МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Сучасні мобільні операційні системи

Лабораторний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за спеціальністю 126 Інформаційні системи та технології

Укладач: Б. Я. Корнієнко

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензент *Ладієва Л. Р.*, канд. техн. наук, доцент,
доцент кафедри технічних та програмних засобів автомаитзації,
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Відповідальний
редактор *Ролік О.І.,* докт. техн. наук, професор

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № _7_від _27_ 04_____2023 р.)
за поданням Вченої ради Факультету інформатики та обчислювальної техніки
(протокол № _11__від _24_ 04_ 2023 р.)

Навчальний посібник розрахований на одержання студентами практичних навичок з програмування Android-додатків і складається з 6 лабораторних робіт, що охоплюють широкий спектр розробки. В навчальному посібнику приділено увагу основним підходами до програмування додатків під операційну систему Android, особливостям середовища програмування, основним елементам інтерфейсу, інструментам і середовищам програмування, основам проектування мобільних додатків.

Містить теоретичні положення, порядок виконання лабораторних робіт, завдання та список літератури з дисципліни «Сучасні мобільні операційні системи».

Для студентів всіх форм навчання, які навчаються за спеціальністю 126 «Інформаційні системи та технології» факультету інформатики та обчислювальної техніки КПІ ім. Ігоря Сікорського.

Реєстр. № НП 22/23-657. Обсяг 2,1 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Сучасне суспільство поступово переходить від електронного бізнесу до мобільного. І якщо електронні технології забезпечують доступність і повноту інформації, то мобільні технології своїм першочерговим завданням вважають забезпечення своєчасності інформації та її релевантність. Мобільні технології в даний час активно розвиваються. При цьому можна виділити три тенденції в цьому процесі. По-перше, мобільні телефони, які набули великої популярності у користувачів, отримують все більше можливостей виконувати обчислення (смартфони), що дозволяють застосовувати їх не тільки для голосового зв'язку, а як невеликих комп'ютерів. По-друге, мініатюризація комп'ютерів призвела до створення кишенькових персональних комп'ютерів (КПК), які користувач може носити з собою і в автономному режимі використовувати програми. І, по-третє, активно розвиваються технології та інфраструктура бездротового зв'язку, які дозволяють обмінюватися даними між кишеньковими персональними комп'ютерами і стаціонарними інформаційно-комунікаційними мережами як в рамках локальних (наприклад, технологія Wi-Fi), так і глобальних (технології GPRS, 3G, 4G) комп'ютерних мереж. Все це відбувається на тлі зниження вартості, а значить підвищення доступності мобільних пристроїв.

Такий швидкий розвиток мобільних пристроїв і бездротових технологій зв'язку створює можливість надання їх користувачам інформаційних і обчислювальних послуг в будь-якому місці, де є потреба в їх результатах. З'являється можливість реалізовувати надання інформаційних сервісів будь-яким користувачам, в будь-який час.

В даний час частка мобільних пристроїв, на платформі Android становить більше 80%. Дані пристрої можуть використовуватися не тільки в цілях комунікації і розваг, а і для виконання професійних завдань. Залежно від типу мобільного пристрою і сфери використання, можуть бути розроблені різні додатки. Функціональні можливості простих телефонів дозволяють

організувати ефективний зворотний зв'язок зі споживачем з використанням SMS-розсилки. Додатки, орієнтовані на смартфони, можуть виступати як мобільні клієнти корпоративних мереж. Потреба в різних категоріях мобільних додатків буде постійно збільшуватися, а функціональність мобільних пристроїв - розширюватися. Тому розробка додатків для мобільних пристроїв ще довгий час буде перспективним напрямком розвитку інформаційних технологій.

У зв'язку з розвитком мобільного бізнесу і мережних сервісів актуальним стає навчання фахівців сучасним мобільним операційним системам. Метою даного посібника є систематизація, опис та навчання студентів сучасним мобільним операційним системам. У посібнику розглянуто питання сучасних мобільних операційних систем та їх реалізація для смартфонів орієнтованих на платформу Android.

Лабораторні роботи розраховані на одержання студентами практичних навичок з програмування Android додатків і складається з 6 лабораторних робіт, що охоплюють широкий спектр розробки. Кожна лабораторна робота містить короткі теоретичні відомості і практичні завдання, які студент повинен виконати індивідуально. Лабораторні роботи орієнтовані на студентів, які мають початковий рівень підготовки в галузі об'єктно-орієнтованого програмування, інформаційних технологій, володіють основами проектування, мають початкові навички програмування Java додатків.

В результаті виконання лабораторних робіт студенти ознайомляться з основними підходами до програмування додатків під операційну систему Android, з особливостями середовища програмування, з основними елементами інтерфейсу; вивчать інструменти і середовища програмування, основи проектування мобільних додатків, структуру Android-додатків, що використовується при програмуванні зовнішніх ресурсів; одержать навички роботи з файлами, базами даних, програмними інтерфейсами, що забезпечують функції телефонії, SMS, Wi-Fi, способами створення фонових

служб, сигналізації і підключення механізму повідомлень, елементами об'єктно-орієнтованого аналізу і дизайну в Android додатках, принципами роботи з файловою системою, програмуванням додатків на мові Java, використанням SDK Android та розробці інтерфейсу.

ЛАБОРАТОРНА РОБОТА 1

ACTIVITY - РОБОТА З ЕЛЕМЕНТАМИ ЕКРАНУ

Мета роботи: знайомство з інтерфейсом середовища програмування та вивчення структури проекту.

Теоретичні положення

Мобільний додаток являє собою спеціально розроблене під конкретну мобільну платформу (Android, iOS, Windows Phone) програмне забезпечення. Мобільний додаток розробляють на мові високого рівня і компілюють в код операційної системи, що дає максимальну продуктивність і функціональність. Мобільні додатки поширюються через крамниці (AppStore, Windows Store, Google Play), істотно розширюють способи монетизації бізнесу в порівнянні з веб-додатками, однак характеризуються високою вартістю і тривалим часом розробки.

Сучасні мобільні додатки створюються для лінійки мобільних пристроїв: смартфонів, планшетів, електронних книг, цифрових програвачів, годинників, ігрових приставок, нетбуків, смартбуків, окулярів, а також для телевізоров (тим самим виходячи за рамки виключно мобільних пристроїв). Роботу таких пристроїв забезпечує мобільна операційна система (мобільна платформа), що поєднує в собі функціональність операційної системи для персонального комп'ютера з функціями для мобільних і кишенькових пристроїв: сенсорний екран, стільниковий зв'язок, Bluetooth, Wi-Fi, GPS-навігація, камера, відео - камера, розпізнавання мови, диктофон, музичний плеєр, NFC і інфрачервоне дистанційне керування.

Найбільш поширені операційні системи для мобільних пристроїв: Android (платформа з відкритим вихідним кодом на основі ядра Linux і власної реалізації віртуальної машини Java від Google), iOS (операційна система компанії Apple, заснована на мікроядрі Mach і використовується в смартфонах iPhone), Windows Phone (розробка компанії Microsoft).

Розглянемо питання проектування інтерфейсів і розробки мобільних додатків під платформу Android, що є найбільш поширеною в світі. Архітектуру Android прийнято ділити на рівні ядра, бібліотек і середовища виконання, каркаса додатків,

власне додатків (рис. 1.1).

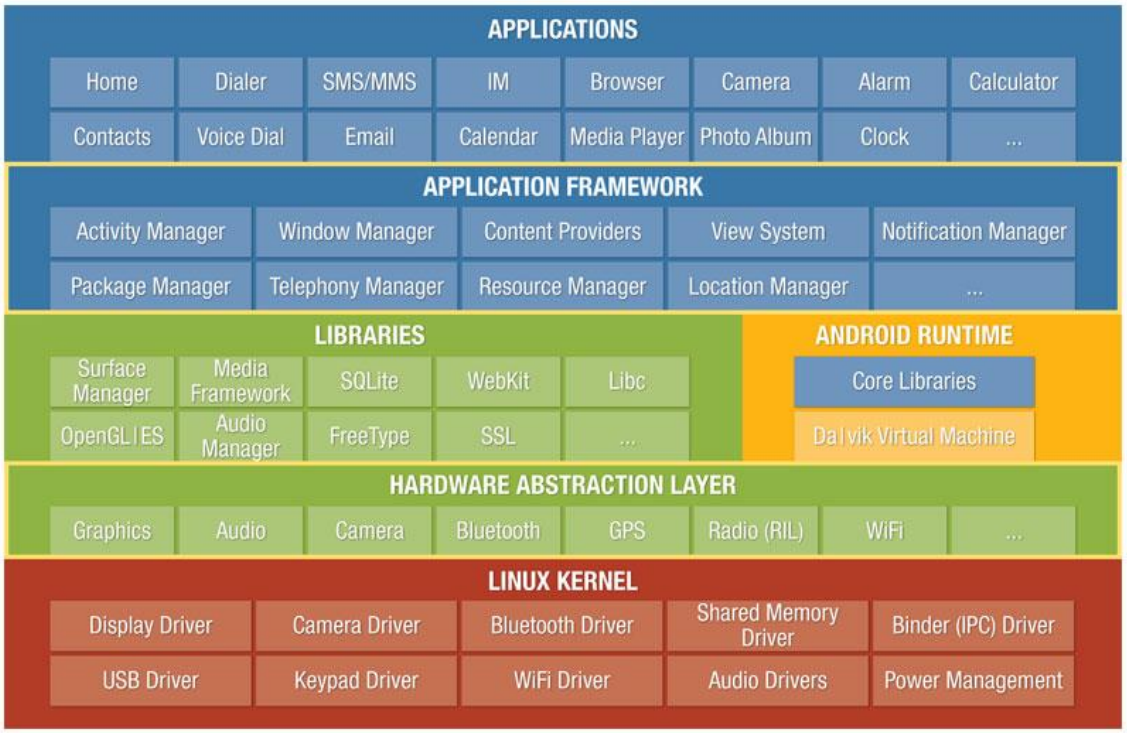


Рис.1.1. Архітектура мобільної платформи Android

Ядро Linux забезпечує функціонування системи і відповідає за безпеку, управління пам'яттю, енергосистемою і процесами, а також надає мережевий стек і модель драйверів.

Набір бібліотек (Libraries) призначений для забезпечення найважливішого базового функціоналу додатків і відповідає за підтримку файлових форматів, кодування і декодування інформації (наприклад, цифровий звук і відео), відображення графіки, підтримку веб-компонентів (WebKit), SQL-СУБД (SQLite) і стандартної для Linux-систем функціональності бібліотек C. На цьому ж рівні розташовується робоче середовище Android (Android Runtime). Кожна програма в операційній системі Android запускається у власному примірнику віртуальної машини Dalvik. Таким чином, всі процеси, працюють ізольовано від операційної системи і один від одного. Завдяки цьому здійснюється захист ядра операційної системи від можливого пошкодження з боку інших її складових.

Рівень каркаса додатків (Application Framework) включає основні служби

Android для управління життєвим циклом додатків, пакетами, ресурсами тощо. Програміст має повний доступ до тих же API (Application Programming Interface), які використовуються основними додатками. Архітектуру цих додатків розроблена з метою спрощення багаторазового використання компонентів. Будь-яка програма може використовувати можливості базових додатків і, відповідно, будь-який інший сторонній додаток може використовувати можливості вашої програми (з урахуванням встановлених дозволів).

Рівень додатків (Applications) включає стандартні програми Android (браузер WebKit, календар Google, клієнт Gmail, додаток Gmaps, SMS-месенджер і e-mail клієнт), а також додатково завантажені програми (з крамниці Android). Android ворізняє основні програми і стороннє програмне забезпечення, в зв'язку з чим ключові компоненти, такі як набір номера, робочий стіл або поштовий клієнт Gmail, можна замінити альтернативними аналогами.

Всього в Android-додатках існує чотири типи компонентів з рівня каркаса додатків: діяльність (Activity), служба (Service), приймач широкомовних намірів (Broadcast Receiver), контент-провайдер (Content Provider). Взаємодія компонентів здійснюється за допомогою об'єктів Intent.

Activity має візуальний інтерфейс для програми - вікно. Activity може також використовувати додаткові вікна, наприклад спливаюче діалогове вікно. Вся діяльності реалізуються як підклас базового класу Activity.

Компонент Service не має візуального інтерфейсу користувача і виконується в фоновому режимі протягом невизначеного періоду часу, поки не завершить свою роботу. Service, як правило, потрібен для тривалих операцій або для забезпечення роботи віддалених процесів, але в загальному випадку це просто режим, який функціонує, коли додаток не в фокусі. Прикладом такого процесу може стати прослуховування музики в той час, коли користувач робить щось інше, або отримання даних по мережі без блокування поточної активності. Додатки можуть підключатися до компоненту Service або запускати його, якщо він не запущений, а також зупиняти вже запущені компоненти.

Broadcast Receiver використовується для відстеження зовнішніх подій і реакції на них. Додаток може мати кілька компонентів Broadcast Receiver, щоб відповісти на будь-які оголошення, які воно вважає важливими. При цьому кожен компонент Broadcast Receiver буде визначати код, який виконається після виникнення конкретної зовнішнього події. За допомогою класу Notification Manager можна повідомити користувачеві інформацію, що вимагає його уваги. Прикладом сповіщень може бути сигнал про те, що інформація завантажена пристрій і доступна до використання.

Content Provider робить певний набір даних, що використовуються додатками, доступним для інших додатків. Дані можуть бути збережені в файловій системі, базі даних SQLite, мережі або будь-якому іншому місці, до якого додаток може мати доступ. Контент-провайдери для безпечного доступу до даних використовують механізм дозволів. За допомогою Content Provider інший додаток може запитувати дані і, якщо виставлені відповідні дозволи, змінювати їх. Наприклад, система Android містить Content Provider, який керує інформацією користувача про контакти.

Intent - спеціальні класи в коді програми, які визначають і описують запити додатків на виконання будь-яких операцій. Наміри додають шар, що дозволяє оперувати компонентами з метою їх повторного використання і заміщення. У деяких випадках це може бути дуже потужним засобом інтеграції додатків. Головна особливість платформи Android полягає в тому, що один додаток може використовувати елементи інших програм за умови, що ці програми дозволяють використовувати свої компоненти. При цьому ваш додаток не включає код іншого додатка або посилання на нього, а просто запускає потрібний елемент іншої програми [1-5].

Створення проєкту в Android Studio

Розробку додатків для платформи Android виконують на мові Java. З 2013 року розробка виконується в офіційному середовищі компанії Google - Android Studio, заснованої на IntelliJ IDEA від JetBrains.

Для створення нового проекту в Android Studio вибираємо на екрані запрошення опцію New Project (для вже відкритого проекту в меню File слід вибрати New Project). У вікні Create New Project (рис. 1.2) заповнюємо поля імені додатки (Application name), яке буде відображатися для користувачів, і кваліфікатора (Company Domain), який буде додаватися до імені пакету. Повна назва проєкту (Package name) формується відповідно до правил іменування пакетів в Java. За необхідності змінюємо папку розташування проекту (Project location).

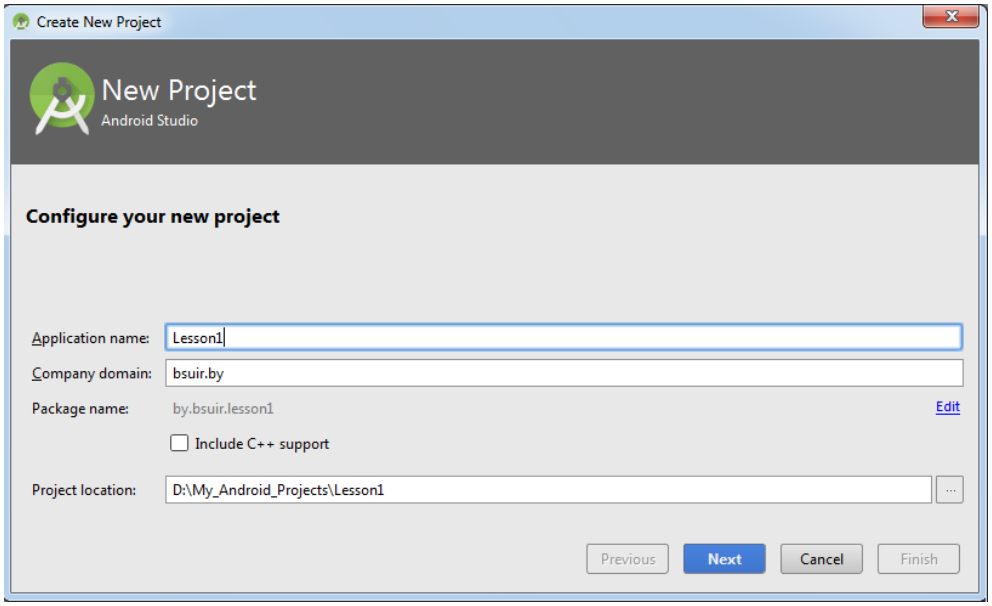


Рис. 1.2. Конфігурація нового проєкту в Android Studio

Далі вказуємо мінімальну версію SDK платформи (рис. 1.3) - найбільш ранню версію Android, яку буде підтримувати додаток. Minimum SDK не може бути вище версії API тієї мобільної платформи, на якій планується тестування і подальше використання програми.

У вікні Add an activity to Mobile в якості основи майбутнього програми вибираємо Empty Activity, в результаті чого буде створено програму з одним Activity.

У формі Customize the Activity (рис. 1.4) згенеровані імена файлів класу Activity (Activity Name), графічної розмітки (Layout Name). Рекомендується

залишити дані імена без змін, оскільки для запускає Activity вони є загальноприйнятими.

Після цього Android Studio створює новий проєкт із зазначеними характеристиками. Структура проєкту (рис. 1.5) включає: файл AndroidManifest.xml (маніфест додатка), папку java (містить весь код програми), папку res (використовується для файлів-ресурсів різного типу: res / drawable / - для зображень (PNG, JPEG і т. д.); res / layout / - для xml-файлів графічної розмітки; res / menu / - для xml-файлів меню; res / values / - для строкових ресурсів, стилів).

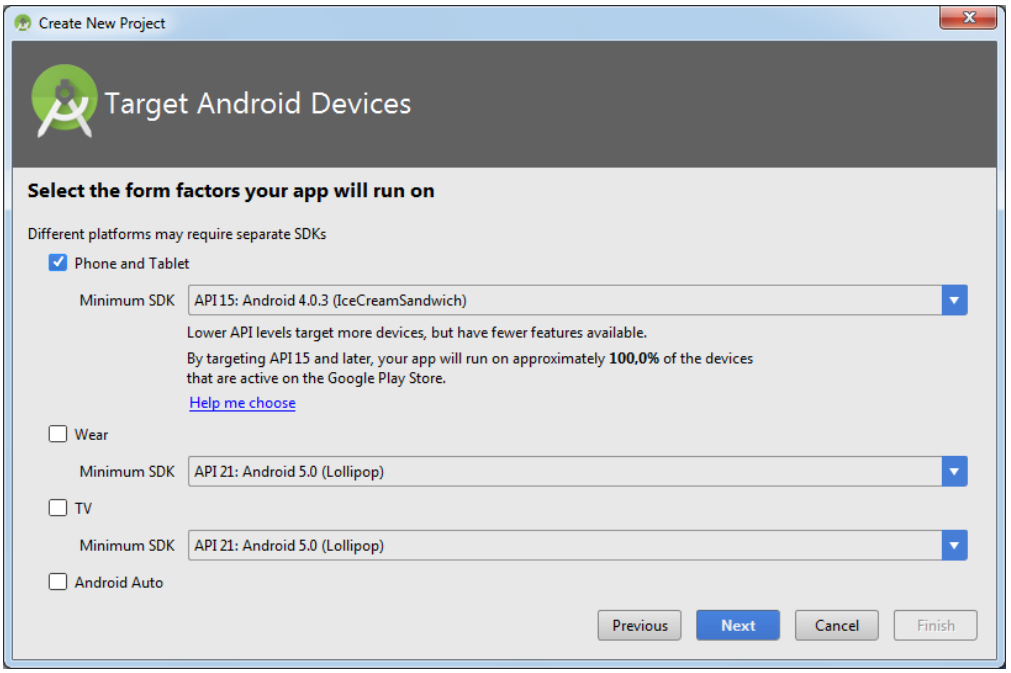


Рис. 1.3. Вказуємо мінімальну версію SDK для нового проєкту в Android Studio

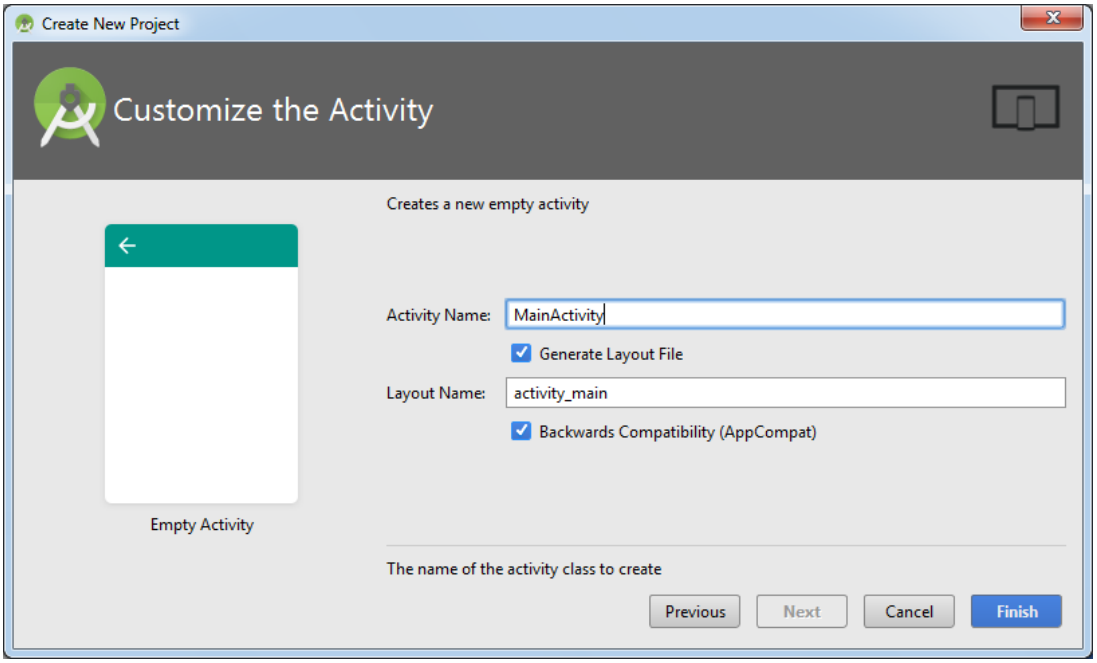


Рис. 1.4. Кастомізація імен для нового проєкту в Android Studio

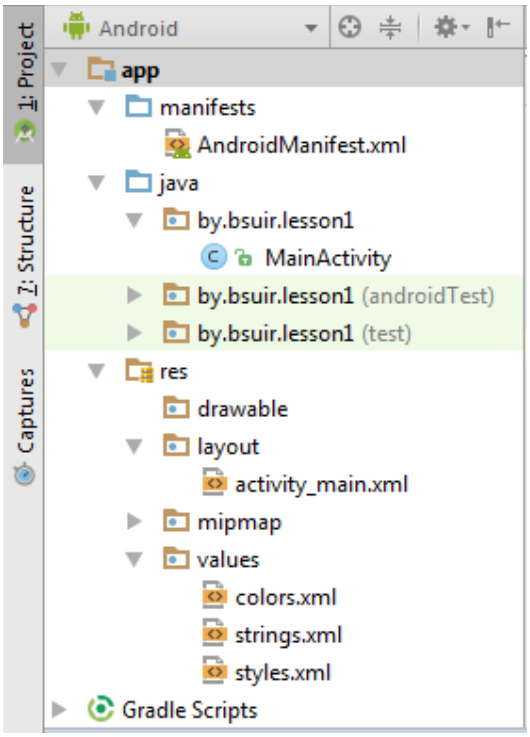


Рис. 1.5. Структура проєкту в Android Studio

Графічна реалізація Activity

Основу Android-додатків становить Activity - робоче вікно. У конкретний

момент часу зазвичай відображається одне Activity і займає весь екран. Робота з набором вікон здійснюється шляхом перемикання різних Activity. Як приклад можна розглянути поштовий додаток: в ньому одне Activity - список листів, інше - перегляд листів, третє - налаштування скриньки.

З програмної точки зору Activity є файлом графічної розмітки (activity_main.xml на рис. 1.5) і відповідним java-класом (MainActivity на рис. 1.5), що реалізує запуск даного Activity з необхідною графічною реалізацією і наступною обробкою подій.

Графічна реалізація Activity формується з різних компонентів (кнопка, поле введення, тощо), що називається View (рис. 1.6).

Графічна реалізація кожного Activity у вигляді необхідного набору і взаємного розташування View-елементів зберігається в xml-файлі папки layout. Даний файл графічної реалізації прописується в відповідному java-класі. При запуску програми Activity читає цей файл і відображає його вміст.

Після створення проєкту файл activity_main.xml відкривається за замовчуванням (рис. 1.7) в режимі конструктора (вкладка Design), крім якого існує відповідне xml-опис графічної реалізації (вкладка Text).

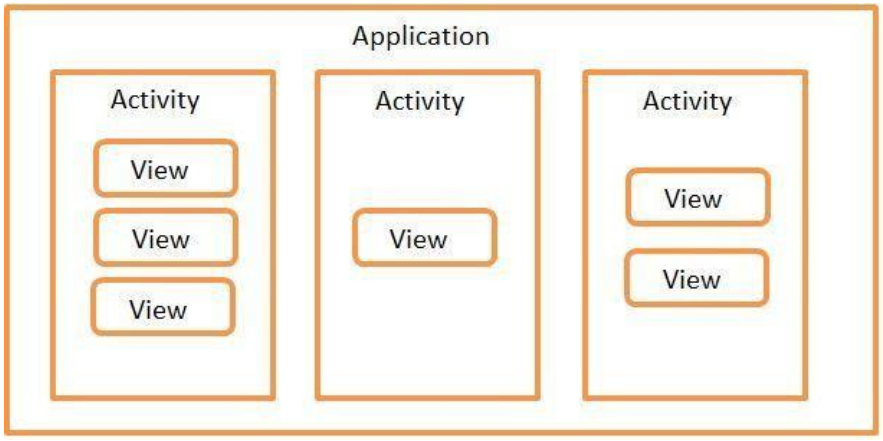


Рис. 1.6. Реалізація додатку Android як набору вікон (Activity) з View-компонентами

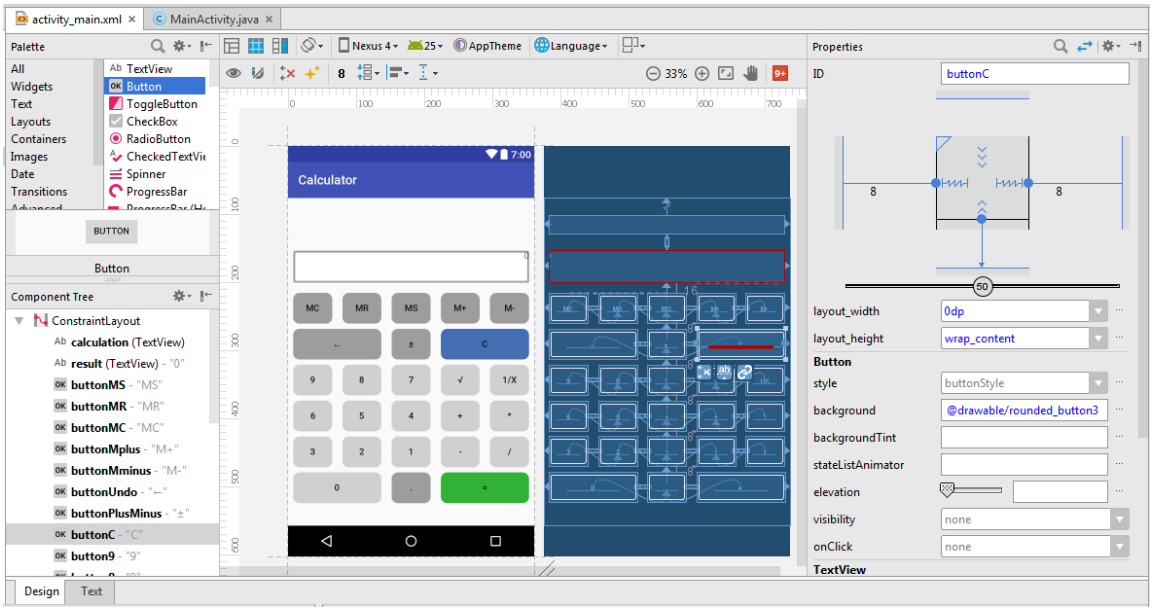


Рис. 1.7. Графічна реалізація Activity, вкладка Design

У центрі вкладки Design за замовчуванням розташовані два графічних редактори, що являють собою екран мобільного пристрою в режимі Design і режимі Blueprint. Режим Blueprint спеціально розроблений для зручності розташування елементів і налаштування їх взаємозв'язків.

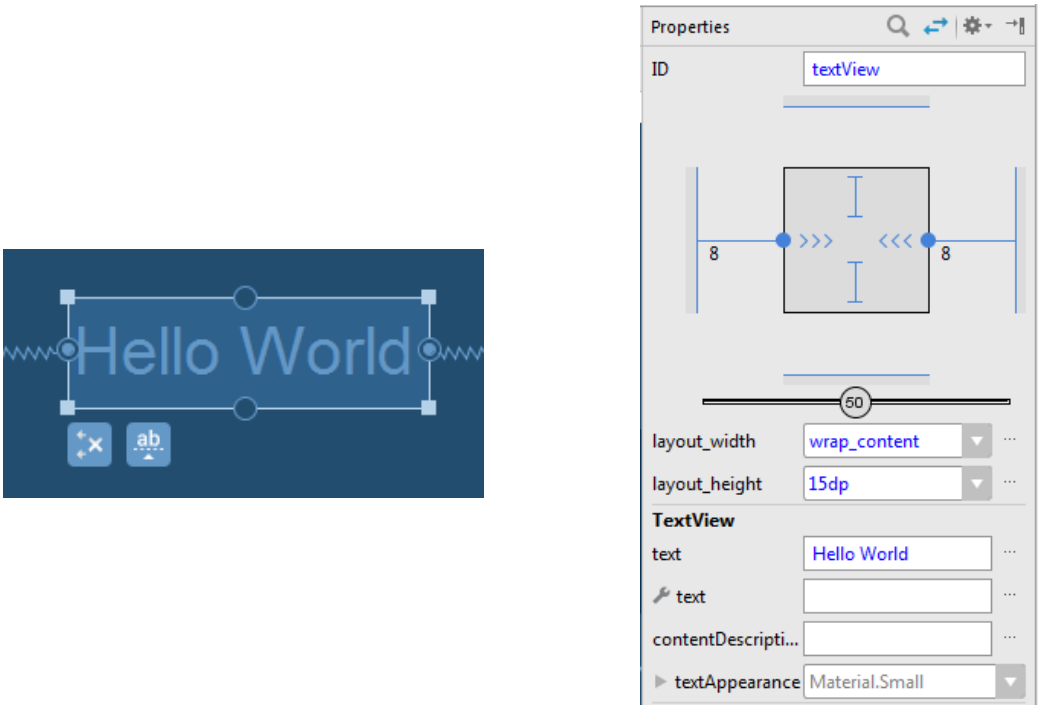
У лівій частині вкладки Design відображаються вікно Palette зі списком всіх можливих віджетів і вікно Component Tree з деревом компонентів, що використовуються в даному графічному поданні.

У правій частині вкладки вкладки Design відображається вікно Properties з набором властивостей для кожного обраного елемента.

Для розміщення View-компонентів використовуються спеціальні контейнери (View Group), звані Layout. Layout бувають різних типів (Linear Layout, Relative Layout, Frame Layout, Table Layout, Constraint Layout тощо). Відповідають за те, як будуть розташовані їхні дочірні View-компоненти на екрані (таблицею, рядком, стовпцем). Android Studio за замовчуванням використовує Constraint Layout як кореневого контейнера для створення розмітки екрану і розміщення компонентів. Даний контейнер має широкий спектр можливостей, що дозволяє реалізовувати складне розташування елементів на екрані.

Для додавання на екран необхідного компонента необхідно знайти його в списку Palette і перемістити мишкою на екран в режимі Design або режимі Blueprint. Після цього компонент з'явиться на обох екранах, а також у вікні Component Tree.

Квадратні опорні точки в кутах компонента (рис. 1.8, а) дозволяють змінювати його розміри.



а *б*

Рис. 1.8. Приклад зображення компонента TextView і його властивостей *а* - зображення компонента TextView в режимі BluePrint; *б*- властивості компонента TextView у вікні Properties

Круглі опорні точки (рис. 1.8, а), розташовані по сторонам віджета, дозволяють створювати прив'язки (constraints) до сторін контейнера або інших компонентів та управляти відступами від країв екрану і інших компонентів.

У вікні Properties (рис. 1.8, б) знаходиться схематичне зображення компоненту, а також зазначені його властивості.

Базовим параметром кожного компонента є id - ідентифікаційний номер. Необхідно давати компонентам унікальні та осмислені імена, щоб з ними в

подальшому було зручно працювати з java-коду. Кожен компонент характеризується шириною (layout_width) та висотою (layout_heigth).

Основні поняття Android проєкту

Структура проєкту:

Src - «вихідний код» додатка (java-класи);

Assets – директорія, може використовуватися для збереження raw-файлів;

Gen - сховище генерації системних файлів, зокрема, тут розташовується файл R.java, в якому зберігаються ідентифікатори всіх ресурсів, створюваних в проєкті;

Libs - різні бібліотеки, які використовуються додатком res-ресурси проекту;

AndroidManifest.xml – файл опису проєкту (підтримувані версії SDK, версія програми тощо);

Project.properties - файл, що включає налаштування проекту, такі як build target.

Ресурси проєкту:

anim / містить XML файли, компільовані в анімаційні об'єкти;

color / містить XML файли, що описують кольори;

drawable / містить растрові файли (PNG, JPEG, orGIF), 9-Patch файли, і XML файли, що описують Drawableshapes або Drawableobjects;

layout / містить XML файли, що описують макети екрану;

menu / містить XML файли, що визначають меню програми;

raw / для зберігання довільних файлів;

values / містить XML файли, компільовані в множину видів ресурсів (strings.xmli);

xml / містить XML файли, що конфігурують компоненти додатку.

Приклад найпростішого файлу AndroidManifest.xml

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Manifest xmlns: android = "http://schemas.android.com/apk/res/android "
package = "com.example.untitled"
android: versionCode = "1"
```

```
android: versionName = "1.0">
<Uses-sdkandroid: minSdkVersion = "19" />
<application android: label = "@ string /
app_name"android: icon = "@ drawable / ic_launcher">
<Activity android: name = "MyActivity"
android: label = "@ string / app_name">
<Intent-filter>
<Action android: name = "android.intent.action.MAIN" />
<Category android: name = "android.intent.category.LAUNCHER" />
</ Intent-filter>
</ Activity>
</ Application>
</ Manifest>
```

Практична частина

Для початку роботи необхідно встановити Java Development Kit та Android Software Development Kit.

Завдання 1. Створення проекту програми. Запустіть середовище програмування в IDE InelliJ Idea (рис. 1.9). Введіть дані проєкту. Збережіть проєкт.

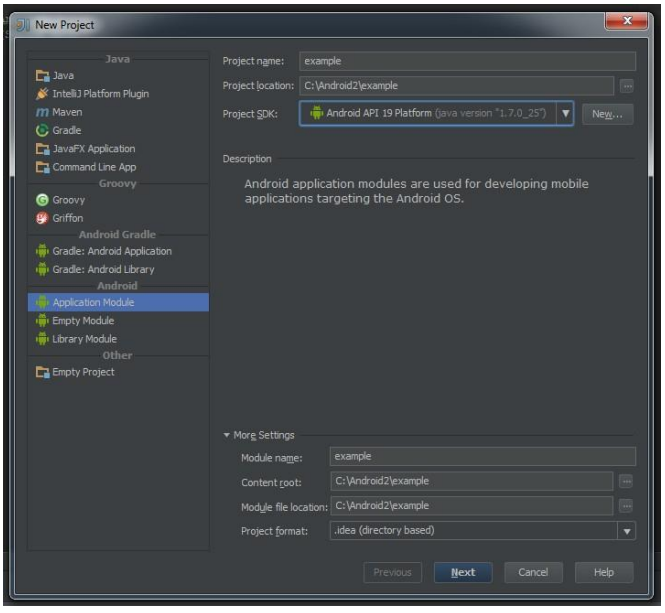


Рис. 1.9. Створення нового проекту в середовищі IntellyJ Idea

Завдання 2. Створення додатків з одним екраном (Activity). Необхідно створити два activity і організувати перехід між ними. Вміст Activity 1 - кнопка з ім'ям btn1. Вміст Activity 2 - TextView з текстом «Параметр: значення_параметра». Значення_параметра - з Activity 1.

При запуску програми користувач повинен потрапляти на екран з Activity 1. Після натискання на кнопку btn1 необхідно здійснити перехід до Activity 2 і передавати параметр з Activity 1. Як значення параметра використовувати своє прізвище.

Ключові класи: Activity, Intent, Button, TextView.

Контрольні питання

1. Що таке мобільний додаток, мобільна платформа?
2. Що собою являє архітектура мобільної платформи Android?
3. Які основні компоненти Android-додатку?
4. Що собою являє структура Android-проєкту?
5. Що містить файл конфігурації AndroidManifest.xml, папка java, папка res?
6. Що таке графічна реалізація Activity?

ЛАБОРАТОРНА РОБОТА 2
ОСНОВИ ВЕРСТКИ

Мета роботи: вивчити основи верстки. Навчитися керувати інтерфейсом мобільного пристрою при розробці програми.

Теоретичні положення

За замовчуванням Layout-файл налаштований під вертикальну орієнтацію екрану. Однак при повороті смартфона включиться горизонтальна орієнтація, що може призвести до некоректного відображення View-елементів. Для усунення даної проблеми необхідно створити ще один Layout-файл для горизонтальної орієнтації екрану. Це завдання вирішується за допомогою опції Create Landscape Variation (рис. 2.1).

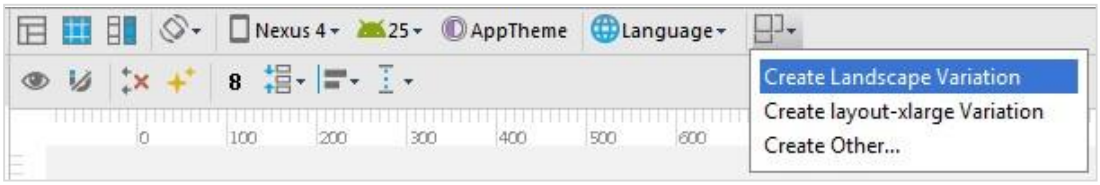


Рис. 2.1. Створення графічної реалізації горизонтальної орієнтації екрану

У структурі проєкту з'явиться новий xml-файл activity_main.xml (land). При цьому поточний зміст файлу activity_main.xml буде скопійовано в горизонтальне положення activity_main.xml (land). Далі змінюємо параметри view- елементів таким чином, щоб в горизонтальній орієнтації екрану вони відображались коректно. При цьому id елементів не змінюємо.

Усі наступні зміни activity_main.xml ніяк не впливатимуть на відповідні дані activity_main.xml (land), в зв'язку з чим створювати горизонтальну реалізацію доцільно після розміщення всіх необхідних компонентів в вертикальному положенні і коригування id-параметрів цих компонетів.

При запуску програми Activity прочитає підключений в коді Layout-файл і відобразить його вміст. При цьому буде врахована орієнтація екрану, і в разі горизонтального розташування автоматично відобразиться файл land.

Файл MainActivity.java. Підключення графічної реалізації до Activity

При запуску діяльності система повинна отримати посилання на кореневий вузол дерева розмітки, який буде використовуватися для промальовування графічного інтерфейсу на екрані мобільного пристрою. Для цього в методі onCreate () необхідно викликати метод setContentView ().

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

Метод setContentView(int) встановлює вміст Activity з Layout-файлу. Але як аргумент вказується не шлях до Layout-файлу (res/layout/activity_main.xml), а константа, яка є id файлу і зберігається в файлі R.java. Імена цих id-констант збігаються з іменами файлів ресурсів (без розширень).

Можна створити новий xml-файл в папці res> layout і прописати його замість activity_main в методі setContentView (int). Після запуску програми відобразиться новий файл розмітки.

Доступ до елементів екрану з коду

Щоб звернутися до елементу екрану з коду, необхідний його id. Він прописаний в вікні Properties або в xml-коді:

```
<Button
android: layout_width = "wrap_content" android: layout_height =
"wrap_content" android: text = "OK" android: id = "@ + id / btnOK" />
```

Знаючи id View-елемента, звернутися до нього з коду можна по константі R.id.btnOK. Для цього знадобиться метод findViewById:

```
public class MainActivity extends ActionBarActivity {private Button btnOK;
@Override
```

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

private void initViews () {
// знаходимо View-елементи
btnOK = (Button) findViewById (R.id.btnOK);
}
```

У наведеному фрагменті коду виявлення View-елементів винесено в окремий метод initViews() для реалізації принципу модульного програмування.

Обробка подій на прикладі натискання кнопки

Механізм обробки натискання кнопки заснований на використанні інтерфейсу View.OnClickListener і реалізації методу onClick, в якому і прописується логіка дій у відповідь на натискання (рис. 2.2).

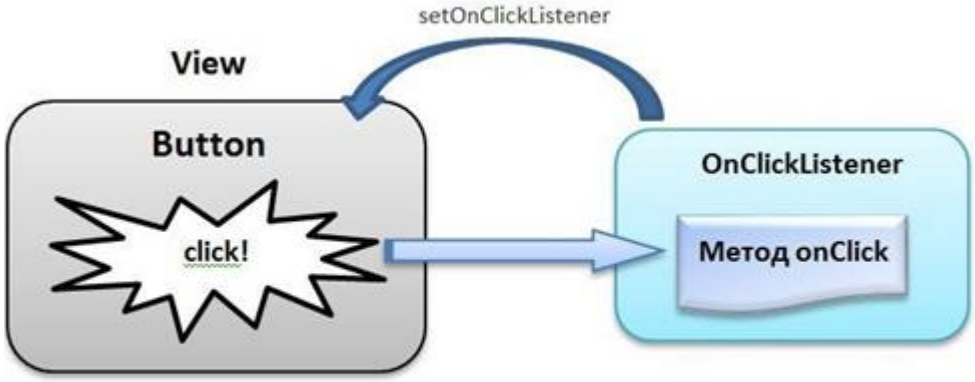


Рис. 2.2. Механізм обробоки натискання кнопки на основі інтерфейсу View.OnClickListener

Для реалізації даного механізму необхідно виконати наступні кроки:

– створити обробник (об'єкт від інтерфейсу View.OnClickListener);

– заповнити метод onClick;

– налаштувати обробник до кнопки (використовуємо метод setOnClickListener). Система обробки подій готова, а саме, коли натискають кнопку, обробник реагує і виконує код з методу onClick [1, 2].

Приклад:

```
OnClickListener listener = new OnClickListener () {@Override
public void onClick (View v) {
// метод, який буде викликаний після натискання
}
};
button.setOnClickListener (listener);
```

Існують різні способи програмної реалізації обробки натискання в залежності від необхідного набору кнопок одного Activity (з використанням в xml-поданні атрибута onClick, свого обробника для кожного View-елемента, одного обробника для декількох View-елементів). Рекомендовано використовувати Activity як єдиного обробника. В даному випадку сам клас Activity реалізує інтерфейс View.OnClickListener:

```
public class MainActivity extends ActionBarActivity
        implements View.OnClickListener{
private Button btnOK, btnCancel; @Override
protected void onCreate (Bundle savedInstanceState) {
super.onCreate (savedInstanceState); setContentView
(R.layout.activity_main); initViews ();
}
private void initViews () {
// знаходимо View-елементи
btnOK = (Button) findViewById (R.id.btnOK); btnCancel = (Button)
findViewById (R.id.btnCancel);
// підключаємо оброблювач до кнопок btnOK.setOnClickListener (this);
```

```
btnCancel.setOnClickListener (this);
    }

    @Override
    public void onClick (View v) {switch (v.getId ()) {
    case R.id.btnOK: btnOK.setText ( "Hello"); btnOK.setEnabled (false);
btnCancel.setEnabled (true); break;
    case R.id.btnCancel:btnCancel.setText ( "Goodbye");
btnCancel.setEnabled (false); btnOK.setEnabled (true);
    break;
    }
    }
    }
```

Для зчитування даних з текстових полів (наприклад, з поля et_message) використовується наступний код:

```
String message = et_message.getText (). ToString ();
```

Ієрархія класів, що знадобляться при розробці програми, наведена на рис. 2.3.

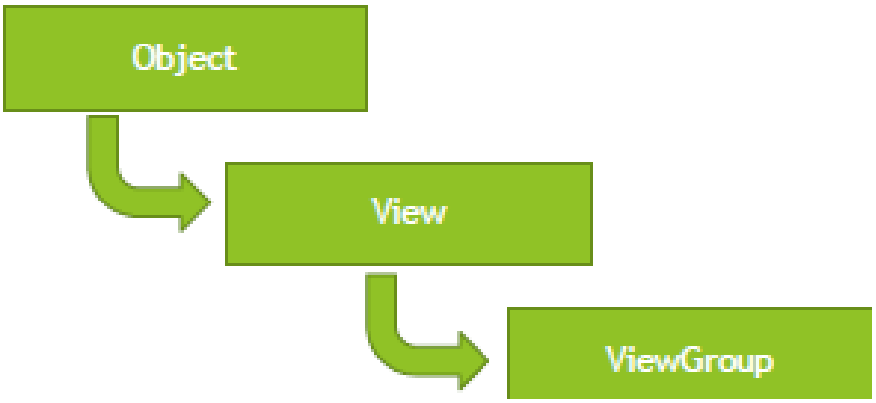


Рис. 2.3. Ієрархія класів View і ViewGroup

Структура для Activity наведена на рис. 2.4.

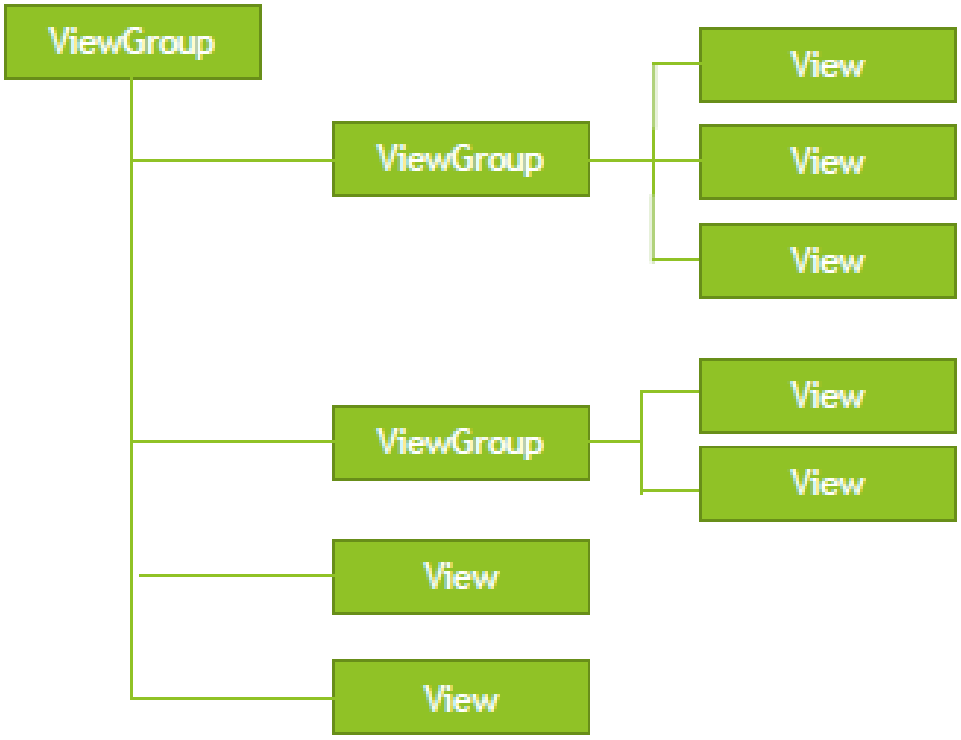


Рис. 2.4. Структура для Activity

Файл розмітки має наступну структуру

<? Xml version = "1.0" encoding = "utf-8"?>

<LinearLayout xmlns: android = "http://schemas.android.com/apk/res/android "

android: orientation = "vertical" android: layout_width = "fill_parent" android: layout_height = "fill_parent">

<LinearLayout android: layout_width = "match_parent" android: layout_height = "0dp" android: layout_weight = "0.25" android: padding = "5dp">

<Button android: layout_width = "0dp" android: layout_weight = "0.33"

android: layout_height = "wrap_content"android: text = "Button1"

android: id = "@ + id / button3" android: layout_gravity = "right" />

<Button android: layout_width = "0dp" android: layout_weight = "0.33"

android: layout_height = "wrap_content"android: text = "Button2" android: id = "@ + id / button" />

<Button android: layout_width = "0dp" android: layout_weight = "0.33"

```
android: layout_height = "wrap_content"android: text = "Button3"
android: id = "@ + id / button2" android: layout_gravity = "center_horizontal" />
</ LinearLayout>
<LinearLayout android: layout_width = "match_parent" android:
layout_height = "0dp" android: paddingLeft = "40dp" android: paddingRight =
"40dp" android: layout_weight = "0.5" android: gravity = "center_vertical">
<Button android: layout_width = "0dp" android: layout_weight = "0.33"
android: layout_height = "wrap_content"android: text = "Button4" android: id
= "@ + id / button3" />
<Button android: layout_width = "0dp" android: layout_weight = "0.33"

android: layout_height = "wrap_content"android: text = "Button5" android: id
= "@ + id / button" />
</ LinearLayout>
<LinearLayout android: layout_width = "match_parent" android:
layout_height = "0dp" android: layout_weight = "0.25" android: padding = "5dp"
android: gravity = "bottom">
...
</ LinearLayout>
</ LinearLayout>
```

Поширені види макетів наведені на рис. 2.5.

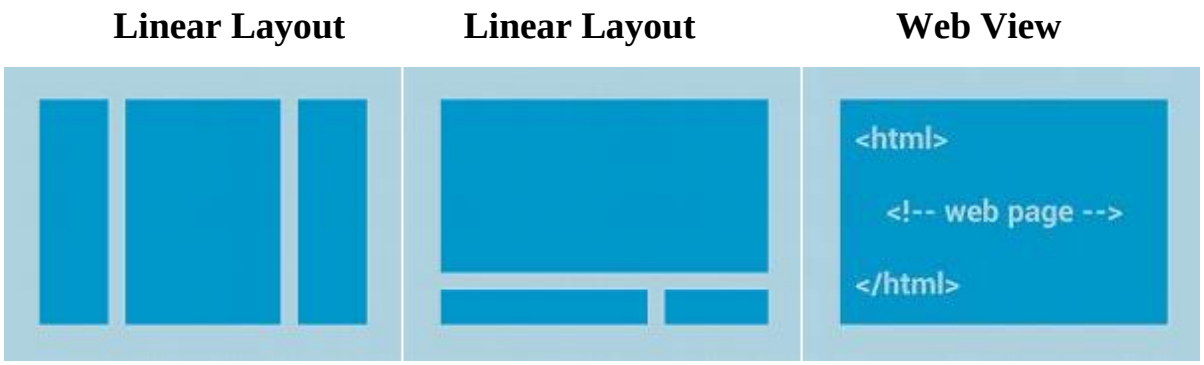


Рис. 2.5. Поширені види макетів

Макети з адаптером наведені на рис. 2.6.

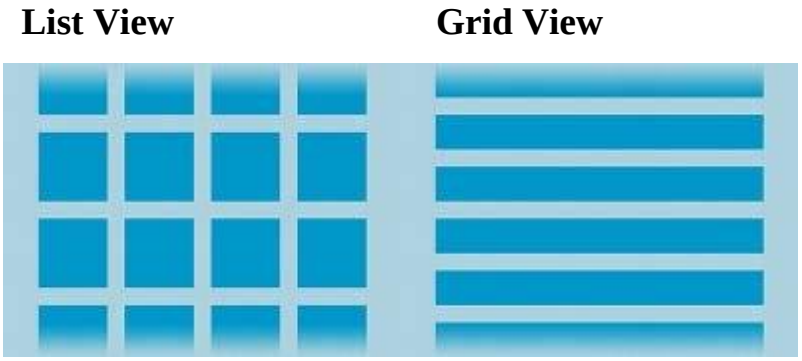


Рис. 2.6. Інтерфейс макетів з адаптером

Практична частина

Завдання 1. Розробити мобільний додаток, що складається з чотирьох Activity.

Після запуску програми користувач повинен потрапляти на екран з Activity1. На цьому екрані має бути представлено меню, що складається з чотирьох кнопок. Висота кнопок повинна складати 20% від висоти екрана. Відстань між кнопками - 2%. Перша і остання кнопка повинні бути на рівній відстані від країв екрану. Ширина кнопок 75%, вирівнювання посередині.

Після натискання на першу кнопку користувач повинен переходити до activity2, його зовнішній вигляд представлений на рис. 2.7. Верстка повинна здійснюватися з використанням LinearLayout, ширина кнопок повинна задаватися в відсотках від ширини екрану.

Після натискання на другу кнопку в Activity1 користувач повинен переходити до Activity3, його зовнішній вигляд наведений на рис. 2.8. Верстка повинна здійснюватися з використанням RelativeLayout (не використовувати LinearLayout).

Третя кнопка в Activity1 повинна створювати Activity3. Зовнішній вигляд Activity3 наведений на рис. 2.9. Кнопка повинна бути вирівняна по центру екрана. Колір обведення кнопки # 505050. Товщина обведення відповідно до місяця вашого народження (від 1 до 12). Радіус заокруглення 24dp. Колір фону екрану #FFFFFF. При натисканні на кнопку її колір повинен змінюватися на

світло-зелений. Натискання на четверту кнопку в Activity1 повинно призводити до закриття програми.

Рис. 2.7. Зовнішній вигляд екрану для першого завдання

Рис. 2.8. Результат першого етапу виконання завдання

Рис.2.9. Інтерфейс програми на етапі Activity3

Контрольні питання

1. Що таке Layout?
2. Які існують види Layout?
3. Які параметри мають View-елементи?
4. Як створити Layout-файл для роботи в горизонтальній орієнтації екрану мобільного пристрою? У яких випадках це необхідно?
5. Для чого потрібні методи setContentView, findViewById?
6. Які існують способи обробки подій в Activity?

ЛАБОРАТОРНА РОБОТА 3

ЗБЕРІГАННЯ ІНФОРМАЦІЇ В БАЗІ ДАНИХ SQLite

Мета роботи: вивчити роботу Android-програми з базою даних.

Теоретичні положення

База даних SQLite доступна на будь-якому Android-пристрої, її не потрібно встановлювати окремо. SQLite підтримує типи TEXT (аналог String в Java), INTEGER (аналог long в Java) і REAL (аналог double в Java). Решта типів слід конвертувати, перш ніж зберігати в базі даних. Для зберігання зображень в базі даних вказують шлях до зображень, а самі зображення зберігають в файловій системі.

Для розуміння механізму роботи з SQLite в Android-додатках розглянемо найпростіший приклад (рис. 3.1): створимо таблицю товарів з полями id (номер), name (найменування товару), price (ціна), count (кількість одиниць товару в наявності). Для роботи з таблицею товарів реалізуємо функції додавання (кнопка Add), читання всіх даних таблиці і виведення інформації в лог (кнопка Read), очищення всієї таблиці (кнопка Clear), введення name, price, count як нових даних вже існуючого товару з номером id (кнопка Update), видалення товару за id (кнопка Delete).

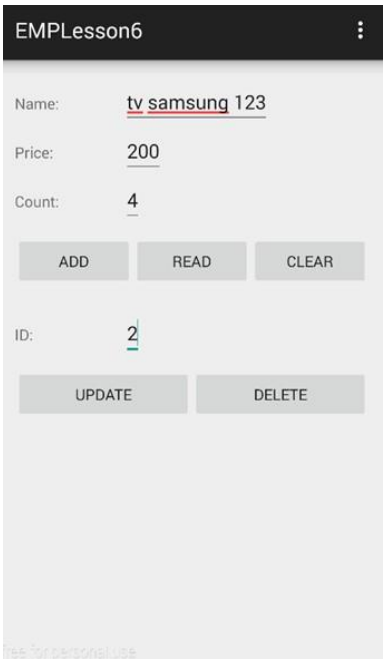


Рис.3.1. Приклад роботи з SQLite в Android-додатках

Робота з базою даних зводиться до наступних завдань:

1. Створення та відкриття бази даних, створення таблиці. Дані завдання реалізуються за допомогою класу SQLiteOpenHelper.

2. Створення інтерфейсу для вставки даних.

Клас ContentValues використовується для додавання нових рядків у таблицю. Кожен об'єкт цього класу являє собою один рядок таблиці і виглядає як асоціативний масив з іменами стовпчиків і значеннями, які їм відповідають.

3. Створення інтерфейсу для виконання запитів (вибірки даних). Запити до бази даних повертають об'єкти класу Cursor. Замість того щоб отримувати дані і повертати копію значень, курсори посилаються на набір вихідних даних. Курсори дозволяють управляти поточною позицією (рядком) в результуючому наборі даних, що повертається при запиті.

4. Закриття бази даних. Як правило виконується в методі onDestroy () java-класу Activity.

За допомогою абстрактного класу SQLiteOpenHelper можна створювати, відкривати і оновлювати бази даних. Це основний клас, з яким необхідно працювати в Android-проєктах. Клас SQLiteOpenHelper містить два обов'язкових абстрактних методи:onCreate (): викликається при першому створенні бази даних; onUpgrade (): викликається при модифікації бази даних (зокрема, при спробі підключення до БД більш нової версії, ніж існуюча) [1].

Реалізація методу onCreate

```
public void onCreate (SQLiteDatabasedb)
{dЬ.execSQL ( "CREATE TABLE" + TABLE_NAME
+ "(_Id INTEGER PRIMARY KEYAUTOINCREMENT,"
+ COL NAME + "TEXT," + COL PHONE + "TEXT);");
```

Реалізація методу onUpgrade

```
@ Override
public void onUpgrade (SQLiteDatabasedb, intoldVersion, intnewVersion)
{
```

```
db.execSQL ( "DROP TABLE IF EXISTS" + TABLE_NAME);
onCreate (db);
}
```

Читання з бази даних

```
Cursor query (String table, String [] columns,String selection, String [] selectionArgs,
String groupBy, String having, String sortOrder)
```

Позиціонування курсора

moveToFirst() - зміщує курсор в перший запис у вибірці;

moveToLast() - зміщує курсор в останню запис у вибірці;

moveToNext() - зміщує курсор в наступний запис і одночасно визначає, чи існує цей запис. Метод moveToNext () повертає true, якщо курсор вказує на інший рядок після переміщення, і false, якщо поточний запис був останнім у вибірці;

moveToPrevious () - зміщує курсор до попереднього запису;

moveToPosition () - зміщує курсор в зазначену позицію;

getPosition () -возвращает поточний індекс позиції курсора.

Практична частина

Завдання 1. Необхідно створити додаток, що взаємодіє з базою даних. Перша активність повинна містити три кнопки. При натисканні на першу кнопку повинна відкриватись нова активність, що виводить інформацію з таблиці «Одногрупники» в зручному для сприйняття форматі.

При запуску програми необхідно:

1. Створити базу даних, якщо її не існує.
2. Створити таблицю «Одногрупники», що містить поля:
   - ID;
   - ПІБ;
   - Час додавання запису.
3. Видаляти усі записи з бази даних, а потім внести 5 записів про

одногрупників. При натисканні на другу кнопку необхідно внести ще один запис в таблицю. При натисканні на третю кнопку необхідно замінити ПІБ в останньому записі на Петренко Петро Петрович.

Завдання 2. Створити нову прикладну програму на основі додатку, створенного в завданні 1. Перевизначити функцію onUpgrade. При зміні версії БД необхідно вилучити таблицю «Одногрупники», створити таблицю «Одногрупники» містить наступні поля:

- ID;
- Прізвище;
- Ім'я;
- По-батькові;
- Час додавання запису.

Змінити версію бази даних.

Контрольні питання

1. Як називається базовий клас для роботи з базою даних SQLite в Android?
2. Які методи обов'язкові для перевизначення при роботі з базою даних SQLite?
3. Для чого використовується клас ContentValues?
4. Для чого використовується клас Cursor?
5. Як реалізуються методи insert, query, delete, update для вставки, читання, видалення і додавання записи в SQLite?

ЛАБОРАТОРНА РОБОТА 4
РОБОТА З МУЛЬТИМЕДІЙНИМИ ФАЙЛАМИ

Мета роботи: вивчити роботу з потоками, навчитися працювати з мультимедійними файлами та з класом AsyncTask.

Теоретичні положення

Програми для платформи Android розповсюджуються у вигляді спеціальних підписаних архівів Android Package (APK), аналогічних подібним архівам для інших систем (APPX для Microsoft Windows, DEB для ОС Linux, заснованих на Debian) і JAR-архівів, що використовуються для розповсюдження Java -Додатків. APK-файл формується автоматично на етапі складання програми в Android Studio [4]. Він містить скомпільований байт-код класів програми у форматі Dalvik executable (у вигляді файлів *. Dex), а також додаткові дані - ресурси додатку та його опис (маніфест).

Архів APK включає наступні папки та файли:

- папка META-INF, що містить файли: − MANIFEST.MF – файл опису архіву (Java-маніфест);
- <name>.SF — файл підпису зі списком хеш-значень відповідних записів з файлу MANIFEST.MF;
- <name>.DSA (або RSA) — файл блоку підпису програми, що включає сертифікат відкритого ключа, парний закритий ключ якого використовувався при створенні підпису;
- папка lib, що містить код бібліотек Android, використовуваних програмою, залежно від платформи скомпілюваному вигляді;
- папка res, яка містить ресурси програми, які не були скомпільовані у файл resources.arsc;
- папка assets, що містить активи, які програма може отримати в процесі своєї роботи за допомогою об'єкта AssetManager;
- файл AndroidManifest.xml — додатковий файл маніфесту ОС Android, описує ім'я програми, його версію, права доступу та бібліотеки ОС Android, які використовуються в додатку;

- classes.dex — класи програми, скомпільовані у форматі DEX, які використовуються ART;
- resources.arsc — файл, що містить передкомпільовані ресурси.

Виконання додатків на платформі Android

Для роботи мобільного додатка та його взаємодії з сервісами операційної системи надаються спеціальні класи Google Android API. Для реалізації необхідної функціональності додаток створює об'єкти необхідних класів. Базові класи, що надаються Google Android API для використання в мобільних додатках:

- View — базовий клас всіх компонентів інтерфейсу користувача, інтерфейс Android утворюється нащадками цього класу;

- Activity — базовий клас, що містить логіку взаємодії з компонентами інтерфейсу користувача певного вікна програми (похідним класам зазвичай прийнято давати назву <screen>Activity.java, де <screen> - назва вікна або екрана, поведінкою якого управляє цей клас);

- ContentProvider — клас, призначений для роботи з моделями даних, найчастіше — з СУБД SQLite;

- Service — клас, призначений для створення сервісів, тобто дій, які мають виконуватися у фонових потоках програми;

- Intent — клас, який використовується для створення об'єктів повідомлень, які використовуються для запиту дій з інших компонентів програми, він може бути використаний для запуску Activity та передачі інформації службам;

- BroadcastReceiver — клас, який отримує та надсилає об'єкти Intent [5].

Класи типу Activity - це класи активного вікна програми. Кожен такий клас описує логіку інтерфейсу користувача деякого вікна програми. Класи Activity програми створюються як похідні від класу Activity, що надається Google Android API. Об'єкти типу Activity взаємодіють з користувачем і створюють вікна, в яких розробник розміщує свій користувальницький інтерфейс за допомогою методу setContentView(). Найчастіше об'єкти Activity

являють собою повноекранні вікна, але їх також можна використовувати й іншими способами: спливаючі вікна, багатоекранний режим або вбудоване вікно [6].

Виконання програми починається зі створення та запуску першого об'єкта Activity, клас якого відзначений спеціальним чином у файлі AndroidManifest.xml. Цей об'єкт Activity послідовно, відповідно до логіки роботи програми та дій користувача, може передавати управління іншим об'єктам Activity, викликаючи їх шляхом направлення операційній системі відповідного повідомлення типу Intent. Кожен об'єкт Activity у процесі свого існування має власний життєвий цикл. Програма виконується, поки не завершено життєві цикли всіх об'єктів Activity.

Життєвий цикл об'єкта типу Activity жорстко контролюється системою і залежить від доступних ресурсів, потреб користувача. При роботі з певним вікном система дає пріоритет тому додатку, який з ним асоційований. І навпаки, якщо користувач не працює з вікном програми протягом певного періоду часу, система зупиняє додаток, що володіє ним, щоб звільнити зайняті ним ресурси. Оскільки Android - операційна система, розроблена спеціально для мобільних пристроїв, обчислювальні та енергетичні ресурси яких обмежені, вона жорстко контролює роботу додатків [6, 7].

Перерахуємо методи життєвого циклу Activity як основної функціональної складової програми:

- onCreate() — викликається при створенні (запуску) даного об'єкта Activity. У цьому методі створюються статичні елементи, завантажуються і ініціалізуються елементи графічного інтерфейсу, зв'язуються дані з елементами управління. У цьому методі розробнику не слід прописувати процес довгої ініціалізації - це може призвести до помилки.

- onStart() — викликається системою за onCreate(), у цьому методі виконуються дії, які Activity здійснює перед показом користувачеві екрана, з яким цей об'єкт асоційований.

- onRestart() — викликається, якщо вікно отримає пріоритет після виклику

методу onStop(). Використовується для спеціальних дій, які потрібно виконати після повторного запуску Activity.

- onResume() — викликається після onStart(). У цей момент користувач взаємодіє з цим вікном і йому передається пріоритет. У ньому можна запускати анімацію, відео тощо.

- onPause() — викликається, коли користувач починає роботу з іншим вікном. Зберігає незафіксовані дані, звільняє ресурси.

- onStop() — викликається при знищенні вікна або переході користувача до іншого вікна. Зупинені об'єкти Activity зберігаються у пам'яті та відновлюються під час запуску.

- onDestroy() — викликається після закінчення роботи Activity, при виклику методу finish() або у разі, коли система знищує даний екземпляр Activity для звільнення ресурсів.

Ресурси програми

Ресурси програми - це додаткові файли і дані, що використовуються в додатку, такі як рядки, зображення, розмітка інтерфейсу, кольору тощо. Ресурси призначені для збільшення незалежності програми від його оточення. Розробник може вказувати альтернативні набори ресурсів, що використовуються додатком при його виконанні, наприклад у середовищах з різними мовами інтерфейсу користувача [8].

Як і файл AndroidManifest.xml, ресурси зберігаються в APK програм у вигляді файлів, що використовують розмітку XML [9]. Одним з найважливіших видів ресурсів є опис макетів вікон, що використовуються додатком. Макет вікна, асоційованого з деяким об'єктом <screen>Activity, описується у файлі з ім'ям activity_<screen>.xml, де <screen> - назва цього вікна. У файлі макету визначаються всі елементи інтерфейсу та його властивості. Макет містить кореневий елемент, для якого всі інші елементи інтерфейсу є дочірніми. Опис кожного елемента інтерфейсу являє собою XML-елемент, що містить дані про його ім'я, вміст, батьківські елементи, знаходження на екрані тощо.

Опис інтерфейсу користувача в окремих файлах макетів дозволяє відокремити його від безпосередньо самого коду програми, що забезпечує можливість зміни опису користувальницького інтерфейсу без зміни коду класів Java. У файлах макетів може бути визначений вид екрана для різних орієнтацій пристрою, різних розмірів екрана, різних мов тощо [10].

APK-файл програми для операційної системи Android повинен обов'язково включати файл AndroidManifest.xml, що містить всю необхідну інформацію про програму, що формується і використовувана інструментами складання Android, операційною системою Android і магазином додатків Google Play. У цьому файлі маніфесту обов'язково має бути зазначено:

- Назва пакета програми. Інструменти складання Android використовують цю інформацію для знаходження елементів коду при складанні програми.

- Компоненти програми: всі файли Activity, Service, BroadcastReceiver та ContentProvider. Кожен компонент повинен бути описаний найпростішими властивостями, такими як ім'я класу Java.

- Дозволи, які потрібні програмі для доступу до захищених частин системи.

- Програмні та апаратні вимоги програми [11].

Оскільки мобільні програми, що працюють у середовищі Android, виконуються на особистому пристрої користувача, для використання системних функцій, що надають доступ до особистої інформації користувача (наприклад, доступ до записника або SMS) або до пристроїв смартфона (наприклад, доступ до камери або доступ в Інтернет), додаток повинен, з одного боку, явно задекларувати дозволи, які йому необхідно отримати для своєї роботи, а з іншого боку, явно отримати ці дозволи. Кожний наданий програмі дозвіл також вказується спеціальним записом у файлі маніфесту операційної системи Android. Існує кілька видів прав доступу:

- Install-Time Permissions — визначають доступ програми під час встановлення. Такі дозволи зазвичай зачіпають функції, що мінімально впливають на користувача.

- Normal Permissions — дозволяє отримати доступ до даних та дій, що виходять за межі ізольованого програмного середовища програми. Однак ці дані та дії становлять дуже невеликий ризик для конфіденційності користувача та роботи інших додатків.

- Runtime Permissions — надають додатку додатковий доступ до обмежених даних і дозволяють додатку виконувати обмежені дії, які більш істотно впливають на систему та інші програми. Ці дозволи додаток повинен запросити під час виконання, перш ніж він зможе отримати доступ до обмежених даних або виконати обмежені дії. Коли програма запитує такий дозвіл під час виконання, система видає запит дозволу у вигляді спливаючого вікна.

Взаємодія з мережею

Будь-яка взаємодія з мережею Інтернет у сучасній версії Android (починаючи з 3.0) має відбуватися поза основним потоком, що обслуговує взаємодію користувача з інтерфейсом програми. Одним з способів виконання асинхронних операцій є використання об'єктів класу AsyncTask.

Цей клас призначений для виконання завдань у фоновому режимі. Коли операції завершаться, він також дозволяє оновлювати уявлення в основному потоці подій. Якщо завдання являє собою серію операцій, що повторюються, об'єкт цього класу також може використовуватися для публікації інформації про прогрес завдання під час її виконання [12].

Для створення власної асинхронної операції розробник повинен створити похідний клас і відповідним чином перевизначити його методи.

AsyncTask визначається трьома узагальненими параметрами: Params — тип об'єкта, що використовується для передачі довільних параметрів завдання методу doInBackground(), Progress — тип об'єкта, що використовується для передачі інформації про прогрес завдання, і Results — тип результату задачи. Якщо один із цих параметрів не використовується, замість нього можна вказати екземпляр класу Void [12]. Клас AsyncTask містить такі методи:

- onPreExecute() — викликається до початку фонової операції та

використовується для її підготовки.

- doInBackground() — запускається у фоновому режимі відразу після завершення методу onPreExecute(). Розробник визначає тип параметрів, які повинні передаватися задачі і тип значення, що повертається.

- onProgressUpdate() — викликається в головному потоці та служить для сповіщення про процес виконання фонового завдання.

- onPostExecute() — викликається після завершення фонового завдання. Отримує результат виконання методу doInBackground() і може взаємодіяти з головним потоком.

Використовуючи об'єкти типу AsyncTask, розробник може організувати клієнт-серверну мережеву взаємодію з використанням як стандартних засобів Java, таких як HttpURLConnection, так і засобів, що надаються сторонніми бібліотеками, наприклад OkHttp.

Отримання даних про розташування пристрою

Для отримання даних про розташування пристрою Google надає спеціальний API - Fused Location Provider API. Це геолокаційний програмний інтерфейс, що використовує сигнали різних давачів, таких як GPS і Wi-Fi, для визначення розташування пристрою. Fused Location Provider бере на себе відповідальність за низькорівневі технології визначення геолокації і надає прості засоби взаємодії з ними [13].

Головна точка, через яку програма може взаємодіяти з Fused Location Provider API - це клас FusedLocationProviderClient [14]. Для отримання та оновлення даних розташування FusedLocationProviderClient використовує асинхронні операції Task [15]. Це клас, об'єкт якого виконує деяку операцію.

Основні методи взаємодії з геолокаційними даними, наведені в класі FusedLocationProvider-Client:

- getLastLocation() — повертає найновіше місце розташування пристрою;

- requestLocationUpdates() — запитує оновлення геолокації;

- removeLocationUpdates() — зупиняє оновлення геолокації.

Ініціалізація екземпляра цього класу здійснюється шляхом отримання

результату роботи методу getFusedLocationProviderClient() класу LocationServices, який є частиною Google Mobile Services. Для додавання в додаток функціональності отримання геолокації розробник повинен реалізувати обробку процесу і результату виконання операцій типу Task, створених виконанням методів класу FusedLocationProviderClient.

Практична частина

Завдання 1. Розгляньте приклад передачі даних.

```
MyAsyncTaskat = newMyAsyncTask ();
at.execute ( "url1", "url2");
doInBackground (String ... urls)
```

Завдання 2. Розгляньте приклад виведення проміжних даних.

```
@Override
Protected void doInBackground (String
        ... urls) {try {
                int cnt = 0; for
                (Stringurl:
                urls) {
                        // обробляємо перший параметр
                            ...
                        // виводимо проміжні результати
                        cnt ++;
                        publishProgress
                        (cnt);
        }
        TimeUnit.SECONDS.sleep (1);
} Catch
        (InterruptedException
```

```
        e) {
        e.printStackTrace ();
        }
        return null;
    }
    @Override
        protected void onProgressUpdate (Integer ... values)
            {super.onProgressUpdate (values); tv.setText (
            "оброблено" + values [0] + "параметрів");
    }
```

Завдання 3. Перевірте приклад створення простої асинхронної Activity

```
    public class MainActivity extends
    Activity {MyAsyncTask at;
    TextView tv;
    public void onCreate (Bundle savedInstance State) {super.onCreate
        (savedInstanceState); setContentView (R.layout.main);
        tv = (TextView) findViewById (R.id.tv);
        MyAsyncTask at = new MyAsyncTask
        ();at.execute ();
    }
    }
```

Завдання 4. На підставі вивчених прикладів розробити додаток, що зберігає статистику пісень, які програються на Webradio. Для збереження пісні і назви необхідно створити базу даних, що містить таблицю з наступними полями:

1) ID
2) Виконавець
3) Назва треку

4) Час внесення запису

При запуску програми необхідно проводити перевірку підключення до Інтернету. У разі якщо сигнал мережі переривається - виводити спливаюче повідомлення (Toast) з попередженням про запуск в автономному режимі (доступний тільки перегляд внесених раніше записів).

Після включення додаток повинен робити асинхронне опитування сервера з інтервалом 20 секунд. Якщо назва треку не збігається з останнім записом в таблиці необхідно зробити запис в базі даних.

URL адреси, за якою можна отримати інформацію про поточний трек і виконавця:

https://webradio.io/api/radio/pi/current-song

У разі успішного виконання запиту результат буде мати вигляд:

{ "Result": "success", "info": "Виконавець - Назва треку"}

У разі помилки API поверне наступний рядок:

{ "Result": "error", "info": "Інформація про помилку"}

Програма повинна дозволяти переглядати внесені в базу даних записи.

Контрольні питання

1. Як за допомогою класу Toast створити спливаюче повідомлення?
2. У яких випадках необхідно логирование?
3. Що являє собою вікно LogCat? Які існують рівні логіро- вання?
4. Як програмно реалізувати логирование?
5. Як створити нове Activity (опишіть роботу з java-класом, layout-файлом, файлом конфігурації AndroidManifest.xml)?
6. Для чого використовується контекст додатки Context?

ЛАБОРАТОРНА РОБОТА 5

РОБОТА З ДАНИМИ - ЗОВНІШНІ ФАЙЛИ

Мета роботи: вивчити інструменти зберігання даних, а також роботу з зовнішніми файлами.

Теоретичні положення

Робота мобільного застосування може супроводжуватися виникненням повідомлень, що спливають, діалогових вікон, роботою з меню тощо. Розглянемо реалізацію найпростішого допоміжний елемент – повідомлення, що спливає.

Повідомлення, що спливає

Додаток може показувати повідомлення, що спливає за допомогою класу Toast:

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

Статичний метод makeText створює View-елемент Toast. Далі розглянемо параметри методу:

1) context - об'єкт, який надає доступ до базових функцій; оскільки Activity є підкласом Context, то в якості об'єкта від Context використовуємо поточний Activity, а в якості першого параметра вказуємо this;

2) text - текст, який треба вивести;

3) duration - тривалість показу (Toast.LENGTH_LONG - довга (3,5 с), Toast.LENGTH_SHORT - коротка (2 с)).

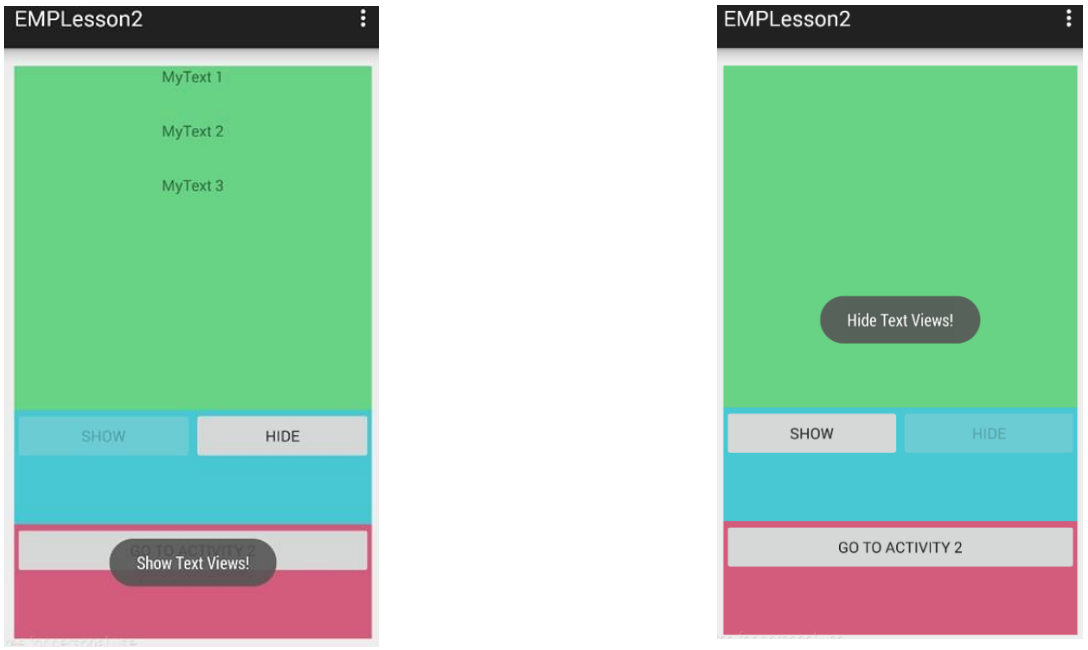
Щоб Toast відобразився на екрані, викликається метод show (). Приклад реалізації (рис. 5.1):

```
Toast.makeText (this, "Show Text Views!", Toast.LENGTH_SHORT) .show
();
```

За замовчуванням повідомлення, що спливають відображається в нижній частині робочого вікна по центру. Для зміни місця розташування повідомлення, що спливає використовується метод setGravity:

```
Toast toast = Toast.makeText (this, "Hide Text Views!",
Toast.LENGTH_SHORT);
toast.setGravity (Gravity.CENTER,
```

0, 0); toast.show ();



а *б*

Рис. 5.1. Приклад відображення повідомлень, що спливають: *а* - висновок повідомлення «Show Text Views!» внизу вікна після натискання кнопки Show і його розміщується за умовчанням; б - висновок повідомлення "Hide Text Views!» в центрі вікна після натискання кнопки Hide

Логування

При тестуванні роботи програми можна відстежувати логи в вікні logcat. Список мають різні рівні важливості: error, warn, info, debug, verbose. Установка Log level в вікні logcat призводить до фільтрації логів зазначеного рівня, а також рівнів вищої важливості. Є можливість створювати, редагувати і видаляти свої фільтри. Логування є одним з інструментів розробника на етапі тестування додатку. Для цього програмно створюються логи, у вікні logcat відстежується логіка роботи програми, а перед випуском фінальної версії створені логи з програми видаляються.

Життєвий цикл Activity

Створене при роботі додатка Activity може бути в одному з трьох станів:

1. Resumed: Activity видно на екрані, воно знаходиться у фокусі,

користувач може з ним взаємодіяти. Цей стан також іноді називають Running.

2. Paused: Activity не в фокусі, користувач не може з ним взаємодіяти, але його видно (воно перекрито іншим Activity, яке займає не весь екран або напівпрозоре).

3. Stopped: Activity не видно (повністю перекривається іншим Activity), воно не в фокусі і користувач не може з ним взаємодіяти.

Коли Activity переходить з одного стану в інший, система автоматично викликає відповідні методи, в які можна додавати свій код. Методи Activity, які викликає система:

– onCreate () - викликається при першому створенні Activity;

– onStart () - викликається перед тим, як Activity буде видно користувачеві;

– onResume () - викликається перед тим, як буде доступно для активності користувача;

– onPause () - викликається перед тим, як буде показано інше Activity;

– onStop () - викликається, коли Activity стає видним користувачеві;

– onDestroy () - викликається перед тим, як Activity буде знищено.

Зміна стану Activity є тригером, який викликає ці методи. Тим самим ми можемо реагувати на подію, що відбулася необхідним чином.

Для того щоб у відповідь на зміни, що відбуваються стану Activity відпрацював необхідний код, необхідно перевизначити описані методи життєвого циклу Activity.

Створення нового Activity

Мобільний додаток, як правило, містить кілька Activity і реалізує перехід між ними з передачею даних при необхідності.

Створити нове Activity можна вручну або скористатися вбудованою функцією додавання нового компонента - Activity - в проект. Далі розглянемо обидва способи.

Алгоритм створення нового Activity вручну включає наступні кроки:

1. Створюємо новий xml-файл розмітки (наприклад, activity_second.xml).

2. Створюємо новий java-клас другого Activity (наприклад, SecondActivity).

3. Прописуємо логіку SecondActivity.java (наслідуючи від суперкласу, підключаємо графічну розмітку):

```
public class SecondActivity extends ActionBarActivity {@Override
protected void onCreate (Bundle savedInstanceState) {super.onCreate (savedInstanceState); setContentView (R.layout.activity_second);
}
}
```

4. Реєструємо нове Activity в маніфест-файлі (це обов'язково):

```
<activity
android: name = ". MainActivity" android: label = "@ string / app_name">
<Intent-filter>
<Action android: name = "android.intent.action.MAIN" />
<Category android: name = "android.intent.category.LAUNCHER" />
</ Intent-filter>
</ Activity>

<activity
android: name = ". SecondActivity" android: label = "@ string / app_name">
```

Програмний спосіб додавання нового компонента в Android-додаток реалізується за допомогою меню File → New або шляхом натискання комбінації клавіш Alt + Insert. Далі необхідно заповнити форму кастомізації нового Activity із зазначенням імен java-файлу, xml-файлу за аналогією зі створенням Activity в новому проекті. Запис у файлі маніфесту буде створена автоматично.

Наступний крок після створення нового Activity - навчитися викликати це Activity, наприклад у відповідь на натискання однойменної кнопки в вихідному

Activity (рис. 5.2).

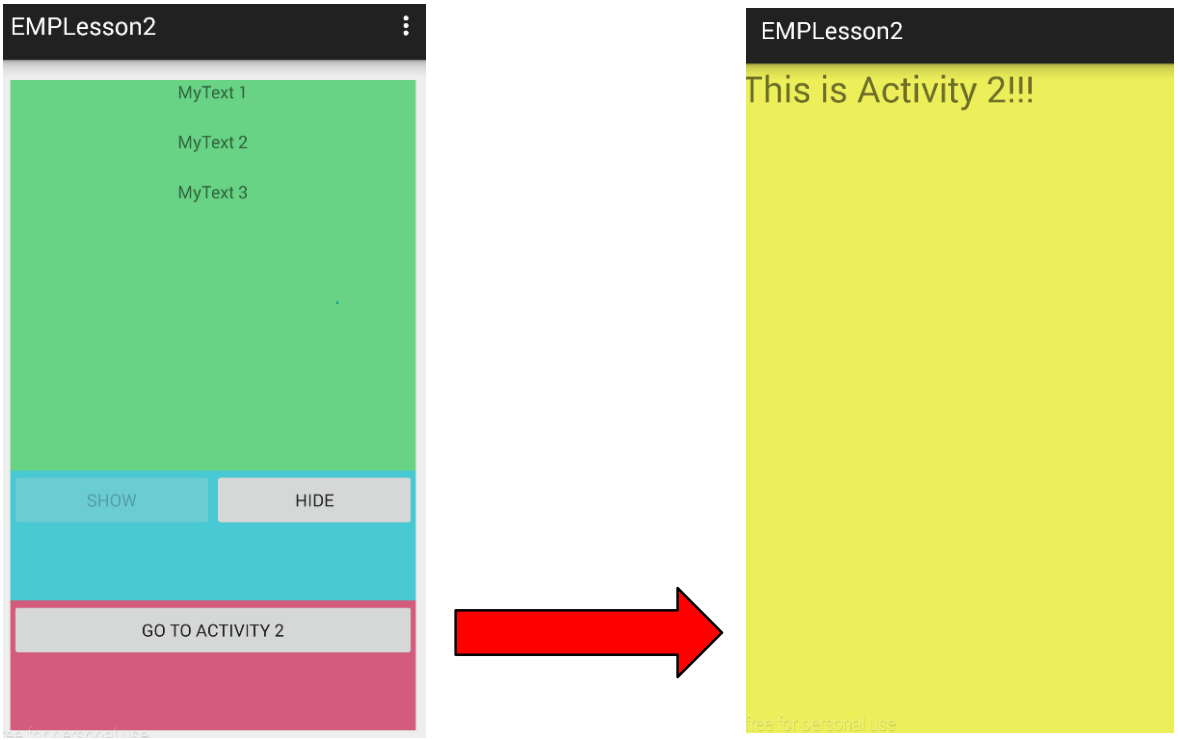


Рис. 5.2. Приклад виклику нового Activity у відповідь на натискання однойменної кнопки в вихідному Activity

Для того щоб з одного Activity викликати інше, існує два способи: явний і неявний виклик. Реалізуємо явний виклик нового Activity з основного:

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

Тоді неявно викликати SecondActivity з MainActivity можна наступним проразом:

```
intent = new Intent ( "android.intent.action.SecondActivity"); startActivity (intent);
```

Параметри об'єкта Intent з наведеного коду збігаються з умовами фільтра в маніфест-файлі, і SecondActivity буде викликано.

Слід зазначити, що у випадку з неявним викликом пошук йде вже по всьому Activity всіх додатків в системі. Якщо таких знаходиться декілька, то система надає вибір, який саме програмою необхідно скористатися.

Передача даних між Activity

Передача даних між Activity здійснюється за допомогою класу Intent. Збереження даних в Intent реалізується за допомогою методу putExtra (), а одержання - getExtra (). Метод putExtra додає до об'єкта пару: перший параметр - це ключ (ім'я), другий - значення. Приклад коду з MainActivity.java:

```
Intent intent = new Intent (this, SecondActivity.class); intent.putExtra ( "Name", et_Name.getText (). toString ()); intent.putExtra ( "Email", et_Email.getText (). toString ()); startActivity (intent);
```

Метод getExtra витягує значення за ключем. Приклад коду з SecondActivity.java:

```
protected void onCreate (Bundle savedInstanceState) {super.onCreate (savedInstanceState); setContentView (R.layout.activity_second); initViews ();
Intent intent = getIntent (); tv_Name.setText (intent.getStringExtra ( "Name")); tv_Email.setText (intent.getStringExtra ( "Email"));
}
```

Метод startActivityForResult

Запуск іншого Activity не обов'язково повинен бути одностороннім. Можна запустити інше Activity і отримати результат. Для цього використовується метод startActivityForResult () замість startActivity () (рис. 5.3). Алгоритм дій наступний:

1. Викликати з вихідного Activity нове Activity не за допомогою методу startActivity, а за допомогою startActivityForResult. Синтаксис методу startActivityForResult: `startActivityForResult (Intent intent, int requestCode);`

Тут requestCode - ідентифікатор для визначення, з якого Activity прийшов результат.

2. Реалізувати в новому Activity метод setResult для вказівки, які дані необхідно повернути в «батьківське» Activity і викликати finish ().

3. У вихідному Activity в методі onActivityResult прописати логіку обробки результату, який прийде з нового Activity.

4.

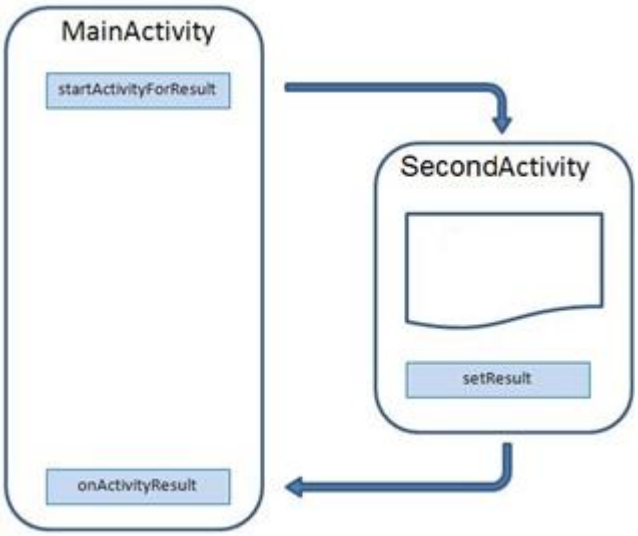


Рис. 5.3. Механізм виклику нового Activity і повернення з нього з результатом

Практична частина

Завдання 1. Вивчіть приклад підключення до мережі. public void myClickHandler (View view) {

...

ConnectivityManager connMgr = (ConnectivityManager) getSystemService (Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo ();if (networkInfo! = null && networkInfo.isConnected ()) {

// fetch data

} Else {

// display error

}

...

}

Завдання 2. Вивчіть код додатків

HttpURLConnection

```
private String downloadUrl (String myurl) throws IOException {InputStream
is = null;
// Only display the first 500 characters of the retrieved
//web page content. int len =
500;
try {
URL url = new URL (myurl);
HttpURLConnection conn = (HttpURLConnection)url.openConnection ();
conn.setReadTimeout (10000/*milliseconds * /);conn.setConnectTimeout
(15000 / * milliseconds * /); conn.setRequestMethod ( "GET"); conn.setDoInput
(true);
// Starts the query conn.connect ();
int response = conn.getResponseCode ();
Log.d (DEBUG_TAG, "The response is:" + response); is =
conn.getInputStream ();
//Convert the InputStream into a string String
contentAsString = readIt (is, len); return contentAsString;
// Makes sure that the InputStream is closed after the app is
// finished using it.
} Finally {
if (is! = null) {
is.close ();
}
}
}
```

Перетворення отриманої інформації до типу Srting

```
public String readIt (InputStream stream, int len) throws
```

```
IOException,UnsupportedEncodingException {
    Reader reader = null;
    reader = new InputStreamReader (stream, "UTF-8");char []
buffer = new char [len];
    reader.read (buffer); return new
String (buffer);
    }
```

Http GET запит

```
    Створюємо HttpClient
    HttpClient client = new DefaultHttpClient ();
    Створюємо об'єкт HttpGet
    HttpGet request = new HttpGet ( "http://www.example.com");
    Виконуємо HTTP запит
    HttpResponse response;
    try {
    response =client.execute (request);
    Log.d ( "Response of GET request", response.toString ());
    } Catch (ClientProtocolException e) {
    // TODO Auto-generated catch blocke.printStackTrace ();
    } Catch (IOException e) {
    // TODO Auto-generated catch blocke.printStackTrace ();
    }
```

Взаємодія з сервером через сокети

```
    public class Requester extends Thread {Socket
requestSocket;
    String message;
    StringBuilder returnStringBuffer = new StringBuilder ();Message lmsg;
    int ch;
    @Override public void run () {try {
```

```
this.requestSocket = new Socket ( "remote.servername.com", 13);
InputStreamReader isr = new
InputStreamReader (this.requestSocket. GetInputStream (), "ISO-8859-1 ");
while ((this.ch = isr.read ())! = -1) {this.returnStringBuffer.append ((char)
this.ch);
}
this.message = this.returnStringBuffer.toString ();this.lmsg =
new Message ();
this.lmsg.obj = this.message; this.lmsg.what = 0;
h.sendMessage (this.lmsg);
this.requestSocket.close ();
}
catch (Exception ee) {
Log.d ( "sample application ", "failed to read data " +
ee.getMessage ());
}
}
}
```

Завдання 3. Робота з зовнішніми файлами.

Розробити мобільний додаток, що дозволяє користувачеві асинхронно завантажувати файли журналу. Файли зберігаються на сервері в форматі PDF і розташовані за адресою:

http://chemengine.kpi.ua

Необхідно передбачити повідомлення про відсутність файлу. У разі якщо файл не знайдений, відповідь від сервера буде містити головну сторінку сайту.

Приклади посилань:

http://chemengine.kpi.ua/issue/view/13836 - повернуто PDF файл

http://chemengine.kpi.ua/issue/view/13837 - файл не знайдений, повернута головна сторінка сайту

Файли повинні зберігатися на пристрої в папці, яка створюється при першому запуску програми. Після закінчення завантаження файлу повинна ставати доступною кнопка «Дивитися» і кнопка «Видалити».

При натисканні на кнопку «Дивитися» має відбуватися відкриття збереженого на пристрої файлу. Передбачити помилку, якщо в ньому не встановлено додаток, що відкриває PDF файли. При натисканні на кнопку «Видалити» завантаження повинен віддалятися з пристрою.

Завдання 4. Зберігання та читання налаштувань.

При запуску програми користувачеві повинно виводитися спливаюче напівпрозоре повідомлення (popupWindow), з короткою інструкцією із користування послугою (можете написати випадковий текст), чекбокс «Більше не показувати» і кнопкою «ОК».

Якщо чекбокс був відзначений і натиснута кнопка ОК, необхідно зробити збереження даного параметра використовуючи SharedPreferences. При наступному запуску програми проводити перевірку параметра, і не виводити спливаюче повідомлення, якщо чекбокс був відзначений.

Контрольні питання

1. Для чого використовуються об'єкти класу Intent?
2. Як виконати явний виклик Activity?
3. Як виконати неявний виклик Activity?
4. Які стани передбачені життєвим циклом Activity?
5. Які методи автоматично спрацьовують при зміні стану Activity? Як можна використовувати ці методи?
6. Як виконується передача даних за допомогою Intent?
7. В якому вигляді зберігаються даних за допомогою класу SharedPreferences?
8. В яких випадках доцільно зберігати дані за допомогою класу SharedPreferences?

ЛАБОРАТОРНА РОБОТА 6
ПОВІДОМЛЕННЯ

Мета роботи: вивчити інструменти зберігання даних та реалізації повідомлень.

Теоретичні положення

Для розробки багатофункціональних інтерфейсів Android-додатків необхідно навчитися створювати свої власні набори View-елементів в окремому xml-файлі з подальшим підключенням створеної графіки.

Створення View-елемента з вмісту layout-файлу

Клас LayoutInflater використовується для створення View-елемента з окремого layout-файлу. Алгоритм дій в даному випадку наступний:

1. Створена xml-файл з необхідним View-елементом або набором View-елементів (наприклад, my_view.xml).

2. У Activity, до якого потрібно додати вміст my_view.xml, знаходимо id того layout, в якому і розміститися View-елемент (наприклад, lin_layout).

3. У методі onCreate створюємо об'єкт класу LayoutInflater: LayoutInflater layoutInflater = LayoutInflater.from (this);

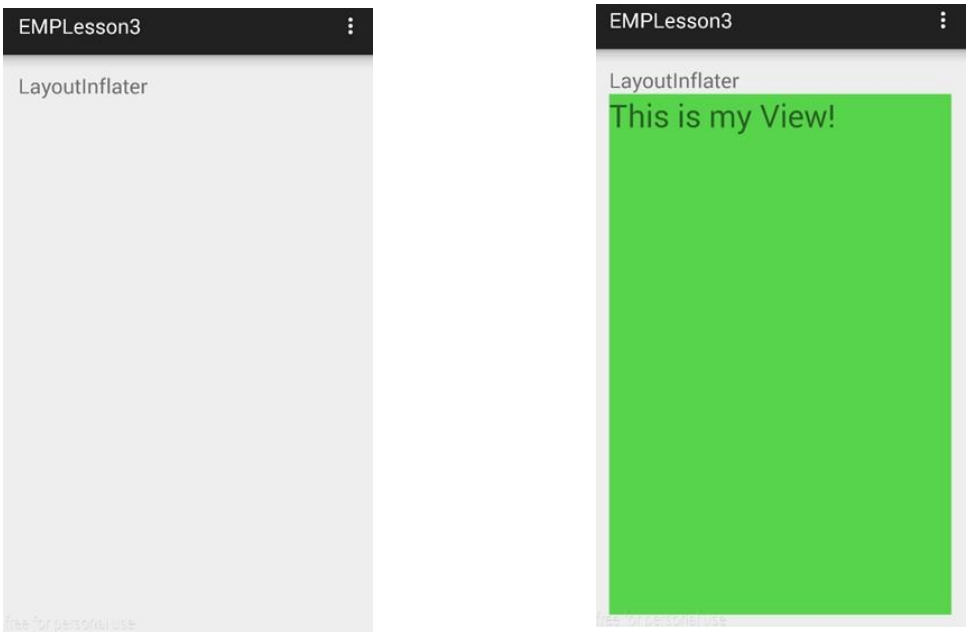
4. Викликаємо з-під об'єкта layoutInflater метод inflater:

View view = layoutInflater.inflate (R.layout.my_view, lin_layout, true); Синтаксис методу inflater:

public View inflate (int resource, ViewGroup root, boolean attachToRoot); View;

Тут resource - id layout-файлу, який буде використаний для створення root - батьківський ViewGroup-елемент для створюваного View; attachToRoot - параметр, що визначає чи приєднувати створюваний View до root. Щоб root став батьком створюваного View необхідно вказати true [1]. Приклад повного варіанту коду з додаванням View-елемента з layout-файлу як дочірнього наведено нижче на рис.

6.1.



а *б*

а - первинний вигляд інтерфейсу; б - інтерфейс з додаванням дочірнього View-елемента з вмісту layout-файлу

Рис. 6.1 - Приклад додаванням View-елемента з вмісту layout-файлу

```
public class MainActivity extends
  ActionBarActivity {private LinearLayout
  lin_layout;
  private LayoutInflater layoutInflater;
  @Override
  protected void onCreate (Bundle savedInstanceState) {
   super.onCreate
   (savedInstanceState);
   setContentView
   (R.layout.activity_main);  initView
   ();
}
```

```
    private void initView () {
        lin_layout = (LinearLayout) findViewById (R.id.lin_layout);
        layoutInflater = LayoutInflater.from (this);
        View view = layoutInflater.inflate (R.layout.my_view, lin_layout, true);
    }
}
```

Список ListView

Список ListView містить однотипні по дизайну пункти і логіку взаємодії з елементами цих пунктів (рис. 6.2). Кожен пункт списку є ViewGroup, створюваний в окремому layout-файлі.

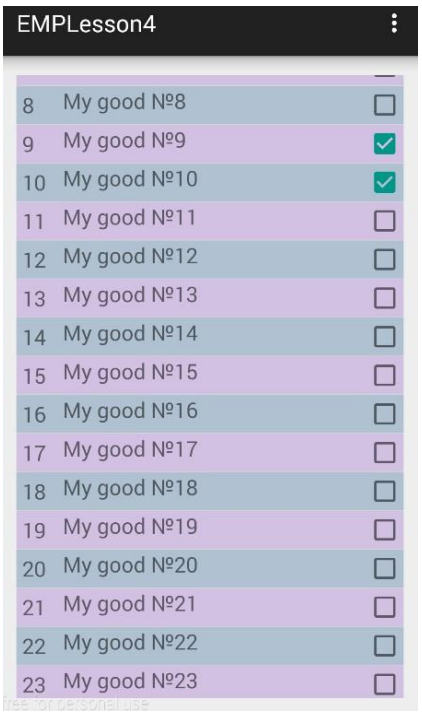


Рис. 6.2 - Приклад реалізації списку ListView

Стратегія створення списку наступна:

1. Створюємо в папці layout xml-файл з графічним відображенням одного пункту списку. Для коректного відображення набору пунктів в результуючому списку в якості layout_height вказуємо wrap_content або висоту в dp.

2. Створюємо набір даних для подальшого наповнення списку.

3. Програмуємо свій власний адаптер. Адаптер - структурний шаблон проектування, призначений для створення класу-оболонки з необхідним інтерфейсом.

Адаптеру призначаємо набір даних для наповнення списку і layout-ресурс з візуальним поданням одного пункту списку.

4. Надаємо адаптер списку ListView. Список при побудові запраши- кість у адаптера пункти, адаптер їх створює (використовуючи дані і layout-файл) і повертає списку. У підсумку ми бачимо готовий список.

Розглянемо детально викладену стратегію. Як приклад розробити конструкцію магазин товарів (Рис. 6.2).

Створимо модель даних - клас Good з полями int id (номер товару), String name (найменування товару), boolean check (прапор, що відображає вибір товару користувачем), конструктором класу і методами getId (), getName (), isCheck (), setId (int id), setName (String name), setCheck (boolean check).

Розробимо графічне представлення одного пункту списку - файл item_good.xml, що містить два TextView і один CheckBox для відображення параметрів товару.

У activity_main.xml в кореневій layout додаємо ListView з розмірами wrap content.

Створюємо клас GoodsAdapter для реалізації кастомизировать адаптера.

У MainActivity.java знаходимо listView по id; створюємо динамічний масив

ArrayList <Good> arr_goods і імітуємо інтернет-магазин шляхом генерації об'єктів класу Good. Створюємо об'єкт goodsAdapter від класу GoodsAdapter: для цього в конструктор класу передаємо параметр context і колекцію об'єктивним товарr_goods. Далі присвоюємо goodsAdapter списку за допомогою методу setAdapter.

public class MainActivity extends ActionBarActivity {private ListView listView;

```
    private ArrayList <Good> arr_goods = new ArrayList <Good> ();
    private final int SIZE_OF_ARR
    = 25; private GoodsAdapter
    goodsAdapter;
  @Override
  protected void onCreate (Bundle savedInstanceState) {super.onCreate
    (savedInstanceState); setContentView (R.layout.activity_main);
    initView ();
    createMyListView ();
  }

  private void initView () {
    listView = (ListView) findViewById (R.id.listView);
  }

private void
    createMyListView ()
    {fillData ();
    goodsAdapter = new GoodsAdapter (this, arr_goods);
    listView.setAdapter (goodsAdapter);
  }

  private void
    fillData () {int i
    = 0;
    while (i <SIZE_OF_ARR) {i ++;
      arr_goods.add (new Good (i, "" + "My good №" + i, false));
    }
  }
```

}

Розглянемо реалізацію класу GoodsAdapter. Наслідуючи від базового адаптера BaseAdapter. Для обробки подій від чекбоксів реалізуємо інтерфейс CompoundButton.OnCheckedChangeListener.

public class GoodsAdapter extends BaseAdapter implements CompoundButton.OnCheckedChangeListener {

```
private Context context;
private ArrayList <Good> arr_goods_adapter;
private LayoutInflater layoutInflater;

public GoodsAdapter (Context context, ArrayList <Good>
   arr_goods_adapter) {this.context = context;
   this.arr_goods_adapter = arr_goods_adapter;
   this.layoutInflater = LayoutInflater.from
   (context);
}

// кількість елементів @Override
public int getCount () {
   return arr_goods_adapter.size ();
}

// елемент по позиції @Override
public Object getItem (int position) {return arr_goods_adapter.get
   (position);
}
```

```
// id по
позиції
@Override
public long getItemId (int position) {return position;
}

// пункт
списку
@Override
public View getView (int position, View convertView, ViewGroup
   viewGroup) {View view = convertView;
   if (view == null) {
      view = layoutInflater.inflate (R.layout.item_good, null, false);
    }

   Good good_temp = arr_goods_adapter.get (position);

   TextView tv_goodId = (TextView) view.findViewById (R.id.tv_goodId);

tv_goodId.setText (Integer.toString (good_temp.getId ()));

TextView tv_goodName = (TextView) view.findViewById
(R.id.tv_goodName); tv_goodName.setText (good_temp.getName ());

CheckBox cb_good = (CheckBox) view.findViewById
(R.id.cb_good); cb_good.setChecked (good_temp.isCheck ());
cb_good.setTag (position);
```

```
        cb_good.setOnCheckedChangeListener (this);

        return view;
        }

        @Override
        public void onCheckedChanged (CompoundButton compoundButton, boolean
isChecked) {
            if (compoundButton.isShown ()) {
                int i = (int) compoundButton.getTag ();
                arr_goods_adapter.get (i) .setCheck
                (isChecked); notifyDataSetChanged ();
            }
          }
        }
```

У конструкторі заповнюємо внутрішні змінні і отримуємо LayoutInflater для роботи з layout-ресурсами. У arr_goods_adapter зберігається список товарів.

Методи, відмічені анотацією @Override, необхідно реалізувати при спадкуванні BaseAdapter. Ці методи використовуються списком і повинні працювати коректно.

Метод getCount повинен повертати кількість елементів списку.

Метод getItem повинен повертати елемент по зазначеній позиції. Використовуючи позицію, отримуємо конкретний елемент з arr_goods_adapter.

Метод getItemId повинен повертати id елемента (повертаємо поточну позицію).

Метод getView повинен повертати View пункту списку. Для цього раніше з- здавай layout-ресурс R.layout.item_good. У цьому методі необхідно з R.layout.item_good створити View, заповнити його даними і віддати списку.

Перед тим як створювати, пробуємо використовувати convertView, який йде на вхід методу. Це вже створене раніше View, але неиспользуемое в даний момент. Наприклад, при прокручуванні списку, частина пунктів йде за екран і їх уже не треба промальовувати. View з цих «невидимих» пунктів використовуються для воссозда- ня графіки та оновлення даних. Це значно прискорює роботу програми, т. К. Не треба промальовувати inflate зайвий раз.

Якщо ж convertView не надійшов (null), то створюємо самі View.

Далі заповнюємо параметри товару, чекбокси присвоюємо обробник, з- що зберігаються в Tag позицію елемента. Tag - це подібність Object-сховища у кожного View, куди можна помістити необхідні дані. У нашому випадку для кожного чекбокса поміщаємо в його Tag номер позиції пункту списку. Далі в обробнику чекбокса буде можливість цей номер позиції витягти і визначити, в якому пункті списку був натиснутий чекбокс.

У підсумку, метод getView повертає списку повністю заповнений view, і список його відобразить як черговий пункт.

Оброблювач для чекбоксів onCheckedChanged. При натисканні на чекбокс в списку він спрацьовує, читає з Tag позицію пункту списку і позначає соответ- ствующий товар як покладений в кошик. Без цього обробника не працював би поміщення товарів у кошик, а на екрані значення чекбоксів в списку губились би при прокручуванні, тому що пункти списку перестворює, якщо вони вухо- дять «за екран» і знову з'являються. Це пересозданіе забезпечує метод getView, а він для заповнення View бере дані з товарів. Значить, при натисканні на чекбокс, обов'язково треба зберегти в даних про товар інформацію про те, що він тепер в кошику. Метод notifyDataSetChanged () повідомляє список про зміну даних, щоб оновити список на екрані [1-3].

Header і Footer в списках

Header і Footer - це View-елементи, які можуть бути додані до списку зверху і знизу. Для цього необхідно створити графічні уявлення header_mygoods.xml і footer_mygoods.xml, перетворити їх в View-елементи і

надати списку за допомогою методів addHeader або addFooter. Обов'язковою умовою відображення Header і Footer є їх додавання до списку до того, як присвоюється адаптер.

Модифікуємо код MainActivity.java:

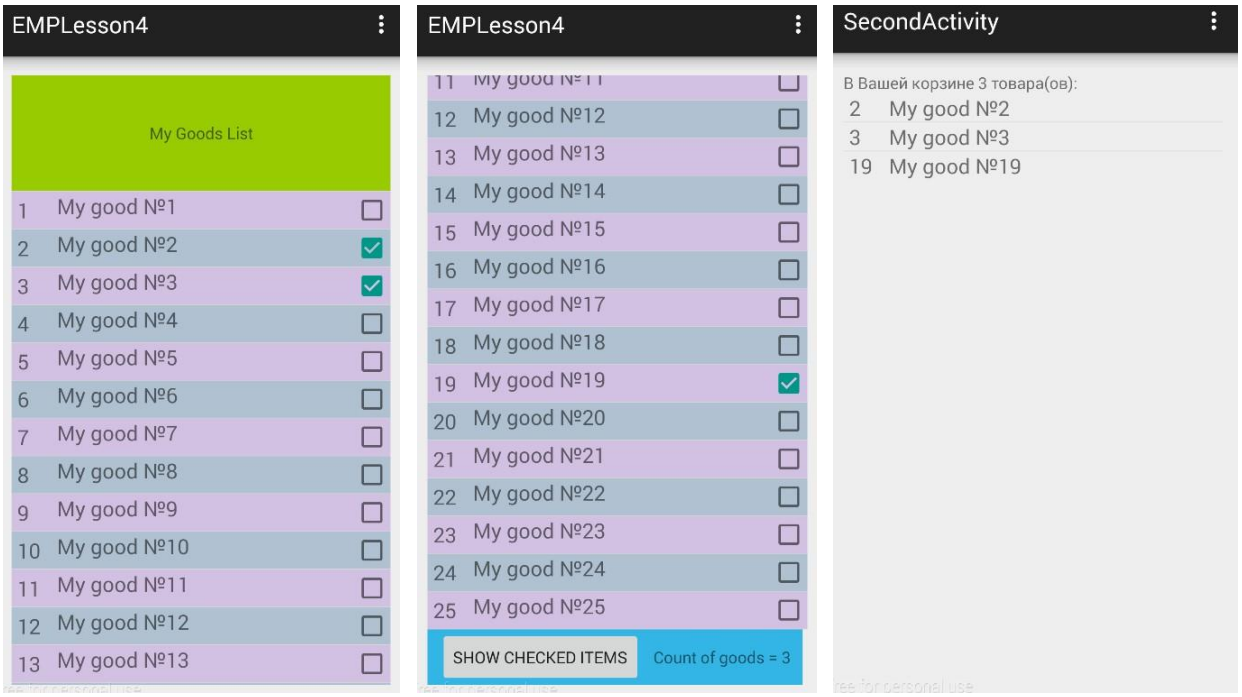
```
public class MainActivity extends ActionBarActivity
        implements View.OnClickListener {

    // додаємо нові змінні класу private LayoutInflater layoutInflater;
    private View view_header, view_footer; private Button btnShow;
    private TextView tv_count;

    // додаємо Header і Footer private void createMyListView () {
        fillData ();
        goodsAdapter = new GoodsAdapter (this, arr_goods, this);

        layoutInflater = LayoutInflater.from (this);
        view_header = layoutInflater.inflate (R.layout.header_mygoods,
        null); view_footer = layoutInflater.inflate
        (R.layout.footer_mygoods, null); btnShow = (Button)
        view_footer.findViewById (R.id.btnShow);
        btnShow.setOnClickListener (this);
        tv_count = (TextView) view_footer.findViewById (R.id.tv_count);

        listView.addHeaderView
        (view_header);
        listView.addFooterView
        (view_footer);

        listView.setAdapter (goodsAdapter);
```

}

Приклад реалізації списку з Header наведено на рис. 6.3, а, з Footer - на рис. 6.3, б.

Механізми зворотного виклику для обробки подій в Android- додатках

В продовження прикладу зі списком товарів магазину поставимо таку задачу: зазначені товари повинні зберігатися в окремому динамічному масСиве для подальшого відображення кошика товарів користувача в новому Activity (рис. 6.3, в).



а *б* *в*

а - список з Header; б - список з Footer; в - корзина товарів

Рис. 6.3 - Приклад реалізації списку ListView

Особливістю реалізації даного завдання є необхідність обробки події, настання якого відстежує клас GoodsAdapter, в класі MainActivity.

Практична частина

Завдання 1. Розробити мобільну прикладну програму, що дозволяє встановити нагадування.

Вимоги до програми:

1) Створення нагадувань і збереження їх в базу даних. В базі даних повинні зберігатися наступні значення (Тема, Текст повідомлення, Дата повідомлення);

2) Перегляд повідомлень;

3) Видалення повідомлень;

4) Дата нагадування повинна встановлюватися за допомогою TimePickerDialog і DatePickerDialog;

5) Стилізувати нагадування в Notification Center і Status bar (встановити власне лого);

6) При натисканні на повідомлення в Notification Center переходити в активацію додатку з повним текстом повідомлення.

Контрольні питання

1. Як створити View-елемент з вмісту Layout-файлу? У яких випадках це необхідно?

2. Як створити і забезпечити роботу списку ListView?

3. Як реалізувати власний кастомізованих список?

4. Що собою являє і як реалізувати Header в списках?

5. Що собою являє і як реалізувати Footer в списках?

6. Які Ви знаєте механізми зворотного виклику для обробки подій в Android-додатках?

7. Як передати динамічний масив об'єктів з одного Activity в інше за допомогою Intent?

СПИСОК ЛІТЕРАТУРИ

1. Статистика Android vs iOS в 2021 году. Разбираем актуальные цифры и факты/Где Трафик? — 2021. http://gdetraffic.com/Analitika/Android_vs_iOS.
2. Friesen, J. Java XML and JSON: Document Processing for Java SE. Second Edition. — Berkeley (CA): Apress, 2019. — 546 p. DOI: 10.1007/978-1-4842-4330-5.
3. Mikkonen T. Programming mobile devices: an introduction for practitioners. - London: John Wiley & Sons Ltd., 2007. - 245 p.
4. Paavilainen J. Mobile business strategies - understanding the technologies and opportunities. - London: IT Press, 2002. - 257 p.
5. Lee V., Schneider H., Schell R. Mobile Applications: architecture, design, and development. - Prentice Hall, 2004. - 368 p.
6. Fling B. Mobile design and development: practical concepts and techniques for creating mobile sites and web apps. - O'Reilly Media, 2009. -336 p.
7. Verbraeck A. Designing mobile service systems. - Amsterdam: IOS Press, 2007. - 249 p.
8. Zheng P., Lionel N. Smart Phone and next-generation mobile computing. - Morgan Kaufmann, 2005. - 350 p.
9. Friesen J. Learn Java for Android development. - Apress, 2010. - 656 p.
10. Jackson W. Android apps for absolute beginners. - Apress, 2011. - 344 p.
11. Burnette E. Hello, Android: introducing Google's mobile development platform. - Pragmatic Bookshelf, 2010. - 300 p.
12. Ableson WF, Sen R., King C. Android in action. - Manning Publications, 2011. - 592 p.
13. Rogers R, Lombardo J, Mednieks Z, Blake Meike G. Android application development: programming with the Google SDK. - O'Reilly Media, 2009. - 336 p.
14. Murphy ML Android programming tutorials. - CommonsWare, 201L- 334 p.

15. Meier R. Professional Android 2 application development. - Wrox, 2010. -576 p.
16. Sayed Y. Hashimi. Pro Android 2. - Apress, 2010. - 500 p.
17. Conder S., Darcey L. Android wireless application development. - Addison-Wesley Professional, 2009. - 600 p.
18. To N., Steele J. The Android developer's cookbook: building applications with the Android SDK (Developer's Library). - Addison-Wesley Professional, 2010. - 400 p.
19. Di Marzio J.F. Android: a programmer's guide. - McGraw-Hill Osborne Media, 2008. - 400 p.
20. Komatineni S., MacLean D., Hashimi S. Pro Android 3. - Apress, 2011. - 1200 p.
21. Корнієнко Б.Я. Дослідження імітаційного полігону захисту критичних інформаційних ресурсів методом IRISK. Моделювання та інформаційні технології. 2018. Вип. 83. С. 34-41.
22. Корнієнко Б.Я. Побудова та тестування імітаційного полігону захисту критичних інформаційних ресурсів. Наукоємні технології. 2017. № 4 (36). С. 316 - 322.
23. Korniyenko B., Yudin A., Galata L. Risk estimation of information system.Wschodnioeuropejskie Czasopismo Naukowe. 2016. № 5. P. 35 - 40.
24. Корнієнко Б.Я., Юдін О.К., Снігур О.С.Безпека аутентифікації у web-ресурсах. Захист інформації. 2012. № 1 (54). С. 20 -25. DOI: 10.18372/2410-7840.14.2056 (ukr).
25. Корнієнко Б.Я., Максімов Ю.О., Марутовська Н.М. Прикладні програми управління інформаційними ризиками. Захист інформації. 2012. № 4 (57). С. 60 – 64. DOI: 10.18372/2410-7840.14.3493 (ukr).
26. Galata, L., Korniyenko, B., Yudin, A.: Research of the simulation polygon for the protection of critical information resources. In: CEUR Workshop Proceedings, Information Technologies and Security, Selected Papers of the XVII International Scientific and Practical Conference on Information

Technologies and Security (ITS 2017), 30 Nov 2017, Kyiv, Ukraine. vol. 2067. pp. 23–31., urn:nbn:de:0074-2067-8.

27. Корнієнко Б.Я. Безпека інформаційно-комунікаційних систем та мереж. Навчальний посібник для студентів спеціальності 125 «Кібербезпека». – К.: НАУ, 2018. – 226 с.
28. Корнієнко Б.Я., Галата Л.П. Оптимізація системи захисту інформації корпоративної мережі. Математичне та комп'ютерне моделювання. Серія: Технічні науки, Випуск 19, 2019. - С. 56-62.
29. Korniyenko B., Galata L. Implementation of the information resources protection based on the CentOS operating system. Conference Proceedings of 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON -2019) July 2 – 6, 2019, Lviv, Ukraine. - pp. 1007-1011.
30. Галата Л.П., Корнієнко Б.Я., Заболотний В.В. Математична модель протидії загрозам у системі захисту критичних інформаційних ресурсів. Наукоємні технології, Том 43, № 3, 2019. – С. 300 – 306.
31. Корнієнко Б.Я. Modeling of information security system in computer network. Безпека інформаційних систем і технологій, Том №1 (1), 2019. – С.36-41.
32. Korniyenko B., Galata L., Ladieva L. Research of Information Protection System of Corporate Network Based on GNS3. Conference Proceedings of 2019 IEEE Inernational Conference on Advanced Trends in Information Theory (IEEE ATIT -2019) Dezember 18 – 20, 2019, Kyiv, Ukraine. - pp. 244-248.
33. Korniyenko B., Galata L., Ladieva L. Mathematical model of threats resistance in the critical information resources protection system. CEUR Workshop Proceedings, Selected Papers of the XIX International Scientific and Practical Conference "Information Technologies and Security" (ITS 2019) Kyiv, Ukraine, November 28, 2019. Vol-2577. P.281-291.
34. Корниенко Б.Я. Кибернетическая безопасность – операционные системы и протоколы. ISBN 978-3-330-08397-4, LAMBERT Academic Publishing, Saarbrucken, Deutschland. − 2017. – 122 с.

35. Korniyenko B.Y., Galata L.P. Design and research of mathematical model for information security system in computer network. Науковий журнал «Наукоємні технології». – 2017, № 2 (34), С. 114 - 118.
36. Корниенко Б.Я. Информационная безопасность и технологии компьютерных сетей : монография. ISBN 978-3-330-02028-3, LAMBERT Academic Publishing, Saarbrucken, Deutschland. − 2016. – 102 с.
37. Korniyenko B., Galata L., Kozuberda O. Modeling of security and risk assessment in information and communication system. Sciences of Europe. – 2016. – V. 2. – No 2 (2). – P. 61 -63.
38. Korniyenko B. The classification of information technologies and control systems. International scientific journal. – 2016. –№ 2. – P. 78 - 81.
39. Корнієнко Б.Я. Інформаційні технології оптимального управління виробництвом мінеральних добрив: монографія. – К.: Вид-во Аграр Медіа Груп, 2014. – 288 с.
40. Korniyenko B., Galata L., Ladieva L. Security Estimation of the Simulation Polygon for the Protection of Critical Information Resources / B. Korniyenko, //CEUR Workshop Proceedings, Selected Papers of the XVIII International Scientific and Practical Conference "Information Technologies and Security" (ITS 2018) Kyiv, Ukraine, November 27, 2018, Vol-2318, - P. 176-187, urn:nbn:de:0074-2318-4
41. Kornienko Y.M., Liubeka A.M., Sachok R.V., Korniyenko B.Y. Modeling of heat exchangement in fluidized bed with mechanical liquid distribution // ARPN Journal of Engineering and Applied Sciences. 2019. №14(12). P. 2203-2210.
42. Korniyenko B., Galata L., Ladieva L. Security Estimation of the Simulation Polygon for the Protection of Critical Information Resources // CEUR Workshop Proceedings, Selected Papers of the XVIII International Scientific and Practical Conference "Information Technologies and Security" (ITS 2018). Kyiv. Ukraine. 2018. №2318. P. 176-187.

43. Корнієнко Б.Я. Дослідження моделі взаємодії відкритих систем з погляду інформаційної безпеки // Наукоємні технології. 2012. №3(15). С. 83-89.
44. Korniyenko B., Yudin O., Novizkij E. Open systems interconnection model investigation from the viewpoint of information security // The Advanced Science Journal. 2013. №8. P. 53-56.
45. Zhulynskyi A.A., Ladieva L.R., Korniyenko B.Y. Parametric identification of the process of contact membrane distillation // ARPN Journal of Engineering and Applied Sciences Volume 14. 2019. №17. P. 3108-3112.
46. Korniyenko, B., Ladieva, L., Galata, L. Control system for the production of mineral fertilizers in a granulator with a fluidized bed. ATIT 2020 - Proceedings: 2020 2nd IEEE International Conference on Advanced Trends in Information Theory, 2020, pp. 307–310.
47. Корнієнко Б. Я., Галата Л. П. Дослідження імітаційного полігону захисту критичних інформаційних ресурсів методом IRISK. Моделювання та інформаційні технології. 2018. Вип. 83. С. 34–42.