МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Кафедра комп’ютерних наук та економічної кібернетики

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«Програмування мобільних додатків»

для студентів денної форми навчання
спеціальності
122 «Комп'ютерні науки»

Вінниця
2021

ЗМІСТ

ВСТУП

Конспект лекцій розроблено відповідно до робочої програми дисципліни «Програмування для мобільних пристроїв» для студентів денної форми навчання ступеня Бакалавр за спеціальністю 126 «Інформаційні системи та технології». Матеріал лекційного курсу присвячено вивченню теоретичних основ розробки додатків, призначених для функціонування на мобільних пристроях під управлінням ОС Android.

В результаті засвоєння лекційного матеріалу студенти буде знати**:**

- загальну характеристику найпопулярніших мобільних платформ;
- архітектуру та особливості платформи Android;
- засоби та можливості середовища Android Studio, зокрема, щодо відлагодження розроблюваних програм з використанням емулятора;
- основи проектування та розробки рішень під платформу Android на мові програмування Java включаючи, зокрема, використання апаратних особливостей мобільних пристроїв.

В результаті засвоєння лекційного матеріалу студенти буде вміти**:**

- використовувати основні класи і типи даних мови Java та можливості SDK, призначені для розробки додатків під мобільну платформу;
- застосовувати середовище Android Studio та відповідний емулятор;
- проектувати інтерфейс програми, в тому числі із застосуванням різноманітних елементів управління;
- використовувати апаратні можливості та особливості мобільних пристроїв (розмір екрану, файлова система, камера, ресурси тощо);
- застосовувати для збереження даних базу даних SQLite;
- розгортати розроблені додатки в Google Play.

Для успішного засвоєння матеріалу студенти повинні володіти базовими знаннями з теорії алгоритмів, програмування та методології об’єктно-орієнтованого програмування. Необхідним є володіння на базовому рівні хоча б однією із сучасних об’єктно-орієнтованих мов програмування (C++, C#, Java).

Освоєння дисципліни є однією із передумов для підготовки та захисту студентами кваліфікаційної роботи.

ЛЕКЦІЯ 1. ЗАГАЛЬНИЙ ОГЛЯД ПЛАТФОРМ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Загальна характеристика платформ для мобільних пристроїв

У сучасному світі існує величезна різноманітність мобільних пристроїв. «Мобільний» означає транспортабельний, переносний. У нашій свідомості, традиційно – це мобільні телефони, планшети, електронні книги, нетбуки, навігатори, розумні годинники, плеєри. А віднедавна сюди ще відносять телевізори, кінотеатри, окуляри віртуальної реальності чи щось подібне.

Як і у будь-якого комп'ютерного пристрою, у мобільних пристроїв є операційні системи. Мобільна операційна система (мобільна ОС) – це операційна система для смартфонів, планшетів, КПК або інших мобільних пристроїв. Мобільні операційні системи поєднують в собі функціональність ОС для ПК з функціями для мобільних і кишенькових пристроїв: сенсорний екран, стільниковий зв'язок, Bluetooth, Wi-Fi, GPS-навігація, камера, відеокамера, розпізнавання мови, диктофон, музичний плеєр, NFC (зв'язок ближнього поля) та інфрачервоне дистанційне управління.

Хоча ноутбуки і можна віднести до мобільних пристроїв, однак операційні системи, під управлінням яких вони працюють, як правило, мобільними не вважаються. Це пояснюється тим, ці ОС розроблялися для стаціонарних настільних комп'ютерів, які традиційно не мали спеціальних «мобільних» функцій, та й не потребували їх. Ця різниця розмита в деяких нових операційних системах, що являють собою гібрид того й іншого.

Портативні пристрої мобільного зв'язку (наприклад, смартфони) містять дві операційні системи. Основну програмну платформу взаємодії з користувачем доповнює друга, низькорівнева пропрієтарна операційна система реального часу, яка обслуговує радіоустаткування. Дослідження показали, що такі низькорівневі операційні системи є вразливими перед шкідливими базовими станціями, здатними отримати контроль над мобільним пристроєм.

1.2 Огляд найпопулярніших і застарілих мобільних ОС

Сучасні операційні системи для мобільних пристроїв: Android, iOS, CyanogenMod, Cyanogen OS, Fire OS, Flyme OS, Firefox OS, Sailfish OS, Tizen, Ubuntu Touch.

Застарілі програмні платформи, які вже нині не підтримуються: Windows Phone, BlackBerry OS, Symbian, Palm OS, webOS, Maemo, MeeGo, LiMo.

Розглянемо деякі з них детальніше.

1.2.1 Android

Операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Підтримується альянсом Open Handset Alliance (OHA).

Хоча Android базується на ядрі Linux, він стоїть дещо осторонь Linux-спільноти та Linux-інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik – віртуальної машини Java, і все програмне забезпечення і застосування спираються на цю реалізацію Java.

З найбільш відомих на вітчизняному ринку – це планшети і смартфони Samsung, HTC, Huawei, SONY, LG, Lenovo.

Перша версія ОС вийшла у світ в 2008 році на смартфоні HTC, і з тих пір невпинно оновлюється.

Основою популярності Android є те, що Android можна встановлювати на пристрої різних виробників. Найбільш поширені такі пристрої на території країн, що розвиваються Азії, Індії, Африки, Європи.

Переваги ОС Android:

- різноманітність програм та ігор на Android;
- швидка інтеграція з сервісами Google;
- абсолютна незалежність від апаратної начинки мобільного пристрою;
- Android є екосистемою з відкритим кодом;
- багатозадачність – це означає, що без проблем працює одночасно кілька програм;
- легкість встановлення програм із різних ресурсів;
- широкі можливості індивідуалізації;

- відсутність обмежень при виборі мобільного оператора;
- підтримка флеш-програвача;
- зручне оновлення через Інтернет, причому оперативне – ведеться безперервна робота над поліпшенням функціоналу ОС, виправляються баги, вносяться зміни в інтерфейс;
- можливість заміни / видалення дефолтних програм;
- максимально спрощено виробництво програм, ігор, плагінів, оновлень. Стрімко набирає обертів нова професія – розробник додатків під Андроїд. Побутує думка, що краща мова програмування для Android – Java, далі C#, Python.

Недоліки ОС Android:

- пристрій під управлінням ОС Android доводиться досить часто заряджати/підзаряджати;
- існують деякі проблеми сумісності нових версій операційної системи зі знятими з виробництва, але присутніми в обігу застарілими пристроями, або ж із пристроями, які випущені маловідомими китайськими виробниками;
- користувачі, у яких на першому місці комфорт від швидкості роботи, можуть виявитися не задоволеними як нею в цілому, так і кількістю налаштувань.

1.2.2 iOS

Для смартфонів, планшетів та ін. пристроїв, розроблена легендарною компанією Apple, виключно під себе, на відміну від Windows Phone (Microsoft) і Android (Google). Вперше операційна система була представлена в 2007 році разом з телефоном iPhone. Спершу створювалася для iPhone і IPod Touch, пізніше для таких пристроїв, як iPad і Apple TV.

Ядро iOS майже ідентичне ядру операційної системи Apple MacOS (раніше іменувалася OS X), тому використовує такий самий набір основних компонентів.

Переваги iOS:

- зручність в роботі;
- шикарний дизайн;
- акцент на надійність і якість ОС;
- оптимізація сприяє збільшенню тривалості роботи гаджета при повному його завантаженні;
- відсутність будь-яких сповільнень, «глюків», як у програмах, так і на робочому екрані пристроїв;

- величезний вибір додатків;
- у програмах, на відміну від Google Play, рідко зустрічається реклама;
- доступність оновлень відбувається відразу після випуску нової версії ОС для всіх пристроїв одночасно;
- розробники щоразу анонсують свої програми для IOS;
- можливість об'єднати роботу оновлених до останньої ОС мобільних пристроїв;
- великий плюс – тривала підтримка старих пристроїв;
- сімейний доступ для покупок в App Store;
- багатозадачність. Передбачена спеціальна панель-меню багатозадачності.

Недоліки iOS:

- закрита, досить консервативна система – все програмне забезпечення, за кількома винятками, є платним. Причому, такий варіант характерний для всіх пристроїв Apple – від смартфонів iPhone до планшетів iPad;
- інколи користувачеві доведеться переплачувати за додатки;
- пряме копіювання файлів чи передача з'явилася лише нещодавно (переміщення можливе було тільки за допомогою iTunes);
- в iOS все зав'язано на інтернеті (коли немає підключення, то ви втрачаєте багато функцій);
- працює тільки на пристроях компанії Apple, що значно зменшує поширення iOS по всьому світі. iPad, iPhone, iPod, Macbook, iMac і iWatch найбільш поширені у заможних країнах – на території США, Європи.

Як операційна система iOS була представлена з iPhone на Macworld Conference & Expo 9 січня 2007 року і випущена в червні того ж року. Спершу, Apple не вказувала її ім'я, просто заявивши, що «iPhone використовує OS X». Спочатку, сторонні програми не підтримувалися. Стів Джобс заявив, що розробники можуть створювати веб-програми, що «будуть вести себе, як рідні програми на iPhone». 17 жовтня 2007 року Apple оголосила, що рідний SDK знаходиться в стадії розробки, і що вони планують поставити його «в руки розробників у лютому». 6 березня 2008 року Apple випустила першу бета-версію, а також нове ім'я для операційної системи: iPhone OS. Продажі мобільних пристроїв Apple викликали інтерес до SDK. Apple також продала більше одного мільйона iPhones під час курортного сезону 2007. У червні 2010 року, Apple перейменувала iPhone OS на iOS.

1.2.3 Windows Phone

Мобільна операційна система, випущена у 2010 році корпорацією Майкрософт. Вона легко пізнається за своєрідними «живими» кольоровими плиточками на стартовому екрані.

Переваги Windows Phone:

- Windows – найпоширеніша ОС для ПК у світі, і мобільні пристрої, що працюють на ОС Windows, легко синхронізуються з комп'ютером, ноутбуком, планшетом;
- достатня кількість програм, встановлених за замовчуванням;
- плавність інтерфейсу, висока швидкість запуску програм, перемикання між відкритими вікнами та ін.;
- доволі зручний дизайн інтерфейсу;
- прості і зрозумілі налаштування;
- підтримка великої кількості пристроїв;
- немає проблем з оперативною пам'яттю.

Недоліки Windows ОС:

- мало додатків або вони не повнофункціональні;
- деяких користувачів старих смартфонів позбавлено підтримки.

Windows Phone встановлена майже на всіх мобільних пристроях Nokia нового покоління, які прийшли на заміну ОС Symbian. Поширені мобільні Nokia в Бразилії, Індії та Європі.

Інтерфейс операційної системи включає 6 витягнутих по горизонталі панелей (Hubs), які на екрані мобільного пристрою можна прокручувати ліворуч і праворуч. Панель «Люди» (People) об'єднує всю інформацію, що стосується будь-якого певного людини, в тому числі його записи та коментарі у соціальних мережах, а також фотографії, надаючи централізований доступ до таких мереж як Facebook і Windows Live. Панель «Картинки» (Pictures) об'єднує фотографії та відеозаписи користувача, що зберігаються в пам'яті пристрою, на комп'ютері та в інтернеті, також відкриваючи доступ до фотографій і відеозаписів друзів. Панель «Ігри» (Games) відкриває доступ до аватарів, використовуваним в Xbox Live, досягнень, профілів інших гравців і мобільних ігор. «Музика+Відео» (Music+Video): об'єднує мультимедійних контент, що зберігається на комп'ютері користувача, музичні онлайн-сервіси та вбудоване FM-радіо і відкриває доступ до сервісу Zune Social для обміну музикою. Панель Marketplace дозволяє завантажувати застосунки та ігри, а Office забезпечує доступ до Office Mobile,

SharePoint і OneNote. Користувачеві надається можливість відкриття, створення і редагування документів.

1.2.4 BlackBerry

Blackberry ОС використовується лише на пристроях даної компанії, які розробляють в Канаді і відповідно найбільше їх саме там. Також популярність Blackberry відзначається в США, Австралії, Європі, зокрема, Великобританії. Через спеціальну утиліту на Blackberry встановлюються додатки Android.

Переваги Blackberry:

- висока якість, оскільки Blackberry позиціонуються, як пристрої бізнес класу;
- відзначається схожість одночасно з Android та iOS;
- гнучкість меню налаштувань;
- безпроблемна синхронізація зі штатними засобами та хмарними рішеннями.

Недоліки ОС Blackberry:

- невелика кількість застосунків та їх оновлення, яке відбувається рідко.

Смартфони Blackberry завжди орієнтувались на потреби корпоративних клієнтів, пропонуючи найкращий захист персональних даних, бізнес сервіси BIS та BES, а також зручний доступ до електронної пошти (E-mail) та Інтернету. Основна мета цих смартфонів – зробити ведення бізнесу легким, зручним і безпечним. Тому перевагу RIM віддали функціональній фізичній QWERTY клавіатурі, яка присутня майже на всіх Blackberry. Сучасні моделі підтримують роботу з документами у форматах Word, PDF, Excel, PowerPoint, ASCII text, HTML, WordPerfect і ZIP. Всі смартфони Blackberry використовують кодування характеристик за стандартом AES для захисту даних від перехоплення. Завдяки цьому телефони Blackberry часто використовують в роботі держорганів. Технологія PUSH-повідомлень спеціально була розроблена для пристроїв BlackBerry, завдяки якій адресат моментально отримує всі e-mail повідомлення, навіть без постійного з’єднання з сервером. Відразу ж після відправлення листа на поштову скриньку користувача оператор мобільного зв’язку сповіщає смартфон про надходження повідомлення, і тільки тоді він з’єднується з сервером, для завантаження пошти.

1.2.5 Firefox OS

Мобільна операційна система, випущена організацією Mozilla у липні 2012. Спочатку проект розвивався під ім’ям Boot to Gecko (B2G), але згодом вирішили, що

ОС буде поставлятися під впізнаваним брендом Firefox, це допоможе зацікавити користувачів до нових смартфонів, що тільки виходять на ринок.

26 липня 2011 представник Mozilla Foundation повідомив про початок робіт над операційною системою, заснованою на рушії Gecko, що використовується в браузері Mozilla Firefox.

У лютому 2012 іспанська телекомунікаційна компанія Telefónica спільно з Mozilla Foundation розробили концепт Open Web Device, що використовує операційну систему Boot to Gecko. Повідомляється також про співпрацю з Qualcomm і Deutsche Telekom.

У грудні 2015 року з'явилася інформація про закриття проекту як платформи для смартфонів. При цьому не виключалося поява інших пристроїв на цій платформі. Проте пізніше стало відомо, що в Mozilla мали на увазі тільки відмову від співпраці з операторами зв'язку, а сама операційна система для смартфонів продовжить існування у вигляді відкритої платформи і може бути використана виробниками пристроїв на їх розсуд. Розроблена в рамках проекту Firefox OS мобільна платформа, базується на ідеї використання браузерного оточення замість робочого стола. На відміну від ChromeOS платформа Firefox OS орієнтована насамперед на мобільні пристрої та надає розширений Web API для створення спеціалізованих мобільних веб-застосунків, які використовують можливості сучасних телефонів. Як основа використовується ядро Linux і компоненти з низькорівневої платформи Android. Замість віртуальної машини Dalvik для запуску застосунків задіяний web-стек Mozilla. Інтерфейс користувача платформи сформований з набору веб-застосунків Gaia. До складу включені такі програми, як веб-браузер, калькулятор, календар-планувальник, додаток для роботи з веб-камерою, адресна книга, інтерфейс для здійснення телефонних дзвінків, клієнт електронної пошти, система пошуку, музичний плеєр, програма для перегляду відео, інтерфейс для SMS/MMS, конфігуратор, менеджер фотографій, робочий стіл і менеджер програм з підтримкою декількох режимів відображення елементів (cards і grid).

Влітку 2014 року Mozilla пішла на дивний крок – дозволила смартфонам на Android запускати додатки для Firefox OS. Для цього потрібно було зайти в браузер Firefox і відкрити фірмовий магазин додатків. По суті, звідти встановлювати не повноцінні програми, а веб-сторінки в браузері без стандартного інтерфейсу з адресним рядком. Корисного програмного забезпечення в цьому маркеті було мало і до того ж

воно працювало гірше рідних (native) додатків. Тому від такого рішення нікому краще не стало – ні Android, ні Firefox OS.

Якщо подивитися всю хронологію розвитку Firefox OS, то можна легко помітити, як розробники самі до кінця не розуміють мету проекту – спочатку це начебто універсальна система для смартфонів різних категорій, потім тільки для бюджетних, а потім ми розробляємо оболонку для телевізорів, «розумних» годинників і кнопкових телефонів. Хоча розробкою проекту займалися всього 4 роки, що дуже мало, і явно на нього не витрачалося багато ресурсів. Така ж доля була у Ubuntu Touch, яка розвивалася приблизно паралельно Firefox OS.

1.2.6 Sailfish

Операційна система з відкритим кодом (але закритим кодом інтерфейсу користувача) та використанням компонентів з відкритим кодом на базі ядра Linux, здебільшого направлена на смартфони. Розробляється з 2012-го року компанією Jolla, заснованою колишніми співробітниками Nokia з метою розробки нових смартфонів, побудованих на базі Linux-платформи MeeGo, у співпраці з проектом Mer, і підтримкою Sailfish Alliance. Особливістю інтерфейсу Sailfish є управління з активним використанням екранних жестів і задіяння вертикальної моделі розміщення контенту, що передбачає використання екранних жестів гортання для переходу від одного екрана до іншого (наприклад, можна «перегорнути» домашній екран і потрапити на екран зі списком застосунків або на екран з оглядом подій). Доступ до меню відкривається екранним жестом при неповному зрушенні вмісту вниз. Домашній екран виступає в ролі інтерфейсу для швидкого запуску і переходу між запущеними застосунками (відображається огляд запущених у цей час застосунків зі зведеною інформацією з активності в кожній з програм). Смартфон Jolla вийшов в листопаді 2013 року. Він отримав бюджетний процесор Snapdragon 400, 1 ГБ оперативної і 16 ГБ постійної. Виділявся він за рахунок Sailfish OS. В системі вже був набір найпопулярніших додатків - браузери Opera і Firefox, WhatsApp, Twitter, Skype та інше.

Jolla вирішила показати, що Sailfish OS – універсальна система, тому в листопаді 2014 року компанія вийшла на краудфандинговую площадку Indiegogo з першим планшетом Jolla Tablet. Планувалося зібрати всього 380 тисяч доларів, а вийшло залучити 2,5 мільйона. Проект виявився успішним.

2015 рік виявився складним для Jolla. Виникли проблеми з поставками деталей для планшета і ніхто з великих компаній не погоджувався міняти Android на

Sailfish OS, а в цей час гроші інвесторів закінчилися і почалися збитки. В якості вирішення компанія змінила стратегію – вирішено відмовитися від апаратних продуктів, щоб зосередитися на розвитку і просуванні операційної системи.

Другу половину 2015 року Jolla спілкувалася з невідомим інвестором, який повинен був внести ще 10 мільйонів євро, але цього не сталося. Тому в листопаді компанія звільнила половину співробітників зі штату приблизно в 100 чоловік і звернулася до суду Фінляндії, щоб реструктуризувати борги.

Ситуація з планшетом теж ускладнилася. До проблем з поставками комплектуючих додалися борги, тому за 2015 рік Jolla випустила трохи більше сотні планшетів. У 2016 році зібрали ще 540 планшетів, а потім компанія оголосила про закриття проекту Jolla Tablet – попередньо замовившим пообіцяли повернути гроші, як з'явиться можливість.

Незважаючи на нову стратегію Jolla – в травні 2016 року компанія все-таки представила новий смартфон. Але він швидше призначався для фанатів і розробників, ніж для масового споживача. Насправді Jolla C став копією індійського смартфона Intex Aqua Fish під новим брендом.

В цілому, Sailfish OS неможливо сприймати як серйозну альтернативу Android або iOS. У ній є все базові функції і навіть нормальна підтримка Android-додатків, але немає головного – аудиторії. Без користувачів розвиток операційної системи неможливий. Всі ці роки розробники випускають оновлення, орієнтуючись на відгуки сотень, можливо тисяч користувачів, які заради інтересу ставлять собі альтернативну систему. При цьому встановити Sailfish OS можна на обмежене коло Android-смартфонів, а купити смартфон з встановленою системою від Jolla дуже складно.

1.2.7 Tizen

Після того, як Nokia відмовилася від MeeGo на користь Windows Phone, Intel не стала розвивати проект наодинці. Приблизно в цей же час Samsung зробила ставку на Android і закинула систему Bada, на яку витратила багато грошей і маркетингових сил.

В один час із закриттям MeeGo і Bada організація Linux Foundation, яка працювала над проектом LiMo, оголосила новий проект на базі відкритого вихідного коду і ядра Linux – Tizen (27 вересня 2011 року). Новий проект об'єднав в собі напрацювання LiMo і MeeGo, остання в свою чергу утворена від платформ Maemo і Moblin.

Tizen задумувалася як кросплатформна операційна система для смартфонів, планшетів, комп'ютерів, телевізорів та інших пристроїв, на кшталт бортових систем автомобілів. За основу платформи взяли HTML5 як просту і перспективну мову для розробки додатків, які можна легко адаптувати під різні пристрої. Журналісти зустріли Tizen прохолодно – це вже не перша мобільна Linux-система з відкритим вихідним кодом. Досить подивитися передісторію цього проекту у вигляді MeeGo, LiMo і старіших попередників. Тим більше, що за Tizen не було якоїсь цікавої ідеї або дорогий маркетингової кампанії. Хіба що Samsung проводила конкурси серед розробників з великими грошовими призами.

В рамках виставки MWC 2013 компанія анонсувала фінальну Tizen 2.0 під кодовою назвою Magnolia. Хоча правильно було б все попередні версії називати бета або альфа, а цей реліз позначити як 1.0. Тому що тільки в цьому оновленні додали базові речі, на кшталт підтримки нативних додатків і повноцінний набір попередньо стандартних програм. Щоправда, на той момент Tizen 2.0 все одно працювала жахливо. Тому привернути увагу їй було нічим.

Незважаючи на обіцянки представити смартфони на Tizen в 2013 році, перший з'явився тільки влітку 2014 року – Samsung Z. Журналісти перший смартфон з Tizen зустріли холодно – смартфон нічим не виділявся, а система була схожа на Android з оболонкою TouchWiz, але без більшості потрібних сторонніх додатків .

І приблизно в цей же час «розумні» годинники Samsung Galaxy Gear, що працюють на модифікованому Android (НЕ Android Wear), з оновленням перевели на Tizen. Схоже, власну операційну систему було простіше адаптувати під екран годинника, ніж перекручуватися зі смартфонів Android, який для цього не призначений.

Пізніше в серпні на ринок вийшов новий годинник Gear S із зігнутим дисплеєм, що працював на Tizen.

Важливо розуміти, що Tizen змінювалася залежно від пристрою – система в годинниках, смартфоні, камері або ноутбуці виглядала і працювала по-іншому. Для користувача це різні операційні системи, які між собою ніяк не пов'язані. Tizen була непопулярною системою, тому ніхто не витрачав час на перевірку її безпеки, хоча ці смартфони продавали для корпоративних цілей. У квітні 2017 року дослідник в області інформаційної безпеки Аміхай Нейдерман опублікував великий звіт після вивчення Tizen - він виявив 40 вразливостей нульового дня і назвав це «можливо, найгіршим кодом, який коли-небудь бачив у житті».

У дослідженні говориться, що кожна з цих вразливостей дає можливість хакерам віддалено отримати доступ до смартфону або іншого пристрою на Tizen. Одна з таких вразливостей дозволяє завантажити будь-шкідливий додаток на телевізор Samsung. Після цієї публікації Samsung не стала заперечувати проблему і зв'язалася з Нейдерманом, щоб усунути всі вразливості.

Tizen перетворилася в запасний варіант для Samsung, якщо почнуться якісь проблеми з Android. У Huawei, наприклад, не виявилося і такої альтернативи – за чутками, її Harmony OS або Hongmeng ще дуже сира, щоб працювати на смартфонах. Тому китайської компанії доводиться випускати флагман Mate 30 на Android без сервісів Google.

Samsung поступово випускає оновлення Tizen, але видно, що на неї не хочуть витрачати багато часу і грошей. Набагато активніше система розвивається в «розумних» годинниках і телевізорах, так що все не так погано.

З усього циклу історій альтернативних операційних систем Tizen виглядає як найвдаліший проект.

ЛЕКЦІЯ 2. ПЛАТФОРМА ANDROID

2.1 Коротка історія платформи

Android – відкрита операційна система для мобільних телефонів, смартфонів, комунікаторів, планшетних комп'ютерів, електронних книг, цифрових програвачів, наручних годинників, нетбуків і смартбуків. Базується на ядрі Linux і підтримує різні апаратні платформи, такі як ARM, MIPS, POWER, x86.

Однією із головних переваг платформи Android є її відкритість. Операційна система Android побудована на основі відкритого вихідного коду і знаходиться у вільному розповсюдженні. Це дозволяє розробникам отримати доступ до вихідного коду Android і зрозуміти, яким чином реалізовані властивості і функції додатків. Будь-який користувач може взяти участь в удосконаленні операційної системи Android. Для цього достатньо відправити звіт про виявлені помилки (http://source.android.com/source/report-bugs.html) або взяти участь в одній з дискусійних груп Open Source Project (http://source.android.com/community/index.html). В Інтернеті доступні різні додатки Android з відкритим вихідним кодом, які пропонує компанія Google та ряд інших розробників.

Спочатку ОС розроблялася компанією Android Inc., яку згодом (липень, 2005) купила Google. Згодом (листопад, 2007) Google ініціювала створення бізнес-альянсу Open Handset Alliance (до його складу увійшли Google, HTC, Intel, Motorola, Nvidia і інші компанії). Розробка і розвиток мобільної платформи Android виконується в рамках проекту AOSP (Android Open Source Project) під керуванням ОНА. З моменту виходу першої версії (вересень, 2008) відбулося декілька оновлень системи. Ці оновлення, як правило, стосуються виправлення виявлених помилок і додавання нового функціоналу в систему. Кожна версія системи отримує власне кодове ім'я на тему десерту (табл. 2.1).

Таблиця 2.1

Історія версій ОС Android

Кодове ім'я версії	Номер	Дата випуску	API рівень
1	**2**	**3**	**4**
Без кодового імені	1.0	23 вересня 2008	1
	1.1	9 лютого 2009	2
Cupcake	1.5	27 квітня 2009	3

Продовження таблиці 2.1

Кодове ім’я версії	Номер	Дата випуску	API рівень
1	**2**	**3**	**4**
Donut	1.6	15 вересня 2009	4
Eclair	2.0–2.1	26 жовтня 2009	5–7
Froyo	2.2–2.2.3	20 травня 2010	8
Gingerbread	2.3–2.3.7	6 грудня 2010	9–10
Honeycomb	3.0–3.2.6	22 лютого 2011	11–13
Ice Cream Sandwich	4.0–4.0.4	18 жовтня 2011	14–15
Jelly Bean	4.1–4.3.1	9 липня 2012	16–18
KitKat	4.4–4.4.4	31 жовтня 2013	19–20
Lollipop	5.0–5.1.1	12 листопада 2014	21–22
Marshmallow	6.0–6.0.1	5 жовтня 2015	23
Nougat	7.0–7.1.2	22 серпня 2016	24–25
Oreo	8.0–8.1	21 серпня 2017	26–27
Pie	9.0	6 серпня 2018	28
Android 10	10.0	3 вересня 2019	29
Android 11	11.0	в розробці	

Основні якісні відмінності Android 10 від попередніх версій:

- підтримка гнучких смартфонів;
- нова навігація у вигляді жестів;
- додано темну тему, що економить заряд на OLED екранах;
- додано програму Pixel themes, завдяки якій можна змінювати годинник на екрані, здійснювати блокування, змінювати шрифт, форму іконок, піктограми в панелі статусу і в панелі керування;
- припинена підтримка Android Beam;
- користувачам дозволено розширено налаштовувати дозвіл на контроль за місцезнаходження пристрою;
- нові права доступу до фонових фотографій, відео- та аудіофайлів;
- фонові програми більше не можуть переміщуватися на передній план;
- покращена конфіденційність;
- динамічний формат глибини для фотографій, який дозволяє змінювати ступінь розмитості фону після зйомки;

- підтримка відеокодека AV1, відеоформату HDR10+ і аудіокодека Opus;
- вбудований MIDI API, який дозволяє взаємодію з музичними контролерами;
- покращена підтримка біометричної автентифікації в програмах.

У жовтні 2012 року виконавчий директор компанії Ларрі Пейдж повідомив, що було активовано більше 500 мільйонів смартфонів і планшетів на базі Android, а також заявив, що щодня активується 1,3 мільйона пристроїв на базі цієї операційної системи. На початок вересня 2013 року було оголошено про те, що в світі вже активовано понад мільярд пристроїв на Android. 29 вересня 2015 року CEO Google Сундар Пічаї зазначив, що число користувачів пристроїв на базі Android перевищила 1,4 млрд. У травні 2017 року компанія повідомила про 2 мільярди активованих Android-пристроїв

ОС Android залишається найпопулярнішою платформою для мобільних пристроїв на сьогоднішній день. Біля 70% смартфонів та біля 40% планшетів у світі працюють під управлінням цієї ОС (за даними сайту http://statcounter.com).

Інтернет-магазин Google Play працює в 190 країнах світу і налічує понад 2,8 млн. додатків, а за весь час роботи сервісу набралося близько 330 млрд завантажень додатків (станом на літо 2018 р.). Станом на 2017 рік на долю Google Play припало біля 70% завантажень мобільних додатків у світі.

Критика платформи

- у деяких Android-пристроях є сервіси Google, що забезпечують можливість передачі ідентифікаційної інформації на сервери компанії, наприклад, інформації про переміщення користувача в реальному часі;
- для доступу до Google Play та інших сервісів від Google необхідно використовувати пропрієтарні додатки, які виробник телефону має право встановлювати на телефон тільки після укладення контракту з Google;
- конкуренти Android звинувачуючи її в надмірній фрагментації, що створює перешкоди для розробників;
- за даними Lookout Security Mobile, за 2011 рік у користувачів Android-смартфонів було вкрадено близько мільйона доларів США (напр., шляхом відправки СМС без відома власника телефону);
- інтерфейс постійно змінюється від версії до версії;
- із кожною наступною версією все менше компонентів ОС мають відкритий вихідний код.

2.2 Архітектура ОС Android

Розглянемо основні компоненти операційної системи Android (рис. 2.1):

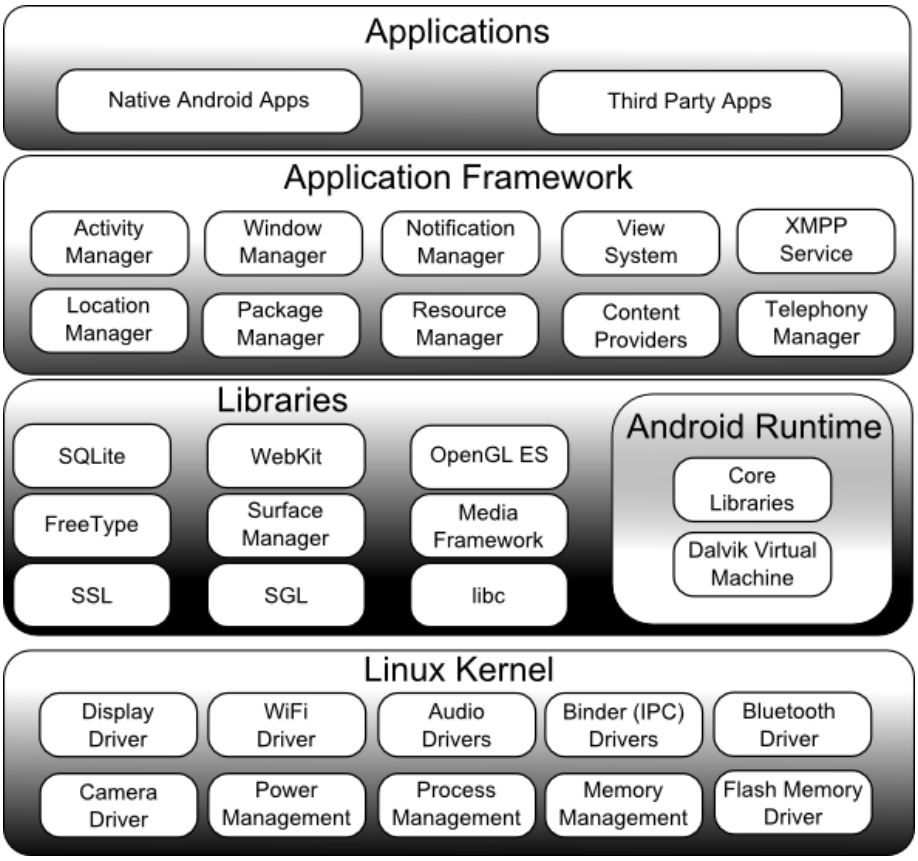


Рисунок 2.1 Архітектура ОС Android

З точки зору архітектури, система Android представляє собою повний програмний стек, в якому можна виділити такі рівні:

Рівень додатків (Applications) – набір встановленого прикладного програмного забезпечення;

Рівень каркасу додатків (Application Framework) забезпечує розробникам доступ до API, наданих компонентами системи рівня бібліотек;

Набір бібліотек і середовище виконання (Libraries & Android Runtime) забезпечує найважливіший базовий функціонал для додатків, містить віртуальну машину Dalvik і базові бібліотеки Java необхідні для запуску Android додатків;

Базовий рівень (Linux Kernel) – рівень абстракції між апаратним рівнем і програмним стеком.

Розглянемо детальніше ці рівні.

Applications. Android постачається з набором основних додатків, що включає в себе календар, карти, браузер, менеджер контактів і ін. Всі перераховані програми написані на Java.

Application Framework. Будучи відкритою платформою, Android дає розробникам можливість створювати гнучкі та інноваційні програми. Розробники можуть використовувати апаратні можливості пристрою, отримувати інформацію про місцезнаходження, виконувати завдання у фоновому режимі, встановлювати оповіщення та багато іншого. Розробники мають повний доступ до тих же API, що використовуються в основних додатках.

Архітектура додатків розроблена з метою спрощення повторного використання компонентів; будь-який додаток може «публікувати» свої можливості і будь-яке інший додаток може потім використовувати ці можливості (з урахуванням обмежень безпеки). Цей же механізм дозволяє замінювати стандартні компоненти на призначені користувацькі.

Libraries. Android включає в себе набір C/C++ бібліотек, які використовуються різними компонентами системи. Ці можливості доступні розробникам в контексті застосування Android Aplication Framework. Деякі основні бібліотеки, перераховані нижче:

- Медіа бібліотеки – ці бібліотеки надають підтримку відтворення і запису багатьох популярних аудіо, відео форматів і форматів зображень, в тому числі MPEG4, MP3, AAC, AMR, JPG, PNG і інших;
- Surface Manager – управляє доступом до підсистеми відображення 2D і 3D графічних шарів;
- LibWebCore – сучасне веб-ядро, на якому побудований браузер Android;
- SGL – основне графічне ядро 2D;
- 3D бібліотеки – реалізовані на основі OpenGL; бібліотеки використовують або апаратне 3D-прискорення (при його наявності), або вмикаються програмно;
- FreeType – підтримка растрових і векторних шрифтів;
- SQLite – механізм бази даних, доступний для всіх додатків.

Android Runtime. Android включає в себе набір основних бібліотек, які забезпечують більшість функцій, доступних в бібліотеках Java. Кожна програма Android працює в своєму власному процесі, зі своїм власним екземпляром віртуальної

машини Dalvik. Dalvik була написана так, що пристрій може працювати ефективно з декількома віртуальними машинами одночасно.

Dalvik проектувалася спеціально під платформу Android. Віртуальна машина оптимізована для низького споживання пам'яті та роботи на мобільному апаратному забезпеченні. Dalvik використовує власний байт-код. Android-додатки перекладаються компілятором в спеціальний машинно-незалежний низькорівневий код. І саме Dalvik інтерпретує і виконує таку програму при виконанні на платформі. Крім того, за допомогою спеціальної утиліти, що входить до складу Android SDK, Dalvik здатна перекладати байт-коди Java в коди власного формату і також виконувати їх у своїй віртуальному середовищі.

Linux Kernel. Android базується на Linux 2.6 з основними системними службами – безпека, управління пам'яттю, управління процесами і модель драйверів. Розробники Android модифікували ядро Linux, додавши підтримку апаратного забезпечення, використовуваного в мобільних пристроях і, найчастіше, недоступного на комп'ютерах.

Розробник зазвичай взаємодіє з двома верхніми рівнями архітектури Android для створення нових додатків. Бібліотеки, система виконання і ядро Linux приховані за каркасом додатків та абстракцій.

Для того, щоб встановити програму на пристроях з ОС Android створюється файл з розширенням *.apk (Android package), який містить виконувані файли, а також допоміжні компоненти, наприклад, файли з даними і файли ресурсів. Після установки на пристрій кожен додаток «живе» в своєму власному ізольованому екземплярі віртуальної машини Dalvik. У версіях, починаючи з Android 4.4 Kitkat, є можливість переключитися з Dalvik на більш швидкий ART (Android Runtime, середовище виконання, розроблене Google). В Android 5.0 Dalvik був повністю замінений на ART.

2.3 Інструменти розробника

Як правило, розробка Android-додатків здійснюється на мові Java. Тому, в першу чергу, необхідно встановити Java Development Kit (JDK).

Java – це об'єктно-орієнтована мова програмування. Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java, яка обробляє байт-код і передає інструкції обладнанню як інтерпретатор. Перевага подібного способу виконання програм полягає в повній незалежності байт-коду від операційної

системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером) викликають негайне їх переривання. Слід зауважити, що фактично, більшість архітектурних рішень, прийнятих при створенні Java, було продиктовано бажанням надати синтаксис, схожий з C/C++. В Java використовується практично ідентичний синтаксис для оголошення змінних, передачі параметрів і операторів. Тому ті, хто вже має досвід програмування на C/C++, зможуть швидко освоїтися і почати писати Java-додатки.

Android Software Development Kit (SDK) містить багато інструментів і утиліт для створення і тестування додатків. Наприклад, за допомогою SDK Manager можна встановити Android API будь-якої версії (рис. 2.2), а також перевірити репозиторій на наявність доступних, але ще не встановлених пакетів і архівів.

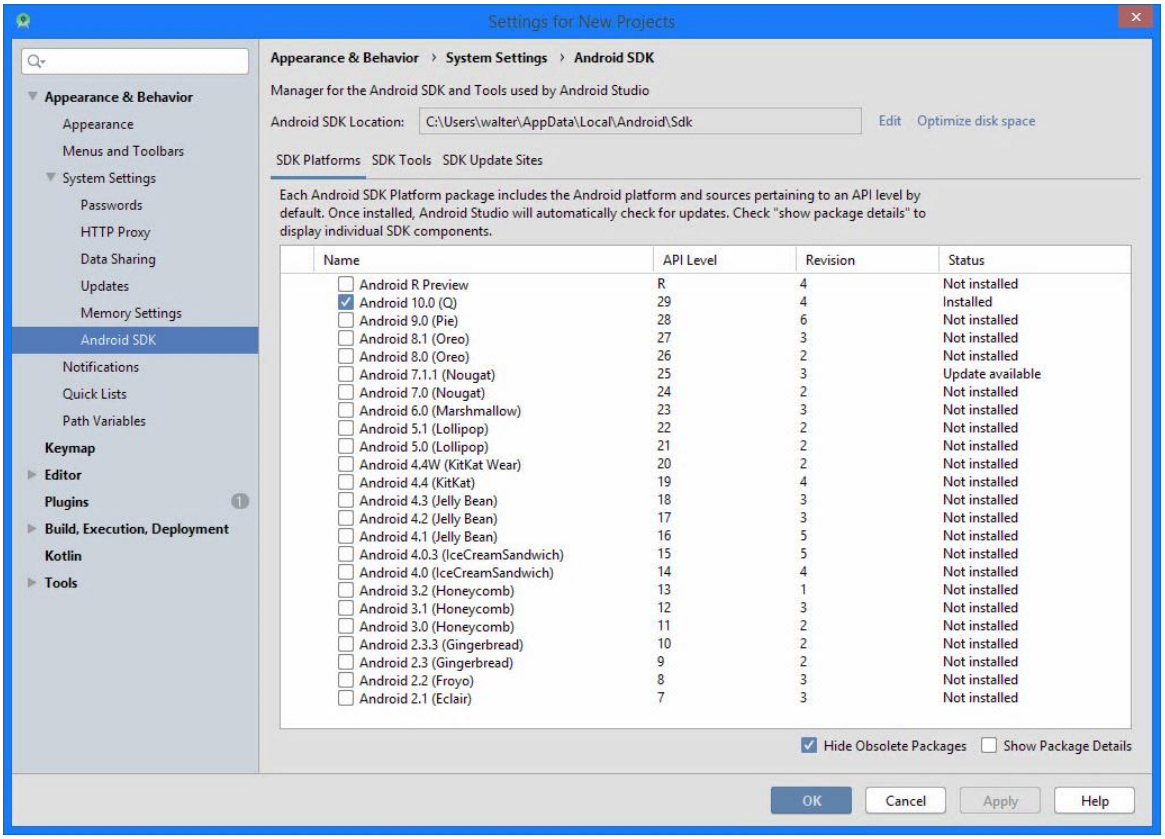


Рисунок 2.2 Вікно Android SDK

JDK (Java Development Kit – це безкоштовно розповсюджуваний комплект розробки додатків на мові Java, що включає в себе компілятор Java, стандартні бібліотеки класів Java, зразки коду, документацію, різні утиліти і систему Java Runtime Environment (JRE). До складу JDK не входить інтегроване середовище розробки IDE (Integrated Development Environment). Тому після того, як буде встановлено JDK, слід встановити IDE.

Найпопулярніші середовища розробки під Android

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation.

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains.

Android Studio – середовище розробки під Android, яке базується на IntelliJ IDEA. Надає інтегровані інструменти для розробки і налагодження. Додатково до всіх можливостей, очікуваним від IntelliJ, в Android Studio реалізовані:

- підтримка процесу збирання додатку на основі Gradle;
- специфічний для Android рефакторинг і швидке виправлення дефектів;
- інструменти для пошуку проблем з продуктивністю, з юзабіліті, з сумісністю версій та інших;
- можливості ProGuard (утиліти для скорочення, оптимізації і обфускації коду) і підписування додатків;
- майстри для створення загальних Android конструкцій і компонентів, які працюють на основі шаблонів;
- WYSIWYG-редактор, який працює на екранах багатьох розмірів та розширень, вікно попереднього перегляду, що показує запущений додаток одразу на декількох пристроях і в реальному часі;
- вбудована підтримка хмарної платформи Google.

Intel XDK – середовище для розробки кросплатформних мобільних додатків; включає в себе інструменти для створення, налагодження та збірки ПЗ, а також емулятор пристроїв; підтримує розробку для Android, Apple iOS, Microsoft Windows 8, Tizen; підтримує мови розробки: HTML5 і JavaScript.

Intel Beacon Mountain – ще одне середовище розробки від компанії Intel. Надає всі інструменти, необхідні для проектування, розробки, налагодження та оптимізації додатків під Android. Підтримує розробку для цільових платформ на основі процесорів

Intel Atom і ARM. Beacon Mountain побудована на основі Android IDE (Eclipse, Android ADT, Android SDK).

Marmalade SDK – кросплатформне SDK від Ideaworks3D Limited. Являє собою набір бібліотек, зразків коду, інструментів і документації, необхідних для розробки, тестування і розгортання додатків для мобільних пристроїв. Використовується, в основному, для розробки ігор.

2.4. Емулятори

Емуляція (англ. Emulation) в обчислювальній техніці – комплекс програмних, апаратних засобів або їх поєднання, призначений для копіювання (або емуляції) функцій однієї обчислювальної системи (гостя) на інший, відмінній від першої, обчислювальної системі (хост) таким чином, щоб поведінка, яка емулюється, як можна ближче відповідала поведінці оригінальної системи (гостя). Метою є максимально точне відтворення поведінки на відміну від різних форм комп'ютерного моделювання, в яких імітується поведінка деякої абстрактної моделі.

Емулятор – це віртуальний мобільний пристрій, який запускається на комп'ютері. За допомогою емулятора можна розробляти і тестувати програми без використання реальних пристроїв. На рис. 2.3 наведено приклад запущеного стандартного емулятора.

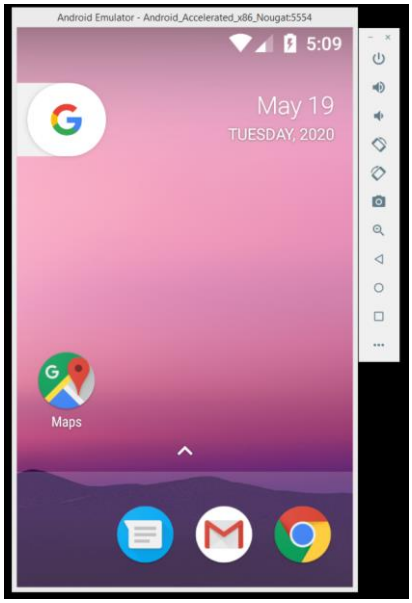


Рисунок 2.3 Стандартний емулятор Android

До переваг використання емуляторів можна віднести простоту їх використання і низьку вартість. Розробнику не потрібно купувати величезну кількість пристроїв з різними характеристиками, щоб перевірити працездатність програми на різних смартфонах. Досить створити кілька емуляторів з необхідними характеристиками і запустити на них додаток. На жаль, емулятори мають і ряд недоліків:

1) вимагають багато системних ресурсів;

2) через відмінності в архітектурі процесорів комп'ютера і смартфона повільно запускаються. Сучасні персональні комп'ютери побудовані на архітектурі x86 і x64, а більшість процесорів смартфонів на Android – ARM. Процес емуляції однієї архітектури на інший надзвичайно складний і відбувається досить повільно;

3) у деяких випадках стандартного емулятора недостатньо. Йдеться про можливості смартфонів, якими звичайні комп'ютери не володіють (наприклад, наявність датчика GPS або акселерометра). У таких випадках повноцінне відлагодження можна провести тільки з використанням реального пристрою.

Альтернативні емулятори

Стандартний емулятор, що поставляється разом з Android SDK, не влаштовує багатьох. Існують проекти, що підтримують розробку та розвиток альтернативних емуляторів. Як приклад можна привести Genymotion (https://www.genymotion.com/) – швидкий емулятор Android (на думку його розробників). Він містить попередньо налаштовані образи Android (x86 з апаратним прискоренням OpenGL). Genymotion доступний для Linux, Windows і MacOS X і вимагає для своєї роботи VirtualBox. Іншими словами, Genymotion є віртуальною машину з встановленою ОС Android, яку користувач запускає так само, як і інші віртуальні машини. Проблема високого споживання системних ресурсів, звичайно ж не зникає, проте швидкість запуску істотно збільшується. В даний час активно розвивається.

Можливість відлагодження на реальних пристроях

Розроблений додаток можна запустити на реальному пристрої, наприклад, на смартфоні. Для цього необхідно виконати попередню роботу. Для відлагодження додатків на реальних пристроях, як правило, необхідно:

1. налаштувати пристрій (включити режим налагодження по USB);
2. налаштувати комп'ютер (для Windows необхідно встановити потрібний драйвер вручну, потрібні права адміністратора);
3. налаштувати середовище розробки і запустити додаток на пристрої.

ЛЕКЦІЯ 3. РОЗРОБКА ПРОГРАМ В СЕРЕДОВИЩІ IDE ANDROID STUDIO

3.1 Створення проекту в середовищі Android Studio

Після запуску Android Studio натискаємо на кнопку *Start a new Android Studio project* (рис. 3.1).

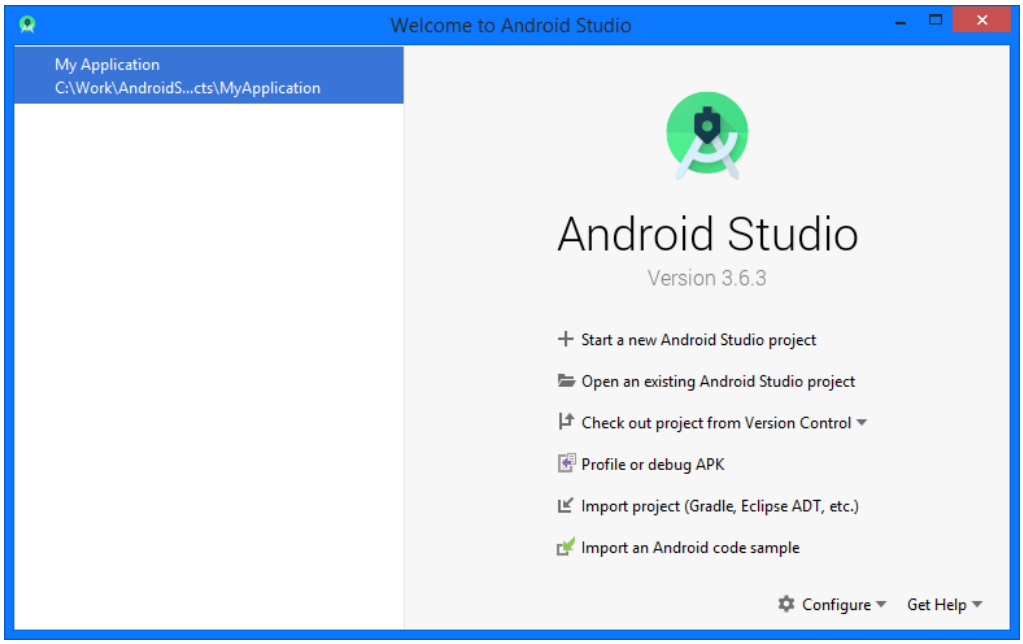


Рисунок 3.1 Стартове вікно Android Studio

Далі необхідно вибрати шаблон розмітки (рис. 3.2).

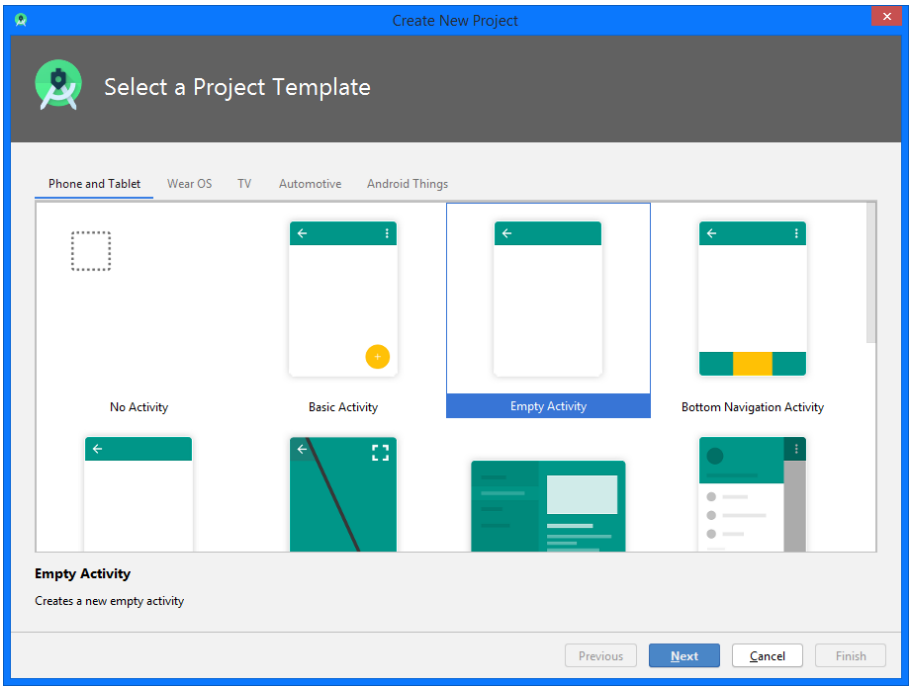


Рисунок 3.2 Вікно вибору шаблону проекту

Наступний крок – конфігурування проекту (рис. 3.3).

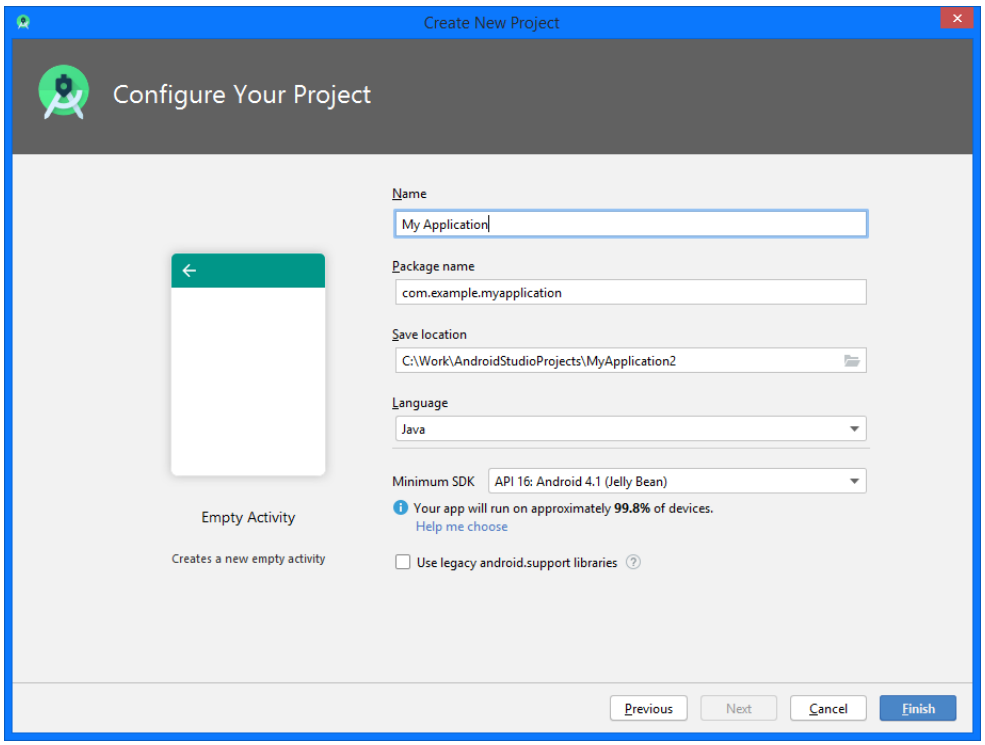


Рисунок 3.3 Вікно конфігурування проекту

Тут:

- **Name** – Найменування проекту. Воно буде відображатися в списку проектів при відкритті Android Studio;
- **Package name** – це поняття із світу мови Java. Якщо коротко, то це префікс для імен класів нашого додатка;
- **Project location** – папка на комп'ютері, де будуть знаходитися всі файли проекту;
- **Language** – використовувана в проекті мова програмування;
- **Minimum SDK** – мінімальна версія Android, на якій можна буде запустити додаток.

Після натискання на кнопку Finish Android Studio згенерує всі необхідні для проекту файли (рис. 3.4).

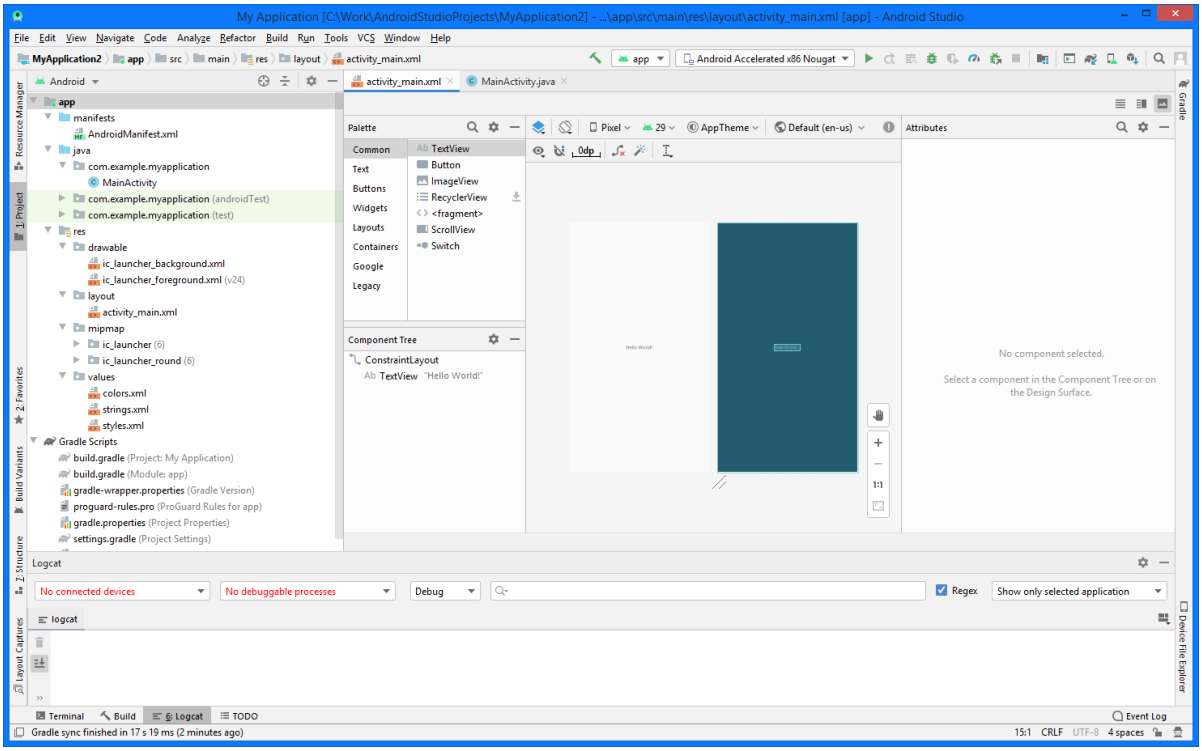


Рисунок 3.4 Створений новий проект в середовищі Android Studio

3.2 Структура проекту

Основні папки та файли проекту показано на рис 3.5. Основним елементами структури проекту є:

- в папці *manifest* розміщується файл маніфесту додатку *AndroidManifest.xml*;
- в папці *java* розміщуються файли класів проекту (програмний код);
- в папці *res* розміщуються файли ресурсів проекту. Зокрема, розмітка екранів проекту повинна знаходиться в папці *res/layout*;
- у вітці дерева проекту *GradleScripts* розміщуються файли інструмента Gradle, призначеного для автоматизації процесу збирання (assembly) проекту.

Варіант відображення структури проекту можна вибрати із списку, який з'являється при натисканні на кнопку (рис. 3.6).

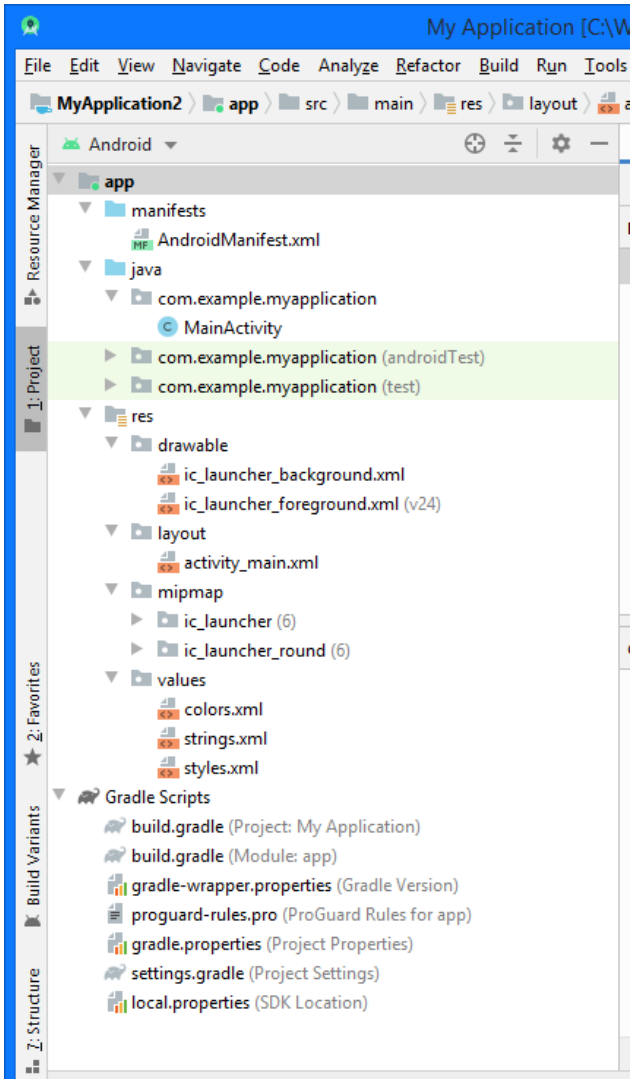


Рисунок 3.5 Структура проекту в середовищі Android Studio

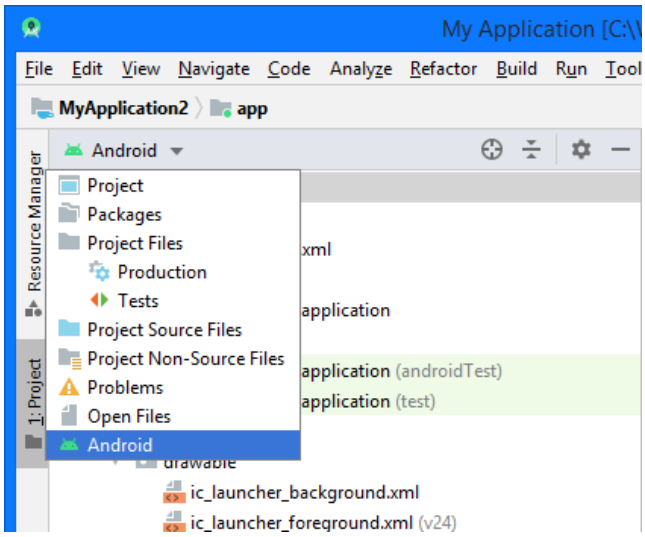


Рисунок 3.6 Вибір варіанту відображення структури проекту

3.3 Конфігурування та запуск емулятора

Для роботи із емуляторами призначений AVD Manager (Android virtual Device Manager), рис. 3.7, який можна відкрити відповідною кнопкою, розміщеною на панелі справа зверху (рис. 3.8).

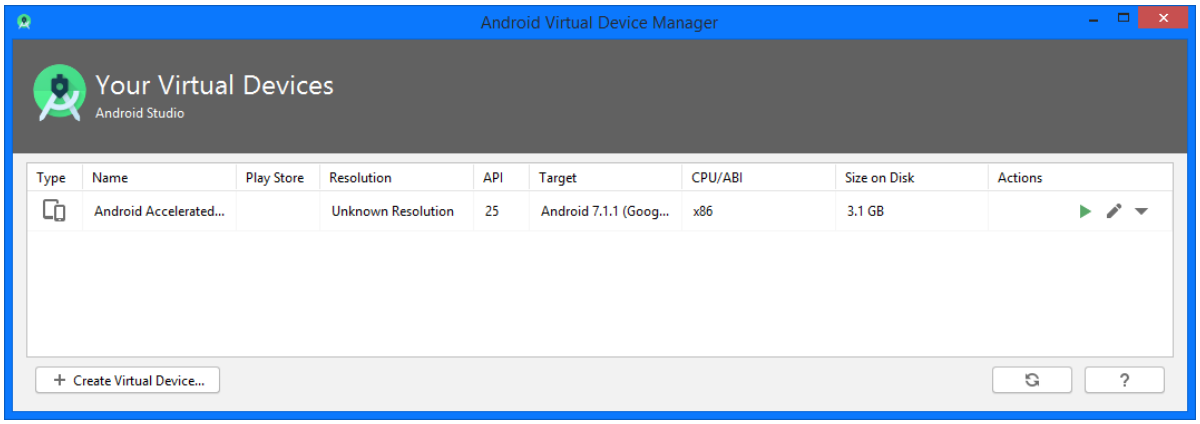


Рисунок 3.7 Вікно AVD Manager

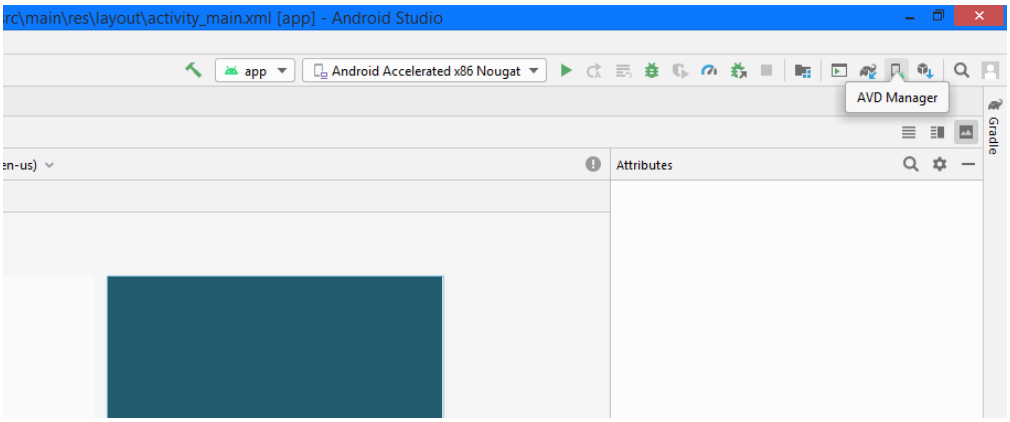


Рисунок 3.8 Розміщення кнопки для відкривання вікна AVD Manager

Поруч із нею знаходиться кнопка для відкривання SDK Manager (рис. 3.9)

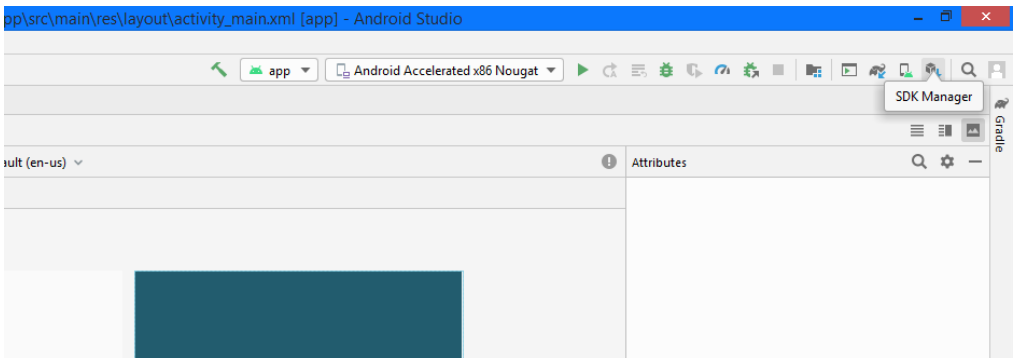


Рисунок 3.9 Розміщення кнопки для відкривання вікна SDK Manager

Елементи управління, призначені для вибору пристрою для відлагодження додатка (фізичний пристрій або емулятор) та запуску процесу емуляції знаходяться на тій же панелі лівіше (рис. 3.10).

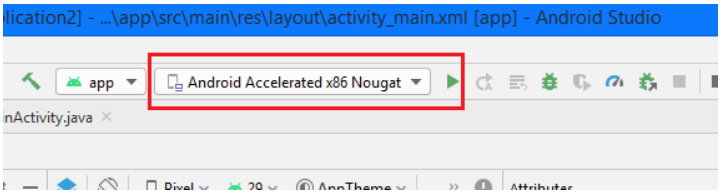


Рисунок 3.10 Розміщення елементів управління
для запуску процесу відлагодження проекту

Для створення нового емулятора слід натиснути на кнопку *Create Virtual Device…* у вікні AVD Manager. З'явиться вікно, де ми можемо вказати параметри пристрою (рис 3.11).

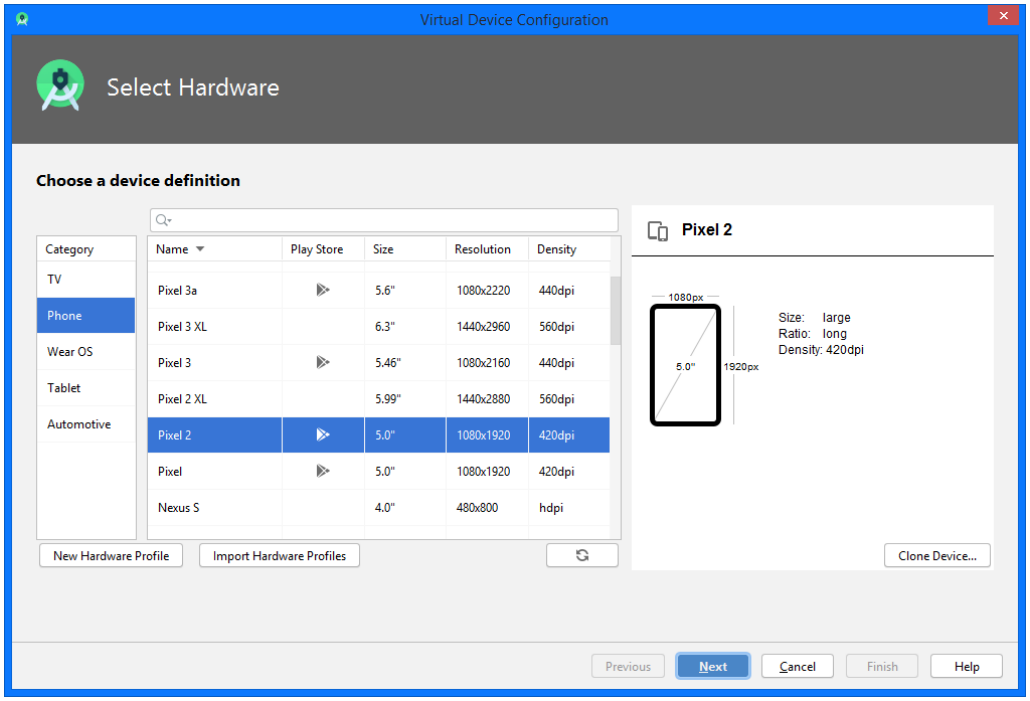


Рисунок 3.11 Вікно задання параметрів пристрою, на основі якого відбуватиметься відлагодження програми

Після натискання на кнопку Next буде запропоновано завантажити необхідний для емуляції образ ОС Android (рисунок 3.12), якщо його не було завантажено раніше.

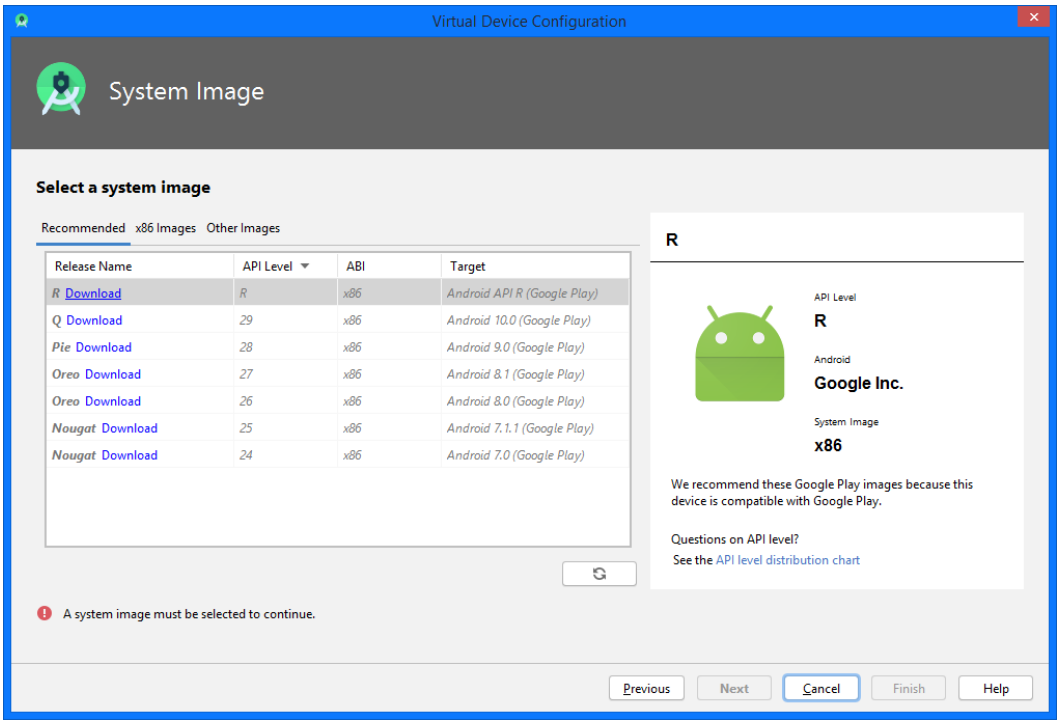


Рисунок 3.12 Вікно для вибору і завантаження
необхідного для емуляції образу ОС Android

Додаток, запущений на виконання на емуляторі фізичного пристрою представлено на рис. 3.13.

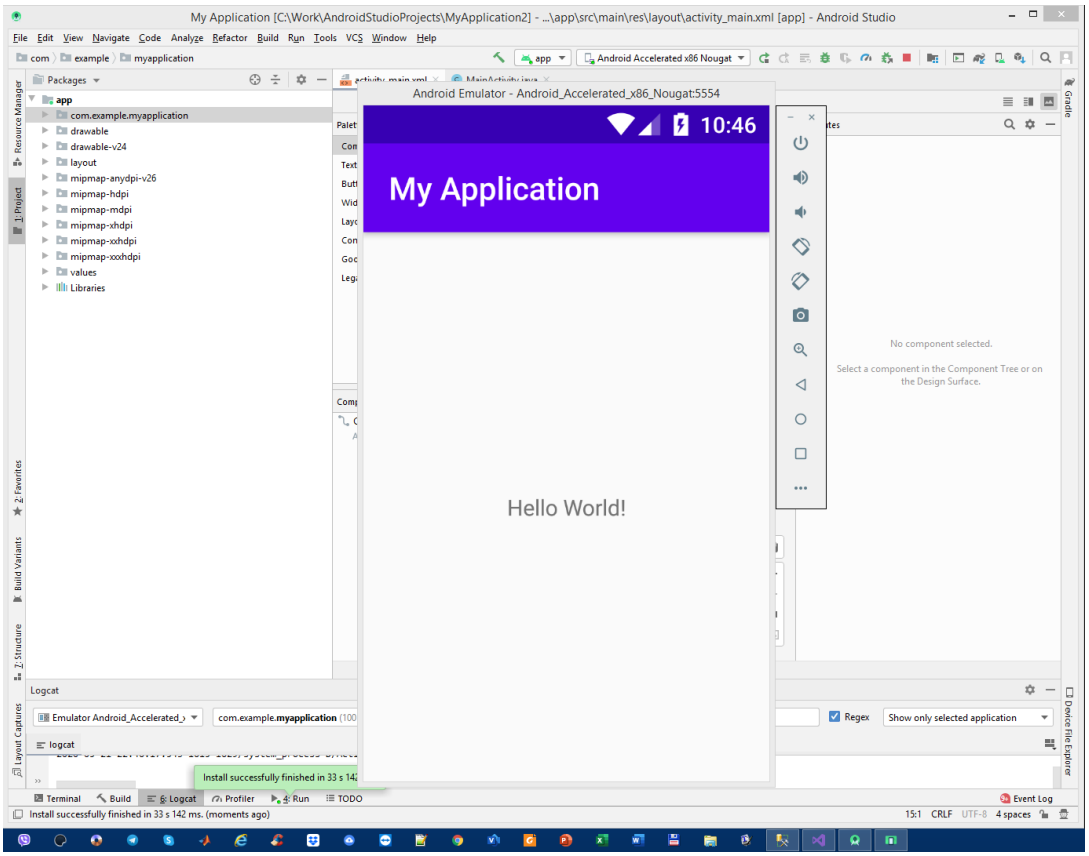


Рисунок 3.13 Запущений на виконання на емуляторі фізичного пристрою додаток

3.4 Запуск додатку з метою відлагодження на фізичному пристрої

Для запуску додатку на фізичному пристрої необхідно здійснити два кроки:

1. Підключити пристрій до комп’ютера за допомогою кабелю USB. Якщо ви розробляєте під управлінням ОС Windows, то вам може знадобитися встановити відповідний драйвер USB для вашого пристрою.
2. Увімкнути режим відлагодження через USB.

Для здійснення другого кроку необхідно перейти на екран налаштувань параметрів розробника (Developer options screen). На ОС Android 4.1 і новіших версій екран налаштувань параметрів розробника доступний за замовчуванням. На ОС Android 4.2 та новіших версій потрібно увімкнути цей екран. Щоб увімкнути параметри розробника, торкніться опції номер збірки (Build number) 7 разів. Цю опцію можна знайти в одному з наступних місць, залежно від версії Android:

- Android 9 (API level 28) та вище: *Settings>About Phone>Build Number*;

- Android 8.0.0 (API level 26) та Android 8.1.0 (API level 26): *Settings>System > About Phone>Build Number*;
- Android 7.1 (API level 25) та нижче: *Settings>About Phone>Build Number*.

У верхній частині екрану є перемикач для ввімкнення/вимкнення параметрів розробника (рис. 3.14).

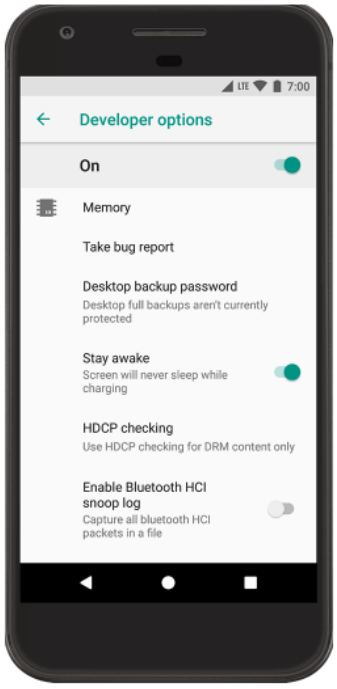


Рисунок 3.14 Екран налаштувань параметрів розробника

Щоб увімкнути режим відлагодження через USB, увімкніть відповідну опцію у меню параметрів розробника. Цю опцію можна знайти в одному з наступних місць, залежно від версії Android:

- Android 9 (API level 28) та вище: *Settings>System>Advanced>Developer Options>USB debugging*;
- Android 8.0.0 (API level 26) та Android 8.1.0 (API level 26): *Settings>System>Developer Options>USB debugging*;
- Android 7.1 (API level 25) та нижче: *Settings>Developer Options>USB debugging*.

3.5 Види Android-додатків

Виділяють такі основні види додатків під ОС Android:

1. **Додатки переднього плану**, які виконують свої функції лише тоді, коли видимі на екрані, в іншому ж випадку їх виконання призупиняється. Такими додатками є, наприклад, ігри, текстові редактори, відеопрогравачі. При розробці цих програм потрібно дуже уважно вивчити життєвий цикл активності, щоб перемикання в фоновий режим і назад відбувалося плавно. Тобто, йдеться про коректне збереження стану додатку. Ще одним важливим моментом, на який обов'язково треба звернути увагу при розробці додатків переднього плану є зручний і інтуїтивно зрозумілий інтерфейс для користувача.

2. **Фонові додатки**, які після налаштування не потребують взаємодії з користувачем а більшу частину часу перебувають і працюють в прихованому стані. Прикладами таких додатків можуть бути служби екранування дзвінків, SMS-автовідповідачі. Здебільшого фонові програми націлені на відстеження подій, породжуваних апаратним забезпеченням, системою або іншими додатками. Можна створювати абсолютно невидимі сервіси, але тоді вони будуть некерованими. Мінімум дій, які необхідно дозволити користувачеві це: санкціонування запуску сервісу, налаштування, припинення і переривання його роботи при необхідності.

3. **Змішані додатки**, які більшу частину часу працюють у фоновому режимі, проте допускають взаємодію з користувачем і після свого налаштування. Зазвичай взаємодія з користувачем зводиться до повідомлення про які-небудь події. Прикладами таких додатків можуть бути мультимедіа-програвачі, додатки для обміну текстовими повідомленнями (чати), поштові клієнти. Можливість реагувати на введення користувачем інформації без втрати працездатності у фоновому режимі є характерною особливістю змішаних додатків. Такі програми зазвичай містять як видимі активності, так і приховані (фонові) сервіси, і при взаємодії з користувачем повинні враховувати свій поточний стан. Можливо їм буде необхідно оновлювати графічний інтерфейс, якщо додаток знаходиться на передньому плані, або ж посилати користувачеві повідомлення з фонового режиму, щоб тримати його в курсі того, що відбувається. Такі особливості необхідно враховувати при розробці подібних додатків.

4. **Віджети** – це невеликі додатки, які відображаються у вигляді графічного об'єкта на робочому столі. Прикладами можуть бути додатки для відображення динамічної інформації, такої як заряд батареї, прогноз погоди, дата і час.

Зрозуміло, що складні додатки можуть містити елементи кожного з розглянутих вище видів. Плануючи розробку конкретної програми необхідно визначити спосіб її використання і тільки після цього переходити до проектування і подальшої розробки.

3.6 Файл маніфесту AndroidManifest.xml

Кожна програма містить файл маніфесту *AndroidManifest.xml*. Даний файл визначає важливу інформацію про програму: назва, версія, іконки, які розширення додаток використовує, реєструє всі використовувані класи activity, сервіси і т.д.

Файл маніфесту може виглядати так:

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

Кореневим елементом є вузол manifest. В даному випадку тільки визначається пакет додатку – *package="com.example.eugene.viewsapplication"*.

Більшість налаштувань рівня додатку визначаються елементом application. Наприклад, через атрибут *android:icon="@mipmap/ic_launcher"* задається іконка програми, яка знаходиться в каталозі *res /mipmap-xxxx*.

Також тут задається назва додатку, яка буде відображатися на мобільному пристрої в списку додатків і в заголовку: *android:label="@string/app_name"*. В даному випадку вона зберігається в стрічковому ресурсі.

Вкладені елементи *activity* визначають всі використовувані в додатку activity. В даному випадку видно, що в додатку є тільки одна activity – MainActivity.

Елемент *intent-filter* в MainActivity вказує, як дана activity буде використовуватися. Зокрема, за допомогою вузла action *android:name="android.intent.action.MAIN"* вказано, що дана activity буде вхідною точкою в додаток і не повинна отримувати будь-які дані ззовні.

Елемент category *android:name="android.intent.category.LAUNCHER"* вказує, що MainActivity представлятиме стартовий екран, який відображається під час запуску програми.

ЛЕКЦІЯ 4. ACTIVITY

4.1 Компоненти Android-додатку

Кожен додаток під управлінням ОС Android запускається в своєму власному процесі. Тому додаток ізольовано від інших запущених додатків, і некоректно працюючий додаток не може нашкодити іншим запущеним на виконання додаткам.

Проте, важливою властивістю додатку є можливість використовувати компоненти інших додатків, якщо вони дають на це відповідні права. Припустимо, нам потрібен якийсь компонент з прокруткою для відображення тексту, і схожий компонент вже реалізований в іншому додатку. Тоді у нас є можливість використовувати реалізований компонент. У цьому випадку наш додаток не копіює необхідний код до себе і не створює посилання на нього. Замість цього додаток робить запит на виконання частини коду іншої програми, де є потрібний нам компонент.

Виділяють чотири типи компонентів Android-програми: *Activities*, *Services*, *Broadcast receivers* і *Content providers*.

Також важливо відзначити об’єкти *Intents*, в Android-додатках майже все працює завдяки їм. *Intent* - це механізм для опису однієї операції (вибрати фотографію, відправити лист, зробити дзвінок, запустити браузер і перейти за вказаною адресою тощо). Найбільш поширений сценарій використання *Intent* – запуск іншого *Activity* в додатку.

4.1.1 Services

Service – це певний процес, який запускається у фоновому режимі. Наприклад, Service може отримувати дані по мережі, виконувати будь-які тривалі обчислення. Хорошим прикладом Service служить програвач музики. Користувач може вибрати будь-яку пісню в програвачі, включити її і закрити плеєр. Музика буде програватися в фоновому процесі. Service для програвання музики буде працювати, навіть якщо Activity плеєра буде закритою.

Подібно до Activity, Service має свої методи життєвого циклу:

- *void onCreate();*
- *void onStart(Intent intent);*
- *void onDestroy().*

У повному життєвому циклі Service існує два вкладених цикли:

1. Повне життя Service – це проміжок між моментом виклику методу *onCreate()* і моментом повернення *onDestroy()*. Подібно до Activity, для Services здійснюють початкову ініціалізацію в *onCreate()* і звільняють всі зайняті ним ресурси в *onDestroy()*;

2. Активне життя Service – починається з виклику методу *onStart()*. Цьому методу передається об'єкт Intent, який передавався в *startService()*.

Як і Activities, Services запускаються в головному потоці процесу додатку. Тому їх слід запускати в окремому потоці, щоб вони не блокували ні інші компоненти ні інтерфейс користувача.

4.1.2 Broadcast receivers

Broadcast receiver – це компонент, який розсилає і реагує на широкомовні повідомлення. Прикладом широкомовних компонентів можуть бути: повідомлення про перехід на літній/зимовий час, повідомлення про мінімальний заряді батареї і т. д.

Broadcast receiver не відображує на екрані користувацький інтерфейс, але може запустити Activity при отриманні певного повідомлення або використовувати NotificationManager для залучення уваги користувача. Привернути увагу користувача можна, наприклад, вібрацією пристрою, програванням звуку або миготінням спалаху.

Приймач широкомовних повідомлень має єдиний метод життєвого циклу: *onReceive*(). Коли широкомовне повідомлення прибуває до одержувача, Android викликає його методом *onReceive*() і передає в нього об'єкт Intent, що містить саме повідомлення. Приймач широкомовних повідомлень є активним тільки під час виконання цього методу. Процес, який в даний час виконує Broadcast receiver, є пріоритетним процесом і буде збереженим, окрім випадків гострої нестачі пам'яті в системі.

Коли програма повертається з *onReceive*(), приймач стає неактивним і система вважає, що робота об'єкта Broadcast receiver закінчена. Процес з активним широкомовним одержувачем захищений від знищення системою. Однак процес, що містить неактивні компоненти, може бути знищений системою в будь-який момент, якщо пам'ять, яку він використовує, буде необхідна для інших процесів.

4.1.3 Content providers

Content providers надають доступ до даних (читання, додавання, оновлення). Content provider може надавати доступ до даних не тільки свого додатку, а й інших. Дані можуть розміщуватися в файловій системі чи в базі даних.

4.2 Activity

4.2.1 Поняття Activity

Ключовим компонентом створення візуального інтерфейсу в додатку є activity (активність). Часто activity асоціюється з окремим екраном або вікном додатку, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Додаток може включати одну або декілька activity (рис. 4.1).

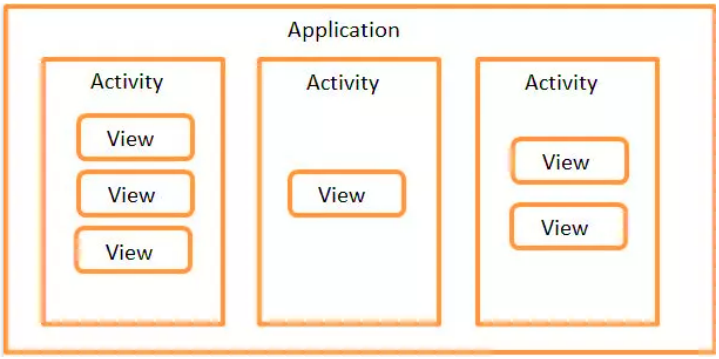


Рисунок 4.1 Представлення додатку як набору activity

Всі об’єкти activity є об’єктами класу *android.app.Activity*, який містить базову функціональність для всіх activity. Як правило activity успадковуються від класу *AppCompatActivity* який, успадковує базовий клас *Activity*:

```
public class MainActivity extends AppCompatActivity {
   // вміст класу
}
```

4.2.2 Життєвий цикл Activity

Всі додатки Android мають строго визначений життєвий цикл. При запуску користувачем додатку операційна система надає цьому додатку високий пріоритет.

Кожна програма запускається у вигляді окремого процесу, що дозволяє системі надавати одним процесам вищий пріоритет, на відміну від інших. Завдяки цьому, наприклад, при роботі з одними додатками не блокуються вхідні дзвінки. Після припинення роботи додатку, система звільняє всі пов'язані з ним ресурси, переводить додаток у розряд низькопріоритетного і закриває його.

Всі об'єкти activity, які є в додатку, управляються системою у вигляді стека activity, який називається back stack. При запуску нової activity вона поміщається на вершину стеку і виводиться на екран пристрою, поки не з'явиться нова activity. Коли поточна activity закінчує свою роботу (наприклад, користувач закриває програму), то вона видаляється із стеку, і відновлює роботу та activity, яка раніше була другою в стеці.

Після запуску activity проходить через ряд подій, які обробляються системою і для обробки яких існує ряд зворотних викликів:

```
protected void onCreate(Bundle saveInstanceState);
protected void onStart();
protected void onRestoreInstanceState(Bundle saveInstanceState);
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onSaveInstanceState(Bundle saveInstanceState);
protected void onStop();
protected void onDestroy();
```

Схематично взаємозв'язок між усіма цими зворотними викликами можна представити так, як це зроблено на рис 4.2.

Тут **onCreate()** – перший метод, з якого починається виконання activity. У цьому методі activity переходить в стан Created. Цей метод обов'язково повинен бути визначений в класі activity. У ньому відбувається початкове налаштування activity. Зокрема, створюються об'єкти візуального інтерфейсу. Цей метод отримує об'єкт Bundle, який містить попередній стану activity, якщо він булв збережений. Якщо activity заново створюється, то даний об'єкт має значення null. Якщо ж activity вже раніше була створена, але перебувала в призупиненому стані, то bundle містить пов'язану з activity інформацію.

В методі **onStart()** відбувається підготовка до виведення activity на екран пристрою. Як правило, цей метод не вимагає перевизначення, а всю роботу робить

вбудований код. Після завершення роботи методу activity відображається на екрані, викликається метод *onResume*(), а activity переходить в стан Resumed.

Після завершення методу onStart() викликається метод **onRestoreInstanceState(),** який призначений для відновлення збереженого стану з об'єкта Bundle, який передається в якості параметра. Але слід враховувати, що цей метод викликається тільки тоді, коли Bundle **не** дорівнює null і містить раніше збережений стан. Так, при першому запуску програми цей об'єкт Bundle матиме значення null, тому і метод onRestoreInstanceState() не викликатиметься.

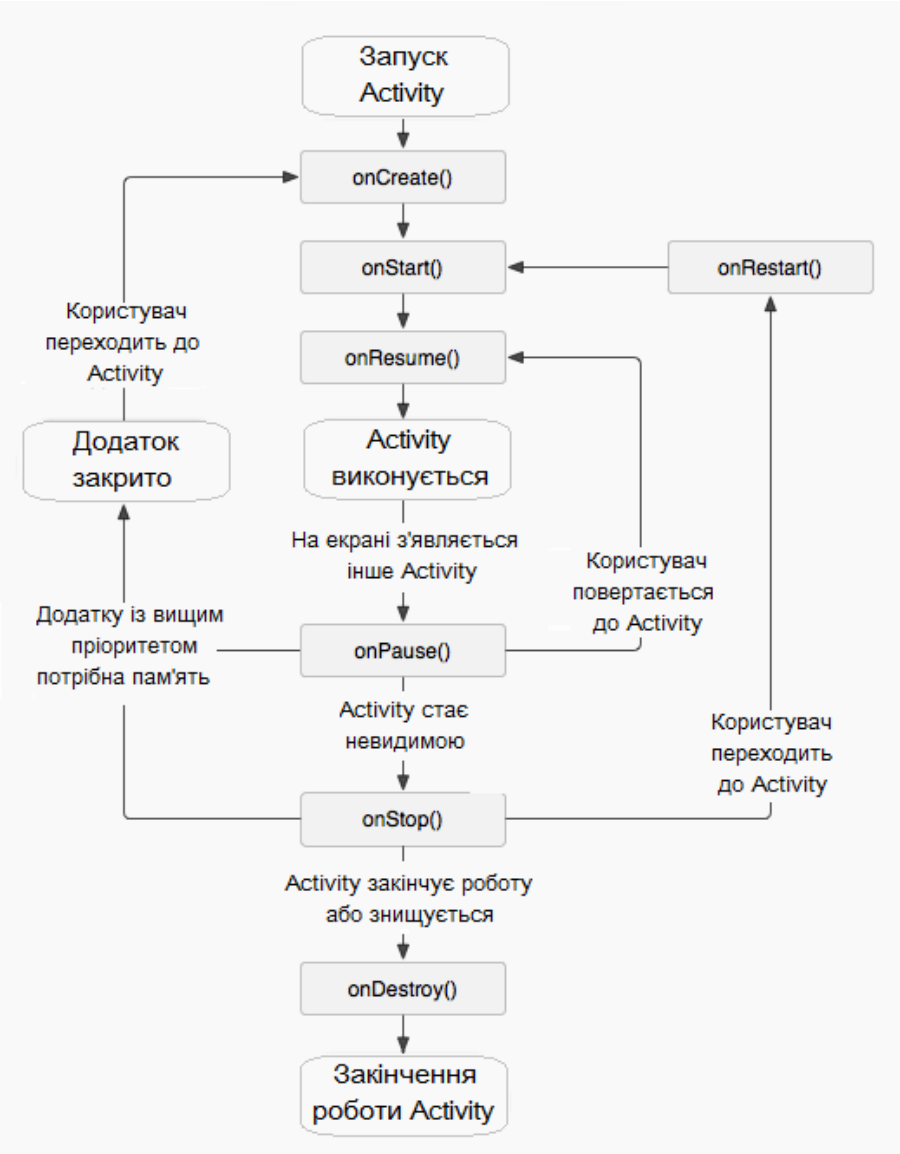


Рисунок 4.2 Життєвий цикл activity

При виклику методу **onResume()** activity переходить в стан Resumed, а користувач може з нею взаємодіяти. І власне activity залишається в цьому стані доти, поки вона не втратить фокус, наприклад, внаслідок перемикання на іншу activity або просто через вимкнення екрану пристрою.

Якщо користувач вирішить перейти до іншої activity, то система викликає метод **onPause()**. У цьому методі можна звільняти використовувані ресурси, припиняти процеси, наприклад, відтворення аудіо, анімацій, зупиняти роботу камери (якщо вона використовується) і т.д., для того, щоб вони менше позначалися на продуктивності системи.

Але треба враховувати, що на роботу даного методу відводиться дуже мало часу, тому не варто тут зберігати якісь дані, особливо якщо при цьому потрібно звернення до мережі, наприклад, відправка даних по інтернету, або звернення до бази даних.

Після виконання цього методу activity стає невидимою, не відображається на екрані, але вона все ще активна. І якщо користувач вирішить повернутися до цієї activity, то система викличе знову метод onResume(), і activity знову з'явиться на екрані.

Інший варіант роботи може виникнути, якщо раптом система бачить, що для роботи активних додатків необхідно більше пам'яті. І система може сама завершити повністю роботу activity, яка невидима і знаходиться в фоні. Або користувач може натиснути на кнопку Back (Назад). В цьому випадку у activity викликається метод onStop().

Метод **onSaveInstanceState()** викликається після методу onPause(), але до виклику onStop(). У onSaveInstanceState проводиться збереження стану програми в переданий в якості параметра об'єкт Bundle.

В методі **onStop()** activity переходить в стан Stopped. Тут слід звільнити використовувані ресурси, які не потрібні користувачеві, коли він не взаємодіє з activity. Тут також можна зберігати дані, наприклад, в базу даних.

При цьому в стані Stopped activity залишається в пам'яті пристрою, зберігається стан всіх елементів інтерфейсу. Наприклад, якщо в текстове поле EditText був введений якийсь текст, то після відновлення роботи activity і переходу її в стан Resumed ми знову побачимо в текстовому полі раніше введений текст.

Якщо після виклику методу onStop() користувач вирішить повернутися до колишньої activity, тоді система викличе метод onRestart(). Якщо ж activity повністю завершила свою роботу, наприклад, внаслідок закриття програми, то викликається метод onDestroy().

Завершується робота activity викликом методу **onDestroy()**, який відбувається або якщо система вирішить завершити роботу activity, або при виклику методу finish().

Також слід зазначити, що при зміні орієнтації екрану система завершує activity і потім створює її заново, викликаючи метод onCreate().

В цілому перехід між станами activity можна представити наступною схемою (рис 4.3):

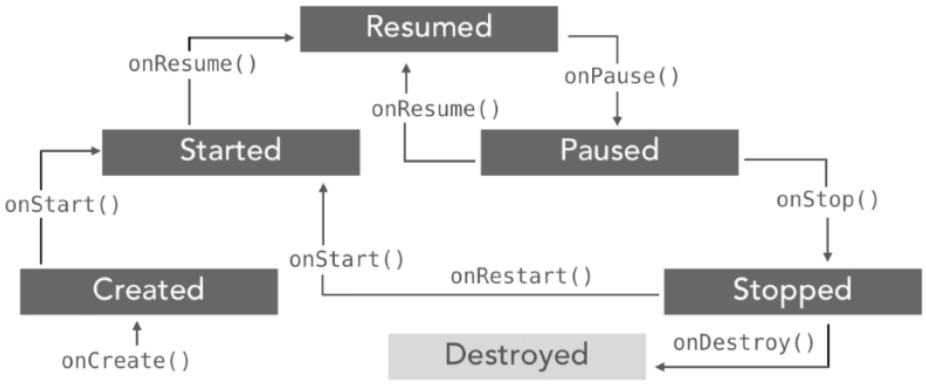


Рисунок 4.3 Діаграма переходів між станами Activity

Розглянемо декілька типових ситуацій. Якщо ми працюємо з Activity і потім перемикається на іншу програму, або натискаємо на кнопку Home, то у Activity викликається наступний ланцюжок методів: *onPause()->onStop().* Activity переходить в стан Stopped. Якщо користувач вирішить повернутися до Activity, то викликається наступний ланцюжок методів: *onRestart()->onStart()->onResume().*

Інша типова ситуація, коли користувач натискає на кнопку Back (Назад), то викликається наступний ланцюжок *onPause()->onStop()->onDestroy()*. В результаті Activity знищується. Якщо ми раптом захочемо повернутися до Activity через диспетчер задач або заново відкривши додаток, то activity буде заново перестворено шляхом виклику методів *onCreate()->onStart()-> onResume().*

4.2.3 Управління життєвим циклом Activity

Ми можемо керувати подіями життєвого циклу activity, перевизначивши відповідні методи. Наприклад:

```
public class MainActivity extends AppCompatActivity {
   private final static String TAG="MainActivity";
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(TAG, "onCreate");
}

@Override
protected void onDestroy(){
    super.onDestroy();
    Log.d(TAG, "onDestroy");
}
@Override
protected void onStop(){
    super.onStop();
    Log.d(TAG, "onStop");
}
@Override
protected void onStart(){
    super.onStart();
    Log.d(TAG, "onStart");
}
@Override
protected void onPause(){
    super.onPause();
    Log.d(TAG, "onPause");
}
@Override
protected void onResume(){
    super.onResume();
    Log.d(TAG, "onResume");
}

@Override
```

```
    protected void onRestart(){
        super.onRestart();
        Log.d(TAG, "onRestart");
    }

    @Override
    protected void onSaveInstanceState(Bundle outState){
        super.onSaveInstanceState(outState);
        Log.d(TAG, "onSaveInstanceState");
    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState){
        super.onRestoreInstanceState(savedInstanceState);
        Log.d(TAG, "onRestoreInstanceState");
    }
}
```

Для логування подій тут використовується клас *android.util.Log*.

В даному випадку обробляються всі ключові методи життєвого циклу. Обробка зводиться до виклику методу *Log.d()*, в який передається TAG – випадкове значення рядка і рядок, який виводиться в консолі logcat внизу середовища Android Studio у вікні Android Monitor. Якщо ця консоль за замовчуванням прихована, то її можна відкрити через пункт меню *View->Tool Windows->Android Monitor*. Під час запуску програми ми зможемо побачити у вікні logcat відлагоджувальну інформацію (рис. 4.4).

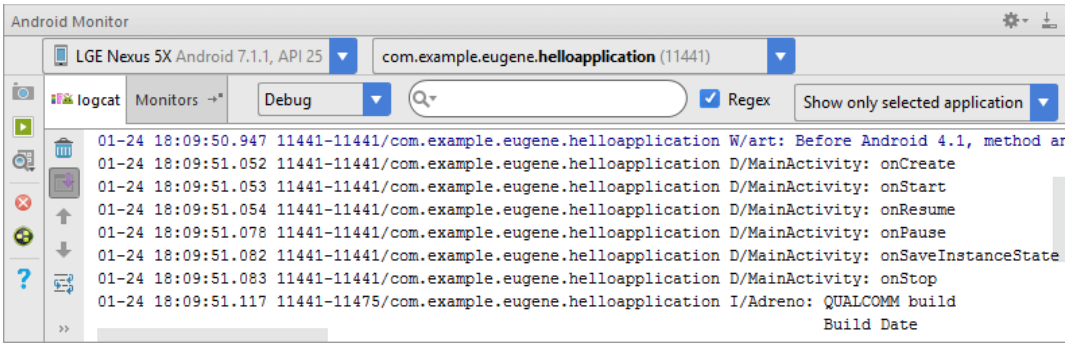


Рисунок 4.4 Відлагоджувальна інормація у вікні logcat середовища Android Studio

4.2.4 Запуск Activity

При організації взаємодії між різними об’єктами activity ключовим класом є *android.content.Intent*. Він представляє собою задачу, яку повинен виконати додаток. Нехай ми добавили в проект нову пусту activity з ім’ям *SecondActivity*. Після цього в файлі маніфесту *AndroidManifest.xml* ми зможемо знайти відповідний рядок:

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

Тут для MainActivity в елементі intent-filter визначено інтент-фільтр, в якому елемент <action android:name="android.intent.action.MAIN" представляє головну точку входу в додаток. Тобто MainActivity залишається основною і запускається додатком за замовчуванням. Для SecondActivity просто вказано, що вона є в проекті, і ніяких intent-фільтрів для неї не задано.

Для того, щоб з MainActivity запустити SecondActivity, слід викликати метод *startActivity*():

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

Як параметр в метод *startActivity*() передається об’єкт Intent, який приймає два параметри: action (дію або завдання) і data (передані в задачу дані). В якості параметра action може виступати багато можливих дій. В даному випадку використовується дія ACTION_MAIN, якf задається константою "android.intent.action.MAIN".

Тепер розглянемо реалізацію переходу від однієї Activity до іншої. Для цього в файлі *activity_main.xml* (тобто в інтерфейсі для MainActivity) визначимо кнопку:

```
 <Button
      android:id="@+id/navButton"
      android:textSize="20sp"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Перейти до SecondActivity"
```

```
android:onClick="onClick" />
```

Для запуску SecondActivity із MainActivity при натисканні на цю кнопку необхідно написати наступний код:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   public void onClick(View view) {
      Intent intent = new Intent(this, SecondActivity.class);
      startActivity(intent);
   }
}
```

4.3 Передача даних між Activity

Для передавання даних між двома Activity використовується об’єкт Intent. Через його метод *putExtra()* можна передати ключ і пов’язане з ним значення.

Наприклад, передача з поточної activity в SecondActivity стрічки "Hello World" з ключем "hello" може виглядати так:

```
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra("hello", "Hello World");
startActivity(intent);
```

Для передавання даних застосовується метод putExtra(), який дозволяє передати дані найпростіших типів – string, int, float, double, long, short, byte, char, масиви цих типів, або об’єкт інтерфейсу Serializable.

Щоб отримати відправлені дані при завантаженні SecondActivity, можна скористатися методом *get*(), в який передається ключ об’єкта:

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

Залежно від типу даних, що передаються, при отриманні ми можемо використовувати ряд методів об’єкта Bundle. Всі вони в якості параметра приймають ключ об’єкта. Основні з них:

- *get* (): універсальний метод, який повертає значення типу Object. Відповідно поле отримання дане значення необхідно перетворити до потрібного типу;
- *getString*(): повертає об’єкт типу string;
- *getInt*(): повертає значення типу int;
- *getByte*(): повертає значення типу byte;
- *getChar*(): повертає значення типу char;
- *getShort*(): повертає значення типу short;
- *getLong*(): повертає значення типу long;
- *getFloat*(): повертає значення типу float;
- *getDouble*(): повертає значення типу double;
- *getBoolean*(): повертає значення типу boolean;
- *getCharArray*(): повертає масив об’єктів char;
- *getIntArray*(): повертає масив об’єктів int;
- *getFloatArray*(): повертає масив об’єктів float;
- *getSerializable*(): повертає об’єкт інтерфейсу Serializable.

Для передачі даних складних типів використовується механізм серіалізації. Нехай в проекті є клас *Product*, який реалізує інтерфейс *Serializable*:

```
public class Product implements Serializable {
   private String name;
   private String company;
   private int price;
   public Product(String name, String company, int price){
      this.name = name;
      this.company = company;
      this.price = price;
   }
…
}
```

Щоб передати його в іншу activity слід написати код:

```
Product product=new Product(name, company, price);
Intent intent=new Intent(this, SecondActivity.class);
intent.putExtra(Product.class.getSimpleName(), product);
startActivity(intent);
```

В якості ключа тут використовується результат виклику методу *Product.class.getSimpleName()*, який по суті повертає назву класу. Щоб отримати об’єкт типу *Product* слід написати:

```
public class SecondActivity extends AppCompatActivity {
  @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      TextView textView = new TextView(this);
      textView.setTextSize(20);
      textView.setPadding(16, 16, 16, 16);

      Bundle arguments = getIntent().getExtras();
      final Product product;
      if(arguments !=null){
         product = (Product)
arguments.getSerializable(Product.class.getSimpleName());
         textView.setText("Name: " + product.getName() + "\nCompany: " +
product.getCompany() +
         "\nPrice: " + String.valueOf(product.getPrice()));
      }
      setContentView(textView);
   }
}
```

ЛЕКЦІЯ 5. ОСНОВИ ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ПРОГРАМИ

5.1 Компоненти екрану

Графічний інтерфейс користувача є ієрархією об’єктів android.view.View і android.view.ViewGroup. Кожен об’єкт ViewGroup представляє собоє контейнер, який містить і впорядковує дочірні об’єкти View. Зокрема, до контейнерів відносять такі елементи, як RelativeLayout, LinearLayout, GridLayout, ConstraintLayout і ряд інших.

Прості об’єкти View є елементами управління та інші віджети, наприклад, кнопки, текстові поля і т. д., через які користувач взаємодіє з програмою.

Більшість візуальних елементів успадковуються від класу View, такі як кнопки, текстові поля та інші, і розташовуються в пакеті android.widget.

Розмітка визначає візуальну структуру користувацького інтерфейсу. Встановити розмітку можна декількома способами:

1. Створити елементи управління програмно в коді Java;
2. Оголосити елементи інтерфейсу в XML
3. Поєднання обох способів – базові елементи розмітки визначити в XML, а решта додавати під час виконання додатку програмним способом.

Найкращим є підхід, за якого візуальний інтерфейс описується в файлах xml, а вся пов’язана з ним логіка – в класі activity. Таким чином ми досягаємо розмежування інтерфейсу і логіки додатка, їх легше розробляти і модифікувати.

5.2 Визначення інтерфейсу у файлі xml. Файли layout

У додатках під Android візуальний інтерфейс часто завантажується із спеціальних файлів xml, які зберігають розмітку. Ці файли є ресурсами розмітки. Подібний підхід нагадує створення веб-сайтів, коли інтерфейс описується в файлах html, а логіка програми – в коді javascript.

Файли розмітки графічного інтерфейсу розташовуються в проекті в каталозі *res/layout*.

При створенні розмітки в XML слід дотримуватися деяких правил: кожен файл розмітки повинен містити один кореневий елемент, який повинен представляти об’єкт View або ViewGroup.

При компіляції кожен XML-файл розмітки компілюється в ресурс View. Завантаження ресурсу розмітки здійснюється в методі *Activity.onCreate()*. Щоб встановити розмітку для поточного об'єкта activity, треба в метод *setContentView*() як параметр передати посилання на ресурс розмітки.

Для отримання посилання на ресурс в коді Java необхідно застосувати вираз R.layout.[Назва_ресурсу]. Назва ресурсу layout повинна збігатися з ім'ям файлу. Наприклад:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

Зазвичай в проекті є декілька різних ресурсів layout (рис. 5.1). Як правило, кожен окремий клас Activity використовує свій файл layout. Але для одного класу Activity можна також використовуватися декілька різних файлів layout.

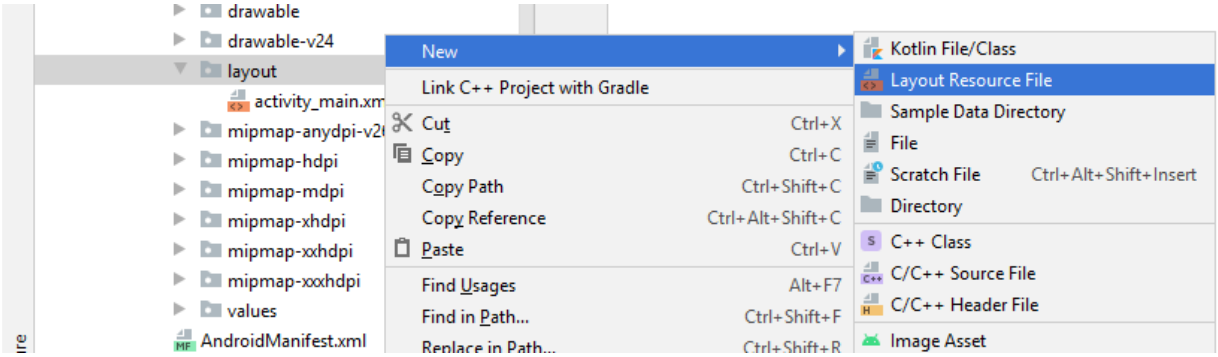


Рисунок 5.1 Додавання нового файла layout в проект

Для доступу до елементів по атрибуту id клас Activity має метод *findViewById*(). У цей метод передається ідентифікатор ресурсу у вигляді *R.id.[ідентифікатор_елементу]*. Цей метод повертає об'єкт View – об'єкт базового класу для всіх елементів, тому результат методу ще необхідно привести до відповідного типу. Далі ми можемо маніпулювати цим елементом. Наприклад, для текстового поля

```
<TextView android:id="@+id/header"
        android:text="Second Activity"
        android:textSize="26dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

можна змінити текст:

```
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // встановлюємо в якості інтерфейсу файл second_layout.xml
        setContentView(R.layout.second_layout);

        // отримуємо елемент textView
        TextView textView = (TextView) findViewById(R.id.header);
        // змінюємо текст
        textView.setText("Hello Android!");
    }
}
```

Тут важливо, що пошук елемента відбувається після того, як в методі *setContentView*() була встановлена розмітка, в якій цей візуальний елемент був визначений.

5.3 Графічні можливості Android Studio

Android Studio має потужний інструментарій, який полегшує розробку графічного інтерфейсу (рис. 5.2).

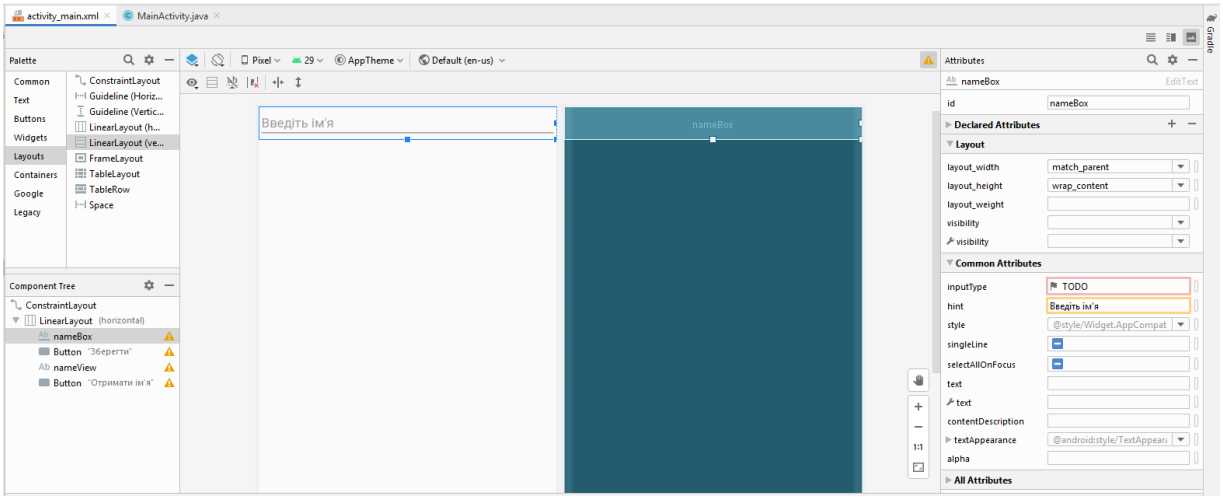


Рисунок 5.2 Частина Android Studio, призначена для розробки інтерфейсу програми

Ми можемо відкрити файл *activity_main.xml* і внизу за допомогою кнопки Design переключитися в режим дизайнера із графічним представленням інтерфейсу у вигляді ескізу смартфона.

Зліва буде знаходитися панель інструментів, із якої ми можемо переносити потрібний елемент мишкою на ескіз смартфона. При цьому всі перенесені елементи будуть автоматично додаватися в файл *activity_main.xml*. За допомогою миші ми можемо змінювати позиціонування вже доданих елементів.

Справа буде вікно Properties – панель властивостей виділеного елемента. Тут ми можемо змінити значення властивостей елемента. І знову ж таки після зміни властивостей зміниться і вміст файлу *activity_main.xml*.

Тобто при будь-яких змінах в режимі дизайнера відбуватиметься синхронізація з файлом *activity_main.xml*. Це все одно, що ми вручну змінювали б код безпосередньо в файлі *activity_main.xml*.

5.4 Різні варіанти компонування елементів інтерфейсу (Layout)

5.4.1 LinearLayout

Контейнер LinearLayout представляє об’єкт ViewGroup, який впорядковує всі дочірні елементи в одному напрямку: по горизонталі або по вертикалі. Все елементи розташовуються один за одним. Напрямок розмітки вказується за допомогою атрибута *android:orientation*.

Якщо, наприклад, орієнтація розмітки вертикальна (*android:orientation = "vertical"*), то всі елементи розташовуються в стовпчик – по одному елементу на кожен рядок. Якщо орієнтація горизонтальна (*android:orientation="horizontal"*), то елементи розташовуються в один рядок:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello"
      android:textSize="26sp" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Android"
      android:textSize="26sp" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Nougat"
      android:textSize="26sp" />
</LinearLayout>
```

LinearLayout підтримує таку властивість, як вага елемента, яка задається атрибутом *android:layout_weight*. Це властивість вказує, яку частину залишку вільного місця контейнера по відношенню до інших об’єктів займе даний елемент. Наприклад, якщо один елемент у нас буде мати для властивості *android:layout_weight* значення 2, а інший – значення 1, то в сумі вони дадуть 3, тому перший елемент буде займати 2/3 простору контейнера, а другий – 1/3.

Якщо всі елементи мають значення *android:layout_weight="1"*, то всі вони будуть рівномірно розподілені по всій площі контейнера.

5.4.2 RelativeLayout

RelativeLayout представляє об’єкт ViewGroup, який розміщує дочірні елементи відносно щодо позиції інших дочірніх елементів розмітки або щодо області самої розмітки RelativeLayout. Використовуючи відносне позиціонування, ми можемо встановити елемент по правому краю або в центрі або іншим способом, який надає даний контейнер. Для установки елемента в файлі xml ми можемо застосовувати атрибути групи *android:layout_* (*android:layout_above, android:layout_below, android:layout_centerHorizontal, android:layout_alignTop, android:layout_alignParentLeft* тощо). Наприклад:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView android:text="Left Top"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:textSize="26sp"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true" />
   <TextView android:text="Right Top"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:textSize="26sp"
      android:layout_alignParentRight="true"
      android:layout_alignParentTop="true" />
   <TextView android:text="Left Bottom"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:textSize="26sp"
      android:layout_alignParentLeft="true"
      android:layout_alignParentBottom="true" />
   <TextView android:text="Right Bottom"
      android:layout_height="wrap_content"
```

```
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
</RelativeLayout>
```

На екрані ми побачимо:

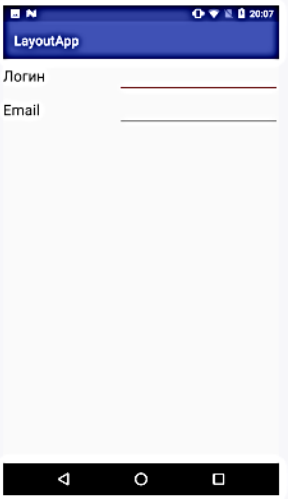
5.4.3 TableLayout

Контейнер TableLayout впорядковує дочірні елементи управління по стовпцях і рядках. Визначимо в файлі *activity_main.xml* елемент TableLayout, який буде включати два рядки і два стовпці:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TableRow>
      <TextView
         android:layout_weight="0.5"
         android:text="Логин"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
      <EditText
         android:layout_weight="1"
         android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
      <TextView
        android:layout_weight="0.5"
        android:text="Email"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
      <EditText
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </TableRow>
</TableLayout>
```

На екрані ми побачимо:



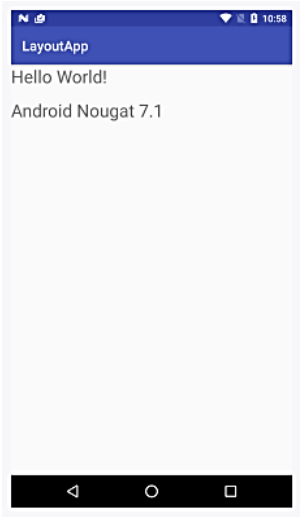
5.4.4 FrameLayout

Контейнер FrameLayout призначений для виведення на екран одного поміщеного в нього візуального елемента. Якщо ж ми помістимо декілька елементів, то вони будуть накладатися один на одного. Припустимо ми вклали в FrameLayout два елементи TextView:

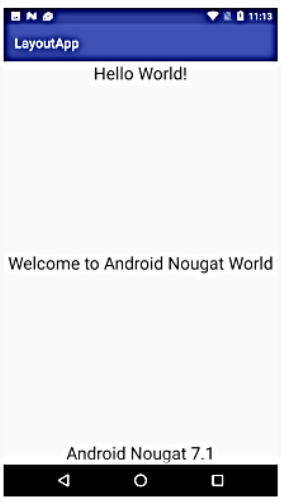
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="26sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Android Nougat 7.1"
        android:textSize="26sp"
        android:layout_marginTop="50dp"/>
</FrameLayout>
```

Тут обидва елементи позиціонуються в одне і той же місце – в лівий верхній кут контейнера FrameLayout. Щоб уникнути накладання, в даному випадку в другого TextView встановлюється відступ зверху в 50 одиниць:



Елементи управління всередині FrameLayout можуть встановлювати своє позиціонування за допомогою атрибута android: layout_gravity. При заданні значення цього атрибуту ми можемо комбінувати ряд значень, розділяючи їх вертикальною лінією: bottom | center_horizontal. В результаті можна отримати таку розмітку:



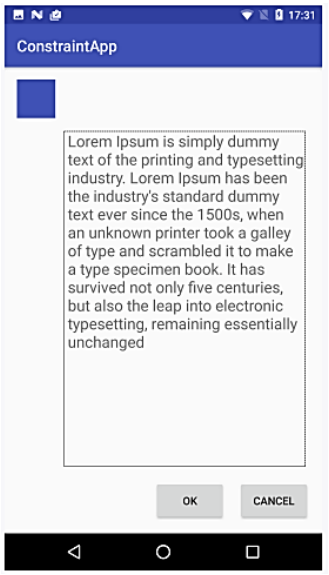
5.4.5 ConstraintLayout

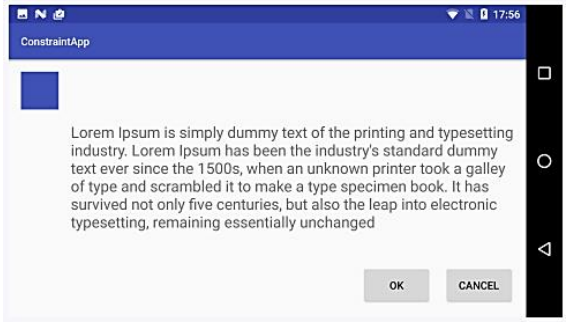
ConstraintLayout представляє собою новий тип контейнерів, який є розвитком RelativeLayout і дозволяє створювати гнучкі та масштабовані інтерфейси. Починаючи з версії Android Studio 2.3 ConstraintLayout був доданий в список стандартних компонентів і навіть є контейнером, який використовується в файлах layout за замовчуванням. Розглянемо приклад його застосування:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <ImageView
      android:text="TextView"
      android:background="#3F51B5"
      android:layout_width="50dp"
      android:layout_height="50dp"
      android:id="@+id/imageView"
      app:layout_constraintLeft_toLeftOf="parent"
      android:layout_marginLeft="16dp"
      app:layout_constraintTop_toTopOf="parent"
```

```
        android:layout_marginTop="16dp" />
    <TextView
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:textSize="20sp"
        android:id="@+id/textView"
        android:text="Lorem Ipsum is simply dummy text of the printing and typesetting
industry ... remaining essentially unchanged"
        app:layout_constraintLeft_toRightOf="@+id/imageView"
        android:layout_marginLeft="16dp"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        android:layout_marginTop="16dp"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"
        app:layout_constraintBottom_toTopOf="@+id/button2"
        android:layout_marginBottom="16dp" />
    <Button
        android:text="Cancel"
        android:layout_width="93dp"
        android:layout_height="53dp"
        android:id="@+id/button1"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="16dp" />
    <Button
        android:text="OK"
        android:layout_width="93dp"
        android:layout_height="53dp"
        android:id="@+id/button2"
        app:layout_constraintRight_toLeftOf="@+id/button1"
        android:layout_marginRight="16dp"
        app:layout_constraintBaseline_toBaselineOf="@+id/button1" />
</android.support.constraint.ConstraintLayout>
```

На екрані це виглядатиме так:





По-перше, тут елемент позиціонується відносно контейнера ConstraintLayout: від верхньої і лівої межі контейнера до відповідних меж ImageView по 16 dip.

По-друге, відносно контейнера позиціонується також кнопка з *id=button1*: від правої і нижньої межі контейнера до відповідних меж Button також по 16 dip.

По-третє, друга кнопка з *id=button2* позиціонується відносно першої кнопки: від правої межі другої кнопки до лівої межі першої кнопки (*app:layout_constraintRight_toLeftOf="@+id/button1"*) також 16 dip. І щоб обидві кнопки знаходилися на одному рівні, у них задано вирівнювання по базовій лінії: *app:layout_constraintBaseline_toBaselineOf="@+id/button1"*.

Нарешті елемент TextView із фрагментом тексту позиціонується одразу відносно як до контейнера, так і до елемента ImageView і до другої кнопки.

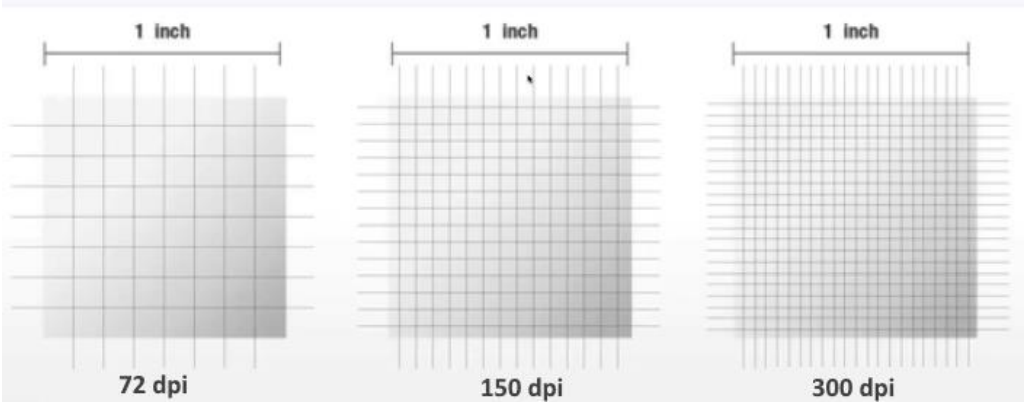
5.5 Одиниці вимірювання розміру елементів екрану

Розрізняють декілька одиниць вимірювання розміру елементів екрану:

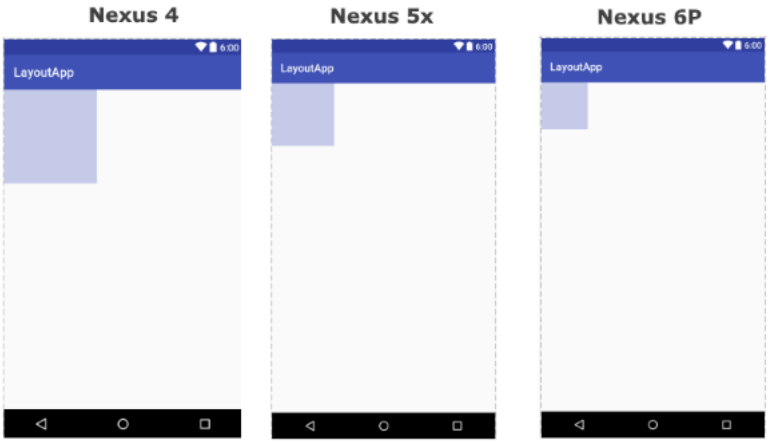
- **px**: пікселі поточного екрану. Дана одиниця вимірювання не рекомендується, оскільки реальне представлення зовнішнього вигляду екрану програми може змінюватися залежно від конкретного пристрою; кожен пристрій має індивідуально визначену кількість пікселів на один дюйм. Відповідно, кількість пікселів на екрані є різною для різних пристроїв;

- **dp**: (device-independent pixels) – пікселі, незалежні від щільності екрану. Це абстрактна одиниця вимірювання, яка базується на фізичній щільності екрану з роздільною здатністю 160 dpi (точок на дюйм). У цьому випадку 1dp=1px. Якщо ж розмір екрана більший або менший, ніж 160dpi, кількість пікселів, які застосовуються для представлення 1dp, відповідно? збільшується або зменшується. Наприклад, на екрані з 240 dpi 1dp=1,5px, а на екрані з 320dpi матимемо: 1dp=2px. Загальна формула для обчислення кількості фізичних пікселів з dp така: px=dp*(dpi/160);
- **sp**: (scale-independent pixels) – пікселі, незалежні від масштабування. Допускає налаштування розмірів користувачем. Рекомендується для роботи із шрифтами;
- **pt**: 1/72 дюйма, базується на фізичних розмірах екрану;
- **mm**: міліметри;
- **in**: дюйми.

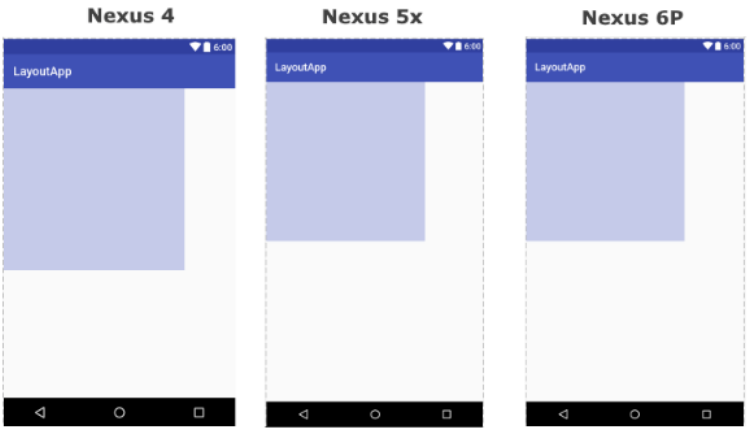
Рекомендованою одиницею для використання є dp. Це пов’язано з тим, що світ мобільних пристроїв на Android досить сильно фрагментований в контексті широкого різноманіття розширення та розмірів екрану. І чим більшою є щільність (кількість пікселів на один дюйм), тим більше пікселів буде на екрані:



Використовуючи стандартні фізичні пікселі, ми неминуче стикнемося із тою проблемою, що розміри елементів сильно варіюватимуться для різних екранів. Наприклад, для 3 пристроїв з різними характеристиками екрану Nexus 4, Nexus 5X та Nexus 6P, квадрат розмірами 300px на 300px на екрані виглядатиме так:



Тепер візьмемо квадрат із сторонами 300dp на 300dp. Як бачимо, тепер квадрат виглядає більш-менш коректно.



Для спрощення роботи всі розміри розбито на декілька груп:

- **ldpi** (low): ~120dpi;
- **mdpi** (medium): ~160dpi;
- **hdpi** (high): ~240dpi (Nexus One);
- **xhdpi** (extra-high): ~320dpi (Nexus 4);
- **xxhdpi** (extra-extra-high): ~480dpi (Nexus 5/5X, Samsung Galaxy S5);
- **xxxhdpi** (extra-extra-extra-high): ~640dpi (Nexus 6/6P, Samsung Galaxy S6).

ЛЕКЦІЯ 6. ЕЛЕМЕНТИ УПРАВЛІННЯ

6.1 Елементи управління TextView, EditView, Button, Checkbox, RadioButton. Створення обробників подій та прив'язка їх до елементів управління

6.1.1 TextView

Для простого перегляду тексту на екрані призначений елемент *TextView*. Він відображає текст без можливості редагування. Деякі його основні атрибути:

- **android:text**: встановлює текст елемента;
- **android:textSize**: встановлює висоту тексту, в якості одиниць виміру для вказівки висоти використовуються *sp*;
- **android:background**: задає фоновий колір елемента у вигляді кольору в шістнадцятковому вигляді або у вигляді колірного ресурсу;
- **android:textColor**: задає колір тексту;
- **android:textAllCaps**: при значенні *true* робить всі символи в тексті великими;
- **android:textDirection**: встановлює напрямок тексту. За замовчуванням використовується напрямок зліва направо, але за допомогою значення *rtl* можна встановити напрямок справа наліво;
- **android:textAlignment**: задає вирівнювання тексту. Може приймати наступні значення:
    - center : вирівнювання по центру;
    - textStart : на початку тексту відносно абзацу;
    - textEnd : в кінці тексту;
    - viewStart : по лівому краю (відносно екрана);
    - viewEnd : по правому краю;
- **android:fontFamily**: встановлює тип шрифту. Може приймати наступні значення:
    - monospace;
    - serif;
    - serif-monospace;
    - sans-serif;
    - sans-serif-condensed;
    - sans-serif-smallcaps;

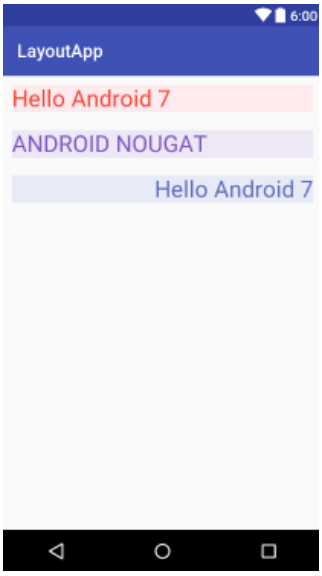
- sans-serif-light;
- casual;
- cursive.

Наприклад, визначимо три текстових поля:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:text="Hello Android 7"
      android:fontFamily="sans-serif"
      android:textSize="26sp"
      android:background="#ffebee"
      android:textColor="#f44336" />
   <TextView
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:text="Android Nougat"
      android:textAllCaps="true"
      android:textSize="26sp"
      android:background="#ede7f6"
      android:textColor="#7e57c2" />
   <TextView
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:text="Hello Android 7"
      android:textAlignment="textEnd"
```

```
        android:textSize="26sp"
        android:background="#e8eaf6"
        android:textColor="#5c6bc0" />
</LinearLayout>
```

На екрані вони відображуватимуться так:



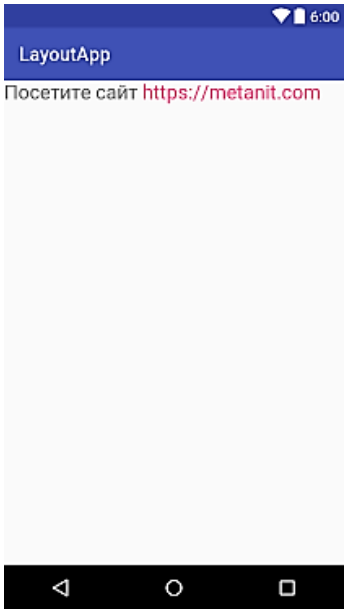
Задання елемента в коді теж не є складним. Наприклад, створимо елемент і виведемо його на екран:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout=new LinearLayout(this);
        TextView textView1=new TextView(this);
        textView1.setBackgroundColor(0xffe8eaf6); // колір фону
        textView1.setTextColor(0xff5c6bc0); // колір тексту
        textView1.setAllCaps(true); // всі букви великі
        textView1.setTextAlignment(TextView.TEXT_ALIGNMENT_CENTER); //
вирівнювання тексту по центру
        textView1.setText("Android Nougat 7"); // текстове значення
        textView1.setTypeface(Typeface.create("casual", Typeface.NORMAL)); // шрифт
        textView1.setTextSize(26); // висота тексту
```

```
        LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams
            (ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);
        layoutParams.setMargins(20,20,20,20); // зовнішні відступи
        textView1.setLayoutParams(layoutParams); // розміри
        linearLayout.addView(textView1);
        setContentView(linearLayout);
    }
}
```

Іноді необхідно вивести на екран посилання або телефон, після натискання на які відбувалася б певна дія. Для цього в *TextView* визначено атрибут **android:autoLink** :

```
<TextView android:id="@+id/display_message"
    android:text="Посетите сайт https://metanit.com"
    android:textSize="21sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoLink="web|email" />
```



Властивість **android:autoLink** може приймати декілька значень:

- none: відключає всі посилання;

- web: включає всі веб-посилання;
- email: включає посилання на електронні адреси;
- phone: включає посилання на номери телефонів;
- map: включає посилання на карту;
- all: включає всі перераховані вище посилання.

Тобто при налаштуванні android: *autoLink="web"* якщо в тексті є адреса *url*, то ця адреса буде виділятися, а при натисканні на нього буде здійснено перехід до веб-браузера, який відкриє сторінку за цією адресою. За допомогою прямої риски ми можемо об'єднувати умови, як в даному випадку: *android:autoLink="web|email*".

6.1.2 EditText

Елемент EditText є підкласом класу *TextView*. Він також представляє текстове поле, але вже з можливістю введення і редагування тексту. Таким чином, в EditText ми можемо використовувати всі ті ж можливості, що і в *TextView*.

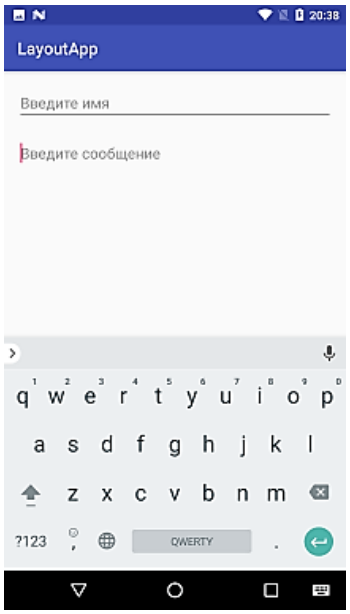
З тих атрибутів, що не розглядалися в підрозділі про *TextView*, слід зазначити атрибут *android:hint*. Він дозволяє задати текст, який буде відображатися в якості підказки, якщо елемент *EditText* порожній. Крім того, ми можемо використовувати атрибут *android:inputType*, який дозволяє задати клавіатуру для введення. Зокрема, серед його значень можна виділити наступні:

- **text:** звичайна клавіатура для введення однорядкового тексту;
- **textMultiLine:** багаторядкове текстове поле;
- **textEmailAddress:** звичайна клавіатура, на якій присутній символ @, орієнтована на введення *email*;
- **textUri:** звичайна клавіатура, на якій присутній символ /, орієнтована на введення інтернет-адрес;
- **textPassword:** клавіатура для введення пароля;
- **textCapWords:** при введенні перший введений символ слова являє велику літеру, інші – малі;
- **number:** числова клавіатура;
- **phone:** клавіатура в стилі звичайного телефону;
- **date:** клавіатура для введення дати;
- **time:** клавіатура для введення часу;
- **datetime:** клавіатура для введення дати і часу.

Використовуємо EditText на практиці:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Введите имя" />
   <EditText
      android:layout_marginTop="16dp"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:hint="Введіть повідомлення"
      android:inputType="textMultiLine"
      android:gravity="top" />
</LinearLayout>
```

Перше поле тут звичайне однорядкове, а друге – багаторядкове. Щоб в другому полі текст вирівнювався по верху, додатково встановлюється атрибут *android:gravity="top"*. В результаті отримаємо:



Однією з можливостей елемента *EditText* також є можливість обробити введені символи під час введення користувача. Для цього визначимо в файлі *activity_main.xml* наступну розмітку:

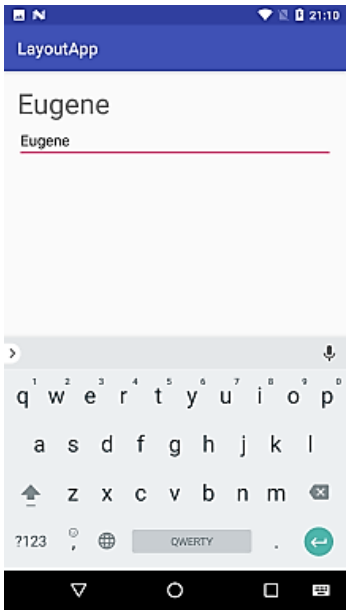
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
    <TextView
        android:id="@+id/textView"
        android:textSize="34sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Введіть ім’я"
        android:id="@+id/editText" />
</LinearLayout>
```

Передбачається, що введені в *EditText* символи тут же будуть відображатися в елементі *TextView*. Для цього також змінимо код *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      EditText editText = (EditText) findViewById(R.id.editText);
      editText.addTextChangedListener(new TextWatcher() {

         public void afterTextChanged(Editable s) {}
         public void beforeTextChanged(CharSequence s, int start,
                                       int count, int after) {
         }
         public void onTextChanged(CharSequence s, int start,
                                   int before, int count) {
            TextView textView = (TextView) findViewById(R.id.textView);
            textView.setText(s);
         }
      });
   }
}
```

За допомогою методу *addTextChangedListener*() тут до елементу *EditText* додається слухач введення тексту – об'єкт *TextWatcher*. Для його використання нам треба реалізувати три методу, але в реальності нам вистачить реалізації методу *onTextChanged*, який викликається при зміні тексту. Введений текст передається в цей метод в якості параметра *CharSequence*. У самому методі просто передаємо цей текст в елемент *TextView*.

В результаті при введенні в EditText все символи також будуть відображатися в *TextView*:



6.1.3 Button

Одним з часто використовуваних елементів є кнопки, які представлені класом *android.widget.Button*. Ключовою особливістю кнопок є можливість взаємодії з користувачем через натискання. Деякі ключові атрибути, які можна задати у кнопок:

- **text:** задає текст на кнопці;
- **textColor:** задає колір тексту на кнопці;
- **background:** задає фоновий колір кнопки;
- **textAllCaps:** при значенні *true* встановлює текст в верхньому регістрі. За замовчуванням якраз і застосовується значення true;
- **onClick:** задає обробник натиснення кнопки.

Змінимо код в *activity_main.xml* наступним чином:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">
   <TextView
      android:id="@+id/textView"
      android:textSize="34sp"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
  <EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введіть ім’я"
    android:id="@+id/editText" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click"
    android:onClick="sendMessage" />
</LinearLayout>
```

За допомогою атрибута **android:onClick** можна задати метод в коді Java, який буде обробляти натискання кнопки. Так, в наведеному вище прикладі це метод *sendMessage*. Тепер перейдемо до коду *MainActivity* і пропишемо в ньому такий метод:

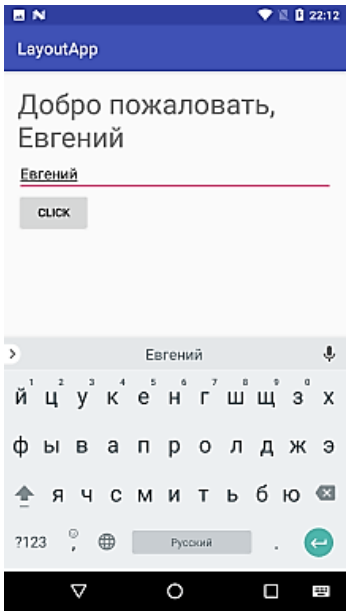
```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  // Обробка натискання на кнопку
  public void sendMessage(View view) {
    TextView textView=(TextView) findViewById(R.id.textView);
    EditText editText=(EditText) findViewById(R.id.editText);
    textView.setText("Ласкаво просимо, "+editText.getText());
  }
}
```

При створенні методу для обробки натискання слід враховувати наступні моменти:

- метод повинен оголошуватися з модифікатором *public*;
- метод повинен повертати значення *void*;

- як параметр метод повинен приймати об’єкт *View*, який представляє натиснуту кнопку.

В даному випадку після натискання на кнопку в *TextView* виводиться текст з *EditText*:



Аналогічний приклад повністю в коді MainActivity:

```
public class MainActivity extends AppCompatActivity {
   EditText editText;
   TextView textView;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      LinearLayout linearLayout=new LinearLayout(this);
      linearLayout.setOrientation(LinearLayout.VERTICAL);
      textView=new TextView(this);
      textView.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
      ));
      linearLayout.addView(textView);
```

```
        editText=new EditText(this);
        editText.setHint("Введіть ім’я");
        editText.setLayoutParams(new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(editText);

        Button button=new Button(this);
        button.setText("CLICK");
        button.setLayoutParams(new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(button);

        button.setOnClickListener(new View.OnClickListener() {
           public void onClick(View v) {
              textView.setText("Ласкаво просимо, "+editText.getText());
           }
        });
        setContentView(linearLayout);
    }
}
```

При програмному створенні кнопки ми можемо визначити у неї слухач натискання *View.OnClickListener* і за допомогою його методу *onClick* також обробити натискання:

```
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Обробка натискання
    }
});
```

6.1.4 Checkbox

Елементами *Checkbox* є прапорці, які можуть перебувати в відміченому і невіміченому стані. Прапорці дозволяють виробляти множинний вибір з кількох значень. Визначимо в файлі розмітки *activity_main.xml* кілька прапорців:

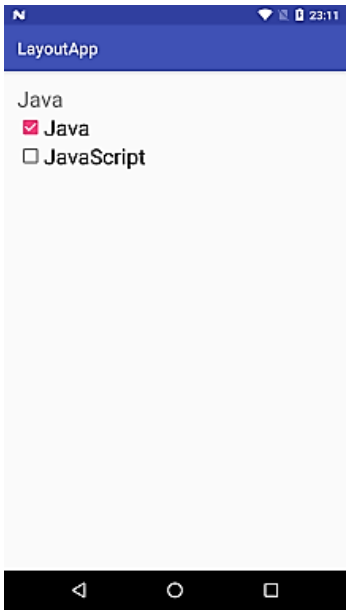
```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">
   <TextView android:id="@+id/selection"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="26sp" />
   <CheckBox android:id="@+id/java"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Java"
      android:textSize="26sp"
      android:onClick="onCheckboxClicked"/>
   <CheckBox android:id="@+id/javascript"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="JavaScript"
      android:textSize="26sp"
      android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Атрибут **android:onClick**, як і в випадку з простими кнопками, дозволяє задати обробник натиснення на прапорець. Визначимо обробник натиснення в коді *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onCheckboxClicked(View view) {
        CheckBox language=(CheckBox) view; // отримуємо прапорець
        boolean checked=language.isChecked(); // перевіряємо, чи він відмічений
        TextView selection=(TextView) findViewById(R.id.selection);
        // Перевіряємо, який саме прапорець відмічено
        switch(view.getId()) {
            case R.id.java:
                if (checked){
                    selection.setText("Java");
                }
                break;
            case R.id.javascript:
                if (checked)
                    selection.setText("JavaScript");
                break;
        }
    }
}
```

В якості параметра в обробник натискання *onCheckboxClicked* передається натиснутий прапорець. За допомогою методу *isChecked*() можна дізнатися, чи виділений прапорець – в цьому випадку метод повертає *true*. За допомогою конструкції *switch ... case* можна отримати *id* натиснутого прапорця і виконати відповідні дії.



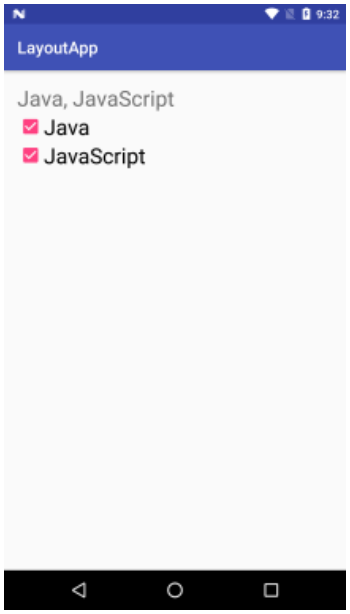
Якщо нам просто треба взяти текст з обраного прапорця, то не обов'язково в даному випадку використовувати конструкцію *switch*, бо ми можемо скоротити весь код наступним чином:

```
public void onCheckboxClicked(View view) {
   CheckBox language=(CheckBox) view;
   TextView selection=(TextView) findViewById(R.id.selection);
   if(language.isChecked())
      selection.setText(language.getText());
}
```

Але в даному випадку у нас є проблема: в текстовому полі відображається тільки один виділений елемент. Змінимо код *MainActivity*, щоб показати обидва виділених елементи:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   public void onCheckboxClicked(View view) {
      // отримуємо прапорці
```

```
        CheckBox java=(CheckBox) findViewById(R.id.java);
        CheckBox javascript=(CheckBox) findViewById(R.id.javascript);
        String selectedItems="";
        if(java.isChecked())
            selectedItems+=java.getText() + ", ";
        if(javascript.isChecked())
            selectedItems+=javascript.getText();
        TextView selection=(TextView) findViewById(R.id.selection);
        selection.setText(selectedItems);
    }
}
```



За допомогою слухача **OnCheckedChangeListener** можна відстежувати зміни прапорця. Цей слухач спрацьовує, коли ми встановлюємо або прибираємо позначку на прапорці. Наприклад, визначимо наступний checkbox:

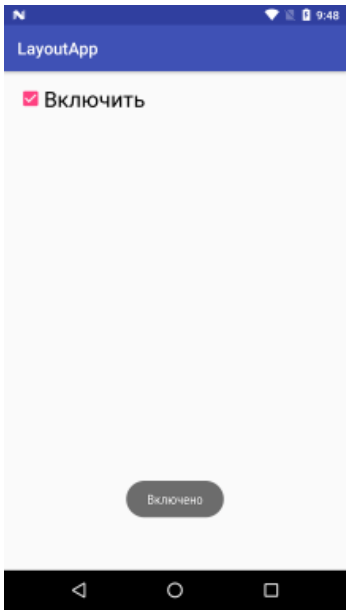
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
```

```
    <CheckBox android:id="@+id/enabled"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="Ввімкнути"
        android:textSize="26sp" />
</LinearLayout>
```

Програмний код:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        CheckBox enableBox = (CheckBox) findViewById(R.id.enabled);
        enableBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                if (isChecked) {
                    Toast.makeText(getApplicationContext(), "Ввімкнено",
Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Вимкнено",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

Слухач **OnCheckedChangeListener** визначено в базовому класі *CompoundButton* і він визначає один метод – *onCheckedChanged*. Перший параметр цього методу *buttonView* – сам змінений прапорець *CheckBox*. А другий параметр *isChecked* вказує, чи відзначений прапорець. При зміні стану прапорця, буде виводитися відповідне повідомлення:



6.1.5 RadioButton

Подібну до прапорців функціональність надають перемикачі, які представлені класом *RadioButton*. Але на відміну від прапорців одноразово в групі перемикачів ми можемо вибрати тільки один перемикач. Щоб створити список перемикачів для вибору, спочатку треба створити об’єкт *RadioGroup* , який буде включати в себе всі перемикачі:

```
<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:orientation="vertical" >
    <TextView android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="26sp"
        />
    <RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/radios"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <RadioButton android:id="@+id/java"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Java"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/javascript"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="JavaScript"
        android:onClick="onRadioButtonClicked"/>
  </RadioGroup>
</LinearLayout>
```

Оскільки клас *RadioGroup* є похідним від *LinearLayout*, то ми також можемо поставити вертикальну або горизонтальну орієнтацію списку, при тому включивши в нього не тільки власне перемикачі, а й інші об’єкти, наприклад, кнопку або *TextView*.

У класі *MainActivity* визначимо обробку вибору перемикачів:

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void onRadioButtonClicked(View view) {
    // якщо перемикач ввімкнено
    boolean checked = ((RadioButton) view).isChecked();
    TextView selection = (TextView) findViewById(R.id.selection);
    // Отримуємо ввімкнений перемикач
    switch(view.getId()) {
      case R.id.java:
        if (checked){
          selection.setText("Выбран Java");
        }
        break;
      case R.id.javascript:
```

```
            if (checked){
                selection.setText("Выбран JavaScript");
            }
            break;
        }
    }
}
```

Окрім обробки натискання на кожен окремий перемикач, ми можемо в цілому визначити для всього *RadioGroup* з його перемикачами слухач *OnCheckedChangeListener* і обробляти в ньому натискання. Для цього приберемо з розмітки у перемикачів атрибути **android:onClick**, а у елемента *RadioGroup* визначимо *id*:

```
<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:orientation="vertical" >
    <TextView android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
    android:textSize="26sp"
    />
  <RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/radios"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/java"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Java"/>
    <RadioButton android:id="@+id/javascript"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="JavaScript"/>
  </RadioGroup>
</LinearLayout>
```

Далі в коді *MainActivity* призначимо об’єкту RadioGroup слухача:

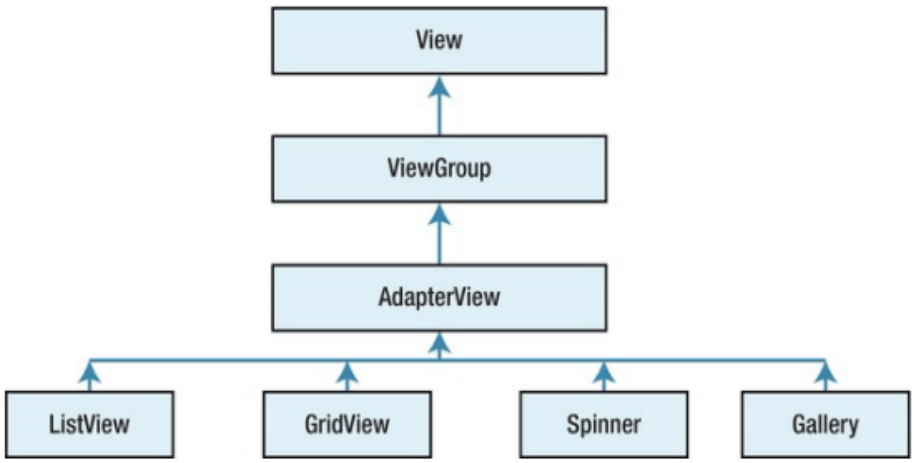
```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    RadioGroup radGrp=(RadioGroup)findViewById(R.id.radios);
    // обробка зміни стану премикача
    radGrp.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(RadioGroup arg0, int id) {
        TextView selection=(TextView) findViewById(R.id.selection);
        switch(id) {
          case R.id.java:
            selection.setText("Выбран Java");
            break;
          case R.id.javascript:
```

```
                selection.setText("Выбран JavaScript");
                break;
            default:
                break;
        }
    });
  }
}
```

Слухач *RadioGroup.OnCheckedChangeListener* визначає метод *onCheckedChanged*(), в який передається об'єкт *RadioGroup* і *id* виділеного перемикача. Далі також ми можемо перевірити *id* і виконати певну обробку.

6.2 Адаптери та списки

Android представляє широку палітру елементів, які представляють списки. Всі вони є спадкоємцями класу *android.widget.AdapterView*. Це такі віджети як *ListView, GridView, Spinner*. Вони можуть виступати контейнерами для інших елементів управління:



При роботі зі списками ми маємо справу з трьома компонентами. По-перше, це самі елементи списків (*ListView, GridView*), які відображають дані. По-друге, це джерело даних – масив, об'єкт *ArrayList*, база даних і т. д., в якому знаходяться самі відображаються дані. І по-третє, це адаптери – спеціальні компоненти, які пов'язують джерело даних з елементом списку.

Розглянемо зв’язок елемента *ListView* з джерелом даних за допомогою одного з таких адаптерів – класу *ArrayAdapter*.

6.2.1 ArrayAdapter

Клас ArrayAdapter є простим адаптером, який пов’язує масив даних з набором елементів *TextView*, з яких, наприклад, може складатися *ListView*. Тобто в даному випадку джерелом даних виступає масив об’єктів. *ArrayAdapter* викликає у кожного об’єкта метод *toString*()і отриманий рядок встановлює в елемент *TextView*. Подивимося на прикладі. Отже, розмітка програми може виглядати так:

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

Тут також визначено елемент *ListView*, який буде виводити список об’єктів. Тепер перейдемо до коду activity і пов’яжемо *ListView* через *ArrayAdapter* з деякими даними:

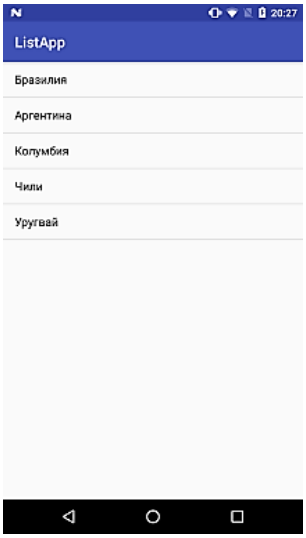
```
public class MainActivity extends AppCompatActivity {
   // набор даних, які ми пов’яжемо із списком
   String[] countries={"Бразилия", "Аргентина", "Колумбия", "Чили", "Уругвай"};
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      // отримуємо ListView
      ListView countriesList=(ListView) findViewById(R.id.countriesList);
      // створюємо адаптер
      ArrayAdapter<String> adapter=new ArrayAdapter(this,
```

```
            android.R.layout.simple_list_item_1, countries)
        // задаємо адаптер для списку
        countriesList.setAdapter(adapter);
    }
}
```

Тут спочатку отримуємо по *id* елемент *ListView* і потім створюємо для нього адаптер. Для створення адаптера використовувався наступний конструктор *ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, countries)*, де:

- *this* : поточний об'єкт *activity*;
- *android.R.layout.simple_list_item_1* : файл розмітки списку, який фреймворк являє за замовчуванням. Він знаходиться в папці Android SDK *platforms/[android-номер_версії]/data/res/layout*. Якщо нас не задовольняє стандартна розмітка списку, ми можемо створити свою і потім в коді змінити *id* на *id* потрібної нам розмітки
- *countries* : масив даних. Тут необов'язково вказувати саме масив, це може бути список *ArrayList<T>*.

В кінці необхідно встановити для *ListView* адаптер за допомогою методу *setAdapter()*. У підсумку ми отримаємо наступне відображення:



6.2.2 Ресурс string-array і ListView

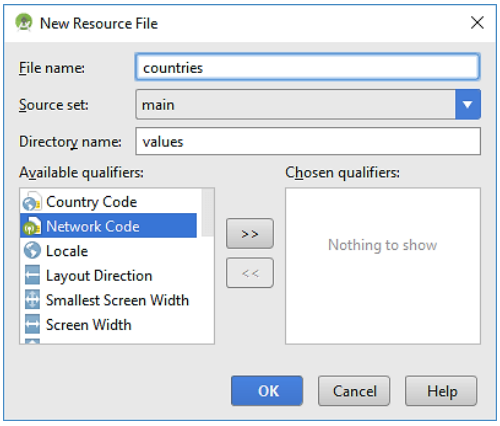
У попередньому підпункті було розглянуто, як виводити масив рядків (стрічок) за допомогою *ArrayAdapter* в *ListView*. При цьому масив рядків визначався програмно в коді *java*. Однак подібний масив рядків набагато зручніше було б зберігати в файлі *xml* у вигляді ресурсу.

Ресурси типу масивів рядків представляють елемент типу *string*-array. Вони можуть знаходиться в каталозі *res/values* в *xml*-файлі з довільним ім'ям. Визначення масивів рядків мають наступний синтаксис:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="ім'я_масиве_стрічок">
      <item>елемент</item>
   </string-array>
</resources>
```

Масив рядків задається за допомогою елемента <*string-array*>, атрибут *name* якого може мати довільне значення, за яким потім будуть посилатися на цей масив. Всі елементи масиву представляють набір значень <*item*>.

Наприклад, додамо в папку *res/values* новий файл. Для цього натиснемо правою кнопкою миші на даний каталог і меню виберемо пункт *New -> Value resource file*. У вікні назвемо файл як countries:



Після додавання файлу в папку *res/values* змінимо його вміст на такий:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="countries">
      <item>Бразилия</item>
      <item>Аргентина</item>
      <item>Колумбия</item>
      <item>Чили</item>
      <item>Уругвай</item>
```

```
    </string-array>
</resources>
```

В файлі розмітки *activity_main.xml* залишається визначення елемента *ListView*:

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

Тепер зв’яжемо ресурс і *ListView* в коді *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // отримуємо ListView
        ListView countriesList = (ListView) findViewById(R.id.countriesList);
        // отримуємо ресурс
        String[] countries = getResources().getStringArray(R.array.countries);
        // створюємо адаптер
        ArrayAdapter<String> adapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_1, countries);
        // задаємо адаптер для списку
        countriesList.setAdapter(adapter);
    }
}
```

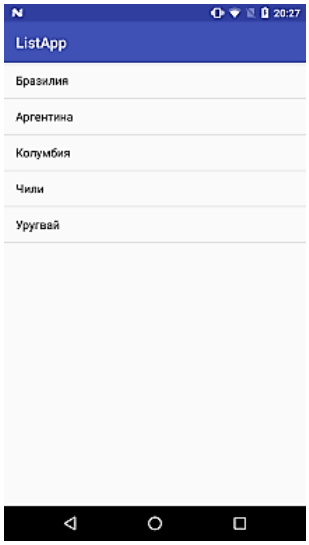
Для отримання ресурсу в коді *java* застосовується вираз *R.array.назва_ресурсу*. Але нам необов’язково додавати список рядків в *ListView* програмно. У цього елемента є атрибут *entries*, який в якості значення може приймати ресурс *string-array*:

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

В цьому випадку код *MainActivity* ми можемо скоротити до стандартного:

```
public class MainActivity extends AppCompatActivity {

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

А результат буде таким же самим:

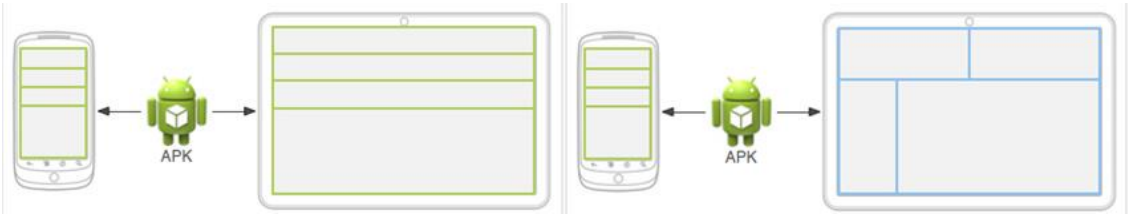


ЛЕКЦІЯ 7. РЕСУРСИ ПРОЕКТУ

7.1 Поняття ресурсу. Типи ресурсів проекту

При розробці мобільних додатків прийнято підхід, який полягає у відокремленні ресурсів, які використовує додаток, від коду. До ресурсів можуть належати: зображення, текстові стрічки, кольори, компонування елементів інтерфейсу користувача тощо. Відокремлення ресурсів від коду дозволяє використовувати альтернативні ресурси для різних конфігурацій пристроїв: мова, розширення екрану і т. д. Для забезпечення сумісності з різними конфігураціями, ресурси необхідно згрупувати в директорії за типом ресурсів і конфігурації пристрою, отримані директорії поміщуються в папку *res*/ проекту.

Для будь-якого типу ресурсів можна визначити дві групи (рис. 7.1).



а) б)

Рисунок 7.1 Варіант компонування інтерфейсу користувача, параметри якого зберігаються в ресурсах: а) компонування по замовчуванню; б) специфічне компонування для певного пристрою

Перша визначає ресурси, які будуть використовуватися незалежно від конфігурації пристрою або в тому випадку, коли під конфігурацію немає відповідних альтернативних ресурсів (ресурси за замовчуванням). Друга група визначає ресурси, які призначені для певної конфігурації фізичного пристрою (альтернативні ресурси). Такі ресурси розміщуються в директорії із назвою, що позначає цю конфігурацію (рис. 7.2).

Для різних типів ресурсів, визначених у проекті, в каталозі *res/* створюються свої підкаталоги. Файли ресурсів не можна розміщувати в папку *res/* безпосередньо, оскільки буде отримано помилку компіляції. Підтримувані підкаталоги:

- *animator*/: xml-файли, що визначають анімацію;
- *anim*/: xml-файли, що визначають tween-анімацію;

- *color*/: xml-файли, що визначають кольори;
- drawable/: графічні файли (.png, .jpg, .gif);
- *mipmap*/: графічні файли, які використовуються для іконок додатку під різні розширення екрану;
- *layout*/: xml-файли, що визначають інтерфейс користувача програми;
- *menu*/: xml-файли, що визначають меню програми;
- *raw*/: файли, які зберігаються в бінарному («сирому») вигляді;
- *values*/: xml-файли, які містять різні використовувані в додатку значення, наприклад, ресурси стрічок;
- *xml*/: довільні xml-файли.

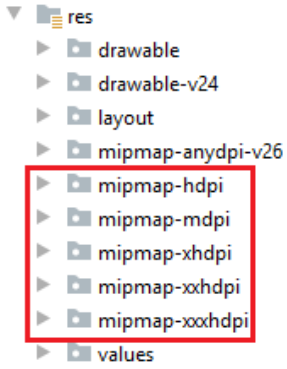


Рисунок 7.2 Можливий варіант розміщення ресурсів проекту, специфікованих для різних розширень екрану фізичного пристрою

Загалом в проекті можна використовувати такі типи ресурсів (табл. 7.1):

Таблиця 7.1

Типи ресурсів Android-проекту

Ресурс	Каталог проекту	Файл	Елемент у файлі
1	2	3	4
Стрічки	/res/values/	strings.xml	<string>
Plurals	/res/values/	strings.xml	<plurals>
Масиви стрічок	/res/values/	strings.xml або arrays.xml	<string-array>

Продовження Таблиці 7.1

Ресурс	Каталог проекту	Файл	Елемент у файлі
1	**2**	**3**	**4**
Логічні значення	/res/values/	bools.xml	<bool>
Кольори	/res/values/	colors.xml	<color>
Списки кольорів	/res/color/	довільне найменування	<selector>
1	**2**	**3**	**4**
Розміри (Dimensions)	/res/values/	dimens.xml	<dimen>
Ідентифікатори ID	/res/values/	ids.xml	<item>
Цілі числа	/res/values/	integers.xml	<integer>
Масиви цілих чисел	/res/values/	integers.xml	<integer-array>
Графічні файли	/res/drawable/	файли із розширенням jpg та png	
Tween-анімація	/res/anim/	файл xml із довільним найменуванням	<set>, <alpha>, <rotate>, <scale>, <translate>
Покадрова анімація	/res/drawable/	файл xml із довільним найменуванням	<animation-list>
Анімація властивостей	/res/animator/	файл xml із довільним найменуванням	<set>, <objectAnimator>, <valueAnimator>
Меню	/res/menu/	файл xml із довільним найменуванням	<menu>
XML-файли	/res/xml/	файл xml із довільним найменуванням	
Бінарні ресурси	/res/raw/	файли мультимедіа (mp3, mp4), текстові та інші файли	
Розмітка графічного інтерфейсу	/res/layout/	файл xml із довільним найменуванням	
Стилі та теми	/res/values/	styles.xml, themes.xml	<style>

По замовчуванню в каталозі *res/* є підкаталоги не для всіх типів ресурсів, які можна використовувати в додатку, проте при необхідності ми можемо додати в папку *res/* потрібний каталог, а в нього потім помістити відповідний ресурс. Коли відбувається компіляція проекту відомості про всі ресурси додаються в спеціальний файл *R.java* проекту. У цьому файлі зберігаються всі оголошення ресурсів у вигляді числових констант. Наприклад, у проекті за замовчуванням є ресурс розмітки інтерфейсу – файл *activity_main.xml* в папці *res*/*layout*. Для цього ресурсу в класі *R* буде створено приблизно таку константу:

```
public final class R {
.....
public static final class layout {
    public static final int activity_main=0x7f030001;
  }
```

7.2 Застосування ресурсів під час розробки додатків

Існує два способи доступу до ресурсів: в файлі вихідного коду і в файлі xml: в коді програми та в тексті xml-файлів.

7.2.1 Посилання на ресурси в коді програми

Тип ресурсу в даному випадку посилається на один із просторів (внутрішніх класів), визначених у файлі R.java:

- *R.drawable (йому відповідає тип в xml drawable);*
- *R.id (id);*
- *R.layout (layout);*
- *R.string (string);*
- *R.attr (attr);*
- *R.plural (plurals);*
- *R.array (string-array).*

Наприклад, для встановлення ресурсу *activity_main.xml* в якості графічного інтерфейсу в коді методу *onCreate*() класу *MainActivity* є такий рядок:

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

З допомогою виразу *R.layout.activity_main* ми посилаємося на ресурс *activity_main.xml*, де *layout* - тип ресурсу, а *activity*_main – ім’я ресурсу.

Подібним чином ми можемо посилатися на інші ресурси. Наприклад, у файлі *res/values/strings.xml* визначено ресурс *app_name*:

```
<resources>
   <string name="app_name">ViewsApplication</string>
</resources>
```

Щоб отримати посилання на даний ресурс в програмному коді ми можемо використовувати вираз *R.string.app_name.*

7.2.2 Посилання на ресурси у файлі xml

Посилання на ресурси в файлах xml мають формат: *@[ім'я_пакета:]тип_ресурсу/ім'я_ресурсу*. Тут:

- *ім'я_пакета* – імєя пакета, в якому знаходиться ресурс (вказувати необов'язково, якщо ресурс знаходиться в тому ж пакеті);
- *тип_ресурсу* – представляє підклас, визначений у класі R для типу ресурсу;
- *ім'я_ресурсу* – ім'я файлу ресурсу без розширення або значення атрибуту *android:name* в XML-елементі (для простих значень).

Наприклад, ми хочемо вивести в елементі інтерфейсу *TextView* рядок, який визначено у вигляді ресурсу в файлі *strings.xml*:

```
<TextView
    android:id="@+id/welcome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/app_name" />
```

В даному випадку властивість *text* елементу *TextView* в якості значення буде отримувати значення стрічкового ресурсу *app_name*.

7.2.3 Метод getResources()

Для отримання ресурсів в класі *Activity* ми можемо використовувати метод *getResources()*, який повертає об'єкт *android.content.res.Resources*. Але щоб отримати сам ресурс, слід в отриманого об'єкта *Resources* викликати один із наступних методів:

- *getString*(): повертає стрічку з файлу strings.xml по числовому ідентифікатору;
- *getDimension*(): повертає числове значення - ресурс dimension;

- *getDrawable*(): повертає графічний файл;
- *getBoolean*(): повертає значення boolean.

Наведемо простий приклад роботи з ресурсами із застосуванням методів об’єкта *Resources*:

```
public class MainActivity extends AppCompatActivity {
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // отримання ресурсів із файлу values/strings.xml
      String app_name = getResources().getString(R.string.app_name);

      TextView textView = new TextView(this);
      textView.setText(app_name);
      setContentView(textView);
   }
}
```

Тут в коді методу *onCreate()* використовуючи метод *getResources*().*getString()* отримуємо найменування проекту із файлу *strings.xml*, яке надалі використовуємо для встановлення напису графічного елемента *textView*. При запуску програми ми побачимо, наприклад, таке:

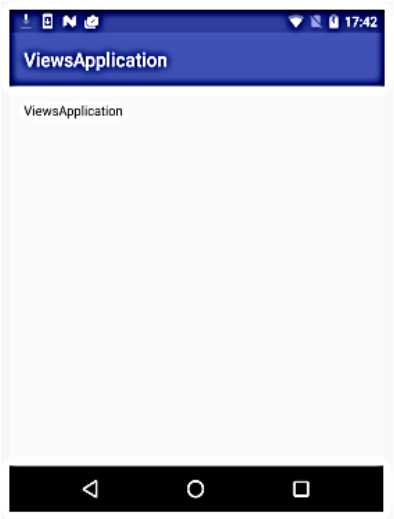


Рисунок 7.3 Результат роботи із текстовим ресурсом
із застосуванням методів об’єкта *Resources*

7.3 Робота із ресурсами основних типів

7.3.1 Стрічкові ресурси

Стрічкові ресурси – один з найважливіших компонентів програми. Ми використовуємо їх при виведенні назви програми, різного тексту, наприклад, тексту кнопок і т. д. XML-файли, що представляють собою стрічкові ресурси, знаходяться в проекті в папці *res/values*. За замовчуванням ресурси даного типу знаходяться в файлі *strings.xml*, який може виглядати наступним чином:

```
<resources>
   <string name="app_name">ViewsApplication</string>
   <string name="ButtonOk_value">Вхід</string>
   <string name="message">Hello Android!</string>
</resources>
```

Тобто, кожен окремий ресурс визначається за допомогою елемента *string*, а його атрибут *name* містить назву ресурсу. В коді додатку ми можемо посилатися на ці ресурси так: R.string.app_name

ОС Android сама співставить дані числові ідентифікатори з відповідними ресурсами рядків. Наприклад в коді Java:

```
public class MainActivity extends AppCompatActivity {
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      TextView textView=(TextView) findViewById(R.id.welcome);
      // отримуємо ресурс
      String message=getResources().getString(R.string.message);
      // встановлюємо текст елемента textView
      textView.setText(message);
   }
}
```

А в xml-файлі можна отримати доступ до стрічкового ресурсу message так:

```
<TextView
      android:id="@+id/welcome"
      android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/message" />
```

7.3.2 Ресурси типу dimension

Оголошення ресурсів даного типу повинне знаходитися в папці res /values в файлі з будь-яким довільним ім'ям. Загальний синтаксис:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="ім'я_ресурса">використовуваний_розмір</dimen>
</resources>
```

Тег <*dimen*> позначає ресурс і включає деяке значення розміру в одній з прийнятих одиниць виміру (dp, sp, pt, px, mm, in). Наприклад, якщо додати в Android Studio в папку *res/values* новий файл *dimens.xml* такого вмісту:

```
<resources>
  <dimen name="activity_horizontal_margin">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
  <dimen name="text_size">16sp</dimen>
</resources>
```

то ми визначимо два ресурси для відступів *activity_horizontal_margin* і *activity_vertical_margin*, які зберігають значення 16dp, і ресурс *text_size*, який зберігає висоти шрифту в 16sp.

В файлі *activity_main.xml* їх можна використати так:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">
  <TextView
    android:textSize="@dimen/text_size"
    android:layout_width="wrap_content"
```

```
    android:text="Hello Android Nougat!" />
</RelativeLayout>
```

Ресурси типу *dimension* використовуються для таких атрибутів візуальних елементів, які в якості значення вимагають вказання конкретного числа.

Для отримання ресурсів в коді Java застосовується метод *getDimension*() класу *Resources*. Наприклад:

```
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      float textSize = getResources().getDimension(R.dimen.text_size);
      int leftPadding =
(int)getResources().getDimension(R.dimen.activity_horizontal_margin);
      int topPadding=(int)getResources().getDimension(R.dimen.activity_vertical_margin);

      TextView textView=new TextView(this);
      textView.setText("Hello world!");
      textView.setTextSize(textSize);
      textView.setPadding(leftPadding, topPadding, leftPadding, topPadding);
      setContentView(textView);
   }
}
```

7.3.3 Ресурси типу color

Ресурси типу *color* повинні зберігатися в файлах каталогу *res/values* і так само як і ресурси стрічок, повинні бути вкладеними в тег <*resources*>. За замовчуванням при створенні найпростішого проекту в папку *res/values* додається файл *colors.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="colorPrimary">#3F51B5</color>
   <color name="colorPrimaryDark">#303F9F</color>
```

```
    <color name="colorAccent">#FF4081</color>
</resources>
```

Колір задається за допомогою елемента *<color>*. Його атрибут *name* задає назву кольору, яка буде використовуватися в додатку, а шістнадцяткове число – значення кольору. Для задання колірних ресурсів можна використовувати такі формати:

1) #RGB (#F00 – 12-бітне значення);
2) #ARGB (#8F00 – 12-бітне значення із додаванням альфа-каналу);
3) #RRGGBB (#FF00FF – 24-бітне значення);
4) #AARRGGBB (#80FF00FF – 24-бітне значення із додаванням альфа-каналу).

Приклад застосування колірних ресурсів у файлі *activity_main.xml:*

```
<TextView
    android:textSize="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/colorPrimary"
    android:background="@color/textViewColor"
    android:text="Hello world!" />
```

Тут за допомогою атрибута *android:textColor* встановлюється колір тексту в *TextView*, а атрибут *android:background* встановлює фон *TextView*. Для отримання значення кольору після "*@color /*" вказується ім’я ресурсу.

В коді *MainActivity* колірні ресурси можна застосовувати так:

```
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int textColor = ContextCompat.getColor(this, R.color.colorPrimary);
        int backgroundColor = ContextCompat.getColor(this, R.color.textViewColor);

        TextView textView = new TextView(this);
        textView.setText("Hello world!");
        textView.setTextSize(20);
        textView.setPadding(16, 16, 16, 16);
        textView.setTextColor(textColor);
        textView.setBackgroundColor(backgroundColor);
```

```
        setContentView(textView);
    }
}
```

Тут застосовується метод *ContextCompat*.getColor(), який в якості першого параметра приймає поточний об’єкт *Activity*, а в якості другого – ідентифікатор колірного ресурсу.

7.3.4 Ресурси зображень

Android підтримує такі формати файлів: *.png* (бажаний), *.jpg* (допустимий), *.gif* (небажаний). Для графічних файлів в проекті за замовчуванням створюється папка *res/drawable*. При додаванні графічних файлів в цю папку для кожного з них Android створює ресурс *Drawable*. Після цього ми можемо звернутися до ресурсу в такий спосіб в коді Java:

R.drawable.ім’я_файла
або в коді Java:
@[ім’я_пакету:]drawable/ім’я_файла

При додаванні графічних файлів у проект варто враховувати, що кожен такий файл буде збільшувати розмір проекту. Крім того, великі зображення негативно впливають на продуктивність. Тому слід використовувати невеликі і оптимізовані (стиснуті) графічні файли.

Для роботи із зображеннями в Android можна використовувати різні елементи, але безпосередньо для виведення зображень призначений ImageView:

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

Тут для відображення файлу в *ImageView* встановлюється атрибут *android:src*. В його значенні вказується ім’я графічного ресурсу, яке повинно збігатися з ім’ям файлу без розширення. Якби ми створювали *ImageView* в коді Java із відповідним застосуванням ресурсу, то activity могла б виглядати так:

```
…
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      ImageView imageView=new ImageView(this);
      // застосовуємо ресурс
      imageView.setImageResource(R.drawable.flowers);
      //setContentView(R.layout.activity_main);
      setContentView(imageView);
   }
}
```

ЛЕКЦІЯ 8. ОФОРМЛЕННЯ ІНТЕРФЕЙСУ ПРОГРАМИ

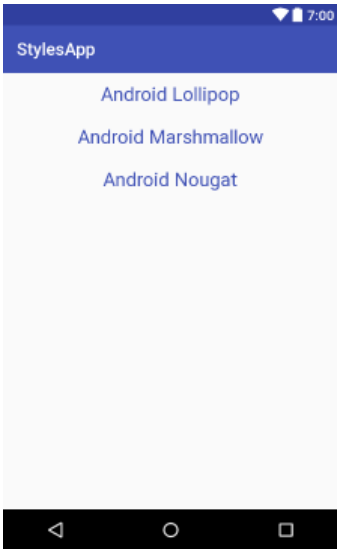
8.1 Стилі

Ми можемо налаштувати елемент за допомогою різних атрибутів, які задають висоту, ширину, колір фону, тексту і так далі. Але якщо у нас декілька елементів використовують одні і ті ж налаштування, то ми можемо об’єднати ці налаштування в стилі. Наприклад, нехай у нас є кілька елементів *TextView*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView
      android:layout_width="match_parent"
      android:layout_height="50dp"
      android:gravity="center"
      android:textSize="22sp"
      android:textColor="#3f51b5"
      android:text="Android Lollipop"/>
   <TextView
      android:layout_width="match_parent"
      android:layout_height="50dp"
      android:gravity="center"
      android:textSize="22sp"
      android:textColor="#3f51b5"
      android:text="Android Marshmallow"/>
   <TextView
      android:layout_width="match_parent"
      android:layout_height="50dp"
      android:gravity="center"
      android:textSize="22sp"
      android:textColor="#3f51b5"
```

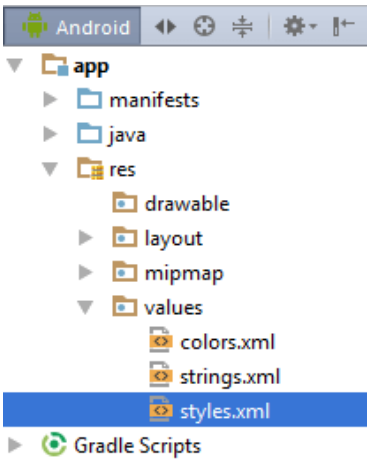
```
    android:text="Android Nougat"/>
</LinearLayout>
```

які на екрані користувачу дають наступну розмітку:



Всі ці *TextView* мають однаковий набір властивостей і, наприклад, якщо нам захочеться змінити колір тексту, то доведеться міняти його у всіх трьох *TextView*. Більш оптимальним є підхід з використанням стилів.

За замовчуванням при створенні проекту в Android Studio в папку *res/values* вже додається файл для стилів *styles.xml*:



Якщо такого файлу немає, то його можна додати, або можна додати ще один файл стилів. Головне щоб стилі перебували в каталозі проекту *res/values/* в файлі xml. При цьому файл може мати довільне ім’я, необов’язково *styles.xml*.

Наявний файл *styles.xml* має наступне визначення:

```
<resources>
   <!-- Base application theme. -->
   <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
      <!-- Customize your theme here. -->
      <item name="colorPrimary">@color/colorPrimary</item>
      <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
      <item name="colorAccent">@color/colorAccent</item>
   </style>
</resources>
```

Стиль задається за допомогою елемента <*style*>. Атрибут **name** вказує на назву стилю, через яке потім можна посилатися на нього. Необов'язковий атрибут parent встановлює для даного стилю батьківський стиль, від якого дочірній стиль буде наслідувати всі свої характеристики.

За допомогою елементів **item** встановлюються конкретні властивості віджета, який приймає в якості значення атрибута **name** ім'я встановлюваної властивості.

Повернемося до нашого завдання по стилізації елементів *TextView* і для її вирішення змінимо файл *styles.xml*:

```
<resources>
   <!-- Base application theme. -->
   <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
      <!-- Customize your theme here. -->
      <item name="colorPrimary">@color/colorPrimary</item>
      <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
      <item name="colorAccent">@color/colorAccent</item>
   </style>
   <style name="TextViewStyle">
      <item name="android:layout_width">match_parent</item>
      <item name="android:layout_height">50dp</item>
      <item name="android:textColor">#3f51b5</item>
      <item name="android:textSize">22sp</item>
      <item name="android:gravity">center</item>
   </style>
</resources>
```

Тут визначено новий стиль *TextViewStyle*, який за допомогою елементів **item** задає значення для атрибутів *TextView*.

Тепер застосуємо стиль, змінимо файл *activity_main.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView
      style="@style/TextViewStyle"
      android:text="Android Lollipop"/>
   <TextView
      style="@style/TextViewStyle"
      android:text="Android Marshmallow"/>
   <TextView
      style="@style/TextViewStyle"
      android:text="Android Nougat"/>
</LinearLayout>
```

Використовуючи визначення style="@style/TextView", текстове поле пов’язується із визначенням стилю. Підсумковий результат буде той же, що і раніше, тільки коду стає менше. А якщо ми захочемо поміняти якісь характеристики, то досить змінити потрібний елемент item у визначенні стилю.

8.2 Теми

8.2.1 Застосування теми

Крім застосування окремих стилів до окремих елементів, ми можемо задавати стилі для всього програми або *activity* у вигляді тем. Ми можемо самі створити тему. Однак Android вже надає кілька попередньо встановлених тем для стилізації застосунків, наприклад, *Theme.AppCompat.Light.DarkActionBar* і ряд інших.

Для визначення тем застосунку відкриємо файл *AndroidManifest.xml*. У ньому ми можемо побачити наступне визначення елемента *application*, що представляє додаток:

```
<application
```

```
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

Задання теми відбувається за допомогою атрибута **android:theme**. В даному випадку використовується ресурс, визначений в стилях – в файлі *res/values/ styles.xml*:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
  <item name="colorPrimary">@color/colorPrimary</item>
  <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
  <item name="colorAccent">@color/colorAccent</item>
</style>
```

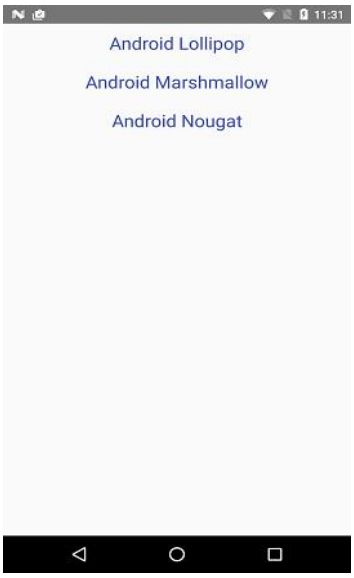
Стиль AppTheme використовує вбудовану тему *Theme.AppCompat.Light.Dark ActionBar*, яка надає візуальні характеристики нашого застосунку. Тепер визначимо стиль, який використовує іншу тему:

```
<resources>
  <style name="NoActionBarTheme"
parent="Theme.AppCompat.DayNight.NoActionBar">
  </style>
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>
  <style name="TextViewStyle">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">50dp</item>
    <item name="android:textColor">#3f51b5</item>
    <item name="android:textSize">22sp</item>
    <item name="android:gravity">center</item>
  </style>
</resources>
```

Нехай новий стиль називається NoActionBarTheme, який посилається на тему Theme.AppCompat.DayNight.NoActionBar. Тепер встановимо його в якості теми застосунку в файлі *AndroidManifest.xml*:

```
<application
   android:theme="@styles/NoActionBarTheme"
```

В результаті отримаємо:



8.2.2 Створення власної теми

Замість використання вбудованих тем, ми можемо створити свою. Для цього створимо в файлі *res/values/styles.xml* новий стиль:

```
<resources>
   <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
      <!-- Customize your theme here. -->
      <item name="colorPrimary">@color/colorPrimary</item>
      <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
      <item name="colorAccent">@color/colorAccent</item>
   </style>
   <style name="TextViewStyle">
      <item name="android:layout_width">match_parent</item>
      <item name="android:layout_height">50dp</item>
      <item name="android:gravity">center</item>
   </style>
```

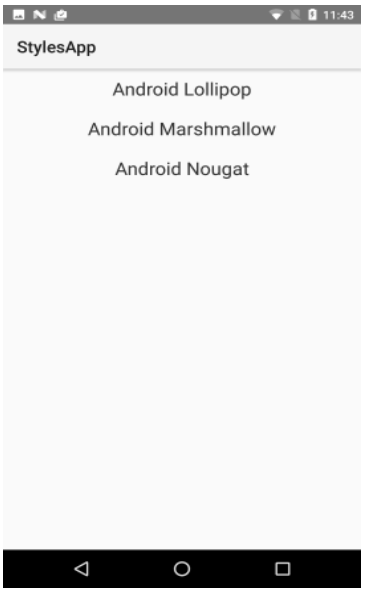
```
    <style name="CustomTheme" parent="Theme.AppCompat.Light">
        <item name="android:textColor">#333333</item>
        <item name="android:textSize">22sp</item>
    </style>
</resources>
```

Отже, ми створили стиль "*CustomTheme*", який успадкований від стилю *Theme.AppCompat.Light*. У цьому стилі ми перевизначили дві властивості: висоту шрифту (**textSize**) – 22*sp*, а також колір тексту (**textColor**) – він тепер світло-сірий.

Тепер визначимо цей стиль як тему додатка в файлі *AndroidManifest.xml*:

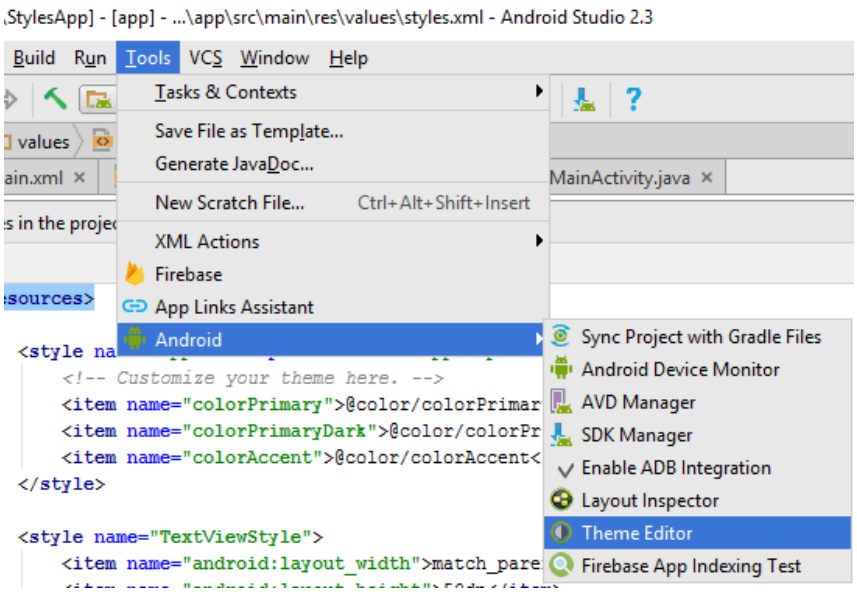
```
<application android:theme="@style/CustomTheme"
```

В результаті отримаємо:

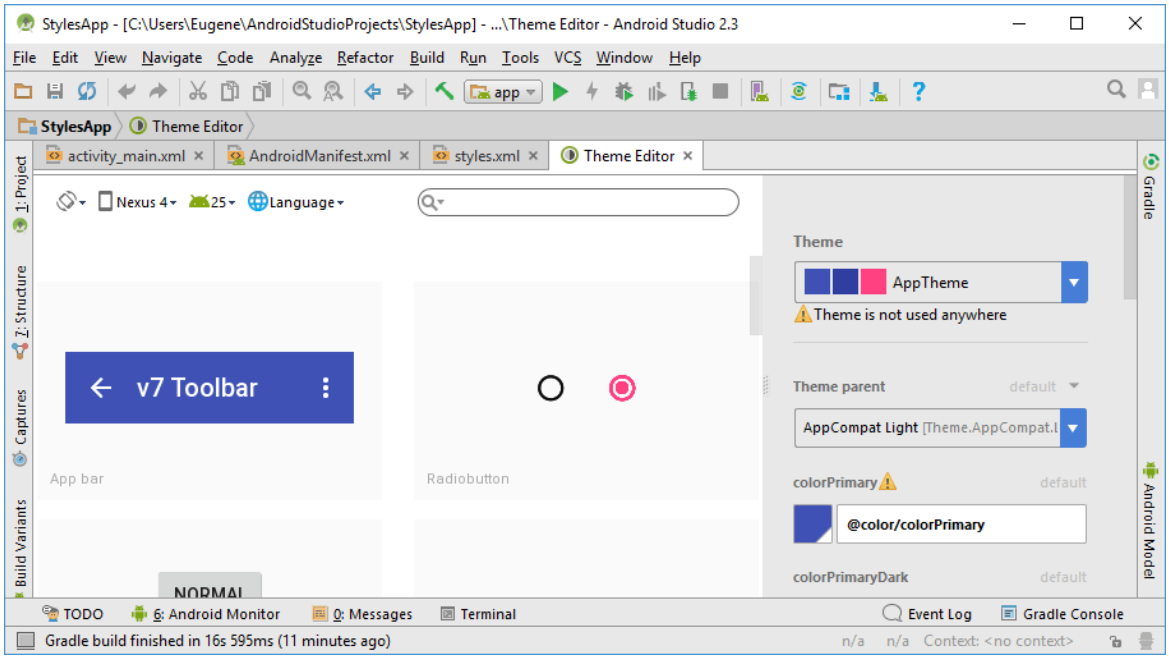


8.2.3 Редактор тем

Для спрощення процесу визначення тем в Android Studio є вбудований графічний редактор тем. Для переходу до нього необхідно вибрати пункт меню *Tools->Android->Theme Editor*:



Після цього відкриється редактор тем, де ми зможемо вибрати будь-яку тему і підредагувати її окремі значення, наприклад, колір:



ЛЕКЦІЯ 9. МЕНЮ

9.1 Створення меню

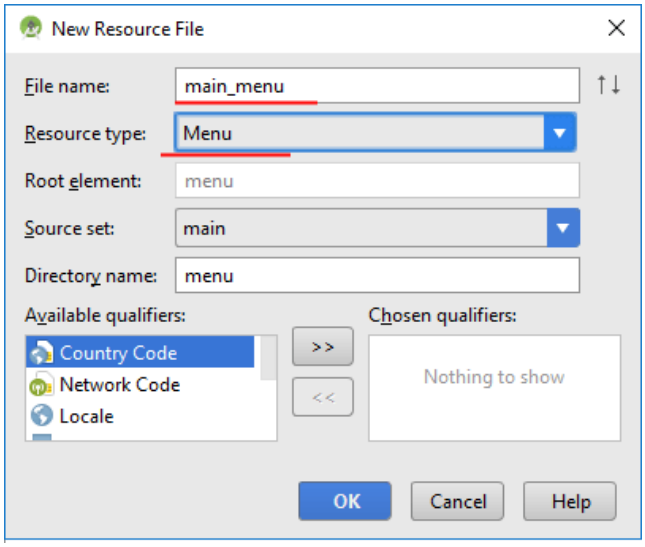
Меню в додатках представляє клас *android.view.Menu*, і кожна *activity* асоціюється з об'єктом цього типу. Об'єкт *android.view.Menu* може включати різну кількість елементів, а ті в свою чергу можуть зберігати піделементи.

9.1.1 Визначення меню в xml

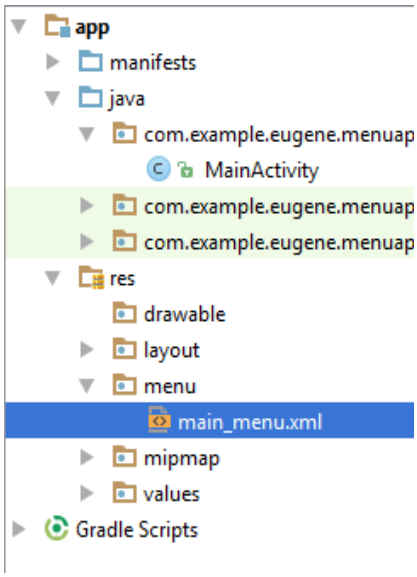
Меню, як і файли інтерфейсу або зображень, також являє собою ресурс. За замовчуванням файли меню знаходяться в проекті в каталозі *res/menu*.

При створенні нового проекту з *Empty Activity* у нас немає ніякого каталогу *res/menu* і відповідно немає ресурсів меню, але ми можемо їх додати вручну. Для цього натиснемо правою кнопкою миші в проекті на каталог *res* і далі в Відкрити список виберемо пункт *New->Android Resource File*.

Далі у вікні вкажемо для імені файлу назва *main_menu*, а для типу ресурсу також виберемо *Menu*:



Після цього в каталозі *res* буде створений підкаталог *menu*, в якому буде знаходитися файл *main_menu.xml*.



По замовчуванню цей файл визначає один порожній елемент *menu*:

```
<?xml version="1.0" encoding="utf-8"?>
 <menu xmlns:android="http://schemas.android.com/apk/res/android">
</menu>
```

Змінимо вміст файлу, визначивши кілька пунктів:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id="@+id/action_settings"
      android:orderInCategory="1"
      android:title="Настройки" />
   <item
      android:id="@+id/save_settings"
      android:orderInCategory="3"
      android:title="Сохранить" />
   <item
      android:id="@+id/open_settings"
      android:orderInCategory="2"
      android:title="Открыть" />
</menu>
```

Тег <*menu*> є кореневим вузлом файлу і визначає меню, що складається з одного або декількох елементів <*item*> і <*group*>.

Елемент <*item*> являє об’єкт *MenuItem*, якій є одним з елементів меню. Цей елемент може містити внутрішній піделементи <*menu*>, за допомогою якого створюється підменю.

Елемент <*item*> включає наступні атрибути, які визначають його зовнішній вигляд і поведінку:

- **android:id**: унікальний *id* елемента меню, який дозволяє його впізнати при виборі користувачем і знайти через пошук ресурсу по *id*;
- **android:icon**: посилання на ресурс *drawable*, який задає зображення для елемента (android:icon="@drawable/ic_help");
- **android:title**: посилання на ресурс рядки, що містить заголовок елемента. Типово має значення "Settings";
- **android:orderInCategory**: порядок проходження елемента в меню.

9.1.2 Наповнення меню елементами

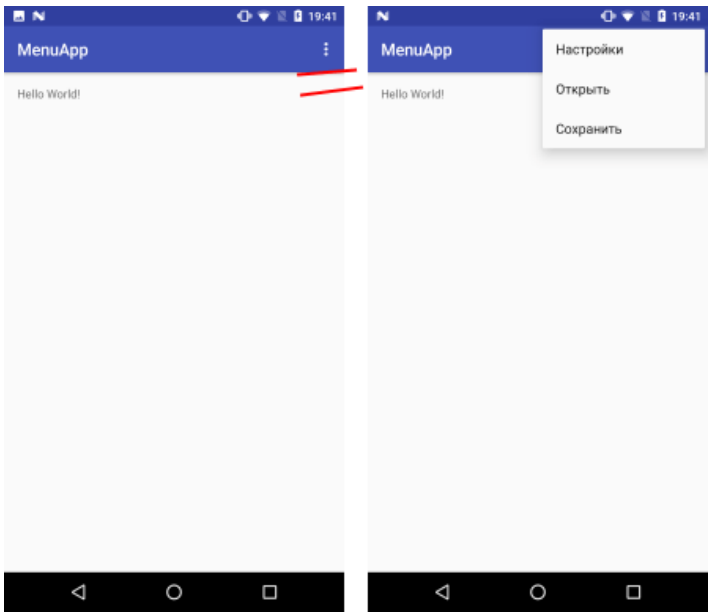
Ми визначили меню з трьома елементами, але саме визначення елементів у файлі ще не створює меню. Це всього лише декларативний опис. Щоб вивести його на екран, нам треба використовувати його в класі *Activity*. Для цього треба перевизначити метод *onCreateOptionsMenu*(). Перейдемо до класу *MainActivity* і змінимо його наступним чином:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main_menu, menu);
        return true;
    }
}
```

Метод *getMenuInflater*() отримує об’єкт *MenuInflater* і викликаємо його метод *inflate*(). Цей метод в якості першого параметра приймає ресурс, який представляє наш

декларативний опис меню в *xml* і наповнює ним об’єкт *menu*, переданий в якості другого параметра.

Запустимо додаток на виконання і натиснемо на кнопку меню в правому верхньому куті. В результаті побачимо:



9.1.3 Обробка натискань на пункти меню

Якщо ми натиснемо на будь-який з пунктів меню, то нічого не станеться. Щоб прив’язати до меню дії, нам треба перевизначити в класі *activity* метод *onOptionsItemSelected*().

Для виведення обраного елемента меню в файлі *activity_main.xml*, визначимо текстове поле з *id=header*:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/header"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="22sp" />
</RelativeLayout>
```

І змінимо клас *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        TextView headerView = (TextView) findViewById(R.id.header);
        switch(id){
            case R.id.action_settings :
                headerView.setText("Настройки");
                return true;
            case R.id.open_settings:
                headerView.setText("Открыть");
                return true;
            case R.id.save_settings:
                headerView.setText("Сохранить");
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Щоб зрозуміти, який пункт меню обраний, спочатку отримуємо його ідентифікатор *int id=item.getItemId()*. Потім в конструкції *switch..case* вибираємо

потрібний варіант і в залежності від вибору здійснюємо відповідні дії, в даному випадку встановлюємо текст *TextView*.

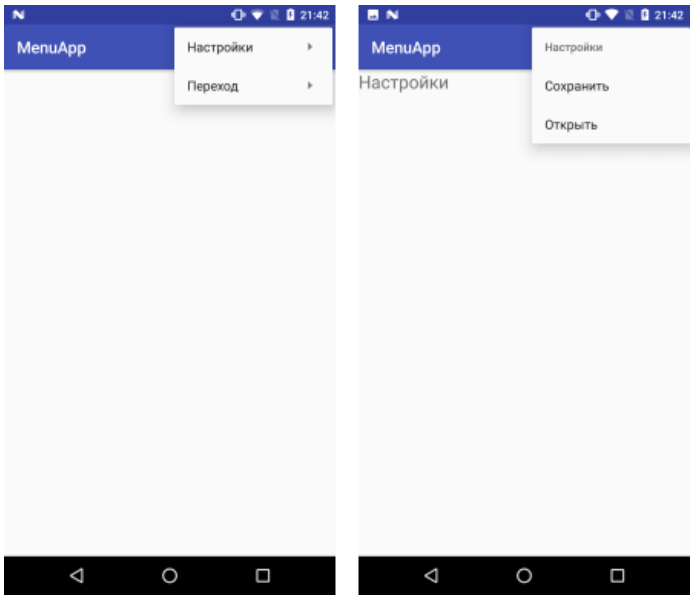
9.2 Групи, підменю і програмне створення меню

9.2.1 Створення підменю

Для створення підменю в файлі розмітки меню визначимо внутрішній елемент *menu*:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id="@+id/action_settings"
      android:title="Настройки">
      <menu>
         <item android:id="@+id/save_settings"
            android:title="Сохранить" />
         <item android:id="@+id/open_settings"
            android:title="Открыть" />
      </menu>
   </item>
   <item
      android:id="@+id/action_move"
      android:title="Переход">
      <menu>
         <item android:id="@+id/forward"
            android:title="Вперед" />
         <item android:id="@+id/back"
            android:title="Назад" />
      </menu>
   </item>
</menu>
```

Після натискання на меню відобразяться елементи верхнього рівня, після натискання на які ми зможемо перейти до підменю:



9.2.2 Групи в меню

Використання елемента *group* дозволяє оформити елементи меню в групу:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <group android:checkableBehavior="single">
      <item
         android:id="@+id/action_settings"
         android:title="Настройки"
         android:checked="true" />
      <item android:id="@+id/save_settings"
         android:title="Сохранить" />
      <item android:id="@+id/open_settings"
         android:title="Открыть" />
   </group>
</menu>
```

У визначенні групи ми можемо встановити атрибут **android:checkableBehavior**. Цей атрибут може набувати таких значень:

- **single**: у кожного елемента створюється радіокнопка;
- **all**: для кожного елемента створюється прапорець;
- **none**.

В даному випадку для кожного елемента буде створюватися радіокнопка (візуально: кружок). Для першого елемента встановлюється зазначена радіокнопка в значення «відмічено» (*android:checked="true"*).

У файлі розмітки інтерфейсу *activity_main.xml* також нехай буде визначено текстове поле:

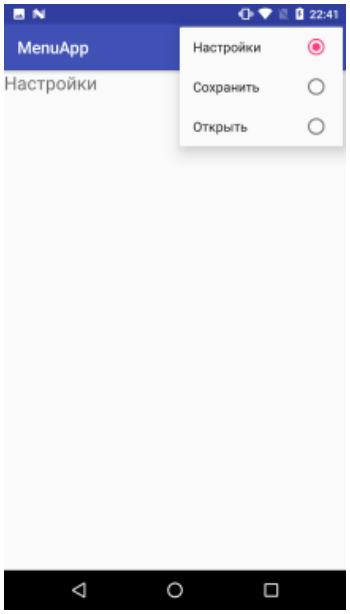
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/header"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="22sp" />
</RelativeLayout>
```

А в класі *MainActivity* визначимо код для виділення радіокнопки в обраного пункту меню:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.main_menu, menu);
      return true;
   }
   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      int id = item.getItemId();
      TextView headerView=(TextView) findViewById(R.id.header);
      if(!item.isChecked()) item.setChecked(true);
```

```
        switch(id){
            case R.id.action_settings :
                headerView.setText("Настройки");
                return true;
            case R.id.open_settings:
                headerView.setText("Открыть");
                return true;
            case R.id.save_settings:
                headerView.setText("Сохранить");
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

В результаті отримаємо на екрані:



9.2.3 Програмне створення меню

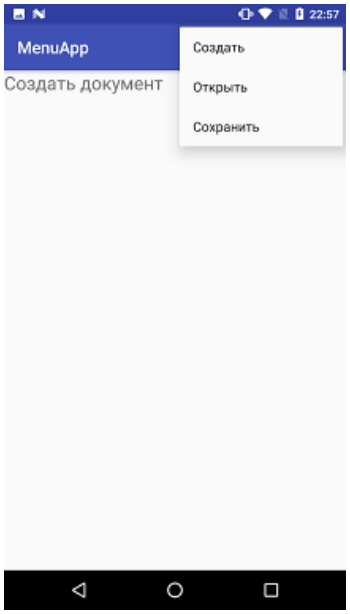
Крім визначення елементів меню в *xml*, можна також створити меню програмним способом. Для додавання нових пунктів меню використовується метод *add*(). Змінимо код *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    @Override
```

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

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   super.onCreateOptionsMenu(menu);

   menu.add(0        // Группа
         ,1    // id
         ,0    //порядок
         ,"Создать"); // заголовок

   menu.add(0,2,1,"Открыть");
   menu.add(0,3,2,"Сохранить");
   return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   int id=item.getItemId();
   TextView headerView=(TextView) findViewById(R.id.header);

   switch(id){
      case 1 :
         headerView.setText("Создать документ");
         return true;
      case 2:
         headerView.setText("Открыть документ");
         return true;
      case 3:
         headerView.setText("Сохранить документ");
         return true;
```

```
        }
        // можна було і так зробити:
        //String title=item.getTitle().toString();
        // headerView.setText(title);
        return super.onOptionsItemSelected(item);
    }
}
```

Метод *add*() додає пункт в меню, приймаючи такі параметри: номер групи, *id*, порядок елемента в меню і заголовок елемента. В результаті отримаємо:



ЛЕКЦІЯ 10. РОБОТА З НАЛАШТУВАННЯМИ ТА СТАНОМ ДОДАТКУ

10.1 Збереження та відновлення стану Activity

Клас *Activity* має в своєму складі методи, призначені для збереження/відновлення стану екземпляра *Activity*:

protected void onRestoreInstanceState(Bundle saveInstanceState);

protected void onSaveInstanceState(Bundle saveInstanceState);

Обидва ці методи в якості параметра приймають об’єкт Bundle, який і містить стан activity.

В якій ситуації може бути доречно використання подібних методів? Найпоширеніша ситуація – переворот екрану і перехід від портретної орієнтації до альбомної і навпаки. Якщо, наприклад, графічний інтерфейс містить текстове поле *TextView*, і ми програмно змінюємо його текст, то після зміни орієнтації екрану його текст може зникнути. Або якщо у нас глобальні змінні, то при зміні орієнтації екрану їх значення можуть бути скинуті до значень за замовчуванням.

Розглянемо приклад. Нехай маємо файл *activity_main* такого змісту:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <EditText
      android:id="@+id/nameBox"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Введіть ім’я"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Зберегти"
      android:onClick="saveName"/>
   <TextView
```

```
    android:id="@+id/nameView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18dip"/>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Отримати ім’я"
    android:onClick="getName"/>
</LinearLayout>
```

Тут визначено поле *EditText*, в яке вводимо ім’я. Також визначена кнопка для його збереження. Для подальшого виведення збереженого імені призначене поле *TextView*, а для отримання збереженого імені – інша кнопка.

Тепер змінимо клас *MainActivity* таким чином:

```
public class MainActivity extends AppCompatActivity {
  String name="undefined";
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void saveName(View view) {
    // отримуємо введене ім’я
    EditText nameBox=(EditText) findViewById(R.id.nameBox);
    name=nameBox.getText().toString();
  }
  public void getName(View view) {
    // отримуємо збережене ім’я
    TextView nameView=(TextView) findViewById(R.id.nameView);
    nameView.setText(name);
  }
}
```

Для зберігання імені в програмі визначена змінна *name*. При натисканні на першу кнопку зберігаємо текст з *EditText* в змінну name, а при натисканні на другу кнопку – отримуємо текст з змінної name в поле *TextView*.

Запустимо додаток, введемо ім'я, збережемо і отримаємо його в *TextView* (рис. 10.1). Та якщо ми перейдемо до альбомного режиму, то *TextView* виявиться порожнім. І навіть якщо ми спробуємо заново отримати значення з змінної name, то ми побачимо, що вона обнулилася (рис. 10.2).

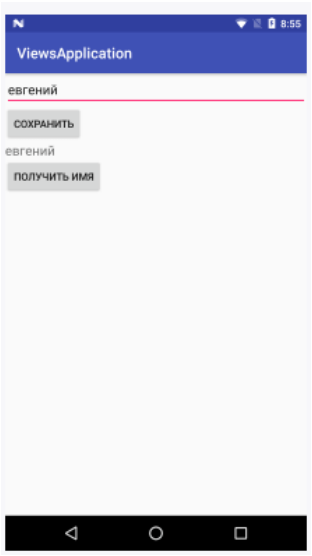


Рисунок 10.1 Вікно додатку при вертикальній орієнтації пристрою

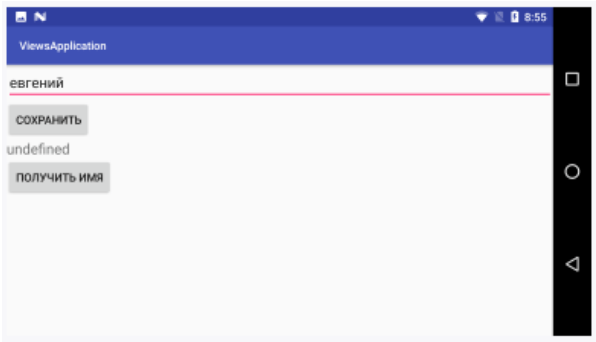


Рисунок 10.2 Вікно додатку при горизонтальній орієнтації пристрою

Щоб уникнути подібних ситуацій слід зберігати і відновлювати стан activity. Для цього змінимо код MainActivity наступним чином:

```
public class MainActivity extends AppCompatActivity {
```

```
String name="undefined";
final static String nameVariableKey="NAME_VARIABLE";
final static String textViewTexKey="TEXTVIEW_TEXT";

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

// збереження стану
@Override
protected void onSaveInstanceState(Bundle outState) {
   outState.putString(nameVariableKey, name);
   TextView nameView=(TextView) findViewById(R.id.nameView);
   outState.putString(textViewTexKey, nameView.getText().toString());
   super.onSaveInstanceState(outState);
}

// отримання раніше збереженого стану
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   name = savedInstanceState.getString(nameVariableKey);
   String textViewText= savedInstanceState.getString(textViewTexKey);
   TextView nameView = (TextView) findViewById(R.id.nameView);
   nameView.setText(textViewText);
}

public void saveName(View view) {
   // отримуємо введене ім'я
   EditText nameBox=(EditText) findViewById(R.id.nameBox);
   name=nameBox.getText().toString();
}
```

```
    public void getName(View view) {
        // отримуємо збережене ім’я
        TextView nameView=(TextView) findViewById(R.id.nameView);
        nameView.setText(name);
    }
}
```

У методі *onSaveInstanceState*() зберігаємо стан. Для цього викликаємо у параметра *Bundle* метод *putString(key, value)*, перший параметр якого – ключ, а другий – значення для збереження даних. В даному випадку ми зберігаємо два рядки, тому викликаємо метод *putString*(). Для збереження об’єктів інших типів даних ми можемо викликати відповідний метод:

- *putInt();*
- *putDouble();*
- *putFloat();*
- *putChar(;)*
- *putByte().*

Кожен із перерахованих методів в якості першого параметра приймає ключ, а в якості другого – значення.

В методі *onRestoreInstanceState()* відбувається зворотний процес – за допомогою методу *getString(key)* по ключу отримуємо із збереженого стану рядок по ключу. Відповідно для отримання даних інших типів ми можемо використовувати аналогічні методи:

- *getInt();*
- *getDouble();*
- *getFloat();*
- *getChar();*
- *getByte().*

10.2 Збереження та отримання налаштувань

10.2.1 Загальні принципи роботи із налаштуваннями

Часто в програмі доводиться зберігати невеликі шматочки даних для подальшого використання (дані про користувача, налаштування конфігурації і т. д.).

Для цього в Android існує концепція *Preferences*. Налаштування представляють собою групу пар ключ-значення, які використовуються додатком.

В якості значень можуть виступати дані наступних типів: *Boolean, Float, Integer, Long, String*, набір рядків.

Налаштування є загальними для всіх activity в додатку, але також можуть бути і налаштування безпосередньо для окремих activity

Налаштування зберігаються в xml-файлах в незашифрованому вигляді в локальному сховищі. Вони невидимі, тому для простого користувача недоступні.

При роботі з налаштуваннями слід враховувати наступні моменти. оскільки вони зберігаються в незашифрованому вигляді, то не рекомендується зберігати в них чутливі дані типу пароля або номерів кредитних карт. Крім того, вони представляють дані, що асоціюються з додатком, і використовуючи панель управління додатком в налаштуваннях ОС користувач може видалити ці дані.

10.2.2 Загальні налаштування

Для роботи із загальними налаштуваппнями в класі *Activity* (точніше, в його базовому класі *Context*) є метод *getSharedPreferences*():

```
import android.content.SharedPreferences;
//.....
SharedPreferences settings = getSharedPreferences("PreferencesName", MODE_PRIVATE);
```

Перший параметр методу вказує на назву налаштувань. В даному випадку це "*PreferencesName*". Якщо налаштувань з подібною назвою немає, то вони створюються при виклику даного методу. Другий параметр вказує на режим доступу. В даному випадку режим описаний константою *MODE_PRIVATE*.

Клас *android.content.SharedPreferences* надає ряд методів для управління налаштуваннями:

- *contains(String key)*: повертає true, якщо в налаштуваннях збережено значення із ключем *key*;
- *getAll()*: повертає всі збережені в налаштуваннях значення;
- *getBoolean(String key, boolean defValue)*: повертає з налаштувань значення типу *Boolean*, якому відповідає ключ *key*. Якщо елемента з таким ключем не виявиться, то повертається значення *defValue*, передане іншим параметром;

- *getFloat(String key, float defValue)*: повертає значення типу *float* з ключем *key*. Якщо елемента з таким ключем не виявиться, то повертається значення *defValue*;
- *getInt(String key, int defValue)*: повертає значення типу *int* з ключем *key*;
- *getLong(String key, long defValue)*: повертає значення типу *long* з ключем *key*;
- *getString(String key, String defValue)*: повертає стрічкове значення з ключем *key*;
- *getStringSet(String key, Set <String> defValues)*: повертає масив рядків з ключем *key*;
- *edit()*: повертає об'єкт *SharedPreferences*.Editor, який використовується для редагування налаштувань.

Для управління налаштуваннями використовується об'єкт класу *SharedPreferences*.Editor, що повертається методом *edit()*. Він має такі методи:

- *clear()*: витирає всі налаштування;
- *remove(String key)*: витирає із налаштувань значення з ключем *key*;
- *putBoolean(String key, boolean value)*: додає в налаштування значення типу *boolean* з ключем *key*;
- *putFloat(String key, float value)*: додає в налаштування значення типу *float* з ключем *key*;
- *putInt(String key, int value)*: додає в налаштування значення *int* з ключем *key*;
- *putLong(String key, long value)*: додає в налаштування значення типу *long* з ключем *key*;
- *putString(String key, String value)*: додає в настройки рядок з ключем *key*;
- *putStringSet (String key, Set <String> values)*: додає в налаштування масив стрічок;
- *commit()*: підтверджує всі зміни в налаштуваннях;
- *apply()*: також, як і метод *commit()*, підтверджує всі зміни в налаштуваннях, проте змінений об'єкт *SharedPreferences* спочатку зберігається в тимчасовій пам'яті, і лише потім в результаті асинхронної операції записується на мобільний пристрій.

Розглянемо приклад збереження та отримати налаштувань додатком. Визначимо в файлі activity_main.xml наступний призначений для користувача інтерфейс:

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

```
    android:orientation="vertical">
    <EditText
        android:id="@+id/nameBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Введіть ім’я"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Зберегти"
        android:onClick="saveName"/>
    <TextView
        android:id="@+id/nameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Отримати ім’я"
        android:onClick="getName"/>
</LinearLayout>
```

На екрані будуть дві кнопки – для збереження і для виведення попередньо збереженого значення, а також поле для введення і текстове поле для виведення збереженого налаштування. Визначимо методи обробники кнопок в класі *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    private static final String PREFS_FILE = "Account";
    private static final String PREF_NAME = "Name";
    SharedPreferences settings;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);
```

```
    }

    public void saveName(View view) {
        EditText nameBox = (EditText) findViewById(R.id.nameBox);
        String name = nameBox.getText().toString();
        // Зберігаємо введено ім’я в налаштуваннях
        SharedPreferences.Editor prefEditor = settings.edit();
        prefEditor.putString(PREF_NAME, name);
        prefEditor.apply();
    }

    public void getName(View view) {
        // отримуємо збережене ім’я
        TextView nameView = (TextView) findViewById(R.id.nameView);
        String name = settings.getString(PREF_NAME,"не определено");
        nameView.setText(name);
    }
}
```

За відсутності налаштувань при спробі їх отримати, додаток виведе значення за замовчуванням. Написаний код дає можливість зберегти і вивести на екран збережене значення.

Інколи виникає потреба автоматично зберігати дані, що вводяться при виході користувача з activity. Для цього ми можемо перевизначити метод *onPause*():

```
public class MainActivity extends AppCompatActivity {
    private static final String PREFS_FILE="Account";
    private static final String PREF_NAME="Name";
    EditText nameBox;
    SharedPreferences settings;
    SharedPreferences.Editor prefEditor;

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

```
        nameBox=(EditText) findViewById(R.id.nameBox);
        settings=getSharedPreferences(PREFS_FILE, MODE_PRIVATE);

        // отримуємо налаштування
        String name=settings.getString(PREF_NAME,"");
        nameBox.setText(name);
    }

    @Override
    protected void onPause(){
        super.onPause();
        EditText nameBox=(EditText) findViewById(R.id.nameBox);
        String name=nameBox.getText().toString();
        // зберігаємо в налаштуваннях
        prefEditor=settings.edit();
        prefEditor.putString(PREF_NAME, name);
        prefEditor.apply();
    }

    public void saveName(View view) {
    }
    public void getName(View view) {
    }
}
```

10.2.3 Приватні налаштування

Крім загальних налаштувань кожна activity може використовувати приватні, доступ до яких з інших activity буде неможливим. Для отримання налаштувань рівня activity використовується метод *getPreferences(MODE_PRIVATE)*:

```
import android.content.SharedPreferences;
//.....................
SharedPreferences settings = getPreferences(MODE_PRIVATE);
```

Тобто, на відміну від загальних налаштувань, тут не використовується назва групи налаштувань в якості першого параметра, так як це відбувається в методі *getSharedPreferences()*. Однак вся інша робота по додаванню, отриманню та зміні налаштувань буде аналогічною розглянутій вище роботі із загальними налаштуваннями.

10.3 PreferenceFragment

Для спрощення роботи з групою налаштувань Android надає спеціальний тип фрагмента – *PreferenceFragment*. Розглянемо як його можна використовувати. Створимо новий проект і спочатку додамо в папку *res* підпапку *xml*. Потім в папку *res / xml* додамо новий файл, який назвемо *settings.xml* із таким вмістом:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <EditTextPreference
      android:key="login"
      android:summary="Введіть логін"
      android:title="Логін" />
   <CheckBoxPreference
      android:key="enabled"
      android:summary="Показати логін"
      android:title="Показати" />
</PreferenceScreen>
```

Тут в кореневому елементі *PreferenceScreen* встановлюються елементи *EditTextPreference* і *CheckBoxPreference*. Через кожен з цих елементів ми можемо взаємодіяти з певною налаштуванням. В даному випадку ми можемо використовувати декілька різних типів налаштувань:

- *EditTextPreference*: використовується елемент *EditText* для введення текстового значення;
- *CheckBoxPreference*: використовується елемент *CheckBox* для встановлення логічних значень *true* або *false*;
- *SwitchPreference*: використовується елемент *Switch* для встановлення логічних значень *true* або *false* ("on" і "off");

- *RingtonePreference*: використовує діалогове вікно для встановлення рінгтона із списку рингтонів для встановлення логічних значень *true* або *false*;
- *ListPreference*: використовує список для вибору одного з визначених значень;
- *MultiSelectListPreference*: також використовує список для вибору значень, але дозволяє вибрати кілька елементів.

Для кожного елемента налаштування необхідно визначити, як мінімум, три атрибути:

1) *android: key*: задає ключ налаштування в *SharedPreferences*;
2) *android: title*: назва настройки для користувача;
3) *android: summary*: задає короткий опис налаштування для користувача.

Нарешті додамо в проект спеціальну activity для задання налаштувань. Назвемо її *SettingsActivity*:

```
public class SettingsActivity extends AppCompatActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_settings);

      getFragmentManager()
            .beginTransaction()
            .add(R.id.prefs_content, new SettingsFragment())
            .commit();
   }

   public static class SettingsFragment extends PreferenceFragment {
      @Override
      public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         addPreferencesFromResource(R.xml.settings);
      }
   }
}
```

SettingsActivity в якості розмітки інтерфейсу буде використовувати ресурс *R.layout.activity_settings*. При запуску *SettingsActivity* буде завантажувати фрагмент *SettingsFragment* в елемент з ідентифікатором *prefs*_content.

Сам фрагмент *SettingsFragment* успадковується від класу *PreferenceFragment*. В його методі *onCreate()* викликається метод addPreferencesFromResource(), в який передається *id* ресурсу *xml* з налаштуваннями (в даному випадку раніше визначений ресурс *R.xml.settings*).

Тепер визначимо в папці *res / layout* наступний файл *activity_settings.xml*:

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

Тут визначено *FrameLayout* з ідентифікатором prefs*_content* - саме той елемент, в який буде завантажуватися фрагмент *SettingsFragment*.

Тепер перейдемо до головної activity - *MainActivity*. У файлі *activity_main.xml* опишемо текстове поле і кнопку:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView
      android:id="@+id/loginText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp" />
   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Налаштування"
      android:onClick="setPrefs"/>
</LinearLayout>
```

І нарешті, змінимо клас *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
```

```
    TextView loginText;
    boolean enabled;
    String login;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        loginText=(TextView) findViewById(R.id.loginText);
    }

    @Override
    public void onResume() {
        super.onResume();
        SharedPreferences prefs=PreferenceManager.getDefaultSharedPreferences(this);
        enabled = prefs.getBoolean("enabled", false);
        login = prefs.getString("login", "не установлено");
        loginText.setText(login);
        if(enabled)
            loginText.setVisibility(View.VISIBLE);
        else
            loginText.setVisibility(View.INVISIBLE);
    }

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

Тут в методі *onResume*() ми отримуємо всі налаштування. Якщо налаштування *enabled* дорівнює *true*, то відображаємо текстове поле з логіном.

У методі *setPrefs()*, який спрацьовує при натисканні на кнопку, відбувається перехід до *SettingsActivity*.

При першому запуску налаштувань не буде, і логін не буде відображуватися. Перейдемо на сторінку налаштувань, встановимо там логін і включимо його відображення, а потім повернемося на головну activity (рис. 10.3).

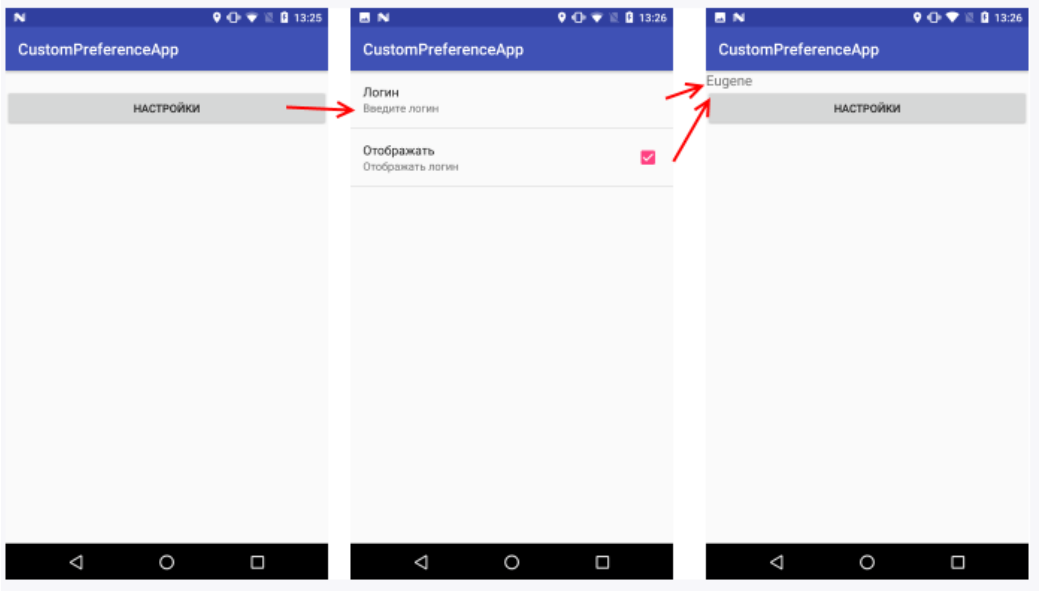


Рисунок 10.3 Демонстрація автоматичного збереження налаштувань додатку

Тобто, вручну нам нічого не треба зберігати, всі налаштування автоматично зберігаються функціоналом *PreferenceFragment*.

ЛЕКЦІЯ 11. РОБОТА З ФАЙЛОВОЮ СИСТЕМОЮ

11.1 Читання і збереження файлів

Робота з налаштуваннями рівня *activity* і застосунку дозволяє зберегти невеликі дані окремих типів (*string, int*), але для роботи з великими масивами даних, такими як графічні файли, файли мультимедіа і т. д., нам доведеться звертатися до файлової системи.

ОС Android побудована на основі Linux. Цей факт знаходить своє відображення в роботі з файлами. Так, в шляхах до файлів, в якості розділювача, Linux використовує слеш *"/"*, а не зворотний слеш *"\"* (як у Windows). А всі назви файлів і каталогів є чутливими до регістру, тобто *"data"* це не те ж саме, що і *"Data"*.

Додаток Android зберігає свої дані в каталозі */data/data/<назва_пакета>/* і, як правило, відносно цього каталогу буде відбуватися робота.

Для роботи з файлами абстрактний клас *android.content.Context* визначає ряд методів:

- *deleteFile(String name)*: видаляє певний файл;
- *fileList*(): повертає всі файли, які містяться в підкаталозі /*files* каталога додатку;
- *getCacheDir*(): повертає посилання на підкаталог /*cache* каталога додатку;
- *getDir(String dirName, int mode)*: повертає посилання на підкаталог в каталозі додатки, якщо такого підкаталогу немає, то він створюється;
- *getExternalCacheDir*(): повертає посилання на папку /*cache* зовнішньої файлової системи пристрою;
- *getExternalFilesDir*(): повертає посилання на каталог /*files* зовнішньої файлової системи пристрою;
- *getFileStreamPath(String filename)*: повертає абсолютний шлях до файлу в файловій системі;
- *openFileInput(String filename)*: відкриває файл для читання;
- *openFileOutput (String name, int mode)*: відкриває файл для запису.

Всі файли, які створюються і редагуються в програмі, як правило, зберігаються в підкаталозі */files* в каталозі застосунку.

Для безпосереднього читання і запису файлів застосовуються також стандартні класи Java з пакету *java.io*.

Отже, можна застосувати функціонал читання-запису файлів в додатку. Нехай у нас буде наступна примітивна розмітка layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <EditText
      android:id="@+id/save_text"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:gravity="start"
      android:layout_weight="4"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:layout_margin="16dp"
      android:layout_gravity="center"
      android:onClick="saveText"
      android:text="Сохранить"/>
   <TextView
      android:layout_marginTop="80dp"
      android:id="@+id/open_text"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:gravity="start"
      android:layout_weight="4"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="0dp"
      android:layout_margin="16dp"
      android:layout_weight="1"
      android:layout_gravity="center"
      android:onClick="openText"
```

```
    android:text="Открыть"/>
</LinearLayout>
```

Поле *EditText* призначене для введення тексту, а *TextView* – для виведення попередньо записаного тексту. Для збереження і відновлення тексту додані дві кнопки.

Тепер в коді *Activity* пропишемо обробники кнопок зі збереженням і читанням файлу:

```
public class MainActivity extends AppCompatActivity {
   private final static String FILE_NAME="content.txt";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   // збереження файла
   public void saveText(View view){
      FileOutputStream fos=null;
      try {
         EditText textBox=(EditText) findViewById(R.id.save_text);
         String text=textBox.getText().toString();
         fos=openFileOutput(FILE_NAME, MODE_PRIVATE);
         fos.write(text.getBytes());
         Toast.makeText(this, "Файл сохранен", Toast.LENGTH_SHORT).show();
      }
      catch(IOException ex) {
         Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
      }
      finally{
         try{
            if(fos!=null)
               fos.close();
         }
         catch(IOException ex){
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
```

```
        }
    }
}
// відкриття файла
public void openText(View view){
    FileInputStream fin = null;
    TextView textView = (TextView) findViewById(R.id.open_text);
    try {
        fin=openFileInput(FILE_NAME);
        byte[] bytes=new byte[fin.available()];
        fin.read(bytes);
        String text=new String (bytes);
        textView.setText(text);
    }
    catch(IOException ex) {
        Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fin!=null)
                fin.close();
        }
        catch(IOException ex){
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
}
```

При натисканні на кнопку збереження буде створюватися потік виведення FileOutputStream fos=openFileOutput(FILE_NAME, MODE_PRIVATE).

В даному випадку введений текст буде зберігатися в файл *"content.txt"*. При цьому буде використовуватися режим **MODE_PRIVATE**.

Система дозволяє створювати файли з двома різними режимами:

- **MODE_PRIVATE** : файли можуть бути доступні тільки власнику додатки (режим за замовчуванням);
- **MODE_APPEND** : дані можуть бути додані в кінець файлую

Тому в даному випадку якщо файл *"content.txt"* вже існує, то він буде перезаписаний. Якщо ж нам треба було дописати файл, тоді треба було б використовувати режим **MODE_APPEND**:

```
FileOutputStream fos=openFileOutput(FILE_NAME, MODE_APPEND);
```

Для читання файлу застосовується потік введення *FileInputStream* :

```
FileInputStream fin=openFileInput(FILE_NAME);
```

Детальніше про використання потоків введення-виведення можна прочитати в будь-якій документації по Java.

У підсумку після натискання на кнопку збереження весь текст буде збережений у файлі */data/data/назва_пакету/files/content.txt*:

11.2 Розміщення файлів у зовнішньому сховищі

У попередньому підрозділі ми розглянули збереження і читання файлів з каталогу програми. За замовчуванням такі файли доступні тільки самому застосунку. Однак ми можемо поміщати і працювати з файлами із зовнішнього сховища. Це також дозволить іншим програмам відкривати дані файли і при необхідності змінювати.

Весь механізм роботи з файлами буде таким же, як і при роботі зі сховищем застосунку. Ключовою відмінністю тут буде отримання і використання шляху до зовнішнього сховища через метод *Environment.getExternalStorageDirectory()*.

Отже, нехай у файлі *activity_main.xml* буде така ж розмітка інтерфейсу:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <EditText
      android:id="@+id/save_text"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:gravity="start"
      android:layout_weight="4"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:layout_margin="16dp"
      android:layout_gravity="center"
      android:onClick="saveText"
      android:text="Сохранить"/>
   <TextView
      android:layout_marginTop="80dp"
      android:id="@+id/open_text"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:gravity="start"
      android:layout_weight="4"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="0dp"
      android:layout_margin="16dp"
      android:layout_weight="1"
```

```
        android:layout_gravity="center"
        android:onClick="openText"
        android:text="Открыть"/>
</LinearLayout>
```

А код класу *MainActivity* буде виглядати наступним чином:

```
public class MainActivity extends AppCompatActivity {
    private final static String FILE_NAME = "content.txt";
    private static final int REQUEST_PERMISSION_WRITE = 1001;
    private boolean permissionGranted;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    private File getExternalPath() {
        return(new File(Environment.getExternalStorageDirectory(), FILE_NAME));
    }
    // збереження файла
    public void saveText(View view){
        if(!permissionGranted){
            checkPermissions();
            return;
        }
        FileOutputStream fos = null;
        try {
            EditText textBox = (EditText) findViewById(R.id.save_text);
            String text = textBox.getText().toString();
            fos = new FileOutputStream(getExternalPath());
            fos.write(text.getBytes());
            Toast.makeText(this, "Файл сохранен", Toast.LENGTH_SHORT).show();
        }
        catch(IOException ex) {
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
```

```
    finally{
        try{
            if(fos!=null)
                fos.close();
        }
        catch(IOException ex){
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
// відкриття файла
public void openText(View view){
    if(!permissionGranted){
        checkPermissions();
        return;
    }
    FileInputStream fin = null;
    TextView textView = (TextView) findViewById(R.id.open_text);
    File file = getExternalPath();
    // якщо файл не існує, відбувається вихід из коду метода
    if(!file.exists()) return;
    try {
        fin =  new FileInputStream(file);
        byte[] bytes = new byte[fin.available()];
        fin.read(bytes);
        String text = new String (bytes);
        textView.setText(text);
    }
    catch(IOException ex) {
        Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fin!=null)
```

```
                fin.close();
            }
            catch(IOException ex){
                Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    }

    // перевіряємо, чи є доступним зовнішнє сховище для читання і запису
    public boolean isExternalStorageWriteable(){
        String state = Environment.getExternalStorageState();
        return  Environment.MEDIA_MOUNTED.equals(state);
    }
    // перевіряємо, чи є доступним зовнішнє сховище хоча б для читання
    public boolean isExternalStorageReadable(){
        String state = Environment.getExternalStorageState();
        return  (Environment.MEDIA_MOUNTED.equals(state) ||
                Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
    }

    private boolean checkPermissions(){
        if(!isExternalStorageReadable() || !isExternalStorageWriteable()){
            Toast.makeText(this, "Зовнішнє сховище не доступне",
Toast.LENGTH_LONG).show();
            return false;
        }
        int permissionCheck = ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
        if(permissionCheck!= PackageManager.PERMISSION_GRANTED){
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
REQUEST_PERMISSION_WRITE);
            return false;
        }
```

```
        return true;
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults){
        switch (requestCode){
            case REQUEST_PERMISSION_WRITE:
                if(grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
                    permissionGranted=true;
                    Toast.makeText(this, "Дозвіл отримано", Toast.LENGTH_LONG).show();
                }
                else{
                    Toast.makeText(this, "Необхідно отримати дозвіл",
Toast.LENGTH_LONG).show();
                }
                break;
        }
    }
}
```

За допомогою виразу *Environment.getExternalStorageDirectory()*, отримуємо доступ до папки програми в зовнішньому сховищі і встановлюємо об’єкт файлу:

```
private File getExternalPath() {
    return(new File(Environment.getExternalStorageDirectory(), FILE_NAME));
}
```

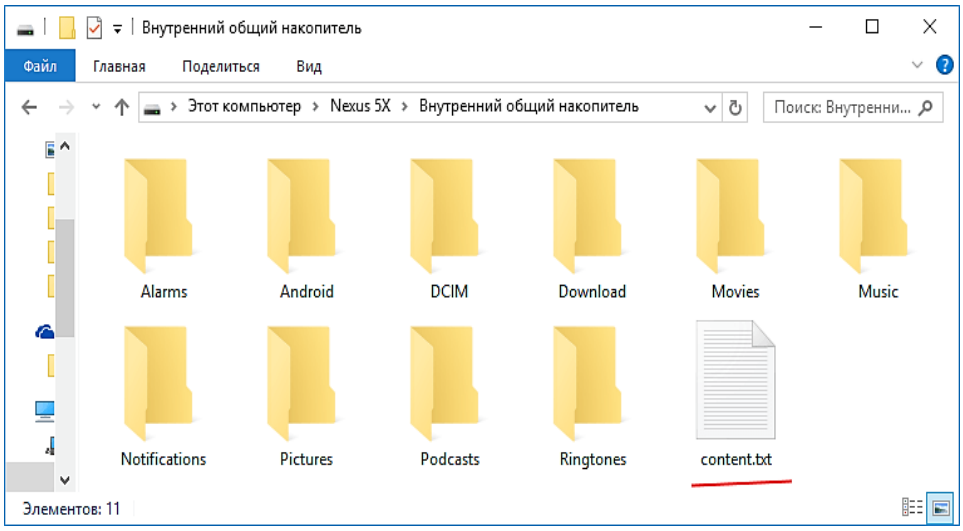
Оскільки для читання/запису в зовнішнє сховище необхідні дозволи, то перед операціями збереження та записи файлу необхідно перевірити наявність дозволів. Для цього визначено метод *checkPermissions*(). При установці дозволів спрацьовує метод *onRequestPermissionsResult*(), в якому в разі вдалої установки дозволів для змінної *permissionGranted* задається значення *true*.

Щоб використовувати зовнішнє сховище, також треба встановити дозволу в файлі маніфесту *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
package="com.example.eugene.filesapp" >
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
//.....
```

І після операції запису на смартфоні ми зможемо побачити створений файл:



11.3 Робота з JSON

Для роботи з форматом *json* немає вбудованих засобів, але є багато бібліотек і пакетів. Одним із найбільш популярних з них є пакет *com.google.code.gson* .

Для його використання в проекті Android, необхідно додати відповідну залежність в файл *guild.gradle*, який відноситься до модуля *app*:

```
compile 'com.google.code.gson:gson:2.8.0'
```

Тобто після додавання, секція залежностей в файлі *build.gradle* може виглядати наступним чином:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
```

```
    compile 'com.google.code.gson:gson:2.8.0'
    testCompile 'junit:junit:4.12'
}
```

Після додавання пакета в проект, додамо новий клас *Phone*, який буде представляти дані:

```
package com.example.eugene.jsonapp;

public class Phone {
    private String name;
    private int price;

    Phone(String name, int price){
        this.name=name;
        this.price=price;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name=name;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price=price;
    }

    @Override
    public String toString(){
```

```
        return name+" "+String.valueOf(price);
    }
}
```

Об’єкти цього класу ми будемо серіалізовувати у формат *json* і навпаки десеріалізовувати з файлу.

Для роботи з *json* додамо наступний клас *JSONHelper*:

```
class JSONHelper {
    private static final String FILE_NAME="data.json";

    static boolean exportToJSON(Context context, List<Phone> dataList) {
        Gson gson=new Gson();
        DataItems dataItems=new DataItems();
        dataItems.setPhones(dataList);
        String jsonString=gson.toJson(dataItems);
        FileOutputStream fileOutputStream=null;
        try {
            fileOutputStream = context.openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
            fileOutputStream.write(jsonString.getBytes());
            return true;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (fileOutputStream !=null) {
                try {
                    fileOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return false;
    }
```

```
static List<Phone> importFromJSON(Context context) {
    InputStreamReader streamReader=null;
    FileInputStream fileInputStream=null;
    try{
        fileInputStream=context.openFileInput(FILE_NAME);
        streamReader=new InputStreamReader(fileInputStream);
        Gson gson=new Gson();
        DataItems dataItems=gson.fromJson(streamReader, DataItems.class);
        return  dataItems.getPhones();
    }
    catch (IOException ex){
        ex.printStackTrace();
    }
    finally {
        if (streamReader !=null) {
            try {
                streamReader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (fileInputStream !=null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return null;
}

private static class DataItems {
    private List<Phone>phones;
```

```
    List<Phone> getPhones() {
        return phones;
    }
    void setPhones(List<Phone> phones) {
        this.phones=phones;
    }
  }
}
```

Тут для роботи із *json* створюється об’єкт *Gson* . Для серіалізації даних в формат *json* у цього об’єкта викликається метод toJson(), в який передаються серіалізовувані дані.

Для спрощення роботи з даними застосовується допоміжний клас *DataItems*. На виході метод *toJson*() повертає рядок, яка потім зберігається в текстовий файл.

Для десеріалізації виконується метод *fromJson*(), в який передається об’єкт *Reader* з серіалізованими даними і тип, до якого треба десеріалізувати дані.

Тепер визначимо основний функціонал для взаємодії з користувачем. Змінимо файл *activity_main.xml* наступним чином:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <EditText
    android:id="@+id/nameText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введите название"/>
  <EditText
    android:id="@+id/priceText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введите цену"/>
  <Button
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Добавить"
        android:onClick="addPhone"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:text="Сохранить"
            android:onClick="save"/>
        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:text="Открыть"
            android:onClick="open"/>
    </LinearLayout>
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Тут визначені два текстових поля для введення назви моделі і ціни об’єкта *Phone* і одна кнопка для додавання даних в список. Ще одна кнопка виконує серіалізацію даних зі списку в файл, а третя кнопка – відновлення даних з файлу. Для виведення самих даних визначено елемент *ListView*. І змінимо відповідним чином клас *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    private ArrayAdapter<Phone> adapter;
    private EditText nameText, priceText;
    private List<Phone> phones;
```

```
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        nameText=(EditText) findViewById(R.id.nameText);
        priceText=(EditText) findViewById(R.id.priceText);
        phones=new ArrayList<>();
        listView=(ListView) findViewById(R.id.list);
        adapter=new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, phones);
        listView.setAdapter(adapter);
    }
    public void addPhone(View view){
        String name=nameText.getText().toString();
        int price=Integer.parseInt(priceText.getText().toString());
        Phone phone=new Phone(name, price);
        phones.add(phone);
        adapter.notifyDataSetChanged();
    }
    public void save(View view){
        boolean result = JSONHelper.exportToJSON(this, phones);
        if(result){
            Toast.makeText(this, "Данные сохранены", Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось сохранить данные",
Toast.LENGTH_LONG).show();
        }
    }
    public void open(View view){
        phones = JSONHelper.importFromJSON(this);
        if(phones!=null){
            adapter=new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, phones);
            listView.setAdapter(adapter);
```

```
            Toast.makeText(this, "Данные восстановлены", Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось открыть данные",
Toast.LENGTH_LONG).show();
        }
    }
}
```

Всі дані знаходяться в списку *phones*, який представляє об’єкт List<Phone>. Через адаптер цей список зв’язується з *ListView*.

Для збереження і відновлення даних викликаються раніше описані методи в класі *JSONHelper*. В результаті отримаємо:

ЛЕКЦІЯ 12. РОБОТА З БАЗАМИ ДАНИХ SQLITE

12.1 Підключення до бази даних SQLite

12.1.1 Загальні основи

В Android є вбудована підтримка однієї з поширених систем управління базами даних – SQLite. Для цього в пакеті *android.database.sqlite* визначено набір класів, які дозволяють працювати з базами даних SQLite. Кожен додаток може створити свою базу даних.

Щоб використовувати SQLite в Android, треба створити базу даних за допомогою відповідного виразу на мові SQL. Після цього база даних буде зберігатися в катало зі застосунку *DATA/data/[назва_застосунку]/databases/[назва_файла_бази_даних]*.

ОС Android по замовчуванню вже містить ряд вбудованих баз SQLite, які використовуються стандартними програмами – для списку контактів, для зберігання фотографій з камери, музичних альбомів і т. д.

Основну функціональність по роботі з базами даних надає пакет *android.database*. Функціональність безпосередньо для роботи з SQLite знаходиться в пакеті *android.database.sqlite*.

База даних в SQLite представлена класом *android.database.sqlite.SQLiteDatabase*. Він дозволяє виконувати запити до БД, виконувати з нею різні маніпуляції.

Клас *android.database.sqlite.SQLiteCursor* надає запит і дозволяє повертати набір рядків, які відповідають цьому запиту.

Клас *android.database.sqlite.SQLiteQueryBuilder* дозволяє створювати SQL-запити. Самі sql-вирази представлені класом *android.database.sqlite.SQLiteStatement*, які дозволяють за допомогою плейсхолдерів вставляти в вирази динамічні дані.

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

У SQLite застосовується наступна система типів даних:

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

Дані, що зберігаються в базі, повинні відповідати типам даних в мові програмуваня Java.

12.1.2 Створення та відкриття бази даних

Для створення або відкриття нової бази даних з коду *Activity* в Android ми можемо викликати метод *openOrCreateDatabase*(). Цей метод може приймати три параметри:

- назва для бази даних;
- числове значення, яке визначає режим роботи (як правило, у вигляді константи **MODE_PRIVATE**);
- необов'язковий параметр у вигляді об'єкта *SQLiteDatabase.CursorFactory,* який представляє фабрику створення курсора для роботи з БД.

Наприклад, створення бази даних *app.db*:

```
SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE, null);
```

Для виконання запиту до бази даних можна використовувати метод *execSQL* класу *SQLiteDatabase*. У цей метод передається SQL-вираз. Наприклад, створення в базі даних таблиці *users*:

```
SQLiteDatabase db=getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE, null);
db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");
```

Якщо нам треба не просто виконати вираз, але і отримати з БД будь-які дані, то використовується метод *rawQuery*(). Цей метод в якості параметра приймає SQL-вираз, а також набір значень для виразу SQL. Наприклад, отримання всіх об'єктів з бази даних:

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

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

Метод *db.rawQuery()* повертає об’єкт *Cursor*, за допомогою якого ми можемо витягти отримані дані.

Можлива ситуація, коли в базі даних не буде об’єктів, і для цього методом *query.moveToFirst()* намагаємося переміститися до першого об’єкту, отриманого з БД. Якщо цей метод поверне значення *false*, значить запит не отримав ніяких даних з БД.

Тепер для роботи з базою даних розробимо простий додаток. Для цього створимо новий проект. У файлі *activity_main.xml* визначимо найпростіший графічний інтерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="16dp"
   android:orientation="vertical">
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click"
      android:onClick="onClick"/>
   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="18sp" />
</LinearLayout>
```

А в класі *MainActivity* визначимо взаємодію з базою даних:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
```

```
    public void onClick(View view){
        SQLiteDatabase db=getBaseContext().openOrCreateDatabase("app.db",
MODE_PRIVATE, null);
        db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");
        db.execSQL("INSERT INTO users VALUES ('Tom Smith', 23);");
        db.execSQL("INSERT INTO users VALUES ('John Dow', 31);");
        Cursor query = db.rawQuery("SELECT * FROM users;", null);
        TextView textView = (TextView) findViewById(R.id.textView);
        if(query.moveToFirst()){
            do{
                String name=query.getString(0);
                int age=query.getInt(1);
                textView.append("Name: "+name+" Age: "+age+"\n");
            }
            while(query.moveToNext());
        }
        query.close();
        db.close();
    }
}
```

При натисканні на кнопку тут спочатку створюється в базі даних *app.db* нова таблиця *users*, а потім в неї додаються два об’єкти в базу даних за допомогою SQL-виразу **INSERT**.

Далі за допомогою виразу **SELECT** отримуємо всіх доданих користувачів з бази даних у вигляді курсору *Cursor*.

Викликом *query.moveToFirst()* переміщаємося в курсорі до першого об’єкту, і так як у нас може бути більше одного об’єкта, то проходимо по всьому курсору в циклі *do ... while*.

Для отримання даних з курсору застосовуються методи *query.getString(0)* і *query.getInt(1)*. У дужках в методи передається номер стовпчика, з якого ми отримуємо дані. Наприклад, вище ми додали спочатку ім’я користувача у вигляді рядка, а потім вік у вигляді числа. Значить, нульовим стовпцем буде йти рядку значення, яке отримуємо за допомогою методу *getString*(), а першим стовпцем йде числове значення, для якого застосовується метод *getInt*().

Після завершення роботи з курсором і базою даних ми закриваємо всі пов’язані об’єкти. Якщо ми не закриємо курсор, то можемо зіткнутися з проблемою витоку пам’яті: query.close(); db.close();

І якщо ми тепер звернемося до додатка, то після натискання на кнопку в текстове поле будуть виведені дані:

12.2 SimpleCursorAdapter і отримання даних

12.2.1 Використання об’єкта SQLiteOpenHelper

Підемо далі і створимо повністю інтерфейс для роботи з базою даних. Отже, створимо новий проект. Для спрощення роботи з базами даних SQLite в Android нерідко застосовується клас *SQLiteOpenHelper*. Для використання необхідно створити клас-спадкоємець від *SQLiteOpenHelper*, перевизначивши як мінімум два його методи:

- *onCreate()*: викликається при спробі доступу до бази даних, але коли ще ця база даних не створена;
- *onUpgrade()*: викликається, коли потрібно оновити схеми бази даних. Тут можна перебудувати раніше створену базу даних в onCreate(), встановивши відповідні правила перетворення від старої БД до нової.

Тому додамо в проект в ту ж папку, де знаходиться клас *MainActivity*, новий клас *DatabaseHelper*:

```
public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME="userstore.db"; // назва бд
```

```
    private static final int SCHEMA=1; // версія
    static final String TABLE="users"; // назва таблиць в бд
    // назви стовпців
    public static final String COLUMN_ID="_id";
    public static final String COLUMN_NAME="name";
    public static final String COLUMN_YEAR="year";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, SCHEMA);
    }

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

Якщо база даних відсутня або її версія (яка задається у змінній **SCHEMA**) вище поточної, то спрацьовує метод *onCreate*().

Для виконання запитів до бази даних нам потрібно об’єкт *SQLiteDatabase*, який представляє базу даних. Метод *onCreate*() отримує в якості параметра базу даних програми.

Для виконання запитів до SQLite використовується метод *execSQL*(). Він приймає SQL-вираз **CREATE TABLE**, який створює таблицю. Тут також при необхідності ми можемо виконати й інші запити, наприклад, додати будь-які

початкові дані. Так, в даному випадку за допомогою того ж методу і виразу SQL **INSERT**, додається один об’єкт в таблицю.

У методі *onUpgrade*() відбувається оновлення схеми БД. В даному випадку для прикладу використаний примітивний похід з видаленням попередньої бази даних за допомогою SQL-виразу **DROP** і подальшим її створенням. Але в реальності якщо вам буде необхідно зберегти дані, цей метод може включати більш складну логіку – додавання нових стовпців, видалення непотрібних, додавання додаткових даних і т. д.

Далі визначимо в файлі *activity_main.xml* наступну розмітку:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/header"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="18dp"/>
   <ListView
      android:id="@+id/list"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

Тут визначено список *ListView*, для відображення отриманих даних, з заголовком, який буде виводити число отриманих об’єктів. Змінимо код класу *MainActivity* наступним чином:

```
public class MainActivity extends AppCompatActivity {
   ListView userList;
   TextView header;
   DatabaseHelper databaseHelper;
   SQLiteDatabase db;
   Cursor userCursor;
   SimpleCursorAdapter userAdapter;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        header = (TextView)findViewById(R.id.header);
        userList = (ListView)findViewById(R.id.list);
        databaseHelper = new DatabaseHelper(getApplicationContext());
    }
    @Override
    public void onResume() {
        super.onResume();
        // відкриваємо підключення
        db = databaseHelper.getReadableDatabase();
        //отримуємо дані з бд у вигляді курсора
        userCursor =  db.rawQuery("select * from "+ DatabaseHelper.TABLE, null);
        // визначаємо стовпці, які будуть виводитися в ListView
        String[] headers = new String[] {DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};
        // створюємо адаптер
        userAdapter = new SimpleCursorAdapter(this,    android.R.layout.two_line_list_item,
userCursor, headers,
new int[]{android.R.id.text1, android.R.id.text2}, 0);
        header.setText("Знайдено елементів: "+String.valueOf(userCursor.getCount()));
        userList.setAdapter(userAdapter);
    }

    @Override
    public void onDestroy(){
        super.onDestroy();
        // Закриваємо підключення і курсор
        db.close();
        userCursor.close();
    }
}
```

У методі *onCreate*() відбувається створення об’єкта *SQLiteOpenHelper*. Сама ініціалізація об’єктів для роботи з базою даних відбувається в методі *onResume*(), який спрацьовує після методу *onCreate*().

Щоб отримати об’єкт бази даних, слід використовувати метод *getReadableDatabase*() (отримання бази даних для читання) або *getWritableDatabase*(). Так як в даному випадку ми будемо лише зчитувати дані з бд, то скористаємося першим методом: db=sqlHelper.getReadableDatabase().

12.2.2 Отримання даних і Cursor

Android надає різні способи для здійснення запитів до об’єкта SQLiteDatabase. У більшості випадків ми можемо застосовувати метод *rawQuery*(), який приймає два параметри: SQL-вираз **SELECT** і додатковий параметр, що задає параметри запиту.

Після виконання запиту *rawQuery*() повертає об’єкт *Cursor*, який зберігає результат виконання SQL-запиту:

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

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

- *getCount()*: Отримує кількість витягнутих з бази даних об’єктів.
- методи *moveToFirst()* і *moveToNext*() дозволяють переходити до першого і до наступного елементів вибірки. Метод *isAfterLast*() дозволяє перевірити, чи досягнуто кінець вибірки;
- Методи *get...(columnIndex*) (наприклад, *getLong*(), *getString*()) дозволяють за індексом стовпчика звернутися до даного стовпця поточного рядка.

12.2.3 CursorAdapter

Додатково, для управління курсором в Android є клас *CursorAdapter*. Він дозволяє адаптувати отриманий за допомогою курсора набір до відображення в спискових елементах на кшталт *ListView*. Як правило, при роботі з курсором використовується підклас *CursorAdapter* – *SimpleCursorAdapter*. Хоча можна використовувати і інші адаптери, типу *ArrayAdapter*.

```
userAdapter = new SimpleCursorAdapter(this, android.R.layout.two_line_list_item,
    userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);
userList.setAdapter(userAdapter);
```

Конструктор класу *SimpleCursorAdapter* приймає шість параметрів:

- **Першим параметром** виступає контекст, з яким асоціюється адаптер, наприклад, поточна *activity*;
- **Другий параметр** – ресурс розмітки інтерфейсу, який буде використовуватися для відображення результатів вибірки;
- **Третій параметр** – курсор;
- **Четвертий параметр** – список стовпців з вибірки, які будуть відображатися в розмітці інтерфейсу;
- **П'ятий параметр** – елементи всередині ресурсу розмітки, які будуть відображати значення стовпців з четвертого параметра;
- **Шостий параметр** – прапори, що задають поведінки адаптера.

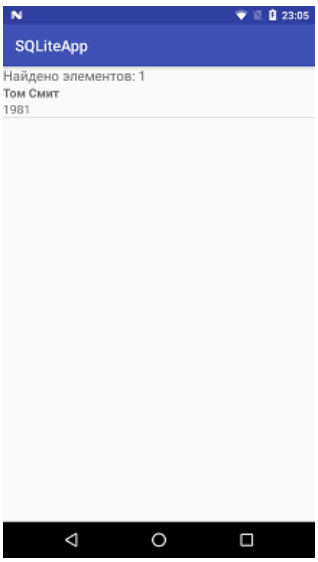
При використанні *CursorAdapter* і його підкласів слід враховувати, що вибірка курсора повинна включати цілочисельний стовпець з назвою *_id*, який повинен бути унікальним для кожного елемента вибірки. Значення цього стовпця при натисканні на елемент списку потім передається в метод обробки *onListItemClick*(), завдяки чому ми можемо по *id* ідентифікувати натиснутий елемент.

В даному випадку у нас перший стовпець як раз називається *"_id"*.

Після завершення роботи курсор повинен бути закритий методом *close*().

І також треба враховувати, що якщо ми використовуємо курсор в *SimpleCursorAdapter*, то ми не можемо використовувати метод *close*(), поки не завершимо використання *SimpleCursorAdapter*. Тому метод *cursor* краще викликати в методі onDestroy() фрагмента або *activity*.

І якщо ми запустимо додаток то побачимо список з одного доданого елемента:



12.3 Додавання, видалення та оновлення даних в SQLite

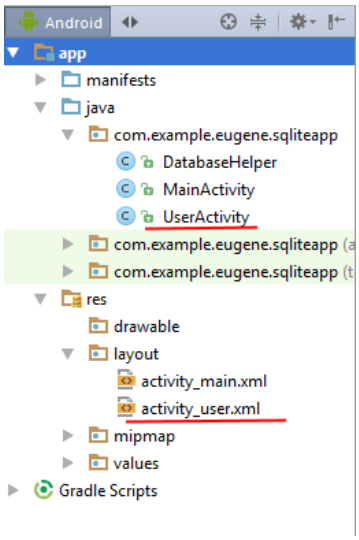
12.3.1 ContentValues

Продовжимо роботу з проектом. Тепер додамо в нього стандартну CRUD-логіку (створення, оновлення, видалення). Щоб не нагромаджувати форму з головною *activity*, всі інші дії по роботі з даними будуть відбуватися на іншому екрані. Додамо в проект новий клас *activity*, який назвемо *UserActivity*.

У файлі *activity_user.xml* визначимо форму для додавання/оновлення/видалення даних:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="16dp">
   <EditText
      android:id="@+id/name"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Введіть ім’я"/>
   <EditText
      android:id="@+id/year"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Введіть рік нарождения"/>
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal">
      <Button
         android:id="@+id/saveButton"
         android:layout_width="0dp"
         android:layout_weight="1"
         android:layout_height="wrap_content"
```

```
            android:text="Зберегти"
            android:onClick="save"/>
        <Button
            android:id="@+id/deleteButton"
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:text="Видалити"
            android:onClick="delete"/>
    </LinearLayout>
</LinearLayout>
```



Також змінимо код *UserActivity*:

```
public class UserActivity extends AppCompatActivity {
    EditText nameBox;
    EditText yearBox;
    Button delButton;
    Button saveButton;
    DatabaseHelper sqlHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    long userId=0;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);
        nameBox=(EditText) findViewById(R.id.name);
        yearBox=(EditText) findViewById(R.id.year);
        delButton=(Button) findViewById(R.id.deleteButton);
        saveButton= (Button) findViewById(R.id.saveButton);
        sqlHelper=new DatabaseHelper(this);
        db=sqlHelper.getWritableDatabase();
        Bundle extras=getIntent().getExtras();
        if (extras !=null) {
            userId=extras.getLong("id");
        }
        // якщо 0, то додаємо
        if (userId > 0) {
            // отримуємо елемент по id з бд
            userCursor=db.rawQuery("select*from "+DatabaseHelper.TABLE +" where " +
                    DatabaseHelper.COLUMN_ID+"=?", new String[]{String.valueOf(userId)});
            userCursor.moveToFirst();
            nameBox.setText(userCursor.getString(1));
            yearBox.setText(String.valueOf(userCursor.getInt(2)));
            userCursor.close();
        } else {
            // ховаємо кнопку видалення
            delButton.setVisibility(View.GONE);
        }
    }

    public void save(View view){
        ContentValues cv=new ContentValues();
        cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());
        cv.put(DatabaseHelper.COLUMN_YEAR,
Integer.parseInt(yearBox.getText().toString()));
```

```
        if (userId > 0) {
            db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID + "=" +
String.valueOf(userId), null);
        } else {
            db.insert(DatabaseHelper.TABLE, null, cv);
        }
        goHome();
    }
    public void delete(View view){
        db.delete(DatabaseHelper.TABLE, "_id=?", new String[]{String.valueOf(userId)});
        goHome();
    }
    private void goHome(){
        // закриваємо підключення
        db.close();
        // перехід до головної activity
        Intent intent=new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
        startActivity(intent);
    }
}
```

При оновленні або видаленні об’єкта зі списку з головною activity в *UserActivity* передаватиметься *id* об’єкта:

```
long userId=0;
//.....
Bundle extras=getIntent().getExtras();
if (extras !=null) {
    userId=extras.getLong("id");
}
```

Якщо з *MainActivity* не було передано *id*, то встановлюємо його значення 0, отже, у нас буде додавання, а не редагування/знищення. Якщо *id* визначено, то отримуємо по ньому з бази даних об’єкт для редагування/видалення:

```
if (id < 0) {
```

```
    userCursor=db.rawQuery("select*from "+DatabaseHelper.TABLE+" where "+
        DatabaseHelper.COLUMN_ID+"=?", new String[]{String.valueOf(id)});
    userCursor.moveToFirst();
    nameBox.setText(userCursor.getString(1));
    yearBox.setText(String.valueOf(userCursor.getInt(2)));
    userCursor.close();
}
```

Інакше просто приховуємо кнопку видалення. Для виконання операцій по вставці, оновленню і видаленню даних *SQLiteDatabase* має методи *insert*(), *update*() і *delete*(). Ці методи викликаються в обробниках кнопок:

```
db.delete(DatabaseHelper.TABLE, "_id=?", new String[]{String.valueOf(id)});
```

У метод *delete*() передається назва таблиці, а також стовпець, по якому відбувається видалення, і його значення. В якості критерію можна вибрати кілька стовпців, тому третім параметром йде масив. Знак питання *?* позначає параметр, замість якого підставляється значення з третього параметра.

12.3.2 ContentValues

Для додавання або оновлення нам треба створити об'єкт *ContentValues*. Даний об'єкт являє собою словник, який містить набір пар «ключ-значення». Щоб додати до цього словника новий об'єкт застосовується метод *put()*. Перший параметр методу – це ключ, а другий – значення, наприклад:

```
ContentValues cv=new ContentValues();
cv.put("NAME", "Tom");
cv.put("YEAR", 30);
```

В якості значення в метод *put*() можна передавати рядки, цілі числа, числа з плаваючою точкою. В даному ж випадку додаються введені в текстове поля значення:

```
ContentValues cv =new ContentValues();
cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());
cv.put(DatabaseHelper.COLUMN_YEAR, Integer.parseInt(yearBox.getText().toString()));
```

При оновленні в метод *update*() передається назва таблиці, об'єкт *ContentValues* і критерій, за яким відбувається оновлення (в даному випадку стовпець *id*):

```
db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID+"="+
String.valueOf(id), null);
```

Метод *insert*() приймає назву таблиці, об'єкт *ContentValues* з додаваними значеннями. Другий параметр є необов'язковим: він передає стовпець, в який треба додати значення *NULL*: db.insert(DatabaseHelper.TABLE, null, cv).

Замість цих методів можна використовувати метод *execSQL*() з точним зазначенням виконуваного SQL-виразу. У той же час методи *delete / insert / update* мають перевагу – вони повертають *id* зміненого запису, за яким ми можемо дізнатися про успішність операції, або -1 у випадку невдалої операції:

```
long result=db.insert(DatabaseHelper.TABLE, null, cv);
if(result > 0){
   // дії
}
```

Після кожної операції виконується метод *goHome*(), який повертає на головну *activity*.

Після цього нам треба виправити код *MainActivity*, щоб вона ініціювала виконання коду в *UserActivity*. Для цього змінимо код *activity_main.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Добавити"
      android:onClick="add"
      android:textSize="18dp"/>
   <ListView
      android:id="@+id/list"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

В даному випадку була додана кнопка для виклику *UserActivity*. Змінимо відповідним чином код класу *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    ListView userList;
    DatabaseHelper databaseHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        userList=(ListView)findViewById(R.id.list);
        userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Intent intent=new Intent(getApplicationContext(), UserActivity.class);
                intent.putExtra("id", id);
                startActivity(intent);
            }
        });
        databaseHelper = new DatabaseHelper(getApplicationContext());
    }
    @Override
    public void onResume() {
        super.onResume();
        db=databaseHelper.getReadableDatabase();
        userCursor=db.rawQuery("select*from "+DatabaseHelper.TABLE, null);
        String[] headers=new String[] {DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};
        userAdapter = new SimpleCursorAdapter(this, android.R.layout.two_line_list_item,
                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);
        userList.setAdapter(userAdapter);
```

```
    }
    // при натисканні на кнопку запускаємо UserActivity для добавлення даних
    public void add(View view){
        Intent intent=new Intent(this, UserActivity.class);
        startActivity(intent);
    }

    @Override
    public void onDestroy(){
        super.onDestroy();
        db.close();
        userCursor.close();
    }
}
```

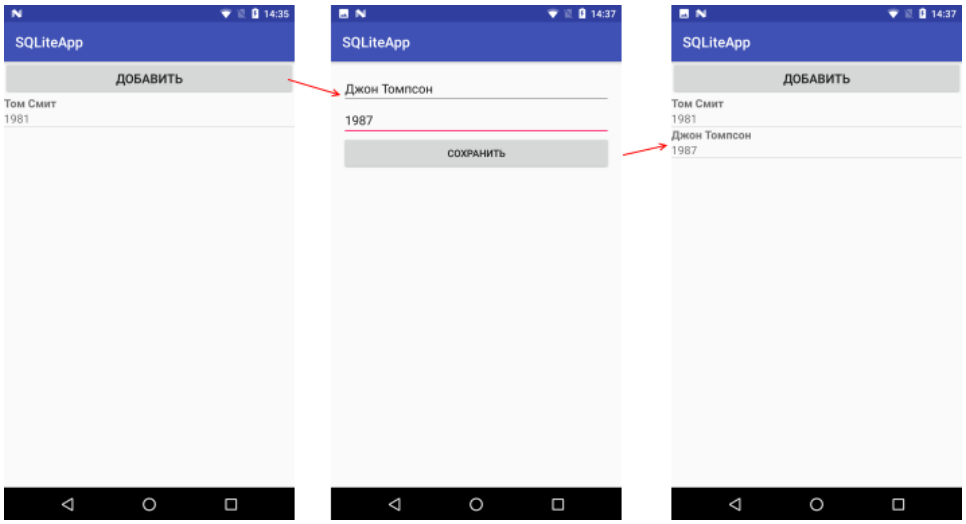
При натисканні на кнопку запускається *UserActivity*, при цьому не передається ніякого *id*, тобто в *UserActivity id* буде дорівнює нулю, значить буде відбуватися додавання даних:

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

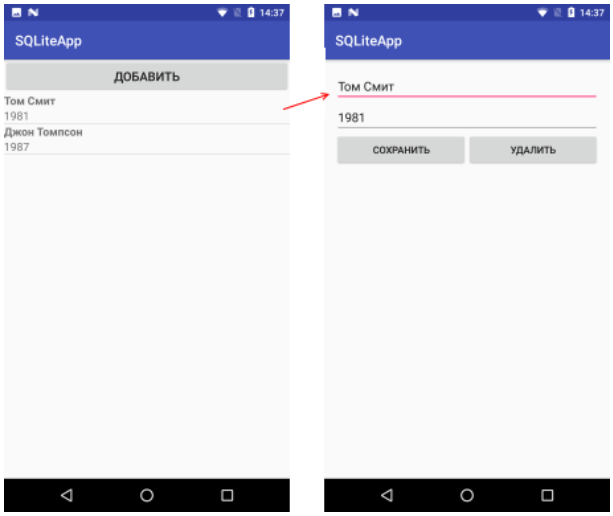
Іншу ситуацію представляє обробник натискання на елемент списку – при натисканні також буде запускатися *UserActivity*, але тепер буде передаватися *id* обраного запису:

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    Intent intent=new Intent(getApplicationContext(), UserActivity.class);
    intent.putExtra("id", id);
    startActivity(intent);
}
```

Запустимо програму і натиснемо на кнопку, яка має перенаправляти на *UserActivity*:



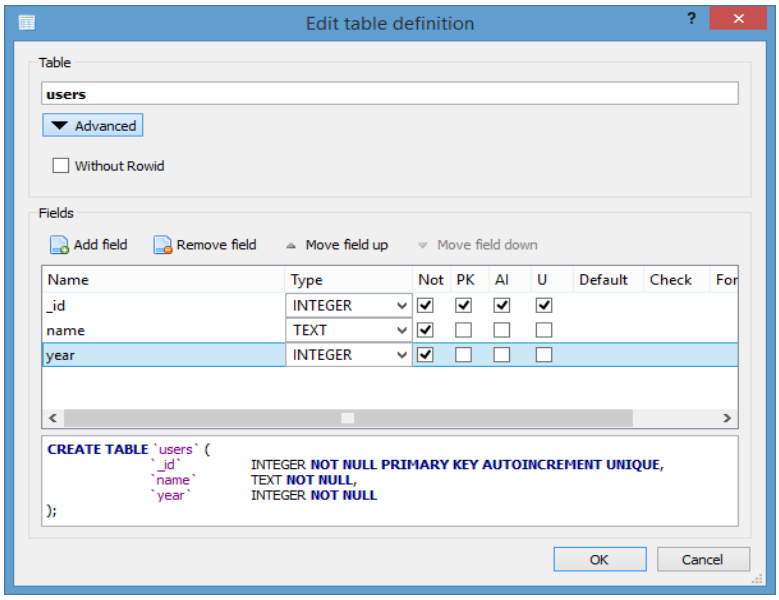
При натисканні в *MainActivity* на елемент списку цей елемент потрапить на *UserActivity*, де його можна буде видалити або відредагувати:



12.4 Використання існуючої БД SQLite

Крім створення нової бази даних ми також можемо використовувати вже існуючу. Візьмемо проект, створений в попередніх темах. Для початку створимо базу даних SQLite. У цьому нам може допомогти такий інструмент як **Sqlitebrowser**. Він безкоштовний і доступний для різних операційних систем за адресою http://sqlitebrowser.org/. Хоча можна використовувати і інші способи для створення початкової БД.

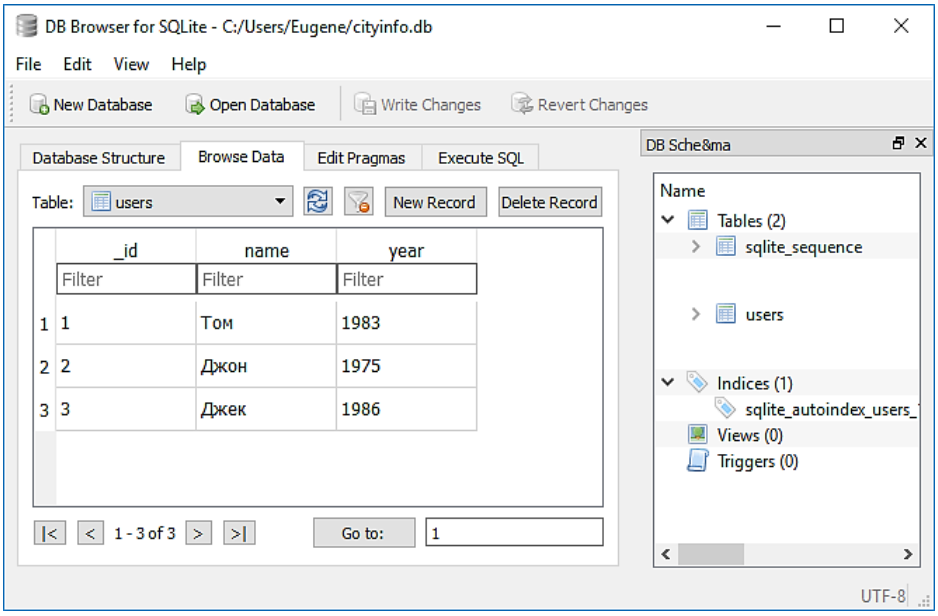
Sqlitebrowser представляє графічний інтерфейс для створення бази даних і визначення в ній всіх необхідних таблиць:



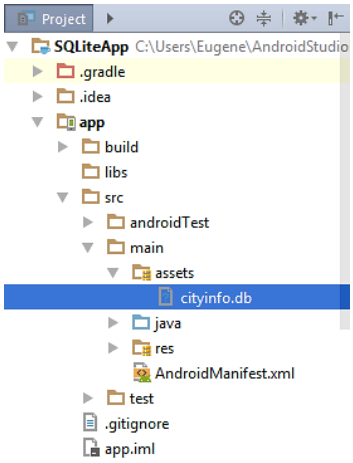
Як видно з рисунку, визначаємо таблицю *users* з трьома полями: *_id*, *name*, *age*. Команда на створення таблиці буде наступною:

```
CREATE TABLE `users` (
   `_id`  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   `name`  TEXT NOT NULL,
   `year`  INTEGER NOT NULL
);
```

Там же в програмі додамо кілька елементів в створену таблицю:



Після створення таблиці додамо в проект в Android Studio папку *assets*, а в папку *assets* – щойно створену базу даних. Для цього перейдемо до повного визначення проекту, натиснемо на папку *main* правою кнопкою миші і в меню виберемо *New->Directory*. Назвемо додавану папку *assets* і потім скопіюємо в неї нашу базу даних:



В нашому випадку база даних називається *"cityinfo.db"*. Змінимо код *DatabaseHelper* наступним чином:

```
class DatabaseHelper extends SQLiteOpenHelper {
   private static String DB_PATH;
   private static String DB_NAME="cityinfo.db";
   private static final int SCHEMA=1;
   static final String TABLE="users";
   static final String COLUMN_ID="_id";
   static final String COLUMN_NAME="name";
   static final String COLUMN_YEAR="year";
   private Context myContext;

   DatabaseHelper(Context context) {
      super(context, DB_NAME, null, SCHEMA);
      this.myContext=context;
      DB_PATH=context.getFilesDir().getPath()+DB_NAME;
   }
```

```
@Override
public void onCreate(SQLiteDatabase db) {
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {
}

void create_db(){
   InputStream myInput=null;
   OutputStream myOutput=null;
   try {
      File file=new File(DB_PATH);
      if (!file.exists()) {
         this.getReadableDatabase();
         // отримуємо локальну бд як потік
         myInput = myContext.getAssets().open(DB_NAME);
         // шлях до нової бд
         String outFileName=DB_PATH;
         // відкриваємо пусту бд
         myOutput=new FileOutputStream(outFileName);
         // побайтово копіюємо дані
         byte[] buffer=new byte[1024];
         int length;
         while ((length=myInput.read(buffer)) > 0) {
            myOutput.write(buffer, 0, length);
         }
         myOutput.flush();
         myOutput.close();
         myInput.close();
      }
   }
   catch(IOException ex){
      Log.d("DatabaseHelper", ex.getMessage());
   }
```

```
    }
    public SQLiteDatabase open()throws SQLException {
        return SQLiteDatabase.openDatabase(DB_PATH, null,
SQLiteDatabase.OPEN_READWRITE);
    }
}
```

По замовчуванню база даних буде розміщуватися в зовнішньому сховищі, що виділяється для програми у папці */data/data/[назва_пакета]/databases/*, і щоб отримати повний шлях до бази даних в конструкторі використовується вираз:
DB_PATH=context.getFilesDir().getPath()+DB_NAME;

Метод *onCreate*() нам не потрібен, так як нам не потрібне створення вбудованої бази даних. Зате тут визначено додатковий метод *create_db()*, мета якого копіювання бази даних з папки *assets* в те місце, яке зазначено в змінній **DB_PATH**.

Крім цього тут також визначено метод відкриття бази даних *open*() за допомогою методу *SQLiteDatabase.openDatabase*().

Новий спосіб організації підключення змінить використання *DatabaseHelper* в *activity*. Тому, слід оновити клас *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    ListView userList;
    DatabaseHelper databaseHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        userList=(ListView)findViewById(R.id.list);
        userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Intent intent=new Intent(getApplicationContext(), UserActivity.class);
                intent.putExtra("id", id);
```

```
            startActivity(intent);
        }
    });
    databaseHelper=new DatabaseHelper(getApplicationContext());
    databaseHelper.create_db();
}
@Override
public void onResume() {
    super.onResume();
    db=databaseHelper.open();
    userCursor=db.rawQuery("select*from "+DatabaseHelper.TABLE, null);
    String[] headers=new String[] {DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};
    userAdapter=new SimpleCursorAdapter(this, android.R.layout.two_line_list_item,
userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);
    userList.setAdapter(userAdapter);
}
// при натисканні на кнопку запускаємо UserActivity для добавлення даних
public void add(View view){
    Intent intent=new Intent(this, UserActivity.class);
    startActivity(intent);
}

@Override
public void onDestroy(){
    super.onDestroy();
    db.close();
    userCursor.close();
}
}
```

Змінимо клас *UserActivity* таким чином:

```
public class UserActivity extends AppCompatActivity {
    EditText nameBox;
    EditText yearBox;
```

```
    Button delButton;
    Button saveButton;
    DatabaseHelper sqlHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    long userId=0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);
        nameBox=(EditText) findViewById(R.id.name);
        yearBox=(EditText) findViewById(R.id.year);
        delButton=(Button) findViewById(R.id.deleteButton);
        saveButton=(Button) findViewById(R.id.saveButton);
        sqlHelper=new DatabaseHelper(this);
        db=sqlHelper.open();

        Bundle extras=getIntent().getExtras();
        if (extras !=null) {
            userId=extras.getLong("id");
        }
        // якщо 0, то додаємо
        if (userId > 0) {
            // отримуємо елемент по id із бд
            userCursor=db.rawQuery("select*from "+DatabaseHelper.TABLE+"where"+
DatabaseHelper.COLUMN_ID+"=?", new String[]{String.valueOf(userId)});
            userCursor.moveToFirst();
            nameBox.setText(userCursor.getString(1));
            yearBox.setText(String.valueOf(userCursor.getInt(2)));
            userCursor.close();
        } else {
            // ховаємо кнопку видалення
            delButton.setVisibility(View.GONE);
```

```
        }
    }
    public void save(View view){
        ContentValues cv=new ContentValues();
        cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());
        cv.put(DatabaseHelper.COLUMN_YEAR,
Integer.parseInt(yearBox.getText().toString()));
        if (userId > 0) {
            db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID+"="+
String.valueOf(userId), null);
        } else {
            db.insert(DatabaseHelper.TABLE, null, cv);
        }
        goHome();
    }
    public void delete(View view){
        db.delete(DatabaseHelper.TABLE, "_id=?", new String[]{String.valueOf(userId)});
        goHome();
    }
    private void goHome(){
        // закриваємо підключення
        db.close();
        // перехід до головної activity
        Intent intent = new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
        startActivity(intent);
    }
}
```

Вся інша робота з даними буде тією ж, щоб і в минулих темах. В результаті отримаємо на екрані:

12.5 Динамічний пошук по базі даних SQLite

Розглянемо, як ми можемо створити в додатку на Android динамічний пошук по базі даних SQLite. Створимо новий проект з порожньою *MainActivity*. Для цього проекту візьмемо базу даних з минулої теми. Дана база даних називається *cityinfo* і має одну таблицю *users* з трьома полями *_id*, *name*, *age*:

```
CREATE TABLE `users` (
   `_id`  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   `name`  TEXT NOT NULL,
   `year`  INTEGER NOT NULL
);
```

Додамо в проект в Android Studio папку *assets*, а в папку *assets* – щойно створену базу даних. У нашому випадку база даних називається *"cityinfo.db"*. Додамо в проект в одну папку з *MainActivity* новий клас *DatabaseHelper*. Перейдемо до файлу *activity_main.xml*, який визначає візуальний інтерфейс, і змінимо його наступним чином:

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

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Пошук" />
    <ListView
        android:id="@+id/userList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ListView>
</LinearLayout>
```

Отже, у нас буде елемент *ListView* для відображення списку і текстове поле для фільтрації. Тепер змінимо код *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
    DatabaseHelper sqlHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;
    ListView userList;
    EditText userFilter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        userList = (ListView)findViewById(R.id.userList);
        userFilter = (EditText)findViewById(R.id.userFilter);
        sqlHelper = new DatabaseHelper(getApplicationContext());
        // створюємо бд
        sqlHelper.create_db();
    }

    @Override
    public void onResume() {
        super.onResume();
        try {
```

```
        db=sqlHelper.open();
        userCursor=db.rawQuery("select * from "+DatabaseHelper.TABLE, null);
        String[] headers=new String[]{DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};
        userAdapter=new SimpleCursorAdapter(this, android.R.layout.two_line_list_item,
                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);
        // якщо в текстовому полі є текст, то здійснюємо фільтрацію
        // дана перевірка необхідна при переході з однієї орієнтації екрану до іншої
        if(!userFilter.getText().toString().isEmpty())
            userAdapter.getFilter().filter(userFilter.getText().toString());
        // встановлення слухача змін тексту
        userFilter.addTextChangedListener(new TextWatcher() {
            public void afterTextChanged(Editable s) { }
            public void beforeTextChanged(CharSequence s, int start, int count, int after) { }
            // при зміні тексту виконуємо фільтрацію
            public void onTextChanged(CharSequence s, int start, int before, int count) {
                userAdapter.getFilter().filter(s.toString());
            }
        });
        // встановлюємо провайдер фільтрації
        userAdapter.setFilterQueryProvider(new FilterQueryProvider() {
            @Override
            public Cursor runQuery(CharSequence constraint) {
                if (constraint == null || constraint.length() == 0) {
                    return db.rawQuery("select*from" DatabaseHelper.TABLE, null);
                }
                else {
                    return db.rawQuery("select * from "+DatabaseHelper.TABLE+" where"+
DatabaseHelper.COLUMN_NAME+" like ?", new String[]{"%"+constraint.toString()+
"%"});
                }
            }
        });
        userList.setAdapter(userAdapter);
```

```
        }
        catch (SQLException ex){}
    }

    @Override
    public void onDestroy(){
        super.onDestroy();
        db.close();
        userCursor.close();
    }
}
```

Перш за все треба відзначити, що для фільтрації даних в адаптері, нам треба отримати фільтр адаптера, а у цього фільтра виконати метод *filter*():

```
userAdapter.getFilter().filter(s.toString());
```

У цей метод *filter*() передається ключ пошуку. Для текстового поля ми можемо відстежувати зміни вмісту за допомогою слухача:

```
userFilter.addTextChangedListener(new TextWatcher() {
    public void afterTextChanged(Editable s) {
    }
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }
    // здійснюємо фільтрацію при зміні тексту
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        userAdapter.getFilter().filter(s.toString());
    }
});
```

У слухачі *TextWatcher* в методі *onTextChanged* якраз і викликається метод *filter*(), в який передається введена користувачем в текстове поле послідовність символів.

Сам виклик методу *filter*() мало на що впливає. Нам потрібно ще визначити провайдер фільтрації адаптера, які і буде інкапсулювати реальну логіку фільтрації:

```
userAdapter.setFilterQueryProvider(new FilterQueryProvider() {
    @Override
    public Cursor runQuery(CharSequence constraint) {
        if (constraint == null || constraint.length() == 0) {
```

```
            return db.rawQuery("select*from"+DatabaseHelper.TABLE, null);
        }
        else {
            return db.rawQuery("select*from"+DatabaseHelper.TABLE+"where" +
                DatabaseHelper.COLUMN_NAME+"like ?", new String[]{"%"+
constraint.toString()+"%"});
            }
    }
});
```

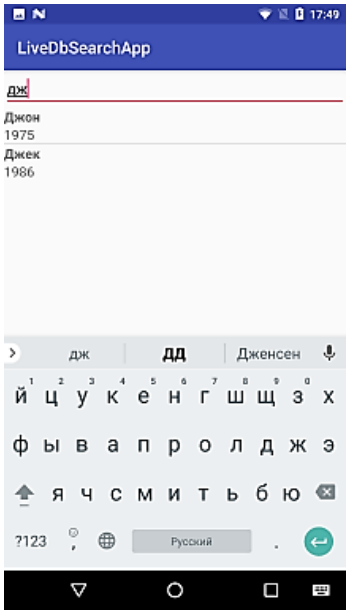
Сутність цього провайдера полягає у виконанні SQL-виразів до БД, а саме конструкцій *"select from"* і *"select from where like"*. Дані найпростіші вирази виконують чутливі до регістру фільтрацію. В результаті адаптар отримує відфільтровані дані.

Слід також відзначити наступний код:

```
if(!userFilter.getText().toString().isEmpty())
    userAdapter.getFilter().filter(userFilter.getText().toString());
```

Даний код нам потрібен при зміні орієнтації (наприклад, з портретної на альбомну). І якщо орієнтація пристрою змінена, але в текстовому полі все ж є деякі текст-фільтри, то виконується фільтрація. Інакше вона б не виконувалася.

Після запуску додатку ми зможемо користуватися фільтрацією даних:



ЛЕКЦІЯ 13. ДІАЛОГОВІ ВІКНА

13.1 DatePickerDialog і TimePickerDialog

В Android існують два діалогових вікна – *DatePickerDialog* і *TimePickerDialog* , які дозволяють вибрати дату і час. Крім установки дати *DatePickerDialog* дозволяє обробити вибір дати за допомогою слухачів *OnDateChangedListener* і *OnDateSetListener*, що дозволяє використовувати обрану дату далі в застосунку. Подібним чином *TimePickerDialog* дозволяє обробити вибір часу за допомогою слухачів *OnTimeChangedListener* і *OnTimeSetListener*.

При роботі з даними компонентами треба враховувати, що відлік місяців в *DatePickerDialog* починається з нуля, тобто січень буде мати номер 0, а грудень – 11. І аналогічно в *TimePickerDialog* відлік секунд і хвилин буде йти з 0 до 59, а годин – з 0 до 23.

Використаємо *DatePickerDialog* і *TimePickerDialog* в додатку. Визначимо наступну розмітку інтерфейсу в *activity_main.xml*:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView android:id="@+id/currentDateTime"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp" />
   <Button android:id="@+id/timeButton"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Изменить время"
      android:onClick="setTime" />
   <Button android:id="@+id/dateButton"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Изменить дату"
      android:onClick="setDate"  />
</LinearLayout>
```

Тут описані дві кнопки для вибору дати і часу та текстове поле, яке відображає вибрані дату і час. І змінимо код *MainActivity*:

```
public class MainActivity extends AppCompatActivity {
   TextView currentDateTime;
   Calendar dateAndTime=Calendar.getInstance();
   @Override
   public void onCreate(Bundle savedInstance) {
      super.onCreate(savedInstance);
      setContentView(R.layout.activity_main);
      currentDateTime=(TextView)findViewById(R.id.currentDateTime);
      setInitialDateTime();
   }

   // показуємо діалоговое вікно для вибору дати
   public void setDate(View v) {
      new DatePickerDialog(MainActivity.this, d,
            dateAndTime.get(Calendar.YEAR),
            dateAndTime.get(Calendar.MONTH),
            dateAndTime.get(Calendar.DAY_OF_MONTH))
            .show();
   }

   // показуємо діалоговое вікно для вибору часу
   public void setTime(View v) {
      new TimePickerDialog(MainActivity.this, t,
            dateAndTime.get(Calendar.HOUR_OF_DAY),
            dateAndTime.get(Calendar.MINUTE), true)
            .show();
   }
   // встановлення початкових дати і часу
   private void setInitialDateTime() {
      currentDateTime.setText(DateUtils.formatDateTime(this,
            dateAndTime.getTimeInMillis(),
            DateUtils.FORMAT_SHOW_DATE | DateUtils.FORMAT_SHOW_YEAR
```

```
                | DateUtils.FORMAT_SHOW_TIME));
    }

    // обробник вибору часу
    TimePickerDialog.OnTimeSetListener t=new TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            dateAndTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
            dateAndTime.set(Calendar.MINUTE, minute);
            setInitialDateTime();
        }
    };

    // обробник вибору дати
    DatePickerDialog.OnDateSetListener d=new DatePickerDialog.OnDateSetListener() {
        public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
            dateAndTime.set(Calendar.YEAR, year);
            dateAndTime.set(Calendar.MONTH, monthOfYear);
            dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth);
            setInitialDateTime();
        }
    };
}
```

Ключовим класом тут є *java.util.Calendar*, який зберігається в стандартній бібліотеці класів Java. У методі *setInitialDateTime*() ми отримуємо з примірника цього класу кількість мілісекунд *dateAndTime.getTimeInMillis*() і за допомогою форматування виводимо на текстове поле.

Метод *setDate*(), що викликається після натискання на кнопку, відображає вікно для вибору дати. При створенні вікна його об’єкту передається обробник вибору дати *DatePickerDialog.OnDateSetListener*, який змінює дату на текстовому полі.

Аналогічно метод *setTime*() відображає вікно для вибору часу. Об’єкт вікна використовує обробник вибору часу *TimePickerDialog.OnTimeSetListener*, який змінює час на текстовому полі. Після запуску додатку, натиснувши на кнопку зміни часу, ми зможемо встановити час:

Для встановлення часу у діалоговому вікні визначена кнопка «Встановити». Аналогічно працює вікно встановлення дати:

13.2 DialogFragment і створення діалогових вікон

Для створення діалогових вікон використовується компонент *AlertDialog* в зв'язці з класом фрагмента *DialogFragment*. Розглянемо їх застосування. Спочатку додамо в проект новий клас фрагмента, який назвемо *CustomDialogFragment*:

```
public class CustomDialogFragment extends DialogFragment {
```

```
    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder.setTitle("Діалоговое вікно").setMessage("Для закриття вікна натисніть
OK").create();
    }
}
```

Клас фрагмента містить всю стандартну функціональність фрагмента з його життєвим циклом, але при цьому успадковується від класу *DialogFragment*, який додає ряд додаткових функцій. І для його створення ми можемо використати два способи:

- перевизначення методу *onCreateDialog*(), який повертає об'єкт *Dialog*;
- використання стандартного методу *onCreateView*().

Для створення діалогового вікна в методі *onCreateDialog*() застосовується клас *AlertDialog.Builder*. За допомогою своїх методів він дозволяє налаштувати відображення діалогового вікна:

- *setTitle()*: встановлює заголовок вікна;
- *setView()*: встановлює розмітку інтерфейсу вікна;
- *setIcon()*: встановлює іконку вікна;
- *setPositiveButton()*: встановлює кнопку підтвердження дії;
- *setNeutralButton()*: встановлює «нейтральну» кнопку, дія якої може відрізнятися від дій підтвердження або скасування;
- *setNegativeButton()*: встановлює кнопку скасування;
- *setMessage()*: встановлює текст діалогового вікна, але при використанні setView даний метод необов'язковий або може розглядатися в якості альтернативи, якщо нам треба просто вивести повідомлення;
- *create()*: створює вікно.

В даному ж випадку діаговое кно просто виводить деякий повідомлення.

Для виклику цього діалогового вікна в файлі *activity_main.xml* визначимо кнопку:

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

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dialog"
        android:onClick="showDialog"/>
</LinearLayout>
```

У коді *MainActivity* визначимо обробник натискання кнопки, який буде запускати діалогове вікно:

```
public class MainActivity extends AppCompatActivity {
   @Override
   public void onCreate(Bundle savedInstance) {
      super.onCreate(savedInstance);
      setContentView(R.layout.activity_main);
   }
   public void showDialog(View v) {
      CustomDialogFragment dialog=new CustomDialogFragment();
      dialog.show(getSupportFragmentManager(), "custom");
   }
}
```

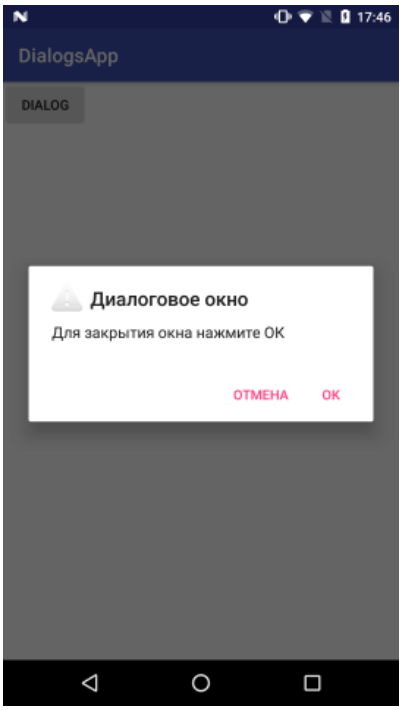
Для виклику діалогового вікна створюється об’єкт фрагмента *CustomDialogFragment*, потім у нього викликається метод *show*(). У цей метод передається менеджер фрагментів *FragmentManager* і рядок – довільний тег.

І поле натискання на кнопку ми зможемо ввести дані в діалогове вікно:

Тепер трохи «кастомізуємо» діалогове вікно:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
   AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
   return builder
         .setTitle("Діалоговое вікно")
         .setIcon(android.R.drawable.ic_dialog_alert)
         .setMessage("Для закриття вікна натисніть ОК")
         .setPositiveButton("OK", null)
         .setNegativeButton("Відмінити", null)
         .create();
}
```

Тут добавляється іконка, яка в якості зображення використовує вбудований ресурс *android.R.drawable.ic_dialog_alert* і встановлюються дві кнопки. Для кожної кнопки можна встановити текст і обробник натискання. В даному випадку для обробника натискання передається *null*, тобто обробник не встановлено. В результаті отримаємо:



Тепер додамо в папку *res/layout* новий файл *dialog.xml*:

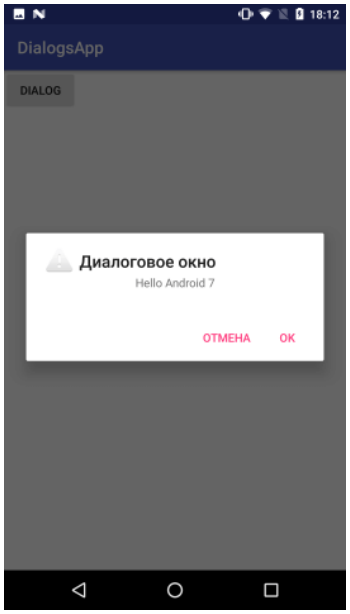
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
<TextView
   android:gravity="center"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Hello Android 7"/>
</LinearLayout>
```

І змінимо код створення діалогового вікна:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
   AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
   return builder
         .setTitle("Диалоговое окно")
         .setIcon(android.R.drawable.ic_dialog_alert)
         .setView(R.layout.dialog)
         .setPositiveButton("OK", null)
```

```
        .setNegativeButton("Отмена", null)
        .create();
}
```

Метод *setView*() встановлює в якості інтерфейсу вікна раніше доданий ресурс *dialog.xml*. При використанні цього методу треба враховувати, що він доступний починаючи з API 21 (*Lollipop*). Тому для його застосування може знадобитися змінити мінімальну версію Android у проекті до 21.



При цьому, як можна побачити на скріншоті, кнопки і заголовок з іконкою не входять до розмітки.

13.3 Передача даних в діалогове вікно

Передача даних в діаговое вікно, як і в будь-який фрагмент, здійснюється за допомогою об’єкта *Bundle*. Так, визначимо в файлі *activity_main.xml* список *ListView*:

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

```
        android:layout_height="match_parent"  />
</LinearLayout>
```

У класі *MainActivity* визначимо для цього списку дані:

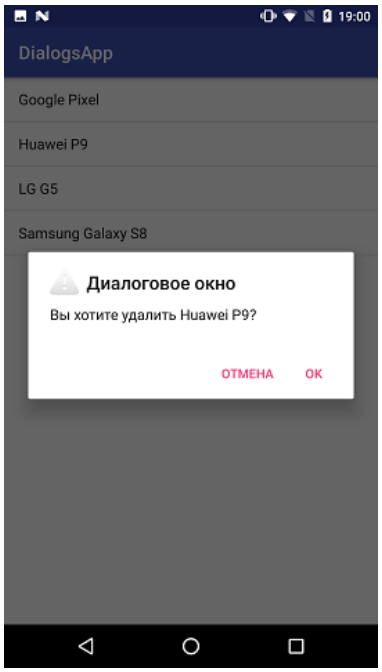
```
public class MainActivity extends AppCompatActivity {
   @Override
   public void onCreate(Bundle savedInstance) {
      super.onCreate(savedInstance);
      setContentView(R.layout.activity_main);
      ListView phonesList=(ListView) findViewById(R.id.phonesList);
      ArrayList<String> phones=new ArrayList<>();
      phones.add("Google Pixel");
      phones.add("Huawei P9");
      phones.add("LG G5");
      phones.add("Samsung Galaxy S8");

      final ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, phones);
      phonesList.setAdapter(adapter);
      phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
         @Override
         public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            String selectedPhone=adapter.getItem(position);
            CustomDialogFragment dialog=new CustomDialogFragment();
            Bundle args=new Bundle();
            args.putString("phone", selectedPhone);
            dialog.setArguments(args);
            dialog.show(getSupportFragmentManager(), "custom");
         }
      });
   }
}
```

У обробнику події натиснення на елемент в списку отримуємо обраний елемент і додаємо його в об’єкт *Bundle*. Далі через метод *dialog.setArguments()* передаємо дані з

Bundle під фрагмент. Тепер визначимо наступний клас фрагмента *CustomDialogFragment*:

```
public class CustomDialogFragment extends DialogFragment {
   @NonNull
   public Dialog onCreateDialog(Bundle savedInstanceState) {

      String phone=getArguments().getString("phone");
      AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
      return builder
            .setTitle("Діалогове вікно")
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setMessage("Бажаєте видалити"+phone+"?")
            .setPositiveButton("OK", null)
            .setNegativeButton("Відмінити", null)
            .create();
   }
}
```

За допомогою методу *getArguments*() отримуємо переданий в *MainActivity* об’єкт *Bundle*. І оскільки був переданий рядок, то для його отримання застосовується метод *getString*(). При натисканні елемент списку буде передаватися в діалогове вікно:



13.4 Взаємодія з Activity

Взаємодія між *Activity* і фрагментом проводиться, як правило, через інтерфейс. Наприклад, в минулій темі *MainActivity* виводила список об'єктів, і тепер визначимо видалення з цього списку через діалогове вікно. Для цього додамо в проект інтерфейс *Datable*:

```
public interface Datable {
   void remove(String name);
}
```

Єдиний метод інтерфейсу *remove()* отримує об'єкт, який видаляється, у вигляді параметра *name*. Тепер реалізуємо цей інтерфейс в коді *MainActivity*:

```
public class MainActivity extends AppCompatActivity
implements Datable{
   private ArrayList<String> phones;
   private ArrayAdapter<String> adapter;
   @Override
   public void onCreate(Bundle savedInstance) {
      super.onCreate(savedInstance);
      setContentView(R.layout.activity_main);
      ListView phonesList=(ListView) findViewById(R.id.phonesList);
      phones=new ArrayList<>();
      phones.add("Google Pixel");
      phones.add("Huawei P9");
      phones.add("LG G5");
      phones.add("Samsung Galaxy S8");
      adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
phones);
      phonesList.setAdapter(adapter);
      phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
         @Override
         public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            String selectedPhone=adapter.getItem(position);
            CustomDialogFragment dialog=new CustomDialogFragment();
            Bundle args=new Bundle();
```

```
            args.putString("phone", selectedPhone);
            dialog.setArguments(args);
            dialog.show(getSupportFragmentManager(), "custom");
        }
    });
  }

  @Override
  public void remove(String name) {
    adapter.remove(name);
  }
}
```

Метод *remove* видаляє з адаптера переданий елемент. Файл *activity_main.xml* як і раніше визначає лише елемент *ListView*:

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

І нарешті визначимо фрагмент *CustomDialogFragment*:

```
public class CustomDialogFragment extends DialogFragment  {
  private Datable datable;

  @Override
  public void onAttach(Context context){
    super.onAttach(context);
    datable = (Datable) context;
  }

  @NonNull
```

```
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        final String phone=getArguments().getString("phone");
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder
                .setTitle("Діалогове вікно")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setMessage("Бажаєте видалити "+phone+"?")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        datable.remove(phone);
                    }
                })
                .setNegativeButton("Отмена", null)
                .create();
    }
}
```

Метод *onAttach*() викликається на початку життєвого циклу фрагмента, і саме тут ми можемо отримати контекст фрагмента, в якості якого виступає клас *MainActivity*. Так як *MainActivity* реалізує інтерфейс *Datable*, то ми можемо перетворити контекст до даного інтерфейсу. Потім в обробнику кнопки ОК викликається метод remove об’єкта *Datable*, який видаляє переданий у фрагмент об’єкта *phone*.

ЛЕКЦІЯ 14. ПУБЛІКАЦІЯ СТВОРЕНОЇ ПРОГРАМИ

14.1 Поняття GooglePlay

Google Play – це сховище і одночасно інтернет-магазин Android-додатків. Додатки можуть бути комерційними (для їх установки потрібно оплата) або безкоштовними (користувач може вільно завантажити додаток). У першому випадку Google Play виступає в якості посередника між розробником і користувачем. У другому – Google Play являє собою звичайне сховище, куди розробники можуть завантажити додатки, а користувачі – завантажити на мобільний пристрій з метою подальшої інсталяції.

Google Play – не панацея. Існують також інші сайти, з допомогою яких можна розповсюджувати ваші програми, наприклад: http://androidapplications.com та http://slideme.org.

Розміщення розробленого додатку на Google Play дозволяє вирішити ряд проблем:

1. поширення програми;
2. оновлення програми (інформацію про нову версію отримають користувачі, що вже купили програму або просто її завантажили - для безкоштовних програм);
3. захист від несанкціонованого поширення;
4. отримання оплати за програму.

14.2 Реєстрація акаунта на Google Play

При реєстрації аккаунта розробника вам доведеться прийняти умови угоди Google Play Developer Distribution Agreement. У ньому досить багато правил і всілякої інформації. Зупинимося лише на найважливіших його моментах:

- ви повинні повністю прийняти умови Угоди. Порушення правил Угоди призведе до «бану» вашого акаунта та видалення ваших продуктів з Google Play;
- ви можете поширювати продукти безкоштовно або за гроші. Якщо ви продаєте програми, то зобов’язані використовувати лише платіжну систему Google Checkout;

- приймаючи умови Google Play, ви погоджуєтеся, що Google буде утримувати 30% вашого прибутку від продажу програми. Крім того, треба буде заплатити ще ряд податків, які залежать від вашої країни;
- якщо протягом 48 годин користувач видалить додаток (якщо воно йому не сподобається), йому будуть повернуті гроші;
- розробникам забороняється збирати будь-які особисті дані користувачів, які завантажили (купили) їх застосування, в процесі використання програми;
- продаючи свій додаток, ви надаєте користувачеві неексклюзивне право використовувати ваш продукт на його пристрої. Але якщо такого короткого угоди мало, ви можете написати окрему угоду користувача (End User Licence Agreement, EULA).

Для реєстрації акаунта перейдемо за посиланням https://play.google.com/apps/publish/signup. Нам буде запропоновано пов'язати наш аккаунт в Google із консоллю розробника (рис. 14.1).

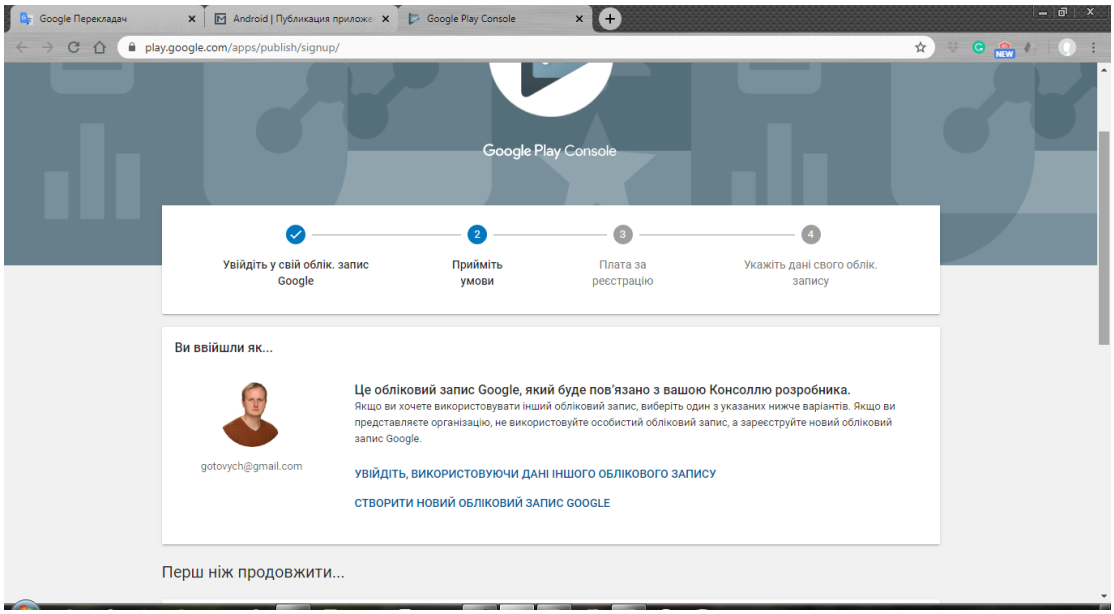


Рисунок 14.1 Консоль розробника Google Play

Тут ми можемо пов'язати поточний аккаунт з аккаунтом розробника, але також можемо і створити новий. При реєстрації треба враховувати, що нам доведеться заплатити реєстраційний збір. Приймемо ліцензійну угоду і натиснемо внизу сторінки на кнопку «Перейти до оплати» (рис. 14.2).

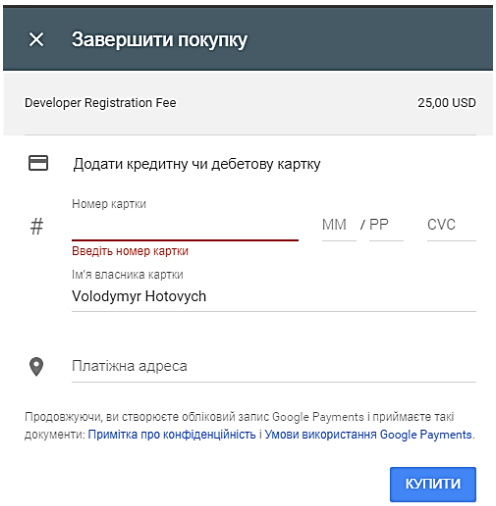


Рисунок 14.2 Вікно оплати за реєстрацію акаунта на Google Play

Для оплати необхідна кредитна або дебетова карта Visa/MasterCard/AMEX/Discover. Після введення персональних даних і відомостей про карту натиснемо на кнопку «Прийняти та продовжити». Після цього нас буде перенаправлено на сторінку заповнення профілю (рис. 14.3).

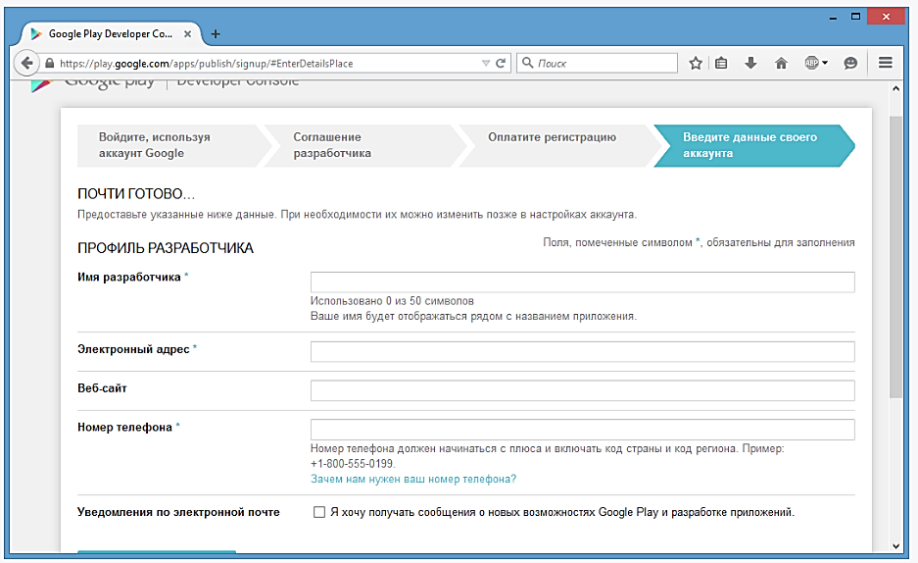


Рисунок 14.3 Сторінка заповнення профіля на Google Play

Введемо тут свої дані і натиснемо на кнопку внизу сторінки. І потім ми вже потрапимо власне в особистий кабінет в консолі розробника (рис. 14.4).

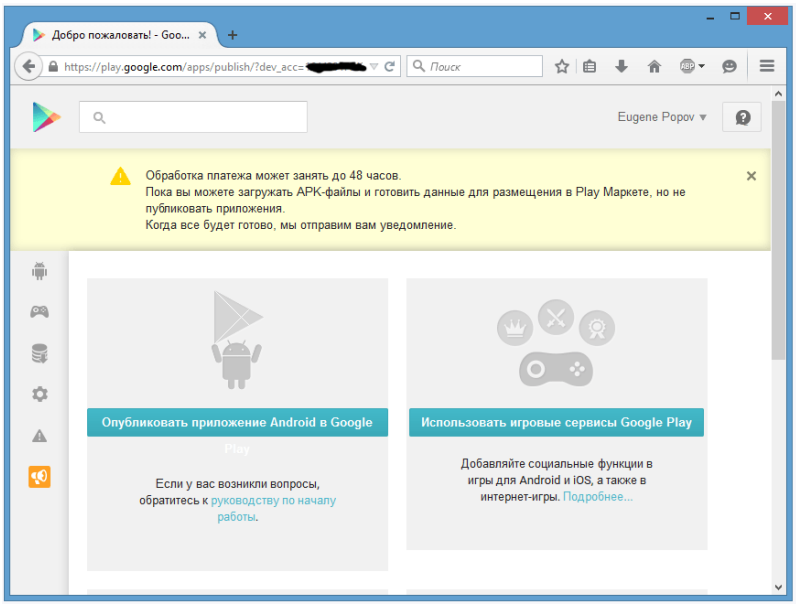


Рисунок 14.4 Особистий кабінет консолі розробника на Google Play

Обробка оплати може зайняти деякий час, тому ми поки будемо обмежені в правах на час обробки. У той же час ми вже можемо завантажити файл apk, повноцінна публікація якого відбудеться після перевірки оплати.

Для завантаження файлу програми натиснемо на кнопку «Опублікувати додаток Android в Google Play». Введемо назву і натиснемо на кнопку «Завантажити APK». Далі відкриється сторінка, на якій ми вже безпосередньо завантажуємо файл програми і вводимо її опис (рис. 14.5).

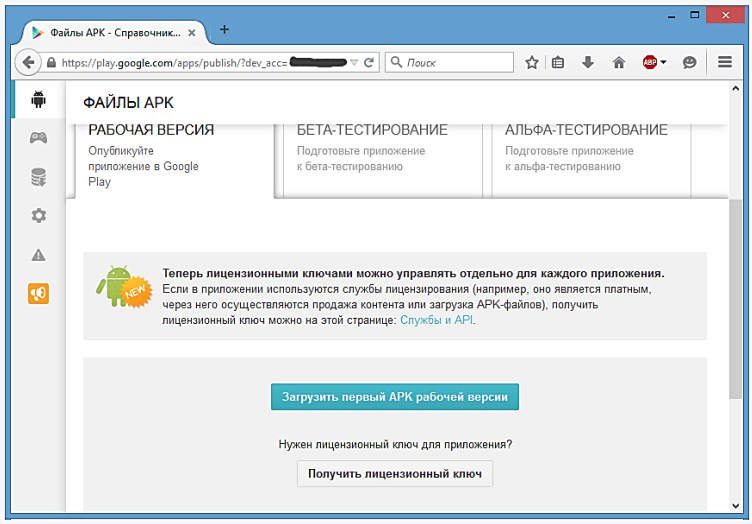


Рисунок 14.5 Сторінка для завантаження apk-файла

14.3 Підготовка додатку до публікації

Перед публікацією додатка слід виконати ряд важливих умов:

1. Тестування на різних пристроях.

Перш за все слід переконатися, що ваш додаток нормально працює на різних смартфонах. Для цього знадобиться кілька пристроїв різних виробників, бажано з різними версіями Android і розширенням екрану. Не варто викладати свій додаток на Google Play, якщо він не протестований хоча б на 3-4 найпопулярніших пристроях від різних виробників.

2. Підтримка різних розширень екрану.

Не забудьте перевірити, як виглядає ваш додаток на екрані, розмір якого відрізняється від розміру екрану емулятора за замовчуванням. Тут все набагато простіше: необхідно створити кілька емуляторів з різними розмірами екрану і запустити додаток на кожному з них.

3. Локалізація.

Щоб розширити аудиторію свого додатка слід виконати локалізацію, тобто, переклад програми на різні мови.

З технічної точки зору виконати локалізацію досить просто. Створіть в каталозі *res* кілька каталогів *values- **: наприклад – *values-en* для англійської мови, *values-de* – для німецької і т. д. Потім скопіюйте в ці каталоги оригінальний файл *strings.xml* з каталогу *res/values*. Локалізацією потрібно займатися, коли програма вже створено, всі помилки виправлені і більше не планується додавати в *strings.xml* нові рядкові ресурси.

Далі в налаштування програми додаєте параметр Мова (Language), що дозволяє вибрати мову. Крім стрічкових ресурсів не забудьте також локалізувати зображення (з каталогів *res/drawable- **), якщо вони містять написи, і меню програми.

4. Піктограма.

Погодьтеся, що комерційний додаток із піктограмою за замовчуванням виглядає, щонайменше, смішно. Змінити піктограму досить просто. Підготуйте файл піктограми і помістіть його в каталог *res/drawable*. Потім змініть атрибут *android: icon* елемента *<application>* у файлі маніфесту: *<application anroid: icon="@ drawable / my_icon" ...*

5. Посилання на магазин.

Щоб перенаправити користувача на GooglePlay (наприклад, для купівлі повноцінної версії програми або для покупки іншої програми), використовуйте URI *market://*, наприклад:

```
Intent i=new Intent(Intent.ACTION_VIEW,
Uri.parse("market://search?q:імя_моєї_програми"));
startActivity(i);
```

6. Підготовка APK-файлу.

Загальні дії

За замовчуванням в процесі налагодження і створення додатка файл apk створюється в папці *Назва_проекту\app\build\outputs\apk*. За замовчуванням файл називається *app-debug.apk* і являє debug-версію додатка. Та для повноцінно публікації даного файлу може виявитися недостатньо. І нам ще додатково треба провести певну підготовку проекту до релізу. Спершу слід вказати в файлі маніфесту в елемента <*manifest*> атрибути *android: versionCode* і *android: versionName*. Також в файлі маніфесту елемент <*application*> не повинен містити атрибута *android: debuggable*.

На даному етапі можна встановити іконку для додатку, яка буде відображатися на екрані гаджета, назву програми (атрибут *android: label*), а також можна задати умви ліцензійної угоди.

У файлі маніфесту також слід вказати назву пакета (атрибут *package* елемента <*manifest*>), яка буде використовуватися для додатка надалі. За замовчуванням при розробці в Android Studio назва пакетів додатків починається із *com.example*. Не варто залишати дану назву, бо назва пакета буде служити в якості унікального ідентифікатора вашого додатка. Назва пакета на початку файлів Java-коду також повинна відповідати назві пакету з файлу маніфесту.

Задання вимог

На етапі підготовки до релізу також можна встановити вимоги до API. Наприклад, наш додаток характеризується певною мінімальною версією ОС Android, тому ми можемо встановити в файлі маніфесту відповідні атрибути в елемента <*uses-sdk*>:

- *android:minSdkVersion* – мінімальна версія Android
- *android:targetSdkVersion* – оптимальна версія API
- *android:maxSdkVersion* – максимальная версія системи

Підписування додатку

Додаток для Android повинен підписуватися сертифікатом, завдяки якому можна ідентифікувати автора програми. Коли ми тестуємо додаток, встановлюючи його через Android Studio на пристрій, то він підписується автоматично. Але для створення реліз-версії нам треба зробити ряд додаткових дій.

При створенні сертифіката слід пам’ятати, що при оновленні програми система буде порівнювати сертифікати старої і нової версії. І оновлення відбуватиметься, якщо сертифікати обох версій співпадатимуть. Але якщо нова версія буде підписана новим сертифікатом, то додаток буде розцінюватися як абсолютно новий, ніяк не пов’язаний зі старою версією і такий, що представляє абсолютно іншу програму. У цьому випадку щоб його встановити, нова версія повинна буде мати відмінну від старої назву пакета.

В середовищі Android Studio виберемо в меню пункт *Build->Generate SignedAPK*. Після цього нам відкриється вікно майстра:

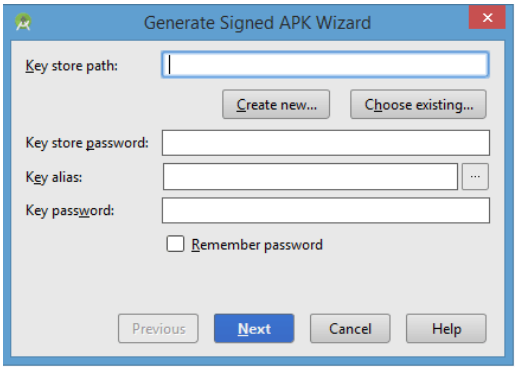


Рисунок 14.6 Вікно майстра генерації підписаного apk

Після натискання на кнопку *Create new...* нам відкриється вікно створення ключа:

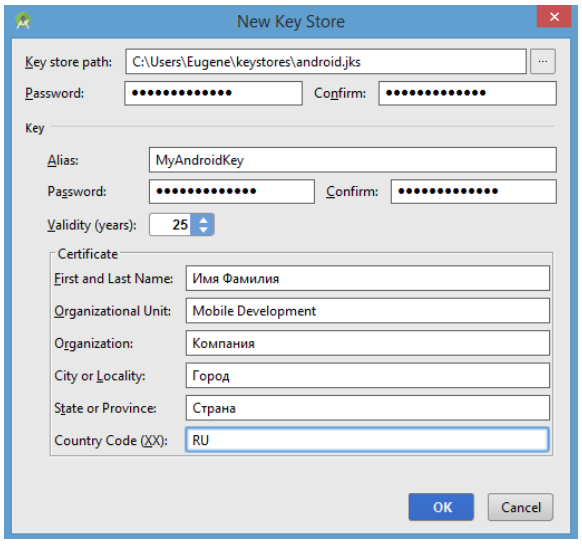


Рисунок 14.7 Вікно створення ключа

Введемо в поле *Key store path* шлях до файлу сетифікату, який буде створений. Якщо зазначеної папки не існує, то її треба створити.

В поле *Password / Confirm* вказуємо пароль.

В полі *Alias* вказуємо псевдонім. Можна поставити довільну назву.

В полі *First and Last Name* вписуємо ім'я і прізвище. Далі вказуємо підрозділ, організацію, місто, країну і код країни.

В кінці натискаємо OK. Після цього автоматично оновиться перше вікно:

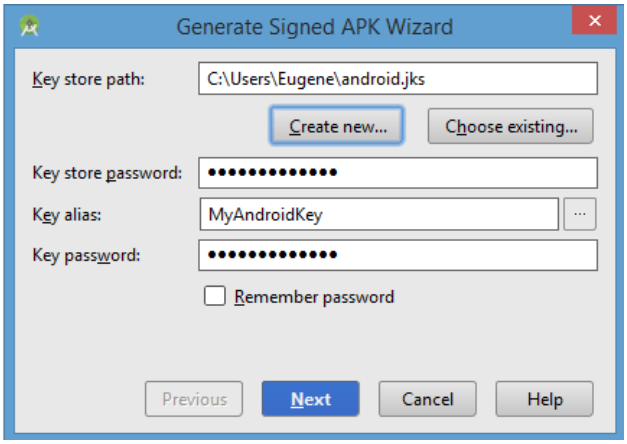


Рисунок 14.8 Вікно майстра генерації підписаного apk із введеними параметрами

Далі натискаємо на кнопку *Next*:

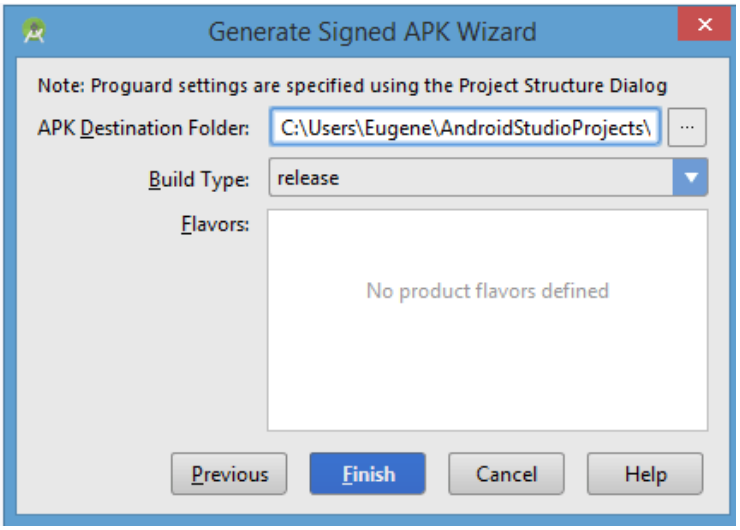


Рисунок 14.9 Вікно майстра генерації підписаного apk

І нарешті, останнє вікно покаже нам шлях до каталогу, де буде знаходитися підписий apk додатку в release-версії. Натиснемо на *Finish*. Тепер за вказаним шляхом можна буде знайти підписаний apk, який матиме назву *app-release.apk*.

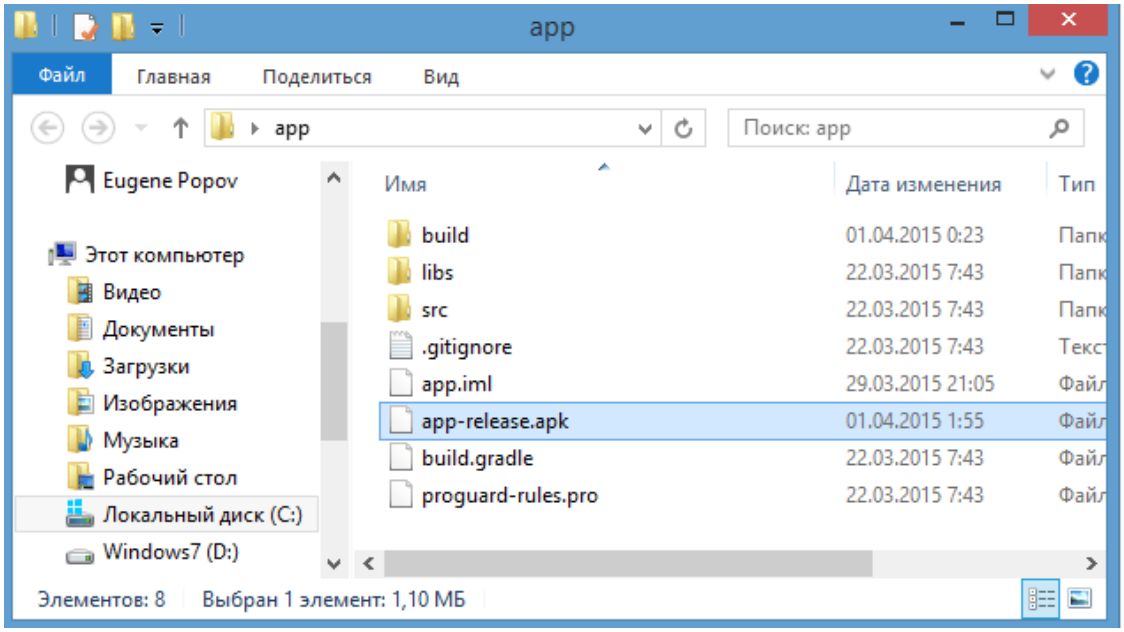


Рисунок 14.10 Вікно, яке вказує шлях до згенерованого підписаного apk

Ми можемо перейменувати даний apk файл, зберігши його розширення і викласти в Google Play або ж відразу завантажити на мобільний пристрій. Після завантаження на телефон/планшет достатньо буде натиснути на нього, і за допомогою стандартного інсталятора пакетів додаток буде встановлено. Також треба враховувати, що якщо ми встановлюємо додаток не з Google Play, то в налаштуваннях треба дозволити установку з інших джерелах: *Безпека->Невідомі джерела (Дозволити встановлення додатків з інших джерел)*.

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. John Horton. Android Programming for Beginners: Build in-depth, full-featured Android 9 Pie apps starting from zero programming experience, 2nd Edition.
2. Голощапов А. Л. Google Android: программирование для мобильных устройств. СПб. : БХВПетербург, 2011. 448 с.
3. Гриффитс Дон, Гриффитс Дэвид Head First. Программирование для Android. СПб.: Питер, 2016. 704 с.
4. Дейтел П., Дейтел Х., Уолд А. Android для разработчиков. 3-е изд. СПб.: Питер, 2016. 512 с.
5. Колисниченко Д. Н. Программирование для Android 5. Самоучитель. СПб.: БХВ-Петербург, 2015. 303 с.
6. Харди Б., Филлипс Б., Стюарт К., Марсикано К. Android. Программирование для профессионалов. 2-е изд. – СПб.: Питер, 2016. 640 с.
7. Офіційна документація для розробників під ОС Android. URL : https://developer.android.com/docs.
8. Android Tutorial. URL : https://www.tutorialspoint.com/android/index.htm.
9. Start Android – учебник по Android для начинающих и продвинутых. URL : https://startandroid.ru/ru/uroki/vse-uroki-spiskom.html.
10. Программирование под ОС Андроид. URL : https://metanit.com/java/android.

ДЛЯ НОТАТКІВ