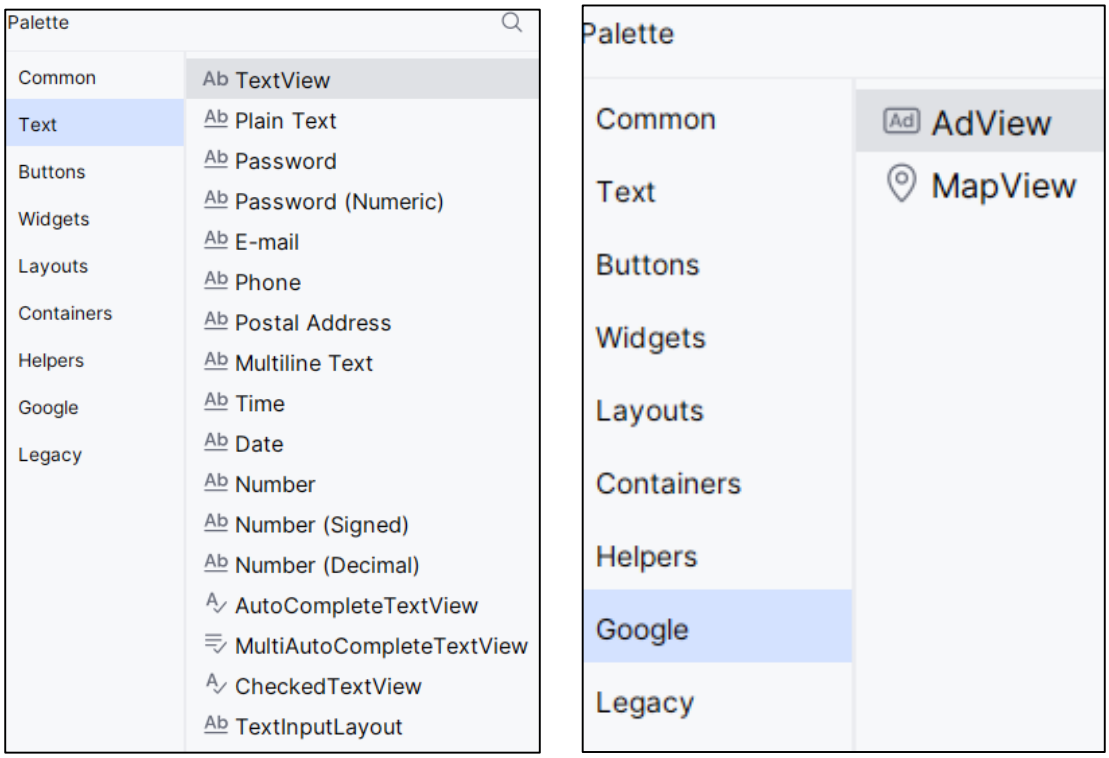


Рисунок 12 – Палітра компонентів UI

Для обробки введених даних необхідно запрограмувати логіку в методах, які обробляють сирцеві дані (наприклад, числа у текстовому форматі), здійснити парсингу у потрібний формат, потрібний для алгоритмів обробки. Якщо дані текстові або числові, їх можна вивести через компонент TextView (група компонентів Text – див. рис. 12).

Приклад

Створимо демонстраційну програму розв'язання квадратного рівняння, коефіцієнти якої вводить користувач з виглядом стартового вікна, як показано на рис. 13.

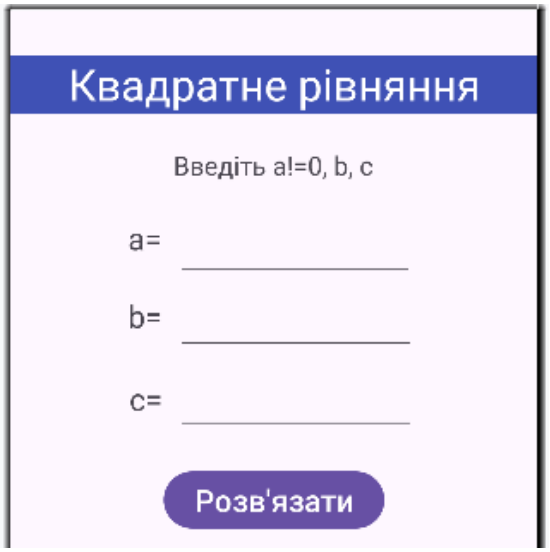


Рисунок 13 – UI програми розв'язанна квадратного рівняння

Для цього створимо новий проєкт як **Empty View Activity** і оберемо мову Java (рис. рис. 11).

Додамо на макет ***ConstraintLayout*** і додамо необхідні елементи:

- Заголовок, запрошення ввести дані, підписи "a=", "b=", "c=" і текстові об'єкти для відображення результатів (значення коренів, одного кореня, повідомлення про відсутність коренів та повідомлення про помилку введення даних) – це об'єкти ***TextView***;
- Три поля для введення значено коефіцієнтів – компоненти типу ***NumberDecimal***; оскільки наші коефіцієнти можуть бути не тільки дробовими, а й із знаком мінус, то треба додати таку можливість у властивість поля вводу (рис. 14).

- Кнопка підтвердження введення коефіцієнтів – компонент ***Button***.

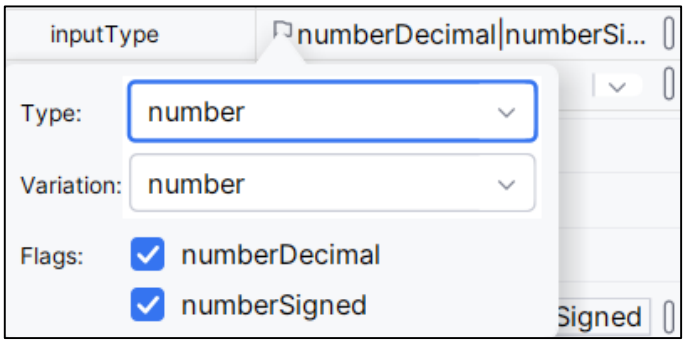
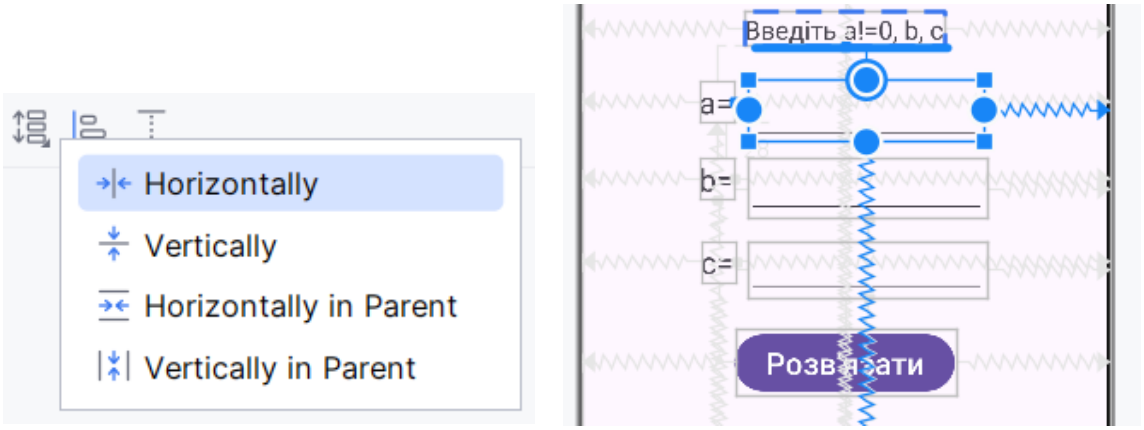


Рисунок 14 – Надання можливості вводити дробові та від'ємні числа

Уже сказано вище, макет ***ConstraintLayout*** дозволяє зробити візуальну прив'язку елементів відносно інших (батьківських), відносно бортів екрану та ін. Наприклад, ***TextView***-елемент з підписом "a=" прив'язаний до батьківського ***TextView***-елемента з підписом "Введіть a!=0, b, c". Таку прив'язку зручно робити звичайним методом перетягування у візуальному редакторі, використовуючи вертикальне та горизонтальне вирівнювання (рис. 15 а) та перетягуючи вузлові точки на рамці елементів до інших елементів (рис. 15 б).



а) б)

Рисунок 15 – Розміщення елементів з ***ConstraintLayout***

Якщо в атрибутах id поля для введення a буде *editTextNumber_a*, id елемента "a=" встановлено *textView_a*, id елемента "Введіть a!=0, b, c" – як *textView_goenter*, то ці id видно у значеннях атрибутів прив'язки, тобто елемент поля для введення *a* після маніпуляцій перетягування візуально прив'язаний до елемента над ним і зліва від нього (рис. 16).

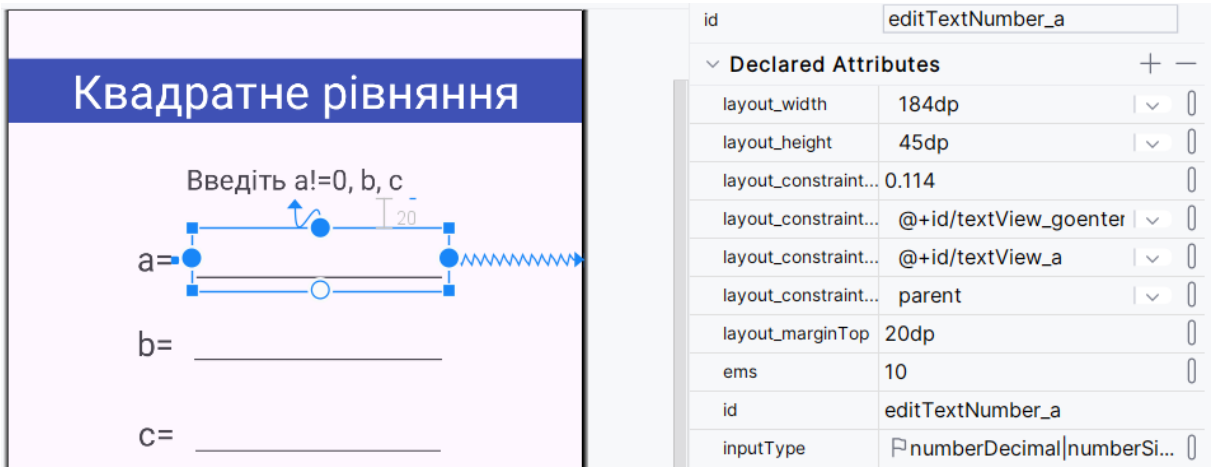


Рисунок 16 – Прив'язка до сусідніх елементів UI

Після додавання і прив'язки елементів у файлі **activity_main.xml** буде відображено макет, ось фрагмент для поля введення коефіцієнта a:

```
<EditText
        android:id="@+id/editTextNumber_a"
        android:layout_width="184dp"
        android:layout_height="45dp"
        android:layout_marginTop="20dp"
        android:ems="10"
        android:inputType="numberDecimal|numberSigned"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.114"
        app:layout_constraintStart_toEndOf="@+id/textView_a"
        app:layout_constraintTop_toBottomOf="@+id/textView_goenter" />
```

Повний текст **activity_main.xml** міститься у Додатку 2.

Для оброблення натиску кнопки створимо метод ***onClickButton()***, у якому обробимо і спарсимо введені дані, обчислимо корені рівняння і покажемо їх або повідомимо про їх відсутність:

```
public void onClickButton(View v) {

    EditText at =(EditText)findViewById(R.id.editTextNumber_a);
    EditText bt =(EditText)findViewById(R.id.editTextNumber_b);
    EditText ct =(EditText)findViewById(R.id.editTextNumber_c);

    double a = Double.parseDouble(at.getText().toString());

    // для виведення результату
    TextView res = (TextView)findViewById(R.id.textView_result);

    // якщо a=0 або не введено, то рівняння вироджене – повідомлення
    if (at.getText().toString()=="" || a==0) {
        res.setText("Коефіцієнт a повинен бути не нуль");
    }
    else {
        double b = Double.parseDouble(bt.getText().toString());
        double c = Double.parseDouble(ct.getText().toString());
        double d, x1, x2;
        d = b*b-4*a*c;
        if (d>0) {
            x1 = (-b-Math.sqrt(d))/(2*a);
            x2 = (-b+Math.sqrt(d))/(2*a);
            res.setText("x1="+x1+"\nx2="+x2);
        }
        else if (d==0) {
            x1 = (-b-Math.sqrt(d))/(2*a);
            res.setText("x="+x1);
        }
        else res.setText("Дійсних коренів немає");
    }
}
```

Прив'яжемо подію натиску кнопки з викликом методу ***onClickButton()***:

Перевіримо роботу програми (рис. 17)

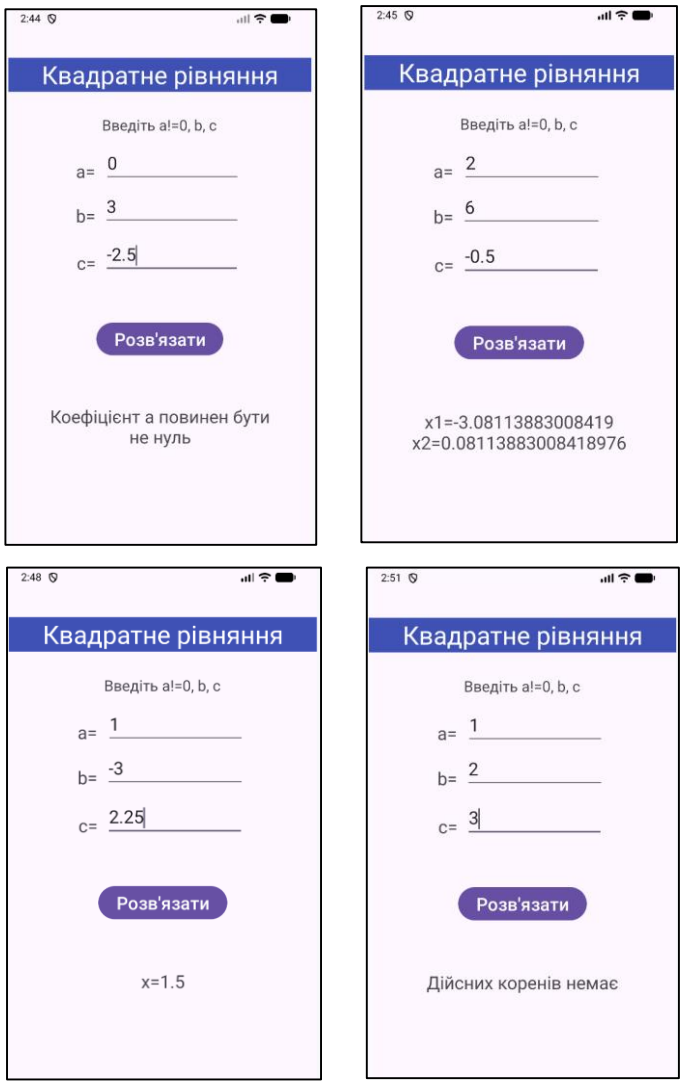


Рисунок 17 – Перевірка роботи програми розв'язання квадратного рівняння

Android Studio надає можливість проводити модульне тестування за допомогою JUnit (рис. 18).

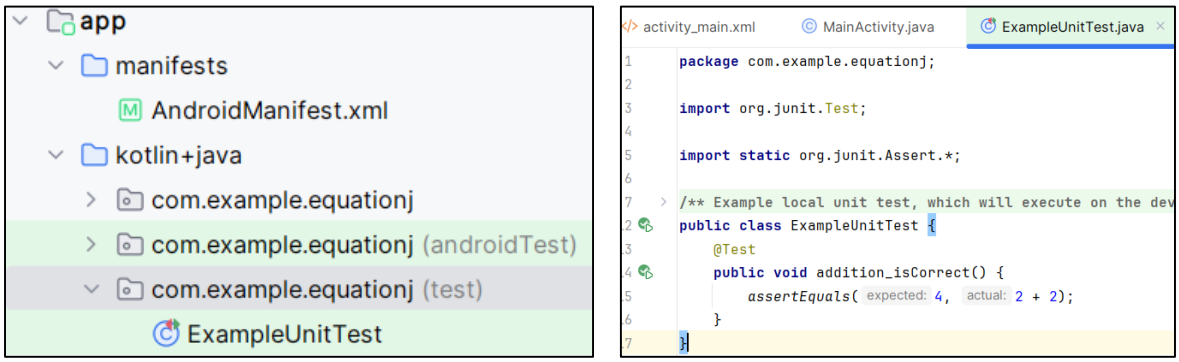


Рисунок 18 – Вбудовані засоби для модульного тестувння

3. Робота з файлами

Мета: опанувати особливості роботи з файлами програм для Android.

Теоретичні відомості

Android використовує файлову систему, схожу на дискові файлові системи інших платформ. Об'єкт типу ***File*** підходить для читання або запису великих обсягів даних в порядку від початку до кінця без пропусків. Цей спосіб оптимальний, наприклад, для зображень або будь-яких інших файлів, що передаються по мережі. Використовуваться бібліотека ***java.io***, що надає небхідний інструментарій для роботи з файлами.

Внутрішнє і ***зовнішнє*** сховища – ці області даних з'явилися в перші роки існування Android, коли на більшості пристроїв були вбудована пам'ять (внутрішнє сховище) і карти пам'яті (наприклад micro SD, зовнішнє сховище). Деякі пристрої ділять вбудовану пам'ять на внутрішній і зовнішній розділи, так що навіть без змінних носіїв в системі дві області зберігання файлів. Доступ до внутнішнього і зовнішнього сховища має специфіку (табл. 1), пов'язану з не завжди доступним зовнішнім, наприклад, карту пам'яті можуть вийняти.

Таблиця 1 – Внутрішнє і зовнішнє сховища в Android

Внутрішнє	Зовнішнє
- Завжди доступне - Збережені файли доступні за замовченням лише застосункам - При видаленні програми Android видаляє з внутрішньої пам’яті всі файли цієї програми	- Доступне не завжди - Збережені файли доступні всюди - При видаленні програми Android видаляє із зовнішньої пам’яті всі файли програми, які зберігається в папці з ***getExternalFilesDir()***

Для запису у зовнішнє сховище слід вказати запит дозволу **WRITE_EXTERNAL_STORAGE** у файлі маніфесту:

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

Якщо програмі буде потрібно зчитати дані із зовнішнього сховища (але не записати туди дані), необхідно буде декларувати дозвіл READ_EXTERNAL_STORAGE. Щоб забезпечити передбачувану подальшу роботу застосунку, треба одразу декларувати цей дозвіл до того, як зміни вступили в силу:

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

Якщо програма використовує дозвіл WRITE_EXTERNAL_STORAGE, вона непрямо отримує дозвіл на читання даних із зовнішнього сховища.

Для збереження файлів у внутрішньому сховищі не потрібно ніяких дозволів. У програми завжди буде дозвіл на читання і запис файлів в її каталог внутрішньої пам'яті.

Збереження даних у внутрішньому сховищі. Використовують методи:

- ***getFilesDir ()*** – повертає об’єкт ***File***, який відповідає внутрішній папці програми
- ***getCacheDir ()*** – повертає об’єкт ***File***, який відповідає внутрішній папці файлів тимчасової кеш-пам'яті програми.

Для створення файлу в одній з цих папок використовується конструктор ***File()***, який передає елемент ***File***, що надається одним із

зазначених методів, за допомогою якого вказується директорія у внутрішній пам'яті. Наприклад:

```
File file = new File(context.getFilesDir(), filename);
```

Крім того, можна викликати метод ***openFileOutput()*** для отримання об'єкта ***FileOutputStream***, що записує у файл внутрішньої пам'яті. Приклад:

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {

        outputStream = openFileOutput(filename,Context.MODE_PRIVATE);
        outputStream.write(string.getBytes());
        outputStream.close();

    } catch (Exception e) {
         e.printStackTrace(); }
```

Якщо потрібно кешувати якісь файли, слід використовувати ***createTempFile()***. У прикладі метод витягує ім'я файлу з URL і створює файл з цим ім'ям в папці внутрішньої кеш-пам'яті програми:

```
public File getTempFile(Context context, String url) {
 File file;
 try {
  String fileName = Uri.parse(url).getLastPathSegment();
  file = File.createTempFile(fileName, null,
context.getCacheDir());

  } catch (IOException e) {
        // помилка створення файна
        }
        return file;
}
```

Папка програми у внутрішній пам’яті вказується з використанням імені пакета програми в певному місці файлової системи Android. Технічно інша

програма може прочитати ваші файли у внутрішній пам'яті, якщо встановити для файлів режим *Readable* (доступно для читання). Однак для цього іншій програмі повинні бути відомі ім'я пакета вашої програми та імена файлів. Інші програми не можуть переглядати внутрішні папки вашої програми і не мають дозволів на читання або запис, якщо ви спеціально не встановите для своїх файлів режим *Readable* або *Writable*. Отже, поки ви будете використовувати режим MODE_PRIVATE для своїх файлів у внутрішній пам'яті, вони будуть недоступні іншим додаткам.

Збереження у зовнішньому сховищі. Оскільки зовнішнє сховище може бути недоступно, перед використанням завжди слід перевіряти його доступність. Стан зовнішнього сховища можна дізнатися через метод ***getExternalStorageState()***. Якщо повертається MEDIA_MOUNTED, ви зможете виконувати з файлами операції читання і запису. Приклад перевірки доступності зовнішнього сховища для читання і запису:

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Перевірка доступності зовнішнього сховища для читання:

```
public boolean isExternalStorageReadable() {
   String state = Environment.getExternalStorageState();

   if (Environment.MEDIA_MOUNTED.equals(state) ||
Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){

       return true;
   }
   return false;
}
```

Хоча зовнішнє сховище може бути змінено користувачем та іншими програмами, існує дві категорії файлів, які в ньому можна зберігати:

- *Загальнодоступні файли* – ті, які повинні бути доступні іншим програмам і користувачеві; коли користувач видаляє вашу програму, ці файли повинні залишатися доступні користувачеві; наприклад, в цю категорію входять фотознімки, зроблені за допомогою вашої програми, а також інші компоненти для завантаження;
- *Особисті файли* – ті, що належать вашій програмі, вони повинні видалятися при видаленні програми; хоча технічно ці файли доступні для користувача та інших програм (оскільки знаходяться в зовнішньому сховищі), вони не мають жодної цінності для користувачів поза програмою; коли користувач видаляє програму, система видаляє всі файли з папки закритих файлів програми на зовнішньому сховищі; наприклад, до цієї категорії відносяться додаткові ресурси, завантажені програмою, тимчасові мультимедійні файли і т.п.

Якщо ви хочете зберегти публічні файли в зовнішньому сховищі, використовуйте методи ***getExternalStoragePublicDirectory ()*** для отримання об'єкту ***File***, що відображає відповідну папку в зовнішньому сховищі.

Цей метод приймає аргумент (тип файлу) і дозволяє включити його в логічну структуру з іншими публічними файлами, наприклад, DIRECTORY_MUSIC або DIRECTORY_PICTURES. Приклад – отримання папки з фотографіями користувача:

```
public File getAlbumStorageDir(String albumName) {
    File file = new
File(Environment.getExternalStoragePublicDirectory(
Environment.DIRECTORY_PICTURES), albumName);

    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

Якщо треба зберегти особисті файли вашої програми, можна створити відповідну папку через метод ***getExternalFilesDir()*** і надати їй ім'я із зазначенням бажаного типу каталогу. Кожна папка, яка створюється таким способом, додається в батьківський каталог, де об'єднані всі файли програми у зовнішньому сховищі. При видаленні програми система видаляє цю папку. Приклад створення папки для персонального фотоальбому:

```
public File getAlbumStorageDir(Context context, String albumName) {
    File file = new File(context.getExternalFilesDir(
Environment.DIRECTORY_PICTURES), albumName);

    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }

    return file;
}
```

Якщо жодне з готових імен підкаталогів не підходить для ваших файлів, ви можете викликати ***getExternalFilesDir ()*** і передати аргумент null. При цьому буде повернуто кореневу папку закритої папки програми у зовнішньому сховищі. ***getExternalFilesDir()*** створює папку всередині папки, яка видаляється при видаленні програми. Якщо збережені файли повинні залишатися доступними після видалення програми (наприклад, знімки), слід використовувати ***getExternalStoragePublicDirectory().***

Незалежно від того, чи використовуєте ви ***getExternalStoragePublicDirectory()*** для загальних файлів або **getExternalFilesDir()** для власних файлів програми, ви повинні використовувати імена папок, які надаються постійними значеннями API-інтерфейсів, наприклад, DIRECTORY_PICTURES. Ці імена папок забезпечують правильну обробку файлів системою. Наприклад, збережені в DIRECTORY_RINGTONES файли автоматично відносяться медіа-сканером системи в категорію рингтонів, а не музики.

Запит доступного простору. Якщо заздалегідь відомо обсяг даних для збереження, можна визначити наявність достатнього простору без винятку ***IOException***, викликавши метод ***getFreeSpace ()*** або ***getTotalSpace ()***.

Ці методи дозволяють визначити:

- поточний обсяг пам'яті
- доступний об’єм у сховищі
- і загальний простір у сховищі

Ця інформація також дозволяє уникнути переповнення сховища понад певного рівня. Однак система не гарантує можливість запису такої ж кількості байт, як зазначено в ***getFreeSpace ()***.

Якщо виведене число на кілька мегабайт перевищує розмір даних, які ви хочете зберегти, або якщо файлова система заповнена менш, ніж на 90%, далі можна діяти спокійно. В іншому випадку запис у сховище здійснювати небажано.

Загалом не обов'язково перевіряти обсяг доступного місця перед збереженням файлу. Можна спробувати відразу записати файл, а потім обробити подію IOException, якщо вона виникне. Це може знадобитися, якщо ви точно не знаєте, скільки потрібно вільного місця. Наприклад, при збереженні PNG-файлу у форматі JPEG.

Видалення файлу. Найпростіший спосіб - викликати у відкритому файлі посилання ***delete ()*** на сам цей файл:

```
myFile.delete();
```

Якщо файл збережений у внутрішньому сховищі, можна викликати ***deleteFile ()*** через ***Context***:

```
myContext.deleteFile(fileName);
```

При видаленні користувачем програми Android видаляє наступне:

- Всі файли, збережені у внутрішньому сховищі
- Всі файли, збережені в зовнішньому сховищі з використанням ***getExternalFilesDir()***.

Однак слід регулярно вручну очищати кеш-пам'ять, щоб видалити файли, створені за допомогою ***getCacheDir()***, а також видаляти будь-які інші непотрібні файли.

Приклад

Створимо застосунок, який табулює функцію $y = \sin x + 2\cos x$ на проміжку [a; b] з кроком dx, де a, b, dx вводить користувач. Підготуємо макет застосунку (рис. 19). Для виведення даних додамо в нижній частині вікна елемент Multiline Text – у нього можна і вводити, і виводити інформацію та застосовувати прокрутку, якщо вона вся не вміщується у виділену область. У прикладі обчислені значення запишемо у текстовий файл у внутрішнє сховище (тобто у маніфесті дозволів у даному випадку не потрібно). Ідентифікатори UI-елементів, потрібних для обробки даних:

- **editTextTextMultiLine_res** – поле для багаторядкового тексту, для виведення результатів читання з файлу;
- **editTextNumberDecimal_a** – поле для вводу значення *a*, може бути від'ємним і дробовим;
- **editTextNumberDecimal_b** – поле для вводу значення **b**, може бути від'ємним і дробовим;
- **editTextNumberDecimal_dx** – поле для вводу значення **b**, може бути дробовим, не може бути від'ємним;
- **button_tab** – кнопка запуску табуляції і запису у файл;
- **button_read** – кнопка читання з файлу і виведення результатів у **editTextTextMultiLine_res**.

Для зручності у поля a, b, dx одразу підставимо значення за замовченням (-5, 5, 0.5) для зручності тестування збірки.

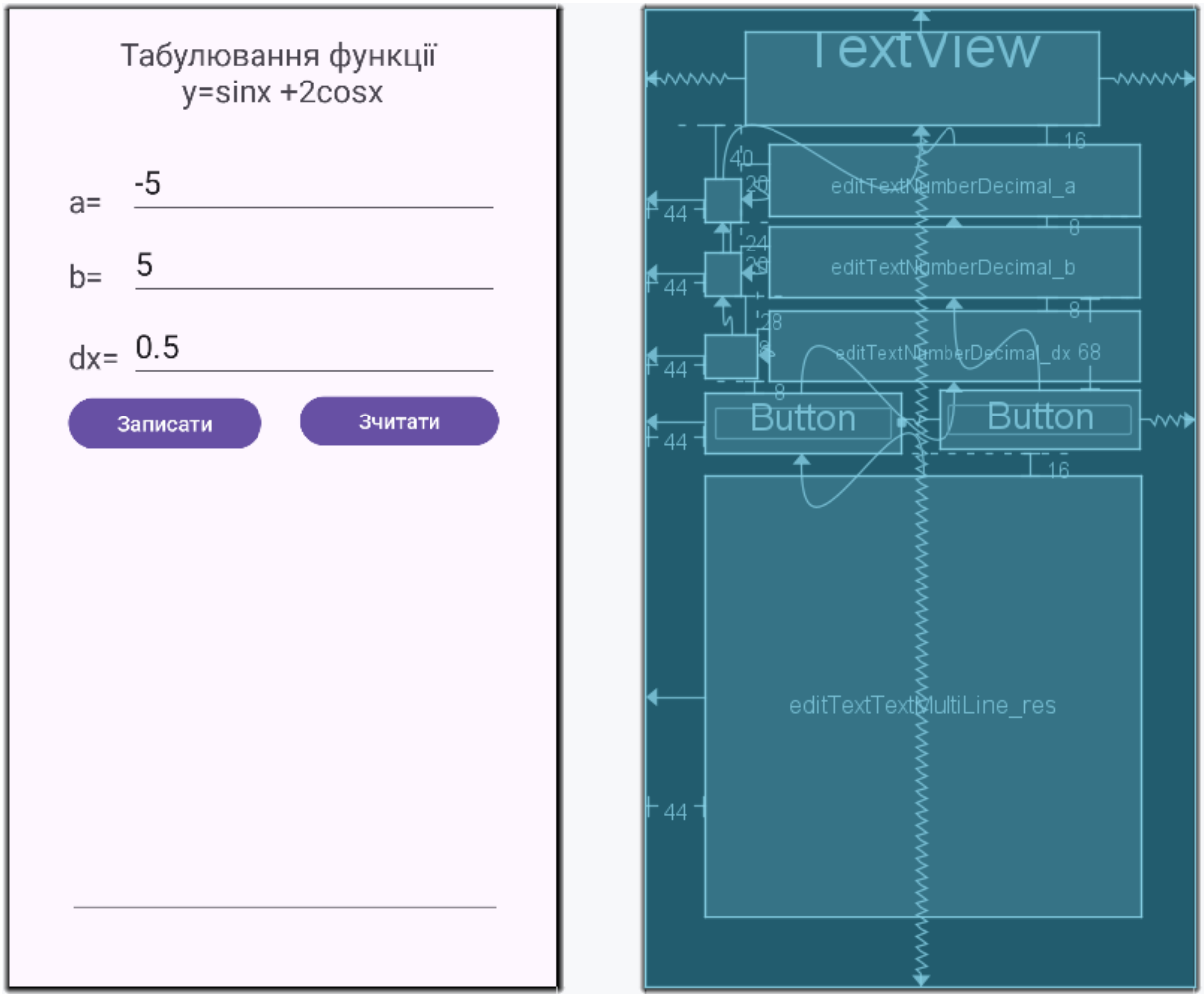


Рисунок 19 – Макет програми табулювання функції

У випадку успішного запису у файл виведемо тост – коротке спливаюче повідомлення внизу екрана (рис. 20 а). Ситуації невдалого запису або читання з файлу відобразимо як тост-повідомлення.

Випадок, коли значення лівого кінця проміжку менше правого кінця і якщо dx виявиться не додатнім, перехопимо це і повидомимо про помилку при введенні даних.

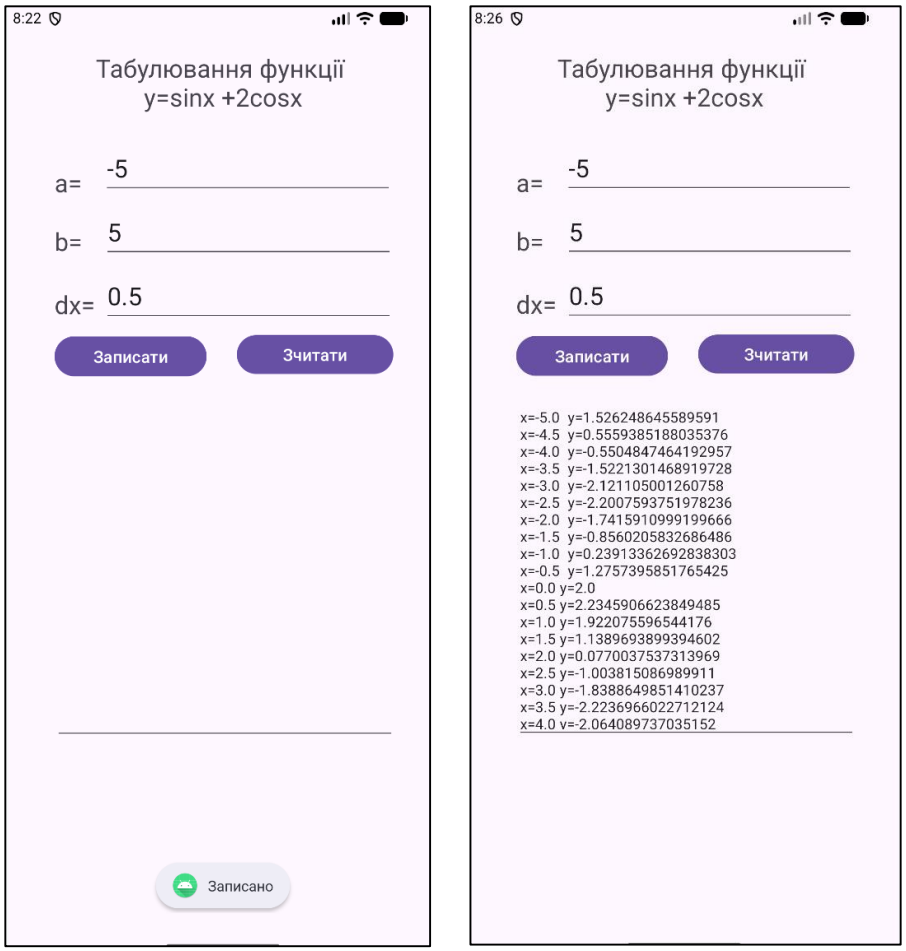
Нехай файл називається ***tabfunc.txt***. Ось код методу запису у файл, до нього залінкована подія натиску кнопки ("Записати"):

```
public void onClickButtonWriteFile(View v) throws
FileNotFoundException {

    TextView res = (TextView)
findViewById(R.id.editTextTextMultiLine_res);
    String content = "";
    String filename = "tabfunc.txt";
    File path = getApplicationContext().getFilesDir();

    EditText at = (EditText)
findViewById(R.id.editTextNumberDecimal_a);
    EditText bt = (EditText)
findViewById(R.id.editTextNumberDecimal_b);
    EditText dxt = (EditText)
findViewById(R.id.editTextNumberDecimal_dx);

    double a = Double.parseDouble(at.getText().toString());
    double b = Double.parseDouble(bt.getText().toString());
    double dx = Double.parseDouble(dxt.getText().toString());

    if (a>b || dx<=0) {
        res.setText("Помилка: має бути a<=b, dx>0");
    }
    else

    try {
        FileOutputStream writestream = new FileOutputStream(new
File(path, filename));
        double x = a, y;
        String line = "";

        // цикл табулювання
        while (x <= b) {
            y = Math.sin(x) + 2 * Math.cos(x);
            line = "x=" + x + "\ty=" + y + '\n';
            writestream.write(line.getBytes());
            x += dx;
        }

        writestream.close();

        // тост-повідомлення про успішний запис у файл
        Toast.makeText(getApplicationContext(), "Записано",
Toast.LENGTH_SHORT).show();

    } catch (Exception e) {
        Toast.makeText(getApplicationContext(), "Не вдалося
записати", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}
```

Нижче подано метод читання з файлу.

```
public void onClickButtonReadFile(View v) {
    TextView res = (TextView)findViewById(R.id.editTextTextMultiLine_res);
    String filename = "tabfunc.txt";
    File path = getApplicationContext().getFilesDir();
    File fread = new File(path, filename);
    byte [] content = new byte[(int)fread.length()];
    try {
        FileInputStream readstream = new FileInputStream(fread);
        readstream.read(content);
        res.setText(new String(content));
        readstream.close();
    }
    catch (Exception e) {
        Toast.makeText(getApplicationContext(), "Не вдалося прочитати",
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}
```

Демонстрація успішного запису і тост-повідомлення (рис. 20 а) та виведення інформації з файлу (рис. 20 б).



а) б)

Рисунок 20 – Демонстрація роботи програми табуляції функції

4. Взаємодія з БД

Мета: ознайомитися можливостями використання API SQLite для рообти з локальною БД.

Теоретичні відомості

SQLite є API для однокористувацького доступу до БД. По суті це робота з файлом засобами SQL, що усуває потребу в модулях, які управляють записом і читанням файлу. Водночас, для проєктів зі складною структурою даних SQLite може не підійти. Обмеження SQlite:

- Не підходить для великих проектів;
- Різна сумісність версій;
- Погано працює на об’ємних транзакціях;
- Обмежена кількість стовпців за замовченням.

Таблиця 2 – Базові типи SQlite

Тип	**Опис**
NULL	пусте значення
INTEGER	ціле
REAL	з плаваючою точкою
TEXT	Рядки або символи в кодах UTF-8, UTF-16BE або UTF-16LE
NUMERIC	може зберігати логічні значення, дату, час
BLOB	бінарні дані

Рекомендують не використовувати тип BLOB для зберігання даних (картинки) в Android. Краще зберігати в базі шляхи до зображень, а самі зображення зберігати в файлової системі.

Оскільки сама БД SQLite є файлом, то по суті при роботі з базою даних ви взаємодієте з файлом. Тому операції R/W можуть бути досить повільним, і рекомендується використовувати асинхронні операції, наприклад, за допомогою класу **AsyncTask**.

Коли програма створить БД, вона зберігається в папці

DATA/дані/Ім’я_пакета/база_даних/і’мя_бази.db

Метод ***Environment.getDataDirectory()*** повертає шлях до папки **DATA**. Основні пакети для роботи:

- **android.database**
- **android.database.sqlite**

База даних SQLite доступна тільки з програмою, яка її створила. Для доступу з інших програм можна використати контент-провайдери (**ContentProvider**).

Визначення схеми і контракту. ***Схема*** – формальна декларація способу організації бази даних. Схема відображається у виразах SQL, що використовуються для створення БД. Може виявитися корисним створити супутній клас (**клас-контракт**), який явно описує структуру схеми.

Клас-контракт є контейнером, що визначає імена для URI-адреси, таблиці і стовпчики. Клас-контракт дозволяє використовувати одні і ті ж постійні значення в усіх інших класах цього ж пакета. Наприклад, при зміні імені стовпця в одному місці це зміна застосовується у всьому коді. Для зручності глобальні визначення бази даних варто розмістити на кореневому рівні класу. Потім потрібно створити внутрішній (inner) клас для кожної таблиці.

За рахунок реалізації інтерфейсу ***BaseColumns*** внутрішній клас може успадковувати поле первинного ключа **_ID**, яке буде затребуване деякими класами Android (наприклад, адаптерами курсора). Це не є обов'язковим, однак може допомогти забезпечити гармонійну роботу бази в інфраструктурі Android.

Приклад: визначення імені таблиці та імена стовпців однієї таблиці:

```
public final class ClassContract {
    public ClassContract () {
    }
    /* Вкладений клас визначає вміст таблиці */
    public static abstract class TableStructure implements
BaseColumns {

       public static final String TABLE_NAME = "tablename";
       public static final String COLUMN_NAME_ENTRY_ID = "idcolumn";
       public static final String COLUMN_NAME_TITLE1 = "title1";
       public static final String COLUMN_NAME_TITLE2 = "title2";
       ...
    }
}
```

Створення БД.

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";

private static final String SQL_CREATE_ENTRIES =
      "CREATE TABLE " + TableStructure.TABLE_NAME + " (" +
       TableStructure._ID + " INTEGER PRIMARY KEY," +
       TableStructure.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_SEP +
       TableStructure.COLUMN_NAME_TITLE1 + TEXT_TYPE + COMMA_SEP +
       ... // Інші опції для команди CREATE
       )";"

private static final String SQL_DELETE_ENTRIES =
              "DROP TABLE IF EXISTS " + TableStructure.TABLE_NAME;
```

Як і при збереженні файлів у внутрішній пам'яті пристрою, Android зберігає БД в закритій області диску, пов'язаній з програмою. Ці дані захищені, тому що ця область за замовчуванням не доступна іншим програмам. Корисний набір API-інтерфейсів міститься в класі **SQLiteOpenHelper**. Якщо ви використовуєте цей клас для отримання посилань на свою базу даних, система виконує потенційно довготривалі операції створення та оновлення бази даних тільки тоді, коли це необхідно, а не під час запуску програми.

Для використання **SQLiteOpenHelper** створіть підклас, який перевизначить методи виклику

- ***onCreate()***
- ***onUpgrade()***
- ***onOpen()***

Також ви можете використовувати ***onDowngrade()***, але це не обов'язково. Для цього потрібно використовувати виклик ***getWritableDatabase()*** або ***getReadableDatabase()***, причому бажано у фоновому режимі (наприклад, з ***AsyncTask*** або ***IntentService***):

```
public class DatabaseHelper extends SQLiteOpenHelper {

    // Якщо змінюєте схему БД, ви повинні підвищити версію БД
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "Mydatabase.db";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion)

    // Ця БД є лише кешем для онлайн даних, отже, політика оновлення
    // повинна просто відкинути дані і почати заново

        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

Для отримання доступу до БД треба створити екземпляр нащадка **SQLiteOpenHelper**:

```
DatabaseHelper myDbHelper = new DatabaseHelper(getContext());
```

Додавання записів у БД. Додайте дані в базу через передачу об'єкта ***ContentValues*** в методі ***insert()***:

```
// Отримати базу в режимі запису
SQLiteDatabase db = myDbHelper.getWritableDatabase();

// Створити нову карту контенту, де імена колонки є ключами
ContentValues values = new ContentValues();

values.put(TableStructure.COLUMN_NAME_ENTRY_ID, id);
values.put(TableStructure.COLUMN_NAME_TITLE1, title1);
values.put(TableStructure.COLUMN_NAME_TITLE2, title2);

// Вставити новий рядок і повернути його первинний ключ
long newRowId;

newRowId =
db.insert(TableStructure.TABLE_NAME,TableStructure.COLUMN_NAME
_NULLABLE, values);
newRowId =
db.insert(TableStructure.TABLE_NAME,TableStructure.COLUMN_NAME
_NULLABLE, values);
```

Перший аргумент ***insert()*** є просто назвою таблиці. Другий аргумент вказує ім'я стовпця, в який інфраструктура вставляє значення ***NULL***, якщо ***ContentValues*** є порожнім (якщо замість цього вказати "null", то інфраструктура не буде вставляти рядок, якщо значень немає)/

Читання з БД. Для читання з бази даних можна використати ***query()*** з передачею критеріїв вибірки і стовпців. Метод поєднує ***insert()*** і ***update().*** Але при цьому список стовпців визначає дані, які треба **отримати**, а не дані для вставки. Результати запиту повертаються в об'єкті **Cursor**.

```
SSQLiteDatabase db = myDbHelper.getReadableDatabase();

// Визначення масиву projection, який визначає,
// які стовпці будуть використані після цього запиту.

String[] projection = {
        TableStructure._ID, TableStructure.COLUMN_NAME_TITLE1,
TableStructure.COLUMN_NAME_UPDATED, ... };
```

Порядок сортування результатів в Cursor:

```
String sortOrder = TableStructure.COLUMN_NAME_UPDATED + "
DESC";
Cursor c = db.query(
        TableStructure.TABLE_NAME, // Таблиця у запиті
        projection,      // Стовпці для повернення
        selection,       // Рядки для вибірки у WHERE
        selectionArgs,   // Значення для WHERE
        null,            // Не групувати рядки
        null,            // Не фільтрувати групи рядків
        sortOrder        // Порядок сортування
);
```

Щоб подивитися на рядок в місці курсору, використовуйте один з методів переміщення курсора, які завжди потрібно викликати перед зчитуванням значень. Зазвичай слід починати з виклику ***moveToFirst(),*** який розміщує "позицію читання" на перший запис в результаті.

Для кожного рядка значення стовпця можна прочитати через один з get методів класу ***Cursor***, наприклад ***getString()*** або ***getLong()***. Для кожного з методів ***get*** треба передати вказівник бажаного стовпця, який може викликати ***getColumnIndex()*** або ***getColumnIndexOrThrow().***

```
c.moveToFirst();

long itemId =
c.getLong(c.getColumnIndexOrThrow(TableStructure._ID));
```

Видалення інформації з БД. Для видалення рядків з таблиці потрібно вказати критерії виділення, які ідентифікують рядки. API бази даних забезпечує механізм для створення критеріїв виділення, що надає захист від включення SQL-коду. Цей механізм ділить специфікацію вибору на пропозицію вибору і аргументи вибору. *Пропозиція* визначає стовпці для розгляду, а також дозволяє об'єднувати операції тестування стовпців. *Аргументи* є значеннями для тестування, які прив'язані до пункту. Оскільки

результат обробляється не як звичайний вираз SQL, він захищений від несанкціонованого виконання SQL-коду.

```
// Визначення частини 'where' в запиті
String selection = TableStructure.COLUMN_NAME_ENTRY_ID + "
LIKE ?";

// Опис аргументів у порядку заповнення
String[] selectionArgs = { String.valueOf(rowId) };

// Застосування SQL-виразу (виклик)
db.delete(table_name, selection, selectionArgs);
```

Оновлення даних у БД. За необхідності, оновлення значень бази застосовуємо метод ***update()***. Оновлення таблиці поєднує значення синтаксису ***insert ()*** і синтаксису ***where*** для ***delete ()***.

```
SQLiteDatabase db = myDbHelper.getReadableDatabase();

// Нове значення для одного стопвця
ContentValues values = new ContentValues();
values.put(TableStructure.COLUMN_NAME_TITLE, coltitle);

// Який рядок оновити на основі ID
/String selection = TableStructure.COLUMN_NAME_ENTRY_ID + "
LIKE ?";

String[] selectionArgs = { String.valueOf(rowId) };

int count = db.update(DatabaseHelper.TableStructure.TABLE_NAME,
values, selection, selectionArgs);
```

5. Побудова графіків і діаграм

Мета: навчитися будувати графіки та діаграми в мобільному застосунку з використанням однієї з популярних бібліотек.

Теоретичні відомості

Існує ряд API для швидкої побудови графіків і діаграм (табл. 3).

Таблиця 3 – Популярніші бібліотеки для побудови діаграм для Android

Бібліотека	Основні можливості	Для чого краще підходить
MPAndroidChart [8-10]	Багатофункціональна, підтримує різні діаграм (лінійні, стовпчасті, кругові, радарні тощо), масштабування перетягування та ін.	Складні діаграми, тонкі налаштування та великі набори даних.
GraphView [11]	Легко інтегрується, підтримує лінійні та стовпчасті діаграми, оновлення в режимі реального часу та пропонує базові функції масштабування та прокручування.	Прості лінійні/стовпчасті діаграми та візуалізація даних у режимі реального часу з простою реалізацією.
AnyChart [12]	Корпоративного рівня, широкий вибір типів діаграм, хороша документація, використовує WebView для рендерингу.	Складні діаграми, глибокі налаштування та великі набори даних.

Побудова графіків і діаграм в Android передбачає внесення змін у конфігураційні файли ***build.gradle.kts***, ***settings.gradle.kts***, а також у файл ***activity_main.xml***, куди треба додати блок з інформацією про зовнішні підключені класи. Для конфігуратора на рівні проєкту (***settings.gradle.kts***) в розділі dependencyResolutionManagement додаємо рядок з адресою репозиторія

```
maven { url = uri("https://jitpack.io") }
```

Для різних типів проєктів синтаксис підключення може також відрізнятися [10], наприклад :

```
maven { url 'https://jitpack.io' }
```

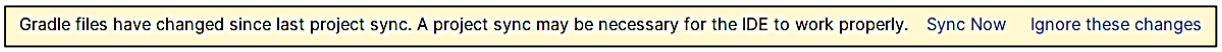
У файлі конфігурації на рівні застосунку в розділі залежностей (*dependencies*) треба додати рядок з імплементацією обраної бібліотеки. Для ***build.gradle.kts*** і використання бібліотеки ***MPAndroidChart*** версії v3.1.0 :

```
implementation("com.github.PhilJay:MPAndroidChart:v3.1.0")
```

Для інших типів збірок синтаксис може відрізнятися, наприклад:

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Після додавання рядка треба синхронізувати зміни з усім проєктом:



Після конфігурування збірки у файлі ***activity_main.xml*** підключаємо клас, який відповідає потрібному типу діаграм (рис. 21).

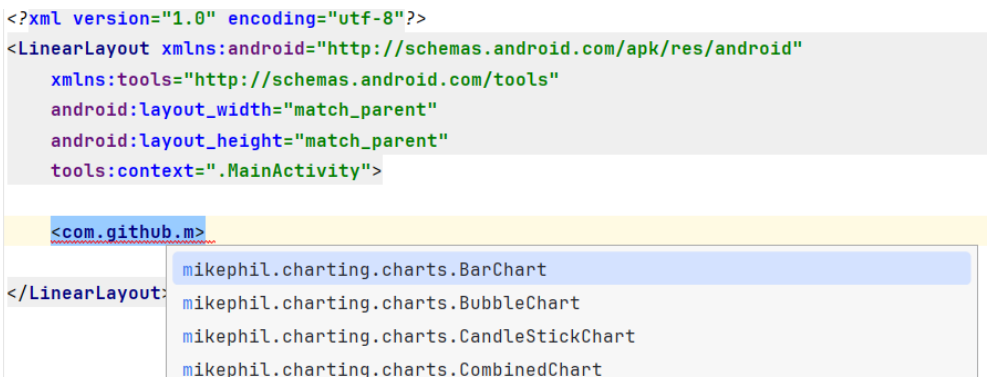


Рисунок 21 – Підключення класу у activity_main.xml

Приклад

Створимо проєкт ***Chart*** застосунку, який будує лінійний графік температури повітря протягом доби у двох містях України, наприклад, Києві та Львові. Використаємо бібліотеку ***MPAndroidChart*** версії v3.1.0.

У файлі ***settings.gradle.kts*** додаємо репозиторій:

```
dependencyResolutionManagement {

repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven { url = uri("https://jitpack.io") }
    }
}
```

У файлі ***build.gradle.kts*** додаємо залежність:

```
dependencies {
    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)

    implementation("com.github.PhilJay:MPAndroidChart:v3.1.0")
}
```

У файлі ***activity_main.xml*** встановлюємо лінійне розміщення (***LinearLayout***) і додаємо посилання на клас ***LineChart*** для лінійних діаграм:

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Температура (Цельсій)"
        android:textAlignment="center"
        android:textSize="16sp"/>

    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/chart"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

Нижче подано файл ***MainActivity.java*** і результат роботи програми. Зверніть увагу на підключені бібліотеки, конфігурацію легенди, осей X і Y, кольору і товщини ліній:

```
package com.example.chart;

import android.graphics.Color;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import
com.github.mikephil.charting.formatter.IndexAxisValueFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private LineChart lineChart;
    private List<String> xValues;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        lineChart = findViewById(R.id.chart);
```

```
        lineChart.getAxisRight().setDrawLabels(false);

        // нехай показники температури – кожні 6 годин
        xValues = Arrays.asList("0:00", "6:00", "12:00", "18:00");

        // конфігурація осей { та Y
        XAxis xAxis = lineChart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setValueFormatter(new
IndexAxisValueFormatter(xValues));
        xAxis.setLabelCount(4);
        xAxis.setTextSize(12f); //float
        xAxis.setGranularity(1f);

        YAxis yAxis = lineChart.getAxisLeft();
        yAxis.setAxisMinimum(-30f);
        yAxis.setAxisMaximum(30f);
        yAxis.setAxisLineWidth(2f);
        yAxis.setAxisLineColor(Color.BLACK);
        yAxis.setLabelCount(10);
        yAxis.setTextSize(12f);

        // Додавання показників температури в список даних
        // для Києва (entries1) та Львова (entries2
        List <Entry> entriesKyiv = new ArrayList<>();
        entriesKyiv.add(new Entry(0, -7f));
        entriesKyiv.add(new Entry(1, -10f));
        entriesKyiv.add(new Entry(2, 6f));
        entriesKyiv.add(new Entry(3, 5f));

        List <Entry> entriesLviv = new ArrayList<>();
        entriesLviv.add(new Entry(0, -2f));
        entriesLviv.add(new Entry(1, -4f));
        entriesLviv.add(new Entry(2, 3f));
        entriesLviv.add(new Entry(3, 1f));

        LineDataSet dataSetKyiv = new LineDataSet(entriesKyiv,
"Київ");
        dataSetKyiv.setColor(Color.BLUE);
        dataSetKyiv.setValueTextSize(20f);

        LineDataSet dataSetLviv = new LineDataSet(entriesLviv,
"Львів");
        dataSetLviv.setColor(Color.GREEN);
        dataSetLviv.setValueTextSize(20f);

        // Додавання наборів даних для побудови графіка
        LineData lineData = new LineData(dataSetKyiv, dataSetLviv);
        lineChart.setData(lineData);

        // для перемальовування
        lineChart.invalidate();
    }
}
```

Результат роботи програми в емуляторі віртуального пристрою Medium Phone API 36.1 показано на рис. 22.

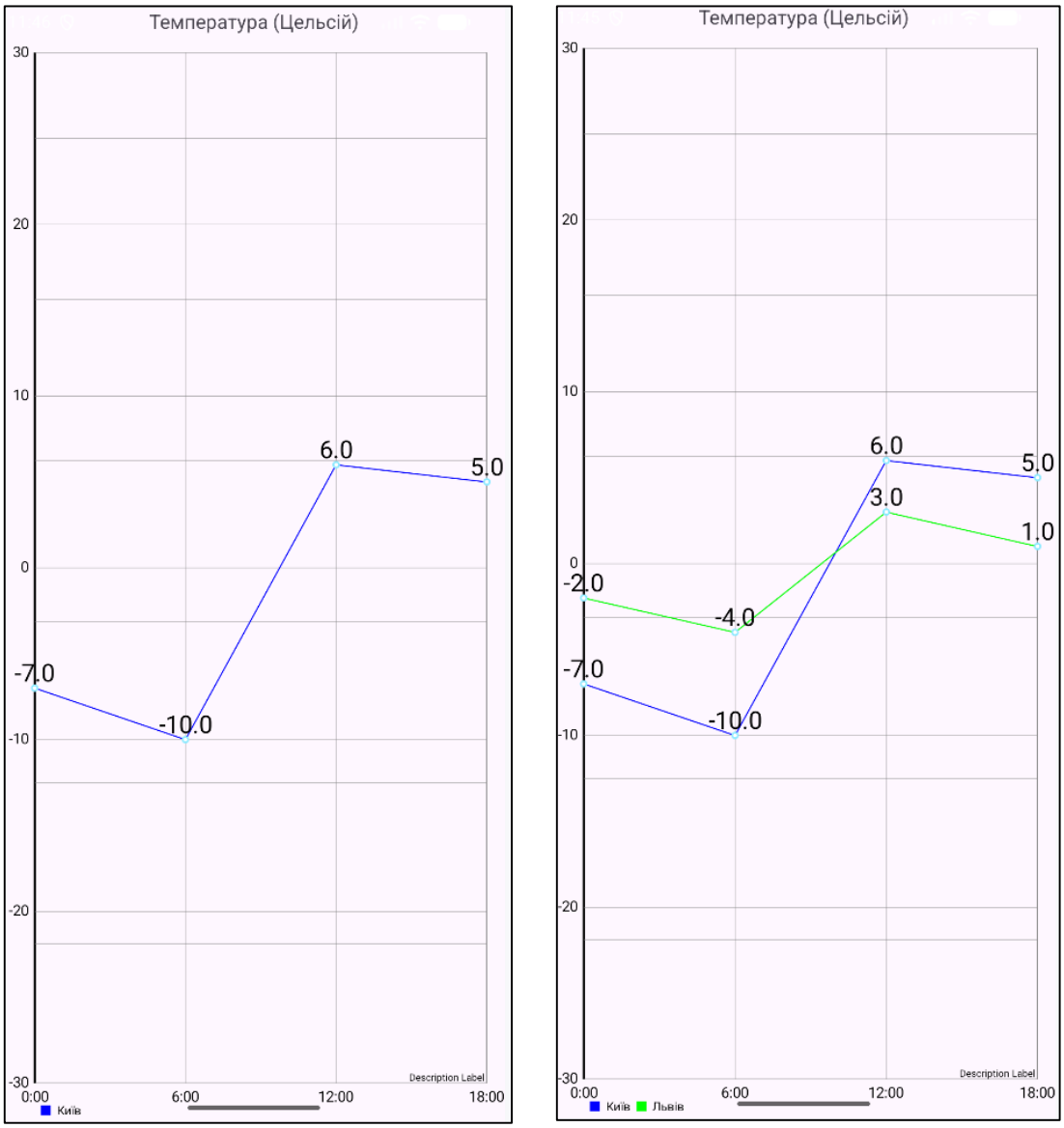


Рисунок 22 – Результат роботи програми відображення лінійного графіка температур

Інформація про класи та їх вміст інших типів діаграм міститься у відповідній API-документації [9].

6. Визначення геопросторових координат

Мета: опанувати механізми визначення геопросторових координат мобільного пристрою та їх використання у застосунках для Android.

Теоретичні відомості

Геопросторові координати – це числові значення, які визначають положення об’єкта на поверхні Землі. Найчастіше використовуються:

- широта (latitude) – положення відносно екватора;
- довгота (longitude) – положення відносно нульового меридіана (Гринвича).

Координати задаються у градусах і дозволяють однозначно визначити місцезнаходження пристрою. У мобільних пристроях координати можуть визначатися за допомогою:

- GPS (супутникова навігація);
- мобільних мереж;
- Wi-Fi мереж;
- гібридних методів.

В сучасних пристроях система автоматично обирає джерело залежно від доступності сигналу, рівня точности та енергоспоживання [13].

В операційній системі Android для роботи з геолокацією використовуються відповідні API. Основними компонентами є:

- ***LocationManager*** – базовий системний сервіс для отримання координат;
- ***Fused Location Provider*** – рекомендований підхід, що об’єднує дані з різних джерел для підвищення точности та оптимізації споживання енергії (реалізований через Google Play Services).

Сучасні застосунки зазвичай використовують *Fused Location Provider* через його ефективність та зручність налаштування.

Для отримання координат застосунок повинен отримати дозвіл користувача. В Android передбачено два основні типи дозволів:

- *ACCESS_COARSE_LOCATION* – приблизне місцезнаходження;
- *ACCESS_FINE_LOCATION* – точне місцезнаходження.

Починаючи з Android 6.0, дозволи запитуються під час виконання програми (runtime permissions).

Як правило, на точність визначення координат впливає наявність відкритого доступу до неба (для GPS), щільність мереж Wi-Fi, рівень сигналу стільникової мережі, налаштування енергозбереження пристрою.

Висока точність супроводжується більшим енергоспоживанням, тому при розробці застосунків важливо враховувати баланс між точністю та ресурсами пристрою.

Приклад

Створимо застосунок, який визначатиме поточні координати пристрою та відображатиме їх у вигляді широти та довготи. Спершу необхідно відкрити середовище **Android Studio** та створити новий проєкт типу **Empty Activity**. Далі обраємо мінімальну версію SDK – не нижче API 23 (для Android 6.0), оскільки з цієї версії реалізовано механізм **runtime permissions**.

Для отримання координат необхідно додати дозволи доступу до місцезнаходження. Для цього їх потрібно розмістити перед тегом <*application*> у **AndroidManifest.xml** файлі:

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

Наступним кроком є створення інтерфейсу користувача. Для цього у файлі **activity_main.xml** створимо простий інтерфейс, який складається з двох об'єктів **TextView** для відображення широти і довготи та об'єкта **Button** для запуску отримання координат.

Встановлюємо лінійне розміщення (**LinearLayout**) і додаємо два об'єкти **TextView** для довготи та широти відповідно, а також об'єкт **Button** для тригеру отримання координат:

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:orientation="vertical"
        android:padding="16dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/tvLatitude"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Latitude: -" />

        <TextView
            android:id="@+id/tvLongitude"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Longitude: -" />

        <Button
            android:id="@+id/btnGetLocation"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Отримати місцезнаходження" />
    </LinearLayout>

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

Нижче описано файл **MainActivity.java** та логіку роботи програми.

```
package com.example.location_coordinates;

import android.os.Bundle;
import android.Manifest;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationManager;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

public class MainActivity extends AppCompatActivity {

    private static final int LOCATION_PERMISSION_REQUEST = 1;

    private TextView tvLatitude, tvLongitude;
    private Button btnGetLocation;
    private LocationManager locationManager;

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

        tvLatitude = findViewById(R.id.tvLatitude);
        tvLongitude = findViewById(R.id.tvLongitude);
        btnGetLocation = findViewById(R.id.btnGetLocation);

        locationManager = (LocationManager)
getSystemService(LOCATION_SERVICE);

        btnGetLocation.setOnClickListener(v -> getLocation());
    }

    private void getLocation() {

        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this,
                    new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
```

```
                    LOCATION_PERMISSION_REQUEST);
            return;
        }

        Location location =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVI
DER);

        if (location != null) {
            double latitude = location.getLatitude();
            double longitude = location.getLongitude();

            tvLatitude.setText("Latitude: " + latitude);
            tvLongitude.setText("Longitude: " + longitude);
        } else {
            Toast.makeText(this,
                    "Не вдалося отримати місцезнаходження",
                    Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String[]
permissions,
                                           @NonNull int[]
grantResults) {
        super.onRequestPermissionsResult(requestCode,
permissions, grantResults);

        if (requestCode == LOCATION_PERMISSION_REQUEST) {
            if (grantResults.length > 0 &&
                    grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                getLocation();
            } else {
                Toast.makeText(this,
                        "Дозвіл не надано",
                        Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

Програма запускається в емуляторі віртуального пристрою Medium Phone API 36.1 (рис. 23). Після натискання кнопки перевіряється наявність дозволу на доступ до точного місцезнаходження. Якщо дозвіл не надано, то система викликає запит *runtime permission*. Після отримання дозволу

передається останнє відоме місцезнаходження через **LocationManager**. Координати (широта та довгота) відображаються у відповідних елементах інтерфейсу [14].

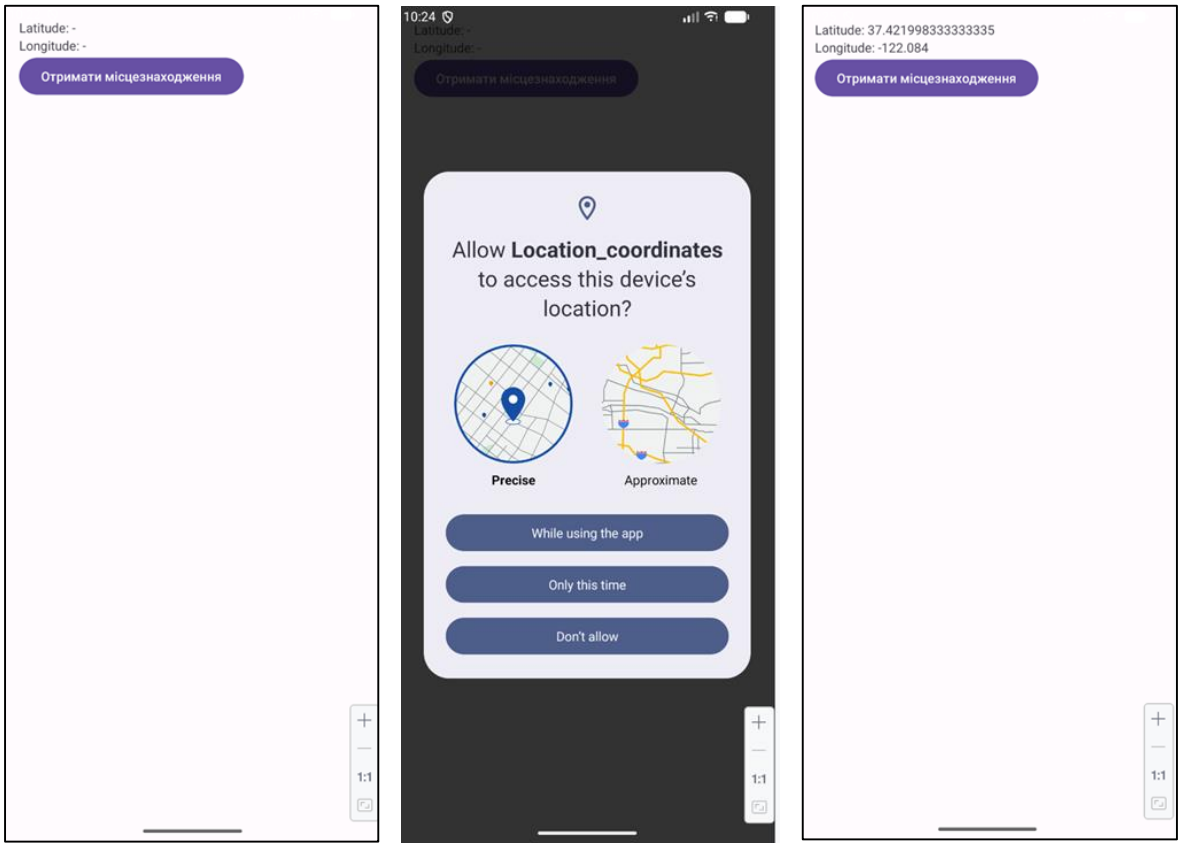


Рисунок 23 – Отримання дозволу на геолокацію та координат

Оскільки програма запускається на емуляторі, то у відповідних полях вказуються координати (37.42, -122.08), які вказують на місцерозташування вказують на штаб-квартиру компанії **Google**, яка розташована в місті Маунтін-В'ю, штат Каліфорнія, США. Для визначення координат власного пристрою рекомендується запускати програму не через емулятор.

7. Геопросторові об’єкти на мапі

Мета: сформувати навички створення, відображення та налаштування геопросторових об’єктів на мапі засобами платформи Android із використанням API Google Maps.

Теоретичні відомості

Геопросторові об’єкти – це абстракції, що використовуються для представлення просторових сутностей на електронній мапі. У мобільних застосунках вони дозволяють візуалізувати окремі точки (маркери), маршрути (полілінії), території (полігони) та кола. Такі об’єкти мають координатну прив’язку та набір візуальних і поведінкових властивостей.

У застосунках платформи Android робота з картами реалізується через Maps SDK for Android. Основними його програмними компонентами є:

SupportMapFragment – фрагмент, який відповідає за відображення карти у Activity.

GoogleMap – основний клас API, через який здійснюється додавання об’єктів на карту, керування виглядом та обробка подій користувача.

LatLng – клас для представлення географічної точки (широта, довгота).

Геопросторові об’єкти можна поділити на кілька видів (табл.4).

Візуальне представлення карти у мобільному застосунку визначається параметрами так званої камери. Камера не є фізичним об’єктом, а логічною моделлю, що описує спосіб відображення картографічної поверхні на екрані пристрою. Керування камерою здійснюється через об’єкт **GoogleMap** засобами Maps SDK for Android.

Таблиця 4 – Основні типи геопросторових об’єктів

Назва	Призначення	Властивості
Marker (маркер)	Використовується для позначення окремої точки на карті. Може реагувати на натискання (подія OnMarkerClickListener)	Position, title, snippet, icon
Polyline (полілінія)	Призначена для відображення маршруту або з’єднання кількох точок.	Список координат, товщина лінії, колір, стиль (суцільна / пунктирна)
Polygon (полігон)	Використовується для відображення замкненої території.	Заливка кольором, прозорість, межі області
Circle (коло)	Використовується для моделювання зони покриття або радіусу доступності.	Коррдинати центральної точки, радіус (у метрах)

Положення та стан камери визначаються такими основними параметрами:

- ***центр (target)*** – географічна точка (**LatLng**), яка відображається в центрі екрану;
- ***масштаб (zoom)*** – рівень деталізації карти: чим більше значення, тим ближче масштабування;
- ***нахил (tilt)*** – кут огляду відносно вертикальної осі (створює ефект псевдо-3D);
- ***азимут (bearing)*** – напрямок орієнтації карти відносно півночі.

Для зміни положення камери використовується клас **CameraUpdateFactory**, який створює об’єкти типу **CameraUpdate**. Далі цей об’єкт передається методам ***moveCamera()*** або ***animateCamera()*** класу **GoogleMap**.

Метод ***moveCamera()*** миттєво змінює положення карти, тоді як ***animateCamera()*** виконує плавну анімацію переходу. Анімація покращує користувацький досвід, оскільки дозволяє візуально простежити зміну позиції або масштабу.

```
LatLng kyiv = new LatLng(50.4501, 30.5234);

mMap.moveCamera(
        CameraUpdateFactory.newLatLngZoom(kyiv, 12)
);

mMap.animateCamera(
        CameraUpdateFactory.newLatLngZoom(kyiv, 15),
        3000,
        null
);
```

Приклад

Розробимо застосунок, за допомогою якого можна додавати маркери та будувати полігон на Google Maps. Передусім створюємо новий проєкт типу Empty Activity. Далі обираємо мінімальну версію SDK API 23. Крім того, підключаємо **Maps SDK for Android** через **Google Cloud Console**. Детальніше це описано у [15].

Потрібно розмістити необхідні дозволи перед тегом <*application*> у **AndroidManifest.xml** файлі:

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

В цьому ж файлі в межах тегу <*application*> додати блок зі згенерованим API-ключем у **Google Cloud Console**.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY"/>
```

За необхідності додати наступний тег <*uses-library*> всередині блоку <*application*>, оскільки, починаючи з Android 9.0 (API level 28), Google

видалила підтримку клієнта **Apache HTTP** зі стандартного набору доступних бібліотек (bootclasspath).

```
<uses-library
    android:name="org.apache.http.legacy"
    android:required="false" />
```

У файл **build.gradle (Module)** додати в блок *dependencies*:

```
implementation("com.google.android.gms:play-services-maps:18.2.0")
```

Далі створимо користувацький інтерфейс, який складається з безпосередньо самої мапи та кнопки «Очистити», яка прибиратиме обрані точки та згенерований полігон.

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

    <fragment
        android:id="@+id/map"

android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

    <Button
        android:id="@+id/btnClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Очистити"
        android:layout_margin="16dp"
        android:layout_gravity="top|end"/>

</FrameLayout>
```

Подаємо нижче файл **MainActivity.java** з кодом реалізації побудови полігонів.

```
package com.example.location_coordinates;

import android.os.Bundle;
import android.widget.Button;
```

```
import androidx.fragment.app.FragmentActivity;
import com.google.android.gms.maps.*;
import com.google.android.gms.maps.model.*;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends FragmentActivity
        implements OnMapReadyCallback {

    private GoogleMap mMap;
    private List<LatLng> points = new ArrayList<>();
    private Polygon polygon;
    private Button btnClear;

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

        btnClear = findViewById(R.id.btnClear);

        SupportMapFragment mapFragment =
                (SupportMapFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.map);

        if (mapFragment != null) {
            mapFragment.getMapAsync(this);
        }

        btnClear.setOnClickListener(v -> clearMap());
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        // Початкове положення камери
        LatLng startPoint = new LatLng(50.4501, 30.5234);

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(startPoint, 12));

        // Обробка натискання на карту
        mMap.setOnMapClickListener(latLng -> addPoint(latLng));
    }

    private void addPoint(LatLng latLng) {

        // Додаємо маркер
        mMap.addMarker(new MarkerOptions()
                .position(latLng)
                .title("Точка " + (points.size() + 1)));

        // Додаємо координату до списку
        points.add(latLng);

        // Якщо точок ≥ 3 – будуємо полігон
```

```
    if (points.size() >= 3) {
            drawPolygon();
        }
    }

    private void drawPolygon() {

        if (polygon != null) {
            polygon.remove();
        }

        PolygonOptions polygonOptions = new PolygonOptions()
                .addAll(points)
                .strokeWidth(5)
                .fillColor(0x3300FF00)  // напівпрозорий зелений
                .strokeColor(0xFF008000);

        polygon = mMap.addPolygon(polygonOptions);
    }

    private void clearMap() {

        mMap.clear();
        points.clear();
        polygon = null;
    }
}
```

Спочатку створюється об’єкт **SupportMapFragment**, який відповідає за відображення карти в інтерфейсі. Після завантаження фрагмента програми отримується доступ до об’єкта **GoogleMap**. Карта спочатку встановлюється у визначеній точці (центр міста Києва) та масштабі. Це робиться для того, щоб користувач бачив початкову область карти.

При кожному натисканні на мапу на місці натискання з’являється візуальний маркер (Marker) для позначення точки, після чого координата точки (LatLng) додається до списку **List<LatLng>**.

Коли кількість точок у списку стає більшою або дорівнює трьом, програма автоматично будує полігон, що з’єднує всі додані точки. Це дає змогу бачити область, обмежену користувачем.

При натисканні на кнопку "Очистити" з карти видаляються всі маркери та полігони, список координат List<LatLng> очищується, що дозволяє почати додавання нових точок з нуля.

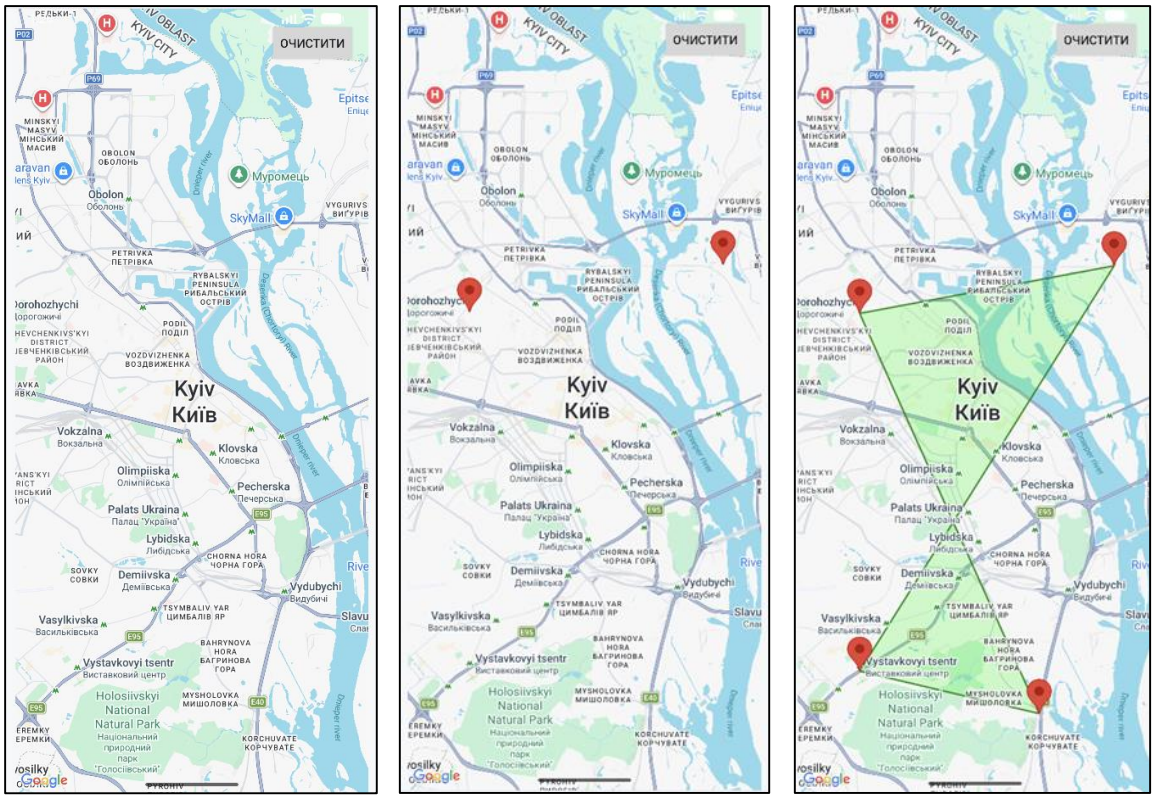


Рисунок 24 – Результат роботи програми для побудови полігону за обраними точками на мапі

Програма забезпечує динамічну взаємодію користувача з картою та демонструє основні операції з маркерами та полігонами.

Приклад використання OpenStreetMap

Google Maps є найвідомішим із геопросторових сервісом, але не єдиним. Наступний приклад демонструє використання OpenStreetMaps [16] для відображення маркера з конкретними координатами на мапі. Створимо проєкт з назвою OSMmap. У файлі ***activity_main.xml*** додамо блок

```
<org.osmdroid.views.MapView
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

У файлі ***build.gradle.kts*** (:app) додамо залежність

```
implementation("org.osmdroid:osmdroid-
android:6.1.18")
```

У файлі ***AndroidManifest.xml*** – дозволи

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

Тоді програма з наступним кодом виведе мапу з інтерактивним маркером (рис. 25).

```
package com.example.maposm;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import org.osmdroid.tileprovider.tilesource.TileSourceFactory;
import org.osmdroid.util.GeoPoint;
import org.osmdroid.views.MapView;
import org.osmdroid.views.overlay.Marker;

public class MainActivity extends AppCompatActivity {
    private MapView map;

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

        map = findViewById(R.id.map);
        map.setTileSource(TileSourceFactory.MAPNIK);

        // можна масштабувати пальцями
        map.setMultiTouchControls(true);

       // стартовий масштаб
        map.getController().setZoom(12.0);

        // координати Майдану Незалежності у Києві
        map.getController().setCenter(new GeoPoint(50.3828671,
30.4701007));
```

```
    Marker marker = new Marker(map);

    // координати ф-ту комп'ютерних наук та кібернетики КНУ
    GeoPoint startPoint = new GeoPoint(50.3828671, 30.5246284);
    marker.setPosition(startPoint);
    marker.setAnchor(Marker.ANCHOR_CENTER,
Marker.ANCHOR_BOTTOM);
    marker.setTitle("Факультет комп'ютерних наук та кібернетики
КНУ");

    map.getOverlays().add(marker);

    // для перемальовування мапи
    map.invalidate();
  }
}
```

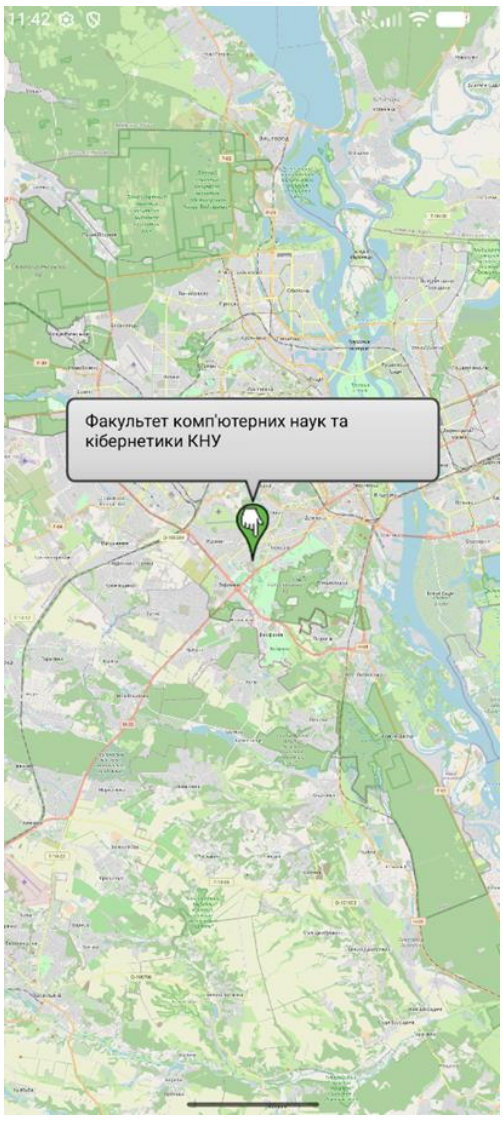


Рисунок 25 – Виведення маркера на мапі OpenStreetMap

8. Геокодування, побудова маршруту

Мета: вивчення принципів геокодування та зворотного геокодування, а також реалізація побудови маршруту між заданими точками на карті з використанням сервісів Google Maps в середовищі Android Studio.

Теоретичні відомості

Геокодування (Geocoding) – це процес перетворення текстової адреси (наприклад, "Київ, Хрещатик, 1") у географічні координати (широту та довготу). ***Зворотне геокодування (Reverse Geocoding)*** – це процес отримання текстової адреси за відомими координатами.

У мобільних застосунках Android геокодування може виконуватись через локальний клас **Geocoder (Android SDK)** або через веб-сервіс **Geocoding API від Google** [17-18]. Для геокодування у середовищі Android Studio використовується наступний клас:

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List <Address> addresses =
geocoder.getFromLocationName("Kyiv", 1);
```

Об’єкт **Address** містить широту (getLatitude()), довготу (getLongitude()), країну, місто, поштовий індекс, повну адресу.

Побудова маршруту передбачає визначення оптимального шляху між двома або більше точками з урахуванням дорожньої мережі, типу транспорту, відстані, часу в дорозі. Для цього використовується **Directions API**, який є складовою Google Maps Platform від **Google Cloud Console** [19].

Запит має наступний вигляд, а відповідь повертається у форматі JSON та містить список маршрутів (routes), відстань (distance), тривалість (duration), polyline (закодований маршрут).

```
https://maps.googleapis.com/maps/api/directions/json?
origin=LAT1,LNG1
&destination=LAT2,LNG2
&key=API_KEY
```

Для відображення маршруту використовується клас **Polyline** з SDK Google Maps. Основними його компонентами є **GoogleMap**, **PolylineOptions**, **LatLng**. Polyline будується на основі декодованого encoded polyline, який повертає **Directions API**.

```
PolylineOptions polylineOptions = new PolylineOptions()
        .addAll(decodedPoints)
        .width(8)
        .color(Color.BLUE);

mMap.addPolyline(polylineOptions);
```

Приклад

Реалізуємо проєкт, в якому користувач водить адресу відправлення, вводить адресу призначення, а потім натискає кнопку «Побудувати маршрут». Після цього на карті становлюються маркери точок вказаних адрес будується маршрут у вигляді Polyline. Спочатку підключимо залежності у файлі **build.gradle (Module)**:

```
implementation("com.google.android.gms:play-services-
maps:18.2.0")
implementation("com.google.android.gms:play-services-
location:21.0.1")
```

Далі розміщуємо необхідні дозволи перед тегом <*application*> у **AndroidManifest.xml** файлі:

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

В цьому ж файлі в межах тегу <*application*> додати блок зі згенерованим API-ключем у **Google Cloud Console**.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY"/>
```

Далі створимо інтерфейс, який складається з безпосередньо самої мапи, двох текстових полів та кнопки «Побудувати маршрут», за допомогою якої будується маршрут між двома адресами.

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

    <EditText
        android:id="@+id/etOrigin"
        android:hint="Початкова адреса"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <EditText
        android:id="@+id/etDestination"
        android:hint="Кінцева адреса"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <Button
        android:id="@+id/btnRoute"
        android:text="Побудувати маршрут"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <fragment
        android:id="@+id/map"

android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"/>
</LinearLayout>
```

У файлі **MainActivity.java** реалізовано логіку взаємодії користувача з картою та сервісами маршрутизації у Додатку 3.

Після запуску застосунку (рис. 26) ініціалізується об’єкт GoogleMap і встановлюється початкове положення камери.

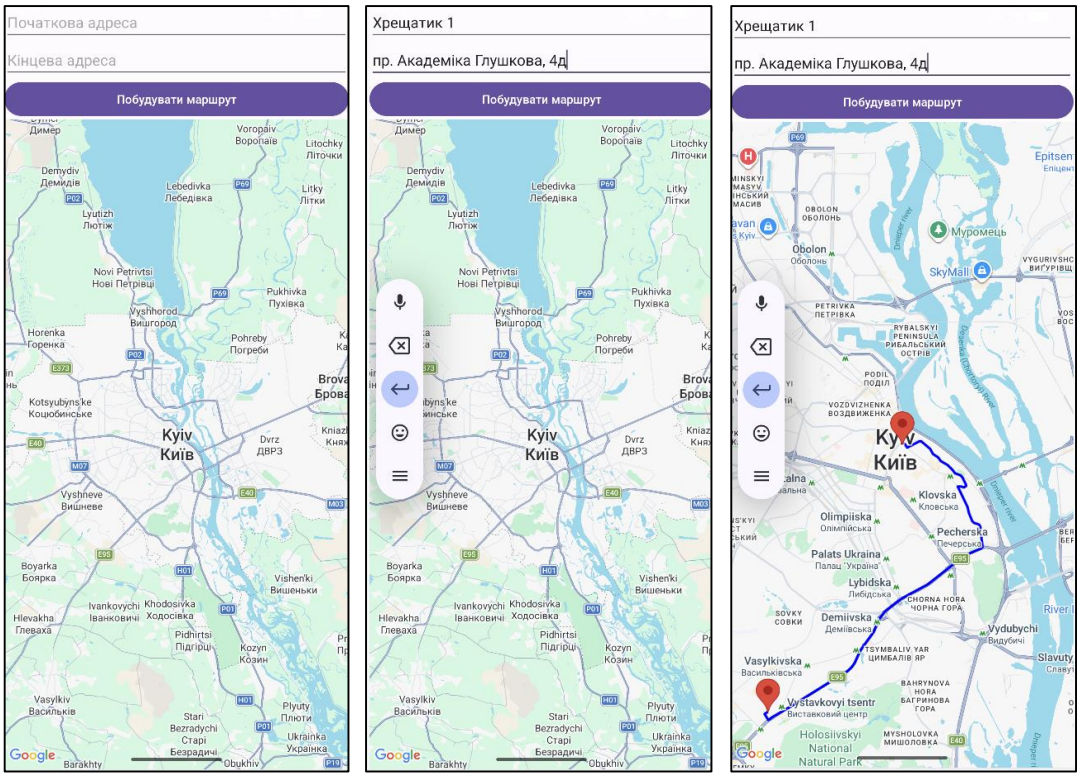


Рисунок 26 – Результат роботи програми для побудови маршруту між двома адресами

Користувач вводить початкову та кінцеву адреси і натискає кнопку побудови маршруту. Далі застосунок виконує *геокодування введених адрес*, який перетворює текстові значення у географічні координати (широту та довготу). Потім формується HTTP-запит до сервісу **Directions API** платформи Google Maps Platform, з якого отримується JSON-відповідь із параметрами маршруту.

Виділяється закодована лінія маршруту (polyline), яку потім декодують у список координат. Після чого будується маршрут на карті у вигляді полілайна та додаються маркери початкової та кінцевої точки.

Бізнес-логіка застосунку забезпечує повний цикл: введення адрес → геокодування → отримання маршруту → візуалізація на карті.

9. Поняття кросплатформної розробки

Кросплатформна розробка – це підхід до створення програмного забезпечення, який дозволяє використовувати єдину кодову базу для кількох операційних систем (Android, iOS, Web, Desktop). Основна ідея полягає у зменшенні витрат на розробку та підтримку, прискоренні випуску продукту та забезпеченні єдиного користувацького досвіду на різних платформах. На відміну від нативної розробки, кросплатформні фреймворки дозволяють писати бізнес-логіку один раз і компілювати або інтерпретувати її для різних середовищ виконання. ***Flutter*** – це UI-фреймворк від Google для створення мобільних, веб- та десктоп-застосунків з єдиної кодової бази. Використовує мову Dart та власний механізм рендерингу інтерфейсу [20].

React Native – фреймворк від Meta для розробки мобільних застосунків з використанням JavaScript та бібліотеки React. Дозволяє створювати інтерфейси з використанням нативних компонентів [21].

.NET MAUI – сучасний кросплатформний фреймворк від Microsoft для створення мобільних і десктоп-застосунків на C# [22].

Переваги кросплатформної розробки**:** єдина кодова база для декількох платформ, скорочення витрат часу та ресурсів, спрощення підтримки та оновлень, швидший вихід продукту на ринок

Недоліки: можливі обмеження у доступі до нативних API, потенційні проблеми з продуктивністю, залежність від стороннього фреймворку.

Кросплатформна розробка є ефективним підходом, коли важлива швидкість розробки та підтримка декількох платформ. Вибір конкретного інструменту залежить від вимог до продуктивності, функціональності та компетенцій команди розробників.

Практичні завдання

Оберіть варіант мобільного проєкта з поданого нижче переліку або запропонуйте свою тему. Лабораторні роботи є етапами виконання проєкту, їх оцінювання відбувається відповідно до чек-листа кожного реалізованого функціоналу. Після виконання Етапу 1 студент/ка може самостійно визначати послідовність виконання інших етапів.

За результатами виконання етапів підготуйте і надішліть ілюстрований звіт. У звіті про виконання Етапу 1 надайте посилання на github/gitlab-акаунт вашого проєкту, зазначте версію API Android та назву віртуального пристрою, на якому тестували роботу програми.

Етап 1. Формування вимог до мобільного програмного проєкту.

Перелік виконаних робіт (завдань):

- Короткий опис концепції застосунку, який розробляється;
- Опис типів користувачів застосунку;
- Опис вимог користувача та функціональних вимог;
- Діаграма прецедентів, станів та активностей;
- Опис етапів виконання проєкту з розбиттям на принти протягом семестру.

Етап 2. Реалізація введення та виведення ланих.

Реалізовані елементи UI у застосунку:

- Введення даних у поле (текстове, числове, дата тощо);
- Фільтрування виведених даних за допомогою вибору категорії зі списку, перемикача вибору одного елемента (radio button) або багатьох елементів (check box);
- Мінімальна валідація коректності введення даних (поле не пусте, формат правилний для дати, числа, email тощо).
- Реалізовано можливість редагування записів;

- Реалізовано можливість видалення записів;
- Кнопка підтвердження введення/оновлення/видалення записів.

Етап 3. Взаємодія з файлом:

- Реалізовано запис у файл;
- Реалізовано читання з файлу;
- Реалізовано можливість перегляду файлу.

Етап 4. Взаємодія з БД.

- Реалізовано запис у БД;
- Реалізовано читання з БД;
- Реалізовано видалення запису БД;
- Реалізовано редагування запису БД.

Етап 5. Побудова графіку/діаграми.

Реалізовано побудову графіків/діаграм мінімум 3 різних типів.

Етап 6. Геопросторовий модуль.

- Реалізовано відображення поточної локації як координат;
- Реалізовано відображення поточної локації на мапі;
- Реалізовано побудову маршруту до іншої локації.

Бонусне завдання. Виконання етапів 2-6 проєкту за допомогою фреймворків кросплатформної розробки.

Варіанти проєктів:

1. Персональний планувальник
2. Помічник тренувань
3. Персональний фінансовий помічник (персональна бухгалтерія)
4. Помічник з правильного харчування
5. Помічник з догляду за домашньою тваринкою
6. Застосунок управління креативними ідеями
7. Застосунок аналізу емоційного стану
8. Мобільний агрегатор новин

9. Мобільний застосунок "Книгарні Києва" (або іншого міста/регіону)

10. Мобільний застосунок "Фітнес-центри Києва" (або іншого міста/регіону)

11. Мобільний застосунок "Аптеки Києва" (або іншого міста/регіону)

12. Мобільний застосунок "Музеї Києва" (або іншого міста/регіону)

13. Мобільний застосунок "Театри Києва" (або іншого міста/регіону)

14. Мобільний застосунок "[Етно]фествалі Києва" (або іншого міста/регіону)

15. Мобільний застосунок "Ветеринарні клініки Києва" (або іншого міста/регіону)

16. Мобільний застосунок "Пункти прокату велосипедів/е-самокатів Києва" (або іншого міста/регіону)

17. Мобільний застосунок "IT-компанії Києва" (або іншого міста/регіону)

18. Мобільний застосунок "Університети і коледжі Києва" (або іншого міста/регіону)

19. Мобільний застосунок "Наукові фахові журнали України"

20. Мобільний застосунок "Студентські наукові конференції України"

21. Застосунок для планування подорожей із використанням картографічних сервісів та геолокації

22. Система рекомендацій контенту на основі вподобань користувача

23. Застосунок для фітнес-трекінгу та аналізу фізичної активності користувача

24. Мобільний погодний застосунок з інтеграцією геолокаційних API та прогнозуванням погодних умов

25. Застосунок для пошуку закладів поблизу з використанням геолокації

26. Мобільний застосунок для відстеження та візуалізації переміщення користувача

27. Застосунок для аналізу та візуалізації активности користувача на мобільному пристрої

28. Мобільний застосунок для планування режиму сну

29. Інтерактивний посібник з теми "Основи вебпрограмування"

30. Мобільний застосунок для контролю догляду за домашніми тваринами

31. Розробка мобільного додатку для інтерактивних туристичних квестів

32. Мобільний помічник для прослуховування аудіокнижок та медитацій

33. Інформаційна система в галузі туризму

34. Інтерактивний навчальний посібник з теми "..."

35. Інша тема, узгоджена з викладачем

Список використаних джерел

1. Digital 2025: The essential guide to the global state of digitalю URL: https://wearesocial.com/uk/blog/2025/02/digital-2025-the-essential-guide-to-the-global-state-of-digital/
2. Android Studio IDE. URL https://developer.android.com/studio
3. Фреймворк Flutter. URL: https://flutter.dev/
4. Фреймворк React Native. URL: https://reactnative.dev/
5. Мова програмування Java. URL: https://www.java.com/
6. Мова програмування Kotlin. URL: https://kotlinlang.org/
7. Документація для розробників на Android. URL https://developer.android.com/reference
8. Бібліотека побудови діаграм і графікав MPAndroidChart. URL: https://github.com/PhilJay/MPAndroidChart
9. API-документація бібліотеки MPAndroidChart. URL: https://javadoc.jitpack.io/com/github/PhilJay/MPAndroidChart/v3.1.0/javadoc/
10. Репозиторій jitpack. URL: https://jitpack.io/
11. Бібліотека побудови діаграм і графікав GraphView. URL: https://github.com/jjoe64/GraphView
12. Бібліотека побудови діаграм і графікав AnyChart. URL: https://www.anychart.com/technical-integrations/samples/android-charts/
13. Android Developers. Location and Maps Overview. URL: https://developer.android.com/develop/sensors-and-location/location
14. Android Developers. Get the last known location. URL: https://developer.android.com/develop/sensors-and-location/location/retrieve-current
15. Maps SDK for Android Quickstart. URL: https://developers.google.com/maps/documentation/android-sdk/start
16. OenStreetMap for developers https://www.openstreetmap.ie/resources-2/openstreetmap-for-developers/
17. Google Maps Platform documantation. URL: https://developers.google.com/maps/documentation/geocoding
18. Android Geocoder (Android Developers). URL: https://developer.android.com/reference/android/location/Geocoder
19. Google Maps Platform – Directions API. URL: https://developers.google.com/maps/documentation/directions
20. Flutter documentation. URL: https://docs.flutter.dev/
21. Get Started with React Native. URL: https://reactnative.dev/docs/environment-setup
22. .NET Multi-platform App UI documentation. URL: https://learn.microsoft.com/en-gb/dotnet/maui/?view=net-maui-10.0

Додатки

Додаток 1. Файл MainActivity.kt з кодом першої програми

```
package com.example.myfirstapplication

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.example.myfirstapplication.ui.theme.MyFirstApplicationTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyFirstApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Greeting(
                        name = "Android",
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    MyFirstApplicationTheme {
        Greeting("Android")
    }
}
```

Додаток 2. Файл activity_main.xml для програми розв'язування квадратного рівняння

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView_title"
        android:layout_width="412dp"
        android:layout_height="46dp"
        android:background="#3F51B5"
        android:text="Квадратне рівняння"
        android:textAlignment="center"
        android:textColor="#FFFFFF"
        android:textSize="34sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.05" />

    <TextView
        android:id="@+id/textView_goenter"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Введіть a!=0, b, c"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView_title"
        app:layout_constraintVertical_bias="0.042" />

    <TextView
        android:id="@+id/textView_a"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="a="
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
    app:layout_constraintHorizontal_bias="0.238"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView_goenter"
    app:layout_constraintVertical_bias="0.048" />

<TextView
    android:id="@+id/textView_b"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="28dp"
    android:text="b="
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.239"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView_a" />

<TextView
    android:id="@+id/textView_c"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="c="
    android:textSize="24sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.241"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView_b"
    app:layout_constraintVertical_bias="0.0" />

<TextView
    android:id="@+id/textView_result"
    android:layout_width="305dp"
    android:layout_height="87dp"
    android:layout_marginBottom="204dp"
    android:textAlignment="center"
    android:textSize="24sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button"
    app:layout_constraintVertical_bias="0.5" />

<EditText
    android:id="@+id/editTextNumber_a"
    android:layout_width="184dp"
    android:layout_height="45dp"
    android:layout_marginTop="20dp"
```

```
        android:ems="10"
        android:inputType="numberDecimal|numberSigned"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.114"
        app:layout_constraintStart_toEndOf="@+id/textView_a"
        app:layout_constraintTop_toBottomOf="@+id/textView_goenter" />

    <EditText
        android:id="@+id/editTextNumber_b"
        android:layout_width="185dp"
        android:layout_height="47dp"
        android:layout_marginTop="12dp"
        android:ems="10"
        android:inputType="number|numberDecimal|numberSigned"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.102"
        app:layout_constraintStart_toEndOf="@+id/textView_b"
        app:layout_constraintTop_toBottomOf="@+id/editTextNumber_a" />

    <EditText
        android:id="@+id/editTextNumber_c"
        android:layout_width="184dp"
        android:layout_height="49dp"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:inputType="number|numberDecimal|numberSigned"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.1"
        app:layout_constraintStart_toEndOf="@+id/textView_c"
        app:layout_constraintTop_toBottomOf="@+id/editTextNumber_b" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickButton"
        android:text="Розв'язати"
        android:textSize="24sp"
        app:layout_constraintBottom_toTopOf="@+id/textView_result"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editTextNumber_c"
        app:layout_constraintVertical_bias="0.5" />

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

Додаток 3. Файл MainActivity.java з кодом програми для побудови маршрутів

```
package com.example.location_coordinates;

import android.graphics.Color;
import android.location.Address;
import android.location.Geocoder;
import android.os.AsyncTask;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.PolylineOptions;

import org.json.JSONArray;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class MainActivity extends AppCompatActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;
    private EditText etOrigin, etDestination;
    private Button btnRoute;

    private static final String API_KEY = "Тут_має_бути_ваш_ключ";

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

        etOrigin = findViewById(R.id.etOrigin);
        etDestination = findViewById(R.id.etDestination);
        btnRoute = findViewById(R.id.btnRoute);

        SupportMapFragment mapFragment =
                (SupportMapFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.map);

        if (mapFragment != null) {
            mapFragment.getMapAsync(this);
```

```
        }

        btnRoute.setOnClickListener(v -> buildRoute());
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Початкове положення камери (Київ)
        LatLng kyiv = new LatLng(50.4501, 30.5234);
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(kyiv, 10));
    }

    private void buildRoute() {

        String originStr = etOrigin.getText().toString();
        String destStr = etDestination.getText().toString();

        LatLng origin = getLocationFromAddress(originStr);
        LatLng dest = getLocationFromAddress(destStr);

        if (origin == null || dest == null) return;

        mMap.clear();

        mMap.addMarker(new
MarkerOptions().position(origin).title("Початок"));
        mMap.addMarker(new MarkerOptions().position(dest).title("Кінець"));

        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(origin, 12));

        String url = getDirectionsUrl(origin, dest);
        new FetchUrl().execute(url);
    }

    private LatLng getLocationFromAddress(String address) {
        try {
            Geocoder geocoder = new Geocoder(this, Locale.getDefault());
            List<Address> addresses = geocoder.getFromLocationName(address,
1);

            if (addresses != null && !addresses.isEmpty()) {
                Address location = addresses.get(0);
                return new LatLng(location.getLatitude(),
location.getLongitude());
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    private String getDirectionsUrl(LatLng origin, LatLng dest) {

        String strOrigin = "origin=" + origin.latitude + "," +
origin.longitude;
        String strDest = "destination=" + dest.latitude + "," +
dest.longitude;
        String mode = "mode=driving";
```

```
        String parameters = strOrigin + "&" + strDest + "&" + mode;

        return "https://maps.googleapis.com/maps/api/directions/json?"
                + parameters + "&key=" + API_KEY;
    }

    private class FetchUrl extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... url) {
            try {
                URL urlObj = new URL(url[0]);
                HttpURLConnection connection = (HttpURLConnection)
urlObj.openConnection();
                connection.connect();

                InputStream stream = connection.getInputStream();
                BufferedReader reader = new BufferedReader(new
InputStreamReader(stream));

                StringBuilder buffer = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                    buffer.append(line);
                }

                reader.close();
                stream.close();
                connection.disconnect();

                return buffer.toString();

            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }

        @Override
        protected void onPostExecute(String result) {
            if (result != null) {
                new ParserTask().execute(result);
            }
        }
    }

    private class ParserTask extends AsyncTask<String, Integer,
List<LatLng>> {

        @Override
        protected List<LatLng> doInBackground(String... jsonData) {

            List<LatLng> points = new ArrayList<>();

            try {
                JSONObject jsonObject = new JSONObject(jsonData[0]);
                JSONArray routes = jsonObject.getJSONArray("routes");

                if (routes.length() > 0) {
```

```
                    JSONObject route = routes.getJSONObject(0);
                    JSONObject overviewPolyline =
                            route.getJSONObject("overview_polyline");
                    String encodedString =
                            overviewPolyline.getString("points");

                    points = decodePolyline(encodedString);
                }

            } catch (Exception e) {
                e.printStackTrace();
            }

            return points;
        }

        @Override
        protected void onPostExecute(List<LatLng> result) {

            if (result == null || result.isEmpty()) return;

            PolylineOptions lineOptions = new PolylineOptions();
            lineOptions.addAll(result);
            lineOptions.width(8);
            lineOptions.color(Color.BLUE);

            mMap.addPolyline(lineOptions);
        }
    }

    private List<LatLng> decodePolyline(String encoded) {

        List<LatLng> poly = new ArrayList<>();
        int index = 0, lat = 0, lng = 0;

        while (index < encoded.length()) {

            int b, shift = 0, result = 0;
            do {
                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            lat += ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));

            shift = 0;
            result = 0;
            do {
                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            lng += ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));

            LatLng p = new LatLng((lat / 1E5), (lng / 1E5));
            poly.add(p);
        }

        return poly;
    }
}
```