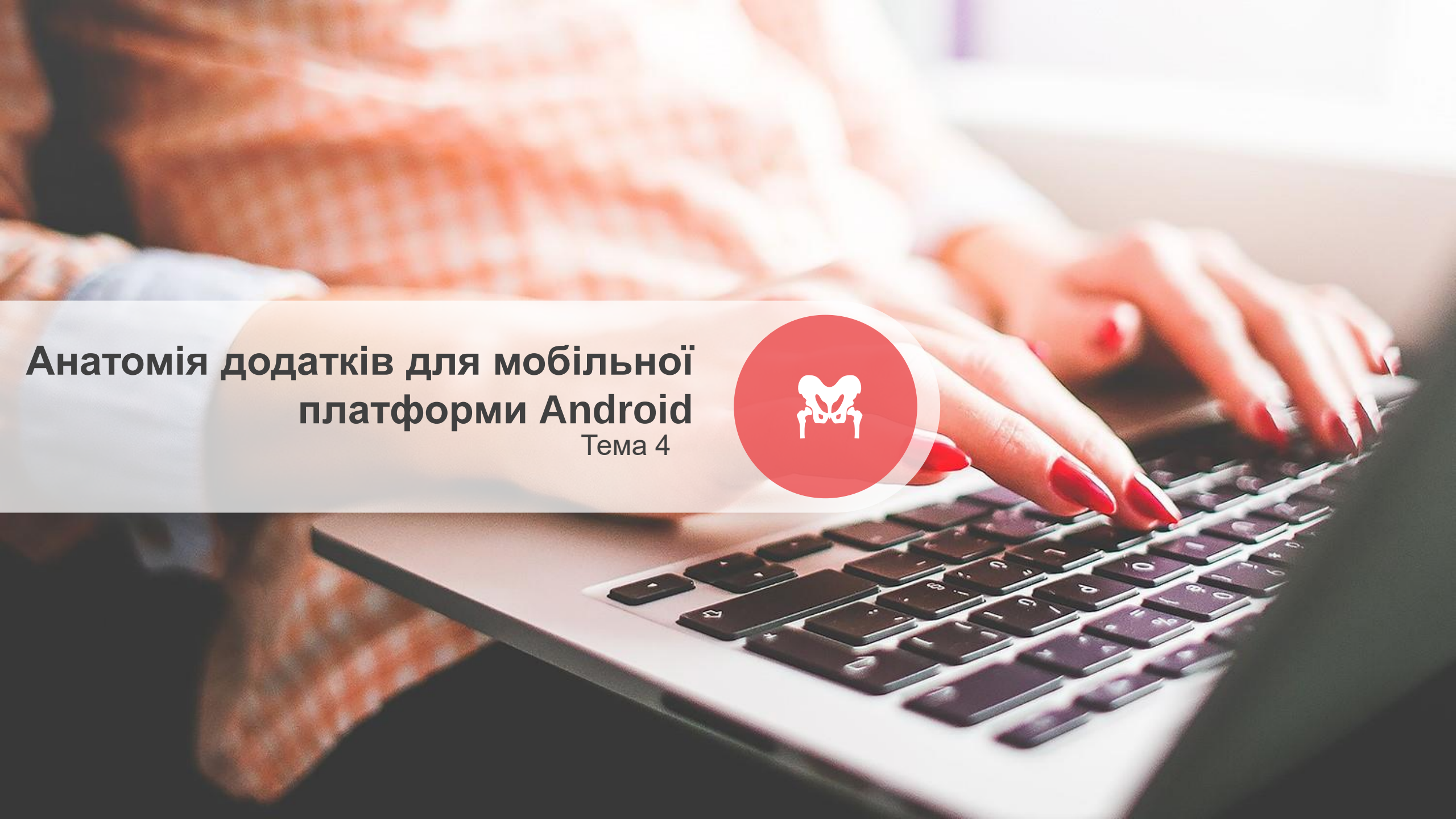
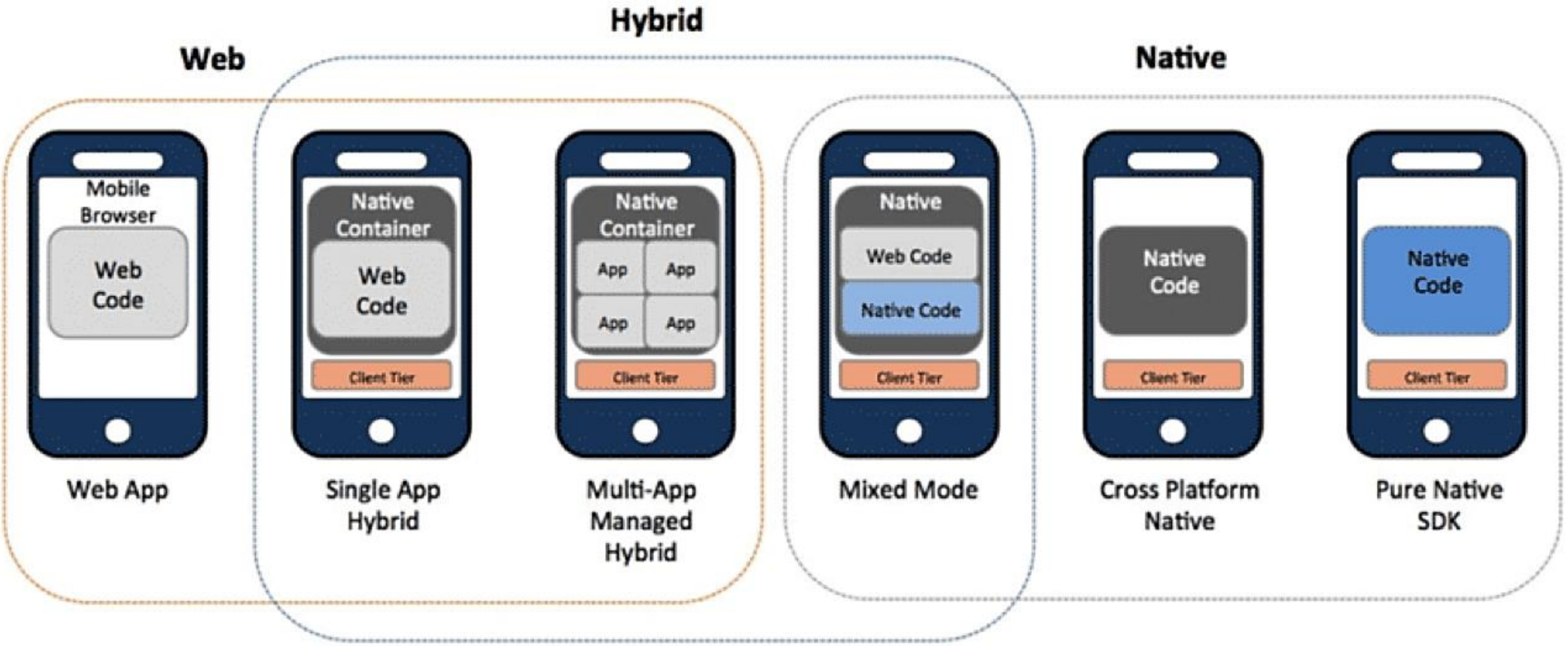
Анатомія додатків для мобільної платформи Android

Тема 4



Класифікація додатків

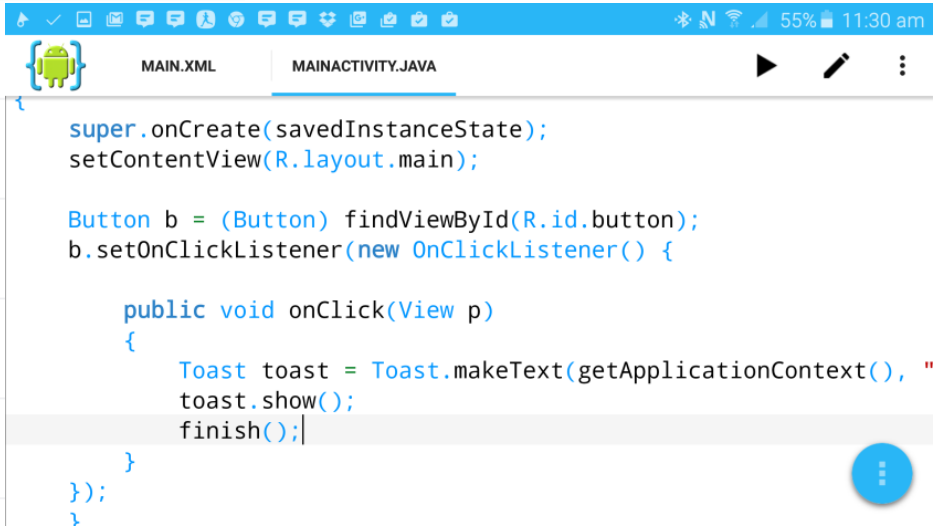
Відповідно до інструментів розробки



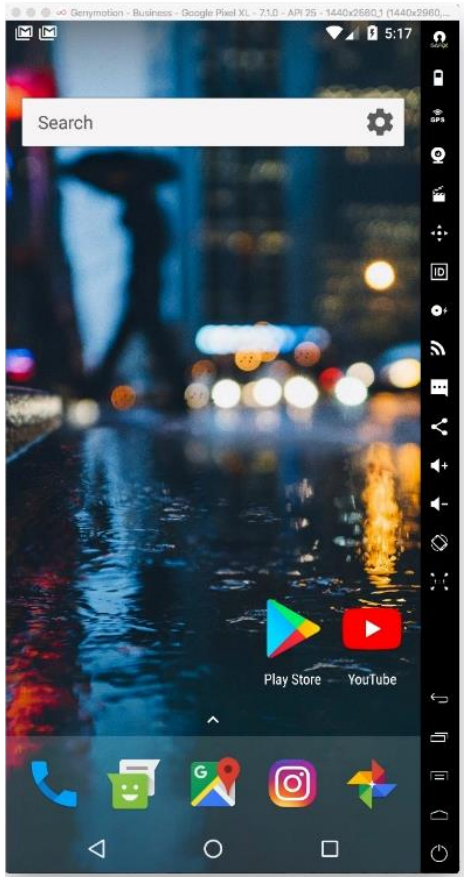
Інструменти розробки нативних додатків

Створення Android-додатків

92		IntelliJ IDEA
89		Android Studio
--		NetBeans IDE
70		Xamarin.Android
--		AIDE
--		JetBrains Rider
--		Komodo IDE
--		RAD studio
--		Eclipse Che
--		Eclipse + Android Development Tools plugin
--		QTrace



GENYMOTION
By Genymobile



Інструменти розробки нативних додатків

Тестування інтерфейсу Android-додатків

Robotium

Дозволяє створювати тестові сценарії для функціонального, системного та поведін-кового тестування.

JAR-файл,

Тестування сірого ящика

UI TESTING FOR ANDROID
espresso

Не підтримує WebView

Тестування білого ящика

Android
uiautomator

Не підтримує WebView, постачається разом з A ndroid SDK.

Тестування чорного ящика

appium
+ Python

Python використовується д ля написання скриптів.

Тестування сірого ящика

Інструменти розробки нативних додатків

Модульне та стрес-тестування Android-додатків

Monkey

- No need source code
- Unusual bugs detection
- Low cost

Testdroid

- Quick testing result
- It takes screenshots
- Supports many devices

Monkey Runner

- A few testing types
- Flexible framework

Інструменти розробки нативних додатків

Модульне та стрес-тестування Android-додатків

Scirocco

- Reports with screenshots
- Quick testing process
- Solid test case is written quickly

Roboelectric

- Business logic testing
- Efficient for refactoring

TestFairy

- Testing video record
- Memorizes technical features of devices

Інструменти розробки нативних додатків

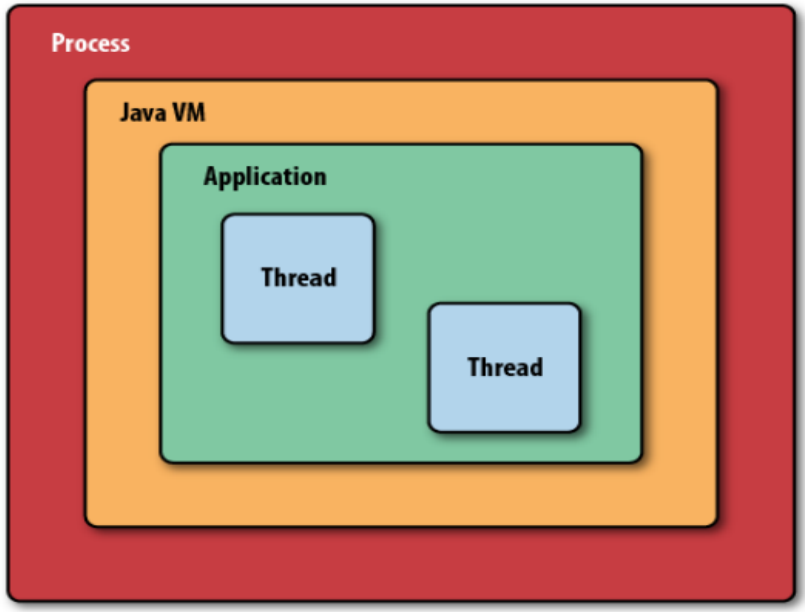
Середовища розробки iOS-додатків

XCode – офіційна IDE для мови Swift від Apple, найбільш потужна платформа розробки для iPhone, iPad, Mac, Apple Watch and Apple TV.
AppCode – середовище розробки від JetBrains для iOS та OS X додатків, має 30-денний пробний період, підтримує як Objective-C, так і Swift.
Code Runner – дозволяє працювати не тільки з мовою Swift, коштує всього $14.99, також доступна демо-версія. Володіє великим набором шаблонів коду.

http://www.businessofapps.com/guide/ios-development-tools/

Концептуальна схема роботи Java-додатку

Точки входу в додаток



Якщо використовуються графічні фреймворки на зразок S wing, для інтерфейсу системи знадобиться ще один потік.

Точка входу класичного Java-застосунку

```
class SillyApp {
        public static void main(String[] args) {
                System. out. println("Hello World!");
        }
}
```

Точка входу для найпростішого сервлету

```
public class DemoServlet extends HttpServlet {
   public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException {
      res.setContentType("text/html"); //setting the content type
      PrintWriter pw=res.getWriter(); //get the stream to write the data
      //writing html in the stream
      pw.println("<html><body>"); pw.println("Welcome to servlet"); pw.println("</body></html>");
      pw.close(); //closing the stream
   }
}
```

Точки входу Android-додатків

Додатки, написані для Android, дотримуються схожого на 2й варіант підходу

Наслідуються базові класи, що постачаються разом з Android для визначення компонентів застосунку. Крім цього, ви створюєте метадані, що описують породжені класи для Android.

Android представляє підхід з багатьма точками входу

Програми повинні очікувати, що система запустить їх в різних місцях, залежно від того, звідки за-й шов користувач і що бажає робити далі.

Замість ієрархії місць програма створює групи ***компонентів***, які можна запустити ззовні цієї програми.

Наприклад, компонент, що зчитує QR-коди, постачає окрему функцію, яку багато інших додатків мо жуть інтегрувати у власний інтерфейс користувача.

Замість очікування на те, що користувач запустить додаток напряму, компоненти самі викликають один одного

Компоненти Android-додатку

Архітектурна основа стандартного додатку

Активності/ Фрагменти

Activity | Fragment

Активність - будівельна одиниця інтерфейсу в Android, аналог класичної веб-сторінки

Служби

Service

Використовуються для створення фонових задач, які можуть бути активними, але не відображатись.

Постачальники контенту

Content Provider

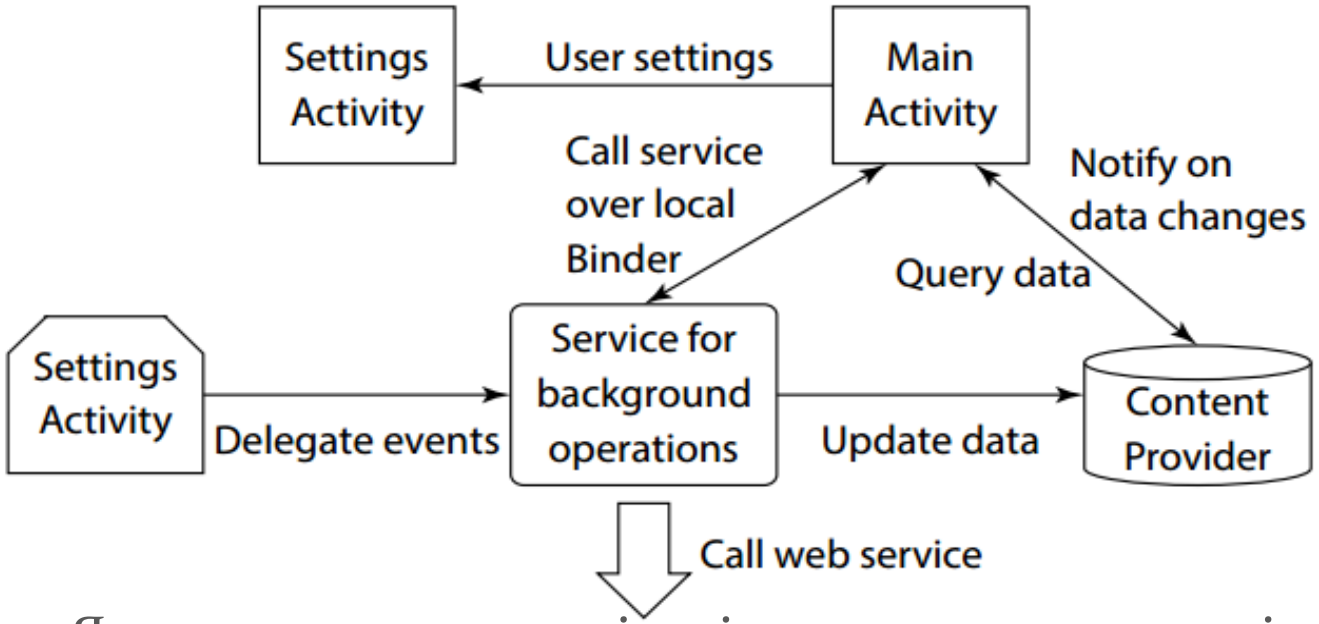
Постачальники контенту управляють доступом до структурованого набору даних.

Широкомовні приймачі / Інтенти

Broadcast Reciever

Середовище виконання Android повідомляє про подію, що трапилась, всім зареєстрованим на неї приймачам.

Приклад простої комбінації компонентів



Зазвичай, розробники додатків для ОС Android починають новий проект, створюючи активність та проектуючи інтерфейс користувача

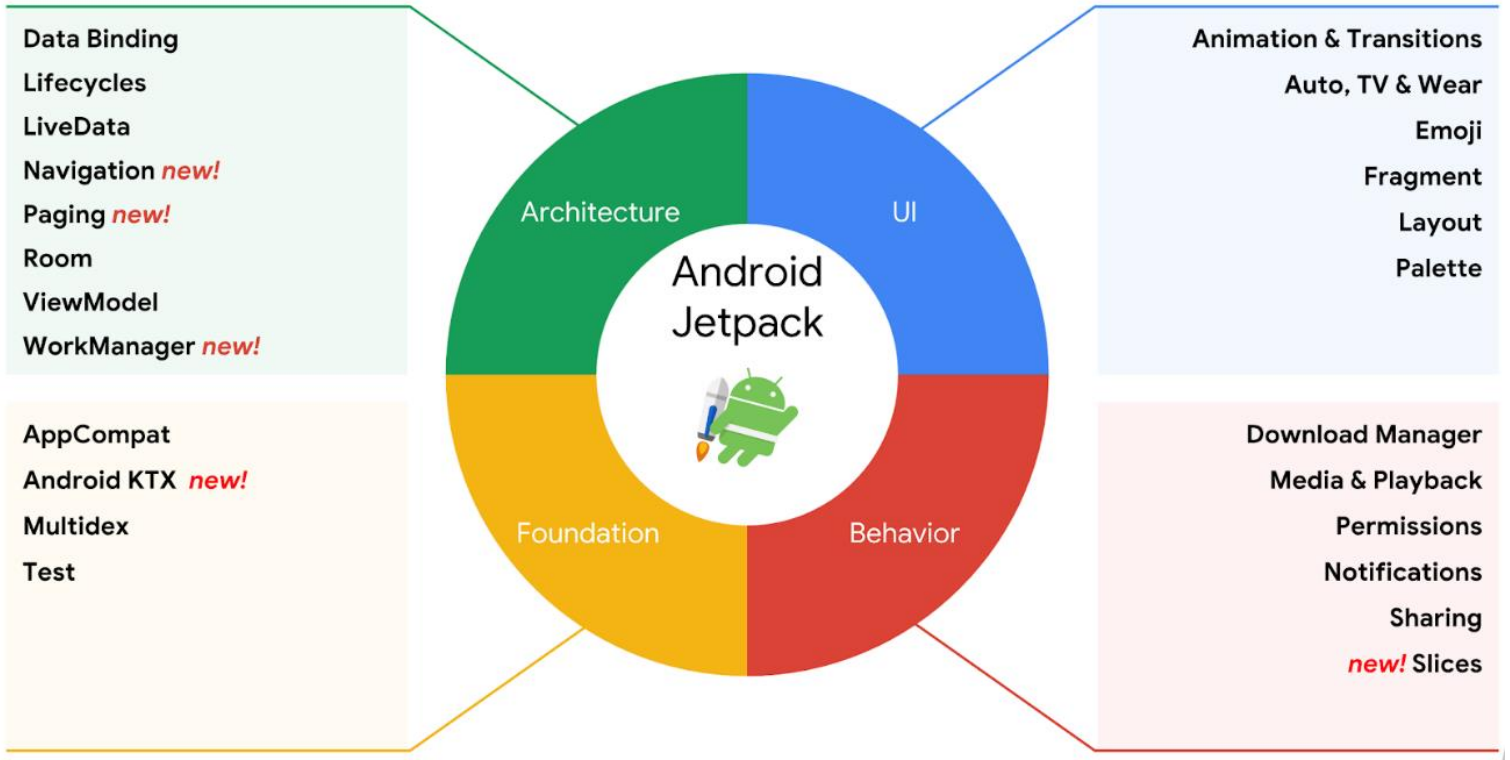
- Якщо в планах є реалізація мережевих викликів, хорошою ідеєю буде їх виділення в окрему службу.
- Якщо потрібно повідомляти про різні системні бродкасти, розумно буде задати широкомовний приймач (BroadcastReciever) та обрати, куди делегувати його події.
- Якщо додаток потребує зберігати дані не у вигляді стандартних пар «ключ-значення», краще створити постачальник контенту, що відповідатиме за цей сценарій.

Контекст

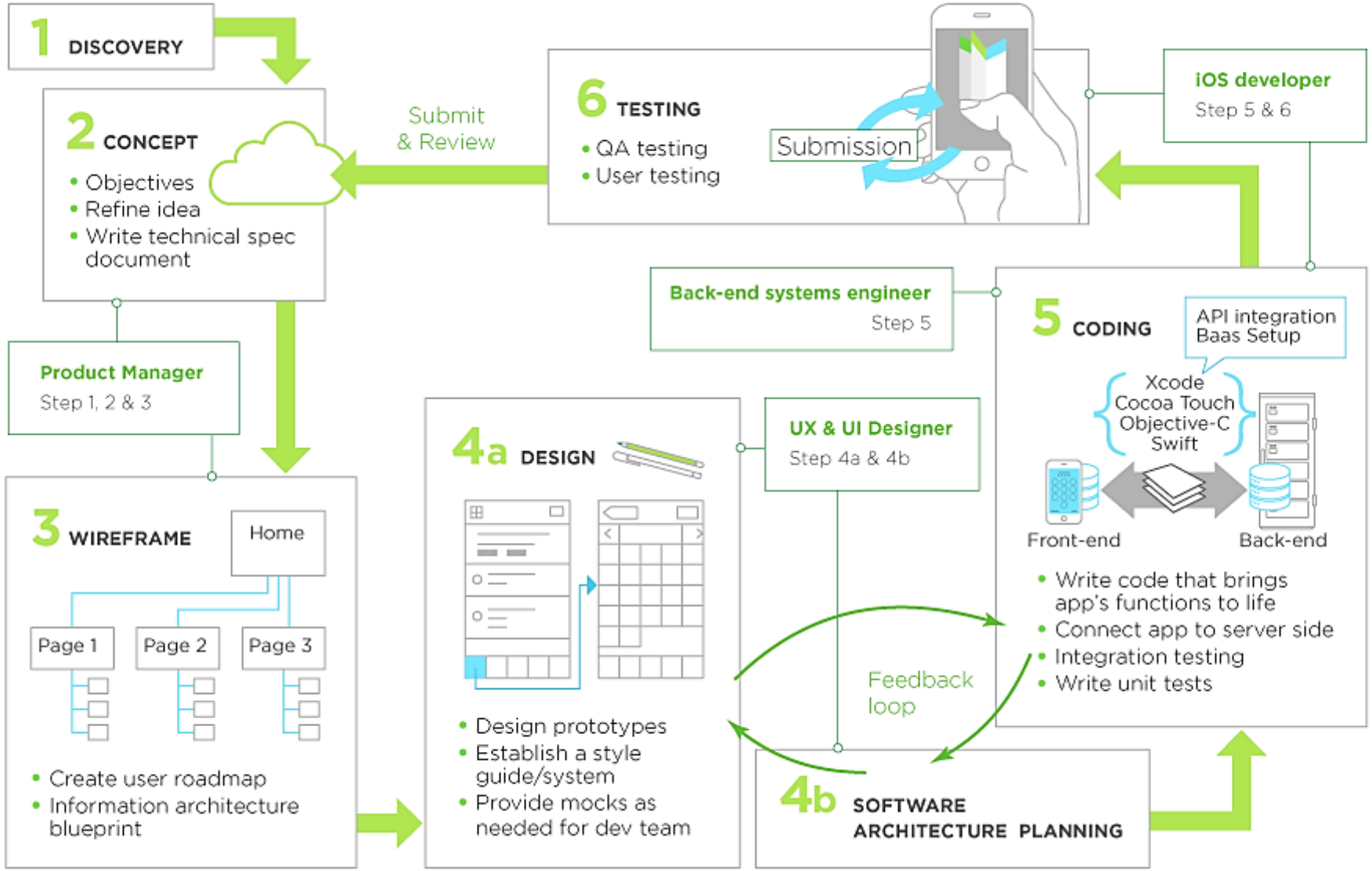
Теми для самоосвіти та самостійного опрацювання

1. ***Android Jetpack.*** Компанія Google підготувала стандартний набір ПЗ для розробника Android-додатків.

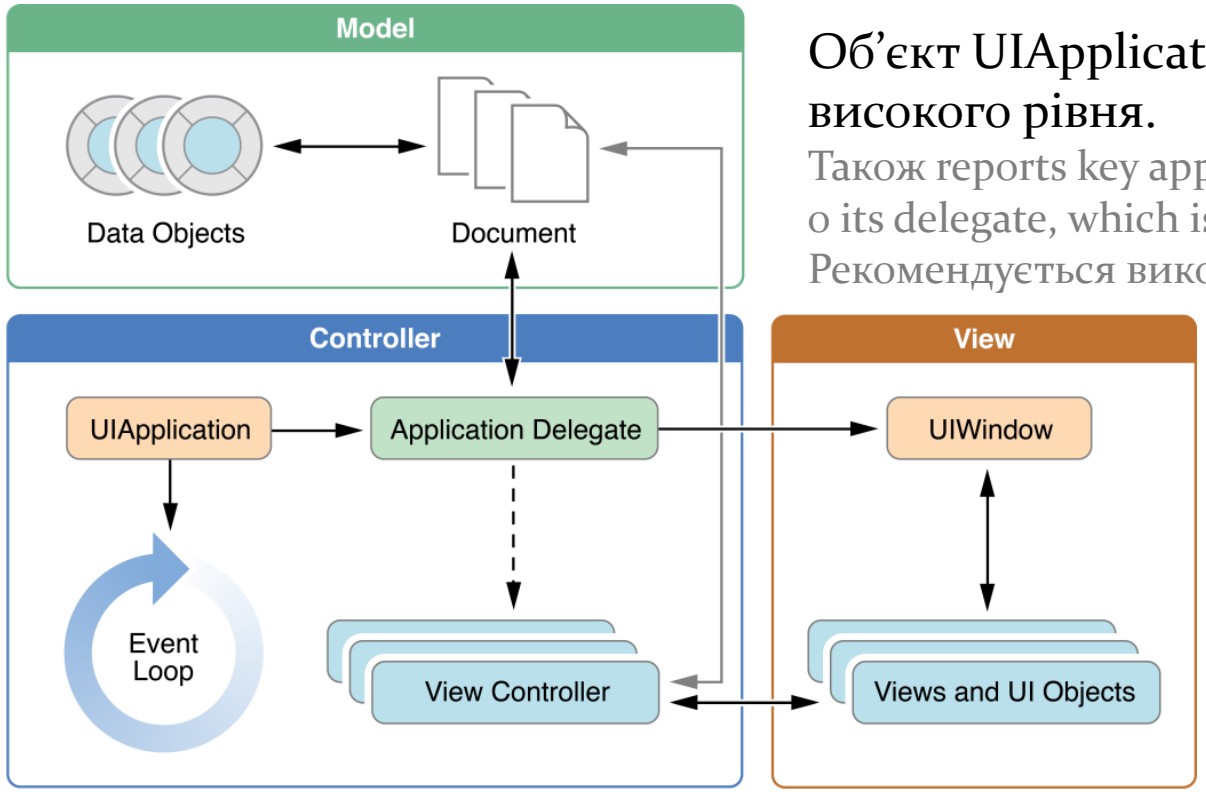
Тема доповіді: Набір бібліотек Android Jetpack: можливості та міграція



Процес розробки мобільного додатку



Структура iOS-додатку



Об'єкт UIApplication управляє подійним циклом та іншими поведінками високого рівня.

Також reports key app transitions and some special events (зокрема, вхідні сповіщення) t o its delegate, which is a custom object you define.

Рекомендується використовувати напряму, без субкласування

Об'єкт App delegate працює в парі з UIApplication, щ об обробляти ініціалізацію додатку, переходи між ст анами та інші високорівневі події.

Це єдиний об'єкт, що гарантовано присутній в кожному д одатку, тому тут часто описують початкові структури даних

Об'єкт UIWindow координує відображення представлень (views) на екрані.

View controller використовують, щоб змінити окремі представлення, а не ціле вікно.

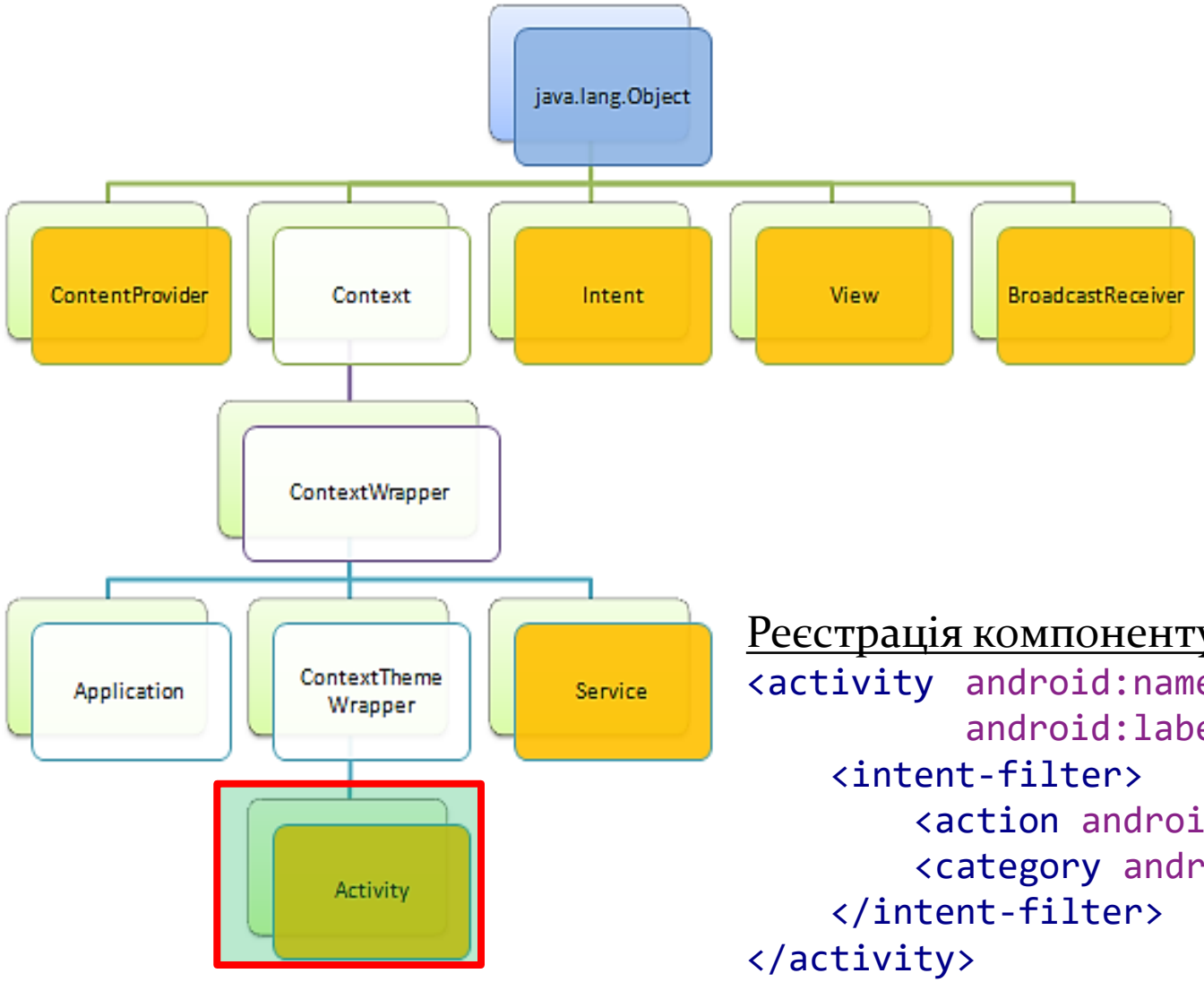
Разом з хостингом представлень, вікна працюють з об'єктом UIApplication, щоб постачати події до представлень та view controllers.

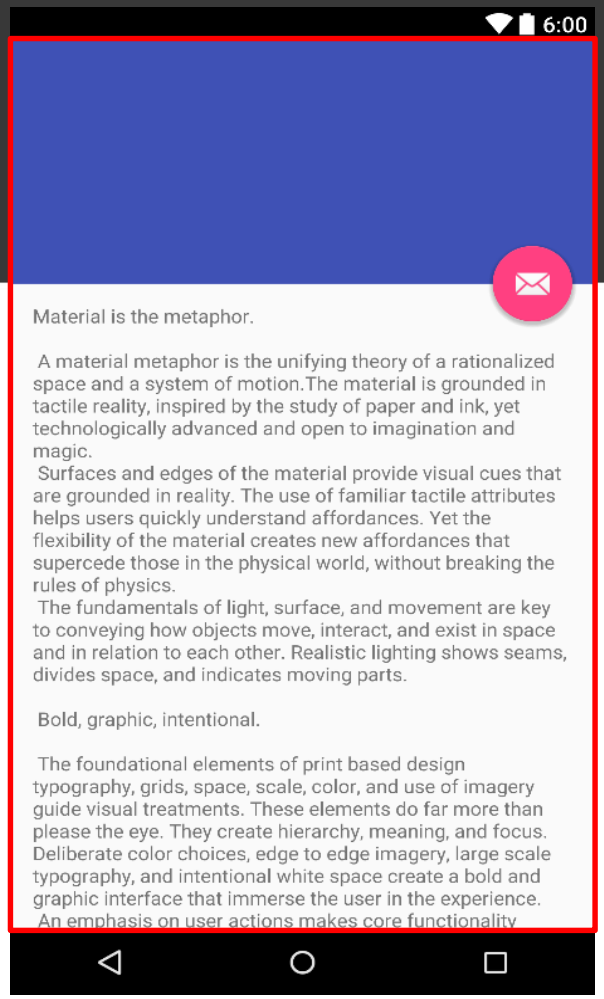
Активності

Побудова та структурування інтерфейсу

Базові компоненти

Активності





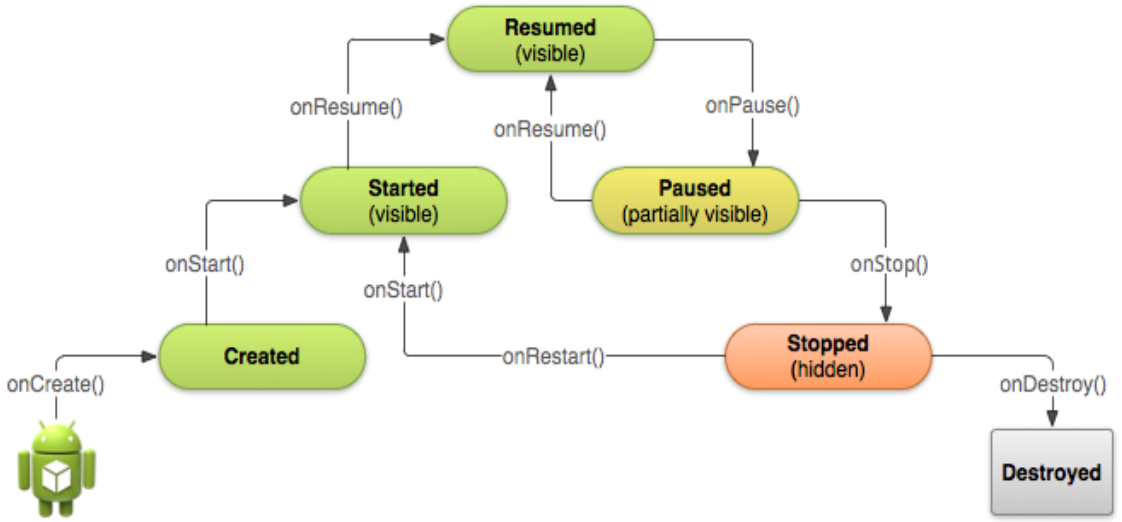
Реєстрація компоненту в файлі маніфесту AndroidManifest.xml

```
<activity  android:name=".MainActivity"
           android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Активність з точки зору дизайну та коду

content_scrolling.xml × ScrollingActivity.java ×

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..."
    tools:context="com.example.admin.myapplication.ScrollingActivity"
    tools:showIn="@layout/activity_scrolling">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:text="Material is the metaphor. A material metaphor is the ..." />

</android.support.v4.widget.NestedScrollView>
```



Коли система запускає активність, виклика
ється конструктор ScrollingActivity, а потім
метод onCreate().

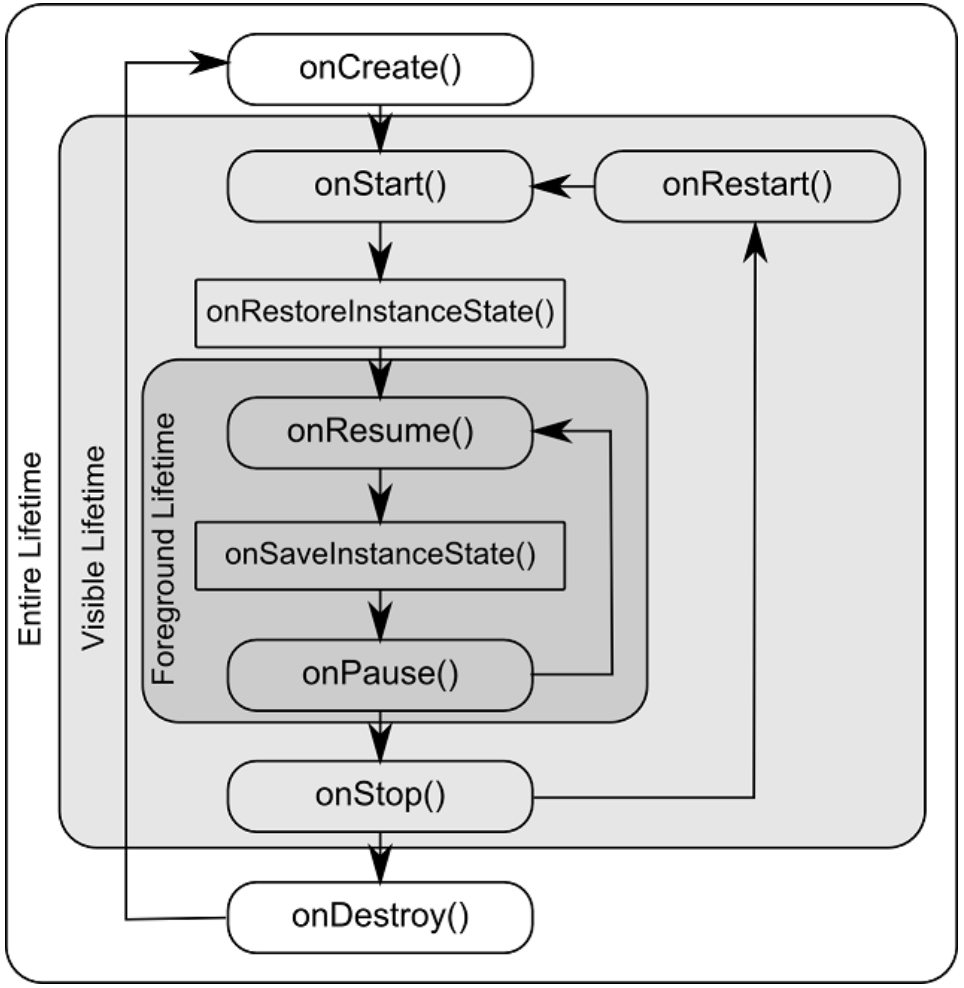
Це спричиняє завантаження та відображення на ек
рані ієрархії віджетів (компонентів інтерфейсу кор
истувача) з файлу content_scrolling.xml (ідентифіка
тор R.layout. content_scrolling).

content_scrolling.xml × ScrollingActivity.java ×

```java
package com.example.admin.myapplication;

import ...

public class ScrollingActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scrolling);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener((view) → {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
        });
    }
```

Методи життєвого циклу активності

У процесі роботи активність може перебувати в одному з трьох станів



Перехід між цими станами відбувається у відповідь на деякі події (events):

- ***Активний стан (запущена активність).*** Активна Activity відображається на екрані, в цей момент з нею можна взаємодіяти.

- ***Призупинений (paused) стан.*** Призупинена активність відображається на екрані, проте не має фокусу. Наприклад, коли з'являється alert dialog, що частково перекриває активність та не дає з нею взаємодіяти, поки не зникне.

- ***Зупинений (stopped) стан.*** Зупинена активність не відображається на екрані, а знаходиться у фоні та, ймовірно, буде перервана системою, якщо пам'яті не вистачатиме. Активність зупиняється, якщо інша Activity виходить на передній план (foreground) та стає активною. Наприклад, коли Вам дзвонять і Ви відповідаєте на дзвінок, запущений до цього додаток зупиняється,

Методи життєвого циклу активності

- **Метод onCreate()**. Повинен бути реалізованим у будь-якому випадку. Зазвичай всередині нього відбувається виклик методу setContentView(). Тут же можуть ініціалізуватись макет інтерфейсу та різного роду змінні. В рамках даного методу можливий запуск фонового потоку (background thread).

- **Метод onResume()**. Типові дії: реєстрація широкомовних приймачів, відновлення (restore) змінних, відбувається перезапуск анімацій

- **Метод OnPause()**. Фіксуються (commit) зміни, зберігаються змінні та налаштування, зупинка анімацій, відміна реєстрації (unregistering) широкомовних приймачів

- **Метод OnStart()**. Реєстрація широкомовних приймачів, реініціалізація станів
- **Метод onRestart()**. Встановлення з'єднань з сервером, виділення ресурсів
- **Метод onStop()**. Розрив з'єднань з сервером, вивільнення ресурсів, відміна реєстрації (unregistering) широкомовних приймачів

Quiz - [D:\Quiz] - [app] - ...\app\src\main\java\com\example\user\quiz\MainActivity.java - Android Studio 2.3.2

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Quiz > app > src > main > java > com > example > user > quiz > MainActivity

Android

app
- manifests
- java
  - com.example.user.quiz
    - MainActivity
  - com.example.user.quiz (androidTest)
  - com.example.user.quiz (test)
- res
- Gradle Scripts

activity_main.xml | MainActivity.java

```
1  package com.example.user.quiz;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
14
```

Android Monitor

Emulator Nexus_5_API_24 Android 7.0, API 24 | com.example.user.quiz (2438)

logcat | Monitors

Verbose | Regex | Show only selected application

```
08-28 05:56:38.888 2500-3328/com.google.android.gms W/PlatformStatsUtil: Could not retrieve Usage & Diagnostics setting. Giving up.
08-28 05:56:40.671 2500-2725/com.google.android.gms I/Icing: Usage reports 0 indexed 0 rejected 0 imm upload false
08-28 05:56:40.734 2500-2725/com.google.android.gms I/Icing: Usage reports 0 indexed 0 rejected 0 imm upload false
08-28 05:56:40.761 2500-2725/com.google.android.gms I/Icing: Usage reports 0 indexed 0 rejected 0 imm upload false
08-28 05:56:40.772 2500-2743/com.google.android.gms I/Icing: Usage reports 0 indexed 0 rejected 0 imm upload false
08-28 05:56:43.988 2500-3328/com.google.android.gms W/PlatformStatsUtil: Could not retrieve Usage & Diagnostics setting. Giving up.
```

Show only selected application
Firebase
No Filters
Edit Filter Configuration

0: Messages | Terminal | 6: Android Monitor | 4: Run | TODO | Event Log | Gradle Console

Gradle build finished in 12s 405ms (a minute ago)

3488:1 CRLF UTF-8 Context: <no context>

Запуск додатку

Android

- app
  - manifests
  - java
    - com.example.user.quiz
      - MainActivity
    - com.example.user.quiz (androidTest)
    - com.example.user.quiz (test)
  - res
- Gradle Scripts

activity_main.xml × | MainActivity.java ×

MainActivity | TAG

```
1  package com.example.user.quiz;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.util.Log;
6
7  public class MainActivity extends AppCompatActivity {
8      private static final String TAG = "MainActivity";
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         Log.d(TAG, "onCreate(Bundle) called");
13         setContentView(R.layout.activity_main);
14     }
15
16     @Override
```

Android Monitor

Emulator Nexus_5_API_24 Android 7.0, API 24 | com.example.user.quiz (8436)

logcat | Monitors | Verbose

```
08-28 06:01:43.772 8436-8436/? D/MainActivity: onCreate(Bundle) called
08-28 06:01:43.956 8436-8436/? D/MainActivity: onStart() called
08-28 06:01:43.960 8436-8436/? D/MainActivity: onResume() called
```

1: Project | 7: Structure | Captures | Favorites

Android

- app
  - manifests
  - java
    - com.example.user.quiz
      - MainActivity
    - com.example.user.quiz (androidTest)
    - com.example.user.quiz (test)
  - res
- Gradle Scripts

activity_main.xml × | MainActivity.java ×

MainActivity | TAG

```
16      @Override
17      public void onStart() {
18          super.onStart();
19          Log.d(TAG, "onStart() called");
20      }
21      @Override
22      public void onPause() {
23          super.onPause();
24          Log.d(TAG, "onPause() called");
25      }
26      @Override
27      public void onResume() {
28          super.onResume();
29          Log.d(TAG, "onResume() called");
30      }
31      @Override
32      public void onStop() {
33          super.onStop();
34          Log.d(TAG, "onStop() called");
35      }
```

Android Monitor

Emulator Nexus_5_API_24 Android 7.0, API 24 | com.example.user.quiz (8436)

logcat | Monitors

```
08-28 06:01:43.772 8436-8436/? D/MainActivity: onCreate(Bundle) called
08-28 06:01:43.956 8436-8436/? D/MainActivity: onStart() called
08-28 06:01:43.960 8436-8436/? D/MainActivity: onResume() called
08-28 06:04:53.076 8436-8436/com.example.user.quiz D/MainActivity: onPause() called
08-28 06:04:53.638 8436-8436/com.example.user.quiz D/MainActivity: onStop() called
08-28 06:04:53.639 8436-8436/com.example.user.quiz D/MainActivity: onDestroy() called
```

Після натиснення кнопки BACK

Android

- app
  - manifests
  - java
    - com.example.user.quiz
      - MainActivity
    - com.example.user.quiz (androidTest)
    - com.example.user.quiz (test)
  - res
- Gradle Scripts

activity_main.xml × | MainActivity.java ×

MainActivity | onResume()

```
22    public void onPause() {
23        super.onPause();
24        Log.d(TAG, "onPause() called");
25    }
26    @Override
27    public void onResume() {
28        super.onResume();
29        Log.d(TAG, "onResume() called");
30    }
31    @Override
32    public void onStop() {
33        super.onStop();
34        Log.d(TAG, "onStop() called");
35    }
36    @Override
37    public void onDestroy() {
38        super.onDestroy();
39        Log.d(TAG, "onDestroy() called");
40    }
41 }
```

Android Monitor

Emulator Nexus_5_API_24 Android 7.0, API 24 | com.example.user.quiz (20641)

logcat | Monitors

```
08-28 06:15:29.415 20641-20641/com.example.user.quiz D/MainActivity: onCreate(Bundle) called
08-28 06:15:29.539 20641-20641/com.example.user.quiz D/MainActivity: onStart() called
08-28 06:15:29.541 20641-20641/com.example.user.quiz D/MainActivity: onResume() called
08-28 06:16:14.161 20641-20641/com.example.user.quiz D/MainActivity: onPause() called
08-28 06:16:14.242 20641-20641/com.example.user.quiz D/MainActivity: onStop() called
```

Після натиснення на кнопку HOME

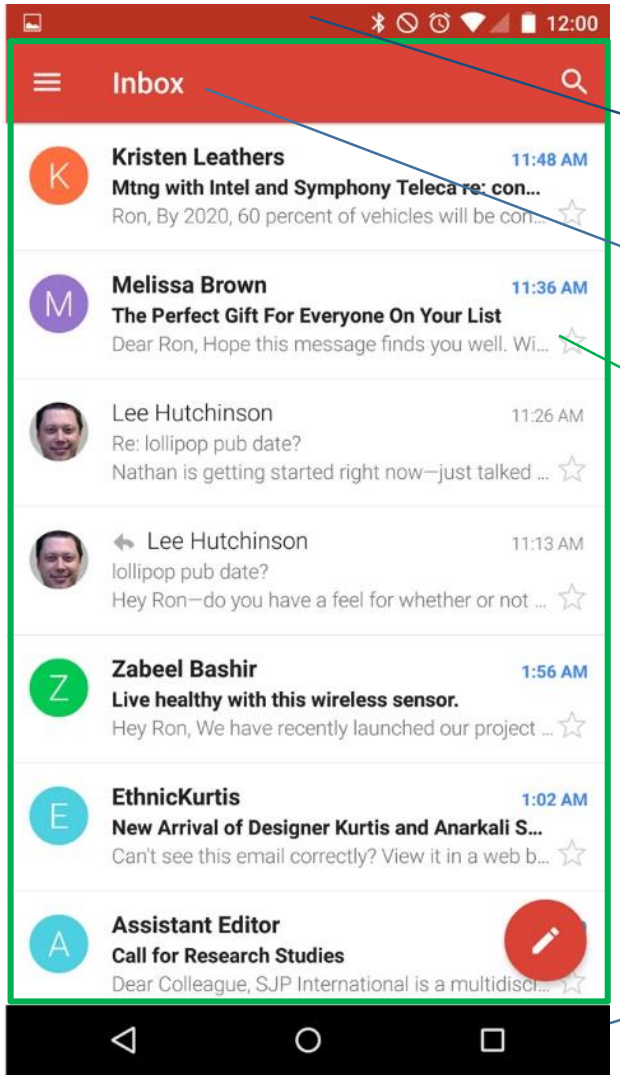
Після повороту пристрою

logcat | Monitors

```
08-28 07:07:05.616 1318-1318/? D/MainActivity: onCreate(Bundle) called
08-28 07:07:05.751 1318-1318/? D/MainActivity: onStart() called
08-28 07:07:05.755 1318-1318/? D/MainActivity: onResume() called
08-28 07:07:16.265 1318-1318/com.example.user.quiz D/MainActivity: onPause() called
08-28 07:07:16.272 1318-1318/com.example.user.quiz D/MainActivity: onStop() called
08-28 07:07:16.272 1318-1318/com.example.user.quiz D/MainActivity: onDestroy() called
08-28 07:07:16.295 1318-1318/com.example.user.quiz D/MainActivity: onCreate(Bundle) called
08-28 07:07:16.308 1318-1318/com.example.user.quiz D/MainActivity: onStart() called
08-28 07:07:16.309 1318-1318/com.example.user.quiz D/MainActivity: onResume() called
```

Структура інтерфейсу ОС Android

Крім активності зазвичай відображаються дві основні панелі:



Панель статусу (Status Bar)

Панель дій (action bar, toolbar)

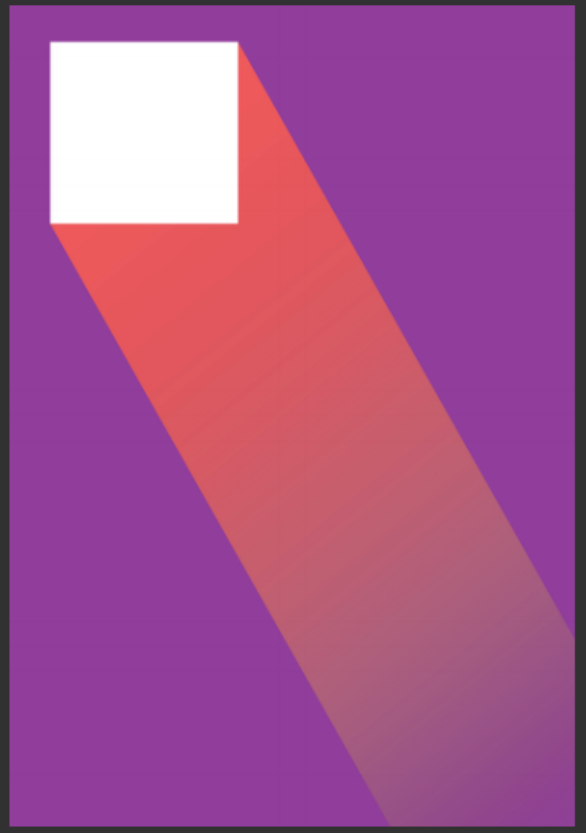
Область активності

Панель навігації (Navigation Bar)

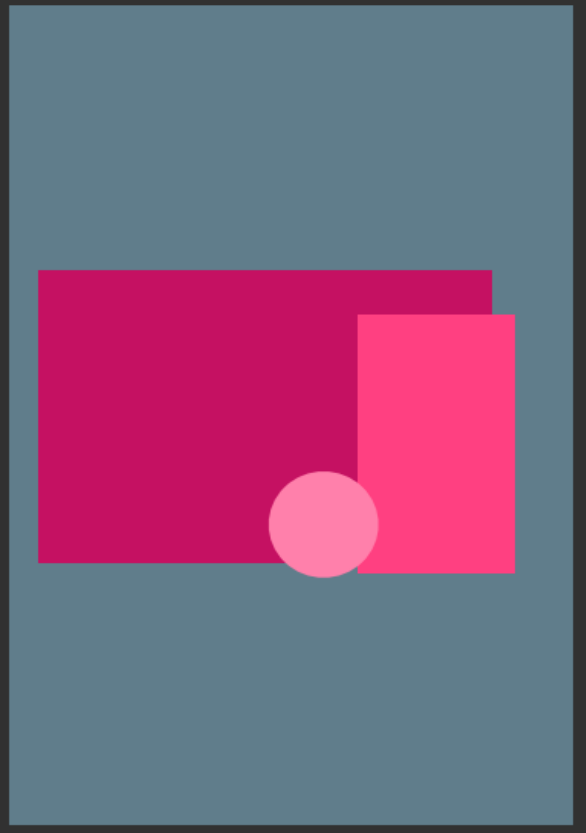
Принципи матеріального дизайну

Tangible surfaces

Print-like design



Meaningful motion

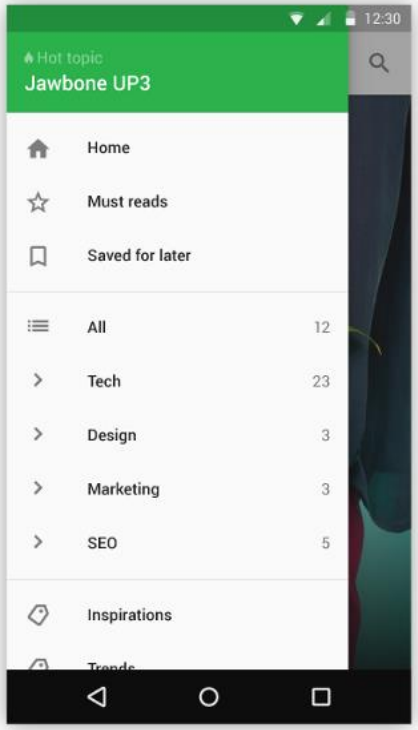


Adaptive design

Концепція поверхні

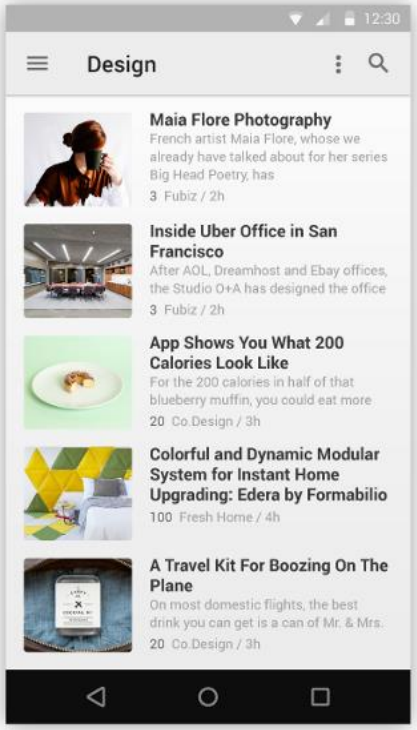
Матеріальний дизайн

Material Design

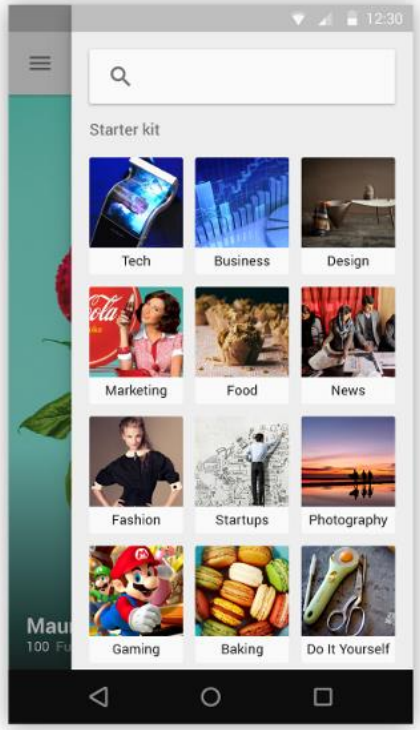


Navigation panel

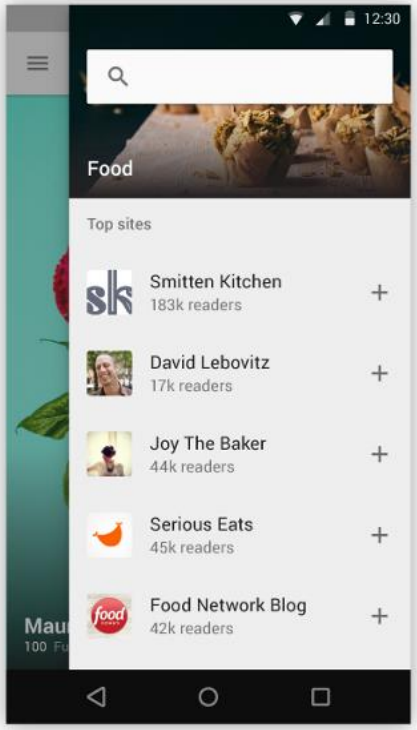
Cards view



Magazine view



Explore panel

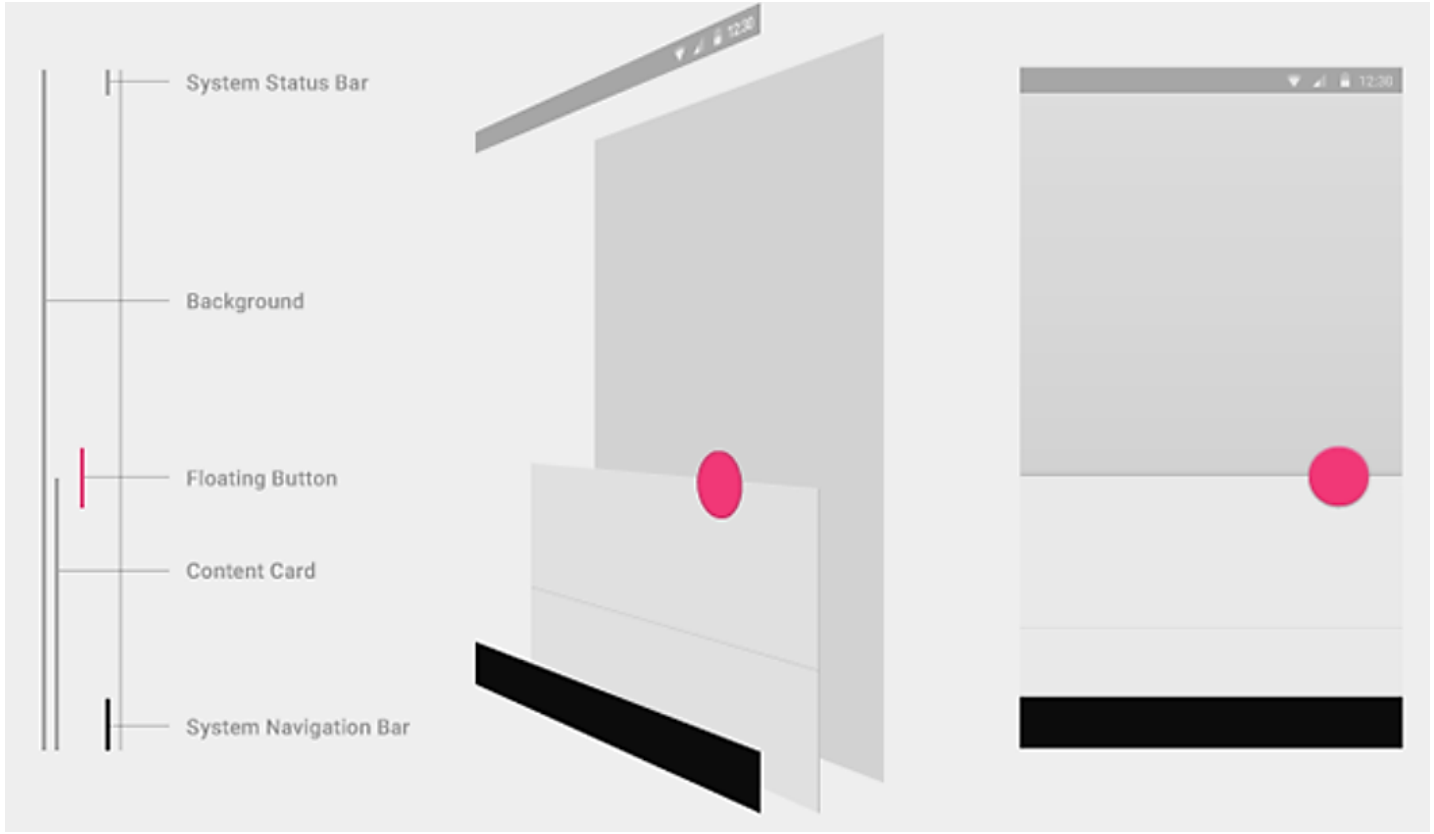


Feed search results

Поверхня – це контейнер для контенту

Концепція поверхні

Деревовидна структура інтерфейсу



Уявимо, що інтерфейс побудовано зі шматків цифрового паперу, які називають поверхнями (surfaces)

Імітація тривимірності відбувається за допомогою тіней

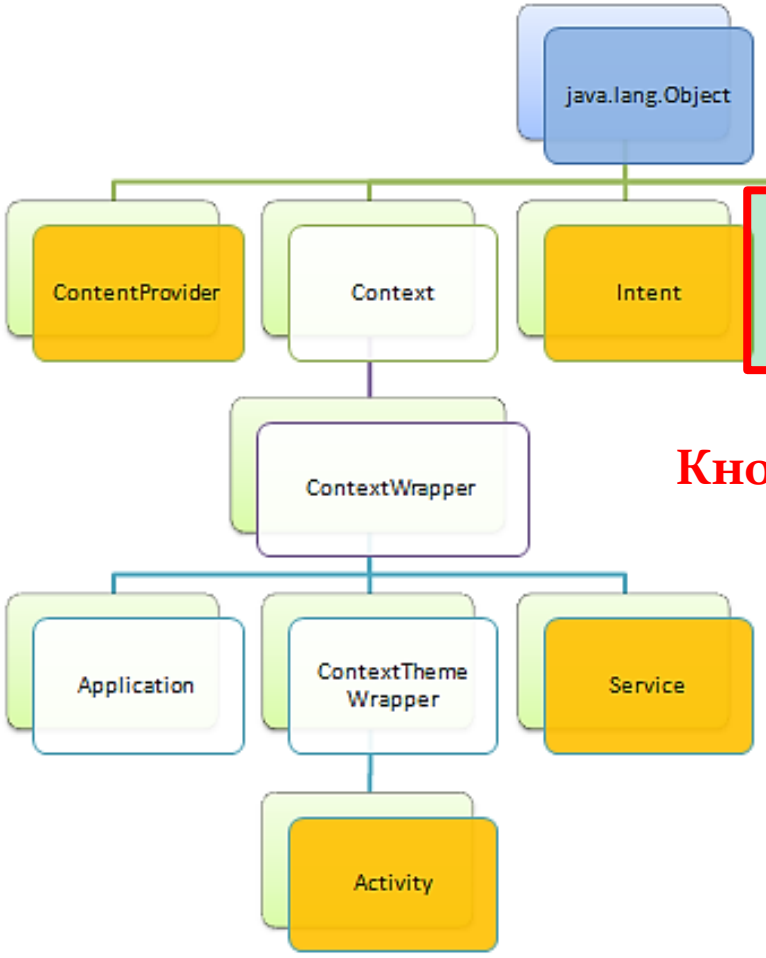
Задавши висоту (elevation), Android згенерує тінь

Контейнери (диспетчери компонувань) – спосіб структурної організації багатьох від жетів.

Контейнери містять дочірні елементи: від жети або інші контейнери.

Представлення (View, віджети)

елементи управління графічного інтерфейсу



↳ android.widget.TextView

Кнопка ↳ android.widget.Button

Частіше всього описуються за допомогою X ML, проте можливе створення лише в Java-к оді.

XML

```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"/>
```

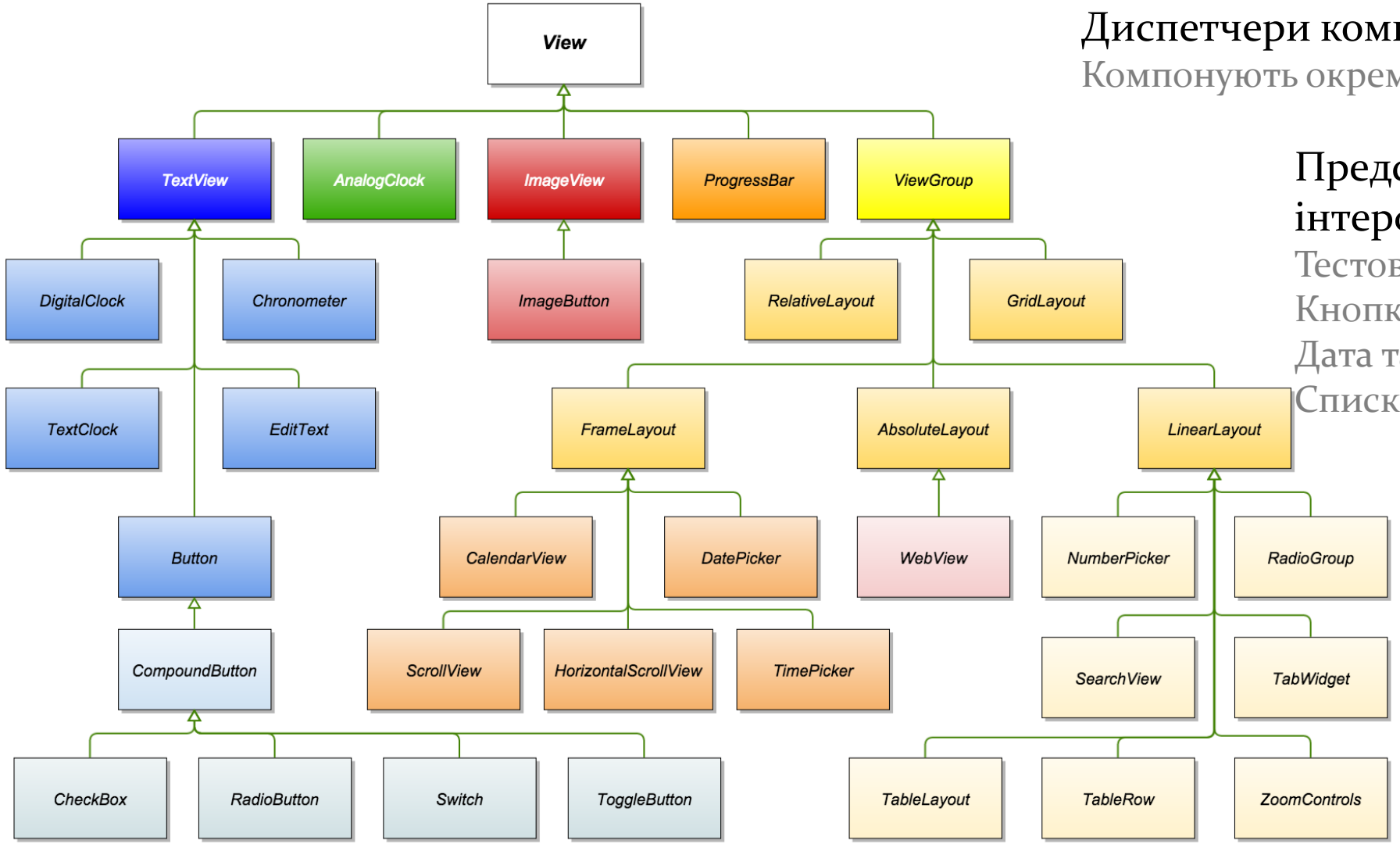
Java

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Кожен віджет володіє набором атрибутів. Тут
android:id – ідентифікатор представлення (віджету)
android:layout_width, android:layout_height – розміри кнопки
android:text – текст на кнопці

Клас View

Архітектура та базові представлення



Диспетчери компонувань (layouts)

Компонують окремі представлення (віджети)

Представлення - окремі елементи інтерфейсу

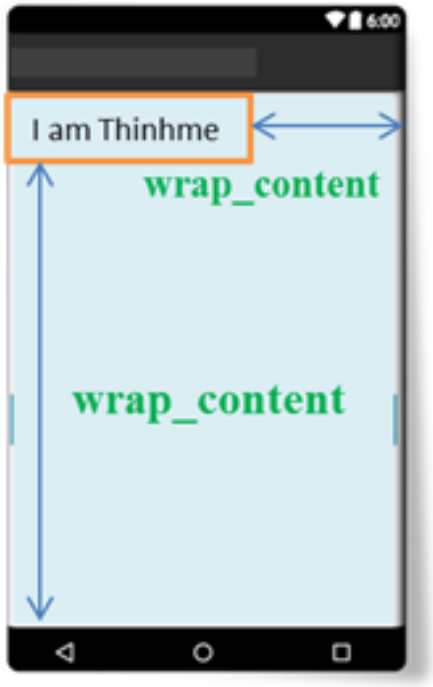
Тестові елементи (TextView, EditText)
Кнопки (Button, RadioButton та ін.)
Дата та час
Спискові віджети та ін.

Атрибути віджетів

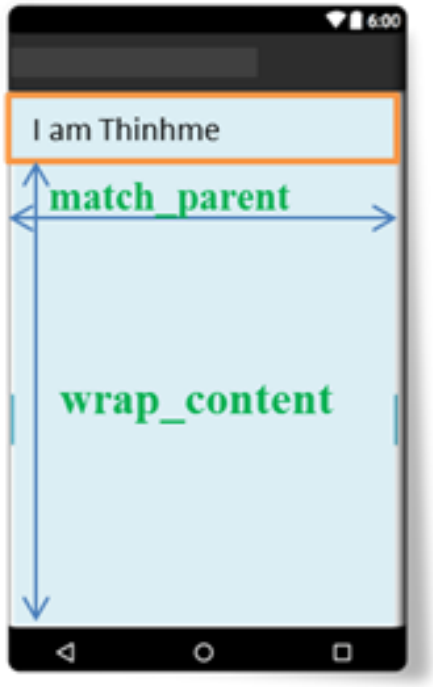
Розміри

Розмір може диктуватись:

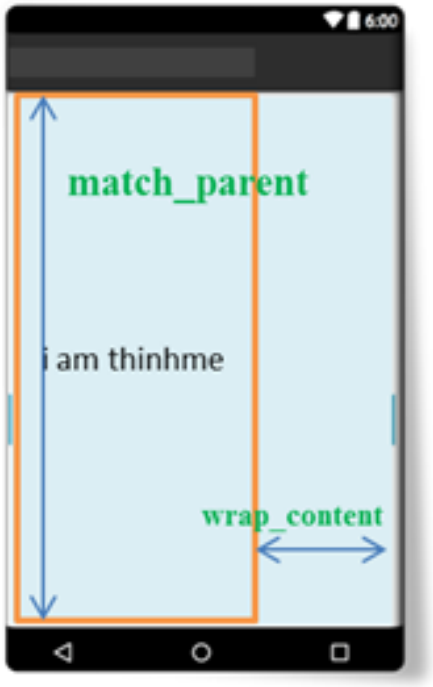
- внутрішнім наповненням (розмір TextView – визначатися текстом всередині нього, wrap_content)
- розміром батьківського контейнеру (match_parent)



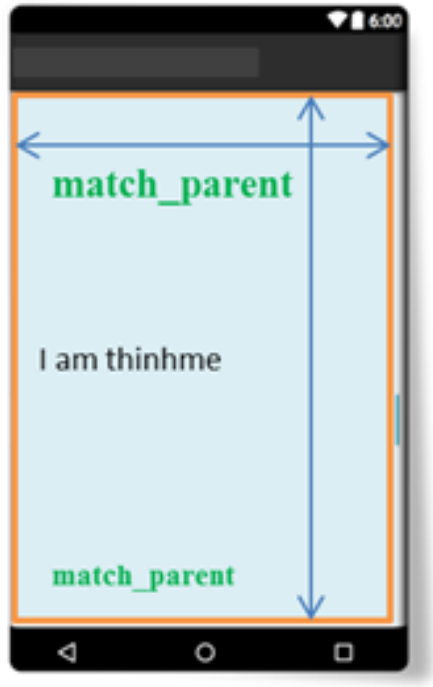
android:layout_width = “wrap_content”
android:layout_height = “wrap_content”



android:layout_width = “match_parent”
android:layout_height = “wrap_content”



android:layout_width = “wrap_content”
android:layout_height = “match_parent”



android:layout_width = “match_parent”
android:layout_height = “match_parent”

Одиниці вимірювання розміру

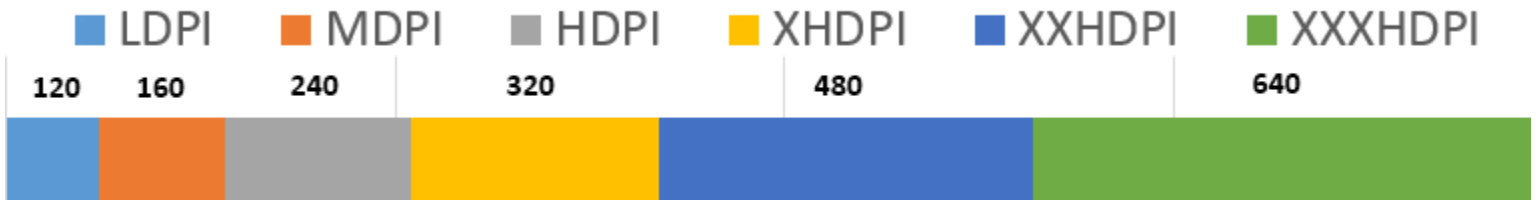
На практиці використовують dp, sp

Dimension	Description	Units / Physical Inch	Density Independent	Same Physical Size on Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density independent pixels	~160	Yes	No
sp	Scale independent pixels	~160	Yes	No

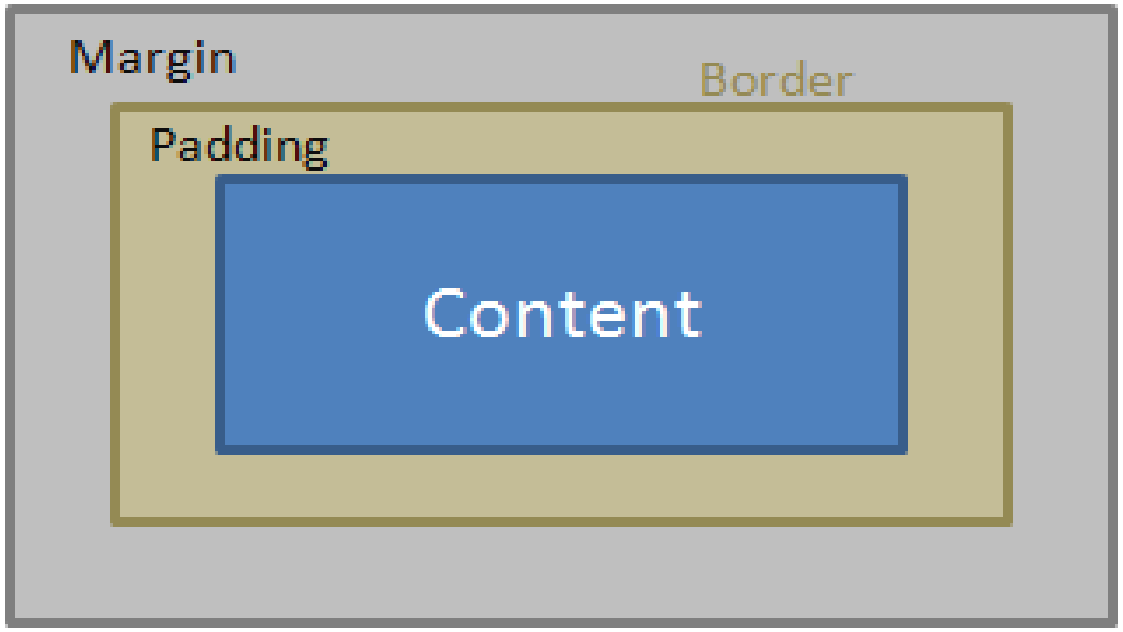
Dimension	Pixels on xhdpi (320dpi) display	Units on xhdpi (320dpi) display
wrap_content	200 px	200 px
dp	200 px	200 px * 160 dpi / 320 dpi = 100 dp
in	200 px	200 px / 320 dpi = 0.625 in
mm	200 px	200 px / 320 dpi * 25.4 mm/in = 15.875 mm
pt	200 px	200 px / 320 dpi * 72 pt/in = 45pt

Розміри віджетів. Щільність екрану

Screen resolution	dpi	Pixel ratio	Image size (pixels)
xxxhdpi	640	4.0	400 x 400
xxhdpi	480	3.0	300 x 300
xhdpi	320	2.0	200 x 200
hdpi	240	1.5	150 x 150
mdpi	160	1.0	100 x 100



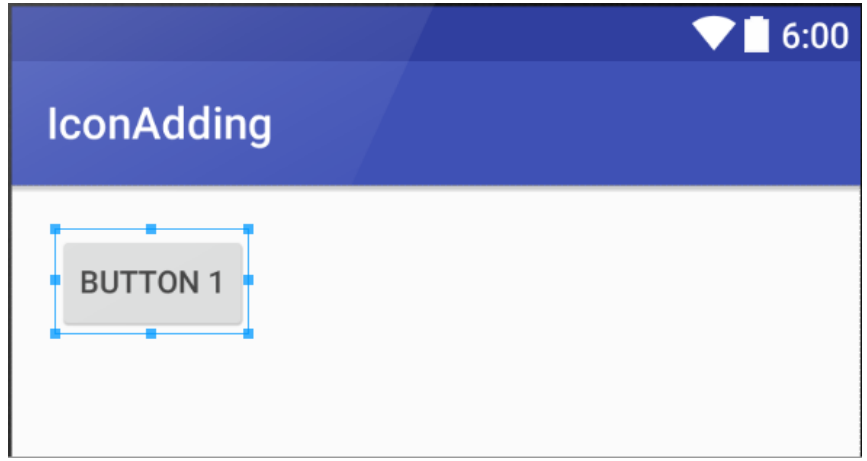
Margins та Padding



Margins – забезпечують відокремлення між в іджетами та іншими суміжними сутностями (іншими віджетами, краями екрану тощо).

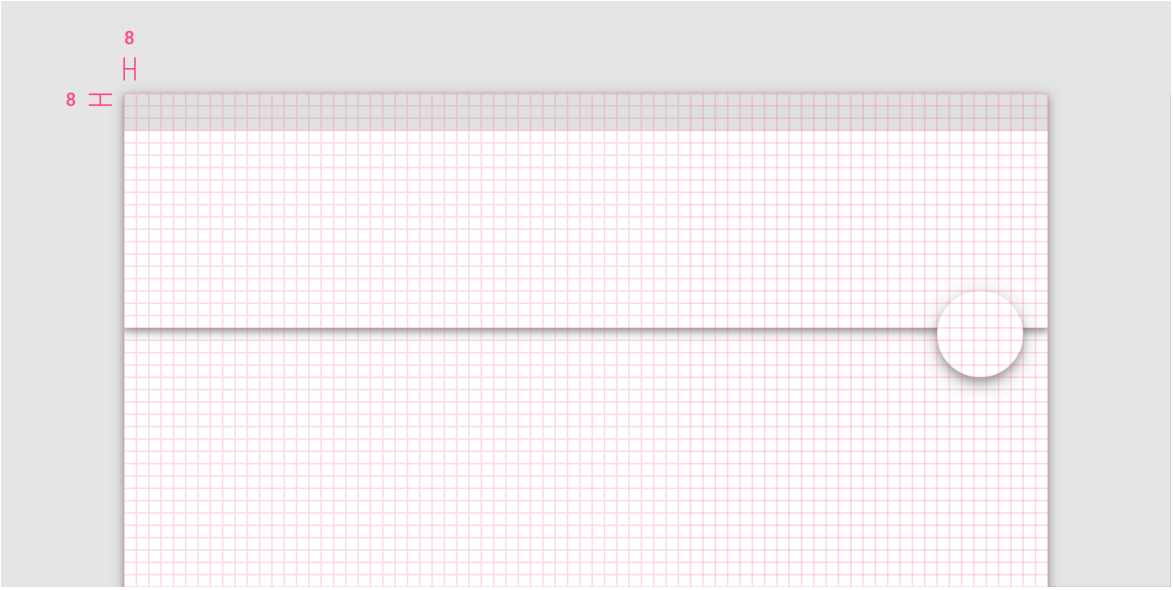
Paddings – забезпечують відокремлення кон тенту віджету від його полів. Зазвичай викор истовуються для віджетів з деяким фоном, н априклад, кнопок.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="4dp"
    android:padding="8dp"
    android:text="Button 1"/>
```



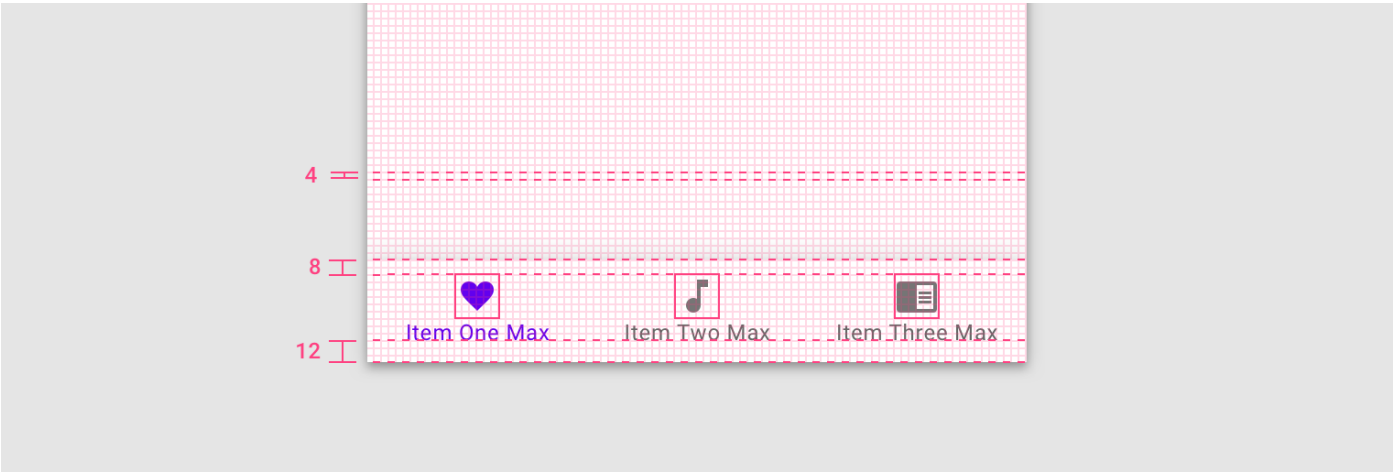
Позиціонування представлень

Базові лінії (baseline)



Сітка 8dp x 8dp

Усі компоненти вирівнюються по сітці з базови ми лініями 8dp для мобільних, планшетних та н астільних додатків.



Сітка 4dp

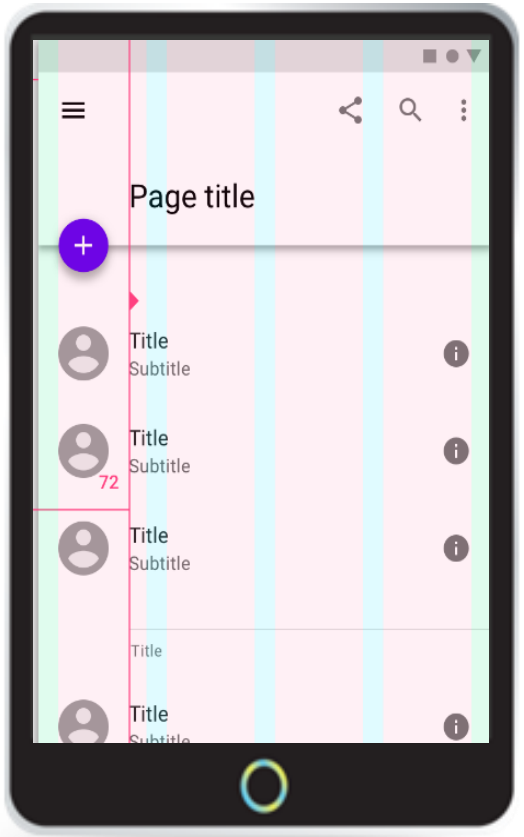
Іконки, шрифти та деякі елементи представлень можуть вирівнюватис ь по 4dp-сітці.

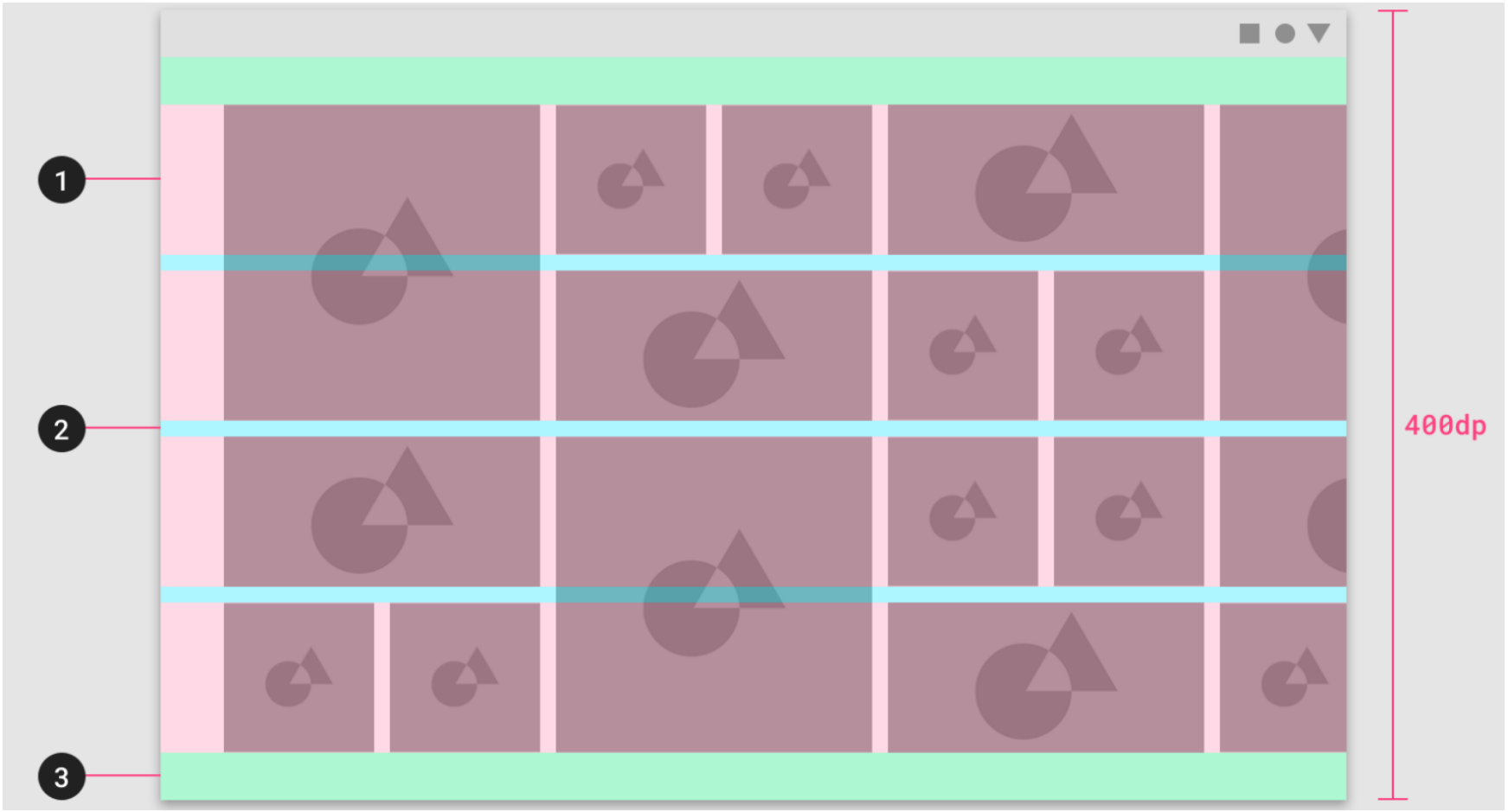
Допускається і базова лінія в 4dp.

Позиціонування представлень

Ключові лінії (keyline) та респонсивне макетування

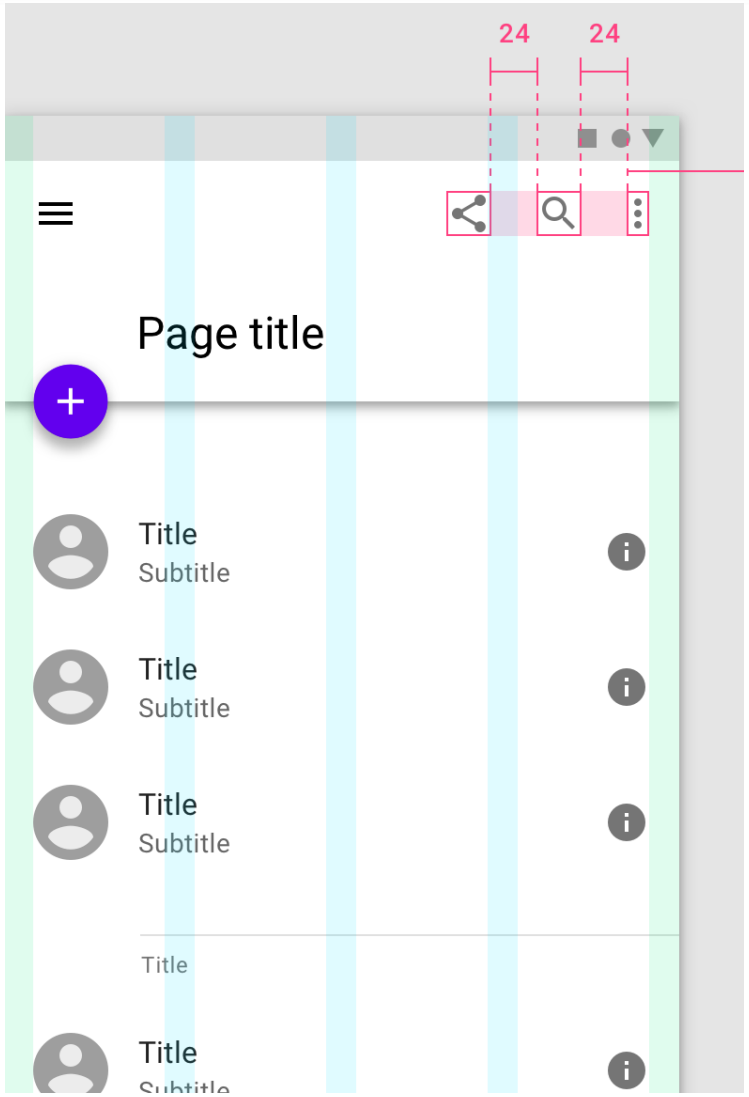
Макетна сітка (layout grid) у матеріальному дизайні складається з трьох елементів: стовпців (columns), канавок (gutters) та полів (margins).



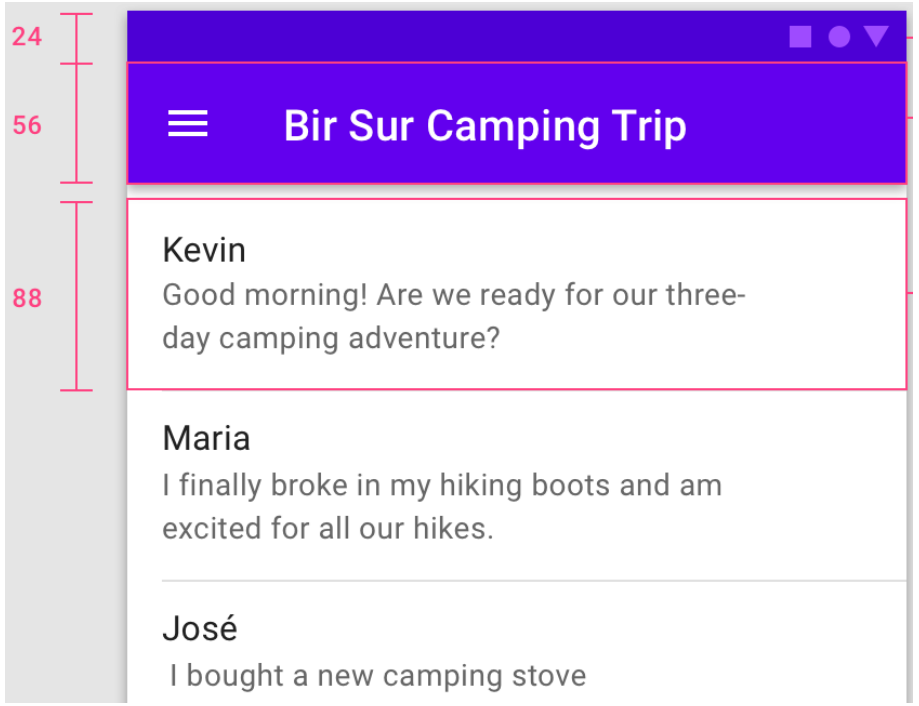


Макетування за допомогою відступів

Paddings



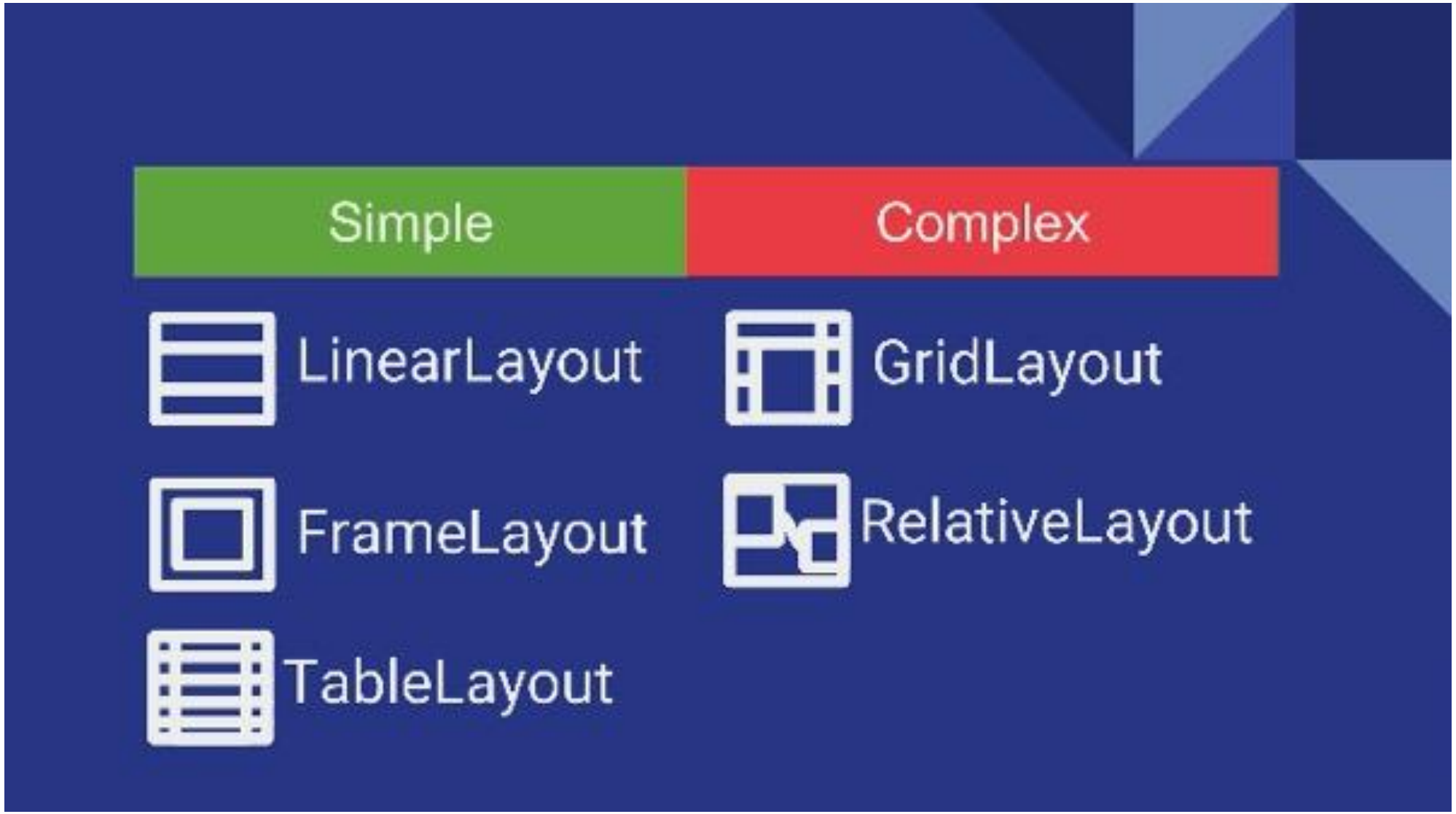
Відступи між елементами інтерфейсу (paddings) вимірюють величинами, кратними 8dp або 4dp.



https://material.io/design/layout/spacing-methods.html#spacing

Базові диспетчери компонувань Android

До появи матеріального дизайну



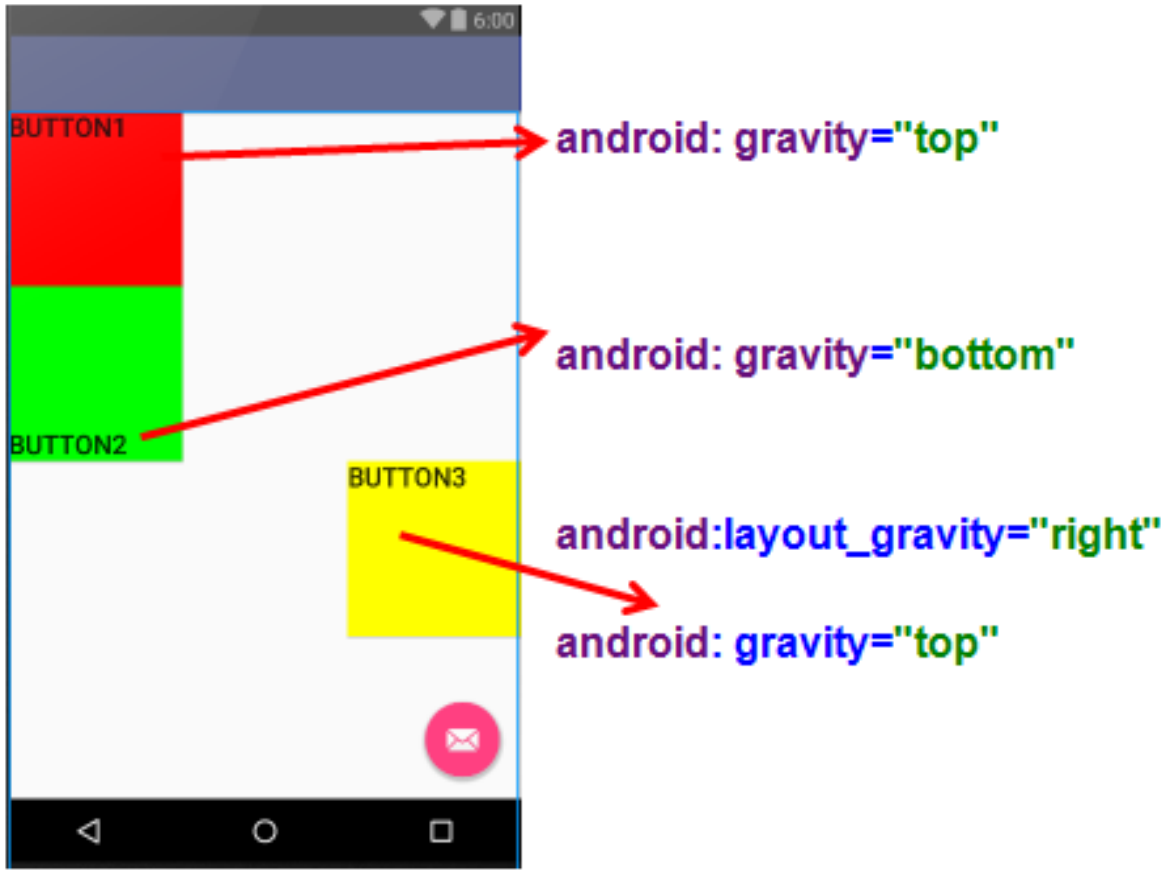
LinearLayout

Приклад XML-розмітки


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

LinearLayout

Основні атрибути



RelativeLayout

атрибути відносного позиціонування (по id)

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/superButton"
        android:text="Кнопка"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/textView"
        android:text="Якийсь текст"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/superButton" />
</RelativeLayout>
```

android:layout_above, android:layout_below
розташовують елемент вище/нижче вказаного елементу.

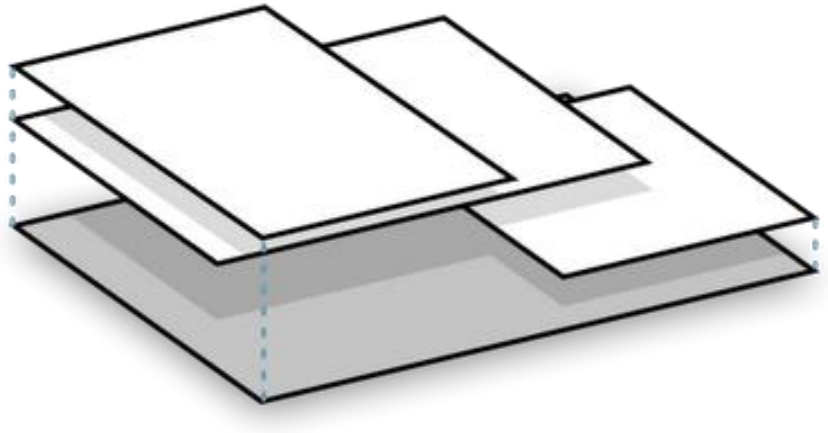
android:layout_toLeftOf, android:layout_toRightOf
встановлює елемент зліва/справа від вказаного елементу

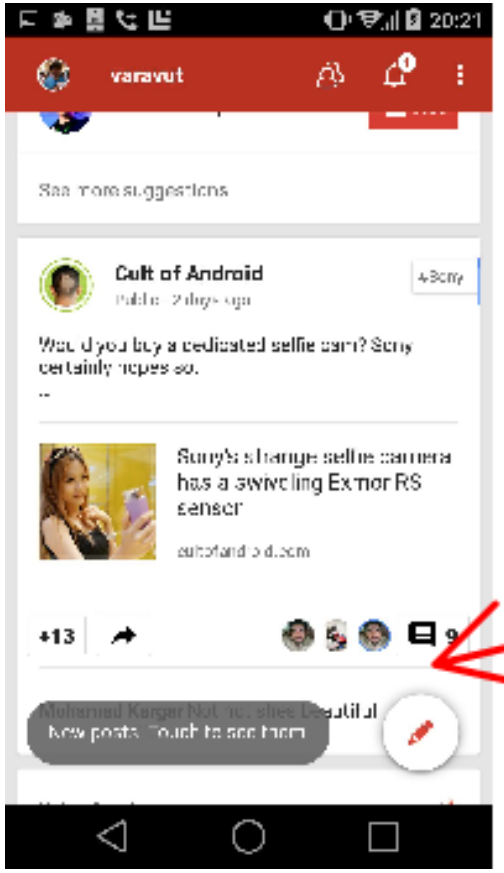
android:layout_alignBaseline
вирівнює базову лінію елементу з базовою лінією вказаного елементу

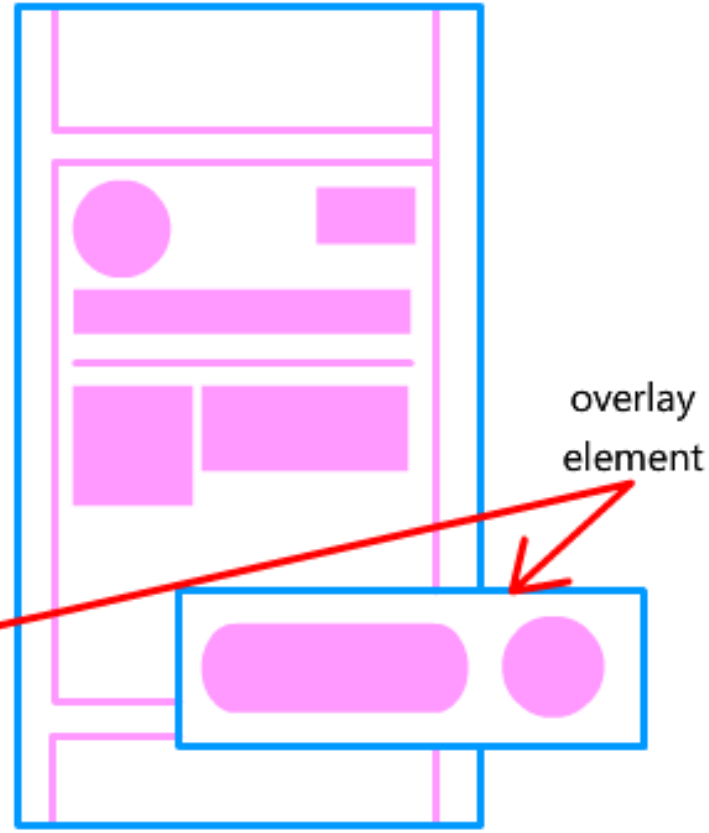
android:layout_alignBottom, android:layout_alignLeft, android:layout_alignRight, android:layout_alignTop
вирівнює низ / лівий край / правий край / верх елементу з відповідною частиною вказаного елементу

Диспетчер компонувань FrameLayout

Накладання віджетів та диспетчерів компонувань один на одного







Диспетчер має тільки ***один*** дочірній елемент, від якого йде решта елементів.

Видимістю елементів можна управляти за допомогою атрибуту android:visibility.

Android Design Support Library

Підключення бібліотек для розробки матеріальних інтерфейсів

I. Включити Maven-репозиторій Google у top-level файл build.gradle.

```
allprojects {
    repositories {
        google()
        // If you're using a version of Gradle lower than 4.1, you must instead use:
        // maven {
        //     url 'https://maven.google.com'
        // }
    }
}
```

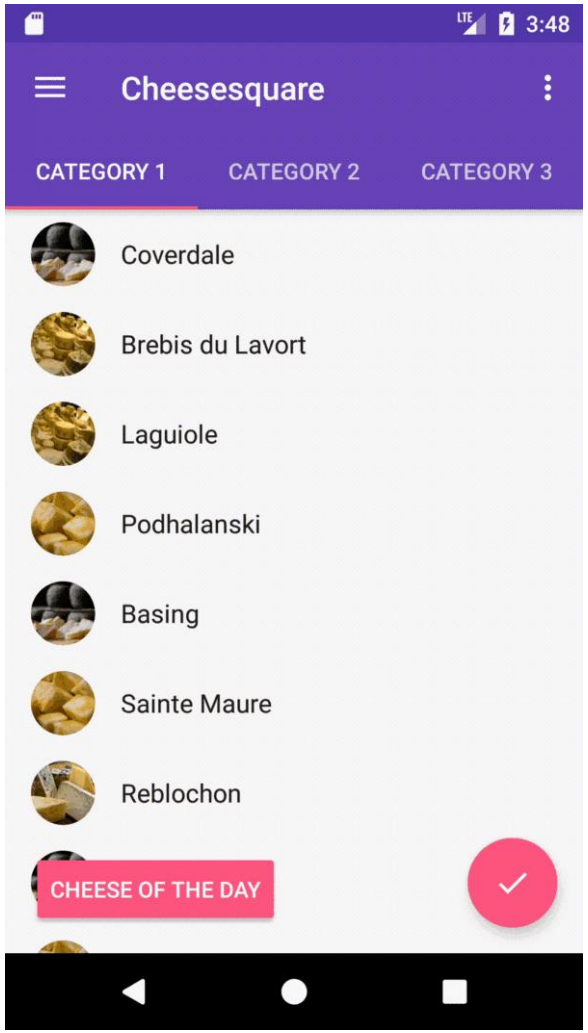
II. Додати бібліотеку в секцію dependencies.

```
dependencies {
    ...
    implementation "com.android.support:support-core-utils:28.0.0"
}
```

З випуском Android 9.0 (API level 28) з'явилась нова версія бібліотеки підтримки - AndroidX, яка є частиною Jetpack.

CoordinatorLayout - “super-powered FrameLayout”

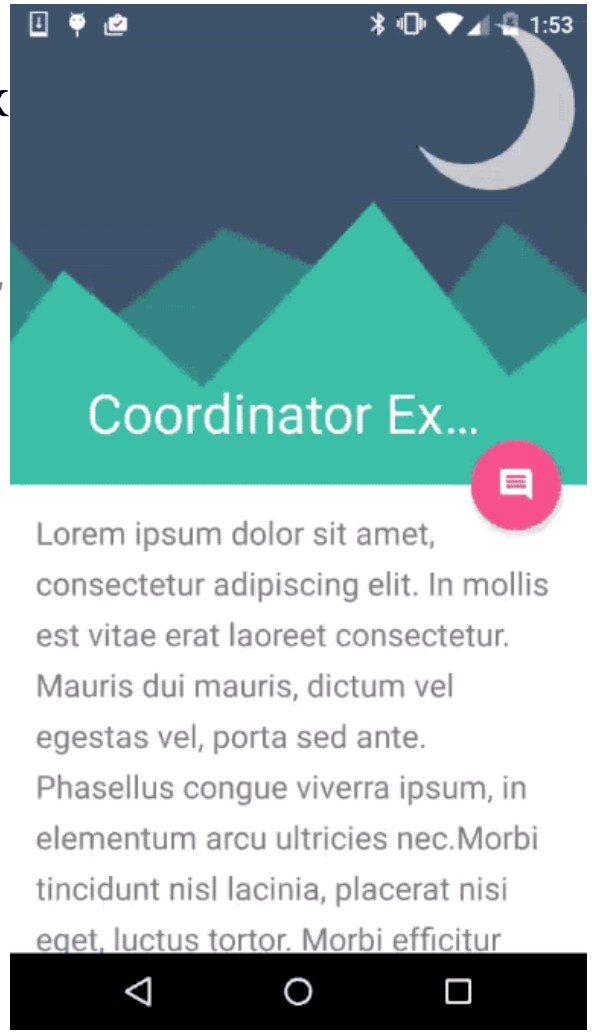
Диспетчери компонувань з бібліотеки підтримки (оглядово)



Використовується для допомоги координування взаємодії між віджетами всередині диспетчеру компонувань.

Для цього задіюються поведінки (Behaviors):

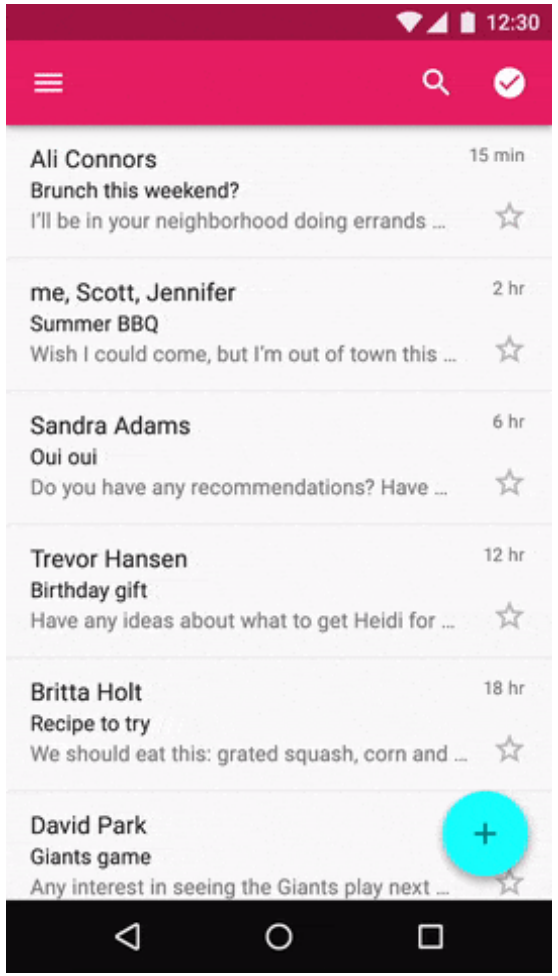
- ***Layout-Based Behaviors.*** Анкоринг (Anchoring) - також форма макето-орієнтованої поведінки (наприклад, FloatingActionButton + AppBarLayout)
- ***Scroll-Based Behaviors.*** Наприклад, при прокрутці RecyclerView, AppBar буде згортатись/розгортатись.



ToolBar та AppBarLayout

Диспетчери компонувань з бібліотеки підтримки (оглядово)

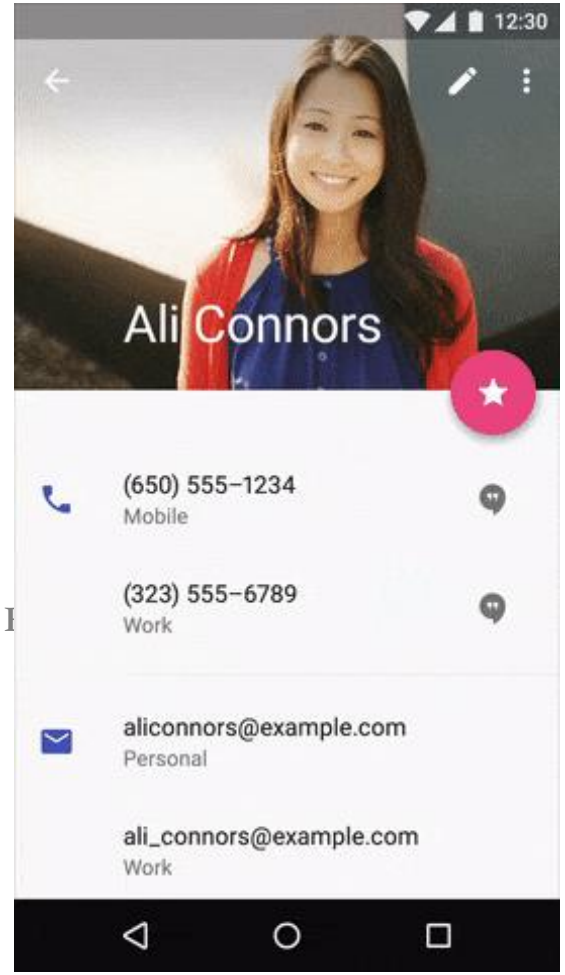
ToolBar



ToolBar – віджет, у якому можна визначити кілька елементів (іконку, меню тощо)

java.lang.Object
↳ android.view.View
↳ android.view.ViewGroup
↳ android.widget.Toolbar

AppBarLayout



AppBarLayout побудовано на основі вертикального LinearLayout.

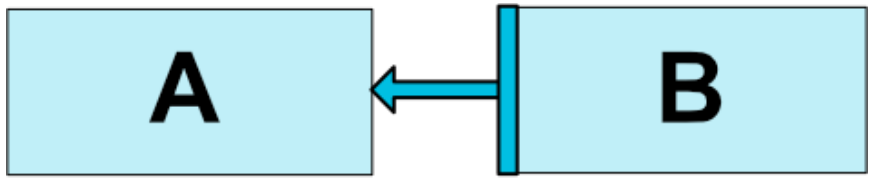
Це представлення залежить від використання в якості пря мого нащадка CoordinatorLayout.

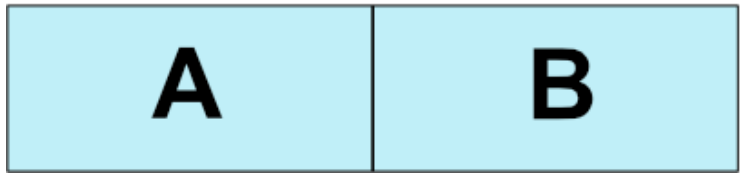
Використання в іншому ViewGroup-елементі не дозволяє к икористовувати більшість функціональності.

java.lang.Object
↳ android.view.View
↳ android.view.ViewGroup
↳ android.widget.LinearLayout
↳ android.support.design.widget.AppBarLayout

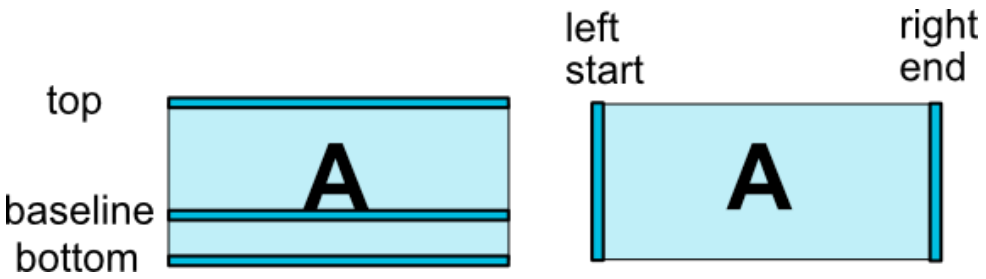
ConstraintLayout - гнучке позиціонування елементів

Відносне позиціонування





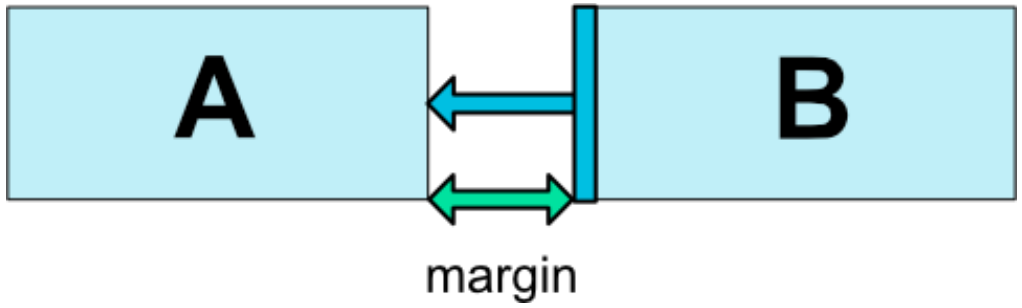

```
<Button android:id="@+id/buttonA" ... />
<Button android:id="@+id/buttonB" ...
        app:layout_constraintLeft_toRightOf=
            "@+id/buttonA" />
```



- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

ConstraintLayout - гнучке позиціонування елементів

Margins



- android:layout_marginStart
- android:layout_marginEnd
- android:layout_marginLeft
- android:layout_marginTop
- android:layout_marginRight
- android:layout_marginBottom

When a position constraint target's visibility is View.GONE, you can also indicate a different margin value to be used using the following attributes:

- layout_goneMarginStart
- layout_goneMarginEnd
- layout_goneMarginLeft
- layout_goneMarginTop
- layout_goneMarginRight
- layout_goneMarginBottom

https://developer.android.com/reference/android/support/constraint/ConstraintLayout

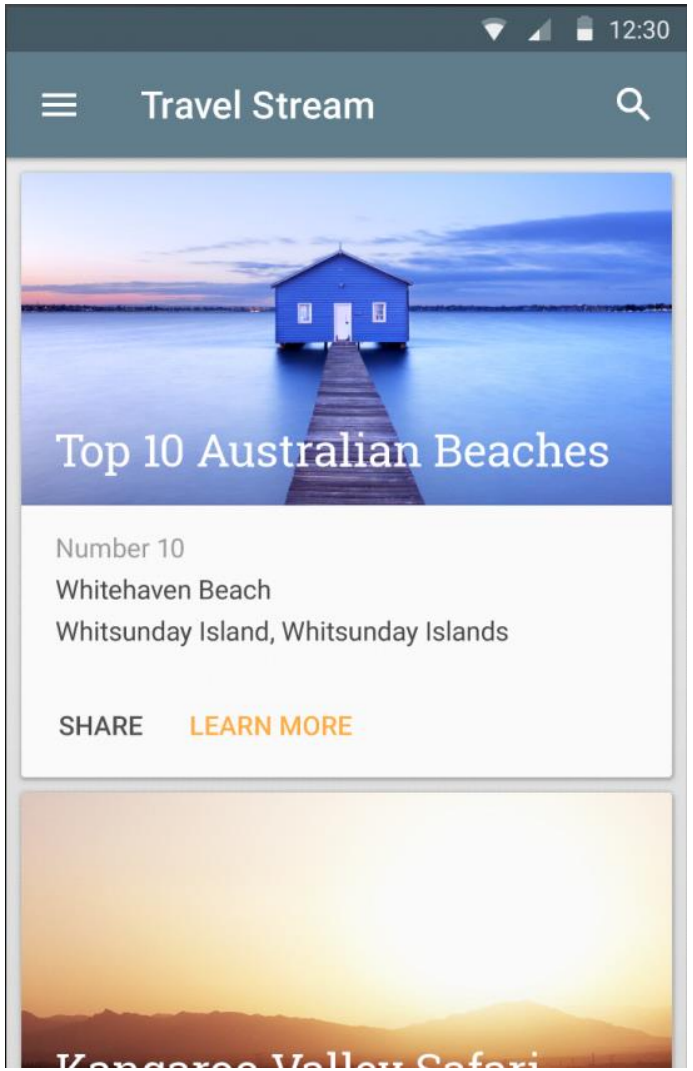
Картки

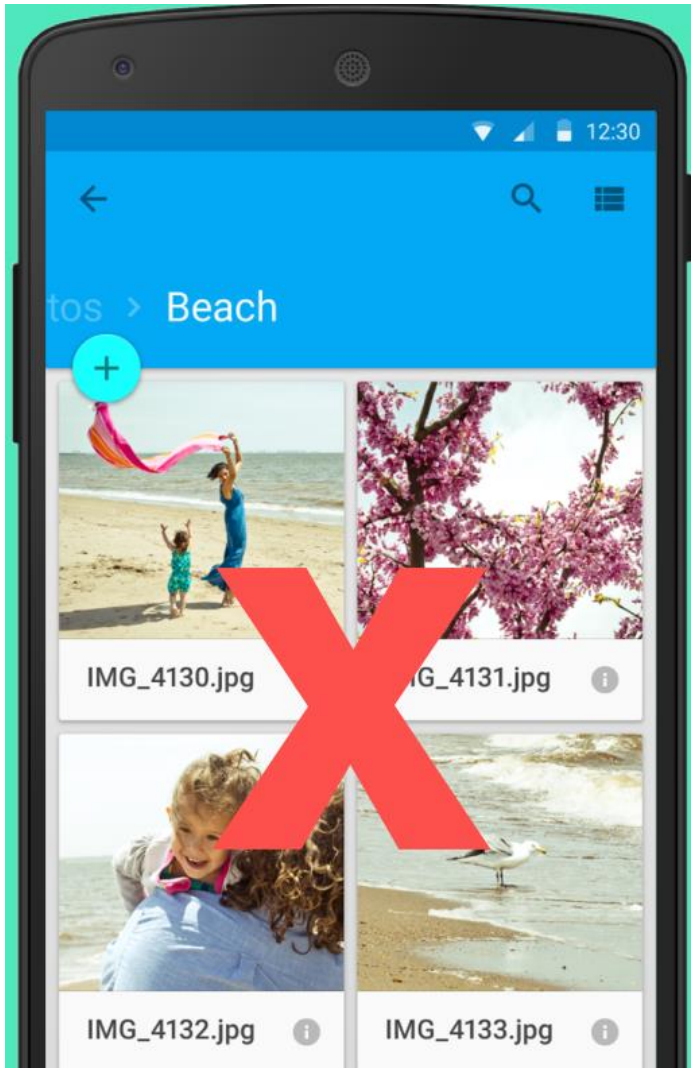
шматок паперу, що містить унікальний набір даних пов’язаної, гетерогенної інформації

```
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- Your card content -->

</android.support.v7.widget.CardView>
```

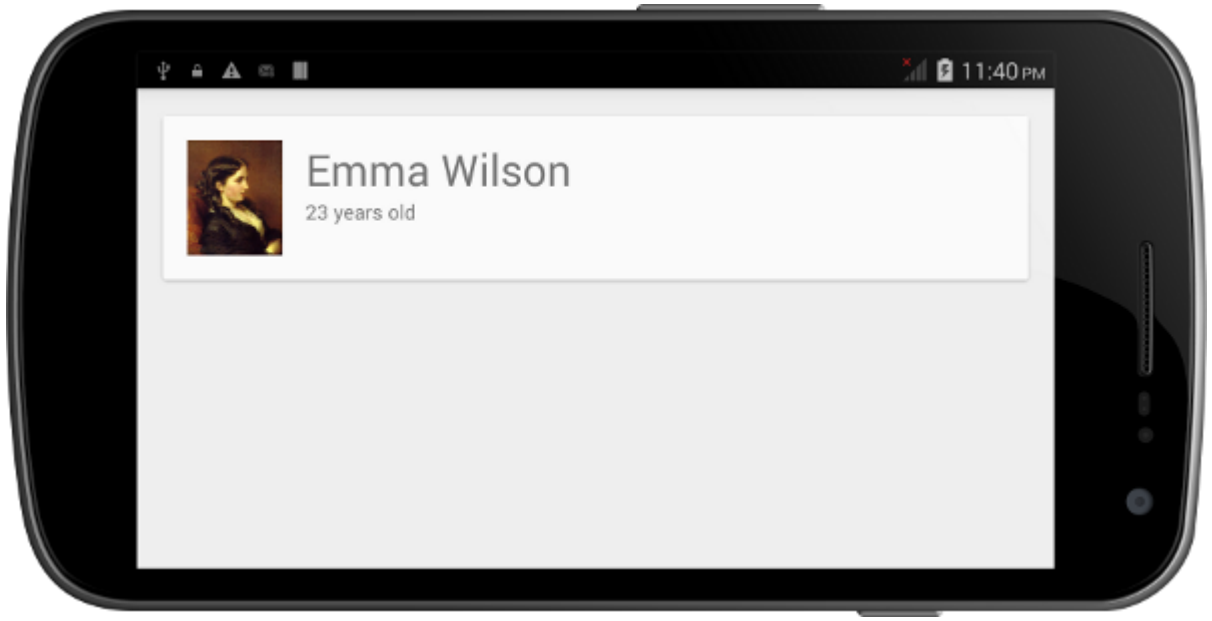
При підключенні:

```
dependencies {
   implementation 'com.android.support:cardview-v7:28.0.0'
}
```





Всередині CardView

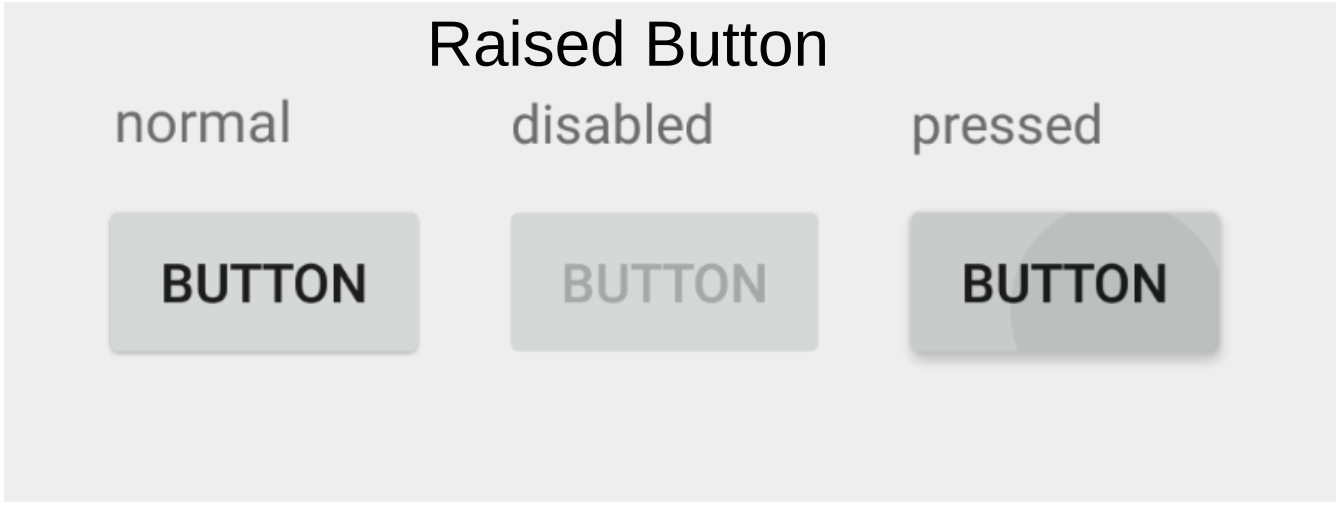


CardView належить до ViewGroup, тому може додаватись до активності або фрагменту через XML-файл макету.

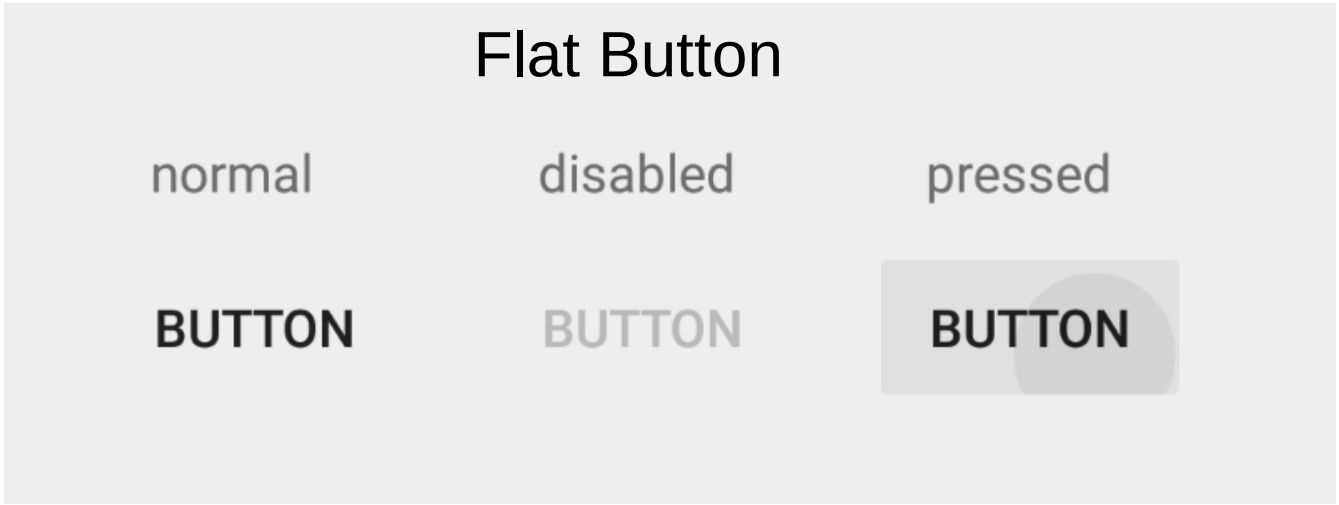
Використовується в якості елементу RecyclerView

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:padding="16dp“>
   <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/person_photo"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginRight="16dp" />
   <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/person_name"
            android:layout_toRightOf="@+id/person_photo"
            android:layout_alignParentTop="true"
            android:textSize="30sp“ />
   <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/person_age"
            android:layout_toRightOf="@+id/person_photo"
            android:layout_below="@+id/person_name" />
</RelativeLayout>
```

Базові віджети. Кнопки




```
<Button
    android:layout_width ="wrap_content"
    android:layout_height ="wrap_content"
    android:text="Button"/>
```

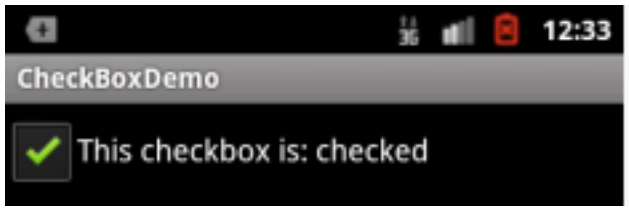


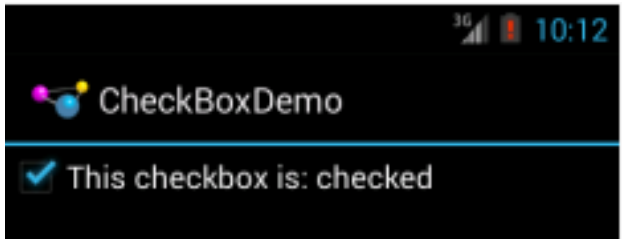

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    style="@style/Widget.AppCompat.Button.
Borderless"/>
```

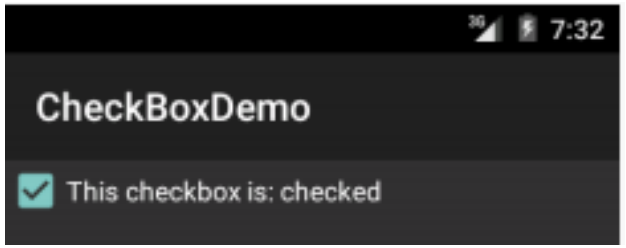
checkbox

У Java-коді для виклику доступні методи: isChecked(), setChecked(), toggle()

```
<CheckBox
  android:id="@+id/check"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/unchecked"/>
```



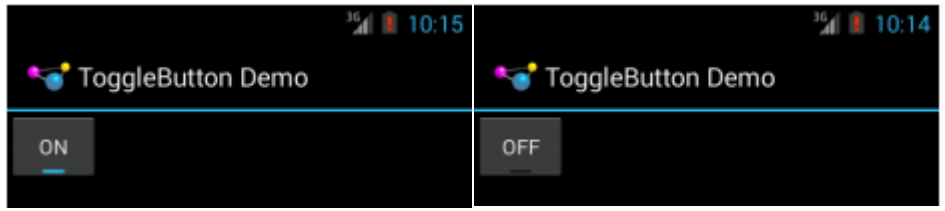


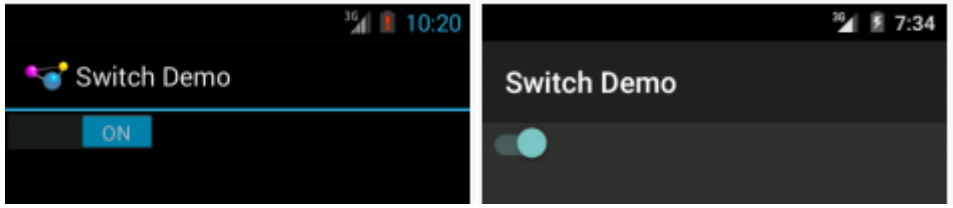


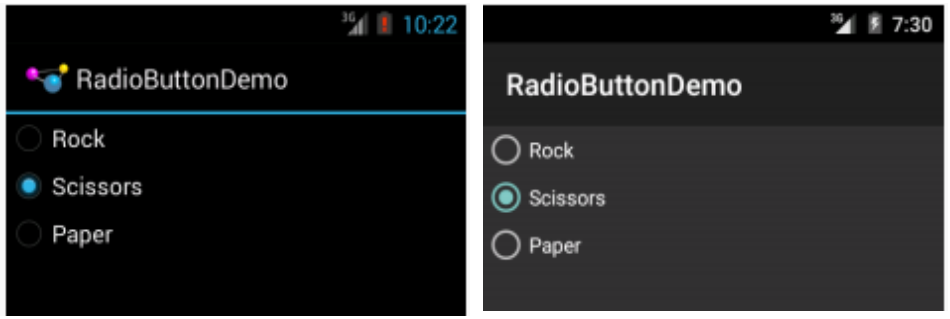

```
public class CheckBoxDemo extends Activity implements CompoundButton.OnCheckedChangeListener {
  CheckBox cb;
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    cb=(CheckBox)findViewById(R.id.check);
    cb.setOnCheckedChangeListener(this);
  }

  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    if (isChecked) {  cb.setText(R.string.checked); }
    else {  cb.setText(R.string.unchecked);  }
  }
}
```

Кнопки-перемикачі

ToggleButton, Switch, RadioButton








```
<RadioGroup android:orientation="vertical" ...>
   <RadioButton .../>
</RadioGroup>
```

```
<ToggleButton
  android:id="@+id/toggle"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

```
<Switch
  android:id="@+id/toggle"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

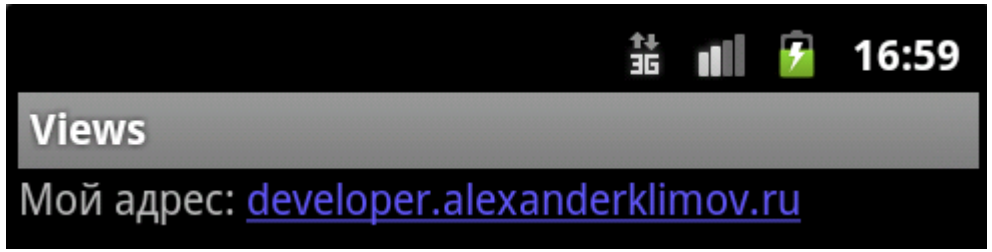
RadioGroup – це LinearLayout з набором radio buttons, чий стан взаємопов’язаний.

Для доступу до RadioGroup з Java по id:

- check() (group.check(R.id.radio1))
- clearCheck()
- getCheckedRadioButtonId() повертає ID поточної обраної радіокнопки (-1, якщо не вибрано жодну)

Текстові віджети

TextView вміє відображати текст, проте не дозволяє його редагувати




```
<TextView android:id="@+id/tv"
    android:layout_width="match_parent"
    android:autoLink="web"
    android:linksClickable="true"
    android:text="Мой адрес: ___________" />
```

- Якщо текстом буде URL або електронна пошта, можна встановити атрибут
  - android:autoLink="email|web" для підсвічування.
  - можуть бути web, email, phone, map, none (за замовчуванням)
  - у коді реалізується методом setAutoLinkMask() класу android.text.util.Linkify

```
TextView tv =(TextView)this.findViewById(R.id.tv);
tv.setAutoLinkMask(Linkify.ALL);
tv.setText("Please visit my website, http://www.androidbook.com
or email me at davemac327@gmail.com.");
```

Обов'язково до самого тексту

Текстові віджети

Поширені атрибути TextView

- Для відображення тексту в TextView у файлі розмітки використовується атрибут
  - android:text="@string/hello"
- Програмно текст можна задати методом setText():

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setText("Hello Kitty!");
textView.setText(R.string.hello);
```

- android:textSize – розмір тексту.
  - Для текстів рекомендується використовувати sp: android:textSize="48sp"
- android:textStyle – стиль тексту.
  - константи: normal, bold, italic.
  - android:textStyle="bold" виводить текст жирним
- android:textColor – колір тексту.
  - 4 формати в шістнадцятковому кодуванні: #RGB; #ARGB; #RRGGBB; #AARRGGBB

Текстові віджети

EditText

normal Hint text

focus Hint text

disabled Hint text

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Hint text"/>
```

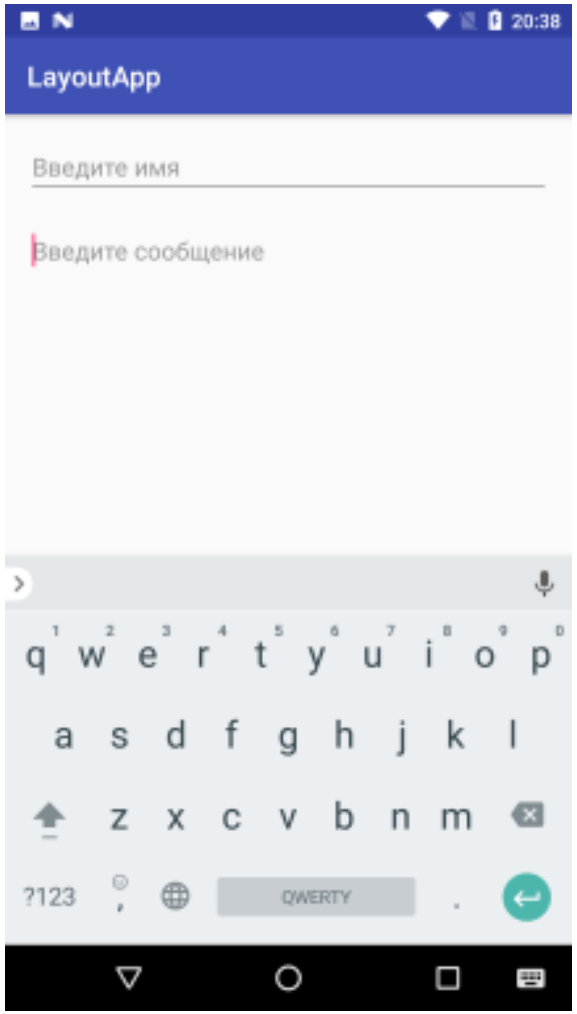
Android 4.4 | Android 5.0 | Android 5.1

First name | First name | First name

The height and vertical alignment of the EditText are different.

Текстові віджети

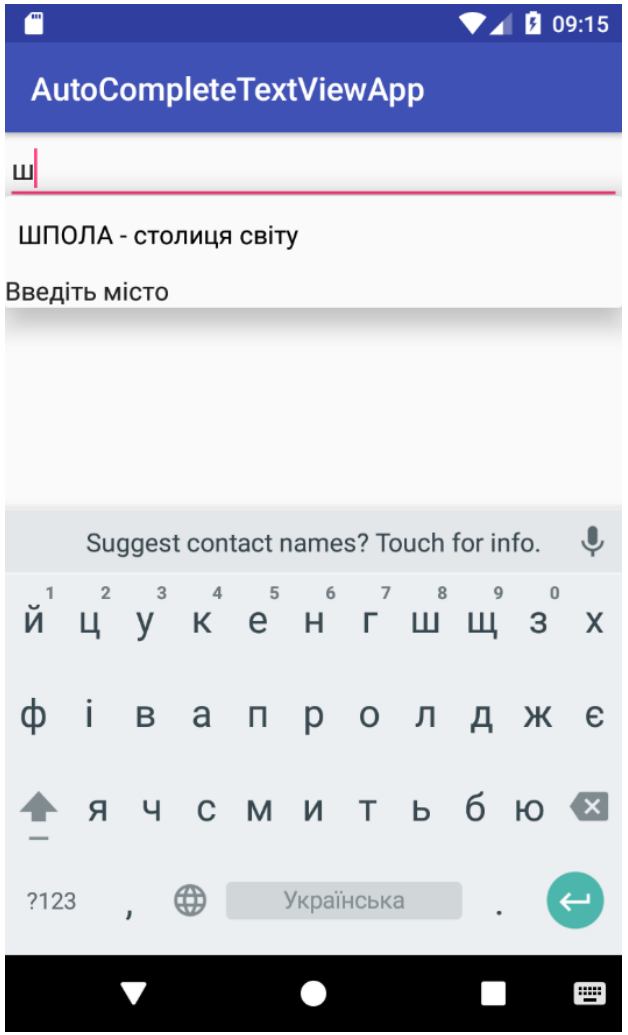
Поширені атрибути EditText



- android:hint – підказка в полі вводу.
- android:inputType – вибирає клавіатуру для вводу
  - text: обычная клавиатура для ввода однострочного текста
  - textMultiLine: многострочное текстовое поле
  - textEmailAddress: обычная клавиатура, на которой присутствует символ @, ориентир ована на ввод email
  - textUri: обычная клавиатура, на которой присутствует символ /, ориентирована на в вод интернет-адресов
  - textPassword: клавиатура для ввода пароля
  - textCapWords: при вводе первый введенный символ слова представляет заглавную бу кву, остальные - строчные
  - number: числовая клавиатура
  - phone: клавиатура в стиле обычного телефона
  - date: клавиатура для ввода даты
  - time: клавиатура для ввода времени
  - datetime: клавиатура для ввода даты и времени

Текстові віджети

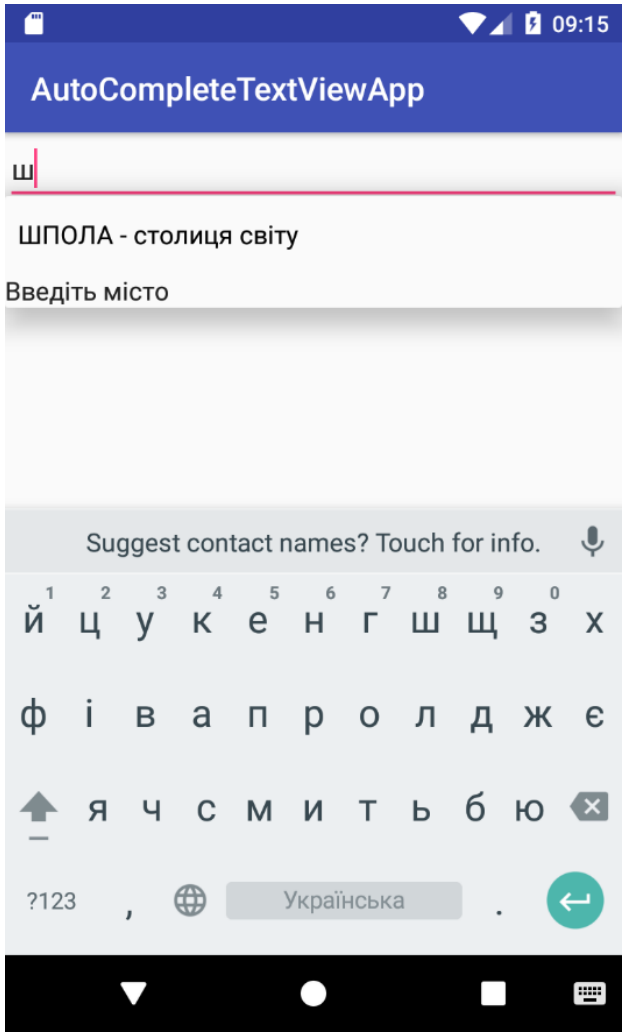
AutoCompleteTextView – поле вводу з автозаповненням




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <AutoCompleteTextView
        android:id="@+id/autocomplete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:completionHint="Введіть місто"
        android:completionThreshold="1"/>

</LinearLayout>
```

Текстові віджети

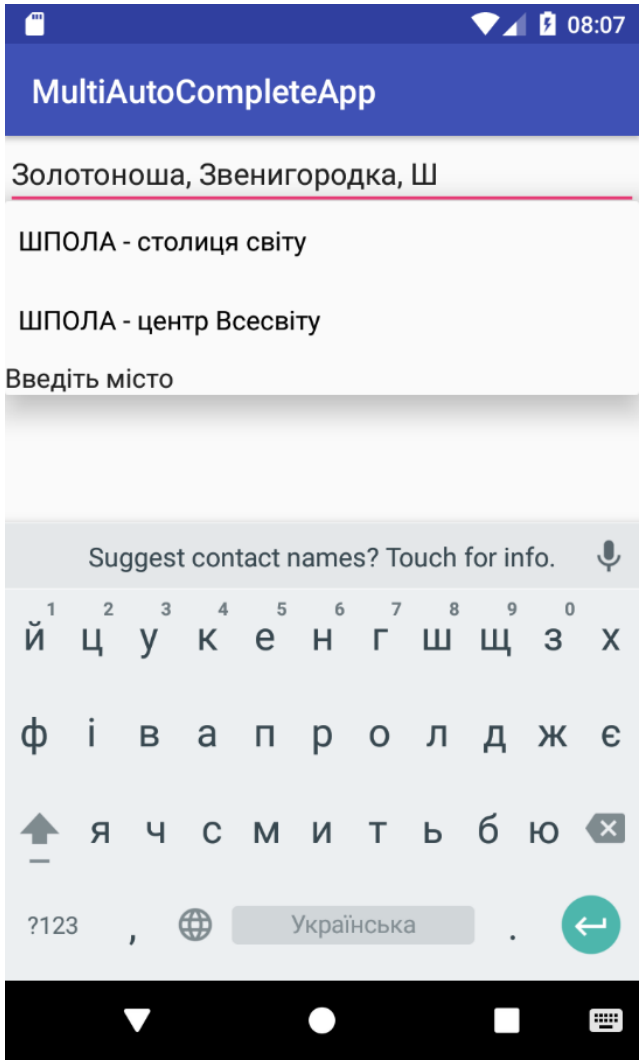
AutoCompleteTextView – поле вводу з автозаповненням




```java
public class MainActivity extends AppCompatActivity {

    String[] cities = {"Черкаси", "Сміла", "Золотоноша", "Звенигородка",
                       "ШПОЛА - столиця світу", "Умань", "Чигирин"};

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

        // Отримуємо посилання на елемент AutoCompleteTextView у розмітці
        AutoCompleteTextView autoCompleteTextView =
                (AutoCompleteTextView) findViewById(R.id.autocomplete);
        // Створюємо адаптер для автозаповнення елемента AutoCompleteTextView
        ArrayAdapter<String> adapter =
                new ArrayAdapter<String>(this,
                        R.layout.support_simple_spinner_dropdown_item, cities);
        autoCompleteTextView.setAdapter(adapter);
    }
}
```

Текстові віджети

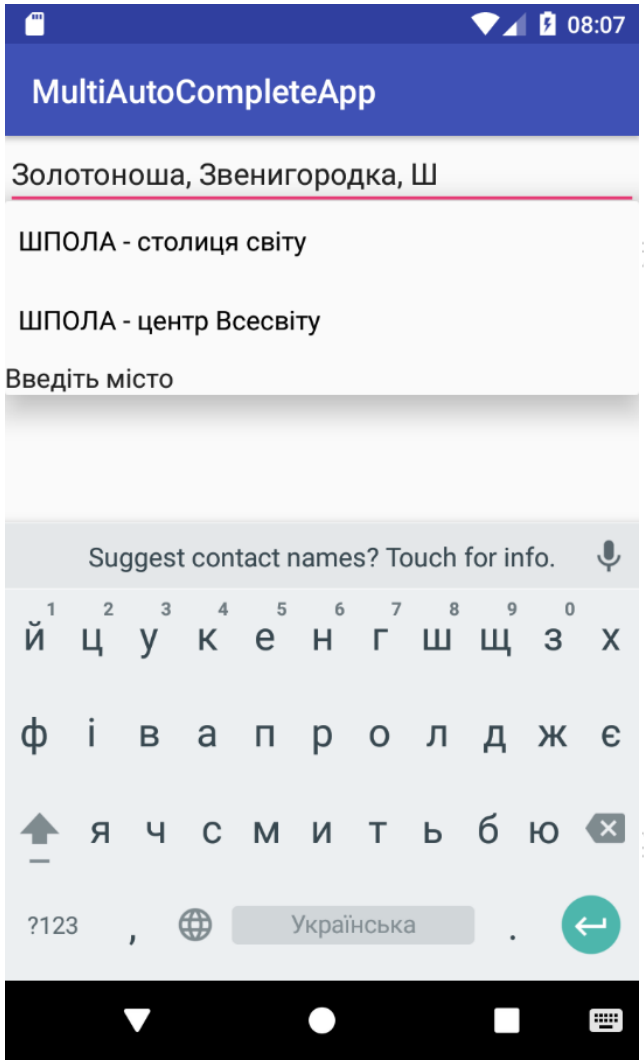
MultiAutoCompleteTextView – поле вводу з ретельнішим автозаповненням




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <MultiAutoCompleteTextView
        android:id="@+id/autocomplete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:completionHint="Введіть місто"
        android:completionThreshold="1"/>

</LinearLayout>
```

Текстові віджети

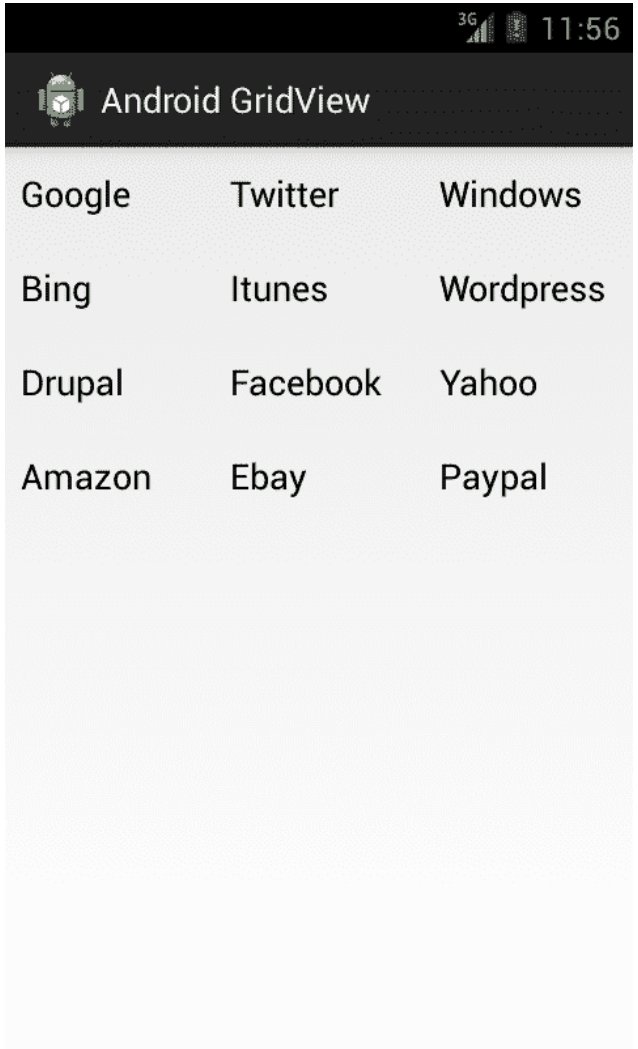
MultiAutoCompleteTextView – поле вводу з ретельнішим автозаповненням




```java
public class MainActivity extends AppCompatActivity {

    String[] cities = {"Черкаси", "Сміла", "Звенигородка", "Золотоноша",
            "ШПОЛА - столиця світу", "ШПОЛА - центр Всесвіту", "Умань"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Отримуємо посилання на елемент MultiAutoCompleteTextView в розмітці
        MultiAutoCompleteTextView autoCompleteTextView =
                (MultiAutoCompleteTextView) findViewById(R.id.autocomplete);
        // Створюємо адаптер для автозаповнення елементу MultiAutoCompleteTextView
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                R.layout.support_simple_spinner_dropdown_item, cities);
        autoCompleteTextView.setAdapter(adapter);
        // визначення коми в якості роздільника
        autoCompleteTextView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
    }
}
```

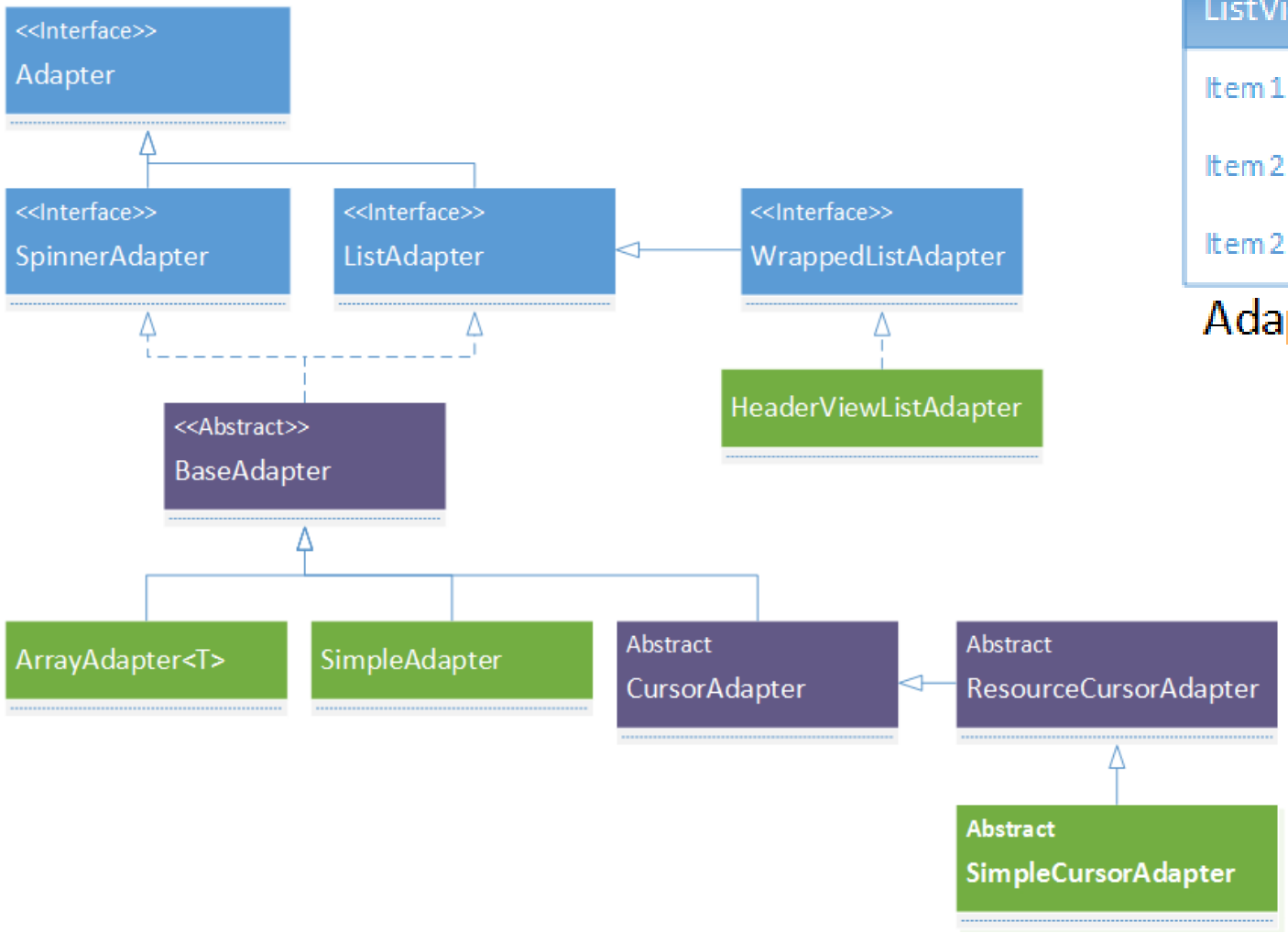
Списковi віджети до Android 5.0

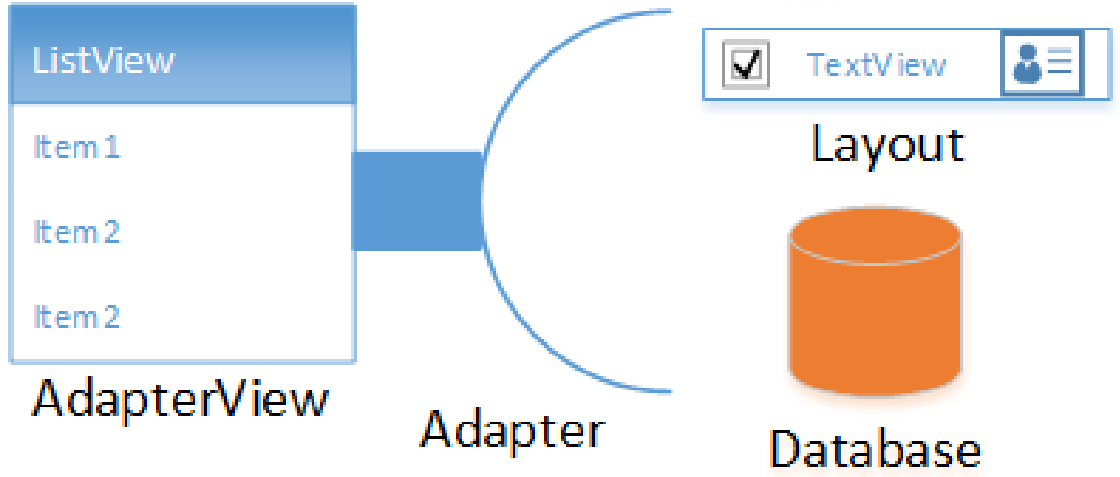
ListView, GridView, Spinner



Поняття адаптера

міст між набором данных та об’єктом, що їх використовує





Також відповідає за створення View-компонента для кожної одиниці даних з набору.

Adapter є інтерфейсом, який реалізують всі інші адаптери:

- ArrayAdapter
- SimpleAdapter
- SimpleCursorAdapter

Вбудовані адаптери

- ArrayAdapter<T> — адаптер для узагальненого масиву довільних об’єктів.
- CursorAdapter — адаптер, що надає дані для списку з курсору.
- SimpleAdapter — простий адаптер, зазвичай застосовується для заповнення списку статичними даними (можливо, з ресурсів).
- ResourceCursorAdapter — розширяє клас CursorAdapter та вміє створювати представлення з ресурсів.
- SimpleCursorAdapter — розширяє клас ResourceCursorAdapter та створює представлення TextView/ImageView з стовпців курсора. Представлення визначені в ресурсах.

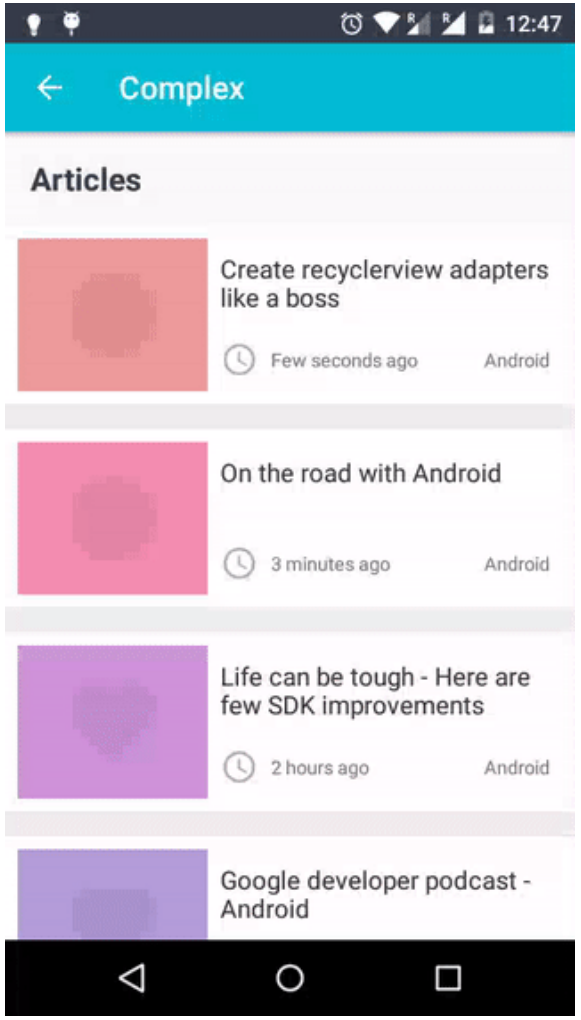
ArrayAdapter

Динамічна зміна даних

ArrayAdapter допускає динамічну зміну даних, що передаються.

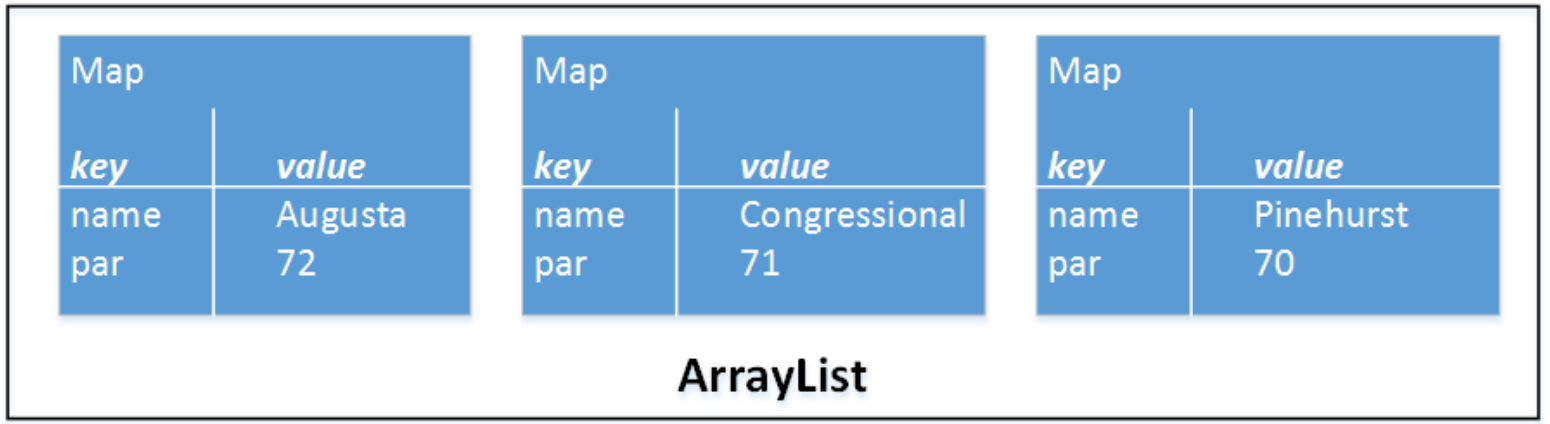
- add() додає новий запис в кінець масиву.
- insert() додає новий запис у задану позицію в масиві.
- remove() видаляє об’єкт з масиву.
- sort() впорядковує масив.

Після виконання цих операцій дані не синхронізовані з ListView, тому п отрібно викликати метод адаптеру notifyDataSetChanged().



SimpleAdapter

Набір даних для SimpleAdatper – це ArrayList of Maps.



Кожен меп містить дані, що будуть показані в одному рядку AdapterView.
Для цього адаптеру дані очікуються статичними, без змін.

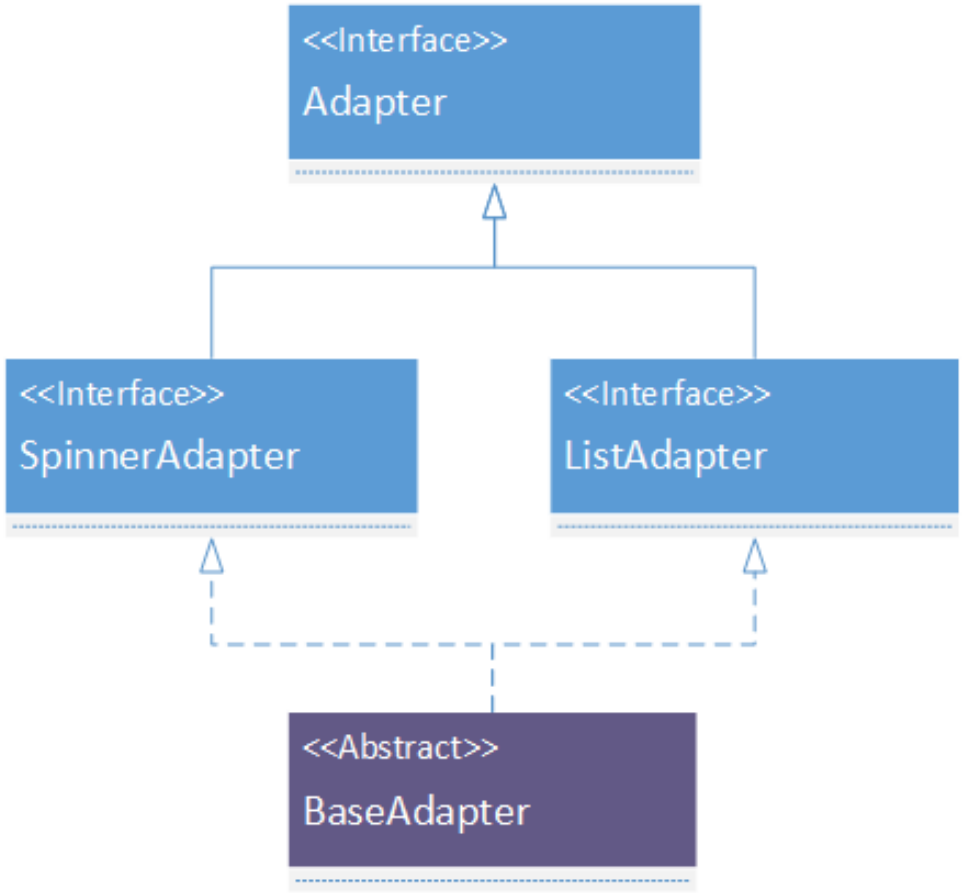
SimpleAdapter зазвичай не вимагає будувати власний клас та переозначені методи.

- Конструктор SimpleAdapter дозволяє створити екземпляр класу та прикріпити його до AdapterView.
- SimpleAdapter дозволяє відобразити більше різноманітних даних, а ArrayAdapter працює з простим t oString() представленням для масиву об'єктів.

- Ви можете розширити SimpleAdapter, проте ціною появи додаткової роботи, яка часто стосується ре алізації одного з інтерфейсів Adapter.

Абстрактний клас BaseAdapter

Джерело непорозумінь



Абстрактний клас BaseAdapter реалізує всі інтерфейси: ListAdapter, SpinnerAdapter, Adatper!

- Інтерфейс Adapter визначає 10 методів, що повинні реалізовуватись розробником.

BaseAdapter постачається для зручності розробників.

- Замість створення класу, який реалізує Adapter, SpinnerAdapter чи ListAdapter, просто розширяють BaseAdapter та реалізовують спеціальні методи для ListView або Spinner.
- BaseAdapter уже забезпечує поширені реалізації для багатьох інтерфейсів, тому переозначати без необхідності нічого не потрібно.

Адаптери

Основні методи для імплементації

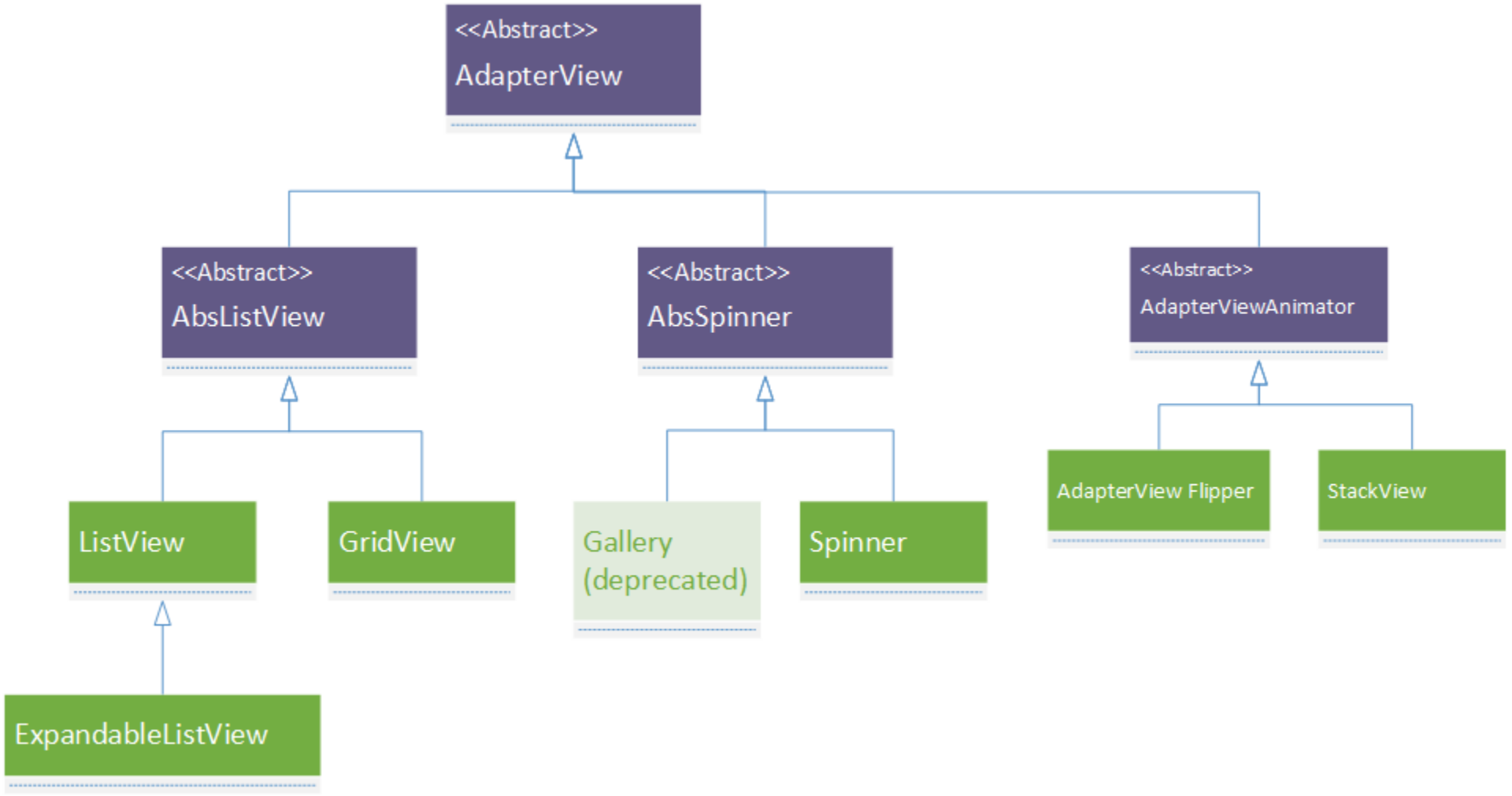
- **getCount():** вказує, скільки елементів (рядків) присутні в наборі даних, що будуть відображатись в AdapterView.

- **getItem(int pos):** отримує дані, пов'язані з елементом (рядком) AdapterView, переданим у якості параметру в метод. Android використає цей метод для збору (fetch) доречних даних при побудові елементу/рядка в AdapterView.

- **getItemId(int pos):** повертає ідентифікатор набору даних для позиції елементу/рядка AdapterView. Зазвичай відповідає рядкам AdapterView.

- **getView(int position, View convertView, ViewGroup parent):** створює View, яке відображає дані для заданої (по позиції) елементу/рядка AdapterView.
  - convertView дозволяє повторно використовувати раніше створені View.
  - parent – View (зазвичай AdapterView), до якого прикріплюється View for the item/row.

Абстрактний клас AdapterView



Приблизна реалізація адаптера

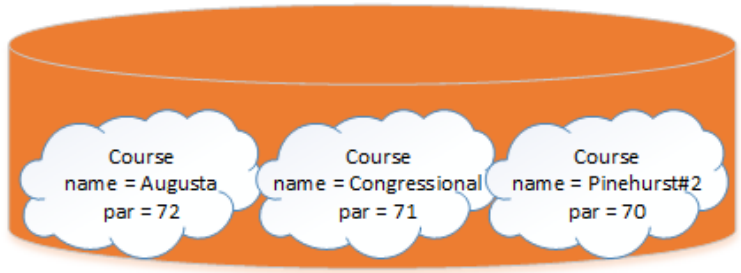
маємо ArrayList об’єктів-курсів гольфу, який треба показати в ListView

```
public class CourseAdapter extends BaseAdapter {

 List<Course> courses = // code to get the courses ArrayList

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

 @Override
 public Object getItem(int position) {
  return courses.get(position);
 }

 @Override
 public long getItemId(int position) {
  return position;
 }
```

Для кожного такого об’єкту потрібно в ідобразити layout (View), що містить че кбокс, назву курсу та бали (par score) з к ожного курсу гольфу.

AdapterView

BaseAdapter



ArrayList

Приблизна реалізація адаптера

(продовження)

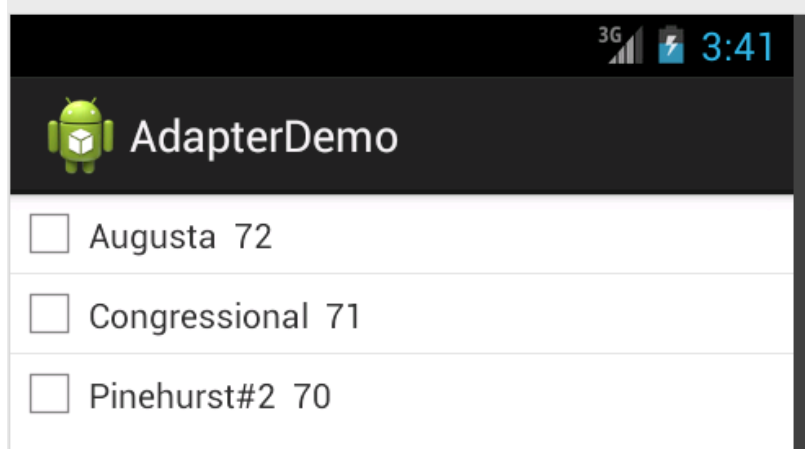
```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
 if (convertView == null) {
  Context context = parent.getContext();
  LinearLayout view = new LinearLayout(context);
  view.setOrientation(LinearLayout.HORIZONTAL);
  view.addView(new CheckBox(context));
  TextView nameTextView = new TextView(context);
  nameTextView.setText(courses.get(position).getName());
  nameTextView.setPadding(0, 0, 10, 0);
  view.addView(nameTextView);
  TextView parTextView = new TextView(context);
  parTextView.setText(Integer.toString(courses.get(position).getPar()));
  view.addView(parTextView);
  return view;
 }
 return convertView;
}

}
```

Метод getView() програмно будує представлення, яке представляє кожен курс при відображенні ListView.

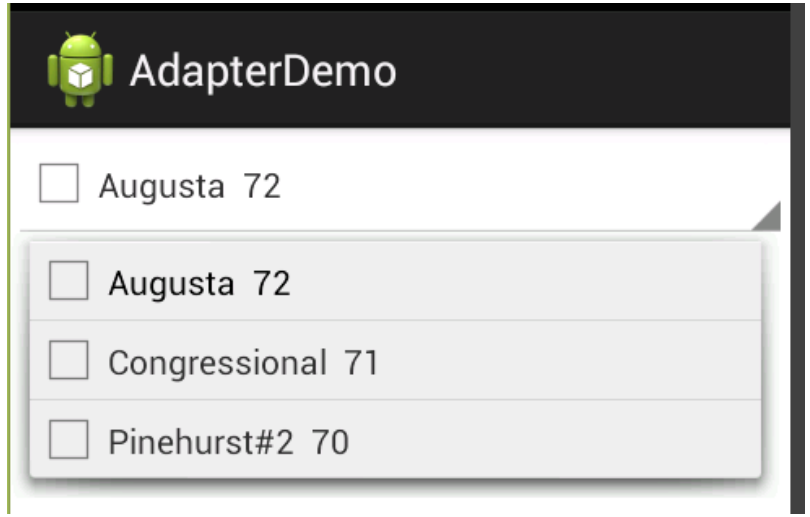
Проте можна використовувати XML-макет для конструювання View.

Прив’язка до ListView або Spinner

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ListView listView = (ListView) findViewById(R.id.crsListView);
  BaseAdapter crsAdapter = new CourseAdapter();
  listView.setAdapter(crsAdapter);
}
```




```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Spinner spinner = (Spinner) findViewById(R.id.crsSpinner);
  BaseAdapter crsAdapter = new CourseAdapter();
  spinner.setAdapter(crsAdapter);
}
```



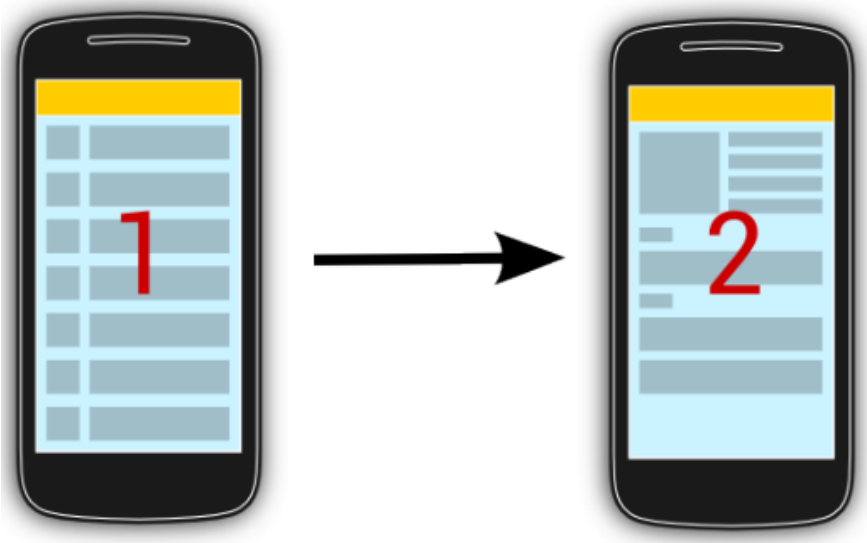
Фрагменти

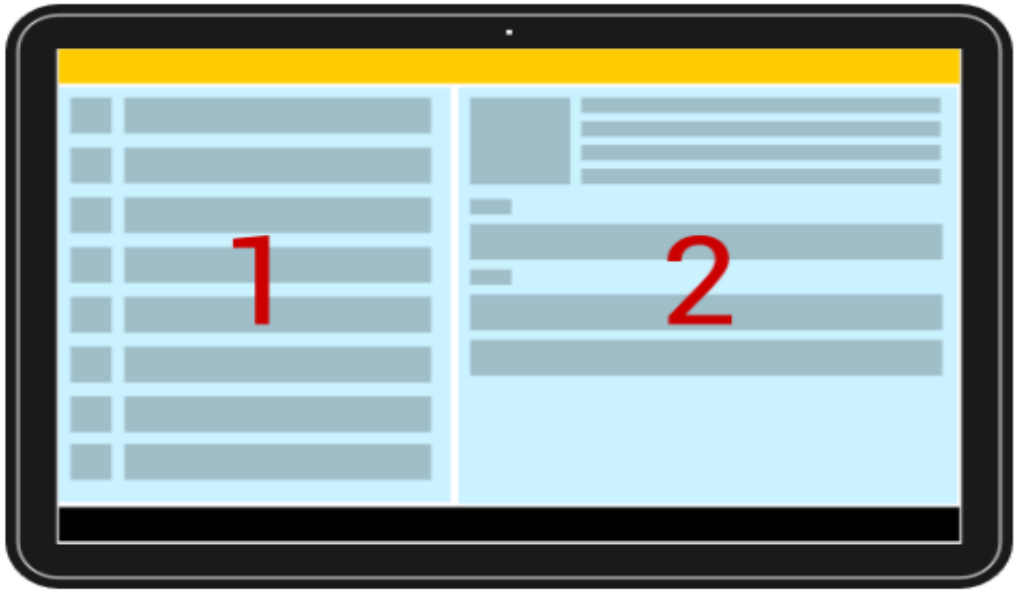
Модульність інтерфейсу користувача

Фрагменти

Потреба у фрагментах з’явилась після появи планшетів на Android

- Фрагменти дозволяють розбивати інтерфейс на функціональні групи, які активність може завантажувати та якими може управляти
- Фрагменти дозволяють враховувати форм-фактор пристрою, у той час як активність управляє інтерфейсом в цілому.
- Фрагменти не заміняють активності, а розширяють їх функціональність.
- Фрагменти завжди існують всередині активності.





Старий activity-oriented спосіб мислення


```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical“
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ScrollView
        android:layout_width="match_parent”
        android:layout_height="0dp“
        android:id="@+id/scrollTitles“   android:layout_weight="1">
        <RadioGroup     android:id="@+id/bookSelectGroup"
            android:layout_height="wrap_content“  android:layout_width="wrap_content">
            <RadioButton  android:id="@+id/dynamicUiBook"
                    android:layout_height="wrap_content"
                    android:layout_width="wrap_content"
                    android:text="@string/dynamicUiTitle"
                    android:checked="true" />
            …
        </RadioGroup>
    </ScrollView>
```

Старий activity-oriented спосіб мислення


```
<!– Опис вибраної книги -->
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:id="@+id/scrollDescription"
    android:layout_weight="1">
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="@string/dynamicUiDescription"
            android:id="@+id/textView"
            android:paddingLeft="@dimen/activity_horizontal_margin"
            android:paddingRight="@dimen/activity_horizontal_margin"
            android:gravity="fill_horizontal"/>
  </ScrollView>
</LinearLayout>
```

Старий activity-oriented спосіб мислення

Відображення інтерфейсу активності:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // load the activity_main layout resource
    setContentView(R.layout.activity_main);
}
```

Нове мислення, орієнтоване на фрагменти

Перший крок – ідентифікувати природні «порції» для існуючого інтерфейсу

Визначаємо новий ресурсний файл fragment_book_list.xml та копіюємо в н ього частину коду.

Хороша практика дизайну – фрагмент має займати весь простір, у якому він р озміщений.

```
<ScrollView
  android:layout_width="match_parent"
  android:layout_height="0dp"
  android:id="@+id/scrollTitles"
  android:layout_weight="1">
  <RadioGroup
    android:id="@+id/bookSelectGroup "
    android:layout_height="wrap_content"
    android:layout_width="wrap_content">
  <RadioButton
    android:id="@+id/dynamicUiBook"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/dynamicUiTitle"
    android:checked="true"/>
  <RadioButton
    android:id="@+id/android4NewBook"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/android4NewTitle"/>

    <!-- Other RadioButtons elided for clarify -->
  </RadioGroup>
</ScrollView>
```

Нове мислення, орієнтоване на фрагменти

Перший крок – ідентифікувати природні «порції» для існуючого інтерфейсу

Видалимо атрибут layout_weight із елементу ScrollView,
Змінимо значення атрибуту layout_height на match_parent.

```
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/scrollTitles">
  <!—RadioGroup and RadioButton elements elided for clarity -->
</ScrollView>
```

З оновленим елементом ScrollView макет фрагменту тепер може адапт уватись до майже будь-якого layout, з яким він пов’язаний.

Нове мислення, орієнтоване на фрагменти

Аналогічно створюємо макет фрагменту fragment_book_desc.xml


```
<!-- Description of selected book -->
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/scrollDescription">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="@string/dynamicUiDescription"
    android:id="@+id/textView"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:gravity="fill_horizontal"/>
</ScrollView>
```

Створення класу для фрагменту

Всі класи фрагментів напряму або опосередковано розширюють клас android.app.Fragment.

```
Import android.app.Fragment;
public class BookListFragment extends Fragment { … }
```

Метод onCreateView() відповідає за повернення ієрархії представлень (view hierarchy) фраг-м енту.

- inflater: посилання на екземпляр LayoutInflater, яке може читати та expand layout-ресурси у контексті активності-контейнера.
- container: посилання на екземпляр ViewGroup всередині макету активності, де ієрархія представлень фрагмента буде прикріплятись (attach)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
  View viewHierarchy =
  inflater.inflate(R.layout.fragment_book_list,
    container, false);
  return viewHierarchy;
}
```

Фрагмент для опису книги

Для фрагменту опису книги визначимо клас BookDescFragment

Ідентичний до BookListFragment за винятком використання макету R.layout.fragment_book_desc.

```
public class BookDescFragment extends Fragment {
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
    View viewHierarchy =
      inflater.inflate(R.layout.fragment_book_desc, container, false);
    return viewHierarchy;
  }
}
```

Конвертація активності для використання фрагментів

Залишки коду в розмітці активності

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

  <!--  List of Book Titles  -->

  <!--  Description of selected book  -->

</LinearLayout>
```

Елемент fragment дозволяє додавати фрагмент до макету, посилаючись на назву класу фрагмен ту в атрибуті name.

```
  <fragment
    android:name="com.jwhh.fragments.BookListFragment"
    android:id="@+id/fragmentTitles"/>
```

Конвертація активності для використання фрагментів

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

  <!-- List of Book Titles -->
  <fragment
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:name="com.jwhh.fragments.BookListFragment"
    android:id="@+id/fragmentTitles"/>

  <!-- Description of selected book -->
  <fragment
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:name="com.jwhh.fragments.BookDescFragment"
    android:id="@+id/fragmentDescription"/>
</LinearLayout>
```

В Android Studio є атрибут tools:layout в елементі fragment.

Він використовується для надання preview макету в графічному дизайнері.

На вигляд додатку він ефекту не спричиняє.

Схожі підходи до розробки

AppDelegate vs MainActivity

AppDelegate та MainActivity є «точками входу» для відповідних додатків.

iOS: AppDelegate

```
class AppDelegate: UIApplicationDelegate {
 func application(_ application: UIApplication,
didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
 //Entry point (In case you do not use storyboards)
 }
}
```

Android: MainActivity

```
public abstract class MainActivity extends AppCompatActivity {
 protected void onCreate(Bundle savedInstanceState) {
 //Entry point
 }
}
```

Схожі підходи до розробки

UIViewController vs Fragment

Обидва класи діють як представники screen UI (або його частини), працюють з подіями життєвого циклу та реалізують взаємодію з користувачем.

Android: Fragment

```
class MyFragment: DialogFragment {
override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?,savedInstanceState: Bundle?): View? {
 //First allocations/inflate
 //View setup will happen on "onViewCreated"
 }
}
```

Події ЖЦ схожі на платформах, проте UIViewController має їх більше.

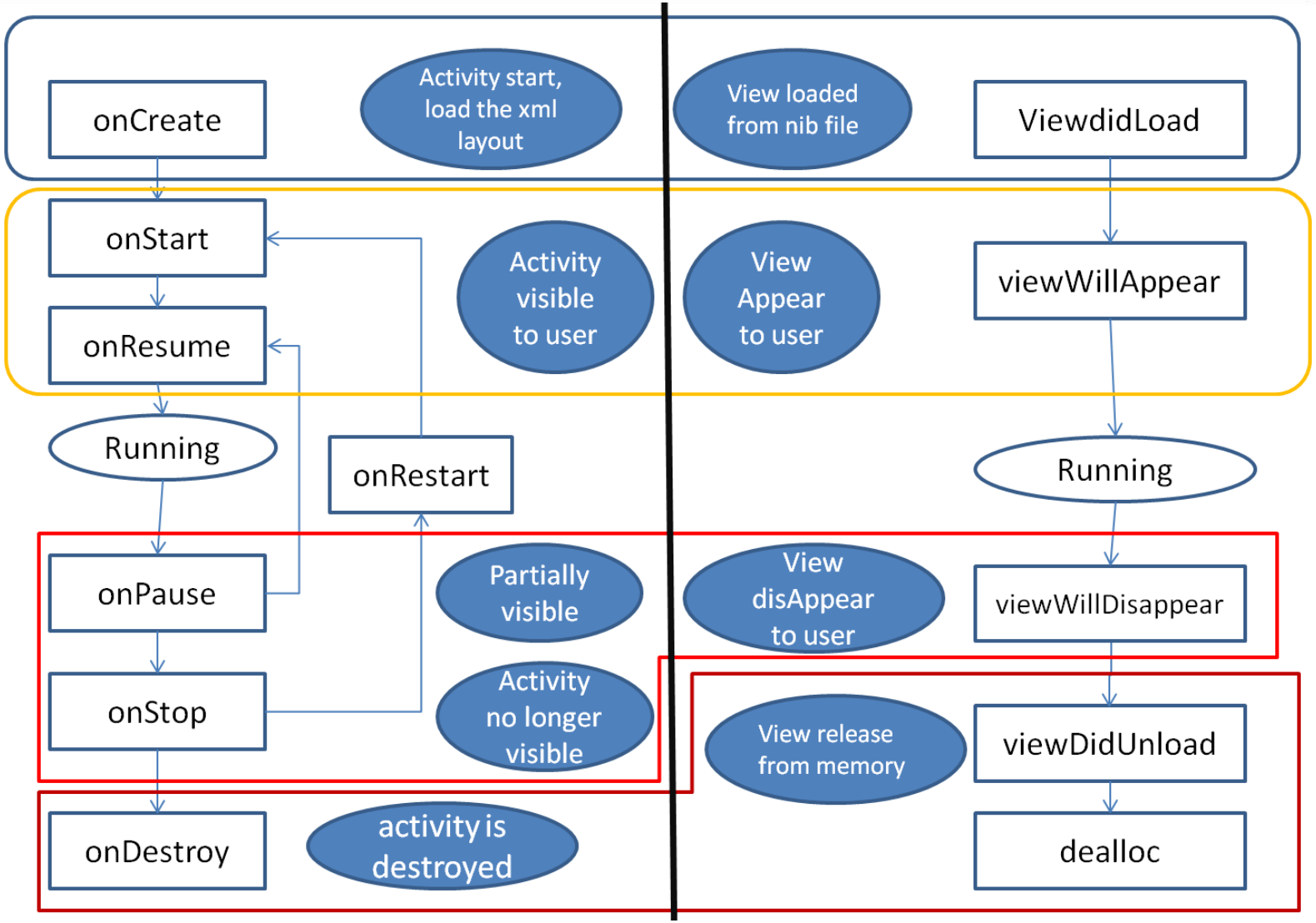
iOS: UIViewController

```
class ViewController: UIViewController{
 override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil:
Bundle?) {
 super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
 //First allocations
 //Some programmers would use "ViewDidLoad" for this purpose
 }
}
```

Порівняння життєвих циклів

Android vs iOS

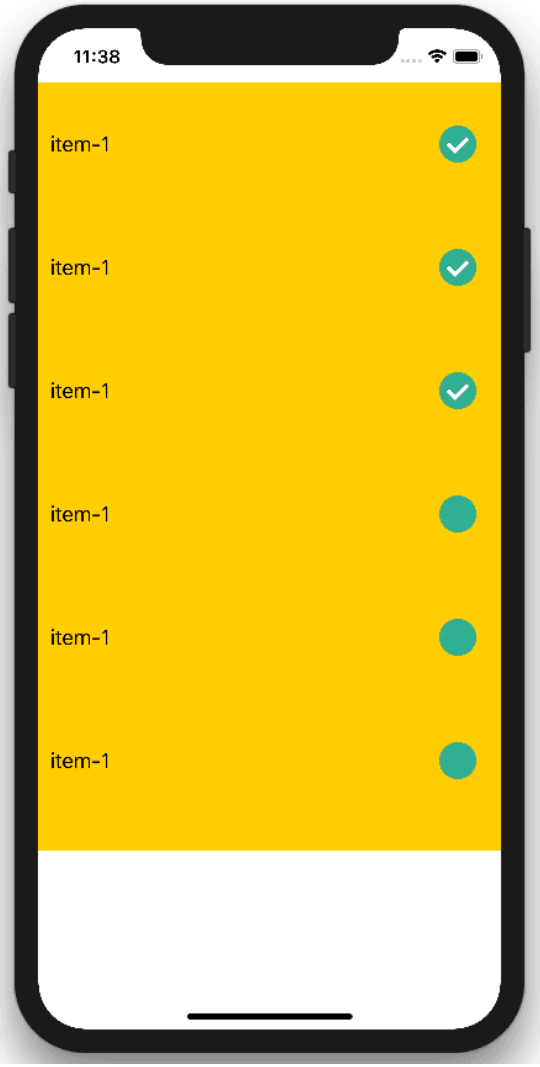
Android



iOS

Схожі підходи до розробки

UITableView vs RecycleView



iOS: UITableView

```
class MyViewController: UIViewController, UITableViewDelegate,
UITableViewDataSource
{
 let tableView = UITableView.init()

override func viewDidLoad() {
  super.viewDidLoad()
  self.tableView.delegate = self
  self.tableView.dataSource = self
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
 //You will get passed a recycle cell, or a new one.
 }
 func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int{
//Number of rows displayed
 }
}
```

Схожі підходи до розробки

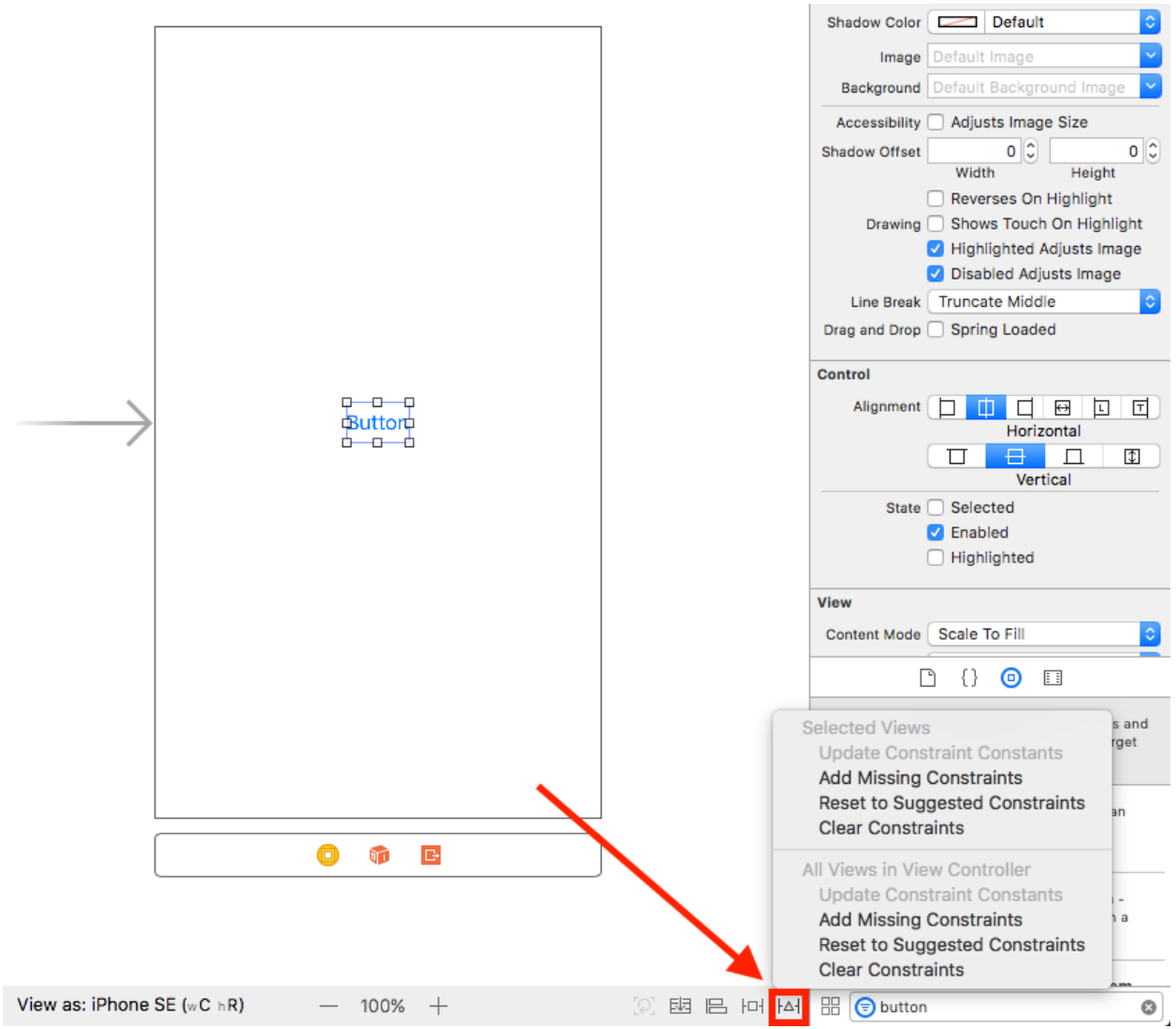
UITableView vs RecycleView

Android: RecycleView

```
class MyFragment : DialogFragment {
 var adapter: MyAdaptor

override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
 super.onViewCreated(view, savedInstanceState)
 homeRecycler.layoutManager = LinearLayoutManager(activity,
LinearLayoutManager.VERTICAL, false)
 homeRecycler.adapter = this.adapter
 }
}

class MyAdaptor : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
 var items: List<HomeFeedItem> = listOf()
override fun onBindViewHolder(holder: RecyclerView.ViewHolder,
position: Int) {
 // Bind ViewHolder to current item props
  if (holder is MyViewHolder) {
    holder.bindViews(items[position])
    return
   }
 }
 override fun getItemViewType(position: Int): Int {
  return items[position].itemType().intValue()
 }

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
 // This is the magic. Inflating will happen only once to improve
performance
 val inflate: CardView =
LayoutInflater.from(context).inflate(R.layout.myLayout, parent,
false) as CardView
 return MyViewHolder(inflate)
 }

fun setFeed(items: List<HomeFeedItem>) {
 //Reload list modals
 this.items = items
 this.notifyDataSetChanged()
 }

 override fun getItemCount() = items.size
}
```

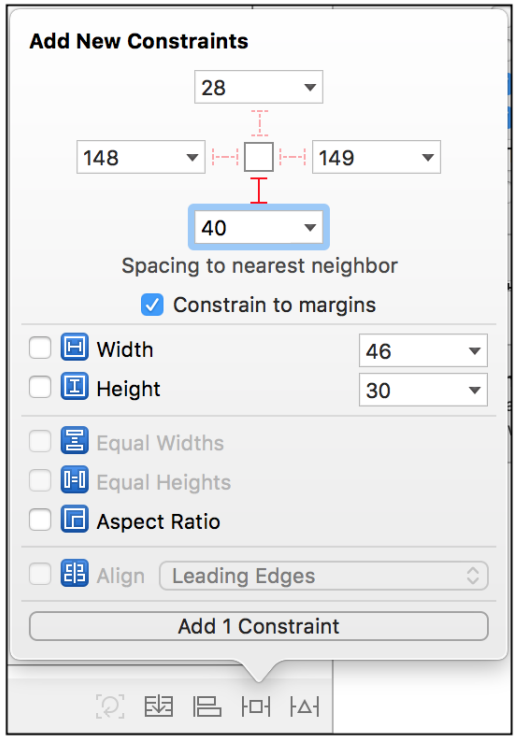
Схожі підходи до розробки

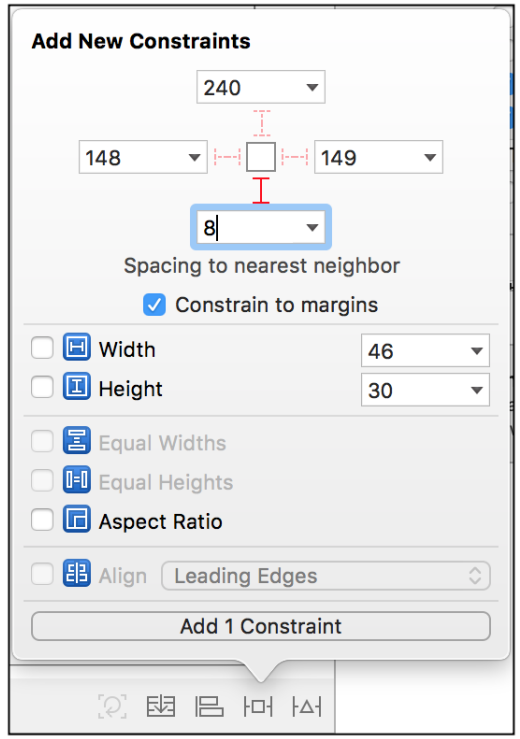
Макетування інтерфейсу: XML vs Constraints



iOS використовує constraint layout (Storyboard).

Обмеження є об’єктами, що визначають зв’язки м іж об’єктами інтерфейсу та можуть додаватись в коді або в Interface Builder.



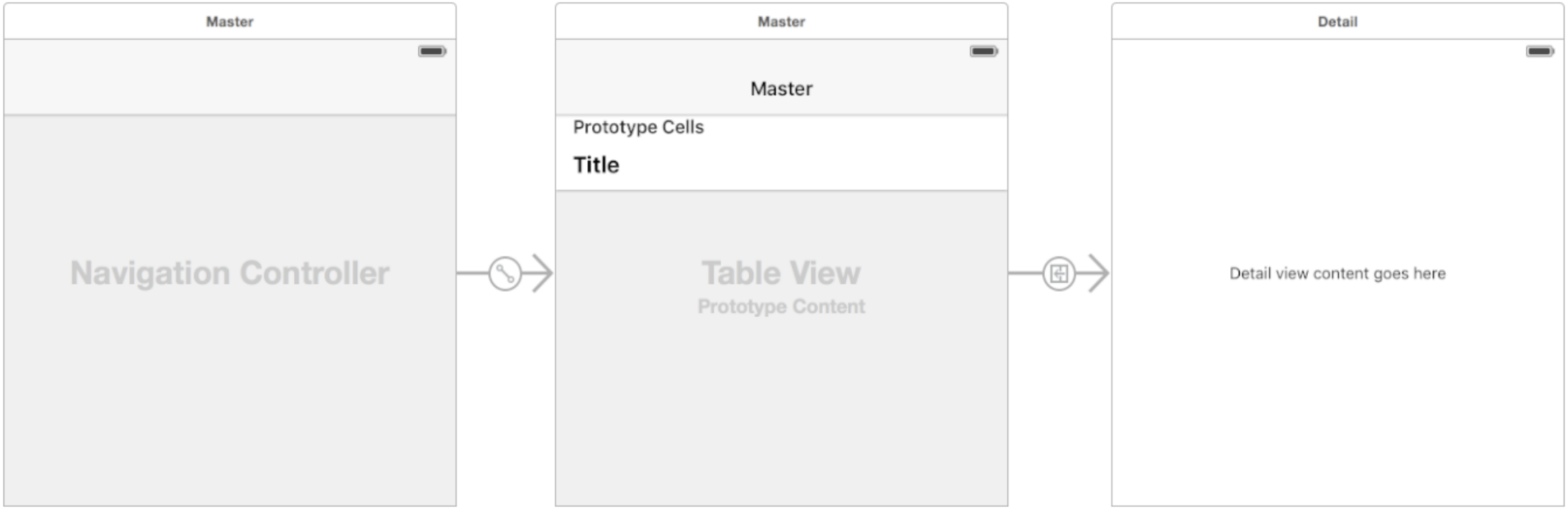


Проте процес макетування дещо відрізняється

Interface Builder оперує основними термінами

Сцени (Scenes) – представляють екранну область. На iPhone та iPod touch екран зазвичай містить 1 с цену, на iPad та Mac екран може компонуватись з кількох сцен.

Сліди між сценами (Segues) – це переходи від однієї сцени до іншої (або між сторібордами)
Controls used to trigger the segues



Комунікація всередині та між додатками

Комунікація всередині та між додатками

Пісочниця додатку в Android та iOS

Android	iOS
App announces permission requirement	Apps have same permissions
Installation-time approval	First-usage time approval
App may have more powerful permissions	Limited permissions

На Android кожен додаток працює з власним “user”-ідентифікатором (UID);

ядро гарантує, що різні “user”-и не зможуть впливати на роботу один одного, мати доступ до файлів один о дного тощо.

Кожен додаток має свою папку.

На iOS всі додатки працюють під одним корист увачем - “mobile”, проте кожному додатку від-во диться власна частина файлової системи.

Також iOS забороняє додаткам здійснювати певні сис темні виклики (sysctl)

Пісочниця також обмежує доступ до камери, контактів, геопозиціонування та ін.

Android та iOS управляють ним за допомогою системи дозволів (app permissions).

Комунікація всередині та між додатками

Керування доступом за допомогою дозволів описується у файлі маніфесту

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.snazzyapp">

   <uses-permission android:name="android.permission.SEND_SMS"/>

   <application ...>
      ...
   </application>
</manifest>
```

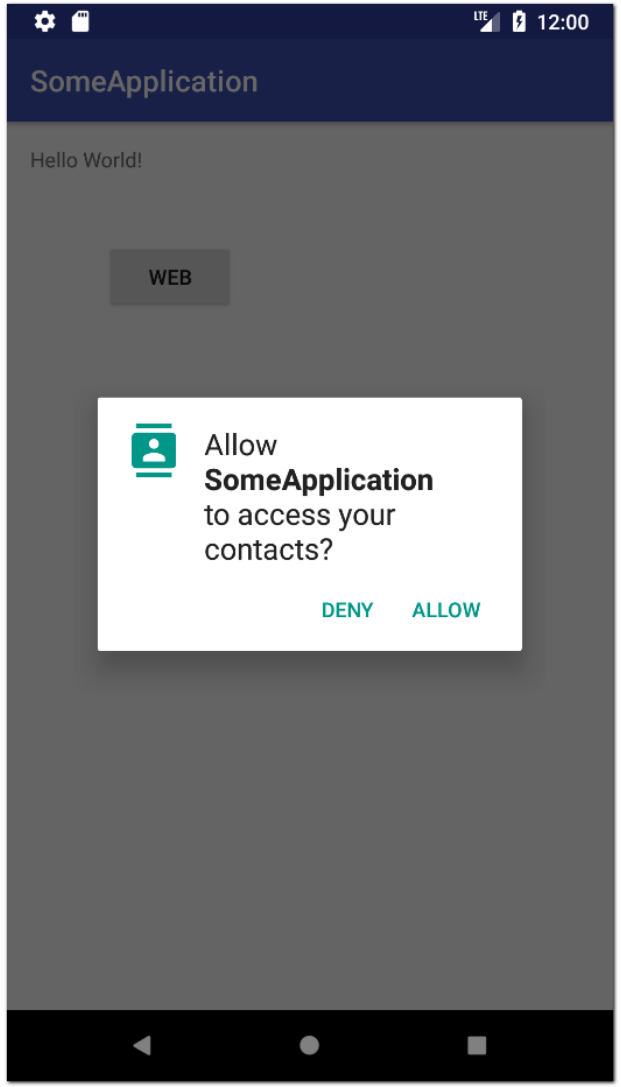
Звичайні дозволи (normal permissions) покривають ситуації, коли додатку треба отрима ти доступ до даних чи ресурсів за межами пісочниці, проте ризик приватності користува ча мінімальний. Наприклад, дозвіл задати часову зону.

Небезпечні дозволи (dangerous permissions) застосовуються в операціях із загрозою осо бистій інформації користувача. Наприклад, здатність зчитувати контакти користувача.

Спеціальні дозволи (special permissions) – кілька дозволів, що не діють як звичайні або небезпечні дозволи: SYSTEM_ALERT_WINDOW та WRITE_SETTINGS .

Комунікація всередині та між додатками

Запит на небезпечний дозвіл



Починаючи з Android 6.0, додаток автоматично отримує всі потрібні normal-дозволи, а dangerous-дозволи необхідно програмно запитувати і під час роботи додатку.

```
int permissionStatus = ContextCompat.checkSelfPermission(this,
                                        Manifest.permission.READ_CONTACTS);

if (permissionStatus == PackageManager.PERMISSION_GRANTED) {
   readContacts();
} else {
   ActivityCompat.requestPermissions(this,
            new String[] {Manifest.permission.READ_CONTACTS},
            REQUEST_CODE_PERMISSION_READ_CONTACTS);
}
```

Комунікація між компонентами

в Android

Одиницею комунікації між компонентами є інтент (клас Intent)

Абстрактний опис операції, яку одна активність хоче виконати з іншою, наприклад, зняти фото чи подзвонити.

Найбільш розповсюджений сценарій – запуск іншої активності в своєму додатку. Також використовуються для

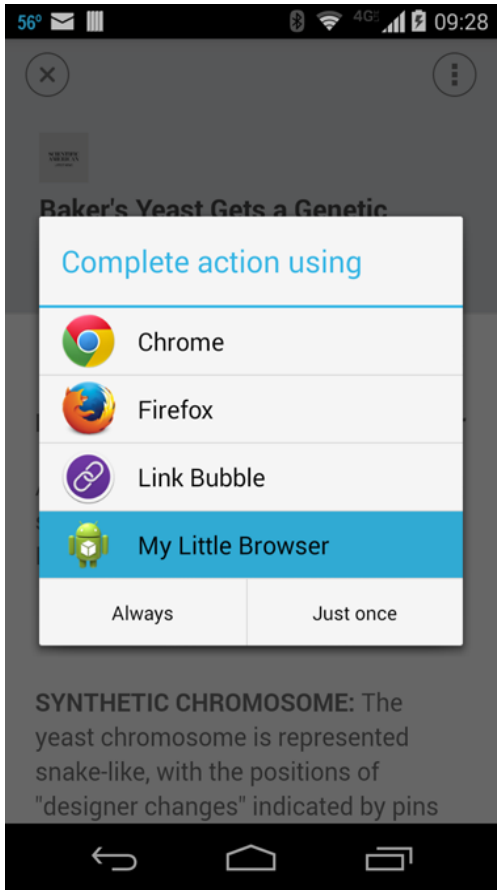
- оголошення про запуск активності або служби, націлених на виконання певних дій (як правило, з певною частиною даних)
- передачі повідомлень про те, що відбулась певна дія або подія.

Будь-який додаток може зареєструвати широкомовний приймач та відстежувати інтенти, реагувати на них.

Android транслює інтенти для оголошення про системні події, наприклад, про зміни у стані підключення до мережі чи в рівні заряду батареї.

Комунікація між активностями

Активності можуть взаємодіяти з іншими активностями



Код, що реалізує одну активність, напряму не викликає код, що реалізує іншу активність.

Додатки описують інтент, який вони бажають виконати та просять систему знайти підходящого виконавця завдання.

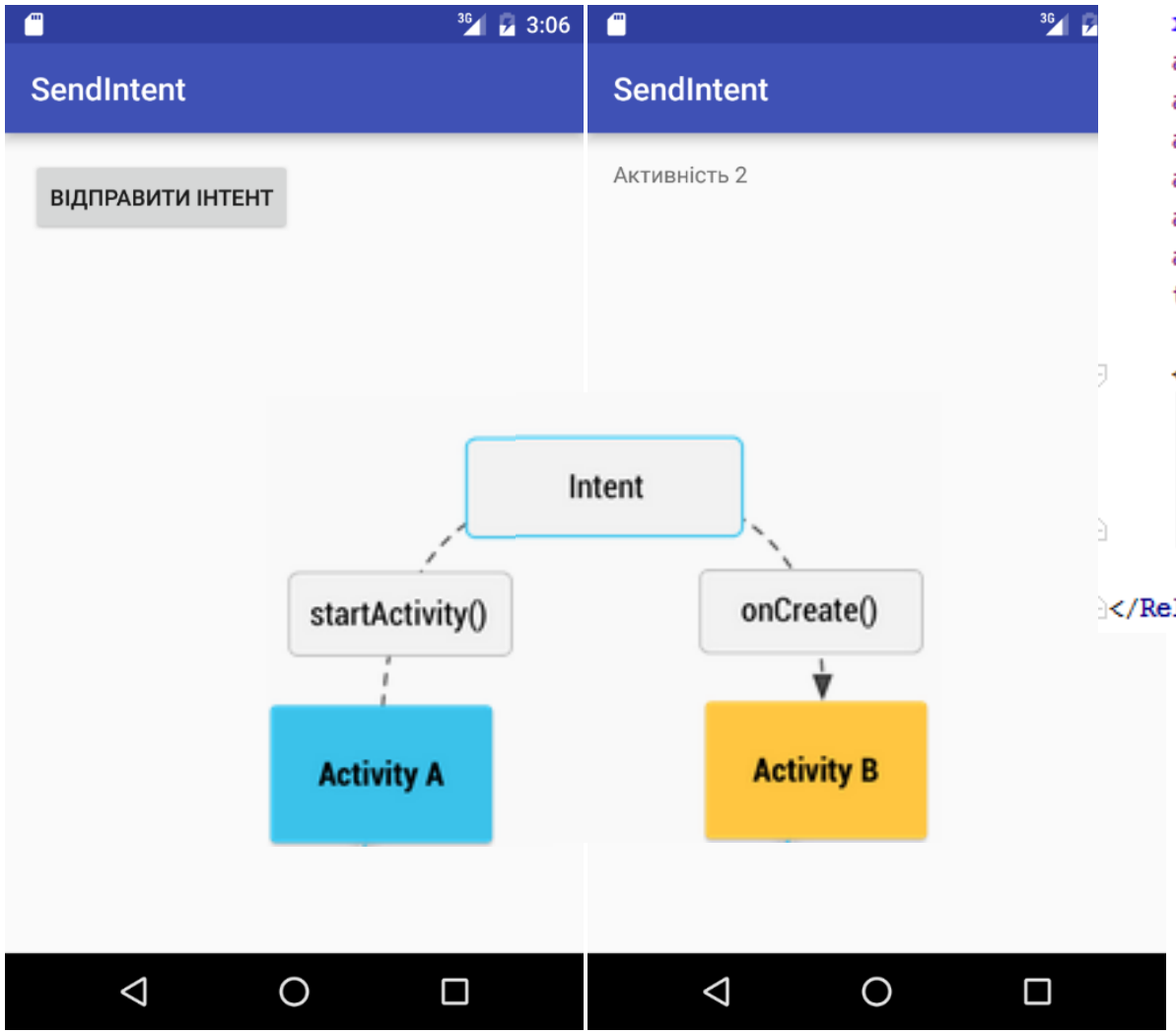
Прояв роботи з інтентом

Коли додаток передає інтент, існує можливість, що кілька активностей зможуть забезпечити виконання бажаної операції.

Робота з таким (неявним) інтентом призводить до появи діалогу

Комунікація між активностями

Запуск однієї активності з іншої




```xml
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.stanislavmarchenko.sendintent.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendIntent" />

</RelativeLayout>
```

```java
public class MainActivity extends AppCompatActivity {

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

    public void sendIntent(View view) {
        Intent intent = new Intent(this, SecondActivity.class);
        startActivity(intent);
    }
}
```

Комунікація між активностями

Два поширених сценарії переходу від однієї активності до іншої

1) ***Явні інтенти:*** Ви знаєте, яку конкретно активність Вам потрібно запустити.
   - явно задають та передають назву цільового компоненту напряму в інтент.

2) ***Неявні інтенти:*** Вам потрібно щось відкрити та дещо зробити. Наприклад, перейти на веб-сторінку, щоб її переглянути. Проте невідомо, який додаток (процес) повинен в иконати дану дію.
   - покладають задачу визначення зареєстрованих компонентів на Android, базуючись на даних і нтенту та передбачену дію (відправку пошти, зйомку фото, відмітку місця на карті тощо).
   - реєстрація компоненту зазвичай відбувається за допомогою інтент-фільтрів

Явні інтенти

Перехід від однієї активності до іншої

Без назви компоненту Android обирає цільовий компонент на основі інформації від інтенту.

```
Intent intent = new Intent(this, LoginActivity.class);
startActivity(intent);
```

Намагаємось запустити LoginActivity.java з MainActivity.java, розміщеному на вершині стеку активностей.

- Можемо додати до інтенту пари «ключ-значення», щоб отримати їх у цільовій активності. Ключ повинен бути типу String, а значення – як примітивного (int, double тощо), так і посилкового типу (CharSequence, String, Bundle, Parceable, Serializable).

```
Intent intent = new Intent(this, MainActivity.class);
intent.putExtra("name", "rishabh");
intent.putExtra("age", 21);
startActivity(intent);
```

Отримування даних з інтенту

Перехід від однієї активності до іншої

Виклик getExtras() для Intent-об’єкта дає об’єкт Bundle, для якого можна викликати багато методів: getString(), getInt(), getChar() тощо, щоб отримати передані дані.

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
  String name = extras.getString("name");
  int age = extras.getInt("age");
}
```

Замість роботи з bundle-об’єктами можна використовувати методи Intent-об’єкта, такі як getStringExtra() та getIntExtra().

Приблизний вигляд маніфесту

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.pycitup.pyc" >
    <application    android:name="com.pycitup.pyc.PycApplication"
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity    android:name="com.pycitup.pyc.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <activity     android:name="com.pycitup.pyc.LoginActivity"
        android:label="@string/title_activity_login" >
      </activity>
   </application>
</manifest>
```

Повернення даних із запущеної активності

Ви запускаєте активність, яка дозволяє користувачу обирати особу зі списку контак-тів, зображення з галереї тощо

Коли вибір зроблено, його результат треба повернути батьківській активності.
Замість startActivity() викликаємо startActivityForResult():

```
Intent intent = new Intent(this, LoginActivity.class);
startActivityForResult(intent, 1);
```

Виклик startActivityForResult() з додатним цілим request-кодом запускає LoginActivity як субактивність MainActivity.

- При поверненні інтенту від субактивності відсилається той же request-код, щоб коректно визначити результат та спосіб його обробки.

Повернення даних до батьківської активності

Якщо бажаєте відправити деякі дані назад до батьківської активності:

```
Intent resultIntent = new Intent();
resultIntent.putExtra("result", "Excellent!");
setResult(RESULT_OK, resultIntent);
finish();
```

Інакше:

```
Intent resultIntent = new Intent();
setResult(RESULT_CANCELED, resultIntent);
finish();
```

Результуючі коди RESULT_OK and RESULT_CANCELED позначають успіх чи невдачу здійснення операції.
Якщо субактивність падає, батьківська активність отримує код RESULT_CANCELED.
Цей response-код може задаватись й іншими значеннями.

Обробка отриманих даних в батьківській активності

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == 1) {
        if (resultCode == RESULT_OK) {
            String result = data.getStringExtra("result");
            Log.d(TAG, result);
        }

        if (resultCode == RESULT_CANCELED) {
            // Operation failed or cancelled. Handle in your own way.
        }
    }
}
```

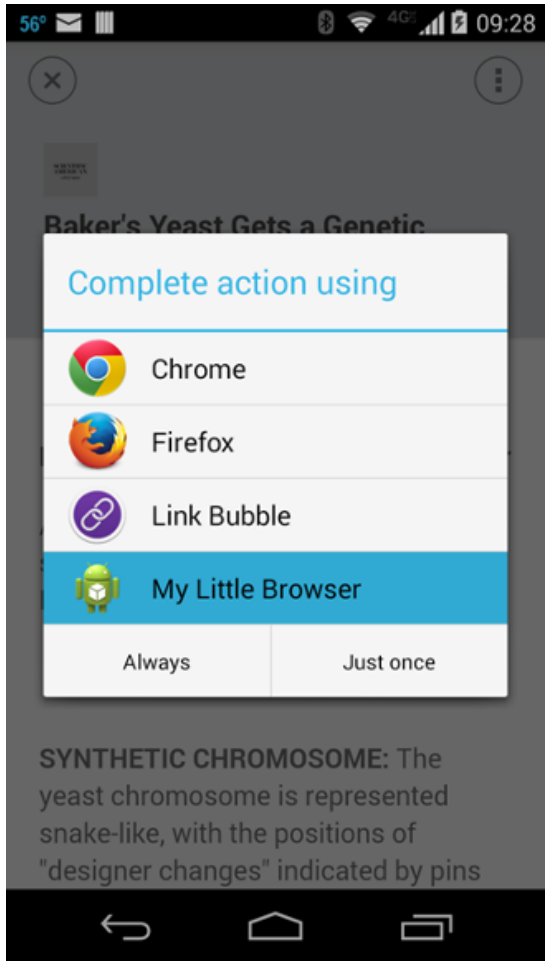
requestCode – ціле значення, що допоможе ідентифікувати коректну операцію, від якої надійшов resultCode.

Неявні інтенти

При роботі з неявними інтентами загалом задається бажана дія та опційно передаються дані для виконання цієї дії.
Зазвичай дані виражаються у формі Uri:

```
// з MainActivity.onCreate()
Intent intent = new Intent(Intent.ACTION_VIEW,
            Uri.parse("http://google.com"));
startActivity(intent);
```

Задається дія (ACTION_VIEW) на базі HTTP URI, яка відображає веб-сторінку.
Система Android шукає всі зареєстровані компоненти (через інтент-фільтри) на конкретну дію та тип даних (тут HTTP URI).
Якщо знайдено не один компонент, запускається діалог.



Інший приклад неявного інтенту

Користувач може захотіти поділитись частиною даних/тексту/оновлення/повідомлення за допомогою месенджерів на зразок Facebook, Gmail, WhatsApp, Twitter, Snapchat тощо.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(android.content.Intent.EXTRA_TEXT, "Standing on the Moon!");
startActivity(intent);
```

Якщо список месенджерів має з'являтись завжди, останній рядок можна замінити на

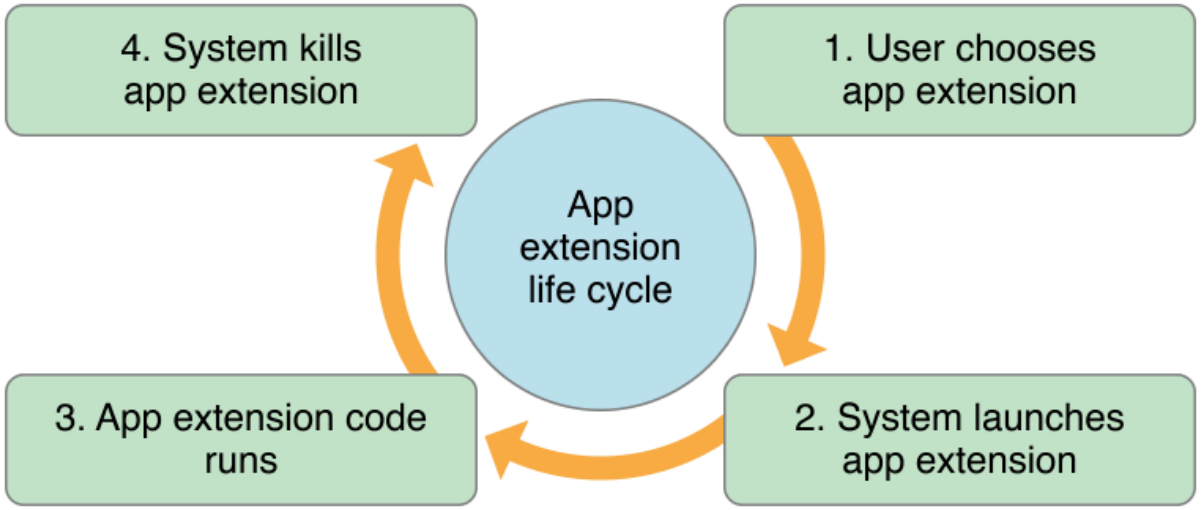
```
Intent chooser = Intent.createChooser(intent, "Chooser Title!");
startActivity(chooser);
```

Для визначення наявності додатку, що обробить інтент слід викликати метод resolveActivity() для об'єкта-інтенту:

```
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

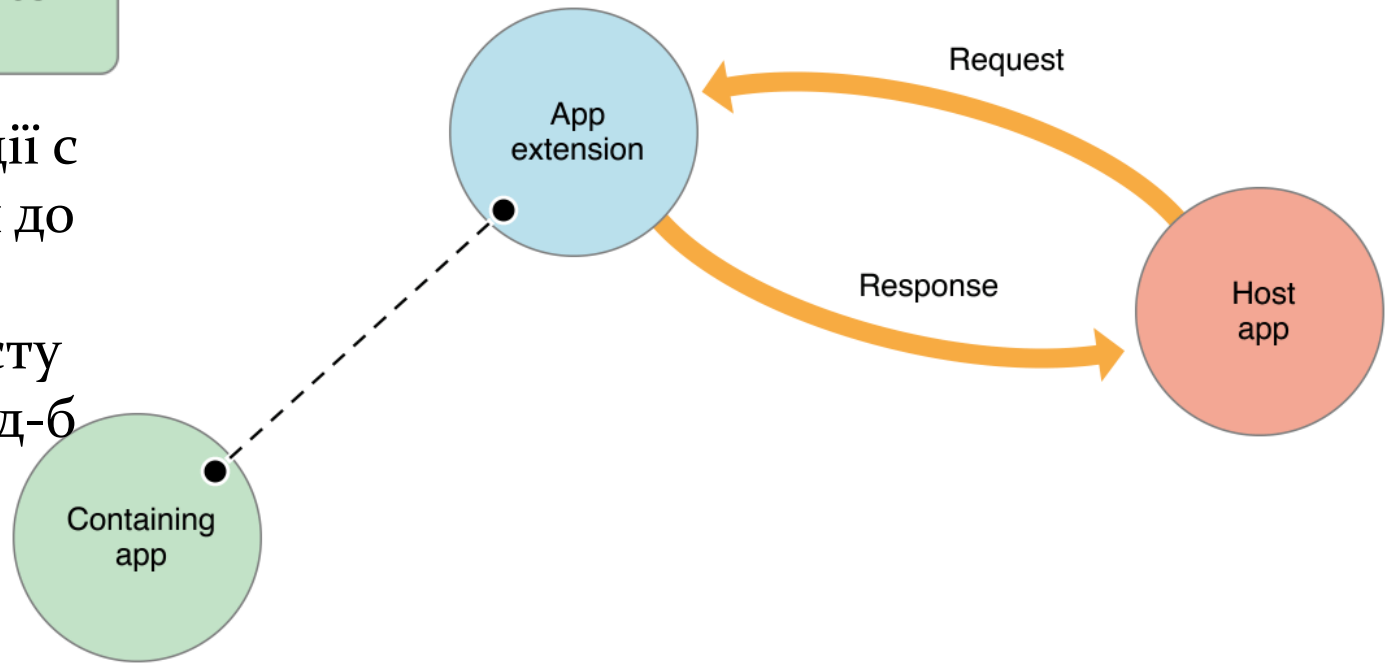
Комунікація в iOS за допомогою розширень додатку

iOS App Extensions



Розширення є специфічною задачею (task), а не додатком, та має власний життєвий цикл. Прямої комунікації між розширенням та its c ontaining app немає;

При типовій «request/response» транзакції с истема відкриває розширення зі сторони до датку-хосту, надаючи контекст. Розширення відображає інтерфейс користу вача, виконує деяку роботу та, якщо перед-б ачено, повертає певні дані хосту.



Контекст

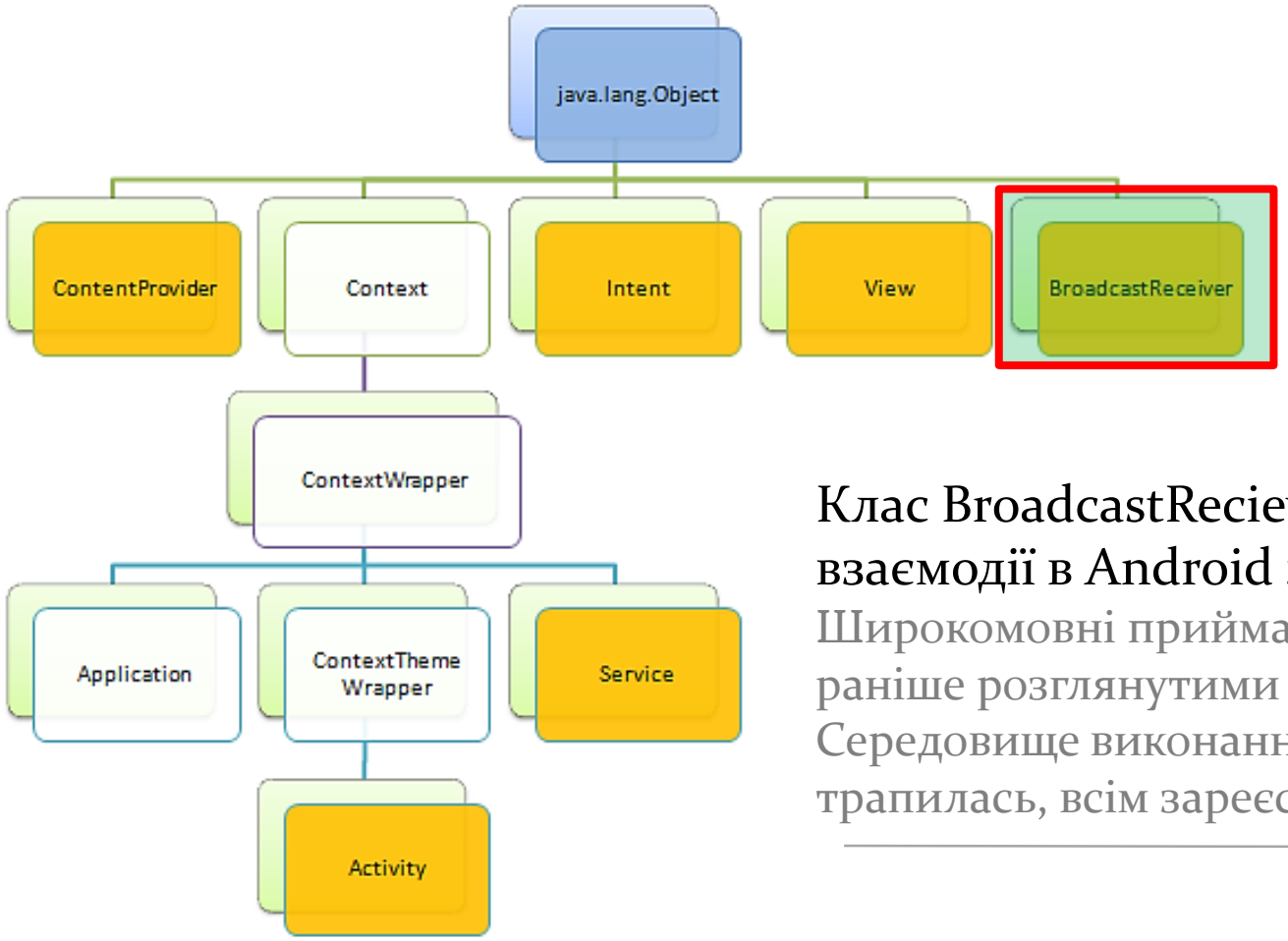
Теми для самоосвіти та самостійного опрацювання

2. Міжпроцесна комунікація в Android та iOS
http://www.ryantzj.com/the-mystical-android-sandbox-and-ipc-inter-process-communication.html

One interesting difference between iOS app extensions and the Android counterpart is that iOS app extensions do not run in the context of their containing app. If the extension wants to share data with or launch the containing app for further processing, it must do so explicitly.

Широкомовні приймачі

Broadcast Receivers



Система або додатки час від часу розсилають бродкасти

Від сповіщення про низький заряд до повідомлення про вимкнення екрану чи зміни стану підключення до Wi-Fi.

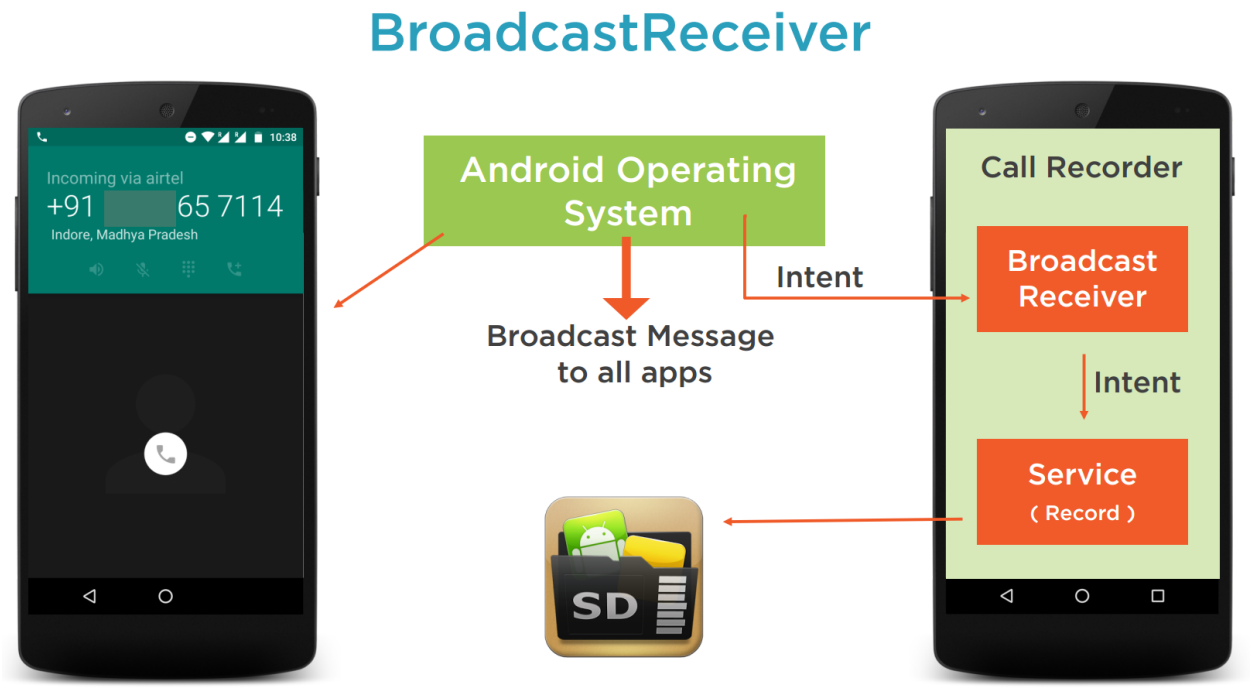
Клас BroadcastReciever реалізує варіант високорівневої міжпроцесної взаємодії в Android за допомогою Intent-об'єктів.

Широкомовні приймачі мають простіший життєвий цикл у порівнянні з раніше розглянутими компонентами.

Середовище виконання (Android runtime) повідомляє про подію, що трапилась, всім зареєстрованим на цю подію приймачам.

Приклад роботи широкомовного приймача

Відбувається вхідний дзвінок



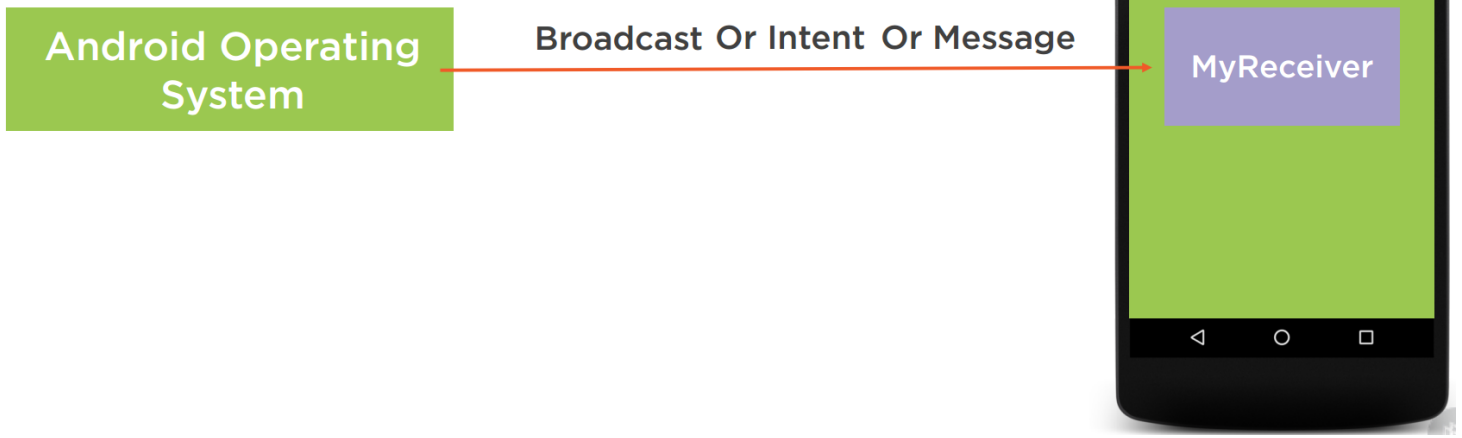
1. ОС Android розсилає (broadcast) сповіщення усім додаткам в системі: чи бажаєте відповісти на цю подію?
2. Якщо якийсь із додатків підписаний на цю подію (тут – записувач дзвінка), ОС доставляє бродкаст за допомогою широкомовного приймача додатку.
3. Від широкомовного приймача передається інтент для запуску служби запису розмови.

1. Дзвінок завершено.
2. ОС надсилає бродкаст про цю подію для записувача дзвінка.
3. Широкомовний приймач додатка зупиняє службу запису та зберігає запис розмови у файл.

Типи широкомовних сповіщень (бродкастів)

- Normal Broadcasts
- Ordered Broadcasts
- Sticky Broadcasts
- Local Broadcasts

Бродкасти можуть надсилатись від ОС, іншого додатку або навіть всередині додатку
Насправді, надсилається інтент, тобто бродкаст = інтент = повідомлення = подія. Можливо, з даними.



Відмінності різних видів бродкастів

- ***Normal Broadcast*** – асинхронний бродкаст, що використовує sendBroadcast(). Будь-який широкомовний приймач отримує бродкаст без визначеного порядку.

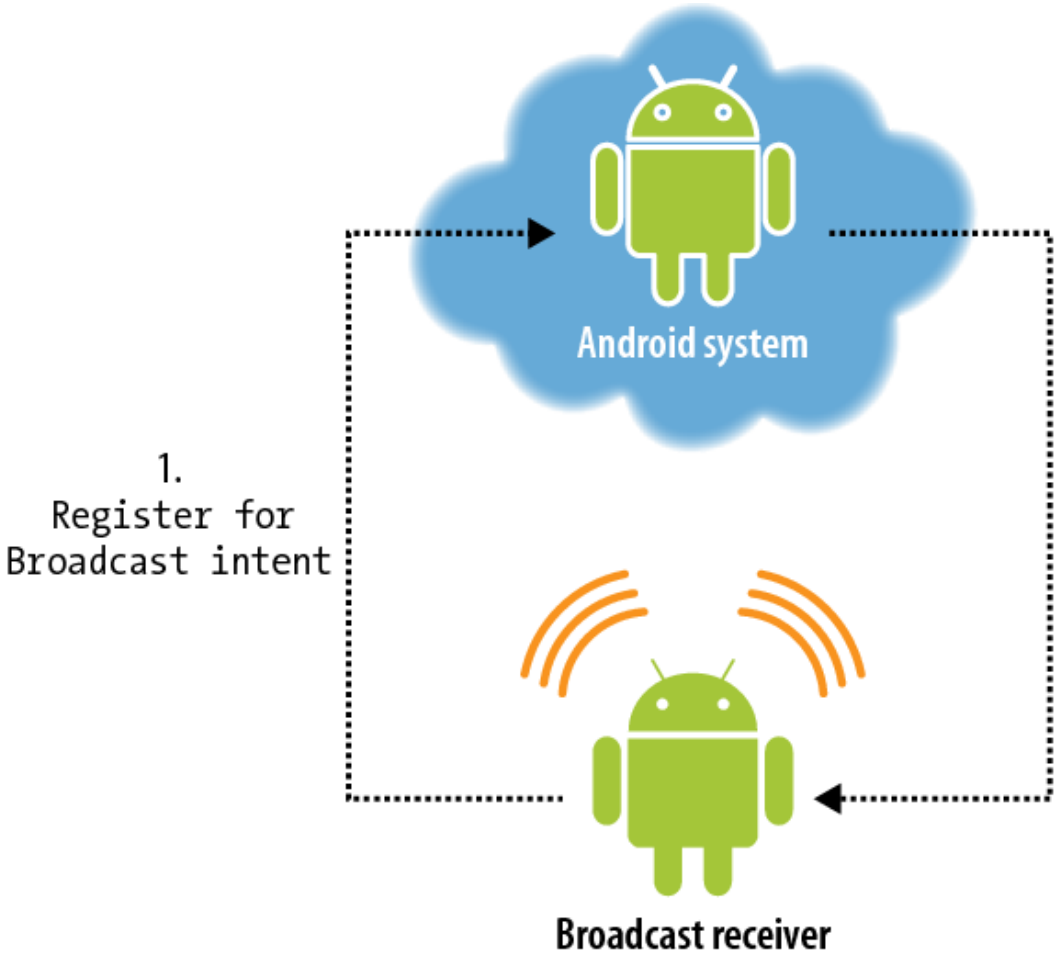
- ***Ordered Broadcast*** – синхронний бродкаст, що використовує sendOrderedBroadcast(). Широкомовний приймач отримує бродкаст за пріоритетом. Такий бродкаст можна відмінити (abort).

- ***Local Broadcast*** використовується лише всередині додатку.

- ***Sticky Broadcast*** – використовує sendStickyBroadcast(Intent). Нормальний бродкаст після відправки та обробки системою стає недоступним. Липкий (sticky) інтент означає, що після виконання бродкасту він залишається в системі. Тому решта компонентів може швидко отримати дані за допомогою registerReceiver(BroadcastReceiver, IntentFilter).

Широкомовні приймачі

Broadcast Receivers



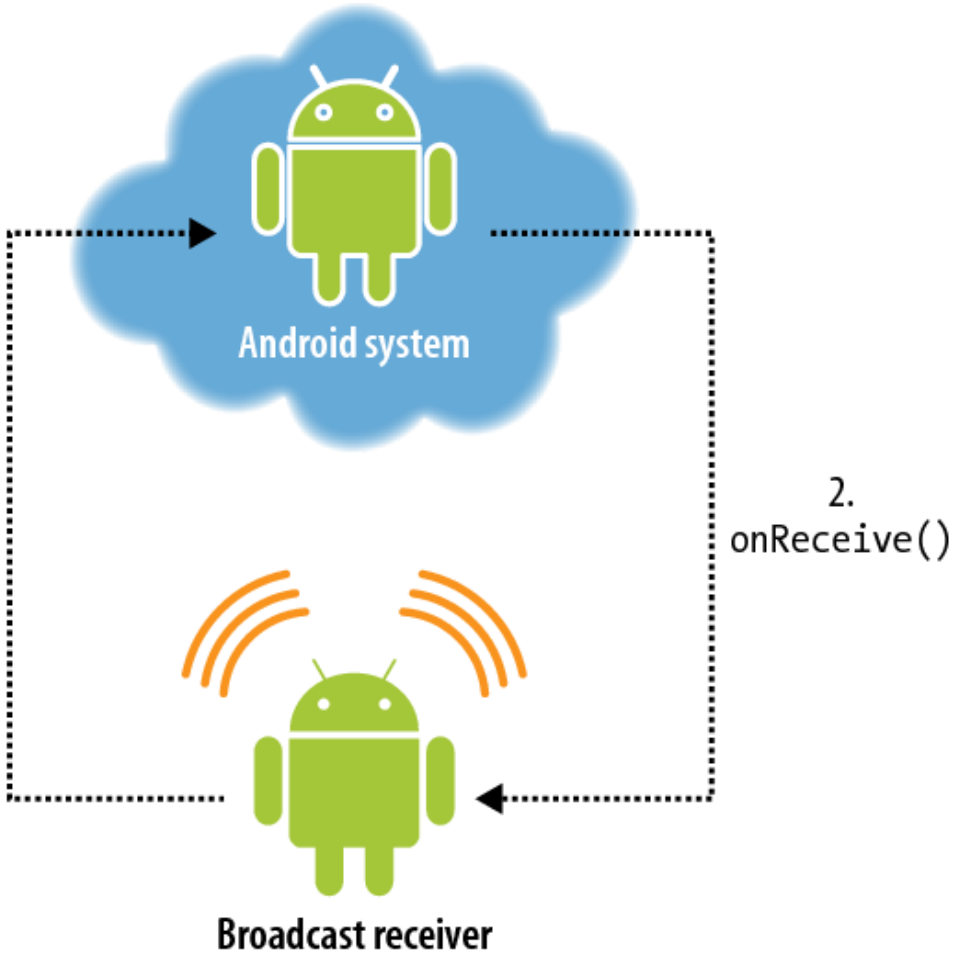
Основні системні події

Intent.ACTION_BOOT_COMPLETED
Intent.ACTION_POWER_CONNECTED
Intent.ACTION_POWER_DISCONNECTED
Intent.ACTION_BATTERY_LOW
Intent.ACTION_BATTERY_OKAY

```
<receiver android:name="MyScheduleReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver android:name="MyStartServiceReceiver" >
</receiver>
```

Широкомовні приймачі

Broadcast Receivers




```java
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class MyReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // assumes WordService is a registered service
                Intent intent = new Intent(context, WordService.class);
                context.startService(intent);
        }
}
```

Зміни в системних бродкастах

Якщо додаток встановлено на Android 7.0+ або цільовий рівень SDK вище 24

Android 9 (API level 28)

- Бродкаст NETWORK_STATE_CHANGED_ACTION не отримує інформацію щодо місця розташування користувача чи його персональні (personally identifiable) дані.
- Якщо додаток встановлено на Android 9+, системні бродкасти від Wi-Fi не міститимуть SSID-и, BSSID-и, connection information чи результати сканування. Для отримання цієї інформації викликайте getConnectionInfo().

Android 8.0 (API level 26)

- Система накладає додаткові обмеження на записані в маніфесті приймачі.
- Якщо рівень цільового SDK 26+, неможливо використовувати маніфест для оголошення приймача для більшості неявних бродкастів (implicit broadcasts – не націлених конкретно на Ваш додаток). Можна використовувати context-registered receiver, коли користувач активно працює в додатку.

Android 7.0 (API level 24)

- Не надсилаються системні бродкасти ACTION_NEW_PICTURE та ACTION_NEW_VIDEO.
- Додатки з рівнем SDK 24+ повинні реєструвати бродкаст CONNECTIVITY_ACTION за допомогою registerReceiver(BroadcastReceiver, IntentFilter). Оголошення приймача в маніфесті не допомагає.

Контекст

Теми для самоосвіти та самостійного опрацювання

Робота широкомовних приймачів та сповіщень подана оглядово. Бажано детально порівняти підсистеми сповіщень в iOS та Android

3. ***Підсистеми для роботи зі сповіщеннями в Android та iOS.***

Принципи виконання фонових процесів

служби та постачальники контенту

Процесна модель Android

Після тапу по іконці виконується перехід від поточної активності (наприклад, лаунчер) до головної активності запущеного додатку

За кулісами цієї події операційна система Android виконує форк (fork – запуск окремої копії) процесу zygote.

Zygote, також відомий як «app process», - батьківський процес для всіх процесів Android-додатків
Запускається при завантаженні системи

Новий процес містить:

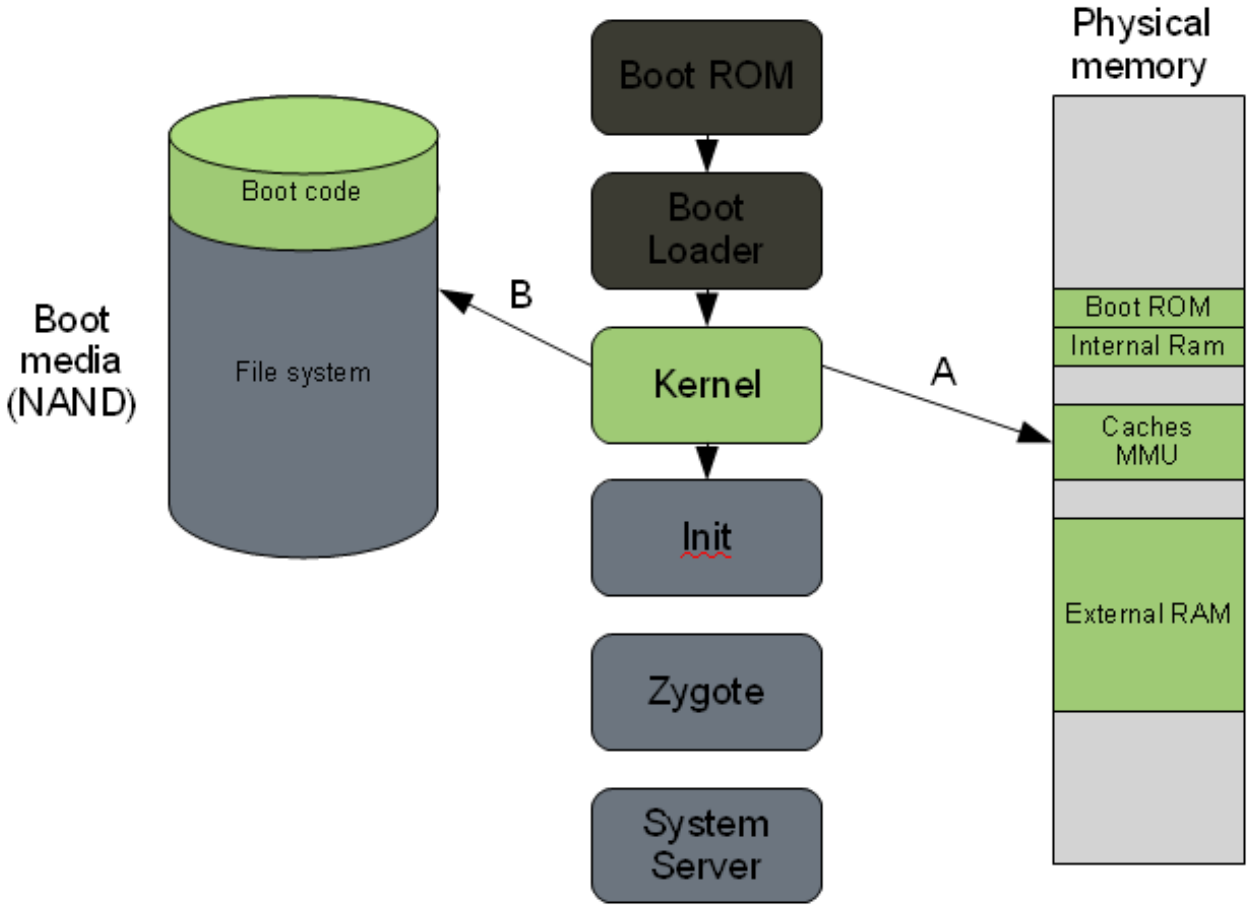
- Копію віртуальної машини (Dalvik або ART)
- Копію класів фреймворку Android (як Activity, Button та ін.)
- Копію власних класів, що була запакована разом з APK-файлом
- Всі об’єкти, створені з Ваших класів чи класів фреймворку

Платформа Android виконує кожен додаток у пісочниці

- Кожен застосунок працює із власним екземпляром віртуальної машини ART

Запуск процесу при холодному старті

+ перебування процесу в пам’яті



Фактори тривалості перебування процесу в пам’яті:

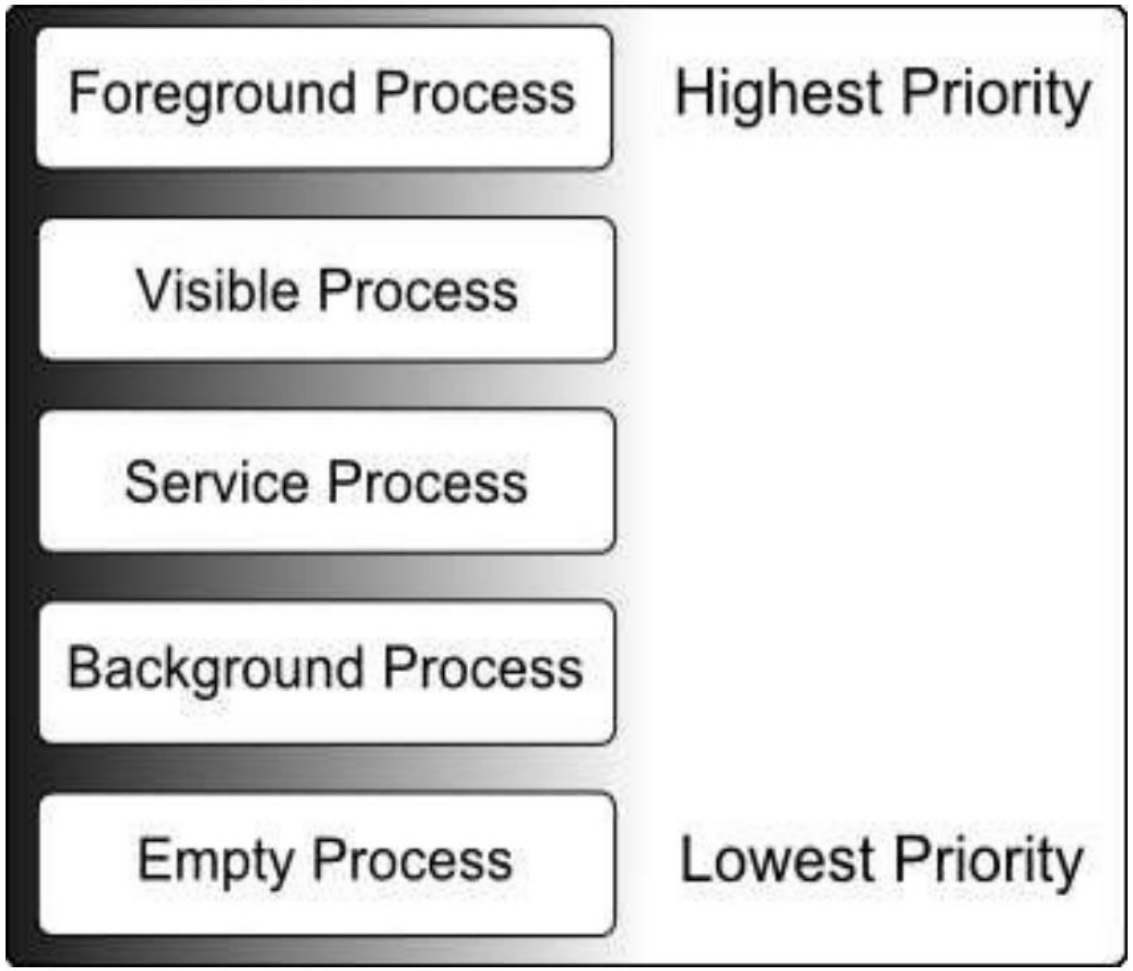
- Інші додатки, на передньому (foreground), та задньому (background) планах
- Скільки пам’яті має пристрій
- Що все ще працює всередині процесу

Існують обґрунтовані припущення, що грають роль і інші фактори, зокрема, час існування процесу (process age).

Натиснення кнопок HOME та BACK спричинить різний ефект: у першому випадку процес залшиться в системі трохи довше, ніж у другому.

Пріоритетність процесів

Ієрархія важливості процесів



Параметри переривання процесу:

- пріоритет процесу
- стан усіх запущених на даний момент процесів.

За допомогою алгоритму компанії Google будується так звана ієрархія важливості (importance hierarchy).

Загальний аналіз процесів ускладнюється ще тим, що процеси можуть залежати один від одного (inte r-dependent processes).

Базовим правилом, що описується в документації по Android, є наступне: процес ніколи не ранжуєть ся нижче, ніж процес, який його в даний момент в икористовує.

Служби (Services)

Життя активності короткотривале, може скінчитись у будь-який момент

Служби спроектовані так, щоб досить довго продовжувати працювати незалежно від активностей.

Служби потрібні при різного роду фонових оновленнях, відтворенні музики, навіть якщо основна активність вже завершила свою роботу.

Також служби використовуються для запланованих задач (scheduled tasks).

Клас Service використовується для створення фонових задач, які можуть бути активними, але невидимими на екрані.

Служби дозволяють додаткам ділитись функціями за допомогою довготривалих (long-term) зв'язків.

Така практика важлива і для інтернет-служб типу FTP та HTTP, які очікують запиту від клієнта, що їх запустить.

Платформа Android намагається уникати відбирання ресурсів у служб, тому дуже ймовірно, що після запуску служби вона буде доступною до моменту, доки пам'яті стане надзвичайно мало.

Системні (Platform) та кастомні служби

- Платформа Android постачає та виконує передвстановлені системні служби, які можуть використовувати додатки з відповідними дозволами.
  - Зазвичай за підключення до системної служби відповідає спеціальний Manager-клас.
  - Доступ до них виконує метод getSystemService().
  - Клас Context визначає кілька констант для доступу до цих служб.

- Крім використання існуючих служб, можна визначати та користуватись власними.
  - Це дозволяє проектувати респонсивні додатки.
  - За їх допомогою можна fetch дані додатку. Як тільки додаток буде запущено користувачем, він може о новити дані.

- Кастомні служби запускаються іншими компонентами Android.

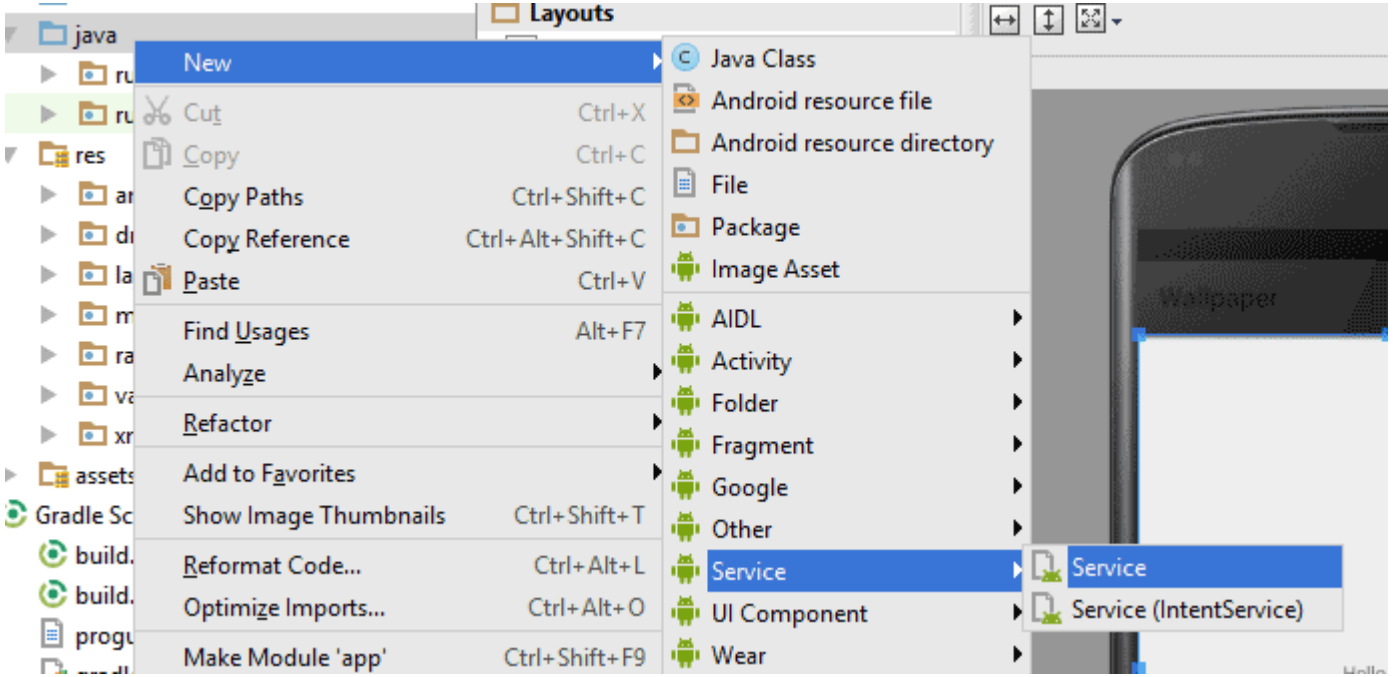
Foreground services

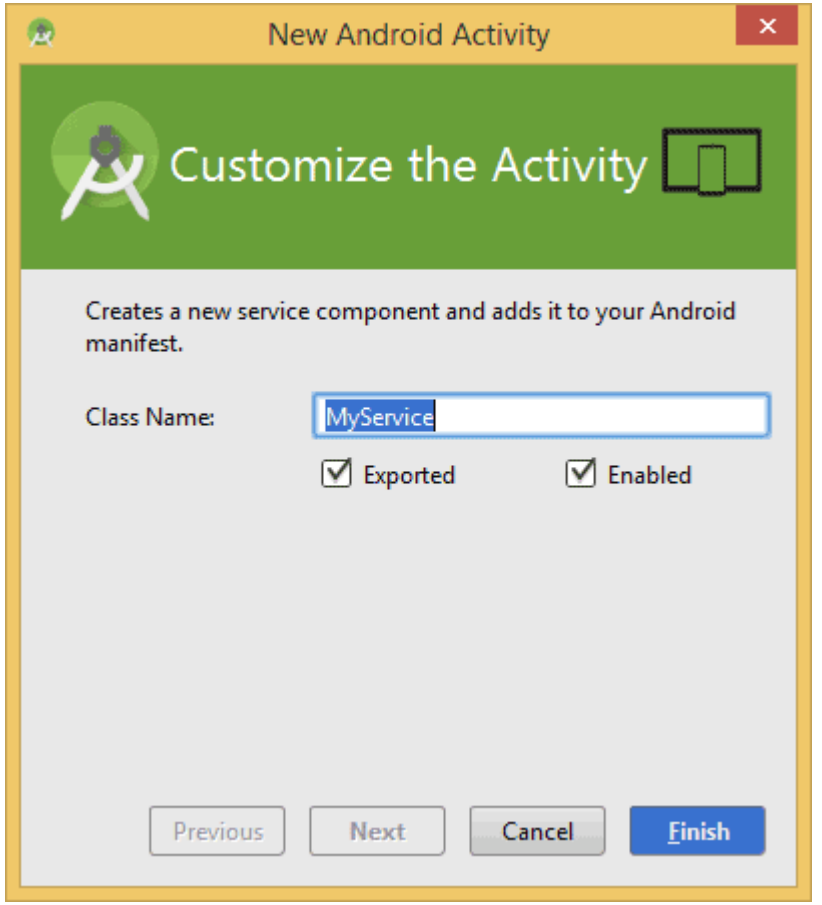
- foreground service – служба, що має пріоритет, аналогічний активній активності, тому не вбивається с истемою навіть при нехватці пам’яті.
- foreground service повинен забезпечувати нотифікації в панелі статусу, що розміщуються під "Ongoing" heading (службу не можна відмінити, зупинити чи видалити з foreground).

```
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker_text),
                                                                System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
                                                        getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

Створення служби

Атрибут exported дає можливість іншим додаткам отримати доступ до вашої служби

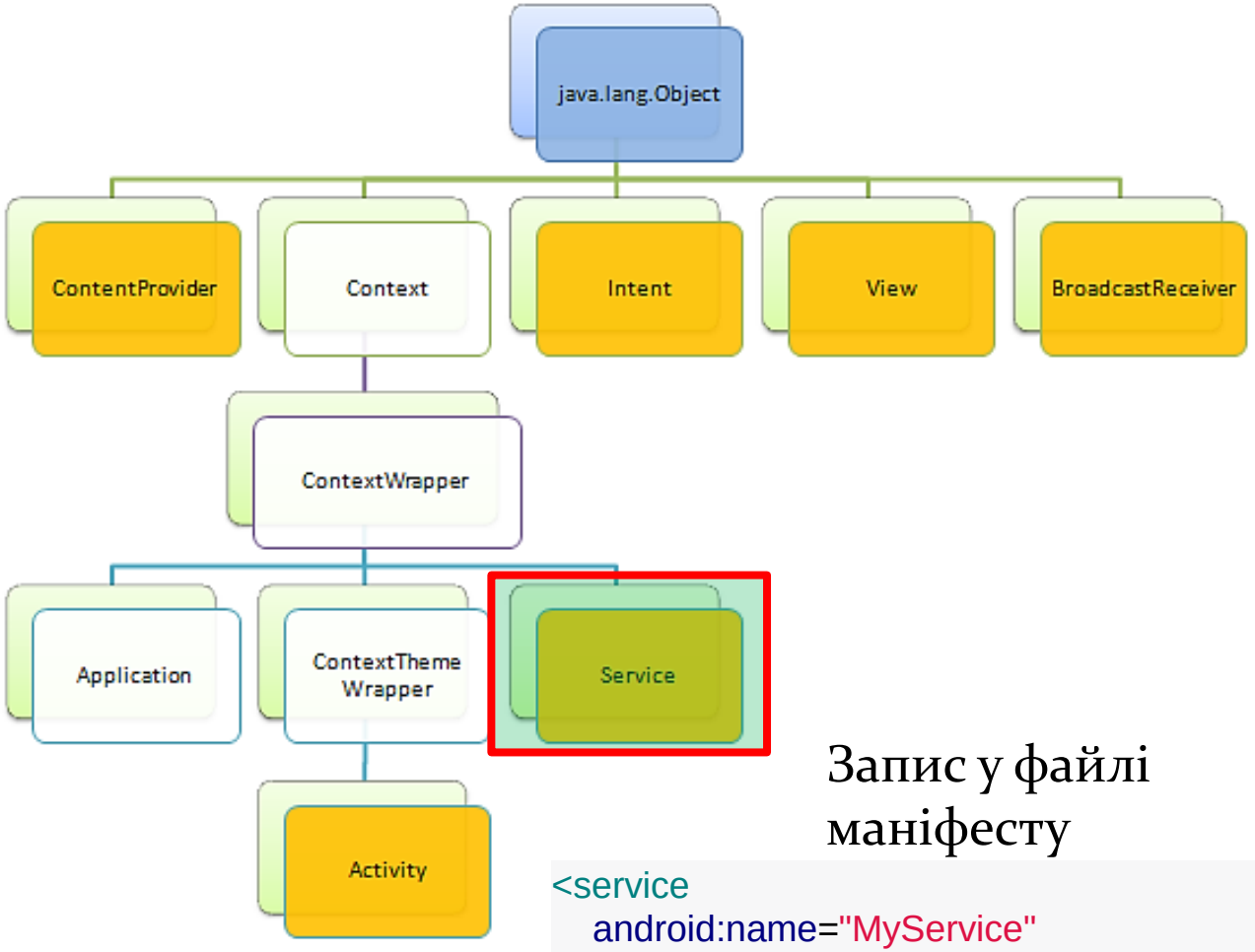




- Присутні й інші атрибути, наприклад, permission, щоб служба запускалась лише вашим додатком.
- Налаштовуються у файлі маніфесту.

http://www.vogella.com/tutorials/AndroidServices/article.html

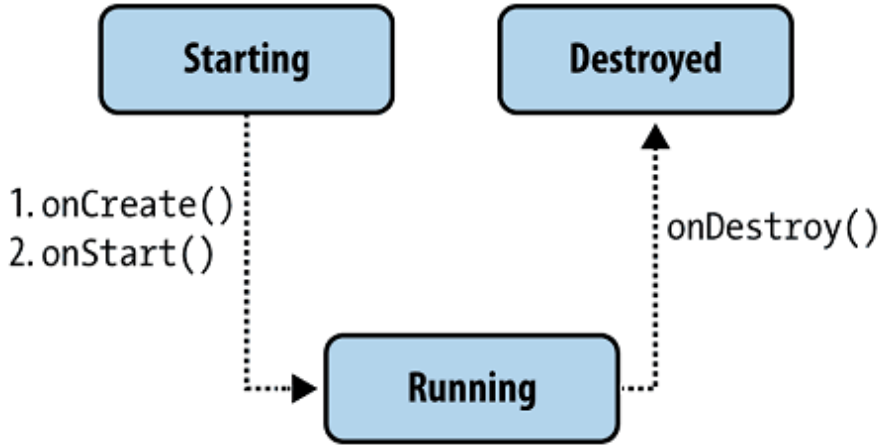
Служби



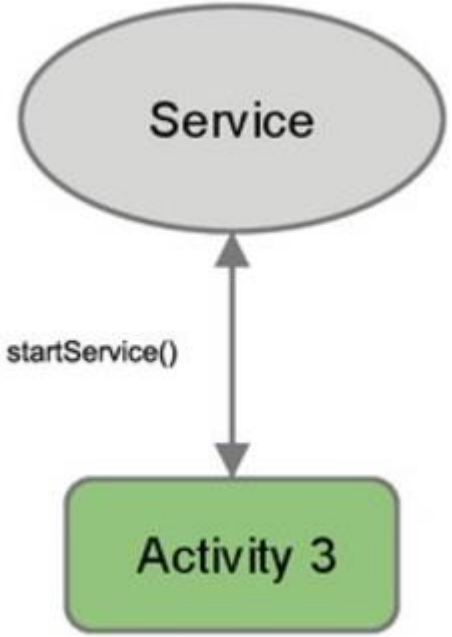

```
public class MyService extends Service {
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      //TODO do something useful
      return Service.START_NOT_STICKY;
   }

   @Override
   public IBinder onBind(Intent intent) {
      //TODO for communication return IBinder implementation
      return null;
   }
}
```

Запис у файлі маніфесту

```
<service
   android:name="MyService"
   android:icon="@drawable/icon“
   android:label="@string/service_name" >
</service>
```

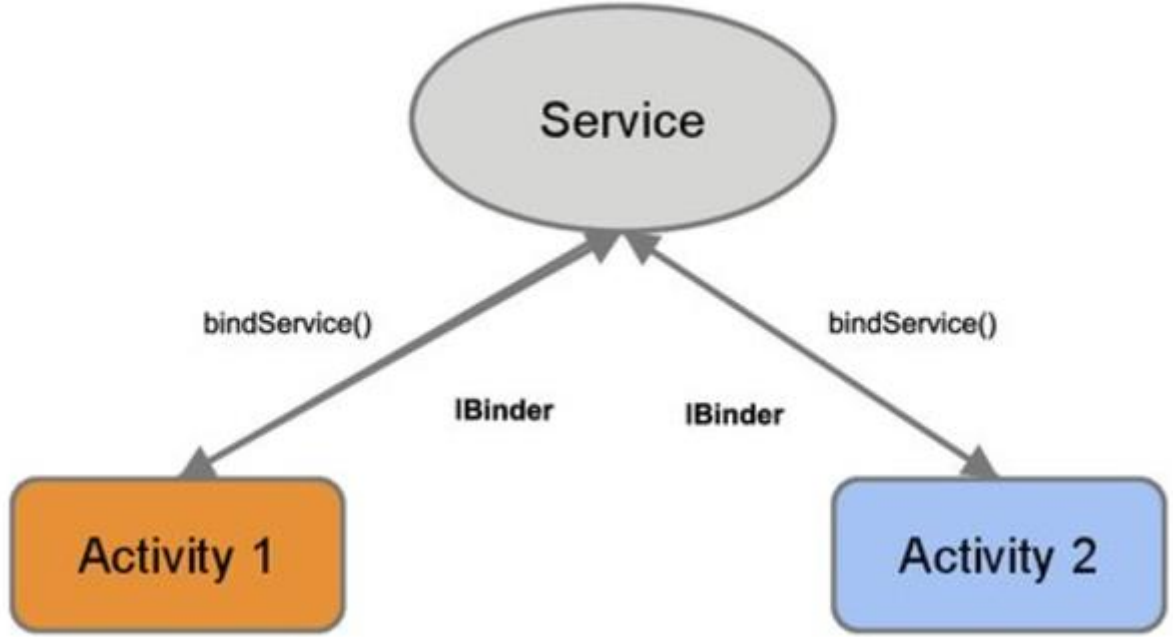


Види взаємодії зі службою



Компонент Android запускає виконання служби:
метод startService(intent).

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent i
putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```



Також можна запустити службу, викликавши метод b indService().

Дозволяє комунікувати напряму через службу.

Запуск служби

Служба запускається лише раз, незважаючи на кількість викликів методу startService()

- Якщо викликається метод startService(intent), а служба ще не виконується, створюється об’єкт слу жби (service object) і викликається метод onCreate() служби.

- Як тільки служба запущена, метод onStartCommand(intent) викликається всередині служби. Він п ередає Intent-об’єкт з виклику startService(intent).

- Якщо викликається startService(intent), поки служба працює, також викликається її метод о nStartCommand().
  - Служба має бути готова до кількаразового виклику onStartCommand().
  - Цей метод викликається в головному потоці Android, тому не може одночасно викликатись з різних потоків.

- Зупинка служби відбувається за допомогою методу stopService()
  - Служба може зупинити саму себе, викликавши метод stopSelf().
  - Наприклад, коли її робота завершена.

Перезапуск служби

При виклику методу onStartCommand() служба повертає ціле число, яке визначає по ведінку її перезапуску, якщо служба зупинена платформою Android

- Перевірити перезапуск дозволяє метод Intent.getFlags().

Option	Description
Service.START_STICKY	Service is restarted if it gets terminated. Intent data passed to the `onStartCommand` method is null. Used for services which manages their own state and do not depend on the `Intent` data.
Service.START_NOT_STICKY	Service is not restarted. Used for services which are periodically triggered anyway. The service is only restarted if the runtime has pending `startService()` calls since the service termination.
Service.START_REDELIVER_INTENT	Similar to Service.START_STICKY but the original `Intent` is re-delivered to the `onStartCommand` method.

Передаються

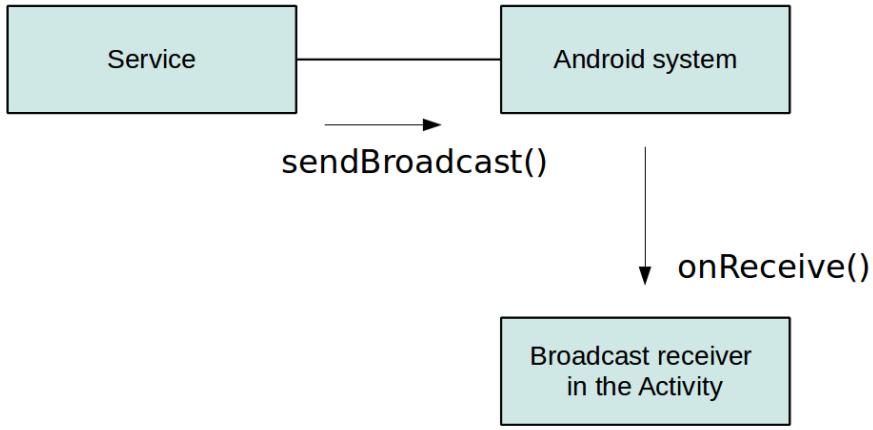
- START_FLAG_REDELIVERY (якщо службу запущено з Service.START_REDELIVER_INTENT)
- START_FLAG_RETRY (якщо службу запущено з Service.START_STICKY).

Комунікація зі службами

Два основних підходи

- ***За допомогою даних від інтента.***
  - Простий сценарій не передбачає прямої комунікації: служба отримує дані з інтента від компоненту, що її запустив, та виконує роботу.
  - Немає потреби у сповіщеннях зі служби.
  - Наприклад, служба оновлює постачальник контенту, який потім сповіщає активність про це.
  - Підхід працює для локальних служб та служб, що працюють у власних процесах.

- ***За допомогою приймача (receiver).***
  - Використовуються бродкасти та зареєстровані приймачі.
  - Наприклад, активність реєструє широкомовний приймач для події, а служба розсилає відповідні події.
  - Типовий сценарій, у якому служба сигналізує активності, що робота служби завершена.



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.service.receiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="18" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.vogella.android.service.receiver.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <service android:name="com.vogella.android.service.receiver.DownloadService" >
        </service>
    </application>

</manifest>
```

Приклад

Маніфест проекту

Завдання: завантаження файлу з мережі Інтернет після натиснення кнопки (з активності).

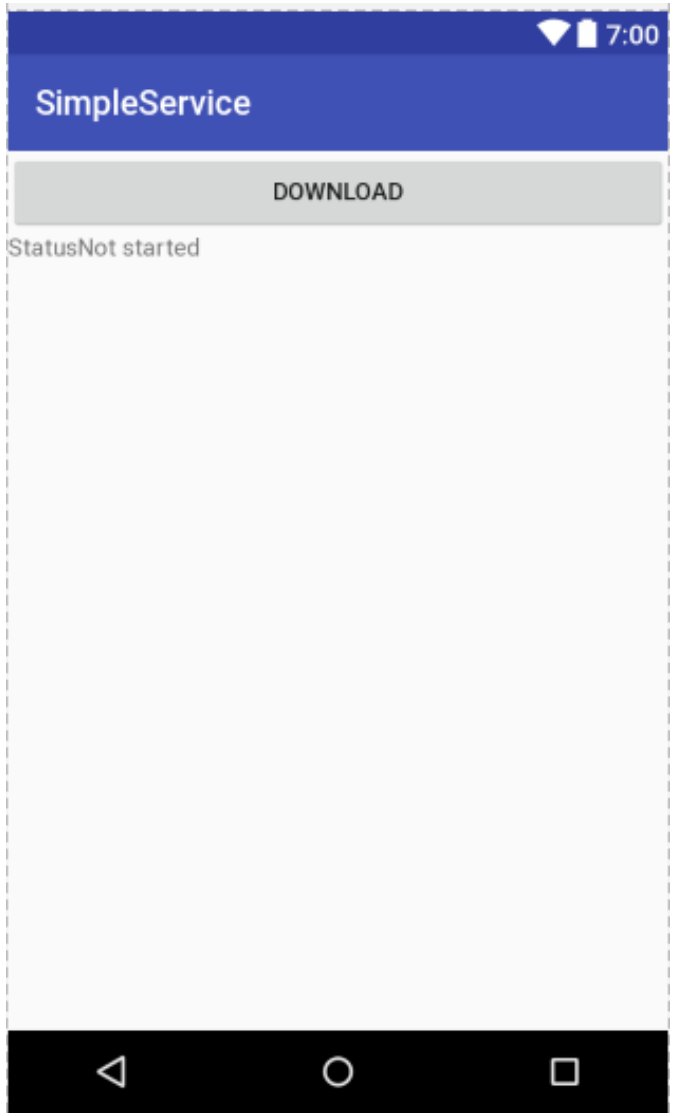
Після завантаження служба сповіщає активність через приймач, що завантаження здійснено.

Використовується клас IntentService, оскільки він забезпечує автоматичну фонову роботу (background processing).

Приклад

Макет інтерфейсу

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

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Download" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Status: " />

        <TextView
            android:id="@+id/status"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Not started" />
    </LinearLayout>

</LinearLayout>
```



Приклад

Служба завантаження

```
package com.vogella.android.service.receiver;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;

import android.app.Activity;
import android.app.IntentService;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.os.Message;
import android.os.Messenger;
import android.util.Log;

public class DownloadService extends IntentService {

    private int result = Activity.RESULT_CANCELED;
    public static final String URL = "urlpath";
    public static final String FILENAME = "filename";
    public static final String FILEPATH = "filepath";
    public static final String RESULT = "result";
    public static final String NOTIFICATION = "com.vogella.android.service.receiver";

    public DownloadService() {
        super("DownloadService");
    }
```

Приклад

Служба завантаження (продовження)

```
// will be called asynchronously by Android
@Override
protected void onHandleIntent(Intent intent) {
    String urlPath = intent.getStringExtra(URL);
    String fileName = intent.getStringExtra(FILENAME);
    File output = new File(Environment.getExternalStorageDirectory(),
            fileName);
    if (output.exists()) {
        output.delete();
    }

    InputStream stream = null;
    FileOutputStream fos = null;
    try {

        URL url = new URL(urlPath);
        stream = url.openConnection().getInputStream();
        InputStreamReader reader = new InputStreamReader(stream);
        fos = new FileOutputStream(output.getPath());
        int next = -1;
        while ((next = reader.read()) != -1) {
            fos.write(next);
        }
        // successfully finished
        result = Activity.RESULT_OK;

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

```
        } finally {
            if (stream != null) {
                try {
                    stream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        publishResults(output.getAbsolutePath(), result);
    }

    private void publishResults(String outputPath, int result) {
        Intent intent = new Intent(NOTIFICATION);
        intent.putExtra(FILEPATH, outputPath);
        intent.putExtra(RESULT, result);
        sendBroadcast(intent);
    }
}
```

Приклад

Головна активність

```
package com.vogella.android.service.receiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private TextView textView;
    private BroadcastReceiver receiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                String string = bundle.getString(DownloadService.FILEPATH);
                int resultCode = bundle.getInt(DownloadService.RESULT);
                if (resultCode == RESULT_OK) {
                    Toast.makeText(MainActivity.this,
                            "Download complete. Download URI: " + string,
                            Toast.LENGTH_LONG).show();
                    textView.setText("Download done");
                } else {
                    Toast.makeText(MainActivity.this, "Download failed",
                            Toast.LENGTH_LONG).show();
                    textView.setText("Download failed");
                }
            }
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = (TextView) findViewById(R.id.status);


    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(receiver, new IntentFilter(
                DownloadService.NOTIFICATION));
    }
    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(receiver);
    }

    public void onClick(View view) {

        Intent intent = new Intent(this, DownloadService.class);
        // add infos for the service which file to download and where to store
        intent.putExtra(DownloadService.FILENAME, "index.html");
        intent.putExtra(DownloadService.URL,
                "http://www.vogella.com/index.html");
        startService(intent);
        textView.setText("Service started");
    }
}
```

Постачальники контенту

Content Providers

Постачальники контенту управляють доступом до структурованого набору даних

Вони інкапсулюють дані та надають механізми забезпечення їх безпеки.

Представляють стандартний інтерфейс для об'єднання даних в одному процесі з кодом, який виконується в іншому процесі.

Модель розробки під Android сприяє створенню даних, доступних як для Вашого, так і для інших додатків.

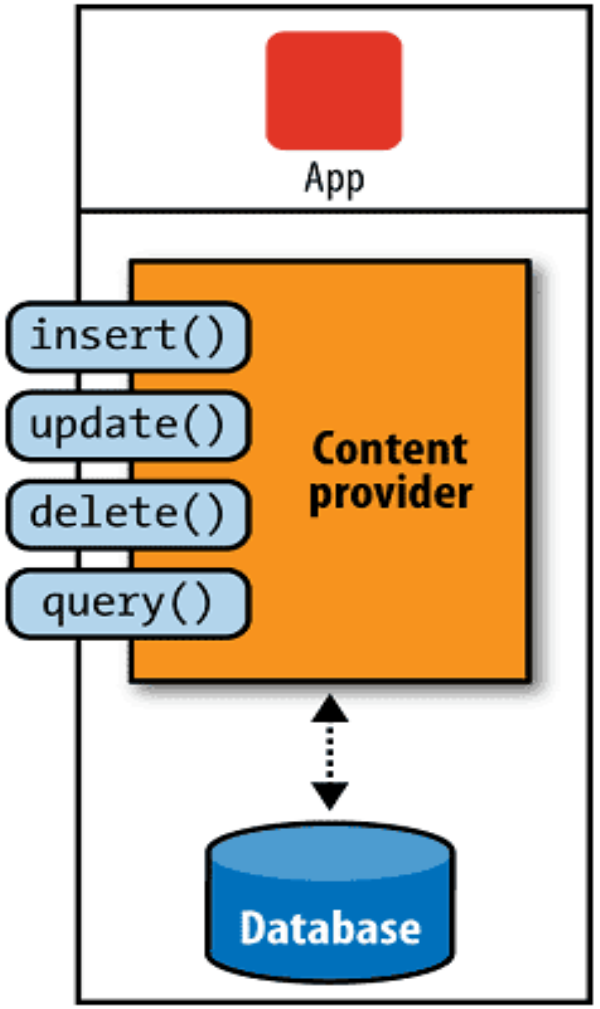
У певній мірі постачальники контенту аналогічні RESTful веб-сервісам

Знаходяться за уніфікованим ідентифікатором ресурсу (Uniform Resource Identifier, URI)

Операції підкласу ContentProvider мають паралелі з веб-операціями RESTful, наприклад, отримування та внесення даних.

Постачальники контенту

Content Providers



Для використання широкомовного приймача необхідно задати URI та дію з відповідними даними (CRUD):

Insert (створити), Query (зчитати), Update (оновити), Delete (видалити)

Постачальник контенту надає дані зовнішнім додаткам у вигляді однієї або декількох таблиць, аналогічних таблицям реляційних БД.

Приклад користувацького словника

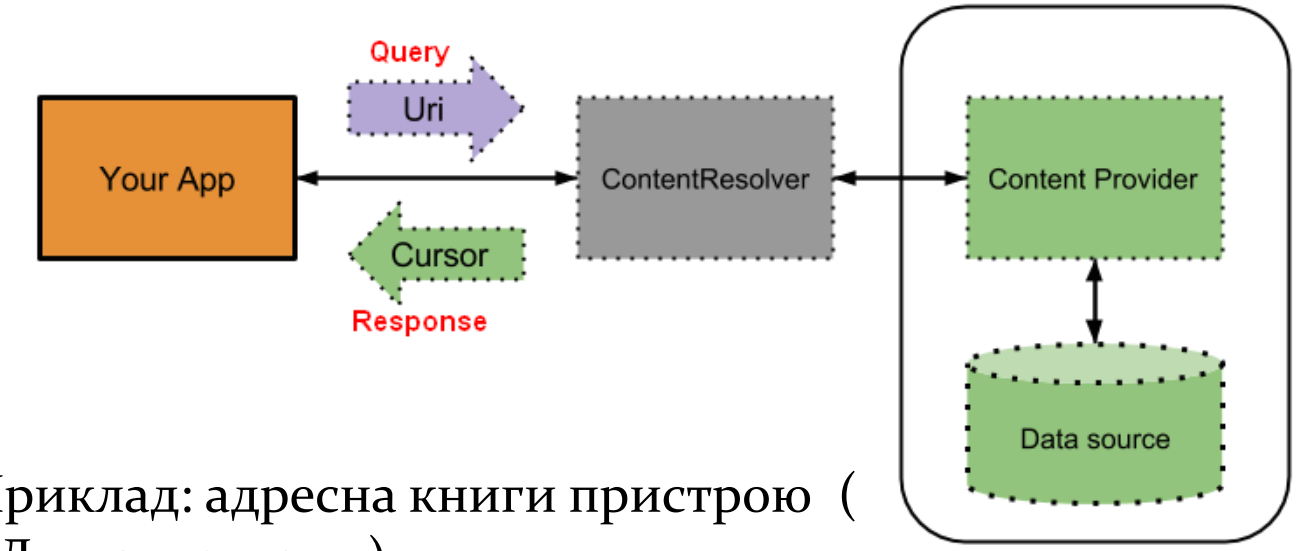
word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

У кожному рядку представлено екземпляр слова, яке відсутнє в стандартному словнику.

Концептуальна модель постачальника контенту

Для доступу додатка до даних із постачальника контенту використовується клієнтський об'єкт ContentResolver.

У ньому пристуні методи, які викликають ідентичні методи в об'єкті постачальника – CRUD-функції.




```
Cursor c = getContentResolver().query(
    android.provider.ContactsContract.Data.CONTENT_URI,
    new String[] {
        android.provider.ContactsContract.Data._ID,
        android.provider.ContactsContract.Phone.NUMBER},
    null,
    null,
    null);;
```

Приклад: адресна книги пристрою (БД з контактами).

База даних Contacts працює в окремому додатку в рамках і ншого процесу.

Використання провайдеру контенту включає виклик його операцій з даними разом з URI