Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
бакалавра
з теми: “РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ «ДОВІДНИК СТУДЕНТА»”

Виконав: студент 4 курсу, KN1-B18 групи
спеціальності 122 Комп’ютерні науки
Іщенко Єгор Степанович
Керівник: Моцик Ростислав Васильович
доцент кафедри комп’ютерних наук,
кандидат педагогічних, доцент
Рецензент: Сморжевський Ю.Л.
доцент кафедри математики,
кандидат педагогічних наук, доцент.

Кам’янець-Подільський – 2022

ЗМІСТ

ВСТУП

Актуальність дослідження. В сучасному освітньому процесі постійно відбуваються зміни, які зумовлені процесами політичного, економічного та соціального характеру. Відбуваються зміни, як пріоритети в сучасній системі освіти та у системі управління освітою, так і, структура та зміст самого освітнього процесу. Крім цього, у галузі освіти, досить часто, застосовуються і новітні технології. Зокрема, істотні зміни відбулись у інформаційній сфері діяльності сучасної людини призводять до суттєвого зниження ефективності застарілих підходів до освітнього процесу студентів освітніх закладів України. Саме тому, сьогодні є доцільною розробка спеціальної сучасної системи, яка дозволить автоматизувати освітній процес студентів та зробити його швидшим, зручнішим та надійнішим. Швидкий доступ до важливої інформації – це проблема кожного навчального закладу. Такі речі роблять освітній процес непередбачуваним, що надалі позначається на загальних показниках успішності студентів освітніх закладів. Автоматизація освітнього процесу може вирішити ці проблеми надавши гнучкі засоби для отримання важливої інформації студентами. Така система, дозволить швидко виявляти зміни в освітньому процесі студента та наддасть певні засоби для ефективного освітнього процесу . Крім того, такий засіб, буде дозволяти студенту стежити за власною відвідуваністю, дозволяючи пересвідчитися в тому, що він повною мірою використовує можливості навчання, які доступні для нього.

На сьогодні є величезна кількість програм у вільному доступі які допомагають викладачам. Основна ціль яких полегшити освітній процес студентам університетів, коледжів й інших навчальних закладів. Але всі ці програми прив’язані безпосередньо з тією чи іншою навчальною дисципліною, викладання якої забезпечує той чи інший викладач. Також є сучасні програми, які можуть використовуватися для усіх навчальних дисциплін. Основне призначення запропонованого мобільного додатку – зробити легшим і доступнішим освітній процес студентів, тобто прототип

інформаційної довідки студентів, яка увесь час доступна як студентам, так і викладачам.

Мета дослідження: створення мобільного застосунку для полегшення освітнього процесу студентів.

Для реалізації мети, виконувалися такі **завдання**:

- здійснити вибір та обгрунтувати сучасні засоби розробки мобільних додатків;
- розробити мобільний додаток «Довідник студента»

Об’єкт досліджнення: розробка та створення мобільного застосунку.

Предмет дослідження: створення мобільного застосунку для отримання відповідної інформації студентом під час освітнього процесу.

Структура роботи. Дипломна робота бакалавра складається зі вступу, двох розділів, списку використаних джерел.

РОЗДІЛ 1. СТВОРЕННЯ ПРОСТОГО ЗАСТОСУНКУ

1.1. Загальні відомості

Мобільний застосунок (Mobile app) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Розвиток ринку мобільних застосунків починається з появою у 1998 році протоколу WAP (Wireless Application Protocol). Саме він об’єднав інтернет і мобільний зв’язок. Тепер можна було вбудувати в телефон браузер, встановити з’єднання із серверами і отримати дані на пристрій. Одним з перших телефоном з WAP-браузером був Nokia 7100, який випустили в 1999 році. Тоді ж почали з’являтися компанії, які розробляють продукти спеціально для мобільних пристроїв.

WAP надав людям не тільки ігри, але і можливість читати новини з мобільного, користуватися електронною поштою, завантажувати карти та бронювати квитки. Для цього створювалися WAP-сайти зі спеціальною розміткою для мобільних екранів. Це були прості сторінки з тексту і посилань, майже без картинок.

У 2001 Symbian стала відкритою операційною системою, і в цей же час з’явився Nokia 7650, на якому можна було встановлювати застосунки від сторонніх розробників. Це повинно було стати проривом на ринку, але не спрацювало сповна через складнощі розробки та обмежені можливості смартфонів.

У цей же час розвивався ринок Java-застосунків. Розробка програми на Java займала менше часу і підходила для Windows Mobile, Android, bada, Palm OS і BlackBerry OS. У Symbian також підтримувалося підмножина Java – J2ME, але функціональність таких програм була сильно обмежена, тому розробкою на Java під Symbian практично ніхто не займався. У Nokia не прагнули допомагати розробникам розвивати ринок. У розробників не було нормального середовища для створення проєктів. Nokia випустили інструмент для розробки, але більша частина його можливостей була платною. Ліцензія

коштувала від 300 до 8000 євро, це сильно впливало на кінцеву вартість програми. У результаті ОС не змогла конкурувати з iOS і Android.

У 2007 році Стів Джобс представив світу перший iPhone. Архітектура iOS була схожа на MacOS, але система була повністю закритою. Джобс не хотів, щоб сторонні розробники могли розробляти програми для iOS, і не збирався відкривати SDK. Замість цього він хотів, щоб розробники створювали вебзастосунки, і дав можливість створювати браузерні закладки на домашньому екрані.

Але «допитливі користувачі» зламали файлову систему і почали писати інсталятори для нативних застосунків. Пізніше рада директорів Apple все ж переконала Джобса легалізувати сторонні застосунки. У підсумку в березні 2008 року iPhone SDK став доступний усім охочим, а в липні презентували App Store. App Store став поштовхом до розвитку індустрії розробки застосунків, але проблемою був Objective-C. Мало хто хотів витрачати час на вивчення нового синтаксису, адже пристрої на iOS займали ще дуже маленьку частку ринку.

Android – операційна система для мобільних пристроїв, заснована на ядрі Linux. Вона є однією з найпопулярніших ОС для смартфонів та планшетів. Також використовується в таких пристроях, як смарт-годинник, електронні книги, музичні плеєри та нетбуки. Її власником, і фактичним розробником, є компанія Google. Історія операційної системи Android починається у 2005 році. Тоді Google купила компанію Android inc, фактичного творця цієї ОС, та у 2008 році була офіційно представлена перша версія операційної системи – Android 1.0. Ця та кожна наступна версія отримувала своє кодове ім’я. Перша версія вийшла з назвою «Apple Pie» (Яблучний пиріг).

До переваг операційної системи Android відносяться:

– легка синхронізація пристроїв на Android один з одним або з іншими пристроями;

– відкритість (можливість встановлювати сторонні програми без використання магазину застосунків, що неможливо на інших платформах);

– наявність різних магазинів застосунків (крім Google Play);

– великий вибір смартфонів на Android (вона не є закритою, тому будь-який виробник може встановлювати її на свої смартфони). Тому на цій ОС можна знайти як бюджетні моделі, так і моделі преміум-класу.

До недоліків відносяться: швидка витрата заряду батареї (найпоширеніша причина критики операційної системи Android), відкритість (завдяки відкритості на Android існує порівняно велика кількість вірусів), розтрата трафіку (під час підключення до мережі система самостійно витрачає інтернет-трафік на свої потреби).

Також безумовно слід враховувати, що на сьогодні під управлінням ОС Android працюють більше 80% всіх смартфонів у світі.

Далі згадаємо популярні мови програмування для розробки мобільних застосунків для Android та iOS:

Java – строго типізована об’єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Платформа: Android, основна IDE: Android Studio.

Swift – мова, розроблена компанією Apple і призначена для розробки застосунків під iOS і OS X. Swift запозичив досить багато з C++ і Objective-C. Платформа: iOS, основна IDE: Xcode.

JavaScript – прототипно-орієнтована сценарна мова програмування. Найбільш широке застосування знайшла в браузерах як мова сценаріїв для додання інтерактивності вебсторінок, а також у кросплатформних фреймворків (React Native, Ionic, Sencha і та ін.). Платформа: iOS, Android і практично будь-яка інша.

C# – об’єктно-орієнтована мова програмування розробки застосунків для платформи Microsoft .NET Framework. В області розробки мобільних

застосунків використовується у фреймворку Xamarin. Платформа: iOS, Android, Windows 10, основна IDE: Visual Studio.

Objective-C – об’єктно-орієнтована мова програмування корпорації Apple, побудована на основі мови C і Smalltalk. Зараз її замінює новий і більш простий Swift. Проте, деякий час розробники на Objective-C будуть дуже затребувані на ринку. Платформа: iOS, macOS, watchOS і tvOS.

Одне із основних питань, що виникає на початку розробки мобільного застосунку, – створювати мобільний (адаптивний) вебсайт, нативний або гібридний застосунок. Від прийнятого рішення буде залежати подальший процес розробки. Для розуміння ключових відмінностей наведемо характеристику нативного та гібридного способів розробки.

Нативні (рідні) застосунки написані мовою програмування, специфічною для платформи, для якої вони розробляються. Зазвичай це Objective-C або Swift для iOS та Java або Kotlin для Android. Нативні застосунки зазвичай мають кращу продуктивність при рендерінгу і анімації, ніж гібридні програми. Але якщо вам необхідна реалізація як для Android, так і для iOS – необхідно буде створити два застосунки. Гібридний застосунок – це мобільний застосунок, який містить вебпредставлення (по суті, ізольований екземпляр браузера) для запуску вебзастосунку із використанням вбудованої оболонки, яка може взаємодіяти із платформою пристрою та вебпредставленням. Це означає, що вебзастосунки можуть працювати на мобільному пристрої, маючи доступ до, наприклад, камери або функцій GPS. Гібридні застосунки можливі завдяки створеним інструментам, які полегшують зв’язок між вебпредставленням і платформою. Ці інструменти не є частиною офіційних платформ iOS або Android, та є сторонніми інструментами, такими як Apache Cordova. Коли гібридний застосунок буде створено, він буде скомпільований, перетворивши ваш вебзастосунок у нативний застосунок.

Цей матеріал зупиняє свою увагу на створенні саме нативних застосунків під операційну систему Android мовою Java.

Платформа Android складається з багатьох компонентів. У неї входять базові застосунки (наприклад, Контакти), набір програмних інтерфейсів (API) для управління зовнішнім виглядом і поведінкою застосунків, а також багатьох допоміжних файлів і бібліотек (рис. 1.1).

Під час побудови застосунків доступні ті ж API, які використовуються базовими застосунками. За допомогою цих API ви керуєте зовнішнім виглядом і поведінкою своїх застосунків. Під інфраструктурою застосунків розташовується рівень бібліотек C і C++. Для роботи з ними використовуються API. Кожен Android-застосунок виконується в окремому процесі. У самій основі системи лежить ядро Linux. В Android воно забезпечує роботу драйверів, а також таких базових сервісів, як безпека і управління пам’яттю.

Рисунок 1.1 – Компоненти платформи Andoid

Пристрої на базі Android не запускають файли .class і .jar. Замість цього для підвищення швидкості та ефективності використання акумуляторів Android-пристрої використовують власні оптимізовані формати компільованого коду. Це означає, що ви не зможете скористатися звичайним середовищем розробки мовою Java – вам також знадобляться спеціальні інструменти для перетворення відкомпільованого коду в формат android,

установки його на Android-пристроях і налагодження програми, коли вона запрацює.

Усі ці інструменти входять до складу Android SDK. Пакет Android Software Development Kit (SDK) містить бібліотеки та інструменти, необхідні для розробки Android-застосунків.

Зокрема, до складу SDK входять:

– SDK Platform – окрема платформа для кожної версії Android;

– SDK Tools – інструменти відлагодження та тестування, а також інші корисні службові програми. Включає набір платформно-незалежних інструментів.

IntelliJ IDEA – одна з найпопулярніших інтегрованих середовищ розробки (IDE) для програмування на Java. Android Studio – версія IDEA, яка включає версію Android SDK і додаткові інструменти графічних інтерфейсів, що спрощують розробку застосунків. Крім редактора і доступу до інструментів та бібліотек Android SDK, Android Studio надає шаблони, що спрощують створення нових класів і застосунків, а також засоби для виконання таких операцій, як упаковка застосунків та їх запуск.

Що стосується запуску застосунку, є два варіанти. Варіант перший – запустити застосунок на фізичному пристрої. Варіант другий – скористатися емулятором Android, вбудованим в Android SDK. Емулятор дозволяє створити один або кілька віртуальних пристроїв Android (Android Virtual Device, AVD) і запустити застосунок в емуляторі так, мов він виконується на фізичному пристрої. За останні роки Google сильно попрацював над своїм емулятором і перетворив його в один із кращих інструментів для розробки: швидкий, гнучкий і корисний під час тестування й налагодженні програм. Емулятор Android може імітувати роботу смартфона, планшета, годинника Wear OS та пристроїв Android TV.

Для того, щоб створити новий AVD, потрібно запустити AVD Manager, вибравши в Android Studio в меню Tools – AVD Manager. Відкриється вікно менеджера, в якому буде відображатися список створених емуляторів (рис. 1.2).

Рисунок 1.2 – Список створених емуляторів

Щоб створити новий емулятор, потрібно натиснути на Create Virtual Device в Менеджері AVD. Відкриється вікно, в якому буде запропоновано вибрати тип пристрою і профіль (рис. 1.3).

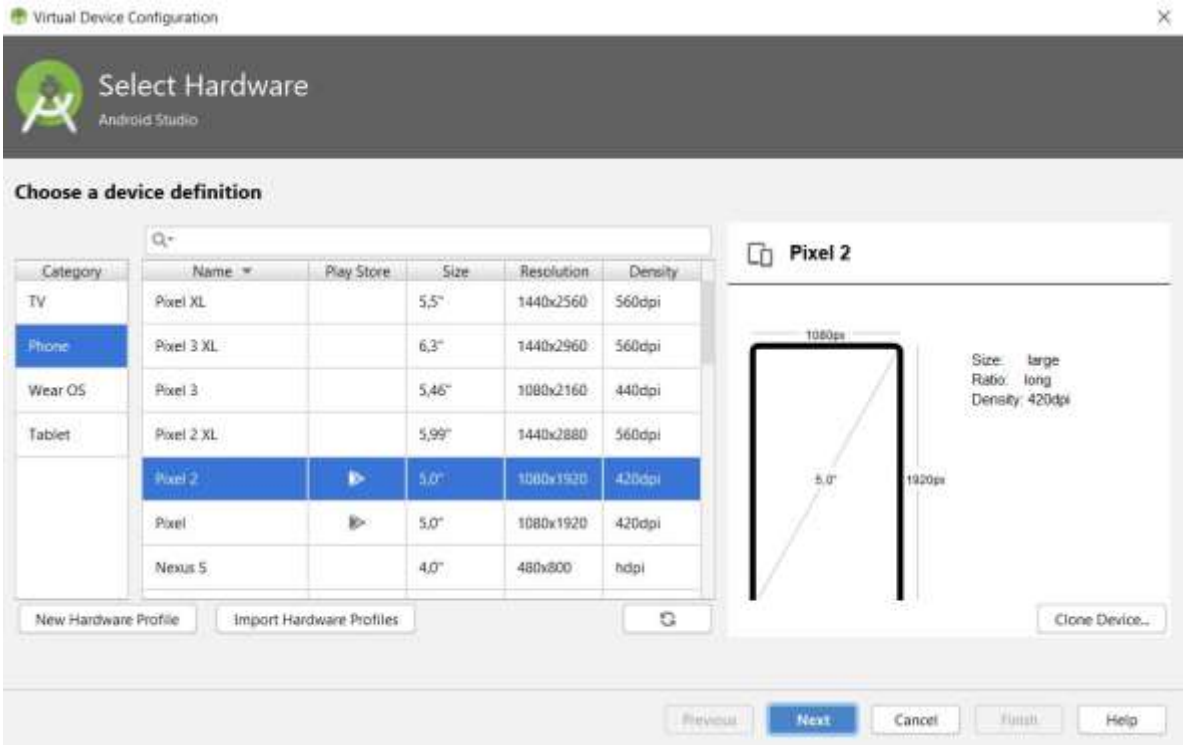


Рисунок 1.3 – Вибір типу пристрою

Після того, як буде обраний профіль, потрібно натиснути на Next для переходу далі. Тут потрібно вибрати, який образ системи використовувати (рис. 1.4).

Рисунок 1.4 – Вибір образу системи

На вкладці Recommended перераховані рекомендовані образи системи. Інші вкладки містять більш повний список доступних образів. Справа наводиться інформація про обраний образ (рівень API, версія). Зауважимо, що образи x86 працюють на емуляторі найшвидше. Рівень API важливий, оскільки якщо він буде менший, ніж той, що зазначений у маніфесті застосунку, застосунок не зможе бути встановлено на цей емулятор.

Якщо образ раніше не був завантажений, поруч з назвою з’явиться кнопка Download, натискання на яку почне процес завантаження. Для завантаження образу потрібно доступ до Інтернету.

Щоб перейти на наступний етап, потрібно натиснути Next. У новому вікні буде запропоновано змінити додаткові властивості AVD (рис. 1.5).

Після того, як AVD буде налаштований, залишиться тільки натиснути Finish. Створений AVD можна буде побачити у вікні Менеджера AVD. Протестувати додаток на емуляторі можна, натиснувши на кнопку Run в Android Studio (рис. 1.6).

Відкривається вікно Select Deployment Target, в якому буде запропоновано обрати, на якому пристрої потрібно запускати застосунок (рис. 1.7).

Рисунок 1.5 – Додаткові властивості AVD

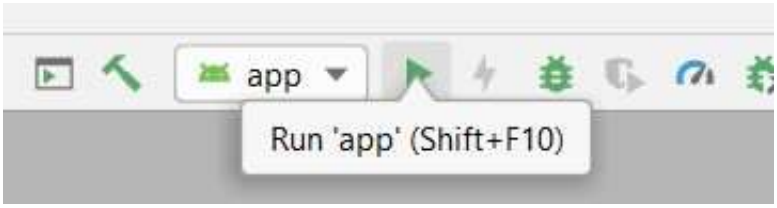


Рисунок 1.6 – Запуск емулятора

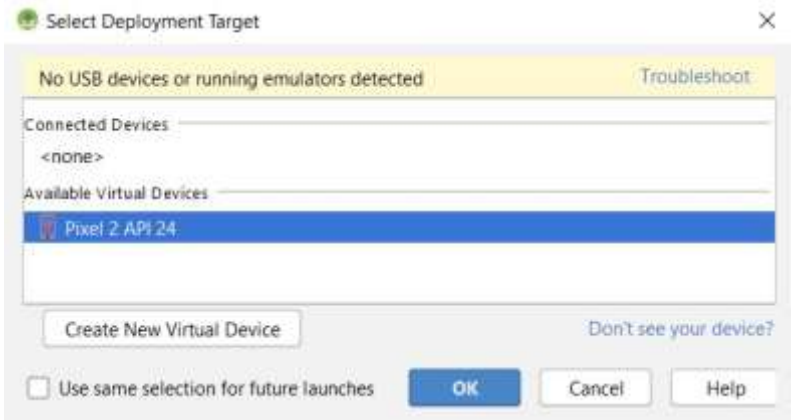


Рисунок 1.7 – Вибір емулятора для запуску застосунку

Після натискання OK почнеться запуск обраного емулятора, якщо він не запущений, або установка APK на емулятор.

Далі розглянемо створення тестового проєкту «Привіт, світ». Для цього запускаємо Android Studio та створюємо новий проєкт File –> New –> New Project (рис. 1.8).

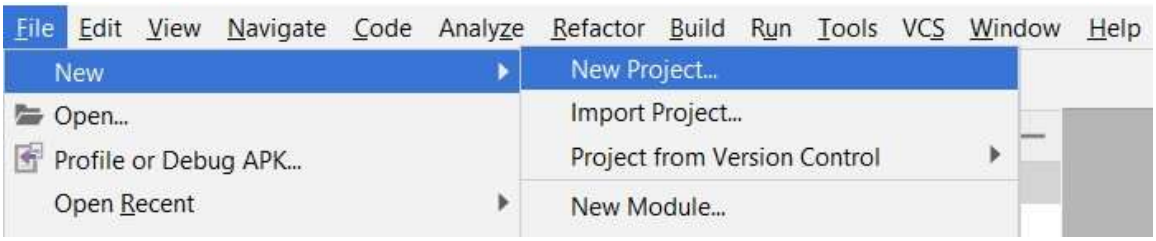


Рисунок 1.8 – Створення нового проєкту в Android Studio

Обираємо вкладку «Phone and tablet», тип активності «Empty activity» та натискаємо «next» (рис. 1.9).

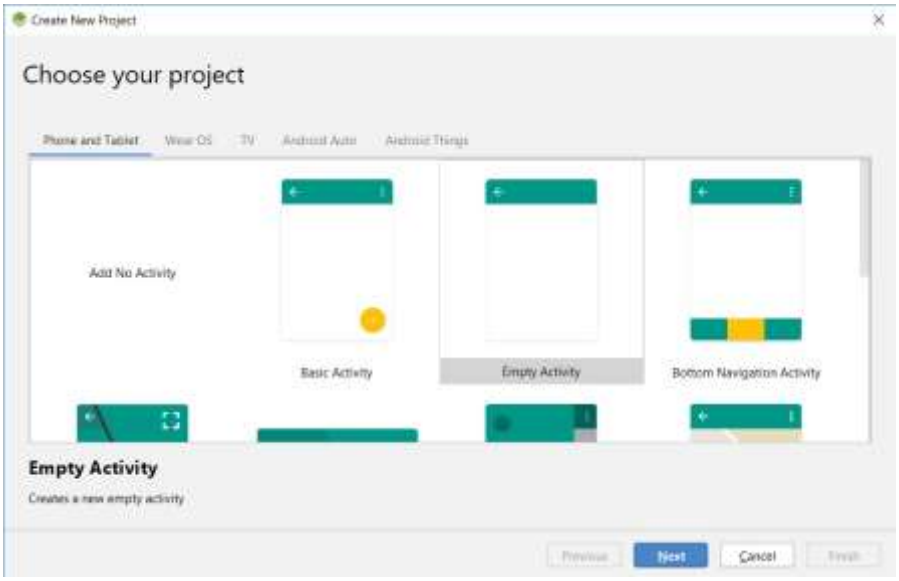


Рисунок 1.9 – Обрання типу активності

На наступному кроці необхідно задати ім’я проєкту, пакета, розташування файлів, мову розробки та мінімальну версію ОС, що буде підтримувати майбутній застосунок (рис. 1.10). Задамо ім’я «MyHello», пакет «com.chnulabs.myhello», мову розробки «Java» та minimum API level 19:Android 4.4. Ім’я пакета грає дуже важливу роль в Android, тому що воно використовується Android-пристроями для однозначної ідентифікації застосунку. Рівні API збільшуються з виходом кожної чергової версії Android.

Якщо тільки ви не хочете, щоб застосунок працював лише на найновіших пристроях, варто вибрати один із нижчих рівнів API.

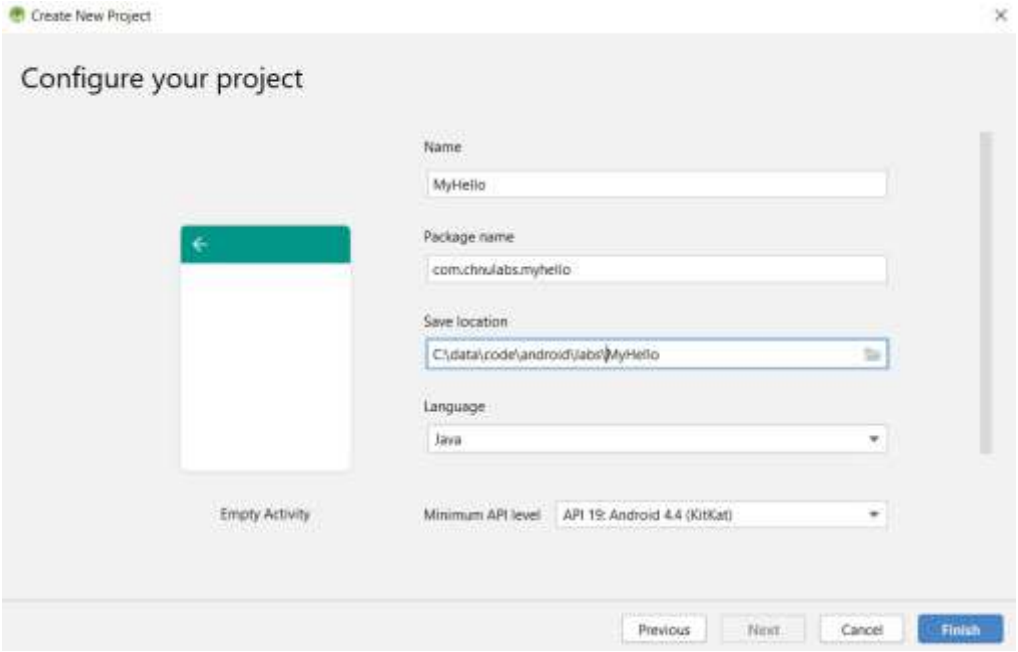


Рисунок 1.10 – Конфігурування проєкту

Кожен Android-застосунок складається з екранів, а кожен екран складається з активності і макета. Активність – одна чітко визначена операція, яку може виконати користувач. Наприклад, у застосунку можуть бути присутніми активності для складання повідомлення електронної пошти, знаходження контакту або створення знімка. Активності зазвичай асоціюються з одним екраном і програмуються на Java (або Kotlin). Макет описує зовнішній вигляд екрана. Макети створюються у вигляді файлів в розмітці XML і повідомляють Android, де розташовуються ті чи інші елементи екрана.

Розглянемо послідовність дій взаємодії пристрою Android, активності та макета:

– пристрій запускає застосунок і створює об’єкт активності;

– об’єкт активності визначає макет;

– активність дає команду на вивід макета на екран;

– користувач взаємодіє із макетом, що відображається на екрані пристрою;

– активність реагує на події макета та виконує відповідний код застосунку;

– у разі необхідності активність у відповідь на дії користувача оновлює дані макета;

– користувач бачить зміни на екрані пристрою.

Після створення нового проєкту (рис. 1.8–1.10) у ньому вже є одна активність із макетом. На рис. 1.11 можна побачити файли java\com.chnulabs.myhello\MainActivity (активність) та res\layout\activity_main.xml (макет).

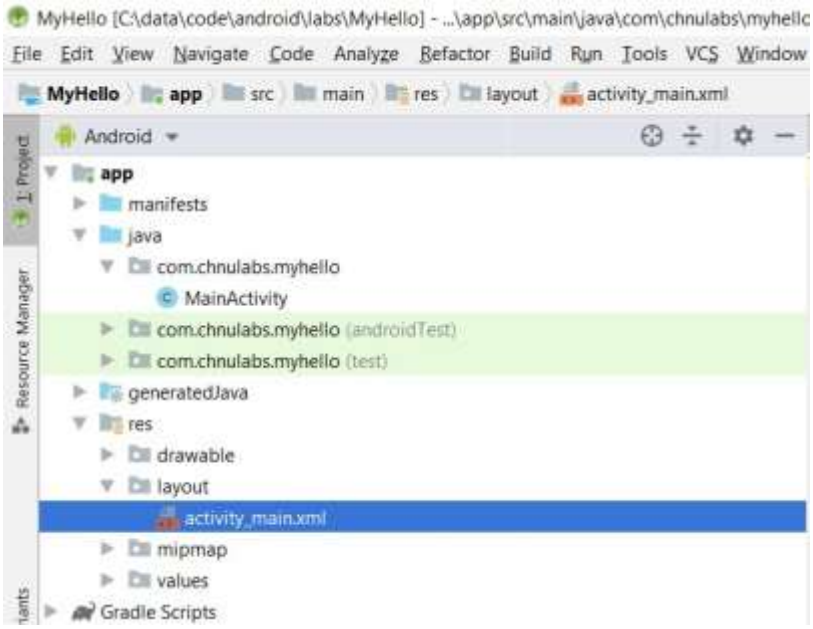


Рисунок 1.11 – Файли активності та макету

Для перегляду і зміни файлів використовуються різні редактори Android Studio. Зробіть подвійний клік на файлі, з яким ви хочете працювати; його вміст з’являється в середині вікна Android Studio (рис. 1.12).

Рисунок 1.12 – Файл MainActivity

Більшість файлів відображається в редакторі коду. По суті це звичайний текстовий редактор, але з підтримкою таких додаткових можливостей, як кольорове виділення синтаксису і перевірка коду.

Під час редагування макета з’являється додаткова можливість: замість редагування розмітки XML можна використовувати візуальний редактор (рис. 1.13). Візуальний редактор дозволяє перетягнути компоненти графічного інтерфейсу на макет і розмістити їх так, як ви вважаєте за потрібне. Редактор коду і візуальний редактор забезпечують різні представлення одного файлу, і ви можете перемикатися між ними (рис. 1.14).

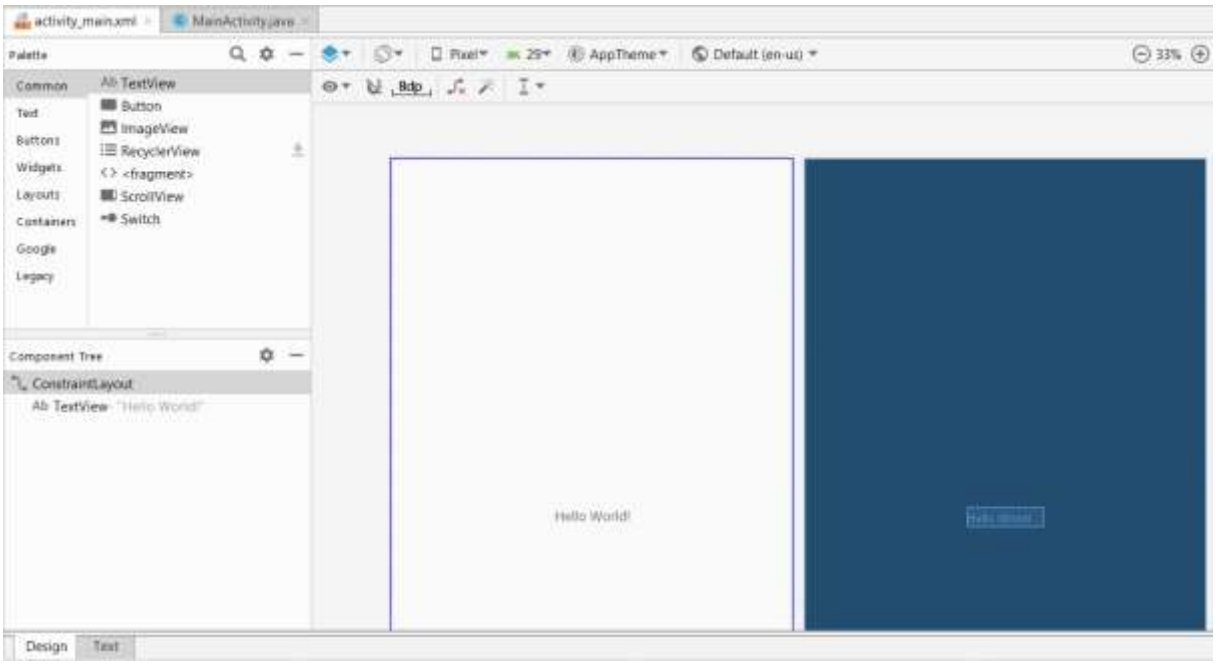


Рисунок 1.13 – Візуальний редактор xml-файлу

Рисунок 1.14 – Редактор коду xml-файлу

Змінимо розмір шрифту для елемента TextView нашого макета. Для цього на вкладці design виділимо його, у вікні властивостей знайдемо textSize та встановимо 36sp (рис. 1.15).

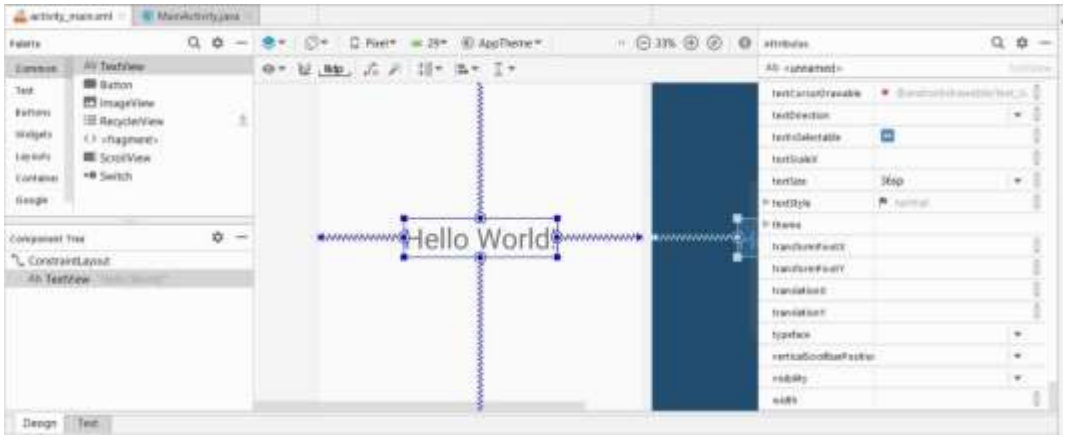


Рисунок 1.15 – Зміна розміру шрифту для елементу TextView

Після цього перейдемо у вкладку text та звернімо увагу на рядок «android:textSize="36sp"», що додався до описання елемента TextView (рис. 1.16)

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="36sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Рисунок 1.16 – Зміни у коді activity_main.xml

Виконаємо наш застосунок на віртуальному пристрої. Для цього виконуємо команду run app із меню run (рис. 1.17) або натиснувши run app на панелі інструментів.

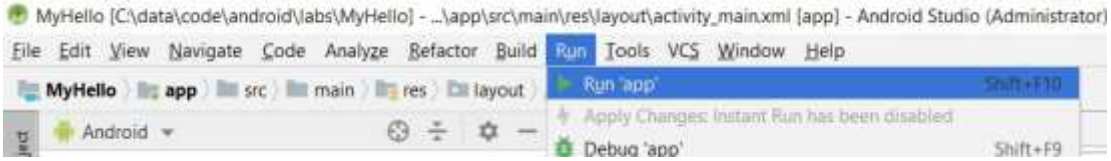


Рисунок 1.17 – Запуск застосунку на виконання

Рисунок 1.18 – Запуск застосунку на виконання через панель інструментів Обираємо віртуальний пристрій та натискаємо ok (рис. 1.19).

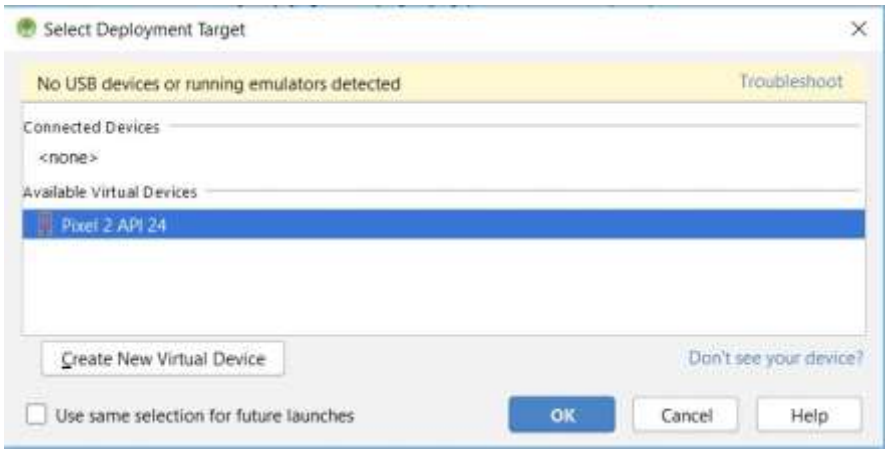


Рисунок 1.19 – Вибір віртуального пристрою

Після запуску емулятора та встановлення на нього нашого застосунку MyHello бачимо на ньому такий екран (рис. 1.20).

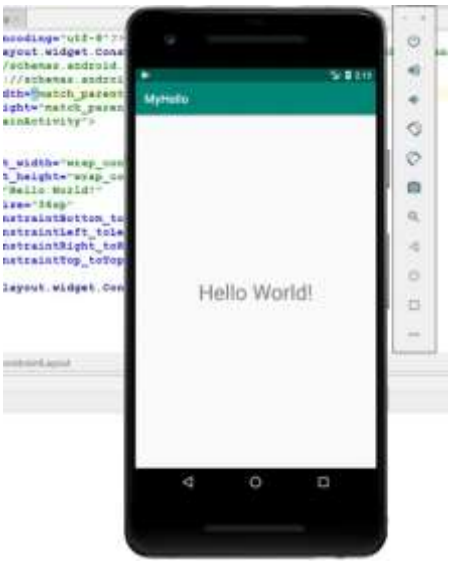


Рисунок 1.20 – Вигляд застосунку MyHello на віртуальному пристрої

Тепер розберемо покроково, що відбувалося після виконання команди run app:

– файли із вихідним кодом Java компілюються у байт-код;

– створюється android-застосунок у вигляді apk-файлу, в який включаються всі необхідні бібліотеки та ресурси;

– запускається емулятор (якщо він ще не запущений);

– після запуску емулятора apk-файл передається на пристрій;

– віртуальний пристрій запускає активність, що пов’язана із застосунком;

– активність визначає макет та відображає її на екран.

1.2. Взаємозв’язок макета та активності

Запускаємо Android Studio та створюємо новий проєкт empty activity «Students», ім’я пакета com.chnulabs.students, API – 19:4.4, мова – Java (рис. 1.21).

Відкриємо макет activity_main.xml у текстовому редакторі, змінимо «Hello world!» на «Hello students» та збільшимо розмір шрифту (рис. 1.22).

Запустимо застосунок (run app) та переконаємось, що текст у макеті змінився (рис. 1.23).

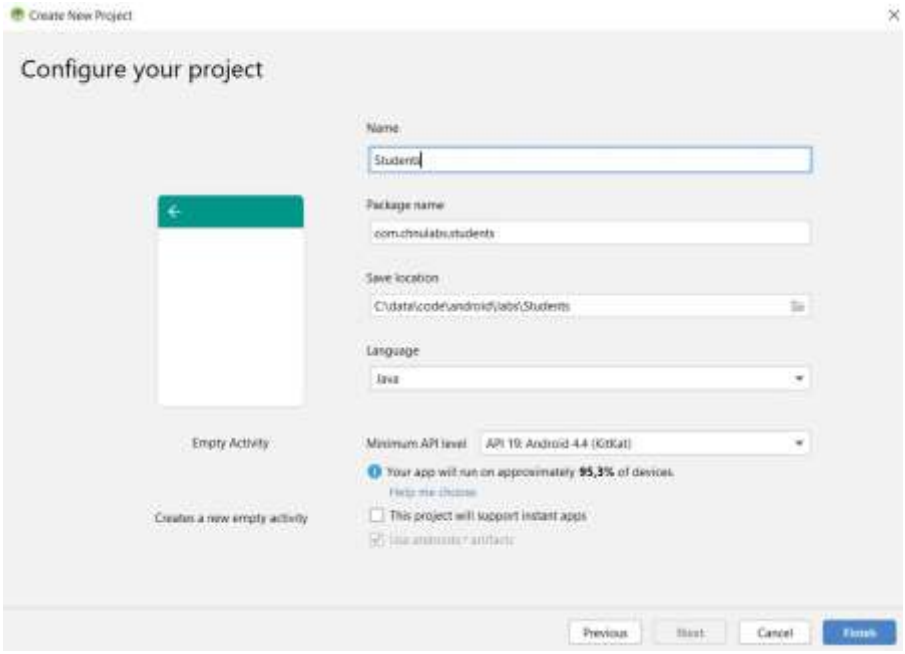


Рисунок 1.21 – Створення нового проєкту Students

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Students!"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 1.22 – Зміна тексту в TextView

Однак розміщення тексту безпосередньо у макеті не є гарним підходом. Це обумовлено декількома причинами, серед яких – необхідність зміни ідентичного тексту у різних макетах при можливій зміні користувацького інтерфейсу, або підтримка застосунком декількох мов.

Рисунок 1.23 – Оновлений текст макета активності

Більш правильно додати змінну до файлу strings.xml та використати її у макеті. У загальному випадку strings.xml включає всі строкові ресурси, необхідні макету, такі як заголовки, написи на кнопках, текстові повідомлення та ін.

Перейдемо до редагування файлу res/values/strings.xml та додамо туди рядок hello_text (рис. 1.24). Також змінимо назву застосунку на «Список студентів».

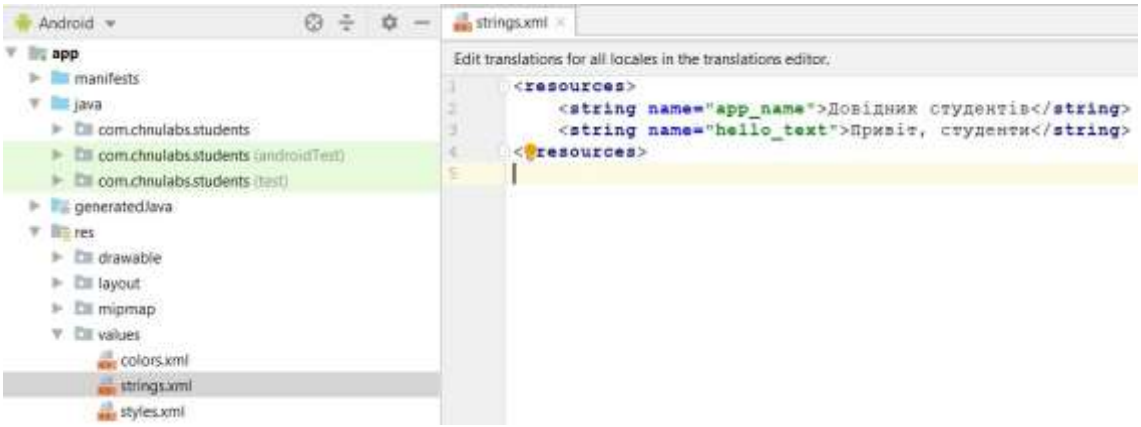


Рисунок 1.24 – Зміни у файлі strings.xml

Далі необхідно внести зміни у макет активності activity_main.xml для того, щоб він використовував дані hello_text, збережені у strings.xml (рис. 1.25).

Запустимо застосунок та переконаймося, що текст макета змінився відповідно до змін у файлі strings.xml (рис. 1.26).

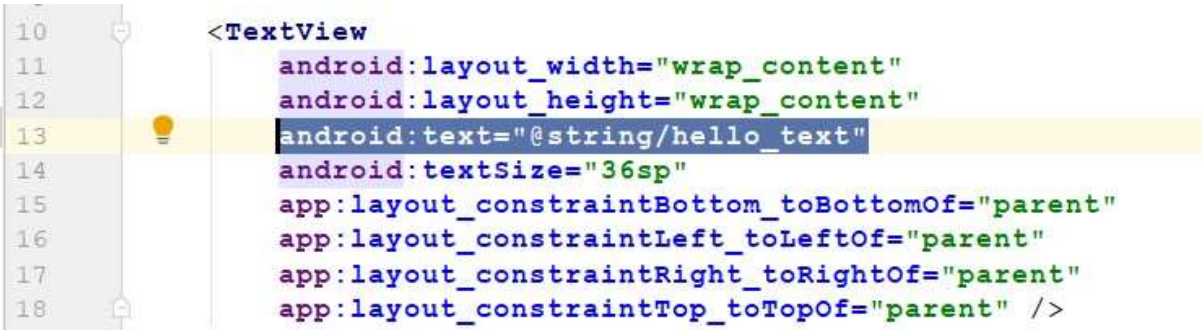


Рисунок 1.25 – Зміни у макеті activity_main.xml

Рисунок 1.26 – Перевірка зміни тексту у макеті активності

Далі змінимо наш макет та активність для відображення списку студентів за студентськими групами. Почнемо із реалізація макета. У макеті активності activity_main.xml у текстовому редакторі змінимо «ConstraintLayout», запропонований Android Studio за замовчуванням, на більш простий «LinearLayout». Він використовується для виведення компонентів графічного інтерфейсу поруч один з одним, по вертикалі або по горизонталі. Якщо компоненти шикуються по вертикалі, вони утворюють стовпець, а якщо по горизонталі – рядок. Також приберемо із TextView частину атрибутів, характерних для використання у контексті «ConstraintLayout» (рис. 1.27).

Використаємо візуальний редактор та додамо до макета випадний список Spinner. Для цього перейдемо на вкладку «design», у вікні «Palette» оберемо розділ «containers» та елемент управління Spinner. Перетягнемо його мишею до екрана макета, розмістивши перед текстом TextView (рис. 1.28).

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_text"
        android:textSize="36sp"/>

</LinearLayout>
```

LinearLayout

Design Text

Рисунок 1.27 – Встановлення LinearLayout для макету активності

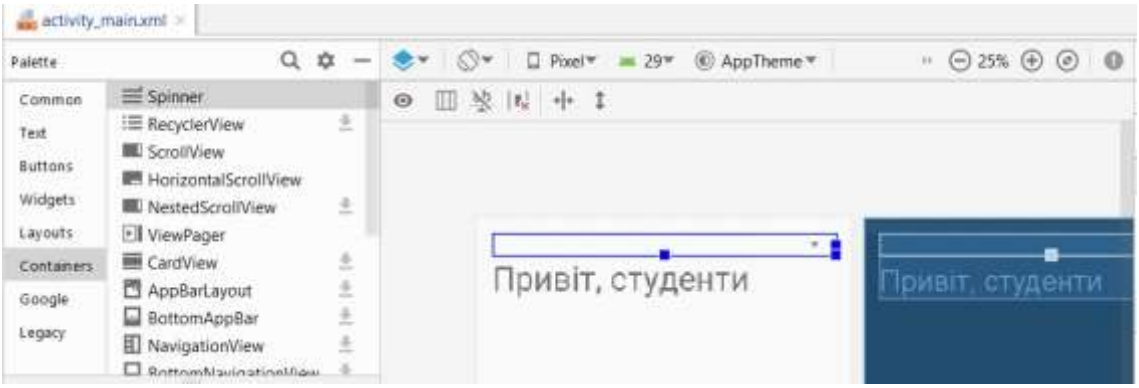


Рисунок 1.28 – Додавання елементу Spinner

Таким самим чином додаємо кнопку Button із розділу «buttons». Розмістимо її між елементом Spinner та TextView (рис. 1.29).

Рисунок 1.29 – Додавання кнопки

Перейдемо до вкладки «Text» редактора макета та виконаємо деякі налаштування властивостей елементів макета. Так, для Spinner змінимо android:layout_width="wrap_content" для того, щоб елемент займав стільки місця, скільки необхідно для вміщення контенту, та додамо відступи зверху та знизу, а також вирівнювання по центру (рис. 1.30).

```
11      <Spinner
12          android:id="@+id/spinner"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:layout_gravity="center"
16          android:layout_marginTop="40dp"
17          android:layout_marginBottom="16dp"/>
18
```

Рисунок 1.30 – Зміни властивостей елементу управління Spinner

Для кнопки додаємо відступ знизу та візьмемо текст напису із strings.xml (рис. 1.31).

```
19      <Button
20          android:id="@+id/button"
21          android:layout_width="match_parent"
22          android:layout_height="wrap_content"
23          android:text="@string/btn_text"
24          android:layout_marginBottom="16dp"/>
```

Рисунок 1.31 – Зміни властивостей кнопки

Зараз значення android:text підсвічено червоним кольором, оскільки ми ще не додали відповідне значення до strings.xml. Виправимо це, також додавши значення тексту за замовчуванням для TextView. Текст «Привіт, студенти» надалі не знадобиться, тому приберемо його (рис. 1.32).

strings.xml | activity_main.xml

Edit translations for all locales in the translations editor.

```
1   <resources>
2       <string name="app_name">Довідник студентів</string>
3       <string name="default_text">Список студентів</string>
4       <string name="btn_text">Показати список</string>
5   </resources>
```

Рисунок 1.32 – Зміни у res/values/strings.xml

Для TextView аналогічно вкажемо android:layout_width="wrap_ content". Також змінимо значення тексту із hello_text на default_text та трохи зменшимо розмір шрифту (рис. 1.33).

```
26      <TextView
27          android:layout_width="wrap_content"
28          android:layout_height="wrap_content"
29          android:text="@string/default_text"
30          android:textSize="18sp" />
31
```

Рисунок 1.33 – Зміни в атрибутах TextView

Той факт, що кнопки, написи та інші елементи мають так багато спільних властивостей, цілком логічний – всі ці компоненти успадковують від одного класу Android View.

Виконаємо застосунок та переглянемо результат в емуляторі (рис. 1.34).

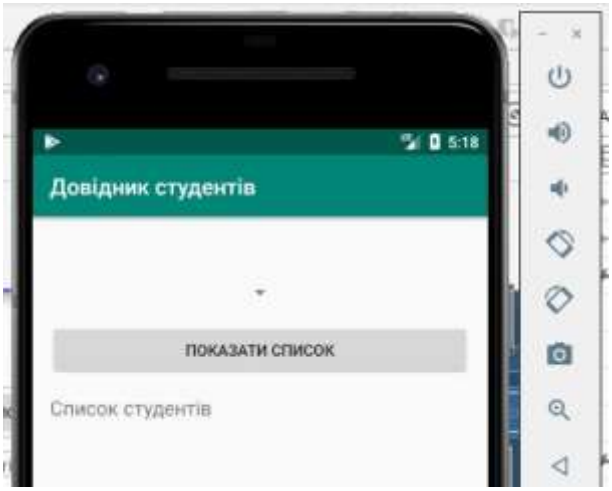


Рисунок 1.34 – Зовнішній вигляд макету активності

Далі необхідно наповнити елемент Spinner значеннями студентських груп. Дана задача буде реалізована через використання можливості збереження масивів рядків у strings.xml. Перейдемо до res/values/strings. xml та додамо список студентських груп (рис. 1.35).

Використаємо створений масив, підключивши його до елемента Spinner у макеті активності activity_main.xml. Для цього у Spinner додаємо атрибут android:entries (рис.1. 36).

Перевіримо результат виконаних змін, запустивши застосунок (рис. 1.37).

strings.xml

Edit translations for all locales in the translations editor.

```
<resources>
    <string name="app_name">Довідник студентів</string>
    <string name="default_text">Список студентів</string>
    <string name="btn_text">Показати список</string>
    <string-array name="students_groups">
        <item>301</item>
        <item>302</item>
        <item>308</item>
        <item>309</item>
    </string-array>
</resources>
```

Рисунок 1.35 – Збереження масиву рядків в strings.xml

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="40dp"
    android:layout_marginBottom="16dp"
    android:entries="@array/students_groups"/>
```

Рисунок 1.36 – Зв’язок Spinner із string-array

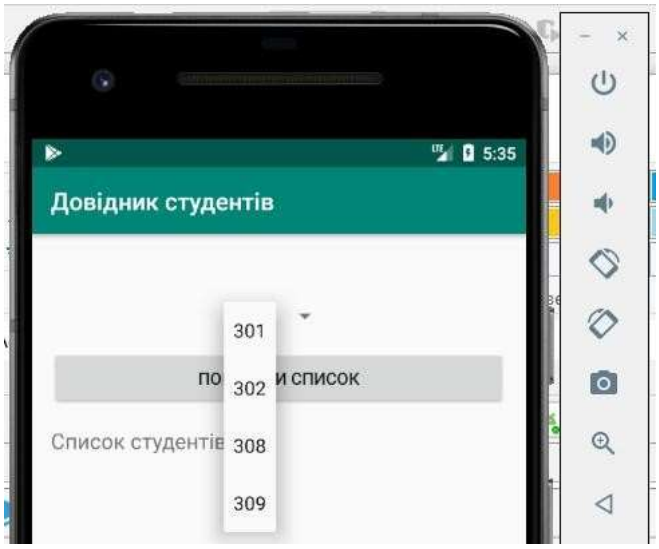


Рисунок 1.37 – Вибір групи в елементі Spinner

Реалізуємо обробку події натискання на кнопку. Для цього перейдемо до файлу активності MainActivity.java та створимо метод onBtnClick, що буде викликатись по натисканню на кнопку. Цей метод має вхідний параметр типу View (рис. 1.38).

```
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

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

    public void onBtnClick(View view) {

    }
}
```

Рисунок 1.38 – Додавання методу onBtnClick до активності

Пов’яжемо цей метод із подією натискання на кнопку. Для цього у макеті activity_main.xml у компоненті Button пропишемо атрибут «android:onClick="onBtnClick"» (рис. 1.39).

Рисунок 1.39 – Прив’язка методу активності до події макета

Для перевірки роботи необхідно виконати у методі активності onBtnClick якусь дію. Використаємо клас-віджет Toast (рис. 1.40). Це просте спливаюче повідомлення, яке з’являється на екрані. Повідомлення виконують чисто інформаційні функції, користувач не може з ними взаємодіяти. Поки повідомлення знаходиться на екрані, активність залишається видимою і доступною для взаємодії з користувачем. Повідомлення автоматично закривається після закінчення часу очікування. Щоб створити повідомлення, викличте метод Toast.makeText() і передайте йому три параметра: Context (зазвичай для поточної активності), CharSequence (виведене повідомлення) та int (тривалість). Після того, як об’єкт повідомлення буде створений, його можна вивести на екран викликом методу show().

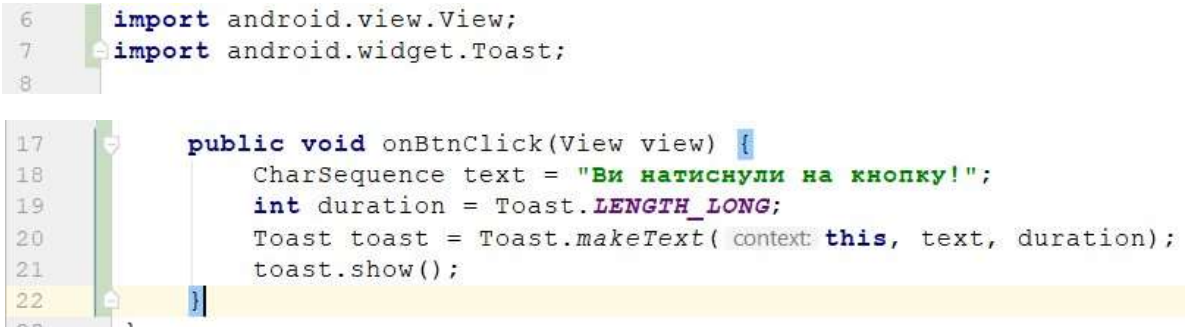


Рисунок 1.40 – Виведення повідомлення по натисканню на кнопку

Перевіримо отриманий результат, запустивши застосунок в емуляторі (рис. 1.41)

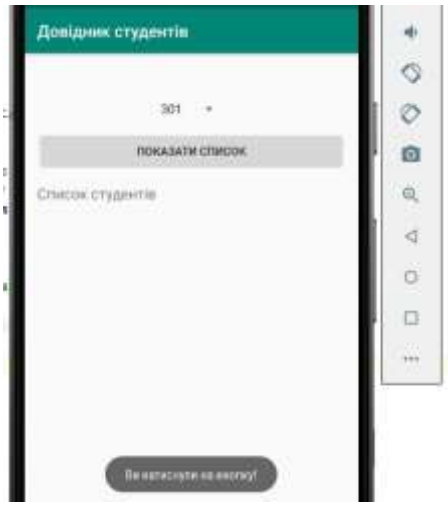


Рисунок 1.41 – Виведення повідомлення по натисканню на кнопку

Тепер залишилось по натисканню на кнопку Button реалізувати виведення в компонент TextView списку студентів обраної в Spinner групи. Почнемо зі створення класу Student, що визначатиме список студентів та набір базових методів по роботі зі студентом. Для цього у пакеті com.chnulabs.students створимо новий java-класс Student New–>Java class (рис. 1.42–1.43).

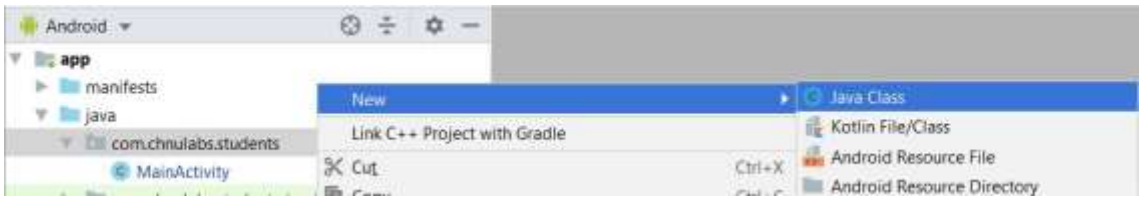


Рисунок 1.42 – Створення нового java-класу Student

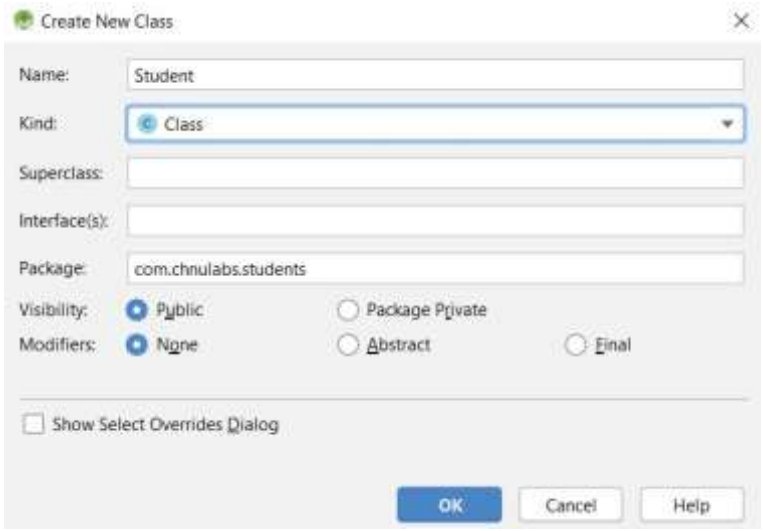


Рисунок 1.43 – Вказання назви класу

Створений клас Student міститиме 2 приватних поля: name – ім’я студента та groupNumber – номер групи; конструктор для створення екземпляра класу з 2-ма відповідними параметрами та методи-гетери для отримання значень цих полів. Крім того, у класі визначено статичну колекцію студентів та метод для отримання із неї студентів відповідної групи (рис. 1.44).

Student.java

```
package com.chnulabs.students;

import java.util.ArrayList;
import java.util.Arrays;

public class Student {
    private String name;
    private String groupNumber;

    public Student(String name, String groupNumber) {
        this.name = name;
        this.groupNumber = groupNumber;
    }

    public String getName() {
        return name;
    }

    public String getGroupNumber() {
        return groupNumber;
    }

    private final static ArrayList<Student> students = new ArrayList<Student>(
        Arrays.asList(
            new Student( name: "Іванов Роман",  groupNumber: "301"),
            new Student( name: "Петров Федір",  groupNumber: "301"),
            new Student( name: "Осадча Оксана",  groupNumber: "302"),
            new Student( name: "Максимов Руслан",  groupNumber: "302"),
            new Student( name: "Смірнов Василь",  groupNumber: "308"),
            new Student( name: "Потапова Марія",  groupNumber: "309"),
            new Student( name: "Гонський Іван",  groupNumber: "309"),
            new Student( name: "Васильев Максим",  groupNumber: "309")
        )
    );

    public static ArrayList<Student> getStudents(String groupNumber) {
        ArrayList<Student> stList = new ArrayList<>();
        for(Student s: students) {
            if (s.getGroupNumber().equals(groupNumber)) {
                stList.add(s);
            }
        }
        return stList;
    }

}
```

Рисунок 1.44 – Клас Student

Перейдемо до макета та перевіримо наявність ідентифікаторів у компонент. Це необхідно для можливості звернення до них із коду активності. На цей момент компонент TextView не має ідентифікатора, виправимо це, додавши до нього атрибут android:id="@+id/text" (рис. 1.45).

```
28      <TextView
29          android:id="@+id/text"
30          android:layout_width="wrap_content"
31          android:layout_height="wrap_content"
32          android:text="Список студентів"
33          android:textSize="18sp" />
```

Рисунок 1.45 – Додавання ідентифікатора до textView

Залишається змінити метод активності onBtnClick для виведення у макет списку студентів обраної групи.

Для отримання посилання на компонент графічного інтерфейсу можна скористатися методом findViewById(). Метод findViewById() отримує ідентифікатор компонента у вигляді параметра і повертає об’єкт View. Далі залишається привести значення, що повертається до правильного типу компонента (наприклад, TextView або Button). Передаючи ідентифікатор компонента, буде використано спеціальний клас R, який генерується інструментарієм Android під час створення або побудови програми. Він знаходиться в папці app/build/generated/source/r/debug вашого проєкту – всередині папки, ім’я якої збігається з ім’ям пакета програми. Android використовує R.java для відстеження ресурсів, використовуваних у застосунку; серед іншого, цей клас дозволяє отримувати посилання на компоненти графічного інтерфейсу з коду активності.

Нижче, на рис. 1.46 наведено оновлений код методу onBtnClick.

```
19      public void onBtnClick(View view) {
20  //          CharSequence text = "Ви натистули на кнопку!";
21  //          int duration = Toast.LENGTH_LONG;
22  //          Toast toast = Toast.makeText(this, text, duration);
23  //          toast.show();
24
25          Spinner spinner = (Spinner) findViewById(R.id.spinner);
26          String grpNumb = (String) spinner.getSelectedItem();
27
28          String txtStudents = "";
29          for(Student s: Student.getStudents(grpNumb)) {
30              txtStudents += s.getName() + "\n";
31          }
32
33          TextView textView = (TextView) findViewById(R.id.text);
34          textView.setText(txtStudents);
35      }
```

Рисунок 1.46 – Код методу onBtnClick

Ми коментуємо попередній варіант коду, де виводилось повідомлення про натискання на кнопку, та додаємо новий. Тут, за допомогою методу findViewById отримується компонент spinner, з якого через метод getSelectedItem у змінну grpNumb записується значення обраної групи. Далі у циклі перебираємо всіх студентів отриманої групи, що повертає getStudents, та додаємо їх імена (отримані через getName()) до змінної txtStudents. Після цього, через findViewById отримуємо компонент textView та встановлюємо йому текст – сформований список студентів обраної групи.

Перевіримо роботу застосунку, запустивши app run (рис. 1.47). У випадному списку виберемо якусь групу та натиснемо кнопку «Показати список».

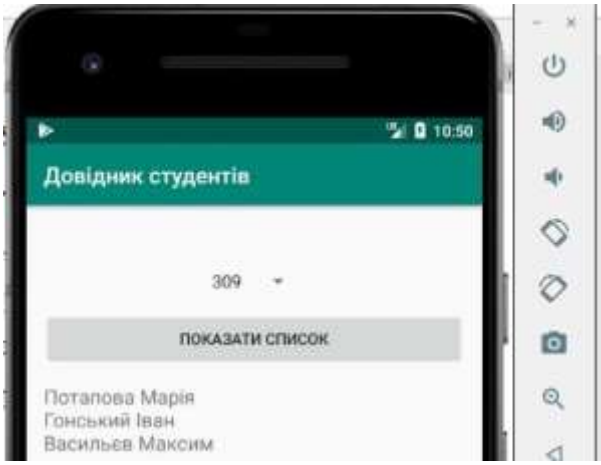


Рисунок 1.47 – Перевірка роботи застосунку «Довідник студентів»

1.3. Інтенти та передача даних між активностями

Активність – одна чітко визначена операція, яка може виконуватися користувачем, наприклад, відображення списку студентських груп. У дуже простому застосунку може бути достатньо однієї активності. Але зазвичай користувачеві потрібно виконувати більше однієї операції – наприклад, не тільки виводити список студентів, але і додавати до нього нових. У таких випадках в застосунку використовуються різні активності: одна, наприклад, для відображення списку студентських груп, а інша – для перегляду студентів.

Розберемо, що необхідно зробити для реалізації цієї задачі:

– додати до застосунку другу активність та макет;

– організувати виклик другої активності із першої;

– організувати передачу даних щодо поточної студентської групи із першої активності до другої;

– перенести логіку щодо отримання та виведення студентів із першої активності до другої.

Під час старту застосунку пристрій завантажуватиме головну активність, яка відображатиме на екрані свій макет. Далі по натисканню на кнопку завантажуватиметься друга активність, що замінюватиме користувацький екран власним макетом. Повернення до головної активності буде можливим за допомогою використання стандартної кнопки пристрою «назад».

Для додавання нової активності виділимо пакет com.chnulabs.students та оберемо New –> Activity –> Empty activity (рис. 1.48).

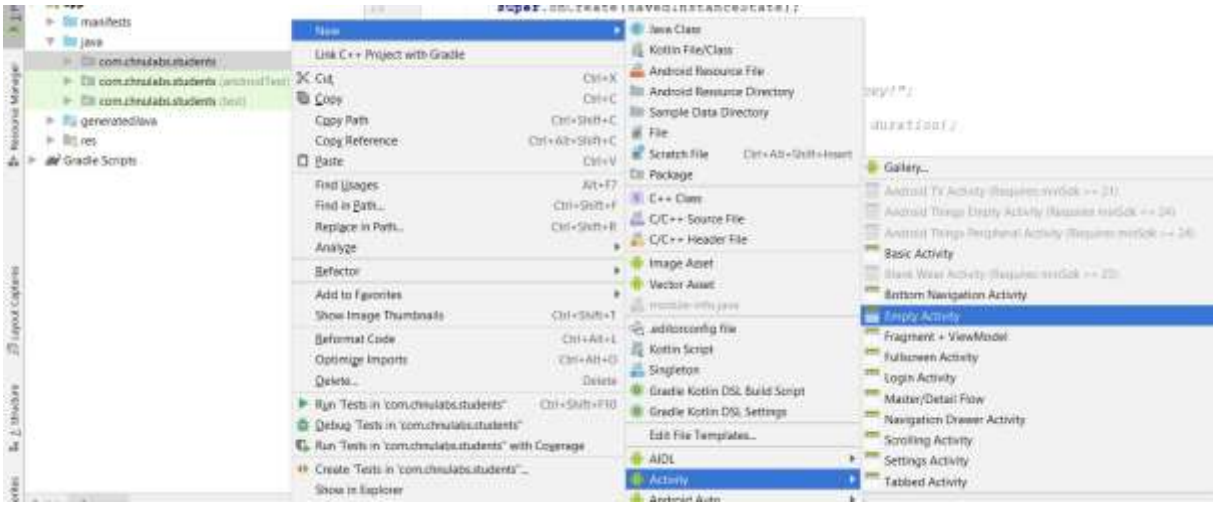


Рисунок 1.48 – Створення Empty activity

Далі вводимо ім’я активносні StudentsListActivity та залишаємо прапорець «Create layout file» (рис. 1.49).

```
activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="40dp"
        android:layout_marginBottom="16dp"
        android:entries="@array/students_groups"/>
```

Рисунок 1.49 – Створення Empty activity (продовження)

Оновимо макет активності – встановимо LinearLayout та перенесемо туди компонент TextView із макета головної активності (рис. 1.50.)

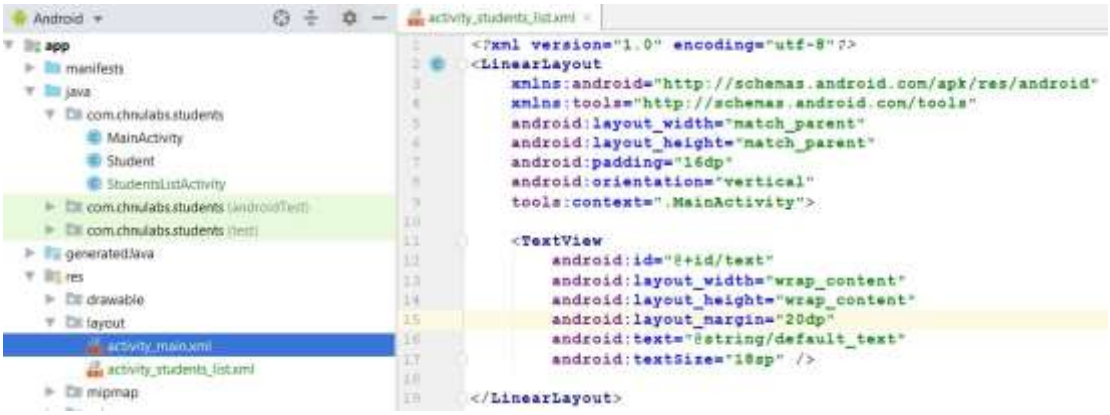


Рисунок 1.50 – Макет активності

Після видалення компонента TextView із макета головної активності він матиме такий вигляд (рис. 1.51):

```
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Показати список"
        android:layout_marginBottom="16dp"
        android:onClick="onBtnClick"/>

</LinearLayout>
```

Рисунок 1.51 – Макет головної активності

Кожен Android-застосунок повинен містити файл з ім’ям AndroidManifest.xml. Ви знайдете його в папці app/src/main свого проєкту. Файл AndroidManifest.xml містить найважливішу інформацію про програму: які активності вона містить, які бібліотеки їй необхідні та інші оголошення. Android створює цей файл під час створення програми. Наш екземпляр AndroidManifest.xml виглядає таким чином (рис. 1.52):

Рисунок 1.52 – AndroidManifest.xml нашого застосунку

Усі активності повинні бути оголошені у файлі AndroidManifest.xml. Якщо активність не оголошена у файлі, то система не знатиме про його існування. А якщо система не знає про активність, то активність не буде виконуватися. Активності оголошуються в маніфесті включенням елемента <activity> в елемент <application>.

На цей момент наш проєкт містить дві активності, одна з яких (MainActivity) зазначена у AndroidManifest.xml як головна та як launcher нашого застосунку. Наступний крок – змусити MainActivity викликати StudentsListActivity, коли користувач натисне на кнопку «Показати список».

Щоб запустити одну Активність з іншої, будемо використовувати інтент. Інтент можна розглядати як свого роду «намір виконати якусь операцію». Це різновид повідомлення, що дозволяє зв’язати різнорідні об’єкти (наприклад, активності) на стадії виконання. Якщо одна активність хоче запустити іншу, вона відправляє для цього інтент системі Android.

Android запускає другу активність і передає їй інтент. Перейдемо до коду головної активності, а саме – методу onBtnClick.

Видалимо весь попередній код, попередньо зберігши його десь у іншому місці (він нам знадобиться надалі у процесі реалізації другої активності). Замість нього наберемо такий код (рис. 1.53).

Рисунок 1.53 – Запуск активності StudentsListActivity у методі onBtnClick

Конструктор інтенту має два параметри. Перший параметр повідомляє Android, від якого об’єкта надійшов інтент; для позначення поточної активності використовується ключове слово this. У другому параметрі передається ім’я класу активності, яка повинна отримати інтент.

Після того, як інтент буде створений, він передається Android викликом startActivity. Цей виклик наказує Android запустити активність, яка визначається інтентом. Під час отримання інтенту Android переконується в тому, що все правильно, і наказує активності запуститися. Якщо знайти активність не вдалося, ініціюється виключення ActivityNotFoundException. Перевіримо роботу застосунку, запустивши run app (рис. 1.54).

Спочатку запускається головна активність та відображає макет із списком студентських груп та кнопкою, але вже без списку студентів. Далі по натисканню на кнопку «Показати список» завантажується друга активність та відображає макет із TextView для списку студентів. На цьому етапі він містить текст за замовчуванням, вказаний у strings.xml.

Для того, щоб реалізувати у другій активності механізм отримання та відображення списку студентів обраної групи, необхідно повідомити другій активності, яка група буда обрана у макеті першої.

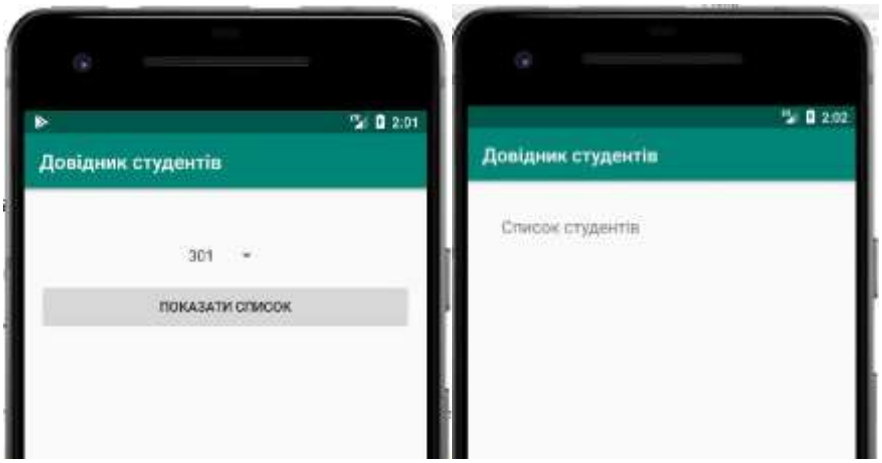


Рисунок 1.54 – Перехід між активностями

У інтент можна додати додаткову інформацію, яка повинна передаватися одержувачу. В цьому випадку активність, яка отримала інтент, зможе на нього якось зреагувати. Для цього використовується метод putExtra(), що має 2 параметри. Перший – ім’я ресурсу для переданої інформації, а другий – його значення. Перевантаження методу putExtra() дозволяє передавати значення багатьох можливих типів. Наприклад, це може бути примітив (скажімо, boolean або int), масив примітивів або String. Багаторазові виклики putExtra() дозволяють включити в інтент набір даних.

Для отримання переданих даних з боку другої активності необхідно спочатку отримати інтент за допомогою методу getIntent(), після чого до отриманого екземпляра можна застосувати методи getStringExtra (або getIntExtra чи ін.). Цей метод має один параметр – ім’я ресурсу.

Реалізуємо передачу значення обраної групи із першої активності в другу. Почнемо із визначення назви ресурсу, що буде передаватись. Зробимо це у класі другої активності StudentsListActivity (рис. 1.55).

Повертаємось до методу onBtnClick та передамо значення обраної групи до інтенту (рис. 1.56).

Тепер реалізуємо прийом переданої групи у другій активності. Для цього розмістимо відповідний код у методі onCreate активності StudentsList Activity. Повний код класу StudentsListActivity наведено на рис. 1.57.

StudentsListActivity.java

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class StudentsListActivity extends AppCompatActivity {

    public static final String GROUP_NUMBER = "groupnumber";

```

Рисунок 1.55 – Визначення назви ресурсу – номера групи

```
    public void onBtnClick(View view) {
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        String grpNumb = (String) spinner.getSelectedItem();

        Intent intent = new Intent( packageContext: this, StudentsListActivity.class);
        intent.putExtra(StudentsListActivity.GROUP_NUMBER, grpNumb);

        startActivity(intent);
    }
```

Рисунок 1.56 – Передача обраної студентської групи до інтенту

StudentsListActivity.java

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class StudentsListActivity extends AppCompatActivity {

    public static final String GROUP_NUMBER = "groupnumber";

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

        Intent intent = getIntent();
        String grpNumber = intent.getStringExtra(GROUP_NUMBER);

        String txtStudents = "";
        for(Student s: Student.getStudents(grpNumber)) {
            txtStudents += s.getName() + "\n";
        }

        TextView textView = (TextView) findViewById(R.id.text);
        textView.setText(txtStudents);
    }
}
```

Рисунок 1.57 – Клас StudentsListActivity

Перевіримо роботу нашого коду, запустивши застосунок на виконання (рис. 1.58).

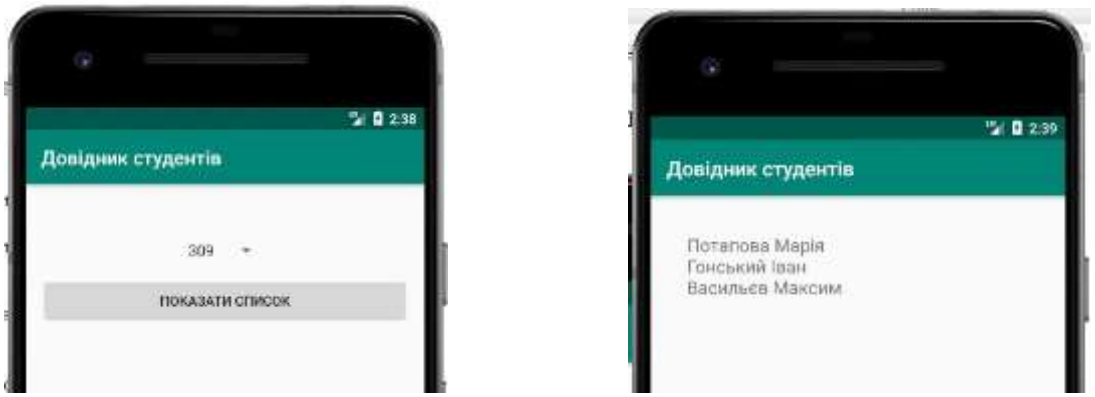


Рисунок 1.58 – Передача даних у інтент та їх обробка

Як вже зазначалося, Android-застосунки складаються з однієї або декількох активностей. Кожна активність представляє одну чітко визначену операцію, яка може виконуватися користувачем. Наприклад, такі застосунки, як Gmail, Viber, Facebook і Twitter, містять активності, що дозволяють відправляти повідомлення, хоча в кожному застосунку ця операція виконується по-своєму.

Ми вже використали інтент для запуску другої активності. Цей принцип стосується і активностей інших застосунків. Активність нашого застосунку передає інтент Android, Android перевіряє його, а потім наказує другій активності запуститися – незважаючи на те, що ця активність знаходиться в іншому застосунку. Наприклад, можна скористатися інтентом для запуску активності Gmail, що відправляє повідомлення, і передати їй текст, який потрібно відправити. Замість того, щоб писати власні активності для відправки електронної пошти, можна скористатися готовим застосунком Gmail. Це означає, що об’єднуючи активності на пристрої в ланцюжок, ви можете будувати програми, що мають значно більшу функціональність.

Ми не можемо знати, які програми встановлені на пристрої, але ця проблема вирішується за допомогою дій (actions). Дії – стандартний механізм, за допомогою якого Android дізнається про те, які стандартні операції можуть виконуватися активностями. Наприклад, Android знає, що всі активності, зареєстровані для дії send, можуть відправляти повідомлення. Тобто, якщо потрібно виконати деяку дію і нас не цікавить, якою саме активністю вона буде

виконана, ми створюємо неявний інтент та повідомляємо Android, яку дію потрібно виконати. Вибір активності, яка виконує цю дію, доручаються Android.

Для створення інтенту із зазначенням дії застосовується синтаксис Intent intent = new Intent(дія), де дія – тип дії, що виконується активністю. Додамо до активності StudentsListActivity кнопку, за натисканням якої список студентів буде відправлятись у вигляді повідомлення. Для цього додаємо до макета activity_students_list.xml кнопку (рис. 1.59).

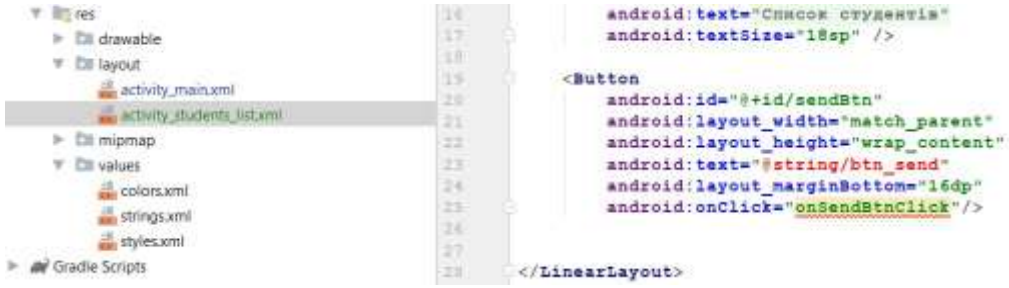


Рисунок 1.59 – Додавання кнопки до макета activity_students_list.xml

Визначимо значення btn_send у strings.xml (рис. 1.60) та пропишемо метод onSendBtnClick в класі активності StudentsListActivity.java (рис. 1.61).

```
        <item>309</item>
    </string-array>
    <string name="btn_send">Відправити повідомленням</string>
</resources>
```

Рисунок 1.60 – Додавання значення btn_send в strings.xml

```
StudentsListActivity.java
            txtStudents += s.getName() + "\n";
        }

        TextView textView = (TextView) findViewById(R.id.text);
        textView.setText(txtStudents);
    }

    public void onSendBtnClick(View view) {
        TextView textView = (TextView) findViewById(R.id.text);

        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, textView.getText().toString());
        intent.putExtra(Intent.EXTRA_SUBJECT, value: "Список студентів");
        startActivity(intent);
    }
}
```

Рисунок 1.61 – Метод onSendBtnClick класу активності StudentsListActivity.

Перевіряємо роботу застосунку. Після запуску переходимо до активності списку студентів, натискаємо кнопку «Відправити повідомлення». Далі для відправки повідомлення обираємо Gmail (рис. 1.62).

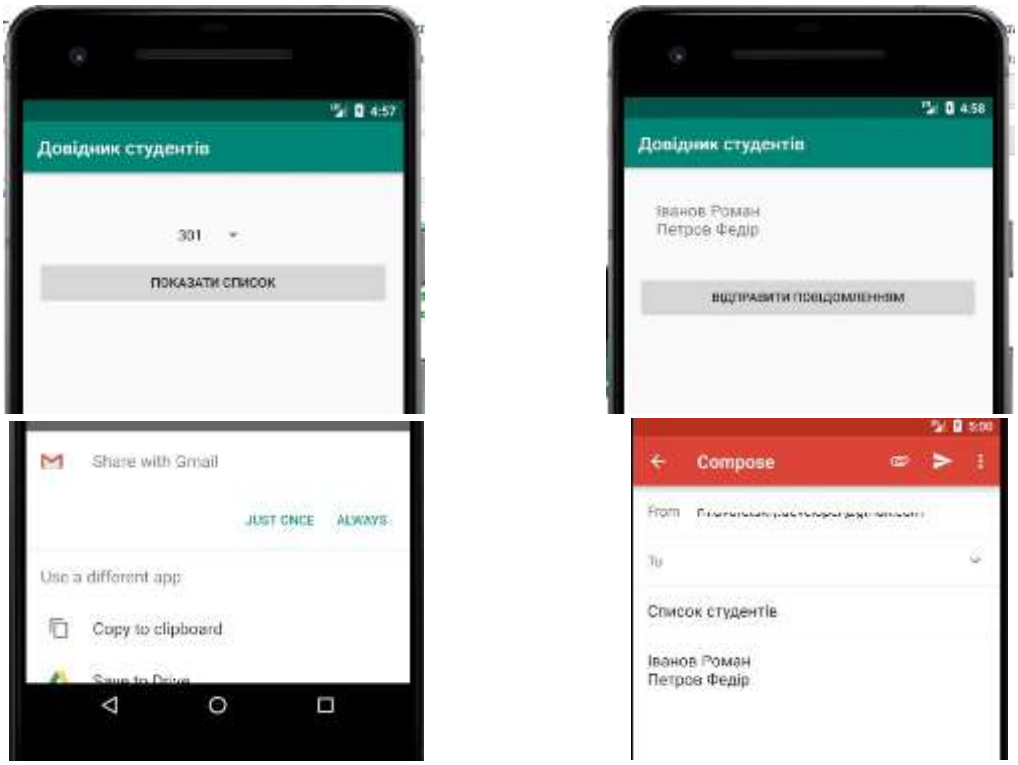


Рисунок 1.62 – Відправка листа із списком студентів через Gmail

1.4. Життєвий цикл активності

Додамо до макета другої активності проєкту activity_students_list.xml кнопку, за натисканням на яку розмір шрифту компонентів TextView, в якій відображається список студентів, збільшуватиметься на 20%. Розмістимо цю кнопку між списком студентів та кнопкою «Відправити повідомлення» (рис. 1.63).

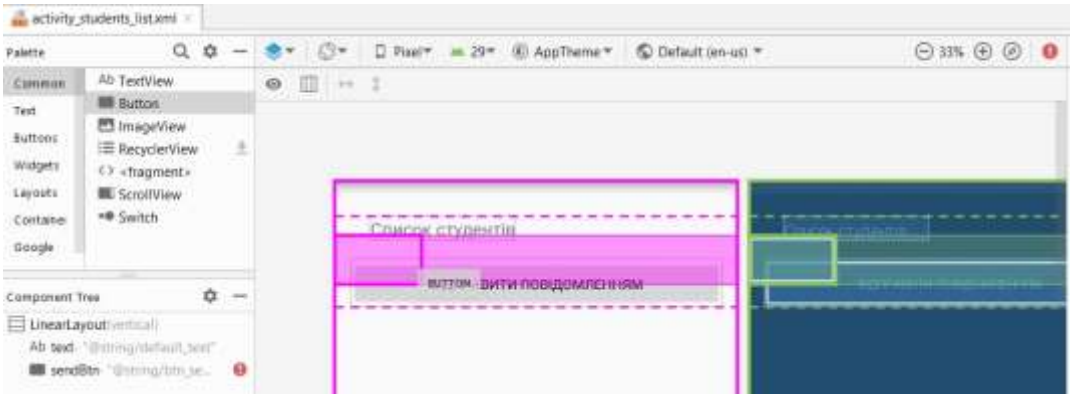


Рисунок 1.63 – Додавання кнопки до activity_students_list.xml у візуальному редакторі

Далі перейдемо на вкладку «text», змінюємо ширину кнопки, її текст, ідентифікатор та додаємо подію onClick для подальшої обробки натискання у коді активності (рис. 1.64).

```
18
19      <Button
20          android:id="@+id/btnPlus"
21          android:onClick="onPlusBtnClick"
22          android:layout_width="wrap_content"
23          android:layout_height="wrap_content"
24          android:text="+" />
25
```

Рисунок 1.64 – Зміна властивостей компонента нової кнопки

Тепер відкриємо код класу другої активності StudentsListActivity та реалізуємо збільшення шрифту відображення списку студентів. Почнімо із того, що визначимо приватне поле для збереження поточного значення розміру шрифту TextView (рис. 1.65).

```
14      private float textSize = 0;
15
```

Рисунок 1.65 – Визначення приватного поля

Далі у методі onCreate виконаємо заповнення змінної поточним значенням розміру шрифту текстового поля для відображення списку студентів (рис. 1.66).

```
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_students_list);
20
21          Intent intent = getIntent();
22          String grpNumber = intent.getStringExtra(GROUP_NUMBER);
23
24          String txtStudents = "";
25          for(Student s: Student.getStudents(grpNumber)) {
26              txtStudents += s.getName() + "\n";
27          }
28
29          TextView textView = (TextView) findViewById(R.id.text);
30          textView.setText(txtStudents);
31
32          textSize = textView.getTextSize();
33      }
```

Рисунок 1.66 – Ініціалізація textSize

І, нарешті, реалізуємо метод onPlusBtnClick, що оброблятиме натискання на кнопку (рис. 1.67).

```
34
35      public void onPlusBtnClick(View view) {
36          textSize = textSize * 1.1f;
37          TextView textView = findViewById(R.id.text);
38          textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
39      }
```

Рисунок 1.67 – Реалізація методу onPlusBtnClick()

Перевіримо роботу нашого застосунку. Запускаємо його на виконання та переходимо до форми списку студентів. Натискаємо декілька разів на кнопку «+», щоб зміна розміру шрифту стала помітною.

Але якщо тепер ми спробуємо виконати поворот пристрою, то розмір шрифту повертається до початкового значення.

Спробуємо розібратись, чому так відбувається:

– користувач запускає застосунок та переходить до потрібної активності;

– відбувається ініціалізація змінної textSize значенням поточного розміру шрифту;

– користувач натискає на кнопку «+»;

– спрацьовує метод активності, що збільшує змінну textSize та встановлює розмір шрифту = textSize;

– користувач повертає пристрій;

– Android фіксує, що орієнтація екрана змінилася, знищує активність разом з усіма її змінними та створює її заново для нової орієнтації екрана;

– відбувається ініціалізація змінної textSize значенням поточного розміру шрифту, тобто того, що було взято із макета під час створення активності;

Під час запуску активності на початку роботи програми Android бере до уваги конфігурацію пристрою. Під цим терміном розуміється як конфігурація фізичного пристрою (розмір екрана, орієнтація екрана, наявність клавіатури), так і параметри конфігурації, задані користувачем (наприклад, локальний

контекст). Система Android повинна знати конфігурацію пристрою під час запуску активності, тому що ця інформація може вплинути на ресурси, необхідні застосунку. Наприклад, у горизонтальній орієнтації може використовуватися інший макет, а у разі вибору російського чи англійського локального контексту може знадобитися інший набір строкових ресурсів.

У процесі зміни конфігурації пристрою всі компоненти програми, що відображають призначений для користувача інтерфейс, повинні бути оновлені відповідно до нової конфігурації. Якщо повернути пристрій, Android помічає, що орієнтація і розміри екрана змінилися, і інтерпретує цей факт як зміну конфігурації пристрою. Поточна активність знищується і створюється заново, щоб застосунок міг вибрати ресурси, відповідні новій конфігурації.

Активність виконується, коли вона знаходиться на першому плані на екрані. Метод onCreate() викликається під час створення активності; саме тут відбувається основна настройка активності. Метод onDestroy() викликається безпосередньо перед знищенням активності. Щоб зберегти поточний стан активності, також необхідно реалізувати метод onSaveInstanceState(). Метод onSaveInstanceState() викликається перед знищенням активності; це означає, що вам випаде можливість зберегти всі значення, які потрібно зберегти, перш ніж вони будуть безповоротно втрачені.

Метод onSaveInstanceState() отримує один параметр типу Bundle. Тип Bundle дозволяє об’єднати різні типи даних в один об’єкт. Для включення map «ім’я/значення» в Bundle використовуються методи Bundle putInt, putString та ін. Метод onCreate() отримує параметр Bundle, завдяки чому можна виконати читання попередньо збережених значень.

Виконаємо необхідні зміни у коді активності StudentsListActivity. Почнемо із реалізації методу onSaveInstanceState (рис. 1.68).

```
50
51      @Override
52      protected void onSaveInstanceState(Bundle outState) {
53          super.onSaveInstanceState(outState);
54          outState.putFloat("textSize", textSize);
55      }
```

Рисунок 1.68 – Збереження змінної textSize в onSaveInstanceState

Також виконаємо читання збережених даних у методі onCreate (рис. 1.69).

```
29          TextView textView = (TextView) findViewById(R.id.text);
30          textView.setText(txtStudents);
31
32          textSize = textView.getTextSize();
33          if (savedInstanceState != null) {
34              textSize = savedInstanceState.getFloat( key: "textSize");
35          }
36      }
```

Рисунок 1.69 – Читання збережених даних змінної textSize

Перевіримо роботу застосунку. Збільшимо розмір шрифту та повернемо пристрій.

Розглянемо іншу задачу. На головній активності реалізуємо таймер, що фіксуватиме проміжок часу, протягом якого на екрані користувача відображається ця активність. По-перше, для цього нам знадобиться приватне поле seconds, в якому буде зберігатись значення кількості секунд з моменту завантаження активності.

```
15
16          private int seconds = 0;
```

Рисунок 1.70 – Додавання приватного поля seconds в головну активність

Для вирішення задачі накопичення секунд буде використано клас Handler. Handler – клас Android, який може використовуватися для планування виконання коду в певний момент у майбутньому. Також клас може використовуватися для передачі коду, який повинен виконуватися в іншому програмному потоці. У нашому прикладі Handler буде використовуватися для планування виконання коду кожну секунду.

Щоб використовувати клас Handler, упакуйте код, який потрібно запланувати, в об’єкт Runnable, після чого використовуйте методи post() і postDelayed() класу Handler для визначення того, коли повинен виконуватися цей код.

Реалізуємо метод runTimer, що визначатиме код, що буде виконуватись кожну секунду (рис. 1.71).

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.textView);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format(Locale.getDefault(),
                    format: "%d:%02d:%02d", hours, minutes, secs);
            timeView.setText(time);
            seconds++;
            handler.postDelayed( r: this,  delayMillis: 1000);
        }
    });
}
```

Рисунок 1.71 – Реалізація класі runTimer

Та під час завантаження активності у методі onCreate запускаємо наш таймер, виконавши runTimer (рис. 1.72).

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

    runTimer();
}
```

Рисунок 1.72 – Запуск таймера в onCreate

Останній крок – збереження значення змінної seconds в методі onSaveInstanceState та відновлення у методі onCreate для запобігання втрати значення під час повороту пристрою (рис. 1.73–1.74).

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("seconds", seconds);
}
```

Рисунок 1.73 – Збереження змінної seconds

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

    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt( key: "seconds");
    }

    runTimer();
}
```

Рисунок 1.74 – Відновлення значення змінної seconds

Перевіримо роботу застосунку (рис. 1.75).

Рисунок 1.75 – Робота таймеру при повороті екрану

Як бачимо, значення таймера зберігається під час повороту пристрою. Але є інша проблема – у разі переходу на іншу активність наш таймер не зупиняється (рис. 1.76).

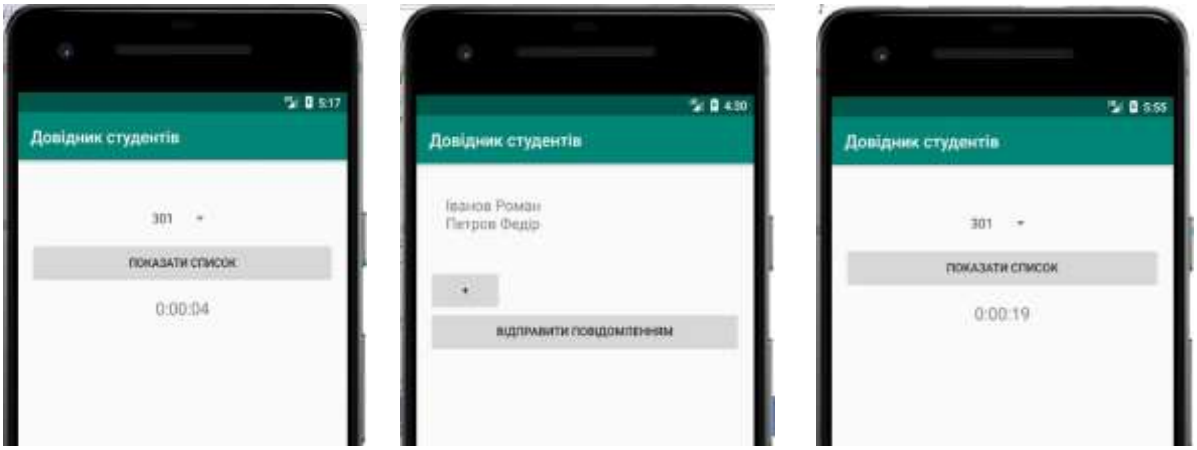


Рисунок 1.76 – Продовження роботи таймеру під час переходу на іншу активність

Отже, нам потрібно організувати зупинку роботи таймера, якщо активність не видима на екрані. Для реалізації цієї задачі для початку спробуємо розібратися із основними етапами життєвого циклу активності (рис. 1.77).

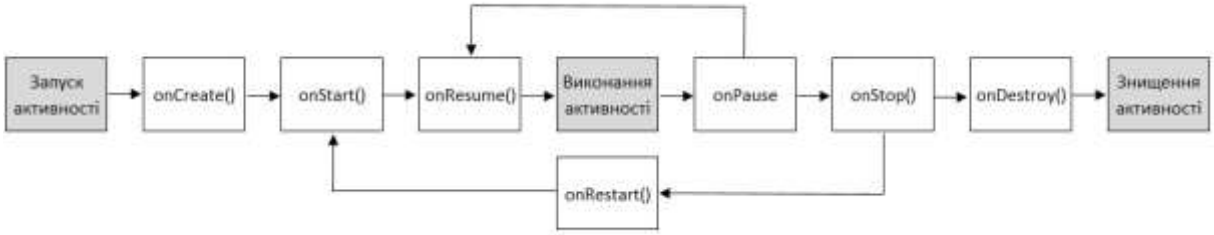


Рисунок 1.77 – Життєвий цикл активності

1. Активність запускається, виконуються її методи onCreate() і onStart(). У цій точці активність видно, але вона ще не володіє фокусом.

2. Викликається метод onResume(). Він виконується тоді, коли активність збирається перейти на перший план. Після виконання методу onResume() активність отримує фокус, а користувач може взаємодіяти з нею.

3. Метод onPause() виконується тоді, коли активність перестає перебувати на першому плані. Після виконання методу onPause() активність залишається видимою, але не володіє фокусом.

4. Якщо активність знову повертається на перший план, викликається метод onResume(). Активність, яка багаторазово втрачає і отримує фокус, може проходити цей цикл кілька разів.

5. Якщо активність перестає бути видимою користувачу, викликається метод onStop(). Після виконання методу onStop() активність не видно користувачеві.

6. Якщо активність знову стане видимою, викликається метод onRestart(), за яким слідують onStart() та onResume(). Активність може проходити через цей цикл багаторазово.

7. Нарешті, активність знищується. Під час переходу від виконання до знищення методи onPause() і onStop() викликаються до того, як активність буде знищена.

Для вирішення нашої задачі виконуємо зупинку таймера, якщо активність перестає бути видимою користувачу, та відновлення його роботи, коли активність знову стане видимою. Скористатися методами onStart та

onStop. Почнемо із додавання приватного поля isRunning до класу нашої активності (рис. 1.78).

```
14  public class MainActivity extends AppCompatActivity {
15
16      private int seconds = 0;
17      private Boolean isRunning = true;
18
```

Рисунок 1.78 – Додавання приватного поля isRunning

Реалізуємо onStart та onStop (рис. 1.79)

```
65      @Override
66      protected void onStart() {
67          super.onStart();
68          isRunning = true;
69      }
70
71      @Override
72      protected void onStop() {
73          super.onStop();
74          isRunning = false;
75      }
```

Рисунок 1.79 – Методи onStart та onStop

Та внесемо зміни у код методу runTimer для врахування значення isRunning (рис. 1.80).

```
47      private void runTimer() {
48          final TextView timeView = (TextView)findViewById(R.id.textView);
49          final Handler handler = new Handler();
50          handler.post(new Runnable() {
51              @Override
52              public void run() {
53                  int hours = seconds/3600;
54                  int minutes = (seconds%3600)/60;
55                  int secs = seconds%60;
56                  String time = String.format(Locale.getDefault(),
57                          format: "%d:%02d:%02d", hours, minutes, secs);
58                  timeView.setText(time);
59                  if (isRunning) {
60                      seconds++;
61                  }
62                  handler.postDelayed( r: this,  delayMillis: 1000);
63              }
64          });
65      }
```

Рисунок 1.80 – Зміни у runTimer

Перевіримо роботу застосунку (рис. 1.81).

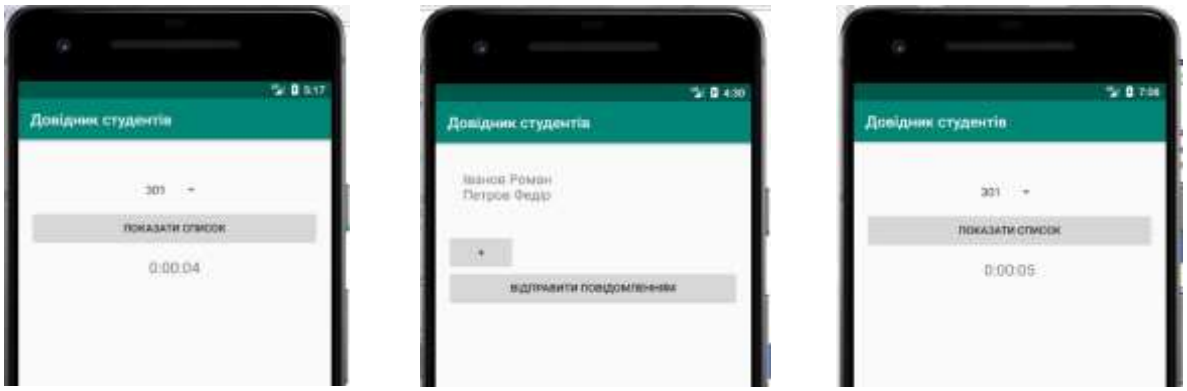


Рисунок 1.81 – Зупинка таймера при зміні активності

Як бачимо, тепер, у разі втрати активністю видимості, таймер зупиняється та продовжує свою роботу під час повернення активності на екран користувача.

РОЗДІЛ 2. ОРГАНІЗАЦІЯ ЗАСТОСУНКА

2.1. Спискові представлення

Коли розробник задумує новий застосунок, у нього зазвичай вже є маса ідей щодо того, що повинно бути в цьому застосунку. Безумовно, всі ці ідеї будуть корисні для користувача. Але як побудувати з них інтуїтивно зрозумілий, добре організований застосунок? Корисний спосіб упорядкування таких ідей полягає в їх класифікації на три типи активностей:

– активності верхнього рівня;
– активності категорій;
– активності деталізації / редагування.

Активності верхнього рівня представляють операції, найбільш важливі для користувача, і надають прості засоби для навігації по них. У більшості застосунків перша активність, яку бачить користувач, є активністю верхнього рівня.

Активності категорій виводять дані, що належать конкретній категорії, – часто у вигляді списку. Такі активності часто допомагають користувачеві перейти до активностей деталізації/редагування. Приклад активності категорії – відображення списку студентських груп.

Активності деталізації/редагування виводять докладну інформацію по конкретному запису, надають користувачеві можливість редагування існуючих записів або введення нових записів. Приклад активності деталізації/редагування – активність, яка виводить детальну інформацію про конкретну студентську групу.

Після того, як ви розділите свої ідеї на активності верхнього рівня, категорій та деталізації/редагування, ця класифікація може використовуватися для планування навігації у застосунку. Як правило, перехід від активностей верхнього рівня до активностей деталізації/редагування повинен здійснюватися через активності категорій.

Для прикладу уявимо ситуацію, що користувач хоче переглянути детальну інформацію по конкретній студентській групі. Для цього він запускає застосунок та обирає команду переходу до списку студентських груп. Далі для відображення даних по конкретній групі користувач обирає її у списку.

У застосунках з такою структурою необхідно організувати навігацію, тобто переходи між активностями. У таких ситуаціях найчастіше застосовуються спискові представленнях. Спискові представлення відображають перелік об’єктів даних, які потім використовується для навігації за застосунком.

Почнімо перетворення нашого застосунку. Для початку змінимо макет activity_main.xml головної активності, що буде відповідати активності верхнього рівня. Прибираємо звідти всі компоненти разом із лінійним layout-ом та поміщуємо ConstraintLayout у якості кореневого елемента (рис. 2.1).

Рисунок 2.1 – Зміна лейауту на ConstraintLayout в макеті головної активності

У strings.xml додаємо текст для напису на кнопці активності верхнього рівня для завантаження активності категорій списку студентських груп (рис. 2.2).

Рисунок 2.2 – Додавання даних до strings.xml

Тепер до макета головної активності додаємо ImageView та кнопку для переходу до майбутньої активності категорій – списку студентських груп (рис. 2.3).

Рисунок 2.3. Активність верхнього рівня

Перелік необхідних властивостей для ImageView та Button наведено на рис. 2.4 та рис. 2.5 відповідно.

id	imageView2
Declared Attributes	
layout_width	0dp
layout_height	wrap_content
layout_constraintEnd_toEnd	parent
layout_constraintStart_toSta	parent
layout_constraintTop_toTop	parent
layout_marginStart	16dp
layout_marginTop	16dp
layout_marginEnd	16dp
id	imageView2
scaleType	fitXY
srcCompat	@drawable/univer

Рисунок 2.4. Властивості ImageView

id	btnShowGroups
Declared Attributes	
layout_width	wrap_content
layout_height	wrap_content
layout_constraintHorizontal_	0.498
layout_constraintTop_toBott	@+id/imageView2
layout_constraintEnd_toEnd	parent
layout_constraintStart_toSta	parent
layout_marginStart	8dp
layout_marginTop	32dp
layout_marginEnd	8dp
id	btnShowGroups
onClick	onBtnShowGroupsClick
text	@string/btn_groups_txt

Рисунок 2.5 – Властивості Button

Додаємо нову активність категорій GroupsListActivity New–>Activity –>Empty activity (рис. 2.6).

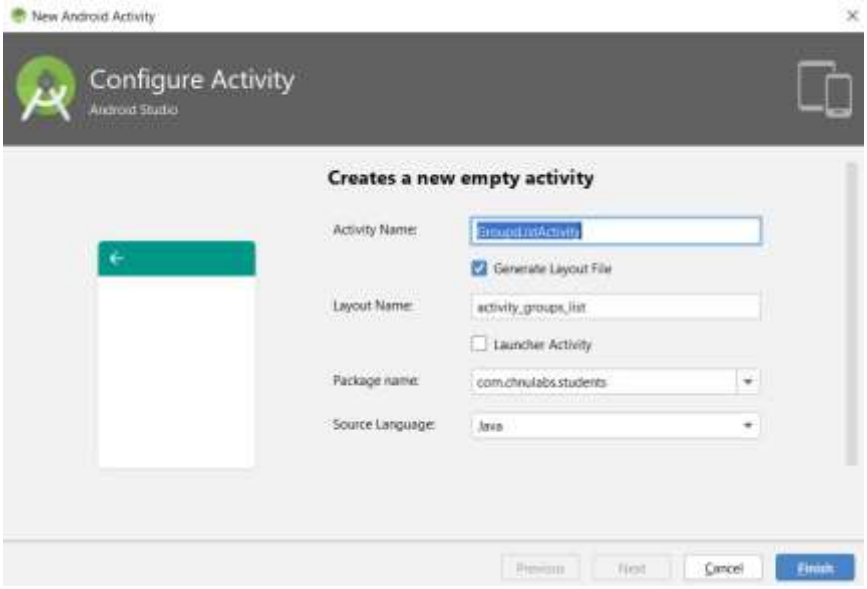


Рисунок 2.6 – Додавання активності категорій GroupsListActivity

У візуальному редакторі перетягуємо на створений макет компонент ListView (рис. 2.7).

Рисунок 2.7 – Додавання компонента ListView

Найпростіший спосіб додавання даних до списку ListView – через присвоєння властивості android:entries масиву рядків.

Рисунок 2.8 – Встановлення властивості android:entries для ListView

Повний перелік властивостей компонента ListView наведено на рис. 2.8.

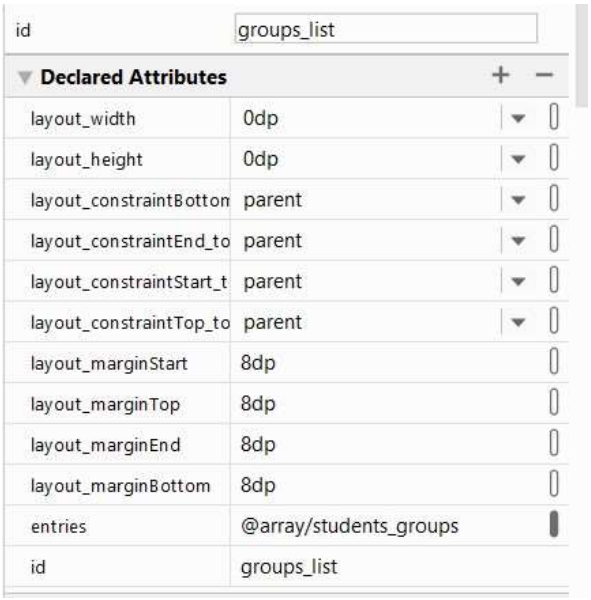


Рисунок 2.8 – Властивості компонента ListView

Пов’яжемо створену активність із головною активністю верхнього рівня. Для цього перейдемо до коду активності MainActivity, видалимо код, що залишився із попередньої роботи, та замінимо його таким (рис. 2.9).

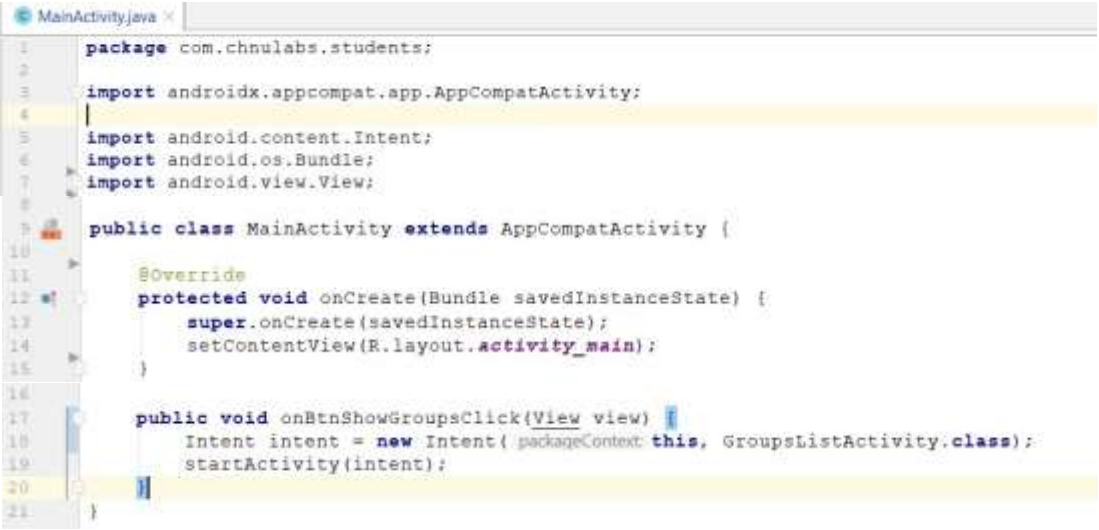


Рисунок 2.9 – Новий код активності MainActivity

Перевірка роботи застосунку (рис. 2.10).

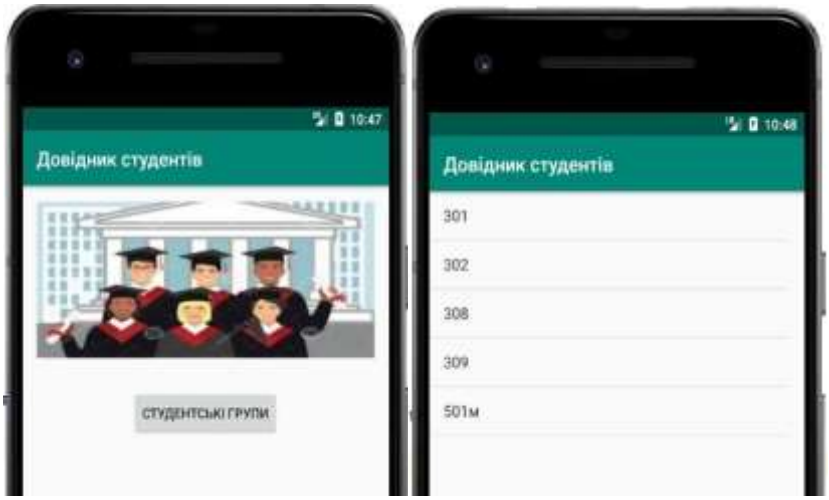


Рисунок 2.10 – Активність верхнього рівня та активність категорій

Щоб пункти списку реагували на клацання, слід реалізувати слухача подій. Слухач подій відстежує події, що відбуваються в застосунку, наприклад, клацання на представленнях, втрату або отримання ними фокуса або натискання фізичної клавіші на пристрої. Реалізація слухача подій дозволить вам виявляти конкретні дії користувача – скажімо, клацання на варіантах списку – і реагувати на них.

Навіщо створювати слухача подій, а не скористатися атрибутом android: onClick в розмітці? Атрибут android:onClick в макетах може використовуватися тільки для кнопок або будь-яких представлень, які є субкласами Button, наприклад CheckBox або RadioButton. Клас ListView не є субкласом Button, тому рішення з атрибутом android:onClick не працює. Саме тому доводиться створювати свою реалізацію слухача.

Якщо ви хочете, щоб елементи списку реагували на клацання, створіть об’єкт OnItemClickListener і реалізуйте його метод onItemClick(). Слухач OnItemClickListener відстежує клацання на елементах списку, а метод onItemClick() визначає реакцію активності на клацання. За параметрами, які передаються методу onItemClick(), можна отримати додаткову інформацію про подію – наприклад, отримати посилання на елемент зі списку, дізнатися

його позицію в списковому представленні (починаючи з 0) та ідентифікатор запису використовуваного набору даних.

Створимо OnItemClickListener для відстеження натискання на елементі списку студентських груп активності GroupsListActivity (рис. 2.11).

GroupsListActivity.java

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;

public class GroupsListActivity extends AppCompatActivity {

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

        AdapterView.OnItemClickListener listener = new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                String group = (String) adapterView.getItemAtPosition(i);
                Intent intent = new Intent( packageContext: GroupsListActivity.this,
                        StudentsGroupActivity.class);
                intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
                startActivity(intent);
            }
        };
    }
}
```

Рисунок 2.11 – Створення слухача для ListView

Наш слухач під час натискання на елемент списку отримуватиме його значення та передаватиме його до інтенту запуску активності StudentsGroupActivity за допомогою putExtra. Залишається пов’язати створеного слухача із компонентом ListView. Зробимо це (рис. 2.12).

```
                Intent intent = new Intent( packageContext: GroupsListActivity.this,
                        StudentsGroupActivity.class);
                intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
                startActivity(intent);
            }
        };

        ListView listView = (ListView) findViewById(R.id.groups_list);
        listView.setOnItemClickListener(listener);
    }
}
```

Рисунок 2.12 – Зв’язок ListView із створеним слухачем

Виконуємо перевірку роботи застосунку. Переходимо до активності списку груп та натискаємо на одному із елементів списку (рис. 2.13).

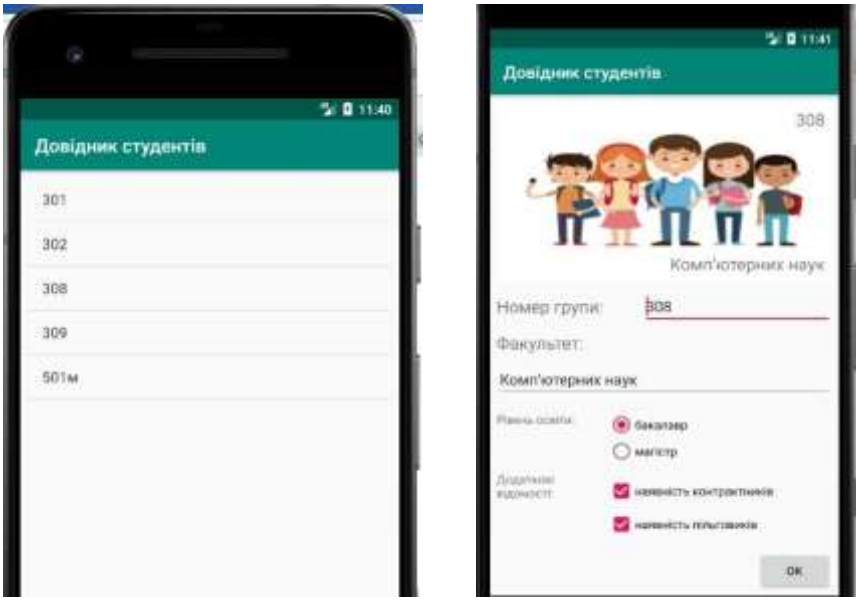


Рисунок 2.13 – Перевірка події натискання на елементі списку

Додавати елементи у список можна не обов’язково із масиву рядків. Як ми пам’ятаємо, у нас реалізований клас StudentsGroup, у якому представлена статична колекція елементів типу StudentsGroup – тобто список спудентських груп. Скористаймося ним.

Але якщо дані спискового представлення повинні надходити з джерела, відмінного від strings.xml (наприклад, з масиву Java або бази даних), необхідно використовувати адаптер. Адаптер грає роль моста між джерелом даних і списковим представленням.

У нашому випадку використаємо адаптер масивів. Щоб використовувати адаптер масиву, слід виконати його форматування і приєднати до спискового представлення. Під час ініціалізації адаптера масиву спочатку вказуємо тип даних масиву, який хочемо зв’язати зі списковим представленням. Потім адаптеру передаються три параметри: Context (зазвичай поточна активність), ресурс макета, який визначає, як повинен відображатися кожен елемент з масиву, і сам масив (або колекція).

На рис. 2.14 наведемо реалізацію адаптеру для студентських груп у методі onCreate активності GroupsListActivity.

```
ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<StudentsGroup>(
        context: this,
        android.R.layout.simple_list_item_1,
        StudentsGroup.getGroups()
);
listView.setAdapter(adapter);
```

Рисунок 2.14 – Адаптер для відображення колекції студентських груп

У класі StudentsGroup реалізуємо гетер для колекції груп groups та метод toString для строкового представлення елементів класу (рис. 2.15).

```
public static ArrayList<StudentsGroup> getGroups() {
    return groups;
}

@Override
public String toString() {
    return number;
}
```

Рисунок 2.15 – Зміни у класі StudentsGroup

У макеті activity_groups_list.xml активності видаляємо прив’язку ListView до масиву рядків файлу strings.xml (рис. 2.16).

```
<ListView
    android:id="@+id/groups_list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:entries="@array/students_groups"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.16 – Видалення прив’язки до strings.xml

Після виконаних змін можна видалити масив рядків у файлі strings.xml (рис. 2.17).

```
<string name="default_text">Список студентів</string>
<string name="btn_text">Показати список</string>
<string-array name="students_groups">
    <item>301</item>
    <item>302</item>
    <item>308</item>
    <item>309</item>
    <item>501м</item>
</string-array>
<string name="btn_send">Відправити повідомленням</string>
<string name="btn_grp_detail_txt">Детальніше ...</string>
```

Рисунок 2.17 – Видалення непотрібних даних в strings.xml

Перевіримо роботу застосунку (рис. 2.18). Для відображення якихось змін на екрані після попереднього запуску, можемо попередньо додати або видалити якусь групу із колекції у класі StudentsGroup (рис. 2.19).

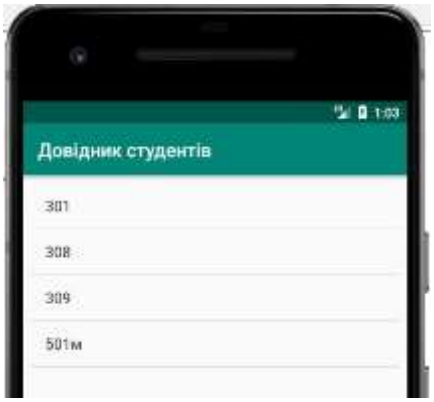


Рисунок 2.18 – Перевірка роботи застосунку після використання адаптера

```
42      private final static ArrayList<StudentsGroup> groups = new ArrayList<>(
43              Arrays.asList(
44                      new StudentsGroup( number: "301",  facultyName: "Комп'ютерних наук",
45                              educationLevel: 0,  contractExistsFlg: true,  privilageExistsFlg: false),
46      //                      new StudentsGroup( "302", "Комп'ютерних наук",
47      //                              0, true, false),
48                      new StudentsGroup( number: "308",  facultyName: "Комп'ютерних наук",
49                              educationLevel: 0,  contractExistsFlg: true,  privilageExistsFlg: true),
50                      new StudentsGroup( number: "309",  facultyName: "Комп'ютерних наук",
51                              educationLevel: 0,  contractExistsFlg: true,  privilageExistsFlg: false),
52                      new StudentsGroup( number: "501м",  facultyName: "Комп'ютерних наук",
53                              educationLevel: 1,  contractExistsFlg: false,  privilageExistsFlg: true)
54              )
55      );
```

Рисунок 2.19 – Зміна списку студентських груп

Виконаємо перехід від активності студентської групи StudentsGroup Activity до активності списку студентів StudentsListActivity та відображення списку студентів також у вигляді списку ListView. Для початку додаємо кнопку до макета activity_students_group2.xml активності Students GroupActivity, за якою будемо переходити до StudentsListActivity (рис. 2.20).

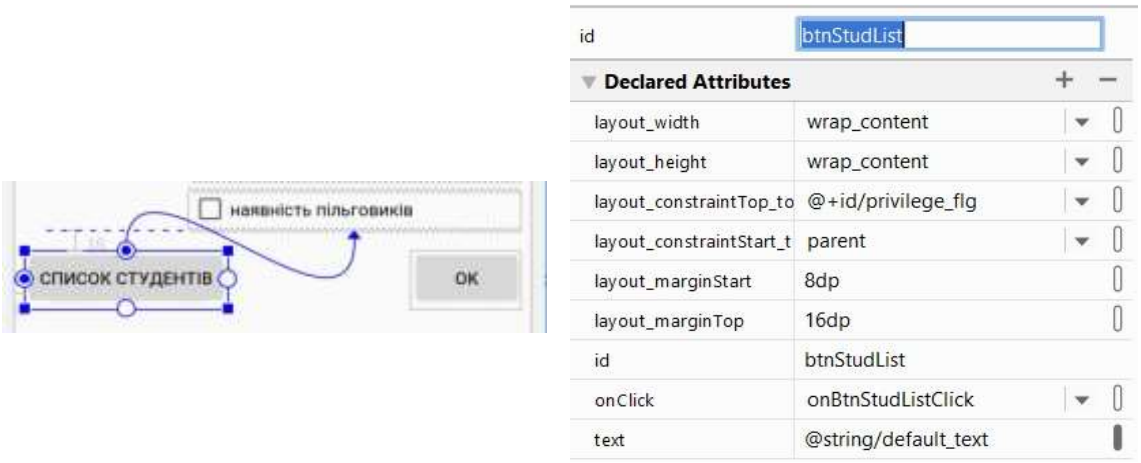


Рисунок 2.20 – Додавання кнопки до активності StudentsGroupActivity

До коду активності StudentsGroupActivity додаємо метод для натискання на кнопку onBtnStudListClick, в якому через інтент запускатимемо активність списку студентів (рис. 2.21).

```
public void onBtnStudListClick(View view) {
    Intent localIntent = getIntent();
    String group = localIntent.getStringExtra(GROUP_NUMBER);

    Intent newIntent = new Intent( packageContext: this, StudentsListActivity.class);
    newIntent.putExtra(StudentsListActivity.GROUP_NUMBER, group);
    startActivity(newIntent);
}
```

Рис. 2.21. Реалізація методу onBtnStudListClick

Перевіримо роботу застосунку. Під час спроби переходу зі списку груп до конкретної групи маємо помилку (рис. 2.22).

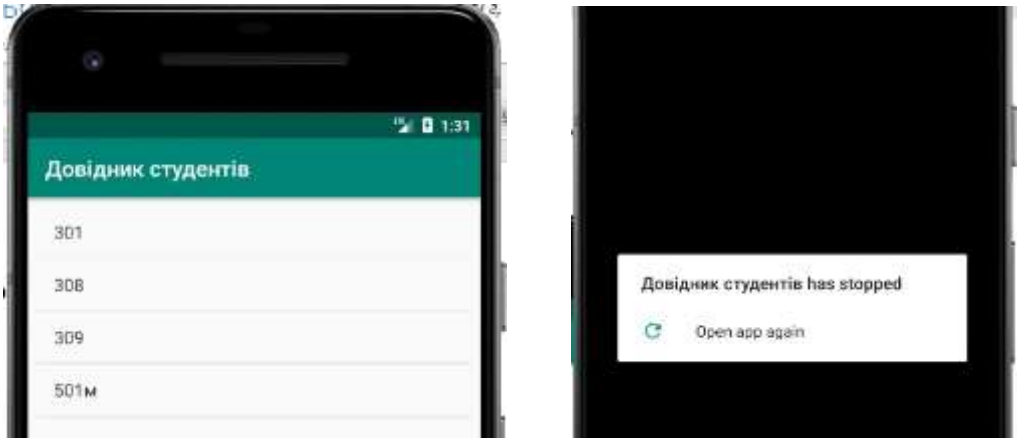


Рис. 2.22. Помилка під час переходу до активності студентської групи

Перейшовши до вкладки Run в AndroidStudio знаходимо нашу помилку та за посиланням переходимо до місця її виникнення у коді (рис. 2.23–2.24).

Рисунок 2.23 – Пошук помилки

Рисунок 2.24 – Місце виникнення помилки

Маємо помилку «com.chnulabs.students.StudentsGroup cannot be cast to java.lang.String», яка виникла внаслідок того, що раніше ми брали список груп із масиву рядків, а тепер із колекції StudentsGroup, і відповідно adapterView.getItemAtPosition(i) повертає тепер не String, а StudentsGroup. Виправимо цю помилку (рис. 2.25).

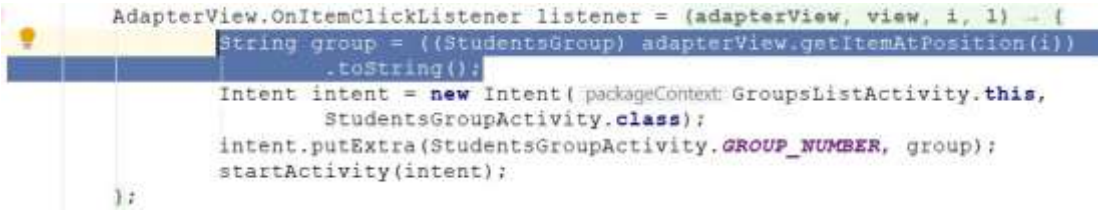


Рисунок 2.25 – Виправлення помилки перетворення типів

Тепер у активності студентської групи натиснемо на кнопку «Список студентів» (рис. 2.26).

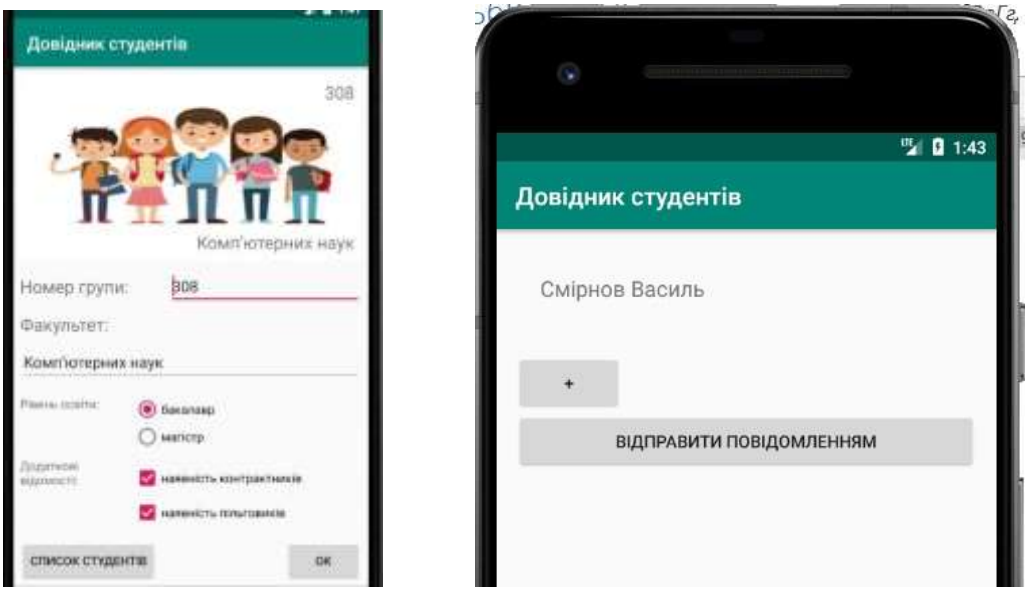


Рисунок 2.26 – Запуск активності списку студентів із студентської групи

Змінимо макет активності списку студентів activity_students_list.xml. Видалимо компоненти TextView і кнопку «+», додаємо компонент ListView (рис. 2.27).

activity_students_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".StudentsListActivity">

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

    <Button
        android:id="@+id/sendBtn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Відправити повідомленням"
        android:layout_marginBottom="16dp"
        android:onClick="onSendBtnClick"/>

</LinearLayout>
```

Рисунок 2.27 – Код макета activity_students_list.xml

У коді активності StudentsListActivity також прибираємо приватне поле textSize, методи onPlusBtnClick і onSaveInstanceState та більшість коду із методу onCreate. Також у метод onCreate прописуємо код для заповнення списку студентів даними (рис. 2.28).

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

    Intent intent = getIntent();
    String grpNumber = intent.getStringExtra(GROUP_NUMBER);

    ListView listView = (ListView) findViewById(R.id.studentsList);
    ArrayAdapter<Student> adapter = new ArrayAdapter<>(
            context: this,
            android.R.layout.simple_list_item_1,
            Student.getStudents(grpNumber)
    );
    listView.setAdapter(adapter);
}
```

Рисунок 2.28 – Змінений метод onCreate активності StudentsListActivity

Також перевизначаємо метод toString() у класі Students (рис. 2.29).

```
@Override
public String toString() {
    return name;
}
```

Рисунок 2.29 – Перевизначення методу toString() у класі Students

2.2 Кнопки та панелі інструментів. Провайдер передачі інформації

Створимо нову активність для додавання нової студентської групи.

Додаємо нову активність new–>activity–>empty activity AddStudentsGroup Activity. Використовуючи візуальний редактор, приводимо макет activity_ add_students_group.xml до такого вигляду (рис. 2.2.1).

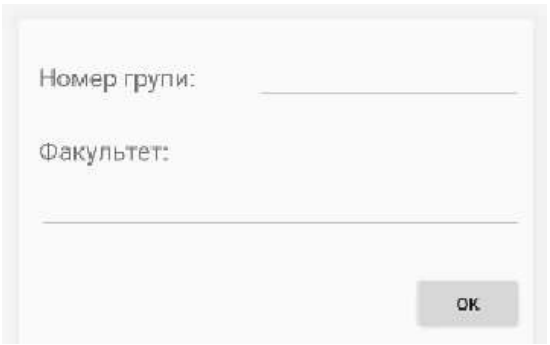


Рисунок 2.2.1 – Вигляд макета активності додавання нової групи

Для можливості додавання до масиву студентських груп у класі StudentsGroup потрібно виконати невеликі зміни. Приберемо із поля groups

```
private static final ArrayList<StudentsGroup> groups = new ArrayList<StudentsGroup>(
```

```
<string name="btn_groups_txt">Студентські групи</string>
```

ключове слово final та створюємо метод для додавання до масиву нового значення (рис. 2.2.2).

Рисунок 2.2.2 Додавання тексту напису на кнопці.

До макета activity_groups_list.xml додаємо кнопку із вищезазначеним написом, ідентифікатором btnGrpAdd та методом onGrpAddClick на подію onClick (рис. 2.2.3).

Рисунок 2.2.3 – Кнопка додавання групи

У коді активності GroupsListActivity реалізуємо метод onGrpAddClick (рис. 2.2.4).

```
41      public void onGrpAddClick(View view) {
42          startActivity(
43                  new Intent( packageContext: this, AddStudentsGroupActivity.class)
44          );
45      }
```

Рисунок 2.2.4 – Метод onGrpAddClick активності GroupsListActivity

У коді активності додавання групи AddStudentsGroupActivity також реалізуємо метод onGrpAddClick, що спрацьовуватиме по натисканню на кнопку «OK» (рис. 2.2.5).

```
18      public void onGrpAddClick(View view) {
19          EditText number = (EditText)findViewById(R.id.addGroupNumber);
20          EditText faculty = (EditText)findViewById(R.id.addFaculty);
21          StudentsGroup.addGroup(
22              new StudentsGroup(number.getText().toString(),
23                  faculty.getText().toString(),
24                   educationLevel: 0,  contractExistsFlg: false,  privilageExistsFlg: false
25              )
26          );
27          NavUtils.navigateUpFromSameTask( sourceActivity: this);
28      }
```

Рисунок 2.2.5 – Додавання елемента до колекції студентських груп

Останній рядок методу виконує повернення до батьківської активності. Пропишемо у AndroidManifest.xml для активності AddStudentsGroup Activity батьківську активність GroupsListActivity (рис. 2.2.6).

```
14          <activity android:name=".AddStudentsGroupActivity"
15              android:parentActivityName=".GroupsListActivity">
16          </activity>
```

Рисунок 2.2.6 – Вказання батьківської активності для AddStudentsGroupActivity

У коді активності GroupsListActivity перенесемо оновлення вмісту списку студентських груп із onCreate в onStart для того, щоб список оновлювався під час відновлення видимості активності, а саме – у процесі повернення до неї від активності AddStudentsGroupActivity (рис. 2.2.7)

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

    AdapterView.OnItemClickListener listener = (adapterView, view, i, l) -> {
            String group = ((StudentsGroup) adapterView.getItemAtPosition(i))
                    .toString();
            Intent intent = new Intent( packageContext: GroupsListActivity.this,
                    StudentsGroupActivity.class);
            intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
            startActivity(intent);
    };

    ListView listView = (ListView) findViewById(R.id.groups_list);
    listView.setOnItemClickListener(listener);
}

@Override
protected void onStart() {
    super.onStart();
    ListView listView = (ListView) findViewById(R.id.groups_list);
    ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<StudentsGroup>(
            context: this,
            android.R.layout.simple_list_item_1,
            StudentsGroup.getGroups()
    );
    listView.setAdapter(adapter);
}
```

Рисунок 2.2.7 – Зміни у методах onCreate та onStart активності GroupsListActivity П

Переходимо до перевірки роботи застосунку. Завантажуємо список студентських груп, додаємо нову та повертаємося до списку, у якому вже присутня нова група (рис. 2.2.8). Також звернімо увагу, що під час додавання батьківської активності у AndroidManifest.xml для AddStudents GroupActivity на панелі інструментів з’явилася кнопка Назад.

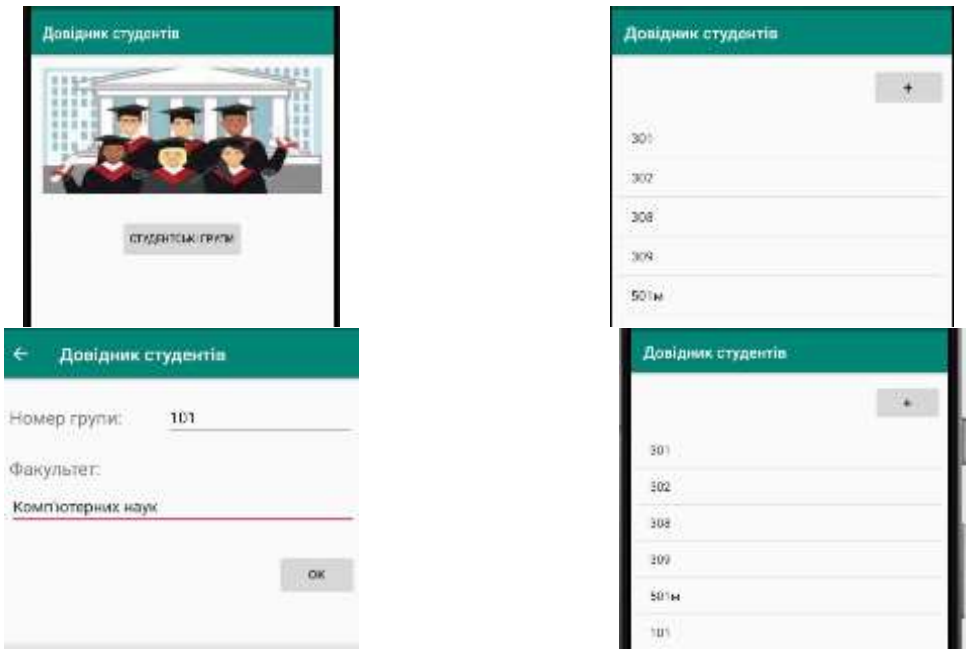


Рисунок 2.2.8 – Додавання нової групи

Нагадаємо, що у процесі створення програми в основному використовуються екрани трьох видів. Екран верхнього рівня – як правило, це перша активність застосунку, яку бачить користувач. Екрани категорій – на екранах категорій виводяться дані, що відносяться до певної категорії, – часто у вигляді списку. На них користувач може перейти до екранів деталізації/редагування. Екрани деталізації/редагування – на цих екранах виводиться докладна інформація по конкретних записах, користувач може редагувати існуючі або створювати нові записи.

Із активності категорій GroupsListActivity списку студентських груп виконується перехід до двох активностей деталізації/редагування: StudentsGroupActivity – для відображення детальних даних студентської групи та AddStudentsGroupActivity – для додавання нової групи. Якщо перша залежить від обраного елемента списку, то друга не залежить від елементів активності списку студентських груп.

Іншими словами, можна сказати, що за допомогою цієї активності користувач виконує дію. У застосунках Android дії зазвичай додаються на

панель застосунку. Ця панель у більшості випадків розташовується в верхньої частини макета активності; іноді її називають панеллю дій. На ній розміщуються найбільш часто використовувані дії.

Почнімо із створення нового меню у вигляді файлу ресурсів (рис. 2.2.9).

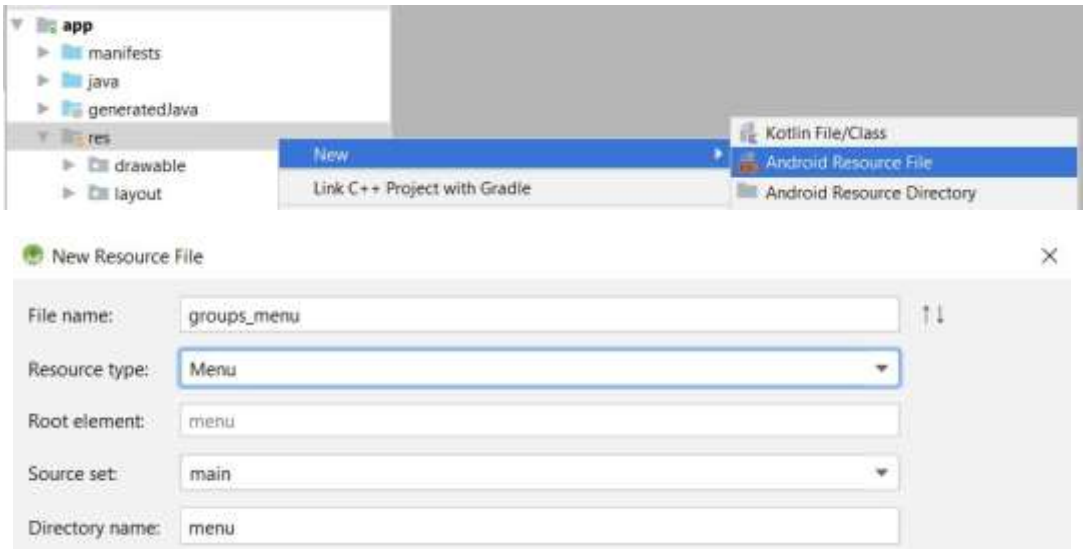


Рисунок 2.2.9 – Меню для активності списку студентських груп

На рис. 2.2.10 наведено xml-код ресурсного файлу меню у текстовому редакторі.

Рисунок 2.2.10 – Xml-код меню

Далі у коді активності GroupsListActivity виконаємо виведення створеного меню на панелі застосунку. Для цього у класі GroupsListActivity перевизначимо метод onCreateOptionsMenu (рис. 2.2.11).

Рисунок 2.2.11 – Перевизначення onCreateOptionsMenu в класі GroupsListActivity

Також змінимо текст напису на панелі активності із стандартного для нашого застосунку «Довідник студентів» на «Список груп». Для цього в AndroidManifest.xml у елементі активності GroupsListActivity додаємо свій android:label. Також, для можливості використання кнопки «назад», додаємо в AndroidManifest.xml до активності GroupsListActivity атрибут parentActivityName (рис. 2.2.12).

```
<activity android:name=".GroupsListActivity"
    android:parentActivityName=".MainActivity"
    android:label="Список груп"/>
```

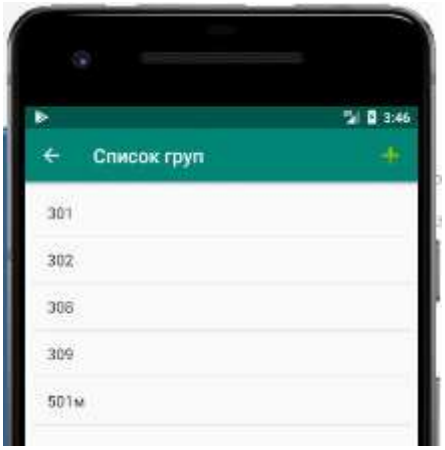
Рисунок 2.2.12 – Зміни в AndroidManifest.xml для активності GroupsListActivity

У активності все ще залишається звичайна кнопка «+» – її ми видаляємо із макета. Але тепер необхідно створити метод для обробки події натискання на кнопку «+» в меню панелі застосунку. Для цього у коді активності GroupsListActivity реалізуємо метод onOptionsItemSelected (рис. 2.2.13). Також видаляємо метод onGrpAddClick, який більше не потрібний.

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_add_group:
            startActivity(
                    new Intent( packageContext: this, AddStudentsGroupActivity.class)
            );
        default:
            return super.onOptionsItemSelected(item);
    }
```

Рисунок 2.2.13 – Реалізація методу onOptionsItemSelected

Перевіряємо роботу застосунку, переходимо до списку груп та пробуємо натиснути пункт меню для додавання нової групи (рис. 2.2.14).



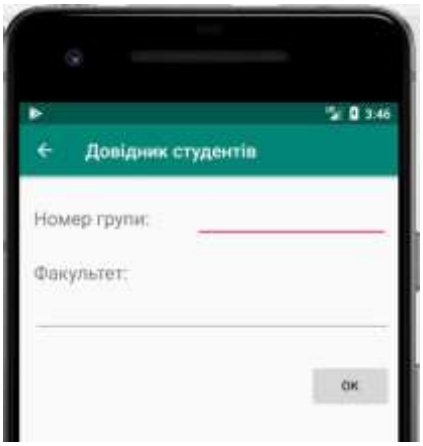


Рисунок 2.2.14 – Перевірка додавання нової групи

А тепер розглянемо, як використовувати провайдера дій на панелі застосунку. Провайдер дій – елемент, який додається на панель застосунку і сам керує своїм зовнішнім виглядом і поведінкою. Зараз зосередимося на використанні провайдера дії передачі інформації. З його допомогою користувачі можуть передавати інформацію з вашого застосунку в інші застосунки – наприклад, в Gmail. Скажімо, користувач може відправити список студентських груп одному зі своїх контактів. Провайдер дії передачі інформації має власний значок, так що вам не доведеться визначати значок самостійно. Під час натискання провайдер надає список застосунків, яким він вміє передавати інформацію. Він додає окремий значок для програми, яку ви найчастіше обираєте для передачі інформації.

Щоб обмінюватися інформацією через провайдера передачі інформації, слід передати йому інтент. Переданий інтент визначає передану інформацію і її тип. Наприклад, можна визначити інтент, який передає текст з дією ACTION_SEND; дію передачі інформації відкриє список застосунків на пристрої, здатних передавати дані такого типу.

Додаємо до strings.xml новий запис для кнопки (рис. 2.2.15).

```
25      <string name="btn_share">Поділитися</string>
```

Рисунок 2.2.15 – Новий напис у strings.xml

Тепер повертаємося до файлу res/menu/groups_menu.xml та додаємо ще один пункт меню (рис. 2.2.16).

groups_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_add_group"
        android:icon="@android:drawable/ic_input_add"
        android:orderInCategory="1"
        android:title="@string/btn_add"
        app:showAsAction="ifRoom" />
    <item android:id="@+id/action_share"
        android:title="@string/btn_share"
        android:orderInCategory="2"
        app:showAsAction="ifRoom"
        app:actionProviderClass="androidx.appcompat.widget.ShareActionProvider" />
</menu>
```

Рисунок 2.2.16 – Додавання пункту меню для відправки листа

Далі, у методі onCreateOptionsMenu активності GroupsListActivity реалізуємо отримання списку студентських груп у вигляді рядка, далі за id отримуємо shareActionProvider та передаємо йому інтент ACTION_SEND із рядком – списком студентських груп. Код методу onCreateOptionsMenu після вказаних змін наведено на рис. 2.2.17.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.groups_menu, menu);

    String text = "";
    for(StudentsGroup group: StudentsGroup.getGroups()) {
        text += group.getNumber() +"\n";
    }

    MenuItem menuItem = menu.findItem(R.id.action_share);
    ShareActionProvider shareActionProvider =
            (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);

    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);

    return super.onCreateOptionsMenu(menu);
}
```

Рисунок 2.2.17 – Код методу onCreateOptionsMenu

Перевіряємо роботу, в активності списку груп натискаємо кнопку поділитися та обираємо Gmail (рис. 2.2.18).

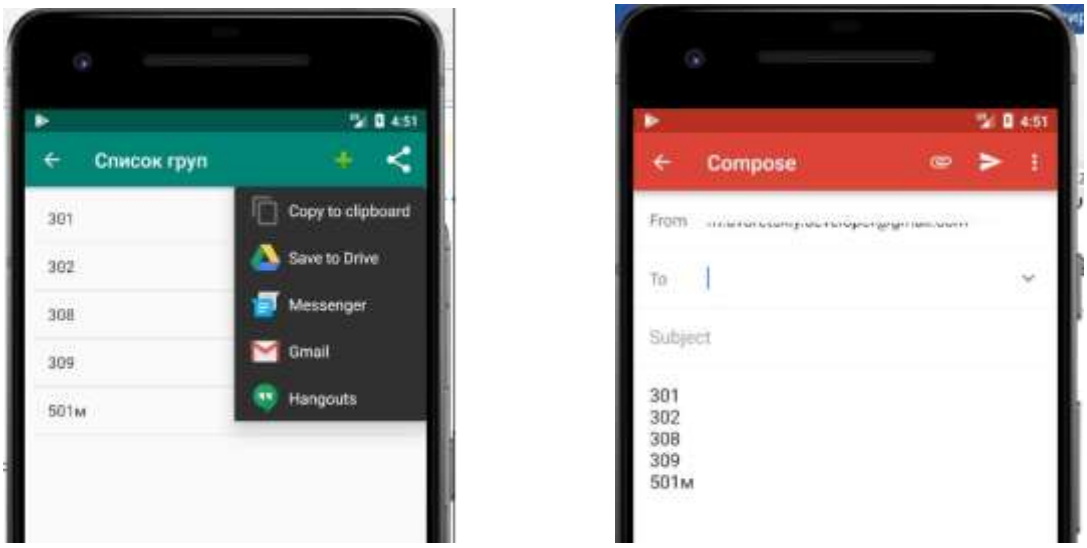


Рисунок 2.2.18 – Перевірка відправки списку студентських груп

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goransson A. Efficient Android Threading: Asynchronous Processing Techniques for Android Applications. O'Reilly Media : 1-st ed. June 13, 2014. 280 p.

2. Meier R. Professional Android, 4-th ed. Wrox, 2018. 928 p.

3. Phillips B., Stewart C., Marsicano K. Android Programming: The Big Nerd Ranch Guide : 3-rd ed. February 9, 2017. 624 p.

4. Zechne M. Beginning Android Games : 3-rd ed. Apress, 2016. 619 p.

5. Дарвин Я. Ф. Android. Сборник рецептов: задачи и решения для разработчиков приложений. Диалектика-Вильямс, 2018. 768 с.

6. Дейтел П., Дейтел Х. Андроид для разработчиков : 3-е изд. Харьков, 2016. 512 с.

7. Филлипс Б., Стюарт К., Марсикано К. Android. Программирование для профессионалов : 3-е изд. Харьков, 2017. 688 с.

8. Хорстманн К., Корнелл Г. Библиотека профессионала Java. Т. 1. Вильямс, 2014. 864 с.

9. Хорстманн К., Корнелл Г. Библиотека профессионала Java. Т. 2. Вильямс, 2014. 1008 с.

10. Шилдт Г. Java Полное руководство. 10-е изд. Диалектика, 2018. 720 с.