

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
“Програмування малих та мобільних платформ”
для магістрів спеціальності

121 Інженерія програмного забезпечення
денної форми навчання

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Програмування малих та мобільних платформ” для магістрів спеціальності 121 "Інженерія програмного забезпечення" / Укл.: О.О. Степаненко, Ю.В. Твердохліб.— Запоріжжя: ЗНТУ, 2016. – 59 с.

Автори: Степаненко О.О., к.т.н., доцент;
Твердохліб Ю.В., асистент

Рецензент: Онищенко В.Ф., к. ф.-м. н., доцент

Відповідальний за випуск: Субботін С.О., д.т.н., проф.

Затверджено
вченою радою інституту
інформатики та радіоелектроніки

Протокол № 1
від “19” березня 2014 р.

Затверджено
на засіданні кафедри
“Програмні засоби”

Протокол № 7
від “06” березня 2014 р.

ЗМІСТ

Вступ	4
Лабораторна робота № 1 Ознайомлення з середою розробки та архітектурою мобільного додатку	5
Лабораторна робота № 2 Графічний інтерфейс користувача. Основні віджети та обробка подій	21
Лабораторна робота № 3 Графічний інтерфейс користувача. Списки	38
Лабораторна робота № 4 Організація обміну даних у гібридних мобільних додатках	51
Література	59

ВСТУП

Людські потреби бути завжди в курсі справ виводять інформаційні технології на пряму лінію по створенню всіх нових пристроїв і гаджетів. Незручність експлуатації комп'ютерів і ноутбуків спричиняє появу різних міні-комп'ютерів, смартфонів та комунікаторів, в основі яких лежить усе та ж операційна система. Лідируючі позиції на сьогоднішній день займають платформи Android і iPhone. Але ці платформи можуть працювати повноцінно тільки при одній умові - якщо для них буде здійснюватись розробка мобільних додатків.

Метою курсу “ Програмування малих та мобільних платформ ” є набуття навиків створення сучасних мобільних додатків високої якості, знайомство з сучасними середами та мовами розробок програмного забезпечення для мобільних пристроїв, вміння правильно та ефективно тестувати розроблені мобільні додатки.

Звіт з лабораторної роботи повинен містити:

- тему;
- мету;
- текстовий та графічний матеріал розробки;
- висновки.

Під час співбесіди при захисті лабораторної роботи студент повинен виявити знання по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинен вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

ЛАБОРАТОРНА РОБОТА №1.

ОЗНАЙОМЛЕННЯ З АРХІТЕКТУРОЮ МОБІЛЬНОГО ДОДАТКУ

Мета: створення середовища розробки мобільного додатку на базі Android, створення екземпляру емулятору мобільного пристрою; вивчення структури, принципів роботи та інсталяція додатку на емулятор та мобільний пристрій.

Короткі теоретичні відомості

1.1 Створення середовища розробки

Для розробки додатків на платформі Android необхідно завантажити та встановити наступне програмне забезпечення, доступне для вільного завантаження:

1. **Java Development Kit (JDK)** – комплект розробника додатків мовою Java, що включає в себе компілятор Java (javac), стандартні бібліотеки класів Java, приклади, документацію, різні утиліти і середовище виконання *Java Runtime Environment, JRE* (доступне для завантаження за адресою: <http://java.sun.com/javase/downloads/index.jsp>).

2. **Eclipse IDE** – інтегроване середовище розробки для Java, що має можливість установки плагіну для розробки Android-додатків (доступно для завантаження за адресою: <http://www.eclipse.org/downloads/>).

3. **Android SDK (Software Development Kit)** – набір бібліотек та інструментів для розробки Android-додатків, включаючи емулятор мобільного пристрою (доступне для завантаження за адресою: <http://developer.android.com/sdk/index.html>). Android SDK включає в себе:

- *API Android SDK* – API-бібліотеки Android, що надаються для розробки додатків;
- документація SDK;
- *Development Tools* – інструментальні засоби для розробки, що надають можливість компілювати та налагоджувати додатки, що створюються;

– *Sample Code* – типові додатки, що демонструють деякі можливості Android.

4. **Android Development Tools (ADT)** – плагін, що автоматизує процес побудови додатків для Android, інтегруючи інструменти розробки безпосередньо в середовище розробки Eclipse.

Плагін ADT додає в Eclipse наступні компоненти:

- майстер створення проекту – *New Project Wizard*, котрий спрощує створення Android-проектів та формує шаблон проекту;

- редактор *Layout Editor* – для розробки графічного інтерфейсу додатку;

- різні редактори ресурсів для створення, редагування та перевірки вірності XML-ресурсів розробника.

Установка всього програмного забезпечення полягає у завантаженні, розпакуванні архіву та запуску виконавчого файлу з розширенням *. exe. Після встановлення JDK, Eclipse IDE і Android SDK необхідно встановити плагін ADT. Для встановлення плагіну запускаємо Eclipse та обираємо пункт меню **Help | Install New Software**.

У діалоговому вікні, що з'явиться, натиснути кнопку Add. У полі **location** ввести «<https://dl-ssl.google.com/android/eclipse/>». Після виконання всіх інструкцій з установки перезапустіть середу Eclipse.

Тепер потрібно зв'язати Eclipse з каталогом Android SDK. Виберіть у головному меню Eclipse пункт **Window | Preferences**, після пункт **Android** та в полі **SDK location** вказати каталог, в якому розташований Android SDK.

1.2 Інструменти для розробки на налагодження додатків

Окрім емулятору, SDK також включає безліч інших інструментальних засобів для налагодження та установки створюваних додатків:

- *android* – інструмент розробки, що запускається з командного рядка, що дозволяє створювати, видаляти і конфігурувати віртуальні пристрої, створювати і оновлювати Android SDK новими платформами, доповненнями і документацією;

– *Dalvik Debug Monitor Service (DDMS)* – інтегрований з **Dalvik Virtual Machine**, стандартної віртуальної машиною платформи Android, цей інструмент дозволяє керувати процесами на емуляторі або пристрої, а також допомагає у налагодженні додатків. За допомогою цього сервісу можна завершувати процеси, обирати певний процес для налагодження, генерувати трасовані дані, здійснити моніторинг використання пам'яті мобільного пристрою, переглядати інформацію про потоки, робити скріншоти емулятора або пристрої та багато іншого;

– *Hierarchy Viewer* – візуальний інструмент, який налагоджувати і оптимізувати користувацький інтерфейс додатку, що розробляється. Він показує візуальне дерево ієрархії класів View, аналізує швидкодію перемальовування графічних зображень на екрані і може виконувати ще багато інших функцій для аналізу графічного інтерфейсу додатків;

– *Layoutopt* – інструмент командного рядка, який допомагає оптимізувати схеми розмітки та ієрархії розміток в створюваному додатку. Необхідний для вирішення проблем при створенні складних графічних інтерфейсів, які можуть зачіпати продуктивність програми;

– *Draw 9-patch* – графічний редактор, який дозволяє легко створювати NinePatch-графіку для графічного інтерфейсу розроблюваних додатків;

– *SQLite3* – інструмент для доступу до файлів даних SQLite, створених і використовуваних Android-додатками;

– *Traceview* – цей інструмент видає графічний аналіз трасованих логів, які можна генерувати з додатків;

– *Mksdcard* – інструмент для створення образу диску, який можна використовувати в емуляторі для симуляції наявності зовнішньої карти пам'яті (наприклад, карти SD).

Для використання інструментів, що надаються Android SDK, необхідно налаштувати змінні оточення: додати в змінну оточення Path шляхи до підкаталогів Android SDK (наприклад, C:\eclipse\adt\sdk\platform-tools та C:\eclipse\adt\sdk\tools).

1.3 Android- емулятор мобільного пристрою

Android Virtual Device (віртуальний пристрій Android) – це емулятор, що запускається на звичайному комп'ютері.

Емулятор використовується для проектування, налагодження і тестування додатків в реальному середовищі виконання.

Перш ніж запустити Android-емулятор пристрою, необхідно створити екземпляр AVD. AVD визначає системне зображення та параметри пристрою, що використовуються емулятором.

Створити екземпляр AVD можна двома способами:

1. В командному рядку утилітою android .
2. За допомогою візуального інструменту Android SDK and AVD Manager. Його можна запустити з кореневого каталогу Android SDK або в середовищі Eclipse , обравши пункт меню **Window | Android Virtual Device Manager**. У вкладці *Android Virtual Devices* необхідно натиснути кнопку **New** і у вікні «*Create new Android Virtual Device*» задайте конфігурацію для створюваного емулятору пристрою (рисунк 1.1):

- *Name* – ім'я створюваного пристрою;
- *Device* – тип пристрою;
- *Target* – версія Android SDK, що підтримується емулятором Android-пристрою;
- *Hardware keyboard present* – якщо прапорець встановлено, то для набору тексту в емуляторі можливо буде використовувати не тільки віртуальну клавіатуру, але й клавіатуру комп'ютера;
- *Display a skin with hardware controls* – показує кнопки керування віртуальним пристроєм в вікні емулятору (home, menu, back і т.ін.). Рекомендовано відмітити.
- *Memory Options* – ємність оперативної пам'яті та пам'яті, що використовується додатком;
- *Internal Storage* – вбудована пам'ять мобільного пристрою;
- *SD Card* – встановлює віртуальну карту пам'яті, для якої можна в текстовому полі *Size* задати необхідну ємність;
- *Emulator Option* – «*Use Host GPU*» (Graphics processing unit) для прискорення роботи емулятору, «*Snapshot*» – при встановленні прапорця, з'являється можливість зберігати стан системи в файл при закритті емулятора та відновлювати його при запуску, оминаючи тривале завантаження.

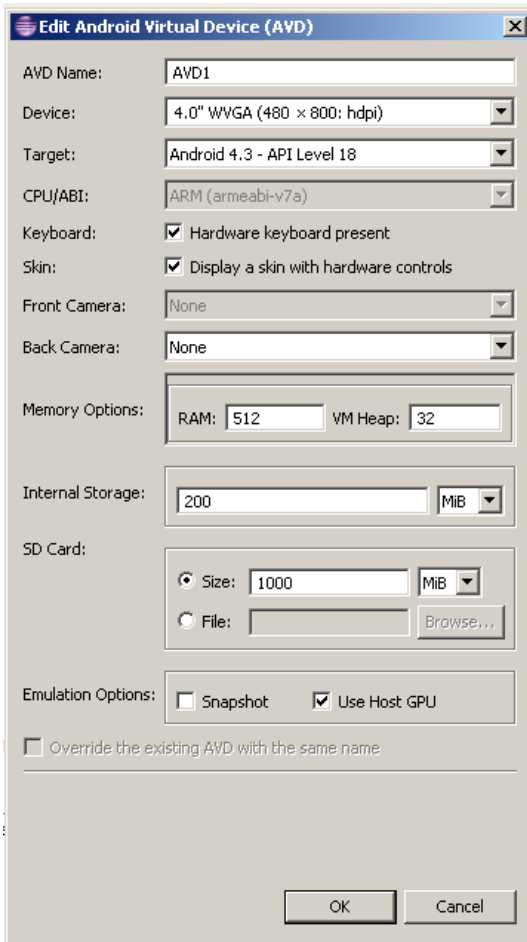


Рисунок 1.1 – Створення нового AVD

Для зміни параметрів пристроїв у вікні *Android Virtual Device Manager* у вкладці *Device Definitions* необхідно натиснути кнопку *New Device*.

- *Name* – ім'я пристрою;
- *Screen Size* і *Resolution* – відповідно розмір діагоналі (у дюймах) і дозвіл (в пікселях) екрана. На їх основі розраховується щільність пікселів;

- *Sensors* – датчики пристрої: акселерометр (*Accelerometer*), гіроскоп (*Gyroscope*), GPS і датчик наближення (*Proximity Sensor*);

- *Cameras* – камери пристрою: *Front* – передня, *Rear* – задня;

- *Input* – пристрої введення: *Keyboard* – клавіатура, *DPad* – джойстик, *Trackball* – трекбол;

- *RAM* – розмір оперативної пам'яті за замовченням;

- *Size* – узагальнений розмір (клас). Розміри дисплеїв в Android залежно від діагоналі поділяються на 4 класи: *small* (< 3.55 in), *normal* (3.55 – 5 in), *large* (5 – 7.5in), *xlarge* (> 7.5 in);

- *Screen Ratio* – співвідношення сторін екрану: «*long*» для широкоформатних дисплеїв (наприклад, WQVGA, WVGA, FWVGA), «*notlong*» – для звичайних (наприклад, QVGA, HVGA, VGA);

- *Density* – клас щільності пікселів. На його основі розраховується величина *dp*. Вона використовується для однакового відображення графічних об'єктів на дисплеях з різною щільністю пікселів. Наприклад, картинка розміром 100×100px буде виглядати маленької на екрані з високою щільністю і великою на екрані з малою, а от 100×100dp виглядає приблизно однаково скрізь.

Існує всього 7 класів щільності: *nodpi* – величина *dp* прирівнюється до 1px, графіка залежить від щільності, *ldpi* – низька щільність пікселів ~ 120dpi, *mdpi* – середня щільність пікселів ~ 160dpi, *hdpi* – висока щільність пікселів ~ 240dpi, *xhdpi* – дуже висока щільність ~ 320dpi, *xxhdpi* – надвисока щільність >400dpi, *tvdpi* – середнє між *mdpi* і *hdpi* (~ 213dpi);

- *Buttons* – визначає якими будуть кнопки управління (home, menu та ін): *hardware* – фізичні (відобразяться у вікні емулятора) або *software* – програмні (як в планшетах).

- *Device State* – орієнтація екрану (*Portrait* – портретна або *Landscape* – альбомна) і навігації по екрану (*DPad* і *Trackball*).

1.4 Програмний стек Android

Детальний опис програмного стеку Android SDK представлено на рисунку 1.2.

Центром платформи Android є ядро Linux, що відповідає за драйвера пристроїв, доступ до ресурсів, управління енергоспоживанням і вирішенням інших завдань ОС.

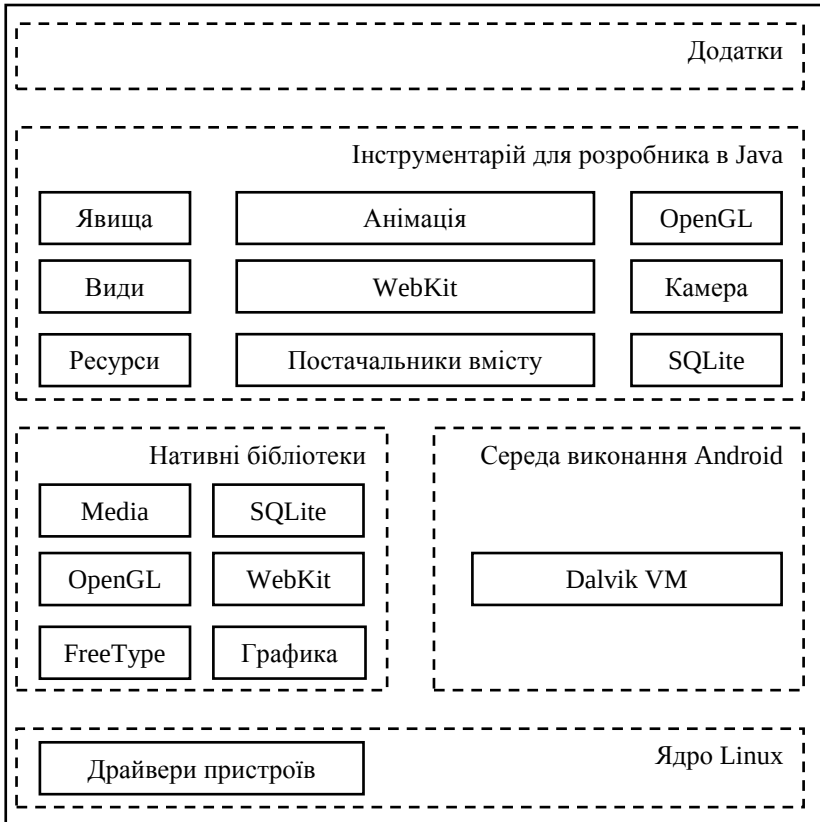


Рисунок 1.2 – Опис програмного стеку Android SDK

На наступному рівні, вище ядра, знаходиться ряд бібліотек C/C++, зокрема OpenGL, WebKit, FreeType, Secure Sockets Layer (SSL), бібліотека часу виконання C (libc), SQLite і Media. Системна бібліотека C, заснована на Berkeley Software Distribution (BSD), налаштована для роботи з вбудованими пристроями, що працюють під Linux. Медіабібліотеки працюють на основі PocketVideo OpenCORE. Ці бібліотеки відповідають за запис та відтворення аудіо- та відеоформатів .

Бібліотека Surface Manager контролює доступ до системи відображення даних і підтримує 2D і 3D. Бібліотека WebKit відповідає за підтримку браузерів. Бібліотека FreeType підтримує шрифти. SQLite – це реляційна база даних, яка знаходиться на самому пристрої. Крім того, SQLite – це незалежна розробка з відкритим кодом, яка не пов'язана безпосередньо з Android. Можна використовувати інструменти, призначені для SQLite, і при роботі з базами даних Android.

Більша частина додатків з цього набору звертається до зазначених кореневих бібліотек через **Dalvik Virtual Machine (Dalvik VM)**, що виконує на платформі Android роль шлюзу. Dalvik оптимізована для одночасного використання декількох екземплярів VM. Коли додатки Java звертаються до цих кореневих бібліотек, кожний додаток працює з власним примірником віртуальної машини.

В основних бібліотеках прикладного інтерфейсу програмування на Java містяться функції для телефонії, роботи з ресурсами, місцезнаходженням, користувальницькими інтерфейсами, постачальниками вмісту (даними), а також диспетчери пакетів (що відповідають за встановлення, безпеку і т.д.). Розробка додатків ведеться на підставі прикладного інтерфейсу програмування Java. В Android підтримується бібліотека Google Skia, що призначена для роботи з 2D – графікою, що написана на C/C++ і бібліотека роботи з 3D – графікою на основі скороченої версії OpenGL ES, оптимізованої для роботи з вбудованими системами. Для роботи з медіа, на платформі Android підтримується більшість поширених форматів аудіо, відео та зображень. В області бездротового зв'язку Android має в своєму розпорядженні спеціальні API для підтримки Bluetooth, EDGE, 3G, Wi-Fi і глобальної системи мобільного зв'язку (GSM), залежними від обладнання.

1.5 Пакети Java для Android

Пакети, що входять до складу Android SDK: android.app (модель додатків), android.bluetooth, android.content (постачальник вмісту для узагальнення обміну та зберігання даних), android.database (реферативна база даних), android.gesture (для роботи з жестами користувача), android.graphics, android.hardware (робота з камерою),

android.net (мережеві API на рівні сокетів), android.location (інформація про місцезнаходження об'єкта), android.media, android.opengl, android.os (служби операційної системи: обмін інформацією між процесами, облік змін в файлах, використання потоку повідомлень тощо), android.preference (можливість користувача керувати налаштуваннями свого додатку), android.provider (набір постачальників вмісту, що відносяться до інтерфейсу android.content), android.sax (набір API для XML), android.speech (робота з розпізнаванням мови), android.telephony (інформація про телефонний виклик), android.text (введення, обробка, оформлення тексту), android.units, android.view, android.webkit (класи, що відносяться до веб – браузеру), android.widget (класи елементів управління користувацького інтерфейсу), com.google.android.maps (класу для роботи з картами Google). Опис даних пакетів детально описаний в документації SDK та розглянуто в рекомендованому переліку літератури по виконанню лабораторних робіт.

1.6 Структура Android-проекту

1.6.1 Створення Android-проекту

Запускаємо Eclipse та обираємо **File|New|Project...**, в списку обираємо **Android|Android Application Project** і натискаємо кнопку **Next**. У відкритому діалоговому вікні майстра *New Android Project* заповніть наступні поля:

- **Application name** (ім'я додатку, котре буде відображатися в його заголовку) – *SimpleProject*;
- **Project name** (ім'я проекту і ім'я каталогу, що буде містити проектні файли) – *SimpleProject*;
- **Package name** (ім'я пакету) – *com.example.simpleproject*;
- **Minimum Required SDK** обирається мінімальна підтримувана платформа, мінімальний рівень API;
- **Target SDK** – вказівка компілятору зібрати додаток для обраного рівня API (виставляється автоматично залежно від обраної версії SDK);
- **Compile With** – тут за замовчуванням ставиться сама остання версія Android;
- **Theme** – тема додатку.

Заповнивши усі поля, натискаємо на кнопку **Next** для переходу в наступне вікно.

- залишаємо прапорець у полі **Create custom launcher icon**, щоб мати можливість встановити власний значок для програми;

- залишаємо прапорець у полі **Create activity**;

- не ставимо прапорець у полі **Mark this project as library**, вона призначена для створення бібліотек;

- залишаємо прапорець у полі **Create Project in Workspace** – всі ваші проекти будуть зберігатися в спеціальній папці. Або ви можете задати свій шлях для проекту .

У наступному вікні майстра можна вибрати і налаштувати свій значок для програми. Після цього натискаємо двічі **Next** та **Finish**.

Щоб побачити додаток в дії, виконуємо команду **Run** (рисунок 1.3). При цьому середовище розробки автоматично інсталує його на Android-емулятор.

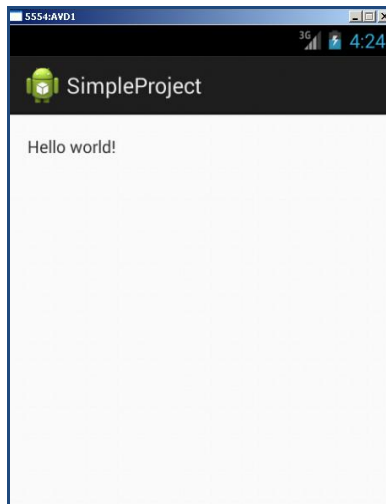


Рисунок 1.3 – Вікно програми SimpleProject

1.6.2 Зміст Android-проекту

ADT – плагін при створенні Android-проекту організує структуру у вигляді дерева каталогів, як і будь-який інший Java-проект (рисунок 1.4):

- **scr** – каталог, що містить вихідний код програми;
- **assets** – довільний збір каталогів і файлів;
- **res** – каталог, що містить ресурси додатку;
- **drawable** – каталог, який містить зображення або файли дескрипторів зображень;
- **anim** – каталог, що містить XML – дескриптори, в який описується анімація, що використовується програмою;
- **layout** – каталог, в якому містяться види даної програми (вони створюються за допомогою XML – дескрипторів, а не шляхом написання коду);
- **menu** – каталог, що містить файли XML – дескрипторів, в який описується меню, що використовується в програмі;
- **values** – каталог, що містить інші ресурси, використовувані програмою (рядки, стилі, кольори), що визначається XML – дескрипторами;
- **xml** – каталог, що містить додаткові XML – файли, використовувані додатком;
- **raw** – каталог, що містить додаткові дані, які використовуються додатком;
- **AndroidManifest.xml** – файл опису програми Android. У цьому файлі визначені явища, постачальники вмісту, служби та приймачі намірів (intent receiver) даного додатку. Цей файл також можна використовувати для декларативного пояснення прав доступу, необхідних для роботи з додатком, і для надання специфічних прав доступу іншими додатками, які користуватимуться службами даного. Крім того, у файлі можуть міститися інформація про інструментування (instrumentation detail), яка може використовуватися при тестуванні даного чи іншого додатка.

Розглянемо файл **MainActivity.java**.

```
package com.example.simpleproject;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
public class MainActivity extends Activity {
```

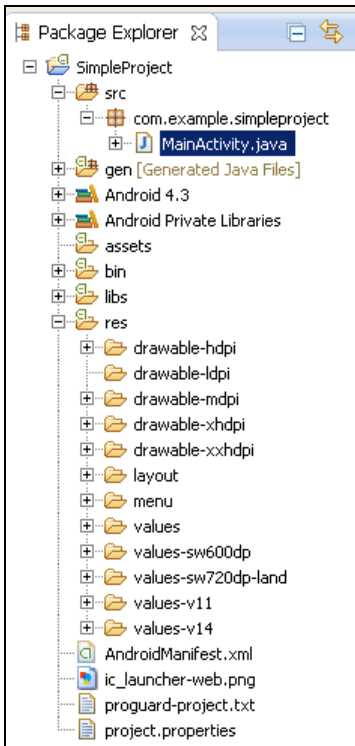


Рисунок 1.4 – Структура програми SimpleProject

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

public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

}
```

У першому рядку йде назва пакету (з Package Name). Далі – рядки імпорту необхідних класів для проекту. Потім йде оголошення самого класу, який успадковується (extends) від абстрактного класу Activity. У самому класі є метод **onCreate()** – він викликається, коли додаток створює і відображає розмітку активності.

Рядок **super.onCreate** (**savedInstanceState**) – це конструктор батьківського класу, що виконує операції для роботи активності.

Другий рядок **setContentView (R.layout.activity_main)** – підключає вміст з файлу розмітки. Як аргумент ми вказуємо ім'я файлу без розширення з папки **res/layout/**(за замовченням **activity_main.xml**).

Ви можете перейменувати файл або створити свій файл з ім'ям **lab.xml**, тоді код буде виглядати так: **setContentView (R.layout.lab)**;

Якщо ваш додаток буде складатися з безлічі екранів, то ви будете створювати для кожного екрана свою розмітку і підключати її описаним вище чином.

Перегляд і редагування властивостей видів додатку можливо з файлу **activity_main.xml**: оберіть потрібний об'єкт в графічній версії файлу і переглядайте властивості даного об'єкта (праворуч).

Додавання (зміна) рядків, зображень та інших ресурсів можливо з **/res/values**: для цього наприклад оберіть файл **strings.xml** і натисніть на кнопку **Add**. Майстер створення запропонує обрати тип елемента, а після його ім'я і значення.

1.6.3 Структура файлу AndroidManifest.xml

Файл маніфесту інкапсулює всю архітектуру Android-додатку, його функціональні можливості і конфігурацію. Розглянемо призначення кожного з них.

– **<manifest>** – це кореневий елемент файлу, що містить наступні атрибути: **xmlns:android** (простір імен Android) з постійним значенням [«http://schemas.android.com/apk/res/android»](http://schemas.android.com/apk/res/android), **package** (ім'я пакету додатку), **android:versionCode** (внутрішній номер версії) і **android:versionName** (номер користувацької версії);

– **<uses-permission>** – описує дозволи безпеки, які потрібно надати вашому пакету. Кількість не обмежена. Наприклад, **<uses-permission android:name = "android.permission.RECEIVE_SMS"/>**;

– **<permission>** – оголошує дозвіл безпеки, що може використовуватися до певних компонентів або функціональності даного додатка, а також права доступу інших додатків до додатку, що розробляється, використовуючи

атрибути: *android:name* (елемент дозволу), *android:label* (ім'я дозволу, що відображається користувачеві), *android:description* (опис), *android:icon* (іконка розширення), *android:permissionGroup* (визначає приналежність до групи дозволів) і *android:protectionLevel* (рівень захисту);

- **<permission-tree>** – оголошує базове ім'я для дерева дозволів (простір імен);

- **<permission>** – визначає ім'я для набору логічно зв'язаних рішень (категорія дозволів);

- **<instrumentation>** – визначає можливість контролювати взаємодію додатку з системою;

- **<uses-sdk>** – визначає сумісність програми з зазначеної версією платформи Android (атрибут *minSdkVersion*);

- **<uses-configuration>** – вказує необхідну для додатку апаратну і програмну конфігурацію мобільного пристрою;

- **<uses-feature>** – визначає функціональність, яка потрібна для роботи додатку;

- **<supports-screens>** – визначає здатність екрану, необхідну для функціонування пристрою;

- **<application>** – кореневий елемент, що містить оголошення компонентів додатку, доступних в пакеті. Цей елемент може також включати глобальні та/або задані за замовчуванням атрибути для програми, такі як мітки, значок, тема, вимоги дозволу, і т.ін. Елемент містить наступні дочірні елементи : **<activity>** з атрибутами *android:name*, *android:label* та інші (кожен **<activity>** може містити елемент **<Intent-filter>**, що надає для компонентів-клієнтів можливість отримання Intent оголошеного типу, відфільтровуючі ті, що не значимі для компоненту, і містить в першу чергу дочірні елементи **<action>**, **<category>**, **<data>**), **<activity-alias>** – псевдонім Activity, **<receiver>** з внутрішнім елементом *intent – filter* (дозволяє додатку повідомляти про заміну даних або про діях , які відбуваються , навіть якщо програма не виконується в даний час) , **<service>** з внутрішнім елементом *intent - filter* (працює у фоновому режимі), **<provider>**-компонент, який управляє постійними даними і відкриває до них доступ іншим додаткам, **<uses-library>** – визначає загальнодоступну бібліотеку для компіляції додатку.

Завдання на роботу

Створити Android-проект за допомоги конструкторів та мастерів, змінивши властивості виду: встановити фон додатку на обраний, змінити надпис, додати зображення. Приклад наведено на рисунку 1.5.

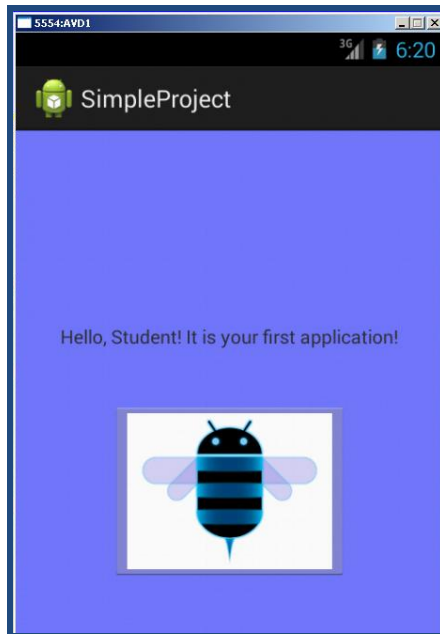


Рисунок 1.5 – Вікно програми після зміни її властивостей

Зміст звіту

- титульний лист, тема та мета роботи;
- текст розробленої програми;
- результати розробленої програми;
- висновки;
- відповіді на контрольні запитання.

Контрольні питання

1. Програмний стек Android.
2. Dalvik Virtual Machine.

3. Відповідність
та рівень API. версії платформи
4. Пакети Java для Android.
5. Створення AVD екземпляру (емулятору).
6. Каталоги ресурсів проекту.
7. Каталог констант додатку /res/values/.
8. Файл AndroidManifest.xml.

ЛАБОРАТОРНА РОБОТА №2

ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА. ОСНОВНІ ВІДЖЕТИ ТА ОБРОБКА ПОДІЙ

Мета роботи: ознайомитися з основними віджетами, їх компоновкою та обробкою подій, викликаних користувачем.

Короткі теоретичні відомості

2.1 Компоновка елементів управління

Компоновка – це архітектура розташування елементів інтерфейсу користувача для окремого вікна (**Activity**). Компоновка визначає структуру розташування елементів (тобто віджетів, **View**) у вікні та містить всі елементи, які надаються користувачу програми. При створенні компоновки треба враховувати те, що екрани мобільних пристроїв мають меншу роздільну здатність, наприклад, ніж монітори, різноманітність мобільних пристроїв з різними розмірами та щільністю пікселів, різні типи сенсорних екранів.

В android-додатку графічний інтерфейс користувача формується за допомогою об'єктів **View** та **ViewGroup**. Клас **View** є базовим класом для **ViewGroup** та складається з набору об'єктів **View**. Об'єкти **View** – це основні модулі для створення графічного інтерфейсу користувача. Клас **View** є базовим для елементів управління (віджетів) – текстових полів, кнопок та ін. Клас **ViewGroup** представляє контейнер, який служить ядром для підкласів, що називаються компоновки (**layouts**). Ці класи формують розташування віджетів на формі та містять дочірні елементи **View** та **ViewGroup** (рисунок 2.1). Таким чином для кожного **Activity** формується дерево ієрархії вузлів **View** та **ViewGroup**.

При запуску програми система отримує посилання на кореневий вузол дерева та використовує її для відображення графічного інтерфейсу користувача на мобільному пристрої. Система також аналізує елементи дерева ієрархії, додаючи їх до елементів-батьків. Для цього в методі **onCreate()** треба викликати метод **setContentView()** передаючи в якості

параметру посилання на ресурс компоновки.
Наприклад, якщо компоновка знаходиться в файлі mail.xml:

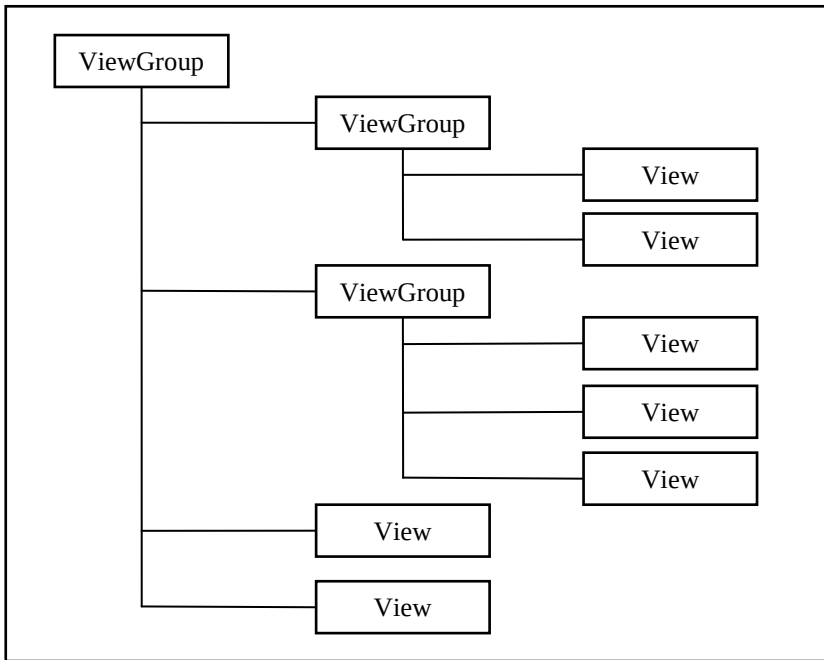


Рисунок 2.1 – Приклад дерева вузлів View та ViewGroup для Activity

```
// ініціалізація компоновки
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

2.2 Типи компоновок

Для створення вікон існує декілька стандартних типів компоновки, які є підкласами ViewGroup:

- FrameLayout;
- LinearLayout;
- TableLayout;
- RelativeLayout.

Розглянемо їх використання детальніше.

2.2.1 Компоновка FrameLayout

FrameLayout є найпростішим типом компоновки. В основному це пустий простір на екрані, яке можна заповнити тільки єдиним дочірнім елементом View чи ViewGroup. Всі дочірні елементи додаються до верхнього лівого кута екрану (рисунок 2.2).

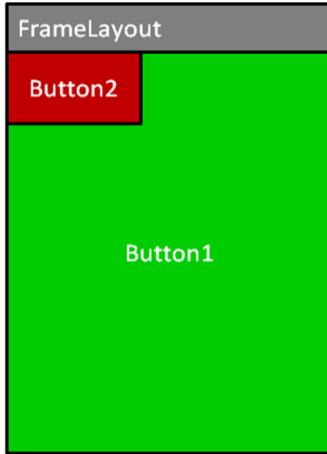


Рисунок 2.2 – Компоновка FrameLayout

В цій компоновці неможна визначити різноманітне місцезнаходження для дочірнього елементу. Всі дочірні елементи додаються поверх попередніх, частково чи повністю затіняючи їх. Виходячи з цього, батьківський елемент займає весь простір екрану та має наступні атрибути: `layout_width="match_parent"` та `layout_height="match_parent"`. Дочірній елемент має дані атрибути з будь-яким параметром: `"match_parent"` (відповідає батьку та приймає значення відповідного параметра як у батька) чи `"wrap_content"` (стандартний розмір елемента). Елементи інтерфейсу будуть розглянуті далі, проте **основними атрибутами для кожного виду є наступні: `android:id`, `android:layout_width`, `android:layout_height`, `android:text`.**

Компоновка FrameLayout використовується рідко, оскільки не дозволяє створити складні вікна з множиною елементів. Вона найчастіше використовується для оверлеїв. Наприклад, якщо у вікні є зображення, що займає весь екран (чи карта, кадр з відеокамери та ін.), а на ньому є елементи

керування (наприклад, елемент масштабування, елемент пошуку, індикатор часу чи стану та ін.).

Наприклад, створимо компоновку `FrameLayout`, розмістивши на ній дві кнопки як на рисунку 2.2.

```
<FrameLayout xmlns:android=http://schemas.android.com/
apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/b1"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:text="Button1"
    />
    <Button
        android:id="@+id/b2"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Button2"
    />
</FrameLayout>
```

2.2.2 Компоновка *LinearLayout*

Компоновка `LinearLayout` вирівнює всі дочірні елементи в одному з напрямів – вертикально чи горизонтально, в залежності від значення атрибуту `android:orientation: "horizontal", "vertical"`.

Наприклад, створимо компоновку `LinearLayout`, розмістивши на ній кнопки вертикально як на рисунку 2.3.

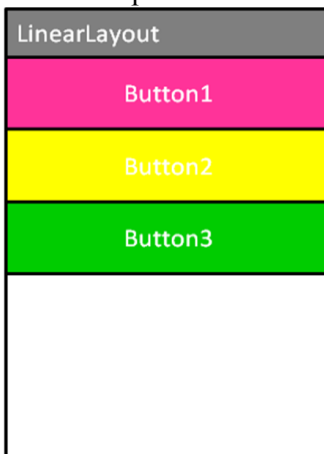


Рисунок 2.3 – Компоновка `LinearLayout`

```

<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/b1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Button1"
    />
    <Button
        android:id="@+id/b2"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Button2"
    />
</LinearLayout>

```

Компоновка `LinearLayout` має атрибут `android:layout_weight`, який визначає вагу дочірнього елемента, тобто надає можливість розширюватися (за замовченням, вага має нульове значення).

Оскільки компоновки можуть бути вкладеними, то у вікні можна розташувати наприклад декілька лінійних компоновок з будь-якою орієнтацією їх елементів. Таке застосування є найбільш розповсюдженим засобом при створенні інтерфейсу користувача для Android-додатків.

2.2.3 Компоновка *TableLayout*

Компоновка `TableLayout` позиціонує всі дочірні елементи у строки та стовбці. `TableLayout` не відображує лінії розділу чарунок. `TableLayout` може мати строки з різним числом чарунок. При формуванні компоновки деякі чарунки можна залишати порожніми. При створенні компоновки для строк використовуються об'єкти `TableRow`. Чарунка може бути об'єктом `View` чи `ViewGroup`.

Компоновка `TableLayout` на практиці застосовується рідко, зазвичай замість неї використовують декілька компоновок `LinearLayout`. `TableLayout` зручно використати, якщо розміщення елементів представлено у вигляді таблиці (рисунк 2.4).

Наприклад, створимо компоновку `TableLayout`, розмістивши на неї кнопки як на рисунку 2.4.

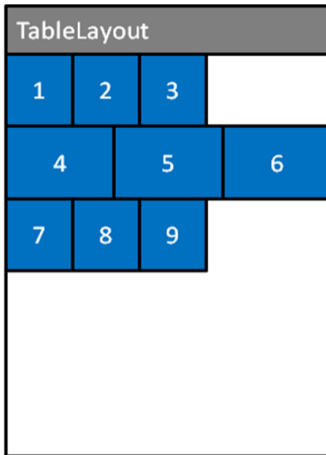


Рисунок 2.4 – Компоновка TableLayout

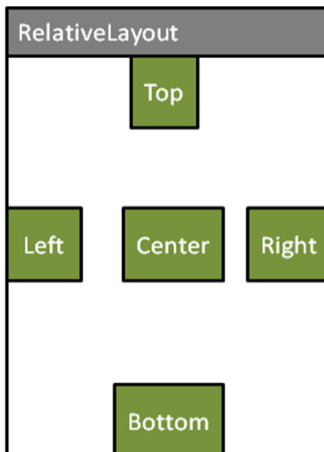
```

<TableLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow
        android:id="@+id/TR1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <Button
            android:id="@+id/b1"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="1"
        />
        ...
    </TableRow>
    <TableRow
        android:id="@+id/TR2"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <Button
            android:id="@+id/b4"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="4"
        />
        ...
    </TableRow>
    ...
</TableLayout>

```

2.2.4 Компонівка *RelativeLayout*

Компоновка *RelativeLayout* дозволяє дочірнім об'єктам визначати свою позицію відносно вказаного за ідентифікатором елемента. В *RelativeLayout* дочірні елементи розташовані так, що якщо перший елемент розташований у центрі екрану, тоді інші елементи будуть вирівняні відносно центру екрану. При об'яві компоновки в XML-файлі, елемент, на який будуть посилатися інші об'єкти для позиціонування, треба об'явити раніш ніж елементи, які звертаються до нього за його ідентифікатором. В компоновці *RelativeLayout* розташування елемента можна визначати відносно іншого елемента, на який посилаються через його ідентифікатор. Наприклад, якщо хочемо отримати ліворуч розташований елемент *TextView* від елемента *button2*, то: `android:layout_toLeftOf="@id/Button2"`. Розташування відносно вказаного: `android:layout_above`, `android:layout_below`, `android:layout_toLeftOf`, `android:layout_toRightOf`. Вирівнювання: `android:layout_alignBaseline`, `android:layout_alignBottom`, `android:layout_alignLeft`, `android:layout_alignRight`, `android:layout_alignTop`. Позиціонування відносно контейнеру: `android:layout_alignParentBottom`, `android:layout_alignParentLeft`, `android:layout_alignParentRight`, `android:layout_alignParentTop`, `android:layout_centerHorizontal`, `android:layout_centerInParent`, `android:layout_centerVertical`.



2.3 Базові віджети

Виджет – це об’єкт View, який служить інтерфейсом для взаємодії з користувачем. Тобто віджет – це елемент керування. Android забезпечує набір готових віджетів, таких як кнопки, текстові поля, перемикачі, полоси прокрутки, індикатори, слайдери, адаптери даних та ін.

Для кожного віджета потрібно визначити його атрибути. Основними атрибутами є наступні: **id**, **layout_width**, **layout_height**, **text**, **gravity**.

Ідентифікатор елемента є унікальним та задається у вигляді `android:id="@+id/view1"`, де `view1` – унікальний id елемента; символ “@” означає, що синтаксичний аналізатор XML повинен розгорнути дану строку і визначити цей вираз як ресурс ідентифікатора; символ “+” означає, що це нове ім’я ресурсу, яке потрібно бути створено і додано у ресурси в файл `R.java`, який середовище Android автоматично генерує.

Ширина елемента задається у вигляді `android:width="wrap_content"` (стандартний розмір) чи `android:width="match_parent"` (розмір відповідає батьківському елементу, тобто якщо елемент-батько займає весь простір екрану за шириною, то і у дочірнього елемента ширина буде займати ширину екрана).

Висота елемента `android:width` також може приймати значення “`wrap_content`” чи “`match_parent`” в залежності від потрібного застосування.

Текст елемента задається у вигляді `android:text="something"` чи `android:text="@string/res_str"`, де `res_str` – ім’я ресурсу (ресурс можна задати в каталогу `/res/values`).

Розташування елемента задається за допомогою `android:gravity` (“`left`”, “`right`”, “`center`” та ін.).

2.3.1 Віджет `TextView`

TextView застосовується для відображення тексту без можливості його редагувати користувачем, крім того він служить для відображення текстових даних в контейнерних віджетах для відображення списків (рисунок 2.5). Від `TextView` успадковуються інші елементи управління, на яких повинен

бути відображений текст (кнопки, прапорці, перемикачі). У елемента є багато атрибутів для роботи з текстом, наприклад:

- `android:textSize` (розмір в px, dp, sp, in, pt, mm);
- `android:textStyle` (“normal”, “bold”, “italic”);
- `android:textColor` (колір тексту в форматі #RRGGBB чи #AARRGGBB);
- `android:autoLink` (створення посилання зі значеннями “web”, “phone”, “email”, “map”, “all”, “none”).

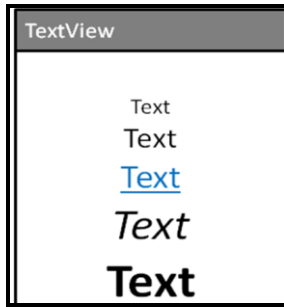


Рисунок 2.5 – Використання віджету TextView

Атрибути тексту можна задати програмно (в файлі *.java):

```
// id елемента T1
TextView T1=(TextView) findViewById(R.id.T1);
// встановлення тексту
T1.setText("something...");
```

2.3.1 Віджет EditText

EditText застосовується для відображення тексту з можливістю його редагування (рисунок 2.6). Основний метод класу – `getText()` з поверненим значенням `EditText`. За допомогою атрибуту `android:hint="Enter text..."` встановлюється підказка для введення тексту. В класі визначено метод для виділення тексту з певних позицій – `setSelection()`. Більшість методів роботи з текстом та його форматування успадковані від базового класу `TextView`.



Рисунок 2.6 – Віджет EditText

2.3.2 Полоси прокрутки

Полоси прокрутки в Android представлені віджетами **ScrollView** та **HorizontalScrollView**, які є контейнерними елементами і успадковуються від **ViewGroup**. Ці елементи – контейнери типу **FrameLayout**, це означає, що в них можна розмістити тільки одне дочірнє представлення. Цей дочірній елемент, в свою чергу, може бути контейнером зі складною ієрархією об'єктів. В якості дочірнього елемента для полос прокрутки зазвичай використовують **LinearLayout** з вертикальною чи горизонтальною орієнтацією елементів.

Віджет **ScrollView** підтримує тільки вертикальну прокрутку, тому для створення вертикальної та горизонтальної прокруток використовують обидва віджета. Звичайно **ScrollView** використовують в якості кореневого елемента, а **HorizontalScrollView** – дочірнього.

// файл компоновки main.xml

```
<ScrollView
    android:id="@+id/s_ver"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <HorizontalScrollView
        android:id="@+id/s_hor"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        // елемент (наприклад, TextView)
        <TextView
            android:id="@+id/t"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:isScrollContainer="true" />
        </HorizontalScrollView>
    </ScrollView>
```

```
// файл project.java
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Project extends Activity{
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView t=(TextView) findViewById(R.id.t);
        t.setText("..."); //ввести чи завантажити
    }
}
```

2.3.3 Відображення графіки

Відображення графіки виконується за допомогою віджету **ImageView**. Клас **ImageView** має змогу завантажувати зображення з різних джерел, таких як ресурси програмного додатку або зовнішні файли. В цьому класі існує декілька методів завантаження зображення (в файлі проекту *.java):

- `setImageResource(int resId)` – завантажує зображення за його ідентифікатором ресурсу;
- `setImageURI(Uri im_uri)` – завантажує зображення за його URI;
- `setImageBitmap(Bitmap bitmap)` – завантажує растрове зображення.

Крім того **ImageView** визначені методи для установки розміру зображення – `setMaxHeight()`, `setMaxWidth()`, `getMinimumHeight()`, `getMinimumWidth()`, а також його масштабування – `getScaleType()`, `setScaleType()`.

Для завантаження зображення в XML-файлі компоновки використовується атрибут **android:scr**. Ресурси зображення частіше за все поміщають в `/res/drawable/` проекту.

//1 вар. (завантаження `pic1` в *.xml)

```
<ImageView
    android:id="@+id=imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scr="@drawable/pic1">
```

//2 вар. (об'ява в *.xml, завантаження `pic2` в *.java)

//файл *.xml

```
<ImageView
    android:id="@+id=imageView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
```

//файл *.java

```
import android.widget.ImageView;
```

```
...
```

```
public class Project extends Activity{
```

```
...
```

```
    final ImageView image2 =(ImageView) findViewById
(R.id.imageView2);
    image2.setImageResource(R.drawable.pic2);
```

```
...
```

```
}
```

2.4 Обробка подій. Кнопки та прапорці

Після додавання віджетів в інтерфейс користувача, потрібно організувати взаємодію віджетів з користувачем. Для цього необхідно визначити обробник подій та зареєструвати його для даного елементу.

Клас View містить в собі колекцію вкладених інтерфейсів, які мають назву **On...Listener()**, в кожному з яких об'явлено єдиний абстрактний метод. Цей метод необхідно перевизначити в вашому класі. Його буде визивати система Android, коли з екземпляром View, до якого було підключено слухача подій, стане взаємодіяти з користувачем.

Клас View містить наступні інтерфейси:

- OnClickListener;
- OnLongListener;
- OnFocusChangeListener;
- OnKeyListener;
- onTouchListener;
- onCreateContextMenuListener.

Якщо потрібно, щоб віджет отримав повідомлення про виконану дію користувачем, потрібно в класі вікна (Activity) реалізувати інтерфейс обробника подій (наприклад, OnClickListener) та визначити метод його зворотного виклику (наприклад, onClick()), де буде міститися код обробки події, та зареєструвати слухач події за допомогою, наприклад, setOnClickListener.

2.4.1 Клас Button

Клас Button (кнопка) – найчастіше використовуваний елемент керування (рисунок 2.7). Зазвичай кнопка потребує написання коду обробки події натиснення onClick.



Рисунок 2.7 – Віджет Button

```
// файл *.xml
...
<Button
    android:id="@+id/b1"
    android:height="wrap_content"
    android:width="match_parent"
```

```

        android:text="Button 1">
// 1 вар. обробки натиснення кнопки (файл *.java)
...
import android.widget.*;
import android.view.View;
public class Project extends Activity{
    public void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.main);
        final Button b1 = (Button) findViewById(R.id.b1);
        final Button b2 = (Button) findViewById(R.id.b2);
        b1.setOnClickListener(new View.OnClickListener(){
            public void onClick(View v){
                // дії після натиснення кнопки 1
            }
        });
        b2.setOnClickListener(new View.OnClickListener(){
            public void onClick(View v){
                // дії після натиснення кнопки 2
            }
        });
    }
}

АБО

// 2 вар. обробки натиснення кнопки (файл *.java)
...
import android.widget.*;
import android.view.View;
public class Project extends Activity{
    public void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.main);
        final Button b1 = (Button) findViewById(R.id.b1);
        final Button b2 = (Button) findViewById(R.id.b2);
        b1.setOnClickListener(b1_click);
        b2.setOnClickListener(b2_click);
    }
    public OnClickListener b1_click=new OnClickListener(){
        public void onClick(View v){
            // дії після натиснення кнопки 1
        }
    };
    public OnClickListener b2_click=new OnClickListener(){
        public void onClick(View v){
            // дії після натиснення кнопки 2
        }
    };
};

```

```
}
```

АБО

```
// 3 вар. обробки натиснення кнопки (файл *.java)
// найліпший в перелічених варіантів
...
import android.widget.*;
import android.view.View;
import android.view.View.OnClickListener;
public class Project extends Activity implements
OnClickListener{
    public void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.main);
        final Button b1 = (Button) findViewById(R.id.b1);
        final Button b2 = (Button) findViewById(R.id.b2);
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }
    public void OnClick (View v){
        switch(v.getId()){
            case R.id.b1:
                // дії після натиснення кнопки 1
                break;
            case R.id.b2:
                // дії після натиснення кнопки 2
                break;
        }
    }
}
```

2.4.2 Клас *CheckBox*

Елемент **CheckBox** (прапорець) – це перемикач з двома станами (рисунок 2.8). Для програмного відстеження зміни стану елемента необхідно реалізувати інтерфейс **CompoundButton.OnCheckedChangeListener**.

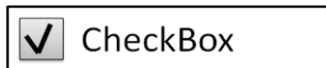


Рисунок 2.8 – Віджет **CheckBox**

```
// файл *.xml
...
<CheckBox
    android:id="@+id/cb1"
    android:height="wrap_content"
```

```

        android:width="match_parent"
        android:text="CheckBox 1">
// файл *.java
...
import android.widget.CheckBox;
import android.widget.CompoundButton;
public class Project extends Activity implements
CompoundButton.OnCheckedChangeListener{
    public void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.main);
        final CheckBox cb1 = (CheckBox) findViewById
(R.id.cb1);
        final CheckBox cb2 = (CheckBox) findViewById
(R.id.cb2);
        cb1.setOnCheckedChangeListener(this);
        cb2.setOnCheckedChangeListener(this);
    }
    public void OnCheckedChangeListener (CompoundButton v, Boolean
isChecked){
        switch(v.getId()){
            case R.id.cb1:
                if(isChecked){ // дії вибору CheckBox 1}
                    break;
            case R.id.cb2:
                if(isChecked){ // дії вибору CheckBox 2}
                    break;
        }
    }
}
}
}

```

2.4.3 Клас *RadioButton*

Віджети `RadioButton` (перемикачі) зазвичай використовують в складі групи контейнеру `RadioGroup`. Контейнер `RadioGroup` успадковується від `ViewGroup` і може бути використаний в якості кореневого елементу компоновки вікна, якщо на екрані є тільки група перемикачів, або в якості вкладеного в інший контейнер, наприклад в `LinearLayout`. Перемикачі дають змогу користувачу обрати тільки один із запропонованих варіантів. Основний метод зміни стану – `toggle()`, який інвертує стан перемикача. Крім того, від базового класу успадковуються інші методи, наприклад, `isChecked()` та `setChecked()`.

```

//файл *.xml
<RadioGroup
    Android:orientation="vertical"

```

```

Android:layout_width="match_parent"
    Android:layout_height="match_parent">
        <RadioButton
            Android:id="@+id/r1"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="Mode 1">
            ...
        </RadioButton>
    </RadioGroup>

//файл *.java
...
import android.widget.*;
import android.view.View;
import android.view.View.OnClickListener;
public class Project extends Activity implements
OnClickListener{
    public void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.main);
        final    RadioButton    r1    =    (RadioButton)
findViewById(R.id.r1);
        final    RadioButton    r2    =    (RadioButton)
findViewById(R.id.r2);
        final    RadioButton    r3    =    (RadioButton)
findViewById(R.id.r3);
        r1.setOnClickListener(this);
        r2.setOnClickListener(this);
        r3.setOnClickListener(this);
    }
    public void OnClick (View v){
        with(v.getId()){
            case R.id.r1:
                // дії після натиснення кнопки 1
                break;
            ...
        }
    }
}

```

Завдання на роботу

Створіть Android-проект простого текстового редактору з кнопками зміни розміру шрифту, вибору одного з типів шрифту (за допомогою RadioGroup) та вертикальної полоси прокрутки. В якості поля введення тексту використати EditText. В якості компоновки рекомендовано використати вкладені LinearLayout. Зразок текстового редактору показано на рисунку 2.9.

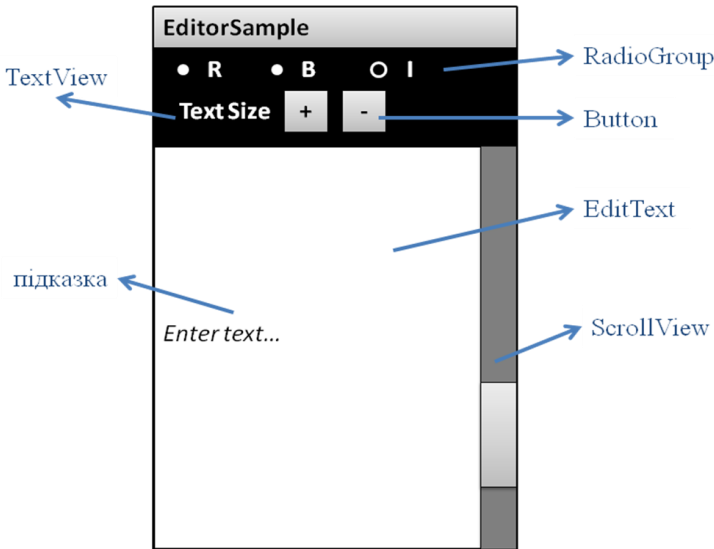


Рисунок 2.9 – Простий текстовий редактор

Зміст звіту

- титульний лист, тема та мета роботи;
- текст розробленої програми;
- результати розробленої програми;
- висновки;
- відповіді на контрольні питання.

Контрольні питання

1. Компонівка елементів управління на Android.
2. Типи компонок. Їх переваги та недоліки.
3. Поняття Activity та View.
4. Перелік основних атрибутів для віджетів.
5. Як створити новий ідентифікатор елементу?
6. Способи встановлення тексту віджету TextView.
7. Встановлення полос прокрутки.
8. Методи завантаження зображення з ImageView.
9. Механізм обробки подій Android-додатка.
10. Застосування віджету Button.
11. Застосування віджету CheckBox.
12. Застосування віджету RadioButton.

ЛАБОРАТОРНА РОБОТА №3

ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА. СПИСКИ

Мета роботи: ознайомитися із віджетами-списками та адаптерами даних.

Короткі теоретичні відомості

3.1 Віджети-списки та прив'язка даних

Для відображення даних у віджетах застосовуються адаптери, які призначені для з'єднання списку даних і відображає ці дані віджета. Самим простим адаптером для використання при з'єднанні даних є шаблонний клас **ArrayAdapter<T>**, наприклад:

```
String[] db = {"s1", "s2", "s3"};
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, db);
```

Конструктор класу **ArrayAdapter** приймає наступні параметри:

- об'єкт **Context** – екземпляр класу, який реалізує **Activity**;

- ідентифікатор ресурсу додатку, вбудовані системні ідентифікатори ресурсу – константи, які визначені в класі **android.R.layout** (**simple_list_item**, **simple_spinner_dropdown_item**, **simple_gallery_item**, **simple_list_item_checked...**);

- масив чи список типу **List<T>** об'єктів для відображення у віджеті.

3.1.1 Віджет **AutoCompleteTextView**

Віджет **AutoCompleteTextView** – це текстове поле з автозаповненням і можливістю редагування введеного тексту. Віджет є підкласом **EditText**, він дозволяє використовувати всі можливості редагування тексту. Віджет має властивість вказання мінімального числа символів, які повинен ввести користувач, для роботи функції автозаповнення списку – **android:completionThreshold**.

Для в'язання віджету з даним необхідно встановити адаптер через `setAdapter()`.

// файл *.xml

```
<LinearLayout xmlns:android = "http://schemas.android.
com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/t1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Choice:" />
    <Button
        android:id="@+id/b1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK" />
    <AutoCompleteTextView
        android:id="@+id/mylist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:completionThreshold="3" />
</LinearLayout>
```

// файл *.java

```
import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;
import android.text.*;

public class MainActivity extends Activity implements
    TextWatcher, OnClickListener{
    private TextView t1;
    private AutoCompleteTextView mylist;
    private String[] db = {"Jacob Anderson", "Joseph
Godwin", "Joshua Harrison", "Emma Lawson"};

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

        t1=(TextView)findViewById(R.id.t1);
```

```

        mylist                =                (AutoCompleteTextView)
findViewById(R.id.mylist);
        mylist.addTextChangedListener(this);
        mylist.setAdapter(new                ArrayAdapter<String>
(this,                android.R.layout.simple_dropdown_item_1line,
db));

        Button b1 = (Button) findViewById(R.id.b1);
        b1.setOnClickListener(this);
    }

    public void afterTextChanged(Editable s) {
        // TODO Auto-generated method stub
    }

    public void beforeTextChanged(CharSequence s, int start,
int count, int after) {
        // TODO Auto-generated method stub
    }

    public void onTextChanged(CharSequence s, int start, int
before, int count) {
        // TODO Auto-generated method stub
    }

    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        if (arg0.getId()==R.id.b1) t1.setText(mylist.getText());
    }
}

```

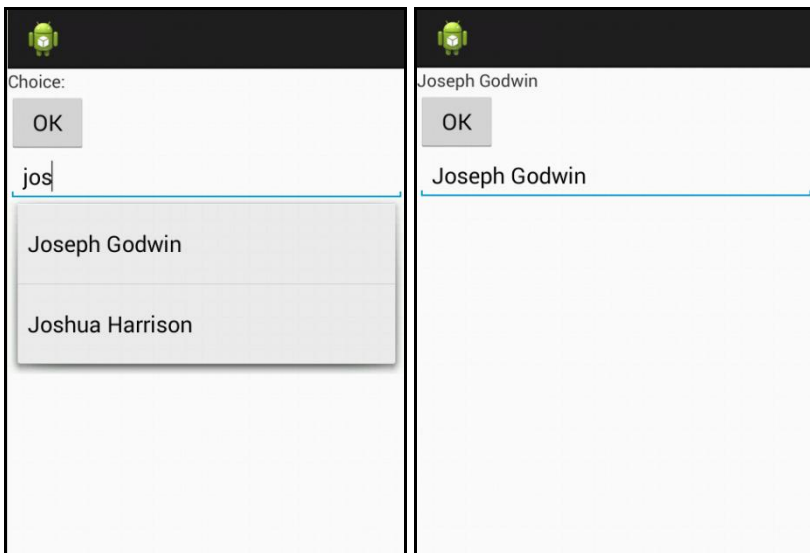


Рисунок 3.1 – Застосування віджету `AutoCompleteTextView`

3.1.2 Віджет `MultiAutoCompleteTextView`

Віджет `MultiAutoCompleteTextView` – це текстове поле з авто заповненням і можливістю редагуванням введеного тексту, який розширює функціональність

`AutoCompleteTextView`. Віджет показує автозаповнення для кожної підстроки тексту, яка розділена знаком пунктуації. Роздільник задається явно викликом метода `setTokenizer()`:

```
MultiAutoCompleteTextView mylist =
    (MultiAutoCompleteTextView) findViewById(R.id.mylist);
mylist.addTextChangedListener(this);

mylist.setAdapter(new ArrayAdapter<String> (this,
    android.R.layout.simple_dropdown_item_1line, db));

//встановлення роздільника
Mylist.setTokenizer(new MultiAutoCompleteTextView.
    CommaTokenizer());
```

3.1.3 Віджет *List*View

Віджет має вигляд вертикального списку із прокруткою. Зв'язані зі списком дані `ListView` отримує від об'єкту `ListAdapter`. На відміну від `AutoCompleteTextView` в якості базового класу використовується **ListActivity** замість `Activity`. Для з'єднання об'єкта `ListActivity` із даними необхідно розробити клас, який реалізує інтерфейс **ListAdapter**. Android забезпечує два стандартних адаптера списку: `SimpleAdapter` (статичне з'єднання даних невеликого обсягу), `SimpleCursorAdapter` (великий масив даних).

```
// файл *.xml
<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/t1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
// файл *.java
import android.os.Bundle;
import android.app.Activity;
```

```

import                                android.view.Menu;
import android.app.ListActivity;
import android.widget.*;
import android.view.View;

public class MainActivity extends ListActivity {
    private String[] db = {"Jacob Anderson", "Joseph
    Godwin", "Joshua Harrison", "Emma Lawson"};
    private TextView t1;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        t1=(TextView)findViewById(R.id.t1);
        setListAdapter(new ArrayAdapter<String>            (this,
        android.R.layout.simple_list_item_1, db));
    }

    public void onItemClick(ListView p, View v, int
    position, long id){
        t1.setText(db[position]);
    }
}

```

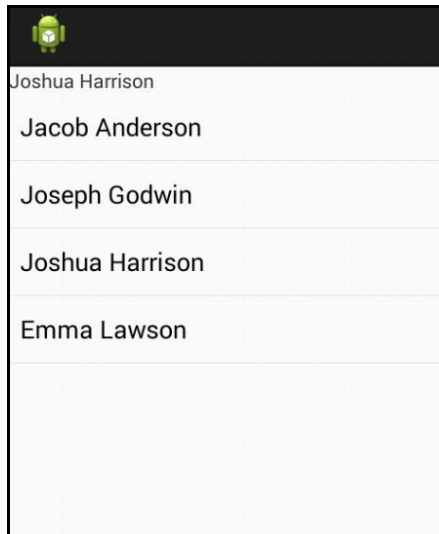


Рисунок 3.2 – Застосування віджету ListView

3.1.4 Створення списку із заданою компоновкою

Крім використання стандартних віджетів-списків можна створити список, визначив власну компоновку однієї строки цього списку. Такий підхід використовується для створення таблиць. Для з'єднання табличних даних використовуються класи `SimpleAdapter` та `SimpleCursorAdapter`. Конструктор `SimpleAdapter` має вигляд:

```
SimpleAdapter(this, list, R.layout.main, new
String[] {items.item1, items.item2, ...}, new int[]
{R.id.item1, R.id.item2, ...})
```

Приклад. Створимо список-довідник, в якому визначено ПІБ та номер телефону. В файлі `main.xml` опишемо два віджета `TextView` з іменами `name` та `phone` (компоновка горизонтальна). Для зберігання строки, яка представляє контакт, створимо окремий файл `Items.java`.

```
// Items.java
package com.samples.ui.lab;
import java.util.HashMap;
public class Items extends HashMap<String, String>{
    private static final long serialVersionUID=1L;
    public static final String NAME = "name";
    public static final String PHONE = "phone";
    public Items(String name, String phone){
        super();
        super.put(NAME, name);
        super.put(PHONE, phone);
    }
}

// lab.java
package com.samples.ui.lab;
import java.util.ArrayList;
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.*;
public class lab extends ListActivity{
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        ArrayList<Items> list = new ArrayList<Items>();
        // заповнення контактів
        list.add(new Items("Jacob Anderson", "420044"));
        list.add(new Items("Joseph Godwin", "336754"));
        list.add(new Items("Joshua Harison", "325588"));
```

```

        list.add(new Items("Emma Lawson",
"425522"));
        // адаптер даних
        ListAdapter a = new SimpleAdapter(this, list,
R.layout.main, new String[] {Items.NAME, Items.PHONE},
new int[] {R.id.name, R.id.phone});
        setListAdapter(a);
    }}

```



Рисунок 3.3 – Створення списку заданої компоновки

3.1.5 Віджет Spinner

Віджет Spinner – це аналог ComboBox для Android. Віджет має два режими `android:spinnerMode`: розгортаємого списку ("dropdown") та діалогу ("dialog"). Застосовується інтерфейс `AdapterView.OnItemSelectedListener`.

Приклад. В файлі проекту `main.xml` визначимо елементи `TextView` (`id=t1`) та `Spinner` (`id=s1`).

```

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
public class lab extends Activity implements
AdapterView.OnItemSelectedListener{
    private TextView t1;
    private final String[] db={...};
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        t1=(TextView)findViewById(R.id.t1);
        final Spinner s1=(Spinner) findViewById(R.id.s1);
        s1.setOnItemSelectedListener(this);
        ArrayAdapter<String> a = new ArrayAdapter<String>
(this, android.R.layout.simple_spinner_item, db);
        a.setDropDownViewResource(android.R.layout.simple_
spinner_dropdown_item);
    }
}

```

```

        s1.setAdapter(a);
    }

    public void onItemClick(AdapterView<?> p, View v, int
    position, long id){
        t1.setText(db[position]);
    }
    public void onNothingSelected(AdapterView<?> p){
        t1.setText("");
    }
}

```

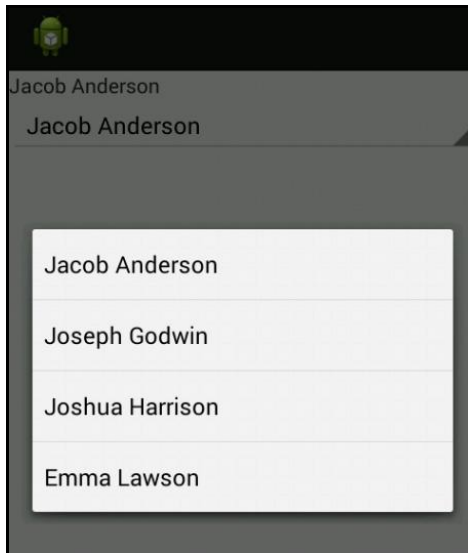


Рисунок 3.4 – Віджет Spinner (режим діалогу)

3.1.6 Відображення графіки в списках

Для відображення графіки в списках існують віджети **GridView**, **Gallery**, **SlidingDrawer**.

Для застосування віджетів необхідно наслідувати клас **BaseAdapter**. В класі адаптера масив даних повинен містити ідентифікатори графічних ресурсів, які будуть розміщуватися в каталозі `res/drawable/`. Наприклад:

```

private static final Integer[] im =
{R.drawable.photo1, R.drawable.photo2,...};

```

Приклад створення класу адаптера **MyImageAdapter.java**.
// MyImageAdapter.java
`import android.view.View;`

```

import android.content.Context;
import android.view.ViewGroup;
import android.widget.*;
public class MyImageAdapter extends BaseAdapter{
    private Context c;
    private static final Integer[] im = {R.drawable.p1,
R.drawable.p2,      R.drawable.p3,      R.drawable.p4,
R.drawable.p5, R.drawable.p6};

    public MyImageAdapter(Context cnt){c = cnt;}

    public int getCount() { return im.length; }

    public Object getItem(int arg0) { return im[arg0];}

    public long getItemId(int pos) {return im[pos];}

    public View getView(int position, View convertView,
ViewGroup parent) {
        ImageView v;
        if(convertView == null){
            v = new ImageView(c);
            v.setLayoutParams(new GridView.LayoutParams(85,85));
            v.setScaleType(ImageView.ScaleType.CENTER_CROP);
            v.setPadding(2, 2, 2, 2);
        }
        else v=(ImageView) convertView;
        v.setImageResource(im[position]);
        return v;
    }
}

```

1. Біджет **GridView**.

```

// main.xml
<TextView
    android:id="@+id/t1" .../>
<GridView
    android:id="@+id/gv"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="auto_fit"
    android:columnWidth="100px"
    android:stretchMode="columnWidth"
    android:gravity="center"
    android:verticalSpacing="35px"
    android:horizontalSpacing="5px" />
...
//main.java
import android.os.Bundle;
import android.app.Activity;

```

```

import                                android.view.View;
import android.widget.*;
public class Main extends Activity
implements AdapterView.OnItemClickListener{
private TextView t1;
private MyImageAdapter ia;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    t1 = (TextView) findViewById(R.id.t1);
    final GridView gv = (GridView) findViewById(R.id.gv);
    ia = new MyImageAdapter(getApplicationContext());
    gv.setAdapter(ia);
    gv.setOnItemClickListener(this);
}

public void onItemClick(AdapterView<?> a0, View a1, int
a2, long a3) {
    // TODO Auto-generated method stub
    t1.setText("Select:  "+a2+"    from    "+a0.getCount()+"";
    id="+ia.getItem(a2));
}}

```



Рисунок 3.5 – Віджет GridView

2. Віджет **Gallery**.

Для застосування даного віджету відкоригуємо файл **MyImageAdapter.java**:

```

public View getView(int position, View convertView,
ViewGroup parent) {
    ImageView v;
    v = new ImageView(c);
    v.setImageResource(im[position]);
    v.setPadding(20, 20, 20, 20);
}

```

```

v.setLayoutParams (new
Gallery.LayoutParams(140, 140));
v.setScaleType (ImageView.ScaleType.FIT_XY);
return v;
}

```

А також у файлі Main.java змінимо строку створення об'єкту:

```
final Gallery gv = (Gallery) findViewById(R.id.gv);
```



Рисунок 3.6 – Віджет Gallery

Завдання на роботу

Створіть Android-проект, який містить в собі список типу ListView. Кожен елемент списку містить текст та зображення. Результат обраного користувачем елементу списку відображається в окремому текстовому полі.

Примітка. Для розробки списку складної компоновки необхідно створити файли *.xml головної форми та окремої строки списку, що складається з текстового поля та зображення (наприклад, activity_main.xml та item.xml); створити декілька *.java файлів: 1) клас, який містить типи списку (наприклад, клас myobject, що містить строку string str та ідентифікатор зображення int image); 2) клас адаптеру списку (наприклад, ItemAdapter.java) з наступним конструктором:

```

public ItemAdapter(Context _c, ArrayList<myobject> _ob)
{
    c = _c;
    ob = _ob;
    layoutinf = (LayoutInflater) c.getSystemService(
Context.LAYOUT_INFLATER_SERVICE);
}

```

та методом getView, який ініціює ресурси списку:

```

public View getView(int pos, View cv, ViewGroup
parent) {
    View v = cv;
    if (v==null){
        v=layoutinf.inflate(R.layout.item,parent,false);
    }

    myobject obs = getObject(pos);

    ((TextView) v.findViewById(R.id.t)).setText(obs.str);

    ((ImageView)v.findViewById(R.id.im)).setImageResource(ob
s.image);

    return v;
}

```

3) головний клас, який встановлює адаптер списку, його ресурси та виконує операцію вибору елементу списку за допомогою методу `setOnItemClickListener` (`android.widget.AdapterView.*`).

Зміст звіту

- титульний лист, тема та мета роботи;
- текст розробленої програми;
- результати розробленої програми;
- висновки;
- відповіді на контрольні питання.

Контрольні питання

1. Як прив'язати (встановити) дані до списку?
2. Як встановити адаптер даних?
3. Які параметри приймає адаптер `ArrayAdapter`?
4. Об'єкт `Context`.
5. Об'єкт `LayoutInflater`.
6. Віджет `AutoCompleteTextView`.
7. Віджет `MultiAutoCompleteTextView`.
8. Віджет `ListView`.
9. Віджет `Spinner`.
10. Відображення графіки у списках.

ЛАБОРАТОРНА РОБОТА №4

ОРГАНІЗАЦІЯ ОБМІНУ ДАНИХ У ГІБРИДНИХ МОБІЛЬНИХ ДОДАТКАХ

Мета роботи: ознайомитися з організацією процесів та взаємодією компонентів в Android-додатках за допомогою об'єктів Intent. Навчитися передавати дані у викликаний Activity та зворотне отримання інформації із викликаного Activity з використанням extra-параметрів об'єктів Intent.

Короткі теоретичні відомості

4.1 Процеси в системі Android

Система Android запускає процес, який містить єдиний потік для виконання, коли хоча б один з компонентів додатка (або весь додаток) буде запитано. Всі компоненти додатка виконуються в цьому процесі та потоку за замовченням.

Всі компоненти ініціюються в основному потоку процесу. Окремі потоки для кожного екземпляру не створюються. Отже, всі методи зворотного виклику, які визначені в компоненті та викликані системою, завжди працюють в основному потоку процесу. Виходячи з цього, компонент не повинен виконувати в методах зворотного виклику довготривалі операції (завантаження файлів з мережі, цикли обчислення і т.д.) або блокувати системний виклик, бо це блокує будь-які компоненти в цьому процесі. Для таких операцій породжують окремі потоки.

Система Android може завершити процес у випадку нестачі пам'яті або якщо пам'ять затребувана більш важливими процесами. Прикладні компоненти, які виконуються в таких процесах, будуть знищені. Процес буде пере запуснено для компонентів у випадку їх повторного виклику.

При виборі процесу для знищення Android оцінює відносну важливість цього процесу з точки зору користувача. Процеси з низькою важливістю знищуються в першу чергу. Існує п'ять рівнів в ієрархії важливості. Наступний список представляє їх в порядку зменшення важливості:

- *активний процес* (Foreground Process). Процес вважається активним, якщо з ним взаємодіє користувач, процес має об'єкт Service чи BroadcastReceiver при виконанні методу зворотного виклику для цього об'єкту;
- *видимий процес* (Visible Process) – компонент із цього процесу може викликатися користувачем. Це може бути процес Activity, який не знаходиться в фокусі, але ще видимий користувачу;
- *сервісний процес* (Service Process) – процес, в якому виконується Service та який не відноситься до жодної з перелічених вище категорій;
- *фоновий процес* (Background Process) – процес, в якому виконується Activity, яку в теперішній час не бачить користувач;
- *пустий процес* (Empty Process) – не містить жодних активних компонентів. Цей процес зберігається тільки як кеш, для того щоб зменшити час запуску виклику компонента.

4.2 Стани Activity

Activity може знаходитися в трьох станах:

- *активний* (active) – Activity знаходиться на передньому плані екрана мобільного пристрою;
- *призупинений* (paused) – Activity не має фокусу, але все ще видимо користувачу;

– *зупинений* (stopped) – Activity повністю закрито іншими Activity при переході з одного стану в інший отримує повідомлення через захищені методи: *onCreate()* – при створенні Activity, *onRestart()* – при відновленні роботи Activity, *onStart()* – викликається безпосередньо при видимості Activity, *onResume()* – викликається безпосередньо при початку взаємодії користувача з Activity, *onPause()* – викликається, коли система намагається запустити інший Activity, *onStop()* – викликається, коли Activity стає невидимим, *onDestroy()* – при знищенні Activity. Ці методи можна реалізувати в класі Activity, для того щоб виконати певні дії при зміні стану даного Activity. Наприклад:

```
protected void onPause() { super.onPause(); ... }
```

4.3 Запуск Activity з використанням об'єктів Intent

Компоненти Android-додатка, в тому числі Activity, запускаються через об'єкти **Intent**. Це засіб пізнього зв'язування

під час виконання між компонентами одного чи декількох додатків. В кожному випадку система Android знаходить відповідний Activity, для того щоб відповісти на Intent та ініціалізує його у випадку необхідності. Об'єкти Intent можна розділити на дві групи:

- *явний Intent* – визначає цільовий компонент за ім'ям;
- *неявний Intent* – не називає адресата.

Явний Intent використовують для повідомлень в межах додатку, наприклад, коли один Activity запускає інший Activity із цього додатку.

Неявний Intent використовують для запуску компонентів інших додатків. В файлі маніфесту додатку зазвичай декларується фільтр Intent.

В системі Android всі Activity зберігаються в стеку. Коли один Activity запускає інший, новий Activity розміщується в стек і стає активним Activity. Під час роботи користувача, поточний Activity виштовхується із стеку та його заміщує попередній Activity, який знов відображується на екрані.

4.4 Запуск Activity за допомоги явного об'єкту Intent

Об'єкт Intent являється структурою даних, яка містить абстрактний опис виконуваної операції. Для того щоб викликати інший Activity, в об'єкт Intent треба передати ім'я цього Activity. Ім'я встановлюється методами `setComponent()`, `setClass()` або `setClassName()`. Для запуску Activity, об'єкт Intent передають в метод `Context.startActivity()`. Цей метод приймає єдиний параметр – об'єкт Intent, який описує Activity, що запускається. Наприклад, викликати Activity з ім'ям `NewActivity` в коді програми можна наступним чином:

```
Intent intent = new Intent();
// встановлюємо ім'я викликаного компоненту
intent.setClass(getApplicationContext(),
                    NewActivity.class);
// запускаємо компонент
startActivity(intent);
```

Для того щоб додаток мав змогу бачити другий Activity, необхідно вказати його в файлі маніфесту додатка. Для цього необхідно вибрати в представленні **Project Explorer** файл **AndroidManifest.xml** та у відкритому редакторі маніфесту перейдіть на вкладку **Application**. На панелі **Application Nodes**

натисніть кнопку **New** та у відкритому діалоговому вікні оберіть елемент **Activity**. На панелі **Application Nodes** буде додано новий вузол Activity. Якщо його виділити, праворуч відобразиться панель із заголовком **Attributes for Activity**. На цій панелі в строчці **Name** натисніть кнопку **Browse** та в відкритому вікні оберіть клас нової Activity (NewActivity). Після цих дій в файлі AndroidManifest.xml з'явиться додатковий елемент `<activity>` з атрибутом `android:name = "NewActivity"`.

4.5 Виклик стандартних Activity для додатку

Для виклику системних компонентів в класі Intent визначено набір констант Standard Activity Actions, що визначають дії запуску стандартних Activity. Наприклад, дія ACTION_DIAL ініціалізує звернення по телефону у вигляді вікна набору телефонного номера, а в класах MediaStore та MediaStore.Audio.Media визначені константи дії для запуску вікон з медіаплеєром, пошуку музики та запису з мікрофону; в класах DownloadManager – завантаження файлів через Інтернет, RingtoneManager – для запуску вікна, що встановлює режим дзвінка та ін. Викликати стандартне Activity можна наступним чином:

```
Intent intent = new Intent("Intent.ACTION_DIAL");
startActivity(intent);
```

4.6 Обмін даними між Activity

Інколи є потреба повернути результат Activity, коли він закривається. Для того щоб запустити Activity та отримати результат його виконання, необхідно викликати метод `startActivityForResult(Intent, int)` зі другим параметром, який ідентифікує запит. Результат повертається через метод `onActivityResult(int, int, Intent)`, що визначений в батьківському Activity.

Крім того, можна передавати додаткові параметри (extra-параметри) у викликаємий Activity. Це пари ключ-значення для інформації, яку треба надати викликаємому компоненту. Об'єкт Intent має ряд методів `put...()` для вставки різного типу додаткових даних та аналогічного набору методів `get...()` для читання даних. Extra-параметри встановлюються та читаються

як об'єкти класу `Bundle` із використанням методів `putExtras()` і `getExtras()`.

Наприклад, запустити `Activity` с ім'ям класу `EditContactActivity` та передати йому два додаткових параметра із головного вікна зі списком контактів `ContactListActivity` можна наступним чином:

```
// ідентифікатор запиту
private static final int IDM_EDIT = 102;
private long mId = -1;
...
// в класі ContactItem визначений список контактів з
// полями NAME та PHONE, а також визначені методи
// getName(), getPhone(), setName(String), setPhone(String)
ArrayList<ContactItem> mList;
private ListAdapter mAdapter;
...
mList = new ArrayList<ContactItem> ();
...
mId=this.getSelectedItemId();
ContactItem mItem = mList.get((int)mId);
Intent intent = new Intent();
// додавання extra-параметрів
intent.putExtra(ContactItem.NAME, mItem.getName());
intent.putExtra(ContactItem.PHONE, mItem.getPhone());
// визначення класу запускаємого Activity
intent.setClass(this, EditContactActivity.class);
// виклик Activity
startActivityForResult(intent, IDM_EDIT);
```

Коли дочірній `Activity` закриється, в ньому можна викликати метод `setResult(int)`, для того щоб повернути дані в батьківський `Activity`. Цей метод повертає код результату закриття `Activity`, який може бути стандартним результатом, що визначений константами `RESULT_CANCELED`, `RESULT_OK` або `RESULT_FIRST_USER`.

```
private EditText mName;
private EditText mPhone;
...
Intent t = new Intent();
// вставляємо ім'я та телефон людини
t.putExtra(ContactItem.NAME, mName.getText().toString());
t.putExtra(ContactItem.PHONE, mPhone.getText().toString());
// повертаємо результат у викликаючий Activity
setResult(RESULT_OK, t);
finish();
```

Крім того, дочірній Activity довільно повернути об'єкт Intent, який містить будь-які додаткові дані. Вся ця інформація в батьківському Activity з'явиться через метод зворотного виклику `Activity.onActivityResult()`, разом із ідентифікатором, який було передано в метод `startActivityResult()` при виклику Activity:

```
protected void onActivityResult( int requestCode, int
resultCode, Intent data){
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK){
        Bundle extras = data.getExtras();
        switch(requestCode){
            case IDM_ADD:
                mList.add(new ContactItem(
                    extras.getString(ContactItem.NAME),
                    extras.getString(ContactItem.PHONE)));
                break;
            case IDM_EDIT:
                mList.set((int)mId, new ContactItem(
                    extras.getString(ContactItem.NAME),
                    extras.getString(ContactItem.PHONE)));
                break;
        }
        ...
    }
}
```

Треба зауважити, що для коректної взаємодії Activity, в файлі маніфесту необхідно додати окрім головного Activity інші Activity, додавши атрибути `android:name`, `android:label`:

```
<application ...>
    <activity android:name=".ContactListActivity"
        android:label="@string/app_name">
        ...
    </activity>
    <activity android:name=".NewContactActivity"
        android:label="@string/title_add">
    </activity>
    <activity android:name=".EditContactActivity"
        android:label="@string/title_edit">
    </activity>
</application>
```

4.7 Intent-фільтри та запуск завдань

Intent-фільтри декларують обмеження компонента по прийому неявних об'єктів Intent, які він має змогу обробляти.

Якщо компонент не має жодних фільтрів, він може приймати тільки явні об'єкти Intent. Компонент з фільтрами може приймати як явні так і неявні Intent. В Intent декларуються тільки три складові об'єкту Intent: дія, дані та категорія. Наприклад, в будь-якому додатку є головний Activity, який встановлюється як точка входу для завдання:

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

Для того щоб Intent запусив компонент, котрому належить фільтр, він повинен чітко пройти всі три тести. Проте, якщо у об'єкта є декілька Intent-фільтра, то у випадку не проходження одного з них, проходить перевірку на наступний Intent-фільтр.

Наприклад, створимо окремий додаток для виклику попереднього додатку ContactListActivity з прикладу, описаного у пункті 4.6, та зробимо зміни в файлі маніфесту:

```
<activity
    android:name=".ContactListActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.
            LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name=" com.samples.app.contact.
            VIEW_CONTACTS" />
        <category android:name="android.intent.category.
            DEFAULT" />
    </intent-filter>
</activity>
```

В коді класу нового проекту буде створюватися об'єкт Intent з визначеною дією: `startActivity(new Intent("com.samples.app.contact.VIEW_CONTACTS"))`;

Завдання на роботу

Створіть Android-проект, який реалізує механізм обміну даними за варіантом (таблиця 4.1).

Таблиця 4 – Теми проектів мобільних додатків

Номер варіанту	Тема проекту
1	Телефонна книга
2	Журнал обліку викладача
3	Кулінарна книга рецептів
4	Здоровий спосіб життя. Фітнес
5	Щоденник справ (записник)
6	Психологічне тестування
7	Медіа-програвач
8	Програма створення списку покупок
9	Тестування з англійської мови
10	Handmade

Зміст звіту

- титульний лист, тема та мета роботи;
- текст розробленої програми;
- результати розробленої програми;
- висновки;
- відповіді на контрольні питання.

Контрольні питання

1. Пріоритети процесів в Android.
2. Як ініціюються в потоці методи одного компоненту?
3. Стани знаходження Activity.
4. Який ланцюг повідомлень захищених методів отримує Activity до взаємодії його з користувачем?
5. Для чого використовуються об'єкти Intent?
6. Види об'єктів Intent.
7. Які необхідно зробити зміни в файлі маніфесту для коректної роботи декількох Activity?
8. Механізм запуску нової Activity та передача йому батьківських параметрів.
9. Механізм повернення даних нової Activity до батька.
10. Intent-фільтри та запуск завдань.

ЛІТЕРАТУРА

1. И.Салмре Программирование мобильных устройств на платформе .NET Compact Framework. “Вильямс”, 2006. - 736 с.
2. Голощапов А. Л. Google Android: программирование для мобильных устройств / Голощапов А. Л. – СПб. : БХВ-Петербург, 2011. – 448 с.
3. Р.Майер. Android 2: программирование приложений для планшетных компьютеров и смартфонов. – М.: Эксмо, 2011. – 672 с.
4. Горнаков С.Г. Программирование мобильных телефонов на Java 2 Micro Edition / С.Г. Горнаков. – М.: ДМК Пресс, 2004. – 336 с.
5. Пирумян В.В. Платформа программирования J2ME для портативных устройств / В.В. Пирумян Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2002. – 352 с.
6. Архипкин В.Я. Bluetooth. Технические требования. Практическая реализация / В.Я. Архипкин, А.В. Архипкин. – М.: Мобильные коммуникации, 2004. – 203 с.
7. Дэрс Л. Android за 24 часа. Программирование приложений под операционную систему Google / Дэрс Л., Кондер Ш. – М. : Рид Групп, – 464 с.
8. Майер Р. Android 2: программирование для планшетных компьютеров и смартфонов / Ретро Майер ; пер. с англ.– М. : Эксмо, 2011. – 672 с.