МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені В.О. Сухомлинського

К. Т. КУЗЬМА

ПРОГРАМУВАННЯ МОБІЛЬНИХ ПРИСТРОЇВ

Навчальний посібник
для дистанційного навчання

Миколаїв
2021

УДК 004.432.2
ББК 32.973
К89

РЕЦЕНЗЕНТИ**:**

УДОВИК І. М., канд. техн. наук, доцент, завідувач кафедри програмного забезпечення комп'ютерних систем Національного технічного університету «Дніпровська політехніка»;

ПОГРОМСЬКА Г. С., канд. пед. наук, доцент, доцент кафедри теорії й методики природничо-математичної освіти та інформаційних технологій Миколаївського обласного інституту післядипломної педагогічної освіти.

Рекомендовано вченою радою
Миколаївського національного університету імені В. О. Сухомлинського
(протокол № 19 від 31.05.2021 р.)

Кузьма К. Т.

К89 Програмування мобільних пристроїв: навчальний посібник для дистанційного навчання / К.Т. Кузьма. – Миколаїв: СПД Румянцева Г. В., 2021. – 128 с.

ISBN 567-859-837-548-4

Навчальний посібник розроблений у відповідності з освітньо-професійною програмою підготовки бакалавра галузі знань 12 «Інформаційні технології» спеціальності 123 «Комп'ютерна інженерія», містить лекційний матеріал та завдання до лабораторних робіт з навчальної дисципліни «Периферійні та мобільні пристрої (Частина 2. Програмування мобільних пристроїв)». Наведений у посібнику матеріал може бути використаний при проведенні аудиторних, індивідуальних і самостійних занять. Посібник буде корисним для студентів інших спеціальностей галузі знань 12 «Інформаційні технології».

УДК 004.432.2
ББК 32.973

ISBN 567-859-837-548-4

Зміст

Вступ

У навчальному посібнику «Програмування мобільних пристроїв» розглядаються теоретичні та практичні питання з основ розробки мобільних додатків для операційної системи Android, вивчення інтегрованого середовища розробки Android Studio.

Навчальний посібник містить теоретичний лекційний матеріал, практичні завдання у вигляді лабораторних робіт, контрольні запитання до кожної лабораторної роботи, тестові завдання для перевірки залишкових знань, що забезпечує ефективність його використання під час дистанційного навчання.

Основні теми посібнику присвячені: етапам розробки мобільного додатку, проєктуванню та реалізації інтерфейсу додатку, роботі із адаптерами, списками, фрагментами, «активностями» (activities), намірами (intents) та системою управління базами даних SQLite.

Навчальний посібник складений у відповідності до програми навчальної дисципліни «Периферійні та мобільні пристрої (Частина 2. Програмування мобільних пристрої)» підготовки бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 123 «Комп'ютерна інженерія».

Кредит 1. Основні етапи розробки мобільного додатка

Тема 1. Введення в програмування для мобільних пристроїв

На першому етапі розробки мобільного додатку визначається варіант його реалізації. Можливі наступні варіанти – «нативний» (рідний), «гібридний» або мобільний (адаптивний) веб-сайт.

«Нативні» (рідні) додатки створюються на тих мовах програмування, які відповідають платформі, під яку вони реалізуються. Наприклад, для iOS: Swift або Objective-C, для Android: Java або Kotlin. Перевагами нативних додатків є краща продуктивність під час рендерінгу. Недоліком, відсутність кросплатформленності: потрібно окремо реалізовувати додатки як для Android так і для iOS.

На рисунку 1.1 графічно відображено варіанти реалізації мобільних додатків.

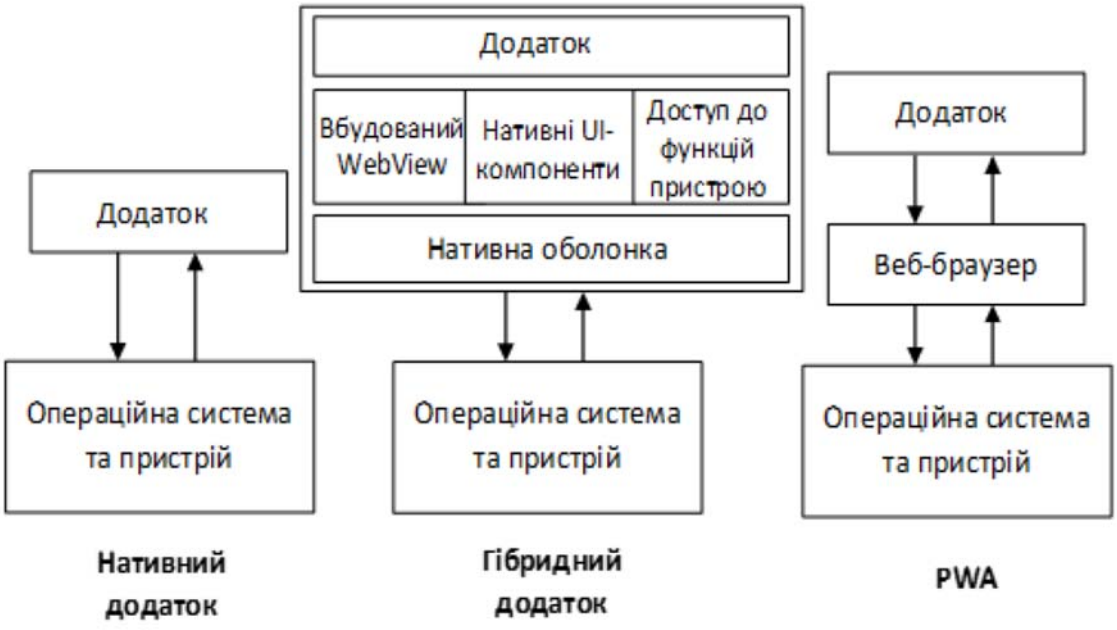


Рис. 1.1. Варіанти реалізації мобільних додатків

«Гібридний» додаток містить вбудоване WebView (веб-представлення), нативні компоненти дизайну (UI), нативну оболонку, яка взаємодіє із операційною системою пристрою та WebView. Таким чином забезпечується доступ мобільного додатку до функцій мобільного пристрою: камери, файлова система, GPS. «Гібридні» застосунки створюються з використанням додаткових, не офіційних, середовищ розробки, наприклад як ApacheCordov.

В посібнику розглядаються питання створення саме «нативних» застосунків.

Тема 2. Огляд платформи Android

Мобільні пристрої, які працюють на ОС Android не виконують файли .class та .jar, а застосовують власні формати компільованого коду. Це дозволяє підвищити швидкість та ефективність використання акумуляторів мобільних пристроїв. Таким чином, одного лише IDE для розробки на мові Java не достатньо для розробки мобільних додатків. Необхідні спеціальні додаткові інструменти, які б забезпечували компіляцію коду в Android-форматі, надавали можливість його встановлення на мобільних пристроях та налаштування програми.

Такій набір додаткових інструментів називається пакетом AndroidSDK. Пакет AndroidSoftwareDevelopmentKit містить бібліотеки та інструменти, необхідні для розробки Android-додатків.

Зараз існує декілька середовищ розробки:

- NetBeans;
- Eclipse;

– Intellij IDEA;

– Android Studio.

Android Studio орієнтована саме під ОС Android, а також не вимагає установки додаткових плагінів. Приклади виконання завдань в даному посібнику будуть розглядатися на Android Studio.

Розглянемо мови розробки нативних додатків.

Java - офіційна мова програмування, підтримуваний середовищем розробки Android Studio. На Java посилається більшість офіціальної документації Google.

Kotlin - мова була офіційно представлений травнем 2017 року на Google I / O і позиціонується Google як другу офіційну мову програмування під Android після Java. Kotlin сумісний з Java і не викликає зниження продуктивності та збільшення розміру файлів. відмінність від Java в тому, що він вимагає менше службового коду, тому легший для читання.

Більш низькорівневі мови підтримуються Android Studio із використанням

користуванням Java NDK (Native Development Kit). Це дозволяє писати нативні додатки, що може стати в нагоді для створення ігор або інших ресурсоємних програм. Приклади, наведені в даному навчально-методичному посібнику, написані на мові Java.

Android Studio є офіційною IDE (інтегрованим середовищем розробки) для розробки програм для Android від Google. Він доступний для безкоштовного завантаження в Windows, Mac OS X та Linux.

Він має дуже простий редактор макетів, який підтримує перетягування для проектування інтерфейсу користувача, де можна

перетягувати макет, віджети, текстові поля тощо та перекидати його на віртуальний мобільний екран для проектування інтерфейсу Android. Він також має опцію властивостей, де можна легко заповнити атрибут, такий як колір, текст тощо, щоб надати привабливий і гарний вигляд інтерфейсу. У Studio також є опція Text, де можна побачити XML-код інтерфейсу та відредагувати його.

Платформа Android складається з багатьох компонентів. У неї входять базові застосунки (наприклад, Контакти), набір програмних інтерфейсів (API) для управління зовнішнім виглядом і поведінкою застосунків, а також багатьох допоміжних файлів і бібліотек (рис.1.2).

Рис. 1.2. Компоненти платформи Android

Кожен Android-застосунок складається з екранів, а кожен екран складається з активності і макета. Активність - одна чітко визначена операція, яку може виконати користувач. Наприклад, в застосунку можуть бути присутніми активності для складання повідомлення електронної пошти, знайти контакт або створення знімка. Активності зазвичай асоціюються з одним екраном і програмуються на Java (або Kotlin). Макет описує зовнішній вигляд екрану. Макети створюються у вигляді файлів в

розмітці XML і повідомляють Android, де розташовуються ті чи інші елементи екрану.

Розглянемо послідовність дій взаємодії пристрою Android, активності та макету:

1) Пристрій запускає застосунок і створює об’єкт активності.

2) Об’єкт активності визначає макет.

3) Активність дає команду на вивід макету на екран.

4) Користувач взаємодіє із макетом, що відображається на екрані пристрою.

5) Активність реагує на події макету та виконує відповідний код застосунку.

6) У разі необхідності активність у відповідь на дії користувача оновлює дані макету;

7) Користувач бачить зміни на екрані пристрою.

Лабораторна робота №1. Налаштування середовища програмування Android Studio. Створення проекту в Android Studio

1.1 Установка програмного забезпечення.

1.2 Створення проекту в Android Studio.

1.3 Завдання.

1.4 Контрольні запитання.

1.1 Установка програмного забезпечення

Першим етапом встановлення програмного забезпечення для розробки мобільних додатків є установка Java SE Development Kit (сторінка

завантаження: https://www.oracle.com/java/technologies/javase-downloads.html). Java Development Kit (JDK) - це реалізація будь-якої з платформ, випущених корпорацією Oracle у вигляді двійкового продукту, спрямованого для розробників Java під Solaris, Linux, macOS або Windows. JDK включає приватний JVM та кілька інших ресурсів для завершення розробки Java-програми. [2] З моменту впровадження платформи Java вона стала на сьогодні найбільш широко використовуваним набором програмного забезпечення (SDK- Software Development Kit). JDK утворює розширену підмножину SDK. JDK включає інструменти для розробки, налагодження та моніторингу програм Java.

Другим етапом є власне встановлення Android Studio . Це безкоштовне інтегроване середовище розробки. Перший стабільний реліз Android Studio 1.0. компанія Google випустила у 2014. Веб-сайт: developer.android.com/studio/index.html. Останній стабільний випуск – версія 4.0.1 у липні 2020 року.

У поточній стабільній версії надаються такі функції:

– підтримка побудови проекту на основі Gradle (інструмент автоматизації упакування, збирання проекту. Він контролює процес розробки в задачах складання та упаковки для тестування, розгортання та публікації);

– Android-рефакторинг та швидкі виправлення;

– інструменти Lint (інструмент, який аналізує вихідний код для позначення помилок програмування, багів, стилістичних помилок та підозрілих конструкцій) для вирішення проблем із продуктивністю, доступністю, сумісністю версій тощо;

– ProGuard інтеграція (ProGuard - це інструмент командного рядка з відкритим кодом, який скорочує, оптимізує код Java);

– майстри на основі шаблонів для створення загальних дизайнів та компонентів Android;

– багатий редактор макетів, що дозволяє користувачам перетягувати компоненти інтерфейсу користувача, можливість перегляду макетів на кількох екранах;

– підтримка створення програм Android Wear (засоби для розробки застосунків для смарт годинників);

– Вбудована підтримка Google Cloud Platform, яка дозволяє інтегруватися з Firebase Cloud Messaging (раніше "Cloud Cloud Messaging") та Google App Engine.

– Віртуальний пристрій Android - емулятор (Android Virtual Device) для запуску та налагодження програм у студії Android.

1.2 Створення проекту в Android Studio

Існують два варіанти створення проекту в Android Studio:

1) На початковому екрані обирається пункт «Start new Android Project», якщо додаток завантажується уперше.

2) Команда «File -> New-> New Project...» в головному меню обирається, якщо додаток вже завантажувався.

Етапи :

I) Налаштування властивостей проекту:

– «Application Name» – ім’я додатку;

– «Company Domain» – пакет класів, в якому будуть розміщуватися головний клас додатку.

– «У разі розміщення» в Google Play, значення даного поля повинне бути унікальним для всього PlayMarket.

– Project Location – фізичне розташування файлів проекту на вінчестері.

II) Встановлюється мінімальна версія SDK – за замовченням @API 15: Android 4.0.3, яка підтримує 100% пристроїв Android.

III) обирається шаблон проекту (найбільш розповсюдженими є Basic Activity та Empty Activity). Обираємо Empty Activity, на наступному етапі встановлюємо налаштування:

а) текстові поля:

– Activity Name: MainActivity – назва головного класу додатку;

– Layout Name: activity_main - назва xml, в якому буде зберігатися визначення візуального інтерфейсу.

б) поля checkbox:

– Generate Layout File: встановлення необхідності генерування xml файлу, в якому буде зберігатися визначення візуального інтерфейсу.

– Backwards Compatibility (AppCompat): якщо true, можливо встановити сумісність різних версій Android.

III) На третьому етапі всі налаштування залишаємо встановленими за замовченням. Після натискання Finish створюється і відкривається новий проект.

Структуру проекту, яка створюється за замовченням, зображено на рисунку 1.3.

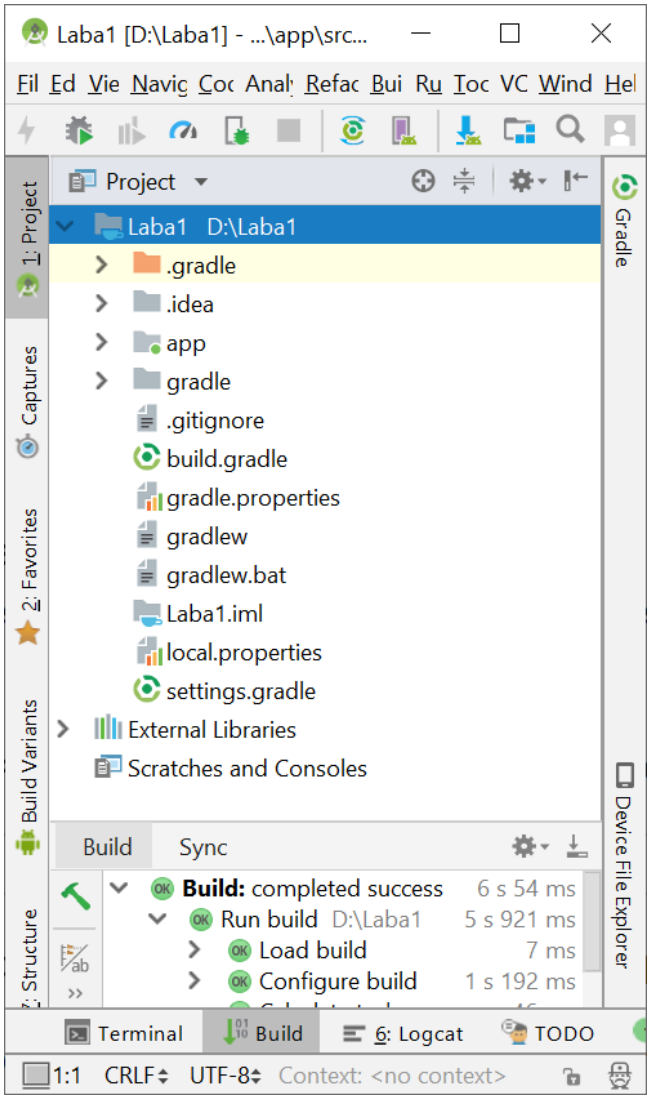


Рис. 1.3. Структура проєкту Android Studio, яка створюється за замовченням

За замовченням проєкт має модуль – app. Модулі описуються у файлі setting.gradle. Конфігурація проєкту, визначену для даного модуля, міститься у файлі build.gradle.

Основні каталоги та файли модуля app:

1) Каталог manifest, який містить файл AndroidManifest.xml. Файл маніфесту зберігає інформацію про назву, версію додатку, реєструє класи, які використовують activity.

2) Каталог java містить вихідні файли додатку, опис класу MainActivity.

3) Каталог res містить папки з ресурсами:

– папка drawable - збереження зображень, які використовуються в додатку;

– папка layout - збереження файлів, що визначають графічний інтерфейс. Файл activity_main.xml визначає інтерфейс для activity – MainActivity;

– папка mipmap містять файли зображень, які призначені для створення іконки програми при різних розширеннях екрану;

– папка values зберігає різні xml-файли, що містять колекції ресурсів - різних даних, які застосовуються в додатку: рядки, числа, кольори.

Графічний інтерфейс додатку визначено у файлі activity_main.xml. файл може відкриватися у двох режимах: режим дизайнера (кнопка «Design» внизу зліва) та текстовому режимі (кнопка «Text»).

Приклад 1. Додаток типу «Hello world!»

Замість стандартного "Hello world!" запишіть значення до властивості android:text елементу TextView "Лабораторна робота №1 Мобільні пристрої!". (рис. 1.4). Графічний дизайнер автоматично оновиться після збереження рядку у текстовому редакторі.

Запуск проєкту може виконуватися на гаджеті або на емуляторі. Для запуску застосовуємо команду «Shift+F10», зелений трикутник (рис. 1.5).

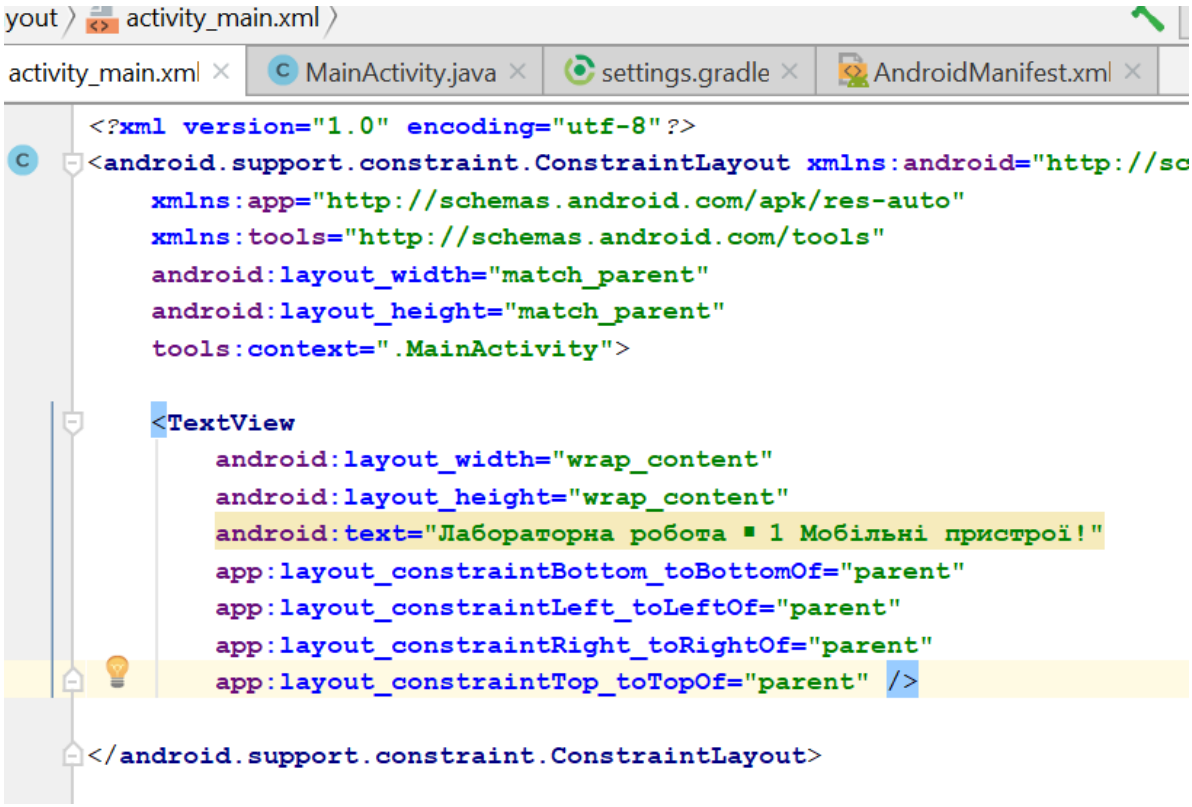


Рис. 1.4. Приклад редагування файлу «activity_main.xml»

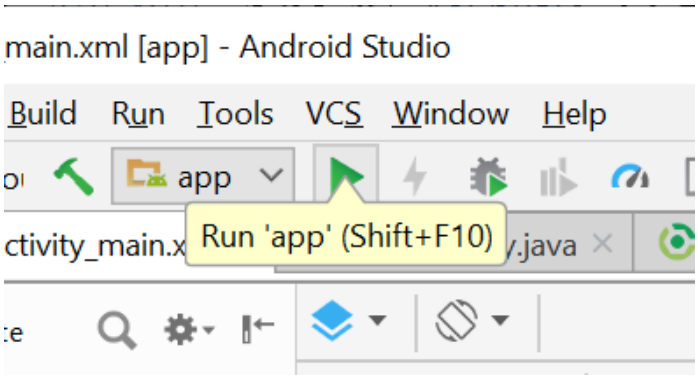


Рис. 1.5. «Запуск проєкту»

Activity - компонент для створення візуального інтерфейсу в Android-додатку. Файл activity_main.xml пов’язаний із графічним інтерфейсом через клас Activity. Код класу генерується автоматично у файлі src/main/java/MainActivity.java (рис. 1.6):

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

import ...

public class MainActivity extends AppCompatActivity {

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

Рис. 1.6. Зміст файлу MainActivity.java

Всі об'єкти activity є об'єктами класу android.app.Activity, який містить базову функціональність для всіх activity.

Клас MainActivity - звичайний клас java. Спочатку йдуть визначення пакету і імпорту зовнішніх пакетів. За замовчуванням він містить тільки один метод onCreate(), в якому фактично і створюється весь інтерфейс програми.

У методі OnCreate() йде звернення до методу батьківського класу і установка ресурсу розмітки дизайну. Щоб встановити ресурс розмітки дизайну, викликається метод setContentView, в який передається ідентифікатор ресурсу.

Ідентифікатор ресурсу: R.layout.activity_main. Фактично це і є посилання на файл activity_main.xml, який знаходиться в каталозі res / layout.

Таким чином, під час запуску програми спочатку запускається клас MainActivity, який в якості графічного інтерфейсу встановлює розмітку з

файлу activity_main.xml. Однак в класі MainActivity ми використовуємо не файли, а ідентифікатори ресурсів: R.layout.activity_main.

Всі ідентифікатори ресурсів визначені в класі R, який автоматично створюється утилітою aapt і знаходиться в файлі R.java в каталозі build/generated/ source/r/debug.

Клас R містить ідентифікатори для всіх ресурсів, розташованих в каталозі res. Для кожного типу ресурсів в класі R створюється внутрішній клас (наприклад, для всіх графічних ресурсів з каталогу res / drawable створюється клас R.drawable) і для кожного ресурсу даного типу присвоюється ідентифікатор. З цього ідентифікатора згодом можна витягти ресурс у файлі коду.

Приклад 2. Обробка натискання кнопки

І) Для додавання текстового створимо елемент EditText (PlainText). Для EditText також треба оголосити певні xml-атрибути:

1) id (ідентифікатор елементу (віджета)- В «графічному режимі» змінюємо значення поля на edit1. В «текстовому режимі» рядок матиме наступний вигляд: android:id="@+id/edit1".

Пояснення: знак (@) застосовується для посилання на об’єкт у файлі XML; слово (id) – вказує на тип ресурсу; знак (/) та ім’я ресурсу. Знак (+) перед типом ресурсу встановлюється, якщо ID ресурсу визначається вперше, в інших випадках даний знак не застосовується. Після призначення id можна звертатися до елементу, використовуючи метод findViewById(R.id.id_value)), де id_value – значення id елементу керування.

Параметри android:layout_width та android:layout_height:

layout_width	wrap_content
layout_height	wrap_content

Значення wrap_content визначає, що ширина та висота буде встановленою достатньою для відображення тексту, введеного до віджету.

3)властивість hint – текст, який буде відображатися за замовчанням в текстовому полі: android:hint="Введіть повідомлення".

II) після EditText Додаємо кнопку:

```
<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Відправити"
  android:onClick="Send"
/>
```

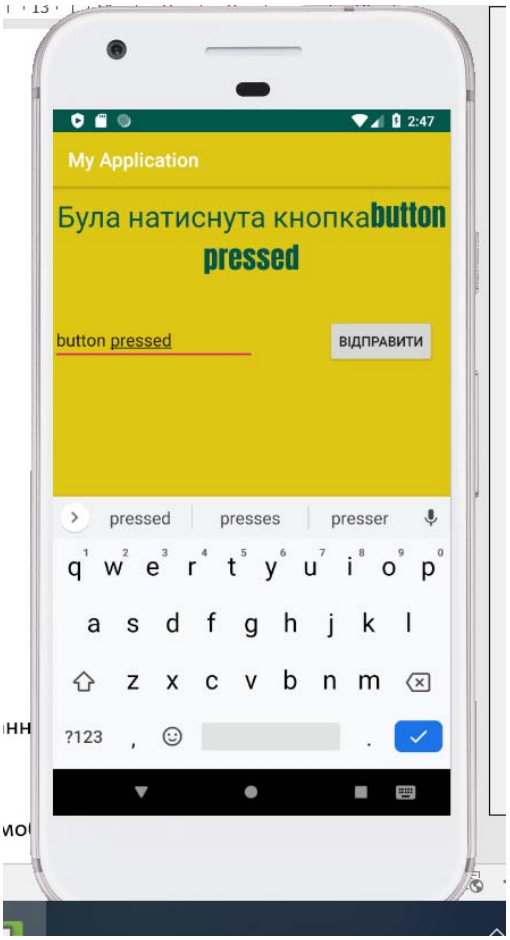


Рис. 1.7. Результат роботи Прикладу 2

Значення атрибуту android:onClick визначає ім’я методу, який визначений в класі activity, та який система викликає при натисканні на кнопку. Для швидкого створення прототипу методу достатньо встановити курсор на рядку android:onClick="Send" в позиції назви методу (в прикладі Send), натиснути Alt+Enter та вибрати команду: “Create Send(‘View’) in ‘MainActivity’ ” . Студія створить шаблон методу в класі MainActivity.

Далі переходимо до файлу MainActivity.java.

Реалізовуємо метод Send в класі MainActivity. Передбачаємо, що текст, який вводиться до «EditText» буде при натисканні на кнопку відображатися у написі «textView»:

```
public void Send(View view)
{
   TextView textView = (TextView)
findViewById(R.id.textView);
   EditText var1 = (EditText) findViewById
(R.id.edit1);
textView.setText("Була натиснута
кнопка"+var1.getText());
}
```

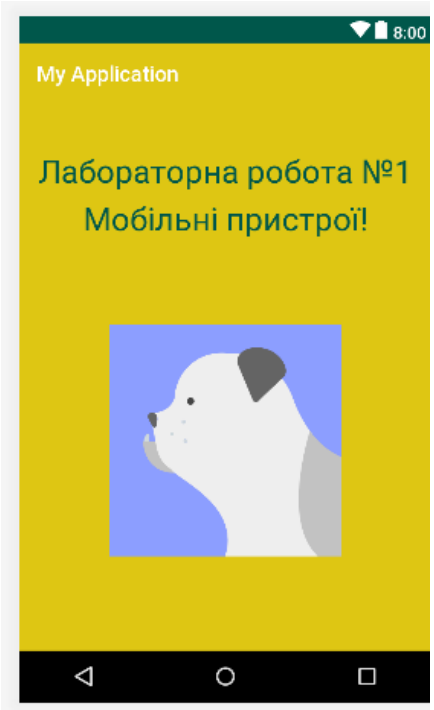


Рис. 1.8. Приклад інтерфейсу

TextView та EditText підкреслюється як помилка. Щоб усунути помилку, необхідно імпортувати в програму типи даних (import android.widget.TextView;). Команда «Alt+Enter». Результат зображено на рис. 1.7.

1.3 Завдання

1.Створити Android-додаток з використанням конструкторів. Змінити властивості виду: фон додатку, надпис, зображення (рис. 1.8).

2. Додати елемент керування «Кнопка» та здійснити обробку події натискання на кнопку (виведення повідомлення в напис «TextView», генерування спливаючої підказки тощо).

1.4 Контрольні запитання

1. Коротка характеристика програмного забезпечення для розробки мобільних додатків.
2. Назвіть поточну версію Android Studio.
3. Файл activity_main.xml (що визначає, які можливості редагування).
4. Поняття activity, зв'язок графічного інтерфейсу з ресурсами.
5. Правило визначення ідентифікатору, пояснити на прикладі ("@+id/button").

Кредит 2. Проектування та реалізація інтерфейсу

Тема 3. Проектування графічного інтерфейсу за допомогою майстра, XML- розмітки

3.1 Основи представлення графічного інтерфейсу

Графічний інтерфейс користувача Android додатку формується на основі об’єктів View та ViewGroup. При чому об’єкти ViewGroup є контейнерами (наприклад, RelativeLayout, LinearLayout, GridLayout, ConstraintLayout), які містять та упорядковуються об’єкти View. Таким чином: клас View є базовим класом для всіх візуальних елементів (елементи керування та віджети), ViewGroup – похідний клас від класу View, містить ядро підкласів, що називаються розмітками (layouts).

Activity – вікна та екрани, які відображаються (аналог форм). Для кожного Activity формується дерево ієрархії вузлів View та ViewGroup (рис. 2.1).

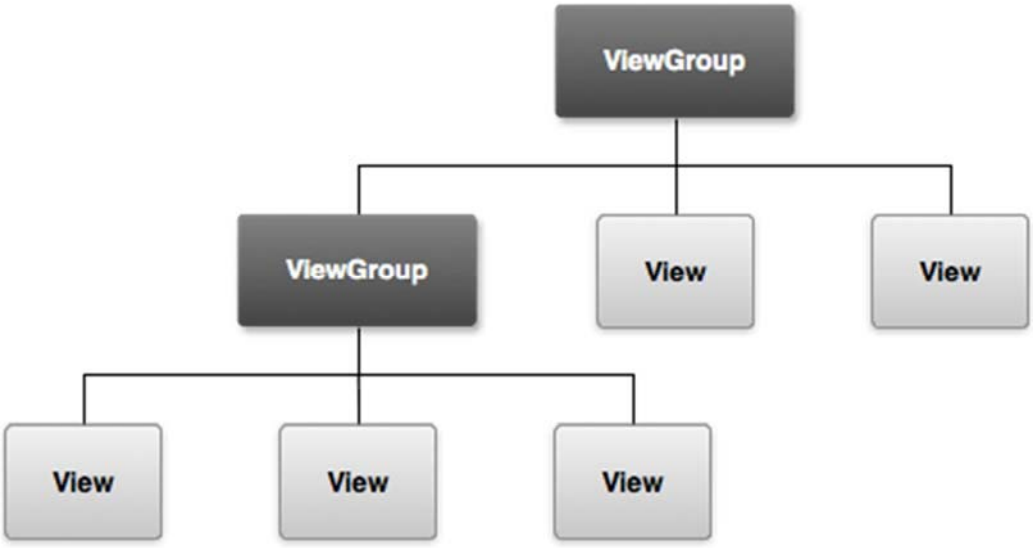


Рис. 2.1. Дерево ієрархії вузлів View та ViewGroup

Під час запуску програми система отримає посилання на кореневий вузол дерева, аналізує елементи дерева ієрархії, додаючи їх до елементів-батьків. Для цього в методі onCreate() викликається метод setContentView(), який в якості параметру приймає ресурс розмітки (розмітка знаходиться в файлі mail.xml).

3.2 Види розмітки

Розмітка (Layout) є розширенням класу ViewGroup і використовується для розміщення дочірніх компонентів на екрані пристрою.

Види Розмітки:

1) ConstraintLayout – вид компоновки, який надає адаптивний та гнучкий спосіб зі створення інтерфейсу додатку. ConstraintLayout, який зараз є видом компоновки за замовченням в Android Studio, дає багато способів розміщення об'єктів. Ви можете розташувати їх відносно головного екрану, відносно один до одного або відносно ліній-вказівників. Це дозволяє створювати великі, складні, динамічні інтерфейси. ConstraintLayout підтримує анімацію.

2) FrameLayout. Найпростіша розмітка, прикріплює кожне нове дочірнє Подання до лівого верхнього кута екрану, накладаючи новий елемент на попередній, затуляючи його. Що б уникнути накладення використовуються відступи. Часто FrameLayout застосовується для створення похідних контейнерів, наприклад, ScrollView, який забезпечує прокрутку. Елементи управління, які поміщаються в FrameLayout, можуть встановити своє позиціонування за допомогою атрибута android: layout_gravity.

3) LinearLayout. Поміщає дочірні Уявлення в горизонтальний або вертикальний ряд. Вертикальна розмітка являє собою колонку, а горизонтальна - рядок з елементами. Напрямок розмітки вказується за допомогою атрибута android: orientation (например, android: orientation = "vertical"). Дана розмітка дозволяє задавати не тільки розміри, але і «відносна вага» дочірніх елементів (передається атрибутом android: layout_weight), завдяки чому можна гнучко контролювати їх розміщення на екрані. Якщо всі елементи мають значення android: layout_weight = "1", то всі ці елементи будуть рівномірно розподілені по всій площі контейнера.

4) RelativeLayout. Найбільш гнучкий серед стандартних видів розмітки. Дозволяє вказувати позиції дочірніх уявлень щодо кордонів вільного простору і інших уявлень. Для позиціонування щодо іншого елемента, вказується id цього елемента.

5) TableLayout. Дозволяє розміщувати дочірні Уявлення всередині осередків «сітки», що складається з рядків і стовпців. Розміри осередків можуть залишатися постійними або автоматично розтягуватися при необхідності. Використовуючи елемент TableRow, створюється окремий рядок. Кількість стовпців визначається максимальною кількістю максимальною кількістю віджетів в рядках одного рівня,

5) Gallery. Представляє елементи у вигляді прокручуваного горизонтального списку (зазвичай графічні елементи).

3.3 Особливості ConstraintLayout

В Android Studio додано два режими: «дизайну» (Design) та «опорних точок» (Blueprint), які відображають макет в режимі дизайну або

сітки. Можна увімкнути будь-який режим або обидва разом відповідно до ваших вимог.

Для розуміння ConstraintLayout необхідно активувати обидва режими (рис. 2.2).

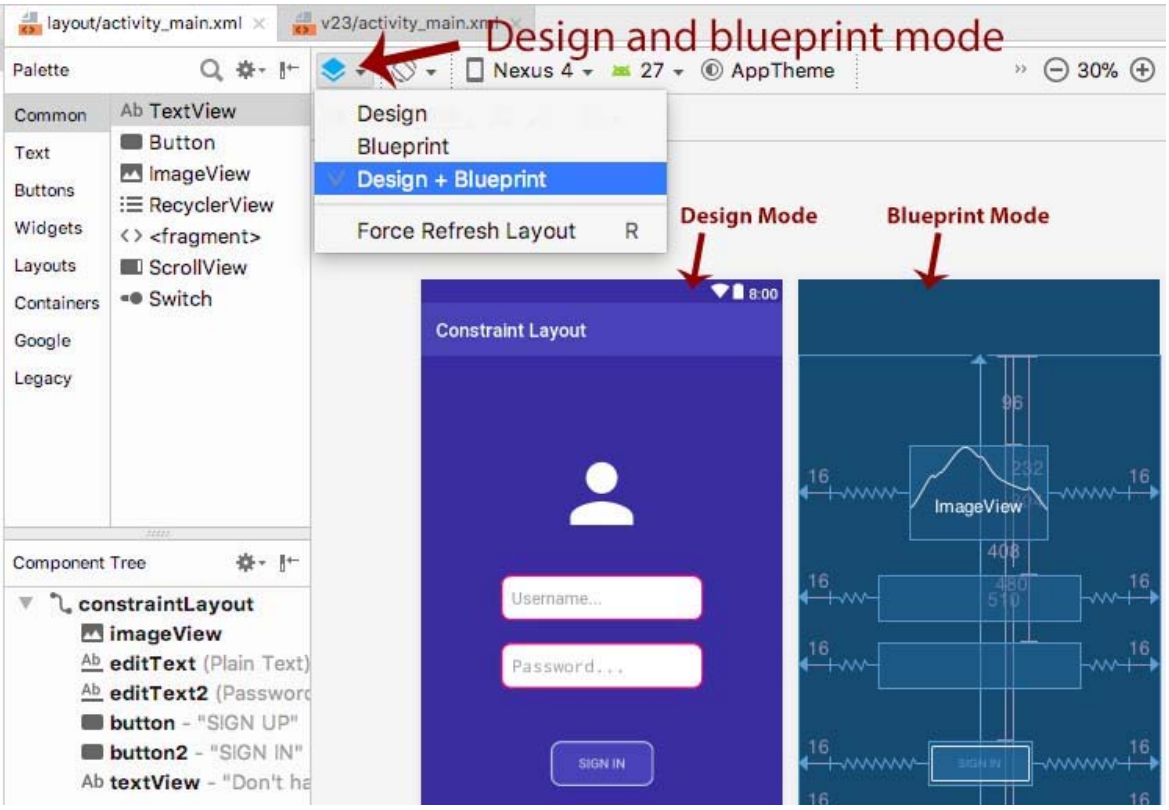


Рис. 2.2. Активація режимів «Дизайн» та «Опорні точки»

«Опорні» точки у макеті розмітки:

Припустимо, ви перетягуєте елемент TextView у візуальному редакторі ConstraintLayout Android Studio. Відразу після перетягування ви помітите повідомлення про помилку з: "Цей View не прив’язаний до розмітки ..." (рис. 2.3). Це означає, що створений View не є прив’язаним до розмітки, і це необхідно виправити. Якщо цю помилку не виправити, елемент керування View не відображатиметься належним чином, коли буде запущено додаток.

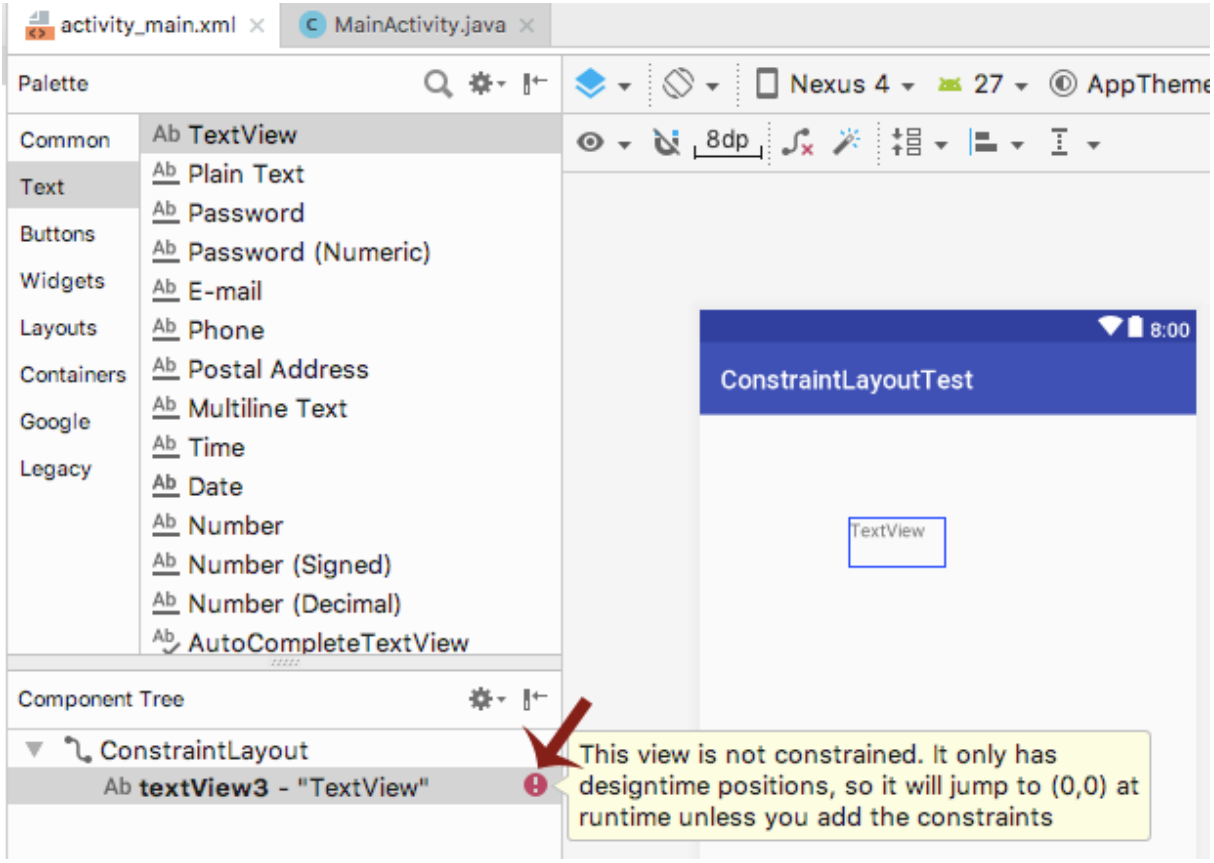


Рис. 2.3. Повідомлення «View не є прив’язаним до розмітки»

Тепер, якщо навести на TextView, можна побачити різні точки, які можна назвати дескрипторами або «опорними» точками в макеті ConstraintLayout (рис. 2.4.).

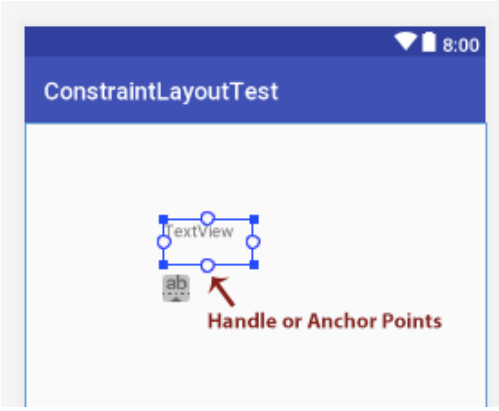


Рис. 2.4. «Опорні» точки в макеті ConstraintLayout

Необхідно натиснути на будь-яку ручку та перетягніть її, щоб встановити зв’язок з чимось іншим навколо неї.

Необхідно з'єднати принаймні дві точки з іншими елементами, щоб зробити TextView прив’язаним.

Щоб змінити розмір View, можна використати ручку зміни розміру, яка знаходиться в кутах (рис. 2.5).

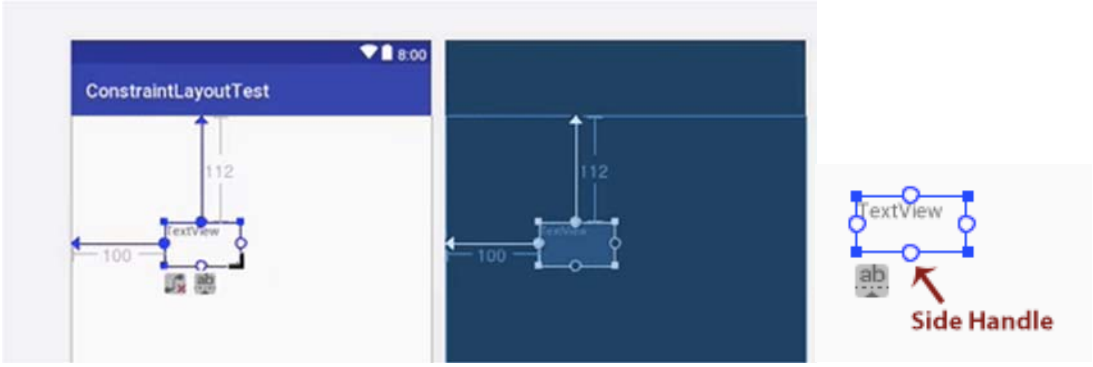


Рис. 2.5. Маркер, який застосовується для зміни розміру View

Бічний маркер (Side Handle) - це кругла точка, яка використовується для встановлення верхньо-лівого та нижньо-правого обмежень.

Праворуч розміщується вікно атрибутів, яке надає багато подробиць про подання, які ми використовували для перегляду в ConstraintLayout.

Також можна використовувати такі інструменти, як Autoconnect, щоб дозволити/заборонити Android Studio зробити автоматичну прив’язку елемента (Autoconnected constrained), очистити всі обмеження, видалити всі обмеження за один раз (clear all constraints) та зробити автоматичну прив’язку для усіх елементів на екрані (infer constraints) (рис. 2.6.).

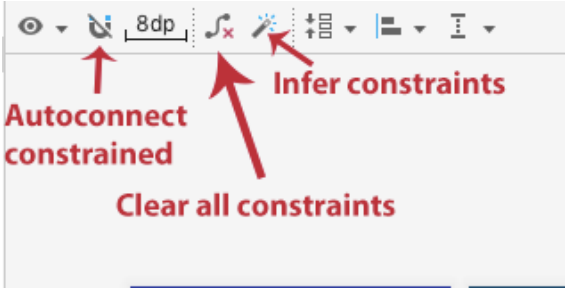


Рис. 2.6. Автоматичне налаштування ConstraintLayout

Лабораторна робота №2. Основи розробки інтерфейсу. Створення та виклик Activity.

2.1 Запуск нової Activity.

2.2 Завдання.

2.3 Контрольні запитання.

2.1 Запуск нової Activity

Розглянемо приклад проекту, в якому запуск нової Activity здійснюється під час натискання на кнопку. При цьому значення текстового поля з основної активності передається до віджету TextView нової активності.

В основній Activity (файл activity_main.xml) додаємо: текстове поле (елемент EditText, id - edit_message) кнопку (елемент Button). Для кнопки значення атрибуту android:onClick встановлюємо ім’я методу обробки натискання на кнопку “sendMessage”.

В класі MainActivity вносимо наступні зміни (рис. 2.7):

1) В класі MainActivity оголошуємо статичну змінну-рядок, яка вказує на ключ даних, що будуть передаватися.

2) Додаємо метод sendMessage () обробки натискання на кнопку.

3) Створюємо об'єкт Intent для виклику нової Activity.

4) Знаходимо текстове поле edit_message в поточній Activity.

5) Зчитуємо вміст текстового поля.

6) Для об'єкту intent викликаємо метод putExtra, передаючи два параметри: перший параметр - ключ, другий - значення даного об'єкту.

7) Виклик методу startActivity() для запуску activity.

```
public class MainActivity extends AppCompatActivity {
    public final static String EXTRA_MESSAGE = "laba2";// змінна-рядок,
    // яка вказує на ключ даних, що будуть передаватися
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // Метод обробки натискання на кнопку
    public void sendMessage(View view) {
        // Створюємо об'єкт Intent для виклику нової Activity
        Intent intent = new Intent(this, Main2Activity.class);
        // Знаходимо  текстове поле edit_message в поточній Activity
        EditText editText = (EditText) findViewById(R.id.edit_message);
        // Зчитуємо вміст текстового поля
        String message = editText.getText().toString();
        // для об'єкту intent викликаємо метод putExtra, передаючи два
параметри: перший параметр - ключ,
        // другий - значення  даного об'єкту
        intent.putExtra(EXTRA_MESSAGE, message);
        // Виклик методу startActivity() для запуску activity
        startActivity(intent);
    }
}
```

Рис. 2.7. Зміст класу MainActivity

Для створення нової активності Main2Activity необхідно для каталогу, в якому знаходиться клас MainActivity (app/java/[назва_пакету]), викликати контекстне меню (права кнопка миші) та обрати команду New->Activity->Empty Activity (рис. 2.8):

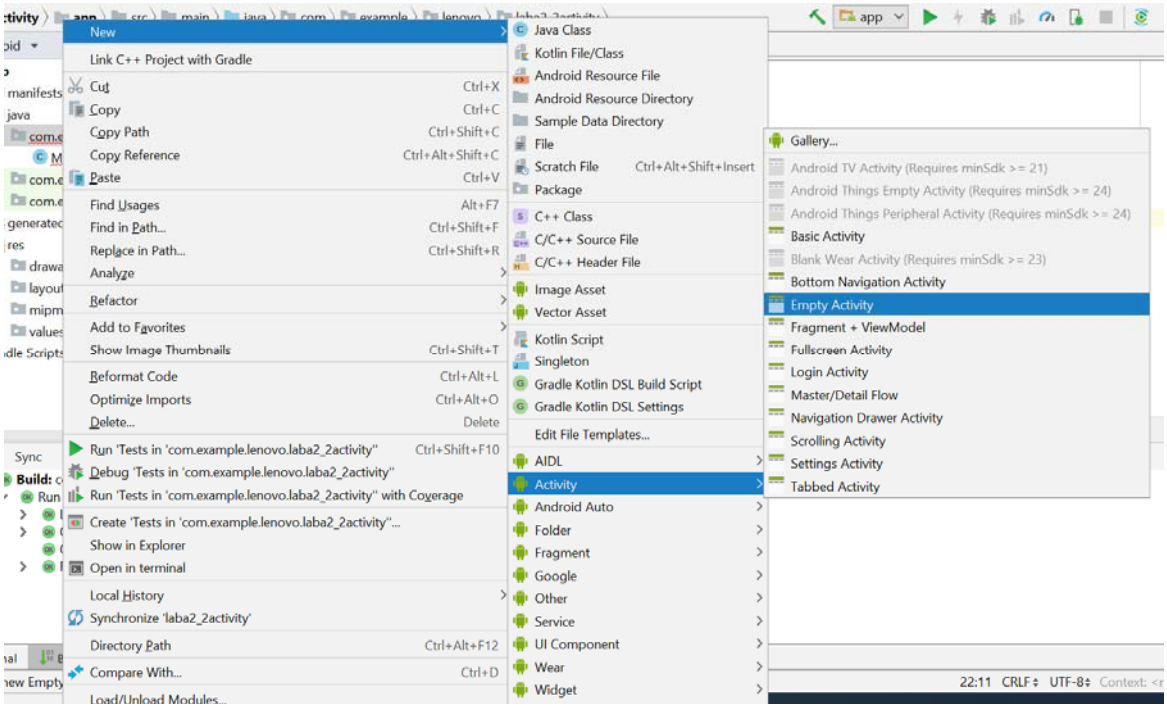


Рис. 2.8. Створення нової активності

Задаємо ім’я активності Main2Activity, останні параметри залишаємо за замовчанням. В каталог */layout додається файл інтерфейсу activity_main2.xml, а до каталогу /java/[назва_пакету] клас* Main2Activity (рис. 2.9).

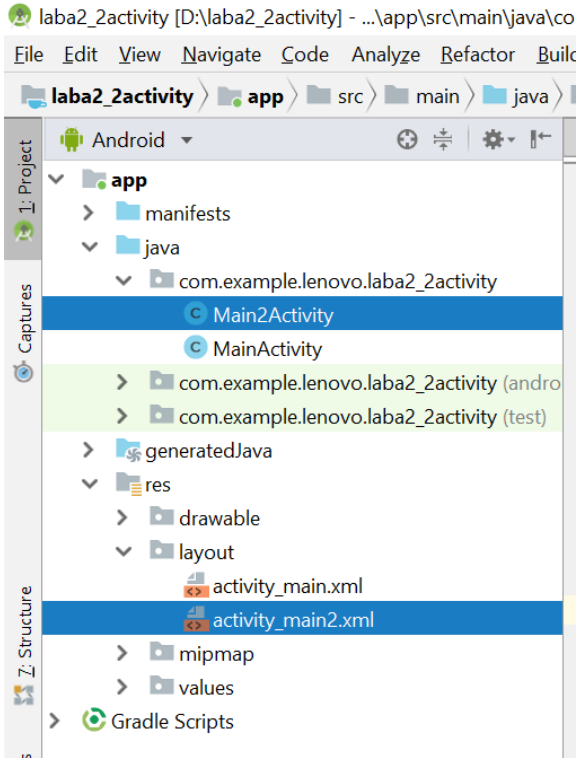


Рис. 2.9. Розміщення файлів інтерфейсу та класу активності

До класу Main2Activity вносимо наступні зміни (рис. 2.10):

1) Отримуємо повідомлення (message) від об'єкту intent через поле EXTRA_MESSAGE класу MainActivity.

2) Створюємо текстове поле та записуємо до нього значення message.

3) Встановлюємо текстове поле в системі компоновки activity.

```
public class Main2Activity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // отримуємо повідомлення від об'єкту intent
        Intent intent = getIntent();
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

        // створюємо текстове поле та записуємо до нього значення message
        TextView textView = new TextView(this);
        textView.setTextSize(40);
        textView.setText(message);

        // Встановлюємо тектове поле в системі компоновки activity
        setContentView(textView);
    }
}
```

Рис. 2.10. Зміст файлу Main2Activity.java

Результат роботи представлено на рис. 2.11.

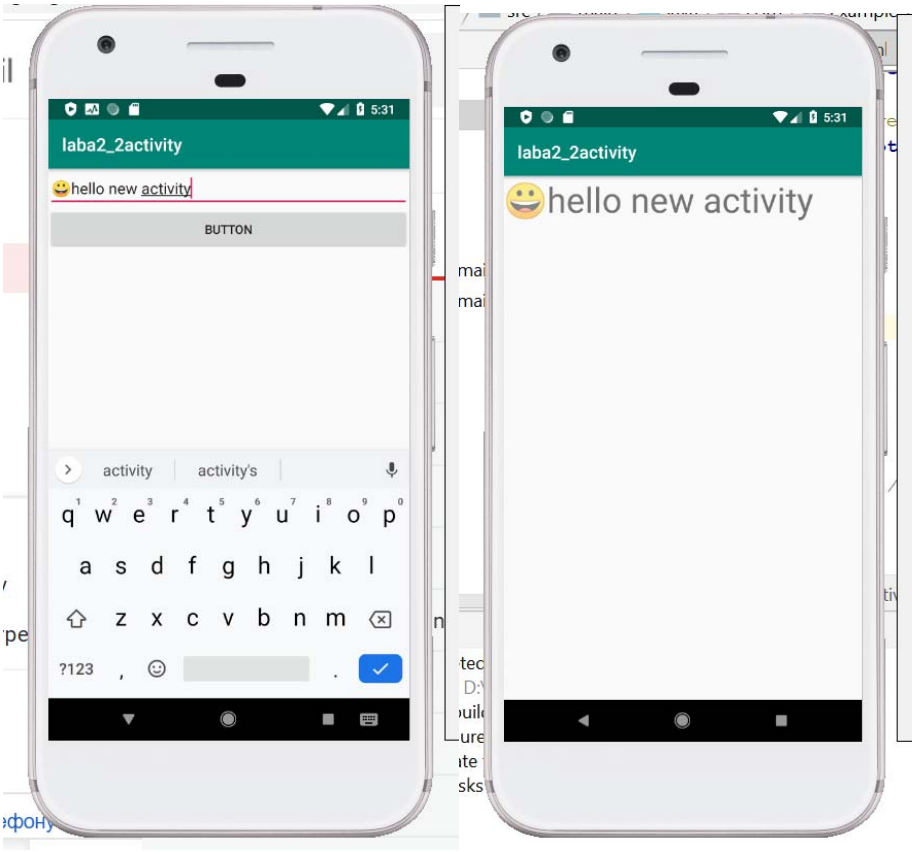


Рис. 2.11. Результат роботи прикладу із створення нової активності

2.2 Завдання

Рис. 2.12. Вигляд екрану до завдання 2

1. RelativeLayout. Реалізуйте зміст файлу розмітки activity_main.xml наступним чином (рис. 2.12.) (режим роботи: «графічний» або «текстовий» обирайте на власний розсуд). Реалізуйте наступні варіанти вирівнювання: текстове поле – по нижньому або верхньому краю, кнопку – по центру або лівому краю.

2. Створити мобільний додаток, який складається з чотирьох activity. Після запуску програми користувач повинен потрапляти на екран з activity1. На цьому екрані має бути представлено меню, що

складається з чотирьох кнопок. Висота кнопок повинна складати 20% від висоти екрана. Відстань між кнопками - 2%. Перша і остання кнопка повинні бути на рівній відстані від країв екрану. Ширина кнопок 75%, вирівнювання посередині.

Після натискання на першу кнопку користувач повинен переходити до activity2, його зовнішній вигляд представлений на рисунку 2.13.

Верстка повинна здійснюватися з використанням LinearLayout, ширина кнопок повинна задаватися в відсотках від ширини екрану.

2.3 Контрольні запитання

1. На основі яких об’єктів формується графічний інтерфейс користувача Android додатку.
2. Дайте пояснення Activity.
3. Поняття компоновки (*Layout). Види Layout.*

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

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

Рис. 2.13. Зміст файлу розмітки *activity_main.xml*

Тема 4. Робота з елементами керування. Атрибути віджетів.

4.1 Властивості EditText

EditText - стандартний віджет для введення даних в Android-додатках. EditText - це підклас TextView з операціями редагування тексту. EditText - це просто розширення TextView. EditText успадковує всі властивості TextView. EditText застосовується у програмах, щоб забезпечити текстове поле для введення, особливо у формах. Найпростіший приклад застосування EditText - це форма входу (Login) або реєстрації (Sign-in).

Види текстових полів в Android Studio, які мають тип EditText представлені на рис. 2.14.

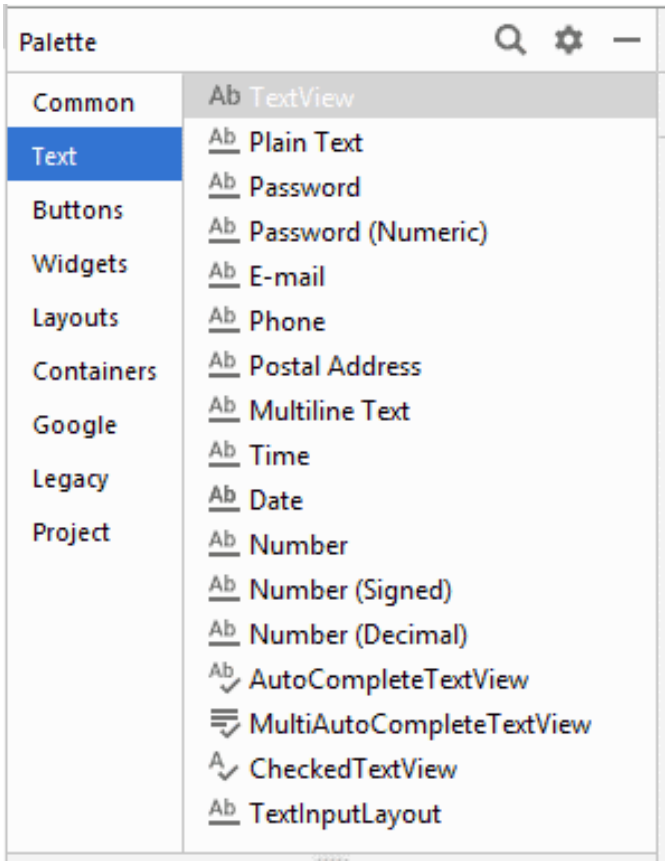


Рис. 2.14. Види текстових полів в Android Studio, які мають тип EditText

Приклади створення EditText в XML та Java:

1) На XML:

```
<EditText
android:id="@+id/simpleEditText"
android:layout_height="wrap_content"
android:layout_width="match_parent"/>
```

2) Отримання значення поля EditText на Java:

```
EditText simpleEditText = (EditText) findViewById(R.id.simpleEditText);
String editTextValue = simpleEditText.getText().toString();
```

Властивості EditText:

1) id: унікальний ідентифікатор.

2) gravity: необов'язковий атрибут, який використовується для контролю вирівнювання тексту, наприклад, зліва, справа, центру, вгорі, внизу, вертикальне вирівнювання, горизонтальне вирівнювання тощо.

3) text: запис текстових даних до EditText.

Приклад «програмного» запису тексту на Java:

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);
editText.setText("Username");//запис тексту
```

4) hint: атрибут для встановлення підказки, тобто те, що ви хочете, щоб користувач ввів у цей текст редагування. Щоразу, коли користувач починає вводити текст редагування, підказка автоматично зникає.

Встановлення hint в EditText на Java:

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);
editText.setHint("Enter Your Name Here");//відображення hint
```

5) textColor: використовується для встановлення кольору тексту. Значення кольору має форму "#argb", "#rgb", "#rrggbb" або "#aarrggbb".

Задання значення textColor для EditText на Java:

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextColor(Color.RED);//текст червоного кольору
```

6) textColorHint**:** встановлення кольору підказки, яка відображується.

Задання значення textColorHint для EditText на Java:

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setHintTextColor(Color.green(0));//hint – зеленого кольору
```

7) textSize: встановлення розміру тексту, який задається в sp(scale independent pixel) або dp(density pixel).

Встановлення textSize для EditText на Java:

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextSize(25);//задання розміру тексту
```

8) textStyle: встановлення стилю для тексту: жирний, курсив та звичайний. Якщо потрібно використовувати два або більше стилів для редагування тексту, тоді використовується оператор "|".

9) background: використовується для встановлення фону тексту редагування. Можна встановити колір або малюнок на задньому плані тексту редагування. В Java застосовується метод setBackgroundColor().

10) padding: атрибут padding використовується для встановлення відступів зліва, справа, зверху або знизу.

Нижче наведено приклад коду, в якому встановлюється чорний колір для фону, білий колір для відображеної підказки та встановлюємо відступ 15dp від усіх бічних сторін текстового поля.

```
<EditText android:id="@+id/simpleEditText"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:hint="Enter Your Name Here"
android:padding="15dp"
android:textColorHint="#fff"
android:textStyle="bold|italic"
android:background="#000"/>
```

4.2 Властивості CheckBox

В Android, CheckBox - це тип кнопки з двома станами: невідмічений або відмічений. Перехід між станами може здійснюватися користувачем. CheckBox застосовується, коли необхідно представити групу варіантів вибору. CompoundButton - це батьківський клас класу CheckBox.

Атрибути CheckBox, які допомагають налаштувати властивості у XML-файлі.

1) id: id - атрибут, який використовується для однозначної ідентифікації прапорця.

2) checked: Позначено - атрибут опції, який використовується для встановлення поточного стану прапорця. Значення повинно бути істинним або хибним. Значення за замовчуванням – false, тобто прапорець не встановлений.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"
    android:checked="true"/> <!—Встановити поточний стан-->
```

Поточний стан можна встановити програмно в Java класі:

```
/*Додати до OnCreate() методу після setContentView()*/
// ініціалізація змінної типу CheckBox
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
// встановлення поточного стану прапорця
simpleCheckBox.setChecked(true);
```

3) gravity: атрибут gravity - це необов'язковий атрибут, який використовується для контролю вирівнювання тексту в CheckBox, наприклад, зліва, справа, центру, вгорі, внизу, вертикальне вирівнювання, горизонтальне вирівнювання тощо.

Приклад використання:

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"
    android:checked="true"
    android:gravity="right|center_vertical"/>  <!—вирівнювання тексту-->
```

4) text: текстовий атрибут використовується для встановлення тексту Встановити текст можна як у xml, так і у класі java.

Приклад xml:

```
android:text="Text Of Checkbox"/> <!—текст, який відображається-->
```

Приклад у класі java:

```
/*Додати до OnCreate() методу після setContentView()*/
// ініціалізація змінної типу CheckBox
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
// відображення тексту CheckBox
simpleCheckBox.setText("Text Attribute Of Check Box");
```

5) textColor: використовується для встановлення кольору тексту у прапорці. Значення кольору у формі "#argb", "#rgb", "#rrggbb" або "#aarrggbb".

```
android:textColor="#f00" <!—червоний колір тексту прапорця-->
```

На Java:

```
…
simpleCheckBox.setTextColor(Color.RED);
```

6) textSize: аналогічно відповідній властивості в EditText.

На Java метод `setTextSize()`

7) textStyle: аналогічно відповідній властивості в EditText.

8) background: аналогічно відповідній властивості в EditText.

9) padding: використовується для встановлення відступів зліва, справа, зверху або знизу використовується для встановлення відступів зліва, справа, зверху або знизу:

- paddingRight: відступ з правого боку прапорця.
- paddingLeft: відступ зліва від прапорця.
- paddingTop: відступ з верхньої сторони прапорця.
- paddingBottom: відступ з нижньої сторони прапорця.
- padding: відступ із усіх боків прапорця.

4.3 RadioButton

В Android RadioButton в основному використовуються разом у RadioGroup. В RadioGroup можливій вибір тільки одного перемикача з декількох. Це означає, одночасно можна перевірити лише один перемикач із групи перемикачів.

Приклад створення RadioGroup та RadioButton на XML:

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <RadioButton
        android:id="@+id/simpleRadioButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
        <RadioButton
        android:id="@+id/simpleRadioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</RadioGroup>
```

Можна програмно перевірити поточний стан перемикача за допомогою методу isChecked (). Цей метод повертає булеве значення true або false.

```
/*додаємо в функції Oncreate() після setContentView()*/
// ініціалізація RadioButton
RadioButton simpleRadioButton =
(RadioButton) findViewById(R.id.simpleRadioButton);
// перевірка поточного стану
Boolean RadioButtonState = simpleRadioButton.isChecked();
```

Атрибути RadioButton в Android:

1) id: - це атрибут, який використовується для однозначної ідентифікації перемикача;

2) checked: використовується для встановлення поточного стану перемикача. Значення даного атрибута за замовчуванням false. Поточний стан значення даного атрибуту встановлюється на JAVA з використанням методу setChecked(**true**).

3) text, gravity, textColor, textSize, textStyle, background, padding відповідають аналогічним властивостям в EditText та CheckBox.

4) drawableBottom, drawableTop, drawableLeft та drawableRight: властивості, які дозволяють встановити малюнок на задньому фоні тексту.

Приклад встановлення піктограми праворуч від тексту RadioButton:

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:padding="20dp"
    android:layout_centerHorizontal="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:drawableRight="@drawable/ic_launcher" />
```

4.4 Робота з ImageView, Switch (On/Off), ProgressBar

В Android клас ImageView використовується для відображення файлу зображення в програмі. ImageView простий у використанні, але його важко освоїти в Android через різні розміри екрану на пристроях Android.

ImageView постачається з різними параметрами конфігурації для підтримки різних типів масштабу. Параметри типу масштабу використовуються для масштабування меж зображення до меж перегляду зображення . Основними властивостями параметрів масштабування (scaleTypes) є center, center_crop, fit_xy, fitStart тощо.

Рис. 2.15. Вигляд екрану з ImageView

Нижче наведено код ImageView у форматі XML: Зображення необхідно зберегти в папці drawable. Результат на рисунку 2.15.

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="214dp"
    android:layout_height="0dp"
    android:layout_marginEnd="71dp"
    android:layout_marginRight="71dp"
    android:layout_marginBottom="10dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:srcCompat="@drawable/image" />
```

Атрибути ImageView:

1) id: id - це атрибут, який використовується для однозначної ідентифікації подання зображення в android. Нижче наведено приклад коду, в якому ми встановлюємо ідентифікатор подання зображення .

2) src: src - це атрибут, який використовується для встановлення вихідного файлу, або ви можете вимовити зображення у своєму View, щоб зробити ваш макет привабливим.

Нижче наведено приклад коду, в якому ми встановлюємо джерело лева перегляду зображень, яке зберігається у папці, що малюється.

Можна також встановити зображення програмно в класі Java . Для цього використовується метод setImageResource ():

```
/*Add in Oncreate() funtion after setContentView()*/
ImageButton simpleImageButton = (ImageButton)findViewById(R.id.simpleImageButton);
simpleImageButton.setImageResource(R.drawable.home); //set the image programmatically
```

3) **background**: атрибут background використовується для встановлення фону ImageView. Можна встановити колір або малюнок на задньому плані ImageView. Нижче наведено приклад коду, в якому ми встановлюємо чорний колір на задньому плані та зображення в атрибуті src для ImageView .

```
/*Add in Oncreate() funtion after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setBackgroundColor(Color.BLACK);//set black color in
//background of a image view in java class
```

4) відступ: атрибут padding використовується для встановлення відступу ліворуч, праворуч, зверху або знизу ImageView.

– **paddingRight:** встановіть відступ з правого боку зображення **.**

– **paddingLeft:** встановіть відступ з лівого боку зображення **.**

– **paddingTop:** встановіть відступ з верхньої сторони зображення **.**

– **paddingBottom:** встановіть відступ з нижньої сторони зображення **.**

– **padding:** встановіть відступ з усіх боків подання зображення **.**

5) scaleType: scaleType - це атрибут, який використовується для керування тим, як зображення має бути змінено за розміром або переміщено відповідно до розміру цього подання ImageView. Значенням атрибута типу масштабу може бути fit_xy, center_crop, fitStart тощо.

Нижче наведено приклад коду типу масштабу, в якому ми встановлюємо тип масштабу перегляду зображення на fit_xy.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion"
android:scaleType="fitXY"/><!--set scale type fit xy-->
```

В Android **Switch** це перемикач віджета, який дозволяє вибирати між двома варіантами. Застосовується для відображення перевіреного та неперевіреного стану кнопки, що забезпечує користувачеві повзунковий елемент керування. Switch є підкласом CompoundButton. В основному це кнопка вимкнення/увімкнення, яка вказує поточний стан Switch. Перемикач застосовується при виборі ввімкнення / вимкнення у звуці, Bluetooth, WiFi тощо (рис. 2.16).

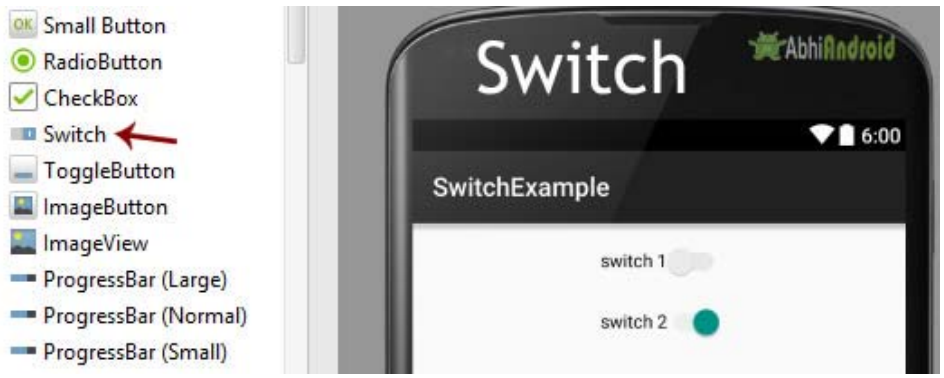


Рис. 2.16. Вигляд екрану з Switch

У порівнянні з ToggleButton Switch забезпечує управління користувачем за допомогою повзунка. Користувач може просто натиснути на перемикач, щоб змінити його поточний стан.

Перемикач дозволяє користувачам змінювати налаштування між двома станами, такими як увімкнення / вимкнення Wi-Fi, Bluetooth тощо з меню налаштувань вашого телефону. Він був представлений після версії Android 4.0 (API 14).

Можна перевірити поточний стан комутатора програмно, використовуючи метод isChecked (). Цей метод повертає булеве значення true (якщо перемикач встановлений) або false.

Нижче наведено приклад коду, в якому ми перевірили поточний стан комутатора.

```
// initiate a Switch
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);
// check current state of a Switch (true or false).
Boolean switchState = simpleSwitch.isChecked();
```

Атрибути перемикача:

1. id: id - це атрибут, який використовується для однозначної ідентифікації комутатора.

2. checked: перевірений атрибут Switch використовується для встановлення поточного стану комутатора. Значення може бути як true, так і false, коли true показує перевірений стан, а false - неперевірений стан комутатора. Значення за замовчуванням перевіреного атрибута - false. Ми також можемо встановити поточний стан програмно.

```
<Switch
android:id="@+id/simpleSwitch"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"/> <!-- set the current state of the Switch-->
```

3. text: текстовий атрибут використовується для встановлення тексту в комутаторі. Ми можемо встановити текст у xml , а також у класі Java .

```
/*Add in Oncreate() funtion after setContentView()*/
// initiate Switch
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);
//displayed text of the Switch
simpleSwitch.setText("switch");
```

4. gravity: атрибут гравітації є необов'язковим атрибутом, який використовується для контролю вирівнювання тексту в Switch. Можна встановити текст вліво, вправо, по центру, вгорі, знизу, по центру по вертикалі, по центру по горизонталі тощо в перемикачі.

5. textOn і testOff: атрибут textOn використовується для встановлення тексту, коли Switch перебуває в перевіреному стані (тобто в стані увімкнення). Можна встановити textOn в XML, а також у класі Java.

У наведеному нижче прикладі встановлюємо textOn як "Так", а textOff як "Ні" для комутатора.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="Notification"
    android:textOff="No"
    android:textOn="Yes"/>
```

6. textColor: атрибут textColor використовується для встановлення кольору тексту комутатора. Значення кольору має форму "#argb", "#rgb", "#rrggbb" або "#aarrggbb".

В Android **ProgressBar** використовується для відображення стану виконуваної роботи, наприклад аналізу стану роботи або завантаження файлу. В Android за замовчуванням індикатор прогресу відображатиметься у вигляді спінінга, але якщо ми хочемо, щоб він відображався як турнік, тоді нам потрібно використовувати атрибут style як горизонтальний. В основному він використовує клас "android.widget.ProgressBar" (рис. 2.17).

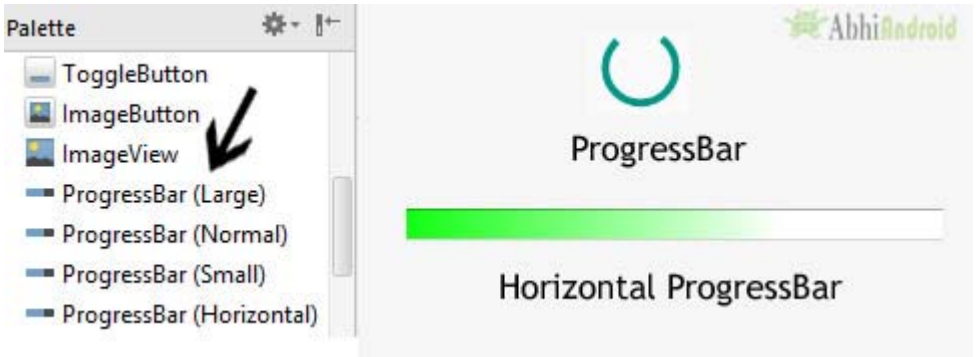


Рис. 2.17. Вигляд екрану з ProgressBar

Індикатор прогресу також можна зробити невизначеним. У цьому режимі на індикаторі прогресу відображається циклічна анімація без

вказівки на прогрес. Цей режим використовується в додатку, коли ми не знаємо обсягу роботи, яку потрібно виконати.

ProgressBar код:

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

Код горизонтального ProgressBar:

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

Важливі методи, які використовуються для панелі виконання:

1. getMax () - повертає максимальне значення індикатора прогресу

2. getProgress () - повертає поточне значення

Атрибути ProgressBar в Android:

1. id: id - це атрибут, який використовується для однозначної ідентифікації індикатора прогресу.

2. max: max - це атрибут, який використовується в android для визначення максимального значення прогресу, який може прийняти. Це має бути ціле значення, наприклад 100, 200 тощо.

Встановити максимальне значення ProgressBar у класі Java:

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar);
// initiate the progress bar
int progressValue=simpleProgressBar.getProgress();
// get progress value from the progress bar
```

4. progressDrawable: атрибут, який використовується в Android для встановлення власного зображення для режиму progress.

Нижче ми встановлюємо користувацький градієнт, який можна малювати для режиму виконання індикатора прогресу. Перш ніж спробувати код нижче, завантажте піктограму прогресу з Інтернету та додайте до папки Drawable.

Крок 1: Додайте цей код у Activity_main. xml або main.xml всередині макета.

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:max="100"
android:progress="60"
android:layout_marginTop="100dp"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
android:progressDrawable="@drawable/custom_progress"/>
<!--custom progress drawable for progress mode-->
```

Крок 2: Створіть новий ресурс XML у папці drawable та назвіть його custom_progress. Додайте до файлу custom_progress.xml код, який створює ефект градієнта на панелі прогресу. Результат відображення представлено на рис. 2.18.

```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item>
        <shape>
            <gradient
                android:endColor="#fff"
                android:startColor="#1f1"
                android:useLevel="true" />
        </shape>
    </item>
</layer-list>
```

Рис. 2.18. Вигляд екрану з ProgressBar

5. background: використовується для встановлення фону панелі виконання. Можна встановити колір або малюнок на задньому плані індикатора виконання.

6. indeterminate: використовується в Android для ввімкнення режиму невизначеності. У цьому режимі на індикаторі прогресу відображається циклічна анімація без вказівки на прогрес. Цей режим використовується в додатку, коли ми не знаємо обсягу роботи, яку потрібно виконати. У цьому режимі фактична робота не відображатиметься.

Лабораторна робота №3

Робота з елементами керування: EditText, CheckBox, RadioButton

3.1 Обробка натискання кнопки з використанням setOnClickListener.

3.2 Вспливаючі підказки (Toast).

3.3 Завдання.

3.4 Контрольні запитання.

3.1 Обробка натискання кнопки з використанням setOnClickListener

Крім стандартного способу обробки натискання клавіші, який пояснявся в лабораторній роботі №1 (через встановлення властивості OnClick) існує інший спосіб з використанням методу setOnClickListener().

Пояснення роботи методу setOnClickListener() на прикладі:

1) Створимо в класі MainActivity змінну типу Button (оголошення виконуємо до методу onCreate()):

```
private Button mCounterButton;
```

2) Після рядку

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

додаємо рядок:

```
mCounterButton= findViewById(R.id.CounterButton);
```

3) Створюємо додаткову змінну, в якій буде зберігатися кількість натискань кнопки. Для цього оголошуємо її після mCounterButton:

```
private int Count = 0;
```

4) Далі додаємо обробник натискання кнопки в методі onCreate() після усього коду, який визначений в ньому. Починаємо вводимо перші символи слова mCounterButton (можна маленькими буквами). Натискаємо Enter, якщо з'явилася підказка. Після цього слова ставимо крапку і вводимо перші літери слова setOnClickListener. На даний момент з’явиться рядок mCounterButton.setOnClickListener(). Ставимо курсор всередині круглих дужок і набираємо new OnClickListener. Тут важливо

набрати символ «О» у верхньому регістрі. Тоді у вас з'явиться потрібна підказка типу View.OnClickListener {...} (android.view.View). Натискаємо Enter і отримуємо потрібну заготовку, всередині якої вставляємо код.

```
mCounterButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        TextView textView = (TextView)
findViewById(R.id.textView);
        textView.setText("Я нарахував" + ++Count
+"натискань");
    }
});
```

3.2 Вспливаючі підказки (Toast)

Toast використовується для відображення інформації протягом певного періоду. Toast містить повідомлення для швидкого відображення, яке зникає через певний проміжок часу. Toast не блокує взаємодію користувача. Toast підкласом класу Object. Повідомлення Toast в Android завжди відображається внизу екрана. Можна створити custom Toast, використовуючи custom layout (файл xml).

Методи Toast:

1) makeText (Context context, CharSequence text, int duration). Цей метод використовується для ініціювання повідомлення, приймає три параметри. Перший призначений для контексту програми, другий - текстове повідомлення, а останній - тривалості відображення.

Константи Toast, які використовуються для встановлення тривалості відображення:

1) LENGTH_LONG: використовується для відображення Toast протягом тривалого періоду часу. Коли встановлюється даний параметр, повідомлення відображатиметься довго.

2) LENGTH_SHORT: Використовується для відображення Toast протягом короткого періоду часу. Коли встановлюється даний параметр, повідомлення відображатиметься короткочасно.

Приклад:

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast",
Toast.LENGTH_LONG);
```

2) show () використовується для відображення Toast на екрані. Відображає текст, який створюється за допомогою методу makeText () Toast.

3) setGravity(int,int,int): використовується для встановлення вирівнювання Toast. Цей метод приймає три параметри: константа напряму вирівнювання, зміщення положення за x та за y.

Приклад встановлення верхнього та лівого вирівнювання.

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```

4) setText (CharSequence s): встановлення значення тексту для Toast. Якщо використовувався метод makeText(), а потім необхідно змінити текст для Toast, тоді необхідно використовувати цей метод.

5) setDuration (int duration): встановлення тривалості Toast.

```
toast.setDuration(Toast.LENGTH_SHORT);
```

6) inflate(int, ViewGroup): Цей метод використовується для встановлення макета з xml. У цьому методі перший параметр є ідентифікатором ресурсу макета, а другий – кореневий елемент View.

7) setView(View): встановлення зовнішнього виду Toast. У цьому методі передається макет, який визначається за допомогою методу inflate(). Нижче ми спочатку отримуємо inflater макета, потім inflate макет і нарешті створюємо новий Toast і передаємо макет у методі setView ().

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast_layout,
(ViewGroup) findViewById(R.id.toast_layout_root));
Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

3.3 Завдання

1) Робота з CheckBox. Створити проект з наступним інтерфейсом (рис. 2.19), виконати перевірку стану CheckBox-ів.

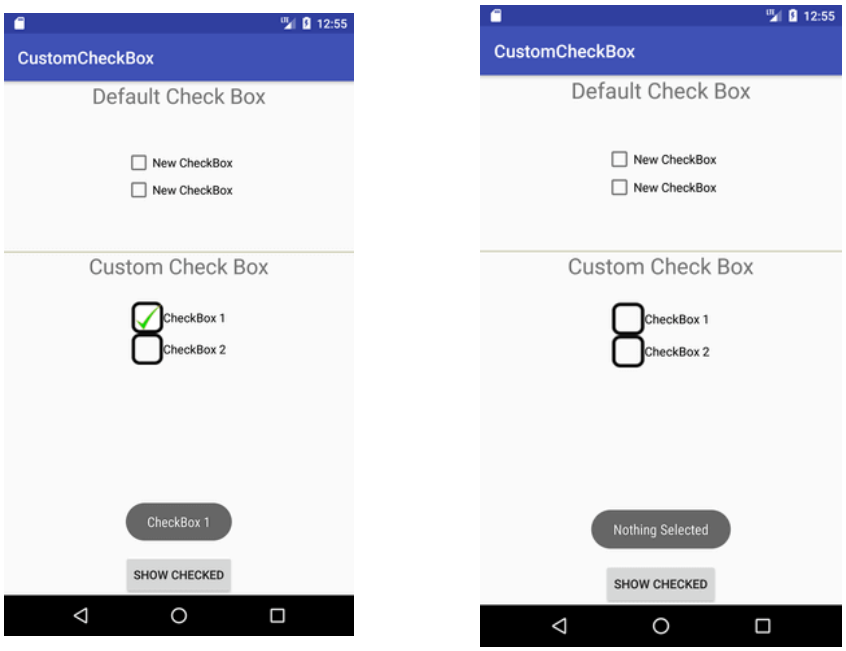


Рис. 2.19. Вид інтерфейсу для роботи з CheckBox

2) Робота з EditText. Створити додаток, який отримує дані з текстових полів, та виводить їх при натискання кнопки у всплиаючому вікні (Toast) (рис. 2.20).

Рис. 2.20 Вигляд екрану до завдання 2

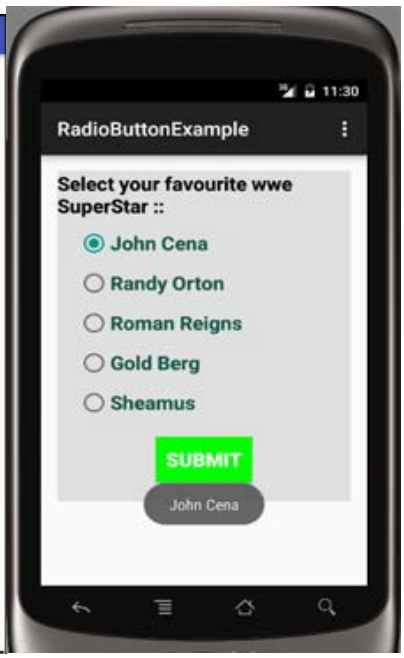


Рис. 2.21 Вигляд екрану до завдання 3

3) Створіть п’ять перемикачів із фоном та іншими атрибутами (рис. 2.21). Радіокнопки використовуються для вибору улюбленої суперзірки за допомогою однієї кнопки «надіслати». Результат вибору виводити в сповіщення Toast.

3.4 Контрольні запитання

1. Основні властивості EditText.
2. Основні Властивості CheckBox.
3. Властивості RadioButton.
4. Особливості обробки натискання кнопки з використанням setOnClickListener().
5. Основні методи Toast.

Кредит 3. Адаптери та списки, фрагменти

Тема 5. Адаптери. Класи BaseAdapter, SimpleAdapter, ArrayAdapter. Використання ListView, GridView, Spinner. Використання віджетів TabHost, WebView

5.1 Адаптер. Види адаптерів

Адаптер - це міст між компонентом інтерфейсу користувача та джерелом даних, який допомагає нам заповнювати даними компоненти інтерфейсу. Він зберігає дані та надсилає дані до представлення адаптера, потім можна отримувати дані з представлення адаптера та відображати у різних представленнях, таких як ListView, GridView, Spinner тощо. Для додаткових налаштувань в віджетах ми використовуємо базовий або спеціальний адаптери.

Для заповнення даних у списку або сітці потрібно реалізувати Adapter. Адаптери діють як міст між компонентом інтерфейсу та джерелом даних. Тут джерелом даних є джерело, звідки отримуємо дані, а компоненти інтерфейсу - це елементи списку або сітки, в яких хочемо відображати ці дані.

Види адаптерів в Android:

1) BaseAdapter - це батьківський адаптер для всіх інших адаптерів.

2) ArrayAdapter - Він використовується, коли список окремих елементів упакований в масив.

3) Custom ArrayAdapter - Він використовується, коли нам потрібно відобразити спеціальний список з власним стилем.

4) SimpleAdapter - це простий адаптер для зіставлення статичних даних з віджетами, визначеними у XML-файлі.

5) Custom SimpleAdapter - Він використовується, коли нам потрібно відобразити індивідуальний список та потрібний доступ до дочірніх елементів списку або сітки

5.2 BaseAdapter в Android

BaseAdapter - це базовий клас загальної реалізації адаптера, який може бути використаний у ListView, GridView, Spinner тощо. Кожного разу, коли потрібен індивідуальний список для ListView або індивідуальні сітки для GridView, необхідно створити власний адаптер і розширити базовий адаптер. Базовий адаптер може розширюватися для створення індивідуального адаптеру з метою відображення спеціального елемента списку. ArrayAdapter також є реалізацією BaseAdapter.

Приклад створення класу Custom Adapter на базі розширення BaseAdapter:

```
public class CustomAdapter extends BaseAdapter {
@Override public int getCount() {return 0;}
@Override public Object getItem(int i) {return null;}
@Override public long getItemId(int i) {return 0;}
@Override public View getView(int i, View view,
ViewGroup viewGroup) {return null;}}
```

У наведеному вище фрагменті коду виконується перевантаження функцій BaseAdapter, які використовуються для встановлення даних у списку, сітці або спінері.

5.3 ArrayAdapter та Custom ArrayAdapter

Якщо список окремих елементів організовані в масив, можна використовувати ArrayAdapter.

Синтаксис ArrayAdapter:

```
ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)
```

Параметри:

1) **context:** Перший параметр використовується для передачі контексту. Тут це ключове слово this використовується для відображення поточного посилання на клас. На місці цього ключового слова можна використовувати getApplicationContext(), getActivity(). getApplicationContext() використовується у Activity, а getActivity () - у фрагменті.

2) **resource:** ідентифікатор ресурсу, який використовується для встановлення макета (файл xml) для елементів списку.

3) **textViewResourceId:** використовується для встановлення ідентифікатора TextView, де потрібно відобразити текст поточного елементу списку.

4) **objects:** масив об'єктів, що використовується для встановлення масиву елементів у textView. Можна встановити об’єкт масиву або списку масивів.

Приклад застосування:

```
String animalList[] = {"Lion","Tiger","Monkey","Elephant","Dog","Cat","Camel"};
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.list_view_items,
R.id.textView, animalList);
```

ArrayAdapter також є реалізацією BaseAdapter, тому, якщо необхідно більше налаштувань, тоді можна створити власний адаптер Custom ArrayAdapter та розширити ArrayAdapter в ньому. Оскільки адаптер масиву є реалізацією BaseAdapter, то можна перевантажити усі функції BaseAdapter в індивідуальному адаптері.

Приклад індивідуального адаптеру MyAdapter, який розширює функції BaseAdapter:

```
public class MyAdapter extends ArrayAdapter {
public MyAdapter(Context context, int resource, int textViewResourceId,
List objects) {super(context, resource, textViewResourceId, objects);}
@Override public int getCount() {return super.getCount();}
@Override
public View getView(int position, View convertView, ViewGroup parent) {
return super.getView(position, convertView, parent);}}
```

5.4 SimpleAdapter, Custom SimpleAdapter

В Android SimpleAdapter - це простий адаптер для зіставлення статичних даних з віджетами, визначеними у файлі XML (макеті). В Android можна запакувати дані в списку виду ArrayList of Maps (тобто hashmap). Кожен запис у ArrayList відповідає одному рядку списку. Карта містить дані для кожного рядка. Також вказується XML-файл (файл користувацьких елементів списку), який визначає віджет, що використовуються для відображення рядка та зіставлення ключів у карті з конкретними представленнями. Кожного разу, коли нам доводиться створювати власний список, нам потрібно реалізувати індивідуальний адаптер. Як зазначалося раніше, ArrayAdapter використовується, коли список окремих елементів упакований в масив. Отже, якщо потрібні додаткові налаштування для ListView або GridView, нам потрібно застосувати простий адаптер.

Код SimpleAdapter у Android:

```
SimpleAdapter (Context context, List<? extends Map<String, ?>> data,
int resource, String[] from, int[] to)
```

Кожного разу, коли нам доводиться створювати власний список, нам потрібно реалізувати спеціальний адаптер. Як зазначалося раніше, ArrayAdapter використовується, коли список окремих елементів запаковано у масив. Отже, якщо потрібно налаштовувати ListView або GridView, тоді необхідно застосувати SimpleAdapter, але коли потрібне додаткове налаштування в елементах списку або сітки, наприклад необхідно обробити будь-яку подію, натискання миші або будь-яку іншу подію віджету, тоді потрібно реалізувати індивідуальний адаптер, який відповідає нашим вимогам і досить простий у реалізації.

BaseAdapter є батьківським адаптером для всіх інших адаптерів, тому, якщо розширювати його у SimpleAdapter, то також можна перевантажувати функції базового адаптера в цьому класі.

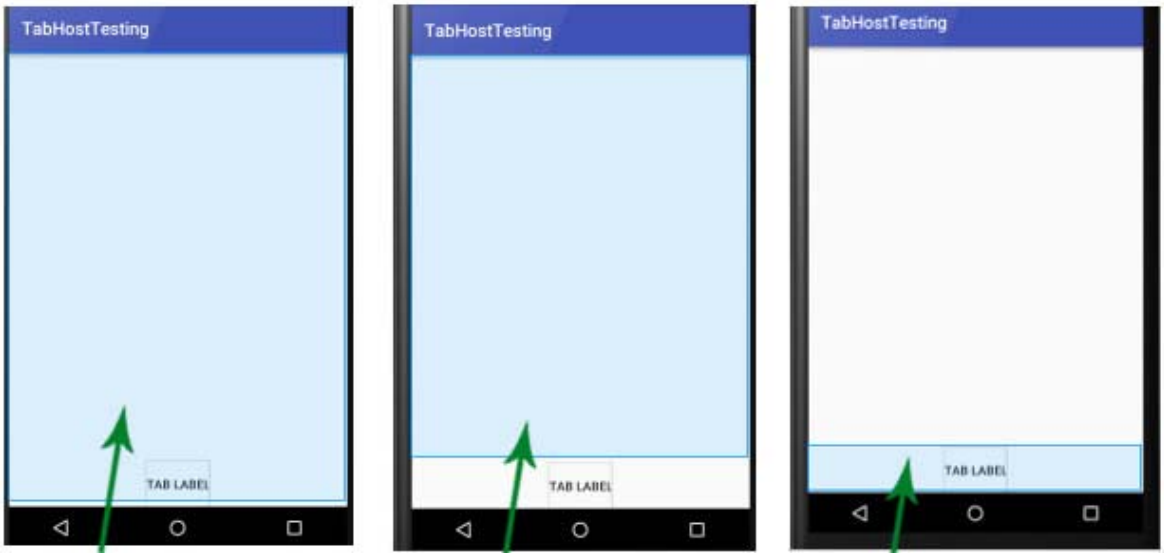
Отже, в стандартному адаптері не можна виконувати такі події, як натискання миші та інші події на дочірньому елементі списку або сітки, але якщо це необхідно, то можна створити власний custom адаптер та розширити SimpleAdapter в ньому.

5.5 Використання віджетів TabHost, WebView

Якщо необхідно ввести або відобразити багато інформації на одній activity, можна застосовувати простий та ефективний метод - використовувати вкладки, які створюються за допомогою віджета TabHost в Android.

В Android TabHost - це контейнер для перегляду вікон із вкладками. Цей об'єкт містить два дочірні елементи, один набір написів вкладок, на які натискає користувач, щоб вибрати певну вкладку, а інший - це об'єкт FrameLayout, який відображає вміст обраної вкладки (рис. 3.1).

Для використання TabHost у MainActivity необхідно розширити клас TabActivity, а не клас Activity.



Повна область TabHost – LinerLayout, яка містить FrameLayout TabWidget

Область – FrameLayout, яка відображає вміст Activity

Область – TabWidget, яка дозволяє обрати користувачу, яку вкладку (Tab) відкрити

Рис. 3.1. Інтерфейс TabHost

Базовий XML код для TabHost:

```
<?xml version="1.0" encoding="UTF-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@android:id/tabhost"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<LinearLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<FrameLayout
android:id="@android:id/tabcontent"
android:layout_width="fill_parent"
android:layout_height="0dip"
android:layout_weight="1" />
<TabWidget
android:id="@android:id/tabs"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_marginBottom="-4dp"
android:layout_weight="0" />
</LinearLayout>
</TabHost>
```

WebView - елемент керування, який відтворює html-код. В якості базових механізмів WebView застосовує WebKit, який забезпечує використання WebView в якості найпростішого веб-браузеру. Механізм «движку» WebKit реалізує приблизно однакове відтворення контенту як і в інших браузерах, що розроблені на WebKit - GoogleChrome, Safari.

Базові методи класу WebView:

- **boolean canGoBack()**: застосовується для перевірки можливості повернутися на одну сторінку назад. Метод вертає «true» у випадку коли в історії навігації перед поточною сторінкою ще є сторінки;
- **boolean canGoForward()**: застосовується для перевірки можливості перейти на одну сторінку вперед. Метод вертає «true» у випадку коли після поточної веб-сторінкою в історії навігації WebView ще є сторінки;

- **void clearCache(boolean includeDiskFiles)**: очищення кешу;
- **void clearFormData()**: очищення даних автоматичного заповнення полів форми;
- **void clearHistory():** очищення історії переходів між веб-сторінками;
- **String getUrl():** дозволяє отримати адресу веб-сторінки**;**
- **void goBack():** дозволяє перейти до попередньої веб-сторінки в історії навігації;
- **void goForward():** дозволяє перейти до наступної веб-сторінки в історії навігації;
- **void loadData (String data, String mimeType, String encoding) :** завантажує до веб-браузеру дані у вигляді html-коду, використовуючи зазначений mime-тип та кодування;
- **void loadDataWithBaseURL (String baseUrl, String data, String mimeType, String encoding, String historyUrl):** аналогічна дія відповідно до методу loadData(), однак в якості першого параметра методу приймає валідну адресу, з якою асоціюються завантажені дані.
- **void loadUrl (String url):** завантажує веб-сторінку за певною адресою;
- **void postUrl (String url, byte [] postData):** відправляє дані за допомогою запиту типу "POST" за певною адресою;
- **void zoomBy (float zoomFactor):** змінює масштаб на значення певного коефіцієнту;
- **boolean zoomIn ():** збільшує масштаб;
- **boolean zoomOut ():** зменшує масштаб.

Лабораторна робота №4. Використання ListView, Spinner

4.1Використання адаптерів в ListView, Spinner.

4.2 Завдання.

4.3 Контрольні запитання.

4.1 Використання адаптерів в ListView, Spinner

Список елементів можна відобразити в Android за допомогою ListView. Він реалізує відображення даних у вигляді прокручуваного списку. Користувач може вибрати будь-який елемент списку, натиснувши його. ListView широко використовується в додатках для Android. Найпоширенішим прикладом ListView є телефонна книга контактів, в якій є список контактів, що відображається у ListView, і якщо натискаєте на контакт, відображається інформація про користувача. Для заповнення даних у ListView використовуються адаптери. Елементи списку автоматично додаються до списку за допомогою адаптера, який отримує вміст із джерела, такого як масив список, масив або база даних. Listview присутній всередині контейнерів. Звідти можна перетягнути на віртуальний мобільний екран, щоб створити його. Можна створити його за допомогою XML-коду.

Атрибути ListView:

Давайте побачимо деякі різні атрибути ListView, які будуть використовуватися при розробці користувацького списку:

1) id: id використовується для однозначної ідентифікації ListView.

2) divider: це малюнок або колір, який можна намалювати між різними елементами списку.

3) spliterHeight: вказує висоту розділювача між елементами списку. Це може бути dp (піксель щільності), sp (незалежний від масштабу піксель) або px (піксель).

4) listSelector: властивість listSelector використовується для встановлення селектора listView. Зазвичай це оранжевий або небесно-блакитний колір, але можна визначити власний колір або зображення як селектор списку відповідно до дизайну.

Приклад коду listSelector:

```
<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="#f00"
android:dividerHeight="1dp"
android:listSelector="#0f0"/>
<!—зелений колір для listSelector -->
```

ListView - це підклас AdapterView, його можна заповнити, прив'язавши до адаптера, який отримує дані із зовнішнього джерела та створює віджет, що представляє кожен запис даних. Якщо список окремих елементів згруповано в масив, можна використовувати ArrayAdapter. За замовчуванням ArrayAdapter взаємодіє з макетом простого TextView. Якщо ви хочете використовувати більш складні віджети, які дозволяють виконувати власні налаштування елементів списку, уникайте ArrayAdapter та використовуйте власні (custom) адаптери.

Нижче наведено приклад заповнення ListView через адаптер масиву:

```
String animalList[] = {"Lion","Tiger","Monkey","Elephant","Dog","Cat","Camel"};
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.list_view_items,
R.id.textView, animalList);
```

В Android Spinner забезпечує швидкий спосіб вибрати одне значення з набору значень. Спінери для Android - це не що інше, як випадаючий список, який можна побачити в інших мовах програмування. У стані за замовчуванням спінер показує вибране на даний момент значення. Spinner асоціюється з видом Adapter, тому для заповнення даних у spinner нам потрібно використовувати один із класу Adapter.

Базовий XML code для Spinner:

```
<Spinner
android:id="@+id/simpleSpinner "
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

Для заповнення даних у спінері потрібно реалізувати клас адаптера. Спінер в основному використовується для відображення текстових полів, тому ми можемо реалізувати для нього ArrayAdapter. Також можна використовувати BaseAdapter та інші користувальницькі адаптери, щоб відобразити список з додатковим налаштуванням.

4.2 Завдання

1. Використовуючи ListView та ArrayAdapter, відобразити список країн або список будь-якого іншого змісту.

2. З використанням SimpleAdapter відобразити список імен тварин з їх зображеннями. Зображення тварини відображається в правій частині екрана, а ім'я відображається в лівій частині екрана. Коли ви натискаєте будь-який елемент списку, ім'я тварини з цього елемента буде відображатися як повідомлення (Toast()) на екрані.

3. Створити список, який відображається в Spinner. Під час вибору елементу списку, значення відображається в toast.

4.3 Контрольні запитання

1. Призначення Адаптерів. Характеристика основних видів адаптерів.
2. BaseAdapter в Android.
3. ArrayAdapter та Custom ArrayAdapter.
4. SimpleAdapter, Custom SimpleAdapter.
5. Використання адаптерів в ListView, Spinner.

Лабораторна робота №5. Використання віджетів TabHost, WebView

5.1 Основні методи TabSpec.

5.2 Основні методи TabHost.

5.3 Робота з WebView.

5.4 Завдання.

5.5 Контрольні запитання.

5.1 Основні методи TabSpec

В Android вкладка має індикатор, вміст і ярлик (TabSpec), який використовується для відстеження вкладок. Цей індикатор допомагає обрати серед варіантів вкладок. Для індикатора вкладки вибір означає обрати ярлик або обрати ярлик та піктограму разом. Для вмісту вкладки вибором є ідентифікатор (id) View, TabHost.TabContentFactory, який створює віджет для перегляду, та Intent, який запускає Activity.

Основні методи TabSpec:

1) setIndicator (CharSequence label): метод використовується для встановлення тексту, який відображатиметься на вкладці. В ньому вказується значення типу CharSequence для відображення мітки для вкладки.

Встановлюється “Tab 1” як індикатор (напис) для вкладки.

```
TabHost tabHost = (TabHost)findViewById(android.R.id.tabhost);
TabHost.TabSpec tabSpec = tabHost.newTabSpec("tab1");
tabSpec.setIndicator("Tab 1");
```

2) setIndicator (CharSequence label, Drawable icon): метод використовується для визначення напису та піктограми як індикатора вкладки. У цьому методі вказується значення напису для відображення мітки для вкладки та Drawable для відображення іконки на вкладці.

```
TabHost tabHost = (TabHost)findViewById(android.R.id.tabhost);
TabHost.TabSpec tabSpec = tabHost.newTabSpec("tab1");
tabSpec.setIndicator("Tab 1",getResources().getDrawable(R.drawable.ic_launcher));
```

3) setContent (Intent intent): визначається намір, який використовується для запуску вмісту вкладки. Щоразу, коли користувач натискає на цю вкладку, відкривається Activity та відображається вміст.

```
TabHost tabHost = (TabHost)findViewById(android.R.id.tabhost);
TabHost.TabSpec tabSpec = tabHost.newTabSpec("tab1");
tabSpec.setIndicator(view);
Intent intent = new Intent(this, MyActivity.class);
tabSpec.setContent(intent);
```

5.2 Основні методи TabHost

1) addTab (TabSpec tabSpec): метод використовується для додавання вкладки у tab widget. Кожного разу, коли створюється нова вкладка за допомогою класу TabSpec, потрібно додати цю вкладку до TabHost.

```
TabHost.TabSpec tabSpec = tabHost.newTabSpec("tab1");
tabSpec.setIndicator(view);
Intent intent = new Intent(this, MyActivity.class);
tabSpec.setContent(intent);
tabHost.addTab(tabSpec);
```

2) clearAllTabs (): метод використовується для видалення всіх вкладок із віджета вкладок (tab widget), пов'язаних із TabHost.

3) setCurrentTab (int index): метод використовується для встановлення поточної вибраної вкладки з віджета вкладок. За замовчуванням хост вкладок встановлює першу вкладку як поточну, але можна змінити її за допомогою цієї функції.

```
TabHost tabHost = (TabHost) findViewById(android.R.id.tabhost);
TabHost.TabSpec tabSpec = tabHost.newTabSpec("tab1");
tabSpec.setIndicator("Tab 1");
Intent intent = new Intent(this, MyActivity.class);
tabSpec.setContent(intent);
TabHost.TabSpec tabSpec1 = tabHost.newTabSpec("tab2");
tabSpec1.setIndicator("Tab 2");
Intent intent1 = new Intent(this, MyActivity.class);
tabSpec1.setContent(intent1);
tabHost.addTab(tabSpec);
tabHost.addTab(tabSpec1);
tabHost.setCurrentTab(1);
```

4) setOnTabChangedListener (OnTabChangeListenerl): метод використовується для реєстрації зворотного виклику, який активується, коли вибраний стан будь-якого з елементів у цьому списку змінюється.

```
tabHost.setOnTabChangedListener(new TabHost.OnTabChangeListener() {
@Override
public void onTabChanged(String tabId) {
// Add code Here
}
});
```

5.3 Робота з WebView

В Android WebView - це віджет, яке використовується для відображення веб-сторінок у програмі. Цей клас є основою, на якій можна створити власний веб-браузер або просто використовувати його для відображення деякого контенту в Інтернеті в межах Activity. Можна також вказати рядок HTML і показати його в додатку за допомогою WebView.

Для того, щоб Activity мав доступ до Інтернету та завантажував веб-сторінки у WebView, необхідно додати дозволи на Інтернет до файлу Android Manifest (Manifest.xml).

```
<!—Додати цей код перед тегом application в AndroidManifest.xml-->
<uses-permission android:name="android.permission.INTERNET" />
```

Методи WebView в Android:

Поширені методи WebView, які використовуються для налаштування веб-віджету в додатку.

1) loadUrl (`String url`) - завантажте веб-сторінку до WebView:

```
/*додаємо до методу Oncreate() після setContentView()*/
WebView simpleWebView=(WebView) findViewById(R.id.simpleWebView);
simpleWebView.loadUrl("http://mdu.edu.ua/");
```

2) loadData () - завантаження статичних HTML даних до WebView:

```
loadData(String data, String mimeType, String encoding)
```

```
/*Додаємо до Oncreate() методу після setContentView()*/
WebView webView = (WebView) findViewById(R.id.simpleWebView);
// статична html-сторінка
String customHtml = "<html><body><h1>Hello, Android</h1>" +
        "<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>" +
        "<p>This is a sample paragraph of static HTML </p>" +
        "</body></html>";
// завантажуємо статичну html-сторінку до web view
webView.loadData(customHtml, "text/html", "UTF-8");
```

3. Завантаження віддаленої URL-адреси до WebView за допомогою WebViewClient:

```
simpleWebView = (WebView) findViewById(R.id.simpleWebView);
// встановлюємо web-view клієнт
 simpleWebView.setWebViewClient(new MyWebViewClient());
// рядок url,який буде завантажуватися до web view
 String url = "http://mdu.edu.ua/";
 simpleWebView.getSettings().setJavaScriptEnabled(true);
 simpleWebView.loadUrl(url);
}
 // class web-view клієнта, який розширує WebViewClient
 private class MyWebViewClient extends WebViewClient {
 @Override
 public boolean shouldOverrideUrlLoading(WebView view, String url)
{
 view.loadUrl(url);
 return true;
 }
```

WebViewClient допомагає відстежувати події в WebView. Необхідно замінити метод shouldOverrideUrlLoading(). Цей метод дозволяє виконувати власні дії, коли обрано певну URL-адресу. Після того, як готові до роботи з WebViewClient, можна встановити WebViewClient у своєму WebView за допомогою методу setWebViewClient ().

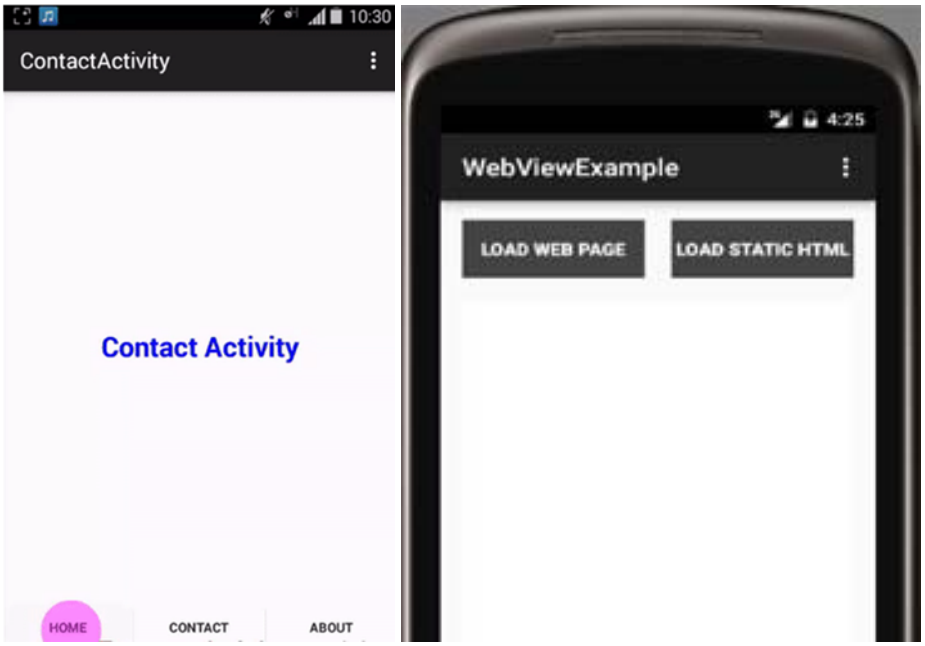
3) canGoBack () - Перейти на одну сторінку назад, якщо існує попередня історія. Цей метод повертає булеве значення: true, або false. Якщо

він повертає true, тоді метод goBack () використовується для переміщення на одну сторінку назад.

4) canGoForward () - Переміщення на одну сторінку вперед, якщо історія пересилання існує. Метод повертає булеве значення. Якщо він повертає true, тоді метод goForword() використовується для переміщення на одну сторінку уперед.

5.4 Завдання

1. Створіть інтерфейс відповідно до рисунку 3.2(а). В додатку реалізувати три вкладки з іменем home, contact та about за допомогою TabHost.TabSpec, в якості вмісту вкладок відображаються статичні дані. Також реалізувати setOnTabChangeListener, який при зміні вкладки за допомогою Toast() відображає назву вкладки.



а) б)

Рис. 3.2. Приклад інтерфейсів для завдань №1-2, лабораторна робота №5

2. Створіть інтерфейс для роботи з WebView (рис. 3.2(б)). Необхідно відобразити дві кнопки, одна - для відображення веб-сторінки, а інша - для відображення статичних даних HTML у веб-віджеті.

5.5 Контрольні запитання

1. Поняття TabHost, особливості застосування.
2. Основні методи TabSpec.
3. Основні методи TabHost.
4. Основні методи WebView.

Тема 6. Фрагменти. Взаємодія між фрагментами

В Android Fragment є частиною activity, яка забезпечує більш модульний дизайн діяльності. Фрагмент - це свого роду sub-activity. Фрагмент представляє поведінку або частину користувальницького інтерфейсу в Activity. Можна об’єднати декілька фрагментів в одній Activity, щоб створити інтерфейс із декількома панелями та повторно використовувати фрагмент у декількох Activites. Завжди потрібно вбудовувати Фрагмент в Activity. Життєвий цикл фрагмента безпосередньо залежить від життєвого циклу діяльності хостової Activity.

Фрагмент інкапсулює функціональність, щоб було легше повторно використовувати його в межах activity та макетів. Пристрої Android існують у різних розмірах та роздільності екрану. Фрагменти спрощують повторне використання компонентів у різних макетах та їх логіку. Можна створювати однопанельні макети для мобільних телефонів та багатопанельні макети для

планшетів. Можна використовувати фрагменти для підтримки різних макетів для альбомної та книжкової орієнтації на смартфоні.

На рис. 3.3 зображено, як два модулі інтерфейсу користувача, визначені фрагментами, можна об’єднати в одній activity для дизайну планшета, але розділити для дизайну телефону.

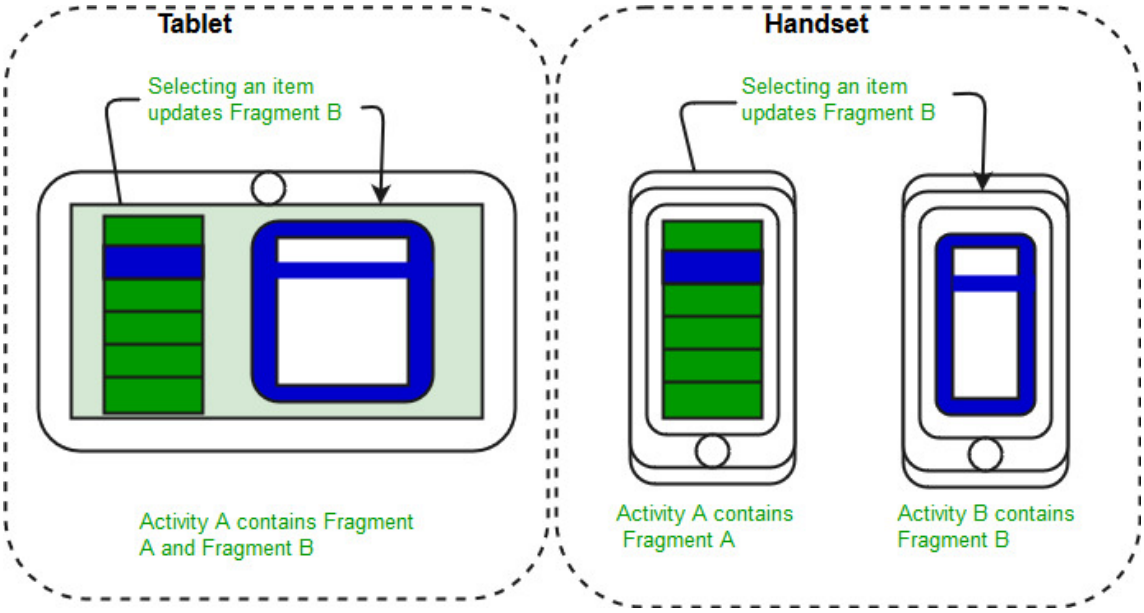


Рис. 3.3. Відображення фрагментів для планшету (а) та телефону (б)

До введення Fragment’s показувалася лише одна активність на екрані в заданий момент часу. За допомогою Fragment’s можна розділити екрани на різні частини та контролювати різні частини окремо. Використовуючи фрагменти, можна включати декілька фрагментів в одну активність. Фрагменти мають власні події, макети та повний життєвий цикл. Це забезпечує гнучкість, а також усуває обмеження окремої активності на екрані в заданий момент часу.

Лабораторна робота №6. Фрагменти

6.1 Створення Fragment Class.

6.2 Приклад роботи з Fragment.

6.3 Завдання.

6.4 Контрольні запитання.

6.1 Створення Fragment Class

Для створення Фрагменту спочатку розширюється Fragment class, а потім перевантажуються ключові методи життєвого циклу, щоб реалізувати логіку додатку, подібно до того, як це було б із класом Activity . Під час створення Fragment ми повинні використовувати зворотний виклик onCreateView () для визначення макета та для запуску Fragment.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;
public class FirstFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return  inflater.inflate(R.layout.fragment_first,  container,
false);
} }
```

Тут параметром inflater є LayoutInflater, що використовується для відтворення макету, параметром контейнера є батьківський віджет ViewGroup (із макета Activity), в який буде вставлено макет Fragment.

Параметр saveInstanceState - це Bundle (набір), який надає дані про попередній екземпляр фрагмента.

```
view = inflater.inflate(R.layout.fragment_first, container, false);
```

Метод inflate () має три аргументи, перший - це макет ресурсу, який відтворюється, другий - ViewGroup, який є батьківським для inflated макету. Передача контейнера є необхідною для того, щоб система застосувала параметри макета до кореневого віджету inflated макета, вказаного батьківським віджетом, в якому він відображається, а третій параметр - це логічне значення, яке вказує, чи слід приєднувати inflated макет до ViewGroup (другий параметр під час відтворення).

Типи фрагментів:

1) Фрагменти одного кадру: Фрагменти одного кадру використовуються для таких пристроїв, як мобільні телефони, тут ми можемо показати лише один фрагмент у Activity.

2) Фрагменти списку: фрагменти, що мають спеціальний вигляд списку, називаються фрагментами списку

3) Транзакція фрагментів: забезпечується можливість перенесення одного фрагменту в інший фрагмент.

Фрагмент існує у трьох станах:

1) Resumed: фрагмент видно в поточній Activity.

2) Paused: Інша Activity знаходиться на передньому плані та має фокус, але Activity, у якій живе цей фрагмент, все ще видима (Activity на передньому плані частково прозора або не охоплює весь екран).

3) Stopped: фрагмент не видно. Або Activity хоста була зупинена, або фрагмент видалено з Activit, але додано до заднього стеку. Зупинений фрагмент все ще живий (вся інформація про стан та елементи зберігається

системою). Однак його користувач більше не бачить, і він буде видалений якщо Activity буде видалена.

6.2 Приклад роботи з Fragment

У прикладі створюються два фрагменти та завантажуємо їх натисканням кнопки. Відображаються дві Button's та FrameLayout в Activity та виконуємо подію setOnClickListener на обох Button's. При натисканні кнопки «Перша кнопка» замінюємо «First Fragment», а після натискання кнопки «Друга кнопка» ми замінюємо « Second Fragment» макетом (FrameLayout). В обох фрагментах ми відображаємо TextView та кнопку, а при натисканні кнопки ми відображаємо назву фрагмента за допомогою тосту.

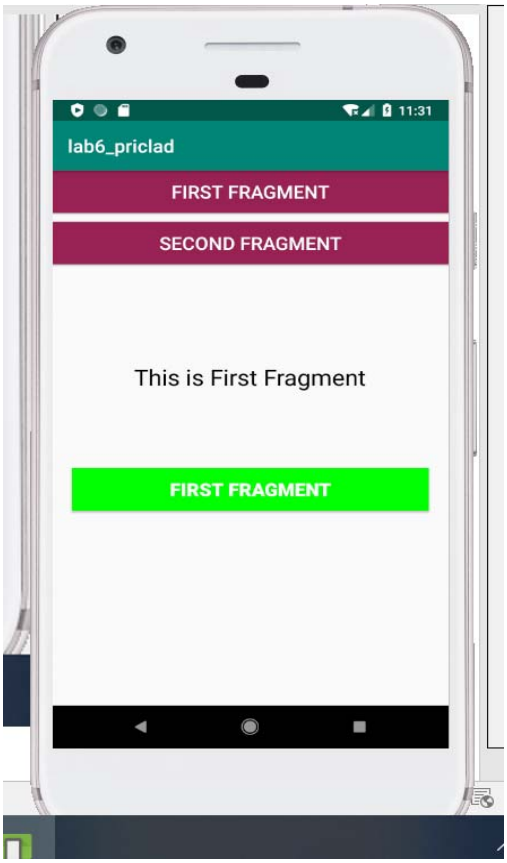


Рис. 3.4. Результат роботи з Fragment

На рис. 3.4 зображено результат роботи прикладу.

1) Створюємо макет основної активності:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/firstFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/button_background_color"
        android:text="First Fragment"
        android:textColor="@color/white"
        android:textSize="20sp" />
    <Button
        android:id="@+id/secondFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:background="@color/button_background_color"
        android:text="Second Fragment"
        android:textColor="@color/white"
        android:textSize="20sp" />
    <FrameLayout
        android:id="@+id/frameLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="10dp" />
</LinearLayout>
```

2) У файл colors.xml додаємо кольори, які будуть задіяні в проєкті.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
        <!—кольори, які застосовані в проєкті -->
        <color name="black">#000</color>
        <color name="green">#0f0</color>
        <color name="white">#fff</color>
        <color name="button_background_color">#925</color>
</resources>
```

3) Створюємо фрагменти **FirstFragment, SecondFragment:** Головне меню, File/New/Fragment/Fragment(Blank). При цьому створюються класи та .xml файли.

Вміст fragment_first.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.andr
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstFragment">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:text="This is First Fragment"
        android:textColor="@color/black"
        android:textSize="25sp" />
    <Button
        android:id="@+id/firstButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:background="@color/green"
        android:text="First Fragment"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Вміст fragment_second.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondFragment">
    <!--TextView and Button displayed in Second Fragment -
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:text="This is Second Fragment"
        android:textColor="@color/black"
        android:textSize="25sp" />
    <Button
        android:id="@+id/secondButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:background="@color/green"
        android:text="Second Fragment"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Вміст FirstFragment.java:

```
package com.example.lenovo.lab6_priclad;
import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;
public class FirstFragment extends Fragment {
    View view;
    Button firstButton;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,Bundle savedInstanceState)
    {
        view = inflater.inflate(R.layout.fragment_first, container,
false);
        firstButton = (Button) view.findViewById(R.id.firstButton);
        firstButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
Toast.makeText(getActivity(), "First Fragment", Toast.LENGTH_LONG).show();
            }
        });
        return view; }
    }
```

Вміст SecondFragment.java:

```
public class SecondFragment extends Fragment {

    View view;
    Button secondButton;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
// Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_second, container,
false);
// get the reference of Button
        secondButton = (Button) view.findViewById(R.id.secondButton);
// perform setOnClickListener on second Button
        secondButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
// display a message by using a Toast
                Toast.makeText(getActivity(), "Second Fragment",
Toast.LENGTH_LONG).show();
            }
        });
        return view;
    }
}
```

Клас MainActivity.java:

```
public class MainActivity extends AppCompatActivity {
    Button firstFragment, secondFragment;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        firstFragment = (Button) findViewById(R.id.firstFragment);
        secondFragment = (Button) findViewById(R.id.secondFragment);
        firstFragment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                loadFragment(new FirstFragment());
            }
        });
        secondFragment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                loadFragment(new SecondFragment());
            }
        });   }
    private void loadFragment(Fragment fragment) {
        FragmentManager fm = getFragmentManager();
        FragmentTransaction fragmentTransaction = fm.beginTransaction();
        fragmentTransaction.replace(R.id.frameLayout, fragment);
        fragmentTransaction.commit();
    } }
```

6.3 Завдання

1. У фрагменті 1 є текстове поле і кнопка. При натисканні на кнопку текст з поля передається (повторюється) в три текстових поля фрагмента 2.

2. На активності створити три закладки та прив'язати до них виклик власного фрагмента. На фрагментах в центрі розташовано одне текстове поле. При зміні закладки вміст текстового поля має передаватися на фрагмент, який відкривається.

6.4 Контрольні запитання

1. Створення фрагменту в коді.

2. Взаємодія між фрагментами. Фрагменти в альбомному і портретному режимі.

3. Життєвий цикл та типи фрагментів.

Кредит 4. Наміри (Intent) в ОС Android

Тема 7. Активності та ресурси

Activity (Активність, діяльність) - екран, з яким взаємодіє користувач, один із основних елементів операційної системи Android. Оскільки Activity - це клас, попередньо написаний у програмуванні на Java, то кожна активність має життєвий цикл: створення, запуск, відновлення, призупинення, зупинка або знищення.

Короткий опис методів життєвого циклу Activity:

1) **onCreate() -** викликається, коли Activity створюється вперше.

2) **onStart() -** викликається після створення Activity або методом перезапуску після onStop (). В цьому методі Activity починає ставати видимою для користувача.

3) **onResume() -** викликається, коли Activity є видимою для користувача, і користувач може взаємодіяти з нею.

4) **onPause() -** викликається, коли вміст Activity не видно, оскільки користувач відновлює попередні дії.

5) **onStop() -** викликається, коли Activity не відображається користувачеві, оскільки в ній відбувається якась інша Activity

6) **onRestart() -** викликається, коли користувач активує екран або відновлює припинену Activity

7) **onDestroy() -** викликається, коли Activity відсутня у фоновому режимі.

На рисунку 4.1 представлено діаграму життєвого циклу **Activity**, яка показує різні стани.

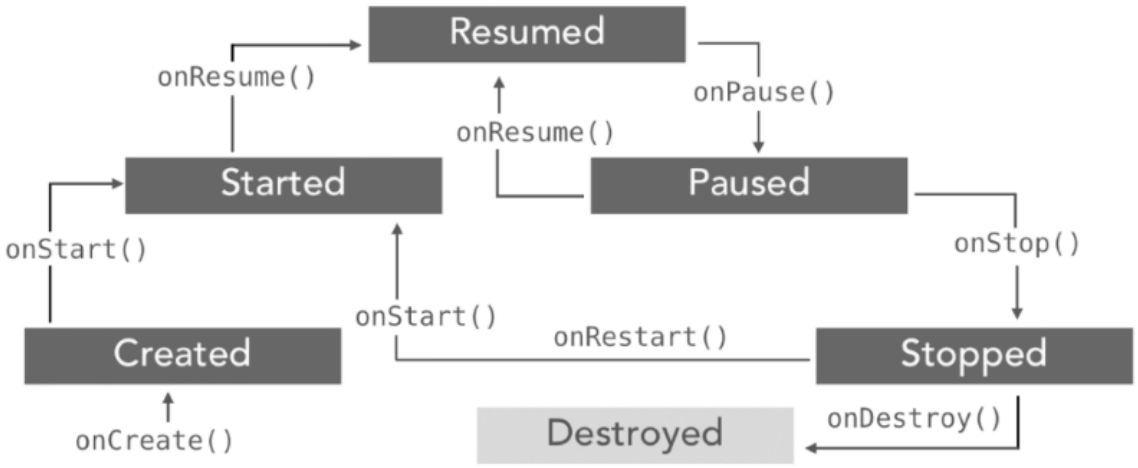


Рис. 4.1. Діаграма життєвого циклу Activity

Усі методи життєвого циклу не потрібно замінювати, але досить важливо розуміти їх. Методи життєвих циклів можна замінити відповідно до вимог.

Список методів та станів життя діяльності:

1) Активність створена: onCreate (Bundle savedInstanceState). Метод onCreate() викликається, коли діяльність отримує пам'ять в ОС. Для використання стану create нам потрібно перевизначити метод onCreate (Bundle savedInstanceState). Тепер виникне питання на увазі, що тут таке Bundle, тому Bundle - це об’єкт сховища даних, який може зберігати будь-які примітивні дані, і цей об’єкт буде нульовим, доки в ньому не будуть збережені деякі дані.

Властивості методу onCreate ():

– під час першого виклику або запуску Activity відбувається метод onCreate (Bundle savedInstanceState), який відповідає за створення активності.

– якщо орієнтація Activity змінюється (з горизонтальної на вертикальну або вертикальну на горизонтальну), або коли Activity

примусово припиняється будь-якою операційною системою, тоді saveInstanceState, тобто об’єкт класу Bundle, зберігає стан Activity.

– в цьому методі краще всього розмістити код ініціалізації.

2) Активність розпочата: onStart (). Метод onStart () викликається відразу після створення Activity. В іншому випадку Activity можна розпочати, викликавши метод перезапуску, тобто після зупинки Activity. Отже, це означає, що onStart () викликається ОС Android, коли користувач перемикається між програмами. Наприклад, якщо користувач використовував програму А, а потім надійшло повідомлення, і користувач натиснув повідомлення та перемістився до програми Б, у цьому випадку програма А буде призупинена. І знову, якщо користувач знову натисне піктограму програми Додаток А, тоді Програма А, яку було зупинено, знову почне працювати.

Властивості методу onStart():

– коли діяльність почне ставати видимою для користувача, тоді буде викликано onStart();

– це викликає відразу після onCreate () при першому запуску активності.

– під час запуску діяльності спочатку викличте метод onCreate (), потім onStart(), а потім onResume();

– якщо активність знаходиться у стані onPause (), тобто не бачиться користувачеві. І якщо користувач знову запустить діяльність, тоді буде викликаний метод onStart().

Приклад onStart()

Приклад Android-додатку, в якому відображаються повідомлення Toast на екрані при виклику методу onStart (). Спочатку створіть новий проект, назвіть діяльність MainActivity і створіть content.xml у папці макета, якщо він відсутній за замовчуванням.

Інтерфейс складається з одного TextView, в якому відображаєтсья текстове повідомлення про виклик onStart () після onCreate ().

Код content.xml:

```
<RelativeLayout 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="fill_parent"
    android:layout_height="fill_parent">
   <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="onStart() Call after onCreate()" />
</RelativeLayout>
```

Під час запуску додатку в емуляторі з'являються два повідомлення Toast, які спочатку викликаються методом onCreate (), а потім onStart ():

3) Відновлення Активності: onResume (). Activity відновлена - це така ситуація, коли вона фактично видима користувачеві. Це означає, що дані, які відображаються в цій діяльності, є видимими для користувача. У життєвому циклі його завжди викликають після початку Activity, а в більшості випадків використання після призупинення Activity (onPause()).

4) Активність призупинена: onPause ():

Активність є призупиненою, коли її вміст не видно користувачеві. В більшості випадків метод onPause() викликається ОС Android, коли

користувач натискає кнопку головного меню, щоб приховати екран (центральна кнопка на пристрої).

Код MainActivity.java:

```
package abhiandroid.com.exampleonstart;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.content);
    Toast.makeText(getApplicationContext(),   "First   onCreate()   calls",
Toast.LENGTH_SHORT).show(); //onCreate called
       }
    @Override
    protected void onStart()  {
    super.onStart();
  Toast.makeText(getApplicationContext(),"Now       onStart()       calls",
Toast.LENGTH_LONG).show(); //onStart Called
       }
}
```

Активність також отримує паузу перед зупинкою у разі , якщо користувач натисне назад навігаційну кнопку . Діяльність буде переведена в призупинений стан з цих причин також, якщо повідомлення або якесь інше діалогове вікно перекриває будь-яку частину (верхню чи нижню) екрану. Подібним чином, якщо інший екран або діалогове вікно прозорі, тоді користувач може бачити екран, але не може взаємодіяти з ним. Наприклад, якщо надходить дзвінок або сповіщення, користувач отримає можливість прийняти дзвінок або проігнорувати його.

5) Активність зупинена: onStop(). Активність називається зупиненою, коли вона не видна користувачеві. Будь-яка Активність

припиняється на випадок, якщо з неї відбувається якась інша діяльність. Наприклад, якщо користувач був на екрані 1 і натиснув якусь кнопку, і перейшов на екран 2. У цьому випадку Активність, що відображає вміст для екрану 1, буде зупинена. Кожна Активність зупиняється перед знищенням у випадку, коли користувач натискає назад кнопку навігації. Тож Активність буде у зупиненому стані, якщо прихована або замінена іншими діями, які були запущені або переключені користувачем. У цьому випадку додаток не буде представляти нічого корисного для користувача безпосередньо, оскільки він збирається зупинитися.

6) Активність перезапущена: onRestart(). Активність викликається у стані перезапуску після стану зупинки. Тож функція onRestart() активності викликається, коли користувач виходить на екран або відновлює Активність, яку було зупинено. Іншими словами, коли операційна система запускає Активність, операція onRestart () ніколи не викликається. Метод викликається лише в тому випадку, коли Активність відновлюється після зупиненого стану.

7) Активність знищена: onDestroy(). Будь-яка Активність вважається зруйнованою, якщо вона не знаходиться у фоновому режимі. Можуть існувати різні випадки, коли саме Активність знищується. По-перше, якщо користувач натиснув кнопку зворотної навігації, тоді активність буде знищена після завершення життєвого циклу паузи та зупинки. У випадку, якщо користувач натискає кнопку головного меню і програма переходить на задній план. Користувач більше не використовує Активність, і вона відображається у списку останніх програм. Отже, у цьому випадку, якщо системні ресурси потрібні де-небудь ще, тоді ОС може знищити Активність.

Після того, як Активність буде знищена, якщо користувач ще раз натисне піктограму програми, Активність буде відтворена та повторюватиметься той самий життєвий цикл знову. Іншим варіантом використання є заставки Splash Screens, якщо є метод call to finish() із onCreate () Активності, тоді ОС може безпосередньо викликати onDestroy () із викликами onPause () та onStop ().

Приклад життєвого циклу діяльності:

У наведеному нижче прикладі використано клас Log, який використовується для виведення повідомлень-логів у Logcat. Одним з важливих застосувань журналу логів є відладка (debugging).

Активність називається HomeActivity. Ініціалізується статична змінна String з назвою базового класу за допомогою методу getSimpleName(). HOME_ACTIVITY_TAG - це ім'я змінної String, яка зберігає ім'я класу HomeActivity.

```
private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();
```

Метод, який виведе повідомлення в Logcat:

```
private void showLog(String text){  Log.d(HOME_ACTIVITY_TAG, text); }
```

Замінюються усі методи життєвого циклу активності в Android та використовується метод showLog().

```
public class HomeActivity extends AppCompatActivity {
private static final String HOME_ACTIVITY_TAG =
HomeActivity.class.getSimpleName();
private void showLog(String text){ Log.d(HOME_ACTIVITY_TAG, text); }
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); showLog("Activity Created");  }
@Override
protected void onRestart(){ super.onRestart();//call to restart after onStop
showLog("Activity restarted");     }
@Override
protected void onStart() {
super.onStart();//soon be visible
showLog("Activity started");     }
@Override
protected void onResume() {
super.onResume();//visible
showLog("Activity resumed");    }
@Override
protected void onPause() { super.onPause();//invisible
showLog("Activity paused");       }
@Override
protected void onStop() {  super.onStop();
showLog("Activity stopped");     }
@Override
protected void onDestroy() { super.onDestroy();
showLog("Activity is being destroyed");   }
}
```

Створюючи Активність, потрібно зареєструвати її в AndroidManifest.xml тому, що у файлі маніфесту є інформація, яку ОС Android зчитує не першому етапі. Під час реєстрації активності в маніфесті також може бути визначена інша інформація, як Launcher Activity (Активність, яка повинна розпочатися, коли користувач натискає піктограму програми).

Під час запуску програми в Емуляторі відкривається порожній білий екран. Екран Hello World видаляється, перевизначившись методом onCreate(). Для перегляду результату необхідно перейти до Logcat, присутнього в моніторі середовища Android Studio та, прокрутивши вгору, знайти три методи, які були названі: Activity Created, Activity started та Activity resumed.

Приклад AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abhiandroid.homeactivity">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".HomeActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Отже:

перший метод onCreate() був викликаний при створенні активності;

– другий метод onStart() був викликаний, коли Активність починає ставати видимою для користувача;

– метод onResume() був викликаний, коли Активність є видимою для користувача і користувач може взаємодіяти з нею.

Після натиснення кнопки назад на емуляторі, необхідно знову перейти до Logcat, прокрутити вниз. Побачимо, що було викликано ще 3 методи: Activity paused, Activity stopped та Activity destroyed.

Отже, це означає:

– метод onPause() був викликаний, коли користувач відновив попередню активність;

– метод onStop () був викликаний, коли діяльність не видна користувачеві;

– останній метод onDestroy () викликався, коли Activity не знаходиться у фоновому режимі

Важлива примітка: У наведеному вище прикладі onRestart() не буде викликаний, оскільки не було ситуації, коли ми можемо відновити метод onStart() знову.

Важливість життєвого циклу Активності: Активність є головною складовою програми Android, оскільки кожен екран є активністю, тому для створення будь-якого простого додатка спочатку неохідно розпочати зі створення Activities. Кожен метод життєвого циклу досить важливий та реалізується у відповідності до вимог, однак onCreate(Bundle state) завжди необхідно реалізовувати, щоб показати або відобразити деякий вміст на екрані.

Тема 8. Наміри в Android: явні і неявні. Створення Активностей за допомогою Намірів.

8.1 Поняття та призначення Intent

Android використовує Intent для взаємодії між компонентами програми, а також взаємодії між програмами. Намір - це об’єкти, які використовуються в Android для передачі інформації між Activities в додатку або між додатками. Android використовує Intents для полегшення зв'язку між такими компонентами, як Activities, Services та Broadcast Receivers.

Наприклад: Намір дозволяє здійснити перенаправлення з однієї activity на іншу activity під час виникнення будь-якої події. Викликавши startActivity, можна виконати це завдання.

```
Intent intent = new Intent(getApplicationContext(),

SecondActivity.class);startActivity(intent);
```

Під час створення Наміру застосовується об’єкт з використанням конструктору, який приймає два параметри. Перший – контекст, - об’єкт, який надає доступ до базових функцій додатку. Другий параметр – ім’я класу.

Варіанти застосування Намірів:

1) Намір для діяльності (Activity). Кожен екран програми Android представляє діяльність. Щоб запустити нову Activity, потрібно передати об'єкт Intent методу startActivity(). Цей об’єкт Intent допомагає запустити нову Activity та передати дані до неї.

2) Наміри для служб (Services). Служби працюють у фоновому режимі програми Android та не мають інтерфейсу. Наміри можуть бути використані для запуску Служби, яка виконує одноразове завдання (наприклад: завантаження якогось файлу). Для запуску Служби потрібно передати Intent до методу startService ().

3) Намір для широкомовних повідомлень (Broadcast Receivers). Існують різні повідомлення, яке отримує додаток; ці повідомлення називаються широкомовними приймачами. (Наприклад, широкомовне повідомлення може бути ініційоване, щоб повідомити, що завантаження файлу завершено та він готовий до використання). Система Android ініціює широкомовні повідомлення щодо декількох подій, таких як перезавантаження системи, попереджувальне повідомлення про низький рівень заряду тощо.

Таким чином, кожного разу, коли потрібно перейти до іншої Activity додатку або потрібно надіслати деяку інформацію до наступної Activity, можна віддати перевагу Intents. Наміри дійсно прості в обробці, полегшують зв'язок між компонентами та Activities додатку. Крім того, можна взаємодіяти з іншим додатком та надсилати деякі дані до нього, використовуючи Intents.

8.2 Типи Intent

Наміри бувають двох типів: Явний (Explicit) та Неявний (Implicit).

4) Явні наміри використовуються для внутрішнього з’єднання в середині програми. В Explicit використовується ім'я компоненту, на який вплине Intent. Наприклад: якщо знаємо назву класу, тоді можна переходити між додатками з однієї активності на іншу, використовуючи Intent. Подібним чином можна запустити службу для завантаження файлу у фоновому режимі. Явний намір працює всередині програми для виконання навігації та передачі даних.

5) Неявний намір. У Implicit Intents немає необхідності вказувати назву компоненту. Вказується дія, яку потрібно виконати, далі ця дія обробляється компонентом іншої програми. Основним прикладом неявного наміру є відкриття будь-якої веб-сторінки.

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("http://www.mdu.edu.ua"));
startActivity(intentObj);
```

Під час застосування «Явного» наміру метод putExtra() об’єкту Intent застосовується для передачі даних між двома Activity. В якості значення

метод putExtra() може передавати дані базових типів: String, int, float, double, long, short, byte, char, масиви даних типів або об’єкт інтерфейсу Serializable.

```
// створення об’єкту Intent для запуска SecondActivity
Intent intent = new Intent(this, SecondActivity.class);
// передача об’єкту з ключем "hello" та значенням "Hello World"
intent.putExtra("hello", "Hello World");
// запуск SecondActivity
startActivity(intent);
```

Щоб отримати відправлені дані, можна застосувати метод get(), який в якості аргументу приймає ключ.

```
Bundle arguments = getIntent().getExtras();
String name = arguments.get("hello").toString();
```

В залежності від типу даних, які отримуються, можна застосовувати різні методи об’єкту Bundle:

– get(): універсальний метод, який повертає значення типу Object. Відповідно поле отримання дане значення необхідно перетворити до потрібного типу;

– getString(): повертає об'єкт типу String;

– getInt(): повертає значення типу int;

– getByte(): повертає значення типу byte;

– getChar(): повертає значення типу char;

– getShort(): повертає значення типу short;

– getLong(): повертає значення типу long;

– getFloat(): повертає значення типу float;

– getDouble(): повертає значення типу double;

– getBoolean(): повертає значення типу boolean;

- getCharArray(): повертає масив об'єктів char;
- getIntArray(): повертає масив об'єктів int;
- getFloatArray(): повертає масив об'єктів float;
- getSerializable(): повертає об'єкт інтерфейсу Serializable.

Лабораторна робота №7. Наміри в Android. Використання Intent для взаємодії Activities

7.1 Фільтри Intent.

7.2 Завдання.

7.3 Контрольні запитання.

7.1 Фільтри Intent

Коли створюється неявний об'єкт Intent, система Android знаходить відповідний компонент за допомогою порівняння вмісту об'єкта Intent із фільтрами Intent, розміщеними у файлі маніфесту інших додатків. Якщо об'єкт Intent відповідає фільтру Intent, система запускає цей компонент та передає йому об'єкт Intent. Якщо підходящими виявляються декілька фільтрів Intent, система виводить діалогове вікно, де користувач може обрати додаток для виконання даної дії. Система передає неявний об’єкт Intent додатку тільки якщо він зможе пройти через один із фільтрів.

Кожен фільтр Intent визначається елементом intent-filter в файлі маніфесту додатку, зазначеному в оголошенні відповідного компоненту додатку. Усередині елемента intent-filter можна вказати тип об'єктів Intent за допомогою одного або декількох елементів: action - оголошує дію, задану для об'єкту Intent, в атрибуті name; data - оголошує тип даних, які

приймаються; category - оголошує прийняту категорію, задану в об'єкті Intent, в атрибуті name:

```
<intent-filter>
<action>
</action>
<data>
</data>
<category>
</category>
</intent-filter>
```

1) Фільтр action.

```
<intent-filter>

<action android:name="android.intent.action.MAIN" />

<action android:name="android.intent.action.EDIT" />

<action android:name="android.intent.action.VIEW" />

</intent-filter>
```

Дії задаються константами дій:

– ACTION_VIEW - найбільш поширена загальна дія, застосовується з методом startActivity(), коли є інформація, яку Activity може показати користувачеві, наприклад, фотографія в додатку галереї;

– ACTION_ANSWER - відкриває активність, яка пов'язана з вхідними дзвінками;

– ACTION_CALL - ініціалізує звернення телефоном;

– ACTION_DELETE - запускає активність, за допомогою якої можна видалити дані, зазначені в шляху URI всередині Intent;

– ACTION_EDIT - відображає дані для редагування користувачем;

– ACTION_INSERT - відкриває активність для вставки в Cursor нового елемента, зазначеного за допомогою шляху URI;

– ACTION_MAIN - запускається як початкова активність завдання;

– ACTION_SEARCH - запускає активність для виконання пошуку;

– ACTION_SEND - завантажує екран для відправки даних, вказаних в намірі;

– ACTION_SENDTO - відкриває активність для відправки контакту, що вказаний в шляху URI, який передається через Intent;

– ACTION_SYNC - синхронізує дані сервера з даними мобільного пристрою.

2) Фільтр data.

Для завдання типу даних, які приймаються об'єктом Intent фільтру може оголошуватися будь-яке(в тому числі нульове) число елементів data.

Наприклад:

```
<intent-filter>
<data android:mimeType="video/mpeg" android:scheme="http" ... />
<data android:mimeType="audio/mpeg" android:scheme="http" ... />
...
</intent-filter>
```

Кожен елемент data може конкретизувати структуру URI і тип даних. Є окремі атрибути - scheme, host, port і path -для кожної складової частини URI: scheme: // host: port / path.

Кожен з цих атрибутів є необов'язковим, але є лінійні залежності: якщо схема не вказана, вузол ігнорується; якщо вузол не вказано, порт ігнорується; якщо не вказана ні схема, ні вузол, шлях ігнорується.

6) Фільтр category.

Щоб об'єкт Intent пройшов тестування категорії, усі категорії, наведені в об'єкті Intent, повинні відповідати категорії android.intent.category.DEFAULT.

Можна задати власні категорії або ж брати стандартні значення, що надаються системою:

– ALTERNATIVE - дія має бути доступною в якості альтернативної тому, що виконується за замовчуванням для елемента цього типу даних. Наприклад, якщо дія за замовчуванням для контакту - перегляд, то в якості альтернативи його також можна редагувати;

– SELECTED_ALTERNATIVE - те ж саме, що і ALTERNATIVE, але замість одиночної дії застосовується в тих випадках, коли потрібен список різних можливостей;

– BROWSABLE – дія, яка дозволена для активності зі сторони браузера;

– DEFAULT - робить компонент оброблювачем за замовчуванням для дії, що виконується з вказаним типом даних всередині фільтра;

– GADGET - активність може запускатися всередині іншої активності;

– LAUNCHER - поміщає Activity в вікно для запуску додатків.

7.2 Завдання

Створити додаток з двома кнопками: «Явний намір», «Неявний намір». При натисканні на кнопку «Явний намір», відкрити нову Activity, в якій розмістити текстове поле «Це втора Activity». Вивести Toast() з повідомленням «Ви переміщуєтесь на другій активності». При натисненні

на кнопку «Неявний намір», відкривається браузер із заданою URL-адресою.

7.3 Контрольні запитання

1. Поняття та призначення намірів.
2. Поняття «Явного» та «Неявного» намірів.
3. Що таке фільтр Intent. Види фільтрів.

Кредит 5. Робота із базами даних

Тема 9. Бази даних в Android. Застосування Shared Preference для роботи з даними.

9.1 Поняття Shared Preference

Shared Preference в Android використовуються для збереження даних на основі пари ключ-значення. За смисловим значенням Shared Preferences є загальнодоступними даними з наданою перевагою.

Shared Preference можна використовувати для збереження примітивних типів даних: string, long, int, float та Boolean.

Preference –файл це xml-файл, який зберігається у внутрішній пам'яті пристрою. Кожна програма має деякі дані, які зберігаються в пам'яті в каталозі data / data / application package name, тобто data / data / com.example.lenovo.lab7_priclad, тому щоразу, коли викликається функція getSharedPreferences (string name, int mode), вона перевіряє чи існує файл, інакше створюється новий xml-файл із переданою назвою.

9.2 Види Shared Preference. Методи отримання доступу

Існує три різні способи збереження даних в Android за допомогою Shared Preference - перший використання Preference, які базуються на Activity, другий - створення власних користувальницьких Preference, а третій – викликається з об’єкту PreferencesManager.

Властивості Activity Preference:

1) Для Activity Preference необхідно викликати функцію **getPreferences (int mode),** доступну в класі Activity

2) Використовуйте лише тоді, коли в Activity потрібен лише один Preference-файл.

3) Він не вимагає імені, оскільки це буде єдиним Preference-файлом для цієї діяльності

4) Розробники не вважають за спосіб кращим. Навіть якщо потрібен лише один файл налаштувань в Activity, вважаються кращим використовувати користувальницьку функції getSharedPreferences (string name, int mode).

Для використання Activity Preference розробник повинен викликати функцію **getPreferences (int mode),** доступну в класі Activity. Функція **getPreferences** (**int mode**) викликає іншу функцію, яка використовується для створення користувацьких налаштувань, тобто **getSharedPreferences (string name, int mode)**. Тільки тому, що Activity містить лише один файл налаштувань, функція getPreferences **(int mode)** просто передає ім’я класу Activity, щоб створити файл Preferences.

Користувальницькі (Custom) Preference:

1) Розробник повинен використовувати getSharedPreferences (string name, int mode) для власних налаштувань.

2) Використовується у випадках, коли в Activity потрібно більше одного файлу налаштувань/

3) Ім'я файлу налаштувань передається в першому параметрі

Custom Preference можна викликати з будь-якої точки програми з посиланням на Context.

Виклик з об’єкту PreferencesManager

З використанням методу *getDefaultSharedPreferences()* викликається з об’єкту PreferencesManager, щоб отримати загальнодоступне налаштування, яке надає Android.

Всі методи: **getPreferences (), getSharedPreferences**(), *getDefaultSharedPreferences()* повертають екземпляр класу *SharedPreferences*, через який можна отримати відповідне налаштування з використанням наступних методів:

getBoolean(String key, boolean defValue);

getFloat(String key, float defValue);

getInt(String key, int defValue);

getLong(String key, long defValue);

getString(String key, String defValue).

9.3 Mode та його тип у Shared Preference

Для створення Preference на основі Activity або Custom Preference розробник повинен «пройти» режим, щоб повідомити систему про конфіденційність файлу Preference.

Контекст - це абстрактний клас, який використовується для отримання глобальної інформації в ресурсах програми Android, такі як рядки, значення, малюнки тощо. При цьому оголошуються режими та статична кінцева константа в класі Context.

Існує три типи режиму у Custom Preference налаштуванні:

1. Context.MODE_PRIVATE - значення за замовчуванням (Не доступне за межами вашої програми)

2. Context.MODE_WORLD_READABLE - читається іншими додатками

3. Context.MODE_WORLD_WRITEABLE - читати / писати з інших додатків.

MODE_PRIVATE - це режим за замовчуванням. MODE_PRIVATE означає, що коли будь-який файл Preference створюється в приватному режимі, він не буде доступний за межами вашої програми. Це найпоширеніший режим, який використовується.

MODE_WORLD_READABLE - Якщо розробник створює спільний файл Preference, використовуючи режим читання в режимі, тоді його може прочитати кожен, хто знає його ім'я, тому будь-яка інша зовнішня програма може легко читати дані вашого додатка. Цей режим не часто використовується в програмі.

MODE_WORLD_WRITEABLE - Це схоже на режим читання, але з обома видами доступу, тобто читання та запис. Цей режим ніколи не використовується в додатку розробником.

Google рекомендує вказувати ім’я файлу Preference як ім’я пакета програми + бажане ім’я. Наприклад, com.example.lenovo.lab8_priclad.User.

Таким чином, для створення **S**hared Preference необхідно викликати **метод getPreferences (int mode)** або **getSharedPreferences (string name, int mode)** .

Обидві функції повертають об'єкт SharedPreferences для збереження та отриманням значень із файлу Preference.

SharedPreferences - це singleton клас, що означає, що цей клас матиме лише один об’єкт протягом життєвого циклу програми. Однак файлів

налаштувань може бути декілька, тому всі файли налаштувань можна читати та писати за допомогою класу SharedPreferences.

Лабораторна робота №8. Бази даних в Android. Застосування Shared Preference (загальних налаштування)

8.1 Запис та читання даних з Shared Preference.

8.2. Завдання.

8.3 Контрольні запитання.

8.1 Запис та читання даних з Shared Preference

Запис даних до SharedPreferences:

Крок 1:

Для збереження даних у SharedPreferences розробнику потрібно викликати функцію **edit()** класу SharedPreferences, яка повертає об'єкт класу **Editor**.

Крок 2:

Клас **Editor** надає різні функції для збереження примітивних даних. Наприклад, редактор має всі примітивні функції даних, включаючи дані типу String як **putInt (String keyName, int value).**

Формат загальної примітивної функції: **put + Primitive Type name (String keyname, Primitive Type value)**

Крок 3:

Тепер переконайтеся, що ім'я ключа має бути унікальним для будь-яких збережених значень, інакше воно буде замінено. Щоб точно зберегти

значення, які ви вкладаєте в різні примітивні функції, ви повинні викликати функцію **commit () Editor** .

Зчитування даних з SharedPreferences:

Крок 1: Для читання даних спочатку розробник повинен отримати посилання на об'єкт SharedPreferences, викликавши **getPreferences (режим int)** або **getSharedPreferences (ім'я рядка, режим int)** .

Крок 2: Потім розробник може отримувати значення за допомогою об'єкта SharedPreferences, викликаючи різні функції примітивного типу, починаючи з **get + Primitive Type name**. Тут кожна функція має додатковий параметр для кожного примітивного типу як значення за замовчуванням, якщо не знайдено значення, що відповідає переданому ключу.

Наприклад, щоб отримати збережене значення int, просто зверніться до функції **getInt ("GameScore", - 1),** де "GameScore" є ключовим, а -1 - значенням за замовчуванням, якщо для GameScore не було збережено значення.

Видалення даних

Аналогічною для збереження даних є функція **remove(String key)** у **SharedPreferences.Editor** для видалення даних, пов'язаних з ключем, з файлу Preferences.

Щоб очистити всі дані, просто викличте функцію **clear (),** доступну в **SharedPreferences.Editor**. Не забудьте викликати метод **commit(),** щоб зберегти зміни. Тож очищення даних ніколи не видаляє файл Preferences, а робить його порожнім.

Приклад запису та зчитування даних до SharedPreferences:

```
public static final String PREFS_GAME =" com.example.lenovo.GamePlay";
public static final String GAME_SCORE= "GameScore";
//======== Code to save data ==================
SharedPreferences sp = getSharedPreferences(PREFS_GAME ,Context.MODE_PRIVATE);
sp.edit().putInt(GAME_SCORE,100).commit();
//========= Code to get saved/ retrieve data ==============
SharedPreferences sp = getSharedPreferences(PREFS_GAME ,Context.MODE_PRIVATE);
int sc  = sp.getInt(GAME_SCORE,0);
Log.d("GamePlay","achieved score  is "+ sc);
```

8.2 Завдання

За допомогою SharedPreferences реалізуйте збереження даних, електронної пошти та пароля, для входу користувача в систему його пристрою. Таким чином, користувачу не доведеться повторно вводити свої дані для входу щоразу, коли він відкриває програму. Інтерфейс додатку зображено на рисунку 5.1.

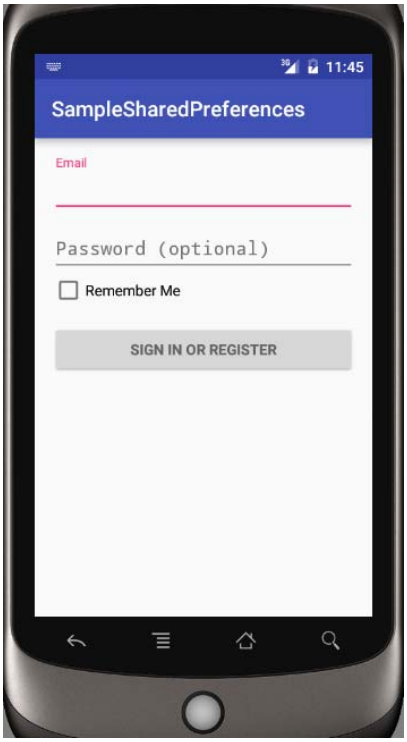


Рис. 5.1 Інтерфейс додатку

8.3 Контрольні запитання.

1. Поняття Shared Preference.
2. Види Shared Preference. Методи отримання доступу.
3. Режим Mode та його тип у Shared Preference.
4. Запис та читання даних з Shared Preference.

Тема 10. Робота з СУБД SQLite.

SQLite - це СУБД побудована на структурованих запитах, з відкритим кодом, невеликим об’ємом. SQLite підтримує вбудовані функції реляційної бази даних.

Для збереження великого обсягу даних, використання sqlite є кращим рішенням, ніж інші системи сховищ, такі як SharedPreferences або збереження даних у файлах.

Android вбудував реалізацію бази даних SQLite. Вона доступна локально та містить дані у текстовому форматі. Вона містить невеликі дані та багатомовний інтерфейс. Отже, для цього не потрібно ніяких процедур адміністрування та налаштування бази даних.

Android OS має власну реалізацію для виконання операцій CRUD (створення, читання, оновлення, видалення), тому Android надає набір класів, доступних у пакетах android.database та android.database.sqlite.

Важлива примітка - створена база даних зберігається в каталозі: data / data / APP_Name / databases / DATABASE_NAME.

В операційній системі Android вже вбудовано деякі баз даних SQLite, які використовуються стандартними програмами – база даних «список контактів», база даних для зберігання фотографій з камери, музичних альбомів тощо.

Основну функціональність для роботи з базами даних надає пакет android.database. Функціональність безпосередньо для роботи з SQLite знаходиться в пакеті android.database.sqlite . База даних в SQLite представлена класом android.database.sqlite.SQLiteDatabase . Він дозволяє виконувати запити до БД, виконувати з нею різні маніпуляції.

Клас android.database.sqlite.SQLiteCursor надає запит та дозволяє повертати набір рядків, які відповідають цьому запиту. Клас android.database.sqlite.SQLiteQueryBuilder дозволяє створювати SQL-запити. Самі sql-вирази представлені класом android.database.sqlite.SQLiteStatement , який дозволяє за допомогою плейсхолдерів вставляти до виразів динамічні дані.

Клас android.database.sqlite.SQLiteOpenHelper дозволяє створити базу даних з усіма таблицями, якщо їх ще не існує.

У SQLite застосовується наступна система типів даних:

- INTEGER: ціле число, аналог типу int в java.
- REAL: число з плаваючою точкою, аналог float та double в java;
- TEXT: набір символів, аналог String та char в java;
- BLOB: представляє масив бінарних даних, наприклад, зображення, аналог типу int в java.

Лабораторна робота №9. Робота із СУБД SQLite

9.1 Створення та оновлення бази даних в Android.

9.2 Запити на додавання, читання, видалення та оновлення операцій в SQLite.

9.3 Завдання.

9.4 Контрольні запитання.

9.1 Створення та оновлення бази даних в Android

Для створення, оновлення та інших операцій вам потрібно створити підклас або клас SQLiteOpenHelper . SQLiteOpenHelper - допоміжний клас

для управління створенням бази даних та управлінням версіями. Він надає два методи onCreate (SQLiteDatabase db), onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion).

SQLiteOpenHelper відповідає за відкриття бази даних, якщо вона існує, створення бази даних, якщо вона не існує, та її оновлення, якщо потрібно. SQLiteOpenHelper вимагає DATABASE_NAME лише для створення бази даних. Після розширення SQLiteOpenHelper вам потрібно буде реалізувати його методи onCreate, onUpgrade та конструктор.

Метод onCreate (SQLiteDatabase sqLiteDatabase) викликається лише один раз протягом життєвого циклу програми. Він буде викликаний щоразу, коли буде перший виклик функції getReadableDatabase () або getWritableDatabase (), доступної в класі SQLiteOpenHelper. Тож клас SQLiteOpenHelper викликає метод onCreate () після створення бази даних та створює екземпляр об'єкта SQLiteDatabase. Ім'я бази даних передається під час виклику конструктора.

onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) викликається лише тоді, коли є оновлення в існуючій версії. Отже, для оновлення версії потрібно збільшити значення змінної версії, яка передається в конструкторі суперкласу.

У методі onUpgrade можна писати запити для виконання будь-якої необхідної дії. Необхідно змінити версію бази даних, якщо додається новий рядок у таблицю бази даних. Якщо є вимога не втрачати наявні дані в таблиці, можна написати запит alter table у методі onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion).

Запит Create: Створення бази даних дуже просто в android за допомогою класу SQLiteOpenHelper. SQLiteOpenHelper - це абстрактний клас із двома абстрактними методами onCreate (SQLiteDatabase db) і onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) та багатьма іншими корисними функціями баз даних. Кожного разу, коли потрібно створити базу даних, необхідно розширити клас SQLiteOpenHelper наступним чином:

```
// Допоміжний клас для виконання запитів, пов’язаних із базою даних
public class SqliteManager extends SQLiteOpenHelper {
public static final String DATABASE_NAME = "abhiandroid.db";
public static final int version = 1;

public SqliteManager(Context context) {
   super(context, DATABASE_NAME, null, version);
  }
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
   String dbQuery = "CREATE TABLE Items (id INTEGER PRIMARY KEY
   AUTOINCREMENT,name TEXT, description TEXT)";
   sqLiteDatabase.execSQL(dbQuery);
  }
@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion,
int newVersion) {
  }
}
```

Якщо потрібно отримати з бази даних будь-яку інформацію, то необхідно застосувати метод rawQuery(), який в якості аргументу приймає SQL-вираз, а також набір значень для sql-виразу. Наприклад, отримання усіх об'єктів з бази даних:

```
SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE, null);
db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");
Cursor query = db.rawQuery("SELECT * FROM users;", null);
if(query.moveToFirst()){

    String name = query.getString(0);
    int age = query.getInt(1);
}
```

9.2 Запити на додавання, читання, видалення та оновлення операцій в SQLite

Під час використання SQLite може існувати два різні способи виконання різних операцій, таких як створення, читання, оновлення та видалення. Один передбачає написання необроблених запитів, а інший використовує параметризовані запити.

Параметризовані запити: це ті запити, які виконуються за допомогою вбудованих функцій для вставки, читання, видалення або оновлення даних. Ці функції, які пов'язані з операціями, надаються в класі SQLiteDatabase.

Необроблені запити: це прості запити sql, подібні до інших баз даних, таких як MySql, сервер Sql тощо, в цьому випадку користувачеві доведеться писати запит як текст і передавати рядок запиту в rawQuery (String sql, String [] selectionArgs) або execSQL (String sql, Object [] bindArgs) для виконання операцій.

Важлива примітка: Документація Android не рекомендує використовувати необроблені запити для виконання операцій вставки, читання, оновлення, видалення. Завжди використовуйте функції вставки, запиту, оновлення, видалення класу SQLiteDatabase.

Нижче наведено приклад необробленого запиту для вставки даних:

```
public void insertItem(Item item) {
  String query = "INSERT INTO " + ItemTable.NAME + " VALUES (0,?,?)";
  SQLiteDatabase db = getWritableDatabase();
  db.execSQL(query, new String[]{item.name, item.description});
  db.close();
}
```

Використовуючи необроблені запити, ніколи не дізнаємось результат роботи, однак із параметризованою функцією запитів повертається значення як у випадку успіху так і при невдалій роботі.

Запит Insert:

Для виконання операції вставки за допомогою параметризованого запиту ми повинні викликати функцію вставки, доступну в класі SQLiteDatabase. Функція insert () має три параметри типу public long:

```
public long insert(String tableName,String  nullColumnHack,
ContentValues values)
```

NullColumnHack може бути передано null. Якщо ми не поміщаємо ім'я стовпця в об'єкт ContentValues, то для цього конкретного стовпця потрібно вставити нульове значення. ContentValues - це ключ- об'єкт на основі пари, який приймає всі примітивні значення типу. Таким чином щоразу, коли дані розміщуються в об'єкті ContentValues, слід знову поставити ім'я стовпця таблиці як ключ, а дані як значення. Функція insert поверне довге ціле (long), тобто кількість доданих рядків, у разі успішного виконання запиту та - 1, в іншому випадку.

Приклад використання запиту insert:

```
/* Елемент - це клас, що представляє будь-який елемент з ідентифікатором, ім'ям та
описом */
public void addItem(Item item) {
  SQLiteDatabase db = getWritableDatabase();
  ContentValues contentValues = new ContentValues();
  contentValues.put("name",item.name);
   // name - column
   contentValues.put("description",item.description);
   // опис - стовпець у таблиці елементів, item.description має значення для опису
  db.insert("Items", null, contentValues);//Items це ім'я таблиці
   db.close();
}
```

Запит Update: функція оновлення дуже схожа на вставку, але вона вимагає два додаткові параметри (whereClause (String) та whereArgs (String [])), але не вимагає nullColumnHack.

```
public int update(String tableName,ContentValues contentValues,String
whereClause,String[] whereArgs)
```

Параметр whereClause – повідомляє базі даних, де оновлювати дані в таблиці, рекомендується передавати знак ? (запитання) разом із назвою стовпця у рядок whereClause. Аналогічним чином масив whereArgs міститиме значення для тих стовпців, навпроти яких були розміщені знаки ? у whereClause. Функція оновлення поверне кількість рядків, які було оновлено, інакше 0.

Приклад використання запиту Update:

```
// Елемент - це клас, що представляє елемент з ідентифікатором, ім'ям
та описом
public void updateItem(Item item) {
  SQLiteDatabase db = getWritableDatabase();
  ContentValues contentValues = new ContentValues();
  contentValues.put("id", item.id);
  contentValues.put("name", item.name);
  contentValues.put("description", item.description);
  String whereClause = "id=?";
  String whereArgs[] = {item.id.toString()};
  db.update("Items", contentValues, whereClause, whereArgs);
}
```

Запит Delete: Подібно до вставки та оновлення, функція видалення доступна в класі SQLiteDatabase, тому видалення дуже схоже на функцію оновлення, крім об'єкта ContentValues, оскільки при видаленні він не потрібний. public int delete (String tableName, String whereClause, String [] whereArgs) функція має три параметри, які повністю схожі на параметри функції оновлення і використовуються так само, як і в функції оновлення.

```
public int delete(String tableName, String whereClause, String [] whereArgs)
```

```
public void deleteItem(Item item) {
SQLiteDatabase db = getWritableDatabase();
String whereClause = "id=?";
String whereArgs[] = {item.id.toString()};
db.delete("Items", whereClause, whereArgs);
}
```

whereClause необов'язковий, передача null призведе до видалення всіх рядків таблиці. Функція delete поверне номер видаленого рядка, якщо whereClause передано, - інакше, поверне 0.

9.3 Завдання

Створити додаток в Android Studio, який проілюструє операції вставки, оновлення, видалення даних з таблиці SQLite. Приклад інтерфейсу додатку (рис. 5.2):

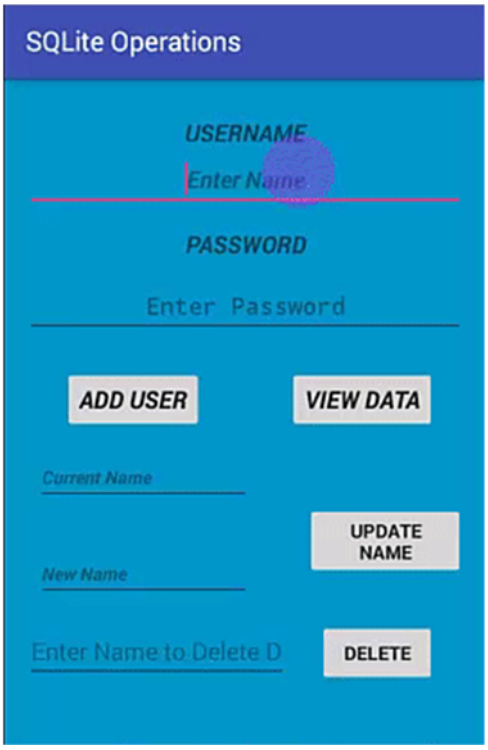


Рис. 5.2. Приклад інтерфейсу додатку

9.4 Контрольні запитання

1. Як працює база даних SQLite?

2. Чому SQLite настільки популярна?

3. За допомогою якого метода можна закрити БД і для чого це роблять?

4. Яку роботу виконує клас SimpleCursorAdapter?

5. Наведіть приклад оновлення та видалення рядків з БД

Тема 11. Використання мережевих сервісів.

Сервіси (служби) представляють собою додатки, які організовано певним чином. На відміну від activity вони не вимагають наявності візуального інтерфейсу. Сервіси дозволяють виконувати довготривалі завдання без втручання користувача. Служби працюють з вищим пріоритетом, ніж неактивні чи невидимі дії, і тому менш імовірно, що система Android припиняє їх роботу. Служби також можна налаштувати на перезапуск, якщо вони зупиняються системою Android, як тільки знову з’явиться достатня кількість системних ресурсів.

Всі сервіси успадковуються від класу Service і проходять такі етапи життєвого циклу:

- метод onCreate(): викликається при створенні сервісу;
- метод onStartCommand(): викликається при отриманні сервісом команди, відправленої за допомогою методу startService();
- метод onBind(): викликається під час закріплення клієнта за сервісом за допомогою методу bindService();
- метод onDestroy(: викликається при завершенні роботи сервісу.

Форми, які може приймати служба:

1) «Запущена». Service називається «запущеною», якщо якийсь компонент додатку запускає її з використанням виклику `startService()`. Після запуску служба може працювати у фоновому режимі протягом необмеженого часу, навіть якщо був знищений компонент, який її запустив. Зазвичай запущена служба виконує одну операцію і не повертає результатів викликаючому компоненту. Коли операція виконана, служба має зупинитися самостійно.

2) *«Прив'язана».* Service називається «прив'язаною», коли компонент програми прив'язується до неї викликом bindService(). Bind-служба пропонує інтерфейс клієнт-серверу, який дозволяє компонентам взаємодіяти зі службою, відправляти запити, отримувати результати та навіть виконувати це між різними процесами. Bind-служба працює до тих пір, поки до неї прив'язаний інший компонент програми. До служби можуть бути прив'язані декілька функцій одночасно, але коли вони скасовують прив'язку, служба знищується.

Хоча, загалом, «запущені» та «пов'язані» служби розглядаються окремо, кожна служба може працювати в обох напрямках - її можна запустити у фоновому режимі (вона може працювати необмежено довго), а також дозволити прив'язку. Справа просто в тому, чи реалізуєте ви пару методів зворотного виклику: onStartCommand(), щоб компоненти могли запускати службу, та onBind(), щоб дозволити прив'язку.

Служба «переднього» плану - це послуга, яка повинна мати той самий пріоритет, що й активна діяльність, і тому не повинна бути знищена системою Android, навіть якщо в системі мало пам'яті. Служба

«переднього» плану повинна подавати повідомлення про рядок стану, яке розміщується під заголовком "Поточний". Це означає, що повідомлення не можна відхилити, якщо послугу не зупинено або не видалено з «переднього» плану. «Фонова» служба виконує операцію, яку користувач не помічає безпосередньо. Наприклад, якщо програма використовує службу для ущільнення свого сховища, це, як правило, є «фоновою» службою.

За замовчуванням служба працює в тому ж процесі, що і основний потік програми. Тому потрібно використовувати асинхронну обробку в службі для виконання ресурсоємних завдань у «фоновому» режимі.

Список використаних джерел

1. Т. О. Мороз, В. С. Ендрес. Переваги гібридних мобільних додатків та прогресивних веб-додатків у бізнесі. Вісник аграрної науки Причорномор'я = Ukrainian Black sea region agrarian science : наук. журн. 2019.N вип. 1. С. 96-102.

2. Java Development Kit. – Режим доступу: https://en.wikipedia.org/wiki/Java_Development_Kit].

3. Разработка мобильных приложений: электронное учеб.-метод. пособие для студентов специальностей 1-40 05 01 03 «Информационные системы и технологии (издательско-полиграфический комплекс)», 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем» / сост. Н. В. Пацей. – Минск : БГТУ, 2020. – 265 с.

4. Дэрси Л. Android за 24 часа. Программирование приложений под операционную систему Google / Дэрси Л., Кондер Ш. – М. : Рид Групп, 2011. – 464 с.

5. Аналіз методів і технологій розробки мобільних додатків для платформи Android : навч. посіб. / О. В. Шматко, А. О. Поляков, В. М. Федорченко. – Хар-ків : НТУ «ХПІ», 2018. – 284 с.

6. Інструкція для розробників під Android. – Режим доступу: https://developer.android.com/guide.

7. Куркин А.В. Программирование под платформу Andriod. Учебное пособие / А.В Куркин. – СПб.: Университет ИТМО, 2015. – 35с.

8. Голощапов А. Л. Google Android: программирование для мобильных устройств / Голощапов А. Л. – СПб. : БХВ-Петербург, 2011. – 448 с.

9. Хашими С. Разработка приложений для Android / С. Хашими, С. Коматинени, Д. Маклин – СПб. : Питер, 2011. – 736 с.

10. Android App Development Tutorial: Beginners Guide With Examples, Code And Tutorials. – Режим доступу: https://abhiandroid.com/.

11. Программирование под Андроид на Java. – Режим доступу: https://metanit.com/java/android/.

12. Харди Б. Программирование под Android. Для профессионалов/ Б. Харди, Б. Филлипс. – СПб.: Питер, 2014. – 592 с.

Приклад тестових завдань

1. Який клас відповідає за всплиаючі підказки?

a) Alert

b) Text

c) Hint

d) Toast

e) Message

2. Що робить властивість

android:backgroundTint = "@android:color/holo_green_light" ?

a) встановлює колір тексту

b) встановлює фон тексту

c) встановлює тінь об’єкту

d) встановлює фон об’єкту

3. Яка мова використовується для Андроід програм?

a) Python

b) C#

c) Java

d) C++

e) JavaScript

4. В чому помилка?

```
btn.setOnClickListener(
        new View.OnClickListener() {
            public void OnClick (View v) {
                btn.setBackgroundTintList(
                    ColorStateList.valueOf(Color.RED)
                );      }       }
```

a) Пропущено @Override

b) Метод OnClick пишеться як onClick

c) Метод OnClickListener пишеться як onClickListener

d) Пропущено @Override та метод OnClick пишеться як Click

5. Якого класу не існує?

a) EditText

b) TextView

c) Button

d) TextEdit

e) MediaPlayer

6. Як можна поміняти тему програми?

a) Такого зробити неможливо

b) Можна змінити в папці values -> styles.xml

c) Можна змінити на пристрої, після установки

d) Виключно прописавши свої стилі і код

7. Які програми потрібні для роботи з Андроїд?

a) Тільки JDK

b) JDK, Android Studio та Visual Studio

c) Тільки Android Studio

d) JDK, Android Studio

8. У яку папку необхідно поміщати звуки?

a) В папку layout

b) В папку drawable

c) В папку raw, яку доведеться створити

d) У будь-яку папку

9. Що таке activity?

a) Це набір всіляких компонентів

b) Це набір тексту і картинок

c) Це набір тексту, картинок і відео

d) Це сховище для фрагментів

e) Це місце для перегляду браузера

10. Яка властивість розтягує елемент на всю ширину екрану?

a) size_parent

b) fill_parent

c) match_parent

d) wrap_parent

e) parent_wrap

11. Навіщо потрібні фрагменти?

a) Дозволяють вбудувати відео

b) Дозволяють вбудувати "підсторінку" з кнопками, картинками і всім іншим

c) Дозволяють вбудувати картинки

d) Це нові сторінки додатка з кнопками, картинками і всім іншим

12 Який метод знаходить об'єкт по id?

a) findId

b) findViewById

c) findViewId

d) findById

e) FindViewID

13. Що виконує наступний код:

Intent intent = new Intent (FirstActivity. This, SecondActivity. Class);

a) Створює прихований намір.

b) Створює неявне намір.

c) Створює явний намір

d) Запускає активність

14. Як називається папка, що містить файл R. java?

a) src

b) res

c) debug

d) gen

15. Папка values містить _____, який визначає значення констант.

a) XML файл

b) Image файл

c) Doc файл

d) нічого з перерахованого.

Контрольні запитання

1. Коротка характеристика програмного забезпечення для розробки мобільних додатків.
2. Файл activity_main.xml (що визначає, які можливості редагування).
3. Поняття activity, зв'язок графічного інтерфейсу з ресурсами.
4. Структура проекту Android Studio.
5. Вміст маніфесту. Кореневий елемент, обов’язкові елементи, основний елемент.
6. Правило визначення ідентифікатору, пояснити на прикладі ("@+id/button").
7. На основі яких об’єктів формується графічний інтерфейс користувача Android додатку.
8. Поняття компоновки (*Layout). Види Layout.*
9. Основні властивості EditText.
10. Основні Властивості CheckBox.
11. Властивості RadioButton.
12. Особливості обробки натискання кнопки з використанням setOnClickListener().
13. Основні методи Toast.
14. Призначення Адаптерів. Характеристика основних видів адаптерів.
15. BaseAdapter в Android.
16. ArrayAdapter та Custom ArrayAdapter.
17. SimpleAdapter, Custom SimpleAdapter.
18. Використання адаптерів в ListView, Spinner.
19. Поняття TabHost, особливості застосування.

20. Основні методи TabSpec. Основні методи TabHost.

21. Основні методи WebView.

22. Взаємодія між фрагментами. Фрагменти в альбомному і портретному режимі.

23. Життєвий цикл фрагменту та типи фрагментів.

24. Поняття та призначення намірів.

25. Поняття «Явного» та «Неявного» намірів.

26. Що таке фільтр Intent. Види фільтрів.

27. Поняття та приклад застосування Shared Preference.

28. Як працює база даних SQLite? Чому SQLite настільки популярна?

29. За допомогою якого метода можна закрити БД і для чого це роблять?

30. Наведіть приклад оновлення та видалення рядків з БД

Навчальне видання

К. Т. Кузьма

кандидат технічних наук,
старший викладач кафедри інформаційних технологій
Миколаївського національного університету
імені В.О. Сухомлинського

ПРОГРАМУВАННЯ МОБІЛЬНИХ ПРИСТРОЇВ

Навчальний посібник
для дистанційного навчання

Формат 60×841/16. Ум. друк. арк. 7,4. Тираж 100 пр. Зам. № 837-548.

Виготовлювач
СПД Румянцева Г. В.
54038, м. Миколаїв, вул. Бузника, 5/1.