ОСНОВИ СТВОРЕННЯ ІНТЕРФЕЙСУ

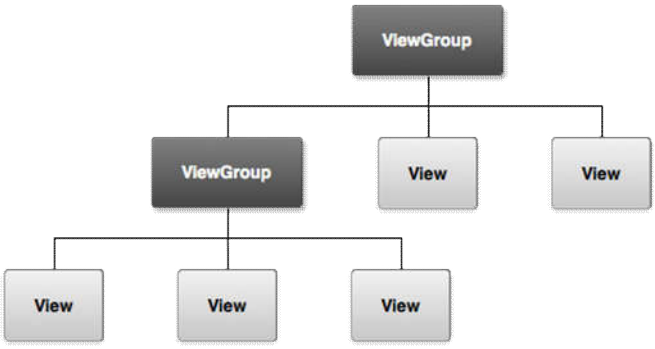
План:

- Вступ в створення інтерфейсу
- Стратегії визначення інтерфейсу
- Додавання файлу layout
- Отримання елементів в коді і їх ідентифікатори
- Графічні можливості Android Studio
- Визначення розмірів
- Ширина і висота елементів
- Програмна встановлення ширини і висоти
- Внутрішні і зовнішні відступи
- Програмне встановлення відступів
- LinearLayout
- Програмне створення LinearLayout
- RelativeLayout
- Gravity і layout_gravity
- Gravity і layout_gravity
- TableLayout
- FrameLayout
- GridLayout
- ConstraintLayout
- ScrollView
- Вкладені layout

Вступ в створення інтерфейсу

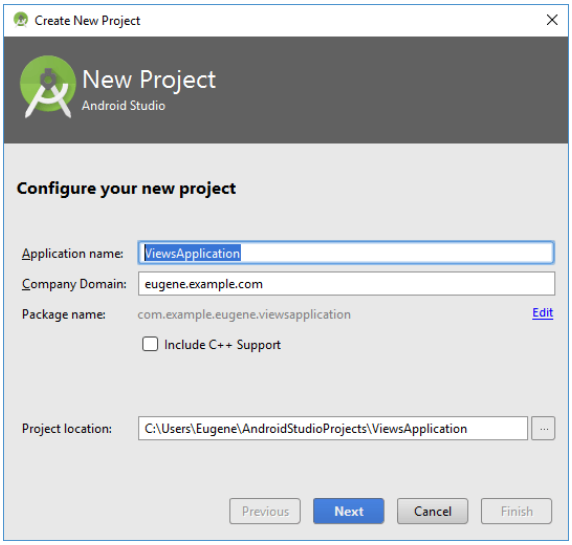
Графічний інтерфейс користувача є ієрархією об'єктів android.view.View і android.view.ViewGroup . Кожен об'єкт ViewGroup представляє контейнер, який містить і впорядковує дочірні об'єкти View . Зокрема, до контейнерів відносять такі елементи, як RelativeLayout, LinearLayout, GridLayout, ConstraintLayout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

Прості об'єкти View є елементами управління та інші віджети, наприклад, кнопки, текстові поля і т.д., через які користувач взаємодіє з програмою:

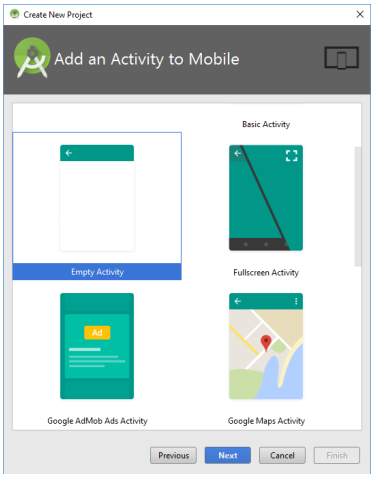


Більшість візуальних елементів, що наслідуються від класу View, такі як кнопки, текстові поля і інші, розташовуються в пакеті android.widget

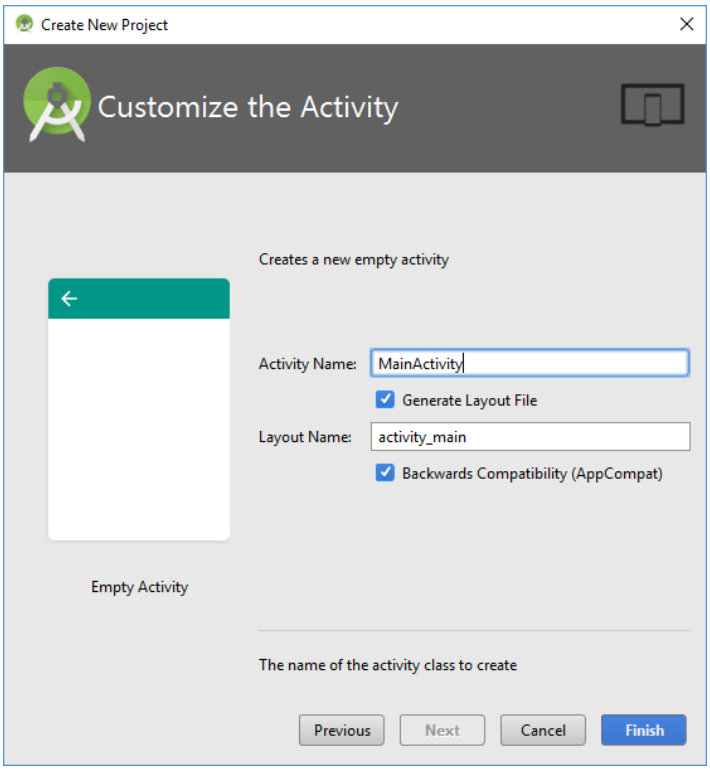
Для роботи з візуальними елементами створимо новий проект. Нехай він буде називатися ViewsApplication:



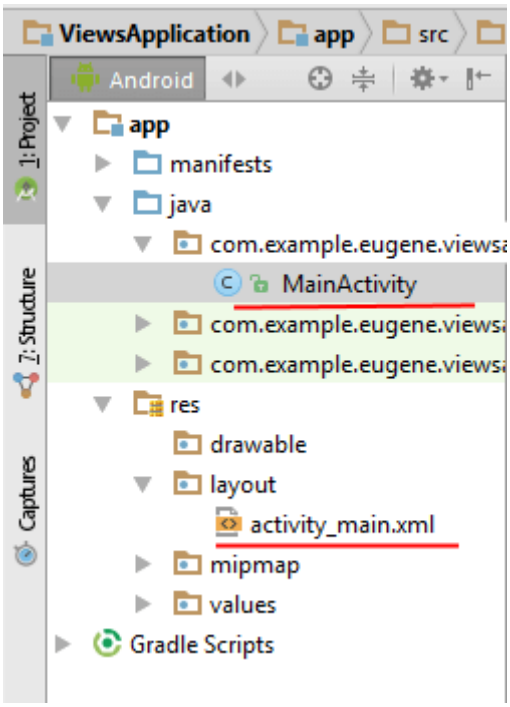
Як шаблон проекту виберемо Empty Activity :



Ім'я activity при створенні проекту залишимо за замовчуванням - MainActivity:



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



Стратегії визначення інтерфейсу

Розмітка визначає візуальну структуру користувальницького інтерфейсу. Встановити розмітку можна двома способами:

- Створити елементи управління програмно в коді java
- Оголосити елементи інтерфейсу в XML

- Поєднання обох способів - базові елементи розмітки визначити в XML, а решта додавати під час виконання

Створення інтерфейсу в коді java

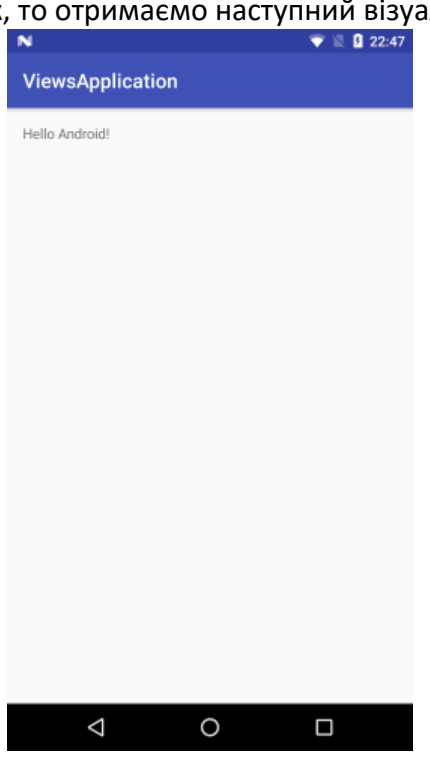
Визначимо в класі **MainActivity** найпростіший інтерфейс:

```
package com.example.ozm.helloworld;

import android.support.v7.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);
    }
}
```

Тут весь інтерфейс представлений елементом TextView, який призначени1 для виводу тексту. За допомогою методів, які, як правило, починаються на **set** , можна встановити різні властивості TextView. Наприклад, в даному випадку метод setText()встановлює текст в поле, а setTextSize()задає висоту шрифту.

Для встановлення елементу в якості інтерфейсу додатку в коді Activity викликається метод **setContentView ()** , в який передається візуальний елемент.

Якщо ми запустимо додаток, то отримаємо наступний візуальний інтерфейс:



Хоча ми можемо використовувати подібний підхід, в той же час більш оптимально визначати візуальний інтерейса в файлах xml, а всю пов'язану логіку визначати в класі activity. Тим самим ми досягнемо розмежування інтерфейсу і логіки додатка, їх легше буде розробляти і згодом модифікувати. І в наступній темі ми це розглянемо.

Визначення інтерфейсу у файлі XML. файли layout

У додатках Android візуальний інтерфейс нерідко завантажується зі спеціальних файлів xml, які зберігають розмітку. Ці файли є ресурсами розмітки. Подібний підхід нагадує створення веб-сайтів, коли інтерфейс визначається в файлах html, а логіка програми - в коді javascript.

Оголошення призначеного для користувача інтерфейсу в файлах XML дозволяє відокремити інтерфейс програми від коду. Що означає, що ми можемо змінювати визначення інтерфейсу без зміни коду java. Наприклад, в додатку можуть бути визначені розмітки в файлах XML для різних орієнтацій монітора, різних розмірів пристроїв, різних мов тощо Крім того, оголошення розмітки в XML дозволяє легше візуалізувати структуру інтерфейсу і полегшує налагодження.

Файли розмітки графічного інтерфейсу розташовуються в проекті в каталозі res/layout. За замовчуванням при створенні проекту вже є один файл ресурсів розмітки activity_main.xml , який може виглядати приблизно так:

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        tools:layout_editor_absoluteX="16dp"
        tools:layout_editor_absoluteY="16dp" />
</android.support.constraint.ConstraintLayout>
```

У файлі визначаються всі графічні елементи і їх атрибути, які складають інтерфейс. При створенні розмітки в XML слід дотримуватися деяких правил: кожен файл розмітки повинен містити один кореневий елемент, який повинен представляти об'єкт **View** або **ViewGroup** .

В даному випадку кореневих елементом є елемент ConstraintLayout, який містить елемент TextView.

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

Для отримання посилання на ресурс в коді java необхідно вжити вислів R.layout.[название_ресурса]. Назва ресурсу layout буде збігатися з ім'ям файлу, тому щоб використовувати файл **activity_main.xml** як джерело візуального інтерфейсу, нам треба змінити код **MainActivity** наступним чином:

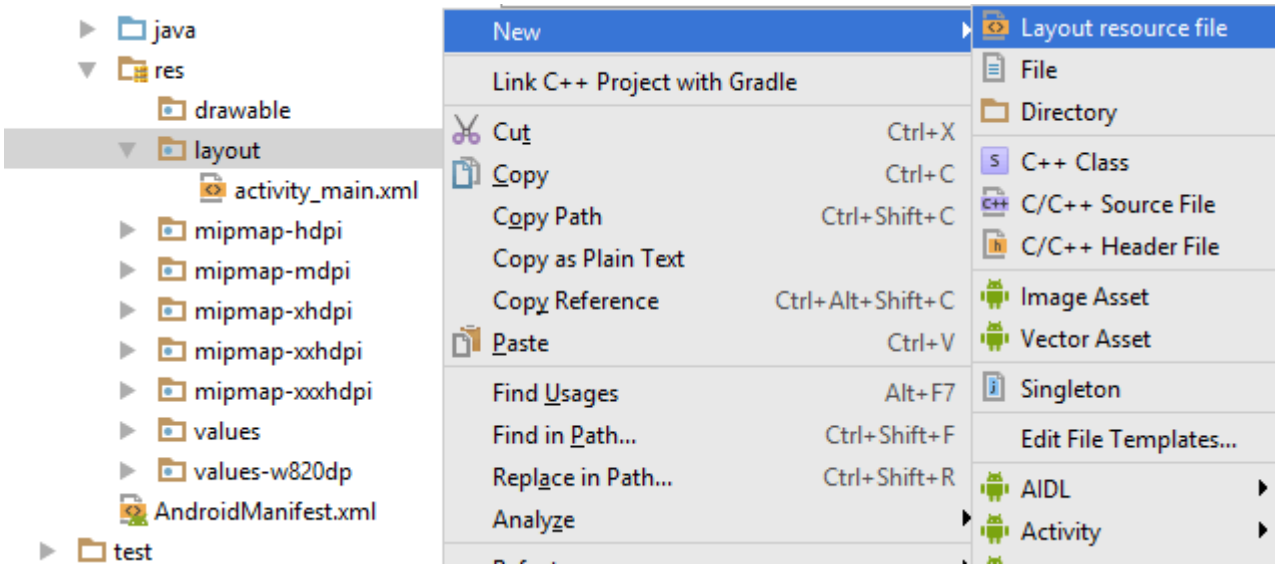
```
package com.example.ozm.helloworld;

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);
    }
}
```

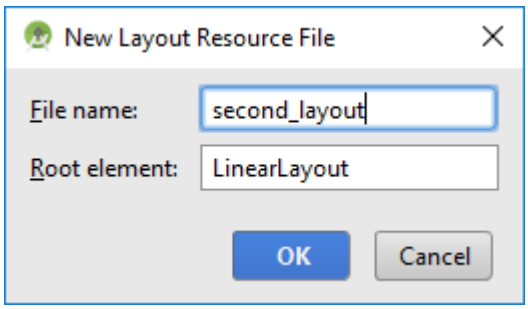
Додавання файлу layout

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

Наприклад, додамо в проект 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 в спеціальному віконці буде запропоновано вказати ім'я і кореневий елемент для файлу layout:



Як назва вкажемо **second_layout** , а в якості типу кореневого елемента залишимо **LinearLayout** .

Після цього в папку *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 і змінимо його вміст в такий спосіб:

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

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

<TextView android:id="@+id/header"
    android:text="Second Activity"
    android:textSize="26dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

Тут визначено текстове поле TextView, яке виводить на екран найпростіший текст "Second Activity".

Отримання елементів в коді і їх ідентифікатори

Крім тексту, ширини і висоти текстове поле встановлює ще один важливий атрибут - id. Знак + в запису android:id="@+id/header" означає, що якщо для елемента не визначений id зі значенням header, то його слід визначити.

Щоб використовувати цей файл в якості основного інтерфейсу, перейдемо до MainActivity і змінимо її код:

```
package com.example.ozm.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

        // встановлюємо в якості інтерфейсу файл second_layout.xml
        setContentView(R.layout.second_layout);

        // отримуємо елемент textView
        TextView textView = (TextView) findViewById(R.id.header);
        // перевстановлюємо у ньому текст
        textView.setText("Hello Android 7!");
    }
}
```

За допомогою методу setContentView()встановлюється розмітка з файлу *second_layout.xml* .

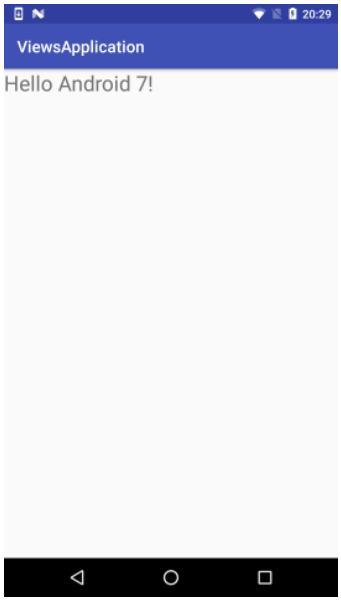
Інший важливий момент, який варто відзначити - отримання візуального елемента TextView. Так як в його коді ми визначили атрибут android:id, то через цей id ми можемо його отримати.

Для отримання елементів з id клас Activity має метод **findViewById ()** . У цей метод передається ідентифікатор ресурсу у вигляді **R.id. [ідентіфікатор_елемента]** . 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, в даному випадку змінюємо його текст.

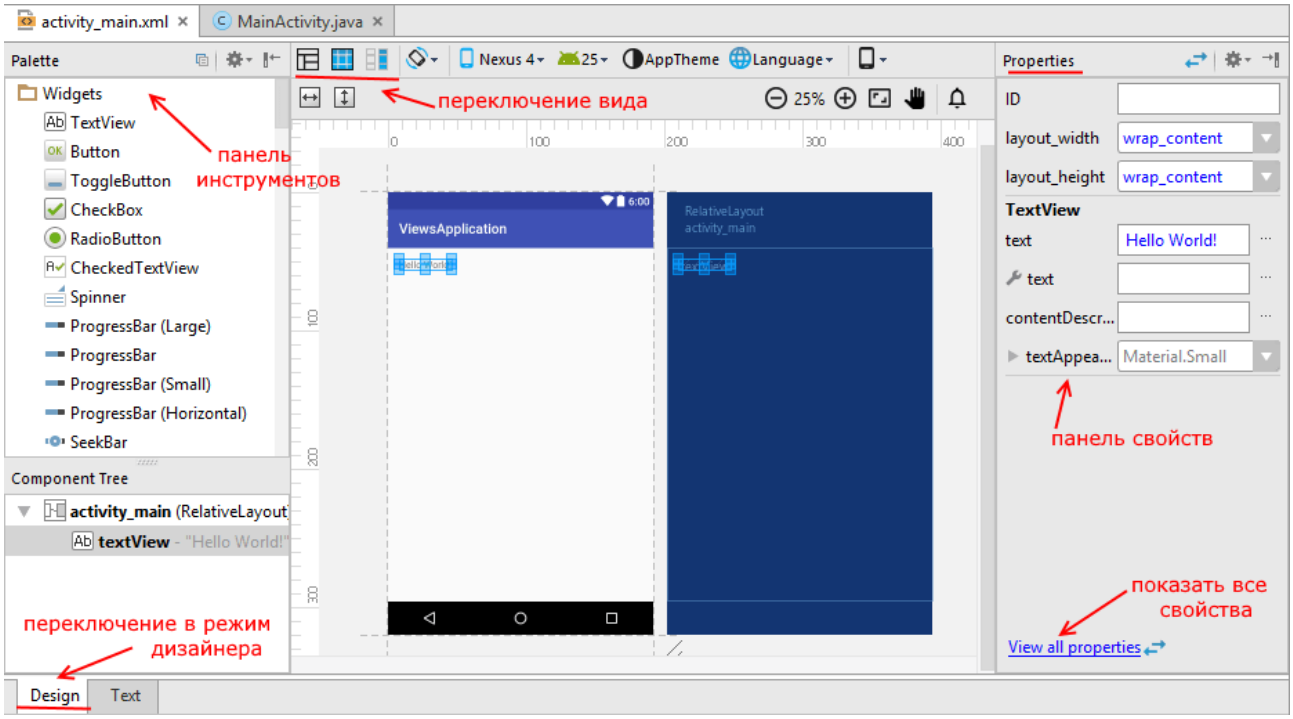
Причому що важливо, отримання елемента відбувається після того, як в методі setContentView була встановлена розмітка, в якій цей візуальний елемент був визначений.

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



Графічні можливості Android Studio

У минулому параграфі ми розглянули, як в коді xml і java визначати інтерфейс. Але треба зазначити, що Android Studio має досить просунутий інструментарій, який полегшує розробку графічного інтерфейсу.



Ми можемо відкрити файл **activity_main.xml** і внизу за допомогою кнопки **Design** переключитися в режим дизайнера до графічного поданням інтерфейсу у вигляді ескізу смартфона.

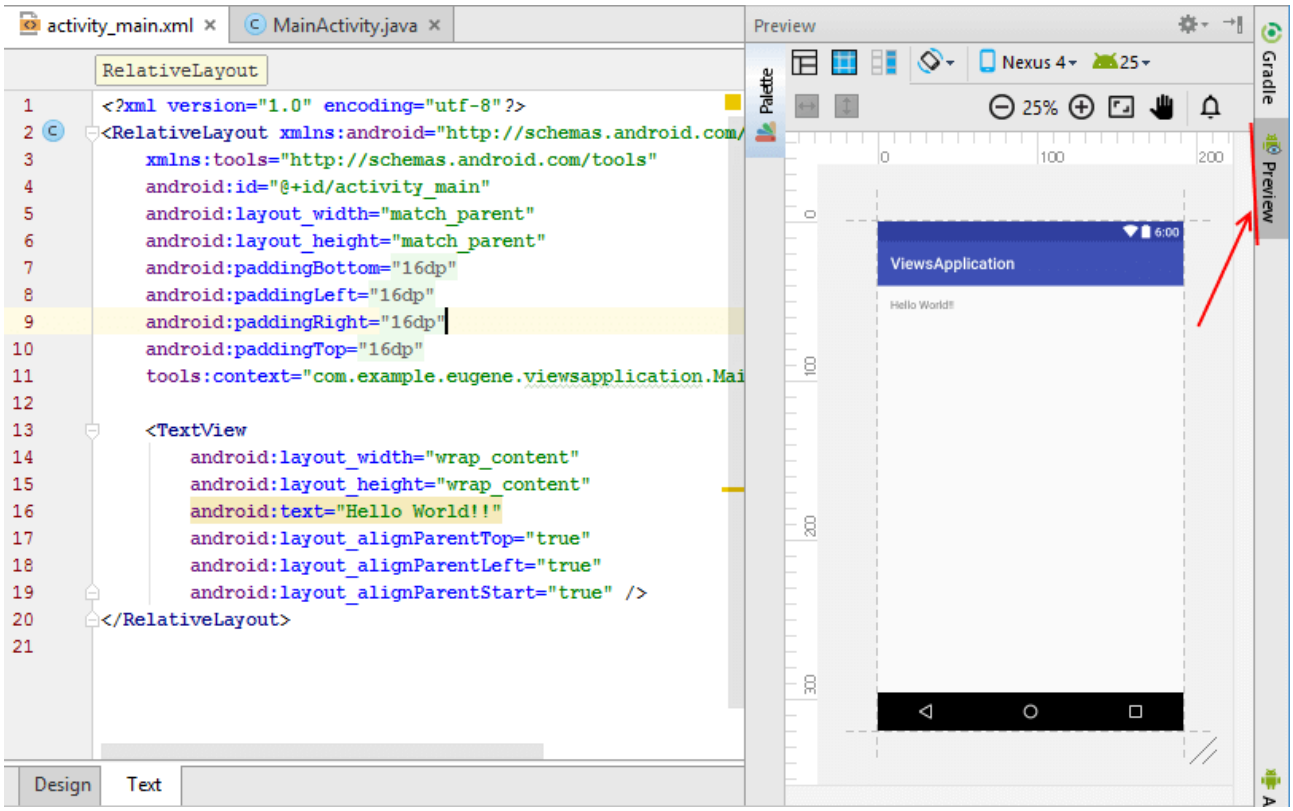
Зліва буде знаходитися панель інструментів, з якою ми можемо переносити потрібний елемент мишкою на ескіз смартфона. І все перенесені елементи будуть автоматично

додаватися в файл activity_main.xml. За допомогою миші ми можемо змінювати позиціонування вже доданих елементів, переносячи їх в інше місце на смартфоні.

Справа буде вікно Properties - панель властивостей виділеного елемента. Тут ми можемо змінити значення властивостей елемента. І знову ж таки після зміни властивостей зміниться і вміст файлу activity_main.xml.

Тобто при будь-яких змінах в режимі дизайнера відбуватиметься синхронізація з файлом activity_main.xml. Все одно, що ми вручну змінювали б код безпосередньо в файлі activity_main.xml.

Але навіть якщо ми вважаємо за краще працювати з розміткою інтерфейсу в текстовому вигляді, то навіть тут ми можемо включити попередній перегляд для файлу activity_main.xml. Для цього після перемикання в текстовий режим необхідно натиснути на вкладку **Preview** справа в Android Studio:



Це дуже зручно, так як відразу дозволяє переглянути, як буде виглядати додаток. А при будь-яких змінах область попереднього перегляду буде автоматично синхронізуватися з вмістом файлу activity_main.xml.

Визначення розмірів

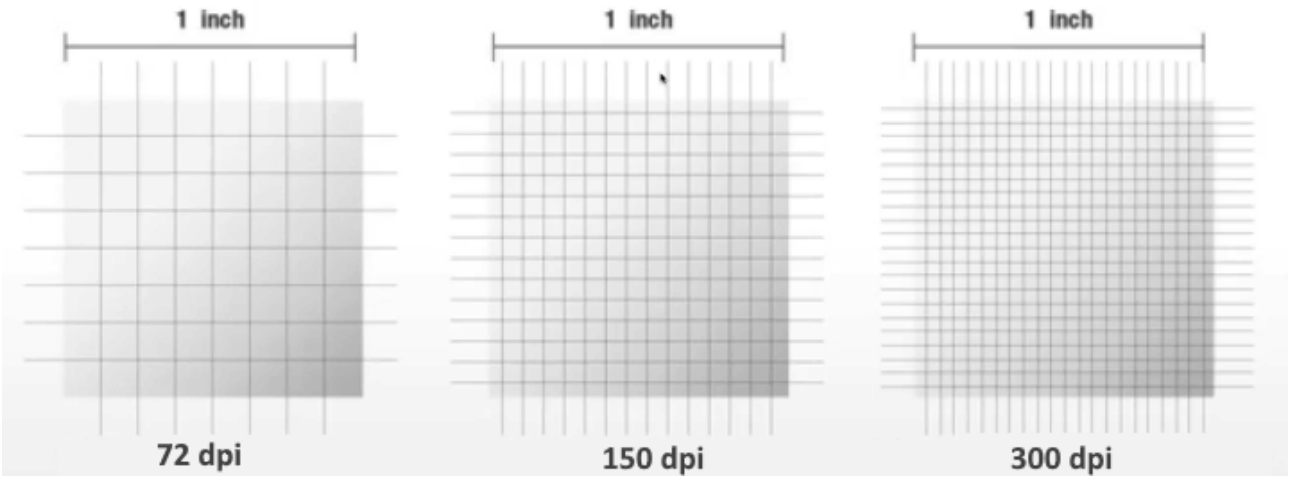
В ОС Android ми можемо використовувати різні типи вимірювань:

- px: Пікселі поточного екрану. Однак ця одиниця виміру не рекомендується, так як реальне уявлення зовнішнього вигляду може змінюватися в залежності від пристрою; кожен пристрій має певний набір пікселів на дюйм, тому кількість пікселів на екрані може також змінюватися
- dp: (Device-independent pixels) незалежні від щільності екрану пікселі. Абстрактна одиниця виміру, заснована на фізичній щільності екрану з роздільною здатністю 160 dpi (точок на дюйм). В цьому випадку 1dp = 1px. Якщо розмір екрану більше або менше, ніж 160dpi, кількість пікселів, які застосовуються для відтворення 1dp відповідно збільшується або зменшується. Наприклад, на екрані з 240 dpi 1dp = 1,5px,

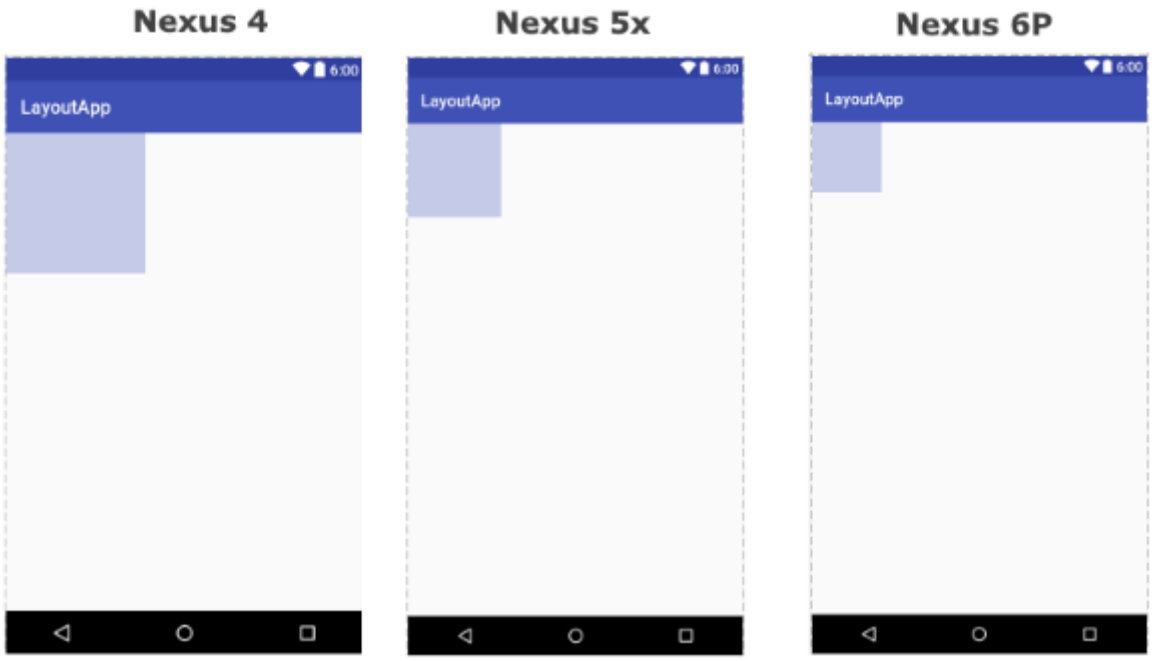
а на екрані з 320dpi 1dp = 2px. Загальна формула для отримання кількості фізичних пікселів з dp: **px = dp * (dpi / 160)**

- sp: (Scale-independent pixels) незалежні від масштабування пікселі. Припускають настройку розмірів, вироблену користувачем. Рекомендуються для роботи зі шрифтами.
- pt: 1/72 дюйма, базуються на фізичних розмірах екрану
- mm: міліметри
- in: дюйми

Кращими одиницями для використання є dp. Це пов'язано з тим, що світ мобільних пристроїв на Android сильно фрагментований в плані дозволу і розмірів екрану. І чим більше щільність пікселів на дюйм, тим відповідно більше пікселів нам буде доступно:

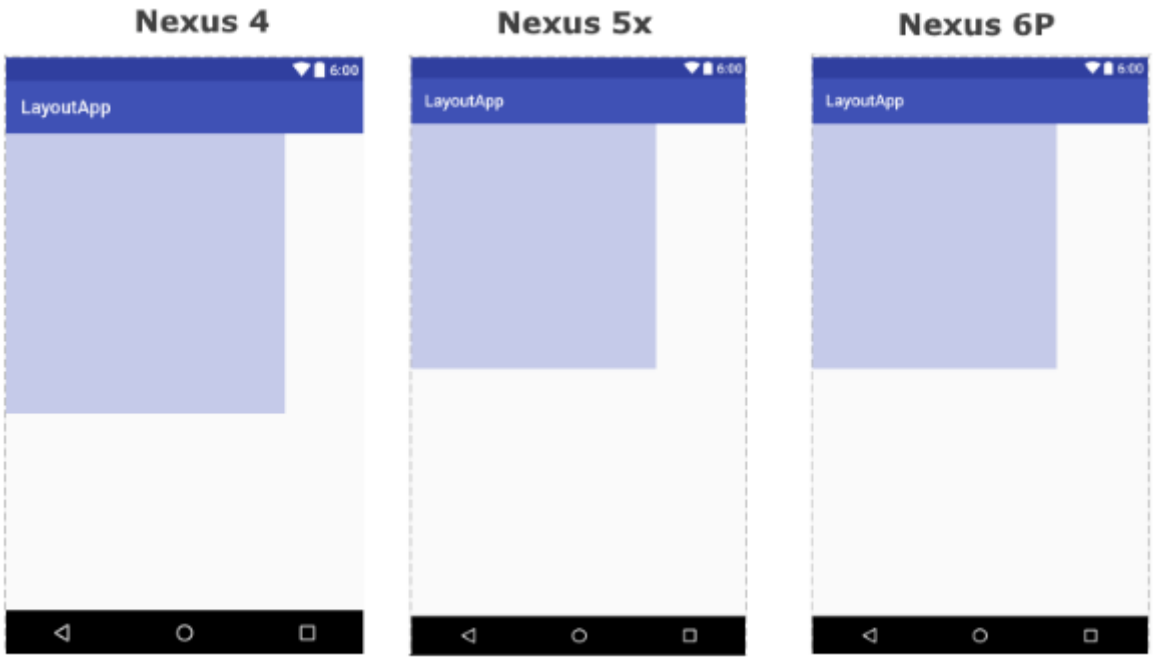


Використовуючи ж стандартні фізичні пікселі ми можемо зіткнутися з проблемою, що розміри елементів також будуть сильно варіюватися в залежності від щільності пікселів пристрою. Наприклад, візьмемо 3 пристрої з різними характеристиками екрану Nexus 4, Nexus 5X і Nexus 6P і виведемо на екран квадрат розміром 300px на 300px:



В одному випадку квадрат по ширині буде займати 40%, в іншому - третина ширини, в третьому - 20%.

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



Тепер же розміри квадрата на різних пристроях виглядають більш Консистентне.

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

- **ldpi (low)** : ~ 120dpi
- **mdpi (medium)** : ~ 160dpi
- **hdpi (high)** : ~ 240dpi (до цієї групи можна віднести такий 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)

Ширина і висота елементів

Всі візуальні елеметов, які ми використовуємо в додатку, як правило, упорядковуються на екрані за допомогою контейнерів. В Android подібними контейнерами служать такі класи як RelativeLayout, LinearLayout, GridLayout, TableLayout, ConstraintLayout, FrameLayout. Всі вони по різному розташовують елементи і керують ними, але є деякі загальні моменти при компонуванні візуальних компонентів, які ми зараз розглянемо.

Для організації елементів всередині контейнера використовуються параметри розмітки. Для їх завдання в файлі xml використовуються атрибути, які починаються з префікса **layout_** . Зокрема, до таких параметрів належать атрибути **layout_height** і **layout_width** , які використовуються для установки розмірів і можуть приймати одне з наступних значень:

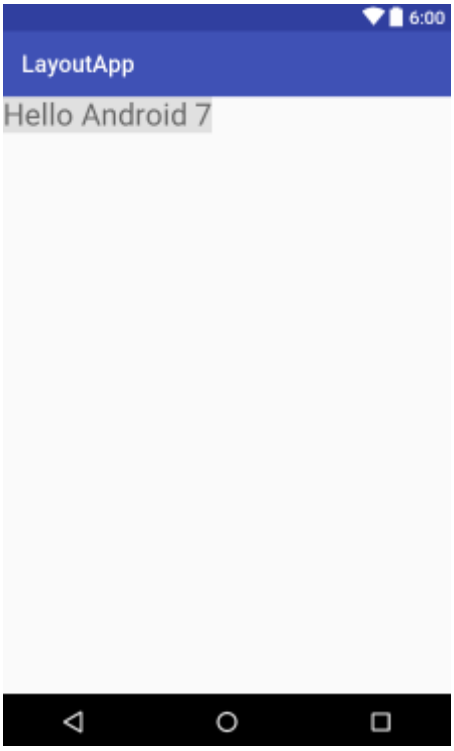
- точні розміри елемента, наприклад 96 dp
- значення **wrap_content** : елемент розтягується до тих меж, які є достатніми, щоб вмістити весь його вміст
- значення **match_parent** : елемент заповнює всю область батьківського контейнера

наприклад:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <TextView
        android:text="Hello Android 7"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:background="#e0e0e0" />
</RelativeLayout>
```

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

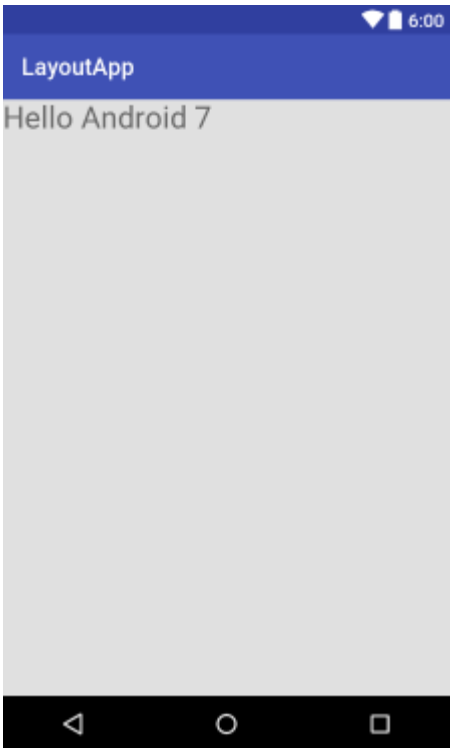
А елемент TextView розтягується до тих значень, які є достатніми для рамещеніе його тексту.



Тепер змінимо висоту і ширину на match_parent:

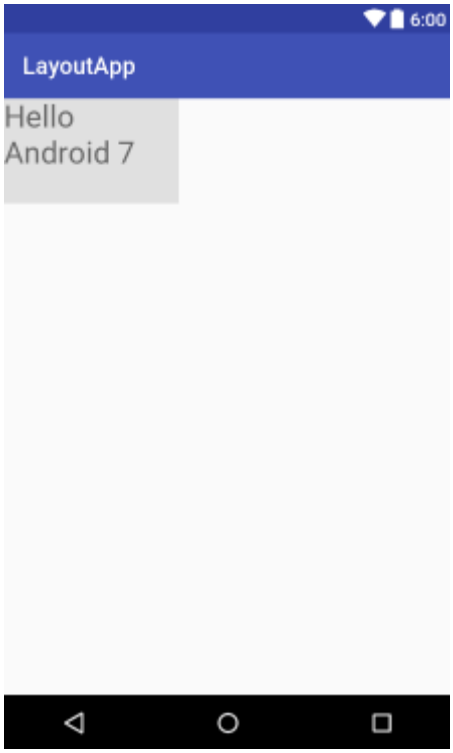
```
<TextView
    android:text="Hello Android 7"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:textSize="26sp"
    android:background="#e0e0e0" />
```

Тепер TextView буде заповнювати весь простір контейнера:



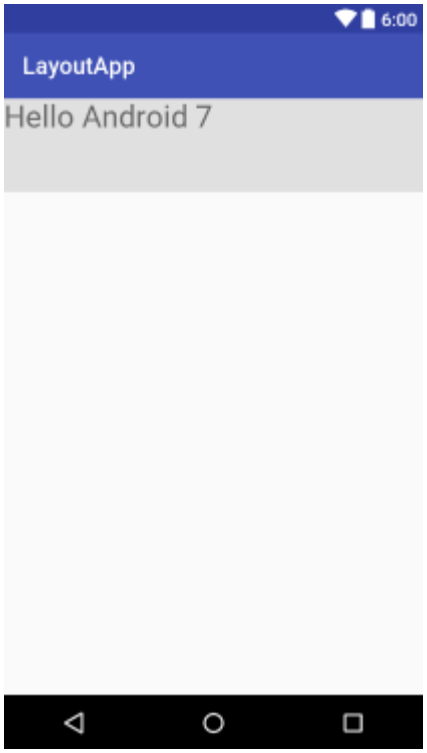
Або ми можемо встановити точні значення:

```
<TextView
    android:text="Hello Android 7"
    android:layout_height="90dp"
    android:layout_width="150dp"
    android:textSize="26sp"
    android:background="#e0e0e0" />
```



Також ми можемо комбінувати кілька значень, наприклад, розтягнути по ширині і встановити точні значення для висоти:

```
<TextView
    android:text="Hello Android 7"
    android:layout_height="80dp"
    android:layout_width="match_parent"
    android:textSize="26sp"
    android:background="#e0e0e0" />
```



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

```
<TextView android:text="Hello Android 7"
    android:minWidth="200dp"
    android:maxWidth="250dp"
    android:minHeight="100dp"
    android:maxHeight="200dp"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:background="#e0e0e0" />
```

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

Програмне встановлення ширини і висоти

Якщо елемент, наприклад, той же TextView створюється в коді java, то для установки висоти і ширини можна використовувати метод **setLayoutParams ()** . Так, змінимо код MainActivity:

```
package com.example.ozm.helloworld;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

        RelativeLayout relativeLayout = new RelativeLayout(this);
        TextView textView1 = new TextView(this);
        textView1.setText("Hello Android 7");
        textView1.setTextSize(26);

        // встановлюємо розміри
        textView1.setLayoutParams(new ViewGroup.LayoutParams
                (ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
        // додаємо TextView в RelativeLayout
        relativeLayout.addView(textView1);
        setContentView(relativeLayout);
    }
}
```

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

Також ми можемо передати точні значення або комбінувати типи значень:

Внутрішні і зовнішні відступи

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

Padding

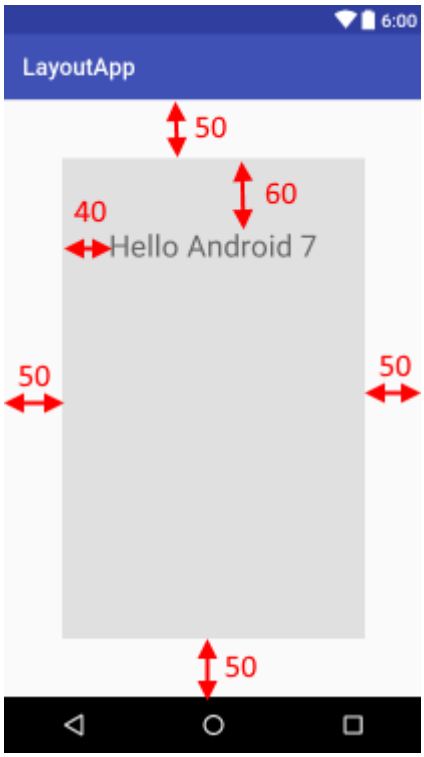
Для установки внутрішніх відступів застосовується атрибут android:padding. Він встановлює відступи контенту від усіх чотирьох сторін контейнера. Можна встановлювати відступи тільки від однієї сторони контейнера, застосовуючи такі атрибути: android:paddingLeft, android:paddingRight, android:paddingTopiandroid:paddingBottom.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:padding="50dp">

    <TextView android:text="Hello Android 7"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:textSize="26sp"
```

```
        android:background="#e0e0e0"
        android:paddingTop="60dp"
        android:paddingLeft="40dp"/>

</RelativeLayout>
```

У контейнера RelativeLayout встановлений тільки один загальний внутрішній відступ в 50 одиниць.

У TextView встановлений відступ між верхньою межею елемента і його внутрішнім вмістом (тобто текстом) в 60 одиниць і відступ зліва в 40 одиниць:



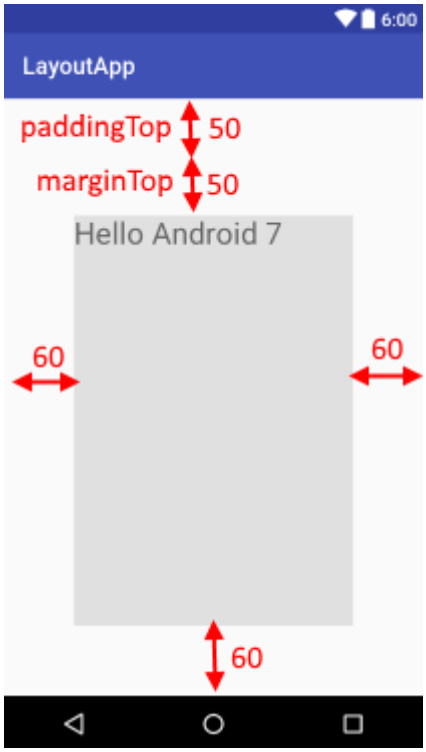
Margin

Для установки зовнішніх остступов використовується атрибут layout_margin. Даний атрибут має модифікації, які дозволяють задати відступ тільки від однієї сторони: android:layout_marginBottom, android:layout_marginTop, android:layout_marginLefti android:layout_marginRight(відступи відповідно від нижньої, верхньої, лівої і правої меж):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:paddingTop="50dp">

    <TextView android:text="Hello Android 7"
        android:layout_height="match_parent"
        android:layout_width="match_parent"

        android:layout_marginTop="50dp"
        android:layout_marginBottom="60dp"
        android:layout_marginLeft="60dp"
```

```
            android:layout_marginRight="60dp"

            android:textSize="26sp"
            android:background="#e0e0e0"/>

    </RelativeLayout>
```

Тут у TextView задаються відступи від кожної сторони RelativeLayout. І варто зазначити, що у самого RelativeLayout встановлений внутрішній відступ зверху в 50 одиниць, який також враховується при формуванні зовнішнього відступу зверху у TextView, тому загальна відстань від верхньої межі TextView до верхнього краю RelativeLayout становитиме 100 одиниць:



Програмне встановлення відступів

Для програмної установки внутрішніх відступів у елементи визивается метод **setPadding (left, top, right, bottom)** , в який передаються чотири значення для кожної зі сторін.

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

```
package com.example.ozm.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

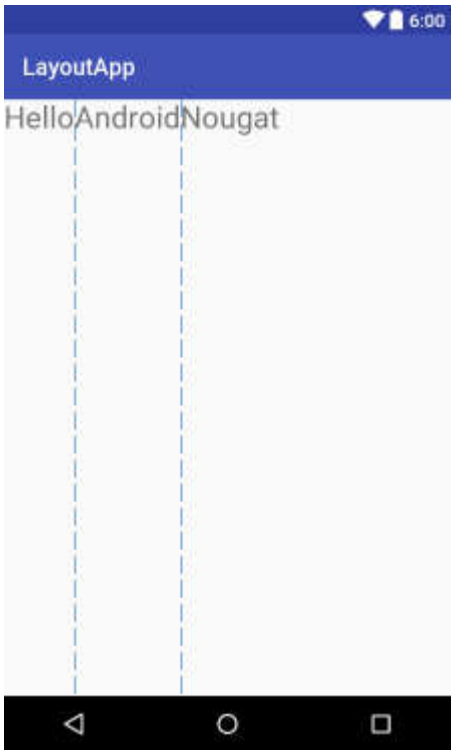
```
            RelativeLayout relativeLayout = new RelativeLayout(this);
            TextView textView1 = new TextView(this);
            textView1.setBackgroundColor(0xFFBDBDBD);
            textView1.setText("Hello Android 7");
            textView1.setTextSize(26);

            RelativeLayout.LayoutParams layoutParams = new
RelativeLayout.LayoutParams
                    (ViewGroup.LayoutParams.MATCH_PARENT, 200);
            // встановлення зовнішніх відступів
            layoutParams.setMargins(30,40,50,60);
            // встановлюємо розміри
            textView1.setLayoutParams(layoutParams);
            // встановлення внутрішніх відступів
            textView1.setPadding(30,30,30,30);
            // додаємо TextView в RelativeLayout
            relativeLayout.addView(textView1);
            setContentView(relativeLayout);
        }
    }
```

LinearLayout

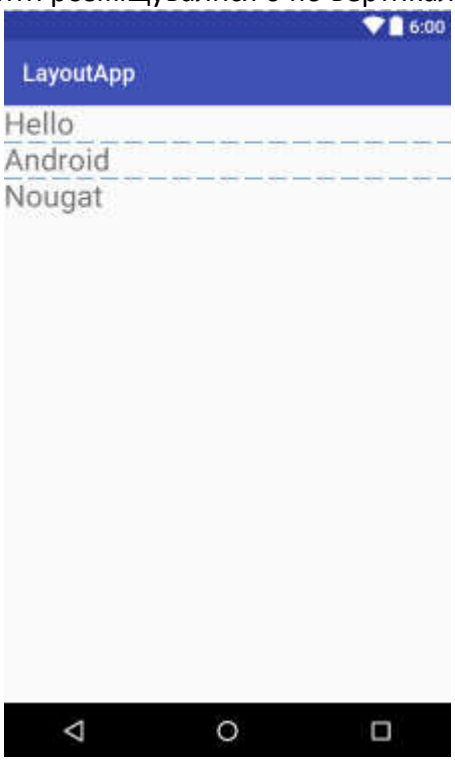
Контейнер **LinearLayout** представляє об'єкт ViewGroup, який впорядковує всі дочірні елементи в одному напрямку: по горизонталі або по вертикалі. Все Елем розташовані один за іншим. Напрямок розмітки вказується за допомогою атрибута **android:orientation** .

Якщо, наприклад, орієнтація розмітки вертикальна (android:orientation="vertical"), то всі елементи розташовуються в стовпчик - по одному елементу на кожному рядку. Якщо орієнтація горизонтальна (android:orientation="horizontal"), то елементи розташовуються в один рядок. Наприклад, розташуємо елементи в 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:text = "Hello"
        android:textSize = "26sp" />
    <TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Android"
        android:textSize = "26sp" />
    <TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Nougat"
        android:textSize = "26sp" />
</LinearLayout>
```



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

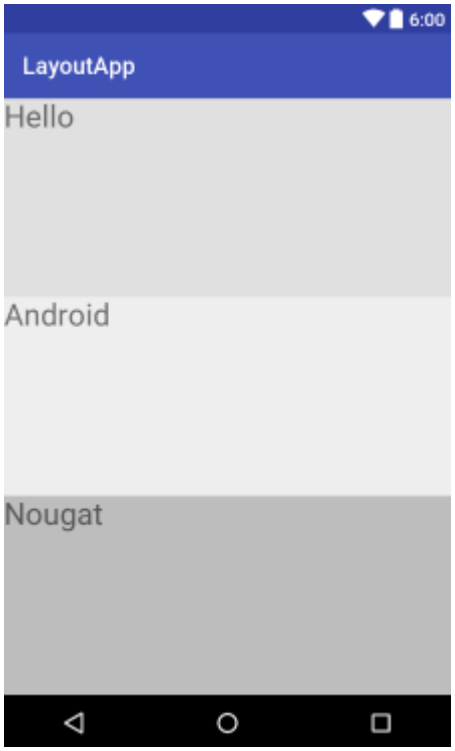


LinearLayout підтримує таку властивість, як вага елемента, яке передається атрибутом **android:layout_weight** . Це властивість приймає значення, яке вказує, яку частину залишку вільного місця контейнера по відношенню до інших об'єктів займе даний елемент. Наприклад, якщо один елемент у нас буде мати для властивості android:layout_weightзначення 2, а інший - значення 1, то в сумі вони дадуть 3, тому перший елемент буде займати 2/3 простору, що залишилося, а другий - 1/3.

Якщо всі елементи мають значення android:layout_weight="1", то 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="Nougat"
        android:background="#bdbdbd"
        android:layout_weight="1"
        android:textSize="26sp" />
</LinearLayout>
```

В даному випадку LinearLayout має вертикальну орієнтацію, тому всі елементи будуть розташовуватися зверху вниз. Всі три елементи мають значення android:layout_weight="1", тому сума ваг всіх елементів буде дорівнює 3, а кожен елемент отримає по третині простору в LinearLayout:



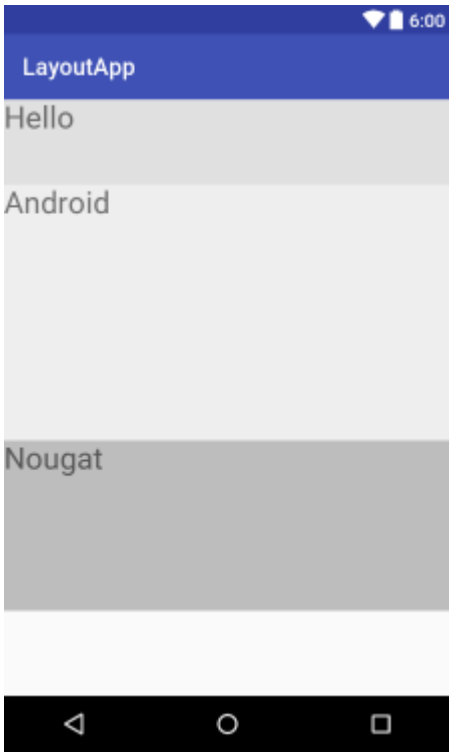
При цьому так як у нас вертикальний стек, то нам треба також встановити для властивості layout_heightзначення **0dp** . Якби LinearLayout мав горизонтальну орієнтацію, то для властивості layout_widthтреба було б встановити значення **0dp** .

Ще один атрибут **android:weightSum** дозволяє вказати суму ваг усіх 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="Nougat"
        android:background="#bdbdbd"
        android:layout_weight="2"
        android:textSize="26sp" />
</LinearLayout>
```

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

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



Програмне створення LinearLayout

Створення LinearLayout в коді MainActivity:.

```
package com.example.ozm.helloworld;

import android.support.v7.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);
        TextView textView1 = new TextView (this);
        textView1.setText ( "Hello");
        linearLayout.addView (textView1, new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT));

        TextView textView2 = new TextView (this);
        textView2.setText ( "Android");
        linearLayout.addView (textView2, new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT));

        setContentView (linearLayout);
    }
}
```

RelativeLayout

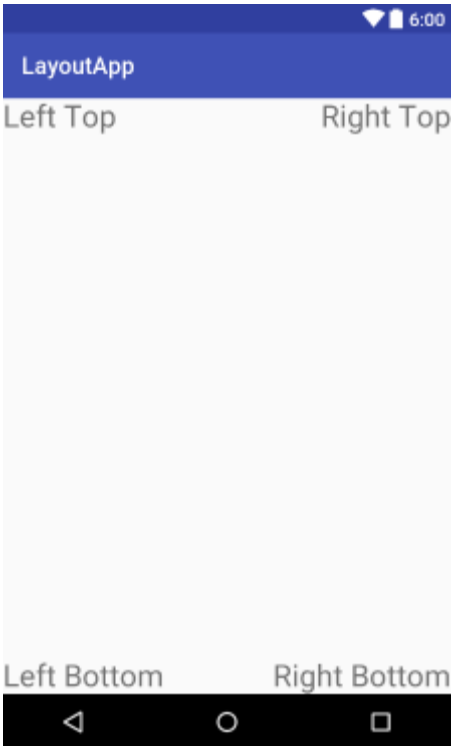
RelativeLayout представляє об'єкт ViewGroup, який має в своєму розпорядженні дочірні елементи щодо позиції інших дочірніх елементів розмітки або щодо області самої розмітки RelativeLayout. Використовуючи відносне позиціонування, ми можемо встановити елемент по правому краю або в центрі або іншим способом, який надає даний контейнер. Для установки елемента в файлі xml ми можемо застосовувати 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 зліва від елемента з вказаним Id
- **android:layout_toRightOf** : розташовується праворуч від елемента з вказаним 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_alignTop** : вирівнює елемент по верхній межі іншого елемента із зазначеним Id
- **android:layout_alignBaseline** : вирівнює базову лінію елемента за базовою лінії іншого елемента із зазначеним Id
- **android:layout_alignParentBottom** : якщо атрибут має значення true, то елемент притискається до нижньої межі контейнера
- **android:layout_alignParentRight** : якщо атрибут має значення true, то елемент притискається до правого краю контейнера
- **android:layout_alignParentLeft** : якщо атрибут має значення true, то елемент притискається до лівого краю контейнера
- **android:layout_alignParentTop** : якщо атрибут має значення true, то елемент притискається до верхньої межі контейнера
- **android:layout_centerInParent** : якщо атрибут має значення true, то елемент розташовується по центру батьківського контейнера
- **android:layout_centerHorizontal** : при значенні true вирівнює елемент по центру по горизонталі
- **android:layout_centerVertical** : при значенні true вирівнює елемент по центру по вертикалі

Наприклад, позиціонування відносно контейнера RelativeLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    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 і кнопку:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    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="Отправить"
```

```
            android:layout_alignRight="@id/edit_message"
            android:layout_below="@id/edit_message"
            />
    </RelativeLayout>
```

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

Створимо елемент RelativeLayout програмно в коді MainActivity:

```
package com.example.ozm.helloworld;

import android.support.v7.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(1);

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

        // встановлюємо параметри додатку для EditText
        RelativeLayout.LayoutParams editTextParams = new
RelativeLayout.LayoutParams(
                RelativeLayout.LayoutParams.MATCH_PARENT,
                RelativeLayout.LayoutParams.WRAP_CONTENT
        );
        editTextParams.addRule(RelativeLayout.CENTER_IN_PARENT);
```

```
            relativeLayout.addView(editText, editTextParams);

            // встановлюємо параметри додатку для Button
            RelativeLayout.LayoutParams buttonParams = new
RelativeLayout.LayoutParams(
                    RelativeLayout.LayoutParams.WRAP_CONTENT,
                    RelativeLayout.LayoutParams.WRAP_CONTENT
            );
            buttonParams.addRule(RelativeLayout.BELOW, editText.getId());
            buttonParams.addRule(RelativeLayout.ALIGN_RIGHT,
editText.getId());
            relativeLayout.addView(button, buttonParams);

            setContentView(relativeLayout);
        }
    }
```

Щоб задати положення елемента в контейнері, застосовується клас **RelativeLayout.LayoutParams** . Через конструктор встановлюються значення для для ширини і висоти. Наприклад, у елемента EditText для ширини встановлюється значення MATCH_PARENT, а для висоти - WRAP_CONTENT.

За допомогою методу **addRule()** ми можемо додавати додаткові правила для позиціонування елемента. Цей метод в якості параметра 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_alignRightконстанта ALIGN_RIGHT.

Варто зазначити, що з метою спрощення коду для установки id у EditText викликається метод **setId** , в який передається просте число. І Android Studio може підкреслювати даний метод як помилки, але за фактом тут помилки немає, проте в реальності, для визначення всіх ідентифікаторів, як правило, використовуються окремі файли ресурсів.

Потім встановлений id передається в якості другого параметра в метод addRule при установці правил для кнопки:

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

Тим самим ми вказуємо щодо якого елемента треба задати розташування.

Gravity і layout_gravity

Для управління позиціонуванням елементу при визначенні інтерфейсу ми можемо використовувати такі атрибути як **gravity** та **layout_gravity**.

Gravity

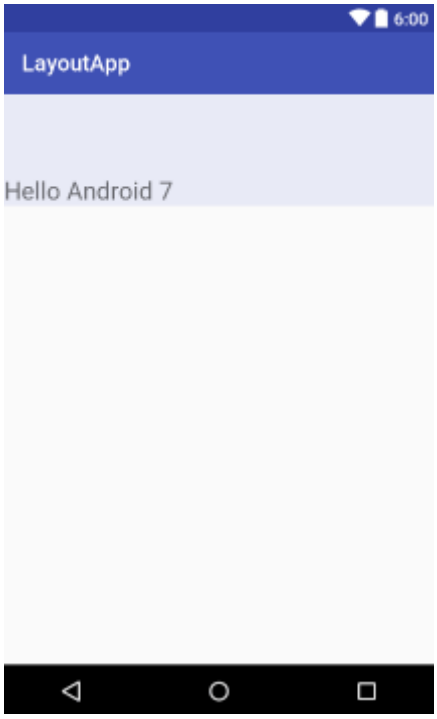
Атрибут **gravity** задає позиціонування вмісту всередині об'єкта. Він може приймати наступні значення:

- top: Елементи розміщуються вгорі
- bottom: Елементи розміщуються внизу
- left: Елементи розміщуються в лівій стороні
- right: Елементи розміщуються в 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 і ліву кордон елементів
- start: Елемент позиціонується на початку (у верхньому лівому куті) контейнера
- end: Елемент позиціонується в кінці контейнера (у верхньому правому куті)

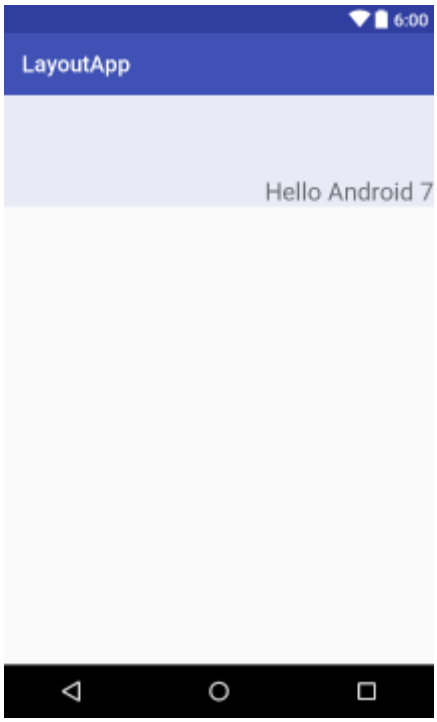
Наприклад, помістимо текст в самий низ в елементі TextView:

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

    <TextView
        android:gravity="bottom"

        android:layout_width="match_parent"
        android:layout_height="200px"
        android:text="Hello Android 7"
        android:background="#e8eaf6"/>
</RelativeLayout>
```



При необхідності ми можемо комбінувати значення, розділяючи їх вертикальною лінією:

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

    android:layout_width="match_parent"
    android:layout_height="200px"
    android:text="Hello Android 7"
  android:background="#e8eaf6"/>
```



Layout_gravity

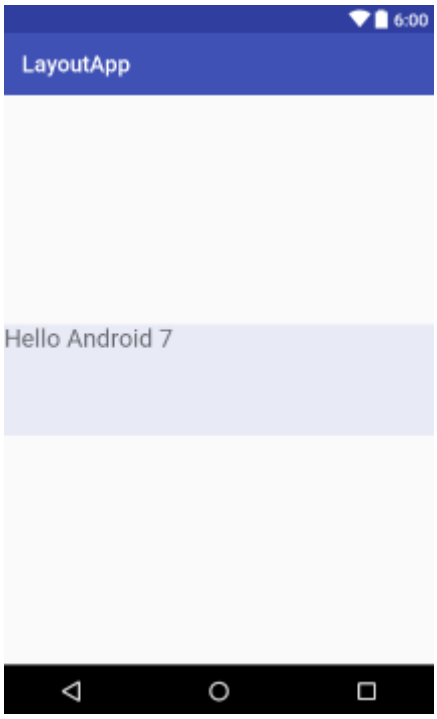
На відміну від gravity атрибут **layout_gravity** встановлює позиціонування в контейнері. Він приймає ті ж значення, тільки позиціонування йде щодо зовнішнього контейнера:

- top: Вирівнює елемент по верхній межі контейнера
- bottom: Вирівнює елемент по нижній межі контейнера
- left: Вирівнює елемент по лівій межі контейнера
- right: Вирівнює елемент по правій межі контейнера
- center_vertical: Вирівнює елемент по центру по вертикалі
- center_horizontal: Вирівнює елемент по центру по горизонталі
- center: елемент ппозиціонується в центрі
- fill_vertical: елемент розтягується по вертикалі
- fill_horizontal: елемент розтягується по горизонталі
- fill: елемент заповнює весь простір контейнера
- clip_vertical: обрізає верхню і нижню границю елементу
- clip_horizontal: обрізає праву і ліву границю елементу
- start: елемент позиціонирється на початку (в верхньому лівому куті) контейнера
- end: елемент позиціонирється в кінці контейнера (в 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="horizontal">

    <TextView
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="200px"
        android:textSize="22sp"
        android:text="Hello Android 7"
```

```
        android:background="#e8eaf6"/>
</LinearLayout>
```



При цьому треба враховувати, що layout_gravity застосовується тільки до класу LinearLayout або до його класами-спадкоємцям, наприклад, FrameLayout. У RelativeLayout цей атрибут не матиме ніякого впливу.

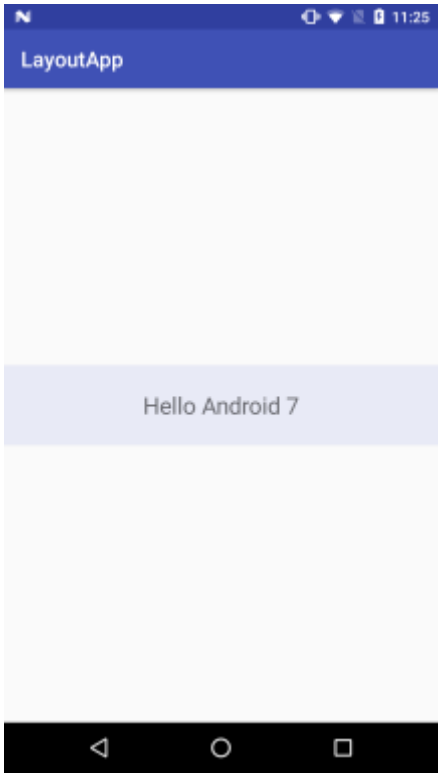
Також всередині LinearLayout варто враховувати орієнтацію контейнера. Наприклад, при вертикальній орієнтації все елементи будуть представляти вертикальний стек, що йде зверху вниз. Тому значення, які належать до позиціонування елемента по вертикалі (наприклад, top або bottom) ніяк не впливатимуть на елемент. Також при горізонтальнйо орієнтації LinearLayout не зроблять ніякого впливу значення, які позиціонують елемент по горизонталі, наприклад, left і right.

Gravity і layout_gravity

Щоб встановити параметр gravity у елемента треба викликати метод **setGravity ()** . Для установки програмно параметра layout_gravity треба задати поле **gravity** у об'єкта **LinearLayout.LayoutParams**:

```
package com.example.ozm.helloworld;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.Gravity;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

        LinearLayout linearLayout = new LinearLayout(this);
        TextView textView = new TextView(this);
```

```
            textView.setText("Hello Android 7");
            textView.setTextSize(22);
            textView.setBackgroundColor(0xffe8eaf6);

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

            // встановлення висоти і ширини
            LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
                    (LinearLayout.LayoutParams.MATCH_PARENT, 200);
            // встановлення layout gravity
            layoutParams.gravity = Gravity.CENTER;
            textView.setLayoutParams(layoutParams);

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

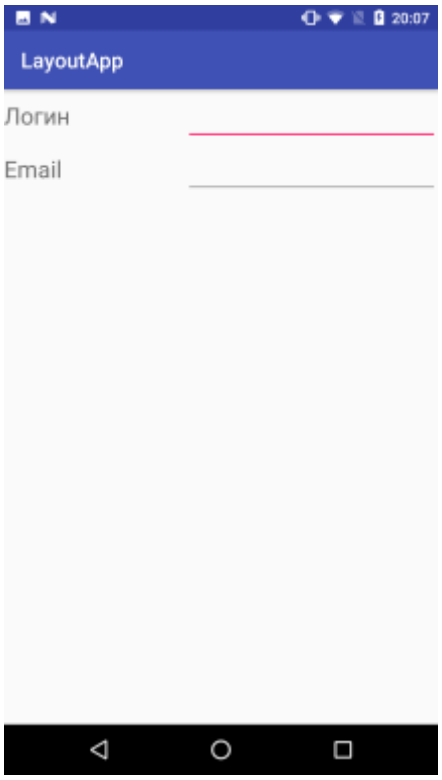


TableLayout

Контейнер **TableLayout** структурує елементи управління по стовпцях і рядках. Визначимо в файлі **activity_main.xml** елемент TableLayout, який буде включати два рядки і 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 буде означати кількість стовпців. Наприклад, в даному випадку у нас визначені два рядки і в кожній по два елементи. Якби в якій-небудь з них було б три віджета, то відповідно стовпців було б також три, навіть якщо в іншому рядку залишилося б два віджету.

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

Якщо якийсь елемент повинен 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 = "Відправити"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:layout_span = "2" />
    </ TableRow>
</ TableLayout>
```

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

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

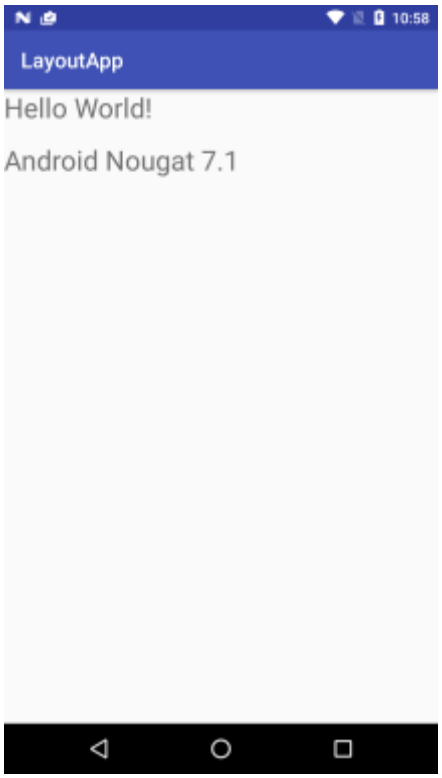
FrameLayout

Контейнер FrameLayout призначений для виведення на екран одного розміщеного в нього візуального елемента. Якщо ж ми розмістимо кілька елементів, то вони будуть накладитися на друге.

Допустимо, вкладемо в FrameLayout два елементи TextView:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    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="Android Nougat 7.1"
        android:textSize="26sp"
        android:layout_marginTop="50dp"/>

</FrameLayout>
```

Тут обидва елементи позиціонуются в одне і те же місце - в лівому верхньому куті контейнера FrameLayout, і для того, щоб уникнути накладання, в цьому випадку в другому TextView встановлюється відступ зверху в 50 одиниць.

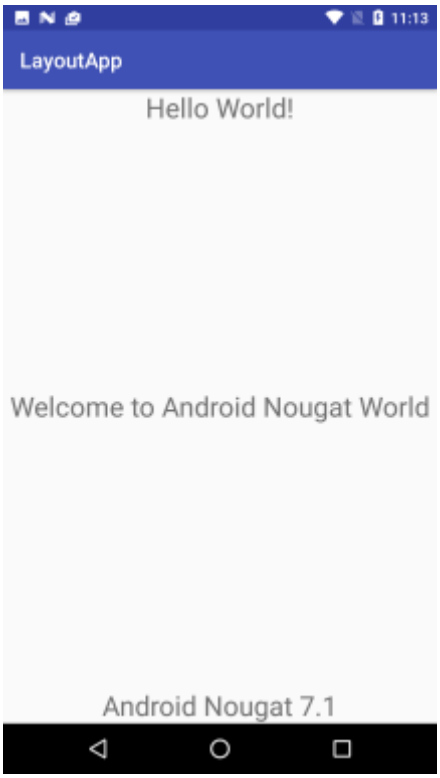


Нерідко FrameLayout застосовується для створення вихідних контейнерів, наприклад ScrollView, який забезпечує прокрутку.

Елементи управління, які розміщуються в 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:id="@+id/activity_main"
    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 Android Nougat World"
        android:textSize="26sp"
        android:layout_gravity="center"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Android Nougat 7.1"
        android:textSize="26sp"
        android:layout_gravity="bottom|center_horizontal"/>

</FrameLayout>
```

При вказанні значення ми можемо комбінувати ряд значень, розділяючи їх вертикальною чертою: **bottom|center_horizontal.**



Программное создание FrameLayout в коде MainActivity:

```
import android.support.v7.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 textView1 = new TextView(this);
        textView1.setText("Hello World!");

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

        textView1.setLayoutParams(layoutParams);
        textView1.setTextSize(26);
        frameLayout.addView(textView1);
        setContentView(frameLayout);
    }
}
```

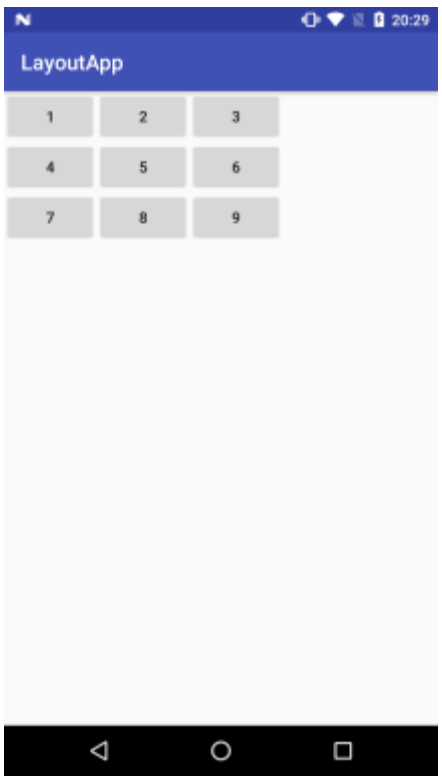
GridLayout

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

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

    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="7" />

    <Button android:text="8" />

    <Button android:text="9" />
</GridLayout>
```

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

При цьому ширина стовпців встановлюється автоматично по ширині самого широкого елемента.

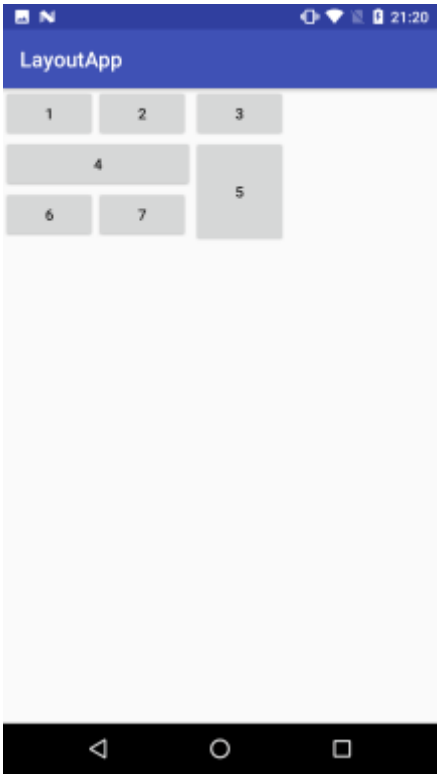


Однак ми можемо явно задати номер стовпця і рядка для певного елемента, а при необхідності розтягнути на декілька стовпців або рядків. Для цього ми можемо застосовувати такі атрибути:

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

наприклад:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:rowCount="3"
    android:columnCount="3">

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

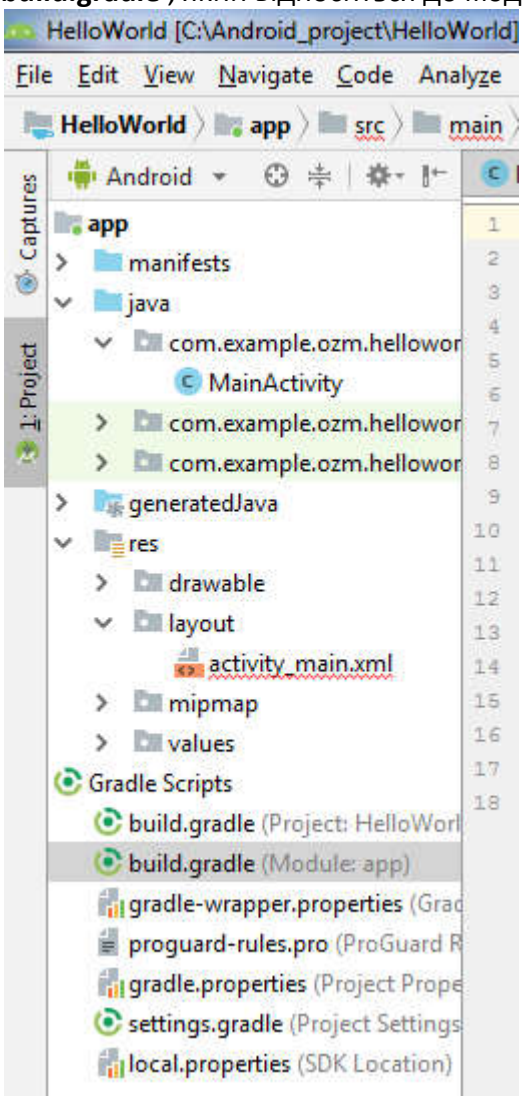


ConstraintLayout

ConstraintLayout представляє новий тип контейнерів, який є розвитком RelativeLayout і дозволяє створювати гнучкі і масштабовані інтерфейси.

Починаючи з версії Android Studio 2.3 ConstraintLayout був доданий в список стандартних компонентів і навіть є контейнером, який використовується в файлах layout за замовчуванням. Однак якщо ви використовуєте Android Studio 2.2 або нижче, то в цьому випадку функціональність ConstraintLayout треба додатково додавати.

Розглянемо додавання ConstraintLayout для Android Studio до версії 2.3 (якщо у вас версія 2.3 і вище, то нічого додавати не треба). Для цього перейдемо в проекті до файлу **build.gradle** , який відноситься до модулю проекту, а не до всього проекту:



У цьому файлі є вузол Dependencies приблизно такого змісту:

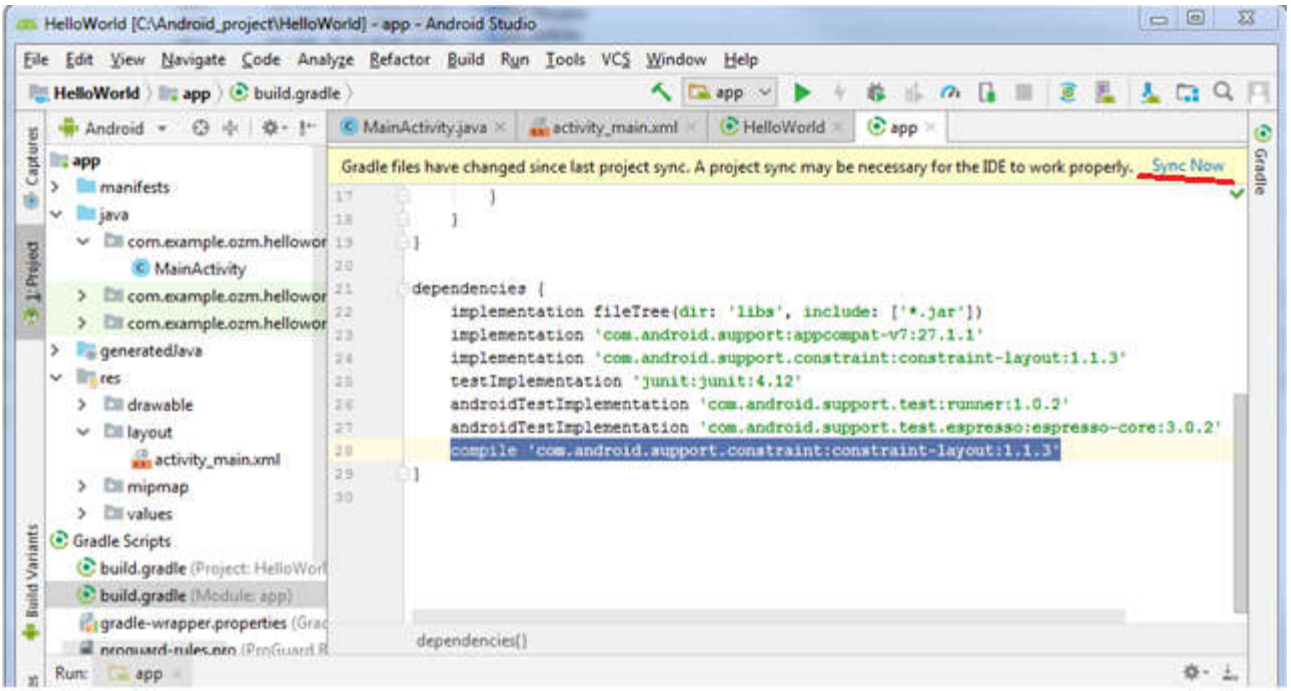
```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

Додамо в кінець цього вузла рядок:

```
dependencies {

// Решта вміст
   compile 'com.android.support.constraint:constraint-layout:1.1.3'}
```

В даному випадку ми додаємо пакет версії "1.1.3", але природно версія може відрізнятися.

Після збереження файлу в Android Studio відобразиться повідомлення з посиланням "Sync Now". Натиснемо на це посилання для синхронізації проекту з файлом build.gradle:



Для позиціонування елемента всередині ConstraintLayout необхідно вказати обмеження (constraints). Є кілька типів обмежень. Зокрема, для установки позиції щодо певного елемента іспльзуются 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** : права межа позиціонується щодо правої межі іншого елемента
- **layout_constraintTop_toTopOf** : верхня межа позиціонується щодо верхньої межі іншого елемента
- **layout_constraintBottom_toBottomOf** : нижня межа позиціонується щодо нижньої межі іншого елемента
- **layout_constraintBaseline_toBaselineOf** : базова лінія позиціонується щодо базової лінії іншого елемента

- **layout_constraintTop_toBottomOf** : верхня межа позиціонується щодо нижньої межі іншого елемента
- **layout_constraintBottom_toTopOf** : нижня межа позиціонується щодо верхньої межі іншого елемента
- **layout_constraintStart_toEndOf** : аналогlayout_constraintLeft_toRightOf
- **layout_constraintStart_toStartOf** : аналогlayout_constraintLeft_toLeftOf
- **layout_constraintEnd_toStartOf** : аналогlayout_constraintRight_toLeftOf
- **layout_constraintEnd_toEndOf** : аналогlayout_constraintRight_toRightOf

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

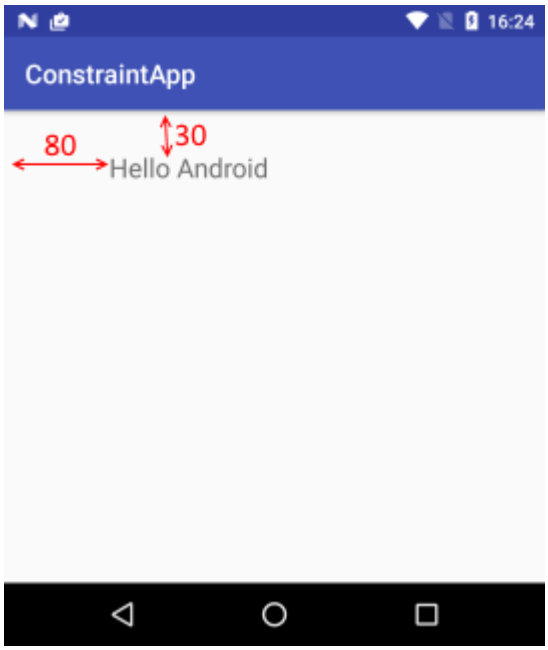
Щоб вказати відступи від елемента, щодо якого проводиться позиціонування, застосовуються такі атрибути:

- **android:layout_marginLeft** : відступ від лівої межі
- **android:layout_marginRight** : відступ від правої межі
- **android:layout_marginTop** : відступ від верхньої межі
- **android:layout_marginBottom** : відступ від нижньої межі
- **android:layout_marginStart** : відступ від лівої межі
- **android:layout_marginEnd** : відступ від правої межі

Розглянемо на простому прикладі. Змінимо файл розмітки інтерфейсу **activity_main**:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android"

        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="30dp"

        android:layout_marginLeft="80dp"
        app:layout_constraintLeft_toLeftOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Тут встановлюються два обмеження щодо батьківського контейнера ConstraintLayout, тому обмеження мають значення **parent**. Тому все відступи, які визначені у елемента TextView, встановлюються щодо верхнього і лівого краю контейнера:

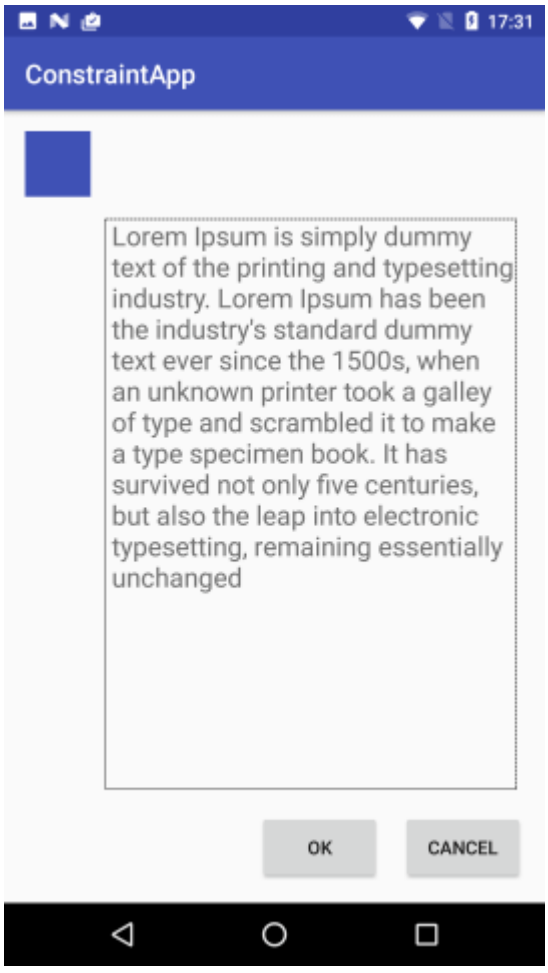


Причому самі по собі відступи нічого не дадуть, нам обов'язково треба встановити обмеження, яке і буде вказувати, щодо якого елемента йде відступ.

Розглянемо більш складний приклад:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:text="TextView"
        android:background="#3F51B5"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:id="@+id/imageView"

        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginLeft="16dp"

        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="16dp" />

    <TextView
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:textSize="20sp"
        android:id="@+id/textView"
        android:text="Lorem Ipsum is simply dummy text of the printing and
typesetting industry ... remaining essentially unchanged"

        app:layout_constraintLeft_toRightOf="@+id/imageView"
        android:layout_marginLeft="16dp"

        app:layout_constraintTop_toBottomOf="@+id/imageView"
        android:layout_marginTop="16dp"

        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"
```

```
            app:layout_constraintBottom_toTopOf="@+id/button2"
            android:layout_marginBottom="16dp" />

        <Button
            android:text="Cancel"
            android:layout_width="93dp"
            android:layout_height="53dp"
            android:id="@+id/button1"

            app:layout_constraintRight_toRightOf="parent"
            android:layout_marginRight="16dp"

            app:layout_constraintBottom_toBottomOf="parent"
            android:layout_marginBottom="16dp" />

        <Button
            android:text="OK"
            android:layout_width="93dp"
            android:layout_height="53dp"
            android:id="@+id/button2"

            app:layout_constraintRight_toLeftOf="@+id/button1"
            android:layout_marginRight="16dp"
            app:layout_constraintBaseline_toBaselineOf="@+id/button1" />
</android.support.constraint.ConstraintLayout>
```



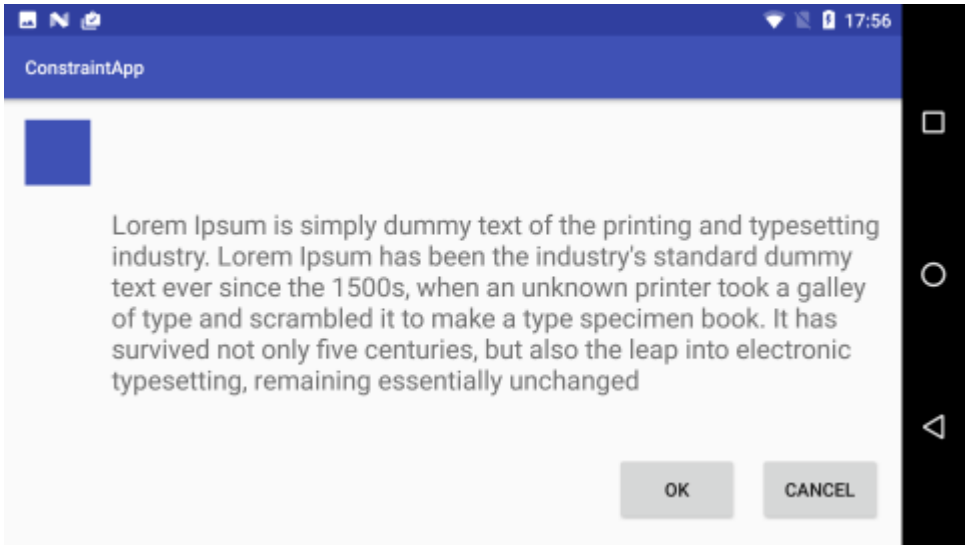
По-перше, тут елемент позиціонується щодо контейнера ConstraintLayout: від верхньої і лівої межі контейнера до відповідних меж ImageView по 16 dip.

По-друге, щодо контейнера позиціонується також кнопка з id = button1: від правої і нижньої межі контейнера до відповідних меж Button також по 16 dip.

По-третє, друга кнопка з id = button2 позиціонується щодо першої кнопки: від правої межі другої кнопки до лівої межі першої кнопки (app:layout_constraintRight_toLeftOf="@+id/button1") також 16 dip. І щоб обидві кнопки знаходилися на одному рівні, у них йде вирівнювання по базовій лінії: app:layout_constraintBaseline_toBaselineOf="@+id/button1".

І елемент TextView з шматком тексту позиціонується відразу щодо контейнера, елемента ImageView і другої кнопки.

І, наприклад, якщо ми перейдемо до альбомної орієнтації інтерфейсу, то все пропорції збережуться і всі обмеження будуть діяти також.



ScrollView

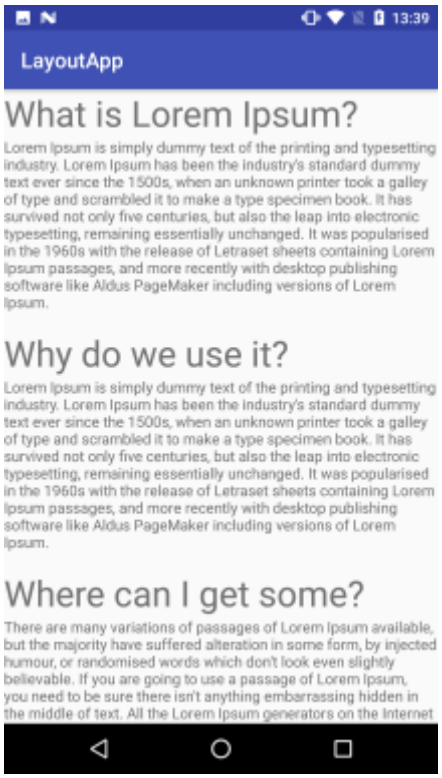
Контейнер **ScrollView призначений** для створення прокрутки для такого інтерфейсу, всі елементи якого одночасно не можуть розміщуватися на екрані пристрою. ScrollView може містити лише один елемент, тому, якщо ми хочемо розмістити кілька елементів, то їх потрібно помістити в який-небудь контейнер.

Наприклад, определим ряд TextView с большими текстами:

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    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 of the 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 of the 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="Where can I get some?"
                android:layout_marginTop="16dp"
                android:textSize="34sp"/>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="There are many variations of passages of Lorem
Ipsum available ... or non-characteristic words etc."
                android:textSize="14sp"/>
        </LinearLayout>
    </ScrollView>
```

Так як в ScrollView можна помістити лише один елемент, то всі TextView укладені в LinearLayout. І якщо площа екрана буде недостатньо, щоб розмістити все вміст LinearLayout, то стане доступною переміщення:



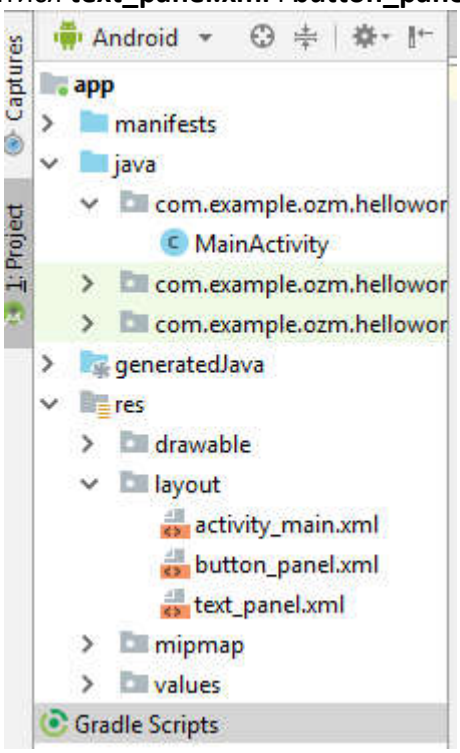
Создание ScrollView программно в коде MainActivity:

```
package com.example.ozm.helloworld;

import android.support.v7.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);

        ScrollView scrollView = new ScrollView(this);

        TextView textView1 = new TextView(this);
        textView1.setText("Lorem Ipsum is simply dummy text of the
printing and typesetting industry...like Aldus PageMaker including versions of
Lorem Ipsum.");
        textView1.setLayoutParams(new ViewGroup.LayoutParams
                (ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
        textView1.setTextSize(26);
        scrollView.addView(textView1);
        setContentView(scrollView);
    }
}
```

Вкладені layout

Одна layout може містити іншу layout. Для цього застосовується елемент **include** .

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



У файлі **text_panel.xml** визначимо наступний код:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    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="20sp"
        android:text="0 Clicks"/>
</LinearLayout>
```

По суті тут просто визначено поле TextView для виведення тексту.
У файлі **button_panel.xml** визначимо наступну розмітку:

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

Тут визначена кнопка, натискання якої ми будемо обробляти.
Основним файлом розмітки, який визначає інтерфейс програми, як і раніше є **activity_main.xml**. Змінимо його:

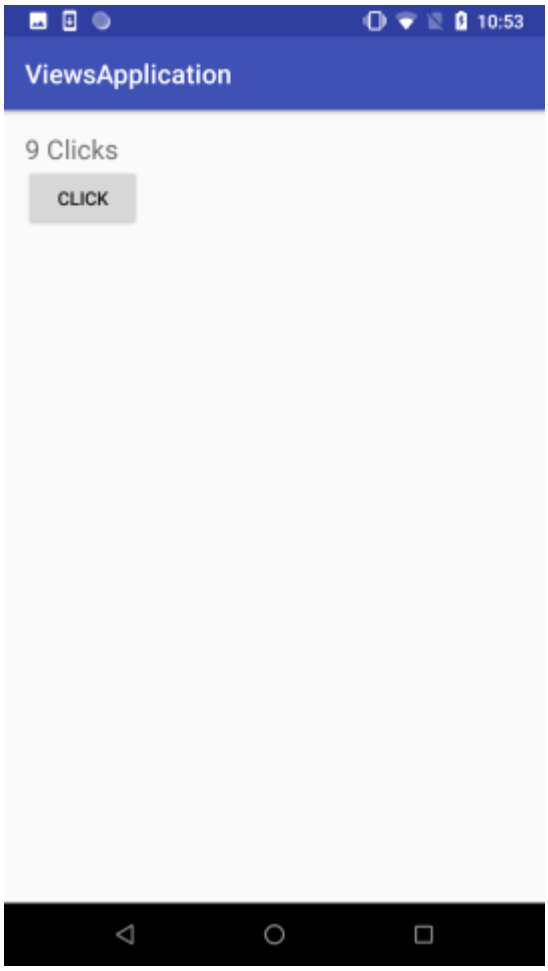
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <include layout="@layout/text_panel" />
    <include layout="@layout/button_panel" />

</LinearLayout>
```

За допомогою LinearLayout весь інтерфейс тут організовується у вигляді вертикального стека. За допомогою елементів **include** всередину LinearLayout додається вміст файлів text_panel.xml і button_panel.xml. Для вказівки назви файлу застосовується атрибут **layout** .
Це все одно, що якби ми безпосередньо замість елемента include додали вміст файлів. Однак такий спосіб має свої переваги. Наприклад, якась частина розмітки, група елементів управління може повторюватися в різних activity. І щоб не визначати по сто раз ці елементи, можна винести їх в окремий файл layout і за допомогою include підключати їх.
Також змінимо код **MainActivity** :

```
package com.example.ozm.helloworld;
```

```
import android.support.v7.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 в TextView.



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

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

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    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"
        android:text="Clicks"/>
</LinearLayout>
```

І змінимо файл **activity_main.xml**:

```
<?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"
    android:padding="16dp">

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

    <include layout="@layout/button_panel"
        android:id="@+id/top_button_panel"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/bottom_button_panel" />
    <include layout="@layout/button_panel"
        android:id="@+id/bottom_button_panel"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="10dp"
        android:layout_marginTop="10dp" />

</RelativeLayout>
```

Тепер елементи розміщені в RelativeLayout, причому файл button_panel.xml додається два рази. Важливо, що при додаванні цього файлу кожному елементу include ідентифікації абонента id. З цього id ми зможемо дізнатися, про який саме елементі include йдеться.
Також змінимо MainActivity:

```
package com.example.eugene.viewsapplication;

package com.example.ozm.helloworld;

import android.support.v7.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 topButtonPanel = findViewById(R.id.top_button_panel);
        View bottomButtonPanel = findViewById(R.id.bottom_button_panel);
        final TextView clicksText = findViewById(R.id.clicksText);

        Button topButton = topButtonPanel.findViewById(R.id.clickBtn);
        Button bottomButton = bottomButtonPanel.findViewById(R.id.clickBtn);

        topButton.setText("+");
        bottomButton.setText("-");

        topButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                clicks++;
                clicksText.setText(clicks + " Clicks");
            }
        });
        bottomButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                clicks--;
                clicksText.setText(clicks + " Clicks");
            }
        });
    }
}
```

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