МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

В.О. Бречко, С.С. Бульба, О.І. Баленко

Проєктування мобільних застосунків

Навчально-методичний посібник
для студентів спеціальності
123 «Комп'ютерна інженерія»

Затверджено
редакційно-видавничою
радою НТУ «ХПІ»,
протокол № 2 від 27.06.24
поз. 90

Харків
НТУ «ХПІ»
2024

УДК 004.451.057.5+004.58](075.8)

Рецензенти:

Олександр СЕРКОВ., д-р техн. наук, проф. НТУ «ХПІ»,

Катерина ТРУБЧАНІНОВА., д-р техн. наук, проф. Українського державного університету залізничного транспорту

Автори:

Вероніка БРЕЧКО, к-т. техн. наук, доц.;

Сергій БУЛЬБА, к-т. техн. наук, доц.;

Олексій БАЛЕНКО, к-т. техн. наук, доц.

«Проєктування мобільних застосунків», навчально-методичний посібник для студентів спеціальності 123 «Комп’ютерна інженерія». – Харків: НТУ «ХПІ», 2024, 123 стр.

Розглянуто теоретичні основи розробки додатків, призначених для функціонування на мобільних пристроях під управлінням ОС Android, архітектуру та особливості платформи Android, засоби та можливості середовища Android Studio, зокрема, щодо відлагодження розроблюваних програм з використанням емулятора, основи проектування та розробки рішень під платформу Android на мові програмування Java.

ЗМІСТ

ВСТУП

Мобільні застосунки стали невід'ємною частиною сучасного світу, охоплюючи широке коло потреб користувачів, від особистих до бізнес-завдань. Проектування мобільних застосунків – це складний процес, який охоплює дослідження, планування, дизайн, розробку, тестування та впровадження продукту, що відповідає очікуванням користувачів та вимогам ринку.

Перше і найважливіше завдання – це визначення основної мети застосунку та його цільової аудиторії. Це допомагає встановити, які функції та можливості повинні бути включені в застосунок, а також на якому етапі розвитку платформи (iOS, Android або обидві) слід зосередитися. Проектування мобільних застосунків починається з ретельного аналізу ринку та конкурентів. Це дозволяє зрозуміти, які рішення вже існують, які проблеми вони вирішують та які можливості можуть бути впроваджені для покращення користувацького досвіду.

User Experience (UX) та User Interface (UI) дизайн є критично важливими компонентами успішного мобільного застосунку. UX дизайн фокусується на створенні зручного та інтуїтивно зрозумілого інтерфейсу, що забезпечує позитивний досвід користувачів. UI дизайн включає візуальні аспекти застосунку, такі як кольори, шрифти та іконки, які мають бути привабливими та послідовними.

Залежно від аудиторії та функціональності, розробники можуть вибирати між нативною (специфічною для платформи) або кросплатформною (один код для кількох платформ) розробкою. Вибір залежить від багатьох факторів, включаючи бюджет, терміни та вимоги до продуктивності.

Архітектура визначає структуру застосунку, включаючи вибір моделей даних, зв'язок між компонентами, підключення до серверів та API, а також забезпечення безпеки та масштабованості. Добре спроектована архітектура робить застосунок гнучким, легко підтримуваним та масштабованим.

Після завершення етапу проектування починається розробка програмного забезпечення. Цей процес включає написання коду, інтеграцію з бекенд-

сервісами, тестування на різних пристроях та в умовах реального використання. Автоматизоване тестування та залучення реальних користувачів до бета-тестування допомагають виявити та виправити помилки перед запуском.

Запуск застосунку включає публікацію в App Store та Google Play, що супроводжується маркетинговою кампанією для залучення користувачів. Після запуску необхідно забезпечити постійну підтримку та оновлення застосунку, враховуючи відгуки користувачів та зміни на ринку.

Важливо не лише запустити застосунок, але й постійно аналізувати його використання, збирати аналітику та оптимізувати продуктивність і UX на основі реальних даних. Це дозволяє постійно покращувати застосунок і підтримувати його актуальність.

Проектування мобільних застосунків – це багатоступеневий процес, що вимагає врахування багатьох факторів, починаючи від цільової аудиторії та закінчуючи технічними аспектами розробки та підтримки. Успішний мобільний застосунок повинен бути корисним, зручним, продуктивним і постійно вдосконалюватися, щоб відповідати очікуванням користувачів та вимогам ринку.

Розглянуті теми використовуються на практичних заняттях та в лабораторному практикумі за дисципліною «Проєктування мобільних за стосунків» – по кожній темі сформовано ряд індивідуальних практичних завдань, необхідних для закріплення матеріалу. Оформлення звітів виконується згідно загальноприйнятих вимог

1. ОЗНАЙОМЛЕННЯ З ІНСТРУМЕНТАМИ РОЗРОБНИКА ПІД ОС АНДРОЇД

Мета: Навчитись налаштовувати робоче місце розробника, створювати та збирати найпростіші проекти, вивчити структуру проекту.

1.1. ТЕРМІНОЛОГІЯ

Коли ви починаєте розробку під ОС Android, важливо ознайомитися з основною термінологією та інструментами, які використовуються в процесі створення мобільних застосунків. Нижче наведено ключові терміни, що допоможуть вам краще зрозуміти та ефективно використовувати Android-екосистему.

Android Studio – офіційне інтегроване середовище розробки (IDE) для створення Android-застосунків. Воно надає інструменти для написання коду, тестування, відладки та випуску застосунків.

SDK (Software Development Kit) – набір інструментів та бібліотек, необхідних для розробки програмного забезпечення під певну платформу. Android SDK включає в себе компілятори, емулятори, документацію та інші інструменти для створення Android-застосунків.

AVD (Android Virtual Device) – віртуальний пристрій, який емулятор Android використовує для тестування застосунків без необхідності використання фізичного пристрою. Ви можете налаштувати AVD під різні конфігурації (версії Android, роздільну здатність екрану тощо).

Gradle – система автоматизованої збірки, яка використовується в Android Studio для керування проектами. Gradle дозволяє автоматично завантажувати залежності, створювати різні збірки (наприклад, для тестування та релізу), та виконує інші задачі, пов'язані зі збіркою проєкту.

APK (Android Package) – формат файлу, в якому пакуються Android-застосунки для інсталяції на пристрої. APK містить всі необхідні файли, включаючи скомпільований код, ресурси, маніфест і метадані.

Activity (активність) – основний компонент Android-застосунку, який представляє один екран з інтерфейсом користувача. Кожна Activity може виконувати різні завдання, такі як відображення списку, введення даних або показ деталей об’єкта.

XML (Extensible Markup Language) – формат розмітки, який використовується для визначення макетів інтерфейсу користувача, ресурсів (такі як строки, стилі, кольори) та конфігурацій у проектах Android.

Емулятор — віртуальний мобільний пристрій, що працює на вашому комп'ютері.

Ознайомлення з цією термінологією є першим кроком до ефективного використання інструментів розробника під ОС Android. Володіння цими поняттями та інструментами допоможе вам створювати якісні мобільні застосунки, відповідати на вимоги сучасних користувачів та забезпечувати високий рівень продуктивності та надійності вашого програмного забезпечення.

1.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Створення найпростішого проекту

Щоб почати розробку Android-застосунків, ви можете створити найпростіший проект, який дозволить вам ознайомитися з основами роботи в Android Studio. Цей проект буде складатися з одного екрана з простою функціональністю, наприклад, відображенням текстового повідомлення на екрані.

Для створення найпростішого проекту запустіть Android Studio та оберіть пункт "Start a new Android Studio project", рис. 1.

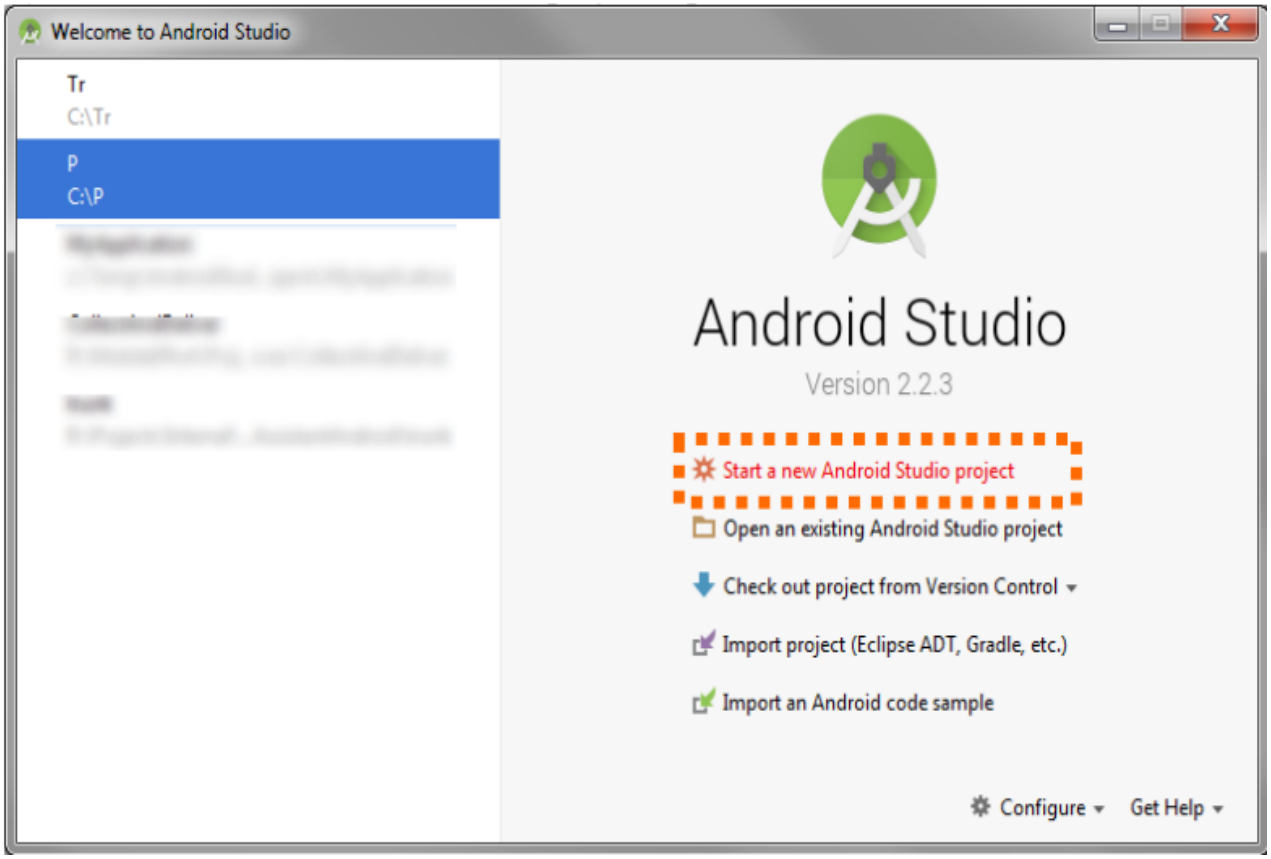

Рисунок 1.1 – Екран привітання Android Studio

На сторінці конфігурації проекту необхідно вибрати ваш бренд у стилі імені сайту. Це ім'я буде переформовано у Java-стиль (слова будуть у зворотньому напрямку). Із полів Company Domain та Application name сформується назва Java-пакету, що повинен бути унікальним серед усіх програм, рис. 2.

При створенні проектів лабораторних робіт задавайте у імені Company Domain otp.ntukhpi.com, та додавайте ваше прізвище із ініціалами як продовження, результатом імені пакету буде наступний рядок:

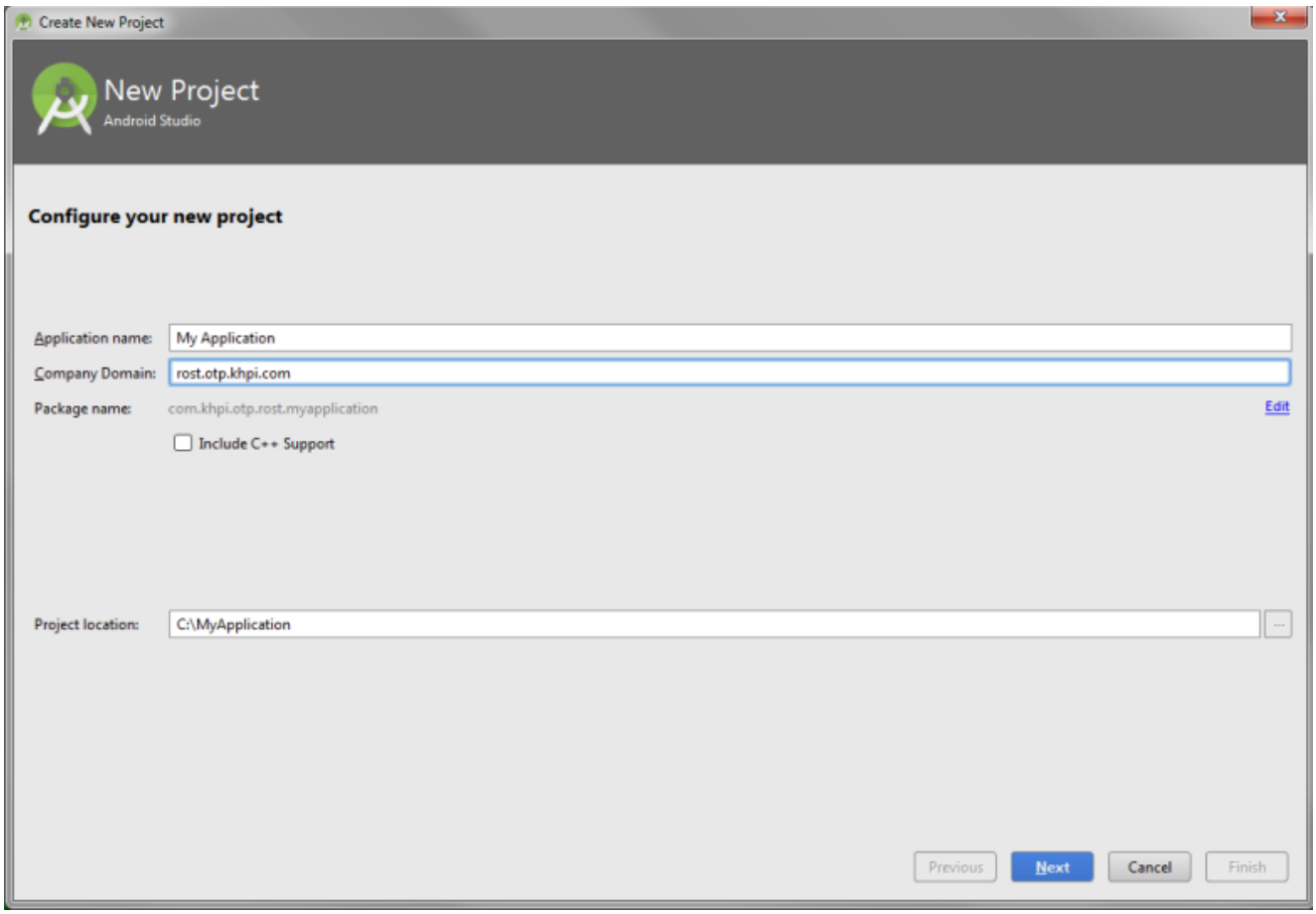

Рисунок 1.2 – Ім'я та розташування проекту

Наступний екран слугує для вибору цільових пристроїв. Це тип пристроїв, на яких планується робота вашого застосунку. На поточний момент середовище розробки пропонує наступні елементи:

- Phone and Tablet – мобільні та планшети;

- Wear – наручні гаджети, годинники, брелоки тощо;

- TV – телевізори;

- Android Auto – версія для автомобілів;

- Glass – окуляри, орієнтовно Google Glass.

В процесі навчання ми обираємо завжди пункт "Phone and Tablet". Версію обираємо керуючись попередньо поданою інформацією щодо вибору версій ОС

та версії API. Зверніть увагу, при виборі різних пунктів вам показуватиметься процент усіх пристроїв, який ви покриваєте, рис. 3.

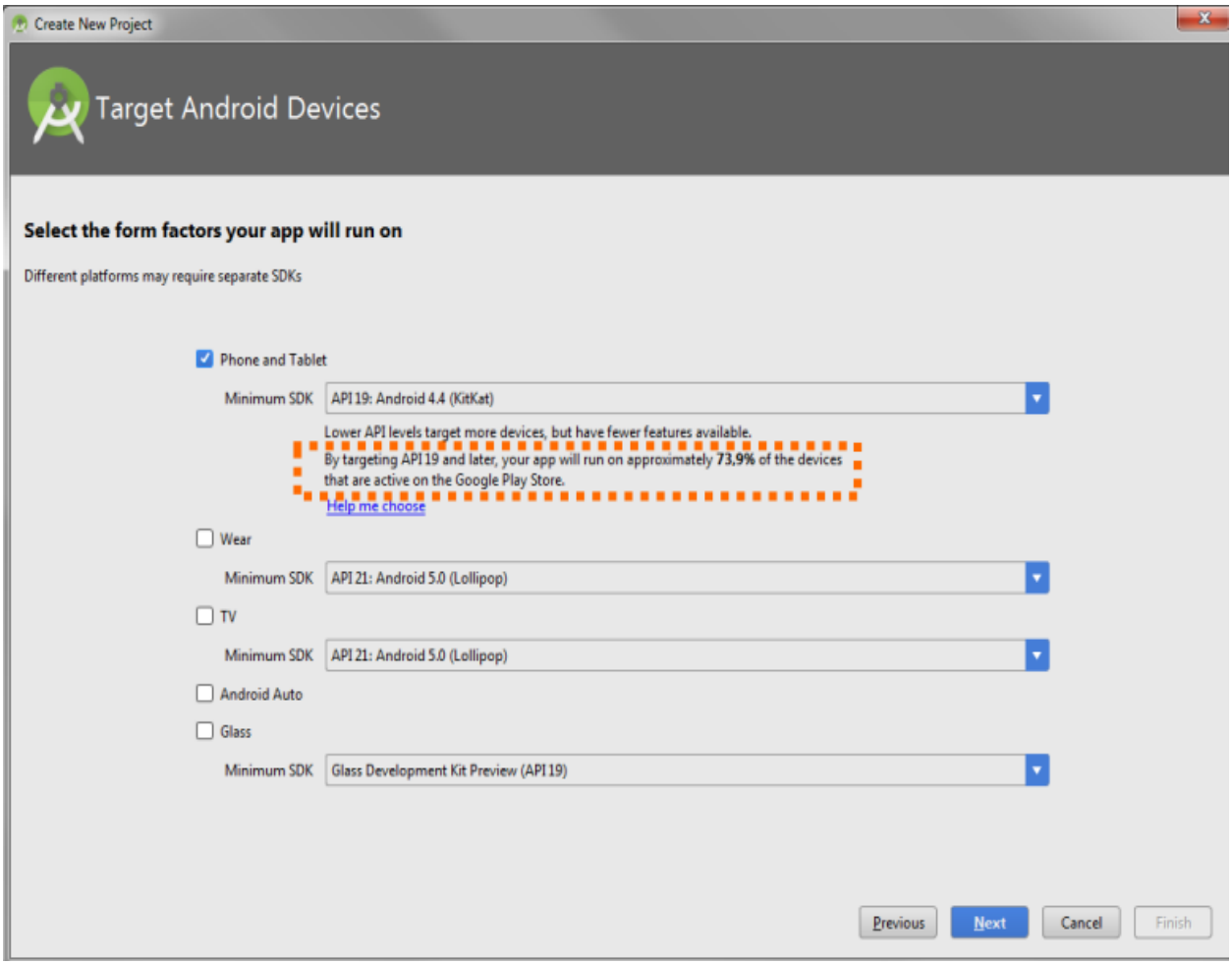

Рисунок 1.3 – Підтримка пристроїв у проекті

На наступному екрані ви обираєте вигляд вашого застосунку. Рекомендуємо обирати один із наступних варіантів:

- Basic Activity – активність має декілька базових елементів;

- Empty Activity – активність без базового функціоналу та елементів управління.

На наступному екрані всі значення залишаємо типовими та натискаємо 'Finish'. Відбувається генерація та базова збірка проекту. В залежності від

потужності вашого ПК це може зайняти до 5 хвилин. Подальші збірки займатимуть набагато менше часу.

У головному вікні середовища розробки зліва відображатиметься дерево проекту, а по центру – візуальний редактор дизайну активності. Ви можете вибрати елементи управління з вікна Palette та перетянути на чисту область у вашій активності.

Структура найпростішого проекту

Структура проекту складається з наступних логічних блоків:

- маніфест;

- тексти програми на мові Java;

- ресурси.

Менеджер емуляторів та запуск існуючого емулятора

Android Studio дозволяє створювати та конфігурувати емулятор, на який згодом буде встановлено написані вами програми.

Для створення емуляторів є спеціальна програма AVD Manager, вона запускається з панелі інструментів Android Studio. Її також можна запустити відокремлено, перейшовши у папку Android SDK та запустивши AVD Manager.exe.

Наприклад, на рис. 4 створено два емулятори. Новий емулятор можна додати за допомогою кнопки "Create Virtual Device…". Запуск існуючого виконується зеленою стрілочкою справа у колонці Actions.

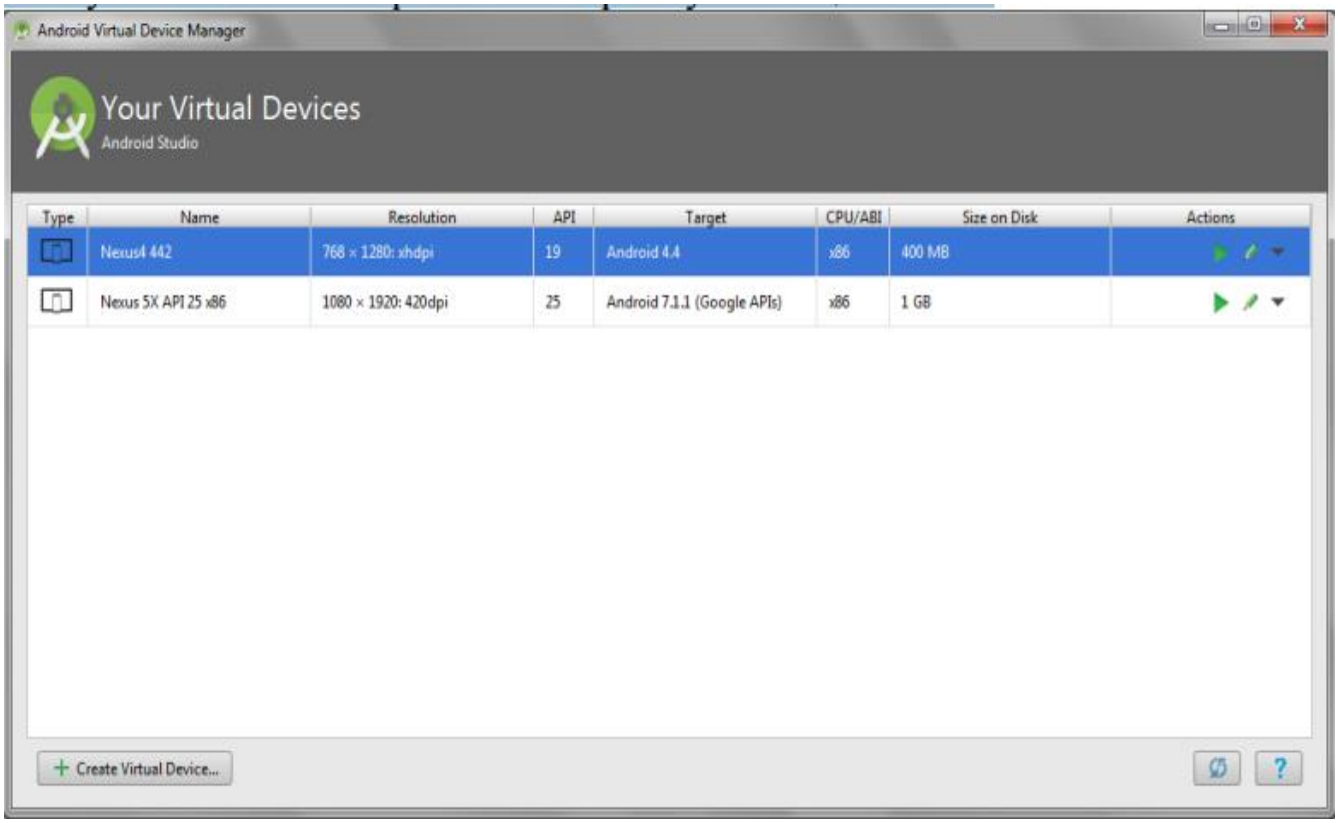

Рисунок 1.4 – Менеджер віртуальних пристроїв з двома емуляторами

Слід зазначити, що після налаштування параметрів відбудеться перший запуск, який зазвичай триває значно довше, ніж наступні. Це пов'язано з початковими налаштуваннями.

У нього є бокова панель для управління діями над пристроєм. Основні кнопки Back, Home, Overview розміщені внизу. За допомогою дотиків мишкою можна управляти емулятором, наче пальцем, взаємодіючи з екраном. Конфігурація емулятора Створюється новий емулятор за допомогою кнопки 'Create Virtual Device…'. Після цього з'являється вікно як показано на рис. 5.

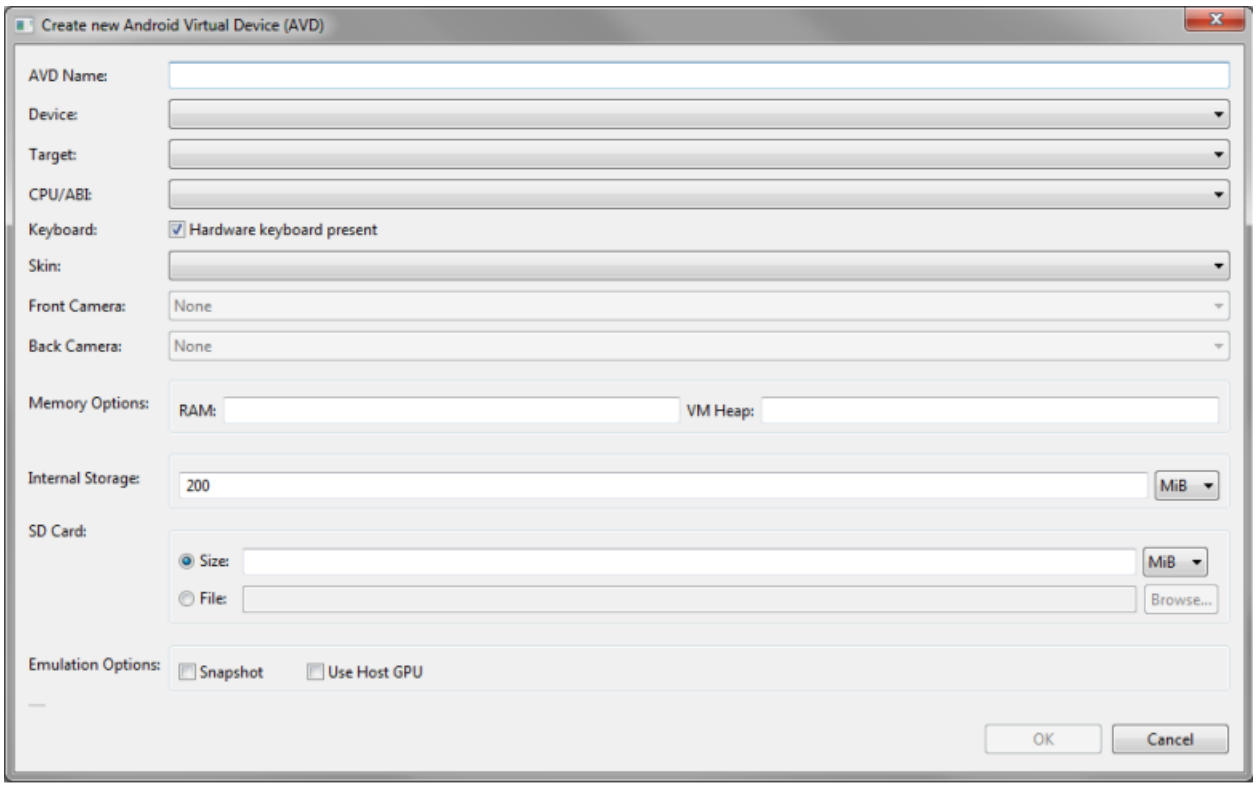

Рисунок 5 – Вікно налаштувань емулятора

- AVD Name – ім'я емулятора, яке відображатиметься у списку;

- Device визначає скоріше зовнішній вигляд та роздільну здатність екрану емулятора;

- Target задає версію операційної системи;

Обов'язково слід зазначити, що попередньо повинен бути завантажений образ емулятора під вибрану версію у секції SDK Manager.

- CPU/ABI відповідає за образ емулятора певної розрядності та архітектури, відповідно до поля Target

- Keyboard. Якщо відмічено, вводити можна з клавіатури, проте екранна клавіатура не з'явиться;

- Skin - картинка навколо екрану емулятора;

- Front/Back Camera - емуляція або підключення до фізичної камери комп'ютера;

- Memory Options, Internal Storage, SD card визначають об'єми пам'яті під відповідні секції;

- Use Host GPU – відмічайте обов'язково, якщо у вас пристойна відеокарта.

Слід зазначити, що вікно налаштувань емулятора згортається до середини і тому тип помилки при заповненні полів може бути невидимим. Розтягніть вікно по висоті.

Запущений емулятор показаний на рис. 6.

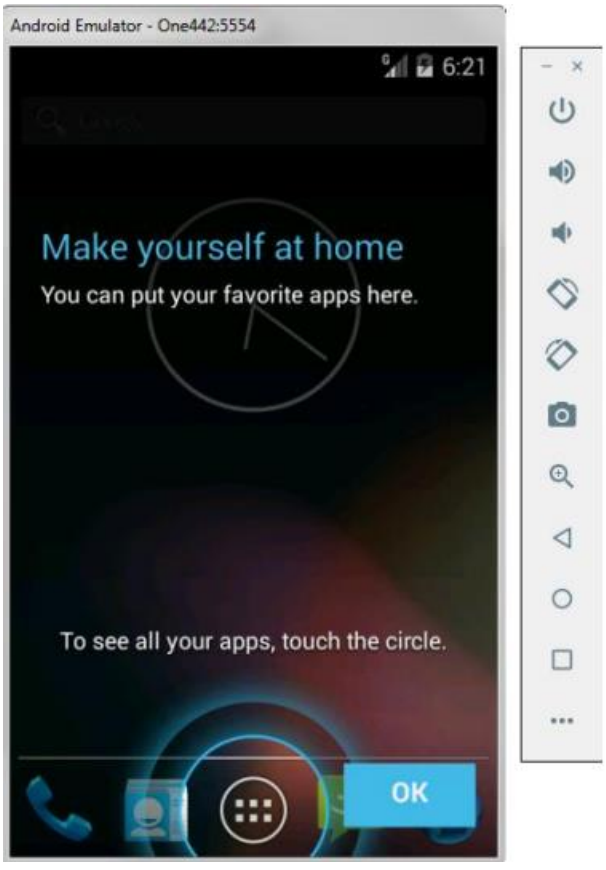

Рисунок 1.6 – Щойно створений емулятор

Запуск проекту на емуляторі

За допомогою меню Run-Run 'app' виконаємо запуск програми на емуляторі. Android Studio нам запропонує вибрати один зі сконфігурованих емуляторів або реальний пристрій, на якому запускатиметься застосунок.

Збірка та інсталяція відбувається аналогічним чином як для пристрою, так і для емулятора – формується APK та встановлюється на емулятор.

Основна активність програми запуститься і програміст, немов користувач пристрою, може переглядати якість реалізованого функціоналу. Мишкою виконується натискання на відповідні елементи управління. Слід зазначити, що ведуть вони себе аналогічно до реальних пристроїв.

Основна структура проекту зображена на рис. 7.

Рисунок 1.7 – Основні елементи структури найпростішого Android-проекту

Опис основних папок і файлів

.idea/ – папка містить конфігураційні файли Android Studio, які використовуються для управління проектом. Зазвичай її вміст не змінюється вручну.

app/ – Головна папка проекту, де знаходяться всі основні файли та каталоги.

build/ – папка, яка містить результати збірки проекту, включаючи зкомпільовані файли, APK, та інші артефакти. Вона автоматично створюється під час збірки проекту.

src/ – папка містить основний код і ресурси проекту.

main/ – основний підкаталог, який включає підпапки java/, res/, та файл AndroidManifest.xml.

java/ – містить файли з вихідним кодом Java або Kotlin.

com/example/myfirstapp/ – простір імен (package), де знаходиться головний клас проекту.

MainActivity.java – файл основної Activity, яка визначає поведінку основного екрану вашого застосунку.

res/ – папка, де зберігаються ресурси застосунку, такі як макети, зображення, текстові рядки та стилі.

drawable/ – містить графічні ресурси (іконки, фони тощо).

layout/ – містить XML-файли, які визначають макети інтерфейсу користувача.

activity_main.xml – макет головного екрана вашого застосунку.

mipmap/ – містить зображення іконок застосунку для різних роздільних здатностей екрана.

values/ – містить XML-файли, які визначають загальні значення, такі як кольори, текстові рядки та стилі.

colors.xml – визначає кольори, які використовуються у вашому застосунку.

strings.xml – Містить текстові рядки для вашого застосунку, які можуть бути локалізовані для різних мов.

themes.xml – Визначає теми (стилі) для вашого застосунку.

AndroidManifest.xml – Ключовий файл, який містить загальну інформацію про застосунок, включаючи оголошення компонентів (Activity, Service), дозволи, потрібні застосунку, та інші метадані.

build.gradle – Скрипт збірки, який використовується для налаштування процесу збірки для модуля app. Тут можна вказувати залежності, конфігурації збірок та інші параметри.

proguard-rules.pro – Файл конфігурації для ProGuard, який використовується для оптимізації та обфускації коду під час випуску застосунку.

build.gradle (поза папкою app) – Скрипт збірки для всього проекту. Він містить налаштування для всього проекту, включаючи репозиторії та залежності.

settings.gradle – Файл, який визначає конфігурацію збірки проекту та список модулів, що входять до складу проекту.

Розуміння структури проекту Android є критично важливим для успішної розробки. Кожен елемент структури має свою роль, від коду та ресурсів до конфігураційних файлів і файлів збірки. Знання того, де знаходяться і як використовувати ці файли, допоможе вам швидше освоїти розробку Android-застосунків та ефективніше керувати вашим проектом.

1.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Що таке Емулятор?

2. У якій папці знаходяться вихідні java-файли?

3. Як називається поле, де можна виводити текст?

4. На якій мові пишуть розмітку елементів для активності?

5. Що міститься у файлі маніфесту?

1.4. ЗАГАЛЬНЕ ЗАВДАННЯ

Створити та запустити проект, що відповідає наступним вимогам:

- ім'я пакету проекту у форматі com.ntukhpi.otp..;

- на екрані емулятора відображається номер вашої групи, ваше прізвище та ініціали.

1.5 ЛІТЕРАТУРА

1. Сайт для розробників під ОС Андроїд [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com]

2. Створення та управління віртуальними пристроями [Режим доступу: https://developer.android.com/studio/run/managing-avds.html]

3. Запуск застосунку на Андроїв емуляторі [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/studio/run/emulator.html]

2. ЕКРАН. РОБОТА З НАЙПРОСТІШИМИ ЕЛЕМЕНТАМИ УПРАВЛІННЯ

Мета: Навчитись користуватись найпростішими елементами управління та передавати дані між активностями

2.1. ТЕРМІНОЛОГІЯ

При розробці Android-застосунків, елементи управління (або UI-компоненти) є основними будівельними блоками інтерфейсу користувача. Нижче наведено основну термінологію, пов'язану з роботою з простими елементами управління в Android.

Графічний інтерфейс користувача (GUI) – це інтерфейс, який дозволяє користувачеві взаємодіяти з програмою через візуальні елементи, такі як кнопки, меню, текстові поля, іконки тощо. GUI надає можливість здійснювати операції за допомогою миші, клавіатури або сенсорного екрану.

View – це базовий клас для всіх елементів інтерфейсу користувача в Android. Всі UI-компоненти, такі як кнопки, текстові поля, зображення тощо, наслідують від класу View.

Layout – це спосіб організації і розміщення елементів View на екрані. Існує кілька типів Layout в Android, наприклад, LinearLayout, RelativeLayout, ConstraintLayout тощо.

OnClickListener – це інтерфейс, який використовується для обробки подій кліку на елементи управління, такі як Button. Використовується для визначення того, що повинно відбуватися, коли користувач натискає на елемент.

Resources – це зовнішні файли, які включають в себе все, що не є кодом Java або Kotlin, але використовується у застосунку, наприклад, макети, рядки, зображення, стилі тощо. Вони зберігаються у відповідній папці res/.

Drawable – це ресурс, який може бути намальований на екрані, наприклад, зображення або фон. Зберігається в папці res/drawable/.

2.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Розробка застосунків на платформі Android включає використання різноманітних елементів управління (UI-елементів), що дозволяють взаємодіяти з користувачем.

Кожен елемент управління має власний набір подій, які можуть виникати під час взаємодії користувача з ним. Наприклад, натискання кнопки викликає подію click, після чого виконується певна функція або серія команд. Також, кожен елемент управління має набір атрибутів (HTML) та властивостей (CSS, JavaScript), які визначають його зовнішній вигляд та поведінку. Наприклад, для кнопки можна встановити атрибути type, disabled, value тощо.

Поля введення часто використовуються для збору інформації, тому важливо забезпечити їхню валідацію, тобто перевірку правильності введених даних до їх відправки на сервер. Валідація може бути як на стороні клієнта (з допомогою JavaScript), так і на стороні сервера.

В сучасних інтерфейсах користувача часто використовуються реактивні підходи, де зміни в одному елементі управління можуть автоматично викликати зміни в іншому. Це забезпечує більш інтерактивний та динамічний досвід для користувача.

Ці основи є фундаментальними для створення ефективних та інтуїтивно зрозумілих користувацьких інтерфейсів. Від того, наскільки добре налаштовані ці елементи, залежить зручність і ефективність роботи користувача з програмою або веб-додатком.

Нижче наведені основні елементи управління та принципи роботи з ними, елементи управління задаються в XML-файлах розмітки або створюються програмно в коді на Java/Kotlin..

1. **TextView** використовується для відображення тексту на екрані.

TextView використовується для відображення тексту. Це простий і часто використовуваний елемент, що дозволяє розміщувати на екрані текстову інформацію. **TextView** може бути використаний для показу статичного тексту,

таких як заголовки, параграфи або інша інформація, яка не потребує введення з боку користувача.

Атрибути, що часто використовуються:

- **text**: текст, що відображається – android:text="Hello, World!
- **textColor**: колір тексту – android:textColor="#FF0000" (червоний колір)
- **textSize**: розмір тексту – android:textSize="18sp"
- **gravity**: визначає вирівнювання тексту всередині TextView (по вертикалі та горизонталі) – `android:gravity="center"`
- **maxLines**: визначає максимальну кількість рядків, які може займати текст – android:maxLines="2"
- **ellipsize**: визначає поведінку при обмеженні кількості рядків або розміру TextView, наприклад, можна додати три крапки (...) в кінці, якщо текст не вміщується – android:ellipsize="end"
- **padding**: визначає внутрішні відступи від країв TextView до тексту – android:padding="16dp"
- **typeface**: Визначає тип шрифту (наприклад, normal, bold, italic) – android:typeface="bold"
- **lineSpacingExtra** і **lineSpacingMultiplier**: визначають інтерліньяж (міжрядковий інтервал) для тексту – `android:lineSpacingExtra="4dp"` (додає 4 dp до міжрядкового інтервалу)

Xml:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Привіт, світ!"
    android:textSize="18sp" />
```

У Java або Kotlin:

```
TextView textView = findViewById(R.id.textView);
textView.setText("Новий текст");
```

2. **EditText** дозволяє користувачу вводити текст.

EditText є розширенням TextView і дозволяє користувачам вводити текст. Це основний компонент для збору текстових даних від користувача.

Атрибути, що часто використовуються:

- **text**: текст, що відображається – android:text="Enter your name"
- **textColor**: колір тексту – android:textColor="#000000" (чорний колір)
- **textSize**: розмір тексту – android:textSize="16sp"
- **hint**: текст-підказка, що відображається, коли поле порожнє – android:hint="Your name"
- **inputType**: тип введення, наприклад, текст, число, пароль тощо – android:inputType="textPersonName" або android:inputType="textPassword"
- **maxLength**: визначає максимальну кількість символів, які можуть бути введені – android:maxLength="50"
- **singleLine**: визначає, чи буде текст введено в один рядок (true) або дозволить багаторядковий ввід (false) – android:singleLine="true"
- **lines**: визначає кількість рядків, що відображаються в полі – android:lines="1"
- **textAlignment**: визначає вирівнювання тексту всередині EditText (ліворуч, по центру, праворуч) – android:textAlignment="center"

Xml:

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

```
    android:hint="Введіть текст" />
```

3. Button використовується для виконання дій при натисканні.

Xml:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
android:text="Натисніть мене" />
```

У Java або Kotlin:

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
       @Override
        public void onClick(View v) {
                // Дія при натисканні кнопки
        }
});
```

Варіанти використання:

- **Форма:** введення текстових даних, таких як ім'я, електронна пошта, адреса.
- **Паролі:** використання типу введення textPassword для прихованого відображення введених символів.
- **Пошук:** поле пошуку, яке підтримує автозаповнення або підказки.

EditText є ключовим елементом для створення інтерактивних форм і забезпечення можливості введення тексту в Android-додатках.

4. CheckBox в Android — це елемент управління, який дозволяє користувачеві вибирати або скасовувати вибір однієї або декількох опцій.

CheckBox зазвичай використовується в тих випадках, коли користувачеві потрібно вибрати декілька опцій із запропонованого списку.

Атрибути, що часто використовуються:

- **checked**: Визначає, чи буде чекбокс обраний (true) чи ні (false) при початковому відображенні – android:checked="false"
- **buttonTint**: визначає колір самого чекбоксу (кнопки) – android:buttonTint="@color/colorAccent"
- **gravity**: визначає вирівнювання тексту відносно чекбоксу – android:gravity="center_vertical"
- **padding**: Визначає відступи навколо тексту або чек боксу – android:padding="8dp"

Xml:

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Погоджуюсь" />
```

У Java або Kotlin:

```
CheckBox checkBox = findViewById(R.id.checkBox);
boolean isChecked = checkBox.isChecked();
```

Варіанти використання:

- **Форми:** використовується в формах для вибору декількох опцій, наприклад, підписка на кілька розсилок.
- **Налаштування:** використовується для ввімкнення або вимкнення певних опцій, таких як включення/виключення сповіщень.
- **Анкети:** у анкетах або опитуваннях для вибору кількох відповідей.

CheckBox є зручним і гнучким елементом, який дозволяє користувачам легко вибирати потрібні опції у ваших Android-додатках.

5. **RadioButton** дозволяє вибирати один варіант з декількох можливих.

RadioButton в Android — це елемент інтерфейсу користувача, який дозволяє вибрати один варіант з кількох доступних. Коли ви використовуєте RadioButton, зазвичай ви поміщаєте його в RadioGroup, який забезпечує, що тільки один RadioButton з групи може бути вибраний в будь-який момент часу.

Xml:

```
<RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/radioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Варіант 1" />

    <RadioButton
        android:id="@+id/radioButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Варіант 2" />
</RadioGroup>
```

У Java або Kotlin:

```
RadioGroup radioGroup = findViewById(R.id.radioGroup);
int selectedId = radioGroup.getCheckedRadioButtonId();
RadioButton radioButton = findViewById(selectedId);
String selectedText = radioButton.getText().toString();
```

При розробці додатків для платформи Android використання базових елементів управління є критично важливим для забезпечення інтуїтивно зрозумілого та зручного інтерфейсу користувача. Використання елементів, таких як TextView, EditText, Button, CheckBox, RadioButton та Spinner, дозволяє створювати додатки, які можуть ефективно взаємодіяти з користувачем, забезпечуючи при цьому високу функціональність та зручність використання.

2.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Для чого потрібні елементи управління?
2. Який елемент управління використовується для вводу тексту?
3. Який елемент управління використовується для виконання дії?
4. Як називається поле у елементі вводу тексту для доступу до тексту?
5. Як типово виглядає елемент рейтинг на активності?

2.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

Створити дві активності та передати між ними дані. Створити відповідні поля для вводу та обробки на першій та другій активності, а також опрацювати механізм передачі даних між ними.

Механізм розрахунку виконується виключно на другій активності.

Завдання вибирається згідно із номером у журналі за формулою:

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Реалізувати калькулятор над цілими числами із операціями "+", "-" над двома числами. На першій активності ввести два числа та вибрати операцію для виконання з випадаючого списку. Друга активність відображає введені на першій числа в двох полях із операцією між ними, а по натисканні на кнопку "Підрахувати" показує результат.

2. Задати поля для фільтрації дати: мінімальну та максимальну дату на першій активності, поле для вибраної дати користувачем та кнопку "Вибрати". По натисканню на кнопку відображається друга активність з якої користувач вибирає дату та натискає кнопку "Вибрати". Вибрана дата відображатиметься у полі вибраної дати першої форми.

3. Реалізувати калькулятор над цілими числами із операціями "*", "/" над двома числами. На першій активності ввести два числа та вибрати операцію для виконання з випадаючого списку. Врахувати, що на нуль ділити не можна. Друга активність відображає введені на першій числа та операцію в рядку, а по натисканні на кнопку "Підрахувати" показує результат.

4. На другій активності порахувати кількість хвилин, що пройшло між двома, вибраними на першій активності, датами. Обробити ситуацію, щоб різниця між датами була позитивна.

5. На першій активності у полях, що доступні лише для читання, введено заздалегідь відомі числа. Біля кожного такого поля є елемент вибору (CheckBox). Програма відображає усі вибрані елементи на другій активності, а після натискання кнопки "Підрахувати" підраховує та зберігає результат для показу на першій активності.

6. На першій активності ввести прізвище, ім'я та по батькові в окремі поля, на другій – вивести отримані дані та виконати скорочення тексту до вигляду прізвище та ініціали через крапку. Передати цей рядок у першу активність та показати у спеціальному полі.

7. Рейтинг встановлюється на підставі отриманої оцінки (вводить користувач) та максимально можливого значення, що виведено у незмінному полі. На другій активності показати рейтинг, що відповідає результату підрахунку та передати його на першу активність.

8. Реалізувати калькулятор на базі двох полів та перемикача (toggle) між операціями "+" та "*". На другій активності виконати підрахунок та передати результат на першу активність.

9. Користувач вводить дату на першій активності та кількість тижнів, що необхідно додати. По натисканню на кнопку дані передаються на другу активність, де виконується підрахунок результату. Результат передається назад у першу активність.

10. Користувач вводить повну назву навчального корпуса та номер аудиторії. На другій активності назва корпуса та аудиторія скорочуються до формату "ВК-313".

2.5. ЛІТЕРАТУРА

1. Запуск іншої активності та передача даних [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/training/basics/firstapp/starting-activity.html]

2. Отримання даних від викликаної активності [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/training/basics/intents/result.html]

3. Елемент вводу випадаючий список (Spinner) [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/topics/ui/controls/spinner.html]

4. Робота та операції з календарем [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/java/util/Calendar.html].

5. Як додавати дні до поточної дати у календарі [Електронний ресурс] / мова: англійська // [Режим доступу: https://www.mkyong.com/java/java-how-to-adddays-to-current-date

3. ЖУРНАЛЮВАННЯ ТА ПОВІДОМЛЕННЯ

Мета: Навчитись використовувати журналювання, ознайомитись з життєвим циклом активностей за допомогою журналювання та навчитись показувати сповіщення користувачеві.

3.1. ТЕРМІНОЛОГІЯ

Журналювання є важливою частиною розробки програмного забезпечення, особливо для налагодження та моніторингу. У Android журналювання дозволяє розробникам отримувати інформацію про те, що відбувається всередині додатка.

Журналювання – запис зневаджувальних текстових повідомлень до якогось сховища для подальшого вивчення.

Сповіщення – допоміжний елемент графічного інтерфейсу системи, що створюється для сповіщення чи взаємодії з користувачем окремо від активностей.

Кнопка "**Огляд"** – використовується для показу переліку кешованих задач.

Кнопка "**Головний екран**" – використовується для переходу на головний екран телефону.

Logcat – інструмент для перегляду журналів.

3.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Журналювання

Журналювання використовується для запису інформації, послідовності дії, проміжних результатів без необхідності підключати зневаджувач до пристрою чи емулятору. Записаний журнал можна переглянути пізніше, передати іншій

особі чи зберегти для подальшого використання. Також Android Studio надає можливість переглядати поточний журнал в реальному часі.

Журналювання виконується за допомогою класу **android.util.Log**, який надає декілька статичних методів. Різниця між цими методами в тому, якого типу інформацію він записує, а саме:

– детальна (Verbose) – **Log.v(…)**

– зневаджувальна (Debug) – **Log.d(…)**

– інформація (Information) – **Log.i(…)**

– попередження (Warning) – **Log.w(…)**

– помилка (Error) – **Log.e(…)**

– критична помилка (Fatal) – **Log.wtf(…)**

Кожен метод має такі параметри:

– тег (Tag) – для пошуку окремих записів в журналі серед багатьох інших,

– повідомлення – текст повідомлення для запису в журнал,

– об’єкт виняткової ситуації для виводу детальної інформації про таку подію.

Перегляд журналу

Вікно для відображення поточного журналу та журналу недавнього запуску відкривається через меню View -> Tool Windows -> Android Monitor, як зображено на рис. 3.1.

Android Monitor був інструментом, який надавав розробникам можливість переглядати журнали, профілювати додатки, а також перевіряти використання ресурсів у реальному часі. Однак, Android Monitor був замінений на Logcat та інші інструменти в Android Studio.

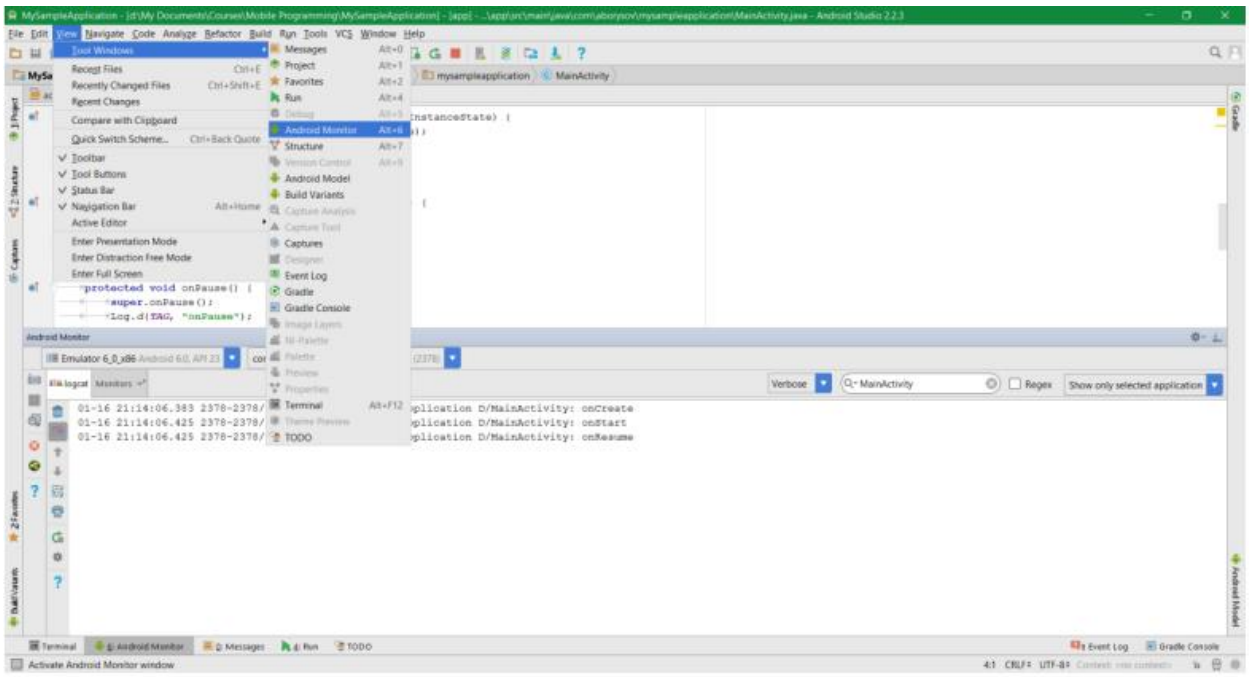

Рисунок 3.1 – Відкриття вікна Android Monitor

У вікні Android Monitor можливо вибрати чи змінити такі параметри:

– пристрій, з якого потрібно отримувати журнал.

– застосунок, який веде запис повідомлень.

– рівень повідомлень для відображення.

– текст для фільтрації повідомлень, наприклад, тег.

Android Monitor як окремий інструмент більше не використовується, але його функціональність була інтегрована в сучасні інструменти Android Studio, такі як Logcat і Android Profiler. Ці інструменти надають розробникам все необхідне для налагодження, профілювання та моніторингу додатків.

Створення спливаючого сповіщення-тосту

Спливаючі повідомлення показуються поверх інших вікон на декілька секунд. Вони не передбачають будь-якої взаємодії з користувачем та використовуються для швидкого некритичного невеликого повідомлення користувачеві про добрий чи поганий результат якоїсь дії.

Повідомлення такого типу створюються за допомогою класу **android.widget.Toast** та його статичного метода **makeText(…)**.

Щоб створити Toast в Android, використовується метод Toast.makeText(). Базовий синтаксис:

```
Toast.makeText(context, text, duration).show();
```

де:

- **context**: Контекст, в якому створюється Toast (наприклад, this або getApplicationContext()).
- **text**: Текст повідомлення, яке потрібно відобразити.
- **duration**: Тривалість показу повідомлення. Може бути або Toast.LENGTH_SHORT (зазвичай 2 секунди), або Toast.LENGTH_LONG (зазвичай 3.5 секунди)

Наступний фрагмент коду створює сповіщення цього типу:

```
Toast.makeText(this, "The file is moved successfully",
                                Toast.LENGTH_SHORT).show();
```

Також можна налаштувати вигляд Toast, змінивши його розташування або додаючи кастомний макет.

Щоб змінити розташування Toast на екрані, використовуйте метод setGravity():

```
Toast toast = Toast.makeText(this, "This is a Toast message",
Toast.LENGTH_SHORT); toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();
```

де:

- **Gravity.CENTER**: місце розташування на екрані (можна використовувати інші значення, такі як Gravity.TOP або Gravity.BOTTOM).
- **0, 0**: зсуви по горизонталі та вертикалі відносно зазначеного місця розташування.

Щоб створити кастомний вигляд для Toast, можна використовувати власний макет XML. Спочатку створюється XML файл для макету (res/layout/custom_toast.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/and
roid"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp"
    android:background="#333333">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_custom_icon" />

    <TextView
        android:id="@+id/toast_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFFFFF" />
</LinearLayout>
```

Далі, використання цього макету для Toast у вашому коді:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
(ViewGroup)
findViewById(R.id.custom_toast_container));
TextView text = layout.findViewById(R.id.toast_text);
```

```
text.setText("This is a custom Toast message");
Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

Toast є зручним і простим способом для короткочасного відображення повідомлень користувачам. Ви можете використовувати стандартний вигляд або створювати кастомні макети для специфічних потреб вашого додатка.

Створення сповіщення через панель статусу (Snackbar)

Snackbar — це компонент інтерфейсу користувача в Android, який показує коротке повідомлення в нижній частині екрана. На відміну від Toast, Snackbar може включати кнопку для взаємодії та підтримує більш складні дії.

Задачею панелі статусу є тимчасовий показ інформації користувачеві та зникнення через певний час або коли проблема зникла. Найбільш простий приклад використання – це спливаюче повідомлення про неможливість приєднання до інтернету.

Панель статусу в Snackbar містить текстову інформацію самого повідомлення та справа розміщається кнопка з дією, якою може скористатись користувач.

Щоб створити Snackbar, потрібно використовувати Snackbar.make(), а потім викликати show() для відображення. Основний синтаксис:

Snackbar.make(view, text, duration).show();

де:

- **view**: основний вид, до якого прив'язується Snackbar. Зазвичай це кореневий вид активності або фрагмента.
- **text**: текст повідомлення, яке потрібно відобразити.

- **duration**: тривалість показу повідомлення. Може бути або Snackbar.LENGTH_SHORT (зазвичай 2 секунди), або Snackbar.LENGTH_LONG (зазвичай 3.5 секунди).

Наступний фрагмент коду створює сповіщення цього типу:

```
Snackbar.make(view, "Couldn't sign in ", Snackbar.LENGTH_LONG)
                                .setAction("SIGN-IN", null).show();
```

Snackbar є потужним інструментом для відображення сповіщень у нижній частині екрана, з можливістю додавання кнопок для взаємодії та кастомізації вигляду. Це корисний компонент для реалізації простих сповіщень, які можуть включати дії, що вимагають участі користувача.

Створення діалогового сповіщення

Діалоги використовуються здебільшого для отримання якогось рішення від користувача, підтвердження дії, вибору з декількох варіантів. Також вони можуть використані для повідомлення про якусь подію, яке користувач перегляне врешті-решт.

Діалоги складаються з наступних елементів:

– заголовок В якості заголовку найчастіше виступають іконка застосунку, його назва або якесь питання чи привід відображення діалогу.

– головний вміст В якості вмісту виступає якийсь текст повідомлення, поле для текстового вводу, список вибору тощо.

– кнопки.

Кожен діалог містить від однієї до трьох кнопок різного призначення. Найчастіше це кнопки підтвердження та відміни якоїсь дії.

Життєвий цикл діалогу співпадає з його батьківською активністю чи до явного закриття його користувачем або розробником.

Слід обов'язково зазначити, що в Андроїд також відміну можна згенерувати натиснувши клавішу "назад" або натиснувши на область поза діалогом.

Для створення діалогу використовується клас **android.app.AlertDialog.Builder** та **android.app.AlertDialog**.

Наприклад:

```
new AlertDialog.Builder(this)
     .setMessage("Чи бажаєш ще тих французьких булок?")
     .setPositiveButton("Бажаю", new OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
         Toast.makeText(MainActivity.this, "Ось ці булки", Toast.LENGTH_SHORT).show();
        }
     })
     .setNegativeButton("Іншим разом", null)
     .show();
```

Приклад додавання нейтральної кнопки (наприклад, "Later"):

```
new AlertDialog.Builder(this)
    .setTitle("Dialog Title")
    .setMessage("Dialog message")
    .setPositiveButton("OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Handle positive button click
        }
    })
    .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
        @Override
```

```
        public void onClick(DialogInterface dialog, int
which) {
            // Handle negative button click
        }
    })
    .setNeutralButton("Later", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int
which) {
            // Handle neutral button click
        }
    })
    .show();
```

AlertDialog є простим і потужним інструментом для створення діалогових вікон в Android. Він підтримує заголовок, повідомлення, кілька кнопок для дій і можливість кастомізації вигляду за допомогою макетів XML. Це дозволяє реалізовувати різноманітні сценарії взаємодії з користувачем в додатку.

Створення діалогових сповіщень у Android може бути здійснено за допомогою AlertDialog, DialogFragment, або кастомних діалогів з власними макетами. Це дозволяє вам реалізовувати різноманітні сценарії взаємодії з користувачем, включаючи підтвердження, введення даних та інші важливі повідомлення.

3.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Як переходити між екранами та застосунками?

2. Які основні методи, що відповідають зміні стану життєвого циклу активності?

3. Які типи інформації доступні в Андроїд при журналюванні?

4. Які параметри передаються в функції журналювання класу android.util.Log?

5. Життєвий цикл активності.

6. За допомогою яких інструментів можна створювати діалогові сповіщкння?

7. Для чого проводиться журналювання?

3.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Створити новий проект з використанням шаблону Empty Activity, використати типові налаштування, задавши ім’я пакету проекту за правилами попередніх робіт.

2. В класі MainActivity перевизначити методи, які відповідають зміні стану життєвого циклу активності, а саме: - onCreate - onStart - onResume - onPause - onStop - onDestroy - onRestart

3. В класі створити константу TAG, яка відповідає імені класу, наприклад:

```
private static final String TAG = MainActivity.class.getName();
```

4. Додати журналювання в кожен перевизначений метод з виводом імені перевизначеного методу, наприклад:

```
/**
* {@inheritDoc}
*/
@Override
protected void onResume()
       { super.onResume();
       Log.d(TAG, "onResume");
}
```

5. Додати на екран кнопку, яка відобразить повідомлення з прізвищем та ім’ям студента-автора відповідно до індивідуального завдання.

6. Запустити проект та взяти результат журналювання життєвого циклу активності відповідно до індивідуального завдання.

7. Зазначити в висновках, які методи життєвого циклу виконались та не виконались.

Завдання вибирається згідно із номером у журналі за формулою

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Запустити застосунок та натиснути системну кнопку **Назад**. В якості повідомлення використати спливаючий діалог з однією кнопкою.

2. Запустити застосунок та натиснути системну кнопку **Головний екран**. В якості повідомлення використати сповіщення-тост (**Toast**).

3. Запустити застосунок та натиснути системну кнопку **Огляд**. В якості повідомлення використати панель статусу (**Snackbar**).

4. Запустити застосунок, натиснути кнопку **Огляд** та закрити застосунок, використавши жест скидання. В якості повідомлення використати спливаючий діалог з однією кнопкою.

5. Запустити застосунок, натиснути кнопку **Огляд** та повернутися до застосунку, вибравши його зображення. В якості повідомлення використати сповіщення-тост (**Toast**).

6. Запустити застосунок, натиснути кнопку **Головний екран**, натиснути кнопку **Огляд** та повернутися до застосунку, вибравши його зображення. В якості повідомлення використати панель статусу (**Snackbar**).

7. Запустити застосунок, натиснути кнопку **Назад**, натиснути кнопку **Огляд** та повернутися до застосунку, вибравши його зображення. В якості повідомлення використати спливаючий діалог з однією кнопкою.

8. Запустити застосунок, натиснути кнопку **Назад** та запустити застосунок знову за допомогою його іконки в списку застосувань. В якості повідомлення використати сповіщення-тост (**Toast**).

9. Запустити застосунок, натиснути кнопку **Головний екран** та запустити застосунок знову за допомогою його іконки в списку застосувань. В якості повідомлення використати панель статусу (**Snackbar**).

10.Запустити застосунок, натиснути кнопку **Огляд**, натиснути кнопку **Головний екран** та запустити застосунок знову за допомогою його іконки в списку застосувань.

3.5. ЛІТЕРАТУРА

1. Як переходити між екранами та додатками [Електронний ресурс] / мова: багатомовний // [Режим доступу: https://support.google.com/nexus/answer/6073614?ref_topic=6126560]

2. Клас android.util.Log [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/android/util/Log.html]

3. Клас android.support.design.widget.Snackbar [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/android/support/design/widget/Snackbar.html]

4. API Guides – Dialogs [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/topics/ui/dialogs.html].

4. РОЗМІТКА ТА РЕСУРСИ

Мета: Навчитись застосовувати контейнери розмітки із використанням спільних ресурсів та застосуванням специфікаторів.

4.1. ТЕРМІНОЛОГІЯ

У Android розмітка та ресурси є основними складовими для створення користувацького інтерфейсу та управління різними аспектами додатка.

Розмітка в XML є основним способом визначення структури користувацького інтерфейсу в Android. Розмітка описує, як виглядає UI елементи (кнопки, текстові поля, списки тощо) і як вони розташовані на екрані.

View — це базовий будівельний блок для UI елементів в Android. Це може бути все, від простих елементів, таких як текстові поля, до більш складних, таких як списки.

ViewGroup є контейнером для інших View і ViewGroup, що дозволяє вам організовувати компоненти інтерфейсу. ViewGroup може розміщувати дочірні елементи в певному порядку.

LayoutInflater — це клас, який використовується для створення (або "надування") View з XML розмітки під час виконання програми.

Ресурси в Android включають в себе все, що не є кодом, але може бути використано у додатку. Це можуть бути зображення, рядки тексту, кольори, розміри, стилі тощо.

Клас R є автоматично згенерованим класом, який містить ідентифікатори для всіх ресурсів вашого додатка. Він дозволяє звертатися до ресурсів у коді за допомогою таких синтаксисів, як `R.layout.main_activity` або `R.drawable.icon`.

4.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

У Android розміщення елементів інтерфейсу (UI) здійснюється за допомогою різних макетів (layouts), які визначають, як елементи будуть розташовані на екрані.

Лінійне розміщення елементів (LinearLayout)

З назви зрозуміло, що це розмітка для лінійного розміщення елементів один відносно одного, у горизонтальному або вертикальному напрямку. В одному напрямку може бути лише один піделемент, наприклад, якщо розмітка вертикальна, то компоненти йдуть у висоту, зверху вниз.

Створіть нову розмітку та виберіть для неї тип LinearLayout або видаліть увесь текст розмітки та внесіть наступний текст:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
      <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Hellow"/>
      <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="My"/>
      <Button
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:text="XPI"/>
      </LinearLayout>
```

За напрямок відповідає атрибут ***android:orientation***, у нас стоїть значення "вертикальний", значить усі елементи розміщатимуться зверху вниз, по одному рядку на кожен елемент.

Всередині тегу LinearLayout в нас міститься один елемент Button, кнопка.

Перемкнувшись на закладку Design ви побачите зліва вигляд вашої кнопки у розмітці LinearLayout. Зверніть увагу, що ширини кнопки у 50 незалежних пікселів недостатньо для коректного відображення нашого тексту. Поставивши 150, розмір кнопки буде прийнятним.

Атрибут ***layout_gravity*** задає межу вирівнювання ("притягування") елементу, наприклад, значення 'left' означатиме вирівнювання по лівій стороні.

Тепер перейдемо до найбільш уживаних значень атрибутів match_parent та wrap_content.

match_parent означає "для батька", тобто розмір елемента необхідно розтягнути до розмірів батьківського.

wrap_content означає "для себе", тобто розмір елемента повинен вирахуватись відносно свого вмісту.

Наприклад, висота кнопки у розмітці задається як wrap_content, а ширину ми встановили 150 dp.

Слід зазначити, що кінцевий елемент у ієрархії повинен визначати свій розмір явним чином.

Якщо замінити орієнтацію розмітки на горизонтальну, ***horizontal***, то наша кнопка Hellow буде знову розтягнута на весь екран, а інших не буде видно. Така ситуація виникла через match_parent у кнопки 'Hellow', якщо ми задамо wrap_content, то усі елементи помістяться в один рядок, а його висота буде взятою по найвищому елементові.

Розміщення елементів RelativeLayout

RelativeLayout дозволяє розміщувати елементи відносно інших елементів або самого контейнера.

Можна використовувати властивості, такі як layout_alignParentTop, layout_below, layout_toRightOf, щоб позиціонувати елементи.

Наведемо наступний **приклад** розмітки:

```
<RelativeLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
>
<Button
      android:text="High"
           adroid:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_alignParentTop="true"
      android:id="@+id/button2"
           android:layout_alignParentStart="true" />
 <Button
           android:text="Low"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:id="@+id/button3"
           android:layout_below="@+id/button2"
           android:layout_alignParentStart="true" />
 <TextView
           android:text="I want center"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_centerVertical="true"
           android:layout_centerHorizontal="true"
           android:layout_toEndOf="@+id/button3"
           android:id="@+id/textView" />
</RelativeLayout>
```

Розберемо отриману структуру в дизайнері. Відображаються дві кнопки та елемент підпис. Причому кнопки розміщені одна відносно одної, а підпис відносно усього контейнеру розмітки.

Основні атрибути вирівнювання говорять самі за себе (типи boolean):

– top, bottom, left, right, center;

– center_vertical, center_horizontal;

– start, end.

Відносно сусідів (вказується ідентифікатор):

– layout_toEndOf, layout_toLeftOf, layout_toRightOf, layout_toStartOf.

В абсолютних позиціях (boolean):

– layout_alignEnd, layout_alignLeft, layout_alignRight, layout_alignStart, layout_alignTop

– layout_alignParentBottom, layout_alignParentEnd, layout_alignParentLeft, layout_alignParentRight, layout_alignParentStart, layout_alignParentTop.

Розміщення елементів ConstraintLayout

ConstraintLayout є більш потужним і гнучким макетом, який дозволяє створювати складніші дизайни, прив'язуючи елементи до інших елементів або меж контейнера.

Він використовує "обмеження" (constraints) для визначення розташування і розмірів елементів.

Він є більш просунутим, ніж інші макети, такі як LinearLayout або RelativeLayout, оскільки дозволяє створювати складні та ефективні інтерфейси користувача без вкладених макетів.

Основні концепції **ConstraintLayout**:

1) **Обмеження (Constraints)** — це правила, які ви задаєте для визначення положення елементів відносно інших елементів або меж контейнера (**parent**). Кожен елемент може бути прив'язаний до інших елементів

або до країв контейнера через верхню, нижню, ліву, праву або центральну частину.

2) **Bias (Зміщення)**: дозволяє зміщувати елемент між двома обмеженнями. Наприклад, якщо елемент має обмеження зліва і справа, то ви можете змістити його ліворуч або праворуч за допомогою **layout_constraintHorizontal_bias**.
3) **Chains (Ланцюжки)** – це група елементів, зв'язаних між собою горизонтально або вертикально. Ланцюжки дозволяють задавати поведінку вирівнювання елементів (наприклад, розподілити простір між елементами рівномірно).
4) **Guidelines (Керівництва)** — це невидима лінія, до якої можна прив'язати елементи. Це корисно для створення макетів, де кілька елементів повинні вирівнюватися по певній осі.

Приклад:

```
<ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Item 1"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <TextView
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Item 2"
```

```
        app:layout_constraintTop_toBottomOf="@id/text1"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />
</ConstraintLayout>
```

Розміщення елементів FrameLayout

FrameLayout призначений для відображення одного елемента, або для накладення елементів один на одного. Основна ідея полягає в тому, що всі додані до **FrameLayout** елементи накладаються один на одного, при цьому перший доданий елемент знаходиться під усіма іншими.

Кожен новий доданий елемент буде розташований поверх попереднього.

FrameLayout ігнорує порожній простір, тобто навіть якщо елемент має властивості **wrap_content** або **match_parent**, якщо його вміст менший, він не розтягуватиметься до розмірів контейнера.

За замовчуванням елементи в **FrameLayout** розташовуються у верхньому лівому куті контейнера. Щоб змінити розташування, можна використовувати властивість **layout_gravity**.

Приклад:

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/background_image" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Overlay Text"
        android:layout_gravity="center" />
```

```
</FrameLayout>
```

Розміщення елементів GridLayout

GridLayout розміщує елементи у вигляді сітки з рядків та стовпців, можна визначити кількість рядків і стовпців і вказати, скільки місця кожен елемент займатиме. Можна вказати кількість рядків за допомогою властивості **rowCount** і кількість стовпців за допомогою **columnCount**. Якщо не вказувати **rowCount** або **columnCount**, GridLayout автоматично визначає кількість рядків або стовпців на основі розміщених елементів.

Кожен елемент у GridLayout займає одну або кілька комірок у сітці. За замовчуванням елементи займають одну комірку, але можна розтягувати їх на кілька комірок, використовуючи атрибути **layout_rowSpan** та **layout_columnSpan**.

Налаштування проміжків між рядками та стовпцями здійснюється за допомогою атрибутів **android:layout_margin** та **android:layout_gravity**.

Приклад:

```
<GridLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2">

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Item 1"
        android:layout_columnSpan="2" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Item 2" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Item 3" />
</GridLayout>
```

Розміщення елементів в Android залежить від вибору макету і використання правильних властивостей для розміщення елементів. Правильний вибір макету та властивостей дозволяє створювати гнучкі, адаптивні та зручні інтерфейси для користувачів.

Підрахунок власних розмірів елемента

Зручно показати такий підрахунок на прикладі кнопки. У кнопки є текст у нього є певний шрифт, а отже, і його висота. Між текстом та рамкою є проміжок, що називається "підшивка" або "набивка" (англ. padding) і не слід забувати про власне сам каркас, що теж має певну товщину.

Схематично залежності для розрахунку вказані на рис. 1.

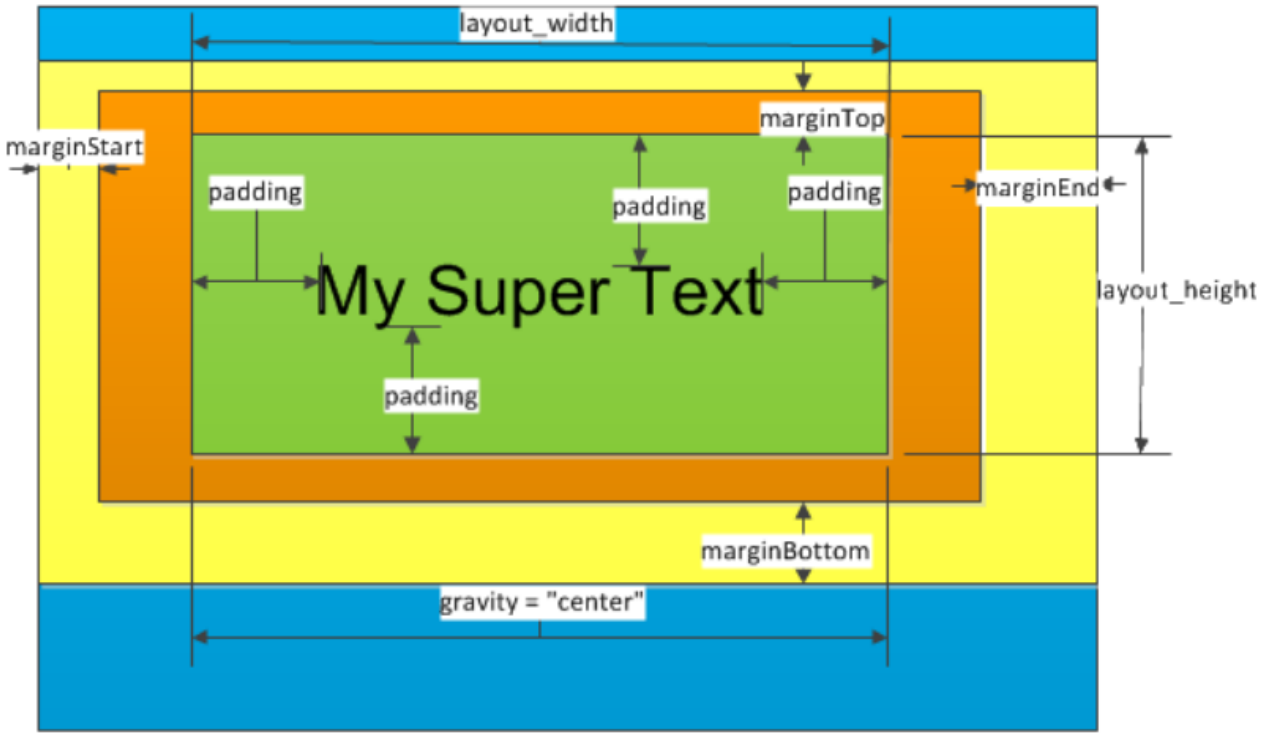

Рисунок 4.1 – Структура складових при підрахунку розміру елемента

Встановлення та відмальовування фону

У якості фону в HTML програмісту знайомо, що можна використовувати картинку, яка розтягується на весь елемент. В Андроїд можна використовувати більш складний варіант – створити власний об'єкт, що дозволяє зображати складні фонові малюнки. Для цього необхідно на папці ***drawable***, натиснувши праву кнопку миші, вибрати ***New***, ***Drawable resource file***. Задаємо логічне ім'я новому примітиву.

Додаємо у тег ***selector*** власний набір примітиву (shape):

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
     <item>
           <shape android:shape="rectangle">
               <stroke android:dashWidth="2dp"
                     android:width="10dp"
                     android:dashGap="5dp" />
               <solid android:color="@android:color/holo_green_dark"/>
               <gradient android:angle="45"
                     android:startColor="@color/colorAccent"
                     android:endColor="@color/colorPrimaryDark"/>
               <corners android:radius="25dp">
           </shape>
     </item>
</selector>
```

Атрибут android:shape відповідає за базовий об'єкт, тег stroke відповідає за переривисту лінію, solid – заливка, а gradient – за створення градієнтної заливки. Скруглення кутів досягається за допомогою тегу corners.

В основній розмітці необхідно для елемента встановити атрибут background. Наприклад:

```
<TextView
          ...
               android:background="@drawable/myShapeName"/>
```

Розмітка та ресурси є основними елементами у створенні користувацького інтерфейсу в Android. Розмітка визначає, як елементи виглядають і розташовані

на екрані, а ресурси включають в себе текст, зображення, кольори та інші дані, які можуть бути використані в додатку. Використання ресурсів дозволяє створювати гнучкі та локалізовані додатки, які легко підтримувати та змінювати.

4.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Для чого використовується LinearLayout?
2. Коли використовується RelativeLayout?
3. Для чого використовуються специфікатори?
4. Які орієнтації екрану ви знаєте?
5. Які ресурси ви знаєте та як їх застосовувати?

4.4. ЗАВДАННЯ

У розмітці активності не повинно бути закодованих чисел та констант. Усі дані повинні братись із ресурів @color/your_id, @drawable/your_id, @dimen/your_id, @string/your_id. Завдання вибирається згідно із номером у журналі за формулою

TaskNumber = (№ mod 10) +1;

1. Показати на екрані три кнопки з градієнтом від червоного до чорного. Задайте читабельний колір тексту. Для книжкової орієнтації кнопки розміщені вертикально, для альбомної – горизонтально.
2. Показати на екрані 2 кнопки з штрихованим контуром. Кнопки повинні бути прив'язаними до верху та до низу екрану відповідно для книжкової орієнтації та до лівої і правої меж для альбомної.
3. Розмістити кнопку по центру екрану з градієнтом в напрямку орієнтації екрану.
4. Створити три кнопки з округленими кутами та розмістити їх по лівій, по правій межах, а третю розмістити по центру. Текст усіх кнопок повинен бути на одній лінії.

5. Для книжкової орієнтації першу кнопку зі округленими краями розмістити вверху по центру, а дві по кутах внизу. Для альбомної орієнтації перша кнопка повинна бути внизу по центру, а дві інші по верхніх кутах.

6. Для режиму ldpi на екрані повинні бути три скруглені кнопки, а для інших режимів кнопки повинні бути прямокутними.

7. Для режиму hdpi на екрані замість двох кнопок повинні бути два елементи TextView. Кнопки повинні бути градієнтні.

8. Дві кнопки зі штрихованим контуром розміщені в ряд одна за одною зліва направо для книжкової орієнтації, а для альбомної орієнтації одна з кнопок повинна мати градієнт.

9. Для альбомного розвороту застосунок має три кнопки висотою в екран. Для книжкового розвороту три кнопки розміщені одна за одною зверху вниз та кожна з кнопок має градієнт.

10.Для режиму xhdpi в альбомній орієнтації три кнопки розміщені горизонтально по центру одна біля одної. Для інших режимів кнопки розміщені на головній діагоналі. Усі кнопки градієнтні.

4.5. БОНУСНІ ЗАВДАННЯ

За одне бонусне завдання, виконане виключно на цьому занятті, викладач додатково нараховує додаткові бонусні бали. Один студент може отримати лише один з бонусів.

1. Підготувати розмітку для чат-вікна як показано на рис. 4.2. Власні повідомлення вирівняні по правому краю, а отримані – по лівому. До 5 (п'яти) бонусних балів.

2. Розмістити кнопки різних кольорів, які розміщені в кутах екрану, рис. 4.3. До 2 (двох) бонусних балів.

3. Змоделювати поличку з книг. Кожна книжка стоїть із написом шириною в одну букву рис. 4.4. До 3 (трьох) бонусних балів.

Рисунок 4.2 – До конусного завдання №1

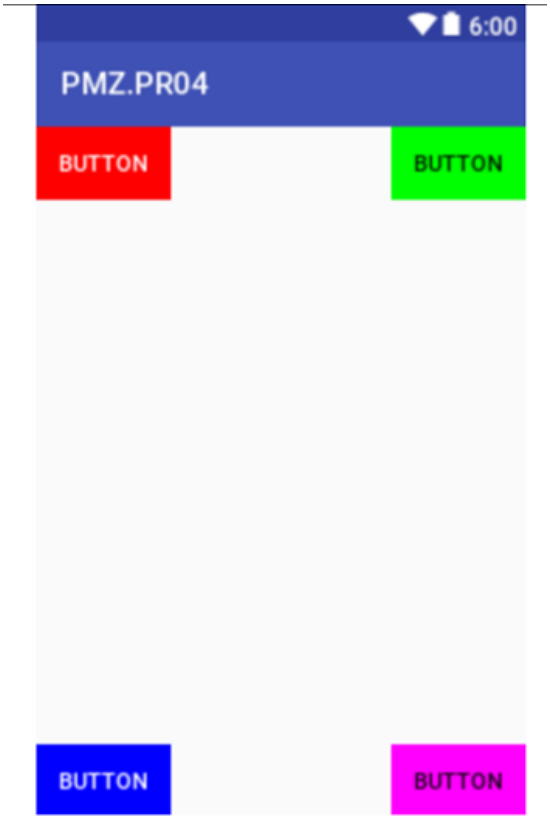

Рисунок 4.3 – До конусного завдання №2

Рисунок 4.4 – До конусного завдання №3

4.6 ЛІТЕРАТУРА

1. Основні означення щодо підтримки різних екранів[Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/practices/screens_support.html#terms].
2. Алгоритм вибору найближчого ресурсу[Електронний ресурс] / мова: // [Режим доступу: https://developer.android.com/guide/topics/resources/providingresources.html#BestMatch]
3. Огляд ресурсів у Android [Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/guide/topics/resources/overview.html].
4. Специфікатори ресурсів[Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/guide/practices/screens_support.html#qualifiers]

5. СПИСКИ, КАРТИНКИ ТА ПРОКРУТКА

Мета: Навчитись відображати власні елементи у списках та використовувати навігаційну шухляду.

5.1. ТЕРМІНОЛОГІЯ

ListView – раніше широко використовуваний компонент для відображення списків. Кожен елемент списку має однакову структуру і може бути будь-яким видом (View), як наприклад

RecyclerView – сучасніший і більш потужний аналог ListView. RecyclerView дозволяє більш гнучко і ефективно управляти списками, особливо коли йдеться про великі набори даних.

Adapter – клас, що підключає дані до елементів ListView або RecyclerView. Він відповідає за створення і прив'язку даних до окремих елементів у списку.

ViewHolder – оптимізація, що використовується в RecyclerView для уникнення повторного пошуку компонентів (наприклад, findViewById). Зберігає посилання на компоненти, які часто використовуються у кожному елементі списку.

ImageView – стандартний компонент для відображення зображень. Підтримує різні формати зображень і дозволяє налаштовувати способи відображення, такі як масштабування і кадрування.

Drawable – абстракція для графічних об'єктів, які можуть бути намальовані на екрані. Це може бути як простий колір, так і складна графіка, включаючи растрові зображення, векторну графіку або анімації.

Glide / Picasso – бібліотеки для завантаження і кешування зображень в Android. Вони автоматично обробляють завантаження, масштабування і кешування зображень, що зменшує кількість пам'яті, яку використовує додаток, і прискорює завантаження зображень.

Навігаційна шухляда (Navigation Drawer) – показує основні параметри навігації на екрані ліворуч.

ScrollView – контейнер, що забезпечує прокрутку вмісту вертикально. Може містити лише один дочірній елемент, який, в свою чергу, може містити інші компоненти.

HorizontalScrollView – контейнер для горизонтальної прокрутки. Працює аналогічно ScrollView, але прокручує вміст по горизонталі.

NestedScrollView – покращена версія ScrollView, яка підтримує вкладену прокрутку, що дозволяє краще управляти взаємодією прокрутки з іншими елементами.

5.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

ListView

ListView — це один із найстаріших і найвідоміших компонентів користувацького інтерфейсу в Android, який використовується для відображення прокручуваного списку елементів. Кожен елемент у списку може бути відносно простим, наприклад, текстом, або складнішим, наприклад, текстом із зображенням.

Adapter — це клас, що підключає дані до елементів ListView. Він відповідає за створення представлення (views) для кожного елемента списку та заповнення їх даними.

Основні типи адаптерів:

- **ArrayAdapter**: Використовується для простих списків, де дані представлені у вигляді масиву або списку.
- **SimpleAdapter**: Більш гнучкий, дозволяє використовувати дані у вигляді списку об’єктів Map.
- **CursorAdapter**: Використовується для відображення даних з бази даних SQLite, які отримуються через Cursor.

View Recycling (Переробка представлень). Коли ListView має великий обсяг даних, важливо оптимізувати використання пам'яті. ListView

використовує техніку під назвою **view recycling**.. Коли елемент списку прокручується за межі екрана, його представлення (view) не видаляється, а переробляється для нового елемента списку, який з'являється на екрані. Це значно покращує продуктивність, особливо у великих списках.

ViewHolder Pattern (Патерн ViewHolder). **ViewHolder** — це патерн, який використовується для зменшення кількості викликів методу findViewById у ListView. Ідея полягає в тому, щоб створити клас ViewHolder, який буде зберігати посилання на всі компоненти представлення одного елемента списку (наприклад, TextView, ImageView тощо). Це знижує витрати на пошук компонентів під час прокрутки.

Customization (Налаштування). Хоча ListView може бути дуже простим у використанні, його можна налаштувати для відображення більш складних макетів для кожного елемента списку. Це може включати додавання зображень, кнопок або інших елементів інтерфейсу. Створення власного адаптера (шляхом розширення класу BaseAdapter) дозволяє мати повний контроль над тим, як кожен елемент виглядає і як він взаємодіє з користувачем.

OnItemClickListener – це інтерфейс, який використовується для обробки кліків на елементах списку. Дозволяє визначити, що відбувається, коли користувач натискає на певний елемент у списку.

Приклад використання ListView. Розглянемо простий приклад, де ListView відображає список текстових елементів:

```
// Дані для списку
String[] values = new String[] {"Item 1", "Item 2", "Item 3", "Item 4", "Item 5"};

// Знайдіть ListView у макеті
ListView listView = (ListView) findViewById(R.id.listView);

// Створення адаптера для підключення даних
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
    this,                    // Контекст
```

```
        android.R.layout.simple_list_item_1,  //  Стандартний  макет
елемента списку
        values                    // Дані для адаптера
    );

    // Підключення адаптера до ListView
    listView.setAdapter(adapter);

    // Обробка кліків на елементах списку
    listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
        @Override
        public  void  onItemClick(AdapterView<?>  parent,  View  view,
int position, long id) {
            String            item            =            (String)
parent.getItemAtPosition(position);
            Toast.makeText(getApplicationContext(),  "Clicked:  "  +
item, Toast.LENGTH_SHORT).show();
        }
    });
```

Незважаючи на свою популярність, ListView має деякі **обмеження**:

- **Обмежена гнучкість**. ListView обмежений у можливостях розміщення елементів. Наприклад, він не підтримує різні макети для різних елементів списку без складного налаштування адаптера.

- **Відсутність ефективної переробки (view recycling) для складних списків.** У порівнянні з RecyclerView, ListView менш ефективний при роботі з великими наборами даних або складними макетами.

- **Відсутність підтримки анімацій і динамічних змін списку**. RecyclerView пропонує більш гнучкі можливості для управління анімаціями і динамічними змінами даних у списку.

Навігаційна шухляда (Navigation Drawer) Навігаційна шухляда показує основні параметри навігації на екрані ліворуч. Вона захована, але розкривається, коли користувач проводить пальцем від лівого краю екрана, або використовує "Hamburger Menu". Готовий до використання варіант на основі елемента NavigationView, надає шаблон Navigation Drawer Activity. Фрагмент activity_main.xml:

```
<android.support.design.widget.NavigationView
      android:id="@+id/nav_view"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:layout_gravity="start"
      android:fitsSystemWindows="true"
      app:headerLayout="@layout/nav_header_main"
      app:menu="@menu/activity_main_drawer" />
```

Тег **NavigationView** в атрибуті **app:headerLayout** містить посилання на розмітку верхній частини панелі в файлі **layout/nav_header_main.xml**. Фрагмент nav_header_main.xml:

```
<ImageView
      android:id="@+id/imageView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:paddingTop="@dimen/nav_header_vertical_spacing"
      app:srcCompat="@mipmap/bird" />
<TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:paddingTop="@dimen/nav_header_vertical_spacing"
      android:text="Android Studio"
     android:textAppearance="@style/TextAppearance.AppCompat.Body1"
     />
<TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
```

```
      android:layout_height="wrap_content"
      android:text="android.studio@android.com" />
```

В атрибуті app:menu – посилання на ресурс меню menu/activity_main_drawer.xml. Фрагмент activity_main_drawer.xml:

```
<group android:checkableBehavior="single">
   <item
      android:id="@+id/sort1"
      android:icon="@android:drawable/ic_menu_sort_alphabetically"
     android:title="Sort 1" />
   <item
      android:id="@+id/sort2"
      android:icon="@android:drawable/ic_menu_sort_by_size"
      android:title="Sort 2" />
</group>
<item android:title="Scroll">
   <menu>
      <item
         android:id="@+id/command1"
         android:icon="@android:drawable/ic_media_previous"
         android:title="Up" />
      <item
         android:id="@+id/command2"
         android:icon="@android:drawable/ic_media_next"
         android:title="Down" />
   </menu>
</item>
```

Використовуючи вищезазначені ресурси, отримуємо навігаційну шухляду як показано на рис. 1.

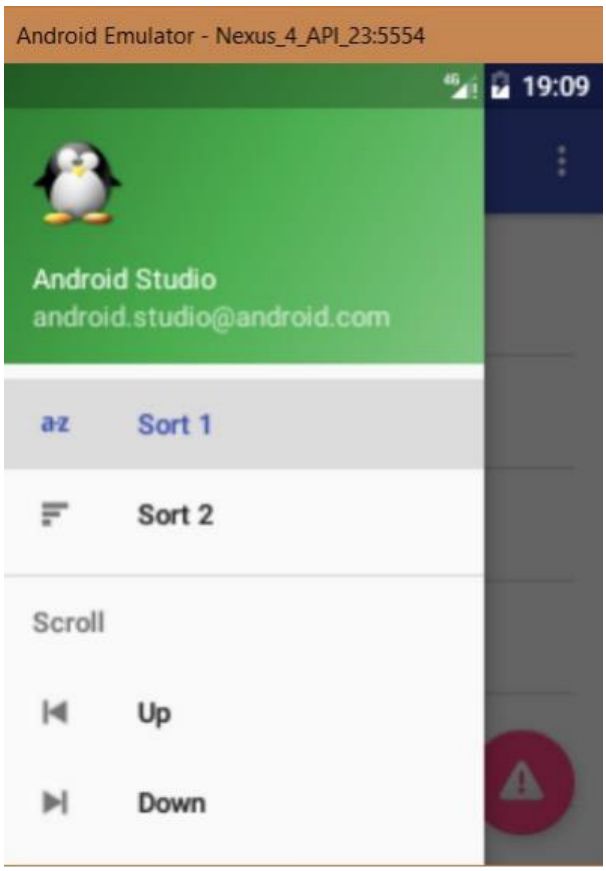

Рисунок 5.1 – Приклад власної навігаційної шухляди

Для обробки пунктів меню в класі активності MainActivity необхідно реалізувати інтерфейс OnNavigationItemSelectedListener:

```
public class MainActivity extends AppCompatActivity
      implements NavigationView.OnNavigationItemSelectedListener {
     ...
      @Override
      public boolean onNavigationItemSelected(MenuItem item) {
      // Handle navigation view item clicks here.
      ...
    }
    }
```

Екранний елемент ListView

Зручний елемент екрану (віджет) ListView часто використовується для подання переліку елементів із прокруткою. Для використання необхідно розмістити сам елемент, а потім заповнити його елементами.

Наведемо XML-код для елемента управління:

```
<ListView
     android:id="@+id/listView"
     android:layout_width="match_parent"
     android:layout_height="wrap_content" >
</ListView>
```

Візьмемо посилання на об'єкт нашого списку у методі onCreate() активності:

```
...
ListView listView = (ListView)findViewById(R.id.listView);
...
```

Компоненту ListView потрібні дані для наповнення. Джерелом наповнення можуть бути масиви, бази даних:

```
final String[] items = new String[] {
     "Item1", "Item2", "Item3"
};
```

Щоб адаптувати різноформатні дані до необхідного вигляду використовується адаптер:

```
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
     android.R.layout.simple_list_item_1, items);
listView.setAdapter(adapter);
...
```

Адаптер забезпечує заповнення списку даними і створюється за допомогою конструкції виду:

```
new ArrayAdapter(Context context, int textViewResourceId, String[]
objects);
```

context – поточний контекст;

textViewResourceId – ідентифікатор ресурсу з розміткою для кожного рядка; можна використовувати системну розмітку з ідентифікатором android.R.layout.simple_list_item_1 або створити власну розмітку;

objects – масив рядків.

Метод ListView.setAdapter() пов'язує підготовлений список з адаптером.

Примітка: з професійної точки зору усі дані краще зберігати у ресурсах:

```
<string-array name="list_items">
      <item>Item1</item>
      <item>Item2</item>
      <item>Item3</item>
</string-array>
```

Для отримання масиву елементів використовують такий Java-код:

```
String[] items =getResources().getStringArray(R.array.list_items);
```

Розмітка елемента у списку

Доступна системна розмітка зі заздалегідь налаштованими параметрами:

– android.R.layout.simple_list_item_1 – один TextView;

– android.R.layout.simple_list_item_2 – два TextView;

– android.R.layout.simple_list_item_checked – CheckedTextView з галочкою;

– android.R.layout.activity_list_item – ліворуч від TextView знаходиться значок ImageView з ідентифікатором android.resource.id.Icon.

Можна визначити власний шаблон для елемента списку. Наприклад, файл "item.xml":

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

 <ImageView
     android:layout_width="@dimen/picture_width"
     android:layout_height="@dimen/picture_height"
```

```
      android:layout_gravity="center_vertical"
      android:src="@mipmap/smile" />

<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">

      <TextView
            android:id="@+id/text1Name"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="@string/text1"
                  android:textSize="@dimen/text1Size"
                  android:textStyle="bold"
                  android:layout_marginLeft="@dimen/margin" />

     <TextView
            android:id="@+id/text2Name"
            android:layout_width="wrap_content"
                android:layout_height="wrap_content"
            android:text="@string/text2"
                 android:textSize="@dimen/text2Size"
            android:layout_marginLeft="@dimen/margin" />
      </LinearLayout>
</LinearLayout>
```

Результат такої розмітки наведено на рис. 5.2.

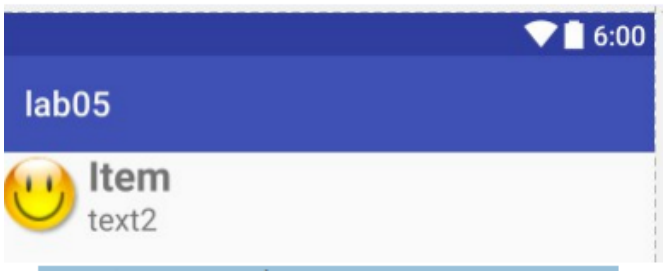

Рисунок 5.2 – Розмітка елемента списку

Текст і зображення будуть визначені в Java-коді. Тоді для елемента ListView, визначеного в content_main.xml:

```
<ListView
      android:id="@+id/list"
      android:layout_height="wrap_content"
      android:layout_width="match_parent">
</ListView>
```

після заповнення списку даними, активність може виглядати, наприклад так, як вказано на рис. 5.3.

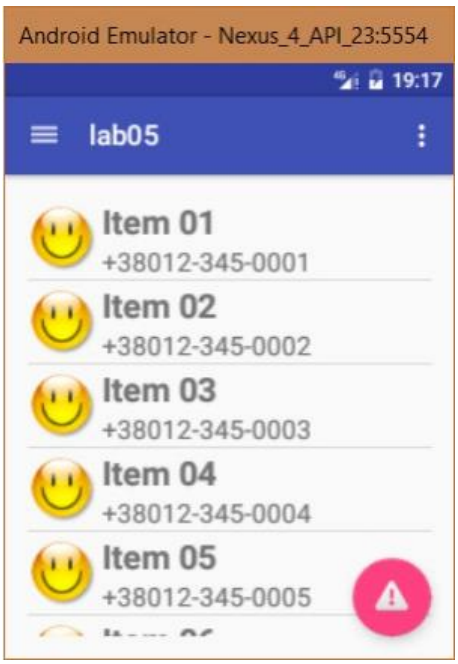

Рисунок 5.3 – Приклад реалізованого списку

Для представлення даних використовувався клас:

```
public class DataEntity {
      private String name;
      private String phone;

     public DataEntity(String name, String phone) {
          this.name = name;
          this.phone = phone;
      }
      public String getName() {
```

```
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

Адаптер:

```
public class DataAdapter extends ArrayAdapter<DataEntity> {
    int resource;

    public DataAdapter(Context context, int resource,
List<DataEntity> items) {
        super(context, resource, items);
        this.resource = resource;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup
parent) {
        LinearLayout itemView;
        DataEntity item = getItem(position);
        if (convertView == null) {
            itemView = new LinearLayout(getContext());
            String inflater = Context.LAYOUT_INFLATER_SERVICE;
            LayoutInflater vi;
            vi = (LayoutInflater)
    getContext().getSystemService(inflater);
            vi.inflate(resource, itemView, true);
        } else {
```

```
            itemView = (LinearLayout) convertView;
        }
        TextView itemText1 = (TextView) itemView
             .findViewById(R.id.text1Name);
        TextView itemText2 = (TextView) itemView
             .findViewById(R.id.text2Name);
        itemText1.setText(item.getName());
        itemText2.setText(item.getPhone());
        return itemView;
    }
}
```

Для заповнення списку ListView listView даними з колекції List list у методі MainActivity#onCreate використовувався код:

```
listView = (ListView) findViewById(R.id.list);
final DataAdapter adapter = new DataAdapter(this, R.layout.item, list);
listView.setAdapter(adapter);
```

При виборі одного з пунктів списку користувачем виконується код методу onItemClick() класу AdapterView.OnItemClickListener:

```
listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent,
                View view, int position, long id) {
        Toast.makeText(MainActivity.this, String.valueOf(id),
            Toast.LENGTH_SHORT).show();
    }
});
```

Для створення в застосунках складних списків і карток за допомогою стилів Material Design зручно використовувати віджети RecyclerView и CardView [4.]

Огляд адаптерів

Інтерфейс Adapter описує базові методи, які повинні містити адаптери: getCount(), getItem(), getView().

Інтерфейс ListAdapter реалізується адаптером, що використовується у ListView (метод setAdapter) та містить опис методів для роботи з роздільниками (separator) списку.

Інтерфейс **SpinnerAdapter** містить метод getDropDownView, який повертає елемент випадаючого списку.

Інтерфейс **WrapperListAdapter** використовується з вкладеними адаптерами. Метод getWrappedAdapter дозволяє витягти з основного адаптера вкладений.

Клас **HeaderViewListAdapter** – готовий адаптер для роботи з Header і Footer. Містить ще один адаптер (ListAdapter), до якого можна отримати доступ за допомогою методу getWrappedAdapter інтерфейсу WrapperListAdapter.

Абстрактний клас **BaseAdapter** спадкоємцям залишає на реалізацію методи getView(), getItemId(), getItem(), getCount() з ListAdapter. Тобто зручно використовувати для створення власного адаптера.

Клас **ArrayAdapter** приймає список або масив об'єктів і вставляє рядок у TextView. Містить методи add, insert, remove, sort, clear і метод setDropDownViewResource для приєднання layout-ресурсу, що призначений для відображення пунктів випадаючого списку.

Клас **SimpleAdapter** – готовий адаптер, обробляє список Map-об'єктів, де кожен Map-об'єкт – це список атрибутів. У масиві to[] вказуємо id елементів екрану, а в масиві from[] – ключі з об'єктів Map, значення яких будуть вставлені у відповідні масиву from[] елементи екрану.

Якщо в ListView кожен пункт списку містить декілька TextView, зручно використовувати SimpleAdapter. Крім того, SimpleAdapter містить методи по наповненню View-елементів значеннями з Map – setViewImage, setViewText, setViewBinder. Метод setViewBinder дозволяє написати свій парсер значень з

Map до View-елементів. Також містить реалізацію методу setDropDownViewResource.

Абстрактний клас **CursorAdapter** реалізує абстрактні методи класу BaseAdapter, містить свої методи для роботи з курсором і залишає спадкоємцям методи для створення і наповнення View: newView, bindView.

Абстрактний клас **ResourceCursorAdapter** містить методи для налаштування використовуваних адаптером layout-ресурсів. Реалізує метод newView з CursorAdapter.

Клас **SimpleCursorAdapter** – готовий адаптер, схожий на SimpleAdapter, тільки використовує список об'єктів Cursor, замість Map, тобто набір рядків з полями. Відповідно в масиві from[] визначені найменування полів, значення яких потрібно використовувати у відповідних View з масиву to[]. Містить метод convertToString, що повертає строкове значення стовпця, який задається методом setStringConversionColumn. Або можна створити власний конвертер методом setCursorToStringConverter і адаптер буде використовувати його при виклику convertToString.

Готові адаптери – HeaderViewListAdapter, ArrayAdapter, SimpleAdapter, SimpleCursorAdapter. Для обробки масиву рядків зручно використовувати ArrayAdapter. Якщо в список потрібно додати дані з БД і є курсор, зручно використовувати SimpleCursorAdapter.

Якщо готові адаптери не підходять для вирішення будь-якої задачі, набір абстрактних класів і інтерфейсів допоможуть створити власний адаптер.

Для роботи з деревом ExpandableListView використовується аналогічна ієрархія адаптерів – ExpandableListAdapter, BaseExpandableListAdapter і т.д.

5.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Для чого використовується навігаційна шухляда?
2. Для чого зручно використовувати клас ArrayAdapter?
3. Як застосовувати клас SimpleAdapter?
4. Як розробити шаблон для елементів ListView?

5. Охарактеризуйте готові адаптери для ListView.

5.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Реалізувати "Hamburger Menu" на основі NavigationView, використовуючи шаблон "Navigation Drawer Activity".

2. На головній активності відобразити список елементів відповідно до індивідуального завдання.

3. Забезпечити реакцію на натискання кнопки FloatingActionButton згідно підпункту А індивідуального завдання.

4. Забезпечити обробку команд меню NavigationView:

- для підпунктів "Up" і "Down" меню "Scroll" забезпечити два варіанта прокрутки списку головної активності відповідно до підпункту Б;
- для елементів меню, що відзначаються, "Sort 1" і "Sort 2" забезпечити сортування списку відповідно до підпункту В.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Завдання вибирається згідно із номером у журналі за формулою:

```
TaskNumber = ((N - 1) mod 10) + 1
```

1. Елемент списку повинен містити три текстових елемента розташованих один за одним (вертикальна розмітка). Лівіше текстових елементів розташувати картинку. Текст генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – видалити ті елементи, у яких хоча б один рядок тексту починається з приголосної; Б) прокрутка – на перший і останній елемент, що відповідає умові видалення п/п А;

В) сортування – у прямому і зворотному лексикографічному порядку за текстом другого елементу.

2. Елемент списку повинен містити два текстових елементи розташованих один за одним (горизонтальна розмітка). Нижче текстових елементів по центру

розташувати картинку. Текст із декількох слів генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – видалити ті елементи, у яких кожне слово рядка тексту першого елемента починається з голосної.

Б) прокрутка – на перший і останній елемент, що відповідає умові видалення п/п А.

В) сортування – у зворотному і прямому лексикографічному порядку за текстом першого елементу.

3. Елемент списку повинен містити три текстових елементи: два розташовані один за одним (горизонтальна розмітка), третій – під першими двома (вертикальна розмітка). Праворуч розташувати картинку. Текст із декількох слів генерувати випадковим чином з символів латиниці та цифр.

А) при натисканні кнопки – видалити всі елементи, у яких текст першого елемента містить цифри;

Б) прокрутка – на перший і останній елемент, що відповідає умові видалення п/п А;

В) сортування – у прямому і зворотному лексикографічному порядку за текстом третього елемента.

4. Елемент списку повинен містити картинку в центрі та під нею – один текстовий елемент. Текст із декількох слів генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – видалити ті елементи, у яких текст починається та закінчується голосною;

Б) прокрутка – на перший і останній елемент, що відповідає умові видалення п/п А;

В) сортування – у зворотному та прямому лексикографічному порядку за другим словом тексту.

5. Елемент списку повинен містити картинку праворуч і поряд з нею (горизонтальна розмітка) один текстовий елемент. Текст із декількох слів генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – видалити ті елементи, у яких текст починається та закінчується голосною;

Б) прокрутка – на перший і останній елемент, що відповідає умові видалення п/п А;

В) сортування – в зворотному та прямому лексикографічному порядку за другим словом тексту.

6. Елемент списку повинен містити картинку ліворуч і поряд з нею (горизонтальна розмітка) три текстових елементи (вертикальна розмітка). Текст генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – додавати новий елемент.

Б) прокрутка – на перший і останній елемент відповідно з однаковими початковими символами текстових елементів.

В) сортування – у зворотному і прямому порядку за другим символом першого текстового елемента.

7. Елемент списку повинен містити картинку в центрі та під нею (вертикальна розмітка) два текстових елемента (горизонтальна розмітка). Текст генерувати випадковим чином з символів латиниці.

А) при натисканні кнопки – видаляти перший елемент і додавати новий елемент.

Б) прокрутка – на перший і останній елемент з приголосними наприкінці всіх текстових елементів.

В) сортування – у прямому та зворотному порядку за останнім символом другого текстового елемента.

8. Елемент списку повинен містити текстові елементи ліворуч та праворуч і картинку в центрі (горизонтальна розмітка). Текст генерувати випадковим чином з цифр.

А) при натисканні кнопки – видаляти перший елемент з "1" на початку одного із текстових елементів.

Б) прокрутка – на перший і останній елемент, що в тексті містить "0".

В) сортування – у зворотному і прямому порядку за першим символом першого текстового елемента.

9. Елемент списку повинен містити дві картинки ліворуч і праворуч і між ними текстовий елемент (горизонтальна розмітка). Текст генерувати випадковим чином з символів латиниці та цифр.

А) при натисканні кнопки – додавати новий елемент і видаляти останній елемент з цифрою наприкінці тексту.

Б) прокрутка – на останній елемент з двома "9" в тексті і перший елемент з двома "6".

В) сортування – у прямому і зворотному порядку за першою цифрою в тексті.

10.Елемент списку повинен містити текстові елементи знизу і зверху і картинку між ними (вертикальна розмітка). Текст генерувати випадковим чином з символів латиниці та цифр.

А) при натисканні кнопки – додавати новий елемент.

Б) прокрутка – на перший і останній елемент, доданий після натискання кнопки.

В) сортування – у прямому і зворотному порядку за сумою цифр в тексті.

БОНУСНІ ЗАВДАННЯ

1. Розробити проект за шаблоном "Navigation Drawer Activity" власноруч.
2. Змінити фон парних і непарних рядків елементів списку.
3. Використовувати різноманітні картинки для кожного елемента списку.
4. Налаштувати верхню частину панелі Navigation Drawer: змінити фон контейнера; використовувати свою фотографію.
5. Замість ListView використовувати розширену і більш гнучку версію – віджет RecyclerView.

5.5. ЛІТЕРАТУРА

1. Рекомендації щодо створення навігаційної шухляди Navigation Drawer [Електронний ресурс] / мова: англійська // [Режим доступу: https://material.io/guidelines/patterns/navigation-drawer.html]

2. Створення навігаційної шухляди - Navigation Drawer [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/training/implementing-navigation/navdrawer.html].

3. Приклад використання ListView [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/topics/ui/layout/listview.html].

4. Створення списків і карток [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/training/material/listscards.html].

5. Огляд можливостей використання ListView [Електронний ресурс] / мова: англійська // [Режим доступу: http://www.vogella.com/tutorials/AndroidListView/article.html].

6. Огляд можливостей використання RecylerView [Електронний ресурс] / мова: англійська //[Режим доступу: http://www.vogella.com/tutorials/AndroidRecyclerView/article.html].

6. ФРАГМЕНТИ ТА НАЛАШТУВАННЯ

Мета: Навчитись розробляти застосунки із використанням фрагментів та використовувати налаштування.

6.1. ТЕРМІНОЛОГІЯ

Фрагмент (Fragment) — це частина інтерфейсу користувача або логіки, яка може бути повторно використана у різних частинах додатка. Фрагменти дозволяють створювати динамічні та адаптивні інтерфейси, особливо на великих екранах (планшети, тощо).

FragmentManager відповідає за управління фрагментами, додаючи їх, видаляючи, замінюючи та виконуючи інші операції з ними. Це основний механізм для маніпулювання фрагментами в активності.

FragmentTransaction використовується для виконання динамічних змін з фрагментами в межах FragmentManager. Він дозволяє виконувати операції, такі як додавання, заміна або видалення фрагментів.

Back Stack — це механізм, який дозволяє фрагментам зберігати свій стан у стеку, щоб користувач міг повернутися до попереднього фрагмента, використовуючи кнопку «Назад».

Спільні налаштування (англ. **Shared Preferences**) — параметри для зберігання у пісочниці в форматі ключ-значення.

Налаштування (англ. **Settings**) — екран з набором елементів, що впливають на роботу застосунку.

PreferenceFragmentCompat — це фрагмент, який спеціально розроблений для відображення і управління налаштуваннями. Він використовується разом із XML-файлами, що визначають структуру налаштувань.

PreferenceScreen — це верхній рівень у ієрархії налаштувань. Він містить всі інші елементи налаштувань і відображає їх у вигляді списку.

Preference — це базовий клас для окремих налаштувань. Існують різні типи налаштувань, такі як CheckBoxPreference, EditTextPreference, SwitchPreference та інші.

Журнал (англ. **Log**) — текстовий режим відображення зневаджувальної інформації.

6.2. ЗАГАЛЬНІ ВІДОМОСТІ

Фрагменти (Fragments) в Android є одним із ключових компонентів для створення адаптивних і динамічних інтерфейсів користувача. Вони дозволяють організувати користувацький інтерфейс у компоненти, які можуть бути повторно використані і динамічно додаватися, видалятися або замінюватися під час виконання додатка.

Фрагмент — це модульний компонент Android-додатка, який представляє частину користувацького інтерфейсу або логіки в межах діяльності (Activity). Фрагмент може мати власний життєвий цикл, макет та обробку подій, що дозволяє створювати складні й динамічні інтерфейси.

Використання фрагментів дозволяє адаптувати інтерфейс додатка для різних розмірів екранів (наприклад, для телефонів і планшетів). На планшеті можна одночасно відображати кілька фрагментів, тоді як на телефоні кожен фрагмент може відображатися окремо.

Життєвий цикл фрагмента. Фрагменти мають власний життєвий цикл, який тісно пов'язаний із життєвим циклом активності, в якій вони знаходяться.

Основні методи життєвого циклу фрагмента включають:

- **onAttach(Context context)**: Викликається, коли фрагмент приєднується до активності.
- **onCreate(Bundle savedInstanceState)**: Викликається для ініціалізації фрагмента. Використовується для налаштування неінтерактивних частин фрагмента (напр. збереження стану).

- **onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)**: Викликається для створення ієрархії представлень (view) фрагмента.
- **onActivityCreated(Bundle savedInstanceState)**: Викликається, коли діяльність, до якої прив'язаний фрагмент, завершила своє створення.
- **onStart()**: Викликається, коли фрагмент стає видимим для користувача.
- **onResume()**: Викликається, коли фрагмент починає взаємодіяти з користувачем.
- **onPause()**: Викликається, коли фрагмент перестає бути активним (напр. коли фрагмент замінюється іншим).
- **onStop()**: Викликається, коли фрагмент більше не видимий для користувача.
- **onDestroyView()**: Викликається, коли видаляється ієрархія представлень фрагмента.
- **onDestroy()**: Викликається перед знищенням фрагмента.
- **onDetach()**: Викликається, коли фрагмент від'єднується від активності.

FragmentManager. Відповідає за управління фрагментами, які додаються до активності. З його допомогою можна додавати, видаляти, замінювати фрагменти та керувати ними динамічно під час виконання додатка.

FragmentTransaction. Використовується для виконання операцій з фрагментами, таких як додавання, заміна, видалення. Операції, виконані через FragmentTransaction, можуть бути додані до Back Stack (історії навігації), що дозволяє користувачу повертатися до попереднього стану фрагментів за допомогою кнопки «Назад».

Використання фрагментів. Фрагменти можна використовувати як статично, так і динамічно.

Статичне використання: Фрагмент додається до макету активності через XML-файл, і він присутній під час усього життєвого циклу активності.

```
<FrameLayout
```

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

    <fragment
        android:id="@+id/example_fragment"
        android:name="com.example.MyFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

Динамічне використання: Фрагменти додаються або замінюються програмно під час виконання додатка за допомогою FragmentManager і FragmentTransaction.

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

MyFragment fragment = new MyFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Комунікація між фрагментами. Фрагменти можуть спілкуватися між собою через активність, яка їх містить. Рекомендується створити інтерфейс у фрагменті і реалізувати цей інтерфейс в активності, що дозволяє передавати дані між фрагментами через активність.

Створення користувацького інтерфейса. Фрагменти зазвичай використовуються як частина користувацького інтерфейса, при цьому в activity додається макет (layout) фрагмента. Це можна зробити наступним чином:

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Метод **inflate()** у прикладі вище приймає три аргумента:

• Ідентифікатор ресурса макет, який потрібно відобразити.

• Об’єкт класа **ViewGroup**, кий повинен стати батьківським елементом для макета фрагмента.

• Логічне значення, яке показує, чи слід прикріпити макет фрагмента до об’єкта **ViewGroup**.

Додавання фрагмента в activity. Зазвичай, фрагмент додає частину користувацького інтерфейса в activity і цей інтерфейс вбудовується в загальну ієрархію компонентів активності.

Для розробника **є дві можливості** додати фрагмент в макет активності.

Визначити фрагмент в файлі макета активності. В цьому випадку можна вказати властивості макета фрагмента в xml файлі відповідної активності:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Додати фрагмент в існуючий об'єкт ViewGroup у програмному коді. Для виконання транзакцій з фрагментами (додавання, видалення, заміна фрагмента) необхідно використовувати API-інтерфейси з класа **FragmentTransaction**. Визначити екземпляр класа **FragmentTransaction** можна наступним чином:

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Після цього, можна додати фрагмент методом **add(),** вказавши фрагмент, що додається та об'єкт **ViewGroup** у який додається фрагмент:

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Перший аргумент метода **add()** – це контейнерний об'єкт **ViewGroup** для фрагмента, який вказано за допомогою ідентифікатора ресурса. Другий аргумент – фрагмент, який необхідно додати. Метод **commit()** необхідний для коректного завершення транзакції.

Додавання фрагмента без користувацького інтерфейса. Фрагменти можуть використовуватись виконання дій в фоновому режимі. Для додавання фрагменту без користувацького інтерфейса використовується метод **add(Fragment, String)**. В методі **add(Fragment, String)** передається унікальний строковий параметр («тег»). Фрагмент буде додано, але, оскільки від не пов'язаний з елементом View, він не буде приймати виклик метода **onCreateView()**. Тому, в реалізації цього метода немає необхідності.

Якщо у фрагмента немає користувацького інтерфейса, строковий тег є єдиним засобом його ідентифікації.

Управління фрагментами. Для управління фрагментами в activity використовується клас **FragmentManager**. Щоб отримати його, потрібно викликати метод **getFragmentManager()** з коду activity.

Дії, які дозволяє виконувати **FragmentManager**:

• Отримувати фрагменти, які є в активності, за допомогою метода **findFragmentById()** (для фрагментів, які мають користувацький інтерфейс) або **findFragmentByTag()** (для всіх фрагментів).

• Видаляти фрагменти з стека переходів назад методом **popBackStack()** (імітується натиснення кнопки «Назад» на пристрої).

• Регіструвати процесс-listener змін в стеку переходів назад за допомогою метода **addOnBackStackChangedListener().**

Транзакції з фрагментами. Перевагою використання фрагментів є можливість їх створення, видалення та інших дій у відповідь на певну поведінку користувача. Будь-який набір змін у activity називається транзакція. Транзакцію можна виконати за допомогою API інтерфейсів класа **FragmentTransaction**. Кожну транзакцію можна зберегти у стеку переходів назад, яким керує activity.

Взаємодія з activity. Зазвичай екземпляр фрагмента пов'язано з певною активністю. Так, фрагмент може звернутися до свого activity за допомогою метода **getActivity()** та отримати, наприклад, ідентифікатор макета:

```
View listView = getActivity().findViewById(R.id.list);
```

Аналогічним чином, активність може викликати методи фрагмента, отримавши посилання на об'єкт Fragment від FragmentManager за допомогою метода findFragmentById() ало findFragmentByTag():

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

6.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. З чого складається фрагмент?
2. Яким чином реалізувати динамічну заміну фрагментів?
3. Які є рівні меню та як додати меню у фрагмент?
4. Для чого потрібні спільні налаштування?
5. Опишіть типове призначення перевизначених функцій для фрагментів.

6.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

Необхідно розробити програму із використанням двох фрагментів логіки зі застосуванням збереження відповідних даних у набір налаштувань. Вважається, що студенти знайомі зі структурою наданого прикладу із лекційного матеріалу.

Обов'язково обробити поворот екрану, щоб дані не пропадали з полів вводу.

Переходи між фрагментами здійснюються за допомогою меню.

Завдання вибирається згідно із номером у журналі за формулою

TaskNumber = (№ mod 10) +1;

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Розробити застосунок для підрахунку результату операції над двома числами із додатковими умовами:

   a. перший фрагмент: два поля для вводу чисел та пункт меню "порахувати".

   b. другий фрагмент: відбувається калькуляція та показується результат у рядок, наприклад:

   5 + 532 = 537

   c. тип операції "+", "-" зберегти у спільних налаштуваннях.

2. У першому фрагменті використати числове та рядкове поле вводу. Передати дані для підрахунку у другий фрагмент, де провести об'єднання у єдиний рядок у залежності від прапорця, що заданий у спільних налаштуваннях число вставляється спочатку або в кінці.

3. У спільні налаштування вноситься тип операції множення або ділення. У залежності від цієї операції необхідно з першого фрагменту передати два числа, а у другому виконати підрахунок, використавши вказану операцію. Результат зберегти у спільні налаштування та відобразити у журналі з першого фрагменту.

4. У другому фрагменті порахувати кількість хвилин, що пройшло між двома вказаними датами на першому фрагменті. Результат записати у спільні налаштування.

5. На першому фрагменті у декількох полях вводяться числа. У спільні налаштування внести операцію, що потрібно послідовно застосувати до чисел. Калькуляцію проводити у другому фрагменті та записати результат у спільні налаштування.

6. У першому фрагменті ввести прізвище, ім'я та по батькові в окремі поля вводу. Передати ці дані у другий фрагмент та у ньому підготувати скорочення даних до одного з форматів у залежності від елементу в спільних налаштуваннях.

Прізвище І.Б.

І.Б. Прізвище

7. Рейтинг встановлюється на підставі отриманої кількості балів та незмінному полі максимальної кількості балів. Вказані дані вводяться в першому фрагменті. У спільні налаштування записати величину бонусного балу. На другому фрагменті порахувати процент отриманих балів разом з бонусним відносно максимального. Результат передати у перший фрагмент та вивести у журнал.

8. У спільні налаштування записано операцію + чи *. Виконати підрахунок даних, введених на першому фрагменті, у другому фрагменті. Результат записати у спільні налаштування та вивести на першому фрагменті.

9. Користувач на першому фрагменті вводить дату та число тижнів. Відповідно до операції + чи – із спільних налаштувань. На другому фрагменті виконується підрахунок із виводом, а результат передається у перший фрагмент. У журнал в режимі інформації перший фрагмент виводить результат та вихідні дані.

10. Користувач на першому фрагменті вводить повну назву навчального корпуса та номер аудиторії. Другий фрагмент відповідає за отримання

скороченої назви у форматі в залежності від значення у спільних налаштуваннях:

ВК 313

313 ВК

БОНУСНЕ ЗАВДАННЯ

Реалізувати додатковий фрагмент з набором налаштувань відповідно до вашого індивідуального завдання. Якщо робота виконується на занятті викладач може додати до 4 балів додатково. У разі виконання в домашньому режимі – до 2 балів.

6.5. ЛІТЕРАТУРА

1. Графічний життєвий цикл фрагменту та активності [Електронний ресурс] / мова: англійська //[Режим доступу: https://github.com/xxv/androidlifecycle/blob/master/complete_android_fragment_lifecycle.png].

2. Приклад роботи з фрагментом [Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/training/basics/fragments/index.html].

3. Using Shared Preferences на ресурсі розробників Андроїд[Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/guide/topics/data/data-storage.html#pref].

4. Робота з меню в Андроїд [Електронний ресурс] / мова: англійська // [Режим доступу: https://code.tutsplus.com/tutorials/android-sdk-implement-an-optionsmenu--mobile-9453].

5. Робота з налаштуваннями[Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/topics/ui/settings.html].

7. РОБОТА З БАЗАМИ ДАНИХ

Мета: Навчитись працювати з базою даних SQLite.

7.1. ТЕРМІНОЛОГІЯ

SQLite — це вбудована реляційна база даних, яка використовується в Android для зберігання структурованих даних. SQLite не вимагає налаштування серверної частини і зберігає дані у файлі на пристрої. Вона підтримує більшість стандартних SQL-операцій і дозволяє ефективно управляти даними.

Таблиця (Table) — це структура, що складається з рядків і стовпців, де зберігаються дані. Кожен рядок відповідає одному запису, а стовпці визначають тип даних, які можуть бути збережені (наприклад, текст, числа).

Рядок (Row) — це один запис у таблиці. Рядок складається з кількох полів (стовпців), які можуть містити різні типи даних.

Стовпець (Column) — це окреме поле у рядку таблиці, що представляє тип даних (напр., ім'я користувача, дата, номер тощо).

Запит (Query) — це інструкція SQL, яка використовується для взаємодії з базою даних. Запити можуть виконувати операції вибору, вставки, оновлення або видалення даних.

SQLiteOpenHelper — це абстрактний клас, який полегшує роботу з базою даних SQLite в Android. Він надає методи для створення, оновлення та управління базою даних.

Cursor — це інтерфейс, який використовується для читання результатів запитів до бази даних. Він надає доступ до даних, повернутих запитом, дозволяючи переміщатися між рядками результатів і читати дані з окремих стовпців.

ContentProvider — це клас, який використовується для надання даних додатка іншим додаткам. Він дозволяє спільний доступ до даних між додатками, використовуючи URI (уніфікований ідентифікатор ресурсу) для ідентифікації ресурсів, таких як бази даних.

Room — це бібліотека, яка є частиною Android Jetpack і надає об'єктно-реляційне відображення (ORM) для SQLite. Room спрощує роботу з базою даних, дозволяючи використовувати об'єкти і методи Java для доступу до даних замість використання сирого SQL.

ContentResolver — це клас, який дозволяє додатку взаємодіяти з іншими додатками через ContentProvider. Він надає методи для виконання CRUD-операцій через URI.

7.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Допоміжний клас SQLiteOpenHelper

Для спрощення роботи з базами даних SQLite в Android нерідко застосовується клас **SQLiteOpenHelper**. Для використання необхідно створити клас-спадкоємець від **SQLiteOpenHelper**, перевизначивши як мінімум два його методи:

– **onCreate()**: викликається при спробі доступу до бази даних, але коли ця база даних ще не створена;

– **onUpgrade()**: викликається, коли потрібно оновити схеми бази даних. У цьому випадку є змога перебудувати (оновити) раніше створену базу даних в onCreate(): встановивши відповідні правила перетворення від старої версії бази до нової.

Слід також перевизначити конструктор, вказавши у ньому ім'я до своєї бази та номер поточної версії. На базі цього номеру система робитиме висновки про очікувану та поточну версію і необхідність оновлення структури.

```
public class DatabaseHelper extends SQLiteOpenHelper {
       private static final String DATABASE_NAME = "userstore.db"; // назва бд
       private static final int SCHEMA = 1; // версія бази даних
       static final String TABLE = "users"; // назва таблиці в бд
       // назва стовбців
       public static final String COLUMN_ID = "_id";
       public static final String COLUMN_NAME = "name";
       public static final String COLUMN_YEAR = "year";

       public DatabaseHelper(Context context) {
```

```
        super(context, DATABASE_NAME, null, SCHEMA);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL("CREATE TABLE users (" + COLUMN_ID
                + " INTEGER PRIMARY KEY AUTOINCREMENT," + COLUMN_NAME
                + " TEXT, " + COLUMN_YEAR + " INTEGER);");
        // додавання початкових даних
        db.execSQL("INSERT INTO "+ TABLE +" (" + COLUMN_NAME
                + ", " + COLUMN_YEAR + ") VALUES ('Том Смит', 1981);");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS "+TABLE);
        onCreate(db);
    }
}
```

Клас **ContentValues** використовується для визначення полів таблиці і значень, які ми в ці поля будемо вставляти. Потім, за допомогою методу **getWritableDatabase()** підключаємося до БД і отримуємо об'єкт **SQLiteDatabase**. Він дозволить нам працювати з БД. Ми будемо використовувати його методи **insert** - вставка запису, **query** - читання, **delete** - видалення. У них багато різних параметрів на вхід, але ми поки використовуємо лише мінімум з них.

Параметри методу query:

– **columns** – список полів, які ми хочемо отримати;

– **selection** – рядок умови WHERE;

– **selectionArgs** – масив аргументів для selection. У selection можна використовувати знаки ?, які будуть замінені цими значеннями;

– **groupBy** – групування результатів;

– **having** – використання умов для агрегатних функцій;

– **orderBy** – сортування результатів.

Отримання даних і Cursor

Android надає різні можливості для здійснення запитів до об'єкта **SQLiteDatabase**. У більшості випадків ми можемо застосовувати метод **rawQuery()**, який приймає два параметри: SQL-вираз SELECT і додатковий параметр, що задає параметри запиту. Після виконання запиту **rawQuery()** повертає об'єкт **Cursor**, який зберігає результат виконання SQL-запиту:

```
userCursor = db.rawQuery("select * from "+ DatabaseHelper.TABLE, null);
```

Клас **Cursor** пропонує ряд методів для управління вибіркою, зокрема:

– **getCount()**: отримує кількість витягнутих з бази даних об'єктів;

– методи **moveToFirst()** та **moveToNext()** дозволяють переходити до першого і до наступного елементам вибірки;

– метод **isAfterLast()** дозволяє перевірити, чи досягнуто кінець вибірки;

– методи **getXxxx(columnIndex)** (наприклад, **getLong()**, **getString()**)дозволяють за індексом стовпчика звернутися до даного стовпця поточного ряду.

ContentProvider в Android — це один з основних компонентів для надання доступу до даних одного додатка з інших додатків. Він забезпечує стандартизований спосіб обміну даними між додатками, що важливо для збереження захищеності і цілісності даних.

ContentProvider використовується, коли додаток хоче дозволити іншим додаткам доступ до своїх даних або коли сам додаток потребує доступу до даних, які зберігаються в інших додатках. Наприклад, контакти, медіафайли або інші особисті дані, збережені на пристрої, доступні через відповідні **ContentProvider** системи.

ContentProvider використовує URI для визначення, які саме дані потрібні іншому додатку. URI містить інформацію, необхідну для ідентифікації ресурсу, і складається з наступних частин:

- **Scheme**: Вказує на протокол, який використовується. Для ContentProvider це завжди content://.

- **Authority**: Унікальний ідентифікатор ContentProvider, зазвичай це назва пакету додатка.
- **Path**: Вказує на конкретні дані або таблицю в базі даних.
- **ID**: Ідентифікатор конкретного запису в таблиці.

ContentProvider надає методи для виконання CRUD-операцій (Create, Read, Update, Delete):

- **insert()**: Додає новий запис у базу даних.
- **query()**: Читає дані з бази даних і повертає їх у вигляді Cursor.
- **update()**: Оновлює існуючі записи в базі даних.
- **delete()**: Видаляє записи з бази даних.

Ці методи використовуються іншим додатком для взаємодії з даними, доступ до яких надає ContentProvider

Приклад використання **ContentResolver** для отримання даних:

```
ContentResolver resolver = getContentResolver();
Uri uri = Uri.parse("content://com.example.app.provider/table_name");
Cursor cursor = resolver.query(uri, null, null, null, null);
```

Визначення власного ContentProvider

Для створення власного ContentProvider, розробник повинен успадкувати свій клас від базового класу ContentProvider і реалізувати необхідні методи (наприклад, query(), insert(), update(), delete()).

Створення класу, що успадковується від ContentProvider, і реалізація методів для управління даними:

```
public class MyContentProvider extends ContentProvider {

    private static final String AUTHORITY =
"com.example.app.provider";
    private static final String PATH = "table_name";
    public static final Uri CONTENT_URI =
Uri.parse("content://" + AUTHORITY + "/" + PATH);

    @Override
```

```
        public boolean onCreate() {
            // Ініціалізація ресурсів, таких як база даних
            return true;
        }

        @Override
        public Cursor query(Uri uri, String[] projection, String
selection, String[] selectionArgs, String sortOrder) {
            // Логіка для читання даних з бази даних
            return null;
        }

        @Override
        public Uri insert(Uri uri, ContentValues values) {
            // Логіка для вставки даних у базу даних
            return null;
        }

        @Override
        public int update(Uri uri, ContentValues values, String
selection, String[] selectionArgs) {
            // Логіка для оновлення даних у базі даних
            return 0;
        }

        @Override
        public int delete(Uri uri, String selection, String[]
selectionArgs) {
            // Логіка для видалення даних з бази даних
            return 0;
        }

        @Override
        public String getType(Uri uri) {
            // Визначення типу даних, що повертаються
```

```
            return null;
        }
    }
```

Декларування в AndroidManifest.xml

Декларація ContentProvider в AndroidManifest.xml із зазначенням authority, який ідентифікує ContentProvider для інших додатків.

```
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider"
    android:exported="true" />
```

Безпека даних

При наданні доступу до даних через ContentProvider, важливо забезпечити належний рівень безпеки, щоб захистити конфіденційні дані. Доступ до ContentProvider може бути обмежений за допомогою дозволів (permissions), що визначають, хто може отримувати доступ до певних даних.

Переваги використання ContentProvider:

- **Спільний доступ до даних**: ContentProvider дозволяє зручно ділитися даними між додатками.
- **Абстрагування**: Він надає стандартний інтерфейс для доступу до даних, незалежно від того, як ці дані зберігаються (у базі даних, у файлах, у мережі тощо).
- **Безпека**: Забезпечує механізм контролю доступу до даних.

Недоліки використання ContentProvider:

- **Складність**: Реалізація ContentProvider може бути складною, особливо якщо потрібно обробляти складні запити або великі обсяги даних.
- **Продуктивність**: Використання ContentProvider може впливати на продуктивність додатка, якщо не оптимізувати запити та обробку даних.

ContentResolver

ContentResolver в Android — це клас, який надає додатку інтерфейс для взаємодії з ContentProvider, що дозволяє здійснювати CRUD-операції (створення, читання, оновлення та видалення) над даними, які можуть зберігатися в базі даних, файлах, або інших сховищах. ContentResolver є ключовим елементом, який забезпечує доступ до даних між додатками в Android, використовуючи унікальні URI для ідентифікації конкретних ресурсів.

Призначення ContentResolver

ContentResolver використовується для звернення до даних, що надаються **ContentProvider**, як у вашому додатку, так і в інших додатках. Він абстрагує деталі взаємодії з **ContentProvider**, дозволяючи розробникам здійснювати запити до даних за допомогою методів класу **ContentResolver** без необхідності знати, як саме ці дані зберігаються.

CRUD-операції з ContentResolver

Методи ContentResolver дозволяють здійснювати основні операції з даними:

- **query()**: Використовується для читання даних з ContentProvider. Він повертає Cursor, який містить результат запиту:

```
Cursor cursor = getContentResolver().query(uri, projection, selection, selectionArgs, sortOrder);
```

- **insert()**: Використовується для додавання нових даних до ContentProvider:

```
ContentValues values = new ContentValues();
values.put("column_name", "value");
Uri newUri = getContentResolver().insert(uri, values);
```

- **update()**: Оновлює існуючі дані в ContentProvider:

```
int rowsUpdated = getContentResolver().update(uri, contentValues, selection, selectionArgs);
```

- **delete()**: Видаляє дані з ContentProvider:

```
int rowsDeleted = getContentResolver().delete(uri, selection, selectionArgs);
```

- **getType()**: Повертає MIME-тип даних, на які вказує URI. Це може бути використано для визначення типу даних, що містяться за URI:

```
String type = getContentResolver().getType(uri);
```

Асинхронні методи

Оскільки операції з даними можуть бути часозатратними, Android пропонує асинхронні методи для роботи з ContentResolver:

- **registerContentObserver()**: Реєструє спостерігача для відстеження змін у даних, що зберігаються за вказаним URI:

```
getContentResolver().registerContentObserver(uri, true, observer);
```

- **notifyChange()**: Використовується ContentProvider для повідомлення про зміни даних, що дозволяє іншим додаткам, які зареєстрували ContentObserver, бути сповіщеними:

```
getContentResolver().notifyChange(uri, null);
```

Безпека та дозволи

При використанні ContentResolver, важливо враховувати питання безпеки даних. Доступ до ContentProvider може бути обмежений за допомогою дозволів (permissions), що гарантує, що тільки авторизовані додатки можуть виконувати певні дії.

7.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Що таке база даних?
2. Які методи створення та оновлення бази даних ви знаєте?

3. Які дії ми можемо робити з полем у таблиці бази даних SQLite?
4. Чи можна видалити строку бази даних не по _id?
5. Які дані та у яку функцію потрібно передати для відкриття бази?

7.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

У розмітці активності не повинно бути закодованих чисел та констант. Усі дані повинні братись із ресурів @color/your_id, @drawable/your_id, @dimen/your_id, @string/your_id.

Завдання вибирається згідно із номером у журналі за формулою

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Створити застосунок з однією таблицею, яка складається трьох полів: ідентифікаційний номер, будинок та вартість будинку. Вивести значення таблиці. Вивести сумарну вартість усіх будинків.

2. Створити застосунок з однією таблицею, яка складається трьох полів: серія паспорта, номер та прізвище. Вивести значення таблиці. Видалити рядок таблиці за номером паспорта.

3. Створити застосунок з однією таблицею, яка складається п’яти полів: прізвище, ім'я, по батькові, рік народження, місто проживання. Вивести значення таблиці. Вивести інформацію про людину починаючи з молодшого до старшого.

4. Створити застосунок з однією таблицею, яка складається з трьох полів: ідентифікаційний номер автомобіля, рік випуску авто та прізвище власника. Заповнити таблицю не менше шести рядків. Вивести значення таблиці. Змінити поле таблиці за ідентифікаційний номером.

5. Створити застосунок з однією таблицею, яка складається двох полів: марка автомобіля та рік випуску авто. Заповнити таблицю не менше п’яти

рядків. Вивести значення таблиці, очистити таблицю та знову вивести значення. Сортувати автомобілі по марці авто.

6. Створити застосунок з однією таблицею, яка складається з п’яти полів: прізвище, ім'я, по батькові, рік народження, місто проживання. Вивести значення таблиці. Вивести всіх людей які проживають в одному місті.

7. Створити застосунок з однією таблицею, яка складається з двох полів: номер автомобіля та рік випуску авто. Заповнити таблицю за допомогою генератора випадкових чисел, не менше п’яти рядків. Вивести значення таблиці. Вивести рік випуску авто за його номером.

8. Створити застосунок з однією таблицею, яка складається з трьох полів: ідентифікаційний номер, марка авто та власник. Заповнити таблицю не менше шести рядків. Вивести значення таблиці. Очистити таблицю. Знайти власника авто за номером авто.

9. Створити застосунок з однією таблицею, яка складається з двох полів: заробітна плата за рік та витрати за рік . Заповнити таблицю, не менше п’яти рядків. Вивести значення таблиці. Вивести залишок від заробітної плати.

10.Створити застосунок з однією таблицею яка складається з трьох полів: прізвище, ім'я та по батькові. Заповнити таблицю, не менше п’яти рядків. Вивести значення таблиці, сортувати значення за алфавітом по полю прізвище.

7.5. ЛІТЕРАТУРА

1. Збереження даних в базах даних SQL [Електронний ресурс] / мова: російська // [Режим доступу: https://developer.android.com/training/basics/data-storage/databases.html].

2. Методи для управління базою даних [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html]

8. РОБОТА З ЛОКАЛІЗАЦІЄЮ

Мета: Освоїти навички роботи з локалізації для Андроїд програм

8.1 ТЕРМІНОЛОГІЯ

Локалізація — це процес налаштування додатка під конкретний регіон або мову. Це передбачає переклад текстових ресурсів, а також налаштування форматів чисел, дат, часу, валюти, і навіть напрямку тексту.

Псевдолокалізація — це метод тестування, який використовується для виявлення проблем з інтернаціоналізацією на ранніх стадіях розробки. Вона включає в себе заміну текстових ресурсів на "псевдопереклади", які зазвичай представляють собою текст з додатковими символами або замінниками.

Інтернаціоналізація — це процес підготовки додатка до локалізації. Вона включає в себе створення коду і структури додатка таким чином, щоб він міг бути легко адаптований до різних мов і культур без значних змін у програмному коді.

Ресурси (Resources). Усі текстові та графічні ресурси, які мають бути локалізовані, зберігаються у спеціальних файлах ресурсів (res). Це можуть бути рядки (strings.xml), макети, зображення, або навіть стилі. Android дозволяє створювати кілька версій цих ресурсів для різних мов або регіонів.

8.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Ресурси у системі Андроїд розміщені у підпапках папки res.

Імена каталогів ресурсів дуже важливі, основні назви наводяться нижче:

animator/ - визначають анімації властивостей.

anim/ - анімації перетворень.

color/ - перелік станів кольору.

drawable/ - растрові зображення (png .jpg, .gif) або примітиви малювання.

mipmap/ - значки запуску з різною щільністю.

layout/ - розмітки користувацького інтерфейсу.

menu/ - меню програми, меню параметрів, контекстні меню або вкладені меню.

raw/ - довільні файли для збереження у вихідній формі.

values/ - прості значення, такі як рядки, цілі числа і кольору.

xml/ - довільні XML-файли, які можна читати під час виконання викликом методу Resources.getXML().

Локалізація – це процес адаптації програмного забезпечення для конкретного регіону або мови, який включає в себе переклад користувацького інтерфейсу, переклад необхідної документації, контроль формату дати і часу, увагу до грошових одиниць, увагу до правових особливостей, розкладка клавіатури користувача та інші подібні аспекти.

Андроїд підтримує кілька специфікаторів конфігурації, що відповідають за локалізацію, дозволяючи додавати декілька до імені каталогу, як вказано у табл.8.1.

Таблиця № 8.1 – Деякі специфікатори, щодо локалізації

Конфігурація	**Приклад**	**Опис**
MCC и MNC	mcc310 mcc310-mnc004 mcc208-mnc00	Код країни для мобільного зв'язку (MCC), код мережі мобільного зв'язку (MNC).
Мова та регіон	en fr en-rUS fr-rFR fr-rCA	Мова задається двохбуквеним кодом мови за стандартом ISO 639-1, дволітерний код регіону ISO 3166-1
Макет	ldrtl ldltr	ldrtl - напрямок макета справа наліво ldltr - напрямок макета зліва направо

Система Андроїд вибирає підходящу папку у залежності від поточної мови пристрою під час роботи програми та завантажує ресурси з неї. Наприклад, з папки res/values-uk/strings.xml будуть взяті рядки українською.

Параметри, що ідентифікують Україну - uk, та українську мову – UKR.

Для створення ресурсу конкретного регіону або мови, необхідно використовувати специфікатор, який визначає комбінацією мови та регіону.

Загальний синтаксис:

```
<ім'я папки> [-країна][-rМОВА]
```

У прикладі вище values-uk - папка, де зберігаються локалізовані ресурси для України.

Приклад локалізованого ресурсу рядків для української мови:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
     <string name="app_name"> I’мя</string>
     <string name="hello_world">Текст!</string>
     <string name="action_settings">Налаштування</string>
     <string name="word">Слово</string>
</resources>
```

Додаткові відомості

Configuration sysConfig = getResources().getConfiguration(); – надання загальної інформації про ОС Android.

setImageResource() – способи доступу ресурсів з коду

@string/hello – способи доступу ресурсів за допомогою синтаксису XML де string – тип ресурсу, а hello – ім'я ідентифікатора.

Окремі ресурси можна отримувати за допомогою методів у класі Resources, примірник якої можна отримати за допомогою методу getResources().

<xliff:g> містить коментар, що допоможе перекладачеві коректно перекласти даний ресурс на іншу мову.

R.resource_type.resource_name – посилання на ресурси з коду java.

Context – зазвичай відповідає активності. За допомогою цього об'єкту можна отримати доступ до ресурсів, а по ній встановити місце знаходження/мову пристрою.

```
String loc =
context.getResources().getConfiguration().locale.getDispl
ayName();
```

Локалізація в Android — це процес адаптації додатка до різних мов і культурних стандартів, що дозволяє зробити додаток доступним для більш широкої аудиторії. Використання локалізованих ресурсів, коректне форматування дати, часу і чисел, підтримка RTL та використання інструментів для локалізації є ключовими аспектами цього процесу. Інтернаціоналізація, як підготовчий етап, дозволяє спростити процес локалізації, роблячи додаток гнучким і легко адаптованим до нових мов і регіонів.

8.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Що таке локалізація?
2. Які типи ресурсів ви знаєте?
3. Які імена специфікаторів конфігурації ви знаєте?
4. З чого складається ідентифікація ресурсів?
5. Які способи доступу до ресурсів ви знаєте?

8.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

Реалізувати програму з локалізацією для України та США. Типовою мовою буде англійська.

Застосувати зміни відповідно до індивідуального завдання.

Завдання вибирається згідно із номером у журналі за формулою:

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Розмістити на екрані елемент зі статичним текстом, в залежності від локалізації відобразити назву країни.

2. Розмістити на екрані елемент із зображенням, в залежності від локалізації відобразити прапор країни.

3. Отримати системне місце знаходження/мову пристрою.

4. Відобразити текстовий елемент з кодом системної мови.

5. Відобразити текстовий елемент з кодом регіону місця знаходження пристрою.

6. Відобразити текстовий елемент з грошовою одиницею системи.

7. Відобразити текстовий елемент з повним ім’ям Президента країни відповідно локалізації.

8. Відобразити зелений колір фону застосунку для англійської мови та блакитний для української.

9. Відобразити дві кнопки вертикально для української мови та горизонтально для англійської.

10. Розмістити три кнопки на екрані. Якщо мова українська – в ряд, для англійської – в стовпчик.

8.5. ЛІТЕРАТУРА

1. Графічний життєвий цикл фрагменту та активності [Електронний ресурс] / мова: англійська //[Режим доступу: https://github.com/xxv/androidlifecycle/blob/master/complete_android_fragment_lifecycle.png].

2. Приклад роботи з фрагментом [Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/training/basics/fragments/index.html].

3. Using Shared Preferences на ресурсі розробників Андроїд[Електронний ресурс] / мова: англійська //[Режим доступу: https://developer.android.com/guide/topics/data/data-storage.html#pref].

4. Робота з меню в Андроїд [Електронний ресурс] / мова: англійська // [Режим доступу: https://code.tutsplus.com/tutorials/android-sdk-implement-an-optionsmenu--mobile-9453].

5. Робота з налаштуваннями[Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/topics/ui/settings.html].

9. НАДАННЯ ПОСЛУГ ІНШИМ ЗАСТОСУНКАМ

Мета: Навчитись використовувати послуги, що надано іншими застосунками в системі. Навчитись використовувати широкомовні повідомлення.

9.1. ТЕРМІНОЛОГІЯ

Content Provider — це компонент, який дозволяє додаткам ділитися даними з іншими додатками. Дані можуть зберігатися в базі даних, файлах або будь-якому іншому сховищі, а доступ до них здійснюється через URI (уніфіковані ідентифікатори ресурсів).

Service — це компонент, який виконує фонові завдання для довготривалих операцій. Він може виконувати роботу в фоновому режимі навіть після того, як користувач вийшов з застосунку. Інші застосунки можуть підключатися до сервісу, щоб скористатися його функціоналом.

Broadcast Receiver — це компонент, який дозволяє застосунку отримувати повідомлення від системи або інших застосунків. Ці повідомлення можуть бути системними подіями (наприклад, зміна стану батареї) або спеціальними подіями, переданими іншими застосунками.

Intent — це механізм, який дозволяє одному застосунку запустити інший застосунок або взаємодіяти з іншими компонентами в межах того ж застосунку. Інтент може передавати дані між компонентами та ініціювати певні дії.

Intent Filter — це механізм, який визначає, на які типи інтентів компонент застосунку може реагувати. Використовуючи фільтри інтентів, застосунок може бути налаштований на обробку певних типів дій, таких як відкриття файлу певного типу або прийом конкретних повідомлень.

AIDL (Android Interface Definition Language) – використовується для опису інтерфейсу, який може бути використаний для взаємодії між різними процесами (IPC — Inter-Process Communication). Це дозволяє різним

застосункам спілкуватися між собою через сервіси, навіть якщо вони працюють у різних процесах.

Content Sharing (Спільний доступ до вмісту). Android дозволяє застосункам надавати спільний доступ до контенту (файлів, зображень, відео тощо) через механізм **Intent**. Інші застосунки можуть отримувати ці файли для перегляду або редагування.

9.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Широкомовні повідомлення

Система та інші застосунки використовують широкомовні повідомлення для того, щоб інші застосунки дізналися про якусь подію та могли реагувати відповідно. Повідомлення представлене звичайним класом Intent, який було вивчено на попередніх роботах.

Існує кілька типів повідомлень за одержувачем:

– явні; коли вказано, який застосунок та компонент всередині, мають обробити це повідомлення. Наприклад:

```
final Intent intent = new Intent(this, MainActivity.class);
intent.setAction("com.khpi.pmz.lab09.ACTION_VIEW");
startActivity(intent);
```

– неявні; коли не вказано, кому повинно дійти повідомлення, тому таке повідомлення одержують всі, хто підписаний на нього. Наприклад:

```
final Intent intent = new Intent(Intent.ACTION_DIAL);
intent.setData(Uri.parse("tel:" + phoneNumber));
startActivity(intent);
```

В залежності від повідомлення, його отримувачем можуть бути різні компоненти:

– активності, наприклад, при використанні наступних повідомлень:

- *Intent.ACTION_VIEW*
- *Intent.ACTION_DIAL*
- *Intent.ACTION_EDIT*

- *Intent.ACTION_CALL*
- *Intent.ACTION_SEND*

– спеціальні отримувачі, наприклад, при використання наступних повідомлень:

- *Intent.ACTION_TIMEZONE_CHANGED*
- *Intent.ACTION_DATE_CHANGED*
- *Intent.ACTION_AIRPLANE_MODE_CHANGED*
- *ConnectivityManager.CONNECTIVITY_ACTION*

Спеціальні отримувачі є класами, що успадковані від класу *android.content.BroadcastReceiver*. В залежності від типу отримувача повідомлення, його можна відправляти різними способами:

– *startActivity()*, коли отримувачем є якась активність,

–*sendBroadcast(), sendOrderedBroadcast()*, коли отримувачем є *BroadcastReceiver*,

– *LocalBroadcastManager.sendLocalBroadcast()*, коли отримувачем є *BroadcastReceiver* в тому ж застосунку.

Користувацькі, тобто не системні, застосунки зазвичай не розсилають повідомлення для *BroadcastReceiver* в інші застосунки та не можуть розсилати більшість існуючих системних чи несистемних повідомлень.

Для підписання на отримання повідомлень використовується клас *IntentFilter*. Існують наступні способи підписання на окремі повідомлення:

– статичний;

– динамічний.

Статичний спосіб застосовують, коли підписання відбувається через маніфест, використовуючи спеціальні теги та атрибути. Такий спосіб зазвичай використовується для активностей, наприклад:

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

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

Динамічний метод використовують, коли об'єкт класу *IntentFilter* формується під час виконання застосунку та передається у функцію *registerReceiver()* для реєстрації. Такий спосіб використовується тільки для *BroadcastReceiver*, наприклад:

```
BroadcastReceiver connectivityReceiver = new
ConnectivityChangeReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction(ConnectivityManager.CONNECTIVITY_ACTION)
;
registerReceiver(connectivityReceiver, filter);
```

Видалення зареєстрованого отримувача виконується за допомогою функції *unregisterReceiver()*, наприклад:

```
unregisterReceiver(connectivityReceiver);
```

Практичне використання повідомлень

Фрагмент відкриття головного вікна системних налаштувань

```
final Intent intent = new
Intent(Settings.ACTION_SETTINGS);

intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(intent);
```

Фрагмент відкриття системних Wi-Fi налаштувань

```
final Intent intent = new
Intent(Settings.ACTION_WIFI_SETTINGS);

intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(intent);
```

Фрагмент відкриття типового застосунку для дзвінків з заповненим номером телефону

```
private void dialPhoneNumber(@NonNull String phoneNumber)
{

    final Intent intent = new Intent(Intent.ACTION_DIAL);

    intent.setData(Uri.parse("tel:" + phoneNumber));

    startActivity(intent);

}
```

Фрагмент відкриття типового застосунку для СМС повідомлень

```
private void openSmsComposer(@NonNull String phoneNumber,
@Nullable String body) {

   final Intent intent = new
Intent(Intent.ACTION_SENDTO);

   intent.setData(Uri.parse("sms:" + phoneNumber));

   if (!TextUtils.isEmpty(body)) {

      intent.putExtra("sms_body", body);

 }

 startActivity(intent);

}
```

Фрагмент дзвінка на заданий номер телефону

```
@RequiresPermission(permission.CALL_PHONE)

private void callPhoneNumber(@NonNull String phoneNumber)
{

    final Intent intent = new Intent(Intent.ACTION_CALL);

    intent.setData(Uri.parse("tel:" + phoneNumber));

    startActivity(intent);
```

```
}
```

Цей фрагмент потребує дозволу android.permission.CALL_PHONE.

Фрагмент прихованої відправки СМС повідомлення

```
@RequiresPermission(permission.SEND_SMS)
private void sendSilentSms(@NonNull String phoneNumber,
@Nullable String body) {
    final SmsManager manager = SmsManager.getDefault();
    manager.sendTextMessage(phoneNumber, null, body,
null, null);
}
```

Цей фрагмент потребує дозволу android.permission.SEND_SMS.

9.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Що таке широкомовне повідомлення?

2. Для чого використовується невизначений одержувач повідомлення?

3. Які функції використовуються для відправлення повідомлення?

4. Які способи підписування на повідомлення існують?

5. Що трапиться, якщо отримувачів повідомлення декілька? Поясніть для різного типу одержувача.

9.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Створити новий проект з використанням шаблону Empty Activity, використати типові налаштування, задавши ім’я пакету проекту за правилами попередніх робіт.

2. Обробити відсутність з’єднання з інтернетом з відображенням Snackbar з можливістю відкриття системних налаштувань.

а) В класі MainActivity створити клас ConnectivityChangeReceiver та успадкувати його від BroadcastReceiver. Додати функцію по обробці зміни

стану підключеності до інтернету. Обробити відповідне широкомовне повідомлення в створеному класі-приймачі.

```
private void handleConnectivityUpdated(boolean
connectivityAvailable) {
     // TODO handle connectivity update
}
private class ConnectivityChangeReceiver extends
BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent)
{
       if
    (ConnectivityManager.CONNECTIVITY_ACTION.equals(inten
    t.getAction())) {
        final boolean noConnectivity =
        intent.getBooleanExtra(ConnectivityManager.EXTRA
    _NO_CONNECTIVIT Y,  false);
             handleConnectivityUpdated(!noConnectivity);
        }
     }
}
```

б) Додати до класу *MainActivity* поле для зберігання об’єкту-приймача. Реалізувати функції реєстрації та видалення реєстрації. Викликати створені функції в *onCreate* та *onDestroy* відповідно.

```
private ConnectivityChangeReceiver iConnectivityReceiver;

private void registerConnectivityReceiver() {
    if (iConnectivityReceiver == null) {
        iConnectivityReceiver = new
ConnectivityChangeReceiver();
        IntentFilter filter = new IntentFilter();

    filter.addAction(ConnectivityManager.CONNECTIVITY_ACT
ION);
        this.registerReceiver(iConnectivityReceiver,
filter);
     }
}
```

```
private void unregisterConnectivityReceiver() {
    if (iConnectivityReceiver != null) {
        this.unregisterReceiver(iConnectivityReceiver);
        iConnectivityReceiver = null;
    }
}

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

@Override
protected void onDestroy() {
     unregisterConnectivityReceiver();
     super.onDestroy();
}
```

в) Додати до проекту бібліотеку для створення *Snackbar*.

```
dependencies {
     compile 'com.android.support:design:25.3.0'
}
```

г) Визначити функцію по створенню *Snackbar* з однією дією, який буде відображати інформацію про відсутність з'єднання з інтернетом та надавати можливість користувачеві відкрити системні налаштування.

```
private void openConnectivitySetting() {
    // TODO open connectivity settings page
}
private Snackbar createSnackbarForConnectivityAbsence() {
    return Snackbar
        .make(getWindow().getDecorView(),"No
connectivity",
                        Snackbar.LENGTH_INDEFINITE)
        .setAction("SETTINGS", new OnClickListener() {
            @Override
            public void onClick(View v) {
               openConnectivitySetting();
            }
         });
```

```
}
```

д) Реалізувати функцію відображення та приховування об’єкту *Snackbar*, що було оголошено раніше. Додати до класу *MainActivity* поле для зберігання цього об’єкту.

```
private void handleConnectivityUpdated(boolean
connectivityAvailable) {
    if (connectivityAvailable) {
        if (iConnectivitySnackbar != null &&
                        iConnectivitySnackbar.isShownOrQ
                        ueued()) {
            iConnectivitySnackbar.dismiss();
        }
    } else {
        if (iConnectivitySnackbar == null) {
            iConnectivitySnackbar                    =
    createSnackbarForConnectivityAbsence();
        }
        if (!iConnectivitySnackbar.isShownOrQueued()) {
           iConnectivitySnackbar.show();
        }
    }
}
```

е) Реалізувати функцію відкриття системних налаштувань.

```
private void openConnectivitySetting() {
    // define settings in priority order with fallbacks
    final String[] settingsActions = {
        Settings.ACTION_WIFI_SETTINGS,
        Settings.ACTION_WIRELESS_SETTINGS,
        Settings.ACTION_SETTINGS
    };

    final PackageManager manager = getPackageManager();

    for (final String action : settingsActions) {
        final Intent intent = new Intent(action);
        if (manager.resolveActivity(intent,
                    PackageManager.MATCH_DEFAULT_ONLY)
                    != null) {

        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

```
                startActivity(intent);

                // stop searching matching intents
                break;
            }
        }
}
```

3. Відобразити список контактів з номерами телефонів з можливістю зателефонувати обраному контакту.

а) змінити розмітку *MainActivity* на *ListView*.

```
<?xml version="1.0" encoding="utf-8"?>
<ListView
xmlns:android="http://schemas.android.com/apk/res/android
"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/contacts_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.khpi.pmz_lab09.MainActivity" />
```

б) реалізувати функцію отримання списку контактів.

```
private List<Map<String, Object>> getContacts() {
    final String[] wantedColumns = {
        Phone._ID,
        Phone.DISPLAY_NAME_PRIMARY,
        Phone.NUMBER
    };

    final String sortOrder = Phone.DISPLAY_NAME_PRIMARY;
    final   List<Map<String,   Object>>   data   =   new
ArrayList<>();

    final        ContentResolver        resolver        =
getContentResolver();

    final            Cursor            cursor            =
resolver.query(Phone.CONTENT_URI,   wantedColumns,   null,
null, sortOrder);
    if (cursor != null) {
        try {
            while (cursor.moveToNext()) {
```

```
                final Map<String, Object> item = new
HashMap<>();
                item.put(Phone._ID, cursor.getString(

                cursor.getColumnIndexOrThrow(Phone._ID))
                );

                item.put(Phone.DISPLAY_NAME_PRIMARY,
            cursor.getString(
                        cursor.getColumnIndexOrThrow
(Phone.DISPLAY_NAME_PRIMARY)));

                item.put(Phone.NUMBER, cursor.getString(
                    cursor.getColumnIndexOrThrow(Phone.N
                UMBER)));

                data.add(item);
             }
         } finally {
            cursor.close();
         }
     }
     return data;
}
```

в) створити адаптер для ListView, заповнити його прочитаними контактами.

```
final     ListView     contactsList     =     (ListView)
findViewById(R.id.contacts_list);
final List<Map<String, Object>> data = getContacts();
contactsList.setAdapter(new SimpleAdapter(this,
    data,
    android.R.layout.simple_list_item_2,
    new              String[]{Phone.DISPLAY_NAME_PRIMARY,
Phone.NUMBER},
    new int[]{android.R.id.text1, android.R.id.text2}
));
```

г) додати відповідний дозвіл для доступу до контактів. Переконатись, що *targetSdkVersion* не більше **22**.

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

д) обробити натиснення на елемент списку. При натисненні викликати системний екран набору номеру, після чого користувач самостійно натисне на кнопку виклику.

```
contactsList.setOnItemClickListener(new
OnItemClickListener() {
    @SuppressWarnings("unchecked")
    @Override
    public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {
        final Map<String, Object> item =
            (Map<String,                          Object>)
parent.getAdapter().getItem(position);

        final    String    phoneNumber    =    (String)
    item.get(Phone.NUMBER);
        dialPhoneNumber(phoneNumber);
    }
});
```

Завдання вибирається згідно із номером у журналі за формулою:

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Відкрити типовий застосунок для СМС повідомлень з обраним контактом та якимось типовим змістом.

2. Додати поле для вводу та кнопку на екран для пошуку по імені контакту. При натисненні на кнопку застосунок залишить у списку тільки ті контакти, імена яких містять текст, що введено в поле для вводу.

3. Додати поле для вводу та кнопку на екран для пошуку по номеру телефону контакту. При натисненні на кнопку застосунок залишить у списку тільки ті контакти, номера яких містять текст, що введено в поле для вводу.

4. Подзвонити на обраний контакт без підтвердження з боку користувача.

5. Відправити приховане СМС обраному контакту з якимось типовим змістом. 6. Відправити приховане СМС обраному контакту з використанням діалогу *AlertDialog* для задання змісту повідомлення.

7. Відобразити тип номеру телефону поряд з ним в списку контактів.

8. Відстежити зміну часового поясу замість відсутності з'єднання з інтернетом.

9. Відстежити зміну дати замість відсутності з'єднання з інтернетом.

10.Відстежити зміну режиму «у літаку» замість відсутності з'єднання з інтернетом.

9.5. ЛІТЕРАТУРА

1. Broadcasts [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/components/broadcasts.html]

2. Intents and Intent Filters [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/guide/components/intents-filters.html]

3. Список полів контактів з номерами [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.Phone.html].

10. РОБОТА З СЕРВЕРОМ

Мета: Навчитись виконувати звернення до серверів в мережі INTERNET.

10.1. ТЕРМІНОЛОГІЯ

HTTP (HyperText Transfer Protocol) та **HTTPS (HTTP Secure)** — це протоколи, які використовуються для передачі даних між додатком і сервером через інтернет. HTTPS є більш захищеним, оскільки він використовує шифрування для захисту даних під час передачі.

REST API (Representational State Transfer) — це архітектурний стиль, що використовується для створення веб-сервісів. REST використовує стандартні HTTP-методи для взаємодії з ресурсами на сервері, що дозволяє додатку здійснювати запити та отримувати відповіді у форматах, таких як JSON або XML

JSON (JavaScript Object Notation) — це легкий формат обміну даними, який часто використовується для передачі даних між сервером та клієнтом (мобільним додатком). Він є зручним для використання і легким для парсингу.

Volley — це бібліотека Android для роботи з мережевими запитами. Вона спрощує процес надсилання HTTP-запитів, управління чергою запитів та обробки відповідей з сервера.

Retrofit — це потужна бібліотека Android для роботи з REST API. Вона дозволяє легко конвертувати HTTP API у виклики Java-інтерфейсів і працювати з відповідями у форматі JSON.

OkHttp — це ефективна HTTP-клієнтська бібліотека для Android, яка забезпечує низькорівневий доступ до мережевих операцій. Вона часто використовується в поєднанні з Retrofit або іншими мережевими бібліотеками.

WebSocket — це протокол для двосторонньої комунікації між клієнтом і сервером у реальному часі. Він дозволяє встановити постійне з'єднання, через яке дані можуть передаватися негайно, без необхідності повторного встановлення з'єднання.

Сокет-програмування дозволяє додаткам встановлювати пряме з'єднання з сервером або іншим клієнтом для обміну даними на більш низькому рівні, ніж HTTP. Це може бути корисно для розробки ігор, чату або інших додатків, що потребують миттєвого обміну даними.

Firebase Realtime Database — це хмарне рішення від Google, яке дозволяє зберігати та синхронізувати дані між клієнтами в реальному часі. Це особливо корисно для додатків, які потребують постійного оновлення даних, таких як чати або соціальні мережі.

Кешування — це процес збереження даних на пристрої для швидкого доступу до них без необхідності повторного звернення до сервера. Це знижує навантаження на сервер і покращує швидкість роботи додатка.

10.2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Робота з сервером в Android охоплює широкий спектр технологій і практик, які дозволяють мобільним додаткам взаємодіяти з віддаленими серверами для обміну даними, синхронізації інформації, аутентифікації користувачів, та виконання інших завдань, що потребують мережевої комунікації.

Клас HttpURLConnection

Клас **java.net.HttpURLConnection** є підкласом **java.net.URLConnection** і дозволяє реалізувати роботу по відправці і отриманні даних з мережі по протоколу HTTP. Дані можуть бути будь-якого типу і довжини. Даний клас слід використовувати для відправки та отримання потокових даних невідомого заздалегідь розміру.

Алгоритм використання наступний:

- отримати об'єкт **HttpURLConnection** через виклик **URL.openConnection()** і привести результат до **HttpURLConnection**;
- підготувати необхідний запит. Основне в запиті - мережева адреса. Також в запиті можна вказати різні метадані: облікові дані, тип контенту, кукі сесії тощо;

- опціонально завантажити тіло запиту. У цьому випадку використовується метод **setDoOutput(true)**. Передача даних, записаних в потік, повертається через метод **getOutputStream()**;

- прочитати відповідь. Тема відповіді зазвичай включає метадані, такі як тип і довжина контенту, дати зміни, кукі сесії. Прочитати дані з потоку можна через метод **getInputStream()**. Якщо у відповіді немає тіла, то метод повертає порожній потік.

- розірвати з'єднання. Після прочитання відповіді від сервера **HttpURLConnection** слід закрити через виклик методу **disconnect()**. Тим самим звільнюються ресурси, які були зайняті з'єднанням.

За замовчуванням **HttpURLConnection** використовує метод GET. Для використання POST необхідно викликати **setDoOutput(true)** і посилати дані через **openOutputStream()**. Інші HTTP-методи (OPTIONS, HEAD, PUT, DELETE і TRACE) налаштовуються через метод **setRequestMethod(String)**. Цей метод може бути використаний для явного завдання також методів GET і POST.

Передача даних

Приклад передачі даних методом GET:

```
HttpURLConnection urlConnection = (HttpURLConnection)
                                        url.openConnection();
urlConnection.setRequestMethod("GET");
urlConnection.setRequestProperty("User-Agent",
                                         "Mozilla/5.0");
```

Також можна додати будь-які інші властивості, притаманні для запитів, наприклад:

```
urlConnection.setRequestProperty("Content-Type", "application/x-
                                       www-form□urlencoded");
urlConnection.setRequestProperty("Content-Language", "en-US");
urlConnection.setRequestProperty("Accept", "application/json");
```

Приклад передачі даних методом POST:

```
HttpURLConnection urlConnection = (HttpURLConnection)
url.openConnection();
```

```
urlConnection.setRequestMethod("POST");
urlConnection.setRequestProperty("User-Agent", "Mozilla/5.0");
urlConnection.setRequestProperty("Accept-Language", "en-
US,en;q=0.5");
String urlParameters = "par1=data1&par2=data2"; // параметри, що
передаються
urlConnection.setDoOutput(true);
DataOutputStream wr = new
DataOutputStream(urlConnection.getOutputStream());
wr.writeBytes(urlParameters);
wr.flush();
wr.close();
```

Для обмеження часу виконання запиту слід встановити наступні налаштування:

```
urlConnection.setConnectTimeout(10000); // наприклад - 10 seconds
urlConnection.setReadTimeout(10000); // наприклад - 10 seconds
urlConnection.connect(); // оновити з’єднання
```

В разі підключення до мережі через проксі-сервер, необхідно використовувати **URL.openConnection(Proxy)** при створенні з'єднання.

Кожен об'єкт **HttpURLConnection** може використовуватися тільки для однієї пари запиту / відповіді. Операції зі з’єднаннями слід проводити в окремій нитці (доцільно використовувати клас **android.os.AsyncTask**) в методі **doInBackground()**:

```
class RequestTask extends AsyncTask<String, String, String> {
     private Exception exception;
     protected void onPreExecute() {
          progressBar.setVisibility(View.VISIBLE);
     }
     @Override
     protected String doInBackground(String... params) {
...
     }
     protected void onPostExecute(String response) {
          progressBar.setVisibility(View.GONE);
     }
}
```

При отриманні відповіді доцільно перед читанням перевірити код відповіді та виконати читання з відповідного потоку:

```
 int response = urlConnection.getResponseCode();
    BufferedReader bufferedReader = null;
    int response = urlConnection.getResponseCode();
    if (response == 200) {
    bufferedReader = new BufferedReader(new
    InputStreamReader(urlConnection.getInputStream()));
} else {
    bufferedReader = new BufferedReader(new
    InputStreamReader(urlConnection.getErrorStream()));
    }
```

Права доступу до Інтернет

Для того, щоб застосунок мав доступ до мережі INTERNET також необхідно в файлі **AndroidManifest.xml** додати рядок з описом дозволу:

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

При роботі з сервером Android-додаток зазвичай реалізує **архітектуру клієнт-сервер**. У цій моделі клієнт (мобільний додаток) надсилає запити до сервера, сервер обробляє ці запити, виконує необхідні операції (наприклад, доступ до бази даних), і відправляє відповідь назад клієнту.

Безпека в роботі з сервером

При розробці Android-додатків, які взаємодіють з серверами, важливо дотримуватися принципів безпеки:

- Використання HTTPS для захисту даних під час передачі.
- Використання OAuth або інших протоколів для безпечної аутентифікації та авторизації.
- Захист даних користувача і забезпечення конфіденційності.

Робота з сервером в Android включає широкий набір інструментів і методів для забезпечення комунікації між додатком і сервером. Це дозволяє створювати динамічні додатки, що можуть взаємодіяти з віддаленими ресурсами, забезпечуючи користувачів актуальною інформацією, можливістю спільної роботи і іншими функціями, які потребують обміну даними через мережу.

10.3. ЗАПИТАННЯ ДЛЯ ЕЛЕМЕНТАРНОГО РІВНЯ

1. Як підготувати запит для передачі на сервер?

2. Яким чином явно вказати метод передачі даних на сервер?

3. Яким чином прочитати відповідь з сервера?

4. Які відмінності виконання передачі даних методами GET і POST при розробці застосунку?

5. Для чого необхідно явно розірвати з'єднання після отримання відповіді від сервера?

10.4. ЗАВДАННЯ

ЗАГАЛЬНЕ ЗАВДАННЯ

Виконати індивідуальне завдання з явним завданням методу передачі, аналізом коду відповіді та обмеженням часу очікування відповіді.

Завдання вибирається згідно із номером у журналі за формулою

```
TaskNumber = (№ mod 10) +1;
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Визначити, чи є на сайті сторінка з заданим ім'ям, якщо є – вивести її зміст. Використати явно метод POST.

2. Визначити, чи є сайт з заданою адресою, якщо є – вивести зміст головної сторінки. Використати явно метод GET.

3. Вивести розмір головної сторінки заданого сайту. Використати явно метод GET.

4. Вивести розмір вказаної сторінки на сайті. Використати явно метод POST.

5. Визначити, чи є на сайті сторінка з іменем index.htm, якщо є – вивести її розмір. Використати явно метод GET.

6. Вивести 5 перших рядків головної сторінки сайту. Використати явно метод GET.

7. Вивести 6 перших рядків будь-якої вказаної сторінки сайту. Використати явно метод POST.

8. Вивести рядок «HTML сторінка», якщо на вказаній сторінці сайту першим зустрічається тег <HTML>. Використати явно метод POST.

9. Визначити самий довгий рядок на вказаній сторінці сайту. Використати явно метод POST.

10.Визначити кількість рядків на головній сторінці вказаного сайту. Використати явно метод GET.

10.5. ЛІТЕРАТУРА

1. AsyncTask [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/android/os/AsyncTask.html].

2. HttpURLConnection [Електронний ресурс] / мова: англійська // [Режим доступу: https://developer.android.com/reference/java/net/HttpURLConnection.html]

3. HTTP запрос на Android [Електронний ресурс] / мова: російська //[Режим доступу: https://bytiger.com/ru/blog/35].

4. AsyncTask. Знакомство, несложный пример [Електронний ресурс] / мова: російська //[Режим доступу: http://startandroid.ru/ru/uroki/vse-uroki spiskom/149-urok-86-asynctask-znakomstvo-neslozhnyj-primer.html].

5. URLConnection [Електронний ресурс] / мова: англійська //[Режим доступу: https://joshuadonlan.gitbooks.io/onramp android/content/networking/urlconnection.html.

Навчальне видання

БРЕЧКО Вероніка Олександрівна
БУЛЬБА Сергій Сергійович
БАЛЕНКО Олексій Іванович

«Проєктевання мобільних застосунків»

Навчально-методичний посібник

Роботу до видання рекомендував проф.. Заполовський М.Й.

Незалежна електронна публікація