МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ (Частина 1)
до лабораторних занять з навчальної дисципліни

«МОБІЛЬНІ ТЕХНОЛОГІЇ»

для студентів денної та дистанційної форми навчання
спеціальність – «комп'ютерні науки»

Затверджено
на засіданні групи забезпечення спеціальності
Протокол № 16 від «14» 06.2023
Голова групи _____ Кузніченко С.Д.

Затверджено
на засіданні кафедри __ІТ__
Протокол № 8 від «13» 06.2023
Завідувач кафедри _____ Казакова Н. Ф.

Одеса 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ (Частина 1)

до лабораторних занять з навчальної дисципліни

«МОБІЛЬНІ ТЕХНОЛОГІЇ»

для студентів денної та дистанційної форми навчання

спеціальність – «комп’ютерні науки»

Одеса 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ (Частина 1)

до лабораторних занять з навчальної дисципліни

«МОБІЛЬНІ ТЕХНОЛОГІЇ»

для студентів денної та дистанційної форми навчання

спеціальність – «комп'ютерні науки»

Затверджено

на засіданні групи забезпечення спеціальності

Протокол № 16 від «14» 06.2023

Одеса – 2023

МЕТОДИЧНІ ВКАЗІВКИ до виконання лабораторних робіт (частина 1) з навчальної дисципліни «Мобільні технології» для студентів денної та дистанційної форми навчання спеціальність – «комп’ютерні науки».

Укладач: Штефан Н. З., старший викладач кафедри інформаційних технологій. ОДЕкУ, 2023, 79 с., укр. мова.

ЗМІСТ

ВСТУП

Дане видання призначене для вивчення та практичного засвоєння студентами всіх форм навчання з основ розробки мобільних додатків. Ця галузь ІТ-технологій охоплює проектування, робробку додатку та UX/UI-дизайн .

В даних методичних вказівках розміщенні основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт. Таким чином для успішного виконання лабораторної роботи та при підготовці до її захисту відповідно до графіка студенти повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку з кожної роботи студент здає викладачу цілком оформлений звіт (студент завантажує в електроному вигляді у систему Moodle), а також демонструє на екрані комп'ютера результати виконання лабораторної роботи (в аудіторії/за допомогою Zoom).

Звіт має містити:

- титульний аркуш;
- тему та мету роботи;
- завдання до роботи;
- знімки екрану, що відображають результати роботи;
- відповіді на контрольні питання до роботи;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують у електроному форматі А4(210×297 мм). Поля сторінки з усіх боків – 20 мм.

Під час співбесіди при захисті лабораторної роботи студент повинен виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах.

Студент повинен вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен

відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

ПРАВИЛА ТЕХНІКИ БЕЗПЕКИ ТА ОХОРОНИ ПРАЦІ

Кожен студент перед початком учбової практики мусить вивчити правила роботи з комп'ютерною технікою в лабораторіях кафедри інформаційних технологій, пройти співбесіду з інструктором по техніці безпеки та розписатися в журналі по техніці безпеки.

Лабораторна робота №1

Тема: Встановлення інструментарію розробника Android. Створення першої програми на Android»

Мета:

- ознайомитись із інструментами розробки Android-додатків;
- на прикладі найпростіших програм розібрати структуру типового додатку Android;
- навчитися запускати програму на емуляторі.

Література:[1], [3]

ХІД РОБОТИ:

ВСТАНОВЛЕННЯ ІНСТРУМЕНТАРІЮ РОЗРОБНИКА ANDROID JDK

Для встановлення JDK необхідно перейти за посиланням *http://goo.gl/t3G2k1* та завантажити інсталяційний файл відповідно до своєї ОС.

СЕРЕДОВИЩЕ РОЗРОБКИ

На даний момент рекомендованою IDE є Android Studio, в основі якої лежить InelliJ IDEA. Завантажити IDE можна за посиланням *http://developer.android.com/sdk/index.html* (у процесі встановлення потрібно обов'язково вибрати Android SDK).

Якщо у вас виникла помилка, пов'язана з відсутністю JDK (JDK not found) – вам необхідно в змінних середовищах вказати шлях до встановленого раніше JDK (докладніше – зморіть посилання *http://stackoverflow.com/questions/16574189/androidstudio-installation-on-windows-7-fails-no-jdk-found*).

ВСТАНОВЛЕННЯ ANDROID SDK

Пакет Android Software Development Kit (SDK) містить бібліотеки та інструменти, необхідні для розробки додатків Android.

Android SDK включає:

- SDK Platform – окрема платформа для кожної версії Android;
- SDK Tools – інструменти налагодження та тестування, а також інші корисні службові програми. Також включає набір платформно-залежних інструментів;
- приклади додатків – містить реальні приклади коди, щоб краще зрозуміти, як користуватися API;
- документація – надає автономний доступ до новітньої документації API;
- допоміжний інструментарій Android – додаткові API, які відсутні у стандартній платформі;
- Google Play Billing – інтеграція сервісу білінгу до додатків.

Android SDK можна встановити окремо (для цього гуглить Android SDK) або в процесі встановлення Android Studio, поставивши галочку у вікні компонентів для встановлення.

За замовчуванням SDK Android включає не всі інструменти, які необхідні для того, щоб приступити до розробки додатків. Інструменти, платформи та інші компоненти представлені в Android SDK у вигляді окремих пакетів, які за потреби можна завантажити за допомогою менеджера SDK Android. Тому, перш ніж приступити до роботи, до пакета SDK Android необхідно додати деякі додаткові пакети.

НАЛАГОДЖЕННЯ НА ФІЗИЧНОМУ ПРИСТРОЇ

Для тестування та налагодження android-програми на фізичному пристрої необхідно увімкнути на пристрої режим розробника (гугліть для свого пристрою), а також завантажити Google USB Driver (див.

http://developer.android.com/sdk/win-usb.html) та встановити його (див. *http://developer.android.com/tools/extras/oem-usb.html*). Деякі фірми-виробники пристроїв (наприклад, Samsung) надають власні драйвера та установник для них, знову ж таки, гуглить для свого пристрою.

НАЛАГОДЖЕННЯ НА СТАНДАРТНОМУ ЕМУЛЯТОРІ

До складу Android SDK входить стандартний емулятор пристрою під керуванням ОС Android. Цей емулятор не відрізняється швидкодією, але є безкоштовним та має широкі можливості для емуляції.

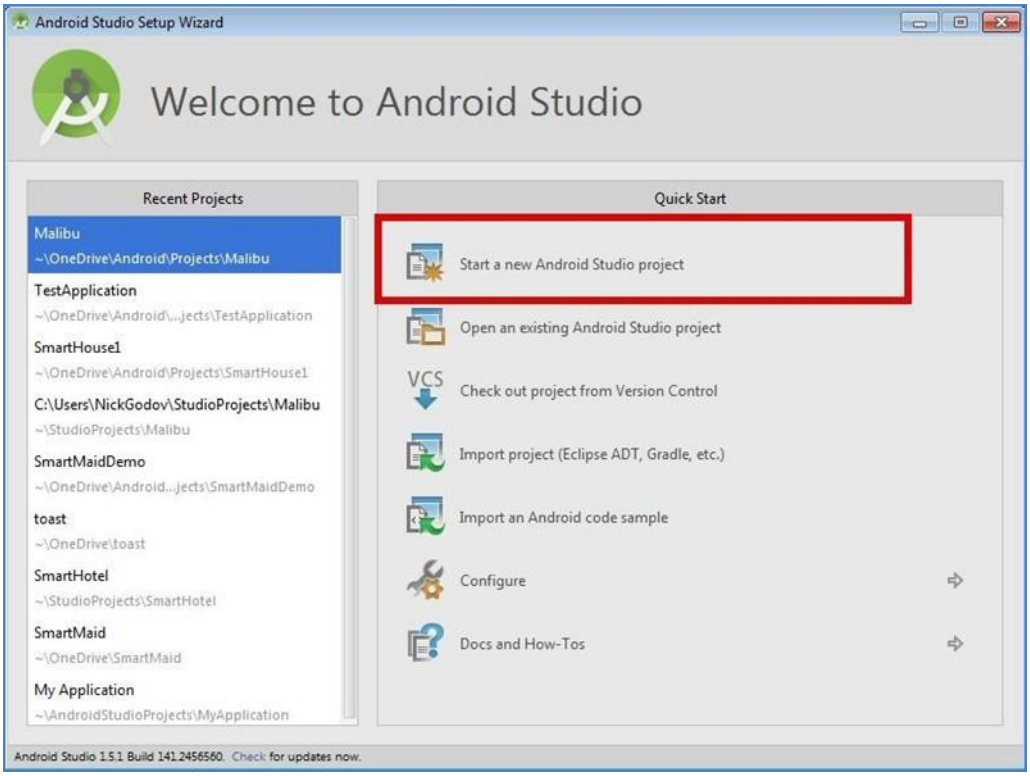
НАЛАГОДЖЕННЯ НА ЕМУЛЯТОР GENYMOTION (БАЖАНО)

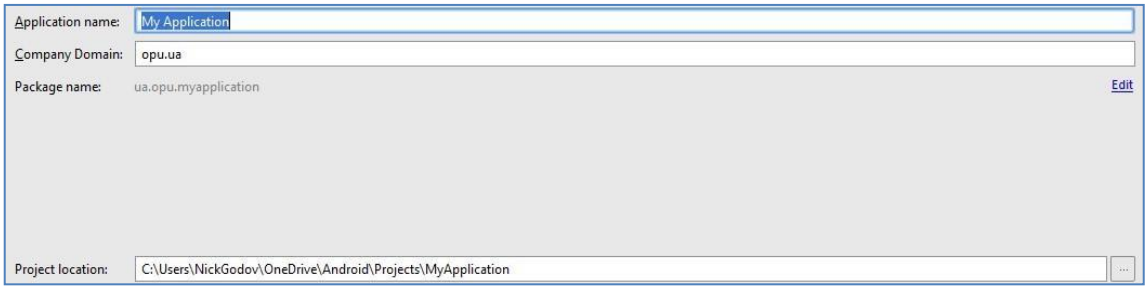
Альтернативою стандартного емулятора Android є емулятор Genymotion –*https://www.genymotion.com/#!/*. Для встановлення та скачування пристроїв необхідно зареєструватися. Після встановлення Genymotion необхідно встановити плагін для вибраної IDE. Інструкція з встановлення плагіна – *https://www.genymotion.com/#!/download*. Запущений емулятор Genymotion визначається як підключений пристрій.

СТВОРЕННЯ ПЕРШОГО ДОДАТКУ НА ANDROID

Створення нового Android-проекту за допомогою майстра. Щоб створити новий проект, натисніть **Start a new Android Studio project** або **File-New-New Project**, якщо середовище розробки вже запущено.

Далі вказуємо назву програми та сайт фірми, яка розробляєидодаток. Це необхідне формування назв пакета. Зверніть увагу, що назва пакету будується із сайту фірми та назви програми. Ім'я пакета має бути унікальним, т.к. за допомогою пакетів додаток однозначно ідентифікується. Також внизу необхідно вказати папку, де зберігатиметься проект.

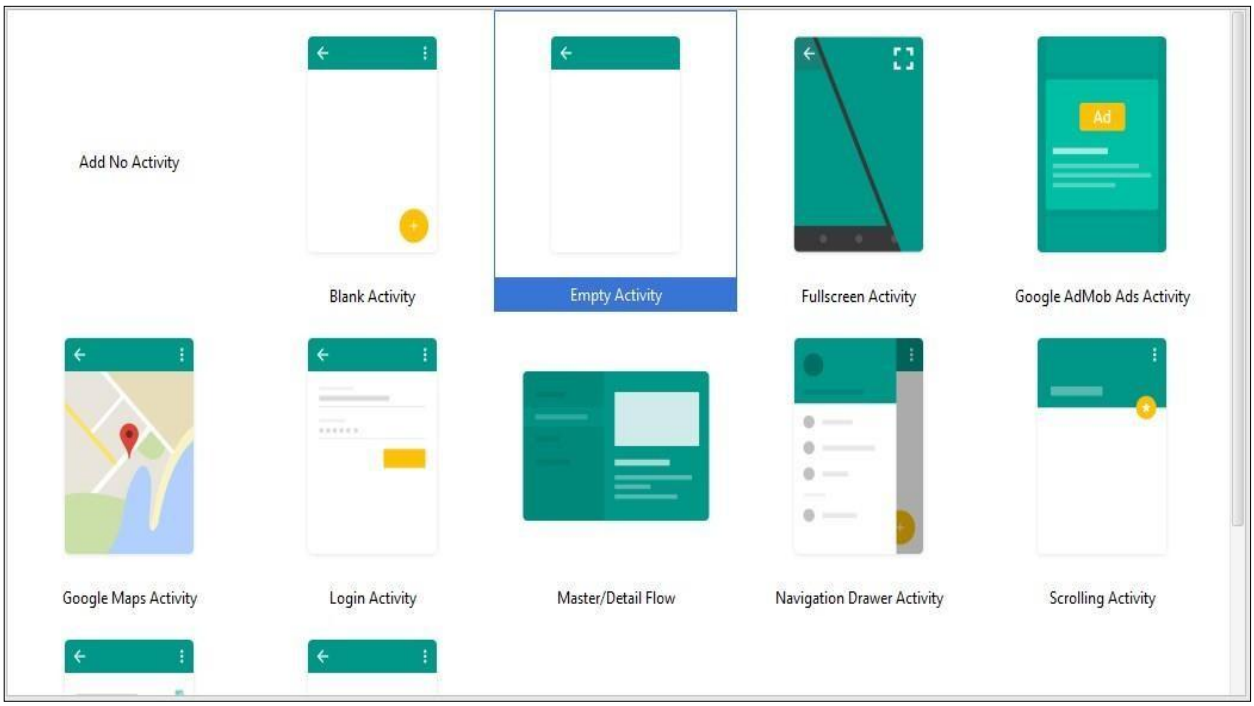


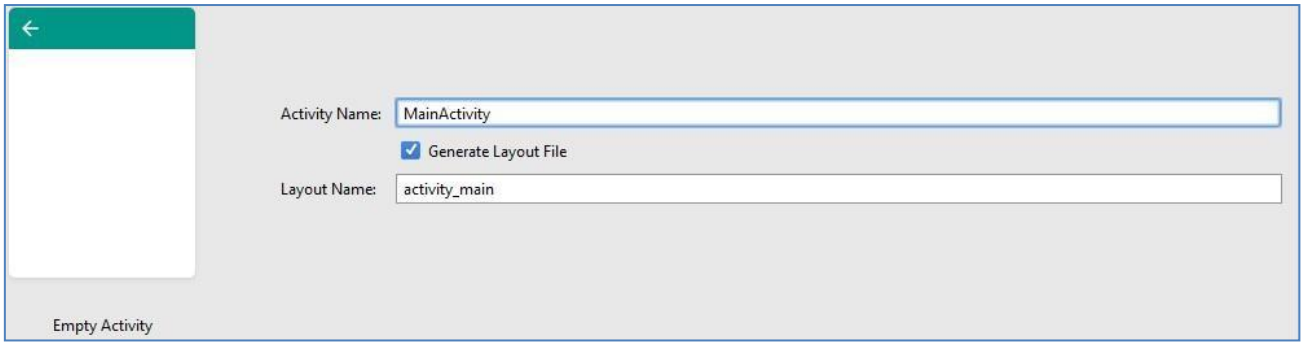


Далі ми вказуємо цільовий пристрій для програми (у нашому випадку це телефони та планшети), а також мінімальний рівень API (про нього ми говорили на лекції).

На наступному етапі нас просять вибрати, як виглядатиме стартове вікно програми. Вибираємо **Empty Activity**.

Наступний крок, нас просять вибрати ім'я класу для стартового вікна та назву макета для стартового вікна. Що це таке, ми розберемо пізніше, зараз нічого не чіпайте і просто натисніть **Finish**.

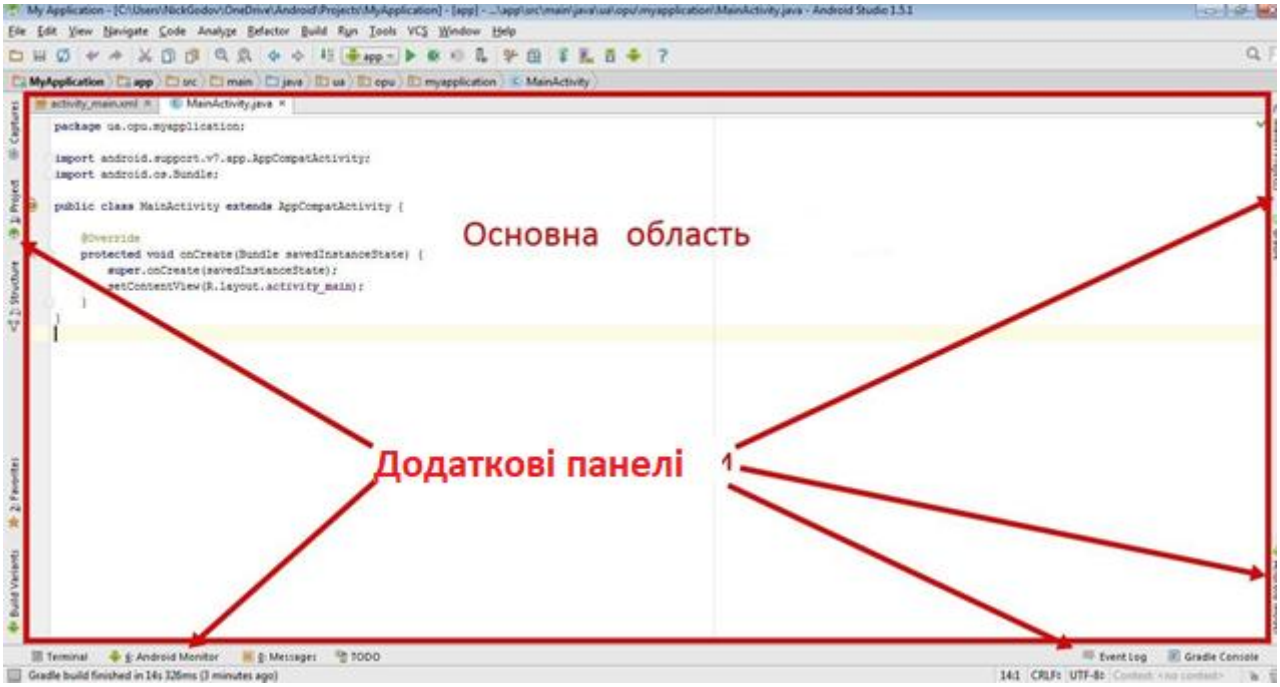




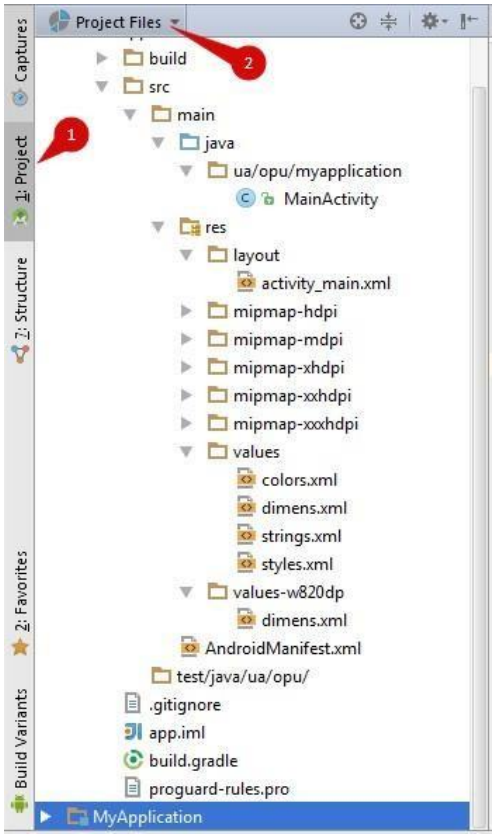
Структура Android-проекту

Після того, як ми натиснули кнопку Finish, Android Studio згенерує новий проект зі стартовим екраном.

Погляньмо на вікно Android Studio. Як ми бачимо, крім стандартного меню вгорі, також ряду кнопок з найбільш важливими операціями, що часто вживаються, у нас є основна область, де відбувається робота з проектом (написання вихідного коду, редагування макета, XML-файлів і тд), а також є бічні закладки, які можна показати та сховати кліком миші.



Натисніть на вкладку **Project** і виберемо пункт **Project Files**. Вкладка **Project** дозволяє нам переглянути структуру проекту:



Наш проект складається з кількох папок та файлів. Основні папки та файли:

1) src – вихідний код Java. Тут є основний файл для роботи. Тут же будуть нові класи.
2) gen – файли згенеровані Java.
3) res – файли ресурсів. Містить кілька підкаталогів о res/drawable-dpi – у цих чотирьох папках містяться ресурси, призначені для різних розширень екрану. Якщо зайти в кожну папку, можна знайти там значок ic_launcher.png, який є значком вашої програми;
4) res/layout – у цій папці містяться xml-файли, що описують зовнішній вигляд форм та різних елементів форм. Після створення проекту вже є файл activity_main.xml;
5) res/menu – тут є ресурси для меню;
6) res/values – тут розташовуються якісь строкові ресурси, ресурси кольорів та вимірювань, які можна використовувати у проекті. Також є схожі з ними папки values-large, values-v11, values-14, призначені для певних видів пристроїв.
7) AndroidManifest.xml – файл-маніфест із основними властивостями проекту. Вони зокрема прописуються дозволи використання інтернету.

Каталог gen в Android-проект містить генеровані значення. Зокрема, файл **R.java** – генерований клас, який містить посилання ресурси з папки res проекту. Ці ресурси містяться в директорії res і можуть бути XML-файлами, значеннями, меню, схемами, значками, малюнками чи анімаціями.

При створенні нових ресурсів відповідні посилання будуть автоматично створені в **R.java**. Посилання є статичними значеннями-інтервалами, система Android надає методи доступу до відповідних ресурсів. Наприклад, використовуйте метод **getString(R.string.yourString)** для доступу до рядка з ідентифікатором посилання **R.string.yourString.** Не рекомендується вручну змінювати файл **R.java**

AndroidManifest.xml. У цьому файлі повинні бути оголошені всі Activity, служби, приймачі та контент-провайдери програми. Також він повинен містити необхідні дозволи. Наприклад, якщо програма потребує доступу до мережі, це повинно бути визначено тут. AndroidManifest.xml можна розглядати як опис для розгортання Android-додатку.

Програма Hello, World! вже вбудована у будь-який новий проект. Просто потрібно запустити проект та отримати готову програму.

Розкрийте папку src та знайдіть файл **MainActivity.java**. Двічі клацніть файл, щоб відкрити його в редакторі коду. У файлі **MainActivity.java** вже є мінімальний код, згенерований середовищем розробки.

Подивимося на код:

activity_main.xml × MainActivity.java ×

```
package ua.opu.myapplication;

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);
    }
}
```

Перед вами відкрито файл класу, де ім'я класу збігається з ім'ям файлу з розширенням java (це правило, встановлене мовою Java). У першому рядку йде назва пакета – він ставився під час створення пакета (Package Name). Далі йдуть рядки імпорту необхідних класів для проекту

Далі йде оголошення самого класу, який успадковується (extends) від класу AppCompatActivity. У самому класі бачимо метод onCreate() – він викликається, коли програма створює та відображає Activity.

Сборка проекту

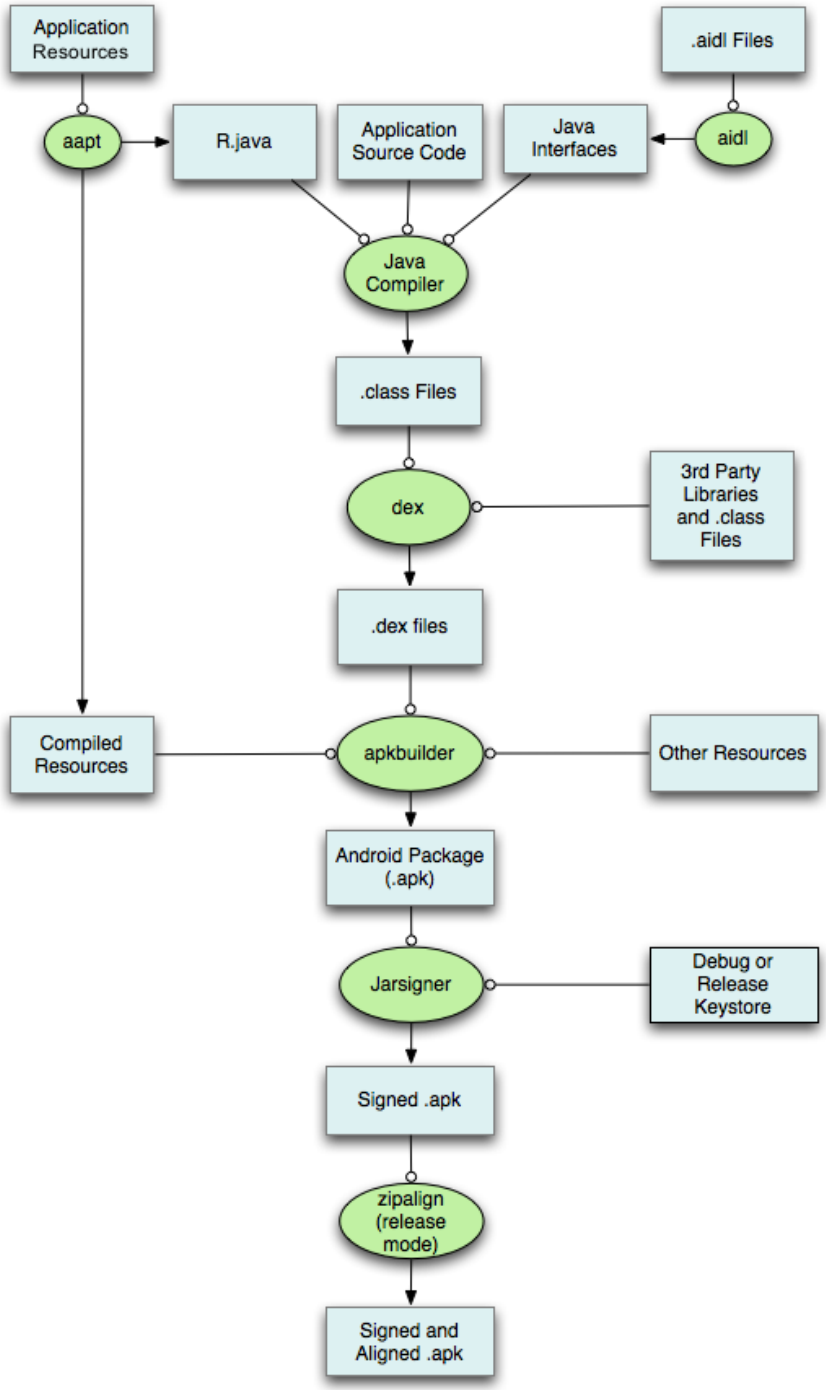
У складі Android SDK є система складання програми, за допомогою якої здійснюється складання, запуск, тестування та упаковка програми. Система складання може бути запущена або з Android Studio або за допомогою командного рядка.

Система складання має такі переваги:

- кастомізація, налаштування та розширення процесу складання;
- створення множинних APK-файлів, що настроюються для додатка для одного і того ж додатка;
- повторне використання коду та ресурсів.

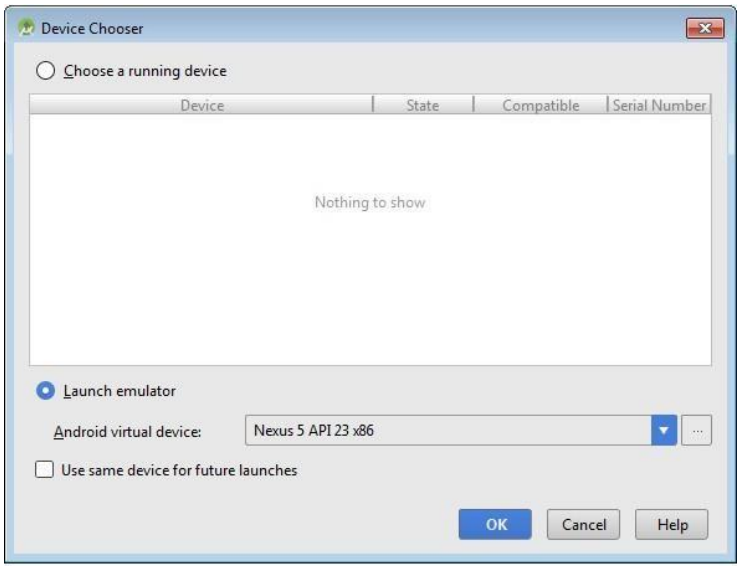
Як збирач, Android Studio використовує збирач Gradle. Детальний процес складання показаний на зображенні внизу і може бути описаний таким чином:

1. Android Asset Packaging Tool (aapt) бере всі ресурси програми, такі як файл маніфесту та інші XML-файли та компілює їх. Одним із виходів aapt виходить файл R.java, який дозволяє посилатися на ресурси у вихідному коді;
2. Утиліта aidl конвертує .aidl інтерфейси в java-інтерфейси;
3. Весь вихідний код, включаючи R.java, а також .aidl файли компілюються компілятором Java. На виході виходять .class-файли.
4. Утиліта dex конвертують .class-файли в Dalvik байт-код, файли з розширенням .dex. Сторонні бібліотеки, а також сторонні файли .class також конвертуються в байт-код;
5. Усі не компілювані ресурси (наприклад, зображення), компілювані ресурси, а також .dex-файли подаються на вхід утиліти apkbuilder, який упаковує все в apk-файл;
6. APK-файл підписується debug чи release ключем перед встановленням на пристрій;
7. На останньому кроці використовується утиліта zipalign, яка підписує apk.



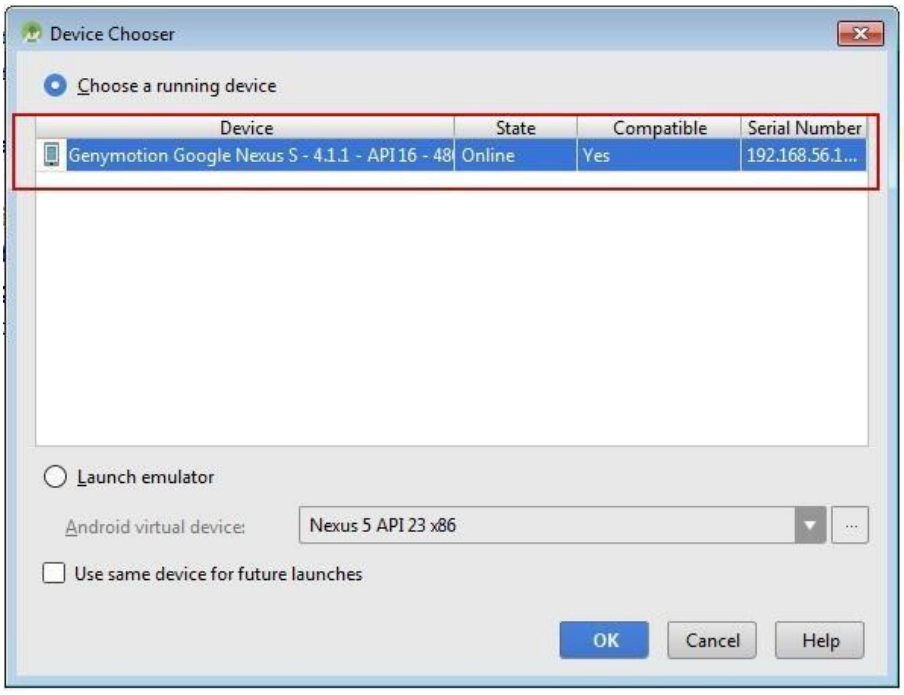
Запуск проекту

Для запуску проекту вибираємо у меню Run | Run. При спробі запустити проект, Android Studio просить вказати пристрій, на якому буде запущено програму. Це може бути як реальний фізичний пристрій, так і віртуальний пристрій.

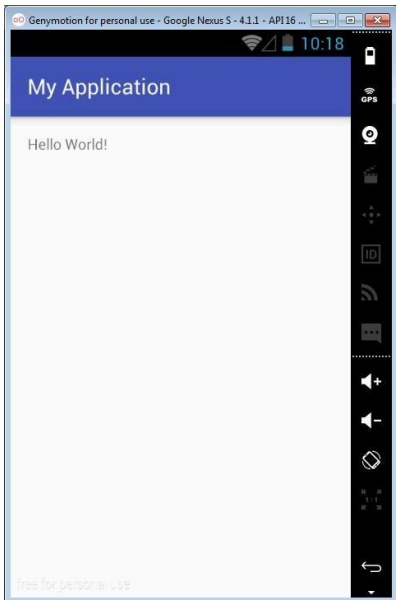


Ви можете використовувати реальний пристрій або використовувати будь-який емулятор. Також у вікні Device Chooser можна запустити стандартний емулятор пристрою. У цьому прикладі я використовуватиму емулятор Genymotion.

Спершу запускаємо новий пристрій від Genymotion, після чого спробуємо запустити Android-проект.



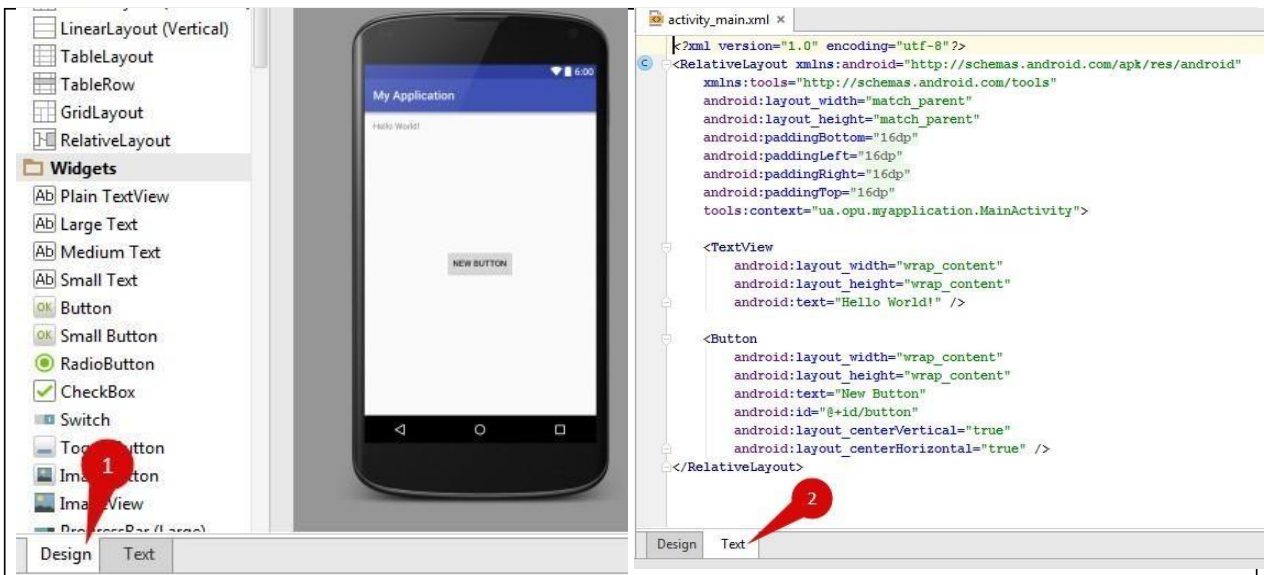
Як ми бачимо, Android Studio розпізнав запущений емулятор і вказав його у списку доступних пристроїв. Натискаємо OK. Через деякий час, ми побачимо, що наша програма була успішно встановлена на віртуальний пристрій.



Зміна макету програми

Знайдіть у проекті папку res\layouts та відкрийте файл activity_main.xml.

Перед вами з'явиться редактор макетів. Він дозволяє настроювати макет вікна як у графічному, так і текстовому режимі



Використовуйте графічний режим (вкладка Design), знайдіть кнопку та перетягніть її у вікно. Тепер у вікні є кнопка. Але якщо ми її натиснемо, нічого не станеться, т.к. ми не прописали реакцію натискання кнопки.

Для роботи з елементами GUI в Android використовується стандартний патерн "Слухач".

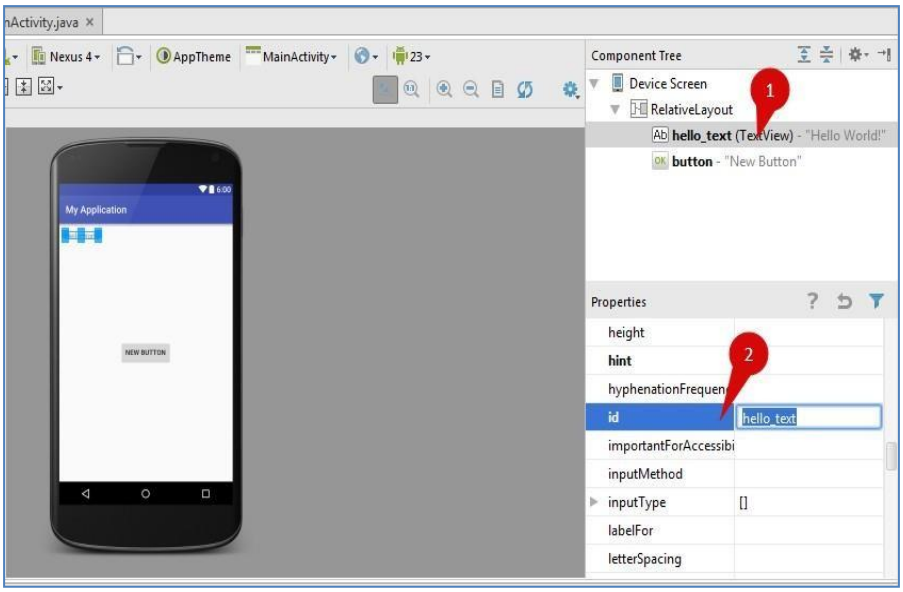
Хід роботи:

1. У MainActivity.java додаємо наступний метод

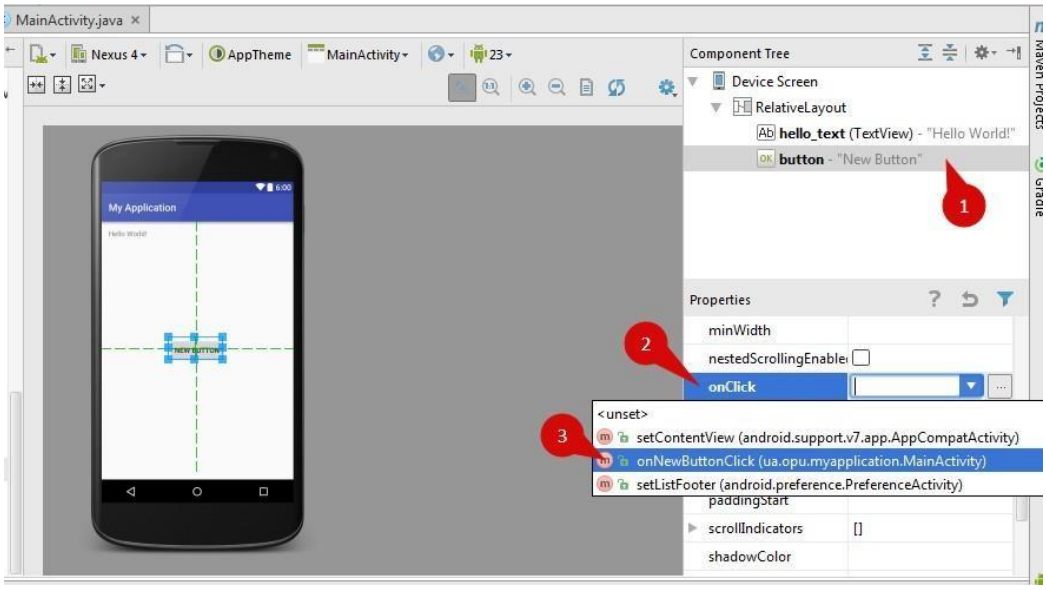
```
public class MainActivity extends AppCompatActivity{

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

    public void onNewButtonClick(View view) {

    }
}
```

Зверніть увагу на метод сигнатури. Метод має бути public void і приймати на вхід об'єкт view класу View.

1. У activity_main.xml виділяємо праворуч вгорі текстовий напис і присвоюємо їй якийсь id



Там же виділяємо нашу додану кнопку та виставляємо властивість onClick. З списку вибираємо наш створений метод



Переходимо в MainActivity.java та прописуємо вміст нашого методу

```
public void onNewButtonClick(View view) {
    TextView tv = (TextView) findViewById(R.id.hello_text);
    tv.setText("Нажато!");
}
```

Більш детально пояснення методу буде дано на лекції, на даний момент вам потрібно знати, що розмітка вікна є ресурсом, всі елементи розмітки (кнопки, текстові написи, списки, що випадають і тд) теж є ресурсами.

Щоб змінити напис *Hello World*, ми повинні з вихідного коду звернутися до конкретного ресурсу (текстовий напис). Щоб це зробити, ми спочатку повинні привласнити цьому ресурсу якийсь ID, після чого за допомогою методу **findViewById** класу **Activity** отримати посилання на потрібний ресурс.

Посилання на всі ресурси зберігаються у *класі R.java*, що генерується за допомогою утиліти **aapt**. Якщо ми відкриємо *файл R.java*, ми зможемо знайти наш ресурс – текстовий рядок з **id hello_text**

```
public static final class id {
    public static final int hello_text=0x7f0c0050;
    public static final int action_bar=0x7f0c0041;
    public static final int action_bar_activity_content=0x7f0c0000;
    public static final int action_bar_container=0x7f0c0040;
    public static final int action_bar_root=0x7f0c003c;
    public static final int action_bar_spinner=0x7f0c0001;
```

Тобто, всередині класу **R** було згенеровано статичний клас **id**, в якому було згенеровано статичний int з ім'ям *hello_text*. Таким чином, ми звертаємось до цього ресурсу за допомогою синтаксису *R.id.hello_text*

В коді не зазначено який саме це ресурс – рядок, кнопка, список тощо, просто вказано, що назва *hello_text*. Метод **findViewById()** повертає посилання на об'єкт класу **View**. Тому, щоб працювати з цим об'єктом далі,

ми повинні зробити приведення типів – перетворити об'єкт класу **View** на об'єкт **класу TextView**.

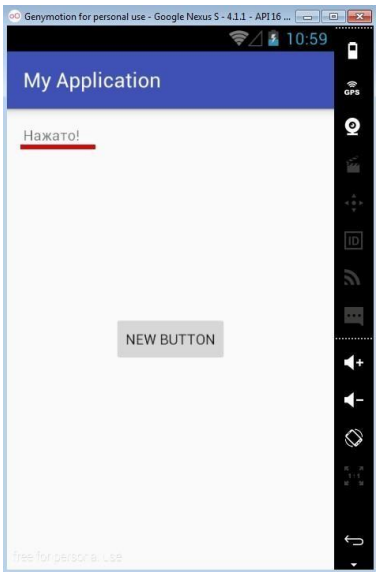
Таким чином, рядок

```
TextView tv = (TextView) findViewById(R.id.hello_text);
```

означає – привласнити змінної tv класу TextView посилання на об'єкт **TextView**, який ховається під *id hello_text*.

Після виконання цього рядка коду, в змінній TV зберігається посилання на текстове поле. Ми викликаємо метод setText() цього поля. Цей метод встановлює новий текст нашого текстового поля.

Запускаємо програму, натискаємо на кнопку



У папці bin проекту можна знайти готовий файл APK.

Завдання на лабораторну роботу:

1. Встановити інструментарій для Android-розробника
2. Створити програму, описану в ході роботи.
3. Під час здачі лабораторної роботи бути готовими відповісти на теоретичні питання.
4. Крім зміни тексту, при натисканні на кнопку повинен змінюватися колір фону програми

Теоретичні питання:

1. Які базові інструменти розробника необхідно встановити для створення програм для Android?
2. Що таке Activity?
3. Що знаходиться в папках gen, src та res?
4. Навіщо потрібний AndroidManifest.xml?
5. Що таке ресурси? Які ви знаєте ресурси?
6. Що зберігається у файлі R.java?
7. Як можна отримати посилання ресурс у вихідному коді?

Лабораторна робота №2

Тема: “Файл маніфесту. Робота із Activity. Стек Activity. Механізм Intent'ів. Явні та неявні intent'і»

Мета лабораторної роботи:

- розібратися з файлом маніфесту та його роллю в Android-додатку;
- ознайомитися зі створенням та аспектами роботи Activity;
- робота з кількома Activity, стек Activity;
- механізм намірів (Intent) для взаємодії компонентів Android;
- явні чи неявні Intent'и. Їх відмінності та сценарії використання.

Література [1,2]

Хід работи:

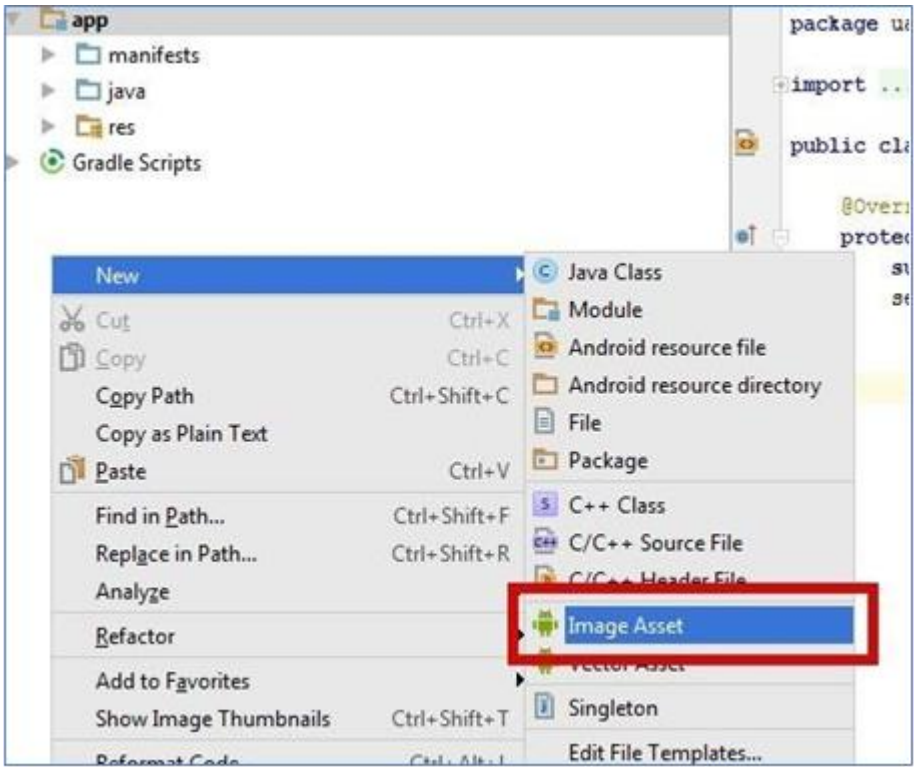
Спробуємо зрозуміти роботу Activity та механізм Intent`ів на прикладі розробки невеликої та дуже простої погодної програми «My Weather», яка показує погоду на поточний тиждень.

Зовнішній вигляд програми буде приблизно таким:

По-перше, створимо новий проект, встановлюємо мінімальну версію API рівну 16.

1. Зміна іконки додатку

Натискаємо у вікні проекту праву кнопку миші та вибираємо **New – Image Asset**.



Після цього вибираємо потрібну картинку (або вибираємо з кліпарта), налаштовуємо різні параметри іконки та натискаємо ОК. Запускаємо програму в емуляторі і переконуємося, що іконка була змінена.

2. Створення нового Activity

Activity – це окремий екран, який має свою окрему логіку та UI. Кількість Activity у програмі може бути 1 або більше. Activity, яка запускається першою, є головною. З неїможна запустити іншу активність цієї програми, а також активності інших програм.

За логіку вікна відповідає контролер вікна – клас, який успадковується від класу Activity або його похідних. За UI відповідає файл розмітки – xml файл, який зберігається у папці **res\layout.** Файл розмітки, що використовується для вікна, вказується за допомогою виклику методу ***setContentView()*** у методі контролера ***onCreate().***

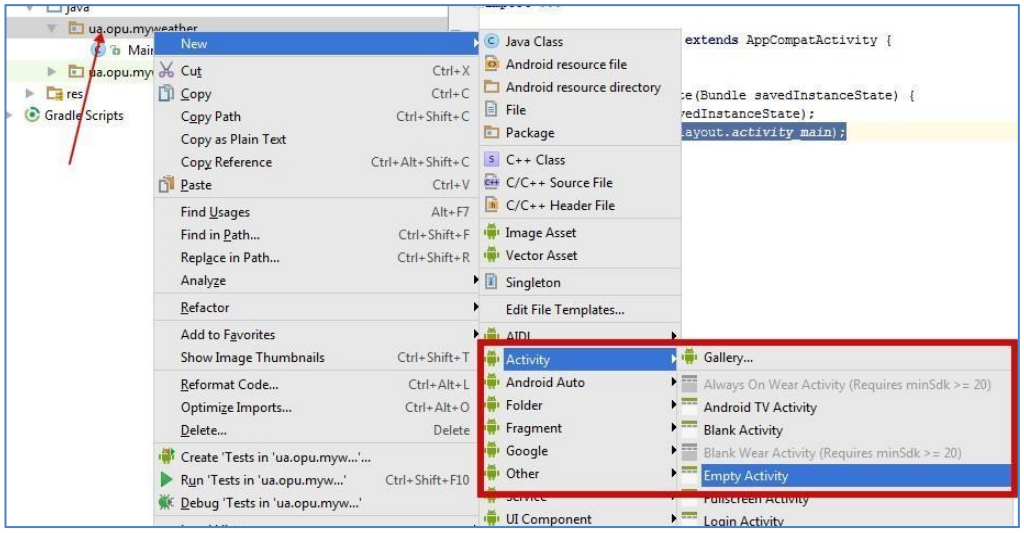
Для нашої програми необхідні два вікна – перше вікно (назвемо його *SplashActivity*), на якому поки що буде показано картинку та кнопку переходу у друге вікно; друге вікно (*MainActivity*), де буде показаний список із 7 пунктів, кожен з яких показуватиме погоду на якийсь день тижня.

Кожна програма повинна мати стартове вікно (*Launcher Activity*), яке запускатиметься при старті програми.

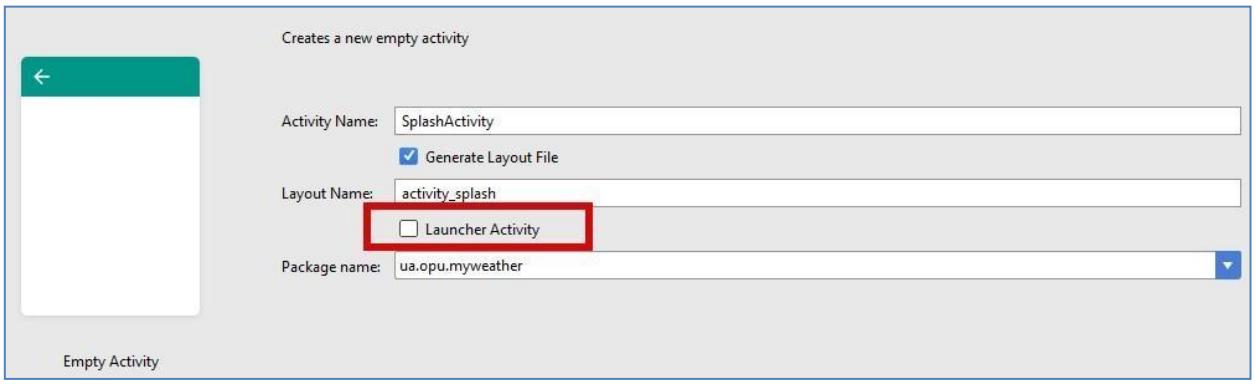
SplashActivity буде стартовим вікном, тому після створення необхідно буде поміняти стартове activity з *MainActivity* на *SplashActivity*.

Створити нову *Activity* можна вручну або за допомогою майстра.

Створимо за допомогою майстра. Для цього відкриваємо package з вихідним кодом та натискаємо праву кнопку миші та вибираємо



У вікні вказуємо ім'я класу, назву файлу розмітки. Також ви можете вказати галочку *Launcher Activity*, щоб зробити вікно стартовим. Ми вибирати його не будемо,а зробимо вікно стартовим вручну.



3. Редагування файлу манифесту

Щоб визначити стартове вікно, а також безліч інших параметрів програми, необхідно відкрити та відредагувати спеціальний файл, який називається **маніфестом програми.**

Файл знаходиться **в папці manifests** і має певну назву ***AndroidManifest.xml*** .

Файл маніфесту інкапсулює всю архітектуру Android-програми, його функціональні можливості та конфігурацію. У процесі розробки програми вам доведеться постійно редагувати цей файл, змінюючи його структуру та доповнюючи новими елементами та атрибутами. Файл маніфесту містить таку інформацію:

- оголошує ім'я Java-пакета програми, яка є унікальним ідентифікатором;
- описує компоненти програми – активіти, служби, приймачі широкомовних намірів та контент-провайдери, що дозволяє викликати класи, які реалізують кожен із компонентів, та оголошує їх наміри;
- містить список необхідних дозволів для звернення до захищених частин API та взаємодії з іншими програмами;
- оголошує дозволи, які сторонні програми повинні мати для взаємодії з компонентами цієї програми;
- оголошує мінімальний рівень API Android, необхідний роботи програми;
- перераховує пов'язані бібліотеки;

На даний момент для нас важливо, що у файлі маніфесту мають бути описані всі вікна програми, а також у файлі маніфесту вказується стартове вікно.

Відкриємо файл маніфесту. Як бачимо, це звичайний xml-файл. Нас цікавить елемент <application>. Всередині <application> є кілька елементів <activity>. Кожен елемент <activity> описує окреме вікно.

Атрибут **android:name** вказує назву ***.java-файлу***. Символ «.» спочатку означає кореневий пакет. Тобто, якщо файл знаходиться в кореневому пакеті, то потрібно писати ". MyFile", а якщо файл буде в пакеті "mypack", то потрібно писати ". mypack. MyFile".

Нам потрібно у файлі маніфесту вказати, що стартовим вікном буде SplashActivity. Для цього необхідно скопіювати елемент <intent-filter> з опису MainActivity у SplashActivity. Що означає цей елемент, ми розглянемо наприкінці лабораторної роботи.

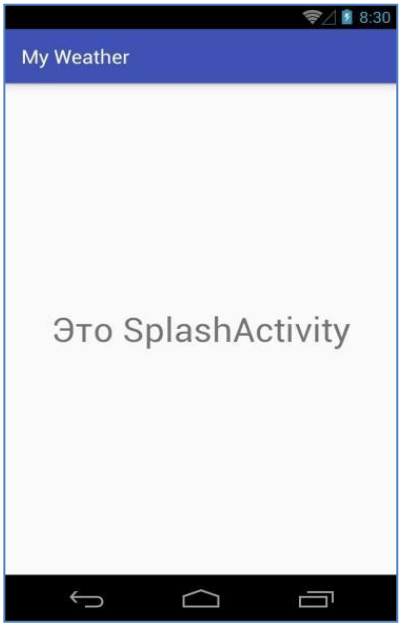
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ua.opu.myweather" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Weather"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >

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

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

        <activity android:name=".SplashActivity" >
        </activity>

    </application>

</manifest>
```

Описание MainActivity.java

Описание SplashActivity.java

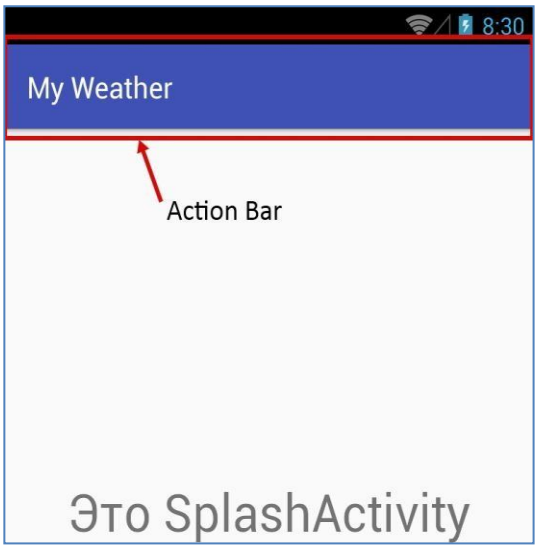
Файл маніфесту має виглядати так.

Додаємо на розмітку ***activity_splash.xml*** якийсь текст і запускаємо програму в емуляторі. Як бачимо, першим стартує вікно *SplashActivity*.



4. Сокриття Action Bar

Action Bar – панель у верхній частині програми, яка, як правило, містить назву програми та різні кнопки для керування програмою.



Для стартового вікна нам не потрібен ***Action Bar***, тому для вікна *SplashActivity* його потрібно забрати. ***Action Bar*** можна прибрати або показати для всієї програми в маніфесті, а можна прибрати або показати для окремого вікна, якщо відредагувати клас вікна.

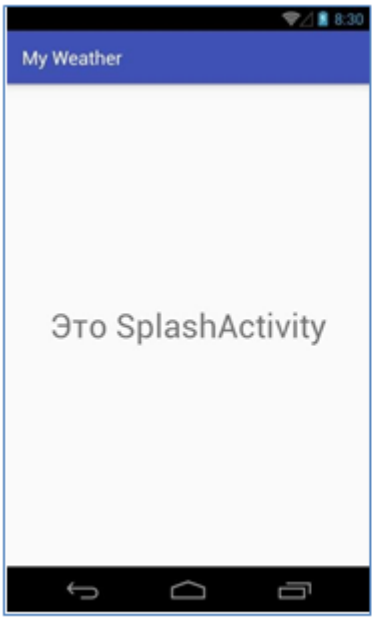
Відкриємо **SplashActivity.java.** Для отримання доступу до ***Action Bar`y*** використовуються два методи – **getActionBar()** та **getSupportActionBar()**, обидва методи повертають об'єкт класу **ActionBar**. Так як наше вікно успадковується від класу **AppCompatActivity**, який призначений для підтримки старих пристроїв, ми повинні використовувати метод **getSupportActionBar()**.

```java
package ua.opu.myweather;

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

public class SplashActivity extends AppCompatActivity {

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

        // Если мы поддерживаем старые устройства (меньше API 15)
        getSupportActionBar();
        // Если мы не поддерживаем старые устройства
        getActionBar();

    }
}
```

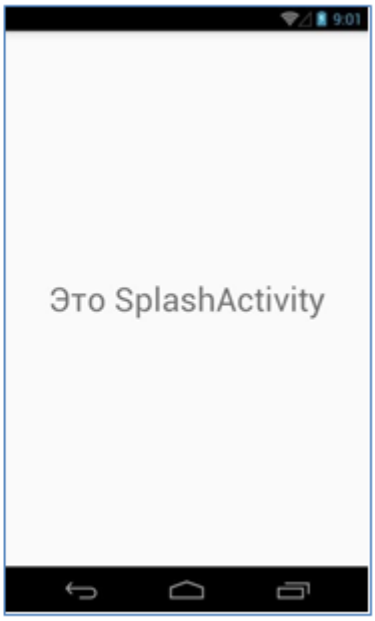
Після отримання об'єкта ***Action Bar`a*** ми викликаємо його **метод hide()**, який ховає **ActionBar** для цього вікна. Також, перш ніж отримати об'єкт **Action Bar`a**, ми повинні переконатися, що нам Android не поверне нам **null** (таке можливо за деяких обставин, про які ми поки що не говоритимемо).

```java
package ua.opu.myweather;

import ...

public class SplashActivity extends AppCompatActivity {

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

        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }
    }
}
```

Запускаємо емулятор та дивимось на результат.



Було

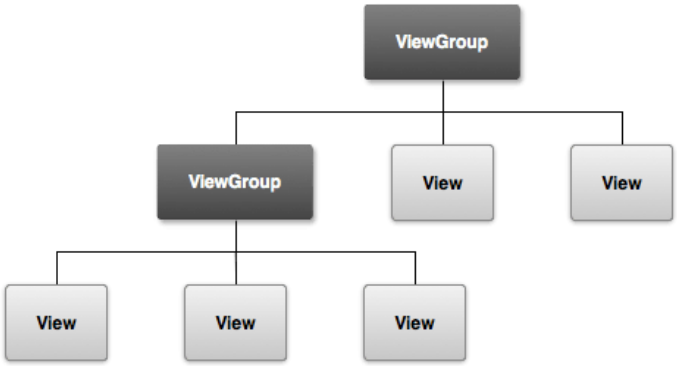


Стало

5. Редагування файлу размітки для SplashActivity

Тепер відредагуємо розмітку для нашого вікна, щоб він відповідав тому, що ми задумали на початку. Відкриваємо файл **activity_splash.xml**.

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**. Об'єкти **View** являють собою елементи керування та інші віджети, наприклад, кнопки, текстові поля та інше 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**.

ViewGroup - контейнер, що дозволяє розташувати один або кілька **View**. Контейнери успадковуються від **android.view.ViewGroup** класу, який

у свою чергу успадковується від **android.view.View**. Це означає, що дочірніми елементами контейнера може бути як **View**, а й самі контейнеры.

Приклади **ViewGroups:**

- FrameLayout – контейнер для відображення одного елемента;
- LinearLayout – контейнер для відображення одного або кількох елементів в одну лінію, горизонтально чи вертикально. Для вибору орієнтації використовується атрибут android:orientation із двома можливими значеннями
- «horizontal» та «vertical»;
- TableLayout – контейнер для розташування елементів у вигляді таблиці;
- RelativeLayout – контейнер для розташування елементів щодо батька чи один одного;
- ScrollView – контейнер, який успадковується від FrameLayout і відрізняється тим, що дозволяє прокручувати елементи, якщо вони займають більше місця ніж розмір екрана.

Так як у нас за задумом елементи (картинка та кнопка) розташовуються один під одним вертикально, будемо використовувати **LinearLayout** з орієнтацією **vertical.**

activity_splash.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="ua.opu.myweather.SplashActivity"
    android:orientation="vertical">

</LinearLayout>
```

Тепер додаємо елементи. Для відображення зображення нам потрібен

віджет **ImageView**, а для кнопки ми використовуємо **віджет Button**.

Для зображення встановлюємо атрибут src, де вказуємо джерело зображення. Додаємо наше підготовлене зображення до ***drawables*** і вказуємо його в атрибуті ***src.***

Нам потрібно, щоб картинка займала максимальний об'єм по горизонталі та вертикалі. Для встановлення розмірів віджетів використовуються атрибути **layout_height** (висота віджету) та **layout_width** (ширина віджету), які використовуються для встановлення 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.

Таким чином, встановлюємо для зображення ширину та висоту ***match_parent***. А для кнопки – ширину встановимо match_parent, а висоту – ***wrap_content***.

Якщо ми подивимося на результат, то побачимо наступну проблему - оскільки ми вказали, що картинка по висоті та ширині займає весь об'єм батьківського контейнера, то для кнопки просто не залишається місця, т.к. Android читає розмітку зверху вниз, тому він спочатку встановлює картинку на весь екран, а кнопка зникає.

Щоб це виправити, нам потрібно сказати Android`у, що спочатку повинна бути відмальована кнопка, а потім місце, що залишилося, повинна зайняти картинка. Для цього використовується атрибут **android:weight** – вага віджету. Вільний простір розподіляється між елементами пропорційно до їхніх **weight-значень.**

Встановимо для зображення **weigh**t рівним 1, а для кнопки нічого не будемо встановлювати.

```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/splash_image"
    android:src="@drawable/splash_logo"
    android:layout_weight="1" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Узнать погоду"
    android:id="@+id/weather_button"
    android:textColor="#ffffff"
    android:background="@color/colorPrimary" />
```

Тепер ми бачимо, що все гаразд.

Крім цього, ми можемо встановити колір тла для кнопки **(android:background),** колір тексту (**android:textColor**), жирне зображення **(android:textStyle=”bold”),** напис на кнопці (**android:text**).

Окремо варто згадати про відступи. Атрибут **android:padding** встановлює відступ контенту від меж контейнера. Можна встановлювати відступи тільки від однієї сторони контейнера, застосовуючи такі властивості: **android:paddingLeft, android:paddingRight, android:paddingTop** та **android:paddingBottom**.

Іншим атрибутом є **layout_margin**, що задає відступи від умовного кордону всередині контейнера в одиницях. Даний атрибут має модифікації, які дозволяють задати відступ тільки від однієї сторони: android:layout_marginBottom, android:layout_marginTop, android:layout_marginLeft та **android:layout_marginRight** (відступи відповідно від нижньої, верхньої, лівої та правої меж).

Для кнопки встановимо android:id="@+id/weather_button".

6. Відкриття вікна MainActivity. Механізм Intent

Отже, за нашим задумом, перебуваючи у вікні *SplashActivity*, потрібно відкрити вікно *MainActivity*.

Через особливості архітектури ОС Android, ми не можемо написати щось на зразок і відкрити нове вікно безпосередньо, це може робити лише операційна система.

```
MainActivity a = new MainActivity();
```

Таким чином, щоб відкрити нове вікно, ми повинні попросити зробити операційнусистему, яка створить нове вікно і виведе його на екран. І тому ми використовуємо механізм намірів **(Intent)**.

Об'єкти типу Intent можуть бути використані для спілкування між окремими частинами Android-програми, а також між різними програмами

системи. Потужність цього механізму полягає в тому, що з його допомогою можна звертатися до будь-якого встановленого в системі додатка.

Є два типи об'єктів Intent:

- ***явні об'єкти*** Intent вказують компонент, який потрібно запустити, на ім'я (повне ім'я класу). Явні об'єкти Intent зазвичай використовуються для запуску компонента з вашої власної програми, оскільки ви знаєте ім'я класу activity або служби, яку необхідно запустити. Наприклад, можна запустити нову діяльність у відповідь на дію користувача або запустити службу, щоб завантажити файл у фоновому режимі.
- ***неявні об'єкти*** Intent не містять імені конкретного компонента. Натомість вони в цілому оголошують дію, яку потрібно виконати, що дає можливість компоненту з іншої програми обробити цей запит. Наприклад, якщо потрібно показати користувачеві місце на карті, то за допомогою неявного об'єкта Intent можна запросити, щоб це зробило іншу програму, в якій така можливість передбачена.

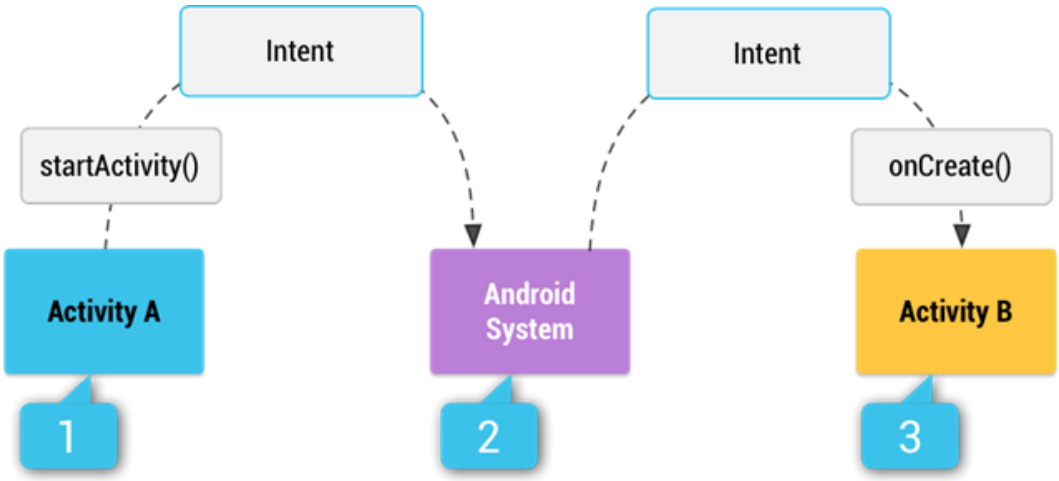
Компонент *Activity* є одним екраном у додатку. Для запуску нового екземпляра компонента *Activity* необхідно передати об'єкт Intent методом **startActivity()**. Об'єкт *Intent* описує операцію, яку потрібно запустити, а також містить всі необхідні дані.

Якщо після завершення операції потрібно отримати результат, викличте метод **startActivityForResult()**. Ваш Activity отримає результат у вигляді окремого об'єкта Intent у зворотному виклику методу активіти **onActivityResult().**

Схематичне зображення процесу передачі неявного об'єкта Intent системою длязапуску інший операции:

- Activity A створює об'єкт Intent з описом дії і передає його методуstartActivity();

- Система Android шукає у всіх додатках фільтри Intent, які відповідають даному об'єкту Intent. Коли програму з відповідним фільтром знайдено;
- система запускає відповідну операцію (Activity B), викликавши її методonCreate() і передавши йому об'єкт Intent.

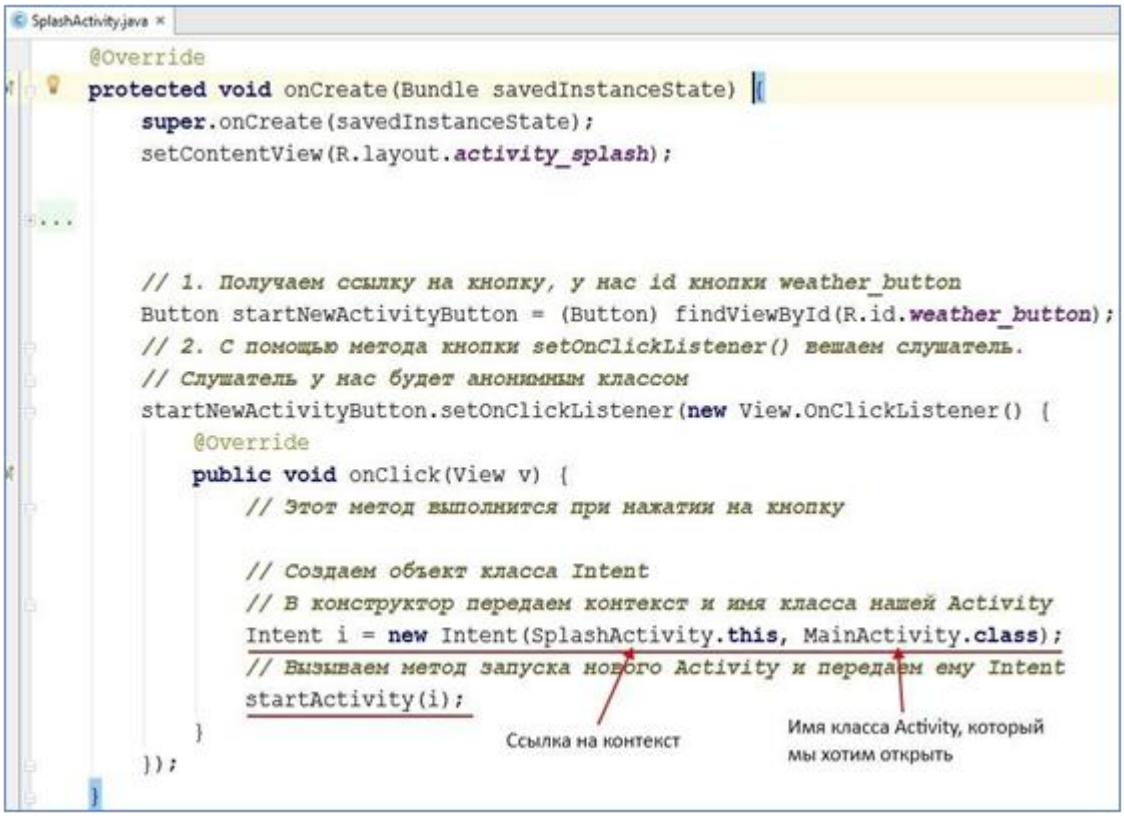


Отже, з усього вищенаписаного, ми маємо зробити таке:

1) створити об'єкт класу *Intent*;
2) явно повідомити об'єкт компонент (вказати назву класу нашого *MainActivity)*;
3) передати об'єкт *класу Intent* методу startActivity().

Все це ми повинні зробити натисканням на кнопку, тому, перш за все, ми повинні отримати посилання на кнопку і прописати слухач натискання на кнопку.

У цьому прикладі нас цікавить перший параметр конструктора *класу Intent*. Він вказує на наш поточний клас і означає посилання на контекст. Що таке контекст і навіщо він потрібен читайте нижче.



7. Контекст (Context) у ОС Android

Контекст (Context) – базовий абстрактний клас, реалізація якого забезпечується системою Android. Цей клас має методи для доступу до специфічних для конкретного застосування ресурсів і класів і служить для виконання операцій на рівні програми, таких як запуск активностей, відправлення широкомовних повідомлень, отримання намірів та інше. Від класу *Context* успадковуються такі великі та важливі класи, як *Application, Activity та Service*, тому всі його методи доступні з цих класів.

Контекст є базовим класом для класів *Application, Activity і Service*, отже його методи входять у їх склад. Саме тому для передачі контексту як параметр можна використовувати як посилання на сам контекст (**getBaseContext)**, так і посилання на класи, що успадковуються (**getApplicationContext, getContext, this, MainActivity.this, getActivity**).

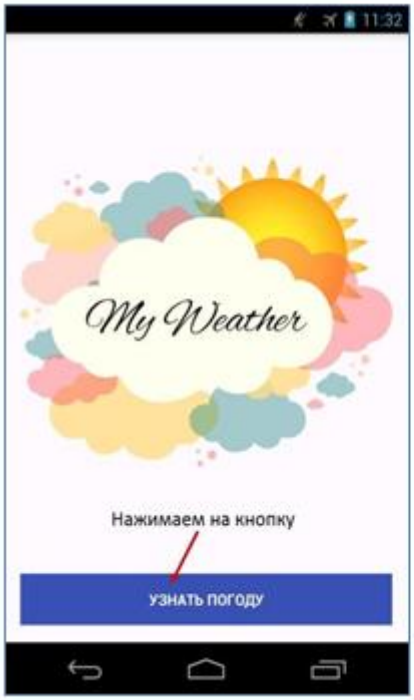
Іншими словами, контекст – це місток, що зв'язує ланку між вашим додатком та операційною системою. Класи, що реалізують контекст, створюються операційною системою.

Так як *Activity* є підкласом контексту, то ми можемо використовувати посилання на об'єкт *Activity* (наприклад, через ключове слово **this**) як контекст. Також це справедливо для класів *Application і Service*, які ми розглянемо в наступних лабораторних роботах.

Отримати доступ до ресурсів програми, записати або рахувати файл, отримати доступ до компонентів пристрою (екран, звук, датчики, кнопки), працювати з повідомленнями – для цього потрібен контекст.

В даному випадку нам потрібен контекст, щоб відправити ОС намір відкрити друге вікно. Як контекст ми передаємо посилання на об'єкт класу *SplashActivity*.

Запускаємо емулятор і дивимося результат:



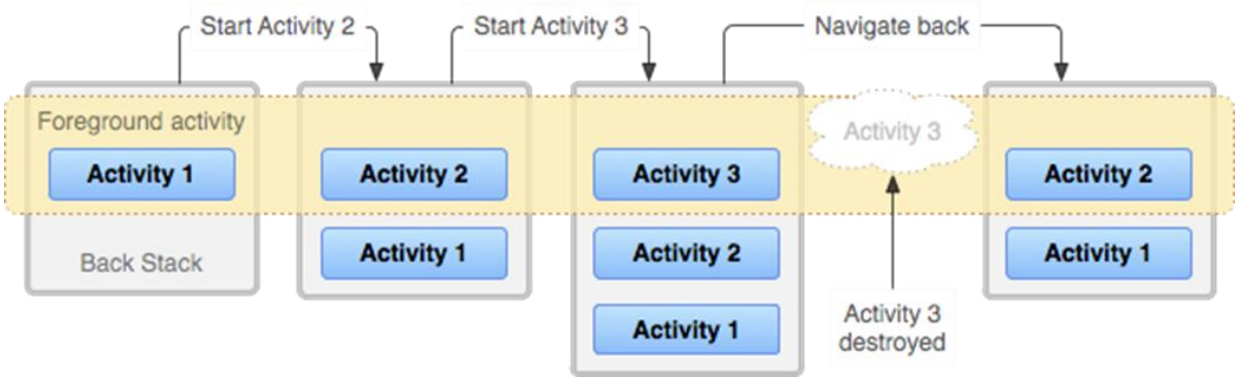
Отже, як бачимо, при натисканні на кнопку, ОС відкрила другий екран (*MainActivity*). А що сталося із першим екраном? Він знищився? А як працює кнопка Back, яка повертає нас на перший екран? Щоб це з'ясувати, потрібно зрозуміти, як працює стек *Activity*.

8. Стек Activity. Task`и

Як ми вже знаємо, кожен Android додаток складається з фундаментальних об'єктів системи - Activity. Кількість Activity в додатку буває різним, від одного до багато. При переходах між різними Activity користувач завжди може повернутися на попередню закриту Activity при натисканні кнопки back на пристрої. Подібна логіка реалізована за допомогою стека (Activity Stack). Його організація "last in, first out" – тобто, останній увійшов, перший вийшов.

При відкритті нової Activity вона стає вершиною, а попередня йде у режим stop. Стек не може перемішуватися, він має можливість додавання на вершину нової Activity та видалення верхньої поточної. Одна і та ж Activity може перебувати в стеку, скільки завгодно разів.

Task – це набір Activity. Кожен таск містить свій стек. У стандартній ситуації, кожен додаток має свій таск та свій стек. При згортанні програми, таск йде в background, але не вмирає. Він зберігає весь свій стек і при черговому відкритті програми через менеджер або через launcher, існуючий таск відновиться і продовжить свою роботу



Робота стеку Activity

Якщо продовжувати натискати кнопку **back**, то стек видалятиме Activity доти, доки залишиться головна коренева. Якщо ж на ній користувач натисне **back**, програма закриється і таск помре. Якщо ми будемо мати багато

завдань у background або просто сильно навантажувати свій пристрій, не мала ймовірність того, що таск помре через брак системних ресурсів.

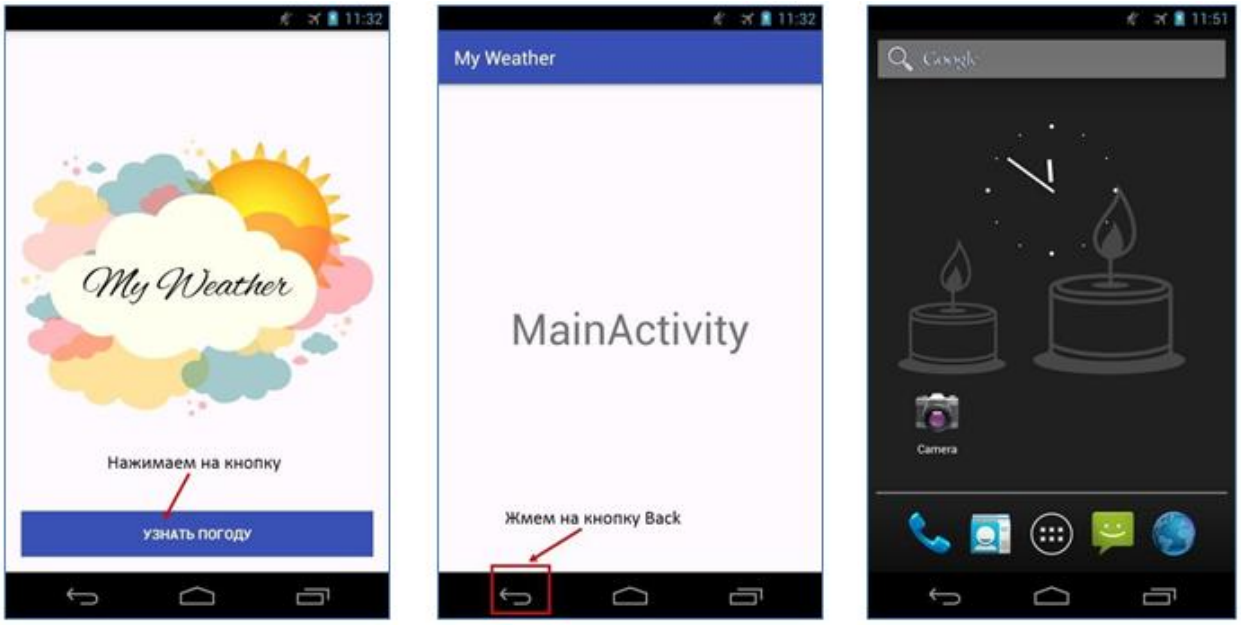
Як правило, стек та таск управляються за допомогою спеціальних прапорів, які можна передати об'єкту класу *Intent* під час запуску нової Activity. За допомогою різних прапорів ми можемо очищати стек, створювати новий таск тощо.

У нашому випадку я хочу, щоб наш перший екран – **SplashActivity**, не був доступний при натисканні **кнопки back**. Для цього я маю зробити так, щоб при відкритті **MainActivity** мій перший екран не залишався в стеку, а просто знищувався.

Це можна зробити декількома способами, наприклад, виставити спеціальні прапори для *Intent`a.* Але ми зробимо простіше і після виклику методу **startActivity()** викличемо метод **finish()** який попросить ОС знищити дане вікно.

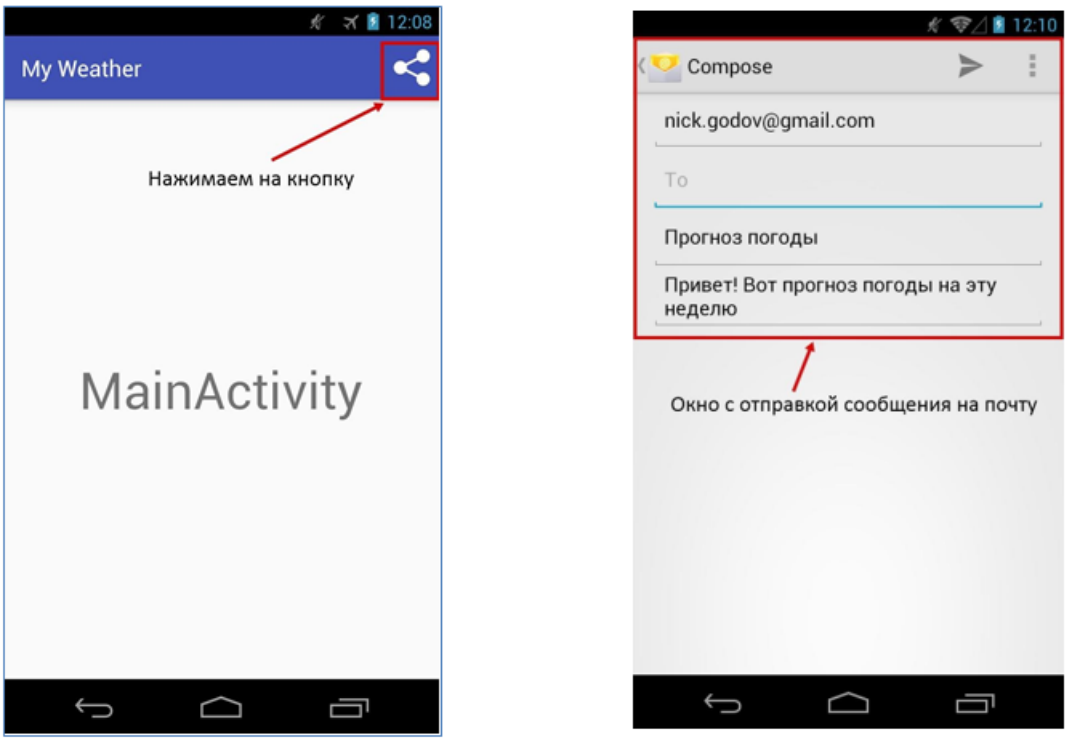
```
// 1. Получаем ссылку на кнопку, у нас id кнопки weather_button
Button startNewActivityButton = (Button) findViewById(R.id.weather_button);
// 2. С помощью метода кнопки setOnClickListener() вешаем слушатель.
// Слушатель у нас будет анонимным классом
startNewActivityButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Этот метод выполнится при нажатии на кнопку

        // Создаем объект класса Intent
        // В конструктор передаем контекст и имя класса нашей Activity
        Intent i = new Intent(SplashActivity.this, MainActivity.class);
        // Вызываем метод запуска нового Activity и передаем ему Intent
        startActivity(i);
        // Просим ОС уничтожить данное окно
        SplashActivity.this.finish();
    }
});
```

Запускаємо емулятор і дивимося на результат. Як ми бачимо, після натисканнякнопки **Back** ми не повертаємося в перше вікно, а виходимо в *Launcher*.



9 Створення меню і пункту меню в Action Bar

Для **MainActivity** додамо пункт меню для *Action Bar`a "Поділитися"*, який дозволяє поділитися прогнозом погоди (нижче наведено приклад, якщо ми ділимося прогнозом за допомогою електронної пошти).



Насамперед, додамо меню та пункт меню на Action Bar.

Ми повинні зрозуміти, що меню – це також ресурс. Меню та пункти меню описуються у форматі xml і повинні зберігатися у папці menu у папці res.

Якщо при створенні проекту у вашому Activity меню не передбачено, то ця папка буде відсутня, тому якщо її немає, просто створюємо нову папку menu.

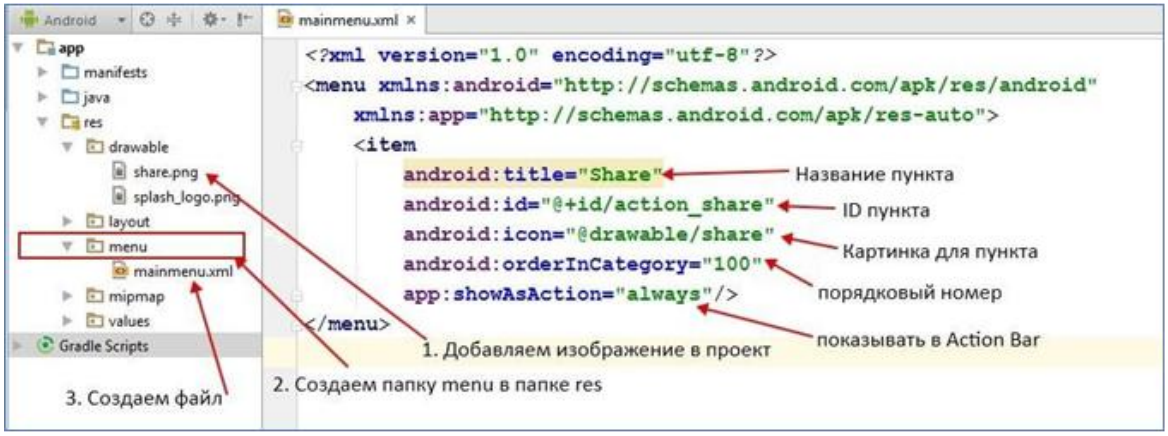
Тому, щоб створити меню:

1)додаємо іконку для пункту меню у проект;

2)створюємо папку menu у папці res;

3)створюємо файл mainmenu.xml (можете назвати його якось інакше, це не важливо).

На структурі цього XML файлу зупинятися докладно не будемо, дивіться зображення внизу, там все має бути зрозуміло.



Тепер ми повинні додати це меню у вікно *MainActivity* і передбачити реакцію натискання пункту меню. Робимо ми це у контролері вікна **MainActivity.jav**a.

1. Додаємо меню у вікно *MainActivity*. Для цього є метод базового класу **onCreateOptionsMenu()**, який потрібно перевизначити. Якщо цього вікна меню не передбачено, просто не перевизначаєте цей метод.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    // Данный метод вызывается, когда окно пытается создать меню.
    // Если в данном окне нет меню, тогда его просто не переопределяем.
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        // Указываем ссылку на xml-файл с описанием меню
        inflater.inflate(R.menu.mainmenu, menu);
        return true;
    }
```

2. Щоб запрограмувати реакцію на натискання якогось пункту меню, необхідно перевизначити метод **onOptionsItemSelected()**. Якщо у вікні це не передбачено, то просто не перевизначаємо цей метод.

У методі потрібно вказати, який саме пункт меню було натиснуто, для цього вказуємо **ID** пункту, який ми прописували у файлі **mainmenu.xml**.

```
public class MainActivity extends AppCompatActivity {

...

    // Данный метод вызывается, когда пользователь кликает
    // на какой-то из пунктов меню
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            // Если пользователь кликнул на пункт "Поделиться"
            case R.id.action_share:
                // Неявный Intent. Обратите внимание, что мы в конструкторе
                // с помощью константы указываем Действие, которое мы хотим совершить
                Intent i = new Intent(Intent.ACTION_SEND);
                // Различные параметры Intent`a
                // Тип сообщения
                i.setType("message/rfc822");
                // Передаем тему письма
                i.putExtra(Intent.EXTRA_SUBJECT, "Прогноз погоды");
                // Передаем текст письма
                i.putExtra(Intent.EXTRA_TEXT,"Привет! Вот прогноз погоды на эту неделю");
                // вызываем метод startActivity()
                startActivity(i);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

ID пункта меню, на нажатие которого мы реагируем

Неявный Intent

10. Використання неявних Intent`ов. Intent-фільтри

Давайте тепер розберемося, що відбувається в цьому коді, і що взагалі хочемо зробити з цим пунктом меню.

Після натискання кнопки Share ми хочемо надіслати прогноз погоди іншій людині. Надіслати якусь інформацію можна по-різному: електронною поштою, месенджером, Bluetooth і т.д. і ми не знаємо, що буде встановлено на телефоні користувача і який спосіб відсилання він віддасть перевагу. У таких випадках ми будемо використовувати неявний Intent.

Основна відмінність неявного Intent`а полягає в тому, що ми вказуємо не ім'я компонента, а вказуємо дію, яку ми хочемо зробити, а ОС вибирає необхідну Activity, яка може цю дію виконати (або видає діалогове вікно та просить користувача вибрати додаток).

Ще один приклад – уявіть, що у вас у додатку є кнопка «Зв'язатися з розробником» і ви хочете, щоб при натисканні на кнопку відкривалося вікно браузера з адресою сайту розробника.

Для цього нам немає потреби самим створювати вікно з браузером усередині або намагатися зрозуміти який браузер встановлений у користувача, щоб відкрити сторінку. Ми просто говоримо нашій ОС «я хочу відкрити вікно браузера з такою прописаною адресою». А вже ОС підбирає потрібну програму, яка може відкрити вікно браузера.

У нашому випадку ми говоримо ОС «Мені потрібно надіслати деяку інформацію». Усі стандартні дії перераховані як публічних констант класу Intent. Для надсилання даних ми прописуємо Intent.ACTION_SEND. Якби ми хотіли відкрити додаток для телефонів камери, ми повинні були б вказати Intent.ACTION_CAMERA_BUTTON і так далі.

Після цього ми вказуємо різні дані, які «пристібаються» до наміру. Ми вказуємо:

1) що тип інформації, яку хочемо вислати – простий текст;
2) що тема повідомлення – «Прогноз погоди»;

3) текст повідомлення «Привіт! …»

```
Intent i = new Intent(Intent.ACTION_SEND);
// Различные параметры Intent`a
// Тип сообщения
i.setType("text/plain");
// Передаем тему письма
i.putExtra(Intent.EXTRA_SUBJECT, "Прогноз погоды");
i.putExtra(Intent.EXTRA_TEXT,"Привет! Вот прогноз погоды на эту неделю");
// вызываем метод startActivity()
startActivity(i);
```

Название действия, которое нужно совершить

Указываем нужные параметры

тип сообщения - простой текст

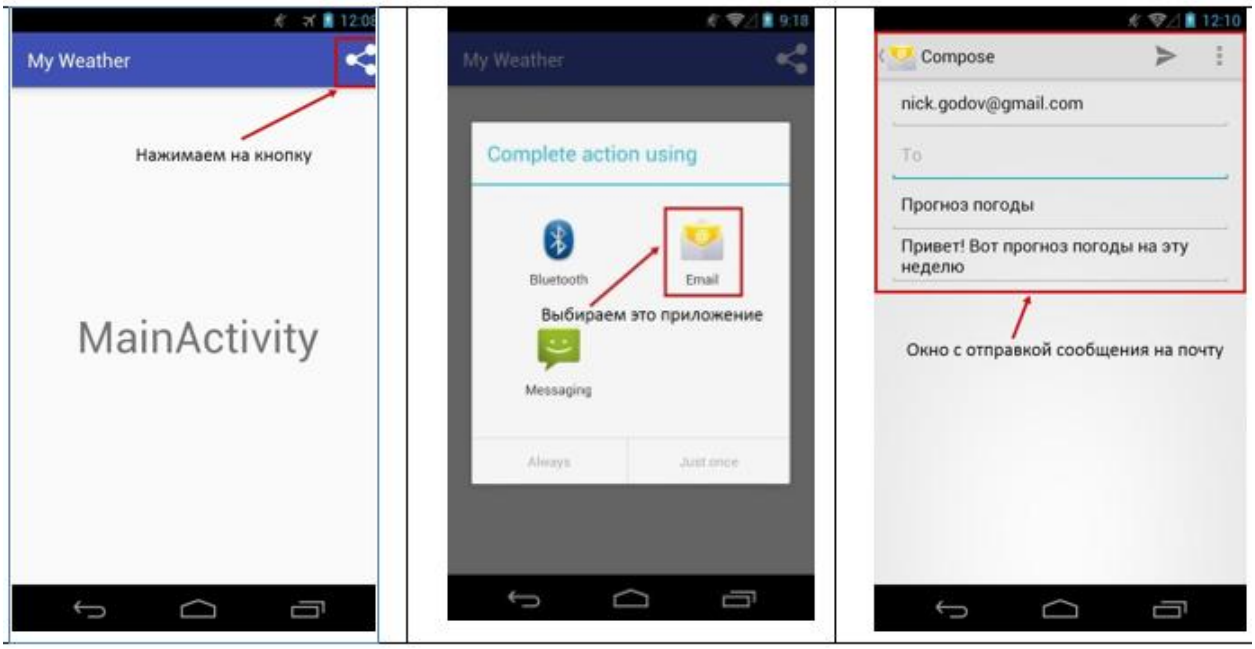
Тема сообщения (для электронной почты)

Текст сообщения

Вызываем метод startActivity()

Вказавши всі необхідні параметри, ми викликаємо метод **startActivity()** і передаємо ОС наш конфігурований намір. ОС вибирає потрібне *Activity* у додатках, встановлених на пристрої і, або запускає це *Activity*, або видає діалогове вікно.

Запустимо емулятор і подивимося, що вийшло



Отже, при натисканні на кнопку ОС видає діалогове вікно з вибором програми, ми вибираємо Email, після чого відкривається вікно надсилання повідомлення. Відкривається не просто програма, а конкретна *Activity цієї програми*. Тобто *Activity* із *програми А* може відкрити *Activity із програми В*.

Тепер відповімо на запитання – а звідки ОС знає, які Activity можуть виконати ту чи іншу дію?

Для цього, при створенні програми, ми повинні вказати для ОС «ось це Activity може виконати таку дію». І коли додаток встановлюється, то ОС зчитує ці вказівки та зберігає в собі інформацію – які Activity яких додатків можуть виконати ті чи інші дії.

Ці вказівки ми залишаємо нашій ОС у файлі маніфесту за допомогою елемента **<intent-filter>** (фільтр намірів). Давайте відкриємо наш файл маніфесту:


```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="My Weather"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >

    <activity android:name=".MainActivity" >
    </activity>

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


</application>
```

Ми бачимо, що всередині елемента activity, який описує *SplashActivity*, ми маємо елемент **<intent-filter>**. Усередині цього елемента є елемент action та category.

Елемент action якраз і визначає – які дії може зробити *Activity*. Прописаний action **android.intent.action.MAIN** означає, що це вікно є стартовим.

Елемент category описує інші відомості про те, яким компонентом має виконуватися обробка об'єкта Intent. У нашому випадку, запис **android.intent.category.LAUNCHER** означає, що ця *Activity* є початковою *Activity* програми, вона додаток вказано у лаунчері. Наприклад, створимо ще одне *Activity* і пропишемо в маніфесті наступне

```
<activity android:name=".TestIntentFilterActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:mimeType="text/plain"/>
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

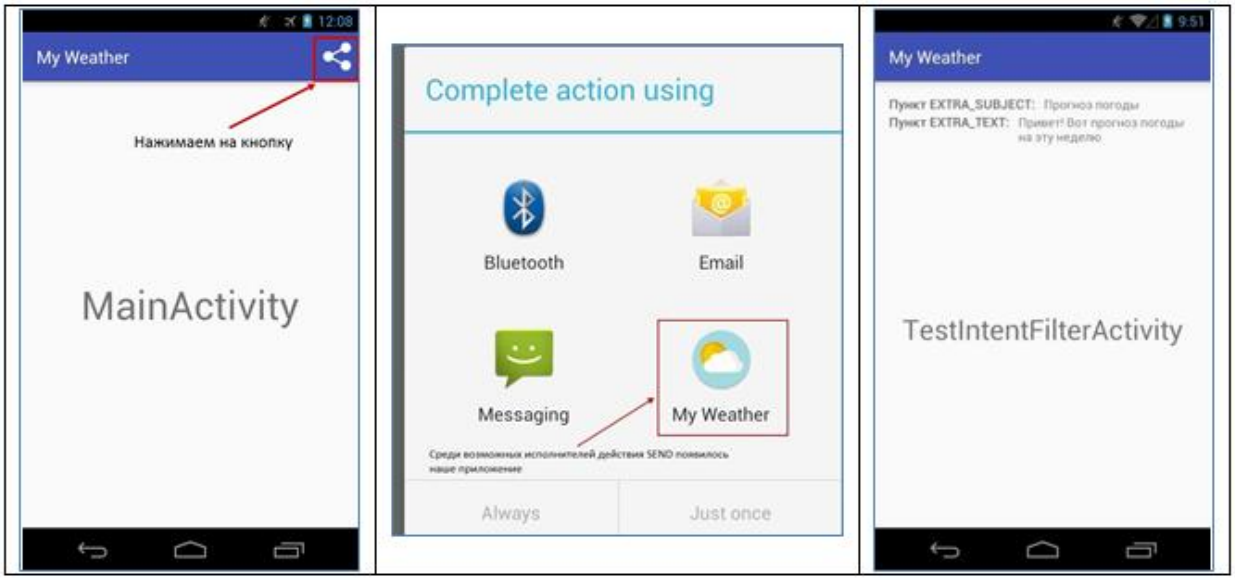
может выполнить действие SEND

если тип данных в SEND обычный текст

категория DEFAULT

Таким чином ми створили *Activity* і прописали в маніфесті, що наше *Activity* може виконати **дію SEND**, якщо потрібно вислати дані типу text/plain (елемент data відповідає за тип даних для даного action).

Запустимо емулятор і дивимося, що вийшло:



Як бачимо, коли ОС пропонує вибрати нам додаток, серед можливих варіантів з'явилося наше. Вибираємо нашу програму і відкриється вікно *TestIntentFilterActivity*.

У вихідному коді **TestIntentFilterActivity.java** ми отримуємо Intent, що прийшов, за допомогою методу **getIntent()**, після чого за допомогою методу Intent`a **getStringExtra()** отримуємо дані, які ми вислали в класі **MainActivity.java**.

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

    Intent i = getIntent();

    TextView tv1 = (TextView)findViewById(R.id.tv_subject);
    tv1.setText(i.getStringExtra(Intent.EXTRA_SUBJECT));

    TextView tv2 = (TextView)findViewById(R.id.tv_text);
    tv2.setText(i.getStringExtra(Intent.EXTRA_TEXT));


}
```

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ:

1. Створити програму з 2-3 вікон на кшталт прикладу (тематику вибирайте самі), із зображеннями, кнопками та іншими віджетами.
2. Вікна повинні відчиняти інші вікна за допомогою явних намірів.
3. Стартове вікно має містити ActionBar і 2-3 пункти меню ActionBar. Кожен із пунктів меню повинен відправляти неявний Intent з якоюсь своєю дією.

Теоретичні питання:

1. Що таке Activity?
2. Навіщо потрібний файл маніфесту і що міститься всередині нього?
3. Що таке Action Bar і як його сховати?
4. Як влаштовано файл розмітки, що таке менеджер розмітки?
5. Як встановлюються розміри віджетів?
6. Що таке Intent? Навіщо він потрібен? Які види Intent`ів ви знаєте?
7. Що таке стек Activity? Як працює цей стек?
8. Як створити меню та пункт меню в Action Bar?
9. Що таке неявні Intent`и, навіщо вони потрібні, як ОС обробляє неявні Intent`и?
10. Навіщо потрібний елемент маніфесту <intent-filter>?

Лабораторна робота № 3

Тема: «Робота із списковими елементами. Адаптери списків, створення власного адаптера та використання стандартних»

Мета лабораторної роботи:

• розібратися з файлом маніфесту та його роллю в Android-додатку;

• ознайомитися зі створенням та аспектами роботи Activity;

• робота з кількома Activity, стек Activity;

• механізм намірів (Intent) для взаємодії компонентів Android;

• явні чи неявні Intent'и. Їх відмінності та сценарії використання.

Література [1,2]

ХІД РОБОТИ:

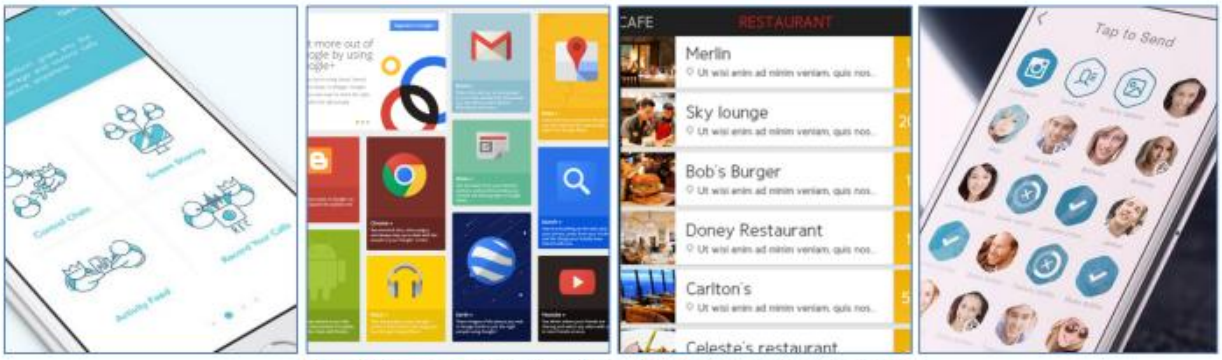
На минулих лабораторних роботах ми почали створювати наш погодний додаток My Weather, створили два Activity, навчилися перемикатися між ними і розібралися –як працює пункт меню Share.

Тепер наше завдання полягає в тому, щоб запрограмувати друге вікно, в якому програма виводить на екран погоду за тиждень. Для створення такого функціоналу в Android передбачені спеціальні віджети – спискові елементи.

Виходячи зі своєї назви, спискові елементи призначені для виведення різної інформації у вигляді списків різних форм та видів. Причому йдеться не тільки про банальний вертикальний список, а й про відображення даних у вигляді сітки, що випадає, а також різні гнучкі сітки.

Спискові елементи, так чи інакше, є практично в кожному мобільному додатку, тому архітектура ОС Android дозволяє дуже гнучко налаштовувати як структуру, так і зовнішній вигляд цих елементів, дозволяючи перетворити банальний перелік різних елементів на дизайнерські шедеври.

Ще однією важливою особливістю подібних елементів є динамічна зміна змісту та зовнішнього вигляду спискових елементів. Під час роботи програми спискові елементи можуть змінювати кількість елементів у списку, їх вміст, динамічні додавати нові елементи тощо.



Приклади використання різних спискових елементів

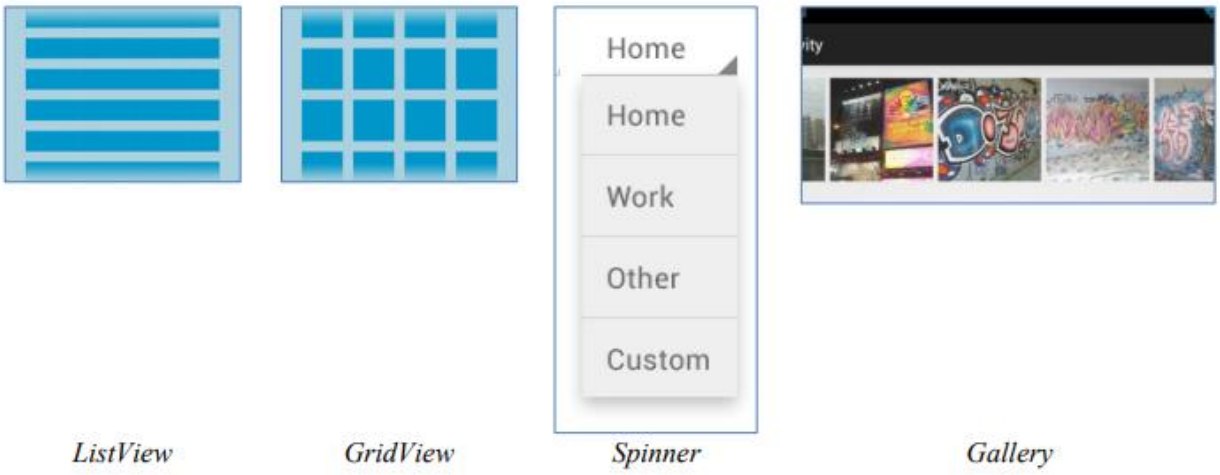
Спискові елементи успадковуються від класу AdapterView (крім класу RecyclerView, який у даному курсі не розглядається), що найчастіше використовуються:

- **ListView** - відображає список в один стовпець з можливістю прокручування;

- **GridView** – відображає сітку з рядків та стовпців з можливістю прокручування;

- **Spinner** – відображає елементи у вигляді списку, що випадає;

- **Gallery** – відображає елементи у вигляді горизонтального списку (починаючи з API 16 вважається застарілим).



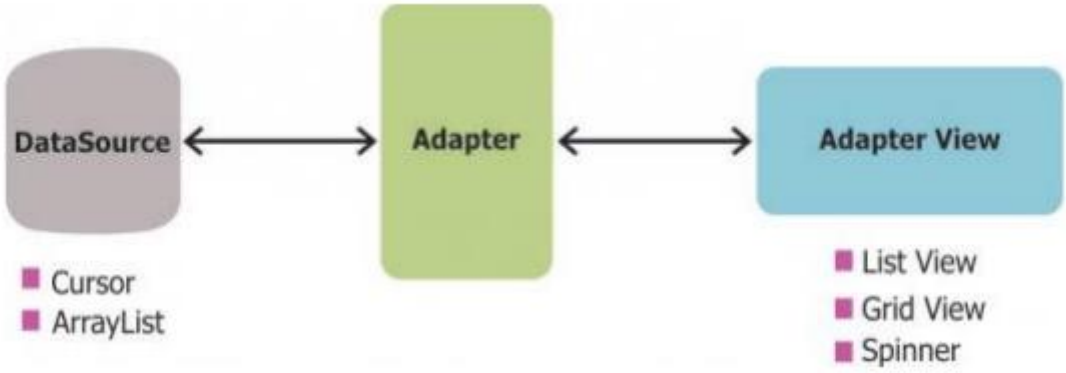
ListView *GridView* *Spinner* *Gallery*

Для того, щоб забезпечити гнучкість у налаштуванні та відображенні спискових елементів, в Android використовується поділ обов'язків між класами. Грубо кажучи, клас ListView відповідає за реалізацію прокручування, відмальовування всього списку, за отримання та втрату фокусу та інші важливі, але приховані від користувача речі, тоді як дані, які потрібно відобразити у списку та за зовнішній вигляд одного елемента списку відповідає інший клас.

Таким «допоміжним» класом для спискових елементів є адаптер (adapter, назва перейшла від патерну проектування «адаптер». Як працює адаптер: Адаптеру потрібні дані та layout-ресурс пункту списку. Далі ми передаємо адаптер списковому елементу.

Списковий елемент, при побудові списку, запитує адаптера пункти списку, адаптер їх створює (використовуючи дані і layout) і повертає списку. У результаті на екран виводиться готовий перелік. Таким чином, адаптер є свого роду посередником між даними та списковим елементом. Адаптер

визначає – 1**) які дані, і в якому вигляді виводити і 2) як виглядатиме один елемент списку**.



Сценарій використання спискового елемента та адаптера:

1. Розробник визначає клас адаптера (такий клас повинен успадковуватися від абстрактного класу **BaseAdapter**) або використовує один із стандартних адаптерів;

2. У класі адаптера передбачається передача даних для виведення у списку (зазвичай дані передаються через конструктор);

3. У класі адаптера реалізуються спеціальні методи (у класі **BaseAdapter** є абстрактними), де розробник вказує кількість елементів, дані виведення у кожному елементі, вказує layout-ресурс одному елементу тощо.

4. Розробник створює об'єкт класу адаптера;

5. Розробник отримує посилання на об'єкт спискового елемента за допомогою *методу findViewById()*, після чого за допомогою методу спискового елемента *setAdapter()* передає списковому елементу створений об'єкт класу адаптера.

6. У разі зміни даних (додавання, видалення елемента, зміна вмісту елемента) необхідно викликати *метод notifyDataSetChanged()* адаптера, щоб дані у списковому елементі оновилися.

Спеціально для демонстрації роботи спискових елементів, створимо тестову програму та розберемо за пунктами цей сценарій використання.

Створимо новий проект та створимо розмітку для стартового екрану.

“activity_main.xml”

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
 android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
 android:orientation="vertical">

<TextView
     android:layout_width="match_parent"
      android:layout_height="wrap_content"
     android:text="Список"
      android:id="@+id/textView"
     android:textSize="25sp"
           android:textStyle="bold"
      android:gravity="center_vertical|center_horizontal" />

 <FrameLayout
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:background="#000000"
     android:padding="1dp"
```

```
     android:layout_weight="1">

 <!-- Наш списочний елемент ListView -->
 <ListView
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:id="@+id/listView"
      android:headerDividersEnabled="false"
      android:layout_weight="1"
      android:layout_gravity="center"
      android:background="@android:color/white" />
 <!-- ************************************ -->

 </FrameLayout>
 <LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
 <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Добавить"
      android:id="@+id/add"
      android:layout_weight="1"
      android:textSize="12dp" />
 <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Удалить"
      android:id="@+id/delete"
      android:layout_weight="1"
      android:textSize="12dp" />
      <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Очистить"
      android:id="@+id/clear"
      android:layout_weight="1"
      android:textSize="12dp" />

     </LinearLayout>
</LinearLayout>
```

1. **Розробник визначає клас адаптера (такий клас повинен успадковуватися від абстрактного класу BaseAdapter) або використовує один із стандартних адаптерів**

Спочатку створимо клас для адаптера. Створюємо новий клас під назвою, наприклад, *FirstAdapter*. Для того, щоб використовувати цей клас як

адаптер, ми повинні вказати, що ми успадковуємося від **класу BaseAdapter**. У **класі BaseAdapter** визначено чотири абстрактні методи, які нам необхідно буде реалізувати

FirstAdapter.java

```
public class FirstAdapter extends BaseAdapter {

    // Возвращает количество элементов в списке
    @Override
    public int getCount() {
        return 0;
    }

    // Должен вернуть элемент данных, которые указаны в элементе списка position
    @Override
    public Object getItem(int position) {
        return null;
    }

    // Должен вернуть id элемента данных, которые указаны в элементе списка position
    @Override
    public long getItemId(int position) {
        return 0;
    }

    // Должен возвращать View пункта списка
    // В данном методе мы должны из
    // layout-ресурса создать View, заполнить его данными и отдать списку
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        return null;
    }}
```

На даний момент нас цікавлять *методи getCount() і getView(). Метод getCount()* викликається списком, щоб дізнатися скільки пунктів (елементів) буде містити список. *Метод getView()* відповідає за зовнішній вигляд та зміст одного пункту списку і викликається списком щоразу, коли йому потрібно вивести на екран якийсь із пунктів списку.

2. **У класі адаптера передбачається передача даних для виведення у списку (зазвичай дані передаються через конструктор)**

У нашому тестовому додатку за дані відповідатиме простий **ArrayList** (у реальних додатках дані можуть бути представлені у будь-якому вигляді). Створимо та ініціалізуємо наш масив, а також отримаємо посилання на описаний у розмітці **ListView**.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    ArrayList<String> list;

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

        // Данные
        list = new ArrayList<>();

        // Списковый элемент ListView
        final ListView listView = (ListView) findViewById(R.id.listView);
    }
}
```

Тепер необхідно передбачити передачу **колекції List** через конструктор адаптера.

FirstAdapter.java

```
public class FirstAdapter extends BaseAdapter {

    ArrayList<String> list;

    // Конструктор
    public FirstAdapter(ArrayList<String> list) {
        this.list = list;
    }

    // остальной код ...}
```

3. **У класі адаптера реалізуються спеціальні методи (у класі BaseAdapter є абстрактними), де розробник вказує кількість елементів, дані виведення у кожному елементі, вказує layout-ресурс одному елементу тощо.**

Як було зазначено вище, у межах даної лабораторної роботи нас цікавлять *методи getCount() і getView(). Метод getCount()* повертає кількість елементів у списку. У нашому випадку, кожен елемент виводитиме один елемент **колекції list**, отже, кількість елементів у списку дорівнюватиме розміру **колекції list**

FirstAdapter.java

```
public class FirstAdapter extends BaseAdapter {

    ArrayList<String> list;

    // Конструктор
    public FirstAdapter(ArrayList<String> list) {
        this.list = list;
    }

    // Возвращает количество элементов в списке
    @Override
    public int getCount() {
        return list.size();
    }

    // остальной код ...
}
```

Основну увагу слід зосередити на *методі getView()*, який є основним для розробки адаптера.

Метод getView() повертає об'єкт **класу View,** який відповідає за пункт списку та містить різні віджети з вмістом пункту списку. Важливо зрозуміти, що *метод getView()* викликається при відображенні та виведенні кожного видимого пункту списку. Одним із аргументів методу є ціле число position, яке вказує – який за рахунком пункт списку необхідно вивести. Знаючи це, ми можемо вказати, які дані потрібно виводити для того чи іншого пункту.

За зовнішній вигляд пункту списку відповідає окремий **xml-файл** розмітки, тому у *методі getView()* мають бути реалізовані такі кроки:

1) Необхідно отримати xml-розмітку та конвертувати її в об'єкт View;

2) Отримати доступ до об'єктів віджетів усередині створеного об'єкта View;

3) Заповнити віджети даними;

4) Отриманий об'єкт View передати вихід методу.

Перш за все, ми повинні створити xml-файл розмітки для списку. Для простоти зробимо так, що наш пункт списку матиме лише два текстові поля.

list_item.xml

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="888"
        android:id="@+id/item_id"
        android:textSize="35dp"
        android:textStyle="bold"
        android:layout_marginRight="20dp"
        android:textColor="#000000" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Какой-то текст"
        android:id="@+id/item_text"
        android:gravity="left|center_vertical"
        android:textColor="#000000" />
</LinearLayout>
```

Це звичайний ресурс розмітки, який зберігається в папці res\layout, тільки він застосовується не визначення зовнішнього вигляду активити, а визначає зовнішній вигляд пункту списку.

Пункти 2-4 не вимагають детального пояснення, однак важливо зупинитися на пункті 1. Щоб конвертувати xml-розмітку в об'єкт View, необхідно скористатися спеціальною службою ОС Android. Вона називається ***Layout Inflater Service*** та дозволяє подати на вхід xml-розмітку та отримати на виході контейнер View з дочірніми об'єктами-віджетами.

Служба ***Layout Inflater Service*** є об'єктом **класу LayoutInflater**, об'єкт цього класу не можна створити безпосередньо, а можна отримати посилання на спеціально налаштований об'єкт. Т.к. цією службою управляє ОС, щоб отримати посилання на неї, нам необхідно передати в адаптер об'єкт контексту (**клас Context**, див. докладніше лабораторній роботі №2)

FirstAdapter.java

```
public class FirstAdapter extends BaseAdapter {

    ArrayList<String> list;
    Context context;

    // Конструктор
```

```
    public FirstAdapter(ArrayList<String> list, Context context) {
        this.list = list;
        this.context = context;
    }
    // остальной код ...

    // Должен возвращать View пункта списка
    // В данном методе мы должны из
    // layout-ресурса создать View, заполнить его данными и отдать списку
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        // Мы должны получить xml-разметку и конвертировать ее в объект View
        // Для этого мы должны использовать специальную службу ОС Android, которая
        // называется Layout Inflater Service.
        // Чтобы это сделать, мы должны передать в этот класс контекст, после чего
        // через контекст запросить ссылку на объект

        // 1. Получили ссылку на объект Layout Inflater Service
        LayoutInflater inflater = (LayoutInflater)
                context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        // 2. С помощью метода inflate мы преобразовали xml-разметку
        // в контейнер View, который содержит внутри себя элементы из xml-разметки
        View view = inflater.inflate(R.layout.list_item, parent, false);

        // 3. С помощью метода findViewById() получаем ссылку на два TextView внутри
        // нашего созданного view
        TextView id = (TextView) view.findViewById(R.id.item_id);
        TextView text = (TextView) view.findViewById(R.id.item_text);

        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss.SSS");
        String currentDateandTime = sdf.format(new Date());

        id.setText(list.get(position));
        text.setText(currentDateandTime);

        return view;
    }
}
```

Таким чином, наш адаптер практично налаштований і готовий до використання (ми до нього знову повернемося трохи пізніше). Повернемося до **класу MainActivity.java.**

4. Розробник створює об'єкт класу адаптера

Об'єкт класу адаптера створюється як стандартний об'єкт класу. У конструкторі передаємо наші дані та посилання на контекст, щоб отримати доступ до ***Layout Inflater Service***

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    ArrayList<String> list;

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

        // Данные
        list = new ArrayList<>();

        // Адаптер.
        // При создании объекта мы должны передать:
        // 1) данные, 2) контекст
        final FirstAdapter adapter = new FirstAdapter(list, this);
    }
}
```

5. **Розробник отримує посилання на об'єкт спискового елемента за допомогою методу findViewById(), після чого за допомогою методу спискового елемента setAdapter() передає списковому елементу створений об'єкт класу адаптера**

Отримання посилання на *ListView* було розглянуто в попередніх лабораторних роботах, слід зупинитися окремо на *методі setAdapter(),* який передає об'єкт адаптера об'єкту *ListView*.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    ArrayList<String> list;

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

        // Данные
        list = new ArrayList<>();

        // Адаптер.
        // При создании объекта мы должны передать:
        // 1) данные, 2) контекст
        final FirstAdapter adapter = new FirstAdapter(list, this);

        // Списковый элемент ListView
        final ListView listView = (ListView) findViewById(R.id.listView);
        // Присоединили адаптер к списку
        listView.setAdapter(adapter);
    }
}
```

Тепер необхідно отримати посилання на кнопки та прописати реакцію на їх натискання. Робиться це за допомогою слухачів, і це розглядалося у минулих лабораторних роботах.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    ArrayList<String> list;

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

        // Данные
        list = new ArrayList<>();

        // Адаптер.
        // При создании объекта мы должны передать:
        // 1) данные, 2) контекст
        final FirstAdapter adapter = new FirstAdapter(list, this);

        // Списковый элемент ListView
        final ListView listView = (ListView) findViewById(R.id.listView);
        // Присоединили адаптер к списку
        listView.setAdapter(adapte);

        // region Слушатели кнопок

        Button add = (Button) findViewById(R.id.add);
        add.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Кнопка "Добавить"
                list.add(String.valueOf(list.size()));
            }
        });
        Button delete = (Button) findViewById(R.id.delete);
        delete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Кнопка "Удалить"
                if (!list.isEmpty()) {
                    list.remove(list.size()-1);
                }
            }
        });

        Button clear = (Button) findViewById(R.id.clear);
        clear.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Кнопка "Очистить"
                list.clear();
            }
        });
        // endregion
    }}
```

Якщо ми запустимо програму та спробуємо натиснути на кнопки, побачимо, що список не поповнюється. Це відбувається тому, що ми не

вказали адаптеру, що дані змінилися і необхідно перемалювати список заново.

6. У разі зміни даних (додавання, видалення елемента, зміна вмісту елемента), необхідно викликати метод notifyDataSetChanged() адаптера, щоб дані у списковому елементі оновилися

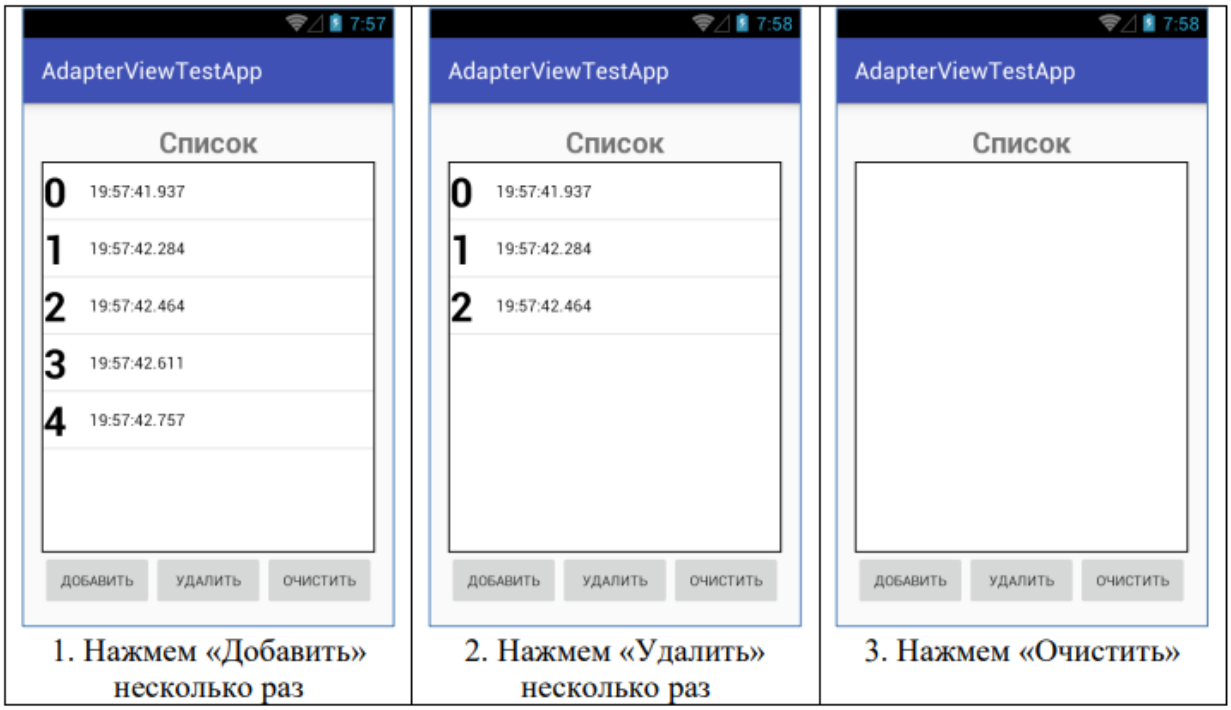
Для оповіщення адаптера про зміни з даними, ми повинні викликати спеціальний метод адаптера, який називається *notifyDataSetChanged()*. У нашому випадку, після додавання або видалення елемента в колекцію List, необхідно викликати цей метод.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    ArrayList<String> list;

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

        // Данные
        list = new ArrayList<>();

        // Адаптер.
        // При создании объекта мы должны передать:
        // 1) данные, 2) контекст
        final FirstAdapter adapter = new FirstAdapter(list, this);

        // Списковый элемент ListView
        final ListView listView = (ListView) findViewById(R.id.listView);
        // Присоединили адаптер к списку
        listView.setAdapter(adapte);

        // region Слушатели кнопок

        Button add = (Button) findViewById(R.id.add);
        add.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Кнопка "Добавить"
                list.add(String.valueOf(list.size));
                // Оповестили адаптер, что данные изменились
                adapter.notifyDataSetChanged();
            }
        });

        Button delete = (Button) findViewById(R.id.delete);
        delete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                if (!list.isEmpty()) {
                    list.remove(list.size()-1);
                    adapter.notifyDataSetChanged();
                }
                // Кнопка "Удалить"
            }
        });

        Button clear = (Button) findViewById(R.id.clear);
        clear.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Кнопка "Очистить"
                list.clear();
                adapter.notifyDataSetChanged();
            }
        });
        // endregion
    }
}
```

Запустимо програму і подивимося на результат:



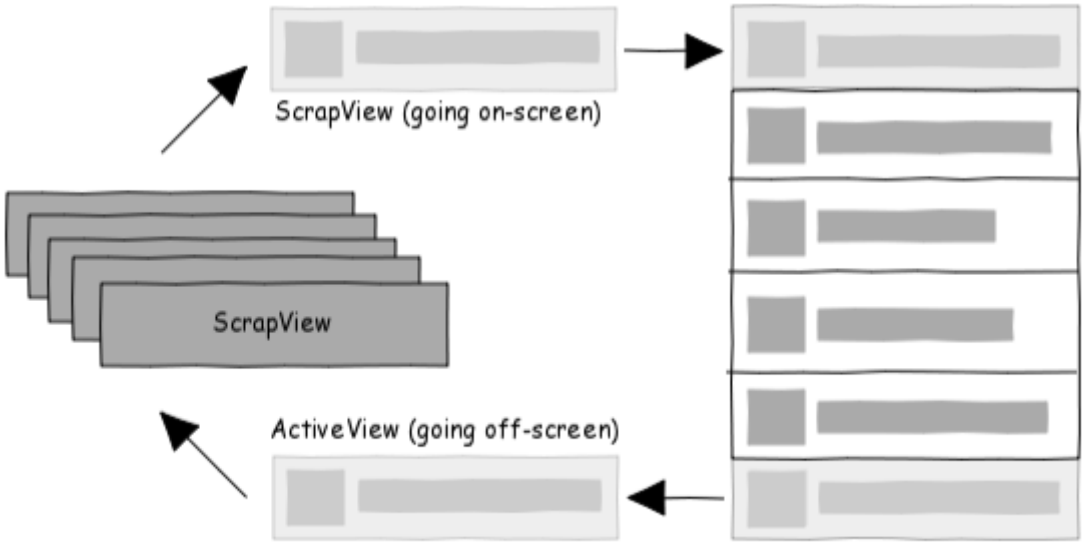
Механізм повторного використання пунктів списку

При великій кількості елементів у списку ви можете помітити, що при скролінг списку програма починає гальмувати.

Це з тим, що **ListView** не створює **View** всім пунктів списку одночасно, а викликає *метод getView()* лише тих пунктів, які у момент на екрані. Коли ви скролите список, **ListView** змушений дуже часто звертатися до *методу*

getView() і перетворювати XML-файл на об'єкт **View**, що є досить довгою операцією. Це призводить до великого навантаження на пристрій.

Щоб подолати цей недолік, у спискових елементах закладено механізм повторного використання пунктів. Він полягає в наступному - як тільки пункт списку виходить за межі екрана, він може бути використаний повторно для показу списку, який потрібно в даний момент вивести на екран.



Простий графічний опис механізму повторного використання пунктів меню

Реалізація цього механізму відбувається автоматично, нам необхідно відредагувати адаптер, саме *метод getView()*. Другим аргументом *методу getView()* є **об'єкт convertView класу View**.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
```

Коли **ListView** викликає *метод getView(),* він може передати через цей аргумент той **View**, який доступний для повторного використання (якщо є). Якщо ж в даний момент немає View для повторного використання, то аргумент дорівнюватиме **null** і ми повинні створити новий **View.**

Таким чином, ми реалізуємо механізм повторного використання **View** так:

1) Перевіряємо, чи дорівнює **convertView null**:

a) якщо **convertView** не дорівнює **null** – це означає, що нам передали **View** для повторного використання і ми не створюємо новий **View**, а використовуємо існуючий. Якщо для всіх пунктів списку використовується та сама розмітка, то ми гарантовано знаємо, що там будуть однакові віджети. Тоді ми просто заповнюємо їх потрібною інформацією;

b) якщо **convertView** дорівнює **null,** то зараз немає **View** для повторного

використання, тому ми створюємо новий **View** з розмітки.

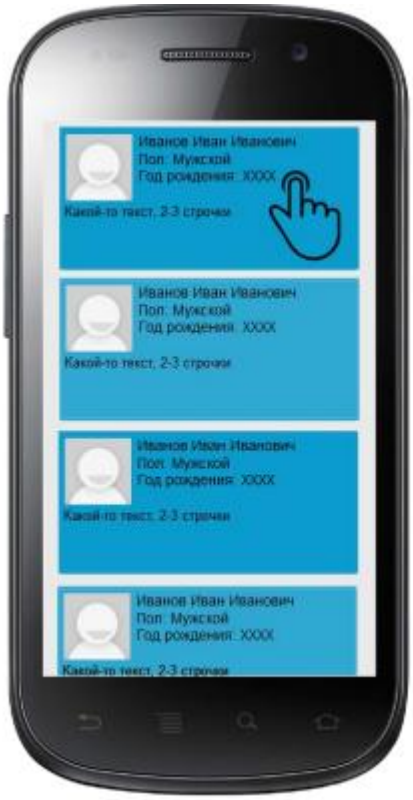
Код *методу getView()* з реалізацією механізму повторного використання виглядає так:

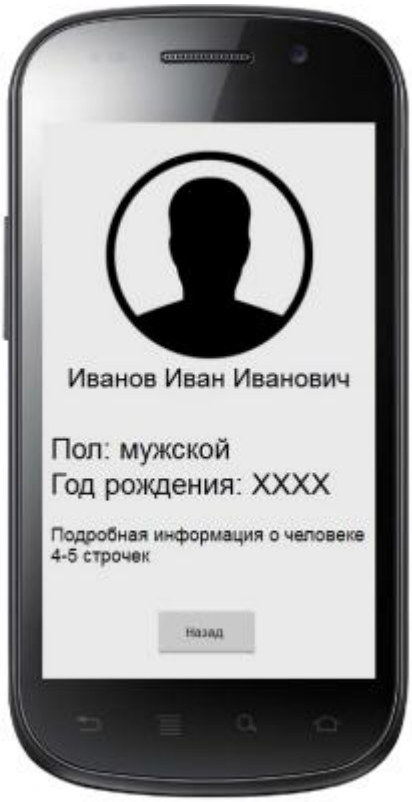
FirstAdapter.java

```
public View getView(int position, View convertView, ViewGroup parent) {

    // convertView - сюда ListView может передать
    // View который можно повторно использовать
    // а может передать null

    // Мы должны:
    // Если пришел null - создать новый объект
    // Если пришел не null - повторно его использовать

    // Мы должны получить xml-разметку и конвертировать ее
    // в объект View.
    // Для этого мы должны использовать специальную службу
    // ОС Android, которая называется Layout Inflater Service

    View view = convertView;

    // Механизм повторного использования пунктов списка
    if (view == null) {
        // 1. Получили ссылку на объект службы Layout Inflater
        LayoutInflater inflater = (LayoutInflater)
                context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        // 2. С помощью Inflater преобразовать layout-ресурс в объект View
        view = inflater.inflate(R.layout.list_item, parent, false);
    }

    // 3. Мы должны каждый пункт заполнить данными
    // Для этого, мы должны получить доступ к нашим TextView
    // внутри view
    TextView id = (TextView) view.findViewById(R.id.item_id);
    TextView text = (TextView) view.findViewById(R.id.item_text);

    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss.SSS");
    String time = sdf.format(new Date());

    id.setText(String.valueOf(position));
```

```
    text.setText(time);

    return view;
}
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ:





1. Розробити невелику програму, що складається із двох екранів. На першому екрані виводиться **ListView** з якимись даними (має бути зображення і кілька текстових полів). Це може бути список співробітників, список музичних груп, автомобілів – будь-що. Кожен пункт списку має бути красиво оформлений, містити одне або кілька зображень та кілька текстових написів.

2. При натисканні на пункт списку з'являється друге активіті з детальною інформацією про співробітника музичної групи автомобіля. Знову ж таки, це має бути добре і охайно оформлено.

- **Нагадую, що слухач на клік можна повісити на будь-який об'єкт класу View, а не тільки на кнопку;**
- **Дані для другої Activity можна передати через Intent (див. механізм Extras при описі неявних активити або гугліть щось на зразок android pass data between activities)**

Теоретичні питання:

1. Що таке спискові елементи та в чому їх відмінність від інших елементів?
2. Які спискові елементи ви знаєте і як вони виводять вміст?
3. Що таке адаптери та яку роль вони відіграють?
4. Опишіть кроки сценарію використання адаптера.
5. Що таке механізм повторного використання View? Як реалізувати цей механізм

Лабораторна робота №4
«Фрагменти в Android. Створення та робота з фрагментами. Розробка мультиекранного Activity за допомогою фрагментів та віджету ViewPager»

Мета:

1.Розібратися з фрагментами, їх призначенням та варіантами використання в Android-додатку;

2.Навчитися створювати фрагменти та маніпулювати ними;

3.Розібратися з менеджером фрагментів в Android;

4. Навчитися використовувати віджет ViewPager для створення сучасних мультиекранних інтерфейсів.

Література [1,3]

ТЕОРЕТИЧНІ ВІДОМОСТІ

Основні відомості про фрагменти.

Пристрої з мобільною платформою Android характеризуються різноманітністю розмірів екранів і дозволів, тобто високим ступенем фрагментації. Якщо для смартфонів з маленькими екранами реалізація **Activity** виглядає прийнятною, то на великих екранах (наприклад, планшетах) немає незадіяного місця. У зв'язку з цим, починаючи з API 11 в Android, були додані фрагменти, щоб розробники могли створювати більш гнучкі користувальницькі інтерфейси на великих екранах (рисунок 4.1).

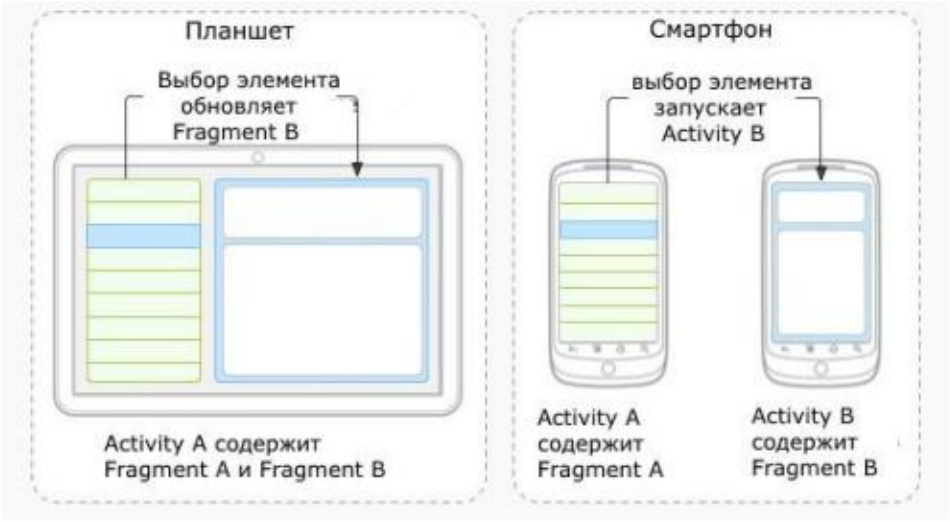


Рисунок 4.1 – Приклад реалізації концепції фрагментів на Android пристроях різного об'єму

Фрагмент існує в контексті Діяльності і має свій життєвий цикл, він не може існувати поза Діяльністю. Вправа може мати кілька фрагментів. Алгоритм створення та підключення фрагмента до Вправи наступний:

1. В окремому xml-файлі створіть графічне представлення фрагмента (наприклад, fragment1.xml).

2. Створіть клас Fragment1.java в папці java. Клас fragment повинен успадковуватися від класу Fragment:

```
public class Fragment1 extends Fragment
```

Змініть метод onCreateView, щоб підключити розмітку фрагмента:

```
public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
     View view = inflater.inflate(R.layout.fragment1, container,
     false);
     return view;
}
```

За бажанням, у методі onCreateView ви можете знайти елементи View фрагмента, встановити обробники для цих елементів та реалізувати логіку обробки подій, подібну до Activity.

3. Додайте фрагмент до дії одним із двох способів: статичним або динамічним.

Статично додайте фрагмент.

У activity_main.xml фрагмент додається як елемент <фрагмент> (рис.4.2). Для кожного фрагмента повинні бути встановлені висота, ширина, id і назва. Ім'я встановлюється на повне ім'я класу з урахуванням пакета:

```
<fragment
      android:layout_width=" match_parent "
      android:layout_height="match_parent"
      android:name="by.bsuir.emplesson7.fragments.Fragment1"
      android:id="@+id/fragment1"/>
```

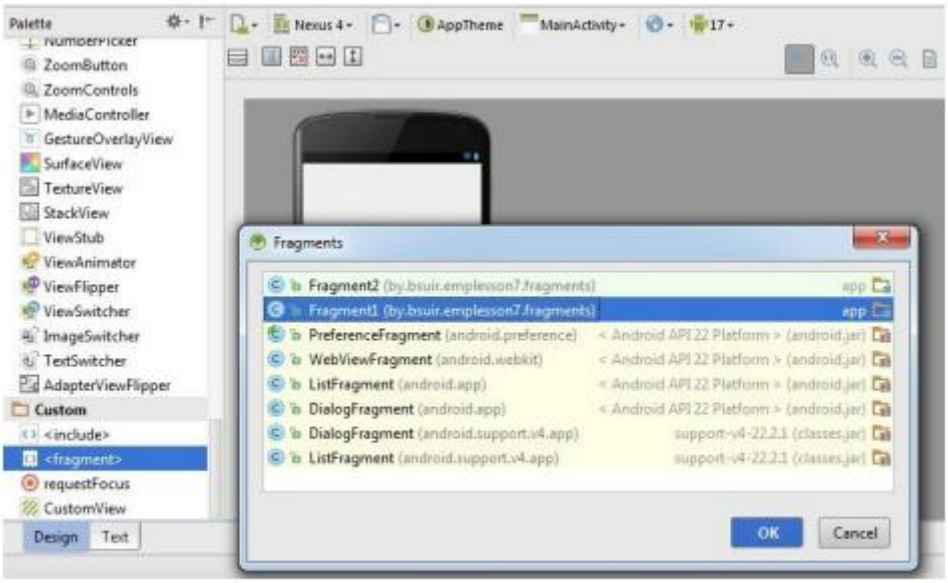


Рисунок 4.2 – Статичне додавання фрагмента до вправи

Крім фрагмента, в розмітку **activity_main.xml** можна додати і інші елементи або фрагменти. Код **класу MainActivity** залишається таким же, як і при звичайному створенні проекту.

Динамічне додавання фрагмент.

Для **activity_main.xml** додайте **FrameLayout**, який в подальшому буде служити контейнером для розміщення фрагмента з коду додатку:

```
<FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

Додайте *снінет* до згенерованого контейнера безпосередньо в коді програми: **MainActivity.java:**

```
private Fragment1 fragment1;
private FragmentManager fragmentManager;
private FragmentTransaction fragmentTransaction;

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

private void initFragments() {
     fragment1 = new Fragment1();
     fragmentManager = getFragmentManager();
     fragmentTransaction = fragmentManager.beginTransaction();
     fragmentTransaction.add(R.id.fragment_container, fragment1);
     fragmentTransaction.commit();
}
```

Використовуйте **клас FragmentManager** для керування фрагментами. Щоб його отримати, викличте *метод getFragmentManager()* з коду операції.

Клас FragmentTransaction дозволяє здійснювати операції з фрагментами:

- **add()** – додати фрагмент;
- **remove()** - видалення фрагмента;
- **replace()**-заміна фрагмента;
- **hide()-** робить фрагмент невидимим;
- **show()** - відображення фрагмента.

Приклад реалізації цих методів для двох фрагментів показаний нижче:

```
 frag1 = new Fragment1();
frag2 = new Fragment2();
fTrans = getFragmentManager().beginTransaction();
fTrans.add(R.id.fragmentContainer, frag1);//добавление фрагмента1
fTrans.remove(frag1); // удаление фрагмента1
fTrans.replace(R.id.fragmentContainer,frag2);//размещение
```

Взаємодія фрагментів і Актівіті.

Доступ до фрагмента з **Activity** здійснюється в **MainActivity.java** за допомогою *методу findFragmentById:*

```
//Для статичних фрагментів: вкажіть ідентифікатор фрагмента
Fragment frag1 = getFragmentManager().findFragmentById(R.id.fragment1);
TextView tv_frag1 = (TextView)
frag1.getView().findViewById(R.id.tv_frag1);
tv_frag1.setText(«Привіт фрагменту 1 із вправи»);

//Для динамічних фрагментів: вкажіть ідентифікатор контейнера
     Fragment frag2 = getFragmentManager().findFragmentById(R.id.
     fragment_container);
TextView tv_frag2
=(TextView)frag2.getView().findViewById(R.id.tv_frag2);
tv_frag2.setText("Hello to Fragment 2 from Activity");

//Доступ до Дії з фрагмента здійснюється в коді класу фрагмента за
допомогою методу getActivity:

TextView                        tv_activity                          =
getActivity().findViewById(R.id.tv_activity);
tv_activity.setText("Привіт з фрагмента1");
```

Обробка в Діяльності (або іншому фрагменті) події з фрагмента виконується через інтерфейс як шар для відстеження подій. Ви не можете безпосередньо зв'язати з одного фрагмента на інший!

Фрагменти: ViewPager.

Щоб реалізувати повзунок і забезпечити ефект перегортання сторінок, додатки Android використовують ViewPager. Кожна сторінка ViewPager є фрагментом. Є можливість сформувати верхнє меню вкладок з заголовками сторінок.

*Алгоритм створення **ViewPager** наступний:*

1. У **activity_main.xml** додати **ViewPager**, при необхідності сформувати

верхнє меню вкладок з заголовками сторінок, додатково прописати **PagerTabStrip**:

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

<android.support.v4.view.ViewPager
 android:id="@+id/pager"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <android.support.v4.view.PagerTabStrip
      android:id="@+id/pagerTabStrip"
      android:layout_width="match_parent"
     android:layout_height="wrap_content"
      android:layout_gravity="top">
 </android.support.v4.view.PagerTabStrip>
 </android.support.v4.view.ViewPager>
</RelativeLayout>
```

2. У розділі **Main.javaActivity** знайдіть **ViewPager** і підключіть до нього **PagerAdapter**:

```
MainActivity extends ActionBarActivity {
 ViewPager pager;
 PagerAdapter pagerAdapter;


     @Override
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         pager = (ViewPager) findViewById(R.id.pager);
         pagerAdapter=new
         MyFragmentPagerAdapter(getSupportFragmentManager());
         pager.setAdapter(pagerAdapter);
     }
    }
```

3. Реалізувати **клас MyFragmentPagerAdapter**:

```
public class MyFragmentPagerAdapter extends
FragmentPagerAdapter {
static final int PAGE_COUNT = 3;
public MyFragmentPagerAdapter(FragmentManager fm) {
super(fm);
}

@Override
public Fragment getItem(int i) {
switch (i){
case 0: return new Fragment1();
case 1: return new Fragment2();
case 2: return new Fragment3();
default: return null;
}
}
@Override
public int getCount() {
return PAGE_COUNT; } // при необходимости добавляем верхнее
меню вкладок с заголовками



@Override
public CharSequence getPageTitle(int i) {
switch (i){
      case 0: return "Frag1";
      case 1: return "Frag2";
      case 2: return "Frag3";
      default: return null; } } }
```

4. Створення графічних розміток і java-коду фрагментів. Приклад структури проекту з **ViewPager** представлений на рисунку 4.4.

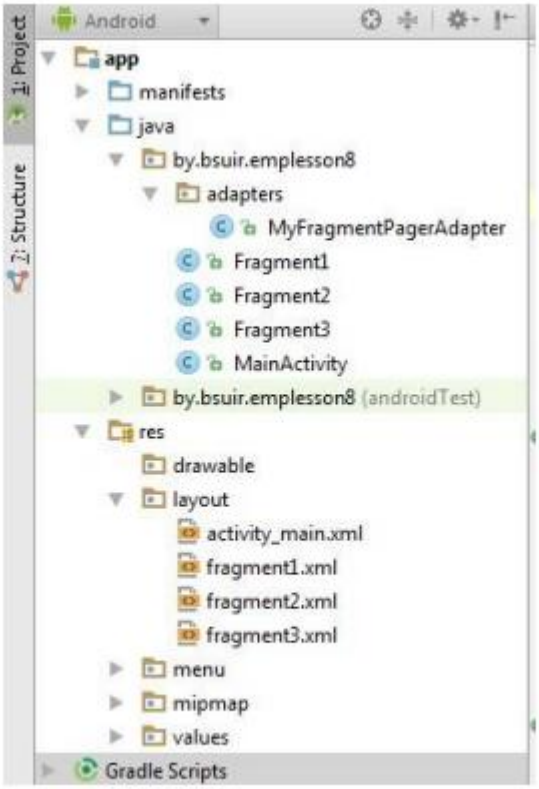


Рисунок 4.4 – Структура проекту з ViewPager

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Реалізуйте наступний додаток:

− у портретній орієнтації перше активіті містить фрагмент А та фрагмент Б, які містять ListView. При виборі пункту з фрагмента А змінюється вміст ListView фрагмента Б. При виборі пункту з фрагмента Б відкривається друге активіті з фрагментом і якимось текстом.

− у ландшафтній орієнтації всі три фрагменти знаходяться на одному екрані, програма працює так само.

Контрольні питання:

1. Що таке фрагменти та навіщо вони потрібні?
2. Чи опишіть основні сценарії використання фрагментів?
3. Чим фрагменти відрізняються від Activity?
4. Що потрібне для створення фрагмента?
5. Як використовувати фрагмент у Activity?
6. Що таке кваліфікатори?
7. Як дізнатися орієнтацію екрана?
8. Який віджет використовується для створення мультиекранних вікон?

ПЕРЕЛІК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Офіційна документація для розробників під ОС Android. URL: https://developer.android.com/docs.
2. Android Tutorial. URL: https://www.tutorialspoint.com/android/index.htm.
3. Аналіз методів і технологій розроблення мобільних додатків для платформи Android : навчальний посібник [Електронний ресурс] / А. О. Поляков, В. М. Федорченко, О. В. Шматко. – Харків : ХНЕУ ім. С. Кузнеця, 2017. – 286 с. ISBN 978-966-676-698-7

4. Аналіз методів і технологій розробки мобільних додатків для платформи Android : навч. посіб. / О. В. Шматко, А. О. Поляков, В. М. Федорченко. – Харків : НТУ «ХПІ», 2018. – 284 с. ISBN 978-966-000-000-0
5. John Horton. Android Programming for Beginners: Build in-depth, full-featured Android 9 Pie apps starting from zero programming experience, 2nd Edition.