МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ВОЛОДИМИРА ДАЛЯ

КОНСПЕКТ ЛЕКЦІЙ

навчально-методичного комплексу
дистанційного курсу дисципліни
«Програмування для мобільних платформ»
Частина І
(для студентів спеціальностей
F2 «Інженерія програмного забезпечення»,
F6 «Інформаційні технології та системи»)
(Електронне видання)

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних технологій та програмування
Протокол № 10 від 09.06.2025 р.

КИЇВ 2025

УДК 330.88:338:65.01

Конспект лекцій навчально-методичного комплексу дистанційного курсу дисципліни «Програмування для мобільних платформ» Частина І (для студентів спеціальностей F2 «Інженерія програмного забезпечення», F6 «Інформаційні технології та системи») (Електронне видання) / Уклад.: Ратов Д. В. - Київ: вид-во СНУ ім. В. Даля, 2025.– 280с.

Укладено на основі ОПП підготовки бакалаврів спеціальностей F2 «Інженерія програмного забезпечення», F6 «Інформаційні технології та системи», робочої навчальної програми з дисципліни „Програмування для мобільних платформ”.

Укладач: Д. В. Ратов, доц., к.т.н.

Рецензент О. І. Захожай, доц., д.т.н.

ЗМІСТ

1. ПОЧАТОК РОБОТИ З ANDROID

1.1. Вступ. Установка Android Studio та Android SDK

Бурхливий розвиток інформаційних технологій останнім часом призвів до того, що з'явилося багато нових пристроїв та технологій, таких як планшети, смартфони, нетбуки, інші гаджети. Вони все міцніше входять у наше життя і стають звичною справою. Лідируючої платформою серед подібних гаджетів на сьогоднішній день є ОС Андроїд.

Android використовується на різних пристроях. Це і смартфони, і планшети, і телевізори, і смарт-годинник та низка інших гаджетів. За різними підрахунками за 2020 рік цією операційною системою користуються близько 85% власників смартфонів, а загальна кількість користувачів смартфонів на ОС Android оцінюється більш ніж у 2,5 млрд. людей у всьому світі.

ОС Андроїд була створена розробником Енді Рубіном (Andy Rubin) як операційна система для мобільних телефонів і спочатку розвивалася в рамках компанії Android Inc. Але у 2005 році Google купує Android Inc. і починає розвивати операційну систему із новою силою. Android постійно еволюціонує, і разом із операційною системою еволюціонують засоби та інструменти для розробки. На даний момент (жовтень 2021 року) останньою версією є Android 12.0, яка вийшла у жовтні 2021 року:

Версія	Кодове ім'я	дата випуску	Рівень API
12.0	12	4 жовтня 2021	31
11.0	11	8 вересня 2020	30
10.0	10	3 вересня 2019	29
9.0	Pie	6 серпня 2018	28
8.1	Oreo	5 грудня 2017	27
8.0	Oreo	21 серпня 2017	26
7.1	Nougat	4 жовтня 2016	25

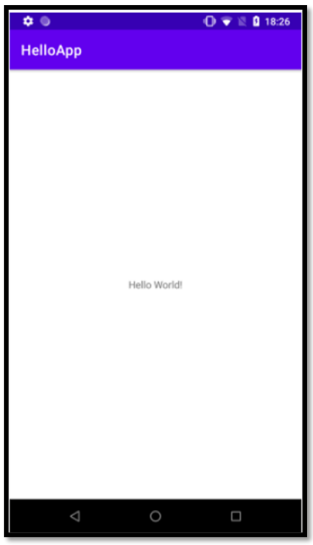
7.0	Nougat	22 серпня 2016	24
6.0	Marshmallow	5 жовтня 2015	23
5.1	Lollipop	9 березня 2015	22
5.0	Lollipop	3 листопада 2014	21
4.4	KitKat	31 жовтня 2013	19
4.3	Jelly Bean	24 липня 2013	18
4.2	Jelly Bean	13 листопада 2012	17
4.1	Jelly Bean	9 липня 2012	16
4.0	Ice Cream Sandwich	16 грудня 2011	15
2.3	Gingerbread	6 грудня 2010	10

1.2. Що потрібне для розробки?

Варто зазначити, що розробляти програми під Android можна за допомогою різних фреймворків та мов програмування. Так, як мови програмування можуть застосовуватися Java, Kotlin, Dart (фреймворк Flutter), C++, Python, C# (платформа Xamarin) тощо. У цьому посібнику ми використовуватимемо саме мову Java, як найпоширеніший і використовуваний. Тому перш ніж приступати до освоєння програмування під Android по даному посібнику, необхідно освоїти хоча б базовий момент мови Java.

1.3. Створення графічного інтерфейсу

Минулої теми ми розглянули створення простої програми, яка пропонує Android Studio за замовчуванням і яка просто виводить на екран рядок Hello Android.



Але чому у нас виводиться саме цей рядок? Чому у нас взагалі створюється такий візуальний інтерфейс?

Виконання програми Android за замовчуванням починається з класу MainActivity, який за замовчуванням відкрито Android Studio:

```
package com.example.helloapp;

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

public class MainActivity extends AppCompatActivity {

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

Кожен окремий екран або сторінка у програмі описується таким поняттям, як activity. У літературі можна використовувати різні терміни: екран, сторінка, активність. В даному випадку я використовуватиму поняття "activity". Так ось, якщо ми запустимо програму на пристрої, то на екрані ми по суті побачимо певну activity, яка представляє цей інтерфейс.

Клас MainActivity по суті є звичайним класом java, на початку якого йде визначення пакета даного класу:

```
package com.example.helloapp;
```

Далі йде імпорт класів з інших пакетів, функціональність яких використовується в MainActivity:

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
```

Потім йде власне визначення класу:

```
publicclass MainActivity extends AppCompatActivity
```

За замовчуванням MainActivity успадковується від AppCompatActivity, який вище підключений за допомогою директиви імпорту. Клас AppCompatActivity насправді представляє окремий екран (сторінку) програми або його візуальний інтерфейс. І MainActivity успадковує весь цей функціонал.

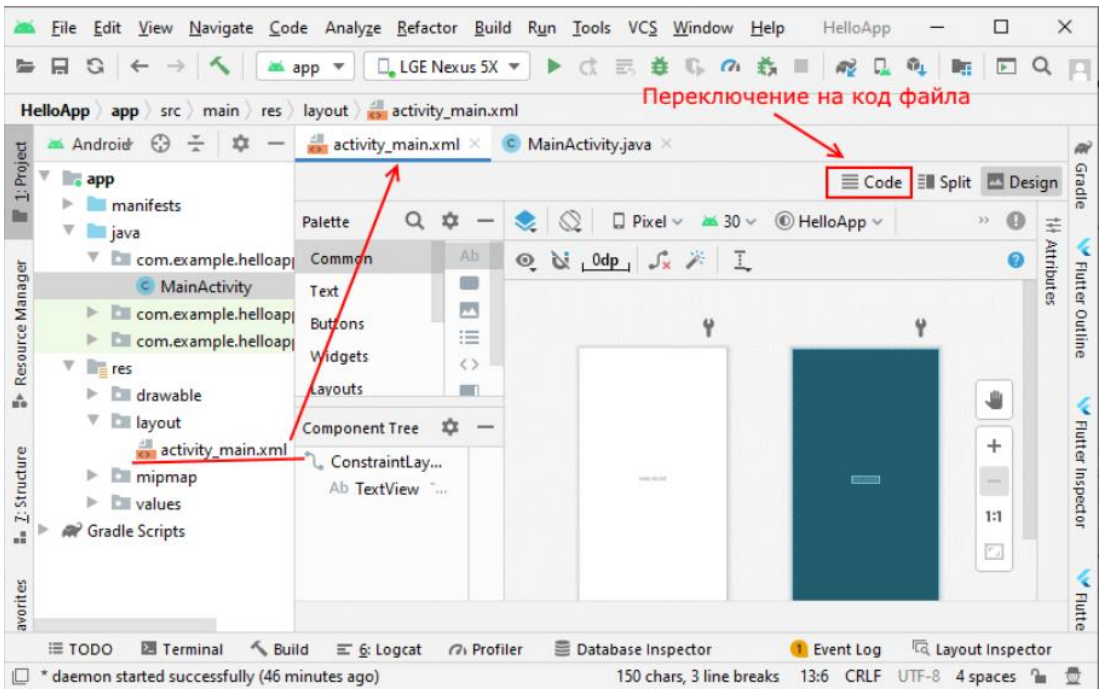
За замовчуванням MainActivity містить лише один метод onCreate(), в якому фактично створюється весь інтерфейс програми:

```
protectedvoid onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
```

До методу setContentView() передається ресурс розмітки графічного інтерфейсу:

```
setContentView(R.layout.activity_main);
```

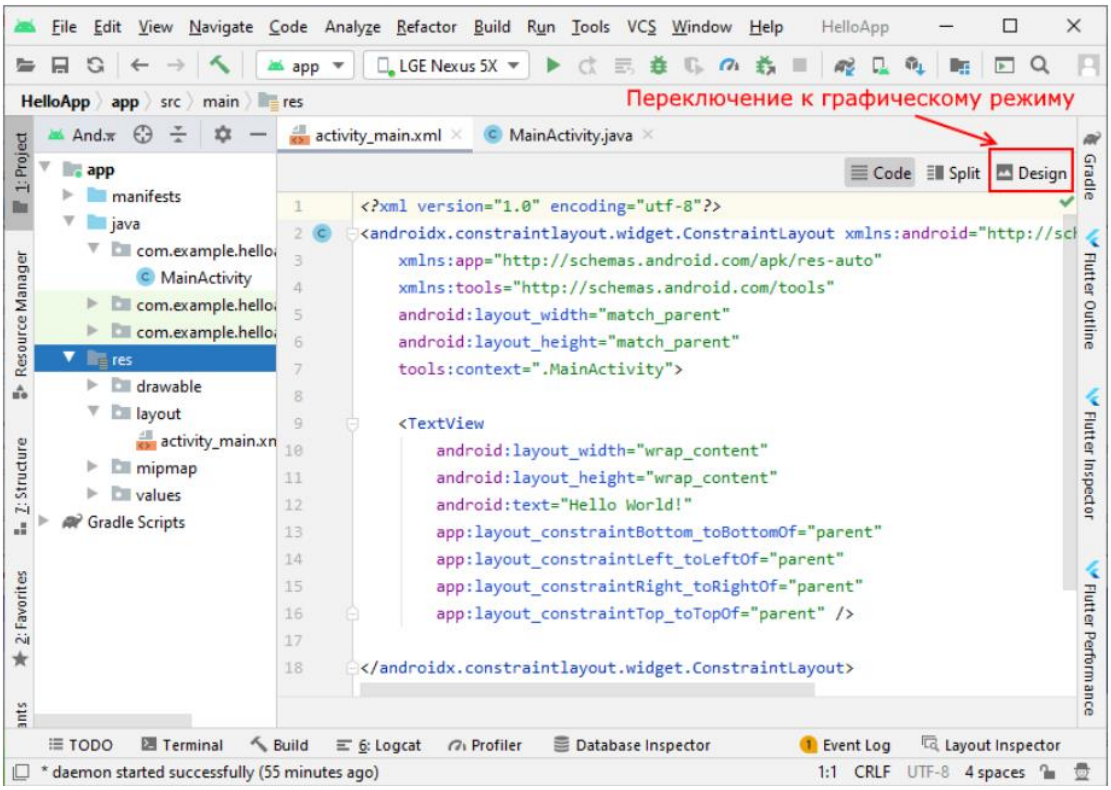
Саме тут і вирішується, який візуальний інтерфейс матиме MainActivity. Але що тут представляє ресурс R.layout.activity_main? Це файл activity_main.xml із папки res/layout (в принципі можна помітити, що назва ресурсу відповідає назві файлу), який також за замовчуванням відкритий в Android Studio:



Файл activity_main.xml

Android Studio дозволяє працювати з візуальним інтерфейсом як у режимі коду, так і у графічному режимі. Так, за замовчуванням файл відкритий у графічному режимі, і ми можемо побачити, як у нас приблизно буде виглядати екран програми. І навіть накидати з панелі інструментів якісь елементи управління, наприклад, кнопки чи текстові поля.

Але також ми можемо працювати з файлом у режимі коду, оскільки activity_main.xml – це звичайний текстовий файл із розміткою xml. Для переключення до коду натисніть кнопку Code над графічним представленням. (Додатково за допомогою кнопки Split можна перейти на комбіноване представлення код + графічний дизайнер)



Тут ми побачимо, що на рівні коду файл activity_main.xml містить таку розмітку:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Весь інтерфейс представлений елементом-контейнером androidx.constraintlayout.widget.ConstraintLayout:

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

ConstraintLayout дозволяє розташувати вкладені елементи у певних місцях екрана. Спочатку елементу ConstraintLayout йде визначення просторів імен XML:

```
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"
```

Кожен простір імен задається так: xmlns:префікс="назва_ресурсу". Наприклад, в

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

Назва ресурсу (або URI - Uniform Resource Indicator) – "http://schemas.android.com/apk/res/android". І цей ресурс зіставляється із префіксом android (xmlns:android).

Навіщо ці простори імен потрібні? Кожен ресурс або URI визначає певну функціональність, яка використовується в програмі, наприклад, надають теги та атрибути, які необхідні для побудови програми.

- `xmlns:android="http://schemas.android.com/apk/res/android"`: містить основні атрибути, що надаються платформою Android, застосовуються в елементах управління та визначають їх візуальні властивості (наприклад, розмір, позиціонування)
- `xmlns:app="http://schemas.android.com/apk/res-auto"`: містить атрибути, визначені в рамках програми
- `xmlns:tools="http://schemas.android.com/tools"`: застосовується для роботи з режимом дизайнера Android Studio

І щоб спростити роботу із цими ресурсами, застосовуються префікси. Наприклад, далі ми бачимо:

```
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">
```

android:layout_widthвизначає ширину контейнера. Цей атрибут (layout_width) розташований у ресурсі "http://schemas.android.com/apk/res/android". І оскільки цей ресурс зіставляється з префіксом android, то для звернення до атрибуту перед ним через двокрапку вказується префікс даного ресурсу.

Значення атрибуту android:layout_weight є "match_parent". Це означає, що елемент (ConstraintLayout) розтягуватиметься по всій ширині контейнера (екрана пристрою).

Атрибут android:layout_height="match_parent" визначає висоту контейнера та також визначений у "http://schemas.android.com/apk/res/android". Значення "match_parent" вказує, що ConstraintLayout буде розтягуватись по всій довжині контейнера (екрана пристрою).

Атрибут tools:context визначає, який клас activity (екрана програми) пов'язаний із поточним визначенням інтерфейсу. В даному випадку це клас MainActivity. Це дозволяє використовувати Android Studio різні можливості в режимі дизайнера, які залежать від класу activity.

1.3.1.1. TextView

Текстове поле встановлює текст, використовуючи атрибут android:text.

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
```

- **android:layout_width**встановлює ширину віджету. Значення wrap_content задає віджету величину, достатню для відображення в контейнері.
- **android:layout_height**встановлює висоту віджету. Значення wrap_content аналогічно до установки ширини задає для віджета висоту, достатню для відображення в контейнері
- **android:text**встановлює текст, який виводитиметься в TextView (в даному випадку це рядок "Hello World!")
- **app:layout_constraintLeft_toLeftOf="parent"**: вказує, що ліва межа елемента вирівнюватиметься з лівої сторони контейнера ConstraintLayout

  Зверніть увагу, що цей атрибут визначений у просторі імен із префіксом app, тобто у "http://schemas.android.com/apk/res-auto".

- **app:layout_constraintTop_toTopOf="parent"**: вказує, що верхня межа елемента вирівнюватиметься по верхній стороні контейнера ConstraintLayout

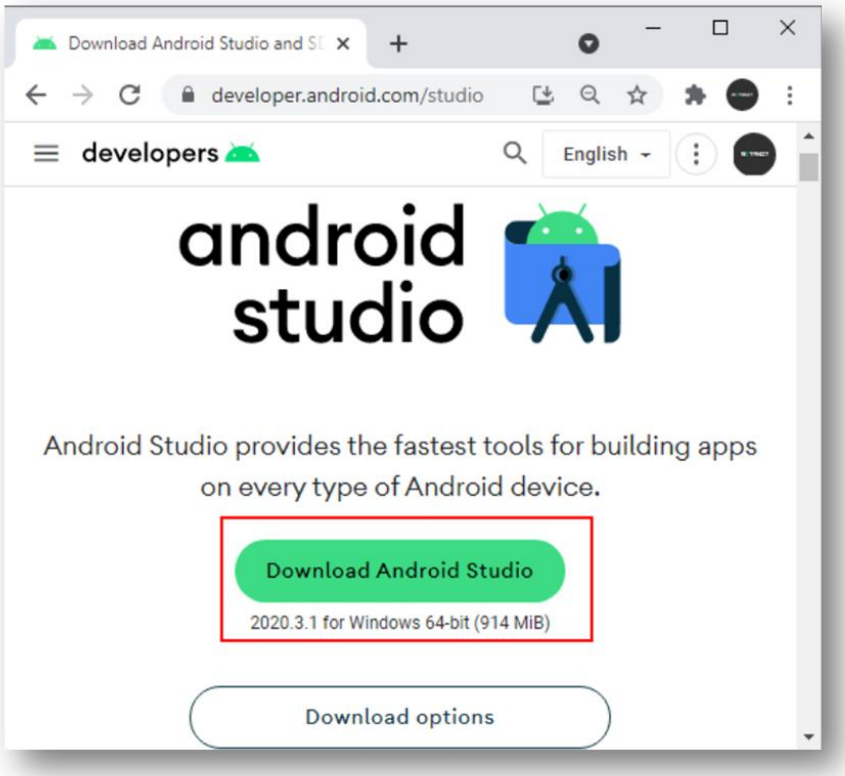
- **app:layout_constraintRight_toRightOf="parent"**: вказує, що права межа елемента буде вирівнюватися праворуч контейнера ConstraintLayout
- **app:layout_constraintBottom_toBottomOf="parent"**: вказує, що нижня межа елемента вирівнюватиметься по нижній стороні контейнера ConstraintLayout

Варто відзначити, що останні чотири атрибути разом призводитимуть до розташування TextView по центру екрана.

Таким чином, при запуску програми спочатку запускається клас MainActivity, який як графічний інтерфейс встановлює розмітку з файлу activity_main.xml. І оскільки в цій розмітці прописаний елемент TextView, який представляє певний текст, ми побачимо його текст на екрані смартфона.

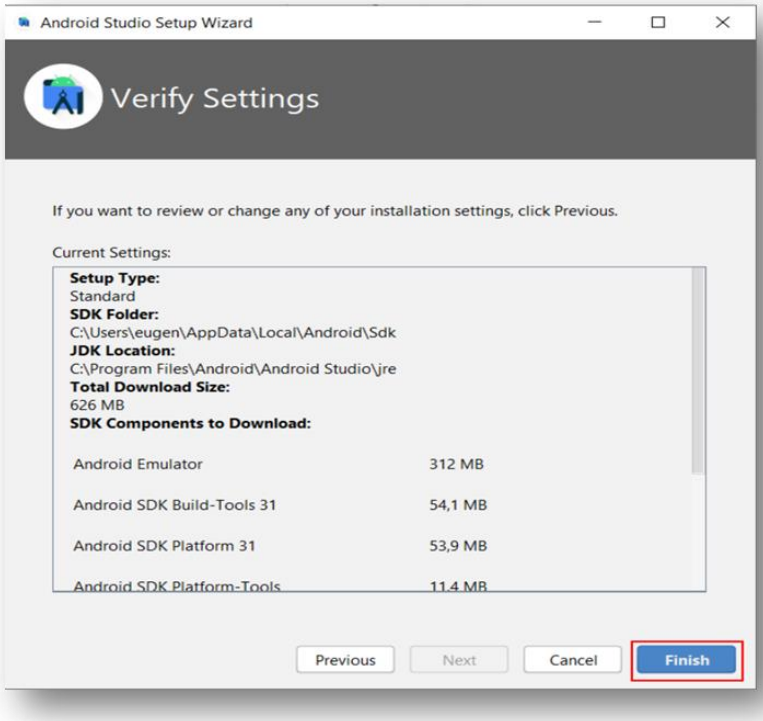
1.4. Установка засобів розробки

Існують різні середовища розробки для Android. Середовищем, що рекомендується, є Android Studio, яка створена спеціально для розробки під ОС Android. Тому ми її й використовуватимемо. Завантажити файл інсталятора можна з офіційного сайту:https://developer.android.com/studio:



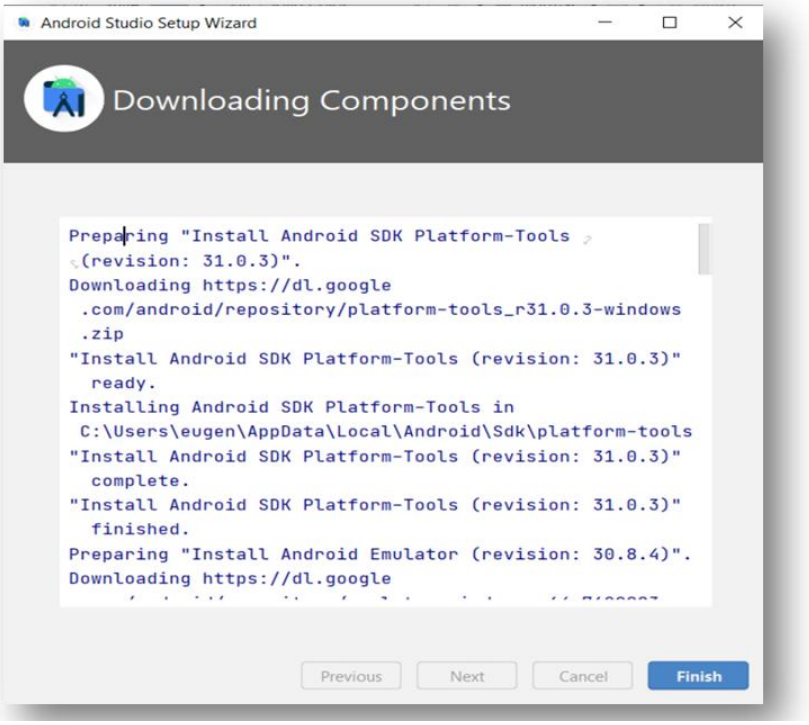
Крім самого середовища Android Studio для розробки також знадобиться набір інструментів, який називається Android SDK. Наприклад, якщо раніше Android SDK ще не було встановлено, то при

першому зверненні до Android Studio вона запропонує встановити додаткові інструменти, які необхідні для розробки. Насамперед це Android SDK і ряд додаткових



компонентів:

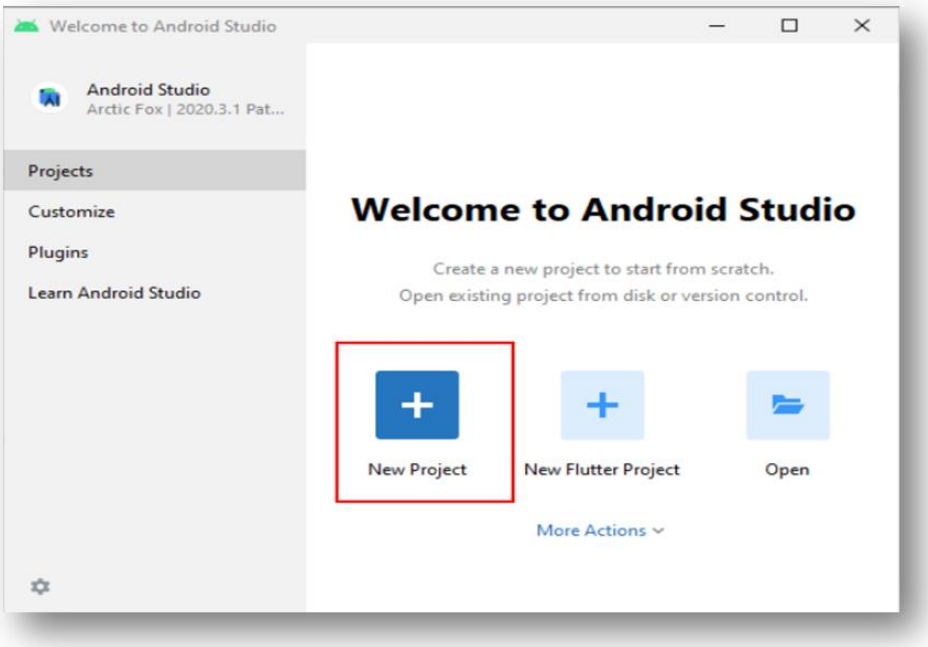
Натисніть кнопку Finish, щоб, нарешті, все це встановити.



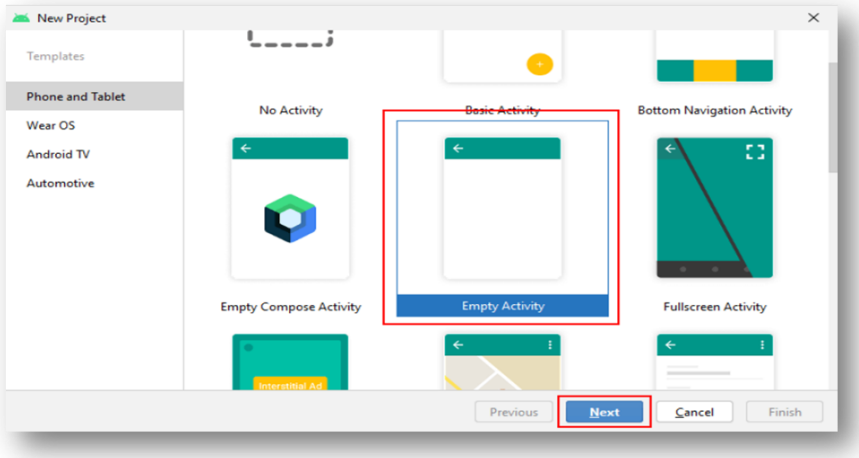
І після завершення установки натисніть кнопку Finish. І ми можемо приступати до створення програм.

1.5. Перший проект у Android Studio

Тепер створимо першу програму в середовищі Android Studio для операційної системи Android. Відкриємо Android Studio і на початковому екрані оберемо пункт New Project:

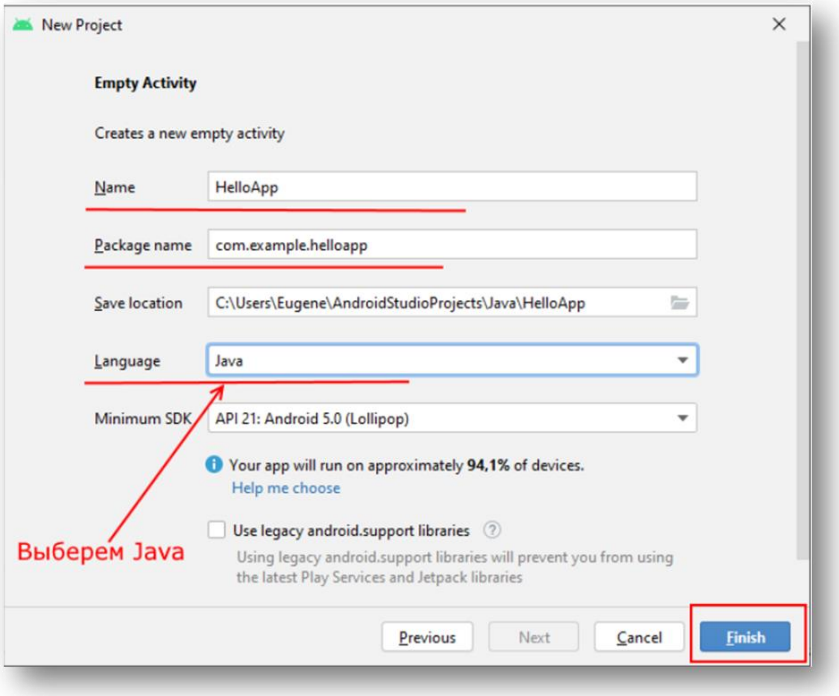


При створенні проекту Android Studio спочатку запропонує нам вибрати шаблон проекту:



Android Studio надає низку шаблонів для різних ситуацій. Виберемо в цьому списку шаблон Empty Activity, який надає найпростіший фукціонал, необхідний для початку, і натиснемо на кнопку Next.

Після цього з'явиться вікно налаштувань нового проекту:

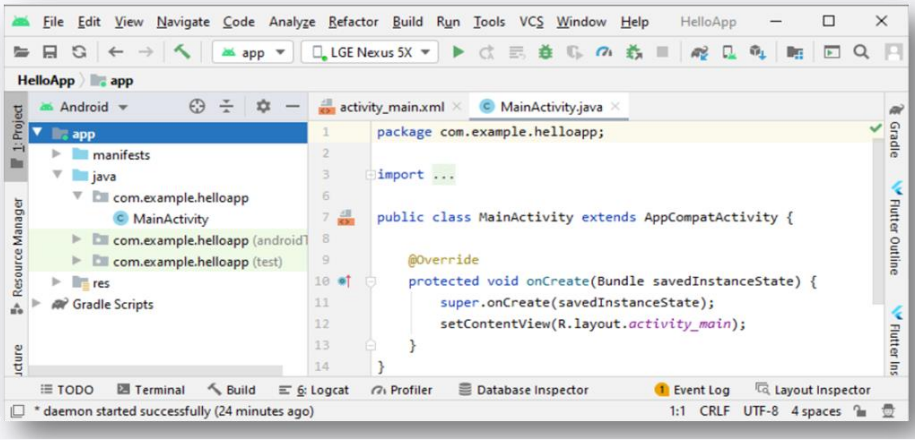


У вікні створення нового проекту ми можемо встановити його початкові налаштування:

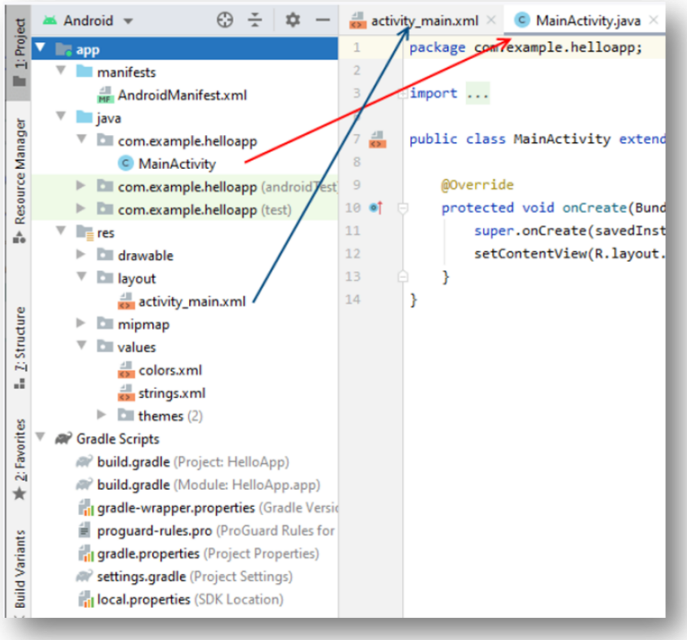
- У полі Name вводиться назва програми. Вкажемо як назву HelloApp
- У полі Package Name вказується ім'я пакета, де розміщуватиметься головний клас програми. В даному випадку для тестових проектів це значення не має значення, тому встановимо com.example.helloapp.
- У полі Save Location встановлюється розташування файлів проекту на жорсткому диску. За замовчуванням можна залишити значення.
- У полі Language як мову програмування вкажемо Java (будуть уважні, оскільки за умовчанням у цьому полі стоїть Kotlin)
- У полі Minimum SDK вказується мінімальна підтримувана версія SDK. Залишимо значення за замовчуванням - API 21: Android 5.0 (Lollipop), яка означає, що наша програма можна буде запустити починаючи з Android 5.0, а це 94% пристроїв. На старіших пристроях запустити не можна.

  Варто враховувати, що чим вище версія SDK, тим менший діапазон пристроїв, що підтримуються.

Далі натиснемо на кнопку Finish, і Android Studio створить новий проект:



Спочатку коротко розглянемо структуру проекту, що він має за замовчуванням



Проект Android може складатися із різних модулів. За замовчуванням, коли створюємо проект, створюється один модуль - app. Модуль має три підпапки:

- **manifests**: зберігає файл маніфесту AndroidManifest.xml, який описує конфігурацію програми та визначає кожен із компонентів цієї програми.
- **java**: зберігає файли коду мовою java, які структуровані окремими пакетами. Так, у папці com.example.helloapp (назва якого було вказано на етапі створення проекту) є за замовчуванням файл MainActivity.java з кодом на мові Java, який представляє клас MainActivity, запускається за замовчуванням при старті програми
- **res**: містить ресурси, що використовуються в програмі. Усі ресурси розбиті на підпапки.

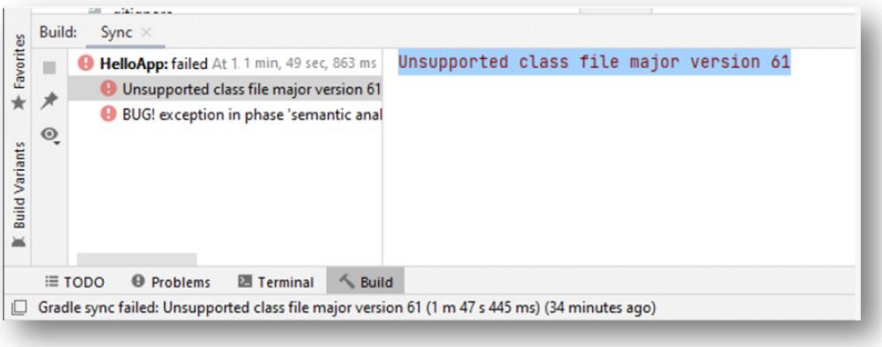
- папка drawable призначена для зберігання зображень, що використовуються в програмі
- папка layout призначена для зберігання файлів, що визначають графічний інтерфейс. За замовчуванням є файл activity_main.xml, який визначає інтерфейс для класу MainActivity у вигляді xml
- папки mipmap містять файли зображень, які призначені для створення іконки програми за різних роздільних здатностей екрана.
- папка values зберігає різні xml-файли, що містять колекції ресурсів - різні дані, що застосовуються в додатку. За замовчуванням тут є два файли та одна папка:
  - файл colors.xml зберігає опис кольорів, що використовуються в програмі
  - файл strings.xml містить рядкові ресурси, що використовуються в програмі
  - папки themes зберігає дві теми програми - для світла (денна) і темна (нічна)

Окремий елемент Gradle Scripts містить низку скриптів, які використовуються при побудові програми.

У всій цій структурі слід виділити файл MainActivity.java, який відкритий в Android Studio і який містить логіку програми і з нього починається виконання програми. Також виділимо файл activity_main.xml, який визначає графічний інтерфейс - по суті те, що побачить користувач на своєму смартфоні після завантаження програми.

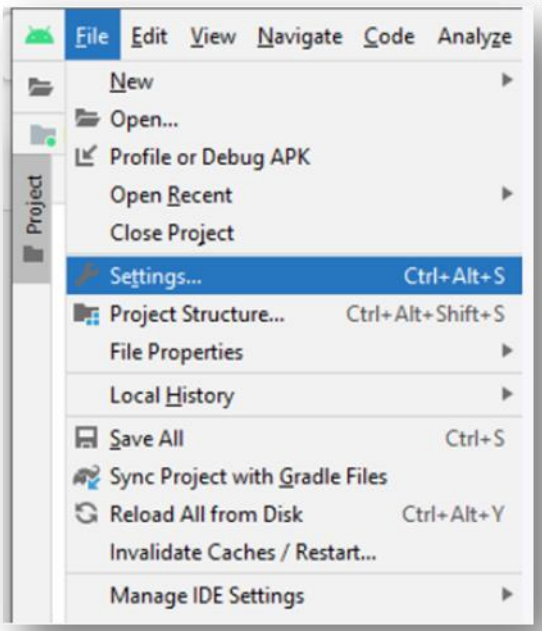
1.6. Можливі проблеми

Для створення програми використовується Java. А для побудови програми використовується інфраструктура Gradle. Однак поточна версія Gradle може бути несумісною з вибраною за замовчуванням версією JDK. І в цьому випадку Android Studio може відображати помилки, наприклад, помилку Unsupported class file major version 61:

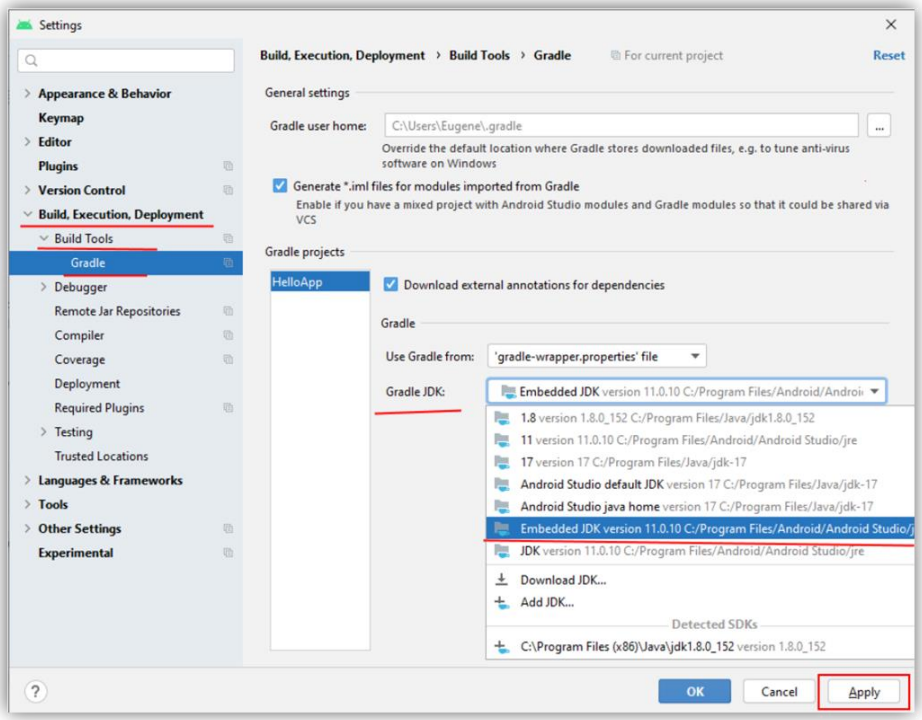


Ця помилка свідчить про те, що версія JDK 17 несумісна із поточною версією Gradle. І треба використати меншу версію.

Для вирішення цієї проблеми перейдемо в студії до меню File->Settings (на MacOS це пункт Android Studio->Preferences)



Потім у вікні налаштувань перейдемо до пункту меню Build, Execution, Deployment -> Build Tools -> Gradle і далі знайдемо поле Gradle JDK, де змінимо версію JDK. Вона повинна мати версію 11 та вище. Як правило, разом з Android Studio встановлюється і версія JDK, що підтримується - на даний момент це JDK 11. І її можна вибрати в списку JDK:



Найбільш оптимальний пункт для вибору версій JDK, яка йде разом з Android Studio, називається Embedded JDK version. Як видно на скріншоті, це версія 11, але при подальших оновленнях Android Studio ця версія може змінитися.

Після зроблених змін спочатку натисніть кнопку Apply, а потім на кнопку OK. І повторимо запуск проекту.

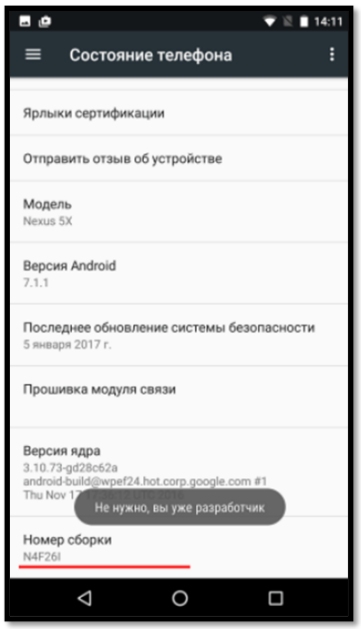
1.7. Запуск проекту

Створений вище проект містить певний примітивний функціонал. Правда, цей функціонал майже нічого не робить, тільки виводить на екран рядок Hello world!. Тим не менш, це вже фактично додаток, який ми можемо запустити.

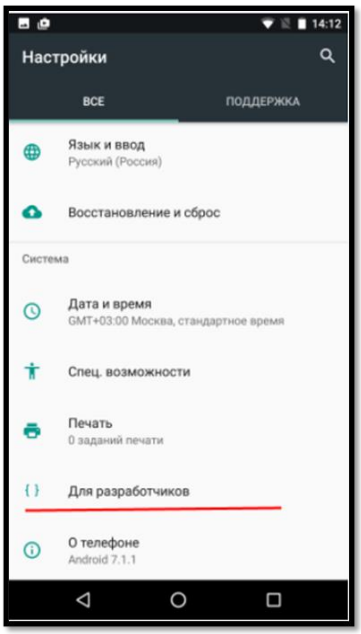
Для запуску та тестування програми ми можемо використовувати емулятори або реальні пристрої. Але в ідеалі найкраще тестувати на реальних пристроях. До того ж емулятори вимагають великих апаратних ресурсів, і кожен комп'ютер може потягнути вимоги емуляторів. А для використання мобільного пристрою для тестування може знадобитися хіба встановити необхідний драйвер.

1.8. Режим розробника на телефоні

За замовчуванням опції розробника на смартфонах приховані. Щоб зробити їх доступними, потрібно зайти в Settings > About phone (Установки > Про телефон) (в Android 8 це в Settings > System > About phone (Установки > Система > Про телефон)) і сім разів натиснути Build Number (Номер складання).



Тепер необхідно увімкнути налагодження USB. Для цього перейдемо в Settings > System > Advanced > Developer options або Установки > Система > Додатково > Для розробників (в Android 8 це в Settings > System > Developer options або Установки > Система > Для розробників).



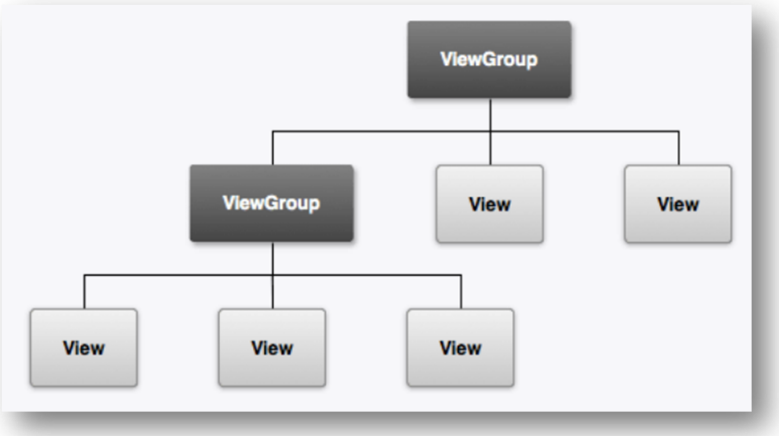
І включимо можливість налагодження по USB:

2. ОСНОВИ СТВОРЕННЯ ІНТЕРФЕЙСУ

2.1. Введення у створення інтерфейсу

Графічний інтерфейс користувача є ієрархією об'єктів android.view.View і android.view.ViewGroup. Кожен об'єкт ViewGroup представляє контейнер, який містить та впорядковує дочірні об'єкти View. Зокрема, до контейнерів відносять такі елементи, як RelativeLayout, LinearLayout, GridLayout, ConstraintLayout та інші.

Прості об'єкти View являють собою елементи керування та інші віджети, наприклад, кнопки, текстові поля і т.д., через які користувач взаємодіє з програмою:



Більшість візуальних елементів, що успадковуються від класу View, такі як кнопки, текстові поля та інші, розміщуються у пакеті`android.widget`

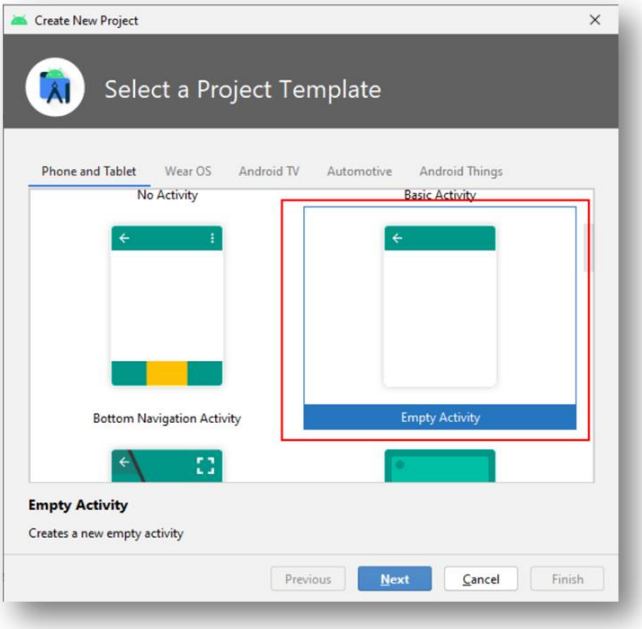
При визначенні візуального ми маємо три стратегії:

- Створити елементи керування програмно в коді java
- Оголосити елементи інтерфейсу в XML
- Поєднання обох способів - базові елементи розмітки визначити в XML, а інші додавати під час виконання

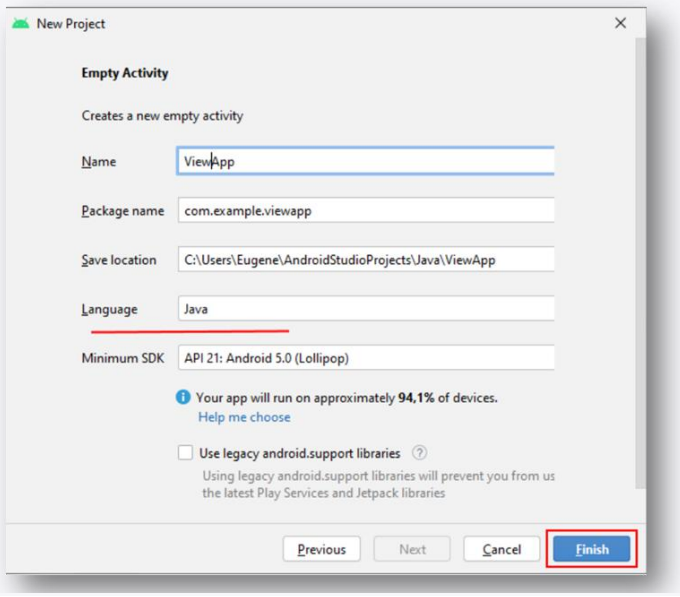
Спочатку розглянемо першу стратегію – визначення інтерейсу в коді Java.

2.2. Створення інтерфейсу в коді java

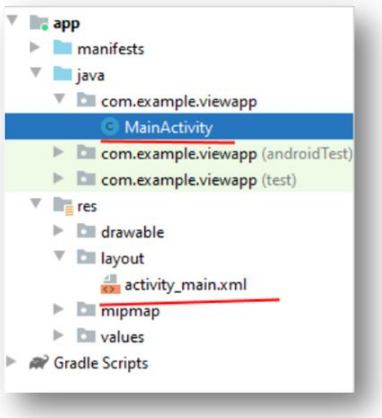
Для роботи із візуальними елементами створимо новий проект. Як шаблон проекту виберемо Empty Activity:



Нехай він буде називатися ViewsApp:



І після створення проекту два основних файли, які нас цікавлять при створенні візуального інтерфейсу - це клас MainActivity і визначення інтерфейсу для цієї activity у файлі activity_main.xml.



Визначимо у класі MainActivity найпростіший інтерфейс:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

// Створення TextView
TextView textView = New TextView(this);
// встановлення тексту в TextView
textView.setText("Hello Android!");
// Встановлення висоти тексту
textView.setTextSize(22);
// встановлення візуального інтерфейсу для activity
setContentView(textView);
}
}
```

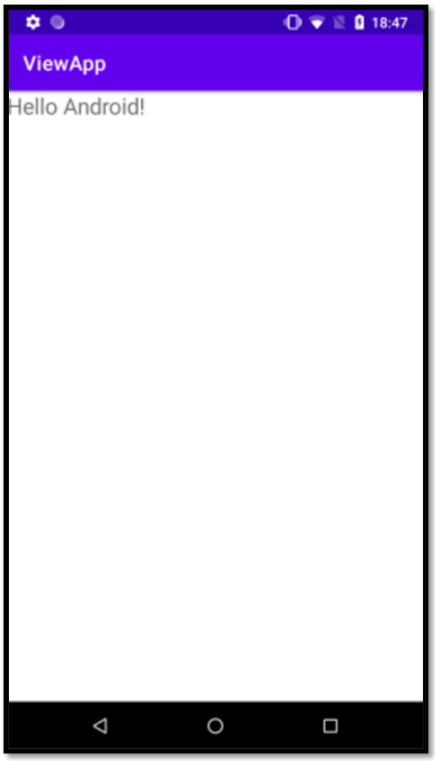
При створенні віджетів у коді Java застосовується їх конструктор, в який передається контекст даного віджету, а точніше об'єкт android.content.Context, як якого виступає поточний клас MainActivity.

```
TextView textView = New TextView(this);
```

Тут весь інтерфейс представлений елементом TextView, який призначений для виведення тексту. За допомогою методів, які зазвичай починаються на set, можна встановити різні властивості TextView. Наприклад, у цьому випадку метод`setText()`встановлює текст у полі, а`setTextSize()`задає висоту шрифту.

Для встановлення елемента як інтерфейс програми в коді Activity викликається метод setContentView(), в який передається візуальний елемент.

Якщо ми запустимо програму, то отримаємо наступний візуальний інтерфейс:



Подібним чином ми можемо створювати складніші інтерейси. Наприклад, TextView, вкладений у ConstraintLayout:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

ConstraintLayout constraintLayout = New
ConstraintLayout(this);
TextView textView = New TextView(this);
textView.setText("Hello Android!");
textView.setTextSize(26);
// встановлюємо параметри розмірів та розташування
```

```
елемента
     ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     // вирівнювання по лівому краю ConstraintLayout
     layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
     // Вирівнювання по верхньому кордоні ConstraintLayout
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
     // встановлюємо параметри textView
     textView.setLayoutParams(layoutParams);
     // додаємо TextView у ConstraintLayout
     constraintLayout.addView(textView);
     // як кореневий
     setContentView(constraintLayout);
     }
     }
```

Для кожного контейнера конкретні дії щодо додавання та позиціонування в ньому елемента можуть відрізнятися. В даному випадку контейнерів виступає клас ConstraintLayout, тому для визначення позиціонування та розмірів елемента необхідно створити об'єкт ConstraintLayout.LayoutParams. (Для LinearLayout це відповідно буде LinearLayout.LayoutParams, а RelativeLayout - RelativeLayout.LayoutParams і т.д.). Цей об'єкт ініціалізується двома параметрами: шириною та висотою. Для вказівки ширини та висоти можна використовувати константу ViewGroup.LayoutParams.WRAP_CONTENT, яка встановлює розміри елемента, необхідні для розміщення на екрані вмісту.

Далі визначається позиціонування. В залежності від типу контейнера набір властивостей, що встановлюються, може відрізнятися. Так, рядок коду

```
     layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
```

вказує, що ліва межа елемента вирівнюватиметься по лівій межі контейнера.

А рядок коду

```
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
```

```
_ID;
```

вказує, що верхня межа елемента вирівнюватиметься по верхній ганиці контейнера. В результаті елемент буде розміщений у верхньому лівому кутку ConstraintLayout.

Для встановлення всіх цих значень для конкретного елемента (TextView) у його методsetLayoutParams()передається об'єкт ViewGroup.LayoutParams (або його спадкоємців, наприклад, ConstraintLayout.LayoutParams).

```
textView.setLayoutParams(layoutParams);
```

Всі класи контейнерів, які успадковуються від android.view.ViewGroup (RelativeLayout, LinearLayout, GridLayout, ConstraintLayout і т.д.), мають методaddView(android.view.View child), який дозволяє додати до контейнера інший елемент - звичайний віджет типу TextView або інший контейнер. І в даному випадку за допомогою даного методу TextView додається до ConstraintLayout:

```
constraintLayout.addView(textView);
```

Знову ж таки зазначу, що для конкретного контейнера конкретні дії можуть відрізнятися, але як правило для всіх характерно три етапи:

- Створення об'єкта ViewGroup.LayoutParams та встановлення його властивостей
- Передача об'єкта ViewGroup.LayoutParams у методsetLayoutParams()елемента
- Передача елемента для додавання до методуaddView()об'єкта контейнера

Хоча ми можемо використовувати подібний підхід, водночас більш оптимально визначати візуальний інтерейс у файлах xml, а пов'язану логіку визначати в класі activity. Тим самим ми досягнемо розмежування інтерфейсу та логіки програми, їх легше буде розробляти та згодом модифікувати. І на наступній темі ми це розглянемо.

2.3. Визначення інтерфейсу у файлі XML. Файли layout

Як правило, для визначення візуального інтерфейсу в проектах під Android використовують спеціальні файли xml. Ці файли є ресурсами розмітки та зберігають визначення візуального інтерфейсу як код XML. Подібний підхід нагадує створення веб-сайтів, коли інтерфейс визначається у файлах html, а логіка програми – у коді javascript.

Оголошення інтерфейсу користувача у файлах XML дозволяє відокремити інтерфейс програми від коду. Що означає, що ми можемо змінювати визначення інтерфейсу без зміни коду java. Наприклад, у

програмі можуть бути визначені розмітки у файлах XML для різних орієнтацій монітора, різних розмірів пристроїв, різних мов тощо. Крім того, оголошення розмітки в XML дозволяє легко візуалізувати структуру інтерфейсу і полегшує налагодження.

Файли розмітки графічного інтерфейсу розміщуються у проекті в каталозі res/layout. За умовчанням при створенні проекту з порожньою activity вже є один файл ресурсів розмітки activity_main.xml, який може виглядати приблизно так:

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

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

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

У файлі визначаються всі графічні елементи та його атрибути, які становлять інтерфейс. При створенні розмітки в XML слід дотримуватись деяких правил: кожен файл розмітки повинен містити один кореневий елемент, який повинен представляти об'єкт View або ViewGroup.

У разі кореневим елементом є елемент ConstraintLayout, який містить елемент TextView.

Як правило, кореневий елемент містить визначення простір імен XML. Наприклад, у коді за замовчуванням у ConstraintLayout ми можемо побачити такі атрибути:

```
    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"
```

Кожен простір імен задається так:`xmlns:префікс="назва_ресурсу"`. Наприклад, в

```
    xmlns:android="http://schemas.android.com/apk/res/android"
```

Назва ресурсу (або URI - Uniform Resource Indicator) - "http://schemas.android.com/apk/res/android". І цей ресурс зіставляється із префіксомandroid(xmlns:android). Тобто через префікс ми можемо посилатися на функціональність цього простору імен.

Кожен простір імен визначає певну функціональність, яка використовується в програмі, наприклад, надають теги та атрибути, які необхідні для створення програми.

- xmlns:android="http://schemas.android.com/apk/res/android": містить основні атрибути, що надаються платформою Android, застосовуються в елементах керування та визначають їх візуальні властивості (наприклад, розмір, позиціонування). Наприклад, у коді ConstraintLayout використовується наступний атрибут із простору імен "http://schemas.android.com/apk/res/android":

```
android:layout_width="match_parent"
```

- xmlns:app="http://schemas.android.com/apk/res-auto": містить атрибути, визначені в рамках програми. Наприклад, у коді TextV-?ew:

```
app:layout_constraintBottom_toBottomOf="parent"
```

- xmlns:tools="http://schemas.android.com/tools": застосовується для роботи з режимом дизайнера Android Studio

Це найпоширеніші простори імен. І зазвичай кожен кореневий елемент (не обов'язково лише ConstraintLayout) їх містить. Однак, якщо ви не плануєте користуватися графічним дизайнером в Android Studio і хочете працювати повністю в коді xml, то відповідно сенсу в просторі імен "http://schemas.android.com/tools" немає, і його можна забрати.

При компіляції кожен XML-файл розмітки компілюється ресурс View. Завантаження ресурсу розмітки здійснюється методом Activity.onCreate. Щоб встановити розмітку для поточного об'єкта activity, треба в метод setContentView() як параметр передати посилання ресурс розмітки.

```
setContentView(R.layout.activity_main);
```

Для отримання посилання на ресурс у коді java необхідно використати вирazR.layout.[назва_ресурсу]. Назва ресурсу layout буде співпадати з ім'ям файлу, тому щоб використовувати файл activity_main.xml як джерело візуального інтерфейсу, можна визначити наступний код у класі MainActivity:

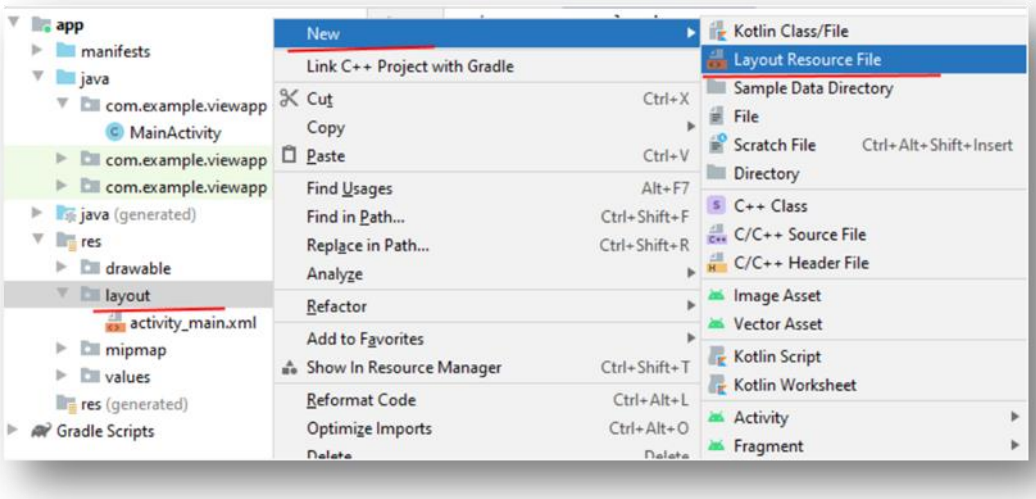
```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        // Завантаження інтерфейсу з файлу activity_main.xml
        setContentView(R.layout.activity_main);
        }}
```

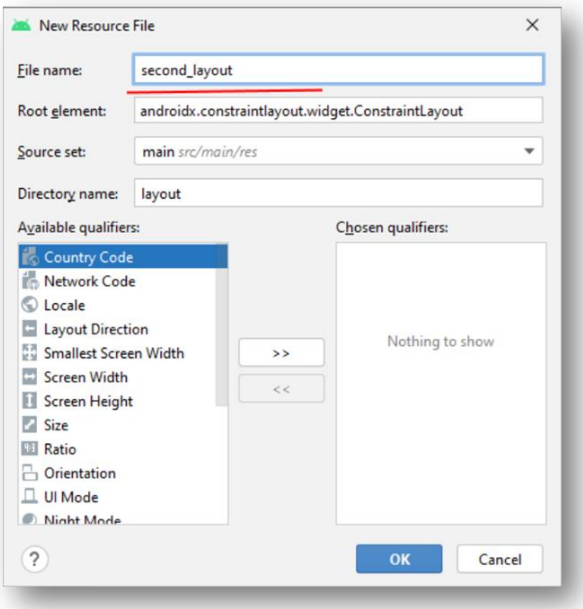
2.4. Додавання файлу layout

Але в нас може бути кілька різних ресурсів layout. Як правило, кожен окремий клас Activity використовує свій файл layout. Або одного класу Activity може використовуватися відразу кілька різних файлів layout.

Наприклад, додамо до проекту новий файл розмітки інтерфейсу. Для цього натиснемо на папку res/layout правою кнопкою миші і в меню виберемо пункт New -> Layout Resource File:



Після цього у спеціальному вікні буде запропоновано вказати ім'я та кореневий елемент для файлу layout:



Як назву вкажемо second_layout. Всі інші налаштування залишимо за замовчуванням:

- у полі Root element вказується кореневий елемент. За промовчанням це androidx.constraintlayout.widget.ConstraintLayout.
- поле Source set вказує, куди розміщувати новий файл. За замовчуванням це main - область проекту, з якою ми власне працюємо при розробці програми.
- поле Directory main вказує папку в рамках каталогу, вибраного в попередній опції, до якого власне міститься новий файл. За промовчанням для файлів з розміткою інтерфейсу це layout.

Після цього до папки res/layout буде додано новий файл second_layout.xml, з яким ми можемо працювати так само, як і з activity_main.xml. Зокрема, відкриємо файл second_layout.xml та змінимо його вміст наступним чином:

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

<TextView
android:id="@+id/header"
android:text="Welcome to Android"
android:textSize="26sp"
android:layout_width="match_parent"
android:layout_height="match_parent" />

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

Тут визначено текстове поле TextView, яке має такі атрибути:

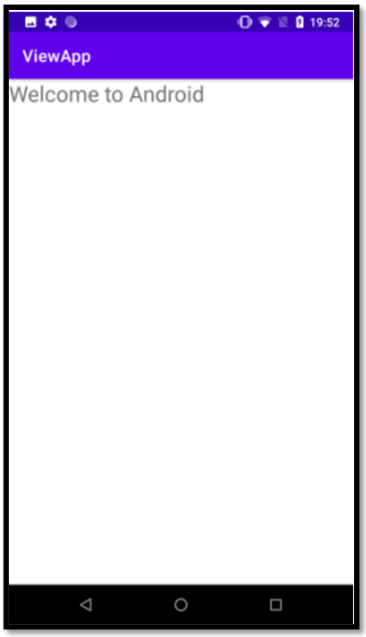
- `android:id`- ідентифікатор елемента, через який ми можемо посилатися на нього в коді. У записі`android:id="@+id/header"`символ @ вказує XML-парсеру використовувати частину рядка атрибута, що залишилася, як ідентифікатор. А знак + означає, що якщо елемент не визначений id зі значенням header, то його слід визначити.
- `android:text`- Текст елемента - на екран буде виводитися рядок "Welcome to Android".
- `android:textSize`- висота шрифту (тут 26 одиниць)
- `android:layout_width`- Ширина елемента. Значення "match_parent" вказує на те, що елемент буде розтягуватися по всій ширині контейнера ConstraintLayout

- `android:layout_height`- Висота елемента. Значення "match_parent" вказує на те, що елемент буде розтягуватися по всій висоті контейнера ConstraintLayout

Застосуємо цей файл як визначення графічного інтерфейсу в класі MainActivity:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.second_layout);
}
}
```

Файл інтерфейсу називається second_layout.xml, тому за умовчанням для нього буде створюватись ресурс`R.layout.second_layout`. Відповідно, щоб його використовувати, ми передаємо його методу setContentView. У результаті ми побачимо на екрані наступне:



Отримання та керування візуальними елементами в коді

Вище певний елемент TextView має дуже важливий атрибут - id або ідентифікатор елемента. Цей ідентифікатор дозволяє звертатися до елемента, визначеного у файлі xml, із коду Java. Наприклад, перейдемо до класу MainActivity та змінимо його код:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// встановлюємо як інтерфейс файл second_layout.xml
setContentView(R.layout.second_layout);
// Отримуємо елемент textView
TextView textView = findViewById(R.id.header);
// встановлюємо у нього текст
textView.setText("Hello from Java!");
}
}
```

За допомогою методу `setContentView()` встановлюється розмітка із файлу second_layout.xml.

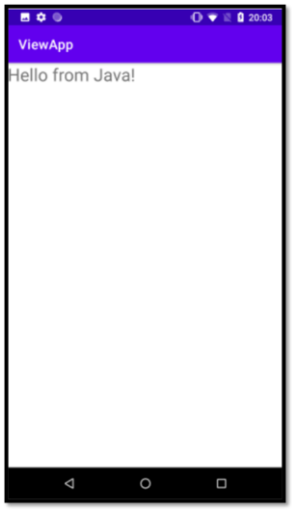
Інший важливий момент, який варто відзначити – отримання візуального елемента TextView. Оскільки в його коді ми визначили атрибут `android:id`, то через цей ID ми можемо його отримати.

Для отримання елементів id клас Activity має метод findViewById(). Цей метод передається ідентифікатор ресурсу як R.id.[идентификатор_элемента]. Цей метод повертає об'єкт View – об'єкт базового класу для всіх елементів, тому результат методу ще необхідно привести до типу TextView.

Далі ми можемо щось зробити з цим елементом, у разі змінюємо його текст.

Причому важливо, отримання елемента відбувається після того, як у методі setContentView була встановлена розмітка, в якій цей візуальний елемент був визначений.

І якщо ми запустимо проект, побачимо, що TextView виводить новий текст:

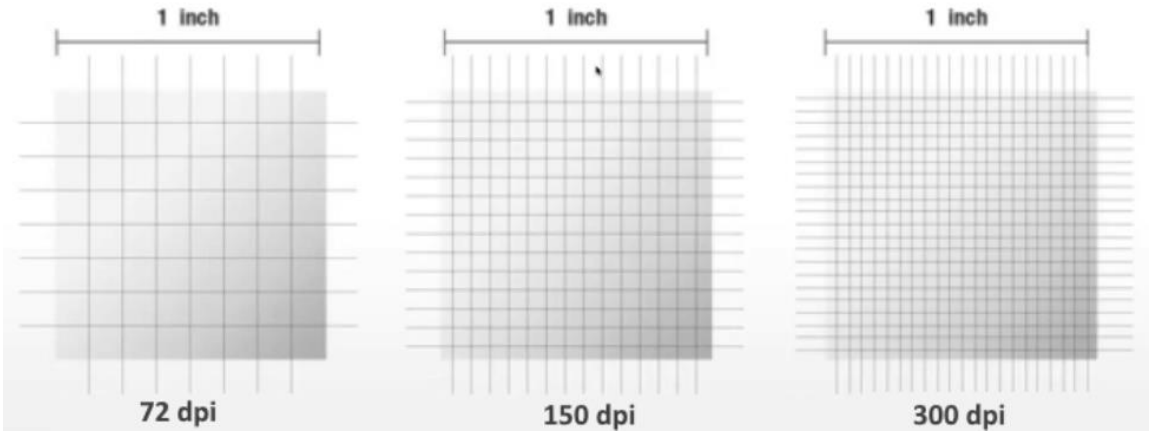


2.5. Визначення розмірів

При розробці програм під Android ми можемо використовувати різні типи вимірювань:

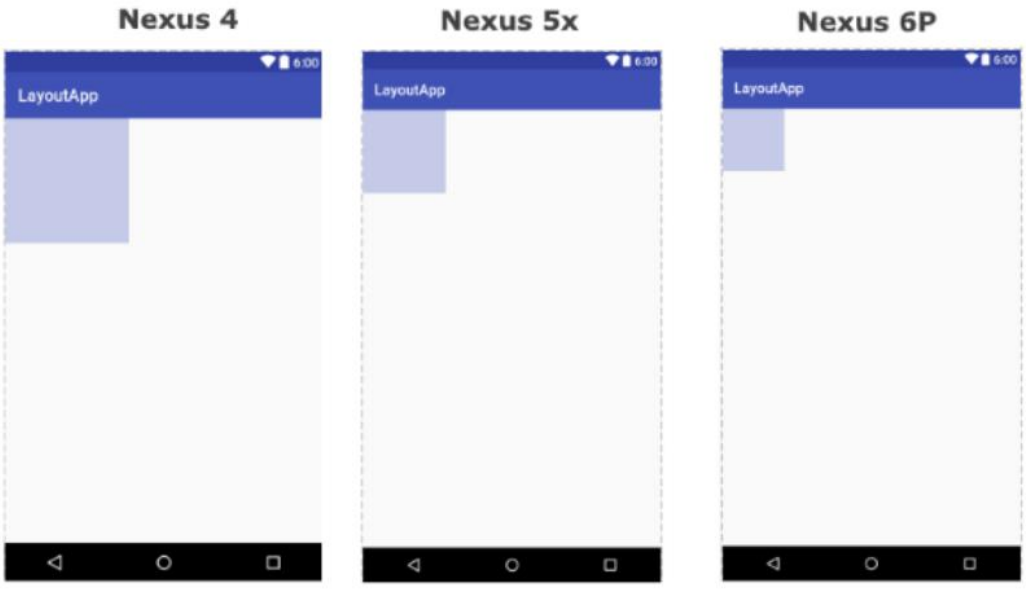
- `px`: пікселі поточного екрана Однак ця одиниця виміру не рекомендується, так як реальне уявлення зовнішнього вигляду може змінюватись в залежності від пристрою; кожен пристрій має певний набір пікселів на дюйм, тому кількість пікселів на екрані також може змінюватися
- `dp`: (device-independent pixels) незалежні від щільності екрану пікселі. Абстрактна одиниця виміру, заснована на фізичній щільності екрана з роздільною здатністю 160 dpi (крапок на дюйм). І тут 1dp = 1px. Якщо розмір екрана більший або менший, ніж 160dpi, кількість пікселів, які застосовуються для відображення 1dp відповідно, збільшується або зменшується. Наприклад, на екрані з 240 dpi 1dp=1,5px, а на екрані з 320dpi 1dp=2px. Загальна формула для отримання кількості фізичних пікселів з dp: px = dp* (dpi/160)
- `sp`: (scale-independent pixels) незалежні від масштабування пікселів. Допускають налаштування розмірів, яке виробляється користувачем. Рекомендуються для роботи зі шрифтами.
- `pt`: 1/72 дюйми, базуються на фізичних розмірах екрану.
- `mm`: міліметри
- `in`: дюйми

Переважними одиницями для використання є dp. Це пов'язано з тим, що світ мобільних пристроїв на Android сильно фрагментований у плані роздільної здатності та розмірів екрану. І чим більша щільність пікселів на дюйм, тим відповідно більше пікселів нам буде доступно:



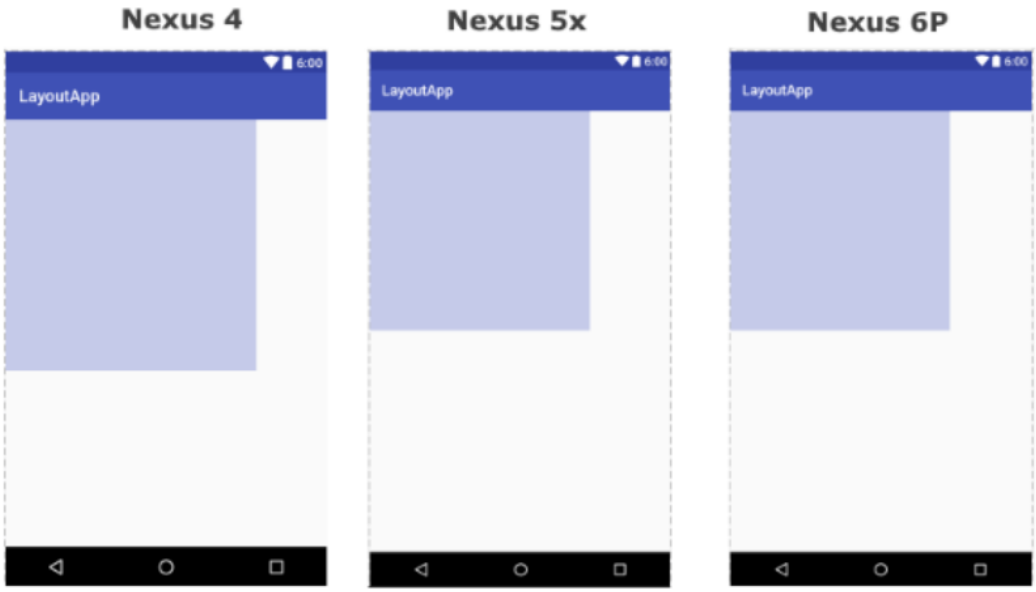
Використовуючи стандартні фізичні пікселі ми можемо зіткнутися з проблемою, що розміри елементів також будуть сильно варіюватися в залежності від щільності пікселів пристрою. Наприклад, візьмемо 3

пристрої з різними характеристиками екрану Nexus 4, Nexus 5X і Nexus 6P і виведемо на екран квадрат розміром 300px на 300px:



В одному випадку квадрат по ширині займатиме 40%, в іншому – третина ширини, у третьому – 20%.

Тепер також візьмемо квадрат зі сторонами 300x300, але замість фізичних пікселів використовуємо одиниці dp:



Тепер розміри квадрата на різних пристроях виглядають більш консистентно.

Для спрощення роботи з розмірами всі розміри розбиті на кілька груп:

- **ldpi (low)**: ~120dpi
- **mdpi (medium)**: ~160dpi

- **hdpi (high)**: ~240dpi (до цієї групи можна віднести такий древній пристрій як Nexus One)
- **xhdpi (extra-high)**: ~320dpi (Nexus 4)
- **xxhdpi (extra-extra-high)**: ~480dpi (Nexus 5/5X, Samsung Galaxy S5)
- **xxxhdpi (extra-extra-extra-high)**: ~640dpi (Nexus 6/6P, Samsung Galaxy S6)

2.6. Встановлення розмірів

Основна проблема, пов'язана з розмірами, пов'язана з їх встановленням Java. Наприклад, деякі методи приймають як значення фізичні пікселі, а не device-independent pixels. У цьому випадку може знадобитися перевести значення з одного типу одиниць до іншого. Для цього потрібно застосувати метод TypedValue.applyDimension(), який приймає три параметри:

```
public static float applyDimension(int unit,
float value,
android.util.DisplayMetrics metrics)
```

Параметр`unit`представляє тип одиниць, з якої треба набути значення в пікселях. Тип одиниць описується однією з констант TypedValue:

- `COMPLEX_UNIT_DIP`- dp або незалежні від щільності екрану пікселі
- `COMPLEX_UNIT_IN`- in або дюйми
- `COMPLEX_UNIT_MM`- mm або міліметри
- `COMPLEX_UNIT_PT`- pt або точки
- `COMPLEX_UNIT_PX`- px або фізичні пікселі
- `COMPLEX_UNIT_SP`- sp або незалежні від масштабування пікселі (scale-independent pixels)

Параметр`value`представляє значення, яке треба перетворити

.

Параметр`metrics`представляє інформацію про метрику, в рамках якої треба виконати перетворення.

Через війну спосіб повертає перетворене значення. Розглянемо абстрактний приклад. Наприклад, нам треба отримати з 60dp звичайні фізичні пікселі:

```
int valueInDp = 60;
int valueInPx = (int) TypedValue.applyDimension(
```

```
    TypedValue.COMPLEX_UNIT_DIP, valueInDp,
getResources().getDisplayMetrics());
```

Як третій аргумент передається виклик методу`getResources().getDisplayMetrics()`, який дозволяє отримати інформацію про метрику, пов'язану з поточним пристроєм. У результаті ми отримаємо з 60dp кілька пікселів.

2.7. Ширина та висота елементів

Всі візуальні елементи, які ми використовуємо в додатку, зазвичай впорядковуються на екрані за допомогою контейнерів. В Android подібними контейнерами є такі класи як RelativeLayout, LinearLayout, GridLayout, TableLayout, ConstraintLayout, FrameLayout. Всі вони по-різному мають у своєму розпорядженні елементи і керують ними, але є деякі загальні моменти при компонуванні візуальних компонентів, які ми зараз розглянемо.

Для організації елементів у контейнері використовуються параметри розмітки. Для їхнього завдання у файлі xml використовуються атрибути, які починаються з префіксу layout_. Зокрема, такі параметри включають атрибути layout_height і layout_width, які використовуються для встановлення розмірів і можуть використовувати одну з наступних опцій:

- Розтяг по всій ширині або висоті контейнера за допомогою значення match_parent (для всіх контейнерів крім ConstraintLayout) або 0dp (для ConstraintLayout)
- Розтягування елемента до тих меж, які є достатніми, щоб вмістити весь його вміст за допомогою значення wrap_content
- Точні розміри елемента, наприклад`96 dp`

match_parent

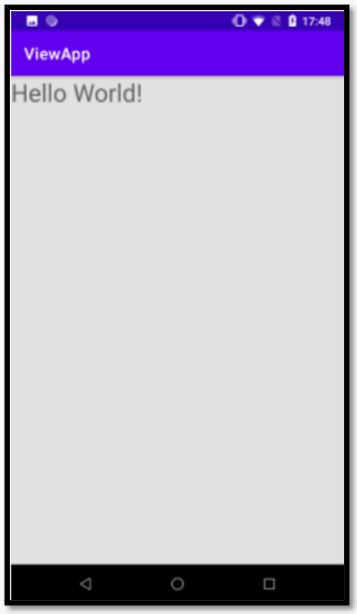
Встановлення значення match_parent дозволяє розтягнути елемент на всій ширині або висоті контейнера. Це значення застосовується до всіх контейнерів, крім ConstraintLayout. Наприклад, рясніть елемент TextView по всій ширині та висоті контейнера LinearLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:text="Hello World!"
```

```
    android:textSize="30sp"
    android:background="#e0e0e0" />
    </LinearLayout>
```

Контейнер самого верхнього рівня, в якості якого в даному випадку виступає LinearLayout, для висоти та ширини має значення match_parent, тобто він буде заповнювати всю область для діяльності - як правило, весь екран.

І TextView також приймає такі атрибути. Значення`android:layout_width="match_parent"`забезпечує розтяг по ширині, а`android:layout_height="match_parent"`- по вертикалі. Для наочності в TextView застосовує атрибут android:background, який представляє фон і в даному випадку забарвлює елемент у колір "#e0e0e0", завдяки чому ми можемо побачити область, яку він займає.



Слід враховувати, що значення match_parent можна застосовувати майже у всіх вбудованих контейнерах типу LinearLayout або RelativeLayout та їх елементах. Однак match_parent не рекомендується застосовувати до елементів усередині ConstraintLayout. Замість "match_parent" у ConstraintLayout можна використовувати значення 0dp, щоб розтягнути елемент по горизонталі або вертикалі:

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

    <TextView
    android:layout_width="0dp"
```

```
    android:layout_height="0dp"
    android:text="Hello World!"
    android:textSize="30sp"
    android:background="#e0e0e0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    />

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

Варто зазначити, що ConstraintLayout сама також розтягується по ширині та висоті екрана за допомогою значення "match_parent" в атрибутах`layout_width`і`android:layout_height`, але до вкладених елементів це значення не рекомендується застосовувати.

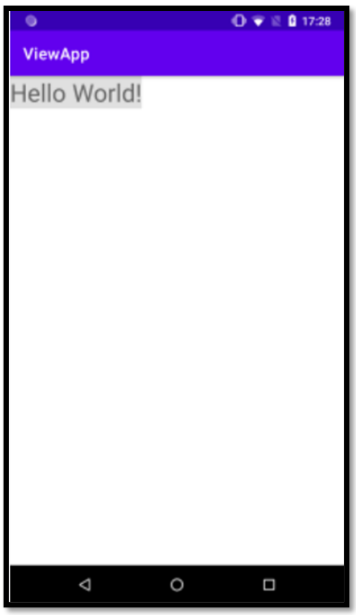
Оскільки ConstraintLayout має деякі особливості при встановленні розмірів, то більш детально робота з розмірами елементів саме в ConstraintLayout більш детально розкрита в одній з наступних тем.

wrap_content

Значення wrap_content встановлює значення для ширини або висоти, які необхідні для розміщення вмісту елемента на екрані:

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

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="30sp"
    android:background="#ffcdd2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    />

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

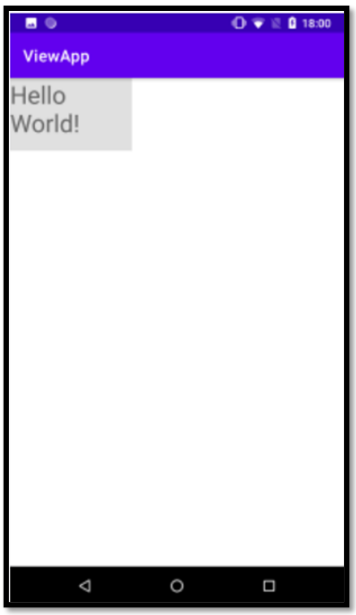
Тут елемент TextView розтягується до тих значень, які є достатніми для розміщення його тексту.



2.8. Встановлення точних значень

Також ми можемо встановити точні значення:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">
<TextView
android:layout_height="90dp"
android:layout_width="150dp"
android:text="Hello World!"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>

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



Крім того, можна комбінувати кілька значень, наприклад, розтягнути по ширині вмісту та встановити точні значення для висоти:

```
<TextView
android:layout_height="80dp"
android:layout_width="wrap_content"
android:text="Hello World!"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>
```

Якщо для встановлення ширини та довжини використовується значення wrap_content, то ми можемо додатково обмежити мінімальні та максимальні значення за допомогою атрибутів minWidth/maxWidth та minHeight/maxHeight:

```
<TextView
android:minWidth="200dp"
android:maxWidth="250dp"
android:minHeight="100dp"
android:maxHeight="200dp"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:text="Hello World!"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>
```

У цьому випадку ширина TextView буде такою, яка є достатньою для вміщення тексту, але не більше значення maxWidth і не менше значення minWidth. Те саме для встановлення висоти.

2.9. Програмне встановлення ширини та висоти

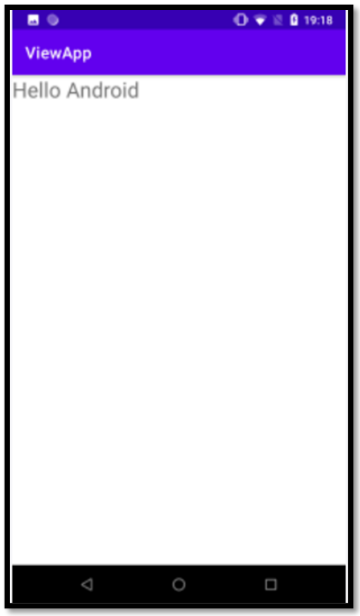
Якщо елемент, наприклад, той же TextView створюється в коді java, то для встановлення висоти та ширини можна використовувати спосіб setLayoutParams(). Так, змінимо код MainActivity:

```
     package com.example.viewapp;

     import androidx.appcompat.app.AppCompatActivity;
     import androidx.constraintlayout.widget.ConstraintLayout;
     import android.os.Bundle;
     import android.widget.TextView;

     public class MainActivity extends AppCompatActivity {

     @Override
     protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     ConstraintLayout constraintLayout = New
ConstraintLayout(this);
     TextView textView = New TextView(this);
     textView.setText("Hello Android");
     textView.setTextSize(26);

     // встановлюємо параметри розмірів та розташування
елемента
     ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     // еквівалент app:layout_constraintLeft_toLeftOf="parent"
     layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
     // еквівалент app:layout_constraintTop_toTopOf="parent"
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
     // встановлюємо параметри textView
     textView.setLayoutParams(layoutParams);
     // додаємо TextView у ConstraintLayout
     constraintLayout.addView(textView);
     setContentView(constraintLayout);
     }}
```

У методsetLayoutParams()передається об'єкт ViewGroup.LayoutParams. Цей об'єкт ініціалізується двома параметрами: шириною та висотою. Для вказівки ширини та висоти можна використовувати одну з констант ViewGroup.LayoutParams.WRAP_CONTENT або

ViewGroup.LayoutParams.MATCH_PARENT (у разі LinearLayout або RelativeLayout).

Оскільки в даному випадку ми маємо справу з контейнером ConstraintLayout, то для встановлення розмірів застосовується значення ConstraintLayout.LayoutParams.WRAP_CONTENT. Насправді клас`androidx.constraintlayout.widget.ConstraintLayout.LayoutParams`, який надає це значення, успадковується від`android.view.ViewGroup.LayoutParams`



Також ми можемо передати точні значення або комбінувати типи значень:

```
    ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
    (ConstraintLayout.LayoutParams.WRAP_CONTENT, 200);
```

2.10. Внутрішні та зовнішні відступи

Параметри розмітки дозволяють встановити відступи як від зовнішніх меж елемента до меж контейнера, так і всередині самого елемента між його межами та вмістом.

2.11. Padding

Для встановлення внутрішніх відступів використовується атрибут android:padding. Він встановлює відступи контенту всіх чотирьох сторін контейнера. Можна встановлювати відступи лише від однієї сторони контейнера, застосовуючи наступні атрибути: android:paddingLeft, android:paddingRight, android:paddingTop та android:paddingBottom.

```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.constraintlayout.widget.ConstraintLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50dp"
    tools:context=".MainActivity">

    <TextView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="Hello World!"
    android:textSize="30sp"
    android:background="#e0e0e0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    />

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

У контейнері ConstraintLayout встановлено лише один загальний внутрішній відступ 50 одиниць. Вкладений елемент TextView позиціонується у верхньому лівому куті контейнера (завдяки атрибутам`app:layout_constraintLeft_toLeftOf="parent"`і`app:layout_constraintTop_toTopOf="parent"`). . Тому TextView буде відсуватися від початкової точки (лівий верхній кут контейнера ConstraintLayout) вниз та ліворуч на 50 одиниць. Крім того, такі ж відступи діятимуть праворуч і знизу, якщо елемент примикатиме до нижньої або правої межі контейнера.

Встановлення одного відступу

```
    android:padding="50dp"
```

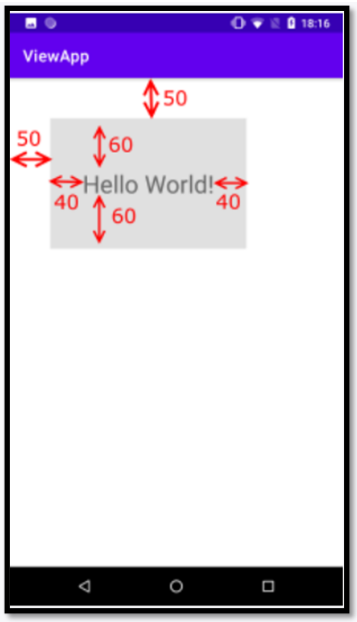
Буде аналогічна установці чотирьох відступів

```
android:paddingTop="50dp"
android:paddingLeft="50dp"
android:paddingBottom="50dp"
android:paddingRight="50dp"
```

Подібним чином можна встановити відступи інших елементів. Наприклад, встановимо всередині TextView зверху і знизу від внутрішнього вмісту (тобто тексту) відступи 60 одиниць і відступи ліворуч і праворуч 40 одиниць:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="50dp"
tools:context=".MainActivity">

<TextView
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:paddingTop="60dp"
android:paddingLeft="40dp"
android:paddingRight="40dp"
android:paddingBottom="60dp"
android:text="Hello World!"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>

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

Варто зазначити, що замість атрибутів`android:paddingLeft`і`android:paddingRight`можна застосовувати атрибути`android:paddingStart`і`android:paddingEnd`, які розроблені спеціально адаптації програми для роботи як для мов з лівосторонньою орієнтацією, так і правосторонньою орієнтацією (арабська, фарсі).



2.12. Margin

Для встановлення зовнішніх відступів використовується атрибут`layout_margin`. Цей атрибут має модифікації, які дозволяють задати відступ лише від однієї сторони:`android:layout_marginBottom`, `android:layout marginTop`, `android:layout_marginLeft`і`android:layout_marginRight`(відступи відповідно від нижньої, верхньої, лівої та правої меж):

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

<TextView
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:layout_marginTop="50dp"
android:layout_marginLeft="60dp"
android:text="Hello World!"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>

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

Тут у TextView задаються відступи від двох сторін ConstraintLayout (ліворуч 60 одиниць та зверху 50 одиниць):

2.13. Програмне встановлення відступів

Для програмної установки внутрішніх відступів елементи викликається метод setPadding(left, top, right, bottom), який передаються чотири значення кожної зі сторін. Також можна окремо задати відступи за допомогою методів getPaddingLeft(), getPaddingTop(), getPaddingRight() та getPaddingBottom().

Для встановлення зовнішніх відступів необхідно реалізувати об'єкт LayoutParams для контейнера, який застосовується. І потім викликати у цього об'єкта LayoutParams метод setMargins(left, top, right, bottom):

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
ConstraintLayout constraintLayout = New
ConstraintLayout(this);
TextView textView = New TextView(this);
// Встановлення кольору текстового поля
textView.setBackgroundColor(0xFFE0E0E0);
// Встановлення тексту текстового поля
textView.setText("Hello Android");
// Встановлення розміру тексту
textView.setTextSize(30);
```

```
     ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     // Встановлення зовнішніх відступів
     layoutParams.setMargins(60, 50, 60, 50);
     // позиціонування у верхньому лівому угду контейнера
     // еквівалент app:layout_constraintLeft_toLeftOf="parent"
     layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
     // еквівалент app:layout_constraintTop_toTopOf="parent"
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
     // встановлюємо розміри
     textView.setLayoutParams(layoutParams);
     // встановлення внутрішніх відступів
     textView.setPadding(40,40,40,40);
     // додаємо TextView у ConstraintLayout
     constraintLayout.addView(textView);
     setContentView(constraintLayout);
     }
     }
```

Оскільки в даному випадку елемент TextView додається до контейнеру типу ConstraintLayout, то для його позиціонування застосовується об'єкт ConstraintLayout.LayoutParams (відповідно для LinearLayout це буде LinearLayout.LayoutParams), у якого викликається метод setMargins().

Але якщо подивитися на останній скріншот, то можна побачити, що, незважаючи на те, що відступи як би задані також, що і в передостанньому прикладі у файлі layout, проте насправді на екрані ми побачимо відступи з зовсім іншими значеннями. Справа в тому, що методи`setPadding()`і`setMargins()`набувають значення в пікселях, тоді

як у файлі layout застосовувалися одиниці dp. І щоб використовувати dp також у коді, необхідно виконати перетворення:

```
     package com.example.viewapp;
     import androidx.appcompat.app.AppCompatActivity;
     import androidx.constraintlayout.widget.ConstraintLayout;
     import android.os.Bundle;
     import android.util.TypedValue;
     import android.widget.TextView;
     public class MainActivity extends AppCompatActivity {
     @Override
     protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     ConstraintLayout constraintLayout = New
ConstraintLayout(this);
     TextView textView = New TextView(this);
     textView.setBackgroundColor(0xFFE0E0E0);
     textView.setText("Hello Android!");
     textView.setTextSize(30);
     // отримуємо відступ у пікселях для 50 dp
     int margin50inDp = (int) TypedValue.applyDimension(
     TypedValue.COMPLEX_UNIT_DIP, 50,
getResources().getDisplayMetrics());
     // отримуємо відступ у пікселях для 60 dp
     int margin60inDp = (int) TypedValue.applyDimension(
     TypedValue.COMPLEX_UNIT_DIP, 60,
getResources().getDisplayMetrics());
     // отримуємо відступ у пікселях для 40 dp
     int padding40inDp = (int) TypedValue.applyDimension(
     TypedValue.COMPLEX_UNIT_DIP, 40,
getResources().getDisplayMetrics());
     ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     // Встановлення зовнішніх відступів
     layoutParams.setMargins(margin60inDp, margin50inDp,
margin60inDp, margin50inDp);
     // вирівнювання по лівому краю ConstraintLayout
     layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
     // Вирівнювання по верхньому кордоні ConstraintLayout
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
     // встановлюємо розміри
     textView.setLayoutParams(layoutParams);
     // встановлення внутрішніх відступів
     textView.setPadding(padding40inDp, padding40inDp,
padding40inDp, padding40inDp);
     // додаємо TextView у ConstraintLayout
```

```
    constraintLayout.addView(textView);

    setContentView(constraintLayout);
    }}
```

2.14. ConstraintLayout

ConstraintLayoutпредставляє контейнер, який дозволяє створювати гнучкі та масштабовані візуальні інтерфейси.

Для позиціонування елемента всередині ConstraintLayout необхідно вказати обмеження (constraints). Є кілька типів обмежень. Зокрема, для встановлення позиції щодо певного елемента використовують наступні обмеження:

- **layout_constraintLeft_toLeftOf**: ліва межа позиціонується щодо лівої межі іншого елемента
- **layout_constraintLeft_toRightOf**: ліва межа позиціонується щодо правої межі іншого елемента
- **layout_constraintRight_toLeftOf**: права межа позиціонується щодо лівої межі іншого елемента
- **layout_constraintRight_toRightOf**: правий кордон позиціонується щодо правого кордону іншого елемента
- **layout_constraintTop_toTopOf**: верхня межа позиціонується щодо верхньої межі іншого елемента
- **layout_constraintTop_toBottomOf**: верхня межа позиціонується щодо нижньої межі іншого елемента
- **layout_constraintBottom_toBottomOf**: нижня межа позиціонується щодо нижньої межі іншого елемента

- **layout_constraintBottom_toTopOf**: нижня межа позиціонується щодо верхньої межі іншого елемента
- **layout_constraintBaseline_toBaselineOf**: базова лінія позиціонується щодо базової лінії іншого елемента
- **layout_constraintStart_toEndOf**: елемент починається там, де завершується інший елемент
- **layout_constraintStart_toStartOf**: елемент починається там, де починається інший елемент
- **layout_constraintEnd_toStartOf**: елемент завершується там, де починається інший елемент
- **layout_constraintEnd_toEndOf**: елемент завершується там, де завершується інший елемент

Можливо, з приводу чотирьох останніх властивостей виникло деяке непорозуміння, що мається на увазі під керівництвом або завершенням елемента. Справа в тому, що деякі мови (наприклад, арабська або фарсі) мають правосторонню орієнтацію, тобто символи йдуть праворуч наліво, а не ліворуч, як у європейських мовах. І залежно від поточної орієнтації – лівостороння чи правостороння – змінюватиметься те, де саме початок, а де завершення елемента. Наприклад, при лівій орієнтації початок - ліворуч, а завершення - праворуч, тому атрибут layout_constraintStart_toEndOf фактично буде аналогічним атрибуту layout_constraintLeft_toRightOf. А при правосторонній орієнтації – атрибуту layout_constraintRight_toLeftOf, оскільки початок праворуч, а завершення – ліворуч

Кожне обмеження встановлює позиціонування елемента або по горизонталі або по вертикалі. І для визначення позиції елемента в ConstraintLayout необхідно вказати щонайменше одне обмеження по горизонталі та одне обмеження по вертикалі.

Позиціонування може проводитися щодо меж самого контейнера ContentLayout (у цьому випадку обмеження має значення parent), або щодо будь-якого іншого елемента всередині ConstraintLayout, тоді як значення обмеження вказується id цього елемента.

Найпростіший приклад:

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

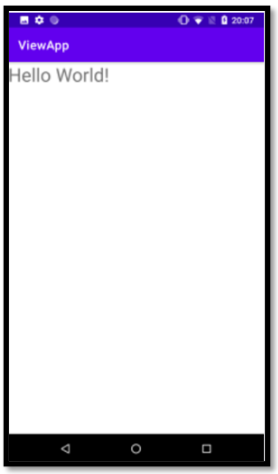
```
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="30sp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

В даному випадку елемент TextView має два обмеження: одне погоризонтальної лінії (app:layout_constraintLeft_toLeftOf="parent"), друге - по вертикальній лінії (app:layout_constraintTop_toTopOf="parent"). Обидва обмеження встановлюються щодо контейнера ConstraintLayout, тому вони набувають значення parent, тобто ConstraintLayout.

Обмеження app:layout_constraintLeft_toLeftOf="parent" встановлює ліву межу TextView біля лівої межі контейнера.

Обмеження app:layout_constraintTop_toTopOf="parent" встановлює верхню межу TextView у верхній межі контейнера.

У результаті TextView розташовуватиметься у верхньому лівому куті контейнера.



Варто звернути увагу, що всі ці атрибути обмежень беруться із простору імен "http://schemas.android.com/apk/res-auto", який проектується на префікс app.

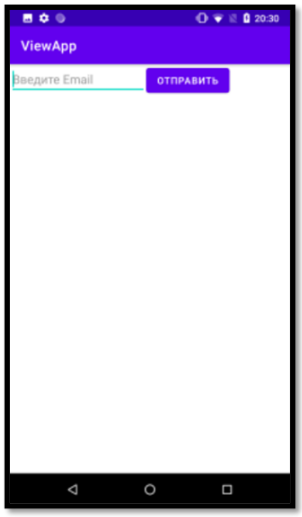
Якщо необхідно встановити обмеження щодо іншого елемента, необхідно вказати id цього елемента:

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

```
    <EditText
    android:id="@+id/editText"
    android:layout_width="180dp"
    android:layout_height="wrap_content"
    android:hint="Введіть Email"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
    <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Надіслати"
    app:layout_constraintLeft_toRightOf="@id/editText"
    app:layout_constraintTop_toTopOf="parent"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Тут для текстового поля введення EditText встановлюються два обмеження щодо батьківського контейнера ConstraintLayout, тому обмеження мають значення parent, а сам EditText вирівнюється по лівій та верхній межі контейнера. Верхня межа кнопки Button також вирівнюється на верхній межі контейнера. А ось ліва межа кнопки вирівнюється правою межею EditText. Для цього як значення атрибута передається id EditText:

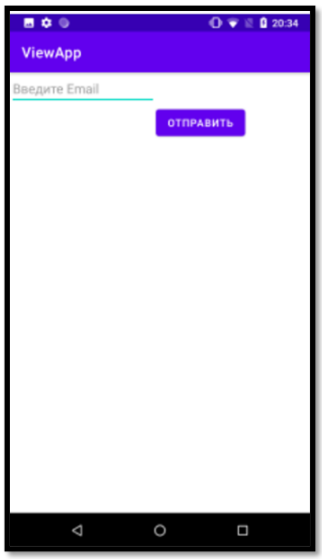
```
    app:layout_constraintLeft_toRightOf="@id/editText"
```



Подібним чином можна складати різні комбінації атрибутів визначення потрібного нам позиціонування. Наприклад, змінимо код кнопки:

```
    <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Надіслати"
    app:layout_constraintLeft_toRightOf="@id/editText"
    app:layout_constraintTop_toBottomOf="@id/editText"/>
```

В даному випадку верхня межа кнопки вирівнюється по нижній межі.



Більш того, ми можемо позиціонувати обидва елементи один щодо іншого:

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

<EditText
android:id="@+id/editText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="Введіть Email"
app:layout_constraintRight_toLeftOf="@+id/button"
app:layout_constraintTop_toTopOf="parent"/>
<Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Надіслати"
app:layout_constraintLeft_toRightOf="@id/editText"
app:layout_constraintTop_toTopOf="parent"/>

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

2.14.1.1. Позиціонування у центрі

Якщо необхідно розташувати елемент у центрі контейнера по вертикалі, треба використовувати пару атрибутів

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
```

Якщо необхідно розташувати елемент у центрі контейнера по горизонталі, треба використовувати наступну пару атрибутів

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
```

Відповідно для розташування в центрі контейнера по вертикалі та горизонталі треба застосувати всі вище вказані чотири атрибути:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android"
android:textSize="30sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>

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

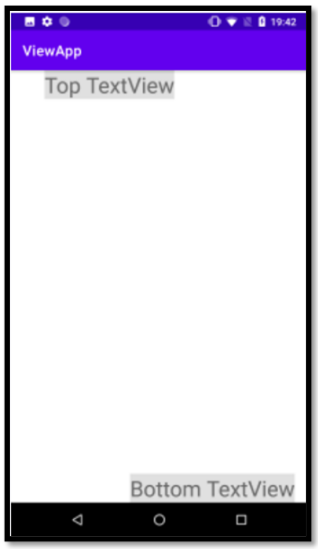
2.15. Зсув

Якщо елементи розташовані центром, ConstraintLayout дозволяє їх зрушувати по горизонталі і по вертикалі. Для зсуву по горизонталі застосовується атрибут layout_constraintHorizontal_bias, а зсуву по вертикалі - атрибут layout_constraintVertical_bias. Як значення вони приймають чис-

ло з плаваючою точкою від 0 до 1. Значення за замовчуванням – 0.5 (розташування по центру). Наприклад:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top TextView"
android:textSize="30sp"
android:background="#e0e0e0"

app:layout_constraintHorizontal_bias="0.2"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom TextView"
android:textSize="30sp"
android:background="#e0e0e0"

app:layout_constraintHorizontal_bias=".9"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>

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

Перший TextView зсувається на 20% від лівої межі контейнера (за замовчуванням значення - 0.5, тому при значенні 0.2 елемент зсувається вліво). Другий TextView зсувається на 90% від лівої межі контейнера. Наприклад, значення 1 означало б, що елемент присунутий до правої межі, а значення 0 - до лівої



Аналогічно працює атрибут layout_constraintVertical_bias, який зсуває по вертикалі.

2.16. Розміри елементів у ConstraintLayout

У ConstraintLayout застосовуються три способи встановлення розмірів:

- Встановлення точних розмірів, наприклад,`123dp`
- Значення WRAP_CONTENT, яке задає для віджету розміри, достатні для розташування вмісту
- Значення 0dp, яке еквівалентне значенню "MATCH_CONSTRAINT" у коді Java. У цьому випадку розміри елемента встановлюються, виходячи із зазначених для нього обмежень. За замовчуванням елемент займає весь доступний простір.

Застосуємо всі три типи установки розмірів:

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

<TextView
android:layout_width="160dp"
android:layout_height="wrap_content"
android:text="Top TextView"
android:textSize="30sp"
android:background="#e0e0e0"
```

```
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Center TextView"
    android:textSize="30sp"
    android:background="#e0e0e0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom TextView"
    android:textSize="30sp"
    android:background="#e0e0e0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>

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

Тут створюються три елементи TextView. Всі вони центруються по горизонталі, але по вертикалі розташовуються по верхній та нижній межі контейнера та в центрі. Для всіх трьох TextView для висоти встановлено значення`wrap_content`, тобто всі три елементи будуть займати ту висоту, яка для них краща, щоб вміст вмісту:

```
android:layout_height="wrap_content"
```

Однак для кожного елемента задані налаштування ширини. Для верхнього TextView встановлені точні розміри – 160 одиниць:

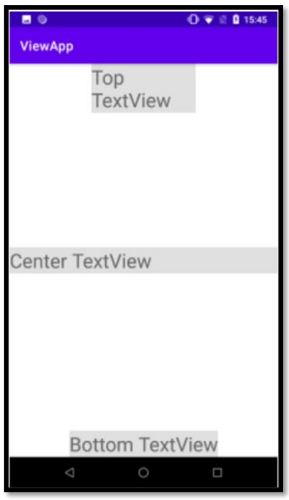
```
android:layout_width="160dp"
```

Для центрального TextView встановлено значення "0dp", завдяки чому елемент за замовчуванням займатиме весь доступний для нього простір (в даному випадку розтягуватиметься по горизонталі):

```
android:layout_width="0dp"
```

Для нижнього TextView встановлено значення "wrap_content", тобто елемент прийматиме ту ширину, яка необхідна для вміщення його вмісту:

```
android:layout_width="wrap_content"
```



Варто зазначити, що у вкладених віджетах ConstraintLayout не рекомендується використовувати значення match_parent, яке дозволяє віджету зайняти весь доступний простір. Натомість рекомендується використовувати 0dp або "MATCH_CONSTRAINT" - разом з іншими обмеженнями вони дадуть необхідний ефект. Так, для розтягування по ширині контейнера застосовують такі атрибути:

```
android:layout_width="0dp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
```

А для розтягування по висоті контейнера застосовують такі атрибути:

```
android:layout_height="0dp"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
```

Наприклад, розтягнення TextView по всій довжині та ширині контейнера:

```
<TextView
android:layout_width="0dp"
android:layout_height="0dp"
android:text="Hello Android"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>
```

Мінімальні та максимальні розміри

Ряд атрибутів задають максимальні та мінімальні розміри:

- **layout_constraintWidth_min**i layout_constraintHeight_min: представляють відповідно мінімальну ширину та висоту

- **layout_constraintWidth_max**і layout_constraintHeight_max: представляють відповідно максимальну ширину та висоту

Як значення вони набувають точного значення dp або значення wrap (аналогічно`wrap_content`). Наприклад:

```
<TextView
android:layout_width="260dp"
android:layout_height="wrap_content"
android:text="Hello Android"
android:textSize="30sp"
android:background="#e0e0e0"

app:layout_constraintHeight_max="200dp"
app:layout_constraintWidth_max="200dp"
app:layout_constraintHeight_min="wrap"
app:layout_constraintWidth_min="wrap"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>
```

Хоча в даному випадку ширина TextView встановлена в 260dp, оскільки максимальна ширина задана в 200dp, реальна ширина не перевищить 200dp.

Розміри у відсотках

Атрибут layout_constraintWidth_percent визначає ширину елемента у відсотках по відношенню до доступного простору по горизонталі. Аналогічно атрибут layout_constraintHeight_percent задає висоту у відсотках відносно доступного простору по вертикалі.

Для їх застосування необхідно дотримуватися таких умов:

- Відповідний атрибут для встановлення розміру (`android:layout width`- якщо ми встановлюємо ширину або`android:layout_height`- якщо ми встановлюємо у висоту у відсотках) повинен мати значення MATCH_CONSTRAINT або 0dp
- Також необхідно встановити атрибут`app:layout_constraintWidth_default="%"`при встановленні ширини та`app:layout_constraintHeight_default="%"`при встановленні висоти

Як атрибути layout_constraintWidth_percent і layout_constraintHeight_percent приймають дробове число від 0 до 1.

Наприклад, нехай TextView займає по вертикалі 25%, а по горизонталі 50% простору:

```
<?xml version="1.0" encoding="utf-8"?>
```

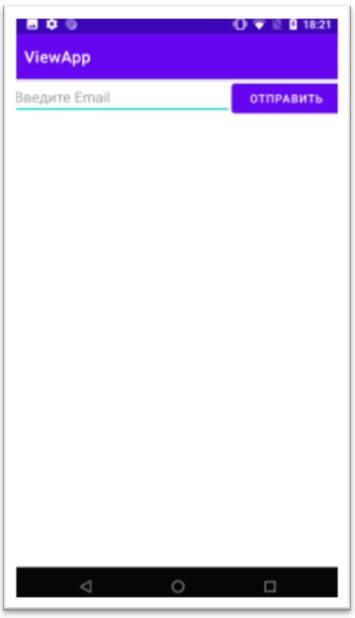
```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<TextView
android:layout_width="0dp"
android:layout_height="0dp"
android:text="Hello Android"
android:textSize="30sp"
android:background="#e0e0e0"

app:layout_constraintWidth_default="%"
app:layout_constraintHeight_default="%"

app:layout_constraintWidth_percent="0.5"
app:layout_constraintHeight_percent="0.25"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>

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

Інший приклад – пропорційний поділ простору між декількома елементами:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<EditText
android:id="@+id/editText"
android:hint="Введіть Email"
android:layout_height="wrap_content"

android:layout_width="0dp"
app:layout_constraintWidth_default="%"
app:layout_constraintWidth_percent="0.66"

app:layout_constraintRight_toLeftOf="@+id/button"
app:layout_constraintTop_toTopOf="parent" />
<Button
android:id="@+id/button"
android:text="Надіслати"
android:layout_height="wrap_content"

android:layout_width="0dp"
app:layout_constraintWidth_default="%"
app:layout_constraintWidth_percent="0.33"

app:layout_constraintLeft_toRightOf="@id/editText"
app:layout_constraintTop_toTopOf="parent" />

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

В даному випадку текстове поле EditText займатиме 66%, а кнопка – 33% ширини:



Встановлення співвідношення висоти та ширини

ConstraintLayout також дозволяє встановлювати у елементів висоту щодо ширини/ширину щодо висоти. Для цього використовується атрибут layout_constraintDimensionRatio. Як значення він набуває ставлення у вигляді`Width:Height`, наприклад,`1:0.5`- тут число 1 є шириною, а 0.5 - висотою. Тобто ширина буде вдвічі більша за висоту. Але при цьому хоча для одного виміру має бути встановлено`0dp`(`MATCH_CONSTRAINT`). Наприклад:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="0dp"
android:text="Hello Android"
android:textSize="30sp"
android:background="#e0e0e0"
app:layout_constraintDimensionRatio="1:0.6"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>

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

В даному випадку ширина TextView буде такою, яка необхідна для вмісту, а висота 60% від ширини.

Якщо і для ширини, і для висоти встановлено 0dp, то в цьому випадку система вибере найбільший вимір, який відповідає всім обмеженням і щодо нього встановить значення іншого виміру. Щоб конкретизувати вимірювання, щодо якого йтиме розрахунок, можна вказати символ W (ширина) або H (висота). Наприклад:

```
<TextView
android:layout_width="0dp"
android:layout_height="0dp"
android:text="Hello Android"
android:textSize="30sp"
android:background="#e0e0e0"

app:layout_constraintDimensionRatio="W, 1:4"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"/>
```

В даному випадку ширина буде в 4 рази менша за висоту.

2.17. Ланцюжки елементів в ConstraintLayout

ConstraintLayout дозволяє організувати розташування елементів у ряд по горизонталі або по вертикалі або те, що в Android називається chains або ланцюжки. Ми можемо по ланцюжку встановити позиціонування одного елемента щодо іншого і таким чином організувати ряд елементів.

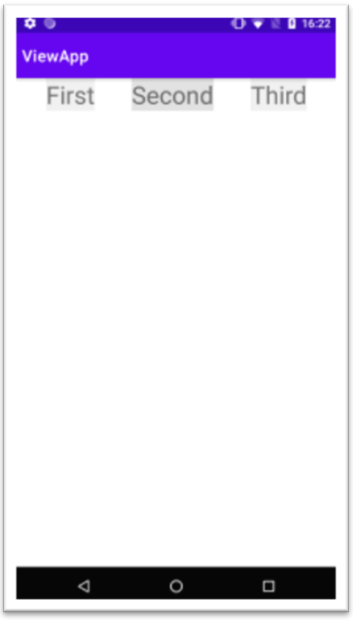
Горизонтальний ланцюжок елементів

Наприклад, ряд елементів по горизонталі:

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

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#efefef"
android:text="First"
android:textSize="30sp"
app:layout_constraintRight_toLeftOf="@id/textView2"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
```

```
    <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

В результаті елементи ланцюжка поступово будуть розтягнуті по всій ширині контейнера:



Горизонтальний ланцюжок елементів досягається за рахунок двох факторів:

- Перший елемент вирівнюється щодо лівої межі контейнера (app:layout_constraintLeft_toLeftOf="parent"), останній елемент

вирівнюється щодо правої межі контейнера (app:layout_constraintRight_toRightOf="parent").

- Завдяки установці атрибутів app:layout_constraintLeft_toRightOf і app:layout_constraintRight_toLeftOf маємо один елемент праворуч або ліворуч від іншого.

Крім того, ConstraintLayout дозволяє настроїти положення елементів усередині ланцюжка. Для цього застосовується атрибут layout_constraintHorizontal_chainStyle, який може приймати такі значення:

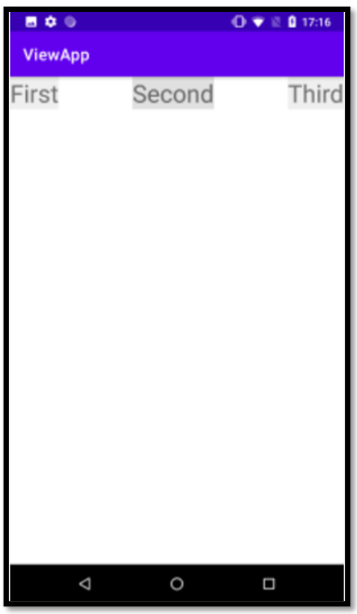
- **spread**: значення за замовчуванням, при якому елементи ланцюжка рівномірно розтягуються по всій довжині ланцюжка, як у прикладі вище
- **spread_inside**: перший та останній елемент ланцюжка примикають до меж контейнера
- **packed**: елементи ланцюжка розташовуються впритул один до одного.

Наприклад, застосуємо значення spread_inside:

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

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#efefef"
android:text="First"
android:textSize="30sp"

app:layout_constraintHorizontal_chainStyle="spread_inside"

app:layout_constraintRight_toLeftOf="@id/textView2"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:id="@+id/textView2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

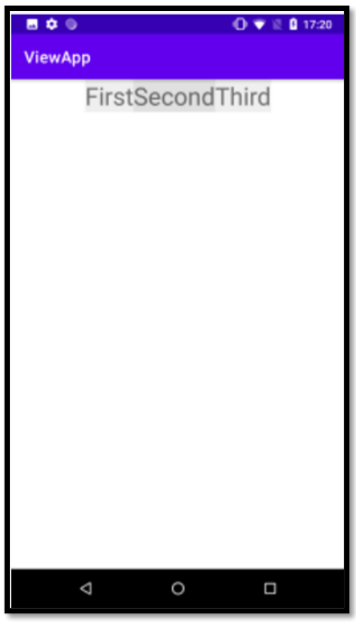
Причому в даному випадку достатньо встановити атрибут у першого елемента ланцюжка:



Значення packed:

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

    <TextView
```

```
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintHorizontal_chainStyle="packed"

    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

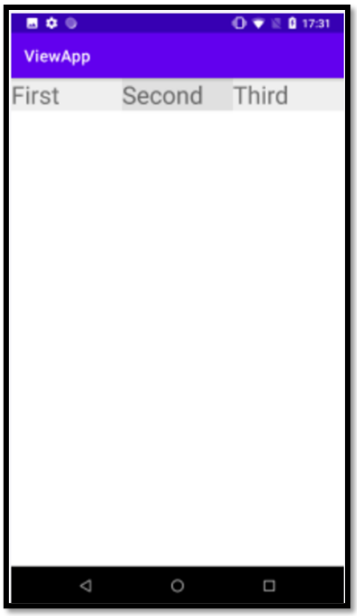


Вага елемента

Варто відзначити, що вище у елементів встановлювалася ширина, необхідна їхнього вмісту. Але ми могли б встановити і нульову ширину, тоді елементи рівномірно розподілялися б по всьому ланцюжку без утворення проміжків між ними.

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

      <TextView
      android:id="@+id/textView1"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:background="#efefef"
      android:text="First"
      android:textSize="30sp"
      app:layout_constraintRight_toLeftOf="@id/textView2"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintTop_toTopOf="parent"/>

      <TextView
      android:id="@+id/textView2"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:background="#e0e0e0"
      android:text="Second"
      android:textSize="30sp"
```

```
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView3"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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



У цьому випадку значення атрибуту app:layout_constraintHorizontal_chainStyle не відіграє жодної ролі, тому що всі елементи таким чином розтягуються по всьому ланцюжку.

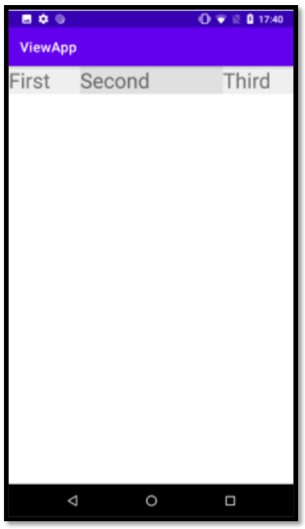
Однак така поведінка може не влаштовувати, наприклад, ми хочемо, щоб один елемент був двічі більший за інший. І тут ми можемо за допомогою атрибута layout_constraintHorizontal_weight. Однак слід враховувати, що при застосуванні ваг у елементів вони повинні мати нульову ширину:

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

```
    tools:context=".MainActivity">

    <TextView
    android:id="@+id/textView1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintHorizontal_weight="1"

    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"

    app:layout_constraintHorizontal_weight="2"

    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/textView3"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"

    app:layout_constraintHorizontal_weight="1"

    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

Як атрибут layout_constraintHorizontal_weight приймає число - вага елемента. Так, в даному випадку вага першого елемента - 1, вага другого - 2, а вага третього - 1. Тому вся ширина контейнера буде умовно поділе-

на на 1 + 2 + 1 = 4 частин, з яких по одній частині займуть перший і третій елемент, а другий займе 2 частини, тобто другий елемент буде вдвічі більше за перший і третій елемент.



В принципі ми можемо залишити елементи і з шириною "wrap_content" або конкретним значенням, відмінним від "0dp", просто в цьому випадку вони не будуть брати участь у розподілі простору контейнера і вага такого елемента ролі не гратиме.

2.18. Вертикальний ланцюжок

Для утворення вертикального ланцюжка також має дотримуватися дві умови:

- Перший елемент вирівнюється відносно верхньої межі контейнера (app:layout_constraintTop_toTopOf="parent"), останній елемент вирівнюється відносно нижньої межі контейнера (app:layout_constraintBottom_toBottomOf="parent").
- Завдяки установці атрибутів app:layout_constraintBottom_toTopOf та app:layout_constraintBottom_toTopOf розташовуємо один елемент поверх іншого.

Щоб настроїти положення елементів усередині ланцюжка, застосовується атрибут layout_constraintVertical_chainStyle, який може приймати такі значення:

- **spread**: значення за замовчуванням, при якому елементи ланцюжка рівномірно розтягуються по всій довжині ланцюжка
- **spread_inside**: перший та останній елемент ланцюжка примикають до меж контейнера
- **packed**: елементи ланцюжка прилягають впритул один до одного.

Наприклад, вертикальний ланцюжок зі значенням за замовчуванням - spread:

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

      <TextView
      android:id="@+id/textView1"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:background="#efefef"
      android:text="First"
      android:textSize="30sp"
      app:layout_constraintBottom_toTopOf="@id/textView2"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
                              app:layout_constraintTop_toTopOf
                        ="parent"/>

      <TextView
      android:id="@+id/textView2"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:background="#e0e0e0"
      android:text="Second"
      android:textSize="30sp"
      app:layout_constraintTop_toBottomOf="@id/textView1"
      app:layout_constraintBottom_toTopOf="@id/textView3"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"/>

      <TextView
      android:id="@+id/textView3"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:background="#efefef"
      android:text="Third"
      android:textSize="30sp"

      app:layout_constraintTop_toBottomOf="@id/textView2"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintBottom_toBottomOf="parent"/>

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

Також достатньо застосувати до першого елемента ланцюжка атрибут layout_constraintVertical_chainStyle, щоб змінити положення елементів:

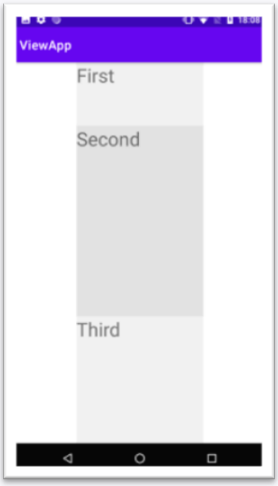
```
    <TextView
    android:id="@+id/textView1"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintVertical_chainStyle="spread_inside"

    app:layout_constraintBottom_toTopOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

І як при горизонтальній орієнтації у вертикальному ланцюжку можна використовувати вагу елементів за допомогою атрибута layout_constraintVertical_weight. Для встановлення ваги у елемента як висота має бути встановлено значення 0dp

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

<TextView
android:id="@+id/textView1"
android:layout_width="200dp"
android:layout_height="0dp"
android:background="#efefef"
android:text="First"
android:textSize="30sp"
app:layout_constraintVertical_weight="1"
app:layout_constraintBottom_toTopOf="@id/textView2"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:id="@+id/textView2"
android:layout_width="200dp"
android:layout_height="0dp"
android:background="#e0e0e0"
```

```
     android:text="Second"
     android:textSize="30sp"
     app:layout_constraintVertical_weight="3"
     app:layout_constraintTop_toBottomOf="@id/textView1"
     app:layout_constraintBottom_toTopOf="@id/textView3"
     app:layout_constraintLeft_toLeftOf="parent"
     app:layout_constraintRight_toRightOf="parent"/>

     <TextView
     android:id="@+id/textView3"
     android:layout_width="200dp"
     android:layout_height="0dp"
     android:background="#efefef"
     android:text="Third"
     android:textSize="30sp"
     app:layout_constraintVertical_weight="2"
                          app:layout_constraintTop_toBottomOf="
@id/textView2"
     app:layout_constraintLeft_toLeftOf="parent"
     app:layout_constraintRight_toRightOf="parent"
     app:layout_constraintBottom_toBottomOf="parent"/>

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

Сукупна вага елементів у даному випадку 1 + 3 + 2 = 6. Тому вся висота контейнера буде ділитися на 6 частин, з яких перший елемент займе 1 частину, другий – 3 частини та третій – 2 частини відповідно до своєї ваги.



2.19. Програмне створення ConstraintLayout та позиціонування

Для створення контейнера в коді Java застосовується однойменний клас ConstraintLayout, для створення об'єкта якого конструктор передаються значення для ширини і висоти елемента:

```
     ConstraintLayout.LayoutParams     layoutParams     =     new
ConstraintLayout.LayoutParams
```

```
    (ConstraintLayout.LayoutParams.WRAP_CONTENT                    ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
```

Перший параметр встановлює ширість елемента, а другий - висоту.`ConstraintLayout.LayoutParams.WRAP_CONTENT`вказує, що елемент матиме ті розміри, які потрібні для того, щоб вивести на екран його вміст. Крім`ConstraintLayout.LayoutParams.WRAP_CONTENT`можна застосовувати константу`ConstraintLayout.LayoutParams.MATCH_CONSTRAINT`, яка аналогічна до застосування значення "0dp" в атрибутах`layout width`і`layout height`і яка розтягує елемент за шириною або висотою контейнера.

Також можна використовувати точні розміри, наприклад:

```
    ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
    (ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, 200);
```

Для налаштування позиціонування всередині ConstraintLayout застосовується клас ConstraintLayout.LayoutParams. Він має багато функціоналу. Розглянемо у разі лише ті поля, які дозволяють встановити розташування елемента:

- **baselineToBaseline**: вирівнює базову лінію елемента базової лінії іншого елемента, id якого присвоюється властивості.
- **bottomToBottom**: вирівнює нижню межу елемента на нижній межі іншого елемента.
- **bottomToTop**: вирівнює нижню межу елемента верхньої межі іншого елемента.
- **leftToLeft**: вирівнює ліву межу елемента по лівій межі іншого елемента
- **leftToRight**: вирівнює ліву межу елемента з правого кордону іншого елемента
- **rightToLeft**: вирівнює праву межу елемента по лівій межі іншого елемента
- **rightToRight**: вирівнює праву межу елемента з правого кордону іншого елемента
- **startToEnd**: вирівнює початок елемента після завершення іншого елемента.
- **startToStart**: вирівнює початок елемента на початку іншого елемента.

- **topToBottom**: вирівнює верхню межу елемента нижньої межі іншого елемента.
- **topToTop**: вирівнює верхню межу елемента верхньої межі іншого елемента.
- **endToEnd**: вирівнює завершення елемента після завершення іншого елемента.
- **endToStart**: вирівнює завершення елемента на початку іншого елемента.

Як значення ці поля приймають ID (ідентифікатор) елемента, щодо якого виконується позиціонування. Якщо розташування встановлюється щодо контейнера ConstraintLayout, застосовується константа ConstraintLayout.LayoutParams.PARENT_ID

Розглянемо найпростіший приклад:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

ConstraintLayout constraintLayout = New
ConstraintLayout(this);
TextView textView = New TextView(this);
// Встановлення тексту текстового поля
textView.setText("Hello Android");
// Встановлення розміру тексту
textView.setTextSize(30);

ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
// позиціонування у верхньому лівому куті контейнера
// еквівалент app:layout_constraintLeft_toLeftOf="parent"
layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
// еквівалент app:layout_constraintTop_toTopOf="parent"
layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
```

```
_ID;
      // встановлюємо розміри
      textView.setLayoutParams(layoutParams);
      // додаємо TextView у ConstraintLayout
      constraintLayout.addView(textView);

      setContentView(constraintLayout);
      }
      }
```

У цьому випадку значення`ConstraintLayout.LayoutParams.WRAP_CONTENT`для ширини і висоти вказує, що елемент матиме ті розміри, які потрібні для того, щоб вивести на екран його вміст.

Далі вирівнюємо ліву межу елемента з лівого боку контейнера:

```
      layoutParams.leftToLeft                                    =
ConstraintLayout.LayoutParams.PARENT_ID;
```

Ця установка аналогічна використанню атрибуту`app:layout_constraintLeft_toLeftOf="parent"`.

Потім вирівнюємо верхню межу елемента по верхній стороні контейнера:

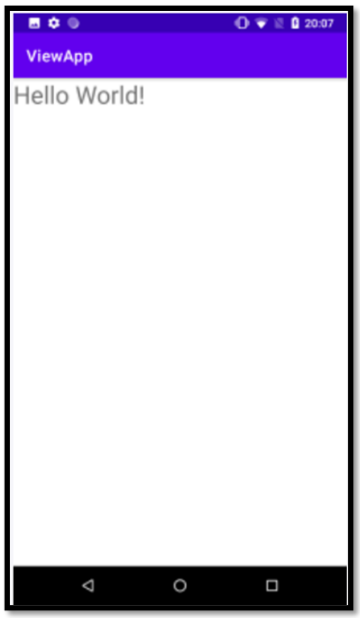
```
      layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
```

Ця установка аналогічна використанню атрибуту`app:layout_constraintTop_toTopOf="parent"`.

І насамкінець застосовуємо об'єкт ConstraintLayout.LayoutParams до TextView:

```
      constraintLayout.addView(textView);
```

В результаті елемент TextView буде розташований у верхньому лівому куті ConstraintLayout:



Розглянемо інший приклад - встановлення розташування елементів щодо один одного:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

ConstraintLayout constraintLayout = New
ConstraintLayout(this);

EditText editText = New EditText(this);
editText.setHint("Введіть Email");
editText.setId(View.generateViewId());

button button = new Button(this);
button.setText("Надіслати");
button.setId(View.generateViewId());

ConstraintLayout.LayoutParams editTextLayout = new
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.WRAP_CONTENT ,
```

```
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     editTextLayout.leftToLeft                                    =
ConstraintLayout.LayoutParams.PARENT_ID;
     editTextLayout.topToTop                                      =
ConstraintLayout.LayoutParams.PARENT_ID;
     editTextLayout.rightToLeft = button.getId();
     editText.setLayoutParams(editTextLayout);
     constraintLayout.addView(editText);

     ConstraintLayout.LayoutParams    buttonLayout    =    new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT                  ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     buttonLayout.leftToRight = editText.getId();
     buttonLayout.topToTop                                        =
ConstraintLayout.LayoutParams.PARENT_ID;
     button.setLayoutParams(buttonLayout);
     constraintLayout.addView(button);

     setContentView(constraintLayout);
     }
     }
```

При розташуванні одного елемента щодо іншого, нам потрібно знати id другого елемента. Якщо елемент визначено в коді Java, спочатку треба згенерувати ідентифікатор:

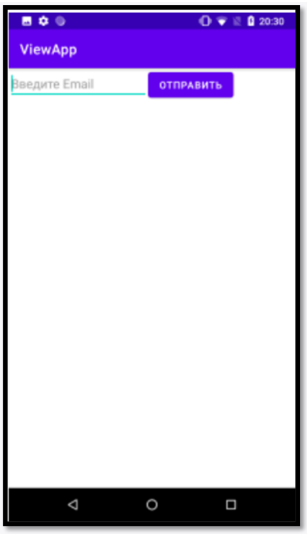
```
     editText.setId(View.generateViewId());
     button.setId(View.generateViewId());
```

Потім можна застосовувати ідентифікатори елементів для встановлення позиціонування. Так, права межа EditText вирівнюється по лівій межі кнопки:

```
     editTextLayout.rightToLeft = button.getId();
```

А ліва межа кнопки вирівнюється правою межею елемента EditText:

```
     buttonLayout.leftToRight = editText.getId();
```



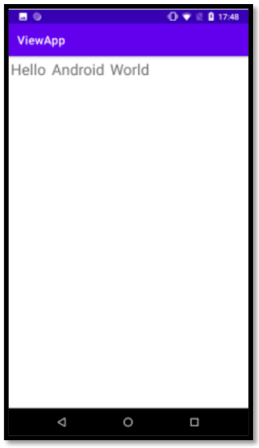
LinearLayout

Контейнер LinearLayout представляє найпростіший контейнер – об'єктViewGroup, який упорядковує всі дочірні елементи в одному напрямку: по горизонталі або по вертикалі. Всі елементи розташовані один за одним. Напрямок розмітки вказується за допомогою атрибута android:orientation.

Якщо, наприклад, орієнтація розмітки вертикальна (android:orientation="vertical"), то всі елементи розташовуються в стовпчику - по одному елементу на кожному рядку. Якщо орієнтація горизонтальна (android:orientation="horizontal"), то елементи розташовуються в один рядок. Наприклад, розташуємо елементи у горизонтальний ряд:

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

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:text="Hello"
    android:textSize="26sp" />
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:text="Android"
    android:textSize="26sp" />
    <TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:text="World"
    android:textSize="26sp" />
    </LinearLayout>
```



Якби ми вказали для LinearLayout атрибут android:orientation="vertical", то елементи розміщувалися по вертикалі:

2.20. **Вага елемента**

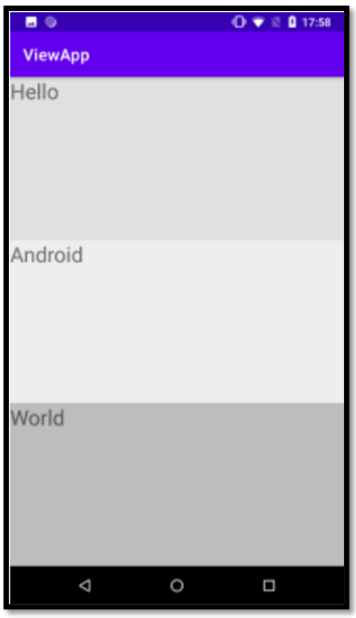
LinearLayout підтримує таку властивість, як вага елемента, який передається атрибутом android:layout_weight. Ця властивість набуває значення, що вказує, яку частину вільного місця контейнера, що залишився, по відношенню до інших об'єктів займе даний елемент. Наприклад, якщо один елемент у нас матиме властивість`android:layout_weight`значення 2, а інший - значення 1, то в сумі вони дадуть 3, тому перший елемент займатиме 2/3 простору, що залишився, а другий - 1/3.

Якщо всі елементи мають значення`android:layout_weight="1"`, всі ці елементи будуть рівномірно розподілені по всій площі контейнера:

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:text="Hello"
    android:background="#e0e0e0"
    android:layout_weight="1"
    android:textSize="26sp" />
    <TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:background="#eeeeee"
    android:text="Android"
    android:layout_weight="1"
    android:textSize="26sp" />
    <TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:text="World"
    android:background="#bdbdbd"
    android:layout_weight="1"
    android:textSize="26sp" />
    </LinearLayout>
```

В даному випадку LinearLayout має вертикальну орієнтацію, тому всі елементи розташовуватимуться зверху вниз. Усі три елементи мають значення`android:layout_weight="1"`, тому сума ваги всіх елементів дорівнюватиме 3, а кожен елемент отримає по третині простору в LinearLayout:



При цьому так як у нас вертикальний стек, то треба також встановити для властивості`layout_height`значення 0dp. Якби

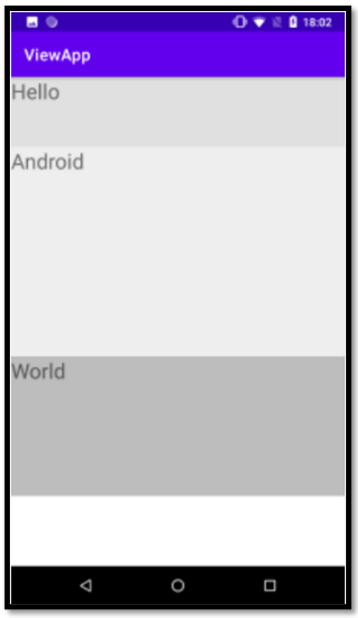
LinearLayout мав горизонтальну орієнтацію, то для якості`layout_width`треба було б встановити значення 0dp.

Ще один атрибут android:weightSum дозволяє вказати суму ваги всіх елементів. Наприклад:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:weightSum="7">

<TextView
android:layout_width="match_parent"
android:layout_height="0dp"
android:text="Hello"
android:background="#e0e0e0"
android:layout_weight="1"
android:textSize="26sp" />
<TextView
android:layout_width="match_parent"
android:layout_height="0dp"
android:background="#eeeeee"
android:text="Android"
android:layout_weight="3"
android:textSize="26sp" />
<TextView
android:layout_width="match_parent"
android:layout_height="0dp"
android:text="World"
android:background="#bdbdbd"
android:layout_weight="2"
android:textSize="26sp" />
</LinearLayout>
```

LinearLayout тут задає суму ваг рівну 7. Тобто весь простір по вертикалі (оскільки вертикальна орієнтація) умовно ділиться на сім рівних частин.

Перший TextView має вагу 1, тобто із цих семи частин займає лише одну. Другий TextView має вагу 3, тобто займає три частини із семи. І третій має вагу 2. Підсумкова сума становить 6. Але оскільки LinearLayout задає вагу 7, то одна частина буде вільна від усіх елементів.

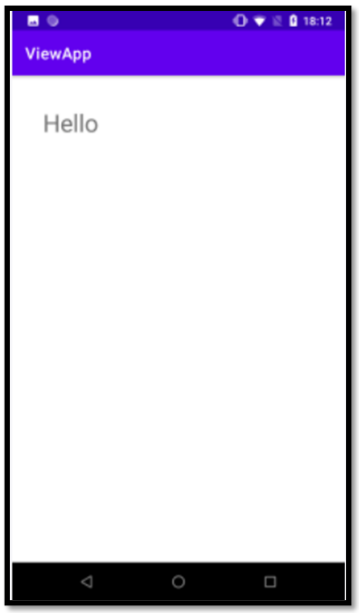


2.21. **Програмне створення LinearLayout**

Створення LinearLayout у коді java:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_main);
LinearLayout linearLayout = New LinearLayout(this);
// горизонтальна орієнтація
linearLayout.setOrientation(LinearLayout.HORIZONTAL);

TextView textView = New TextView(this);
textView.setText("Hello");
textView.setTextSize(30);
// створюємо параметри позиціонування елемента
LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
// встановлюємо відступи
layoutParams.setMargins(100, 100, 0, 0);
textView.setLayoutParams(layoutParams);
// Додаємо елемент у LinearLayout
linearLayout.addView(textView);
```

```
        setContentView(linearLayout);}}
```



Додаткова версія конструктора LinearLayout.LayoutParams() як третій параметр дозволяє вказати вагу елемента:

```
        package com.example.viewapp;

        import androidx.appcompat.app.AppCompatActivity;
        import android.os.Bundle;
        import android.widget.LinearLayout;
        import android.widget.TextView;

        public class MainActivity extends AppCompatActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout linearLayout = New LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);

        // перше текстове поле
        TextView textView1 = New TextView(this);
        textView1.setText("Hello");
        textView1.setTextSize(30);
        // textView1 має вагу 3
        linearLayout.addView(textView1,                                    new
LinearLayout.LayoutParams
        (LinearLayout.LayoutParams.MATCH_PARENT, 0, 3));

        // Друге текстове поле
        TextView textView2 = New TextView(this);
        textView2.setText("Android");
        textView2.setBackgroundColor(0xFFBDBDBD);
```

```
      textView2.setTextSize(30);
      // textView2 має вагу 2
      linearLayout.addView(textView2,                                  new
LinearLayout.LayoutParams
      (LinearLayout.LayoutParams.MATCH_PARENT, 0, 2));

      setContentView(linearLayout);
      }
      }
```

2.22. **Layout_gravity**

Атрибут layout_gravity дозволяє встановити позиціонування щодо LinearLayout. Він приймає такі значення:

- `top`: вирівнює елемент верхньої межі контейнера
- `bottom`: вирівнює елемент нижньої межі контейнера
- `left`: вирівнює елемент по лівій межі контейнера
- `right`: вирівнює елемент з правого кордону контейнера
- `center_vertical`: вирівнює елемент по центру по вертикалі
- `center_horizontal`: вирівнює елемент по центру по горизонталі.
- `center`: елемент позиціонується в центрі
- `fill_vertical`: елемент розтягується по вертикалі
- `fill_horizontal`: елемент розтягується по горизонталі.
- `fill`: елемент заповнює весь простір контейнера
- `clip_vertical`: обрізає верхню та нижню межу елемента

- `clip_horizontal`: обрізає праву та ліву межу елемента
- `start`: елемент позиціонується на початку (у верхньому лівому куті) контейнера
- `end`: елемент розташований у кінці контейнера (у верхньому правому куті)

Наприклад:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">
<TextView
android:layout_gravity="left"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30sp"
android:text="Hello Java!"
android:background="#e8eaf6"/>
<TextView
android:layout_gravity="center"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30sp"
android:text="Hello World!"
android:background="#e8eaf6"/>
<TextView
android:layout_gravity="right"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30sp"
android:text="Hello Android!"
android:background="#e8eaf6"/>
<TextView
android:layout_gravity="center"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30sp"
android:text="Hello Kotlin!"
android:background="#e8eaf6"/>
</LinearLayout>
```

В даному випадку перший елемент TextView позиціонуватиметься по лівій стороні контейнера (`android:layout_gravity="left"`), другий TextView по центру (`android:layout_gravity="center"`), третій - по правій стороні (`android:layout_gravity="right"`) та четвертий - по центру (`android:layout_gravity="center"`)

Варто враховувати орієнтацію контейнера. Наприклад, при вертикальній орієнтації всі елементи будуть представляти вертикальний стек, що йде зверху донизу. Тому значення, які відносяться до позиціонування елемента по вертикалі (наприклад, top або bottom), ніяк не впливатимуть на елемент. Також при горизонтальній орієнтації LinearLayout не вплинуть значення, які позиціонують елемент по горизонталі, наприклад, left і right.

Для встановлення програмного параметра layout_gravity треба задати поле gravity у об'єкта LinearLayout.LayoutParams:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Gravity;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

LinearLayout linearLayout = New LinearLayout(this);
linearLayout.setOrientation(LinearLayout.VERTICAL);
LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
// встановлення layout_gravity
layoutParams.gravity = Gravity.CENTER;
```

```
// перше текстове поле
TextView textView1 = New TextView(this);
textView1.setText("Hello");
textView1.setTextSize(30);
linearLayout.addView(textView1, layoutParams);
setContentView(linearLayout);
}
}
```

Як значення передається одна із констант класу Gravity, які аналогічні значенням атрибута.

2.23. **RelativeLayout**

RelativeLayoutпредставляє об'єкт`ViewGroup`, який має дочірні елементи щодо позиції інших дочірніх елементів розмітки або щодо області самої розмітки RelativeLayout. Використовуючи відносне позиціонування, ми можемо встановити елемент праворуч або в центрі або іншим способом, який надає цей контейнер. Для встановлення елемента у файлі xml ми можемо використовувати такі атрибути:

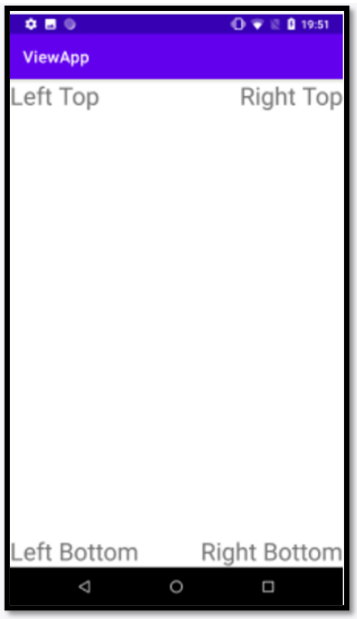
- **android:layout_above**: має елемент над елементом із зазначеним Id
- **android:layout_below**: має елемент під елементом із зазначеним Id
- **android:layout_toLeftOf**: розташовується ліворуч від елемента із зазначеним Id
- **android:layout_toRightOf**: розташований праворуч від елемента із зазначеним Id
- **android:layout_toStartOf**: має початок поточного елемента, де починається елемент із зазначеним Id
- **android:layout_toEndOf**: має початок поточного елемента, де завершується елемент із зазначеним Id
- **android:layout_alignBottom**: вирівнює елемент по нижній межі іншого елемента із зазначеним Id
- **android:layout_alignLeft**: вирівнює елемент по лівій межі іншого елемента із зазначеним Id
- **android:layout_alignRight**: вирівнює елемент по правій межі іншого елемента із зазначеним Id
- **android:layout_alignStart**: вирівнює елемент по лінії, у якої починається інший елемент із зазначеним Id

- **android:layout_alignEnd**: вирівнює елемент по лінії, у якій завершується інший елемент із зазначеним Id
- **android:layout_alignTop**: вирівнює елемент по верхній межі іншого елемента із зазначеним Id
- **android:layout_alignBaseline**: вирівнює базову лінію елемента по базовій лінії іншого елемента із зазначеним Id
- **android:layout_alignParentBottom**: якщо атрибут має значення true, елемент притискається до нижньої межі контейнера
- **android:layout_alignParentRight**: якщо атрибут має значення true, елемент притискається до правого краю контейнера
- **android:layout_alignParentLeft**: якщо атрибут має значення true, елемент притискається до лівого краю контейнера
- **android:layout_alignParentStart**: якщо атрибут має значення true, елемент притискається до початкового краю контейнера (при лівій орієнтації тексту - лівий край)
- **android:layout_alignParentEnd**: якщо атрибут має значення true, елемент притискається до кінцевого краю контейнера (при лівій орієнтації тексту - правий край)
- **android:layout_alignParentTop**: якщо атрибут має значення true, елемент притискається до верхньої межі контейнера
- **android:layout_centerInParent**: якщо атрибут має значення true, то елемент розташований у центрі батьківського контейнера
- **android:layout_centerHorizontal**: при значенні true вирівнює елемент по центру горизонталі
- **android:layout_centerVertical**: при значенні true вирівнює елемент по вертикалі центру

Наприклад, позиціонування щодо контейнера RelativeLayout:

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

<TextView android:text="Left Top"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:textSize="26sp"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true" />
```

```
    <TextView android:text="Right Top"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true" />

    <TextView android:text="Left Bottom"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:layout_alignParentLeft="true"
    android:layout_alignParentBottom="true" />

    <TextView android:text="Right Bottom"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true" />
    </RelativeLayout>
```

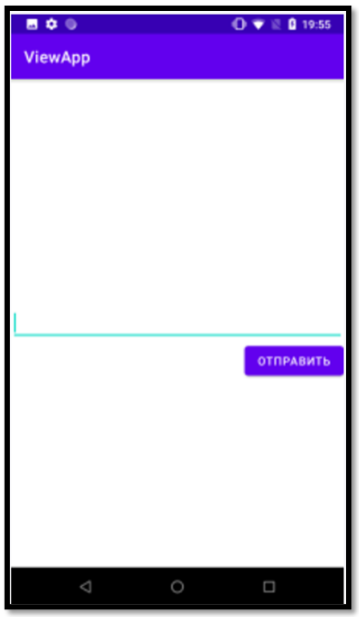


Для позиціонування щодо іншого елемента нам треба вказати id цього елемента. Так, помістимо на RelativeLayout текстове поле та кнопку:

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

    <EditText
```

```
android:id="@+id/edit_message"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Надіслати"
android:layout_alignRight="@id/edit_message"
android:layout_below="@id/edit_message"
/>
</RelativeLayout>
```

В даному випадку поле EditText розташовується по центру RelativeLayout, а кнопка поміщається під EditText і вирівнюється по його правій межі:



Програмне створення RelativeLayout

Створимо елемент RelativeLayout програмно в коді Java:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RelativeLayout;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

RelativeLayout relativeLayout = New RelativeLayout(this);
```

```
     EditText editText = New EditText(this);
     editText.setId(EditText.generateViewId());

     button button = new Button(this);
     button.setText("Надіслати");

     // встановлюємо параметри для EditText
     RelativeLayout.LayoutParams    editTextParams    =    New
RelativeLayout.LayoutParams(
     RelativeLayout.LayoutParams.MATCH_PARENT,
     RelativeLayout.LayoutParams.WRAP_CONTENT
     );
     // Вирівнювання по центру батьківського контейнера
     editTextParams.addRule(RelativeLayout.CENTER_IN_PARENT);
     // додаємо в RelativeLayout
     relativeLayout.addView(editText, editTextParams);

     // встановлюємо параметри положення Button
     RelativeLayout.LayoutParams                buttonParams=new
RelativeLayout.LayoutParams(
     RelativeLayout.LayoutParams.WRAP_CONTENT,
     RelativeLayout.LayoutParams.WRAP_CONTENT
     );
     // вирівнювання праворуч та знизу від поля EditText
     buttonParams.addRule(RelativeLayout.BELOW,
editText.getId());
     buttonParams.addRule(RelativeLayout.ALIGN_RIGHT,
editText.getId());
     // додаємо в RelativeLayout
     relativeLayout.addView(button, buttonParams);

     setContentView(relativeLayout);
     }
     }
```

Щоб встановити положення елемента в контейнері, застосовується клас RelativeLayout.LayoutParams. Через конструктор встановлюються значення для ширини та висоти. Наприклад, елемент EditText для ширини встановлює значення`MATCH_PARENT`, а для висоти -`WRAP_CONTENT`.

За допомогою методу addRule() ми можемо додавати додаткові правила позиціонування елемента. Цей метод як параметр приймає числову константу, яка представляє параметр позиціонування і яка аналогічна атрибуту. Наприклад, атрибуту`android:layout centerInParent`буде відповідати константа`CENTER_IN_PARENT`, а атрибуту`android:layout_alignRight`константа`ALIGN_RIGHT`.

Варто зазначити, що для спрощення коду для установки id у EditText викликається метод generateViewId();, який дозволяє програмно згенерувати id для елемента управління.

Потім встановлений id передається як другий параметр метод addRule при установці правил для кнопки:

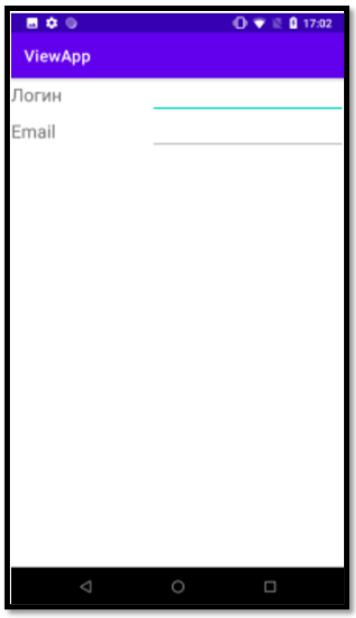
```
    buttonParams.addRule(RelativeLayout.BELOW,
editText.getId());
```

Тим самим ми вказуємо щодо якого елемента треба задати розташування.

TableLayout

Контейнер TableLayout структурує елементи керування як таблиці по стовпчикам і рядкам. Визначимо у файлі activity_main.xml елемент TableLayout, який включатиме два рядки та два стовпці:

```
    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
    <TextView
    android:layout_weight="0.5"
    android:text="Логін"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

    <EditText
    android:layout_weight="1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
    </TableRow>

    <TableRow>
    <TextView
    android:layout_weight="0.5"
    android:text="Email"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

    <EditText
    android:layout_weight="1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
    </TableRow>
    </TableLayout>
```



Використовуючи елемент TableRow, ми створюємо окремий рядок. Як розмітка дізнається, скільки стовпців треба створити? Android знаходить рядок з максимальною кількістю віджетів одного рівня, і ця кількість означатиме кількість стовпців. Наприклад, у цьому випадку у нас визначено два рядки та в кожному по два елементи. Якби в якомусь із них було б три віджети, то відповідно стовпців було б також три, навіть якщо в іншому рядку залишилося б два віджети.

Причому елемент TableRow успадковується від класу LinearLayout, тому ми можемо до нього застосовувати той самий функціонал, що й до LinearLayout. Зокрема, для визначення простору для елементів у рядку використовується атрибут`android:layout_weight`.

Якщо якийсь елемент має бути розтягнутий на ряд стовпців, то ми можемо розтягнути його за допомогою атрибута layout_span, який вказує на скільки стовпців треба розтягнути елемент:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TableRow>
<TextView
android:textSize="22sp"
android:text="Логін"
android:layout_width="100dp"
android:layout_height="wrap_content" />

<EditText
android:textSize="22sp"
android:layout_width="200dp"
android:layout_height="wrap_content" />
</TableRow>
```

```
<TableRow>
<TextView
android:textSize="22sp"
android:text="Email"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />

<EditText
android:textSize="22sp"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
</TableRow>
<TableRow>
<Button
android:text="Надіслати"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_span="2"/>
</TableRow>
</TableLayout>
```

Також можна розтягнути елемент на весь рядок, встановивши атрибут у нього`android:layout_weight="1"`:

```
<TableRow>
<Button
android:text="Надіслати"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_weight="1" />
</TableRow>
```

Програмне створення TableLayout

Створимо TableLayout програмним чином, переклавши на код java перший приклад з цієї статті:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

TableLayout tableLayout = New TableLayout(this);

// перший рядок
TableRow tableRow1 = New TableRow(this);

TextView textView1 = New TextView(this);
textView1.setText("Логін");
tableRow1.addView(textView1, new TableRow.LayoutParams(
TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 0.5f));

EditText editText1 = New EditText(this);
tableRow1.addView(editText1, new TableRow.LayoutParams(
TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 1.0f));

// другий рядок
TableRow tableRow2 = New TableRow(this);
TextView textView2 = New TextView(this);
textView2.setText("Email");
tableRow2.addView(textView2, new TableRow.LayoutParams(
TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 0.5f));

EditText editText2 = новий EditText(this);
tableRow2.addView(editText2, new TableRow.LayoutParams(
TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 1.f));

tableLayout.addView(tableRow1);
tableLayout.addView(tableRow2);
setContentView(tableLayout);}}
```

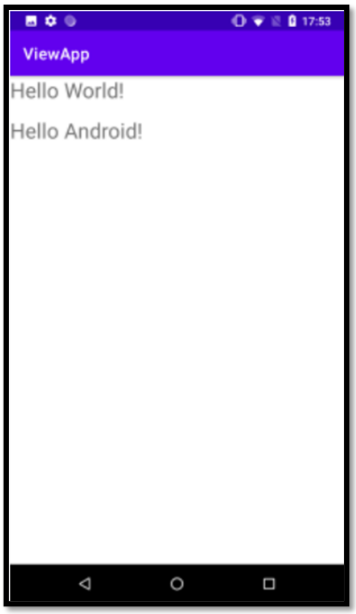
2.24. FrameLayout

Контейнер FrameLayout призначений для виведення на екран одного візуального елемента, що вміщений у нього. Якщо ж ми помістимо кілька елементів, то вони накладатимуться один на одного. Проте також можна розташовувати в FrameLayout кілька елементів.

Допустимо, вкладемо у FrameLayout два елементи TextView:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
android:textSize="26sp"/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android!"
android:textSize="26sp"
android:layout_marginTop="50dp"/>

</FrameLayout>
```

Тут обидва елементи позиціонуються в те саме місце - в лівий верхній кут контейнера FrameLayout, і щоб уникнути накладання, в даному випадку у другого TextView встановлюється відступ зверху в 50 одиниць.



Нерідко FrameLayout застосовується для створення похідних контейнерів, наприклад ScrollView, який забезпечує прокручування.

Елементи керування, що містяться у FrameLayout, можуть встановити своє позиціонування за допомогою атрибута android:layout_gravity:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
android:textSize="26sp"
android:layout_gravity="center_horizontal" />
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Welcome to Java World"
android:textSize="26sp"
android:layout_gravity="center"/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android!"
android:textSize="26sp"
android:layout_gravity="bottom|center_horizontal"/>

</FrameLayout>
```

При вказанні значення ми можемо комбінувати ряд значень, розділяючи їх вертикальною рисою:`bottom|center_horizontal`

Програмне створення FrameLayout у коді MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Gravity;
import android.widget.FrameLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

FrameLayout frameLayout = new FrameLayout(this);
TextView textView = New TextView(this);
textView.setText("Hello World!");

FrameLayout.LayoutParams layoutParams = new
FrameLayout.LayoutParams
(FrameLayout.LayoutParams.WRAP_CONTENT,
FrameLayout.LayoutParams.WRAP_CONTENT);
layoutParams.gravity = Gravity.CENTER_HORIZONTAL |
Gravity.TOP;

textView.setLayoutParams(layoutParams);
textView.setTextSize(26);
frameLayout.addView(textView);
setContentView(frameLayout);
}
}
```

2.25. GridLayout

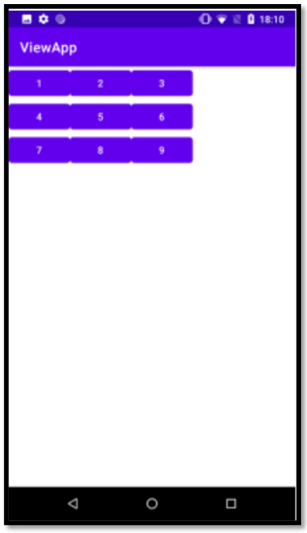
GridLayout представляє ще один контейнер, який дозволяє створювати табличні уявлення. GridLayout складається з колекції рядків, кожен з яких складається з окремих осередків:

```
<GridLayoutxmlns:android="http://schemas.android.com/apk/res/and
roid"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:rowCount="3"
android:columnCount="3">

<Buttonandroid:text="1" />
```

```
    <Buttonandroid:text="2" />
    <Buttonandroid:text="3" />
    <Buttonandroid:text="4" />
    <Buttonandroid:text="5" />
    <Buttonandroid:text="6" />
    <Buttonandroid:text="7" />

    <Buttonandroid:text="8" />

    <Buttonandroid:text="9" />
    </GridLayout>
```

За допомогою атрибутів android:rowCount та android:columnCount встановлюється число рядків та стовпців відповідно. Так, у даному випадку встановлюємо 3 рядки та 3 стовпці. GridLayout автоматично може позиціонувати вкладені елементи керування рядками. Так, у нашому випадку перша кнопка потрапляє в перший осередок (перший рядок перший стовпець), друга кнопка - у другий осередок і так далі.

При цьому ширина стовпців встановлюється автоматично за шириною найширшого елемента.

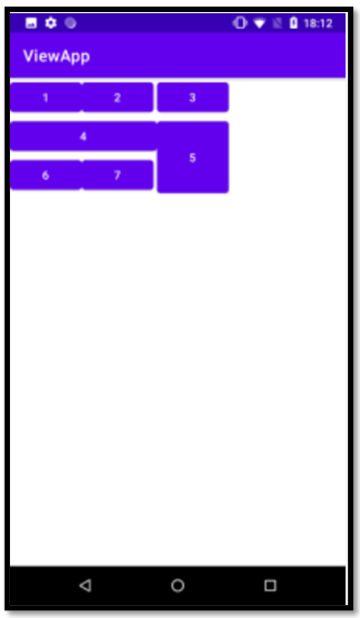


Однак ми можемо явно задати номер стовпця та рядки для певного елемента, а за необхідності розтягнути на кілька стовпців або рядків. Для цього ми можемо використовувати такі атрибути:

- **android:layout_column**: номер стовпця (відлік йде від нуля)
- **android:layout_row**: номер строки
- **android:layout_columnSpan**: кількість стовпців, на які розтягується елемент
- **android:layout_rowSpan**: кількість рядків, на які розтягується елемент

Наприклад:

```
      <GridLayoutxmlns:android="http://schemas.android.com/apk/res/and
roid"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:rowCount="3"
      android:columnCount="3">

      <Button
      android:text="1"
      android:layout_column="0"
      android:layout_row="0"/>
      <Buttonandroid:text="2"
      android:layout_column="1"
      android:layout_row="0"/>
      <Buttonandroid:text="3"
      android:layout_column="2"
      android:layout_row="0"/>
      <Buttonandroid:text="4"
      android:layout_width="180dp"
      android:layout_columnSpan="2"/>
      <Buttonandroid:text="5"
      android:layout_height="100dp"
      android:layout_rowSpan="2"/>
      <Buttonandroid:text="6" />
      <Buttonandroid:text="7"/>
      </GridLayout>
```



Програмне створення GridLayout

Серед методів GridLayout слід зазначити методи setRowCount() та setColumnCount(), які дозволяють задати відповідно кількість рядків та стовпців. Наприклад, визначимо у коді GridLayout, аналогічний першому прикладу у статті:

```
packagecom.example.viewapp;

importandroidx.appcompat.app.AppCompatActivity;
importandroid.os.Bundle;
importandroid.view.Gravity;
importandroid.widget.Button;
importandroid.widget.EditText;
importandroid.widget.GridLayout;
importandroid.widget.LinearLayout;
importandroid.widget.TableLayout;
importandroid.widget.TableRow;
importandroid.widget.TextView;

publicclass MainActivity extends AppCompatActivity {

@Override
protectedvoid onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

GridLayout gridLayout = NewGridLayout(this);
// кількість рядків
gridLayout.setRowCount(3);
// кількість стовпців
gridLayout.setColumnCount(3);

for(inti = 1; i <=9; i++){
button btn = newButton(this);
btn.setText(String.valueOf(i));
gridLayout.addView(btn);
}
setContentView(gridLayout);
}
}
```

У даному випадку GridLayout має три рядки та три стовпці. При додаванні віджетів (в даному випадку кнопок) вони послідовно поміщаються в комірки гриду по одному віджету в комірці.

2.25.1.1. GridLayout.LayoutParams

Для більш детального настроювання розташування віджету в гриді можна використовувати клас GridLayout.LayoutParams. Цей клас має ряд властивостей, які дозволяють налаштувати розташування:

- **columnSpec**: задає стовпець для розташування у вигляді об'єкту GridLayout.Spec
- **rowSpec**: задає рядок для розташування у вигляді об'єкту GridLayout.Spec

- **leftMargin**: задає відступ ліворуч
- **rightMargin**: задає відступ праворуч
- **topMargin**: задає відступ зверху
- **bottomMargin**: задає відступ знизу
- **width**: задає ширину віджету.
- **height**: задає висоту віджета.

Об'єкт GridLayout.Spec дозволяє задати розміщення в осередках стовпця або рядка. Для створення цього об'єкта застосовується статичний метод GridLayout.spec(), який має низку версій. Зазначимо серед них такі:

- `GridLayout.spec(int)`: задає стовпець або рядок, де розташовується віджет. Відлік осередків починається з нуля. Віджет займає лише один осередок
- `GridLayout.spec(int, int)`: перший параметр задає стовпець або рядок, де розташовується віджет. Другий параметр вказує, наскільки осередків розтягується віджет
- `GridLayout.spec(int, android.widget.GridLayout.Alignment)`: перший параметр задає стовпець або рядок, де розташовується віджет. Другий параметр встановлює вирівнювання віджету
- `GridLayout.spec(int, int, android.widget.GridLayout.Alignment)`: перший параметр задає стовпець або рядок, де розташовується віджет. Другий параметр вказує, наскільки осередків розтягується віджет. Третій параметр встановлює вирівнювання віджету

Приклад застосування GridLayout.LayoutParams:

```
     button btn = newButton(this);
     btn.setText("натисніть");
     GridLayout.LayoutParams           layoutParams           =
newGridLayout.LayoutParams();
     // кнопка міститься в нульовий стовпець і розтягується на
2 стовпці
     layoutParams.columnSpec = GridLayout.spec(0,2);
     // кнопка міститься у другий рядок і розтягується на 1
рядок
     layoutParams.rowSpec = GridLayout.spec(1,1);
     layoutParams.leftMargin=5;
     layoutParams.rightMargin=5;
     layoutParams.topMargin=4;
     layoutParams.bottomMargin=4;
     layoutParams.width = GridLayout.LayoutParams.MATCH_PARENT;
     layoutParams.height=GridLayout.LayoutParams.WRAP_CONTENT;
     gridLayout.addView(btn, layoutParams);
```

Наприклад, реалізуємо в коді другий приклад цієї статті:

```
      packagecom.example.viewapp;

      importandroidx.appcompat.app.AppCompatActivity;
      importandroid.os.Bundle;
      importandroid.util.TypedValue;
      importandroid.view.Gravity;
      importandroid.widget.Button;
      importandroid.widget.EditText;
      importandroid.widget.GridLayout;
      importandroid.widget.LinearLayout;
      importandroid.widget.TableLayout;
      importandroid.widget.TableRow;
      importandroid.widget.TextView;

      publicclass MainActivity extends AppCompatActivity {

      @Override
      protectedvoid onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      GridLayout gridLayout = NewGridLayout(this);

      // кількість рядків
      gridLayout.setRowCount(3);
      // кількість стовпців
      gridLayout.setColumnCount(3);

      for(inti = 1; i <=3; i++){
      button btn = newButton(this);
      btn.setText(String.valueOf(i));
      gridLayout.addView(btn);
      }

      Button btn4 = NewButton(this);
      btn4.setText("4");
      GridLayout.LayoutParams
layoutParams4=newGridLayout.LayoutParams();
      layoutParams4.columnSpec = GridLayout.spec(0,2);
      layoutParams4.width = (int) TypedValue.applyDimension(
      TypedValue.COMPLEX_UNIT_DIP,                                  180,
getResources().getDisplayMetrics());
      gridLayout.addView(btn4, layoutParams4);


      Button btn5 = NewButton(this);
      btn5.setText("5");
      GridLayout.LayoutParams
```

```
layoutParams5=newGridLayout.LayoutParams();
     layoutParams5.rowSpec = GridLayout.spec(1,2);
     layoutParams5.height = (int) TypedValue.applyDimension(
     TypedValue.COMPLEX_UNIT_DIP,                              100,
getResources().getDisplayMetrics());
     gridLayout.addView(btn5, layoutParams5);

     Button btn6 = NewButton(this);
     btn6.setText("6");
     Button btn7 = NewButton(this);
     btn7.setText("7");
     gridLayout.addView(btn6);
     gridLayout.addView(btn7);

     setContentView(gridLayout);
     }
     }
```

2.26. **ScrollView**

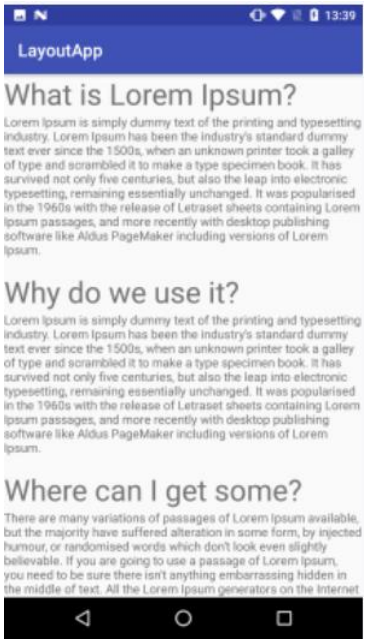
Контейнер ScrollView призначений для створення прокручування для такого інтерфейсу, всі елементи якого миттєво не можуть поміститися на екрані пристрою. ScrollView може вміщати лише один елемент, тому якщо ми хочемо розмістити кілька елементів, їх треба помістити в якийсь контейнер.

Наприклад, визначимо ряд TextView з великими текстами:

```
     <ScrollView
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent">
     <LinearLayout
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical"
     >
     <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="What is Lorem Ipsum?"
     android:textSize="34sp" />
     <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Lorem Ipsum is simply dummy text printing
and typesetting industry...like Aldus PageMaker including
versions of Lorem Ipsum."
     android:textSize="14sp"/>
     <TextView
```

```
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Why do we use it?"
     android:layout_marginTop="16dp"
     android:textSize="34sp"/>
     <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Lorem Ipsum is simply dummy text printing
and typesetting industry...like Aldus PageMaker including
versions of Lorem Ipsum."
     android:textSize="14sp"/>
     <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Whe can I get some?"
     android:layout_marginTop="16dp"
     android:textSize="34sp"/>
     <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Ви маєте багато варіантів проходжень з Lorem
Ipsum available ... or non-characteristic words etc."
     android:textSize="14sp"/>
     </LinearLayout>
     </ScrollView>
```

Так як в ScrollView можна помістити лише один елемент, всі TextView укладені в LinearLayout. І якщо площа екрану буде недостатньою, щоб помістити весь вміст LinearLayout, то стане доступною прокручування:



Створення ScrollView програмно у коді MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.ScrollView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

ScrollView scrollView = New ScrollView(this);

TextView textView = New TextView(this);
textView.setText("Lorem Ipsum is simply dummy text of
printing and typesetting industry...like Aldus PageMaker
including versions of Lorem Ipsum.");
textView.setLayoutParams(new ViewGroup.LayoutParams
(ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
textView.setTextSize(26);
scrollView.addView(textView);
setContentView(scrollView);
}
}
```

2.27. **Gravity**

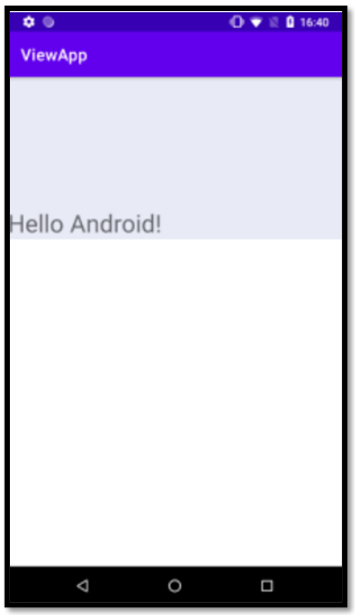
Атрибут gravity визначає позиціонування вмісту всередині візуального елемента. Він може приймати такі значення:

- `top`: елементи розміщуються зверху
- `bottom`: елементи розміщуються внизу
- `left`: елементи розміщуються у лівій стороні
- `right`: елементи розміщуються у правій стороні контейнера
- `center_vertical`: вирівнює елементи по центру по вертикалі.
- `center_horizontal`: вирівнює елементи по центру по горизонталі.
- `center`: елементи розміщуються по центру

- `fill_vertical`: елемент розтягується по вертикалі
- `fill_horizontal`: елемент розтягується по горизонталі.
- `fill`: елемент заповнює весь простір контейнера
- `clip_vertical`: обрізає верхню та нижню межу елементів
- `clip_horizontal`: обрізає праву та ліву межу елементів
- `start`: елемент позиціонується на початку (у верхньому лівому куті) контейнера
- `end`: елемент позиціонується в кінці контейнера(у верхньому правому куті)

Наприклад, помістимо текст у низ в елементі TextView:

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

<TextView
android:gravity="bottom"

android:layout_width="0dp"
android:layout_height="200dp"
android:text="Hello Android!"
android:textSize="30sp"
android:background="#e8eaf6"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />

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



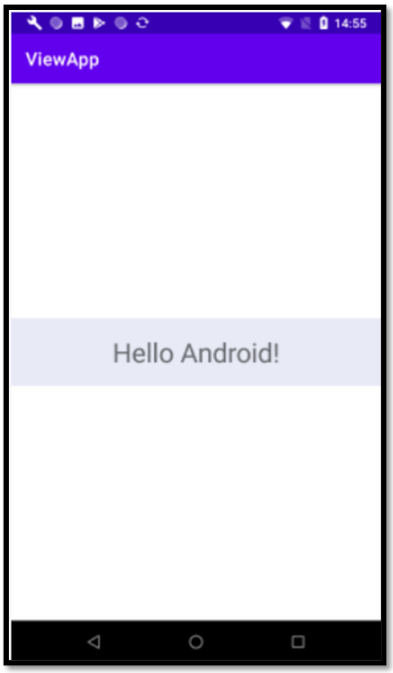
При необхідності ми можемо комбінувати значення, поділяючи їх вертикальною рисою:

```
<TextView
android:gravity="bottom|right"

android:layout_width="0dp"
android:layout_height="200dp"
android:text="Hello Android!"
android:textSize="30sp"
android:background="#e8eaf6"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

Програмне встановлення gravity

Щоб встановити параметр gravity у елемента, потрібно викликати метод setGravity(). В якості параметра метод передається одна з констант класу Gravity, які аналогічні значенням атрибута (за тим виключимо, що назви у верхньому регістрі):

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.view.Gravity;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

ConstraintLayout constraintLayout = New
ConstraintLayout(this);
TextView textView = New TextView(this);
textView.setText("Hello Android!");
textView.setTextSize(30);
textView.setBackgroundColor(0xffe8eaf6);

// встановлення gravity
textView.setGravity(Gravity.CENTER);

// Установка висоти та ширини
ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, 200);
layoutParams.leftToLeft =
ConstraintLayout.LayoutParams.PARENT_ID;
layoutParams.rightToRight =
ConstraintLayout.LayoutParams.PARENT_ID;
layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
layoutParams.bottomToBottom =
ConstraintLayout.LayoutParams.PARENT_ID;
textView.setLayoutParams(layoutParams);

constraintLayout.addView(textView);
setContentView(constraintLayout);}}
```



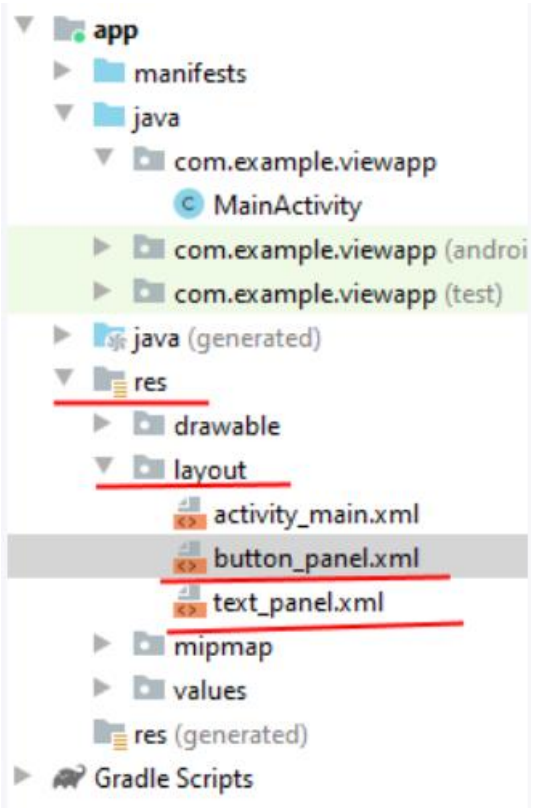
Для поєднання кількох значень також можна використовувати вертикальну межу:

```
textView.setGravity(Gravity.BOTTOM | Gravity.CENTER);
```

2.28. **Вкладені layout**

Один layout може містити інший layout. І тому застосовується елемент include.

Наприклад, додамо в папку res/layout два файли layout, які будуть називатися text_panel.xml і button_panel.xml:



У файлі text_panel.xml визначимо наступний код:

```
<?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"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<TextView
android:id="@+id/clicksText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30sp"
android:text="0 Clicks"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

По суті, тут просто визначено поле TextView для виведення тексту.

У файлі button_panel.xml визначимо таку розмітку:

```
<?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"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Click"
android:onClick="onClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
/>

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

Тут визначено кнопку, натискання якої ми будемо обробляти.

Основним файлом розмітки, який визначає інтерфейс програми, як і раніше, є activity_main.xml. Змінимо його:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp"
```

```
    tools:context=".MainActivity">

    <include
    android:id="@+id/textView"
    layout="@layout/text_panel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/button"
    />
    <include
    android:id="@+id/button"
    layout="@layout/button_panel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    />

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

За допомогою ConstraintLayout весь інтерфейс тут організується як вертикального стека. За допомогою елементів include всередину ConstraintLayout додається вміст файлів text_panel.xml та button_panel.xml. Для вказівки назви файлу використовується атрибут layout.

Це все одно, що якщо б ми замість елемента include додали вміст файлів. Проте такий спосіб має переваги. Наприклад, якась частина розмітки, група елементів управління може повторюватися в різних діях. І щоб не визначати сто разів ці елементи, можна винести їх в окремий файл layout і за допомогою include підключати їх.

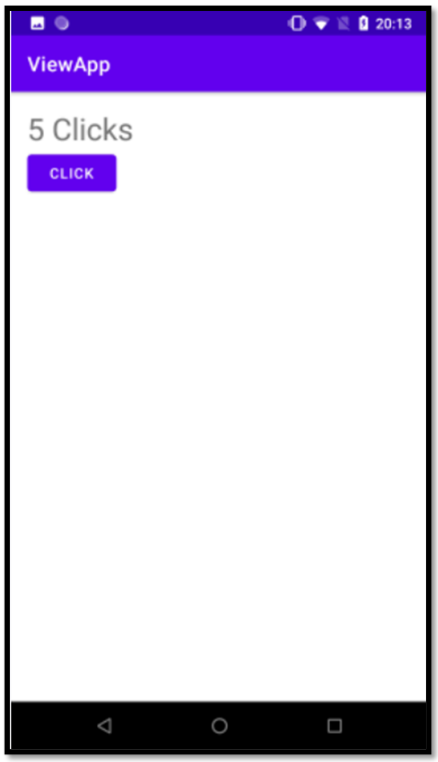
Після додавання в ConstraintLayout до елементів include можна застосовувати всі стандартні атрибути, які застосовуються в цьому контейнері до вкладених елементів, наприклад, налаштувати розміри, розташування. Також варто відзначити, що додавати зовнішні layout можна не тільки в ConstraintLayout, а й в інші контейнери (LinearLayout, RelativeLayout і т.д.)

Також змінимо код MainActivity:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.os.Bundle;
    import android.view.View;
    import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {

int clicks = 0;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}

public void onClick(View view){
TextView clicksText = findViewById(R.id.clicksText);
clicks++;
clicksText.setText(clicks + "Clicks");
}
}
```

У MainActivity ми можемо звертатися до елементів у вкладених файлах layout. Наприклад, ми можемо встановити обробник натискання кнопки, в якому при натисканні змінювати текст TextView.



При цьому ми кілька разів можемо додавати один файл layout інший файл layout. Для цього спочатку змінимо файл button_panel.xml таким чином:

```
<?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"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
    android:background="#3F51B5"
    android:paddingTop="10dp"
    android:paddingBottom="10dp">
    <Button
    android:id="@+id/clickBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    />

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

І змінимо файл activity_main.xml:

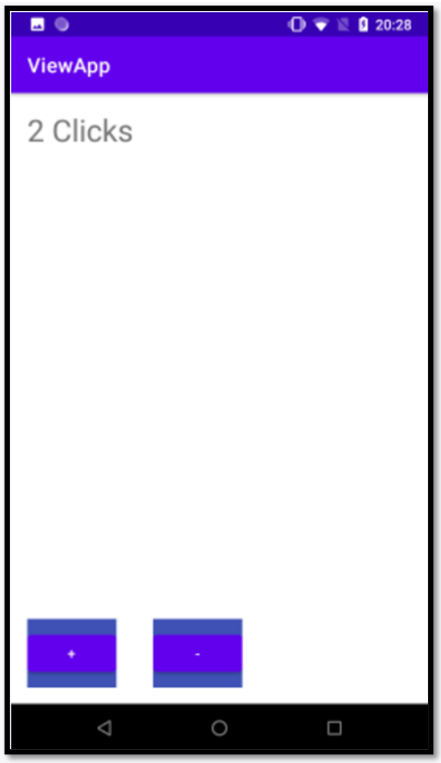
```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">

    <include
    layout="@layout/text_panel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    />
    <include layout="@layout/button_panel"
    android:id="@+id/plus_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/minus_button"/>

    <include layout="@layout/button_panel"
    android:id="@+id/minus_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="36dp"
    app:layout_constraintLeft_toRightOf="@id/plus_button"
    app:layout_constraintBottom_toBottomOf="parent"/>

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

Тепер файл button_panel.xml додається двічі. Важливо, що при додаванні файлу кожному елементу include присвоєний певний id. Тому id ми зможемо дізнатися, про який саме елемент include йдеться.

Також змінимо MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

int clicks = 0;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

View plusButtonView = findViewById(R.id.plus_button);
View minusButtonView = findViewById(R.id.minus_button);
TextView clicksText = findViewById(R.id.clicksText);

Button plusButton =
plusButtonView.findViewById(R.id.clickBtn);
Button minusButton =
minusButtonView.findViewById(R.id.clickBtn);

plusButton.setText("+");
minusButton.setText("-");

plusButton.setOnClickListener(v -> {
clicks++;
clicksText.setText(clicks + "Clicks");
});
minusButton.setOnClickListener(v -> {
clicks--;
clicksText.setText(clicks + "Clicks");
});
}
}
```

Тут спочатку ми отримуємо окремі елементи, include по id. Потім у межах цих елементів отримуємо кнопку. Після цього ми можемо встановити в будь-який текст і повісити обробник події натискання. І таким чином, поведінка обох кнопок відрізнятиметься.


20:28
ViewApp
2 Clicks
+
-

3. ОСНОВНІ ЕЛЕМЕНТИ КЕРУВАННЯ

3.1. TextView

Для простого виведення тексту на екран призначено елемент TextView. Він просто відображає текст без можливості його редагування. Деякі його основні атрибути:

- **android:text**: встановлює текст елемента
- **android:textSize**: встановлює висоту тексту, як одиниці виміру для вказівки висоти використовуються sp
- **android:background**: задає фоновий колір елемента у вигляді кольору у шістнадцятковому записі або у вигляді колірного ресурсу
- **android:textColor**: задає колір тексту
- **android:textAllCaps**: при значенні true робить усі символи в тексті великими
- **android:textDirection**: встановлює напрямок тексту За замовчуванням використовується напрямок ліворуч, але за допомогою значення rtl можна встановити напрямок праворуч наліво
- **android:textAlignment**: визначає вирівнювання тексту. Може приймати такі значення:
  - `center`: вирівнювання по центру
  - `textStart`: по лівому краю
  - `textEnd`: з правого краю
  - `viewStart`: при напрямку тексту зліва направо вирівнювання по лівому краю, при напрямку праворуч наліво - по правому
  - `viewEnd`: при напрямку тексту зліва направо вирівнювання по правому краю, при напрямку праворуч наліво - по лівому
- **android:fontFamily**: встановлює тип шрифту. Може приймати такі значення:
  - monospace
  - serif
  - serif-monospace
  - sans-serif

- sans-serif-condensed
- sans-serif-smallcaps
- sans-serif-light
- casual
- cursive
- cursive

Наприклад, визначимо три текстові поля:

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

<TextView
android:layout_height="wrap_content"
android:layout_width="0dp"
android:layout_margin="10dp"

android:text="Hello Android"
android:fontFamily="sans-serif"
android:textSize="26sp"
android:background="#ffebee"
android:textColor="#f44336"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintRight_toRightOf="parent"/>
<TextView
android:layout_height="wrap_content"
android:layout_width="0dp"
android:layout_margin="10dp"

android:text="Hello Java"
android:textAllCaps="true"
android:textSize="26sp"
android:background="#ede7f6"
android:textColor="#7e57c2"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintRight_toRightOf="parent"/>

<TextView
```

```
    android:layout_height="wrap_content"
    android:layout_width="0dp"
    android:layout_margin="10dp"

    android:text="Hello World"
    android:textAlignment="textEnd"
    android:textSize="26sp"
    android:background="#e8eaf6"
    android:textColor="#5c6bc0"

    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toRightOf="parent"/>

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

Встановлення елемента в коді також не відрізняється складністю. Наприклад, створимо елемент і виведемо його на екран:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;
    import androidx.constraintlayout.widget.ConstraintLayout;

    import android.graphics.Typeface;
    import android.os.Bundle;
    import android.widget.TextView;

    public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
     ConstraintLayout       constraintLayout       =       New
ConstraintLayout(this);
     TextView textView = New TextView(this);
     // Встановлення фонового кольору
     textView.setBackgroundColor(0xffe8eaf6);
     // Встановлення кольору тексту
     textView.setTextColor(0xff5c6bc0);
     // робимо всі букви великими
     textView.setAllCaps(true);
     // встановлюємо врівнювання тексту центром
     textView.setTextAlignment(TextView.TEXT_ALIGNMENT_CENTER);
     // встановлюємо текст
     textView.setText("Hello Android!");
     // Встановлення шрифту
     textView.setTypeface(Typeface.create("casual",
Typeface.NORMAL));
     // встановлюємо висоту тексту
     textView.setTextSize(26);

     ConstraintLayout.LayoutParams    layoutParams    =    new
ConstraintLayout.LayoutParams
     (ConstraintLayout.LayoutParams.WRAP_CONTENT,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
     layoutParams.leftToLeft                                  =
ConstraintLayout.LayoutParams.PARENT_ID;
     layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
     textView.setLayoutParams(layoutParams);

     constraintLayout.addView(textView);
     setContentView(constraintLayout);
     }
     }
```

Іноді необхідно вивести на екран яке-небудь посилання, або телефон, за натисканням на які здійснювалося певну дію. Для цього в TextView визначено атрибут android:autoLink:

```
<TextView
android:text="Відвідайте сайт https://snu.edu.ua"
android:textSize="21sp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:autoLink="web|email"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
```

android:autoLink може набувати кількох значень:

- `none`: відключає всі посилання
- `web`: включає всі веб-посилання
- `email`: включає посилання на електронні адреси
- `phone`: включає посилання на номери телефонів
- `map`: включає посилання на карту
- `all`: включає всі вищенаведені посилання

Тобто при налаштуванні android:autoLink="web" якщо в тексті є згадка адреси url, то ця адреса виділятиметься, а при натисканні на неї буде здійснено перехід до веб-браузера, який відкриє сторінку за цією адресою. За допомогою прямої риси ми можемо поєднувати умови, як у цьому випадку:`android:autoLink="web|email"`

3.2. EditText

Елемент EditText є підклас класу TextView. Він також представляє текстове поле, але вже з можливістю введення та редагування тексту. Таким чином, в EditText ми можемо використовувати ті самі можливості, що і в TextView.

З тих атрибутів, які розглядалися у темі для TextView, слід зазначити атрибут android:hint. Він дозволяє задати текст, який буде відображатися як підказка, якщо елемент EditText порожній. Крім того, ми можемо використовувати атрибут android:inputType, який дозволяє встановити клавіатуру для введення. Зокрема, серед його значень можна назвати такі:

- `text`: звичайна клавіатура для введення однорядкового тексту
- `textMultiLine`: багаторядкове текстове поле
- `textEmailAddress`: звичайна клавіатура, на якій є символ @, орієнтована на введення email
- `textUri`: звичайна клавіатура, на якій є символ /, орієнтована на введення інтернет-адрес
- `textPassword`: клавіатура для введення пароля
- `textCapWords`: при введенні перший введений символ слова представляє велику літеру, інші - малі
- `number`: цифрова клавіатура
- `phone`: клавіатура у стилі звичайного телефону
- `date`: клавіатура для введення дати
- `time`: клавіатура для введення часу
- `datetime`: клавіатура для введення дати та часу

Використовуємо EditText:

```
<?xmlversion="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"
android:layout_width="match_parent"
android:layout_height="match_parent">

<EditText
android:id="@+id/name"
android:layout_width="0dp"
```

```
    android:layout_height="wrap_content"
    android:hint="Введіть ім'я"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintRight_toRightOf="parent"/>
    <EditText
    android:id="@+id/message"
    android:layout_marginTop="16dp"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:hint="Введіть повідомлення"
    android:inputType="textMultiLine"
    android:gravity="top"
    app:layout_constraintTop_toBottomOf="@+id/name"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>

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

Перше поле тут звичайне однорядкове, а друге – багаторядкове. Щоб на другому полі текст вирівнювався по верху, додатково встановлюється атрибут android:gravity="top".

Однією з можливостей елемента EditText є можливість обробити введені символи в міру введення користувача. Для цього визначимо у файлі activity_main.xml таку розмітку:

```
<?xmlversion="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"
android:layout_width="match_parent"
android:layout_height="match_parent">

<TextView
android:id="@+id/textView"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:textSize="34sp"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"/>
<EditText
android:id="@+id/editText"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:hint="Введіть ім'я"
app:layout_constraintTop_toBottomOf="@+id/textView"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"/>


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

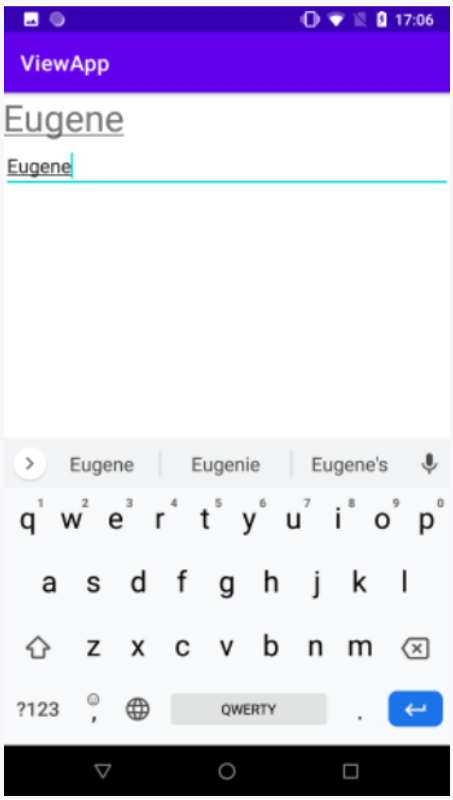
Передбачається, що введені в EditText символи будуть відображатися в елементі TextView. І для цього також змінимо код MainActivity:

```
packagecom.example.viewapp;

importandroidx.appcompat.app.AppCompatActivity;
importandroid.os.Bundle;
importandroid.text.Editable;
importandroid.text.TextWatcher;
importandroid.widget.EditText;
importandroid.widget.TextView;

publicclass MainActivity extends AppCompatActivity {

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

EditText editText = findViewById(R.id.editText);

editText.addTextChangedListener(newTextWatcher() {
```

```
        publicvoid afterTextChanged(Editable s) {}

        publicvoid beforeTextChanged(CharSequence s, int start,
        intcount, int after) {
        }

        publicvoid onTextChanged(CharSequence s, int start, int before, int count) {
        TextView textView = findViewById(R.id.textView);
        textView.setText(s);
        }
        });
        }
        }
```

За допомогою методу addTextChangedListener() тут до елемента EditText додається слухач уведення тексту - об'єкт TextWatcher. Для його використання нам треба реалізувати три методи, але насправді нам вистачить реалізації методу наTextChanged, який викликається при зміні тексту. Введений текст передається цей метод як параметр CharSequence. У самому методі просто передаємо цей текст елемент TextView.

У результаті при введенні EditText всі символи також будуть відображатися в TextView:



3.3. **Button**

Одним з найчастіше використовуваних елементів є кнопки, які представлені класом android.widget.Button. Ключовою особливістю клавіш є можливість взаємодії з користувачем через натискання.

Деякі ключові атрибути, які можна встановити у кнопок:

- `text`: задає текст на кнопці
- `textColor`: задає колір тексту на кнопці
- `background`: задає фоновий колір кнопки
- `textAllCaps`: під час значення true встановлює текст у верхньому регістрі. За замовчуванням застосовується значення true
- `onClick`: задає обробник натискання кнопки

Отже, змінимо код в activity_main.xml так:

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

<TextView
android:id="@+id/textView"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:textSize="34sp"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"/>
<EditText
android:id="@+id/editText"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:hint="Введіть ім'я"
app:layout_constraintTop_toBottomOf="@+id/textView"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Введення"
```

```
    android:onClick="sendMessage"
    app:layout_constraintTop_toBottomOf="@+id/editText"
    app:layout_constraintLeft_toLeftOf="parent" />


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

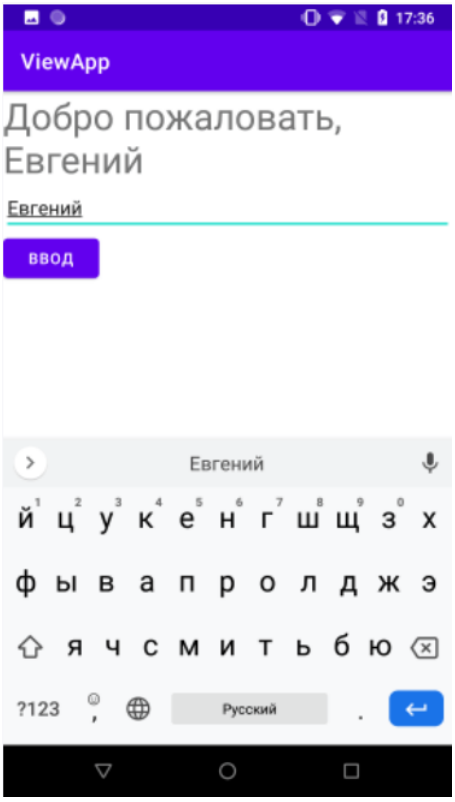
За допомогою атрибуту`android:onClick`можна задати метод у коді java, який оброблятиме натискання кнопки. Так, у наведеному вище прикладі це метод`sendMessage`. Тепер перейдемо до коду MainActivity і пропишемо в ньому такий метод:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.EditText;
    import android.widget.TextView;

    public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    }
    // Обробка натискання кнопки
    public void sendMessage(View view) {
    TextView textView = findViewById(R.id.textView);
    EditText editText = findViewById(R.id.editText);
    textView.setText("Ласкаво просимо," + editText.getText());
    }
    }
```

При створенні методу обробки натискання слід враховувати такі моменти:

- Метод має оголошуватися з модифікатором`public`
- Повинен повертати значення`void`
- Як параметр приймати об'єкт`View`. Цей об'єкт View і є натиснутою кнопкою

У цьому випадку після натискання на кнопку TextView виводиться текст з EditText.



Аналогічний приклад повністю у коді MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

ConstraintLayout constraintLayout = New
ConstraintLayout(this);
textView = new TextView(this);
textView.setId(View.generateViewId());
ConstraintLayout.LayoutParams textViewLayout = new
ConstraintLayout.LayoutParams(
ConstraintLayout.LayoutParams.MATCH_CONSTRAINT,
```

```
ConstraintLayout.LayoutParams.WRAP_CONTENT
     );
     textViewLayout.topToTop                                    =
ConstraintLayout.LayoutParams.PARENT_ID;
     textViewLayout.leftToLeft                                  =
ConstraintLayout.LayoutParams.PARENT_ID;
     textViewLayout.rightToRight                                =
ConstraintLayout.LayoutParams.PARENT_ID;
     textView.setLayoutParams(textViewLayout);
     constraintLayout.addView(textView);

     editText = new EditText(this);
     editText.setId(View.generateViewId());
     editText.setHint("Введіть ім'я");
     ConstraintLayout.LayoutParams    editTextLayout    =    new
ConstraintLayout.LayoutParams(
     ConstraintLayout.LayoutParams.MATCH_CONSTRAINT,
ConstraintLayout.LayoutParams.WRAP_CONTENT
     );
     editTextLayout.topToBottom = textView.getId();
     editTextLayout.leftToLeft                                  =
ConstraintLayout.LayoutParams.PARENT_ID;
     editTextLayout.rightToRight                                =
ConstraintLayout.LayoutParams.PARENT_ID;
     editText.setLayoutParams(editTextLayout);
     constraintLayout.addView(editText);

     button button = new Button(this);
     button.setText("Введення");
     ConstraintLayout.LayoutParams    buttonLayout    =    new
ConstraintLayout.LayoutParams(
     ConstraintLayout.LayoutParams.WRAP_CONTENT,
ConstraintLayout.LayoutParams.WRAP_CONTENT
     );
     buttonLayout.topToBottom = editText.getId();
     buttonLayout.leftToLeft                                    =
ConstraintLayout.LayoutParams.PARENT_ID;
     button.setLayoutParams(buttonLayout);
     constraintLayout.addView(button);

     button.setOnClickListener(new View.OnClickListener() {
     public void onClick(View v) {
     // Обробка натискання
     textView.setText("Ласкаво просимо," + editText.getText());
     }
     });

     setContentView(constraintLayout);
     }
```

```
    }
```

При програмному створенні кнопки ми можемо визначити слухач натискання View.OnClickListener і за допомогою його методу onClick також обробити натискання:

```
    button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
    // Обробка натискання
    }
    });
```

3.4. Додаток Калькулятор

Знаючи деякі основи компонування і такі елементи як TextView, EditText і Button, вже можна скласти повноцінний додаток. У цьому випадку ми зробимо простенький калькулятор.

Для цього створимо новий проект та визначимо у файлі activity_main.xml наступний інтерфейс:

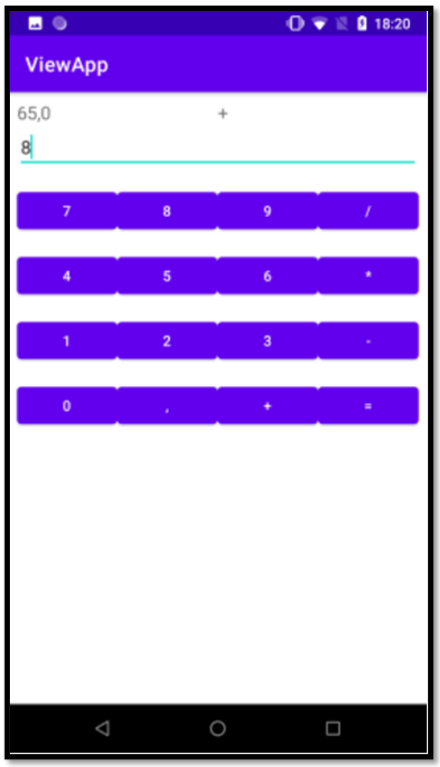
```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp">
    <!-- поле результату -->
    <TextView
    android:id="@+id/resultField"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintHorizontal_weight="1"
    android:textSize="18sp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/operationField"/
>
    <!-- поле знака операції -->
    <TextView
    android:id="@+id/operationField"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintHorizontal_weight="1"
    android:textSize="18sp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintLeft_toRightOf="@+id/resultField"
    />
    <!-- поле введення чисел -->
```

```
    <EditText
    android:id="@+id/numberField"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:inputType="phone"
    app:layout_constraintTop_toBottomOf="@+id/resultField"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"/>

    <LinearLayout
    android:id="@+id/firstButtonPanel"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintTop_toBottomOf="@+id/numberField"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent">
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="7"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="8"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="9"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="/"
    android:onClick="onOperationClick"/>
    </LinearLayout>
    <LinearLayout
    android:id="@+id/secondButtonPanel"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintTop_toBottomOf="@+id/firstButtonPanel
"
```

```
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent">
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="4"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="5"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="6"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="*"
    android:onClick="onOperationClick"/>
    </LinearLayout>
    <LinearLayout
    android:id="@+id/thirdButtonPanel"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintTop_toBottomOf="@+id/secondButtonPane
l"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent">
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="1"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="2"
    android:onClick="onNumberClick"/>
    <Button
```

```
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="3"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="-"
    android:onClick="onOperationClick"/>
    </LinearLayout>
    <LinearLayout
    android:id="@+id/forthButtonPanel"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintTop_toBottomOf="@+id/thirdButtonPanel
"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent">
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="0"
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text=","
    android:onClick="onNumberClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="+"
    android:onClick="onOperationClick"/>
    <Button
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="="
    android:onClick="onOperationClick"/>
    </LinearLayout>

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

У результаті весь інтерфейс буде виглядати так:



Кореневий контейнер компонування представляє елемент ConstraintLayout. Зверху в ньому визначено два текстові поля TextView: одне для виведення результату обчислень та одне для виведення поточного знака операції.

Потім іде елемент EditText, призначений для введення чисел.

І далі розташовані чотири елементи LinearLayout із горизонтальними рядами кнопок. Щоб усі кнопки займали рівний простір усередині контейнера, для них встановлені атрибути`android:layout_weight="1"`і`android:layout_width="0dp"`.

Крім того, для числових кнопок як обробник натискання встановлено метод`onNumberClick`для кнопок зі знаками операцій атрибут`onClick`вказує на метод`onOperationClick`.

Тепер змінимо клас MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
     TextView resultField; // текстове поле висновку результату
     EditText numberField; // поле для введення числа
     TextView operationField; // текстове поле виведення знака
операції
     Double operand = null; // операнд операції
     String lastOperation = "="; // остання операція
     @Override
     protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     // отримуємо всі поля з id з activity_main.xml
     resultField = findViewById(R.id.resultField);
     numberField = findViewById(R.id.numberField);
     operationField = findViewById(R.id.operationField);
     }
     // Збереження стану
     @Override
     protected void onSaveInstanceState(Bundle outState) {
     outState.putString("OPERATION", lastOperation);
     if(operand!=null)
     outState.putDouble("OPERAND", operand);
     super.onSaveInstanceState(outState);
     }
     // отримання раніше збереженого стану
     @Override
     protected        void        onRestoreInstanceState(Bundle
savedInstanceState) {
     super.onRestoreInstanceState(savedInstanceState);
     lastOperation = savedInstanceState.getString("OPERATION");
     operand= savedInstanceState.getDouble("OPERAND");
     resultField.setText(operand.toString());
     operationField.setText(lastOperation);
     }
     // Обробка натискання на числову кнопку
     public void onNumberClick(View view){

     button button = (button)view;
     numberField.append(button.getText());

     if(lastOperation.equals("=") && operand!=null){
     operand = null;
     }
     }
     // Обробка натискання на кнопку операції
     public void onOperationClick(View view){

     button button = (button)view;
     String op = button.getText().toString();
     String number = numberField.getText().toString();
```

```
     // якщо введено щось
     if(number.length()>0){
     number = number.replace(',', '.');
     try{
     performOperation(Double.valueOf(number), op);
     }catch (NumberFormatException ex) {
     numberField.setText("");
     }
     }
     lastOperation = op;
     operationField.setText(lastOperation);
     }

     private void performOperation(Double number, String
operation){

     // якщо операнд раніше не був встановлений (при введенні
першої операції)
     if(operand ==null){
     operand = number;
     }
     else {
     if(lastOperation.equals("=")){
     lastOperation = operation;
     }
     switch(lastOperation){
     case "=":
     operand =number;
     break;
     case "/":
     if(number==0){
     operand =0.0;
     }
     else {
     operand /=number;
     }
     break;
     case "*":
     operand *=number;
     break;
     case "+":
     operand +=number;
     break;
     case "-":
     operand -=number;
     break;
     }
     }
     resultField.setText(operand.toString().replace('.', ','));
```

```
numberField.setText("");
}
}
```

Розберемо цей код. Спочатку у методі`onCreate()`отримуємо всі поля з activity_main.xml, текст яких змінюватиметься:

```
resultField = findViewById(R.id.resultField);
numberField = findViewById(R.id.numberField);
operationField = findViewById(R.id.operationField);
```

Результат операції буде потрапляти в змінну operand, яка представляє тип Double, а знак операції - в змінну lastOperation:

```
Double operand = null;
String lastOperation = "=";
```

Оскільки при переході від портретної орієнтації до альбомної чи навпаки ми можемо втратити всі введені дані, те щоб їх не втратити, ми їх зберігаємо у методі`onSaveInstanceState()`і назад отримуємо у методі`onRestoreInstanceState()`.

При натисканні на числову кнопку викликатиметься метод`onNumberClick`, в якому додаємо введену цифру або знак коми до тексту в полі numberField:

```
button button = (button)view;
numberField.append(button.getText());

if(lastOperation.equals("=") && operand!=null){
operand = null;
}
```

При цьому якщо остання операція була одержанням результату (знак "рівно"), то ми скидаємо змінну operand.

У методі`onOperationClick`відбувається обробка натискання на кнопку зі знаком операції:

```
button button = (button)view;
String op = button.getText().toString();
String number = numberField.getText().toString();
if(number.length()>0){
number = number.replace(',', '.');
try{
performOperation(Double.valueOf(number), op);
}catch (NumberFormatException ex) {
numberField.setText("");
}
}
lastOperation = op;
operationField.setText(lastOperation);
```

Тут отримуємо раніше введене число та введену операцію та передаємо їх у методperformOperation(). Так як метод передається не просто рядок, а число Double, то нам треба перетворити рядок в число. І оскільки теоретично можуть бути введені нечислові символи, то для вилову виключення, що може виникнути при перетворенні, використовується конструкція try...catch.

Крім того, так як роздільником цілої і дробової частини в Double в java є точка, то нам треба замінити кому на точку, так як передбачається, що ми використовуємо як розділювач кому.

А в методіperformOperation()виконуємо власне операцію. При введенні першої операції, коли операнд ще не встановлено, ми просто встановлюємо операнд:

```
if(operand ==null){
operand = number;
}
```

При введенні другої та наступних операцій застосовуємо попередню операцію, знак якої зберігається в змінній останнійоперації, до операнда operand і другого числа, яке було введено в числове поле. Отриманий результат операції зберігаємо у змінній operand.

3.5. **Вспливаючі вікна. Toast**

Для створення простих повідомлень Android використовується клас Toast. Фактично Toast представляє спливаюче вікно з деяким текстом, яке відображається протягом деякого часу.

Не можна створити об'єкт Toast у коді розмітки xml, наприклад, у файлі activity_main.xml. Toast можна використовувати лише у коді java.

Так, визначимо у файлі розмітки activity_main.xml кнопку:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Click"
android:onClick="onClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

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

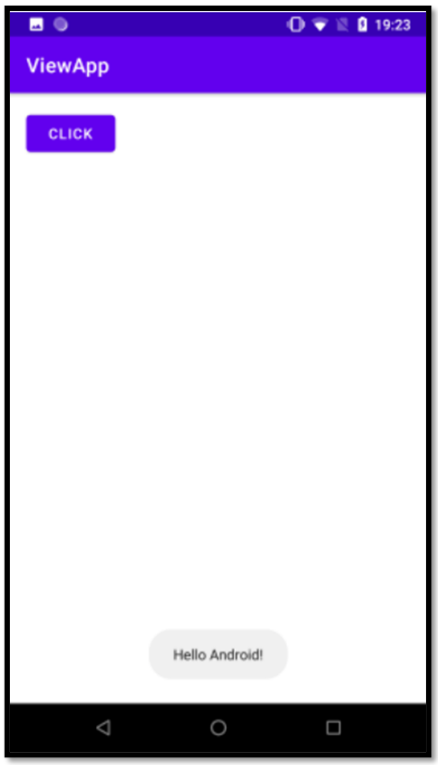
У кнопки встановлено обробник натискання – метод onClick. Визначимо його в коді MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);
}

public void onClick(View view){
Toast toast = Toast.makeText(this, "Hello Android!",
Toast.LENGTH_LONG);
toast.show();
}
}
```

В обробнику відображається спливаюче вікно. Для його створення застосовується метод Toast.makeText(), в який передається три параметри: поточний контекст (поточний об'єкт activity), текст, що відображається, і час відображення вікна.

Як час показу вікна ми можемо використовувати цілісне значення - кількість мілісекунд або вбудовані константи Toast.LENGTH_LONG (3500 мілісекунд) та Toast.LENGTH_SHORT (2000 мілісекунд).

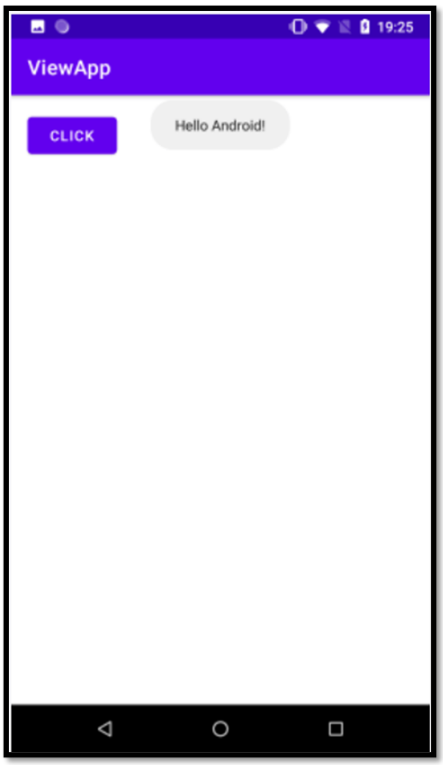
Для відображення вікна викликається метод show():



За замовчуванням вікно відображається внизу інтерфейсу з центрування центром. Але ми можемо кастомізувати позиціонування вікна за допомогою методів setGravity() та setMargin(). Так, змінимо метод onClick:

```
    public void onClick(View view){

    Toast toast = Toast.makeText(this, "Hello Android!",
Toast.LENGTH_LONG);
    toast.setGravity(Gravity.TOP, 0,160); // import
android.view.Gravity;
    toast.show();
    }
```

Перший параметр методу setGravity вказує, в якій частині контейнера треба позиціонувати Toast, другий та третій параметр встановлюють відступи від цієї позиції по горизонталі та вертикалі:

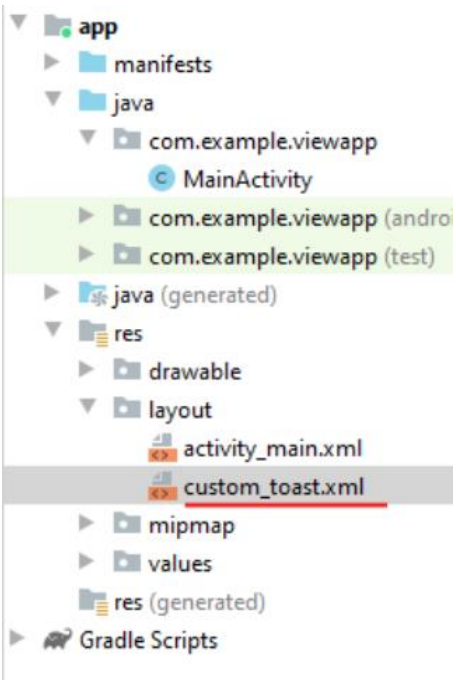


Метод setMargin() приймає два параметри: відступ від лівої межі контейнера у відсотках від шиірини контейнера та відступ від верхньої межі у відсотках від довжини контейнера.

Визначення інтерфейсу Toast

Android підтримує кастомізацію візуального інтерфейсу спливаючих вікон. Для цього додамо до проекту папку res/layout новий файл layout custom_toast.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
android:id="@+id/toast_layout"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="#555"
android:padding="16dp"
>
<TextView android:id="@+id/text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textColor="#81C784"
android:textSize="30sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

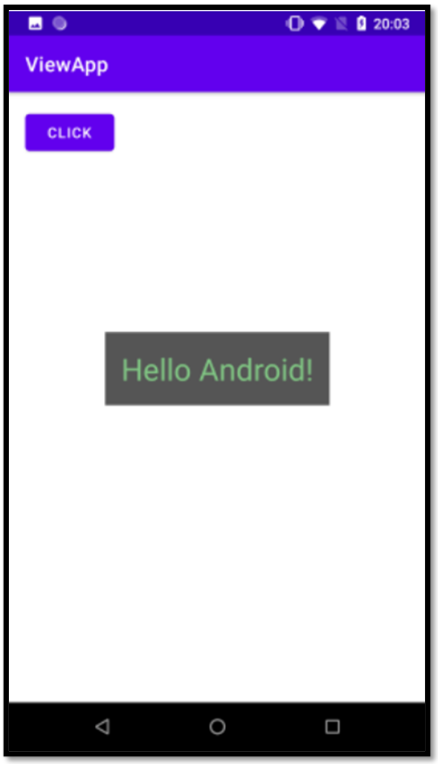


У коді MainActivity налаштуємо відображення Toast:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);
}

public void onClick(View view){
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
findViewById(R.id.toast_layout));

TextView text = layout.findViewById(R.id.text);
text.setText("Hello Android!");

Toast toast = New Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
```

```
        toast.setDuration(Toast.LENGTH_LONG);
        toast.setView(layout);
        toast.show();
        }
        }
```

Тут за натисканням кнопки за допомогою об'єкта LayoutInflater завантажуємо вище певний інтерфейс з файлу custom_toast.xml. Потім налаштовуємо відображення вікна, його текст і виводимо вікно, що спливає, по центру екрана.



3.6. Snackbar

Елемент Snackbar певною мірою схожий на Toast: він також дозволяє виводити спливаючі повідомлення, але тепер повідомлення розтягуються по ширині екрана.

Для застосування Snackbar додамо у файл activity_main.xml визначення кнопки, натискання на яку з'являтиметься Snackbar:

```
        <?xmlversion="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"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp">

        <Button
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="Click"
    android:onClick="onClick"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

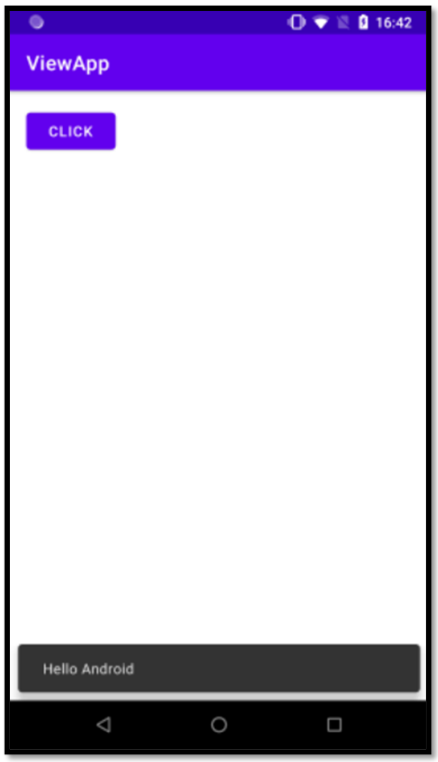
Тут визначено кнопку, за натисканням на яку зображатиметься повідомлення.

І також змінимо клас MainActivity:

```
    packagecom.example.viewapp;

    importandroidx.appcompat.app.AppCompatActivity;
    importandroid.os.Bundle;
    importandroid.view.View;

    importcom.google.android.material.snackbar.Snackbar;

    publicclass MainActivity extends AppCompatActivity {

    @Override
    protectedvoid onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    }

    publicvoid onClick(View view) {
    Snackbar.make(view, "Hello Android", Snackbar.LENGTH_LONG)
    .show();
    }
    }
```

Snackbar створюється за допомогою методу make(), який передаються три параметри: об'єкт View, до якого прикріплюється спливаюче повідомлення, саме повідомлення у вигляді рядка і параметр, який вказує, скільки відображатиметься повідомлення. Останній параметр може приймати числове значення - кількість мілісекунд або одну з трьох констант: Snackbar.LENGTH_INDEFINITE (відображення протягом невизначеного періоду часу), Snackbar.LENGTH_LONG (довге відображення) або Snackbar.LENGTH_SHORT (недовге відображення).

Після створення Snackbar відображається за допомогою методу show:



При цьому, на відміну від Toast, ми не можемо вплинути на позицію повідомлення, воно відображається внизу екрана і займає всю нижню частину.

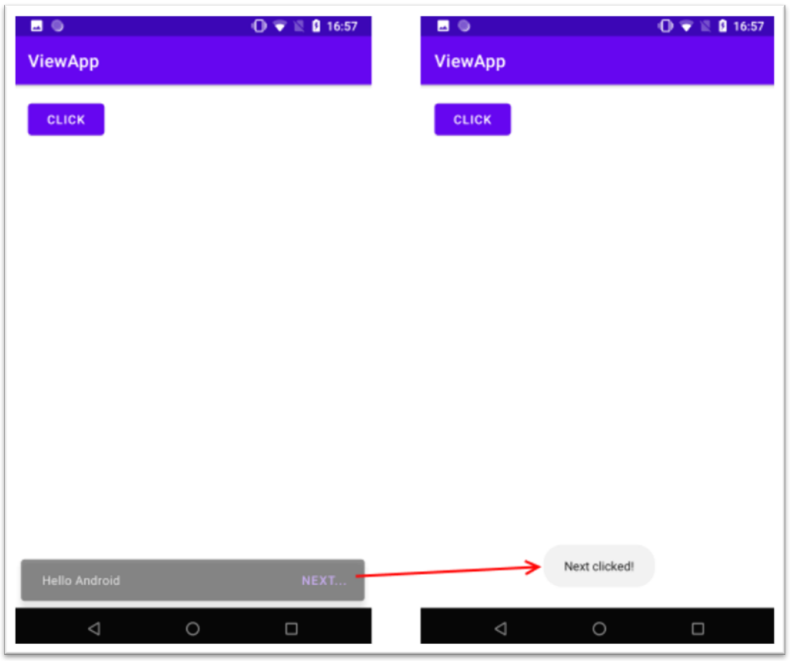
3.7. Прикріплення оброблювача події

Snackbar дозволяє додати віджету дію, щоб користувач міг якось відреагувати на повідомлення. Наприклад, змінимо код MainActivity наступним чином:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import com.google.android.material.snackbar.Snackbar;

public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onClick(View view) {
Snackbar snackbar = Snackbar.make(view, "Hello Android",
Snackbar.LENGTH_LONG);

snackbar.setAction("Next...", new View.OnClickListener (){
@Override
```

```
      publicvoid onClick(View v) {
      Toast   toast   =   Toast.makeText(getApplicationContext(),
"Next clicked!", Toast.LENGTH_LONG);
      toast.show();
      }
      });
      snackbar.show();
      }
      }
```

Для додавання дії Snackbar застосовується метод setAction(). Перший параметр представляє текст кнопки у повідомленні, яку може натиснути користувач - у разі це "Next...". Другий параметр представляє реалізацію інтерфейсу View.OnClickListener (той самий, який використовується для обробки натискання кнопки). У методі onClick() виконуємо дії, які викликаються при натисканні на кнопку в повідомленні. В даному випадку для простоти просто відображаємо спливаюче повідомлення як об'єкт Toast

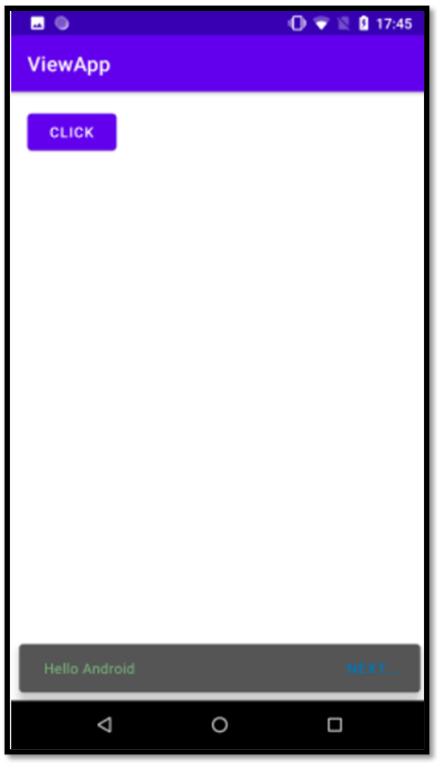


Налаштування візуального вигляду

Ряд методів Snackbar дозволяє налаштувати зовнішній вигляд:

- **setTextColor()**: налаштовує колір тексту
- **setBackgroundTint()**: налаштовує колір фону
- **setActionTextColor()**: налаштовує колір тексту кнопки у спливаючому повідомленні

```
      snackbar.setTextColor(0XFF81C784);
      snackbar.setBackgroundTint(0XFF555555);
      snackbar.setActionTextColor(0XFF0277BD);
```



3.8. **Checkbox**

Елементи Checkbox є прапорцями, які можуть перебувати у зазначеному та невідзначеному стані. Прапорці дозволяють робити множинний вибір із кількох значень. Отже, визначимо у файлі розмітки activity_main.xml елемент CheckBox:

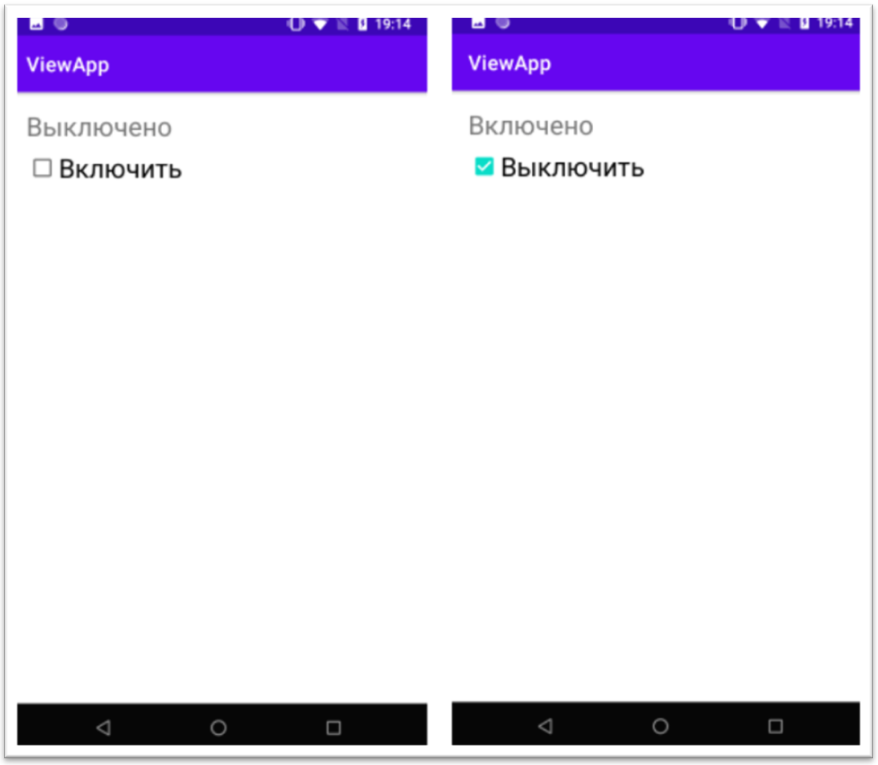
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView android:id="@+id/selection"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<CheckBox android:id="@+id/enabled"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Увімкнути"
android:textSize="26sp"
```

```
    android:onClick="onCheckboxClicked"

    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/selection"/>


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

Атрибут`android:onClick`, як і у випадку із простими кнопками, дозволяє задати обробник натискання на прапорець. Визначимо обробник натискання у коді MainActivity:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.os.Bundle;
    import android.view.View;
    import android.widget.CheckBox;
    import android.widget.TextView;

    public class MainActivity extends AppCompatActivity {

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

    public void onCheckboxClicked(View view) {
    // Отримуємо прапорець
    CheckBox checkBox = (CheckBox) view;
    TextView selection = findViewById(R.id.selection);
    // Отримуємо, чи зазначений цей прапорець
    if(checkBox.isChecked()) {
    selection.setText("Увімкнено");
    checkBox.setText("Вимкнути");
    }
    else{
    selection.setText("Вимкнено");
    checkBox.setText("Увімкнути");
    }
    }
    }
```

Як параметр натискання обробника наCheckboxClicked передається натиснений прапорець. Обробник спрацьовує при кожному натисканні checkBox. Тобто коли ми встановлюємо прапорець, і коли ми знімемо позначку. За допомогою методу isChecked() можна дізнатися, чи виділений прапорець - у цьому випадку метод повертає true.



Подібним чином можна використовувати кілька прапорців:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView android:id="@+id/selection"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<CheckBox android:id="@+id/java"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Java"
android:textSize="26sp"

android:onClick="onCheckboxClicked"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/selection"/>

<CheckBox android:id="@+id/kotlin"
android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="Kotlin"
    android:textSize="26sp"

    android:onClick="onCheckboxClicked"

    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/java"/>

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

На кожен прапорець можна повісити свій натискач. А можна зробити один, як у цьому випадку. У цьому випадку ми можемо обробити кілька прапорців у коді java за допомогою конструкції`switch...case`

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.os.Bundle;
    import android.view.View;
    import android.widget.CheckBox;
    import android.widget.TextView;
    import android.widget.Toast;

    public class MainActivity extends AppCompatActivity {

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

    public void onCheckboxClicked(View view) {
    // Отримуємо прапорець
    CheckBox checkBox = (CheckBox) view;
    // Отримуємо, чи зазначений цей прапорець
    boolean checked = checkBox.isChecked();

    TextView selection = findViewById(R.id.selection);

    // Дивимося, який саме з прапорців позначений
    switch(view.getId()) {
    case R.id.java:
    if (checked)
    Toast.makeText(this,        "Ви        обрали        Java",
Toast.LENGTH_LONG).show();
    break;
    case R.id.kotlin:
    if (checked)
```

```
      Toast.makeText(this,                "Ви                вибрали
Kotlin",Toast.LENGTH_LONG).show();
      break;
      default:
      selection.setText("");
      }
      }
      }
```

За допомогою конструкції switch...case можна отримати id натиснутого прапорця та виконати відповідні дії.

Щоправда, якщо нам просто треба взяти текст із вибраного прапорця, то необов'язково в даному випадку використовувати конструкцію switch, тому що ми можемо скоротити весь код таким чином:

```
      public void onCheckboxClicked(View view) {
      // Отримуємо прапорець
      CheckBox language=(CheckBox) view;
      // Отримуємо, чи зазначений цей прапорець
      TextView selection = findViewById(R.id.selection);
      if(language.isChecked())
      selection.setText(language.getText());
      }
```

Однак у цьому випадку залишається проблема: у текстовому полі відображається лише один виділений елемент. Змінимо код MainActivity, щоб відображати обидва виділені елементи:

```
      package com.example.viewapp;

      import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

public void onCheckboxClicked(View view) {

// Отримуємо прапорці
CheckBox java = findViewById(R.id.java);
CheckBox kotlin = findViewById(R.id.kotlin);
String selectedItems = "";
if(java.isChecked())
selectedItems +=java.getText() + "";
if(kotlin.isChecked())
selectedItems +=kotlin.getText();

TextView selection = findViewById(R.id.selection);
selection.setText(selectedItems);
}
}
```

OnCheckedChangeListener

Застосування слухача OnCheckedChangeListener є альтернативним способом відстеження зміни прапорця. Цей слухач спрацьовує, коли ми встановлюємо або забираємо позначку на прапорці. Наприклад, визначимо наступний checkbox:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView android:id="@+id/selection"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<CheckBox android:id="@+id/enabled"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Увімкнути"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/selection"/>

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

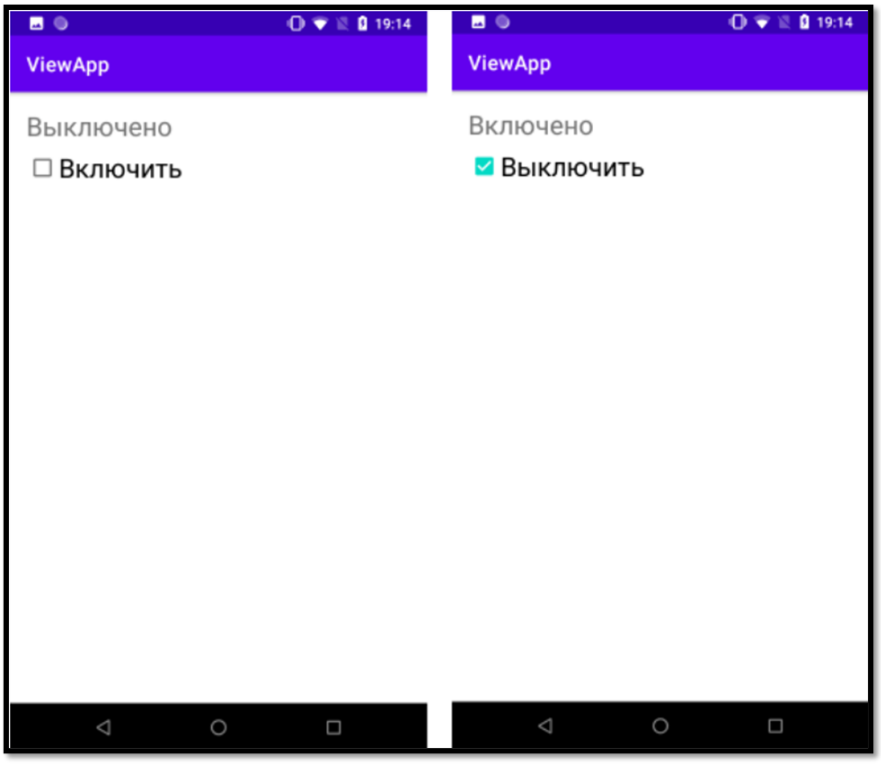
У коді MainActivity підключимо обробник зміни стану:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

TextView selection = findViewById(R.id.selection);
CheckBox enableBox = findViewById(R.id.enabled);
```

```
     enableBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
     public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {

     if(isChecked) {
     selection.setText("Увімкнено");
     buttonView.setText("Вимкнути");
     }
     else {
     selection.setText("Вимкнено");
     buttonView.setText("Увімкнути");
     }
     }
     });
     }
     }
```

Слухач OnCheckedChangeListener визначений у базовому класі CompoundButton і визначає один метод - onCheckedChanged. Перший параметр цього методу`buttonView`- Сам змінений прапорець CheckBox. А другий параметр`isChecked`вказує, чи відмічено прапорець.

При зміні стану прапорця виводитиметься у спливаючому вікні відповідне повідомлення:



ToggleButton

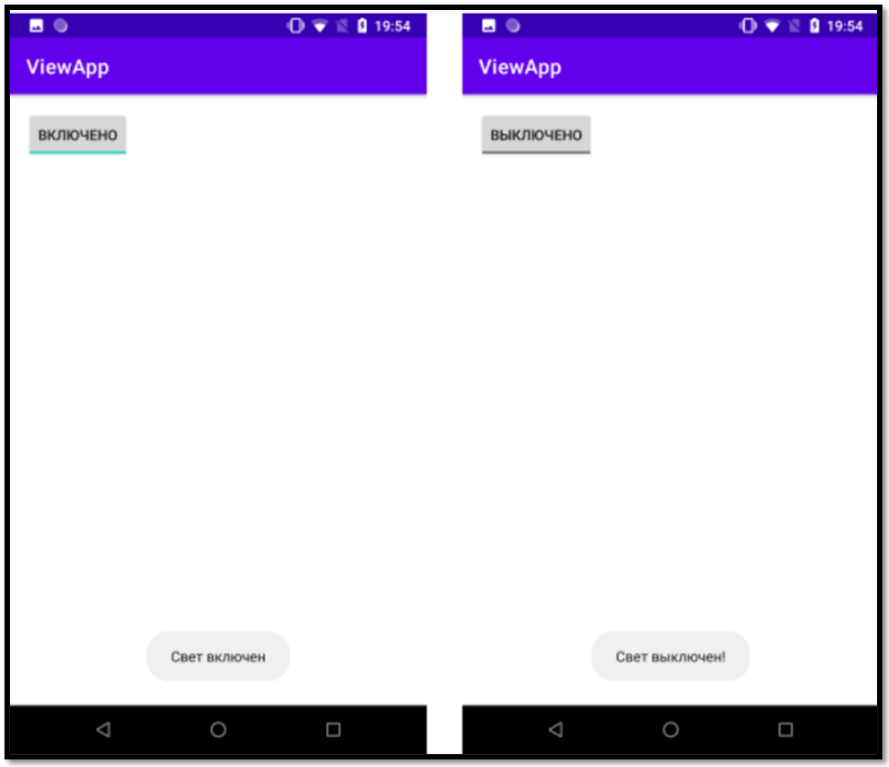
ToggleButtonподібно до елемента CheckBox може перебувати у двох станах: зазначеному та невідзначеному, причому для кожного стану ми можемо окремо встановити свій текст. Наприклад, визначимо наступний елемент ToggleButton:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<ToggleButton
android:id="@+id/toggle"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="Увімкнено"
android:textOff="Вимкнено"
android:onClick="onToggleClicked"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Атрибути android:textOn та android:textOff задають текст кнопки у зазначеному та невідзначеному стані відповідно. І, як і для інших кнопок, ми можемо обробити натискання на елемент за допомогою події onClick. У цьому випадку визначимо у класі Activity обробник події:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onToggleClicked(View view) {
```

```
      // чи включена кнопка
      boolean on = ((ToggleButton) view).isChecked();
      if (on) {
      // Дія якщо включена
      Toast.makeText(this,           "Світло           включено",
Toast.LENGTH_LONG).show();
      } else {
      // дії, якщо вимкнено
      Toast.makeText(this,           "Світло           вимкнено!",
Toast.LENGTH_LONG).show();
      }
      }
      }
```



Створення елемента ToggleButton у коді java:

```
      package com.example.viewapp;

      import androidx.appcompat.app.AppCompatActivity;
      import androidx.constraintlayout.widget.ConstraintLayout;

      import android.os.Bundle;
      import android.view.View;
      import android.widget.Toast;
      import android.widget.ToggleButton;

      public class MainActivity extends AppCompatActivity {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
```

```
      super.onCreate(savedInstanceState);
      //setContentView(R.layout.activity_main);
      ConstraintLayout layout = New ConstraintLayout(this);
      ConstraintLayout.LayoutParams    layoutParams    =    new
ConstraintLayout.LayoutParams
      (ConstraintLayout.LayoutParams.WRAP_CONTENT,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
      ToggleButton toggleButton = New ToggleButton(this);
      toggleButton.setTextOff("Вимкнено");
      toggleButton.setTextOn("Увімкнено");
      toggleButton.setText("Вимкнено");
      toggleButton.setOnClickListener(new  View.OnClickListener()
{
      @Override
      public void onClick(View view) {
      boolean on = ((ToggleButton) view).isChecked();

      if (on) {
      Toast.makeText(getApplicationContext(), "Світло включено",
Toast.LENGTH_LONG).show();
      } else {
      Toast.makeText(getApplicationContext(),            "Світло
вимкнено!", Toast.LENGTH_LONG).show();
      }
      }
      });
      layoutParams.leftToLeft                                    =
ConstraintLayout.LayoutParams.PARENT_ID;
      layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT
_ID;
      layout.addView(toggleButton);
      setContentView(layout);
      }
      }
```

RadioButton

Подібну до прапорців функціональність надають перемикачі, які представлені класом RadioButton. Але на відміну від прапорців одночасно у групі перемикачів ми можемо вибрати лише один перемикач.

Щоб створити список перемикачів для вибору, спочатку потрібно створити об'єкт RadioGroup, який буде включати всі перемикачі:

```
      <?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"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:padding="16dp">
```

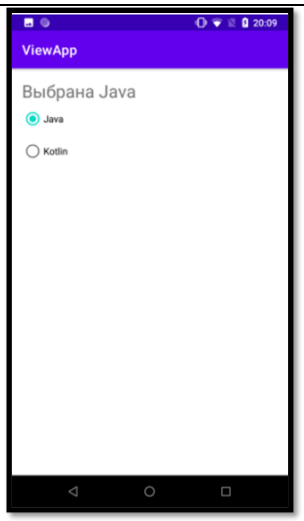
```
<TextView android:id="@+id/selection"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<RadioGroup
android:id="@+id/radios"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/selection"
>

<RadioButton android:id="@+id/java"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Java"
android:onClick="onRadioButtonClicked"/>
<RadioButton android:id="@+id/kotlin"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Kotlin"
android:onClick="onRadioButtonClicked"/>
</RadioGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Оскільки класRadioGroupє похідним відLinearLayout, то ми також можемо встановити вертикальну або горизонтальну орієнтацію списку, при тому включивши в нього не тільки власне перемикачі, але й інші об'єкти, наприклад, кнопку або TextView.

У класі MainActivity визначимо обробку вибору перемикачів:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.RadioButton;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onRadioButtonClicked(View view) {
// якщо перемикач відзначений
boolean checked = ((RadioButton) view).isChecked();
TextView selection = findViewById(R.id.selection);
// Отримуємо натиснутий перемикач
switch(view.getId()) {
case R.id.java:
if (checked) {
selection.setText("Вибрано Java");
}
break;
case R.id.kotlin:
if (checked) {
selection.setText("Вибраний Kotlin");
}
break;
}
}
}
```



OnCheckedChangeListener

Окрім обробки натискання на кожен окремий переколючувач, ми можемо в цілому повісити на весь RadioGroup з його перемикачами слухач OnCheckedChangeListener та обробляти в ньому натискання. Для цього приберемо з розмітки у перемикачів атрибути `android:onClick`, а елемента RadioGroup визначимо id:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">
```

```
    <TextView android:id="@+id/selection"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="26sp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

    <RadioGroup
    android:id="@+id/radios"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/selection">
    <RadioButton android:id="@+id/java"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Java" />
    <RadioButton android:id="@+id/kotlin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Kotlin" />
    </RadioGroup>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Далі у коді MainActivity повісимо на об'єкт`RadioGroup`слухач OnCheckedChangeListener:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.os.Bundle;
    import android.widget.RadioGroup;
    import android.widget.TextView;

    public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // отримуємо об'єкт RadioGroup
    RadioGroup radGrp = (RadioGroup)findViewById(R.id.radios);
    // обробка перемикання стану перемикача
    radGrp.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup arg0, int id) {
```

```
    TextView selection = findViewById(R.id.selection);
    switch(id) {
    case R.id.java:
    selection.setText("Вибрано Java");
    break;
    case R.id.kotlin:
    selection.setText("Вибраний Kotlin");
    break;
    default:
    break;
    }
    }});
    }}
```

Слухач RadioGroup.OnCheckedChangeListener визначає метод onCheckedChanged(), який передається об'єкт RadioGroup і id виділеного перемикача. Далі також ми можемо перевірити id та виконати певну обробку.

3.9. **DatePicker**

DatePickerпредставляє елемент вибору дати. Серед його атрибутів можна назвати такі:

- **android:calendarTextColor**: колір тексту календаря
- **android:calendarViewShown**: вказує, чи відображатиметься календар.
- **android:datePickerMode**: встановлює режим вибору дати
- **android:dayOfWeekBackground**: встановлює фоновий колір панелі вибору дня тижня.
- **android:endYear**: встановлює останній рік, що відображається.
- **android:firstDayOfWeek**: встановлює перший день тижня
- **android:headerBackground**: встановлює колір фону для панелі обраної дати
- **android:maxDate**: встановлює максимальну дату у форматі mm/dd/yyyy.
- **android:minDate**: встановлює мінімальну дату у форматі mm/dd/yyyy.
- **android:spinnersShown**: вказує, чи буде відображатися спіннер у віджеті
- **android:startYear**: встановлює початковий рік, що відображається.

- **android:yearListSelectorColor**: встановлює колір для поля вибору року

Серед методів DatePicker можна назвати такі:

- `int getDayOfMonth()`: повертає номер вибраного дня
- `int getMonth()`: повертає номер обраного місяця (від 0 до 11)
- `int getYear()()`: повертає номер обраного року
- `void init(int year, int monthOfYear, int dayOfMonth, DatePicker.OnDateChangedListener onDateChangedListener)`: встановлює початкову дату Останній параметр встановлює слухач зміни вибраної дати
- `void setOn-?ateChangedListener(DatePicker.OnDateChangedListener onDateChangedListener)`: встановлює слухач зміни вибраної дати
- `void setFirstDayOfWeek(int firstDayOfWeek)`: встановлює перший день тижня
- `void updateDate(int year, int month, int dayOfMonth)`: програмно оновлює вибрану дату

Нехай в activity_main.xml визначено елемент DatePicker:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView android:id="@+id/dateTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />

<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/dateTextView" />

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

Застосуємо деякі методи DatePicker для управління його поведінкою:

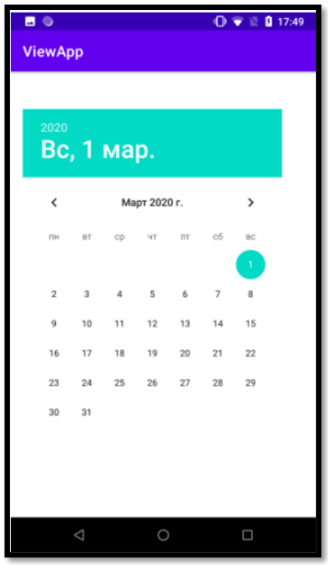
```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

TextView dateTextView = findViewById(R.id.dateTextView);
DatePicker datePicker =
this.findViewById(R.id.datePicker);

// Місяць починаючи з нуля. Для відображення додаємо 1.
datePicker.init(2020, 02, 01, new
DatePicker.OnDateChangedListener() {
@Override
public void onDateChanged(DatePicker view, int year, int
monthOfYear, int dayOfMonth) {

// Відлік місяців починається з нуля. Для відображення
додаємо 1.
dateTextView.setText("Дата: " + view.getDayOfMonth() + "/"
+
(view.getMonth() + 1) + "/" + view.getYear());

// альтернативний запис
// dateTextView.setText("Дата: " + dayOfMonth + "/" +
(monthOfYear + 1) + "/" + year);
}
});
}
}
```

Використовуючи метод`datePicker.init();`встановлюємо дату за замовчуванням – 1 березня 2020 року, оскільки відлік місяців іде з нуля. Крім того, за допомогою останнього параметра – об'єкта DatePicker.OnDateChangedListener встановлюється обробка вибору дати. Щоразу, коли користувач вибирає дату, буде спрацьовувати метод onDateChanged() об'єкта DatePicker.OnDateChangedListener. Цей метод

приймає чотири параметри - view (елемент DatePicker), year (вибраний рік), monthOfYear (вибраний місяць), dayOfMonth (вибраний день).

Далі ми можемо отримати вибрані день, місяць та рік. Причому можна використовувати як параметри методу onDateChanged, так і методи самого DatePicker

Початковий стан перед вибором – встановлено дату 1 березня 2020 року.



Вибір довільної дати (20 травня 2020 року):

DatePicker за замовчуванням відображається в режимі календаря, але ми можемо використовувати додати інший режим - спіннер за допомогою атрибута`android:datePickerMode`:

```
<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/dateTextView" />
```

У цьому випадку спіннер відображається зліва від календаря. Якщо ми зовсім не хочемо відображати календар, то можна встановити атрибут`android:calendarViewShown="false"`

```
<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"
android:calendarViewShown="false"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/dateTextView" />
```

3.10. TimePicker

TimePickerпредставляє віджет для вибору часу, який може відображати час або у 24-годинному або у 12-годинному форматі.

Серед атрибутів TimePicker слід виділити timePickerMode, який дозволяє режим відображення і може приймати одне з двох значень: clock (відображення у вигляді годинника) і spinner (відображення у вигляді спіннера).

Серед методів TimePicker можна назвати такі:

- `int getHour()`: повертає годину (у 24-годинному форматі)
- `int getMinute()`: повертає хвилини
- `boolean is24HourView()`: повертає true, якщо використовується 24-годинний формат
- `void setHour(int hour)`: встановлює годину для TimePicker
- `void setIs24HourView(Boolean is24HourView)`: встановлює 24-годинний формат
- `void setMinute(int minute)`: встановлює хвилини

- `void setOnTimeChangedListener(TimePicker.OnTimeChangedListener on-TimeChangedListener)`: встановлює слухач зміни часу в TimePicker як об'єкт TimePicker.OnTimeChangedListener

Визначимо TimePicker у activity_main.xml:

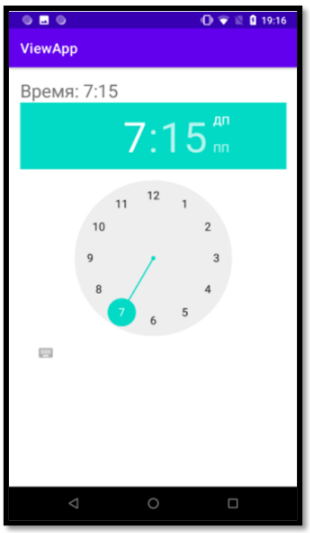
```
<?xmlversion="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextViewandroid:id="@+id/timeTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TimePickerandroid:id="@+id/timePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/timeTextView"/>

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

Застосуємо деякі методи TimePicker для керування його поведінкою:

```
packagecom.example.viewapp;

importandroidx.appcompat.app.AppCompatActivity;

importandroid.os.Bundle;
importandroid.widget.TextView;
importandroid.widget.TimePicker;

publicclass MainActivity extends AppCompatActivity {

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

TextView timeTextView = findViewById(R.id.timeTextView);
TimePicker timePicker = findViewById(R.id.timePicker);

timePicker.setOnTimeChangedListener(newTimePicker.OnTimeChangedLi
```

```
stener() {
      @Override
      publicvoid onTimeChanged(TimePicker view, int hourOfDay, int minute) {

      timeTextView.setText("Час:  "+ hourOfDay + ":" + minute);
      // або так
      //  timeTextView.setText("Час:"  +  view.getHour()  +  ":"  +
view.getMinute());
      }
      });
      }
      }
```

Для додавання слухача зміни часу в TimePicker застосовується метод setOnTimeChangedListener(), який передається об'єкт TimePicker.OnTimeChangedListener. Він має один метод - onTimeChanged(), який викликається при кожній зміні часу в TimePicker. Цей метод приймає три параметри - сам елемент TimePicker, hourOfDay - встановлений час і minute - встановлені хвилини. В даному випадку просто передаємо значення вибраного часу TextView.



TimePicker за замовчуванням відображається в режимі "clock" або годинник. Застосуємо режим "spinner":

```
      <TimePickerandroid:id="@+id/timePicker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:timePickerMode="spinner"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintTop_toBottomOf="@+id/timeTextView"/>
```

3.11. Повзунок SeekBar

Елемент SeekBar виконує роль повзунка, тобто шкалу поділів, де ми можемо змінювати поточну позначку.

Серед його атрибутів можна назвати такі:

- **android:max**: встановлює максимальне значення
- **android:min**: встановлює мінімальне значення
- **android:progress**: встановлює поточне значення, яке знаходиться в діапазоні між мінімальним та максимальним

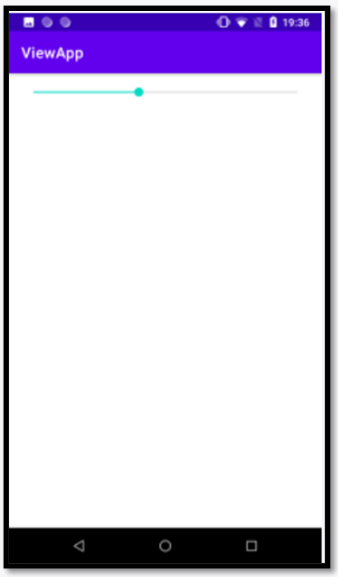
Для управління SeekBar визначає ряд методів, у тому числі виділимо такі:

- **void setProgress(int progress)**: встановлює поточне значення повзунка
- **void setMin(int min)**: встановлює мінімальне значення
- **void setMax(int max)**: встановлює максимальне значення
- **void incrementProgressBy(int diff)**: збільшує поточне значення на diff
- **int getMax()**: повертає максимальне значення
- **int getMin()**: повертає мінімальне значення
- **int getProgress()**: повертає поточне значення
- **void setOn-?eekBarChangeListener(SeekBar.OnSeekBarChangeListener l)**: встановлює слухача зміни значення SeekBar

Визначимо SeekBar у розмітці layout:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<SeekBar
android:id="@+id/seekBar"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:progress="20"
android:max="50"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />

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

Атрибут`android:progress`задає число 20 як поточне значення повзунка, а атрибут`android:max`- максимально можливе значення – число 50. У результаті ми отримаємо наступний елемент:



Тепер використовуємо метод setOnSeekBarChangeListener(), який дозволяє встановити обробники подій зміни значення повзунка. Так, визначимо у файлі layout наступний код:

```
<?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"
android:layout_width="match_parent"
```

```
      android:layout_height="match_parent"
      android:padding="16dp">

      <TextView android:id="@+id/seekBarValue"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="26sp"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintTop_toTopOf="parent" />

      <SeekBar
      android:id="@+id/seekBar"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:progress="20"
      android:max="50"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintTop_toBottomOf="@+id/seekBarValue"
/>

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

Тут визначено елемент TextView, який виводитиме поточне значення повзунка під час його зміни.

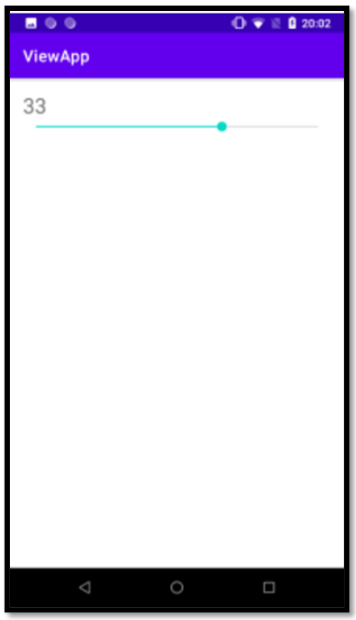
І змінимо код MainActivity:

```
      package com.example.viewapp;

      import androidx.appcompat.app.AppCompatActivity;

      import android.os.Bundle;
      import android.widget.SeekBar;
      import android.widget.TextView;
      import android.widget.TimePicker;

      public class MainActivity extends AppCompatActivity {

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

      SeekBar seekBar = findViewById(R.id.seekBar);
      TextView textView = findViewById(R.id.seekBarValue);
      seekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
      @Override
      public void onProgressChanged(SeekBar seekBar, int
progress, boolean fromUser) {
```

```
        textView.setText(String.valueOf(progress));
        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
        }});
        }}
```

У методsetOnSeekBarChangeListener()передається об'єкт SeekBar.OnSeekBarChangeListener, який дозволяє встановити три методи-обробники:

- onProgressChanged: спрацьовує під час перетягування повзунка за шкалою. Параметр, що передається в метод, дозволяє отримати нове значення повзунка, яке в даному випадку передається в TextView для відображення на екрані
- onStartTrackingTouch: спрацьовує при початку перетягування повзунка за шкалою
- onStopTrackingTouch: спрацьовує при завершенні перетягування повзунка за шкалою



Також ми можемо отримати поточне значення повзунка, використовуючи метод getProgress():

```
    public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {

    textView.setText(String.valueOf(seekBar.getProgress()));
    }
```

4. РЕСУРСИ

4.1. Робота з ресурсами

Ресурс у програмі Android є файл, наприклад, файл розмітки інтерфейсу або деяке значення, наприклад, простий рядок. Тобто ресурси є і файли розмітки, і окремі рядки, і звукові файли, файли зображень тощо. Усі ресурси знаходяться у проекті в каталозі res. Для різних типів ресурсів, визначених у проекті, у каталозі res створюються підкаталоги. Підтримувані підкаталоги:

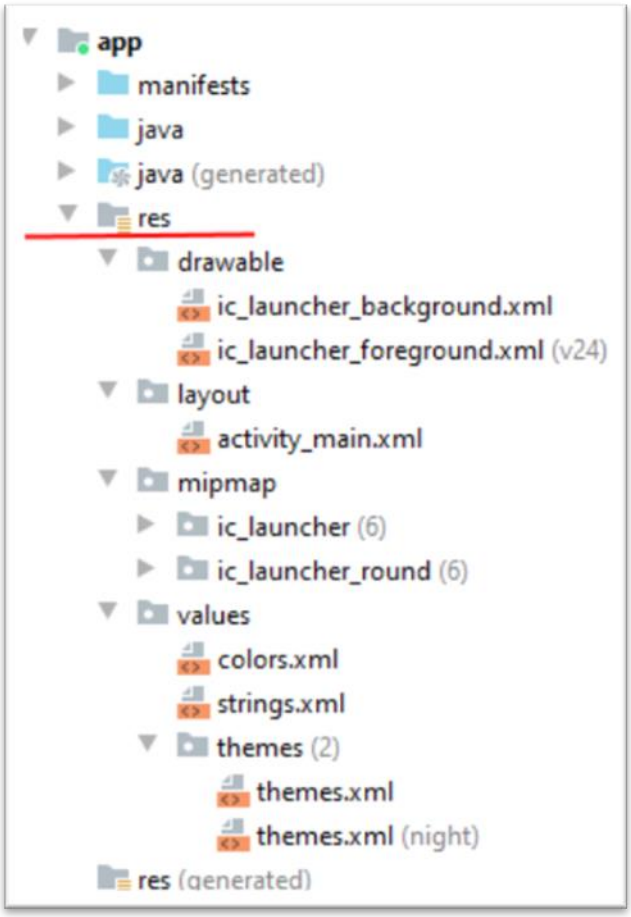
- **animator/**: XML-файли, що визначають анімацію властивостей
- **anim/**: XML-файли, що визначають tween-анімацію
- **color/**: XML-файли, що визначають список кольорів
- **drawable/**: Графічні файли (.png, .jpg, .gif)
- **mipmap/**: Графічні файли, які використовуються для іконок програми під різні роздільні здатності екранів
- **layout/**: xml-файли, що визначають інтерфейс користувача.
- **menu/**: xml-файли, що визначають меню програми
- **raw/**: різні файли, які зберігаються у вихідному вигляді
- **values/**: xml-файли, які містять різні значення, що використовуються в додатку, наприклад, ресурси рядків
- **xml/**: Довільні xml-файли
- **font/**: файли з визначеннями шрифтом та розширеннями .ttf, .otf або .ttc, або файли XML, які містять елемент <font-family>

Загалом ми можемо визначити такі типи ресурсів:

Ресурс	Каталог проекту	Файл	елемент у файлі
Рядки	/res/values/	strings.xml	<string>
Plurals	/res/values/	strings.xml	<plurals>
Масиви рядків	/res/values/	strings.xml або arrays.xml	<string-array>
Логічні значення Boolean	/res/values/	bools.xml	<bool>

кольору	/res/values/	colors.xml	<color>
Список квітів	/res/color/	Довільна назва	<selector>
Розміри (Dimensions)	/res/values/	dimens.xml	<dimen>
Ідентифікатори ID	/res/values/	ids.xml	<item>
Цілі числа	/res/values/	integers.xml	<integer>
Масив цілих чисел	/res/values/	integers.xml	<integer-array>
Графічні файли	/res/drawable/	Файли з розширенням jpg та png	-
Tween-анімація	/res/anim/	Файл xml із довільною назвою	<set>, <alpha>, <rotate>, <scale>, <translate>
Покадрова анімація	/res/drawable/	Файл xml із довільною назвою	<animation-list>
Анімація властивостей	/res/animator/	Файл xml із довільною назвою	<set>, <objectAnimator>, <valueAnimator>
Меню	/res/menu/	Файл xml із довільною назвою	<menu>
XML-файли	/res/xml/	Файл xml із довільною назвою	
Бінарні та текстові ресурси	/res/raw/	Мультимедійні файли (mp3, mp4), текстові та інші файли	
Розмітка графічного інтерфейсу	/res/layout/	Файл xml із довільною назвою	
Стилі та теми	/res/values/	styles.xml, themes.xml	<style>

Наприклад, якщо ми візьмемо стандартний проект Android Studio, який створюється за замовчуванням, там можемо помітити наявність вже декількох папок для різних ресурсів в каталозі res.



За замовчуванням тут є каталоги не для всіх типів ресурсів, які використовуватимуться в Android, проте при необхідності ми можемо додати в папку res потрібний каталог, а потім помістити ресурс.

Коли відбувається компіляція проекту, відомості про всі ресурси додаються до спеціального файлу.R.jar, який потім використовується при роботі з ресурсами

Застосування ресурсів

Існує два способи доступу до ресурсів: у файлі вихідного коду та у файлі xml.

4.2. Посилання на ресурси у коді

Тип ресурсу в даному записі посилається на один із просторів (вкладених класів), визначених у файлі R.java, які мають відповідні їм типи в xml:

- `R.drawable`(йому відповідає тип у xml drawable)
- `R.id`(id)
- `R.layout`(layout)
- `R.string`(string)
- `R.attr`(attr)
- `R.plural`(plurals)
- `R.array`(string-array)

Наприклад, для встановлення ресурсу activity_main.xml як графічний інтерфейс у коді MainActivity у методі onCreate() є такий рядок:

```
setContentView(R.layout.activity_main);
```

Через вираз R.layout.activity_main ми посилаємося на ресурс activity_main.xml, де layout - тип ресурсу, а activity_main - ім'я ресурсу.

Подібним чином ми можемо отримати інші ресурси. Наприклад, у файлі res/values/strings.xml визначено ресурс app_name:

```
<resources>
<stringname="app_name">ViewApp</string>
</resources>
```

Цей ресурс посилається на рядок. Щоб отримати посилання на цей ресурс у коді java, ми можемо використовувати вираз R.string.app_name.

4.3. Доступ до файлу xml

Нерідко виникає необхідність посилатися на ресурс у файлі xml, наприклад, у файлі, який визначає візуальний інтерфейс, наприклад, activity_main.xml. Посилання на ресурси у файлах xml мають таку формалізовану форму: @[ім'я_пакета:]тип_ресурсу/ім'я_ресурсу

- `ім'я_пакету`представляє ім'я пакета, у якому ресурс перебуває (вказувати необов'язково, якщо ресурс перебуває у тому пакеті)
- `тип_ресурсу`представляє підклас, визначений у класі R для типу ресурсу
- `ім'я_ресурсу`ім'я файлу ресурсу без розширення або значення атрибута android:name у XML-елементі (для простих значень).

Наприклад, ми хочемо вивести в елемент TextView рядок, визначений у вигляді ресурсу у файлі strings.xml:

```
<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/app_name"/>
```

В даному випадку властивість text як значення отримуватиме значення рядкового ресурсу app_name.

4.4. Метод getResources

Для отримання ресурсів у класі Activity ми можемо використовувати метод getResources(), який повертає об'єкт android.content.res.Resources. Але щоб отримати сам ресурс, нам потрібно отримати об'єкт Resources викликати один з методів. Деякі з його методів:

- **getString()**: повертає рядок з файлу strings.xml за числовим ідентифікатором
- **getDimension()**: повертає числове значення - ресурс dimen
- **getDrawable()**: повертає графічний файл як об'єкт Drawable
- **getBoolean()**: повертає значення boolean

- **getColor()**: повертає визначення кольору
- **getColorStateList()**: повертає об'єкт ColorStateList - набір кольорів
- **getFont()**: повертає визначення шрифту як об'єкт Typeface
- **getFloat()**: повертає значення float
- **getLayout()**: повертає об'єкт XmlResourceParser, пов'язаний із файлом layout

Це лише деякі методи. Але всі вони як параметр приймають ідентифікатор ресурсу, який треба отримати. Коротко розглянемо їхнє застосування. Візьмемо той самий файл res/values/strings.xml як джерело ресурсів, який у моєму випадку виглядає так:

```
<resources>
<stringname="app_name">ViewApp</string>
</resources>
```

І змінимо код MainActivity:

```
packagecom.example.viewapp;
importandroidx.appcompat.app.AppCompatActivity;
importandroid.os.Bundle;
importandroid.widget.TextView;
publicclass MainActivity extends AppCompatActivity {
@Override
protectedvoid onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_main);
// Отримання ресурсів з файлу values/strings.xml
String app_name =
getResources().getString(R.string.app_name);
TextView textView = NewTextView(this);
textView.setTextSize(30);
textView.setText(app_name);
setContentView(textView);
}
}
```

Тут, використовуючи метод getResources() отримуємо всі ресурси і потім використовуємо їх для встановлення значень властивостей графічних елементів. При запуску програми ми побачимо застосування отриманих ресурсів:

Подібним чином ми можемо програмно отримувати й інші ресурси та використати їх у додатку. Однак слід зазначити, що в даному випадку нам не потрібно використовувати метод getResources() і взагалі робити якісь певні дії щодо отримання ресурсу, оскільки метод setText() класу TextView підтримує пряме встановлення тексту за ідентифікатором ресурсу:

```
textView.setText(R.string.app_name);
```

4.5. Ресурси рядків

Ресурси рядків – один із важливих компонентів програми. Ми використовуємо їх при виведенні назви програми, різного тексту, наприклад тексту кнопок і т.д.

XML-файли, що є ресурсами рядків, знаходяться в проекті в папці res/values. За замовчуванням ресурси рядків знаходяться у файлі strings.xml, який може виглядати так:

```
<resources>
<string name="app_name">ViewApp</string>
</resources>
```

У найпростішому вигляді цей файл визначає один ресурс "app_name", який встановлює назву програми, яку ми бачимо у заголовку програми на екрані пристрою. Але, природно, ми можемо визначити будь-які рядкові ресурси. Кожен окремий ресурс визначається за допомогою елемента string, а його атрибутnameмістить назву ресурсу.

Потім у додатку у файлах коду ми можемо посилатися на ці ресурси:

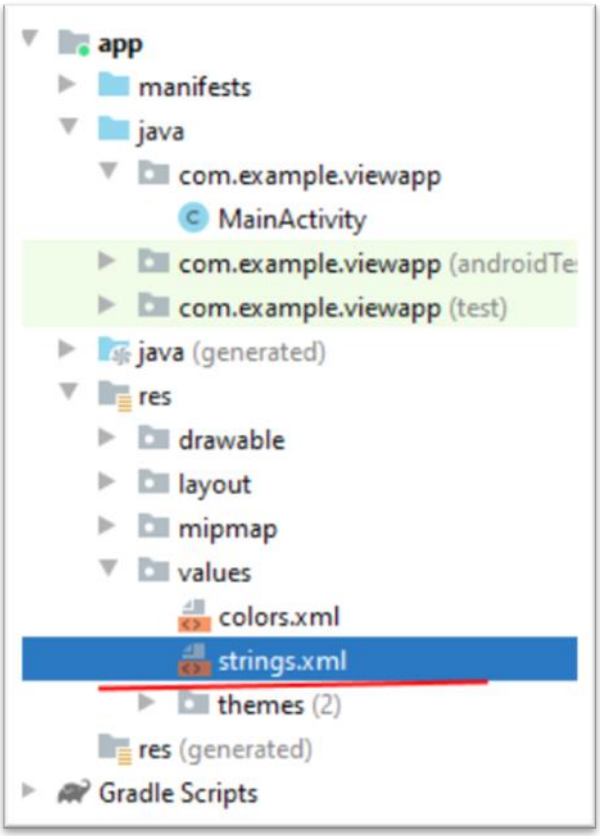
```
R.string.app_name
```

Наприклад, у коді Java:

```
String application_name = getResources().getString(R.string.app_name);
```

Або в XML-файлі:

```
@string/app_name
```

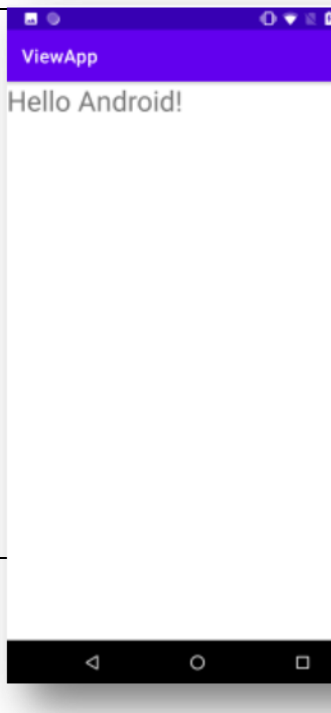


Наприклад, змінимо файл res/values/strings.xml таким чином:

```
<resources>
<string name="app_name">ViewApp</string>
<string name="message">Hello Android!</string>
</resources>
```

Тут доданий ресурс`message`зі значенням "Hello Android!". Тепер використовуємо ресурс у файлі activity_main.xml:

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

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/message"
android:textSize="30sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```



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

За допомогою виразу@string/messageпередаємо атрибутуandroid:textзначення із ресурсу. Аналогічно ми могли б використовувати ресурс у коді Activity:

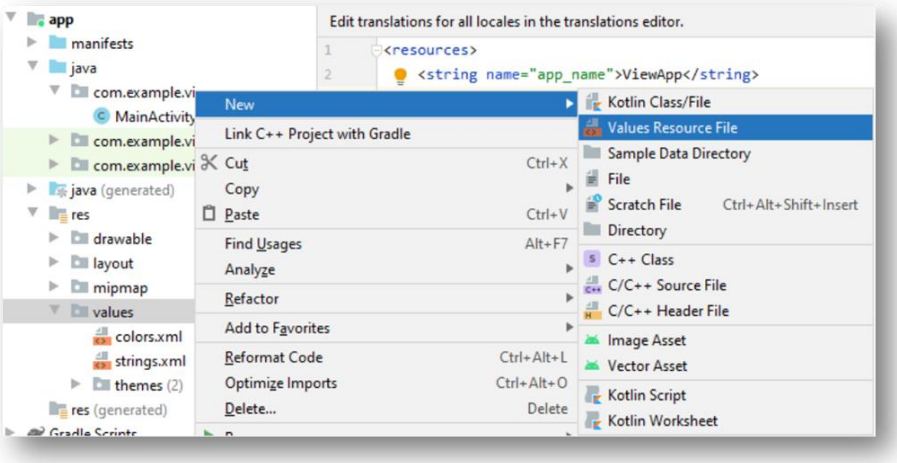
```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

// Отримуємо елемент textView
TextView textView = findViewById(R.id.textView);
// встановлюємо у нього текст
textView.setText(R.string.message);
}
}
```

Якщо нам у принципі треба отримати ресурс у коді Java (необов'язково для встановлення тексту в TextView), то в цьому випадку можна використовувати методgetResources().getString(ідентифікатор_ресурсу)

```
String message = getResources().getString(R.string.message);
```

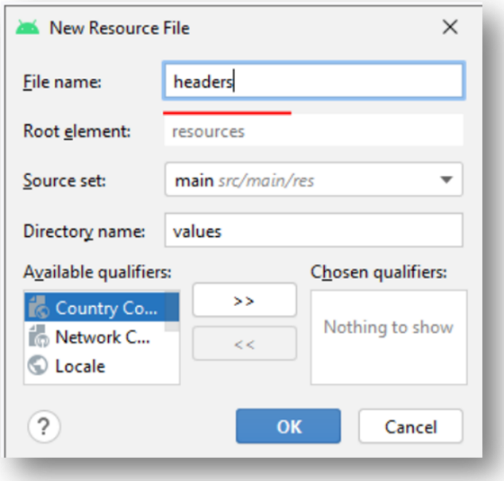
Хоча за замовчуванням для ресурсів рядків застосовується файл strings.xml, але розробники можуть додавати додаткові файли ресурсів до каталогу проекту res/values. При цьому достатньо дотримуватися структури файлу: він повинен мати кореневий вузол<resources>і мати один або кілька елементів<string>.

Так, натиснемо на папку res/values правою кнопкою миші і в списку виберемо пункт New -> Value Resource File:



При цьому слід зазначити, що цей тип файлів буде характерним для будь-якого типу ресурсів, який додається до папки res/values.

Після цього нам буде запропоновано визначити для файлу ім'я:



Назвемо, наприклад, headers (назва файлу довільне), а всіх інших полів залишимо значення за промовчанням. І в папку res/values буде додано новий файл headers.xml. Визначимо у ньому пару ресурсів:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="welcome">Ласкаво просимо</string>
<string name="click_button">Натисніть на кнопку</string>
</resources>
```

І після цього ми зможемо використовувати певні ресурси в коді Activity або у файлі layout.

4.6. **Форматування рядків**

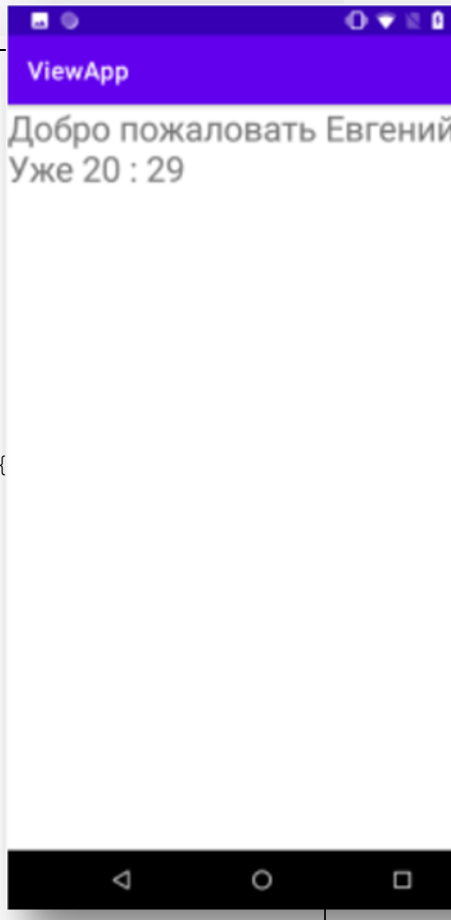
Android дозволяє застосовувати до ресурсів рядків форматування. Наприклад, змінимо файл strings.xml:

```
<resources>
<string name="app_name">ViewApp</string>
```

```
    <string name="message">Hello Android!</string>
    <string name="welcome_message">Ласкаво просимо %1$s! Вже %2$d: %3$d</string>
    </resources>
```

Третій ресурс - welcome_message представляє рядок із форматуванням. Так, вона містить такі символи як %1$s, %2$d та %3$d. Що це означає?%1$sвказує, що це перший аргумент, а символ "s" каже, що цей аргумент є рядком.%2$dпредставляє другий аргумент, а символ "d" наприкінці вказує, що це буде ціле число. Аналогічно%3$dпоказує, що це третій аргумент, який є цілим числом.

Отримаємо ресурс у коді Java




```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.content.res.Resources;
    import android.os.Bundle;
    import android.widget.TextView;

    import java.util.Calendar;

    public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //setContentView(R.layout.activity_main);
    Resources res = getResources();

    String userName = "Євген";
    Calendar calendar = Calendar.getInstance();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    int minute = calendar.get(Calendar.MINUTE);

    String text = getString(R.string.welcome_message, userName, hour, minute);
    TextView textView = New TextView(this);
    textView.setText(text);
    textView.setTextSize(28);
    setContentView(textView);
    }
    }
```

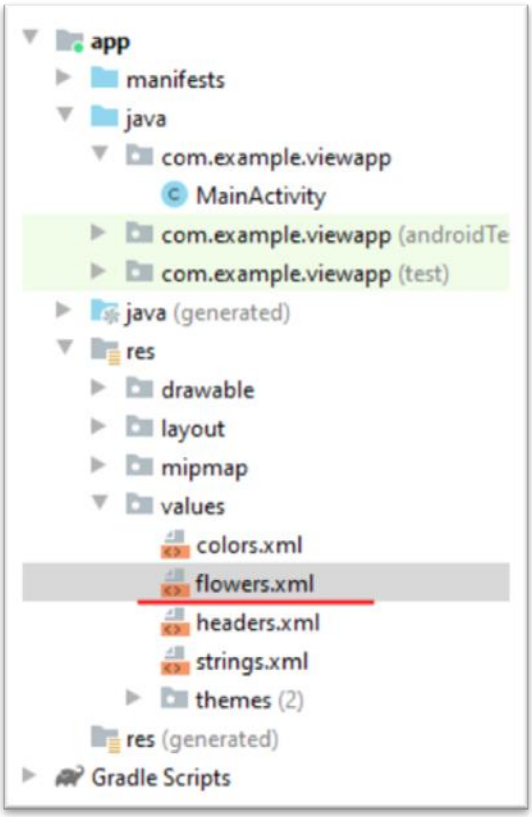
МетодgetString(R.string.welcome_message, userName, hour, minute)отримує ресурс welcome_message і як наступні параметри передає його аргументів значення. Для першого аргументу-рядка використовується змінна userName, а для другого та третього аргументів

передаємо поточну кількість годин та хвилин, отриманих за допомогою класу Calendar.

4.7. **Ресурси Plurals**

Plurals є ще одним видом набору рядків. Він призначений для опису кількості елементів. Навіщо це треба? Наприклад, візьмемо іменник: часто воно змінює закінчення в залежності від чисельного, яке з ним використовується: 1 квітка, 2 квітки, 5 квіток. Для подібних випадків і використовується ресурсplurals.

Подивимося на прикладі. Додамо до папки res/values новий ресурс. Назвемо його flowers:



Змінимо його вміст так:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<plurals name="flowers">
<item quantity="one">%d квітка</item>
<item quantity="few">%d квітки</item>
<item quantity="many">%d квіток</item>
</plurals>
</resources>
```

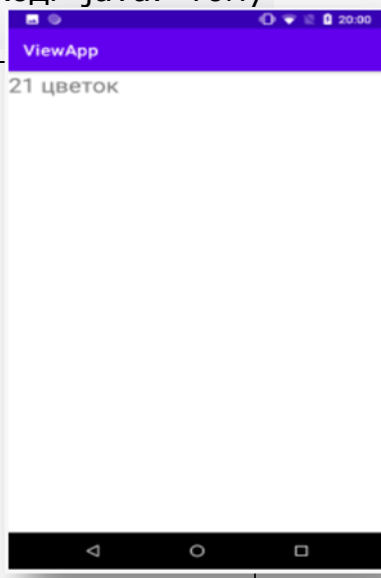
Для завдання ресурсу використовується елемент<plurals>, для якого існує атрибутname, Який отримує значення довільне назва, яким потім посилаються даний ресурс.

Самі набори рядків вводяться дочірніми елементами<item>. Цей елемент має атрибутquantity, який має значення, що вказує, коли цей рядок використовується. Цей атрибут може приймати такі значення:

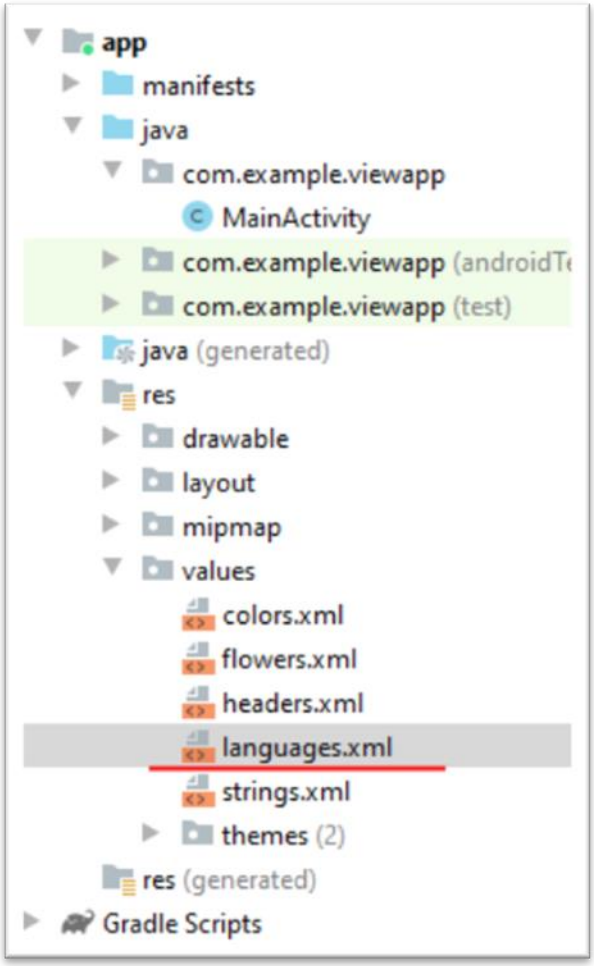
- **zero**: рядок для кількості у розмірі 0
- **one**: рядок для кількості у розмірі 1 (для російської мови - для завдання всіх кількостей, що закінчуються на 1, крім 11)
- **two**: рядок для кількості у розмірі 2
- **few**: рядок для невеликої кількості
- **many**: рядок для великої кількості
- **other**: всі інші випадки

Причому в цьому випадку багато залежить від конкретної мови. А система сама дає змогу визначити, яке значення брати для того чи іншого числа.

Використання цього ресурсу можливо тільки в коді java. Тому змінимо код MainActivity:




```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_main);
String rose = getResources().getQuantityString(R.plurals.flowers, 21,
21);
TextView textView = New TextView(this);
textView.setText(rose);
textView.setTextSize(26);
setContentView(textView);
}
}
```



За допомогою методу getQuantityString ми отримуємо значення ресурсу. Першим параметром передаємо ідентифікатор ресурсу. Другим параметром іде значення. для якого потрібно знайти потрібний рядок. Третій параметр є значення, яке вставлятиметься на місце плейсхолдера %d. Тобто ми отримуємо рядок для 21.

string array

Ще одним видом рядкових ресурсів є string-array або масив рядків. Наприклад, додамо до папки res/values новий файл, який назвемо languages.xml.

Нехай він міститиме наступний код:

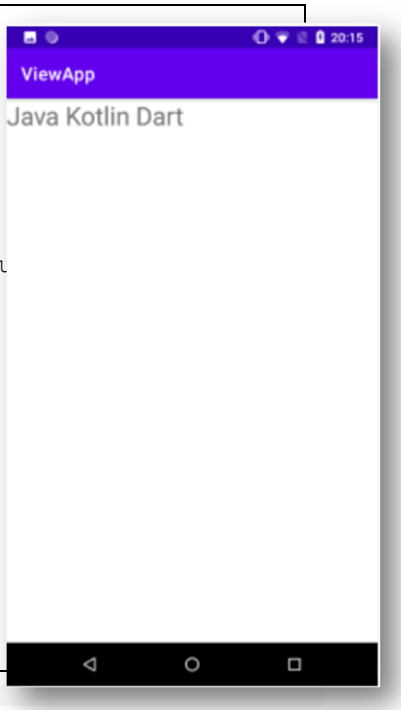
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="languages">
<item>Java</item>
<item>Kotlin</item>
<item>Dart</item>
</string-array>
</resources>
```

Ресурс визначається за допомогою елемента <string-array>. Фактично він визначає набір рядків. А кожен окремий рядок задається за допомогою елемента <item>

У файлі MainActivity.java визначимо код для отримання значень із цього ресурсу:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_main);
Resources res = getResources();
String[] languages = res.getStringArray(R.array.langu
String allLangs = "";
for (String lang: languages) {
allLangs += lang + " ";
}
TextView TextView = New TextView(this);
textView.setText(allLangs);
textView.setTextSize(28);
setContentView(textView);
}
}
```



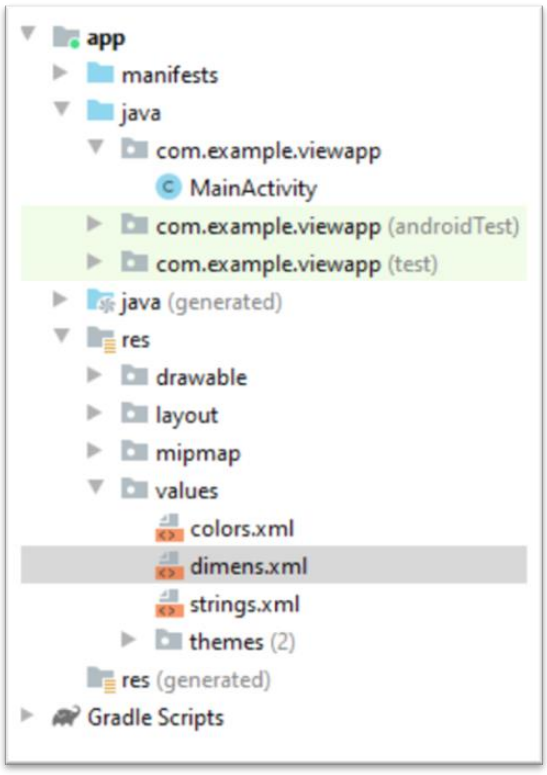
За допомогою методу getStringArray отримуємо ресурс у масив рядків, а потім за допомогою циклу складаємо з масиву один рядок і передаємо його в TextView.

4.8. **Ресурси dimension**

Визначення розмірів має знаходитись у папці res/values у файлі з будь-яким довільним ім'ям. Загальний синтаксис визначення ресурсу:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<dimen name="ім'я_ресурсу">використовуваний_розмір</dimen>
</resources>
```

Як і інші ресурси, ресурс dimension визначається кореневим елементом<resources>. Тег<dimen>позначає ресурс і як значення приймає деяке значення розміру в одній з прийнятих одиниць виміру (dp, sp, pt, px, mm, in). Докладніше установка розмірів розглядалася в одній із минулих статей -Визначення розмірів



Так, додамо в Android Studio до папки res/values новий елемент Values Resources File, який назвемо dimens.xml.

Визначимо в ньому такий вміст:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<dimen name="horizontal_margin">64dp</dimen>
<dimen name="vertical_margin">32dp</dimen>
<dimen name="text_size">32sp</dimen>
</resources>
```

Тут визначено два ресурси для відступівhorizontal_marginivertical_margin, які зберігають відповідно значення 64dp та 32dp, та ресурс text_size, який зберігає висоту шрифту - 32sp. Назви ресурсів може бути довільними.

Використовуємо ресурс у файлі activity_main.xml:

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

<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android"
android:background="#eaeaea"

android:layout_marginTop="@dimen/vertical_margin"
android:layout_marginLeft="@dimen/horizontal_margin"
android:textSize="@dimen/text_size"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />

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

Ресурси dimension використовуються для таких атріубтів візуальних елементів, які як значення вимагають вказівки числового значення. Наприклад, атріубутandroid:layout_heightабоandroid:text Size. Для отримання ресурсу в xml після"@dimen/"вказується ім'я ресурсу.

Для отримання ресурсів коду java застосовується метод getDimension() класу Resources. Наприклад, визначимо в коді Java аналогічний візуальний інтерфейс:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.content.res.Resources;
import android.os.Bundle;
import android.util.TypedValue;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_main);
// Отримуємо ресурси
Resources resources = getResources();
float textSize = resources.getDimension(R.dimen.text_size);
int hMargin = (int)resources.getDimension(R.dimen.horizontal_margin);
int vMargin = (int)resources.getDimension(R.dimen.vertical_margin);

ConstraintLayout constraintLayout = New ConstraintLayout(this);

ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
```

```
layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT_ID;

TextView textView = New TextView(this);
textView.setText("Hello Android");
textView.setBackgroundColor(0XFFEAEAEA);
// встановлюємо розмір шрифту за ресурсом
textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
// встановлюємо відступи відповідно до ресурсів
layoutParams.setMargins(hMargin, vMargin, hMargin, vMargin);

textView.setLayoutParams(layoutParams);
constraintLayout.addView(textView);

setContentView(constraintLayout);
}
}
```

При програмному одержанні ресурсів dimen за допомогою методуgetDimension()слід враховувати, що він повертає значення у фізичних пікселях, навіть якщо у файлі ресурсів ми визначили значення інших величин (32sp, 64dp). У більшості випадків це не викличе якихось труднощів, оскільки більшість методів Java приймають саме пікселі, а не dp або інші одиниці, як наприклад, методsetMargins()що встановлює відступи.

Проте методsetTextSize()приймає саме sp, тому за допомогою додаткового параметра необхідно вказати, що в даному випадку маються на увазі фізичні пікселі, а не sp:

```
textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
```

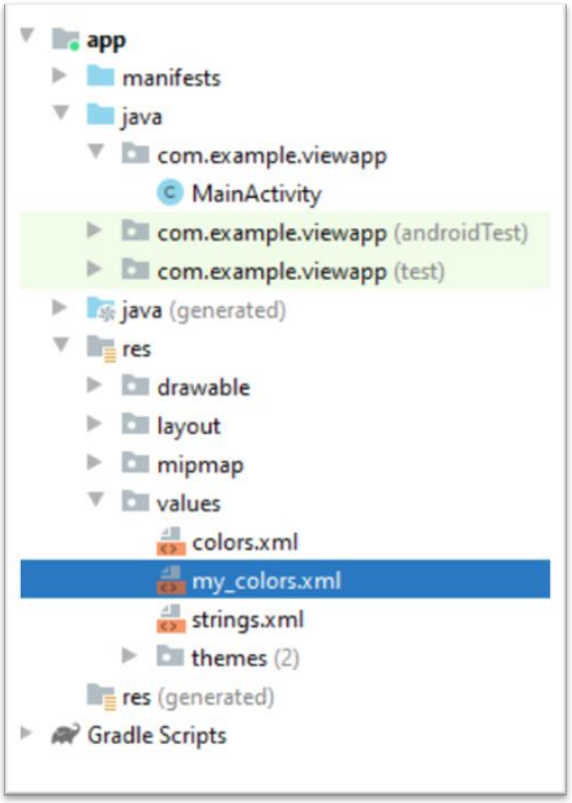
Або, як варіант, за допомогою класу TypedValue програмно перевести фізичні пікселі sp або іншу одиницю вимірювання.

4.9. **Ресурси Color та встановлення кольору**

У Android також можна визначати ресурси кольорів (Color). Вони повинні зберігатися у файлі шляхом res/values і так само, як і ресурси рядків, укладені в тег<resources>. Так, за умовчанням при створенні найпростішого проекту до папки res/values додається файл colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="purple_200">#FFBB86FC</color>
<color name="purple_500">#FF6200EE</color>
<color name="purple_700">#FF3700B3</color>
<color name="teal_200">#FF03DAC5</color>
<color name="teal_700">#FF018786</color>
<color name="black">#FF000000</color>
<color name="white">#FFFFFFFF</color>
</resources>
```

Колір визначається за допомогою елемента<color>. Його атрибутnameвстановлює назву кольору, яка буде використовуватись у додатку, а шістнадцяткове число – значення кольору.



Для завдання колірних ресурсів можна використовувати такі формати:

- #RGB (#F00 - 12-бітове значення)
- #ARGB (#8F00 - 12-бітове значення з додаванням альфа-каналу)
- #RRGGBB (#FF00FF - 24-бітове значення)
- #AARRGGBB (#80FF00FF - 24-бітове значення з додаванням альфа-каналу)

Щоб не чіпати і не псувати файл, визначимо свій новий файл ресурсів і для цього додамо в папку res/values новий файл ресурсів, який назвемо my_colors.xml.

Змінимо файл my_colors.xml, додавши до нього пару кольорів:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="textViewBackColor">#A0EAE1</color>
<color name="textViewFontColor">#00695C</color>
</resources>
```

Застосуємо кольори у файлі activity_main.xml:

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

<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android"

android:background="@color/textViewBackColor"
```

```
        android:textColor="@color/textViewFontColor"

        android:textSize="32sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

За допомогою атрибуту android:textColor встановлюється колір тексту в TextView, а атрибутandroid:backgroundвстановлює тло TextView. Як значення вони використовують колір, наприклад, у тому ж шістнадцятковому форматі. Для отримання кольору після "@color/" вказується ім'я ресурсу.

Також можна використовувати колірні ресурси в коді MainActivity:

```
        package com.example.viewapp;

        import androidx.appcompat.app.AppCompatActivity;
        import androidx.constraintlayout.widget.ConstraintLayout;

        import android.content.res.Resources;
        import android.os.Bundle;
        import android.widget.TextView;

        public class MainActivity extends AppCompatActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        // Отримуємо ресурси
        Resources resources = getResources();
        int textColor = resources.getColor(R.color.textViewFontColor, null);
        int backgroundColor = resources.getColor(R.color.textViewBackColor,
```

```
null);

      ConstraintLayout constraintLayout = New ConstraintLayout(this);

      ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
      (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
      layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
      layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT_ID;

      TextView textView = New TextView(this);
      textView.setText("Hello Android");
      textView.setTextSize(32);

      // використовуємо ресурси color
      textView.setTextColor(textColor);
      textView.setBackgroundColor(backgroundColor);

      textView.setLayoutParams(layoutParams);
      constraintLayout.addView(textView);

      setContentView(constraintLayout);
      }
      }
```

Для отримання кольору ресурсів застосовується метод resources.getColor(), який приймає два параметри. Перший параметр – ідентифікатор ресурсу, колір якого треба отримати. Другий параметр представляє тему. Але оскільки в даному випадку тема не є важливою, для цього параметра передаємо значеннянull

Слід враховувати, що метод resources.getColor() з двома параметрами, який використаний вище, доступний, якщо мінімальна версія Android не нижче Android 6.0 (або Android 23). Однак мінімальна версія Android нижче, то можна використовувати застарілу версію з одним параметром:

```
      int textColor = resources.getColor(R.color.textViewFontColor);
      // замість
      //int textColor = resources.getColor(R.color.textViewFontColor, null);
```

5. ACTIVITY

5.1. Activity та життєвий цикл програми

Ключовим компонентом для створення візуального інтерфейсу в Android є activity (активність). Нерідко activity асоціюється з окремим екраном або вікном програми, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Програма може мати одну або кілька дій. Наприклад, при створенні проекту з порожньою Activity до проекту за замовчуванням додається один клас Activity - MainActivity, з якого і починається робота програми:

```
public class MainActivity extends AppCompatActivity {
// Вміст класу
}
```

Всі об'єкти діяльності є об'єктами класуandroid.app.Activity, Що містить базову функціональність для всіх activity. У додатку з минулої теми ми безпосередньо з цим класом не працювали, а MainActivity успадковувалась від класуAppCompatActivity. Проте, сам клас AppCompatActivity, хоч і не безпосередньо, успадковується від базового класу Activity.

Життєвий цикл програми

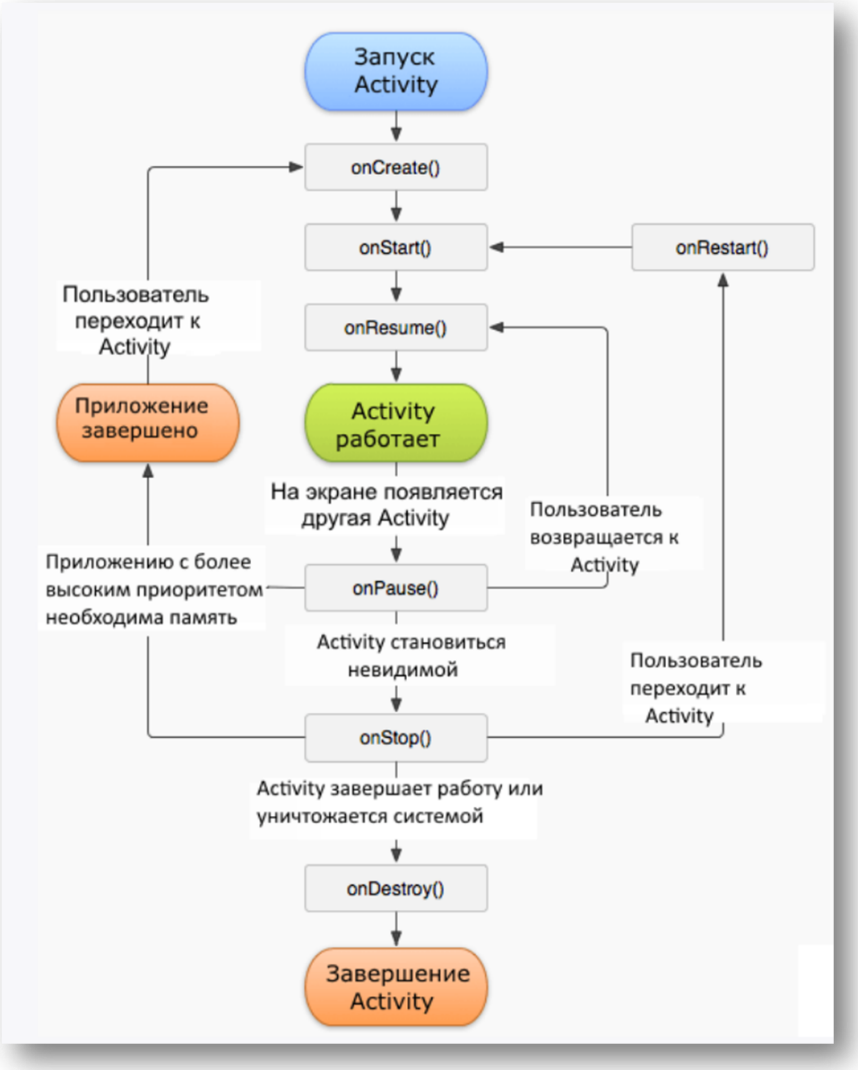
Всі програми Android мають строго визначений системою життєвий цикл. При запуску користувачем програми дає цій програмі високий пріоритет. Кожна програма запускається у вигляді окремого процесу, що дозволяє системі давати одним процесам більш високий пріоритет, на відміну від інших. Завдяки цьому, наприклад, при роботі з одними програмами Android дозволяє не блокувати вхідні дзвінки. Після припинення роботи з додатком система звільняє всі пов'язані ресурси і переводить додаток в розряд низькопріоритетного і закриває його.

Всі об'єкти activity, які є в додатку, керуються системою у вигляді стека activity, який називаєтьсяback stack. При запуску нової activity вона поміщається поверх стека і виводиться на екран пристрою, доки з'явиться нова activity. Коли поточна activity закінчує свою роботу (наприклад, користувач виходить з програми), то вона видаляється зі стека, і відновлює роботу та activity, яка раніше була другою у стеку.

Після запуску діяльності проходить через ряд подій, що обробляються системою та для обробки яких існує ряд зворотних викликів:

```
protected void onCreate(Bundle saveInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
```

Схематично взаємозв'язок між усіма цими зворотними викликами можна представити так



onCreate()

onCreate- перший метод, з якого починається виконання діяльності. У цьому методі діяльність переходить у стан Created. Цей метод обов'язково має бути визначений у класі діяльності. У ньому проводиться початкове настроювання діяльності. Зокрема, створюються об'єкти візуального інтерфейсу. Цей метод отримує об'єктBundle, який містить колишній стан діяльності, якщо він був збережений. Якщо активність заново створюється, цей об'єкт має значення null. Якщо ж activity вже раніше була створена, але перебувала у зупиненому стані, то bundle містить пов'язану з activity інформацію.

Після того, як метод onCreate() завершив виконання, діяльність переходить у станStarted, та і система викликає методonStart()

onStart

У методіonStart()здійснюється підготовка до виведення діяльності на екран пристрою. Як правило, цей метод не вимагає перевизначення, а

всю роботу здійснює вбудований код. Після завершення роботи методу activity відображається на екрані, викликається методonResume, А діяльність переходить у стан Resumed.

onResume

При виклику методуonResumeactivity переходить у станResumedі відображається на екрані пристрою, і користувач може взаємодіяти з нею. І власне activity залишається в цьому стані, поки вона не втратить фокус, наприклад, внаслідок перемикання на іншу activity або просто через вимкнення екрана пристрою.

onPause

Якщо користувач вирішить перейти до іншої діяльності, то система викликає методonPause, а діяльність переходить у станPaused. У цьому методі можна звільняти використовувані ресурси, призупиняти процеси, наприклад, відтворення аудіо, анімацій, зупиняти роботу камери (якщо вона використовується) і т.д., щоб вони менше впливали на продуктивність системи.

Але треба враховувати, що в цьому стані activity як і раніше залишається видимою на екрані, і на роботу даного методу відводиться дуже мало часу, тому не варто тут зберігати якісь дані, особливо якщо при цьому потрібне звернення до мережі, наприклад, надсилання даних інтернету, або звернення до бази даних – подібні дії краще виконувати у методіonStop().

Після виконання цього методу діяльність стає невидимою, не відображається на екрані, але вона все ще активна. І якщо користувач вирішить повернутися до цієї діяльності, то система викличе знову методonResume, і дія знову з'явиться на екрані.

Інший варіант роботи може виникнути, якщо раптом система бачить, що для роботи активних програм необхідно більше пам'яті. І система може сама повністю завершити роботу діяльності, яка невидима і знаходиться в фоні. Або користувач може натиснути кнопку Back (Назад). У цьому випадку у діяльності викликається методonStop.

onStop

У цьому методі діяльність переходить у стан Stopped. У цьому стані діяльність повністю невидима. У методі onStop слід особливо чекати використовувані ресурси, які не потрібні користувачеві, коли він не взаємодіє з діяльністю. Тут також можна зберігати дані, наприклад, базу даних.

При цьому під час стану Stopped activity залишається в пам'яті пристрою, зберігається всі елементи інтерфейсу. Наприклад, якщо в текстове поле EditText було введено якийсь текст, то після відновлення роботи activity

та переходу її в стан Resumed ми знову побачимо в текстовому полі раніше введений текст.

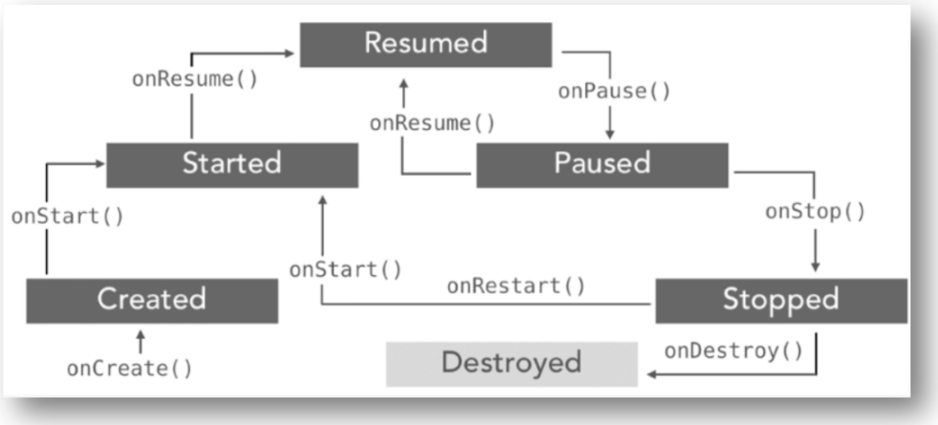
Якщо після виклику методуonStopкористувач вирішить повернутися до попередньої діяльності, тоді система викличе методonRestart. Якщо ж діяльність зовсім завершила свою роботу, наприклад, через закриття програми, то викликається методonDestroy().

onDestroy

Ну і завершується робота activity викликом методуonDestroy, який виникає або, якщо система вирішить убити діяльність через конфігураційні причини (наприклад, поворот екрана або при багатокінному режимі), або при виклику методуfinish().

Також слід зазначити, що при зміні орієнтації екрана система завершує діяльність і потім створює її заново, викликаючи методonCreate.

Загалом перехід між станами діяльності можна виразити наступною схемою:



Розглянемо кілька ситуацій. Якщо ми працюємо з Activity і потім перекључаємося на іншу програму, або натискаємо на кнопку Home, то у Activity викликається наступний ланцюжок методів:onPause -> onStop. Activity виявляється у стані Stopped. Якщо користувач вирішить повернутися до Activity, то викликається такий ланцюжок методів:**onRestart -> onStart -> onResume**.

Інша ситуація, якщо користувач натискає кнопку Back (Назад), то викликається наступний ланцюжок**onPause -> onStop -> onDestroy**. В результаті Activity знищується. Якщо ми раптом захочемо повернутися до Activity через диспетчер завдань або заново відкривши програму, то activity заново перетворюватиметься через методи**onCreate -> onStart -> onResume.**

5.2. Управління життєвим циклом

Ми можемо керувати цими подіями життєвого циклу, визначивши відповідні методи. Для цього візьмемо з попереднього розділу клас MainActivity і змінимо його таким чином:

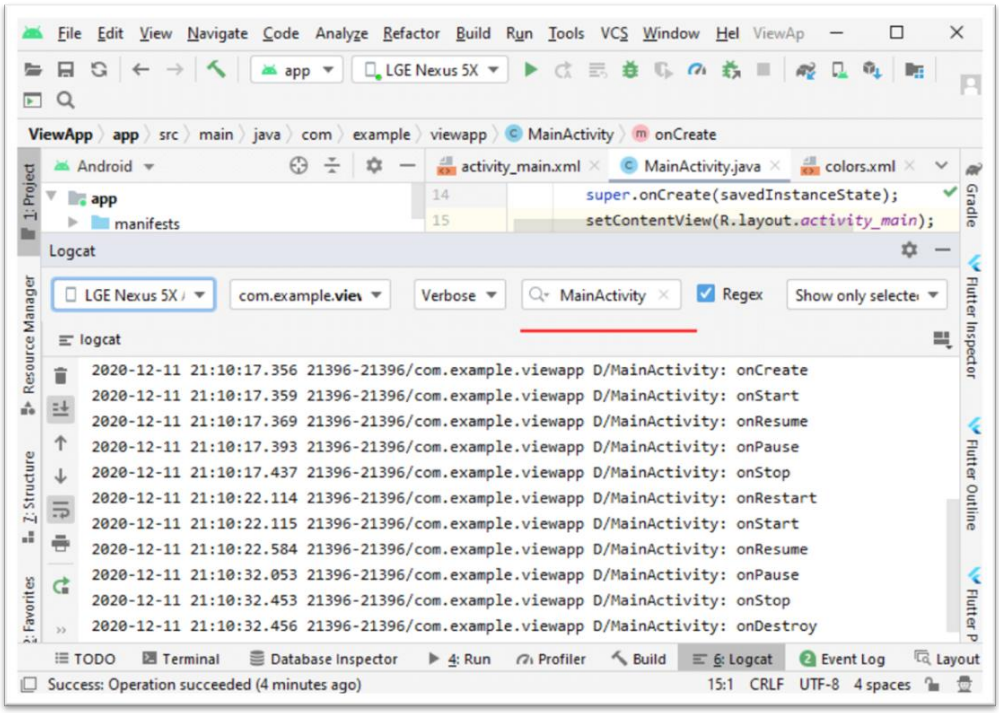
```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;


public class MainActivity extends AppCompatActivity {

private final static String TAG = "MainActivity";

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

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

Для логування подій тут використовується клас**android.util.Log**.

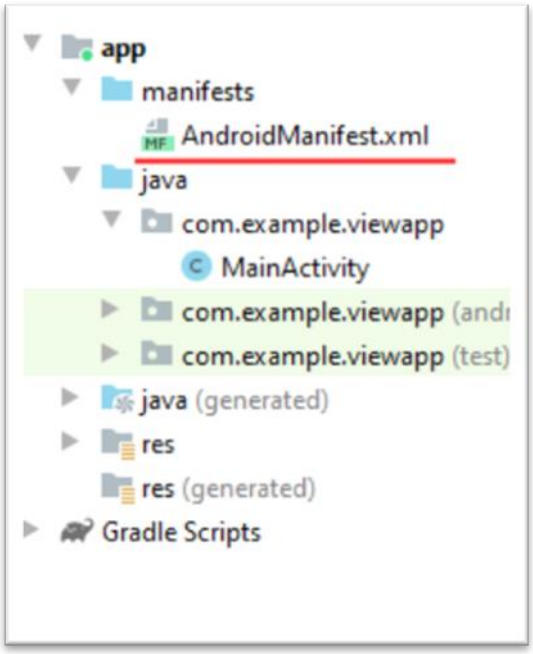
У разі обробляються все ключові методи життєвого циклу. Вся обробка зведена до виклику методуLog.d(), в який передається TAG - випадкове рядкове значення та рядок, що виводиться в консоліLogcatв нижній частині Android Studio, виконуючи роль налагоджувальної інформації. Якщо ця консоль за замовчуванням прихована, ми можемо перейти до неї через пункт меню**View -> Tool Windows -> Logcat.**

І при запуску програми ми зможемо побачити у вікніLogcatналагоджувальну інформацію, що визначається в методах життєвого циклу діяльності:



Файл маніфесту AndroidManifest.xml

Кожна програма містить файл маніфесту AndroidManifest.xml. Цей файл визначає важливу інформацію про програму - назву, версію, іконки, які дозволи програма використовує, реєструє всі використовувані класи activity, сервіси тощо. Цей файл можна знайти у проекті в папці manifests:



Файл маніфесту може виглядати так:

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

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.ViewApp">
<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

</manifest>
```

Елементом кореневого рівня є вузол manifest. В даному випадку тільки визначається пакет програми – package="com.example.viewapp". Власне, це визначення файлу маніфесту за умовчанням. У кожному конкретному випадку може відрізнятися пакет програми, решта вмісту при створенні проекту з порожньою діяльністю буде аналогічною.

Більшість установок рівня програми визначаються елементом application. Ряд установок задаються за допомогою атрибутів. За замовчуванням застосовуються такі атрибути:

- android:allowBackup вказує, чи буде для програми створюватись резервна копія. Значення android:allowBackup="true" дозволяє створити резервну копію.
- android:icon встановлює іконку програми. При значенні android:icon="@mipmap/ic_launcher" іконка програми береться з каталогу res/mipmap
- android:roundIcon встановлює круглу іконку програми. Також береться з каталогу res/mipmap
- android:label задає назву програму, яка відображатиметься на мобільному пристрої у списку програм та в заголовку. В даному випадку воно зберігається у рядкових ресурсах – android:label="@string/app_name".
- android:supportsRtl вказує, чи можуть використовуватись різні RTL API - спеціальні API для роботи з правосторонньою орієнтацією тексту (наприклад, для таких мов як арабська або фарсі).
- android:theme встановлює тему програми. Детально теми будуть розглянуті далі, а поки що достатньо знати, що тема визначає загальний стиль програми. Значення @style/Theme.ViewApp бере тему "Theme.ViewApp" з каталогу res/values/themes

Вкладені елементи activity визначають всі activity, що використовуються в програмі. В даному випадку видно, що в додатку є тільки одна діяльність - MainActivity.

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

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

Елемент intent-filter в MainActivity вказує, як ця дія буде використовуватися. Зокрема, за допомогою вузла action android:name="android.intent.action.MAIN", що дана activity буде вхідною точкою в програму і не повинна отримувати будь-які дані ззовні.

Елемент category android:name="android.intent.category.LAUNCHER" вказує, що MainActivity представлятиме стартовий екран, який відображається при запуску програми.

Файл маніфесту може містити багато елементів, які мають безліч атрибутів. І всі можливі елементи та їх атрибути можна знайти у документації. Тут же розглянемо приклади використання.

5.3. **Визначення версії**

За допомогою атрибутів елементаmanifestможна визначити версію програми та її коду:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.viewapp"
android:versionName="1.0"
android:versionCode="1">

<!-- решта вміст-->

</manifest>
```

Атрибут android:versionName вказує на номер версії, який відображатиметься користувачеві та на яку будуть орієнтуватися користувачі під час роботи з програмою.

Тоді як атрибут android:versionCode є номером версії для внутрішнього використання. Цей номер лише визначає, що одна версія програми нова, ніж якась інша з меншим номером номером версії. Цей номер не відображається користувачам.

При бажанні ми також можемо визначити версію ресурсів, а тут посилатися на ресурс.

Встановлення версії SDK

Для керування версією android sdk у файлі маніфесту визначається елемент <uses-sdk>. Він може використовувати такі атрибути:

minSdkVersion: мінімальна підтримувана версія SDK

targetSdkVersion: оптимальна версія

maxSdkVersion: максимальна версія

Версія визначається номером API, наприклад, Jelly Beans 4.1 має версію 16, а Android 11 має версію 30:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.viewapp"
android:versionName="1.0"
android:versionCode="1">
<uses-sdk android:minSdkVersion="22" android:targetSdkVersion="30" />
<!-- решта вміст-->

</manifest>
```

5.4. Встановлення дозволів

Іноді програма потребує дозволу на доступ до певних ресурсів, наприклад, до списку контактів, камери тощо. Щоб програма могла працювати з тим же списком контактів, у файлі маніфесті необхідно встановити відповідні дозволи. Для встановлення дозволів застосовується елемент <uses-permission>:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.viewapp">
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.CAMERA"
android:maxSdkVersion="30" />
<!-- решта вміст-->

</manifest>
```

Атрибут android:name встановлює назву роздільної здатності: в даному випадку читання списку контактів і використання камери. Опціонально можна встановити максимальну версію sdk за допомогою атрибуту android:maxSdkVersion, який приймає номер API.

Підтримка різних дозволів

Світ пристроїв Android дуже фрагментований, тут зустрічаються як гаджети з невеликим екраном, так і великі широкоекранні телевізори. І трапляються випадки, коли треба обмежити використання програми для певних дозволів екранів. Для цього у файлі маніфесту визначається елемент <supports-screens>:

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

<supports-screens
android:largeScreens="true"
android:normalScreens="true"
android:smallScreens="false"
android:xlargeScreens="true" />
<!-- решта вміст-->

</manifest>
```

Цей елемент приймає чотири атрибути:

- android:largeScreens - екрани з діагоналлю від 4.5 до 10"
- android:normalScreens - екрани з діагоналлю від 3 до 4.5"
- android:smallScreens - екрани з діагоналлю менше 3"
- android:xlargeScreens - екрани з діагоналлю більше 10"

Якщо атрибут має значення true, то програма підтримуватиметься відповідним розміром екрану

5.5. **Заборона на зміну орієнтації**

Програма в залежності від положення гаджета може перебувати в альбомній та портретній орієнтації. Не завжди це зручно. Ми можемо зробити, щоб програма незалежно від повороту гаджета використовувала лише одну орієнтацію. Для цього у файлі маніфесту у необхідної діяльності треба встановити атрибут android:screenOrientation. Наприклад, заборонимо альбомну орієнтацію:

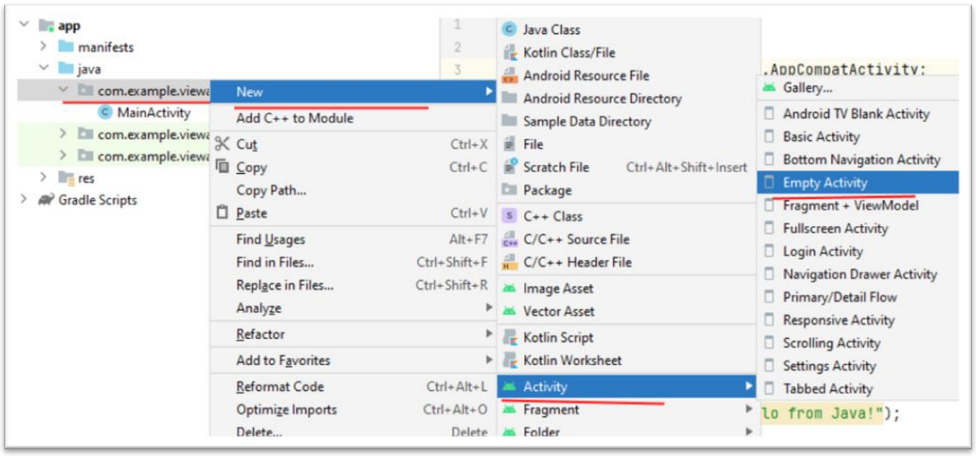
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.viewapp"
android:versionName="1.0"
android:versionCode="1" >

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.ViewApp">
<activity android:name=".MainActivity"

android:screenOrientation="portrait">

<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

Значення android:screenOrientation="portrait" вказує, що ця дія буде перебувати лише в портретній орієнтації. Якщо ж потрібно встановити лише альбомну орієнтацію, тоді треба використовувати значення android:screenOrientation="landscape"

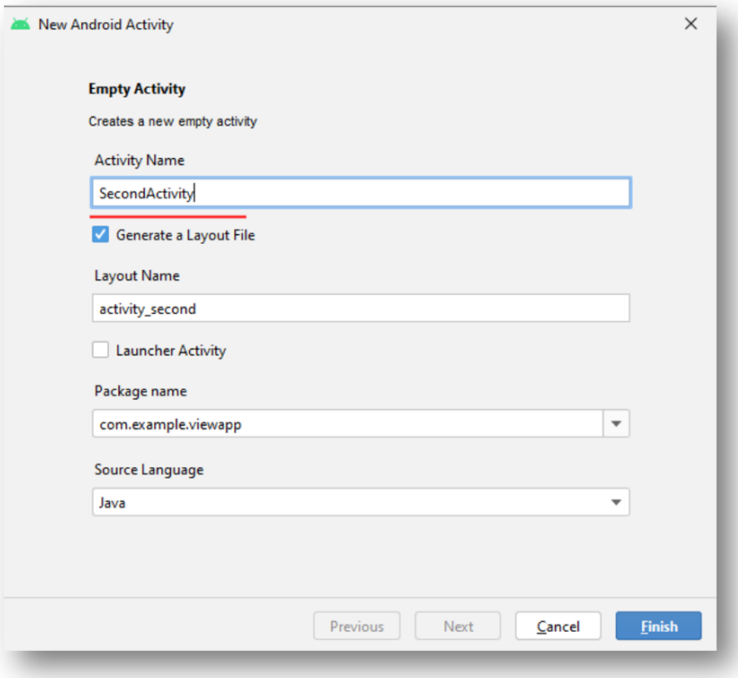
5.6. **Вступ до Intent. Запуск Activity**

Для взаємодії між різними об'єктами activity ключовим класом єandroid.content.Intent. Він є завданням, яке треба виконати додатку.

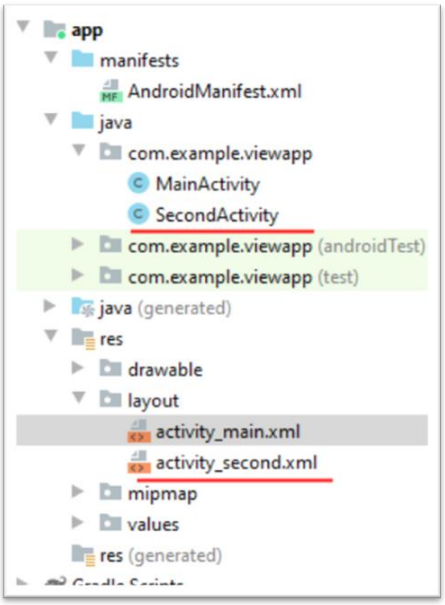
До роботи з Intent додамо новий клас Activity. Для цього натиснемо правою кнопкою миші в папку, в якій знаходиться клас MainActivity, а потім у контекстному меню виберемоNew->Activity->Empty Activity:



Новий клас Activity назвемоSecondActivity, а всі інші налаштування залишимо за замовчуванням:



І після цього до проекту буде додано нову Activity - SecondActivity:



Після цього у файлі маніфестуAndroidManifest.xmlми зможемо знайти наступні рядки:

```
<діяльність android:name=".SecondActivity"></activity>

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

Усі використовувані класи activity повинні бути описані у файлі AndroidManifest.xml за допомогою елемента<діяльність>. Кожен подібний елемент містить як мінімум один атрибутandroid:name, який встановлює ім'я класу діяльності.

Однак по суті activity – це стандартні класи java, які успадковуються від класуActivityабо його спадкоємців. Тому замість вбудованих шаблонів Android Studio можемо додавати звичайні класи, і потім їх успадковувати від класу Activity. Однак у цьому випадку потрібно буде вручну додавати у файл маніфесту дані про діяльність.

Причому для MainActivity в елементіintent-filterвизначається інтент-фільтр. У ньому елемент action значення "android.intent.action.MAIN" представляє головну точку входу до програми. Тобто MainActivity залишається основним і запускається програмою за замовчуванням.

Для SecondActivity просто вказано, що вона у проекті, і жодних intent-фільтрів для неї не задано.

Щоб з MainActivity запустити SecondActivity, треба викликати методstartActivity():

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

Як параметр методу startActivity передається об'єкт Intent. Для свого створення Intent в конструкторі приймає два параметри: контекст виконання (в даному випадку це поточний об'єкт MainActivity) і клас, який використовується об'єктом Intent і представляє дані (фактично клас activity, яку ми запускатимемо).

Тепер розглянемо реалізацію переходу від однієї Activity на іншу. Для цього у файліactivity_main.xml(тобто в інтерфейсі для MainActivity) визначимо кнопку:

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

<Button
android:id="@+id/navButton"
android:textSize="20sp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Перейти до SecondActivity"
android:onClick="onClick"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

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

І визначимо для кнопки у класіMainActivityобробник натискання, яким буде здійснюватися перехід до нової Activity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

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

```
    }
    public void onClick(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    startActivity(intent);
    }
    }
```

В обробнику натискання запускатиметься SecondActivity.

Далі змінимо кодSecondActivity:

Запустимо програму і перейдемо від однієї Activity до іншої:

5.7. Передача даних між Activity. Серіалізація

Для передачі даних між двома Activity використовується об'єкт Intent. Через його метод putExtra() можна додати ключ та пов'язане з ним значення.

Наприклад, передача з поточної діяльності в SecondActivity рядка "Hello World" з ключем "hello":


```
    // Створення об'єкта Intent для запуску SecondActivity
    Intent intent = new Intent(this, SecondActivity.class);
    // передача об'єкта з ключем "hello" та значенням "Hello World"
    intent.putExtra("hello", "Hello World");
    // запуск SecondActivity
    startActivity(intent);
```

Для передачі даних застосовується метод putExtra(), який як значення дозволяє передати дані найпростіших типів - String, int, float, double, long, short, byte, char, масиви цих типів або об'єкт інтерфейсу Serializable.

Щоб отримати відправлені дані при завантаженні SecondActivity, можна скористатися методом get(), який передається ключ об'єкта:

```
Bundle arguments = getIntent().getExtras();
String name = arguments.get("hello").toString(); // Hello World
```

Залежно від типу даних, що надсилаються при їх отриманні, ми можемо використовувати ряд методів об'єкта Bundle. Усі вони як параметр приймають ключ об'єкта. Основні з них:

- **get()**: Універсальний метод, який повертає значення типу Object. Відповідно поле одержання дане значення необхідно перетворити до потрібного типу
- **getString()**: повертає об'єкт типу String
- **getInt()**: повертає значення типу int
- **getByte()**: повертає значення типу byte
- **getChar()**: повертає значення типу char
- **getShort()**: повертає значення типу short
- **getLong()**: повертає значення типу long
- **getFloat()**: повертає значення типу float
- **getDouble()**: повертає значення типу double
- **getBoolean()**: повертає значення типу boolean
- **getCharArray()**: повертає масив об'єктів char
- **getIntArray()**: повертає масив об'єктів int
- **getFloatArray()**: повертає масив об'єктів float
- **getSerializable()**: повертає об'єкт інтерфейсу Serializable

Нехай у нас у проекті буде визначено дві activity: MainActivity та SecondActivity.

У коді SecondActivity визначимо отримання даних:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;
public class SecondActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

TextView TextView = New TextView(this);
textView.setTextSize(26);
textView.setPadding(16, 16, 16, 16);
Bundle arguments = getIntent().getExtras();

if(arguments!=null){
String name = arguments.get("name").toString();
```

```
String company = arguments.getString("company");
int age = arguments.getInt("age");
textView.setText("Name:" + name + "\nCompany: " + company +
"\nAge:" + age);
}
setContentView(textView);
}
}
```

В даному випадку в SecondActivity отримуємо всі дані з об'єкта Bundle та виводимо їх у текстове поле TextView. Передбачається, що даної діяльності будуть передаватися три елементи - два рядки з ключами name і company і число з ключем price.

Тепер визначимо передачу в SecondActivity даних. Наприклад, визначимо для MainActivity наступний інтерфейс у файлі activity_main.xml:

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

<TextView
android:id="@+id/nameLabel"
android:layout_width="0dp"
android:layout_height="20dp"
android:text="Name:"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
<EditText
android:id="@+id/name"
android:layout_width="0dp"
android:layout_height="40dp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/nameLabel"/>
<TextView
android:id="@+id/companyLabel"
android:layout_width="0dp"
android:layout_height="20dp"
android:text="Company:"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/name"/>
<EditText
android:id="@+id/company"
```

```
android:layout_width="0dp"
android:layout_height="40dp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/companyLabel" />
<TextView
android:id="@+id/ageLabel"
android:layout_width="0dp"
android:layout_height="20dp"
android:text="Age:"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/company"/>
<EditText
android:id="@+id/age"
android:layout_width="0dp"
android:layout_height="40dp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/ageLabel"/>
<Button
android:id="@+id/btn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onClick"
android:text="Save"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/age"/>

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

Тут визначено три текстові поля для введення даних та кнопка.

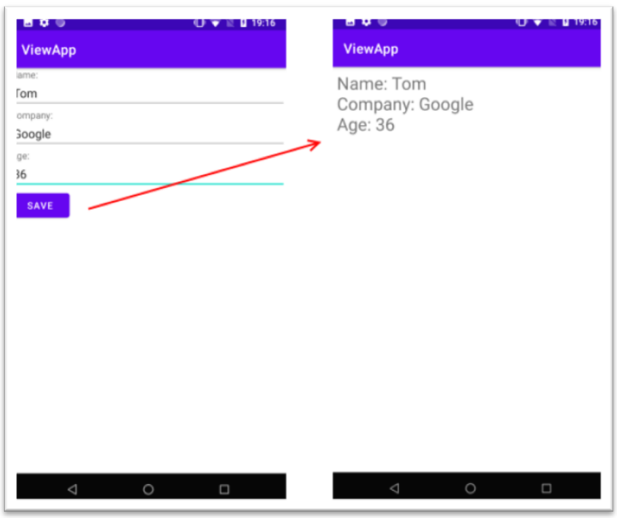
У класі MainActivity визначимо наступний вміст:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

```
        public void onClick(View v) {

        EditText nameText = findViewById(R.id.name);
        EditText companyText = findViewById(R.id.company);
        EditText ageText = findViewById(R.id.age);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int age = Integer.parseInt(ageText.getText().toString());

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra("name", name);
        intent.putExtra("company", company);
        intent.putExtra("age", age);
        startActivity(intent);
        }
        }
```

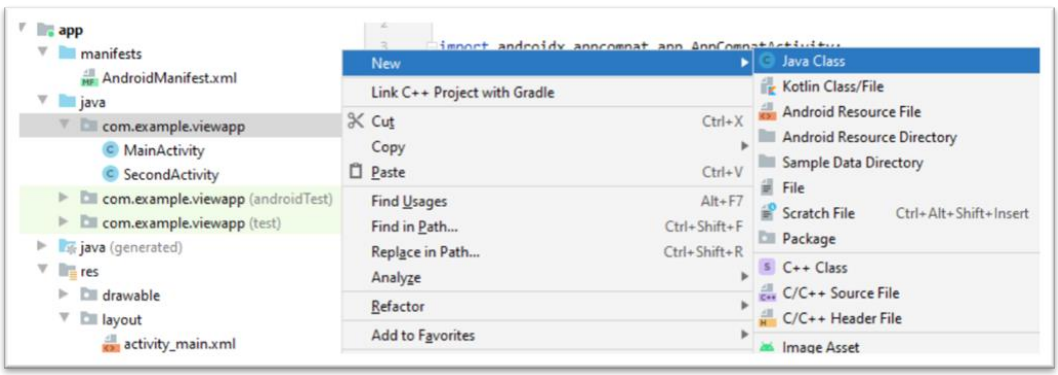
В обробнику натискання кнопки отримуємо введені в текстові поля EditText дані та передаємо їх у об'єкт Intent за допомогою методу putExtra(). Потім запускаємо SecondActivity.

У результаті, при натисканні на кнопку запуститься SecondActivity, яка отримає деякі введені в текстові поля дані.

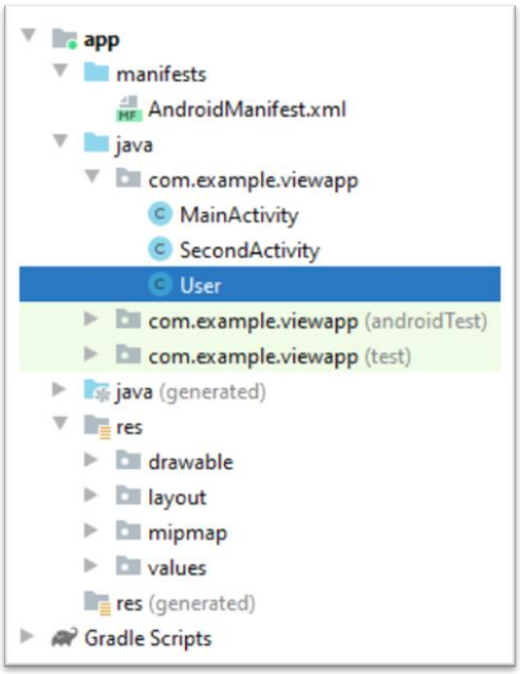


5.8. Передача складних об'єктів

У прикладі вище передавалися прості дані – числа, рядки. Але також ми можемо передавати складніші дані. І тут використовується механізм серіалізації. Для цього натиснемо на папку пакета, де знаходяться класи MainActivity та SecondActivity, правою кнопкою миші та в контекстному меню виберемо New -> Java Class:



Назвемо новий клас User - нехай він представлятиме користувача.



Нехай клас User має такий код:

```
package com.example.viewapp;

import java.io.Serializable;

public class User implements Serializable {

private String name;
private String company;
private int age;

Public User(String name, String company, int age){
this.name = name;
this.company = company;
this.age = age;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getCompany() {
return company;
}

public void setCompany(String company) {
```

```
this.company = company;
}

public int getAge() {
return age;
}

public void setAge(int age) {
this.age = age;
}
}
```

Цей клас реалізує інтерфейс Serializable. Тепер змінимо код MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onClick(View v) {
EditText nameText = findViewById(R.id.name);
EditText companyText = findViewById(R.id.company);
EditText ageText = findViewById(R.id.age);
String name = nameText.getText().toString();
String company = companyText.getText().toString();
int age = Integer.parseInt(ageText.getText().toString());
User user = new User(name, company, age);
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra(User.class.getSimpleName(), user);
startActivity(intent);
}
}
```

Тепер замість трьох розрізнених даних передається один об'єкт User. Як ключ використовується результат методу User.class.getSimpleName(), який по суті повертає назву класу.

І змінимо клас SecondActivity:

```
package com.example.viewapp;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//setContentView(R.layout.activity_second);
TextView TextView = New TextView(this);
textView.setTextSize(26);
textView.setPadding(16, 16, 16, 16);

Bundle arguments = getIntent().getExtras();

User user;
if(arguments!=null){
user = (User) arguments.getSerializable(User.class.getSimpleName());

textView.setText("Name: " + user.getName() + "\nCompany: " +
user.getCompany() +
\nAge: + String.valueOf(user.getAge()));
}
setContentView(textView);
}
}
```

Для отримання даних застосовується метод getSerializable(), оскільки клас User реалізує Serializable інтерфейс. Таким чином ми можемо передати один єдиний об'єкт замість набору розрізнених даних.

Parcelable

Можливість серіалізації об'єктів надається безпосередньо інфраструктурою мови Java. Однак Android також надає інтерфейс Parcelable, який, по суті, також дозволяє серіалізувати об'єкти, як і Serializable, але є більш оптимізованим для Android. І подібні об'єкти Parcelable також можна передавати між двома діями або використовувати якимось іншим чином.

Наприклад, у минулій темі дані передавалися між діями у вигляді об'єктів User, які використовували серіалізацію. Тепер нехай клас User застосовує інтерфейс Parcelable:

```
package com.example.viewapp;

import android.os.Parcel;
import android.os.Parcelable;
```

```
public class User implements Parcelable {

private String name;
private String company;
private int age;

public static final Creator<User> CREATOR = New Creator<User>() {
@Override
public User createFromParcel(Parcel source) {
String name = source.readString();
String company = source.readString();
int age = source.readInt();
return new User(name, company, age);
}

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

Public User(String name, String company, int age){
this.name = name;
this.company = company;
this.age = age;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getCompany() {
return company;
}
public void setCompany(String company) {
this.company = company;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}

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

```
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(name);
    dest.writeString(company);
    dest.writeInt(age);
    }
    }
```

Інтерфейс android.os.Parcelable передбачає реалізацію двох методів: describeContents() та writeToParcel(). Перший метод описує контент і повертає деяке числове значення. Другий метод пише об'єкт Parcel вміст об'єкта User.

Для запису даних об'єкта Parcel використовується ряд методів, кожен з яких призначений для певного типу даних. Основні методи:

- **writeString()**
- **writeInt()**
- **writeFloat()**
- **writeDouble()**
- **writeByte()**
- **writeLong()**
- **writeIntArray()**
- **writeValue()**(Записує об'єкт типу Object)
- **writeParcelable()**(Записує об'єкт типу Parcelable)

Крім того, об'єкт Parcelable повинен містити статичне поле CREATOR, яке представляє об'єкт Creator<User>. Причому цей об'єкт реалізує два методи. Вони потрібні для створення раніше серіалізованих даних вихідних об'єктів типу User.

Так, метод newArray() створює масив об'єкта User.

Метод створенняFromParcel створює з Parcel новий об'єкт типу User. Тобто цей метод протилежний дії методу writeToParcel. Для отримання даних із Parcel застосовуються методи типу readString(), readInt(), readParcelable() і так далі - для читання певних типів даних.

Причому важливо, що дані в createFromParcel зчитуються з об'єкта Parcel саме в тому порядку, в якому вони додаються в цей об'єкт у методі writeToParcel.

Допустимо в activity, яка називається SecondActivity, ми будемо отримувати об'єкт User:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

TextView TextView = New TextView(this);
textView.setTextSize(26);
textView.setPadding(16, 16, 16, 16);

Bundle arguments = getIntent().getExtras();

User user;
if(arguments!=null){
user = arguments.getParcelable(User.class.getSimpleName());

textView.setText("Name: " + user.getName() + "\nCompany: " +
user.getCompany() +
\nAge: + String.valueOf(user.getAge()));
}
setContentView(textView);
}
}
```

Для отримання об'єкта Parcelable, переданого в activity, застосовується метод getParcelable(). Причому жодного наведення типів не потрібне.

Для тестування передачі Parcelable визначимо у файлі activity_main.xml найпростіший інтерфейс MainActivity:

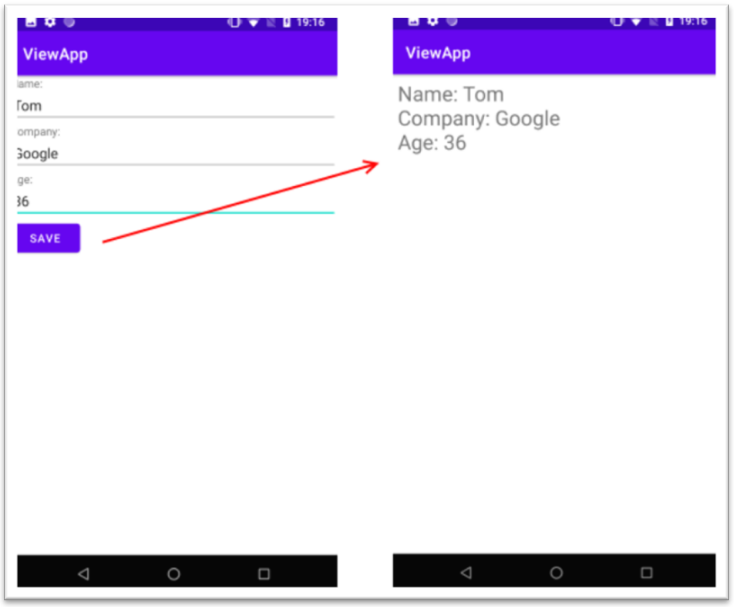
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent">

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

```
        android:layout_height="20dp"
        android:text="Name:"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
        <EditText
        android:id="@+id/name"
        android:layout_width="0dp"
        android:layout_height="40dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nameLabel"/>
        <TextView
        android:id="@+id/companyLabel"
        android:layout_width="0dp"
        android:layout_height="20dp"
        android:text="Company:"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/name"/>
        <EditText
        android:id="@+id/company"
        android:layout_width="0dp"
        android:layout_height="40dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/companyLabel" />
        <TextView
        android:id="@+id/ageLabel"
        android:layout_width="0dp"
        android:layout_height="20dp"
        android:text="Age:"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/company"/>
        <EditText
        android:id="@+id/age"
        android:layout_width="0dp"
        android:layout_height="40dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/ageLabel"/>
        <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Save"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
    app:layout_constraintTop_toBottomOf="@+id/age"/>

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

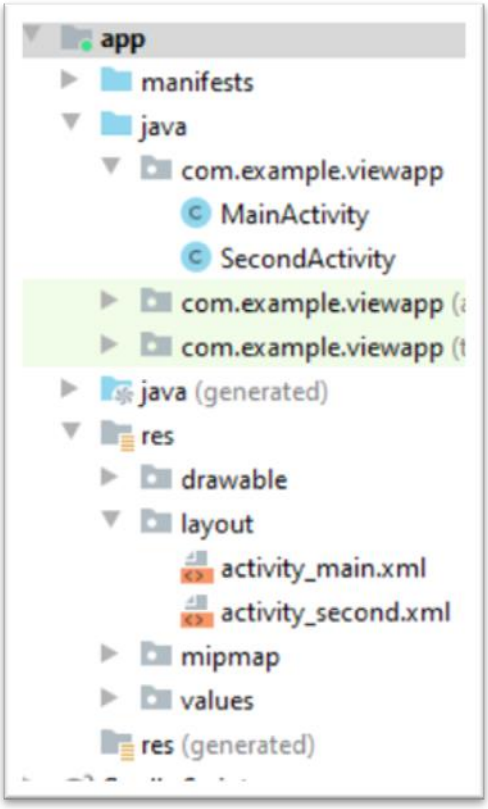
А в коді MainActivity визначимо передачу даних у SecondActivity:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.content.Intent;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.EditText;

    public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    }
    public void onClick(View v) {

    EditText nameText = findViewById(R.id.name);
    EditText companyText = findViewById(R.id.company);
    EditText ageText = findViewById(R.id.age);

    String name = nameText.getText().toString();
    String company = companyText.getText().toString();
    int age = Integer.parseInt(ageText.getText().toString());

    User user = new User(name, company, age);
    Intent intent = new Intent(this, SecondActivity.class);
    intent.putExtra(User.class.getSimpleName(), user);
    startActivity(intent);
    }
    }
```



5.9. Отримання результату з Activity

У минулій темі було розглянуто як викликати нову Activity та передавати їй деякі дані. Але ми можемо не тільки передавати дані активності, що запускається, але й очікувати від неї деякого результату роботи.



Наприклад, нехай у нас у проекті будуть дві activity: MainActivity та SecondActivity. А для кожної activity є свій файл інтерфейсу: activity_main.xml та activity_second.xml відповідно.

У минулій темі ми викликали нову діяльність за допомогою методу startActivity(). Для отримання результату роботи запускається activity необхідно використовувати Activity Result API.

Activity Result API надає компоненти для реєстрації, запуску та обробки результату іншої Activity. Однією з переваг застосування Activity Result API є те, що вона відв'язує результат Activity від самої Activity. Це дозволяє отримати та обробити результат, навіть якщо Activity, яка повертає результат, через обмеження пам'яті або через інші причини завершила свою роботу. Коротко розглянемо основні моменти застосування Activity Result API.

Реєстрація функції для отримання результату

Для реєстрації функції, яка оброблятиме результат, Activity Result API надає метод registerForActivityResult(). Цей метод як параметри приймає об'єкти ActivityResultContract і ActivityResultCallback і повертає об'єкт ActivityResultLauncher, який застосовується для запуску іншої activity.

```
ActivityResultLauncher<I> registerForActivityResult (
ActivityResultContract<I, O> contract,
ActivityResultCallback<O> callback)
```

ActivityResultContractвизначає контракт: дані якого типу подаватимуться на вхід і який тип представлятиме результат.

ActivityResultCallbackпредставляє інтерфейс з єдиним методом onActivityResult(), який визначає обробку отриманого результату. Коли друга діяльність закінчить роботу і поверне результат, то буде викликатися цей метод. Результат передається в метод як параметр. При цьому тип параметра повинен відповідати типу результату, визначеному ActivityResultContract. Наприклад:

```
ActivityResultLauncher<Intent> mStartForResult =
registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
new ActivityResultCallback<ActivityResult>() {
@Override
public void onActivityResult(ActivityResult result) {

// обробка result
}
});
```

Клас ActivityResultContracts надає низку вбудованих типів контрактів. Наприклад, у лістингу коду вище застосовується вбудований тип ActivityResultContracts.StartActivityForResult, який як вхідний об'єкт встановлює об'єкт Intent, а як тип результату - тип ActivityResult.

Запуск діяльності для отримання результату

Метод registerForActivityResult() реєструє функцію-колбек та повертає об'єкт ActivityResultLauncher. За допомогою цього ми можемо запустити діяльність. Для цього біля об'єктуActivityResultLauncherвикликається метод launch():

```
mStartForResult.launch(intent);
```

У метод lauch() передається об'єкт того типу, який визначений об'єктом ActivityResultContracts як вхідний.

5.10.Практичне застосування Activity Result API

Отже, визначимо у файлі activity_main.xml наступний інтерфейс:

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

<TextView
android:id="@+id/textView"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Вкажіть вік"
android:textSize="22sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
<EditText
android:id="@+id/age"
android:layout_width="0dp"
android:layout_height="wrap_content"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/textView"/>
<Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onClick"
android:text="Надіслати"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/age"/>

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

Для введення даних тут визначено елемент EditText, а відправки - кнопка.

Визначимо в класі MainActivity запуск другої діяльності:

```
package com.example.viewapp;
import androidx.activity.result.ActivityResult;
import androidx.activity.result.ActivityResultCallback;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

```
      import android.widget.TextView;

      public class MainActivity extends AppCompatActivity {

      static final String AGE_KEY = "AGE";
      static final String ACCESS_MESSAGE="ACCESS_MESSAGE";

      ActivityResultLauncher<Intent>            mStartForResult            =
registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
      new ActivityResultCallback<ActivityResult>() {
      @Override
      public void onActivityResult(ActivityResult result) {

      TextView textView = findViewById(R.id.textView);
      if(result.getResultCode() == Activity.RESULT_OK){
      Intent intent = result.getData();
      String accessMessage = intent.getStringExtra(ACCESS_MESSAGE);
      textView.setText(accessMessage);
      }
      else {
      textView.setText("Помилка доступу");
      }
      }
      });
      @Override
      protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      }
      public void onClick(View view) {
      // Отримуємо введений вік
      EditText ageBox = findViewById(R.id.age);
      String age = ageBox.getText().toString();
      Intent intent = new Intent(this, SecondActivity.class);
      intent.putExtra(AGE_KEY, age);

      mStartForResult.launch(intent);
      }
      }
```

Коротко розглянемо основні моменти цього коду. Насамперед, ми визначаємо об'єкт ActivityResultLauncher, за допомогою якого будемо запускати другу activity та передавати їй дані:

```
      ActivityResultLauncher<Intent>            mStartForResult            =
registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
      new ActivityResultCallback<ActivityResult>() {
      @Override
```

```
public void onActivityResult(ActivityResult result) {

TextView textView = findViewById(R.id.textView);
if(result.getResultCode() == Activity.RESULT_OK){
Intent intent = result.getData();
String accessMessage = intent.getStringExtra(ACCESS_MESSAGE);
textView.setText(accessMessage);
}
else {
textView.setText("Помилка доступу");
}
}
});
```

Об'єкт ActivityResultLauncher типизується типом Intent, тому що об'єкт цього типу передаватиметься в методlaunch()при запуску другої діяльності.

Тип контракту визначається типомActivityResultContracts.StartActivityForResultщо визначає тип Intent як вхідний тип і тип ActivityResult як тип результату.

Другий аргумент методуregisterForActivityResult()- об'єкт ActivityResultCallback типизується типом результату - типомActivityResulti визначає функцію-колбек onActivityResult(), яка отримує результат та обробляє його. В даному випадку обробка полягає в тому, що ми виводимо в текстове поле відповідь від другої діяльності.

При обробці ми перевіряємо отриманий код результату:

```
if(result.getResultCode() == Activity.RESULT_OK)
```

Як результат, як правило, застосовуються вбудовані константиActivity.RESULT_OKiActivity.RESULT_CANCELED. На рівні умовностейActivity.RESULT_OKозначає, що діяльність успішно обробила запит, aActivity.RESULT_CANCELED- що діяльність відхилила обробку запиту.

За допомогою методуgetData()результату отримуємо передані з другої activity дані у вигляді об'єкта Intent:

```
Intent intent = result.getData();
```

Далі витягаємо з Intent рядок, що мають ключ ACCESS_MESSAGE, та виводимо його у текстове поле.

Таким чином, ми визначили об'єкт ActivityResultLauncher. Далі в обробнику натискання onClick за допомогою цього об'єкта запускаємо другу activity - SecondActivity:

```
public void onClick(View view) {
// Отримуємо введений вік
EditText ageBox = findViewById(R.id.age);
```

```
String age = ageBox.getText().toString();

Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra(AGE_KEY, age);

mStartForResult.launch(intent);
}
```

В обробнику натискання кнопкиonClick()отримуємо введений у текстове поле вік, додаємо його в об'єкт Intent з ключем AGE_KEY та запускаємо SecondActivity за допомогою методуlaunch()

Тепер перейдемо до SecondActivity та визначимо у файлі activity_second.xml набір кнопок:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView
android:id="@+id/ageView"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:textSize="26sp"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
<Button
android:id="@+id/button1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Відкрити доступ"
android:onClick="onButton1Click"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/ageView"/>
<Button
android:id="@+id/button2"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Відхилити доступ"
android:onClick="onButton2Click"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button1"/>
<Button
android:id="@+id/button3"
android:layout_width="0dp"
```

```
android:layout_height="wrap_content"
android:text="Вік недійсний"
android:onClick="onButton3Click"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button2" />
<Button
android:id="@+id/cancel"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Скасувати"
android:onClick="onCancelClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button3" />

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

А в класі SecondActivity визначимо обробники для цих кнопок:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_second);
Bundle extras = getIntent().getExtras();
if (extras! = null) {
TextView ageView = findViewById(R.id.ageView);
String age = extras.getString(MainActivity.AGE_KEY);
ageView.setText("Вік:" + age);
}
}
public void onCancelClick(View v) {
setResult(RESULT_CANCELED);
finish();
}
public void onButton1Click(View v) {
sendMessage("Доступ дозволено");
}
public void onButton2Click(View v) {
```

```
        sendMessage("Доступ заборонено");
        }
        public void onButton3Click(View v) {
        sendMessage("Неприпустимий вік");
        }
        private void sendMessage(String message){

        Intent data = New Intent();
        data.putExtra(MainActivity.ACCESS_MESSAGE, message);
        setResult(RESULT_OK, data);
        finish();
        }
        }
```

Три кнопки викликають методsendMessage(), в який передають відповідь, що відправляється. Це і буде те повідомлення, яке отримати MainActivity в методі onActivityResult.

Для повернення результату необхідно викликати метод setResult(), який передається два параметри:

- числовий код результату
- дані, що надсилаються

Після виклику методуsetResult()Необхідно викликати метод finish, який знищить поточну діяльність.

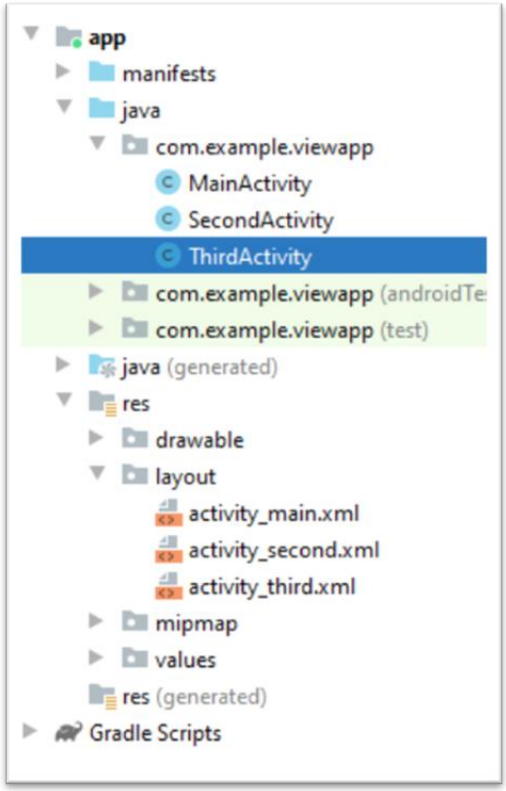
Одна кнопка викликає оброблювачonCancelClick(), В якому передається в setResult тільки код результату - RESULT_CANCELED.

Тобто, умовно кажучи, ми отримуємо в SecondActivity введений у MainActivity вік і за допомогою натискання певної кнопки повертаємо деякий результат у вигляді повідомлення.

Залежно від натиснутої кнопки на SecondActivity ми отримуватимемо різні результати в MainActivity:

5.11.Взаємодія між Activity

У минулих темах ми розглянули життєвий цикл activity та запуск нових activity за допомогою об'єкта Intent. Тепер розглянемо деякі особливості взаємодії між діяльністю в одному додатку. Допустимо, у нас є три activity: MainActivity, SecondActivity та ThirdActivity.



За допомогою Intent, наприклад, після натискання кнопки MainActivity запускає SecondActivity:

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

На SecondActivity також є кнопка, яка запускає ThirdActivity:

```
Intent intent = new Intent(this, ThirdActivity.class);
startActivity(intent);
```

На ThirdActivity також є кнопка, яка повертається до першої діяльності - MainActivity:

```
Intent intent = new Intent(this, MainActivity.class);
startActivity(intent);
```

Якщо ми послідовно запустимо всі activity: з головної MainActivity запустимо SecondActivity, з SecondActivity - ThirdActivity, то в результаті у нас складеться наступний стек activity:

ThirdActivity
SecondActivity
MainActivity

Якщо після цього з ThirdActivity ми захочемо звернутися до MainActivity, то метод startActivity() запустить новий об'єкт MainActivity (а не повернеться до вже існуючого), і стек буде виглядати так:

1. MainActivity
2. ThirdActivity
3. SecondActivity
4. MainActivity

Тобто ми матимемо дві незалежні копії MainActivity. Такий стан небажаний, якщо ми просто хочемо перейти до існуючого. І на цей момент треба враховувати.

Якщо ми натиснемо кнопку Back (Назад), то поточна activity, яка знаходиться на вершині стека, видаляється зі стека, і попередня activity виявляється на вершині стека та відновлює свою роботу. І таким чином, за допомогою кнопки Back (Назад) ми зможемо перейти до попередньої activity в стеку. Наприклад, якщо ми натиснемо кнопку Назад, то MainActivity на вершині стека завершує свою роботу, і на екрані починає відображатися ThirdActivity

1. ThirdActivity
2. SecondActivity
3. MainActivity

Тим не менш, іноді виникає необхідність впалювати переходом між діяльністю. Наприклад, в даному випадку нам небажано при натисканні на кнопку ThirdActivity запускати нову копію MainActivity замість того, щоб просто перейти до MainActivity, яка була запущена першою і знаходиться в самому низу стека. Розглянемо які можливості надає нам Android.

Управління стеком activity

Для управління стеком з діяльності Android пропонує нам використовувати прапори - константи, визначені в класі Intent. Застосування певного прапора дозволить нам певним чином змінити положення у стеку для певної діяльності.

Наприклад, візьмемо попереднє завдання, коли після натискання на кнопку ThirdActivity запускається новий екземпляр MainActivity. Але ми хочемо не запускати нову, а перейти до існуючої.

1. MainActivity
2. ThirdActivity
3. SecondActivity
4. MainActivity

Щоб вийти з цієї ситуації, ми можемо використовувати прапор Intent.FLAG_ACTIVITY_REORDER_TO_FRONT:

```
Intent intent = new Intent(this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```

прапор Intent.FLAG_ACTIVITY_REORDER_TO_FRONT переміщує activity, до якої здійснюється перехід на вершину стека, якщо вона вже є в стеку. І в цьому випадку після переходу з ThirdActivity до MainActivity стек буде виглядати так:

1. MainActivity
2. ThirdActivity
3. SecondActivity

Якщо ж нам просто треба перейти з ThirdActivity до MainActivity, якби ми перейшли назад за допомогою кнопки Back, то ми можемо використовувати прапори Intent.FLAG_ACTIVITY_CLEAR_TOP та Intent.FLAG_ACTIVITY_SINGLE_TOP:

```
Intent intent = new Intent(this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
startActivity(intent);
```

Прапор Intent.FLAG_ACTIVITY_CLEAR_TOP очищає всі activity крім тієї, яка запускається (якщо вона вже є у стеку). А прапор Intent.FLAG_ACTIVITY_SINGLE_TOP вказує, що якщо у вершині стеку вже є activity, яку треба запустити, то вона НЕ запускається (то вона може існувати у стеку лише в одиничному вигляді).

У цьому випадку після переходу з ThirdActivity до MainActivity стек буде повністю очищений, і там залишиться одна MainActivity.

Ще один прапор - Intent.FLAG_ACTIVITY_NO_HISTORY дозволить не зберігати в стеку activity, що запускається. Наприклад, при запуску SecondActivity ми не хочемо її зберігати у стеку:

```
Intent intent = new Intent(this, SecondActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
startActivity(intent);
```

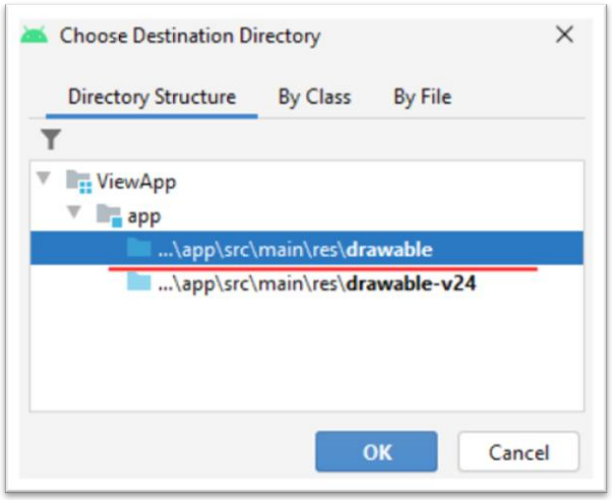
У цьому випадку при переході по ланцюжку MainActivity -> SecondActivity -> ThirdActivity стек буде виглядати так:

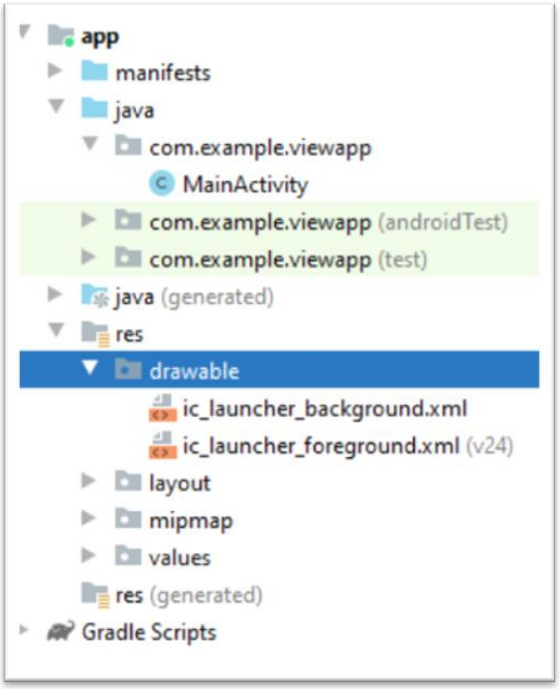
1. MainActivity
2. ThirdActivity

6. РОБОТА ІЗ ЗОБРАЖЕННЯМИ

6.1. Ресурси зображень

Одним із найпоширеніших джерел ресурсів є файли зображень. Android підтримує такі формати файлів: .png (переважний), .jpg (прийнятний), .gif (небажаний). Для графічних файлів у проекті вже за замовчуванням створено папку res/drawable. За замовчуванням вона вже містить ряд файлів – пару файлів іконок:





При додаванні графічних файлів до цієї папки для кожного з них Android створює ресурс Drawable. Після цього ми можемо звернутися до ресурсу в коді Java:

```
R.drawable.ім'я_файлу
```

Або в коді xml:

```
@[ім'я_пакета:]drawable/ім'я_файлу
```

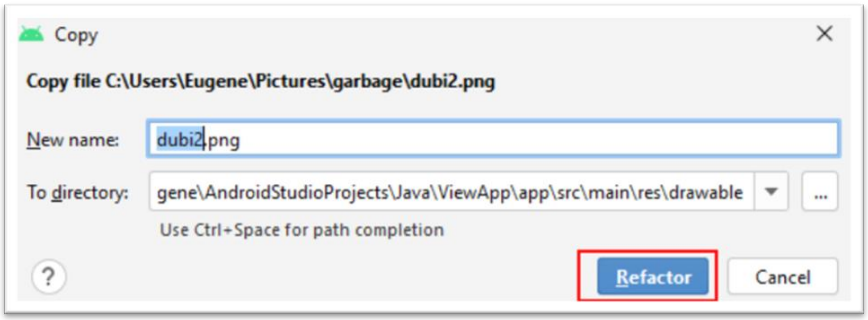
Наприклад, додамо в проект папки res/drawable який-небудь файл зображення. Для цього скопіюємо на жорсткому диску якийсь файл із розширенням png або jpg і вставимо його в папку res/drawable (для копіювання в проект використовується простий Copy-Paste)

Далі нам буде запропоновано вибрати папку – drawable або drawable-24. Для додавання звичайних файлів зображень виберемо drawable:

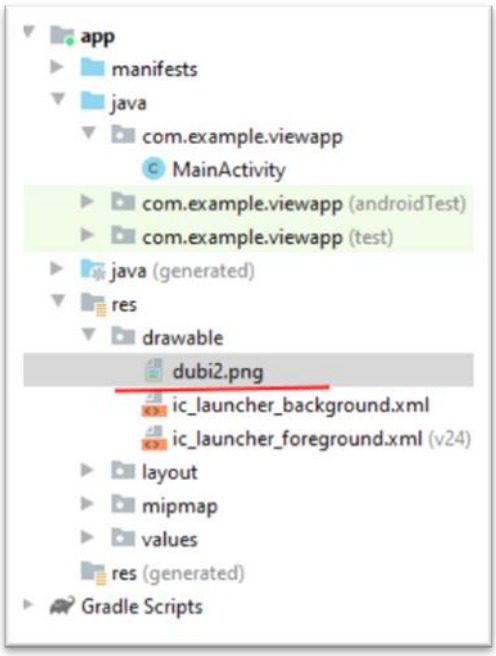
Тут відразу варто врахувати, що файл зображення додаватиметься в додаток, тим самим збільшуючи його розмір. Крім того, великі зображення негативно впливають на продуктивність. Тому краще використовувати невеликі та оптимізовані (стислі) графічні файли. Хоча також варто відзначити, що всі файли зображень, які додаються до цієї папки, можуть

автоматично оптимізуватися за допомогою утиліти aapt під час побудови проекту. Це дозволяє зменшити розмір файлу без втрати якості.

При копіюванні файлу нам буде запропоновано встановити нове ім'я.



Можна змінити назву файлу, а можна залишити як є. У моєму випадку файл називається dubi2.png. І потім натисніть кнопку Refactor. І після цього до папки drawable буде додано обраний нами файл зображення.



Для роботи з зображеннями в Android можна використовувати різні елементи, але для виведення зображень призначений ImageView. Тому змінимо файл activity_main.xml таким чином:<?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"
android:layout_width="match_parent"
android:layout_height="match_parent">

<ImageView
android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:src="@drawable/dubi2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

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

В даному випадку для відображення файлу в ImageView елемент встановлює атрибут android:src. У його значенні вказується ім'я графічного ресурсу, яке збігається з ім'ям без розширення. І після цього вже в Preview або в режимі дизайнера Android Studio можна буде побачити застосування зображення, або при запуску програми:

Якби ми створювали ImageView в коді java і з коду застосовували б ресурс, то діяльність могла б виглядати так:

```
        package com.example.viewapp;

        import androidx.appcompat.app.AppCompatActivity;
        import androidx.constraintlayout.widget.ConstraintLayout;
        import android.os.Bundle;
        import android.widget.ImageView;

        public class MainActivity extends AppCompatActivity {

        @Override
```

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

ConstraintLayout constraintLayout = New ConstraintLayout(this);
ImageView imageView = новий ImageView(this);
// застосовуємо ресурс
imageView.setImageResource(R.drawable.dubi2);

ConstraintLayout.LayoutParams layoutParams = New
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT_ID;
imageView.setLayoutParams(layoutParams);
constraintLayout.addView(imageView);

setContentView(constraintLayout);
}
}
```

В даному випадку ресурс drawable безпосередньо передається в метод imageView.setImageResource(), і таким чином встановлюється зображення. В результаті ми отримаємо той самий результат.

```
imageView.setImageResource(R.drawable.dubi2);
```

Однак, може виникнути необхідність якось обробити ресурс перед використанням або використовувати його в інших сценаріях. У цьому випадку ми можемо спочатку отримати його як об'єкт Drawable, а потім використовувати для наших завдань:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.content.res.ResourcesCompat;

import android.content.res.Resources;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {
```

```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        ConstraintLayout constraintLayout = New ConstraintLayout(this);
        ImageView imageView = новий ImageView(this);
        Resources res = getResources();
        Drawable drawable = ResourcesCompat.getDrawable(res,
R.drawable.dubi2, null);
        // застосовуємо ресурс
        imageView.setImageDrawable(drawable);

        ConstraintLayout.LayoutParams layoutParams = New
ConstraintLayout.LayoutParams
        (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT_ID;
        imageView.setLayoutParams(layoutParams);
        constraintLayout.addView(imageView);

        setContentView(constraintLayout);
        }
        }
```

Для отримання ресурсу застосовується метод ResourcesCompat.getDrawable(), в який передається об'єкт Resources, ідентифікатор ресурсу та тема. У цьому випадку тема нам не важлива, тому для неї передаємо значення null. Повертається ресурс у вигляді об'єкта Drawable:

```
        Drawable drawable = ResourcesCompat.getDrawable(res,
R.drawable.dubi2, null);
```

Потім, наприклад, можна також передати ресурс об'єкту ImageView через його метод**setImageDrawable()**

```
        imageView.setImageDrawable(drawable);
```

-

6.2. ImageView

Минулої теми було розглянуто, як виводити зображення за допомогою елемента ImageView. Тепер розглянемо деякі додаткові моменти роботи з цим елементом.

Деякі основні атрибути елемента ImageView:

- **android:cropToPadding**: при значенніtrueзображення обрізається відповідно до встановлених відступів

- **android:scaleType**: встановлює, як зображення масштабуватиметься щодо меж елемента ImageView

Щоб задати параметри масштабування, використовується одне із значень переліку <="" span="">:

  - CENTER: зображення центрується по центру без масштабування
  - CENTER_CROP: зображення центрується по центру та масштабується із збереженням аспектного відношення між шириною та висотою. Якщо якась частина не поміщається у межі екрана, вона обрізається
  - CENTER_INSIDE: зображення центрується по центру і масштабується із збереженням аспектного відношення між шириною та висотою, але ширина та висота не можуть бути більшими за ширину та висоту ImageView
  - FIT_CENTER: зображення масштабується та центрується
  - FIT_START: зображення масштабується та встановлюється на початок елемента (вгору при портретній орієнтації та вліво – при альбомній)
  - FIT_END: зображення масштабується та встановлюється в кінець елемента (вниз при портретній орієнтації та вправо – при альбомній)
  - FIT_XY: зображення масштабується без збереження аспектного відношення між шириною і висотою, заповнюючи весь простір ImageView
  - MATRIX: зображення масштабується із застосуванням матриці зображення

- **android:src**: ресурс зображення
- **android:alpha**: встановлює прозорість (значення від 0.0 – повністю прозоре до 1.0 – повністю мабуть)
- **android:tint**: колір, який використовується для накладання зображення
- **android:tintMode**: режим, який застосовується для накладання кольору на зображення

Деякі основні методи класу ImageView:

- **Drawable getDrawable()**: повертає ресурсDrawable, який пов'язаний з даними ImageView (абоnull, якщо ресурс для ImageView не встановлено)
- **ImageView.ScaleType getScaleType()**: повертає значення перерахування ImageView.ScaleType, яке вказує, як масштабується зображення щодо меж елемента ImageView
- **void setImageDrawable(Drawable drawable)**: встановлює ресурс зображення за допомогою об'єкта Drawable
- **void setImageResource(int resId)**: встановлює ресурс зображення за допомогою ідентифікатора ресурсу Drawable
- **void setImageURI(Uri uri)**: встановлює ресурс зображення за допомогою адреси Uri цього ресурсу

- **void setScaleType(ImageView.ScaleType scaleType)**: задає масштабування зображення
- **void setImageAlpha(int alpha)**: визначає прозорість зображення - значення від 0.0 до 1.0

Наприклад, встановлення значенняFIT_XYдля атрибуту android:scaleType у файлі activity_main.xml:

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

<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/dubi2"
android:scaleType="fitXY"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

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

У результаті зображення розтягнеться по вертикалі.
та горизонталі:

Для порівняння, аналогічний приклад
зandroid:scaleType="center":

Аналогічний приклад у коді java:

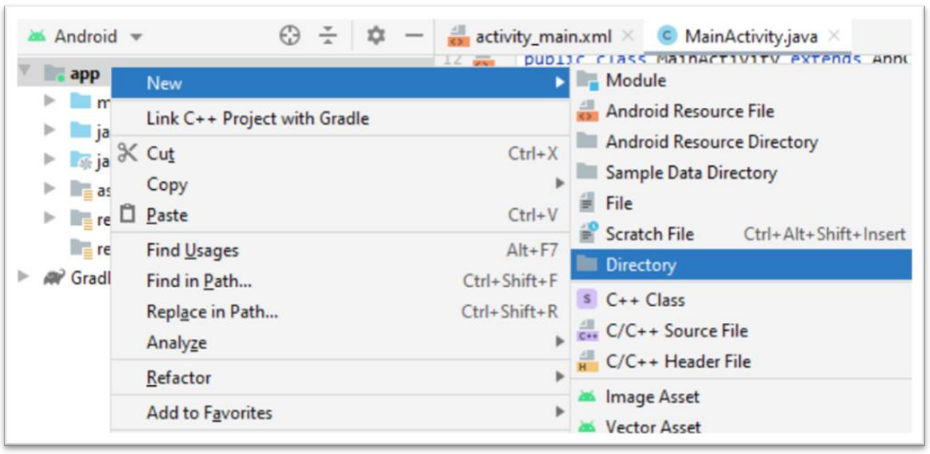
```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

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

ConstraintLayout constraintLayout = New ConstraintLayout(this);
ImageView imageView = новий ImageView(this);
imageView.setImageResource(R.drawable.dubi2);
// задаємо масштабування
imageView.setScaleType(ImageView.ScaleType.FIT_XY);

ConstraintLayout.LayoutParams layoutParams = New
ConstraintLayout.LayoutParams
(ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
layoutParams.topToTop=ConstraintLayout.LayoutParams.PARENT_ID;
imageView.setLayoutParams(layoutParams);
constraintLayout.addView(imageView);

setContentView(constraintLayout);
}
}
```

6.3. Зображення з папки assets

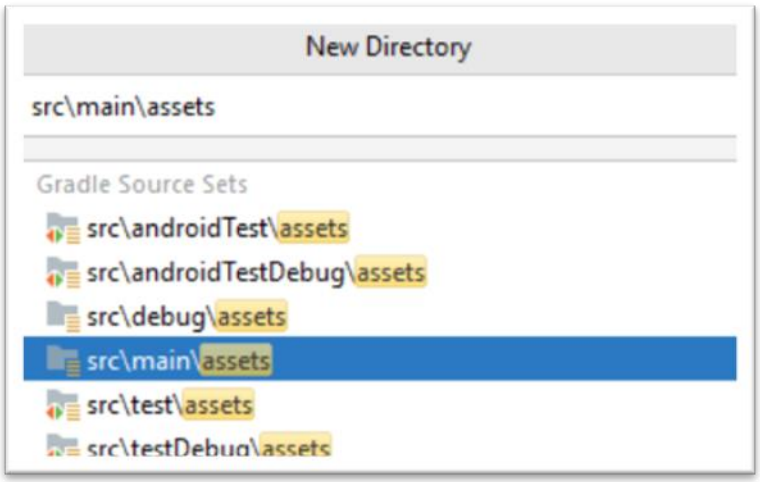
У попередніх темах зображення в проекті поміщалися в папку res/drawables як ресурси і виводилися в елемент ImageView. Однак зображення необов'язково в принципі поміщати саме цю папку. Файли також

можуть бути розміщені в папці assets. Розглянемо, як працювати з файлами зображень.

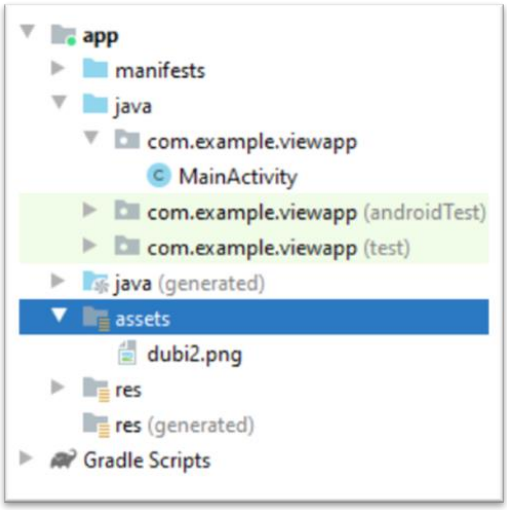
Спочатку додамо до проекту папку assets. Для цього в Android Studio натиснемо на каталог app і в контекстному меню, що з'явилося, виберемо New -> Directory:



Потім у вікні, що з'явилося, виберемо пункт src\main\assets і натиснемо на Enter для її додавання в проект:



Додамо в цю папку якесь зображення:



Нехай у файлі activity_main.xml буде визначено елемент ImageView:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
    android:id="@+id/image"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_margin="16dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Завантажимо зображення з папки assets в елемент ImageView в MainActivity:

```
    package com.example.viewapp;

    import androidx.appcompat.app.AppCompatActivity;

    import android.graphics.drawable.Drawable;
    import android.os.Bundle;
    import android.widget.ImageView;

    import java.io.IOException;
    import java.io.InputStream;

    public class MainActivity extends AppCompatActivity {

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

    ImageView imageView = findViewById(R.id.image);
    String filename = "dubi2.png";
    try(InputStream inputStream =
getApplicationContext().getAssets().open(filename)){
    Drawable drawable = Drawable.createFromStream(inputStream, null);
    imageView.setImageDrawable(drawable);
    imageView.setScaleType(ImageView.ScaleType.FIT_XY);
    }
```

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

Для завантаження файлу потрібно отримати потік InputStream за допомогою виразу getApplicationContext().getAssets().open(filename).

Виклик Drawable.createFromStream(inputStream, null) формує об'єкт Drawable із вхідного потоку.

Метод imageView.setImageDrawable(d) завантажує Drawable у ImageView.

- 

7. БАГАТОПОТОЧНІСТЬ

7.1. Створення потоків та візуальний інтерфейс

Коли ми запускаємо програму на Android, система створює потік, який називається основним потоком програми або UI-потік. Цей потік обробляє всі зміни та події інтерфейсу користувача. Однак для допоміжних операцій, таких як надсилання або завантаження файлу, тривалі обчислення тощо, ми можемо створювати додаткові потоки.

Для створення нових потоків нам доступний стандартний функціонал класу Thread з базової бібліотеки Java з пакетаjava.util.concurrent, які особливої складності не представляють. Тим не менш, труднощі можуть виникнути при оновленні візуального інтерфейсу з потоку.

Наприклад, створимо найпростіший додаток з використанням потоків. Визначимо наступну розмітку інтерфейсу у файлі activity_main.xml:

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

<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
android:textSize="22sp"
app:layout_constraintBottom_toTopOf="@id/button"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent" />
<Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Запустити потік"
app:layout_constraintTop_toBottomOf="@id/textView"
app:layout_constraintLeft_toLeftOf="parent" />

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

Тут визначено кнопку для запуску фонового потоку, а також текстове поле для відображення деяких даних, які будуть генеруватися в запущеному потоці.

Далі визначимо у класі MainActivity наступний код:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

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

TextView textView = findViewById(R.id.textView);
button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// Визначаємо об'єкт Runnable
Runnable runnable = New Runnable() {
@Override
public void run() {
// отримуємо поточний час
Calendar c = Calendar.getInstance();
int hours = c.get(Calendar.HOUR_OF_DAY);
int minutes = c.get(Calendar.MINUTE);
int seconds = c.get(Calendar.SECOND);
String time = hours + ":" + minutes + ":" + seconds;
// відображаємо у текстовому полі
textView.setText(time);
}};
// Визначаємо об'єкт Thread – новий потік
Thread thread = New Thread (runnable);
// Запускаємо потік
thread.start();
}});
}
}
```

Отже, тут до кнопки прикріплено обробник натискання, який запускає новий потік. Створювати та запускати потік у Java можна різними способами. В даному випадку самі дії, які виконуються в потоці, визначаються методом run() об'єкта Runnable:

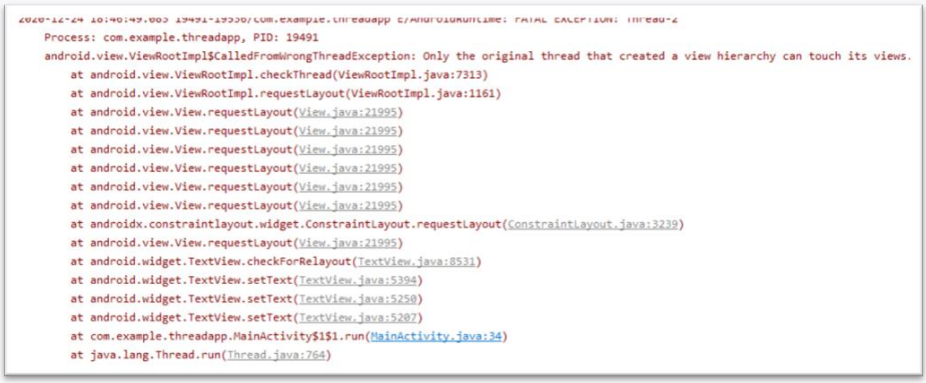
```
Runnable runnable = New Runnable() {
@Override
public void run() {
// отримуємо поточний час
Calendar c = Calendar.getInstance();
int hours = c.get(Calendar.HOUR_OF_DAY);
int minutes = c.get(Calendar.MINUTE);
int seconds = c.get(Calendar.SECOND);
String time = hours + ":" + minutes + ":" + seconds;
// відображаємо у текстовому полі
textView.setText(time);
}
};
```

Наприклад, отримуємо поточний час і намагаємося відобразити його в елементі TextView.

Далі визначаємо об'єкт потоку – об'єкт Thread, який приймає об'єкт Runnable. І за допомогою методу start() запускаємо потік:

```
// Визначаємо об'єкт Thread – новий потік
Thread thread = New Thread (runnable);
// Запускаємо потік
thread.start();
```

Ніби нічого складного. Але якщо ми запустимо програму і натиснемо кнопку, то ми зіткнемося з помилкою:



Оскільки змінювати стан візуальних елементів, звертатися до них ми можемо лише в основному потоці програми або UI потоку.

Для вирішення цієї проблеми взаємодії у вторинних потоках з елементами графічного інтерфейсу клас View() визначає метод post():

```
boolean post (Runnable action)
```

Як параметр він приймає завдання, яке треба виконати, і повертає логічне значення -true, якщо завдання Runnable успішно поміщено в чергу повідомлення, абоfalseякщо не вдалося розмістити в черзі

Також у класу View є аналогічний метод:

postDelayed():

```
boolean postDelayed (Runnable action, long millsec)
```

Він також запускає завдання лише через певний проміжок часу в мілісекундах, який вказується в другому параметрі.

Так, змінимо код MainActivity наступним чином

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;

public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
TextView textView = findViewById(R.id.textView);
button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// Визначаємо об'єкт Runnable
Runnable runnable = New Runnable() {
@Override
public void run() {
// отримуємо поточний час
Calendar c = Calendar.getInstance();
int hours = c.get(Calendar.HOUR_OF_DAY);
int minutes = c.get(Calendar.MINUTE);
int seconds = c.get(Calendar.SECOND);
String time = hours + ":" + minutes + ":" + seconds;
// відображаємо у текстовому полі
textView.post(new Runnable() {
public void run() {
textView.setText(time);
}
});
}
```


```
        };
        // Визначаємо об'єкт Thread – новий потік
        Thread thread = New Thread (runnable);
        // Запускаємо потік
        thread.start();
        }});
        }}
```

Тепер для оновлення TextView застосовується метод post:

```
        textView.post(new Runnable() {
        public void run() {
        textView.setText(time);
        }});
```

Тобто тут у методіrun()передається в методpost()об'єкта Runnable ми можемо звертатися до елементів візуального інтерфейсу та взаємодіяти з ними.

Подібним чином можна працювати з іншими віджетами, які успадковуються від класу View.

7.2. Потоки, фрагменти та ViewModel

У разі використання вторинних потоків слід враховувати наступний момент. Більш оптимальним способом є робота потоків з фрагментом, ніж

безпосередньо з діяльністю. Наприклад, визначимо у файлі activity_main.xml наступний інтерфейс:

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

<Button
android:id="@+id/progressBtn"
android:text="Запуск"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
app:layout_constraintBottom_toTopOf="@id/statusView"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:id="@+id/statusView"
android:text="Статус"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
app:layout_constraintBottom_toTopOf="@id/indicator"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@id/progressBtn" />
<ProgressBar
android:id="@+id/indicator"
style="@android:style/Widget.ProgressBar.Horizontal"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:max="100"
android:progress="0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/statusView"/>

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

Тут визначено кнопку для запуску вторинної задачі та елементи TextView та ProgressBar, які відображають індикацію виконання завдання.

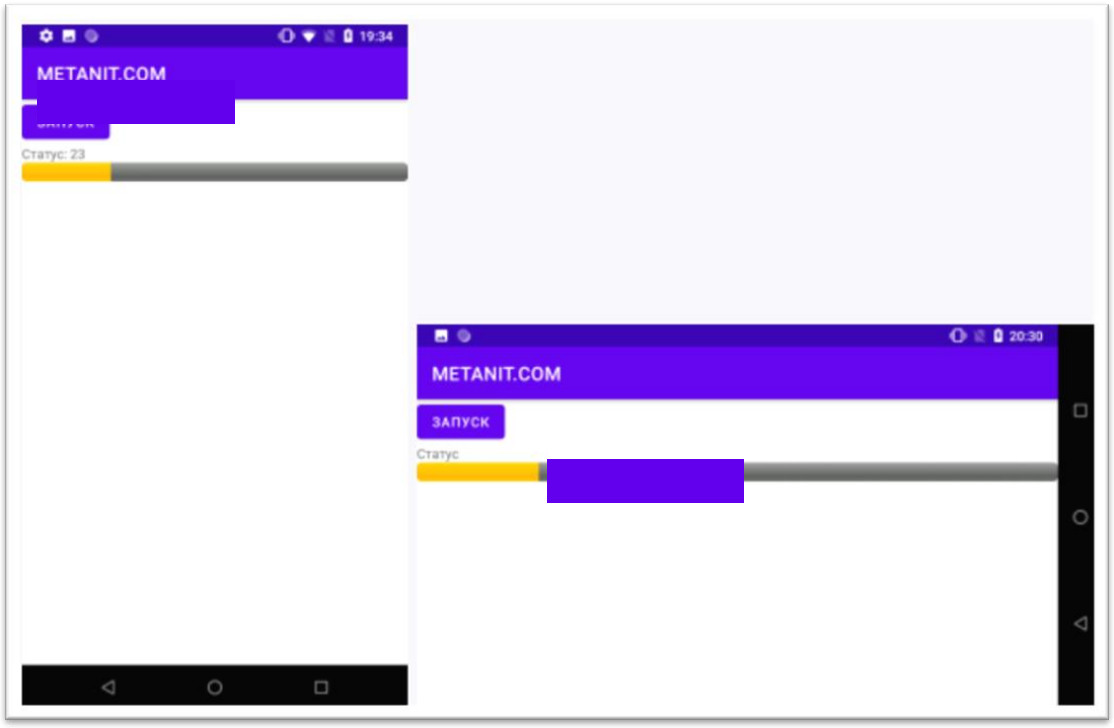
У класі MainActivity визначимо наступний код:

```
package com.example.threadapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

int currentValue = 0;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

ProgressBar indicatorBar = findViewById(R.id.indicator);
TextView statusView = findViewById(R.id.statusView);
button btnFetch = findViewById(R.id.progressBtn);
btnFetch.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

Runnable runnable = New Runnable() {
@Override
public void run() {

for(; currentValue <= 100; currentValue++){
try {
statusView.post(new Runnable() {
public void run() {
indicatorBar.setProgress(currentValue);
statusView.setText("Статус:" + currentValue);
}
});


} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
};
Thread thread = New Thread (runnable);
thread.start();
}
});
}
}
```

Тут, натиснувши кнопку, ми запускаємо завдання Runnable, в якій в циклі від 0 до 100 змінюємо показники ProgressBar і TextView, імітуючи деяку тривалу роботу.

Однак якщо в процесі роботи завдання ми змінимо орієнтацію мобільного пристрою, то відбудеться перестворення activity, і програма перестане працювати належним чином.



У разі проблема впирається у стан, яким оперує потік, саме - змінниcurrentValue, до якої прив'язані віджети в Activity.

7.3. Додавання ViewModel

Для подібних випадків як вирішення проблеми пропонується використовувати ViewModel. Отже, додамо до тієї ж папки, де знаходиться файл MainActivity.java, новий клас MyViewModel з наступним кодом:

```
package com.example.threadapp;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

public class MyViewModel extends ViewModel {

private MutableLiveData<Boolean> isStarted = new
MutableLiveData<Boolean>(false);
private MutableLiveData<Integer> value;
public LiveData<Integer> getValue() {
if (value == null) {
value = new MutableLiveData<Integer>(0);
}
```

```
return value;
}
public void execute(){
if(!isStarted.getValue()){
isStarted.postValue(true);
Runnable runnable = New Runnable() {
@Override
public void run() {

for(int i = value.getValue(); i <= 100; i++){
try {
value.postValue(i);
Thread.sleep(400);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
};
Thread thread = New Thread (runnable);
thread.start();
}
}
}
```

Отже, тут визначено класMyViewModel, який успадкований від класу ViewModel, спеціально призначеного для зберігання та керування станом чи моделлю.

Як стан тут визначено для об'єкта. Насамперед це числове значення, до яких будуть прив'язані віджети MainActivity. І по-друге, нам потрібен певний індикатор того, що потік вже запущений, щоб натискання на кнопку не було запущено зайвих потоків.

Для зберігання числового значення призначена змінnavalue:

```
private MutableLiveData<Integer> value;
```

Для прив'язки до цього значення воно має типMutableLiveData. А оскільки ми зберігатимемо в цій змінній числове значення, то тип змінної типізований типомInteger.

Для доступу ззовні класу до цього значення визначено методetValue, який має типLiveDatai який при першому зверненні до змінної встановлює 0 або просто повертає значення змінної:

```
public LiveData<Integer> getValue() {
if (value == null) {
value = new MutableLiveData<Integer>(0);
}
return value;
}
```

Для індикації, чи запущено потік, визначено змінnyisStarted, яка зберігає значення типуBoolean, тобто фактичнotrueабоfalse. За умовчанням вона має значенняfalse(тобто потік не запущений).

Для зміни числового значення, до якого будуть прив'язані віджети, визначено методexecute(). Він запускає потік, якщо не запущений потік:

```
if(!isStarted.getValue()){
```

Далі перемикає значення змінноїisStartednatrueоскільки ми запускаємо потік.

У потоці також запускається цикл

```
for(int i = value.getValue(); i <= 100; i++){
```

І в даному випадку ми користуємося перевагою класу ViewModel, що дозволяє автоматично зберігати свій стан.

Причому лічильник циклу як початкове значення бере значення зі змінноїvaluei збільшується на одиницю, доки не дійде до ста.

У самому циклі змінюється значення змінноїvalueза допомогою передачі значення методpostValue()

```
 value.postValue(i);
```

Таким чином, в циклі здійсниться прохід від 0 до 100 і при кожній ітерації циклу буде змінюватися значення змінної value.

Тепер задіємо наш класMyViewModeli для цього змінимо код класу MainActivity:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;

import android.os.Bundle;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

ProgressBar indicatorBar = findViewById(R.id.indicator);
TextView statusView = findViewById(R.id.statusView);
button btnFetch = findViewById(R.id.progressBtn);
MyViewModel model = новий
```

```
ViewModelProvider(this).get(MyViewModel.class);

     model.getValue().observe(this, value -> {
     indicatorBar.setProgress(value);
     statusView.setText("Статус:" + value);
     });
     btnFetch.setOnClickListener(v -> model.execute());
     }
     }
```

Щоб задіяти MyViewModel, створюємо об'єкт класу ViewModelProvider, конструктор якого передається об'єкт-власник ViewModel. В даному випадку це поточний об'єкт MainActivity:

new ViewModelProvider(this)

І далі за допомогою методуget()створюємо об'єкт класу ViewModel, який використовуватиметься в об'єкті MainActivity.

```
     MyViewModel model = новий
ViewModelProvider(this).get(MyViewModel.class);
```

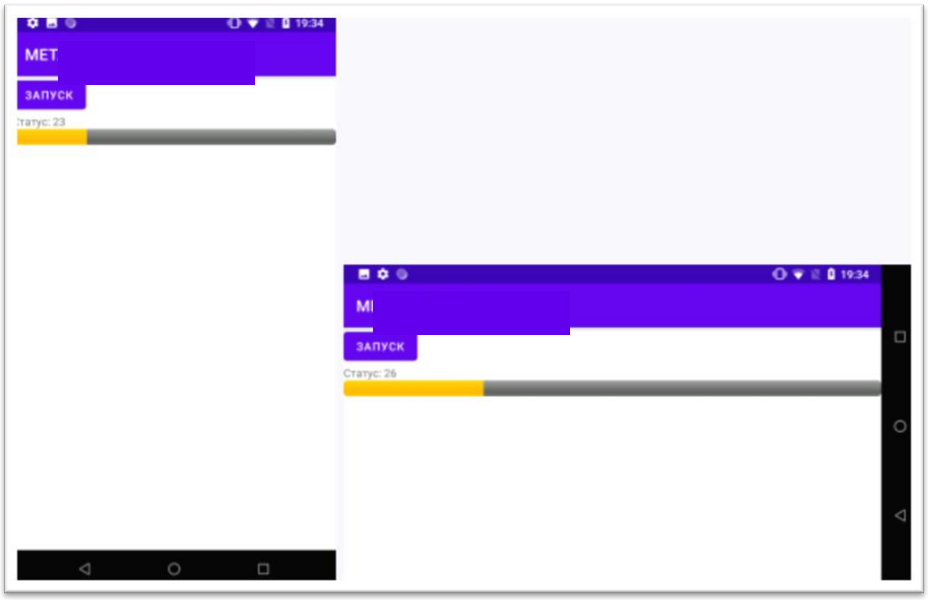
Отримавши об'єкт MyViewModel, визначаємо прослуховування змін його змінноїvalueза допомогою методу observe:

```
     model.getValue().observe(this, value -> {
     indicatorBar.setProgress(value);
     statusView.setText("Статус:" + value);
     });
```

Методobserve()як перший параметр приймає власника функції обсервера - у разі поточний об'єкт MainActivity. А як другий параметр - функцію обсервера (а точніше об'єкт інтерфейсуObserver). Функція обсервера приймає один параметр - нове значення змінної, що відстежується (тобто в даному випадку змінної value). Отримавши нове значення змінної value, ми змінюємо параметри віджетів.

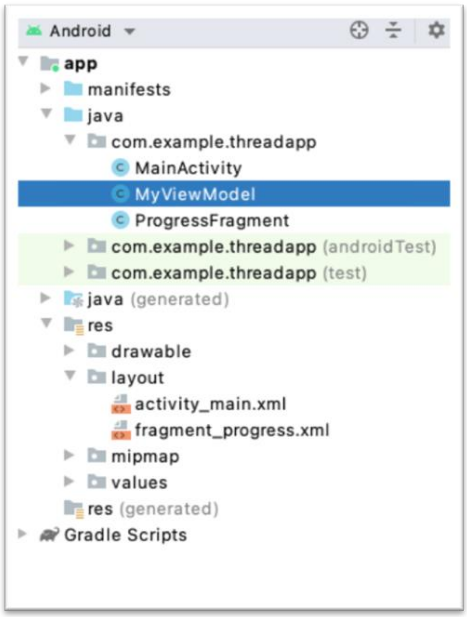
Таким чином, при кожній зміні значення змінної value віджети отримають її нове значення.

Тепер якщо ми запустимо програму, то незалежно від зміни орієнтації мийного пристрою фонове завдання продовжуватиме свою роботу:



7.4. Використання фрагментів

Аналогічно ми можемо використовувати фрагменти. Отже, додамо до проекту новий фрагмент, який назвемо ProgressFragment.



Визначимо новий файл розмітки інтерфейсу fragment_progress.xml:

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

<Button
android:id="@+id/progressBtn"
android:text="Запуск"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toTopOf="@id/statusView"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

    <TextView
    android:id="@+id/statusView"
    android:text="Статус"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toTopOf="@id/indicator"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/progressBtn" />
    <ProgressBar
    android:id="@+id/indicator"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/statusView"/>

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

Сам клас фрагмента ProgressFragment змінимо так:

```
    package com.example.threadapp;

    import android.os.Bundle;
    import androidx.fragment.app.Fragment;
    import androidx.lifecycle.ViewModelProvider;
    import android.util.Log;
    import android.view.LayoutInflater;
    import android.view.View;
    import android.view.ViewGroup;
    import android.widget.Button;
    import android.widget.ProgressBar;
    import android.widget.TextView;


    public class ProgressFragment extends Fragment {

    @Override
    Public    View    onCreateView(LayoutInflater    inflater,
ViewGroup container, Bundle savedInstanceState) {

    View  view  =  inflater.inflate(R.layout.fragment_progress,
```

```
container, false);

     ProgressBar       indicatorBar       =       (ProgressBar)
view.findViewById(R.id.indicator);
     TextView          statusView          =          (TextView)
view.findViewById(R.id.statusView);
     Button                        btnFetch                        =
(Button)view.findViewById(R.id.progressBtn);

     MyViewModel            model            =            новий
ViewModelProvider(requireActivity()).get(MyViewModel.class);

     model.getValue().observe(getViewLifecycleOwner(), value ->
{
     indicatorBar.setProgress(value);
     statusView.setText("Статус:" + value);
     });
     btnFetch.setOnClickListener(v -> model.execute());
     return view;
     }
     }
```

Тут аналогічно застосовується клас MyViewModel. Єдино для отримання асоційованої із фрагментом Activity тут застосовується метод requireActivity(). А для отримання власника життєвого циклу – методgetViewLifecycleOwner.

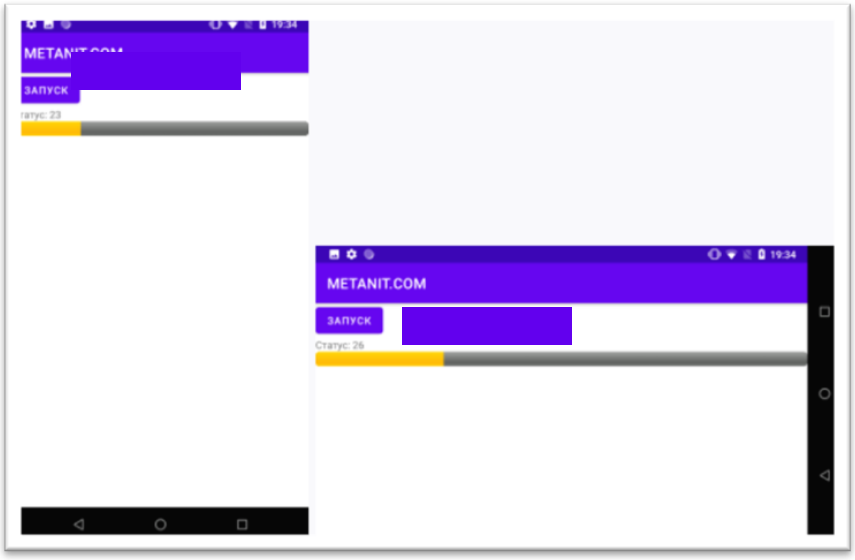
Тепер зв'яжемо фрагмент з діяльністю. Для цього визначимо у файлі activity_main.xml наступний код:

```
     <androidx.fragment.app.FragmentContainerView
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/fragment_container_view"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:name="com.example.threadapp.ProgressFragment" />
```

А сам клас MainActivity скоротимо:

```
     package com.example.threadapp;
     import androidx.appcompat.app.AppCompatActivity;
     import android.os.Bundle;
     public class MainActivity extends AppCompatActivity {
     @Override
     protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     }
     }
```

І код із фрагементом працюватиме аналогічно:



7.5. Клас AsyncTask

У минулих статтях був описаний загальний підхід, який застосовується зараз для запуску в додатку нового потоку та оновлення в ньому інтерфейсу користувача. Розглянемо інший підхід, який представляє клас AsyncTask. Хоча застосування AsyncTask в сучасних програмах Android застарів. Тим не менш, оскільки він, як і раніше, широко застосовується, також розглянемо його.

Щоб використовувати AsyncTask, нам треба:

1. Створити клас, похідний від AsyncTask (як правило, для цього створюється внутрішній клас в діяльності або у фрагменті)
2. Перевизначити один або кілька методів AsyncTask для виконання певної роботи у фоновому режимі
3. При необхідності створити об'єкт AsyncTask та викликати його методexecute()для початку роботи

Отже, створимо найпростішу програму з використанням AsyncTask. Визначимо наступну розмітку інтерфейсу у файлі activity_main.xml:

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:orientation="vertical">

    <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
```

```
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="22sp"
    android:id="@+id/clicksView"
    android:text="Clicks: 0"/>
    <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/clicksBtn"
    android:text="Click" />
    </LinearLayout>

    <Button
    android:id="@+id/progressBtn"
    android:text="Запуск"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

    <TextView
    android:id="@+id/statusView"
    android:text="Статус"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
    <ProgressBar
    android:id="@+id/indicator"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="0" />
    </LinearLayout>
```

Тут визначено кнопку для запуску фонового потоку, а також текстове поле та прогрессбар для індикації виконання завдання. Крім того, тут визначені додаткова кнопка, яка збільшує число кліків, і текстове поле, яке виводить число кліків.

Далі визначимо у класі MainActivity наступний код:

```
    import android.os.AsyncTask;
    import android.os.SystemClock;
    import android.support.v7.app.AppCompatActivity;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.Button;
    import android.widget.ProgressBar;
    import android.widget.TextView;
    import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {

int[] integers=null;
int clicks = 0;
ProgressBar indicatorBar;
TextView statusView;
TextView clicksView;
Button progressBtn;
button clicksBtn;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

integers = new int[100];
for(int i=0;i<100;i++) {
integers[i] = i + 1;
}
indicatorBar = (ProgressBar) findViewById(R.id.indicator);
statusView = findViewById(R.id.statusView);
progressBtn = findViewById(R.id.progressBtn);
progressBtn.setOnClickListener(new  View.OnClickListener()
{
@Override
public void onClick(View v) {

new ProgressTask().execute();
}
});

clicksView = findViewById(R.id.clicksView);
clicksBtn = findViewById(R.id.clicksBtn);
clicksBtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

clicks++;
clicksView.setText("Clicks: " + clicks);
}
});
}

class ProgressTask extends AsyncTask<Void, Integer, Void>
{
@Override
protected Void doInBackground(Void... unused) {
for (int i = 0; i<integers.length;i++) {

publishProgress(i);
```

```
     SystemClock.sleep(400);
     }
     return(null);
     }
     @Override
     protected void onProgressUpdate(Integer... items) {
     indicatorBar.setProgress(items[0]+1);
     statusView.setText("Статус:"                              +
String.valueOf(items[0]+1));
     }
     @Override
     protected void onPostExecute(Void unused) {
     Toast.makeText(getApplicationContext(),          "Завдання
завершено", Toast.LENGTH_SHORT)
     .show();
     }
     }
     }
```

Клас завдання ProgressTask визначено як внутрішній клас. Він успадковується не просто від AsyncTask, а від його типової версіїAsyncTask<Void, Integer, Void>. Вона типізується трьома типами:

- Клас для зберігання інформації, яка потрібна для обробки завдання
- Тип об'єктів, що використовуються для індикації процесу виконання завдання
- Тип результату задачі

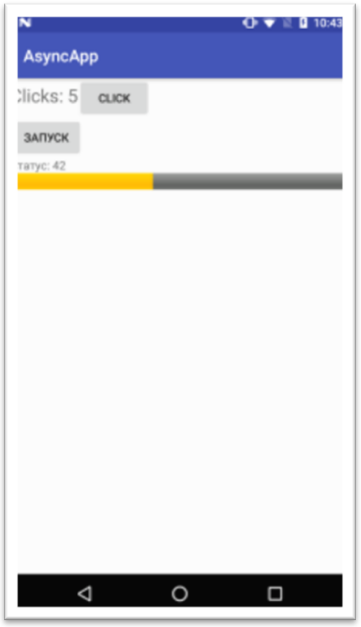
Ці типи можуть бути різними класами. В даному випадку сутність завдання полягатиме в переборі масиву integers, що представляє набір елементів Integer. І тут нам не треба передавати в завдання жодний об'єкт, тому перший тип іде як Void.

Для індикації перебору використовуються цілі числа, які показують, який об'єкт із масиву ми зараз перебираємо. Тому як другий тип використовується Integer.

Як третій тип використовується знову Void, оскільки в даному випадку не треба нічого повертати із завдання.

AsyncTask містить чотири методи, які можна перевизначити:

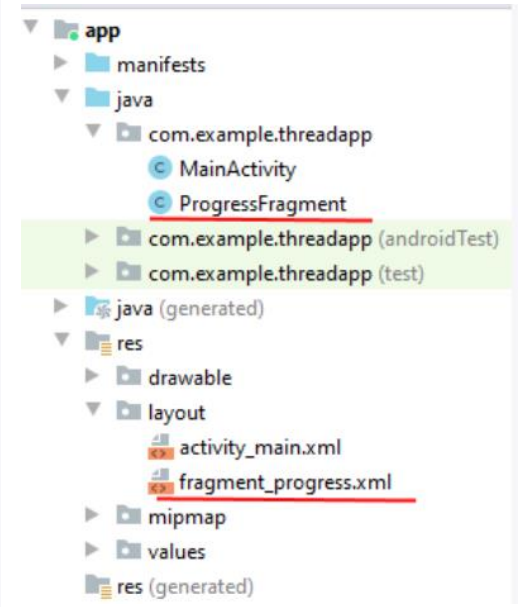
- Метод doInBackground(): виконується у фоновому потоці, має повертати певний результат
- Метод onPreExecute(): викликається з головного потоку перед запуском методуdoInBackground()



- Метод onPostExecute(): виконується з головного потоку після завершення роботи методуdoInBackground()
- Метод onProgressUpdate(): дозволяє сигналізувати користувачеві про виконання фонового потоку

Так як метод doInBackground() не приймає нічого і не повертає нічого, то як його параметр використовуєтьсяVoid...- масив Void, і як повертається типу - теж Void. Ці типи відповідають першому і третьому типуAsyncTask<Void, Integer, Void>.

Метод doInBackground() перебирає масив і за кожної ітерації повідомляє систему за допомогою методу publishProgress(item). Оскільки у разі для індикації використовуються цілі числа, то параметрitemмає представляти ціле число.



МетодonProgressUpdate(Integer... items)отримує передане вище число та застосовує його для налаштування текстового поля та прогресу.

МетодonPostExecute()виконується після завершення завдання і як параметр приймає об'єкт, що повертається методом doInBackground() - тобто в даному випадку об'єкт типу Void. Щоб сигналізувати закінчення роботи, виводиться на екран спливаюче повідомлення.

Запустимо програму. Запустимо завдання, натиснувши кнопку.

При цьому, поки виконується завдання, ми можемо паралельно натискати на другу кнопку і збільшувати кількість кліків, або виконувати якусь іншу роботу в додатку.

7.6. AsyncTask та фрагменти

При використанні AsyncTask слід враховувати наступний момент. Більш оптимальним способом є робота AsyncTask з фрагментом, ніж безпосередньо з діяльності. Наприклад, якщо ми візьмемо проект із минулої теми, запустимо програму та змінимо орієнтацію мобільного пристрою, то відбудеться перестворення activity. У разі зміни орентації

пристрою потік AsyncTask буде продовжувати звертатися до старої діяльності замість нової. Тому в цьому випадку краще використати фрагменти.

Отже, візьмемо проект із минулої теми та додамо до нього новий фрагмент, який назвемо ProgressFragment.

Визначимо новий файл розмітки інтерфейсу fragment_progress.xml:

```
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_progress"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:orientation="vertical">
    <Button
    android:id="@+id/progressBtn"
    android:text="Запуск"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

    <TextView
    android:id="@+id/statusView"
    android:text="Статус"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
    <ProgressBar
    android:id="@+id/indicator"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="0" />
    </LinearLayout>
```

Сам клас фрагмента ProgressFragment змінимо так:

```
    package com.example.eugene.asyncapp;

    import android.widget.Button;
    import android.os.AsyncTask;
    import android.os.Bundle;
    import android.app.Fragment;
    import android.os.SystemClock;
    import android.view.LayoutInflater;
    import android.view.View;
    import android.view.ViewGroup;
    import android.widget.ProgressBar;
    import android.widget.TextView;
```

```
import android.widget.Toast;
import android.view.View.OnClickListener;

public class ProgressFragment extends Fragment {

int[] integers=null;
ProgressBar indicatorBar;
TextView statusView;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setRetainInstance(true);
}

@Override
Public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState) {
View view = inflater.inflate(R.layout.fragment_progress,
container, false);
integers = new int[100];
for(int i=0;i<100;i++) {
integers[i] = i + 1;
}
indicatorBar = (ProgressBar)
view.findViewById(R.id.indicator);
statusView = (TextView)
view.findViewById(R.id.statusView);
Button btnFetch =
(Button)view.findViewById(R.id.progressBtn);
btnFetch.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {

new ProgressTask().execute();
}
});
return view;
}

class ProgressTask extends AsyncTask<Void, Integer, Void>
{
@Override
protected Void doInBackground(Void... unused) {
for (int i = 0; i<integers.length;i++) {

publishProgress(i);
SystemClock.sleep(400);
}
```

```
     return null;
     }
     @Override
     protected void onProgressUpdate(Integer... items) {
     indicatorBar.setProgress(items[0]+1);
     statusView.setText("Статус:"                                    +
String.valueOf(items[0]+1));
     }
     @Override
     protected void onPostExecute(Void unused) {
     Toast.makeText(getActivity(),     "Завдання     завершено",
Toast.LENGTH_SHORT)
     .show();
     }
     }
     }
```

Тут визначено всі ті дії, які були розглянуті в минулій темі та які раніше перебували у класі MainActivity. Варто відзначити виклик setRetainInstance(true) у методі onCreate(), який дозволяє зберігати стан фрагмента незалежно від зміни орієнтації.

Тепер зв'яжемо фрагмент з діяльністю. Для цього визначимо у файлі activity_main.xml наступний код:

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

     <fragment
     android:id="@+id/progressFragment"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:name="com.example.eugene.asyncapp.ProgressFragment
"/>
     </LinearLayout>
```

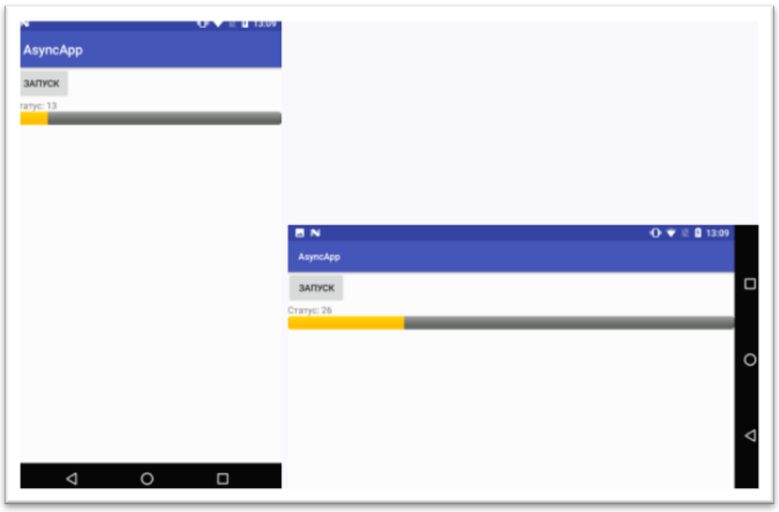
А сам клас MainActivity скоротимо:

```
     package com.example.eugene.asyncapp;

     import android.support.v7.app.AppCompatActivity;
     import android.os.Bundle;

     public class MainActivity extends AppCompatActivity {

     @Override
     protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        }
        }
```

Тепер якщо ми запустимо програму, то незалежно від зміни орієнтації мийного пристрою фонове завдання продовжуватиме свою роботу:

8.



8. СТИЛІ ТА ТЕМИ

8.1. Стилі

Ми можемо налаштувати елемент за допомогою різних атрибутів, які задають висоту, ширину, колір тла, тексту тощо. Але якщо у нас кілька елементів використовують одні й ті самі налаштування, ми можемо об'єднати ці налаштування в стилі.

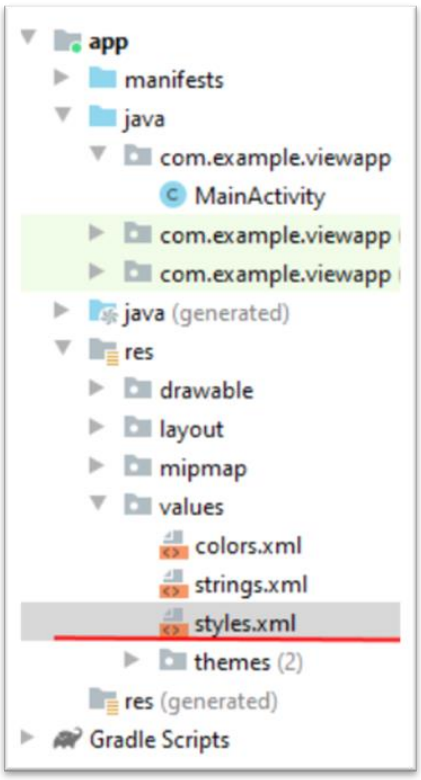
Наприклад, нехай ми маємо кілька елементів TextView:

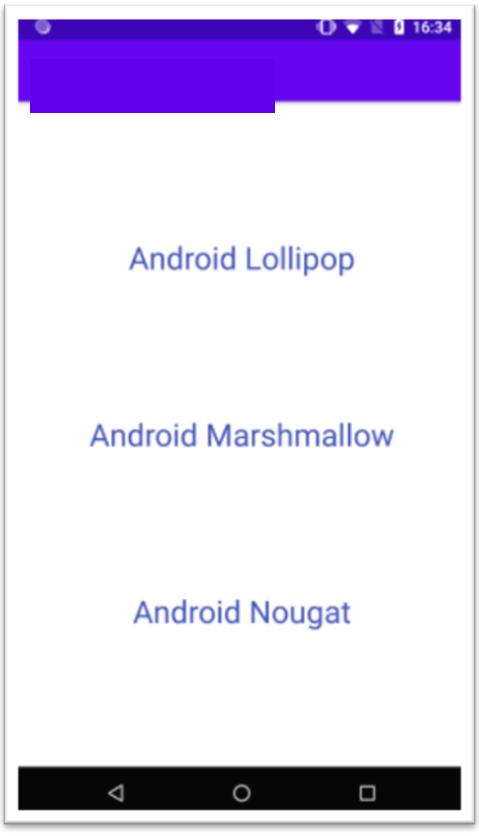
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent">

<TextView
android:id="@+id/textView1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:textSize="28sp"
android:textColor="#3f51b5"
android:text="Android Lollipop"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toTopOf="@+id/textView2"
/>
<TextView
android:id="@+id/textView2"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:textSize="28sp"
android:textColor="#3f51b5"
android:text="Android Marshmallow"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@+id/textView3"
app:layout_constraintTop_toBottomOf="@+id/textView1"
/>
<TextView
android:id="@+id/textView3"
android:layout_width="0dp"
```

```
    android:layout_height="wrap_content"
    android:gravity="center"
    android:textSize="28sp"
    android:textColor="#3f51b5"
    android:text="Android Nougat"

    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2"
    />

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

Всі ці TextView мають однаковий набір властивостей, і, наприклад, якщо нам захочеться змінити колір тексту, доведеться змінювати його у всіх трьох TextView. Цей підхід не є оптимальним, а більш оптимальний підхід є використанням стилів, які визначаються в проекті в папці res/values.

Отже, додамо до проекту папку res/values новий елемент Value Resourse File, який назвемо styles.xml:





Визначимо у файлі styles.xml такий вміст:

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
    <style name="TextViewStyle">
```

```
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#3f51b5</item>
    <item name="android:textSize">28sp</item>
    <item name="android:gravity">center</item>
    </style>
    </resources>
```

Тут визначено новий стиль TextViewStyle, який за допомогою елементів item визначає значення для атрибутів TextView.

Стиль задається за допомогою елемента`<style>`. Атрибут name вказує на назву стилю, через яку потім можна посилатися на нього. Необов'язковий атрибут`parent`встановлює для цього стилю батьківський стиль, від якого дочірній стиль успадковуватиме всі свої характеристики.

За допомогою елементів`item`встановлюються конкретні властивості віджету, який приймає значення атрибута name ім'я встановлюваного властивості.

Тепер застосуємо стиль, змінимо файл activity_main.xml:

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

    <TextView
    android:id="@+id/textView1"

    style="@style/TextViewStyle"

    android:text="Android Lollipop"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/textView2"
    />
    <TextView
    android:id="@+id/textView2"
    style="@style/TextViewStyle"
    android:text="Android Marshmallow"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/textView3"
    app:layout_constraintTop_toBottomOf="@+id/textView1"
    />
    <TextView
    android:id="@+id/textView3"
```

```
    style="@style/TextViewStyle"
    android:text="Android Nougat"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2"
    />

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

Використовуючи визначення`style="@style/TextViewStyle"`текстове поле пов'язують із визначенням стилю. Підсумковий результат буде той же, що й раніше, тільки коду стає меншим. А якщо ми захочемо змінити якісь параметри, то досить змінити необхідний елемент item у визначенні іміджу.

8.2. Теми

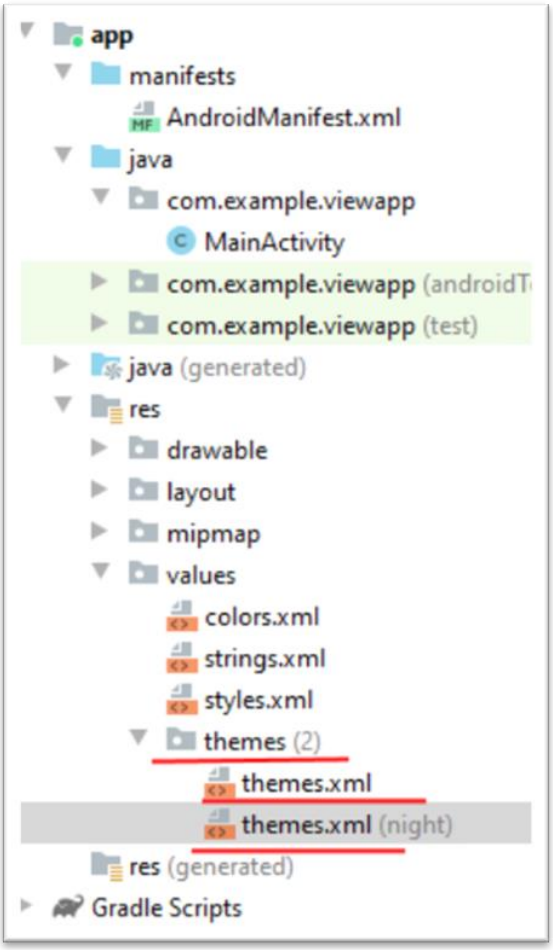
Крім застосування окремих стилів до окремих елементів, ми можемо задавати стилі для всієї програми або activity у вигляді тем. Тема представляє колекцію атрибутів, які застосовуються в цілому до всього додатку, класу activity або ієрархії віджетів.

Ми можемо створити тему. Однак Android вже надає кілька встановлених тем для стилізації програми, наприклад,`Theme.AppCompat.Light.DarkActionBar`та низку інших.

За промовчанням програма вже застосовує теми. Так відкриємо файл AndroidManifest.xml. У ньому ми можемо побачити таке визначення елемента application, що представляє програму:

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

Завдання теми відбувається за допомогою атрибуту`android:theme`. В даному випадку використовується ресурс, який називається у моєму випадку Theme.ViewApp. За промовчанням файли тем визначені в папці res/values. Зокрема, тут можна знайти умовний каталог themes, в якому є два елементи за замовчуванням: themes.xml:



Один файл представляє світлу тему, а інший – темну. Наприклад, відкриємо файл themes.xml зі світлою темою:

```
<resources xmlns:tools="http://schemas.android.com/tools">
<!-- Base application theme. -->
<style name="Theme.ViewApp"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
<!-- Primary brand color. -->
<item name="colorPrimary">@color/purple_500</item>
<item name="colorPrimaryVariant">@color/purple_700</item>
<item name="colorOnPrimary">@color/white</item>
<!-- Secondary brand color. -->
<item name="colorSecondary">@color/teal_200</item>
<item name="colorSecondaryVariant">@color/teal_700</item>
<item name="colorOnSecondary">@color/black</item>
<!-- Status bar color. -->
<item name="android:statusBarColor"
tools:targetApi="l">?attr/colorPrimaryVariant</item>
<!-- Customize your theme here. -->
</style>
</resources>
```

Тут ми можемо побачити, що тема визначається як стиль за допомогою елемента style.. Атрибут parent вказує на батьківську тему, від якої поточна тема бере всі стильові характеристики. Тобто тема

"Theme.ViewApp" використовує іншу тему - "Theme.MaterialComponents.DayNight.DarkActionBar". І крім того, визначає низку своїх власних стилів.

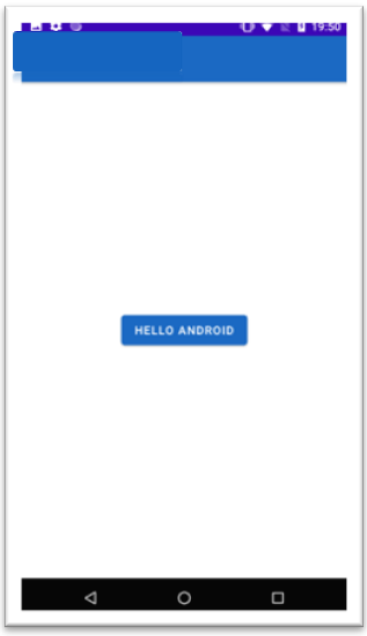
Також можна помітити, що тут визначаються не лише характеристики для атрибутів, а й семантичні імена, наприклад,`colorPrimary`, Який зіставлений ресурс "@color/purple_500".

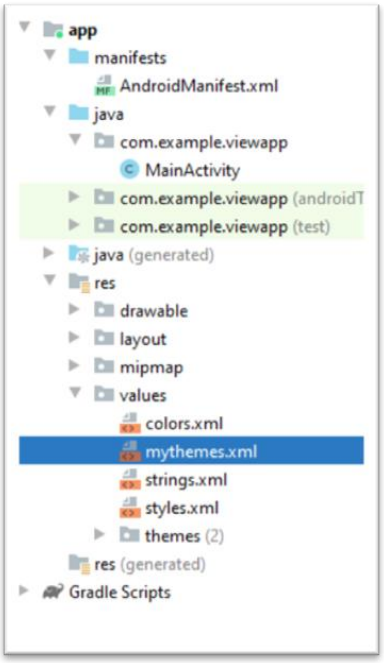
У разі потреби ми можемо змінити ці характеристики або доповнити тему новими стильовими характеристиками. Наприклад, змінимо колір властивості`colorPrimary`, яке застосовується у тому числі як фоновий колір заголовка і кнопки:

```
<item name="colorPrimary">#1565C0</item>
```

І відповідно зміниться колір за промовчанням для фону заголовка та кнопки:

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

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello Android"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
/>

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





8.3. Створення власної теми

Замість використання вбудованих тем, ми можемо створити свою. Для цього додамо до папки res/values новий файл mythemes.xml і визначимо в ньому такий вміст:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="MyTheme"
parent="Theme.AppCompat.Light">
<item
name="android:textColor">#FF018786</item>
<item name="android:textSize">28sp</item>
</style>
</resources>
```

Отже, ми створили стиль MyTheme, який успадкований від стилю`Theme.AppCompat.Light`. У цьому стилі ми перевизначили дві властивості: висоту шрифту (textSize) – 28sp, а також колір тексту (textColor) –`#FF018786`.

Тепер визначимо цей стиль як тему програми у файлі AndroidManifest.xml:

```
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/MyTheme"><!-- застосування теми -->
```

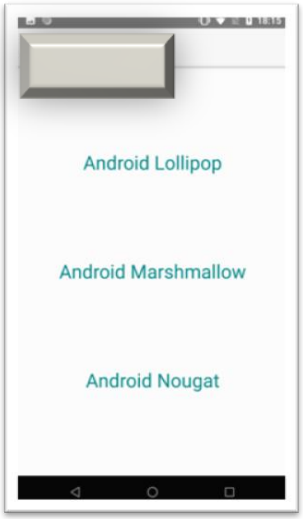
Нехай у нас буде наступна розмітка у activity_main.xml

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

<TextView
android:id="@+id/textView1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Android Lollipop"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toTopOf="@+id/textView2"
/>
<TextView
android:id="@+id/textView2"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Android Marshmallow"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@+id/textView3"
app:layout_constraintTop_toBottomOf="@+id/textView1"
/>
<TextView
android:id="@+id/textView3"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Android Nougat"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toBottomOf="@+id/textView2"
/>

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

Як видно, для елементів TextView не встановлюється атрибут`textSize`і`textColor`, однак оскільки вони визначені в темі, яка застосовується глобально до нашої програми, то елементи TextView підхоплюватимуть ці стильові характеристики:

8.4. Застосування теми до діяльності

Вище теми застосовувалися глобально до всього додатку. Але також їх можна застосувати до окремого класу Activity. Для цього потрібно підкоригувати файл маніфесту AndroidManifest. Наприклад:




```
      <?xml version="1.0" encoding="utf-8"?>
      <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.viewapp"
      android:versionCode="1"
      android:versionName="1.0">

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

      android:theme="@style/Theme.ViewApp">
      <діяльність                    android:name=".MainActivity"
android:theme="@style/MyTheme">
      <intent-filter>
      <action android:name="android.intent.action.MAIN" />

      <category   android:name="android.intent.category.LAUNCHER"
/>
      </intent-filter>
      </activity>
      </application>

      </manifest>
```

Атрибут android:theme елемент<діяльність>вказує на тему, що застосовується до MainActivity. Тобто глобально до застосування застосовується тема "Theme.ViewApp", а до MainActivity - "MyTheme".

8.5. Застосування теми до ієрархії віджетів

Також можна застосувати тему ієрархії віджетів, встановивши атрибут android:theme у елемента, до якого (включаючи його вкладені елементи) ми хочемо застосувати тему. Наприклад, застосування теми до ConstraintLayout та її елементів:

```
<?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"
android:layout_width="match_parent"
android:layout_height="match_parent"

android:theme="@style/MyTheme">

<TextView
android:id="@+id/textView1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Android Lollipop"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
/>

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

Навчальне видання

КОНСПЕКТ ЛЕКЦІЙ
навчально-методичного комплексу
дистанційного курсу дисципліни
«Програмування для мобільних платформ»
Частина І
(для студентів спеціальностей
F2 «Інженерія програмного забезпечення»,
F6 «Інформаційні технології та системи»)

Укладач:
Ратов Денис Валентинович

Редактор *Д.В.Ратов*
Техн. редактор *Д.В.Ратов*
Оригінал - макет *Д.В.Ратов*

Підписано до друку _____
Формат $60\times84\,{}^{1}\!/_{16}$. Папір типограф. Гарнитура *Times*.
Друк офсетний. Ум. друк. арк.___. Обл.-вид.арк._____.
Тираж ___ прим. Вид. №_____. Замовл. №_____. Ціна договірна.

Видавництво Східноукраїнського національного університету
імені Володимира Даля

Свідоцтво про реєстрацію: серія ДК № 1620 від 18.12.03 р.
Адреса університета: вул. Іоанна Павла II, 17
м. Київ, 01042, Україна
e-mail: vidavnictvoSNU.ua@gmail.com.