**МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ**
**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ВНУТРІШНІХ СПРАВ**

**Кафедра протидії кіберзлочинності факультету № 4**

# МЕТОДИЧНІ МАТЕРІАЛИ
## ДО ПРАКТИЧНИХ ЗАНЯТЬ

навчальної дисципліни
«Розробка захищених мобільних застосувань»
обов'язкових компонент освітньої програми
другого (магістерського) рівня вищої освіти

**125 Кібербезпека та захист інформації**
**(безпека інформаційних та комунікаційних систем)**

**Харків 2023**

**ЗАТВЕРДЖЕНО**
Науково-методичною радою
Харківського національного
університету внутрішніх справ
Протокол від 30.08.2023 № 7

**СХВАЛЕНО**
Вченою радою факультету № 6
Протокол від 16.08.2023 № 8

**ПОГОДЖЕНО**
Секцією Науково-методичної ради
ХНУВС з технічних дисциплін
Протокол від 29.08.2023 № 7

Розглянуто на засіданні кафедри протидії кіберзлочинності факультету № 4 (протокол від 15.08.2023 № 19)

**Розробники:**
1. Доцент кафедри протидії кіберзлочинності факультету № 4, кандидат технічних наук, доцент Клімушин П.С.

**Рецензенти:**

1. Завідувач кафедри інформаційних управляючих систем ХНУРЕ, д.т.н., професор Петров К. Е.

2. Провідний науковий співробітник Науково-дослідної лабораторії з проблем розвитку інформаційних технологій ХНУВС, к.т.н., доцент Мордвинцев М.В.

**ЗМІСТ**

## Загальні методичні вказівки

Навчання з дисципліни «Розробка захищених мобільних застосувань» проходить у формі: лекцій(20 год.), практичних (10 год.) та лабораторних занять (10 год.), а також самостійної роботи (90 год.). Метою лекційного курсу є розкриття основних категорій, особливостей тематики та проблемних аспектів відповідних тем.

Аудиторні заняття проводяться у формі практичних занять, на яких студенти під керівництвом викладача засвоюють навички розв'язання практичних задач шляхом побудови алгоритмів і подальшої їх реалізації шляхом розробки програми або розв'язання задачі за допомогою інструментальних засобів. Самостійна робота за кожною темою включає вивчення рекомендованих джерел, навчальної та науково-монографічної літератури, а також розв'язання практичних завдань, що сформульовані в методичних вказівках, та підготовку звіту про виконання лабораторної роботи.

### 1. Розподіл часу навчальної дисципліни за темами (денна форма навчання)

| Номер та назва навчальної теми | Кількість годин, відведених на вивчення навчальної дисципліни | | | | | Вид контролю |
|---|---|---|---|---|---|---|
| | Всього | з них: | | | | |
| | | лекції | Семінарські заняття | Практичні заняття | Самостійна робота | |
| Тема № 1. Бездротові локальні мережі | 9 | 2 | 2 | | 5 | |
| Тема № 2. Сучасні мобільні операційні системи та мові програмування додатків | 9 | 2 | 2 | | 5 | |
| Тема № 3. Інструменти і середовища розробки мобільних додатків | 9 | 2 | 2 | | 5 | |
| Тема № 4. Середовище розробки платформи Android | 9 | 2 | 2 | | 5 | |
| Тема № 5. Побудова інтерактивних програм | 9 | 2 | | 2 | 5 | |
| Тема № 6. Множинні активності та інтенти | 9 | 2 | | 2 | 5 | |
| Тема № 7. Користувацький інтерфейс | 9 | 2 | | 2 | 5 | |
| Тема № 8. Робота з базами даних | 9 | 2 | | 2 | 5 | |
| Тема № 9. Безпека мобільних застосувань | 9 | 2 | | 2 | 5 | |
| Тема № 10. Месенджери миттєвого обміну повідомленнями та захист мобільних і хмарних обчислювальних середовищах | 9 | 2 | 2 | | 5 | |
| **Всього за семестр № 2:** | **90** | **20** | **10** | **10** | **50** | **екзамен** |
| **Всього по дисципліні** | **90** | **20** | **10** | **10** | **50** | |

**(заочна форма навчання)**

| Номер та назва навчальної теми | Кількість годин, відведених на вивчення навчальної дисципліни | | | | | Вид контролю |
|---|---|---|---|---|---|---|
| | Всього | з них: | | | | |
| | | лекції | Семінарські заняття | Практичні заняття | Самостійна робота | |
| Тема № 1. Бездротові локальні мережі | 9 | 1 | | 0,5 | 7 | |
| Тема № 2. Сучасні мобільні операційні системи та мові програмування додатків | 9 | 1 | | 0,5 | 7 | |
| Тема № 3. Інструменти і середовища розробки мобільних додатків | 9 | 1 | | 1 | 7 | |
| Тема № 4. Середовище розробки платформи Android | 9 | 1 | | 1 | 7 | |
| Тема № 5. Побудова інтерактивних програм | 9 | 1 | | 1 | 8 | |
| Тема № 6. Множинні активності та інтенти | 9 | 1 | | 1 | 8 | |
| Тема № 7. Користувацький інтерфейс | 9 | 1 | | 1 | 8 | |
| Тема № 8. Робота з базами даних | 9 | 1 | | 1 | 8 | |
| Тема № 9. Безпека мобільних застосувань | 9 | 0,5 | | 1 | 7 | |
| Тема № 10. Месенджери миттєвого обміну повідомленнями та захист мобільних і хмарних обчислювальних середовищах | 9 | 0,5 | | 1 | 7 | |
| **Всього за семестр № 2:** | **90** | **8** | | **8** | **74** | **екзамен** |
| **Всього по дисципліні** | **90** | **8** | | **8** | **74** | |

## 2. Методичні вказівки до практичних занять

**Практичне заняття № 1. Середовище розробки платформи Android**

**Кількість годин:** 2 год**.**

**Навчальна мета заняття**:

1. Придбання теоретичних знань з теми «Середовище розробки платформи Android», розвиток здібностей до творчого мислення, формування навичок самостійної роботи з аналізу і узагальнення інформації, вміння проектувати компонентну архітектуру мобільного додатку.

**Рекомендована література:**

1. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.

2. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
3. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
4. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
5. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
6. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.
7. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
8. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.

**Матеріально-технічне забезпечення:** комп'ютерний клас.

**Навчальні питання:**

1. Установка та налаштування інструментів розробки додатків Android
2. Побудова простої програми
3. Створення віртуального пристрою Android
4. Запуск програми в емуляторі
5. Модифікація програми

1. ПОРЯДОК ПРОВЕДЕННЯ ЗАНЯТТЯ:

1.1. Проведення експрес-контролю готовності до заняття.

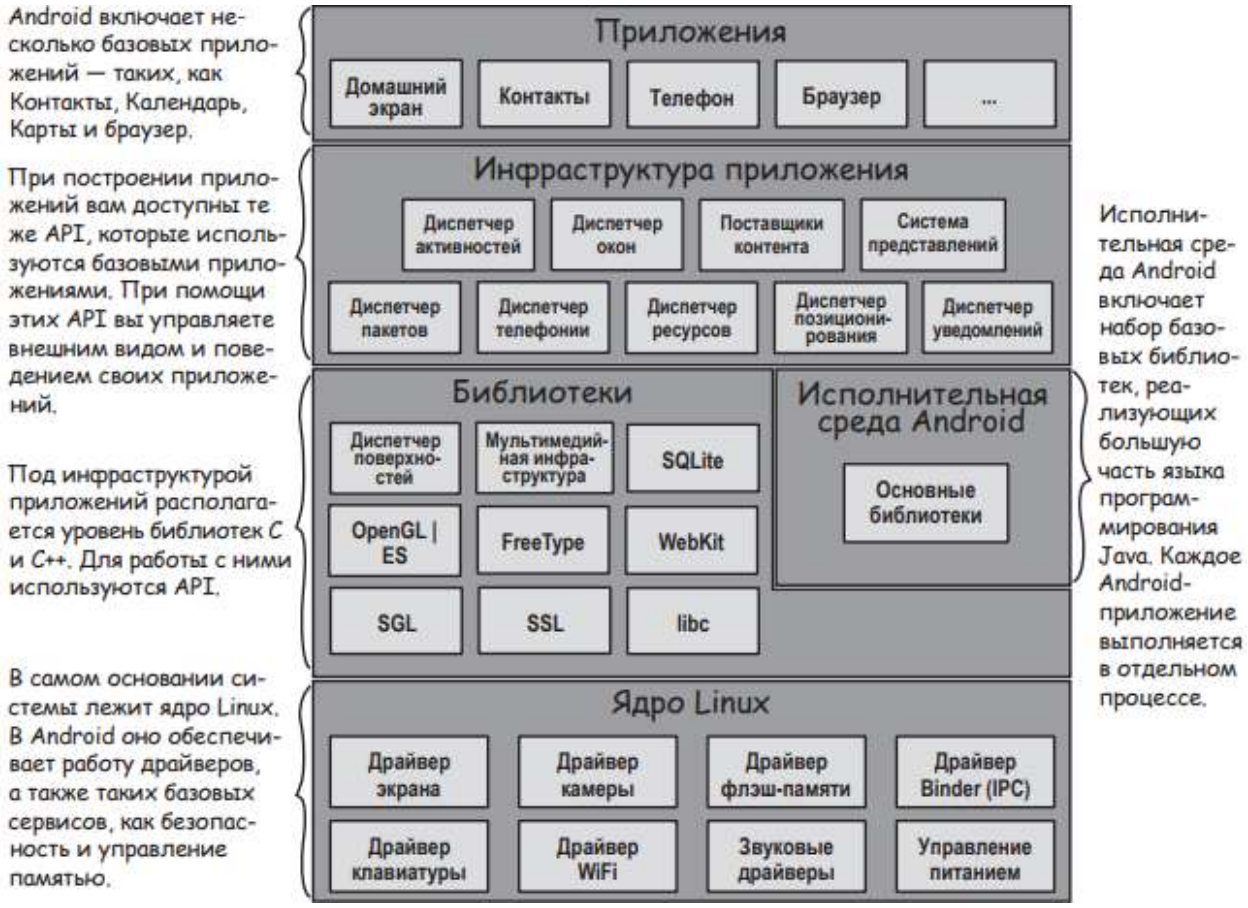
1.2. Ввести текст підготовленої програми і виконати її відлагодження.

1.3. Підібрати тести і виконати відпрацювання розробленого алгоритму на цих тестах.

1.4. Скласти звіт про виконану роботу і здати роботу викладачу.

**1. Установка та налаштування інструментів розробки додатків Android**

**Платформа Android** складається з безлічі компонентів. До неї входять базові програми (наприклад, Контакти), набір програмних інтерфейсів (API) для керування зовнішнім виглядом та поведінкою програми. а також безліч допоміжних файлів та бібліотек підтримки. Структура, що утворюється з цих компонентів, виглядає приблизно так:




Доступ до бібліотек Android надається через API в інфраструктурі додатків, і для створення додатків Android потрібно використовувати саме ці API. Для цього знадобляться лише деякі знання Java та ідея щодо створення програми. Для побудови найпростішого Android-додаток необхідно виконати лише кілька дій:

*1. Підготовка середовища розробки.* Необхідно встановити середовище Android Studio, що включає все необхідне для розробки Android-додатків.

*2. Побудова найпростішій програми*. Створимо в Android Studio простій додаток, який виводитиме текст на екрані.

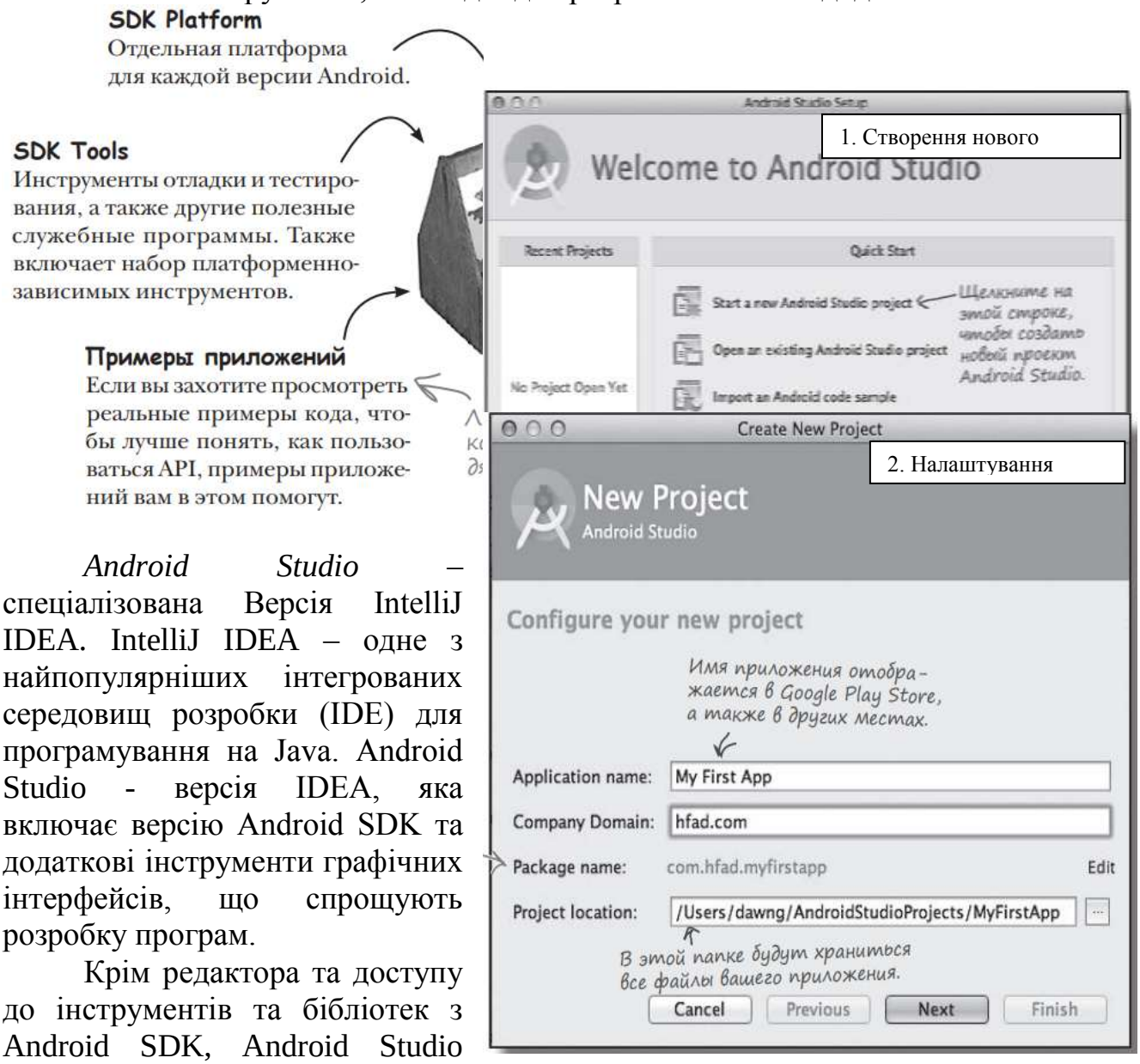
*3. Запуск програми в емуляторі Android*. Скористаємося вбудованим емулятором, щоб побачити програму в дії.

*4. Зміна програми*. Внесемо кілька змін до додатку, створеного на кроці 2, і знову запустимо його.

*Середовище розробки*. Java - найпопулярніша мова, яка використовується для розробки Android-додатків. Пристрої на базі Android не запускають файли .class та .jar. Натомість для підвищення швидкості та ефективності використання акумуляторів Android-пристрою використовують власні оптимізовані формати компільованого коду. Це означає, що не можливо скористатися звичайним середовищем розробки на мові Java - тут знадобляться спеціальні інструменти для перетворення відкомпілюваного коду у формат Android, встановлення його на Android-пристроях та налагодження програми, коли воно запрацює. Все необхідне міститься в Android SDK. Подивимося, що до нього входить.

*Android SDK*. Пакет Android Software Development Kit (SDK) містить

бібліотеки та інструменти, необхідні для розробки Android-додатків:




*Android Studio* – спеціалізована Версія IntelliJ IDEA. IntelliJ IDEA – одне з найпопулярніших інтегрованих середовищ розробки (IDE) для програмування на Java. Android Studio - версія IDEA, яка включає версію Android SDK та додаткові інструменти графічних інтерфейсів, що спрощують розробку програм.

Крім редактора та доступу до інструментів та бібліотек з Android SDK, Android Studio надає шаблони, які спрощують створення нових програм та класів, а також засоби для виконання таких операцій, як упаковка програм та їх запуск.

*Встановіть Java та Android Studio.* Android Studio - середовище розробки на мові Java, тому на комп'ютері має бути встановлена правильна версія Java. Спочатку перевірте системні вимоги Android Studio та визначте, які версії Java Development Kit (JDK) та Java Runtime Edition (JRE) знадобляться.

Коли Java успішно запрацює, завантажте Android Studio. Виконайте інструкції з встановлення, щоб встановити Android Studio на комп'ютері. Після завершення інсталяції відкрийте Android Studio і дотримуйтесь інструкцій з додавання новітніх інструментів SDK та бібліотек підтримки. Коли все зроблено, на екрані з'являється заставка Android Studio. Все готове для побудови першої Android-програми.

## 2. **Побудова простої програми**

Щоразу, коли створюється новий додаток, для нього необхідно створити

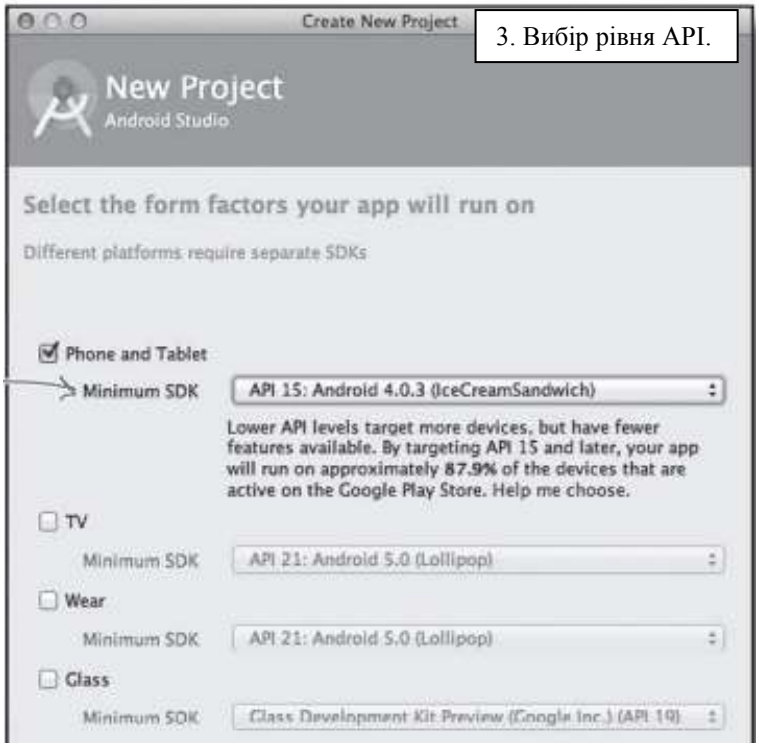
новий проект. Переконайтеся, що середовище Android Studio відкрите, і виконайте наступні шаги.

1. *Створення нового проекту*. На заставці Android Studio перелічені деякі можливі операції. Клацніть на рядку "Start a new Android Studio project".

2. *Налаштування проекту*. Тепер необхідно налаштувати конфігурацію програми: вказати, як вона називатиметься, який домен компанії буде використовуватися і де повинні зберігатися його файли. Android Studio використовує домен компанії та ім'я програми для формування імені пакета, яке буде використовуватися вашим додатком. Наприклад, якщо присвоїти ім'я “My First App” і вказати домен компанії “hfad.com”, то Android Studio сформує ім'я пакета com. hfad.myfirstapp. Ім'я пакета грає дуже важливу роль в Android, тому що воно використовується Android-пристроями для однозначної ідентифікації програми. Введіть ім'я програми “My First App”, домен компанії “hfad.com”, та підтвердьте розташування за мовчанням. Натисніть кнопку Next.

*3. Вибір рівня API.* Тепер необхідно вказати, які рівні API системи Android використовуватимуть програму. Рівні API збільшуються із виходом кожної чергової версії Android. Якщо не бажаєте, щоб програма працювала тільки на нових пристроях, варто вибрати один з більш старих рівнів API. Тут вибираємо API рівня 15; це означає, що програма зможе працювати на більшості пристроїв. Крім того, версія програми створюється тільки для телефонів та планшетів, тому прапорці інших варіантів так і залишаються знятими. Коли це буде зроблено, натисніть кнопку Next.

Відметімо, кожній версії Android надається номер і кодове ім'я. Номер версії визначає конкретну версію Android (наприклад, 5.0), тоді як кодове ім'я є трохи більш загальним «дружнім» ім'ям, яке може об'єднувати відразу кілька версій Android. Під "рівнем API" розуміється версія API, що використовуються програмою. Наприклад, Android версії 5.0 відповідає API 21. При розробці додатків Android необхідно враховувати, з якими версіями Android має бути сумісна програма.



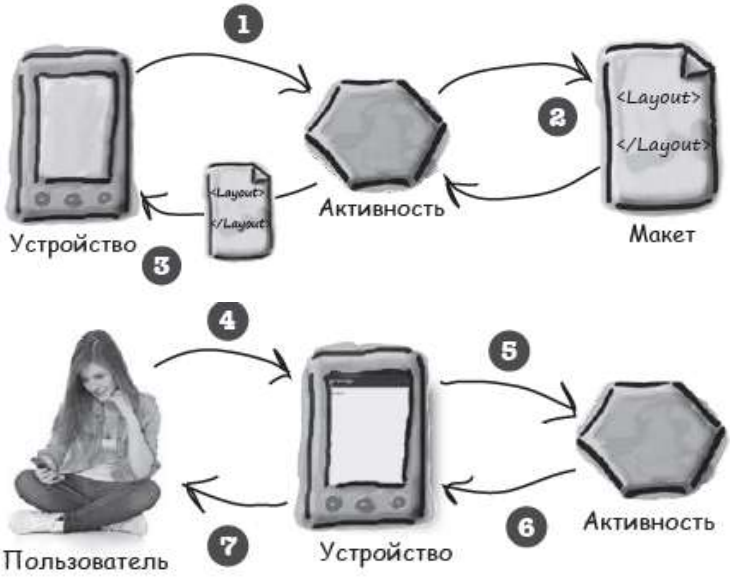
3. Вибір рівня API.

*Активності та макети*. Далі потрібно додати активність у проект. Кожний Android-додаток складається з екранів, а кожен екран складається з активності та макета. Макети визначають спосіб подання користувальницького інтерфейсу. Активності визначають дії.

*Активність* – одна чітко визначена операція, яку може виконати користувач. Наприклад, у програмі можуть бути активні для складання повідомлення електронної пошти, пошуку контакту або створення знімка.

Активності зазвичай асоціюються з одним екраном та програмуються на Java.

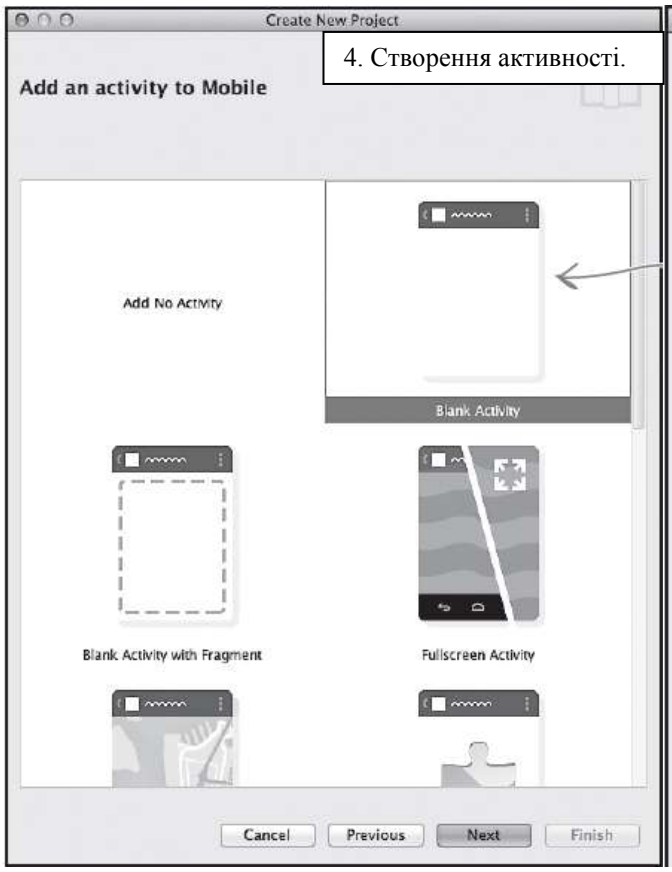
*Макет* описує зовнішній вигляд екрана. Макети створюються у вигляді файлів у розмітці XML і повідомляють Android, де розміщуються ті чи інші елементи екрану. Активності та макети, пристрої та користувач взаємодіють наступним чином.



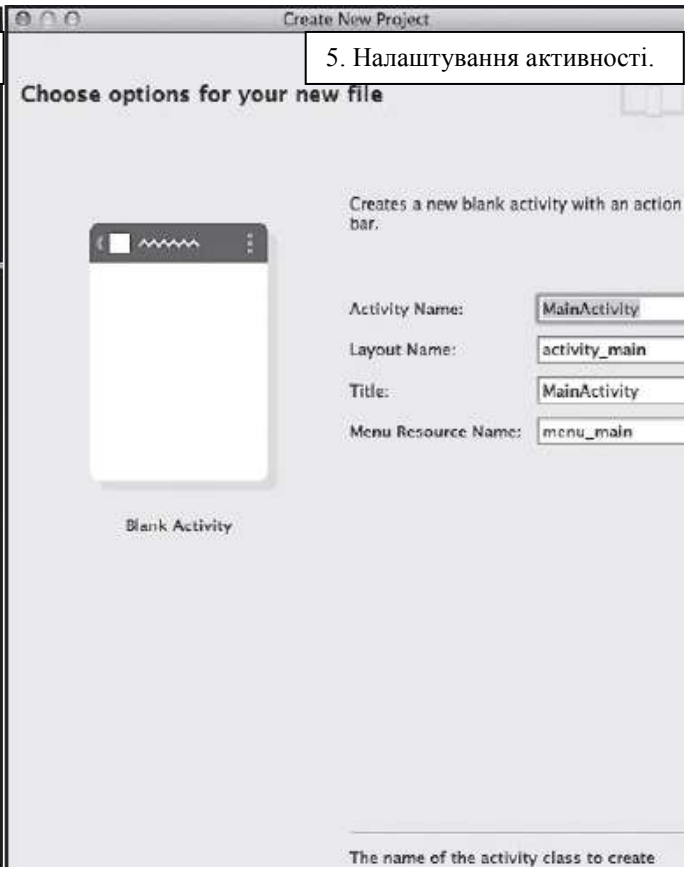

1. Пристрій запускає програму та створює об'єкт активності.
2. Об'єкт активності вказує макет.
3. Активність наказує Android вивести макет на екран.
4. Користувач взаємодіє з макетом на пристрої.
5. Активність реагує на ці взаємодії, виконуючи код програми.
6. Активність оновлює вміст екрана.
7. Користувач бачить це на пристрої.

*4. Створення активності*. На наступному екрані представлений набір шаблонів, які можуть використовуватися для створення активності та макету. Необхідно вибрати одну з них. Так як у додатку будуть використовуватися найпростіша активність та макет, виберіть варіант Blank Activity і натисніть кнопку Next.

*5. Налаштування активності*. Тепер середовище розробки запропонує вибрати імена для активності та макета екрана. Необхідно вказати текст заголовка екрана та ім'я ресурсу меню. Введіть ім'я активності “MainActivity*” та ім'я макету “activity_main”. Активність являє собою клас Java, а макет-файл з розміткою XML, тому для введених імен буде створено файл класу Java з ім'ям MainActivity.java і файл XML с іменем activity_main.xml. З натисканням кнопці Finish, Android Studio побудує програму.

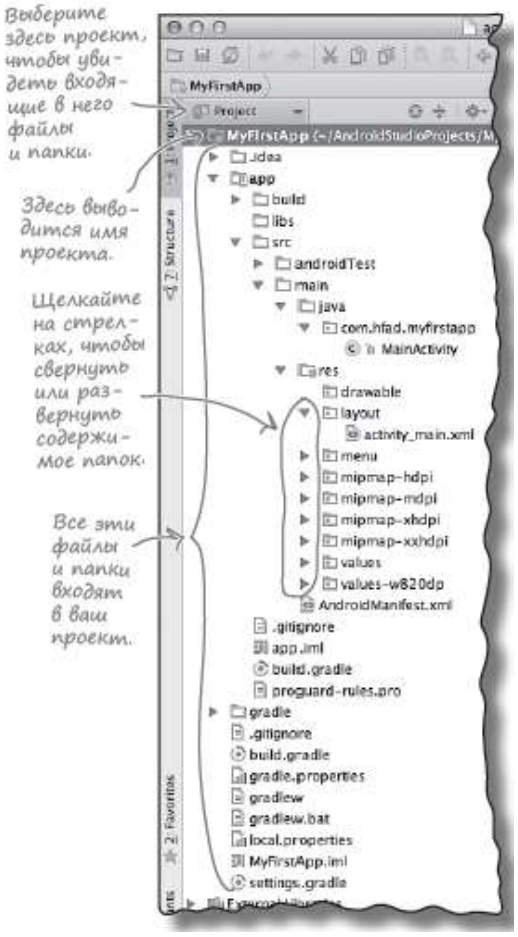



4. Створення активності.




5. Налаштування активності.

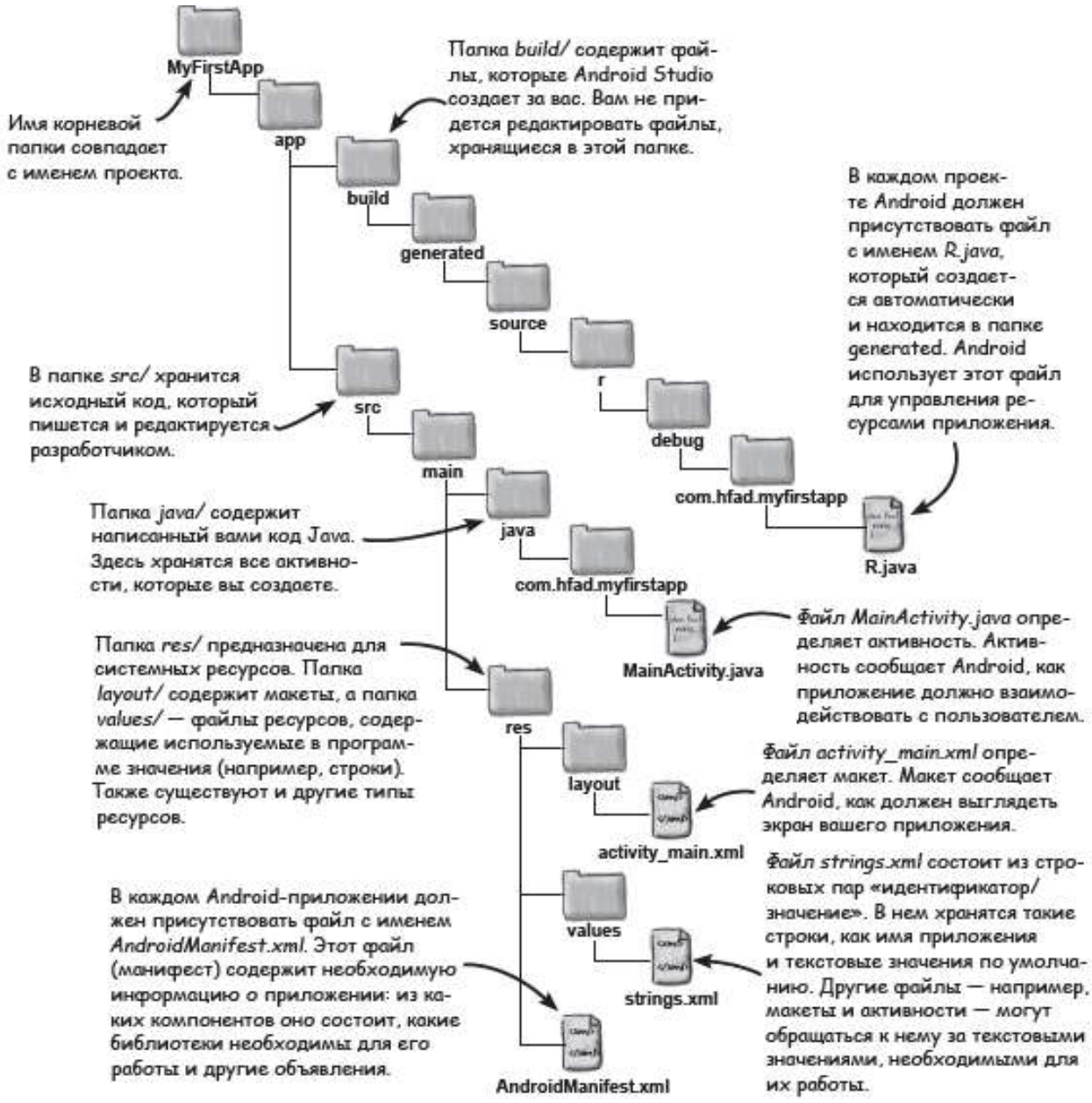
Таким чином, майстер Android Studio створив для програми проект, параметри якого були налаштовані відповідно до вказівок. Було визначино, з якими версіями Android має бути сумісній додаток, а майстер створив усі файли та папки, необхідні для найпростішої працездатної програми. Майстер створив базову активність та базовий макет із шаблонним кодом. Шаблонний код включає розмітку XML для макету та код Java для активності; він виводить текст "Hello world!" у макеті. Цей код можна змінити. З завершенням створення проекту Android Studio автоматично відобразить проект у середовищі розробки.




Android Studio створює всю структуру папок. Android-додаток насправді є набором файлів, розміщених у чітко визначеній структурі папок; Android Studio створює всі ці папки під час створення нової програми. Якщо цікаво, як виглядає ця структура папок, найпростіше переглянути її на панелі біля лівого краю вікна Android Studio. На ній відображаються всі проекти, які зараз відкриті. Щоб згорнути або розгорнути вміст папки, натисніть стрілку ліворуч від піктограми папки.

Уважніше придивимося деякі ключові файли та папки програм Android. Проекти Android Studio використовують систему складання gradle для компіляції та розгортання програм. Gradle проекти мають стандартну структуру. У структурі папок є файли різних типів:

1. Про Вихідні файли Java та XML. Файли активності та макета, які були створені майстром.




2. Файли Java, які з генеровані Android. Додаткові файли Java, які Android Studio також генерує автоматично. Вносити до них зміни вам не доведеться.

3. Файли ресурсів. До цієї категорії відносяться файли зображень на значках за мовчанням, стилі, які можуть використовуватися програмою, та всі загальні строкові дані, до яких може звертатися програма.

4. Бібліотеки Android. У вікні майстра була вказана мінімальна версія SDK, з якою має бути сумісна програма. Android Studio включає додаток бібліотеки Android, актуальні для цієї версії.

5. Файли конфігурації. Файли конфігурації повідомляють Android, що

містить програму і як її слід виконувати.

Редагування коду в Android Studio. Для перегляду та зміни файлів використовуються різні редактори Android Studio. Зробіть подвійне клацання на файлі, з яким потрібно працювати; його вміст з'являється у середині вікна Android Studio. Більшість файлів відображається у редакторі коду. По суті, це звичайний текстовий редактор, але з підтримкою таких додаткових можливостей, як колірне виділення синтаксису та перевірка коду.

Під час редагування макету з'являється додаткова можливість: замість редагування розмітки XML можна використовувати візуальний редактор. Візуальний редактор дозволяє перетягнути компоненти графічного інтерфейсу на макеті і розташувати їх так, як потрібним. Редактор коду та візуальний редактор забезпечують різні уявлення одного файлу, і можливо перемикатися між ними по на свій розсуд.

Нижче наведено фрагмент коду із файлу макета та активності, згенерованих Android Studio.

**activity_main.xml**

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"        Призначити висоту та ширину макету
android:layout_height="match_parent"       за розмірами екрану пристрою
android:paddingLeft="16dp"
android:paddingRight="16dp"                Додати відступи біля країв екрану
android:paddingTop="16dp"
android:paddingBottom="16dp"
tools:context=".MainActivity">
<TextView                                  Додати графічний компонент TextView для виведення тексту
    android:text="@string/hello_world"     Вивести зн. ресур. рядка hello_world
    android:layout_width="wrap_content"    Включити перенесення тексту
    android:layout_height="wrap_content" /> по горизонталі та вертикалі
</RelativeLayout>
```

**MainActivity.java**

```
package com.hfad.myfirstapp;  Ім'я пакета
import android.os.Bundle;          Класи Android, що використовуються в
import android.app.Activity;       MainActivity.
public class MainActivity extends Activity {          Розширює клас Android
@Override
protected void onCreate(Bundle savedInstanceState) {          Метод реалізації
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);    Пакет, що використовуватися
}
}
```

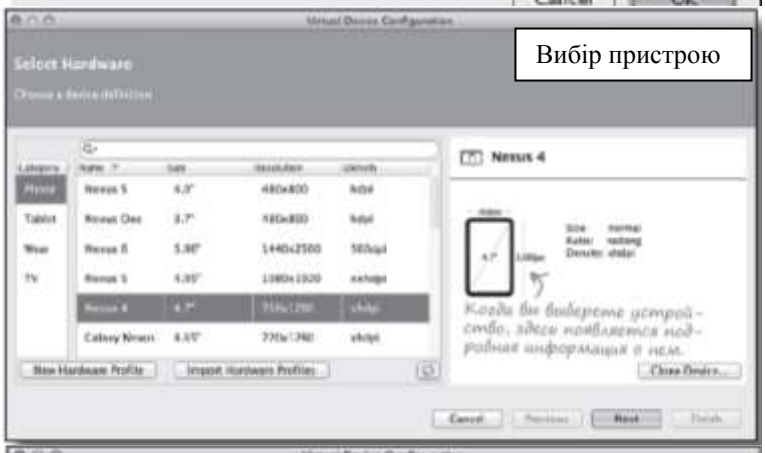
## 3. **Створення віртуального пристрою Android**

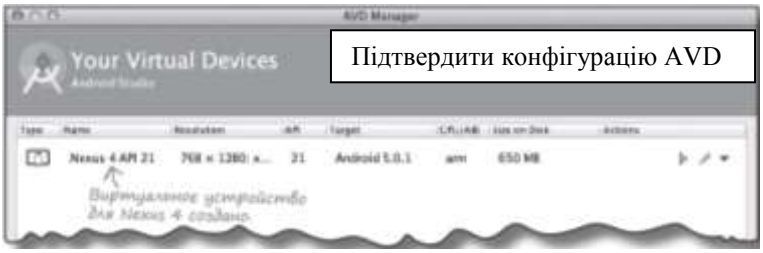
Для перевірки роботи програми скористатися емулятором Android, який вбудований в Android SDK. Емулятор дозволяє створити один або кілька віртуальних пристроїв Android (Android Virtual Device, AVD) і запустити програму в емуляторі так, ніби воно виконується на фізичному пристрої.

Емулятор є додаток, що точно відтворює апаратне оточення пристрою Android: від центрального процесора та пам'яті до звукових мікросхем та екрану. Емулятор побудований на базі існуючого емулятора QEMU, схожий на інші віртуальні машини, такі як VirtualBox або VMWare. Зовнішній вигляд та поведінка AVD залежить від заданих параметрів.

Створення AVD в Android Studio для Nexus 4 з рівнем API 21 складається з кількох кроків. У диспетчері AVD Manager можливо створювати нові AVD, а також переглядати та редагувати вже створені віртуальні пристрої. Щоб запустити його, виберіть у меню Tools пункт Android та виберіть AVD Manager. Натисніть кнопку “Create a virtual device”.

У наступному вікні виберіть у меню Category пункт Phone, потім виберіть у списку Nexus 4. Натисніть кнопку Next.

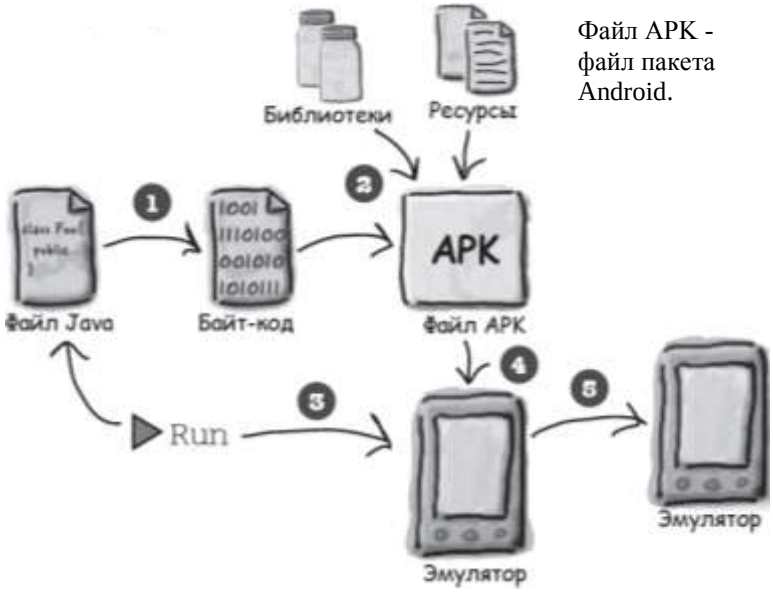



Підтвердити конфігурацію AVD

Далі слід вибрати образ системи, тобто встановлену версію операційної системи Android. Можливо обрати версію Android, яка повинна підтримуватися AVD, та тип процесора (ARM або x86). Необхідно вибрати образ системи для рівня API, сумісного із створюваним додатком. Наприклад, щоб програма працювала на мінімальному рівні API 15, виберіть образ системи з рівнем API щонайменше 15. Вибираємо для нашого випадку строку Lollipop/21/armeabi-v7a з цільовою системою Android 5.0.1. Натисніть кнопку Next.


На наступному екрані буде запропоновано підтвердити конфігурацію AVD. На ньому наведено зведення параметрів щойно вибраних. Підтвердьте значення та клацніть на кнопці Finish. AVD Manager створює AVD і, коли віртуальний пристрій буде створено, відображає його у списку пристроїв. Тепер AVD Manager можна закрити.

### 4. **Запуск програми в емуляторі**



Файл APK - файл пакета Android.

Для запуску програми виберіть команду “Run app” з меню Run. Коли буде запропоновано вибрати пристрій, переконайтеся, що встановлено перемикач “Launch emulator” з щойно створеним віртуальним пристроєм Nexus 4 AVD. Клацніть на кнопки OK.

Команда Run не просто запускає програму. Вона також виконує всі підготовчі операції, необхідні для запуску програми:

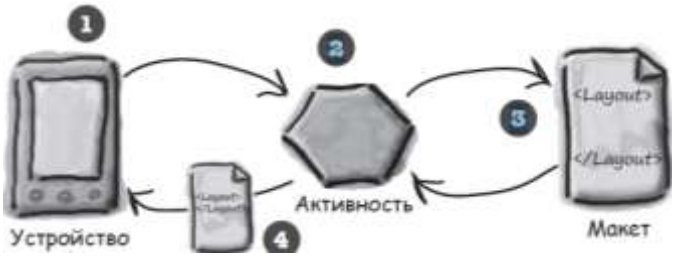
1. Файли з вихідним кодом Java компілюються в байт-код.
2. Створюється пакет Android-програми, або файл APK. Файл APK включає відкомпільовані файли Java, а також всі бібліотеки та ресурси, необхідні для роботи програми.
3. Якщо емулятор не виконується зараз, він запускається з AVD.
4. Коли емулятор буде запущено, а AVD активізується, файл APK передається на AVD та встановлюється.
5. AVD запускає головну активність, пов'язану із додатком. Програма

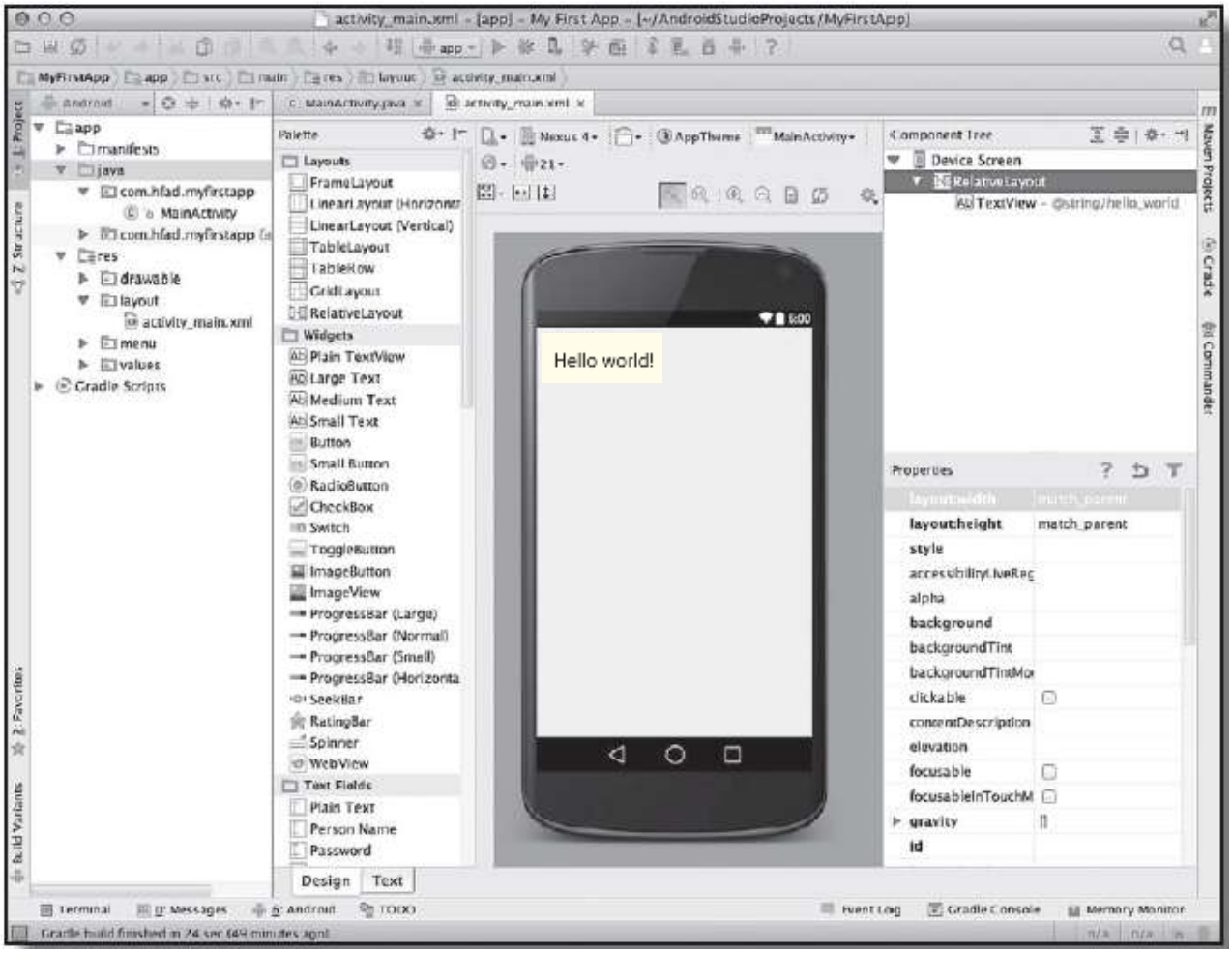
відображається на екрані AVD. Тепер ніщо не заважає тому, щоб перевірити його у роботі.

Запуск емулятора з AVD може займати чимало часу зазвичай кілька хвилин. За ходом операції можна простежити на консолі Android studio. На консолі виводиться докладний звіт про те, що робить система складання gradle, а якщо в процесі запуску виникнуть будь-які помилки - вони будуть виділені в тексті. Консоль розташована в нижній частині екрана Android Studio. Ось що виводиться у вікні консолі під час запуску програми:

```
Waiting for device.
/Applications/adt-bundle-mac/sdk/tools/emulator -avd Nexus_4_API_21 -netspeed full -netdelay none
Device connected: emulator-5554
Device Nexus_4_API_21 [emulator-5554] is online, waiting for processes to start up..
Device is ready: Nexus_4_API_21 [emulator-5554]   <- Виртуальное устройство Android запущено и готово к работе.
Target device: Nexus_4_API_21 [emulator-5554]
Uploading file
        local path: /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/outputs/apk/app-debug.apk
        remote path: /data/local/tmp/com.hfad.myfirstapp
Installing com.hfad.myfirstapp
DEVICE SHELL COMMAND: pm install -r "/data/local/tmp/com.hfad.myfirstapp"
pkg: /data/local/tmp/com.hfad.myfirstapp   <- Загрузка и установка файла APK.
Success
Launching application: com.hfad.myfirstapp/com.hfad.myfirstapp.MainActivity.
DEVICE SHELL COMMAND: am start -n "com.hfad.myfirstapp/com.hfad.myfirstapp.MainActivity" -a
android.intent.action.MAIN -c android.intent.category.LAUNCHER
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
cmp=com.hfad.myfirstapp/.MainActivity }   <- Наконец, наше приложение начинает работу с запуска его главной активности — той самой, которую мастер сгенерировал за вас.
```

Спочатку емулятор запускається в окремому вікні. Емулятор досить довго завантажує AVD, а потім через деякий час у AVD з'являється екран блокування. Зніміть блокування з екрану AVD, махнувши вгору на зображенні замка, з'являється щойно створеній додаток. Ім'я програми відображається у верхній частині екрана, а текст за замовчуванням Hello world! виводиться в основній галузі екрана.






При запуску програми відбувається кілька етапів:

1. Android Studio запускає емулятор, завантажує AVD і встановлює програму.

2. Коли програма запуститься, на базі MainActivity.java створюється об'єкт активності.

3. Активність вказує, що вона використовує макет activity_main.xml.

4. Активність наказує Android вивести макет на екран.

5. **Модифікація програми**

Зараз програма виводить стандартний текст “Hello world!”, включений до нього майстром. Замінимо цей текст, щоб програма вітала користувача якось інакше. Що для цього потрібно зробити? Щоб отримати відповідь на це питання, відступимо на крок назад і подивимося, як зараз будується додаток.

Додаток складається з однієї активності та одного макета. Під час побудови програми повідомили Android Studio, як слід налаштувати його, а майстер зробив усе інше. Майстер згенерував базову активність, а також макет за замовчуванням.

Активність керує тим, що робить додаток. Android Studio створює за нас активність з ім'ям MainActivity.java. Активність визначає, що додаток робить і як воно має реагувати на дії користувача.

Макет керує зовнішнім виглядом програми. Файл MainActivity.java

вказує, що він використовує макет з ім'ям activity_main.xml, який середовище Android Studio згенерувало за нас. Макет визначає, як має виглядати програма. Отже, ми збираємося змінити зовнішній вигляд додатк, змінюючи виведений ним текст. Це означає, що нам доведеться мати справу з компонентом Android, що управляє зовнішнім виглядом програми. Значить, потрібно ближче познайомитись із макетом, тобто почнемо з макета activity_main.xml. Знайдємо файл у папці app/src/main/res/layout та для відкриття його зробімо на ньому подвійне клацання.

Є два способи перегляду і редагування файлів макетів в Android Studio: у візуальному редакторі та у редакторі коду. Якщо вибрати редактор коду, у вікні відображається вміст файлу activity_main.xml. Щоб побачити редактор коду, клацніть на вкладці Text на нижній панелі.

Код складається із двох елементів. Перший елемент - <RelativeLayout> - наказує Android виводити компоненти макета у відносних позиціях. Наприклад, елемент <RelativeLayout> може використовуватися для вирівнювання компонентів по центру макета, вирівнювання по нижньому краю екрана Android-пристрою або позиціонування щодо інших компонентів.

Другий елемент – <TextView> – використовується для виведення тексту. Він вкладений в елемент <RelativeLayout> і в нашому прикладі використовується для виведення згенерованого тексту "Hello world!".

```
<TextView
android:text="@string/hello_world"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

Ключова частина коду в елементі <TextView> знаходиться у першому рядку. Запис android:text означає, що йдеться про властивість text елемента <TextView>, а конструкція визначає, який текст має виводитися у макеті. Але чому використовується синтаксис @string/hello world замість простого Hello world!? І що це взагалі означає?

Почнемо з першої частини @string. Вона просто наказує Android знайти текстове значення у файлі рядкових ресурсів. У прикладі Android Studio створює файл рядкових ресурсів з ім'ям strings.xml, який знаходиться в папці app/src/main/res/values. Друга частина, hello_world, наказує Android отримати значення ресурсу з ім'ям hello_world. Таким чином, @string/hello_world означає: "Знайти рядковий ресурс з ім'ям hello_world і використовувати пов'язане з ним текстове значення". Чому б не включити звичайний текст у activity_main.xml? Хіба це не простіше?

Одна важлива причина: локалізація. Припустимо, ви створили програму, яка мала великий успіх у локальному магазині Google Play Store. Але ви не хочете обмежуватися однією країною або мовою - програма має бути доступною для користувачів з інших країн, які розмовляють іншими мовами.

Виділення текстових значень у strings.xml значно спрощує вирішення подібних завдань. Замість того, щоб змінювати жорстко запрограмовані текстові значення у багатьох різних файлах, достатньо замінити файл strings.xml його локалізованою версією.

Використання файлу strings.xml як центрального сховища текстових значень також спрощує глобальні зміни в тексті в масштабах усієї програми. Якщо директор вимагає змінити текст у додатку через те, що компанія змінила свою назву, достатньо обмежитися файлом strings.xml.

Середовище Android Studio автоматично створило файл рядкових ресурсів з іменем strings.xml; давайте подивимося, містить чи цей файл ресурс hello_world. Знайдіть його в папці app/src/main/res/values на панелі структури проекту та відкрийте подвійним клацанням.

Ось як виглядає код у strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">My First App</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

Як видно з лістингу, у файлі є рядок, який нам і потрібний. Вин описує рядковий ресурс з ім'ям hello_world та значенням “Hello world!”:

```
<string name="hello_world">Hello world!</string>
```

Відредагуйте файл strings.xml, щоб змінити текст, що виводиться. Знайдіть рядок з ім'ям “hello_world” та замініть текстове значення “Hello world!” іншим - наприклад, "Sup doge":

```
<string name="hello_world">~~Hello world!~~ Sup doge</string>
```

Після зміни файлу перейдіть до меню File і виберіть команду Save All, щоб зберегти зміни.

Існують дві ознаки, за якими Android розпізнає strings.xml як файл рядкових ресурсів: 1) файл зберігається у папці app/src/main/res/values; 2) файл містить елемент <resources>, який у свою чергу містить один або кілька елементів <string>.

З формату самого файлу випливає, що він є файлом ресурсів для зберігання рядків. Елемент <resources> повідомляє Android, що файл містить ресурси, а елемент <string> ідентифікує кожен рядковий ресурс. Елемент <resources> ідентифікує вміст файлу як ресурси. Елемент <string> визначає пару ім'я/значення як рядковий ресурс. Це означає, що файл рядкових ресурсів має називатися strings.xml; йому можна надати інше ім'я або розбити рядкові ресурси по кількох файлах. Кожна пара «ім'я/значення» має формат

```
<string name=" ім'я_рядка"> значення_рядка</string>
```

де ім'я_рядка - ідентифікатор рядка, а значення_рядка - власне рядкове значення. Для звернення до значення рядка з макету використовується синтаксис виду "@string/ ім'я_рядка"

Після того, як файл буде відредагований, спробуйте знову запустити програму в емуляторі - виберіть “Run app ” з меню Run. На цей раз додаток повинен вивести повідомлення “Sup doge” замість “Hello world!”.

**Висновки.** Версії Android характеризуються номером версії, рівнем API та кодовим ім'ям. Android Studio – спеціалізована версія середовища IntelliJ IDEA, інтегрована з пакетом Android Software Development Kit (SDK) та

системою складання gradle. Типова Android-додаток складається з активностій, макетів та файлів ресурсів. Макети описують зовнішній вигляд програми. Вони зберігаються у папці app/src/main/res/layout. Активності описують те, що робить додаток, і як воно взаємодіє з користувачем. Створені активності зберігаються в папці app/src/main/java.

Файл strings.xml містить пари «ім'я/значення» для рядків. Вони використовуються для винесення конкретних текстових значень з макетів та активностій, а також для підтримки локалізації. Файл AndroidManifest.xml містить інформацію про програму. Цей файл знаходиться у папці app/src/main.

AVD - віртуальний пристрій Android (Android Virtual Device). AVD виконується в емуляторі Android та моделює фізичний пристрій Android. APK - пакет програми Android, аналог JAR-файлу для програм Android. Файл містить байт-код програми, бібліотеки та ресурси. Встановлення програми на пристрої здійснюється установкою його пакету APK.

Програми Android виконуються в окремих процесах з використанням виконавчого середовища Android (ART). Елемент RelativeLayout використовується для розміщення компонентів графічного інтерфейсу у відносних позиціях у макеті. Елемент TextView використовується для виведення тексту.

**Практичне заняття № 2. Побудова інтерактивних програм**

**Кількість годин:** 2 год.

**Навчальна мета заняття**:

1. Придбання теоретичних знань з теми «Середовище розробки платформи Android», розвиток здібностей до творчого мислення, формування навичок самостійної роботи з аналізу і узагальнення інформації, вміння проектувати компонентну архітектуру мобільного додатку.

**Рекомендована література:**


1. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.
2. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
3. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
4. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
5. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
6. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.

7. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
8. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.

**Матеріально-технічне забезпечення:** комп’ютерний клас.

**Навчальні питання:**

1. Основні крокі в створенні інтерактивних програм
2. Створення інтерактивного проєкту
3. Оновлення макету
4. Зв'язування макету з активністю
5. Оновлення активності

## 1 Основні крокі в створенні інтерактивних програм

Зазвичай додаток повинен реагувати на дії користувача. Побудуємо додаток Beer Adviser для користувача виберу пива, який він віддає перевагу, клацає на кнопці і отримує список рекомендованих сортів. Макет додатку має таку структуру з трьох компонентів графічного інтерфейсу: списку значень, що розкривається, в якому користувач вибирає потрібний вид пива; кнопки, яка при натисканні повертає добірку сортів пива; написи для виведення сортів пива.

Файл strings.xml включає всі строкові ресурси, необхідні макету, - наприклад, текст надписи на кнопці, що знаходиться в макеті. Активність визначає, як додаток має взаємодіяти з користувачем. Вона отримує вид пива, вибраний користувачем, та використовує його для виведення списку сортів, які можуть представляти інтерес для користувача. Для вирішення цього завдання використовується допоміжний клас Java. Клас Java містить логіку програми. Вин включає метод, який набуває вигляду пива у параметрі та повертає список сортів пива вказаного типу. Активність викликає метод, передає йому вид пива та використовує отриманою відповідь.

Отже, приступимо до побудови програми Beer Adviser. Робота складається з кількох кроків:

1. Створення проекту, до цього достатньо включити базовий макет та активність.

2. Оновлення макету. Слідує редагувати макет і включити до нього всі компоненти графічного інтерфейсу, необхідних для роботи програми.

3. Зв'язування макету з активністю. Макет створює лише візуальне оформлення. Щоб додаток міг виконувати розумні дії, необхідно зв'язати макет із кодом Java в активності.

4. Оновлення активності, яка буде повертати користувачеві правильні сорти пива залежно від вибору.

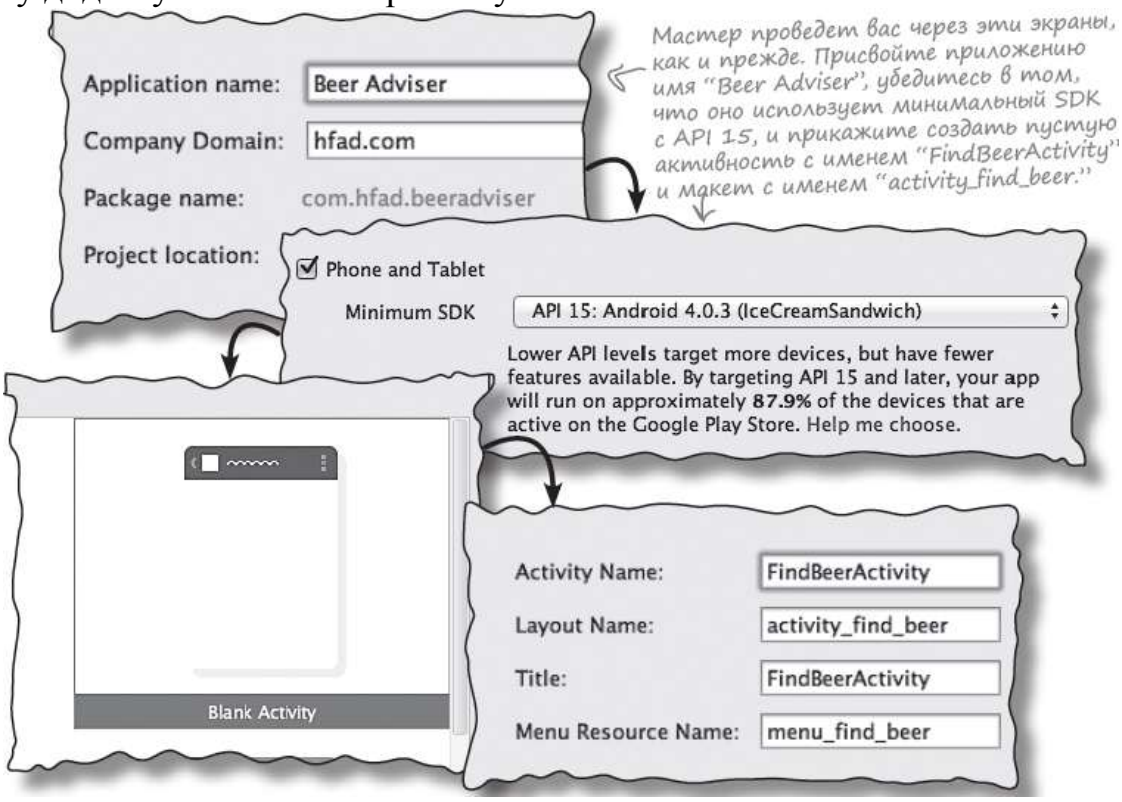
## 2 Створення інтерактивного проєкту

Робота починається зі створення нової програми (це робиться майже так само, як у попередньому додатку):

1. Відкриття Android Studio та вибір на заставці рядок "Start a new Android Studio project" для запуску майстру завдання параметрів проекту: введення імені програми Beer Adviser з подальшою генерацією імені пакета com.hfad.beeradviser.

2. Вибір мінімальної версії SDK з API 15, щоб програма працювала на більшості телефонів та планшетів. Встановлення прапорця “Phone and Tablet”.

3. Вибір простої активності за мовчанням з ім’ям "FindBeerActivity", а макету - ім'я "activity_find_beer". Підтвердження значення за мовчанням для тексту заголовка (Title) та імені ресурсу меню (Menu Resource Name), оскільки у цьому додатку вони не використовуються.




Таким чином, створено новий проект в середовище Android Studio, що містить активність FindBeerActivity.java та макет activity_find_beer.xml. Для редагування файлу макету перейдемо до папки app/src/main/res/layout та відкриймо файл activity_find_beer.xml.

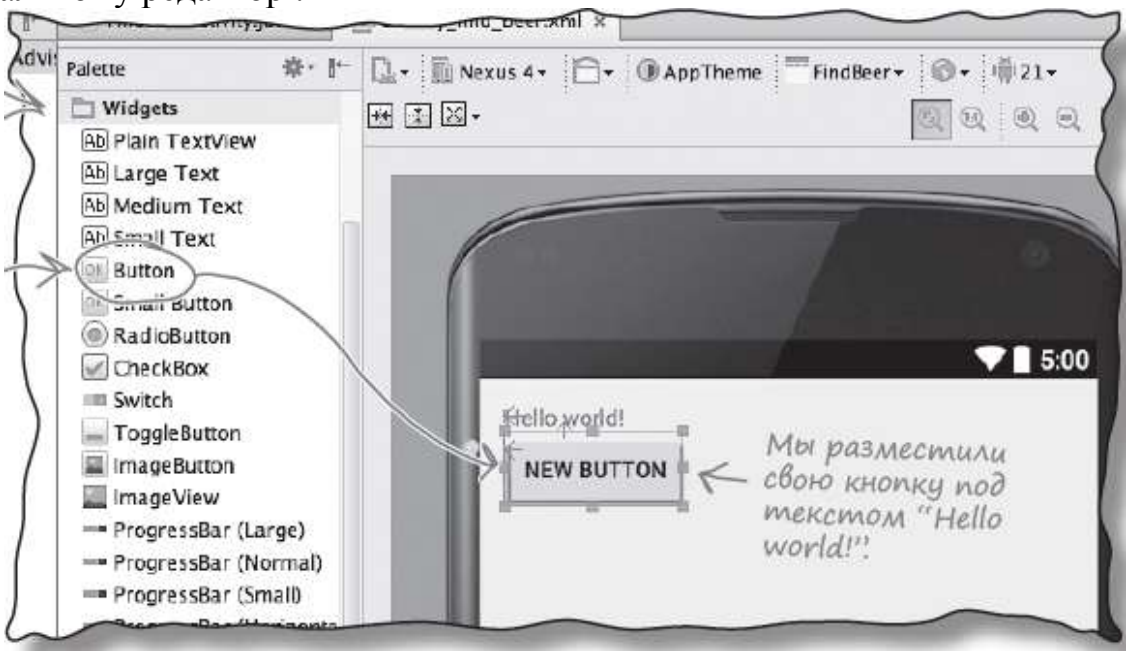
Як і в попередньому додатку, майстер створив стандартний макет з елементом <TextView>, що містить текст “Hello world!”:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".FindBeerActivity">
```

*Эти элементы относятся к макету в целом. Они определяют его ширину и высоту, а также величину отступов от краев макета.*

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Додавання компонентів у візуальному редакторі до макету можна двома способами: у розмітці XML або у візуальному редакторі. Почнемо з додавання кнопки у візуальному редакторі. Ліворуч від візуального редактора розташовується палітра з компонентами графічного інтерфейсу, які можна перетягувати мишею на макет. Перегляньте розділ Widgets (Віджети) і знайдіть у ньому компонент кнопки (Button). Клацніть на ньому та перетягніть на макет у візуальному редакторі.




Таке перезавантаження компонентів графічного інтерфейсу є зручним способом оновлення макету. Переключившись на редактор коду, бачимо, що в результаті додавання кнопки у візуальному редакторі у файлі з'явилося кілька рядків коду. У activity_find_beer.xml з'явилася нова кнопка, тобто редактор додав новий елемент <Button> у файл activity_find_beer.xml:

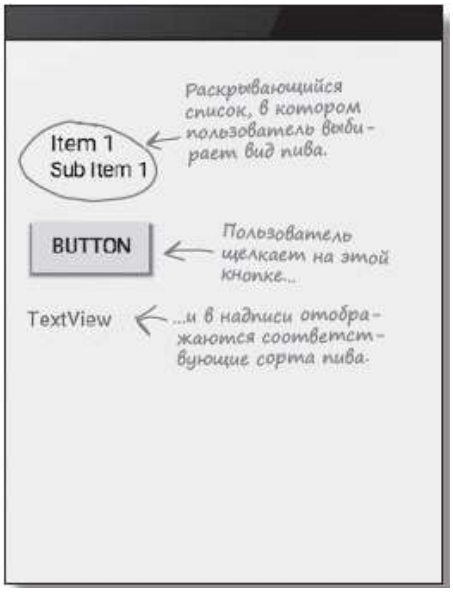

У світі Android кнопка натискається користувачем, щоб ініціювати яку-небудь дію. Кнопка володіє властивостями, керуючими її позицією, розміром, зовнішнім виглядом і методами активності, які вона повинна викликати. Ці властивості існують не тільки в кнопках - вони є і в інших компонентах графічного інтерфейсу, включаючи надписи. Фактично, що кнопки та надписи мають так багато загальних властивостей, цілком логічно - обидва компоненти успадковують від одного класу Android View.

Ім'я, за яким ідентифікується компонент використовується для керування робочим компонентом із коду активності, а також для керування розміщенням компонентів у макеті: android:id="@+id/button". Властивість android:text="New Button" повідомляє Android, який текст має виводитись у компоненті. Властивості android:layout_width, android:layout_height задають базову ширину і висоту компонента. Значення "wrap_content" означає, що розміри компонента повинні підбиратися за розмірами вмісту:

android:layout_width="wrap_content"

android:layout_height="wrap_content"

Як вже було відомо, зміни, що вносяться у візуальному редакторі відображаються в розмітці XML макету. Також справедливе і протилежне: всі зміни, що вносяться у розмітці XML макета, відображаються у візуальному редакторі. Зазначимо, що пряме редагування XML дозволяє точніше керувати макетом, а також скорочує залежність розробника від середовища розробки.




Замінимо код із файлу activity_find_beer.xml з додаванням фрагменту Spinner, якій приведе до змін у візуальному редакторі. Замість макета з написом і розташованим під її кнопкою буде відображатися макет, в якому напис відображається під кнопкою.


Після внесення змін до XML макету перейдемо до візуального редактора. Замість макета з написом і розташованим під її кнопкою повинен відображатися макет, в якому напис відображається під кнопкою.

Над кнопкою розташовується список (spinner). Якщо торкнутися його, на екрані з'являється список, з якого користувач вибирає одне значення. Компоненти графічного інтерфейсу – кнопки, списки, що розкриваються, написи – мають схожі атрибути, оскільки всі вони є спеціалізаціями View. Класи всіх цих компонентів успадковують від одного класу Android View.

Перш ніж запускати програму, необхідно внести ще одну зміна. На даний момент тексти кнопки та написи задаються жорстко запрограмованими рядковими значеннями. Як згадувалося у попередньому додатку, бажано замінити їх посиланнями на файл рядкових ресурсів strings.xml. Використання файлу рядкових ресурсів для статичного тексту спрощує створення версій програми іншими мовами, а якщо раптом потрібно змінити формулювання у всьому додатку, достатньо буде внести зміни в одному централізованому місці.

Відкриваємо файл app/src/main/res/values/strings.xml, він повинен виглядати приблизно так:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Beer Adviser</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

Насамперед видаліть ресурс “hello_world”, оскільки він нам більше не потрібно. Додати новий ресурс з ім'ям "find_beer" та значенням "Find Beer!". Коли це буде зроблено, додайте новий ресурс під назвою “brands”, але доки не вводите значення. Новий код має виглядати приблизно так:

```
<string name="app_name">Beer Adviser</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="find_beer">Find Beer!</string>
<string name="brands"></string>
```

Необходимо удалить строковый ресурс hello_world и добавить два новых ресурса с именами find_beer и brands.

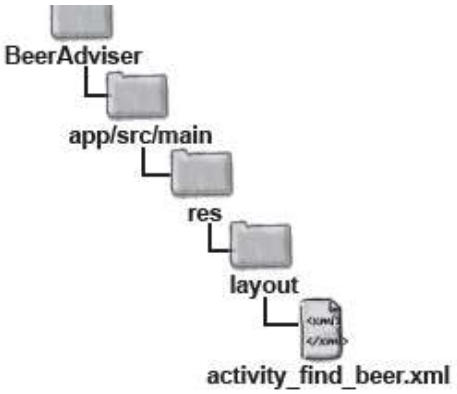
**3 Оновлення макету**

Для внесення змін в макет для використання рядкових ресурсів змінимо елементи кнопки та написи в розмітці XML макета, щоб вони використовували два щойно додані рядкові ресурси. Відкроїмо файл activity_find_beer.xml та внесіть такі зміни:

Замініть рядок android:text="Button" рядком android:text="@string/find_beer".

Замініть рядок android:text="TextView" рядком android:text="@string/brands".

```
<Spinner
    android:id="@+id/color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp" />

<Button
    android:id="@+id/find_beer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/color"
    android:layout_below="@+id/color"
    android:text="@string/find_beer" />

<TextView
    android:id="@+id/brands"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/find_beer"
    android:layout_below="@+id/find_beer"
    android:layout_marginTop="18dp"
    android:text="@string/brands" />
```




На кнопке выводится значение строкового ресурса find_beer.

Значение строкового ресурса brands выводится в надписи. В настоящее время надпись пуста, но все будущие изменения строкового значения будут отражаться автоматически.

Результат виглядає на даний момент наступним чином Збережіть внесені зміни та виберіть команду “Run app” з меню Run. Коли буде запропоновано вибрати спосіб запуску, виберіть запуск в емуляторі.


Додавання значень до списку. На даний момент макет включає список, що розкривається, але в цьому списку немає жодних даних. Щоб список приносив користь, у ньому має відображатись список значень. Список значень для списку, що розкривається, можна визначити практично так само, як було визначимо текст на кнопці та написи: потрібно створити для нього ресурс. Досі у файлі strings.xml визначалися одиночні строкові значення. Усе, що нам потрібно - це визначити масив рядкових значень і передати посилання на нього списку, що розкривається.

Додавання ресурсу масиву дуже схоже на додавання ресурсу рядка, для цього у файлу strings.xml необхідно вказати:

```
<string-array name="имя_массива_строк">
    <item>значение1</item>
    <item>значение2</item>
    <item>значение3</item>
    ...
</string-array>
```

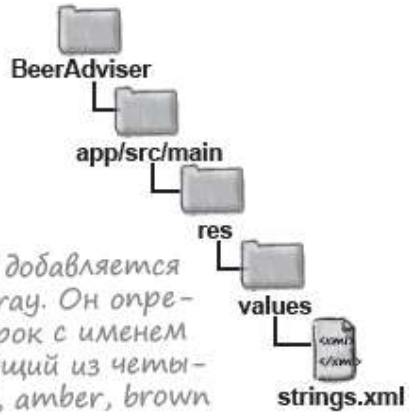
Имя массива.

Значения в массиве. Добавьте столько, сколько потребуется.

де ім'я_масиву - ім'я масиву, а значення1, значення2, значення3 - окремі строкові значення, що входять до масиву. Відкрийте файл strings.xml і додайте до нього наступний фрагмент:

```
...
    <string name="brands"></string>
    <string-array name="beer_colors">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
    </string-array>
</resources>
```

В файл strings.xml добавляется элемент string-array. Он определяет массив строк с именем beer_colors, состоящий из четырех значений: light, amber, brown и dark.

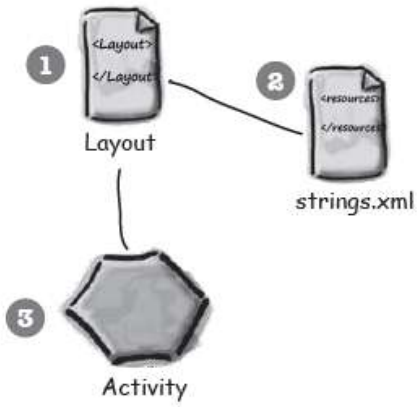



Відкрийте файл макета activity_find_beer.xml і додайте елемент <Spinner> атрибут entries:

```
<Spinner
    ...
    android:layout_marginTop="37dp"
    android:entries="@array/beer_colors" />
```

Это означает "данные раскрывающегося списка берутся из массива beer_colors".

Збережіть зміни та запустіть програму. Результат повинен виглядати приблизно так:




Нижче коротко перераховані основні дії, які були виконані на даний момент:

1. Створено макет, який визначає зовнішній вигляд програми.

Макет включає в себе розкривається список, кнопку та напис.

2. Файл strings.xml містить необхідні рядкові ресурси. Додано текст для кнопки та порожній рядок для сортів пива.

3. Активність визначає, як програма має взаємодіяти з користувачем. Середовище Android Studio згенерувало базову активність, але воно поки так і залишилося у початковому вигляді.

**4 Зв'язування макету з активністю**

Тепер потрібно домогтися того, щоб програма реагувала на значення, вибране в списку, що розкривається. Воно має працювати приблизно так:

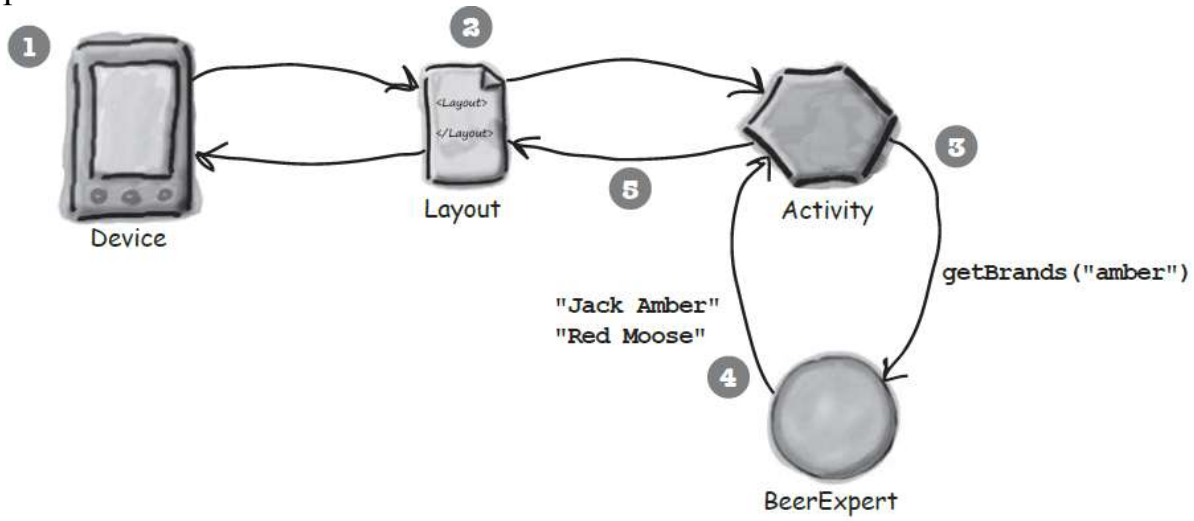
1. Користувач вибирає вид пива в списку, що розкривається. Користувач клацає на кнопці, щоб знайти сорти пива цього виду.

2. Макет вказує, який метод активності повинен викликатися при натисканні на кнопці.

3. Метод активності отримує від списку, що розкривається, обране значення (вид пива) і передає його методу getBrands() класу Java з ім'ям BeerExpert.

4. Метод getBrands() класу BeerExpert знаходить сорти пива, що відповідають заданому виду, і повертає їх активності у вигляді об'єкта ArrayList з рядковими даними.

5. Активність отримує посилання на напис з макета і надає його властивості text список відповідних сортів. Інформація відображається на пристрої.




Щоб змусити кнопку викликати метод активності, необхідно внести зміни у двох файлах:

Зміни у файлі макету activity_find_beer.xml. Необхідно вказати, який

метод активності має викликатись при натисканні на кнопці.

Зміни у файлі активності FindBeerActivity.java. Необхідно написати метод, який буде викликатись при клацанні.

Щоб повідомити Android, який метод повинен викликатись при клацанні на кнопці, достатньо всього одного рядка розмітки XML з атрибутом android:onClick в елемент <button> і вказати ім'я методу, що викликається: android:onClick="имя_метода".

Відкрийте файл макета activity_find_beer.xml і додайте в елемент <button> новий рядок XML, який повідомляє, що при натисканні на кнопці повинен викликатися метод onClickFindBeer():

```
<Button
android:id="@+id/find_beer"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/color"
android:layout_below="@+id/color"
android:text="@string/find_beer"
android:onClick="onClickFindBeer" />
```

Збережіть файл після внесення змін. Тепер макет знає, який метод активності слідує викликати; але потрібно написати сам метод. У процесі створення проекту для програми наказано майстру згенерувати найпростішу активність з ім'ям FindBeerActivity. Код цієї активності зберігається у файлі FindBeerActivity.java. Відкрийте цей файл – перейдіть до папки app/src/main/java та зробіть на ньому подвійне клацання.

Середовище Android Studio згенерувало досить великий обсяг коду Java, якій не є обов'язковим. Запропонуємо замінити його кодом, наведеним нижче.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;

public class FindBeerActivity extends Activity {

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

Класс расширяет класс Android Activity.

Метод onCreate() вызывается при исходном создании активности.

Метод setContentView сообщает Android, какой макет использует активность. В данном случае это макет activity_find_beer.

Цей код - все, що потрібно для створення найпростішої активності.У ньому створюється клас, який розширює клас android.app.Activity і реалізує метод onCreate().

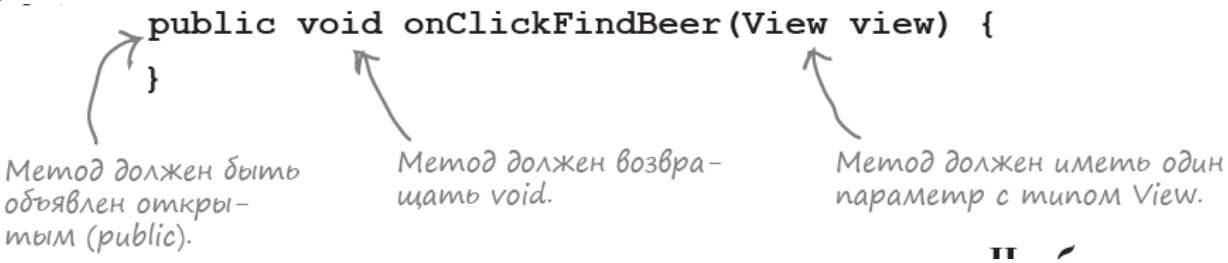
Усі активності мають розширювати клас Activity. Клас Activity містить набір методів, які перетворюють звичайний клас Java у повноцінну активність Android.

Усі активності також повинні реалізувати метод onCreate(). Метод

onCreate() викликається при створенні об'єкта активності та використовується для налаштування основних параметрів - наприклад, вибору макета, з яким пов'язується активність. Це робиться за допомогою методу setContentView(). У наведеному прикладі виклик setContentView(R.layout.activity_find_beer) повідомляє Android, що ця активність використовує макет activity_find_beer.

На попередній сторінці надано атрибут onClick до кнопки в макеті і надали йому значення onClickFindBeer. Тепер потрібно додати цей метод активність, щоб він викликався при натисканні кнопки. Таким чином, активність буде реагувати на натискання користувачем кнопки в інтерфейсі.

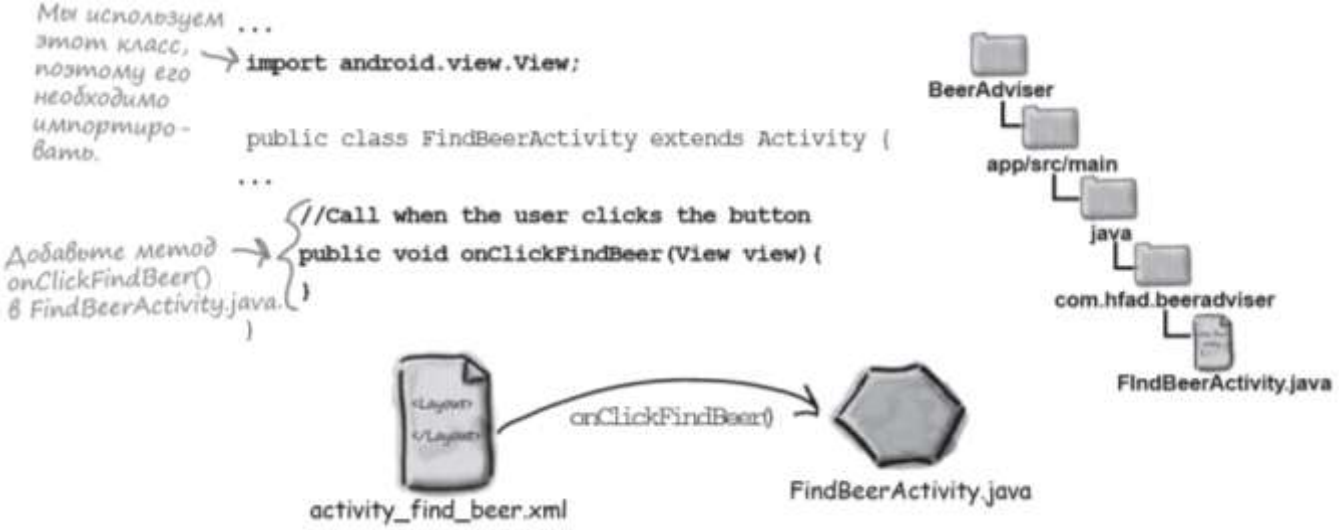
Додавання до активності методу onClickFindBeer(). Метод onClickFindBeer() повинен мати чітко визначену сигнатуру; в іншому випадку він не буде викликатися при клацання на кнопці, вказаній у макеті. Він має таку форму:




Якщо метод має іншу сигнатуру, він не реагуватиме на дотик користувача до кнопки. Справа в тому що Android непомітно для користувача шукає відкритий метод, повертаючий void, ім'я якого збігається з ім'ям методу, вказаного в розмітці XML макету.

Параметр View на перший погляд здається дещо дивним, але для його присутності є вагома причина. Він визначає компонент графічного інтерфейсу, що ініціював виклик методу (у разі це кнопка). Як згадувалося раніше, компоненти графічного інтерфейсу – такі, як кнопки та написи, - всі є спеціалізаціями View.

Отже, відновимо наш код активності. Додати до коду активності метод onClickFindBeer(), наведений нижче:




Отже, створено в активності метод onClickFindBeer(). Далі потрібно подбати про те, щоб під час виконання цього методу щось відбувалося. Додаток

повинен виводити добірку сортів пива, що відповідають виду, вибраному користувачем.

**5 Оновлення активності**

Для цього необхідно спочатку отримати посилання на обидві компоненти графічного інтерфейсу в макеті - список і напис, що розкривається. За допомогою цих посилань можливо отримати значення, вибране у списку (вигляд пива), та вивести текст у написі.

Для отримання посилання на дві компоненти графічного інтерфейсу можна скористатися методом findViewById(). Метод findViewById() отримує ідентифікатор компонента як параметр і повертає об'єкт View. Далі залишається привести значення, що повертається до правильного типу компонента (наприклад, TextView або Button).

Подивимося, як метод findViewById() використовується для отримання посилання на напис з ідентифікатором brands:

TextView brands = (TextView) findViewById(R.id.brands);

brands має тип TextView, тому посилання наводиться до цього типу.

R – спеціальний клас Java, який дозволяє отримувати посилання ресурси у додатку.

Щоб передавати ім'я компоненти, задається ідентифікатор написи R.id.brands. R.java - спеціальний файл Java, який генерується інструментарієм Android при створенні або побудові програми. Він знаходиться в папці app/build/generated/source/r/debug проекту – всередині папки, ім'я якої збігається з ім'ям пакета програми. Android використовує R для відстеження ресурсів, що використовуються у додатку; серед іншого, цей клас дозволяє отримувати посилання компоненти графічного інтерфейсу з коду активності.

Файл R.java містить серію внутрішніх класів по одному для кожного типу ресурсів. Звернення до кожного ресурсу цього здійснюється через внутрішній клас. R включає внутрішній клас з ім'ям id, а цей внутрішній клас входить значення static final brands. Рядок коду (TextView) findViewById(R.id.brands); використовує це значення для отримання посилання на напис brands.

Отримавши посилання на об'єкт View, можливо викликати його методи. Метод findViewById() надає Java-версію компонента графічного інтерфейсу. Це означає, що можливо читати та задавати властивості компоненти за допомогою методів, що надаються класом Java. Для отримання посилання на компонент напису Java використовується синтаксис

TextView brands = (TextView) findViewById(R.id.brands);

Під час виконання цього рядка коду створюється об'єкт класу TextView з іменем brands. Після цього можна викликати методи цього об'єкта TextView. Клас TextView містить метод setText(), який використовується для завдання властивості text, наприклад, щоб у написі brands відображався текст "Gottle of geer":

brands.setText("Gottle of geer");

Для отримання посилання на список, що розкривається, робиться практично так само, як для напису. Знову використовується метод

findViewById(), але цього разу результат наводиться до типу Spinner:

Spinner color = (Spinner) findViewById(R.id.color);

Тобто отримується об'єкт Spinner та можливо викликати його методи. Наприклад, ось як відбувається отримання поточного вибраного варіанту списку і перетворення його до типу String:

String.valueOf(color.getSelectedItem())

Отримує вибраний варіант у списку і перетворює його на String.

Конструкція color.getSelectedItem() повертає узагальнений об'єкт Java. Справа в тому, що значення списку, що розкривається, не повинні бути об'єктами String - це можуть бути, наприклад, зображення. У нашому випадку відомо, що значення є об'єктами String, тому використовуємо метод String.valueOf() для перетворення вибраного варіанту з Object в String.

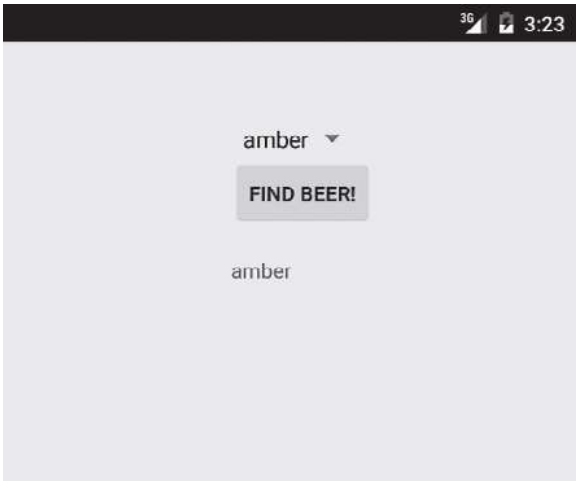
Для оновлення коду методу onClickFindBeer() активності необхідно отримання варіанту, обраного в списку, і відображення його у написі. Метод повинен отримувати вид пива, вибраний у списку, що розкривається, і виводити його в написі.

Нижче наведено код активності разом із методом, який зібраний воєдино на попередній сторінці. Внесіть ці зміни у FindBeerActivity.java і збережіть файл:

```
package com.hfad.beeradviser;
import android.os.Bundle;
import android.app.Activity;
// Використовуємо додаткові класи
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;

public class FindBeerActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_find_beer);
}
// Викликається при натисканні на кнопці
public void onClickFindBeer(View view) {
//Отримати посилання на TextView
TextView brands = (TextView) findViewById(R.id.brands);
//Отримати посилання на Spinner
Spinner color = (Spinner) findViewById(R.id.color);
//Отримати варіант, вибраний у Spinner
String beerType = String.valueOf(color.getSelectedItem());
// Вивести обраний варіант
brands.setText(beerType);
```

Цей код робить наступне:

1. Користувач вибирає вид пива в списку, що розкривається, і клацає на кнопці Find Beer. Це призводить до виклику методу public void onClickFindBeer(View) активності. Макет повідомляє, який метод активності повинен викликатися при натисканні на кнопці, за допомогою якості android:onClick кнопки.




2. Щоб отримати посилання компоненти графічного інтерфейсу TextView і Spinner, активність викликає метод findViewById().

3. Активність отримує поточне обране значення в списку, що розкривається і перетворює його на String.

4. Потім активність задає властивість text компонента TextView, щоб поточний варіант зі списку відображався у написі.

Внесіть зміни до файлу активності, збережіть його і запустіть програму. Тепер при натисканні на кнопці Find Beer значення варіанта, вибраного у списку, виводиться у написи. Вибраний вид пива відображається у написі.

Допоміжний клас Java повинен задовольняти такі вимоги:

- Клас повинен належати пакету com.hfad.beeradviser.
- Класу має бути присвоєно ім'я BeerExpert.
- Клас повинен надавати один метод getBrands(), який отримує бажаний вид пива (у вигляді String) і повертає контейнер List<String> з рекомендованими сортами.

Класи Java бувають надзвичайно складними, вони можуть бути задіяні виклики нетривіальної логіки програми. Або побудуйте власну версію класу або скористайтеся нашою готовою версією, наведеною нижче:

```
package com.hfad.beeradviser;
import java.util.ArrayList;
import java.util.List;
public class BeerExpert {
    List<String> getBrands(String color) {
        List<String> brands = new ArrayList<String>();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
    return brands;
    }
}
```

У другій версії активності доопрацюємо метод onClickFindBeer(), щоб він викликав метод класу BeerExpert для отримання рекомендацій. Усі необхідні зміни містять лише традиційний код Java. Можемо спробувати написати код і запустити програму.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
...
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Получить рекомендации от класса BeerExpert
        List<String> brandsList = expert.getBrands(beerType);
        StringBuilder brandsFormatted = new StringBuilder();
        for (String brand : brandsList) {
            brandsFormatted.append(brand).append('\n');
        }
        //Display the beers
        brands.setText(brandsFormatted);
    }
}
```

Получить контейнер List с сортами пива.

Построить String по данным из List.

Каждый сорт выводится с новой строки.

Вывести результаты в надписи.

Реализация BeerExpert содержит только традиционный код Java, поэтому не беспокойтесь, если ваш код немного отличается от нашего.

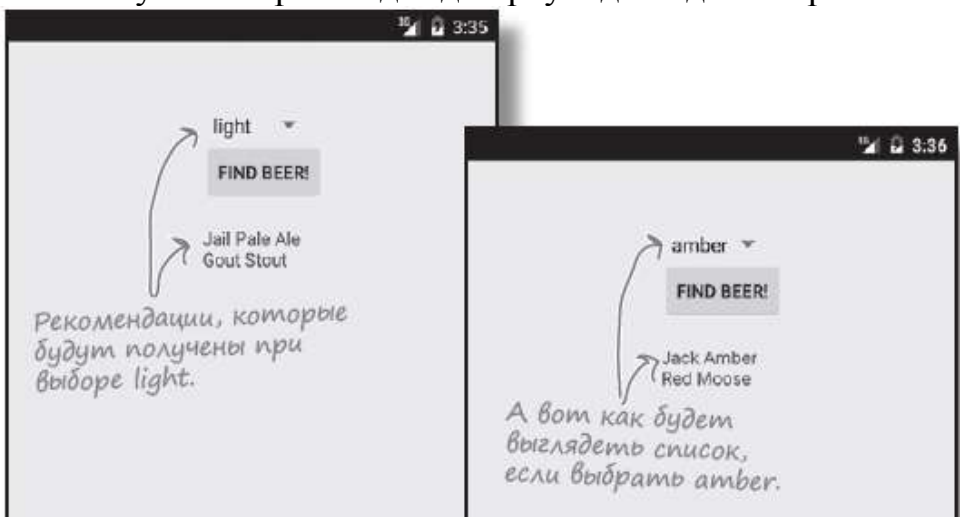
При виконанні коду відбувається наступне

1. Коли користувач клацає на кнопці Find Beer, викликається метод onClickFindBeer() з класу активності. Метод створює посилання на список, що розкривається, і напис і отримує поточне значення, вибране у списку

2. Метод onClickFindBeer() викликає метод getBrands() з класу BeerExpert, передаючи йому вид пива, вибраний у списку, що розкривається. Метод getBrands() повертає список сортів пива.

3. Метод onClick Find Beer() форматує список сортів і використовує для завдання властивості text написи.

Після того як програма буде змінена, запустіть його. Проекспериментуйте, вибираючи різні види пива та клацаючи на кнопці Find Beer. Коли вибираємо види пива і клацаєте на кнопці Find Beer, додаток при допомоги класу BeerExpert видає добірку відповідних сортів.




**Висновки.** Елемент Button використовується для додавання кнопки. Елемент Spinner використовується для додавання розкривного списку. Всі компоненти графічного інтерфейсу успадковують від класу Android View.

Масив рядкових значень створюється конструкцією наступного виду:

<string-array name="array">
<item>string1</item>
...
</string-array>

Для звернення до string-array у макеті використовується синтаксис:

"@array/array_name"

Щоб при натисканні на кнопці викликався метод, увімкніть у макет наступний атрибут:

android:onClick="clickMethod"

При цьому в активності має існувати відповідний метод:

public void clickMethod(View view){
}

Клас R.java генерується середовищем. Він дозволяє отримувати посилання на макети, компоненти графічного інтерфейсу, рядки та інші ресурси у коді Java. Метод findViewById() повертає посилання на компонент. Метод setText() задає текст компонента. Метод getSelectedItem() повертає варіант, вибраний у списку, що розкривається. Щоб додати допоміжний клас до проекту Android, виконайте команду File→New...→Java Class.

**Практичне заняття № 3. Множинні активності та інтенти**

**Кількість годин:** 2 год.

**Навчальна мета заняття:**

1. Придбання теоретичних знань з теми «Структура та компоненти мобільного додатку», розвиток здібностей до творчого мислення, формування навичок самостійної роботи з аналізу і узагальнення інформації, вміння проектувати компонентну архітектуру мобільного додатку.

**Рекомендована література:**


1. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.
2. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
3. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
4. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
5. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
6. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.
7. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
8. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.


**Матеріально-технічне забезпечення:** комп'ютерний клас.

**Навчальні питання:**

1. Послідовність дій створення додатків із множинною активністю
2. Створення базової програми з однією активністю та макетом
3. Додавання другої активності та макета
4. Організація виклику другої активності з першої
5. Організація передачі з першої активності в другу

1. ПОРЯДОК ПРОВЕДЕННЯ ЗАНЯТТЯ:

1.1. Проведення експрес-контролю готовності до заняття.

1.2. Ввести текст підготовленої програми і виконати її відлагодження.

1.3. Підібрати тести і виконати відпрацювання розробленого алгоритму на цих тестах.

1.4. Скласти звіт про виконану роботу і здати роботу викладачу.

**1 Послідовність дій створення додатків із множинною активністю**

Більшості додатків однієї активності недостатньо. Одна активність для простих програм це нормально. Однак у складнішій ситуації одна активність просто не справляється з усіма справами. Як будувати програми з кількома активностями та як організувати взаємодія між активностями з використанням

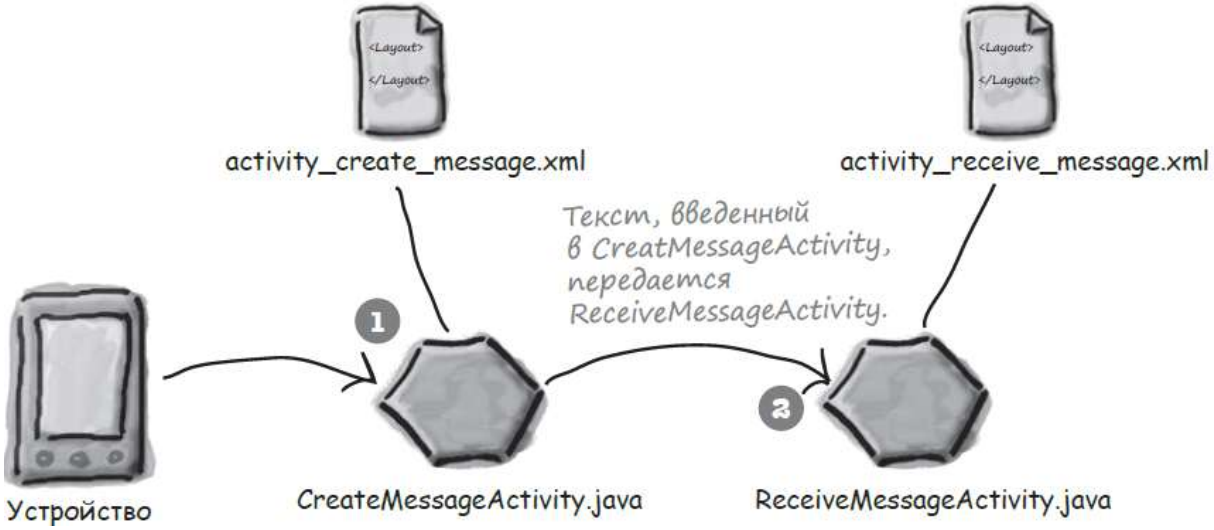
інтентів є метою цій теми. Інтенція - це суб'єктивна спрямованість на певний предмет, активність свідомості суб'єкта. Активність – одна цілеспрямована операція, яка може виконуватись користувачем. Якщо об'єднати кілька активностей для виконання чогось складнішого, вийде завдання. Побудуємо додаток із двома активностями. Перша активність дає змогу ввести текст повідомлення. Клацніть на кнопці в першій активності запускає другу активність, якою передається повідомлення. Далі друга активність виводить отримане повідомлення. Послідовність дій:

1. Створення базової програми з однією активністю та макетом.
2. Додавання другої активності та макета.
3. Організація виклику другої активності з першої.
4. Організація передачі з першої активності в другу.

Структура додатку складається з двох активностей та двох макетів:

1. На початку роботи програми запускається активність CreateMessageActivity. Ця активність використовує макет activity_create_message.xml.

2. Користувач клацає на кнопці в CreateMessageActivity. Кнопка запускає активність ReceiveMessageActivity, яка використовує макет activity_receive_message.xml.




**2 Створення базової програми з однією активністю та макетом.**

Проект програми створюється так само, як і в попередніх розділах. Створіть в Android Studio новий проект для програми з ім'ям Messenger і ім'ям пакета com.hfad.messenger. Виберіть мінімальний рівень API 15, щоб програма працювала на більшості пристроїв. Щоб код не відрізнявся від нашого, створіть порожню активність з ім'ям “CreateMessageActivity” та макет з ім'ям “activity_create_message”.

Оновлений макету розмітка XML із файлу activity_create_message.xml. має вигляд:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".CreateMessageActivity" >
    <Button
        android:id="@+id/send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="36dp"
        android:layout_marginTop="21dp"
        android:onClick="onSendMessage"
        android:text="@string/send" />
    <EditText
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/send"
        android:layout_below="@+id/send"
        android:layout_marginTop="18dp"
        android:ems="10" />
    </RelativeLayout>
```

Клацніт
метод от

Рядкови

Замініть
згенеров
елемент


Описує ширину текстового поля <EditText>. Ширина поля має бути достатньою для розміщення 10 літер "M".

Оновлення strings.xml означає, що потрібно додати рядок з іменем “send” у strings.xml і привласнити йому значення Send Message - текст, який повинен відображатися на кнопці:

<string name="send">Send Message</string>

Додавання методу до активності в наступному рядку елемента <Button>

android:onClick="onSendMessage

означає, що метод onSendMessage() активності буде спрацьовувати при натисканні на кнопці. Для цього відкрийте файл CreateMessageActivity.java та замініть код, згенерований Android Studio, наступним:

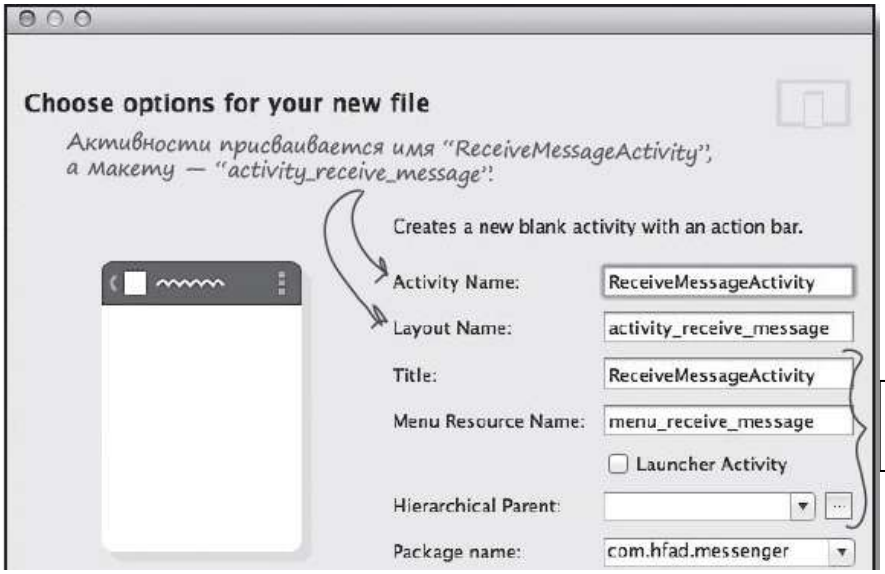
```
package com.hfad.messenger;
// Замінюємо код, який було згенеровано Android Studio, оскільки більшість цього коду не обов'язкова
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
public class CreateMessageActivity extends Activity {
    @Override
```

```
    // Метод onCreate() викликається під час створення активності
      protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_create_message);
      }
      // Викликати onSendMessage() при натисканні на кнопці
      public void onSendMessage(View view) {
      }
}
```

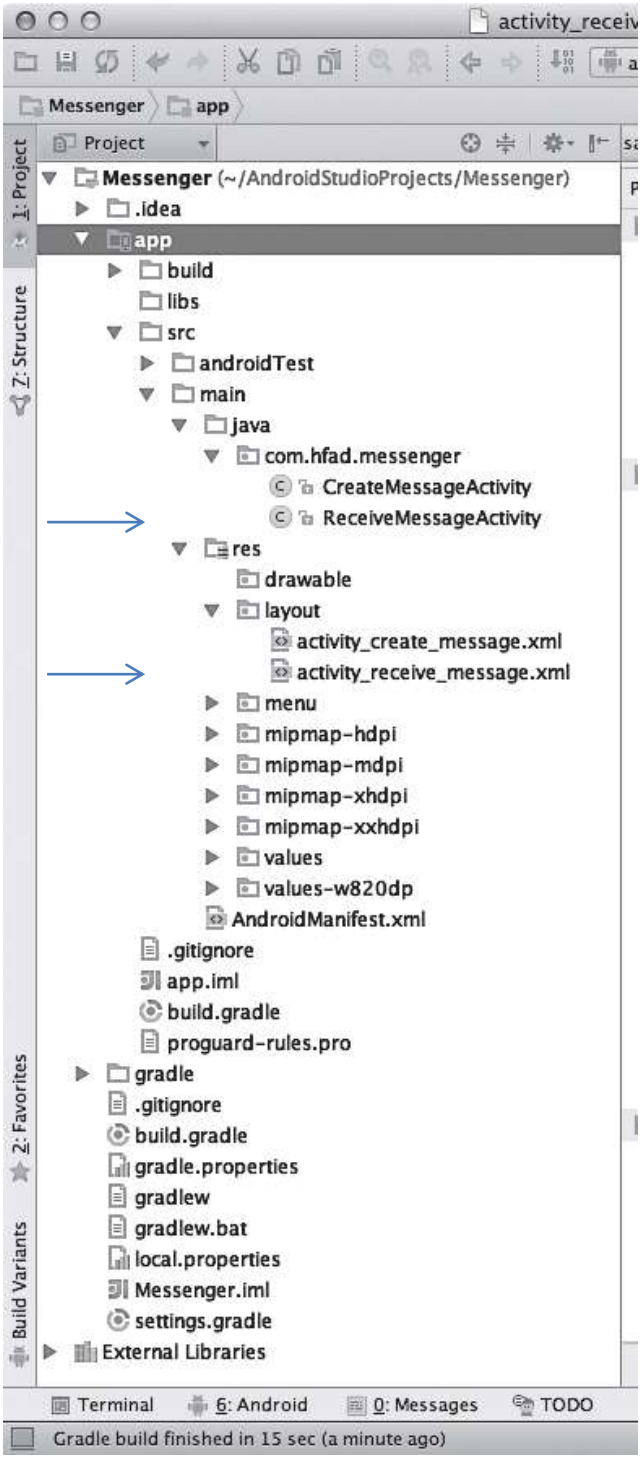
Отже, перша активність готова; переходимо до другої.

**3 Додавання другої активності та макета**

Android Studio має майстер для створення нових активностей і макетів у додатках. Щоб створити нову активність, виконайте File → New → Activity та виберіть варіант Blank Activity. На екрані з'являється вікно, в якому можливо вибрати параметри нової активності. Кожній створюваній новій активності та макету необхідно присвоїти ім'я. Задайте для активності ім'я “ReceiveMessageActivity”, а для макету – ім'я “activity_receive_message”. Переконайтеся, що пакет має назву “com.hfad.messenger”. Підтвердить інші значення за замовчуванням, а коли все буде готове, клацніть на кнопці Finish.



Android Studio створює новий файл активності разом із новим макет. На панелі структури проекту, бачимо, що у папці app/src/main/java з'явився новий файл з ім'ям ReceiveMessageActivity.java, а в папці app/src/main/res/layout - файл з ім'ям activity_receive_message.xml.




Кожна активність використовує окремий макет. CreateMessageActivity використовує макет activity_create_message.xml, а ReceiveMessageActivity – макет activity_receive_message.xml.

Android Studio також змінює конфігурацію програми у файлі з ім'ям AndroidManifest.xml. Кожна Android-програма повинна містити файл з ім'ям AndroidManifest.xml, якій знаходитися в папці app/src/main вашого проекту.

Файл AndroidManifest.xml містить важливу інформацію про програму: які активності воно містить, які бібліотеки йому необхідні та інші оголошення. Android створює цей файл під час створення програми. Наш екземпляр AndroidManifest.xml виглядає так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.
android.com/apk/res/android"package="c
om.hfad.messenger" >
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
      <activity
        android:name=".CreateMessageActivity"
        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=".ReceiveMessageActivity"
            android:label="@string/title_activity_receive_message" >
        </activity>
</application>
</manifest>
```

Усі активності мають бути оголошені у файлі AndroidManifest.xml. Якщо активність не оголошена у файлі, система не знатиме про його існування. А якщо система не знає про активність, то активність не виконуватиметься.

Активності оголошуються в маніфесті включенням елемента <activity> до елемента <application>. Власне кожна активність у додатку повинна мати відповідний елемент <activity>. Загальний формат оголошення виглядає так:

```
    <application
        ...
        ...>
        <activity
            android:name="имя_класса_активности"
            android:label="@string/метка_активности"
            ...
            ...>
        ...
        </activity>
        ...
    </application>
```

Наступний рядок є обов'язковим та використовується для визначення імені класу активності:

```
android:name="имя_класса_активности"
```

де имя_класа_активности — ім'я класу з префіксом “.”, у разі .ReceiveMessageActivity. Ім'я класу має префікс “.”, тому що Android будує повне ім'я класу, поєднуючи ім'я класу з ім'ям пакета.

Наступний рядок не є обов'язковим; вона задає мітку активності, зручну для користувача:

```
android:label="@string/мітка_активності"
```

Мітка виводиться у верхній частині екрана під час активності. Якщо прибрати її, то Android використовуватиме замість неї ім'я програми. До оголошення активності також можуть включатися й інші властивості - наприклад, дозволи безпеки або можливість використання активностей інших додатків.

На даний момент створено додаток із двома активностями, кожна з яких має власний макет. При запуску програми буде виконуватися наша перша активність CreateMessageActivity. Наступний крок – змусити CreateMessageActivity викликати ReceiveMessageActivity, коли користувач клацає на кнопці Send Message.

Щоб запустити одну активність із іншої, скористайтесь інтентом. Інтент можна як свого роду «намір виконати якусь операцію». Це різновид повідомлень, що дозволяє зв'язати різноманітні об'єкти (наприклад, активності) на стадії виконання. Якщо одна активність хоче запустити іншу, вона відправляє інтент для системи Android. Android запускає другу активність та передає їй інтент.

Щоб запустити активність, створіть інтент та використовуйте його у методі startActivity(). Процедура створення та надсилання інтенту складається всього з двох рядків коду. Для початку створіть інтент:

Intent intent = new Intent(this, Target.class);

Перший параметр повідомляє Android, від якого об'єкта надійшов інтент; Для визначення поточної активності використовується ключове слово this. У другому параметрі передається ім'я класу активності, що має отримати інтент. Після того, як інтент буде створено, він передається Android наступним викликом:

startActivity(intent);

Цей виклик наказує Android запустити активність, що визначається інтентом.

При отриманні інтенту Android переконується, що все правильно, і наказує активності запуститися. Якщо знайти активність не вдалося, ініціюється виключення ActivityNotFoundException.

А тепер застосуємо цю схему на практиці та використовуємо інтент для виклику ReceiveMessageActivity. Активність повинна запускатися, коли користувач клацає на кнопці Send Message, тому додаємо два рядки коду методом onSendMessage(). Внесемо зміни, виділені жирним шрифтом:

```
package com.hfad.messenger;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
public class CreateMessageActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }
    //Викликати onSendMessage() при натисканні на кнопці
    public void onSendMessage(View view) {
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        startActivity(intent);
```

Під час запуску програми відбувається наступне:

1. При запуску програми починає працювати його головна активність CreateMessageActivity. У конфігурації активності вказано, що вона використовує макет activity_create_message.xml. Цей макет відображається у новому вікні.

2. Користувач клацає на кнопці. Метод onSendMessage() класу CreateMessageActivity реагує на клацання.

3. Метод onSendMessage() наказує Android запустити активність ReceiveMessageActivity за допомогою інтенту. Android переконується в тому, що інтент правильний і після цього наказує ReceiveMessageActivity запуститися.

4. При запуску активність ReceiveMessageActivity повідомляє, що вона використовує макет activity_receive_message.xml; цей макет відображається у новому вікні.

Збережіть зміни та запустіть програму. Активність CreateMessageActivity запускається, а при натисканні на кнопці Send Message вона запускає ReceiveMessageActivity.


**4 Організація виклику другої активності з першої**

На даний момент запрограмоване активність CreateMessageActivity так, щоб вона запускала ReceiveMessageActivity при натисканні на кнопку Send Message. Тепер необхідно забезпечити передачу тексту з CreateMessageActivity в ReceiveMessageActivity, щоб активність ReceiveMessageActivity могла вивести отриманий текст. Для цього необхідно зробити три речі:

1. Змінити макет activity_receive_message.xml так, щоб він міг використовуватись для виведення тексту. Зараз він є макет за замовчуванням, згенерований майстром.

2. Оновити активність CreateMessageActivity.xml, щоб вона отримувала введений користувачем текст. Цей текст має бути доданий до інтенту перед його відправкою.

3. Оновити код ReceiveMessageActivity.java, щоб він виводив текст, надісланий в інтенті.

Ось як виглядає макет activity_receive_message.xml, згенерований Android Studio:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">
    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

У макет необхідно внести кілька змін. По-перше, потрібно привласнити елементу <TextView> ідентифікатор message, щоб до нього можна було звертатися з коду активності, а по-друге, потрібно заблокувати виведення тексту Hello world!

У макет необхідно внести кілька змін. Насамперед необхідно призначити ідентифікатор елементу <TextView>. Ідентифікатори повинні призначатися всім компонентам графічного інтерфейсу, з якими працюватиме в коді активності, - ідентифікатор використовується для звернення до компоненту з коду Java. Також необхідно скасувати висновок у макеті тексту “Hello world!”. Щоб внести обидві зміни, наведіть макет до наступного:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">
    <TextView
        android:id="@+id/message"
        ~~android:text="@string/hello_world"~~
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Замість видалення рядка android:text="@string/hello_world" також можна було б у файлі strings.xml присвоїти строковому ресурсу hello_world порожнє значення. Отже, після внесення змін до макету можна переходити на роботу з активностями.

До інтенту також можна додати додаткову інформацію, яка повинна передаватися одержувачу. І тут активність, яка отримала інтент, зможе нього якось зреагувати. Для цього використовується метод putExtra()

```
intent.putExtra("повідомлення", значення);
```

де повідомлення - ім'я ресурсу для інформації, що передається, а значення - саме значення. Перевантаження методу putExtra() дозволяє передавати значення багатьох можливих типів. Наприклад, це може бути примітив (скажімо, boolean або int), масив примітивів або String. Багаторазові виклики putExtra() дозволяють включити до інтенту кілька екземплярів додаткових даних. Для отримання додаткової інформації з інтенту існує пара зручних методів, які допомагають у вирішенні цього завдання.

Перший метод: getIntent() повертає інтент, що запустив активність; з отриманого інтенту можна прочитати будь-яку інформацію, відправлену разом з ним. Конкретний спосіб читання залежить від типу надісланої інформації. Наприклад, якщо інтент включає рядкове значення під назвою “message”:

```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

Звичайно, з інтенту можна читати не лише строкові значення. Наприклад, виклик

```
int intNum = intent.getIntExtra("name", default_value);
```

може використовуватися щоб отримати значення int з ім'ям name. Параметр default_value вказує, яке значення int слід використовувати за замовчуванням.

Тепер потрібно змінити ReceiveMessageActivity для використання переданого тексту. ReceiveMessageActivity відображатиме повідомлення у своєму написі під час створення активності. Оскільки метод onCreate() активності викликається відразу при її створенні, код буде додано до цього методу. Щоб отримати повідомлення з інтенту, спочатку отримаємо об'єкт інтенту викликом getIntent(), а потім самі дані, що передаються викликом getStringExtra().

Нижче наведено повний код ReceiveMessageActivity.java (замініть код, згенерований Android Studio, та збережіть всі зміни):

```
package com.hfad.messenger;
import android.os.Bundle;
import android.app.Activity;
//Необхідно імпортувати класи Intent та TextView.
import android.content.Intent;
import android.widget.TextView;
public class ReceiveMessageActivity extends Activity {
    //Ім'я додаткового значення, яке передається в інтенті.
    public static final String EXTRA_MESSAGE = "message";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receive_message);
        Intent intent = getIntent();
  //Отримати інтент та витягти з нього повідомлення викликом getStringExtra().
        String messageText = intent.getStringExtra(EXTRA_MESSAGE);
        TextView messageView = (TextView)findViewById(R.id.message);
```

```
            //Додати текст до напису з ідентифікатором message.
            messageView.setText(messageText);
        }
    }
```

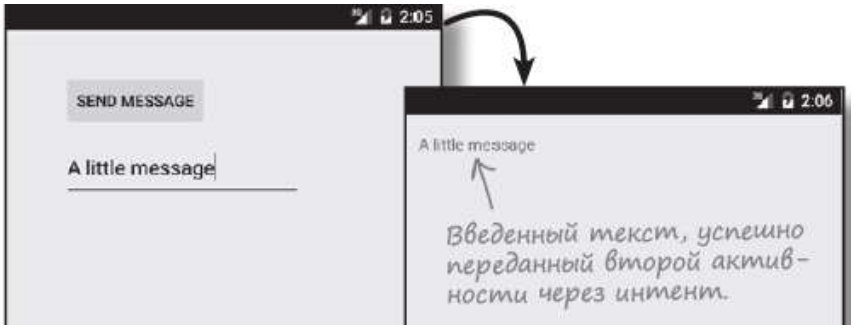
При натисканні на кнопці Send Message відбувається наступне:

1. Коли користувач клацає на кнопці, викликається метод onSendMessage(). Код методу onSendMessage() створює інтент для запуску активності ReceiveMessageActivity, додає в інтент текст повідомлення та передає його Android разом із інструкцією із запуску активності.

2. Android перевіряє інтент на правильність і наказує ReceiveMessageActivity запуститися.

3. Під час запуску активність ReceiveMessageActivity вказує, що вона використовує макет activity_receive_message.xml. Цей макет відображається на пристрої. Активність змінює макет та виводить у ньому текст, отриманий з інтенту.

Переконайтеся, що внесено зміни в обох активах, збережіть зміни та запустіть програму. Запускається активність CreateMessageActivity; Якщо ввести текст і натиснути кнопку Send Message, запускається ReceiveMessageActivity. Текст, введений у першій активності, з'являється у написі.



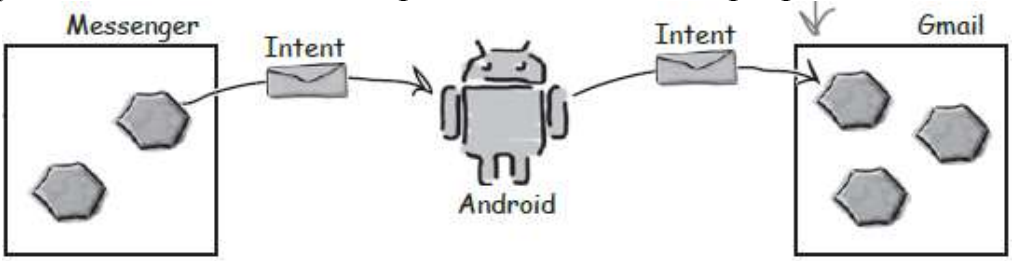

## 5 Організація передачі з першої активності в другу

Тепер, коли наша програма навчилася відправляти повідомлення іншої активності, його можна змінити так, щоб воно надсилало повідомлення іншим людям. Для цього програма інтегрується з іншими програмами, що підтримують надсилання повідомлень та вже встановлених на пристрої. В залежності від того, які програми встановлені у користувача, можна організувати надсилання повідомлень із програми через Gmail, Google+, Facebook, Twitter та ін.

Інтенти можуть запускати активності з інших програм. Перша активність передає інтент Android; Android перевіряє інтент, а потім наказує другий активності запуститися.

Цей принцип відноситься і до активності інших додатків. Активність програми передає інтент Android, Android перевіряє його, а потім наказує другий активності запуститися - незважаючи на те, що ця активність знаходиться в іншому додатку. Наприклад, можна скористатися інтентом для запуску активності Gmail, що надсилає повідомлення, і надіслати їй текст, який

потрібно надіслати. Замість того, щоб писати власні активи для надсилання електронної пошти, можна скористатися готовою програмою Gmail.




Це означає, що об'єднуючи активності на пристрої в ланцюжок, ви можете будувати програми, що мають значно більшу функціональність.

Перш ніж викликати активності з інших програм, необхідно відповісти на три питання:

– Як дізнатися, які активності доступні на пристрої користувача?
– Як дізнатися, які з цих активностей підходять для того?
– Як дізнатися, як використовувати ці активності?

На щастя, ці проблеми вирішуються з допомогою дій (actions). Дії - стандартний механізм, за допомогою якого Android дізнається про те, які стандартні операції можуть виконуватись активностями. Наприклад, Android знає, що всі активності, зареєстровані для дії send можуть надсилати повідомлення.

А тепер потрібно навчитися створювати інтенти, які використовують дії для отримання набору активностей, які можуть використовуватись для виконання стандартних функцій - наприклад, для надсилання повідомлень. Для цього потрібно:

1. Створити інтент із зазначенням дії. Інтент повідомить Android, що потрібна активність, яка вміє відправляти повідомлення. Інтент включатиме текст повідомлення.

2. Дозволити користувачеві вибрати програму, що використовується. Швидше за все, на пристрої встановлено відразу кілька додатків, здатних надсилати повідомлення, тому користувач повинен вибрати одне з них. Необхідно, щоб користувач міг вибрати програму щоразу,коли він клацає на кнопці Send Message.

Створення інтенту із зазначенням дії. Створення інтенту для запуску конкретної активності виконується командою виду:

Intent intent = new Intent(this, ReceiveMessageActivity.class);

Такі інтенти називаються явними; тобто Android явно повідомляєте, який клас має запустити система. Якщо потрібно виконати деяку дію і вас не цікавить, якою вона буде виконана, створіть неявний інтент. При цьому повідомляємо Android, яку дію потрібно виконати, а всі подробиці вибору активності доручаються Android. Для створення інтенту із зазначенням дії застосовується наступний синтаксис:

Intent intent = new Intent (дія);

де дія - тип дії, що виконується активністю. Android надає цілу низку

стандартних варіантів дій. Наприклад, дія Intent.ACTION_DIAL використовується для набору номера, Intent.ACTION_WEB_SEARCH - для виконання веб-пошуку, а Intent.ACTION_SEND - для надсилання повідомлень. Отже, якщо при створенні інтенту для надсилання повідомлення, скористаємося командою такого вигляду:

Intent intent = new Intent(Intent.ACTION_SEND);

Після визначення дії до інтенту можна включити додаткову інформацію. - додати текст, який утворює тіло повідомлення, що надсилається. Завдання вирішується наступним фрагментом коду:

intent.setType("text/plain");

intent.putExtra(Intent.EXTRA_TEXT, текст);

де текст - текст, що надсилається. Виклик повідомляє Android, що активність повинна вміти обробляти дані з типом даних MIME "text/plain", а також передає текст.

Якщо потрібно додати кілька видів додаткової інформації, використовуємо багаторазові виклики методу putExtra(). Наприклад, якщо ви хочете вказати тему повідомлення, використовуйте виклик виду

intent.putExtra(Intent.EXTRA_SUBJECT, тема);

де тема – тема повідомлення.

Змінимо файл CreateMessageActivity.java так, щоб у ньому створювався неявний інтент використання дії відправки. Внесіть зміни, виділені жирним шрифтом, та збережіть свою роботу:

```
package com.hfad.messenger;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;
public class CreateMessageActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }
//Викликати onSendMessage() при натисканні на кнопці
public void onSendMessage(View view) {
    EditText messageView = (EditText)findViewById(R.id.message);
    String messageText = messageView.getText().toString();
    Intent intent = new Intent(this, ReceiveMessageActivity.class);
    intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE,
messageText);
//Замість того, щоб створювати інтент, призначений безпосередньо для
ReceiveMessageActivity, створюємо інтент із зазначенням дії відправлення.
    ~~Intent intent = new Intent(Intent.ACTION_SEND);~~
    ~~intent.setType("text/plain");~~
```

```
            intent.putExtra(Intent.EXTRA_TEXT, messageText);
            startActivity(intent);
        }
}
```

Під час виконання коду відбувається наступне:

1. При виклику методу onSendMessage() створюється інтент. Метод startActivity() передає інтент Android. Інтенту призначається дія ACTION_SEND та тип MIME text/plain.

2. Android бачить, що інтент може передаватися тільки активності, здатною обробляти дію ACTION_SEND і дані text/plain. Android перевіряє всі активності та шукає серед них ті, які зможуть обробити інтент. Якщо жодна дія не здатна обробити інтент, ініціюється виключення ActivityNotFoundException.

3. Якщо тільки одна активність здатна обробити інтент, Android наказує цій активності запуститися та передає їй інтент.

4. Якщо знайдеться кілька активностей, здатних обробити інтент, Android відкриває діалогове вікно для вибору активності та пропонує користувачеві вибрати.

5. Коли користувач вибере активність, яку хоче використовувати, Android наказує активності запуститися і передає їй інтент. Активність виводить додатковий текст, який міститься в інтенті, у тілі нового повідомлення.

**Висновки.** Завданням називаються дві і більше активності, об'єднані в ланцюжок. Елемент <EditText> визначає текстове поле з можливістю редагування та введення тексту. Клас текстового поля успадковує від класу Android View.

Нова активність в Android Studio створюється командою File → New... → Activity. Для кожної створюваної активності у файлі AndroidManifest.xml повинен бути створений запис.

Інтент являє собою різновид повідомлень, що використовуються для організації взаємодії між компонентами Android. Явний інтент призначений для конкретного компонента. Явний інтент створюється командою

Intent intent = new Intent(this, Target.class);

Активності запускаються викликом startActivity(intent). Якщо жодна відповідна активність не знайдена, метод ініціює виключення ActivityNotFoundException. Використовуйте метод putExtra() для включення додаткової інформації в інтент. Використовуйте метод getIntent() для отримання інтенту, який запустив активність. Використовуйте методи get*Extra() для читання додаткової інформації, пов'язаної з інтентом. Метод getStringExtra() читає String, getIntExtra() читає int, і т.д.

Дія описує стандартну операцію, яку може виконувати активність. Так, для надсилання повідомлень використовується позначка Intent.ACTION_SEND.

Щоб створити неявний інтент із зазначенням дії, використовуйте запис

Intent intent = new Intent(action);

Для опису типу даних в інтенті використовується метод setType(). Android виробляє дозвіл інтентів на підставі імені компонента, дії, типу даних і категорій, вказаних в інтенті. Вміст інтенту порівнюється з фільтрами інтентів з

AndroidManifest.xml кожної програми. Щоб активність отримувала неявні інтенти, вона має містити категорію DEFAULT.

Метод createChooser() дозволяє перевизначити стандартне діалогове вікно вибору активності в Android. При використанні цього методу можна вказати текст заголовка, а у користувача немає можливості призначити активність за замовчуванням. Якщо метод не знаходить жодної активності, здатної отримати переданий інтент, він виводить повідомлення. Метод createChooser() повертає об'єкт Intent. Для читання значень рядкових ресурсів використовується синтаксис getString(R.string.stringname);.

**Практичне заняття № 4. Користувацький інтерфейс**

**Кількість годин:** 2 год.

**Навчальна мета заняття**:

1. Придбання теоретичних знань з теми «Структура та компоненти мобільного додатку», розвиток здібностей до творчого мислення, формування навичок самостійної роботи з аналізу і узагальнення інформації, вміння проектувати компонентну архітектуру мобільного додатку.

**Рекомендована література:**


1. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.
2. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
3. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
4. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
5. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
6. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.
7. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
8. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.


**Матеріально-технічне забезпечення:** комп'ютерний клас.

**Навчальні питання:**

1. Ключові макети: відносний, лінійний та табличний
2. Компоненти графічного інтерфейсу
3. Спискові уявлення та адаптери
4. Фрагменти

1. ПОРЯДОК ПРОВЕДЕННЯ ЗАНЯТТЯ:

1.1. Проведення експрес-контролю готовності до заняття.

1.2. Ввести текст підготовленої програми і виконати її відлагодження.

1.3. Підібрати тести і виконати відпрацювання розробленого алгоритму на цих тестах.

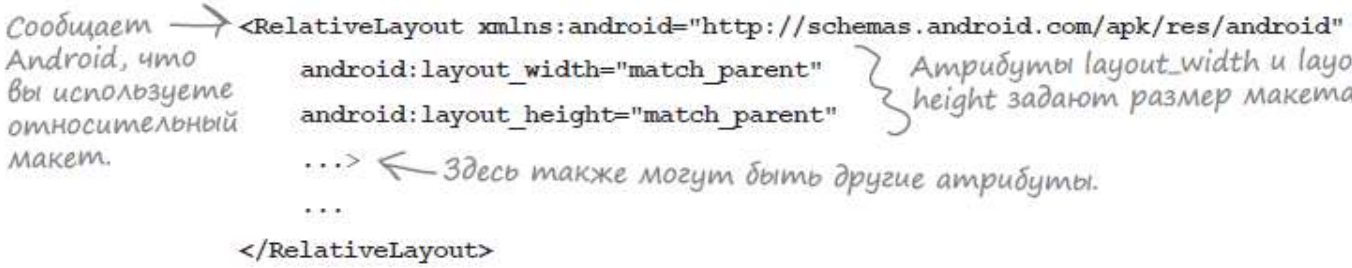
1.4. Скласти звіт про виконану роботу і здати роботу викладачу.

**1 Ключові макети: відносний, лінійний та табличний**

Інтерфейс користувача складається з макетів та компонентів графічного інтерфейсу. Макет визначає зовнішній вигляд екрану, а для опису використовується формат розмітки XML. Макети зазвичай містять компоненти графічного інтерфейсу - кнопки, текстові поля і т. д. Користувач взаємодіє з ними, щоб програма виконувала потрібні операції.

Макети поділяються на кілька різновидів, і кожний з них дотримується своїх правил для прийняття рішень про позиціонування уявлень, що містяться в ньому (views). Нижче наведено коротке зведення трьох найважливіших різновидів макетів: відносний, лінійний та табличний.

У *відносному макеті* (RelativeLayout) уявлення, що входять у нього, розміщуються у відносних позиціях. Позиція кожного уявлення визначається щодо інших уявлень у макеті чи щодо його батьківського макета. Наприклад, напис можна розмістити щодо верхнього краю батьківського макета, список, що розкривається, розмістити під текстовим поданням, а кнопку - щодо нижнього краю батьківського макета.




Атрибут xmlns:android використовується визначення простору імен Android; йому завжди надається значення "http://schemas.android.com/apk/res/android".

Атрибути android:layout_width та android:layout_height визначають ширину та висоту макета. Ці атрибути є обов'язковими для всіх типів макетів та уявлень.

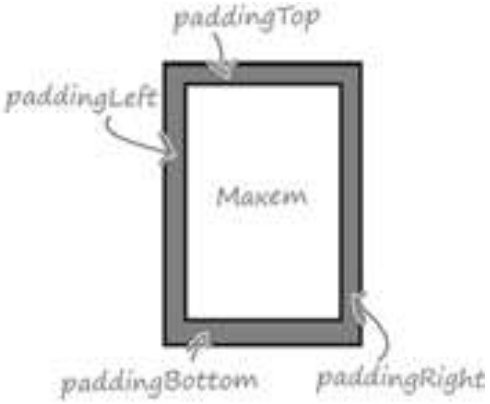
Атрибутам android:layout_width і android:layout_height можна встановити як узагальнені значення "match_parent", "wrap_content", так і конкретні розміри, наприклад 10dp (10 апаратно-незалежних пікселів). Значення "wrap_content" означає, що розміри макета мають бути мінімально достатніми для того, щоб розмістити всі уявлення, а "match_parent" означає, що розміри макета вибираються за розмірами батька - в даному випадку це розмір екрану за вирахуванням відступів. Найчастіше ширині та висоті макета задається значення "match_parent".

Відступи. Якщо ви бажаєте, щоб макет оточував деякий порожній

простір, скористайтеся атрибутами padding. Ці атрибути повідомляють Android, яку відстань має відокремлювати сторони макета від батька. Наступний фрагмент наказує Android додати до всіх сторін макета відступи завбільшки 16dp:

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Добавить отступы величиной 16dp.

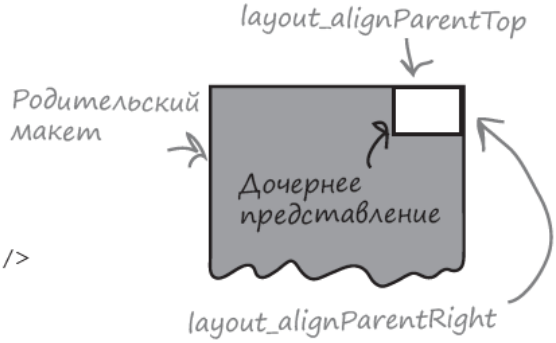



Атрибути android:padding* не є обов'язковими, але вони можуть бути використані з будь-яким макетом або поданням. Також можна задати розмір відступів в ресурсному файлі розмірів. Використання ресурсів полегшує керування відступами всіх макетів вашої програми. Щоб використовувати ресурсний файл розмірів, вкажіть в атрибутах відступів з макету імена ресурсів розмірів.

При використанні відносного макета необхідно повідомити Android, де мають розташовуватися уявлення стосовно інших уявлень у макеті, або його батькових виставив. Батькові вистави є макет, що містить цю виставу. Якщо ви хочете, щоб подання завжди відображалося у певній позиції екрана незалежно від його розмірів та орієнтації, позиціонуйте подання щодо його батька. Наприклад, щоб кнопка завжди розташовувалась у правому верхньому куті макета, використовуйте таку розмітку:

```
<RelativeLayout ... >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
</RelativeLayout>
```

Макет содержит кнопку, поэтому макет является родителем кнопки.




Рядки
android:layout_alignParentTop="true"
android:layout_alignParentRight="true"
означають, що верхня сторона кнопки вирівнюється по верхньому краю макета, а права сторона кнопки вирівнюється праворуч макета. Таке розміщення буде застосовуватися незалежно від розміру екрана або орієнтації пристрою.

Перед вами зведення атрибутів, що найчастіше використовуються при позиціонуванні уявлень щодо батьківського макета. Увімкніть потрібний атрибут в уявлення, положення якого потрібно визначити, та надайте йому значення "true":

android:attribute="true"

Атрибути для позиціонування уявлень щодо батьківського макета:

| *Атрибут* | *Що робить* |
|---|---|
| android: layout_alignParentBottom | Нижній край вистави вирівнюється по нижньому краю батька |
| android: layout_alignParentLeft | Лівий край вистави вирівнюється по лівому краю батька. |
| android: layout_alignParentRight | Правий край уявлення вирівнюється правому краю батька. |
| android: layout_alignParentTop | Верхній край вистави вирівнюється по верхньому краю батька. |
| android: layout_centerInParent | Вирівнюється по центру всередині батька (по горизонт. та вертикалі). |
| android: layout_centerHorizontal | Вирівнюється центром всередині батька (по горизонталі). |
| android: layout_centerVertical | Вирівнюється центром всередині батька (по вертикалі). |

Крім позиціонування щодо батьківського макета, ви також можете розміщувати уявлення щодо інших уявлень. Ця можливість застосовується у тих випадках, якщо уявлення повинні зберігати вирівнювання незалежно від розміру чи орієнтації екрана.

Щоб визначити позицію подання щодо іншого подання, слід призначити ідентифікатор того подання, яке використовується як якорь. Для цього використовується атрибут android:id:

android:id="@+id/button_click_me"

Синтаксис "@+id" наказує Android увімкнути ідентифікатор у вигляді ресурсу у файл ресурсів R.java. Якщо пропустити "+", Android не додасть ідентифікатор як ресурс, і під час виконання коду виникнуть помилки.

У наступному прикладі створюється макет із двома кнопками: перша кнопка розташована по центру макета, а друга кнопка розміщується під першою:

```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/click_me" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button_click_me"
        android:layout_below="@+id/button_click_me"
        android:text="@string/new_button_text" />
</RelativeLayout>
```

Эта кнопка используется в качестве якоря для второй, поэтому ей необходимо присвоить идентификатор.

Рядки

android:layout_alignLeft="@+id/button_click_me"

android:layout_below="@+id/button_click_me"

гарантують, що лівий край другої кнопки вирівнюватиметься по лівому краю першої та друга кнопка завжди буде розміщуватися під першою.

Атрибути для позиціонування уявлень щодо інших уявлень:

*Атрибут* *Що робить*

android:layout_above Подання розміщується над якірним уявленням.

android:layout_below Подання розміщується під якірним уявленням.

android:layout_alignTop Верхній край вистави вирівнюється по верхньому краю якірного уявлення.

android:layout_alignBottom Нижній край вистави вирівнюється по нижньому краю якірного уявлення.

android:layout_alignLeft Лівий край вистави вирівнюється по лівому краю якірного уявлення.

android:layout_alignRight Правий край уявлення вирівнюється правому краю якірного уявлення.

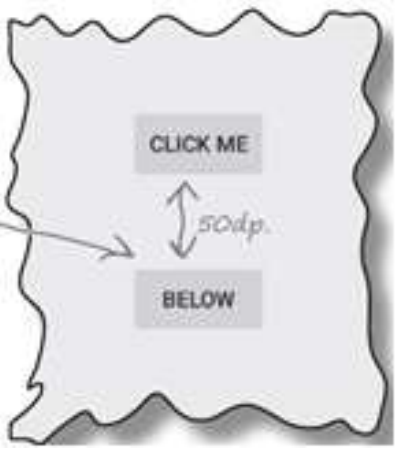
android:layout_toLeftOf Правий край уявлення розташовується біля лівого краю якірного уявлення.

android:layout_toRightOf Лівий край уявлення розташовується біля правого краю якірного уявлення.

Коли ви застосовуєте атрибути для розміщення, вистави розташовуються впритул один до одного. Щоб уявлення поділялися проміжками, додайте до уявлень інтервали. Допустимо, ви хочете, щоб одне уявлення розміщувалося під іншим, але вони розділялися додатковим проміжком величиною 50dp. Для цього до верхнього краю нижньої вистави додається інтервал величиною 50dp:

```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        ... />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button_click_me"
        android:layout_below="@+id/button_click_me"
        android:layout_marginTop="50dp"
        android:text="@string/button_below" />
</RelativeLayout>
```

*При добавлении интервала к верхнему краю нижней кнопки два представления разделяются дополнительным промежутком.*




Нижче наведено інтервали, які можуть використовуватися для створення додаткових інтервалів між уявленнями. Додайте атрибут у виставу і надайте йому значення - величинуінтервалу:

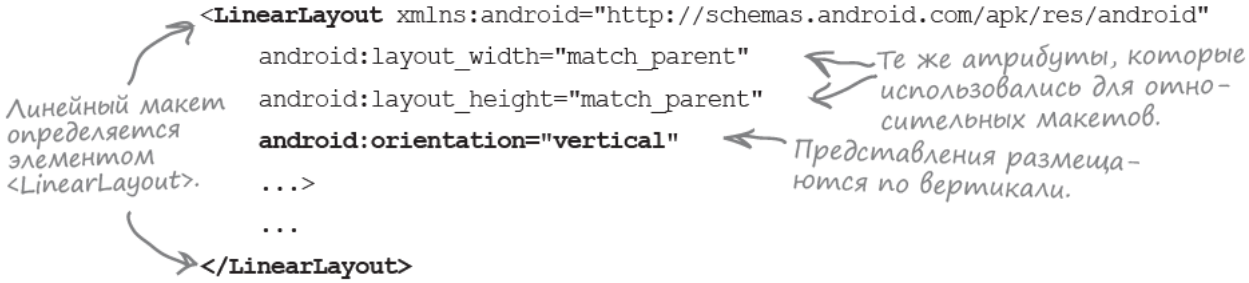
android:attribute="10dp"

*Атрибут* *Що робить*

android:layout_marginTop Додає додатковий інтервал у верхнього краю

уявлення.
android:layout_marginBottom Додає додатковий інтервал у нижнього краю уявлення.
android:layout_marginLeft Додає додатковий інтервал у лівого краю вистави.
android:layout_marginRight Додає додатковий інтервал у правого краю уявлення.

У *лінійному макеті* (LinearLayout) вистави розміщуються поруч по вертикалі або горизонталі. Якщо використовується вертикальне розташування, уявлення відображаються в один стовпець. У варіанті з горизонтальним розміщенням уявлення виводяться в один рядок. Лінійний макет визначається за допомогою елемента <LinearLayout>:




Атрибути android:layout_width, android:layout_height та android:orientation є обов'язковими. Атрибути android:layout_width та android:layout_height визначають ширину та висоту макета, як і для відносних макетів. Атрибут android:orientation задає напрямок розміщення уявлень.

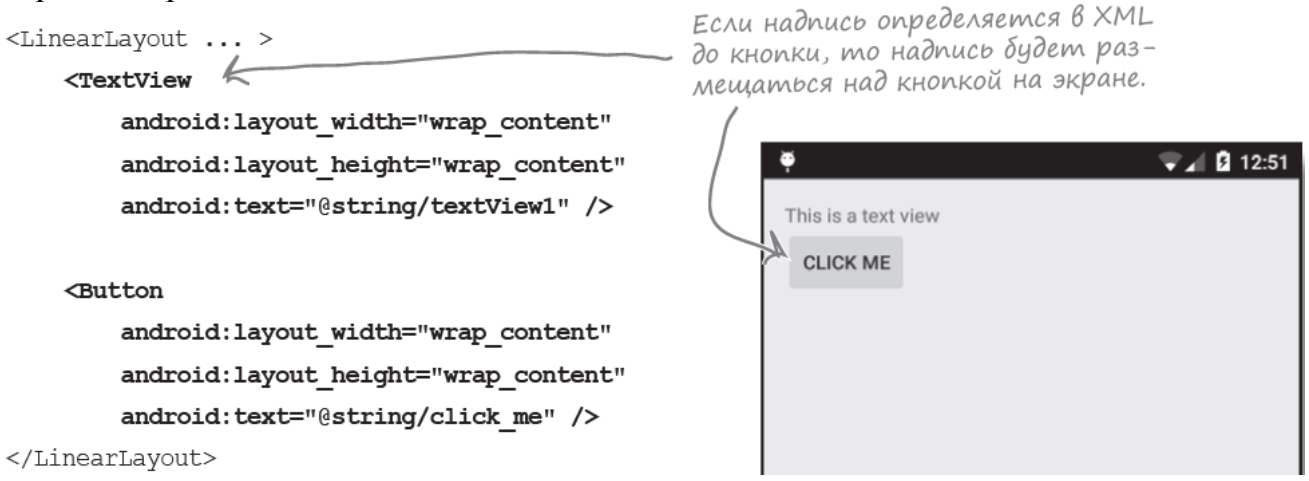
Щоб розмістити уявлення по вертикалі, використовуйте атрибут:

android:orientation="vertical"

Для горизонтального розміщення уявлень використовується наступний атрибут:

android:orientation="horizontal"

У лінійному макеті вистави відображаються у порядку їх прямування в розмітці XML. При визначенні лінійного макету подання включаються до макету в тому порядку, в якому вони повинні йти на екрані. Отже, якщо ви хочете, щоб напис розміщувався над кнопкою, напис повинен визначатися першою в розмітці:




У лінійному макеті ідентифікатори уявлень знадобляться лише в тому випадку, якщо ви збираєтесь явно звертатися до них із коду активності. Справа

в тому що у лінійному макеті позиція кожного подання визначається його порядком у розмітці XML. Щоб вказати, де має розміщуватися уявлення, розробнику не потрібно звертатися до інших уявлень.

Як і з відносними макетами, ширина та висота уявлень задаються атрибутами android:layout_width та android:layout_height. Атрибут:

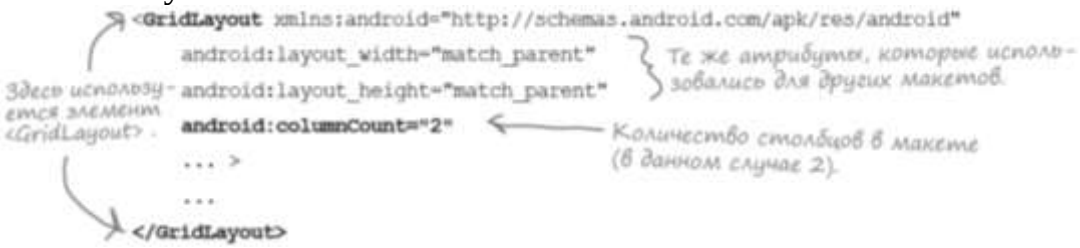
android:layout_width="wrap_content"

означає, що ширина вистави має бути мінімально необхідною для того, щоб у ньому помістилося всій вміст. Як, наприклад, під час виведення тексту на кнопці або у написі. Атрибут:

android:layout_width="match_parent"

означає, що ширина вистави визначається шириною батьківського макета.

У *табличному макеті* (GridLayout) екран ділиться на рядки та стовпці, на перетині яких знаходяться комірки. Визначення табличного макета дуже схоже визначення інших типів макетів, тільки в цьому випадку використовується елемент <GridLayout>:




Кількість стовпців у табличному макеті задається наступним атрибутом:

android:columnCount="число"

де число - кількість стовпців. Також можна задати максимальне число рядків із використанням атрибуту:

android:rowCount="число"

але на практиці зазвичай краще довірити обчислення кількості рядків Android в залежності від кількості подань у макеті. Android створює стільки рядків, скільки потрібно для відображення всіх уявлень.

У табличний макет подання додаються приблизно так само, як і в лінійний:

```
<GridLayout ... >
        <TextView
                android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/textview" />
<Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/click_me" />
<EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:hint="@string/edit" />
</GridLayout>
```

Як і при використанні лінійного макета, немає необхідності призначати уявленням ідентифікатори, якщо ви не збираєтеся явно посилатися на них у коді активності. Уявленням не потрібно звертатися один до одного в макеті,

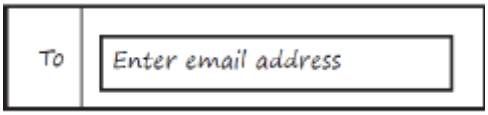
тому для цієї цілі ідентифікатори не знадобляться.

За умовчанням табличний макет розміщує уявлення в порядку їхнього прямування в XML. Якщо створити табличний макет з двома стовпцями, табличний макет помістить першу виставу в першій позиції, друге уявлення в другій позиції і т.д.

Таке рішення має один недолік: виняток одного з уявлень з макета може навести до серйозної зміни зовнішнього вигляду макета. Щоб уникнути подібних проблем, ви вказуєте, де має знаходитися кожне подання і скільки стовпців воно має позичати.

Створення нового макету починається з побудови ескізу. Це допоможе нам зрозуміти, скільки рядків і стовпців знадобиться, де має бути кожне уявлення і скільки стовпців воно має займати.


Табличний макет має складатися із двох стовпців. Отже, необхідне розташування уявлень досягається з табличним макетом, що складається з двох стовпців:




```
<GridLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >
    </GridLayout>
```

Після визначення основи табличного макета можна переходити до додавання уявлень. Перший рядок табличного макета складається з напису (у першому стовпці) та текстового поля (у другому стовпці).

```
<GridLayout...>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/to" />
        <EditText
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:hint="@string/to_hint" />
</GridLayout>
```

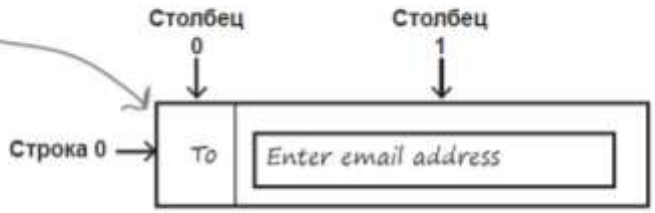
Атрибути android:layout_row і android:layout_column використовуються для позначення рядків і стовпців, у яких мають бути представлені. Індекси рядків та стовпців починаються з 0; отже, щоб подання розташовувалося в першому стовпці та першому рядку, потрібно задати такі значення:

```
android:layout_row="0"
android:layout_column="0"
```

*Индексы столбцов и строк начинаются с 0, поэтому эти атрибуты обозначают первую строку и первый столбец.*

Застосуємо ці позначення до розмітки макета: розмістимо напис у стовпці 0, а текстове поле у стовпці 1:

```
<GridLayout...>
    <TextView
        ...
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        ...
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />
</GridLayout>
```




Другий рядок табличного макета складається з текстового поля, яке починається у першому стовпці та поширюється на другий стовпець. Подання займає весь вільний простір.

Щоб уявлення займало кілька стовпців, необхідно спочатку вказати, з якого стовпця та рядка має починатися уявлення. Наше текстове поле має починатися у першому стовпці другого рядка, тому атрибути виглядають так:

```
android:layout_row="1"
android:layout_column="0"
```

Подання у нашому ескізі займає два стовпці. Щоб досягти цього, можна скористатися атрибутом android:layout_columnSpan наступного виду:

```
android:layout_columnSpan="число"
```

де число - кількість стовпців, які має займати виставу. У нашому прикладі атрибут має виглядати так:

```
android:layout_columnSpan="2"
```

Поєднуючи все сказане, ми приходимо до наступної розмітки поля Message:

```
<GridLayout...>
    <TextView... />
    <EditText.../>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />
</GridLayout>
```

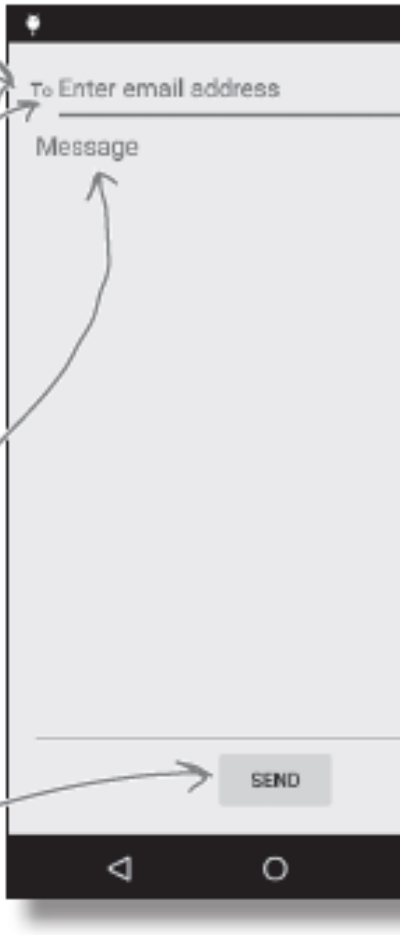
*Представления, добавленные на предыдущей странице для строки 0.*

*Текстовое поле занимает все свободное пространство, а текст отображается в верхней части.*

*Текстовое поле начинается в столбце 0 и занимает два столбца.*

Після додавання уявлень перших двох рядків залишається лише додати кнопку. Кнопка має бути вирівняна по горизонталі в центрі області, що складається з двох стовпців. Повний код табличного макета:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="2"
        android:layout_column="0"
        android:layout_gravity="center_horizontal"
        android:layout_columnSpan="2"
        android:text="@string/send" />
</GridLayout>
```

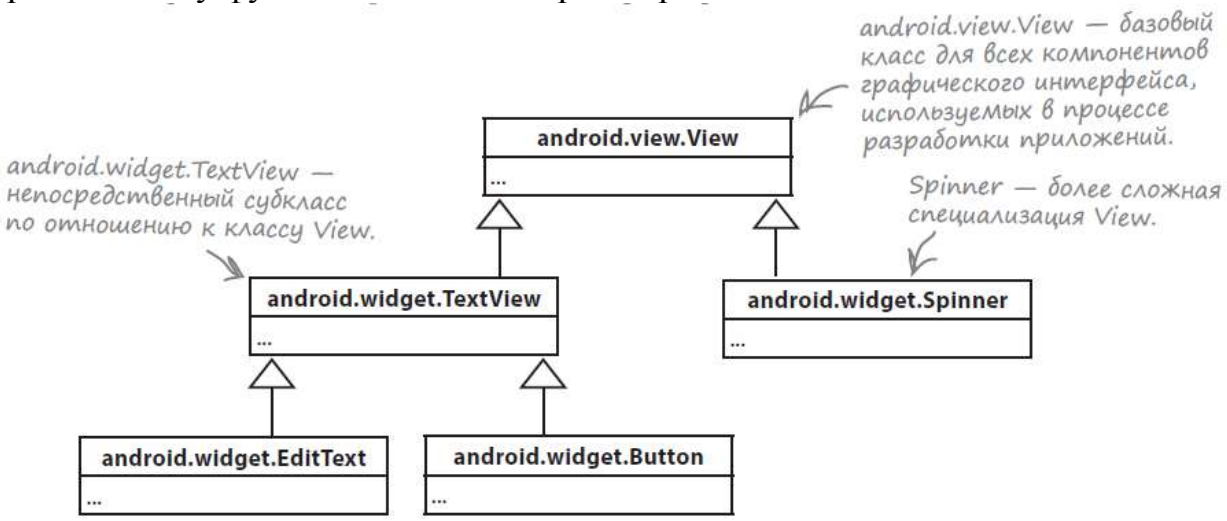



Кнопка занимает область из двух столбцов, начиная со столбца 1 строки 2. Кнопка выравнивается по центру области.

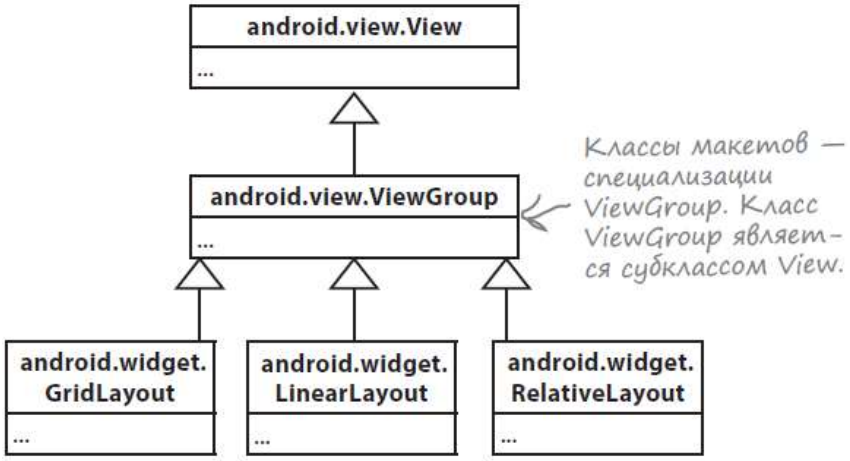
## 2 Компоненти графічного інтерфейсу

Компоненти графічного інтерфейсу є спеціалізаціями уявлень – у внутрішній реалізації вони є субкласами класу android.view.View. Це означає, що всі компоненти, що використовуються в інтерфейсі програми, мають спільні атрибути і поведінку. Наприклад, усі вони можуть відображатись на екрані, а також можуть повідомляти інформацію про свою ширину та висоту. Кожен

компонент графічного інтерфейсу, що використовується в інтерфейсі програми, бере цю базову функціональність та розширює її.




В ієрархії Android макет є спеціалізацією групи уявлень, що представлена класом android.view.ViewGroup. Група уявлень - особливий різновид уявлень, здатних містити інші уявлення.




Об'єкт View займає прямокутну область екрана. Він включає функціональність, необхідну для всіх уявлень для нормального існування в світі Android. Нижче наведено найважливіші деякі аспекти цієї функціональності:

*1 Читання та запис властивостей*. Кожна вистава представлена об'єктом Java; це означає, що ви можете задавати і читати його властивості в коді активності - наприклад, отримати значення, вибране в списку, або змінити текст напису. Конкретний набір властивостей та методів, які можуть використовуватись у коді, залежить від типу уявлення. Кожному уявленню можна призначити ідентифікатор, яким до нього можна звертатися з коду.

2 *Розмір та позиція*. За значеннями ширини та висоти, заданими в програмі, Android визначає необхідні розміри уявлення. Також можна вказати, чи потрібно забезпечити уявлення відступами. Після того, як уявлення з'явиться на екрані, ви зможете отримати дані про його позицію, а також визначити фактичні розміри.

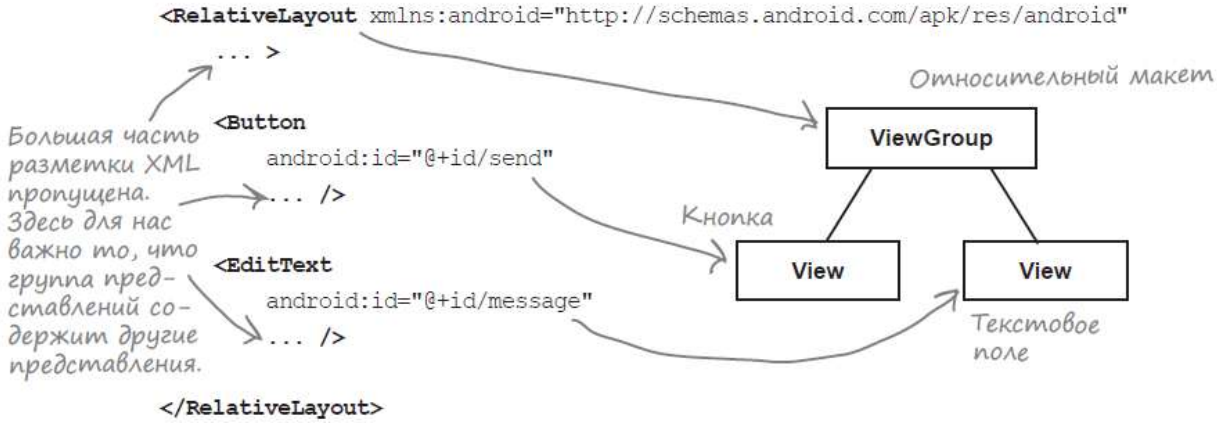
*3 Обробка фокусу*. Android керує передачею фокусу залежно від дій користувача. Зокрема, під час передачі фокусу враховуються події

приховування, видалення чи появи уявлень.

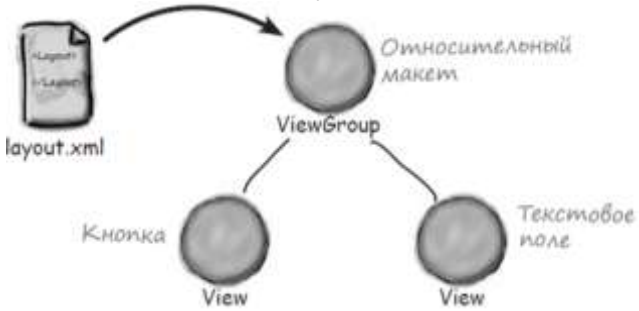
*4 Обробка подій та слухачі*. Кожне уявлення може реагувати на події. Також розробник може створювати слухачів (listeners) для реакції на події, що відбуваються у поданні. Наприклад, всі уявлення здатні реагувати на отримання або втрату фокусу, а кнопка (а також усі її субкласи) може реагувати на клацання.

Так як група уявлень також є спеціалізацією уявлення, це означає, що всі макети та компоненти графічного інтерфейсу також підтримують загальну функціональність.

Макет, що визначається в розмітці XML, формує ієрархічне дерево уявлень та груп уявлень. Наприклад, уявіть відносний макет із кнопкою та текстовим полем. Відносний макет є групою уявлень, а кнопка та текстове поле – уявленнями. Група уявлень є батьком текстового поля, а уявлення є нащадками стосовно групи:




При побудові XML-розмітка макета автоматично перетворюється на об'єкт ViewGroup, що містить дерево елементів View. У наведеному вище прикладі кнопка перетворюється на об'єкт Button, а напис перетворюється на об'єкт TextView. І Button, і TextView є субклас View.




Все це дозволяє працювати з уявленнями з макета в коді Java. Кожне уявлення непомітно для розробника перетворюється на об'єкт Java View. Розглянемо такі основні уявлення: надпис, текстове поле, кнопка, двопозиційна кнопка, вимикач, прапорець, перемикач, список, що розкривається, графічне уявлення, зображення/текст на кнопках, повідомлення тощо.

*Напис.* Використовується для виведення тексту. Напис визначається у макеті елементом <TextView>. Атрибут android:text вказує, який текст повинен виводитися - зазвичай у формі рядкового ресурсу:

```
<TextView
    android:id="@+id/text_view"
```

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

API TextView містить численні атрибути для керування зовнішнім виглядом тексту, наприклад розміром тексту. Для зміни розміру використовується атрибут android:textSize:

```
android:textSize="14sp"
```

Розмір тексту визначається в апаратно-незалежних пікселах (sp). Апаратно-незалежні пікселі враховують факт включення великих шрифтів на пристрої користувачів. Текст з розміром 14 sp на пристрої, налаштованому на використання великих шрифтів, буде фізично більше такого ж шрифту на пристрої з малими шрифтами. Для зміни тексту, що виводиться у написі, використовується програмний код такого вигляду:

```
TextView textView = (TextView) findViewById(R.id.text_view);
textView.setText("Some other string");
```

*Текстове поле*. Аналог напису, але з можливістю редагування. Текстове поле XML визначається елементом <EditText>. Атрибут android:hint задає текст підказки, яка пояснює користувачеві, яку інформацію слід вводити у поле

```
<EditText
    android:id="@+id/edit_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_text" />
```

Атрибут android:inputType визначає тип даних, які мають бути введені в поле. Ця інформація дозволяє Android допомогти користувачеві в процесі введення. Наприклад, якщо поле призначене для введення числових даних, використовуйте атрибут

```
android:inputType="number"
```

для вибору цифрової клавіатури. На наш погляд, найбільш корисні такі типи:

```
phone - Надає клавіатуру для введення номерів.
textPassword - Надає клавіатуру для введення тексту, дані, що вводяться маскуються.
textCapSentences - Перше слово в реченні починається з великої літери.
textAutoCorrect - Автоматично виправляє текст, що вводиться.
```

У значенні атрибута можна перерахувати кілька типів, розділених символом |. Наприклад, щоб перше слово речення починалося з великої літери, а в текст, що вводиться автоматично виправлялися помилки, використовуйте атрибут:

```
android:inputType="textCapSentences|textAutoCorrect"
```

Для отримання тексту, який міститься в текстовому полі, використовується програмний код такого вигляду:

```
EditText editText = (EditText) findViewById(R.id.edit_text);
String text = editText.getText().toString();
```

*Кнопка.* Зазвичай використовується для того, щоб додаток виконувало будь-які дії при натисканні на кнопці. Кнопка XML визначається елементом <Button>. Атрибут android:text вказує, який текст має відображатись на кнопці:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
```

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


Щоб кнопка реагувала на клацання, включіть у XML макеті атрибут android:onClick і надайте йому ім'я викликаного методу з коду активності:

```
android:onClick="onButtonClicked"
```

Потім активністю визначається метод наступного виду:

```
/** Викликається при натисканні на кнопці */
public void onButtonClicked(View view) {
// Зробити щось по клацанню на кнопці
```

*Двопозиційна кнопка*. Клацнувши на двопозиційній кнопці, користувач вибирає одне із двох станів: off або on.

Двопозиційна кнопка визначається в XML елементом <ToggleButton>. Атрибути android:textOn та android:textOff визначають текст, який має виводитися на двопозиційній кнопці в залежності від її стану:

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

Щоб двопозиційна кнопка реагувала на клацання, включить атрибут android:onClick в XML макеті. Надайте йому ім'я викликаного методу з коду активності:

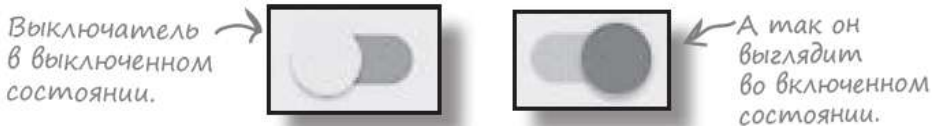
```
android:onClick="onToggleButtonClicked"
```

Потім в активності визначається метод наступного виду:

```
/** Вызывается при щелчке на двухпозиционной кнопке*/
public void onToggleClicked(View view) {
    // Получить состояние двухпозиционной кнопки.
    boolean on = ((ToggleButton) view).isChecked();
    if (on) {
        // Вкл
    } else {
        // Выкл
    }
}
```

Возвращает true, если двухпозиционная кнопка находится во включенном состоянии, или false, если она находится в выключенном состоянии.

*Вимикач* є важелем, який працює за тим же принципом, як і двопозиційна кнопка.




Вимикач визначається XML елементом <Switch>. Атрибути android:textOn та android:textOff вказують, який текст повинен відображатися залежно від стану вимикача:

```
<Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
```

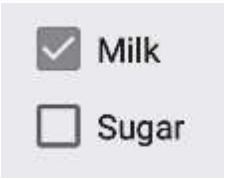
```
android:layout_height="wrap_content"
android:textOn="@string/on"
android:textOff="@string/off" />
```

Щоб вимикач реагував на клацання, включить атрибут android:onClick в XML макеті. Дайте йому ім'я викликаного методу з коду активності:

```
android:onClick="onSwitchClicked"
```

Потім активністю визначається метод наступного виду:

```
/** Викликається при натисканні на вимикачі. */
public void onSwitchClicked(View view) {
    // Включений стан?
    boolean on = ((Switch) view).isChecked();
    if (on) {
    // Увімк
    } else {
    // Вимк
    }
}
```




*Прапорці* (check boxes) надають користувачеві набір незалежних варіантів. Користувач може вибрати будь-які варіанти по своєму розсуду. Кожен прапорець може встановлюватися або зніматися незалежно від решти прапорців. Прапорець визначається XML елементом <CheckBox>. Атрибут android:text використовується для визначення тексту, що виводиться поряд з прапорцем:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk" />
<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar" />
```

Щоб перевірити, чи встановлено певний прапорець, використовуйте метод isChecked(). Якщо цей метод повертає true, значить прапорець встановлений:

```
CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox_milk);
boolean checked = checkbox.isChecked();
if (checked) {
//Дії для встановленого прапорця
}
```

Щоб обробляти клацання на прапорцях (за аналогією зі клацаннями на кнопках), увімкніть атрибут android:onClick в XML макеті і надайте йому ім'я методу, що викликається з коду активності:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk"
    android:onClick="onCheckboxClicked"/>
<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
android:text="@string/sugar"
android:onClick="onCheckboxClicked"/>
```

Потім в активності визначається метод наступного виду:

```
public void onCheckboxClicked(View view) {
    // Чи було встановлено прапорець, у якому клацнув користувач?
    boolean checked = ((CheckBox) view).isChecked();
    // Визначити, на якому прапорці було зроблено клацання
    switch(view.getId()) {
        case R.id.checkbox_milk:
            if (checked)
                // Кава з молоком
            else
                // Чорна кава
            break;
        case R.id.checkbox_sugar:
            if (checked)
            // З цукром
            else
                // Без цукру
            break;
```

*Перемикачі* (radio buttons) надають набір варіантів, з якого користувач може вибрати один варіант:

Почніть з визначення групи перемикачів - особливого різновиду групи уявлень - елементом <RadioGroup>. Усередині цього елемента окремі перемикачі визначаються елементами <RadioButton>:

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen" />
    <RadioButton android:id="@+id/radio_astronauts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/astronauts" />
</RadioGroup>
```

Щоб визначити, який перемикач у групі встановлено, використовуйте метод getCheckedRadioButtonId():

```
RadioGroup radioGroup = findViewById(R.id.radioGroup);
int id = radioGroup.getCheckedRadioButtonId();
if (id == -1) {
    //Жоден перемикач не встановлений
}
else {
    RadioButton radioButton = findViewById(id);
}
```

Щоб обробляти клацання на перемикачах, увімкніть атрибут android:onClick в XML макеті і надайте йому ім'я методу, що викликається з

коду активності:

```
<RadioGroup android:id="@+id/radio_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <RadioButton android:id="@+id/radio_cavemen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/cavemen"
            android:onClick="onRadioButtonClicked" />
        <RadioButton android:id="@+id/radio_astronauts"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/astronauts"
            android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

Потім в активності визначається метод наступного виду:

```
public void onRadioButtonClicked(View view) {
RadioGroup radioGroup = findViewById(R.id.radioGroup);
        int id = radioGroup.getCheckedRadioButtonId();
        switch(id) {
            case R.id.radio_cavemen:
                    // Встановлено перемикач Cavemen
                    break;
           case R.id.radio_astronauts:
                    // Встановлено перемикач Astronauts
                    break;
      }
}
```

*Список, що розкривається*, містить набір значень, з яких користувач може вибрати лише одне. Список, що розкривається, визначається в XML елементом Spinner>. Щоб додати в список, що розкривається, статичний масив елементів, використовуйте атрибут android:entries і надайте йому масив рядків

```
<Spinner
        android:id="@+id/spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/spinner_values" />
```

Масив рядків додається до файлу strings.xml так:

```
<string-array name="spinner_values">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
</string-array>
```

Щоб отримати значення поточного вибраного варіанта, використовуйте метод getSelectedItem() і перетворіть результат до типу String:

```
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        String string = String.valueOf(spinner.getSelectedItem());
```

*Графічне уявлення* використовується для виведення зображень. Для початку увімкніть файл зображення в проект як ресурс. Відкривши папку app/src/main/res у своєму проекті, ви побачите, що в ній знаходиться папка з drawable ім'я. Вона використовується за умовчанням для зберігання ресурсів зображення. Щоб додати файл із зображенням, просто перетягніть його до папки.

За бажанням ви можете використовувати різні файли зображень залежно від густини екрана пристрою. На екранах з високою щільністю пікселів будуть використовуватися зображення з більш високою роздільною здатністю, а на екранах з низькою щільністю пік селів зображення зі зниженою роздільною здатністю. Для цього створіть у app/src/main/res вкладені папки drawable для різних варіантів щільності пікселів. Ім'я папки відповідає щільності пікселів пристрою:

android-ldpi Екрани низької щільності (близько 120 dpi).
android-mdpi Екрани середньої густини (близько 160 dpi).
android-hdpi Екрани високої густини (близько 240 dpi).
android-xhdpi Екрани надвисокої густини (близько 320 dpi).
android-xxhdpi Екрани надвисокої щільності (близько 480 dpi).
android-xxxhdpi Екрани над-над-надвисокої щільності (близько 640 dpi).

Потім розмістіть зображення з різними дозволами у папках drawable*; простежте за тим, щоб файлам з різними дозволами надавалися імена, що збігаються. Android вибирає використовуване зображення на стадії виконання, залежно від густини пристрої, на якому працює програма. Наприклад, якщо пристрій оснащений екраном надвисокої щільності, система використовуватиме графіку з папки drawable-xhdpi. Якщо зображення додано лише до однієї з папок, Android використовує один файл на всіх пристроях. Зазвичай для цього використовується папка drawable.

Графічне подання визначається XML елементом <ImageView>. Атрибут android:src вказує, яке зображення має виводитись. Атрибут android:contentDescription дозволяє додати текстовий опис зображення, щоб зробити програму більш доступною:

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

Значення атрибуту android:src задається у формі @drawable/ім'я_зображення, де ім'я_зображення - ім'я файлу зображення (без розширення). Ресурси зображень мають префікс @drawable. Префікс @drawable повідомляє Android, що ресурс зображення зберігається у одній чи кількох папках drawable.

```
ImageView photo = (ImageView)findViewById(R.id.photo);
int image = R.drawable.starbuzz_logo;
String description = "Це є logo";
photo.setImageResource(image);
photo.setContentDescription(description);
```

Вихідне зображення та його текстовий опис задаються в коді активності

методами setImageResource() та setContentDescription():

Цей фрагмент коду шукає ресурс зображення з ім'ям starbuzz_logo у папках drawable* та призначає його джерелом даних для графічного представлення з ідентифікатором photo. Для посилань на ресурс зображення у коді активності використовується синтаксис R.drawable.ім'я_зображення, де Ім'я_зображення - ім'я файлу зображення (без розширення).

Крім виведення зображень у графічних уявленнях, також можна виводити *зображення на кнопках*. Щоб вивести на кнопці текст, праворуч від якого знаходиться графічне зображення, використовуйте атрибут android:drawableRight та вкажіть потрібне зображення:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableRight="@drawable/android"
    android:text="@string/click_me" />
```

Щоб зображення розташовувалося ліворуч від тексту, скористайтесь атрибутом android:drawableLeft:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/android"
    android:text="@string/click_me" />
```

Для того, щоб зображення було під текстом, скористайтеся атрибутом android:drawableBottom:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableBottom="@drawable/android"
    android:text="@string/click_me" />
```

При встановленні атрибуту android:drawableBottom зображення виводиться над текстом:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/android"
    android:text="@string/click_me" />
```

*Повідомлення* виконують суто інформаційні функції, Користувач не може з ними взаємодіяти. Поки повідомлення знаходиться на екрані, активність залишається видимою та доступною для взаємодії з користувачем. Повідомлення автоматично закривається після закінчення тайм-ауту. Повідомлення створюються лише у коді активності; визначити їх у макеті неможливо. Щоб створити сповіщення, викличте метод Toast.makeText() і передайте йому три параметри: Context (зазвичай для поточної активності), CharSequence (виводиться повідомлення) та int (тривалість). Після того як об'єкт повідомлення буде створено, його можна вивести на екран викликом методу show(). Приклад коду зі створенням повідомлення, що ненадовго з'являється на екрані:

CharSequence text = "**Hello, I'm a Toast**!"; За замовчуванням повідомлення

відображаються в нижній частин екрану.

```
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(this, text, duration);
toast.show();
```

## 3 Спискові уявлення та адаптери

Ми розглянули основні структурні елементи, які використовуються при побудові додатків; тепер настав час привести знання в порядок. У цьому питанні ми покажемо, як взяти розрізнені ідеї та перетворити їх на класний додаток. Ми покажемо, як списки даних можуть стати основою структури вашої програми та що зв'язування списків дозволяє створювати потужні та зручні програми. Принагідно ви в загальних рисах дізнаєтеся, як за допомогою слухачів подій та адаптерів зробити ваш додаток більш динамічним.

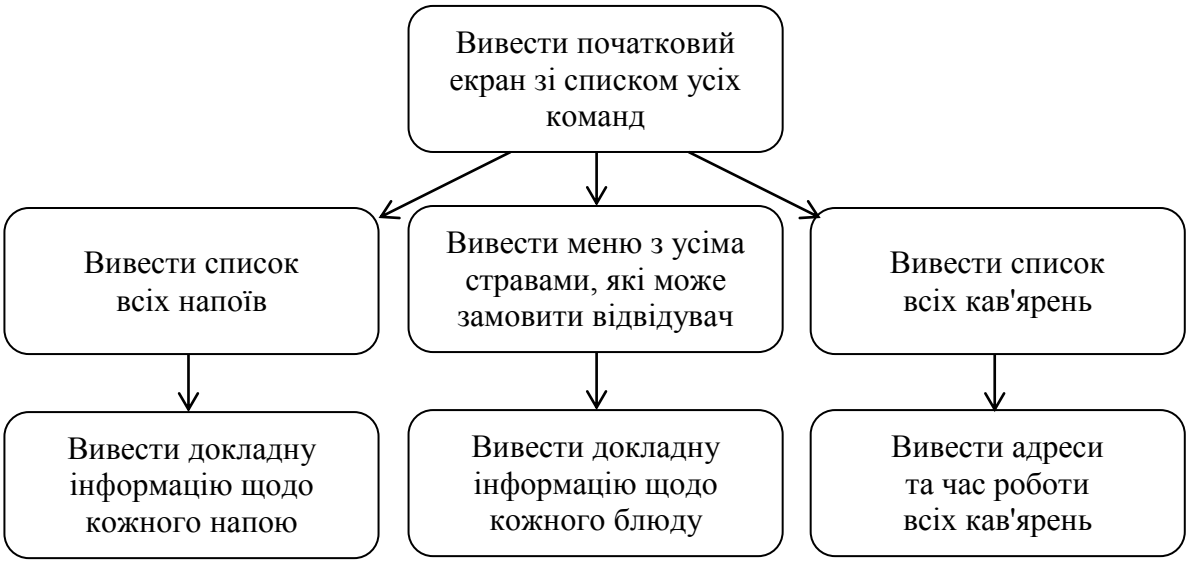
Корисний спосіб упорядкування таких ідей полягає в їхній класифікації на три типи активностей: активності верхнього рівня, активності категорій та активності деталізації/редагування. Припустимо, керівництво мережі кав'ярень Starbuzz хоче створити новий додаток, який приверне до їхніх закладів більше народу. Ось лише деякі можливості, які, як вони вважають, мають бути реалізовані у новому додатку:

*Активності верхнього рівня* є операціями, найбільш важливими для користувача, і надають прості засоби для навігації до них. У більшості програм перша активність, яку бачить користувач є активністю верхнього рівня.

*Активності категорій* виводять дані, що належать до конкретної категорії, - часто у вигляді список. Такі активності часто допомагають користувачеві перейти до активності деталізації/редагування. Приклад активності категорії виведення списку всіх напоїв, що є у Starbuzz.

*Активності деталізації/редагування* виводять докладну інформацію щодо конкретної записи, надають користувачеві можливість редагування існуючих записів або введення нових записів. Приклад активності деталізації/редагування - активність, яка виводить детальну інформацію щодо конкретного напою.

Після того, як активності будуть поділені на категорії, класифікація використовується для побудови ієрархії, описує переходи між активностями. Після того, як ви розділите свої ідеї на активності верхнього рівня, категорій та деталізації/редагування, ця класифікація може використовуватися для планування навігації за додатком. Як правило, перехід від активностей верхній рівень до активностей деталізації/редагування повинен здійснюватися через активність категорій.




У додатках із такою структурою необхідно організувати навігацію, тобто переходи між активностями. У таких ситуаціях найчастіше застосовуються спискові уявлення. Спискові уявлення відображають перелік об'єктів даних, які потім використовуються для навігації за програмою. Наприклад, на попередній сторінці було зазначено, що нам знадобиться активність категорії для виведення списку напоїв у кав'ярнях Starbuzz. Ця активність може виглядати як компонент ListView зі списком напоїв.

Активність використовує спискову виставу для виведення всіх напоїв, що продаються у кав'ярнях Starbuzz. Щоб перейти до конкретного напою, користувач клацає на відповідному рядку і на екрані з'являється докладний опис напою. Якщо клацнути у рядку Latte списку ListView, ви побачите докладний опис цього напою.

Побудуємо активність верхнього рівня, яку бачитиме користувач при запуску програми; активність категорії, яка виводить перелік напоїв; та активність деталізації/редагування, яка виводить докладну інформацію про один напій.

При запуску програми користувач бачить активність верхнього рівня - головну точку входу програми. Ця активність включає зображення логотипу Starbuzz та навігаційний список з командами для отримання інформації про напої, їжу та кав'ярні. Коли користувач клацає на одному з пунктів списку, програма використовує його вибір для переходу до іншої активності. Наприклад, якщо користувач клацнув на команді Drinks, додаток запускає активність категорій із списком напоїв. Ця активність виводить список всіх напоїв, що продаються в кав'ярнях Starbuzz. Користувач вибирає один з напоїв, щоб отримати докладнішу інформацію про нього.

Тобто користувач переходить від активності верхнього рівня до активності з детальною інформацією про напій, клацаючи на команді "Drinks" в активності верхнього рівня. Після цього він вибирає конкретний напій у списку.

Програма складається з трьох активностей. TopLevelActivity – активність верхнього рівня – забезпечує основну навігацію по розділах програми. DrinkCategoryActivity – активність категорії зі списком напоїв. Третя активність, DrinkActivity, містить докладну інформацію щодо конкретного напою.

У цій версії дані напоїв зберігатимуться у класі Java. В одному з наступних розділів інформація буде перенесена до бази даних, але ми хочемо зосередитися на побудові програми, не відволікаючись на підтримку баз даних.

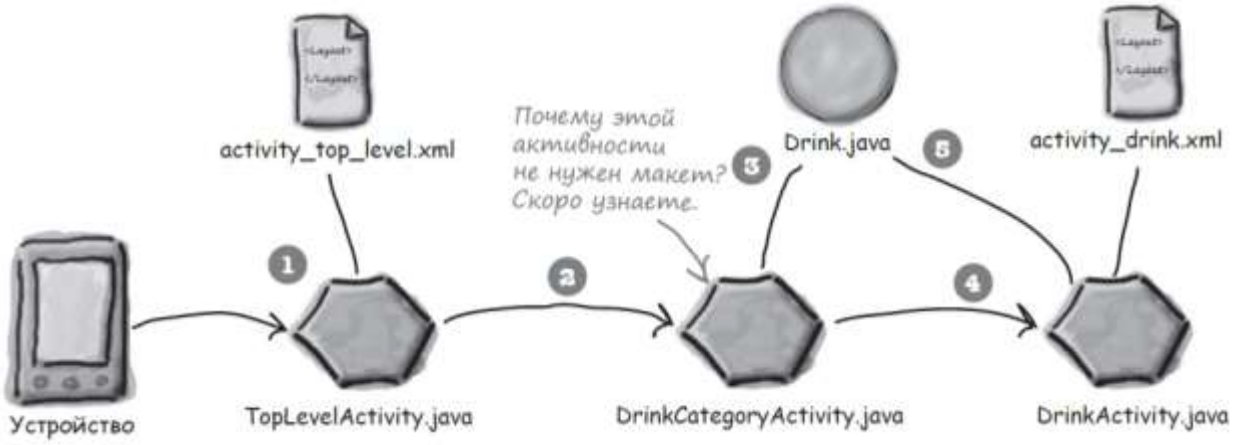
1. Під час запуску програми відкривається активність TopLevelActivity. Активність використовує макет activity_top_level.xml. Активність виводить список команд для переходу до розділів напоїв, блюд та кав'ярень.

2. Користувач вибирає команду Drinks у TopLevelActivity. Команда відкриває активність DrinkCategoryActivity,яка відображає список напоїв.

3. Інформація про напої зберігається у файлі класу Drink.java. DrinkCategoryActivity отримує дані напоїв із цього класу.

4. Користувач вибирає напій у DrinkCategoryActivity. При виборі запускається активність DrinkActivity, яка використовує макет activity_drink.xml.

5. DrinkActivity отримує детальну інформацію про напій із файлу класу Drink.java.




Нижче наведено основні етапи побудови програми:

1. Додавання класу Drink та ресурсів зображень. Клас містить докладну інформацію про напої; також у додатку використовуються ресурси зображень напоїв та логотипу Starbuzz.

2. Створення активності TopLevelActivity та її макета. "Точка входу" програми: активність повинна відображати Starbuzz логотип і навігаційний список команд. При виборі команди Drink активність TopLevelActivity повинна

відкривати DrinkCategoryActivity.

3. Створення DrinkCategoryActivity. Активність DrinkCategoryActivity містить перелік усіх наявних напоїв. При виборі напою має відкриватися активність DrinkCategory.

4. Створення активності DrinkActivity та її макет. Активність DrinkActivity виводить інформацію про напій, вибраний користувачем у списку DrinkCategoryActivity.

Створіть новий проект Android для програми з ім'ям "Starbuzz" та ім'ям пакету com.hfad.starbuzz. Виберіть мінімальний рівень SDK дорівнює API 15. Додаток має містити активність під назвою “TopLevelActivity” та макет з ім'ям “activity_top_level”.

Для початку додамо додаток клас Drink. Drink.java – звичайний файл класу Java, з якого активності отримуватимуть дані напоїв. Клас визначає масив із трьох об'єктів, що становлять напої; кожен об'єкт складається з назви, опису та ідентифікатора ресурсу зображення. Додайте клас до пакету com.hfad.starbuzz у папці app/src/main/java вашого проекту, надавши йому ім'я Drink.

```
package com.hfad.starbuzz;

public class Drink {
    private String name;
    private String description;
    private int imageResourceId;

    //drinks — массив с элементами Drink
    public static final Drink[] drinks = {
        new Drink("Latte", "A couple of espresso shots with steamed milk",
                R.drawable.latte),
        new Drink("Cappuccino", "Espresso, hot milk, and a steamed milk foam",
                R.drawable.cappuccino),
        new Drink("Filter", "Highest quality beans roasted and brewed fresh",
                R.drawable.filter)
    };

    //Для каждого напитка хранится имя, описание и ресурс изображения
    private Drink(String name, String description, int imageResourceId) {
        this.name = name;
        this.description = description;
        this.imageResourceId = imageResourceId;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }

    public String toString() {
        return this.name;
    }
}
```

Каждый объект Drink состоит из полей имени, описания и идентификатора ресурса изображения. Идентификаторы ресурсов принадлежат изображениям напитков, которые будут добавлены в проект на следующей странице.

drinks — массив из трех объектов Drink.

Изображения напитков. Мы добавим их на следующем этапе.

Конструктор Drink.

Get-методы для приватных переменных.

В качестве строкового представления Drink используется название напитка.


Код Drink включає три ресурси зображень напоїв із ідентифікаторами

R.drawable.latte, R.drawable.cappuccino та R.drawable.filter. Вони потрібні для того, щоб користувач міг побачити фотографію напою.

R.drawable.latte посилається на файл зображення з ім'ям latte, R.drawable.cappuccino - на файл зображення з ім'ям cappuccino, а R.drawable.filter - на файл зображення з ім'ям filter.

Ці файли зображень необхідно додати до проекту разом із зображенням логотипу Starbuzz, щоб його можна було використати в активності верхнього рівня. Для цього завантажте файли starbuzz-logo.png, cappuccino.png, filter.png та latte.png за адресою https://tinyurl.com/HeadFirstAndroid та перетягніть їх у папку app/src/main/res/drawable у проекті Starbuzz.

При додаванні зображень до програми необхідно вирішити, чи збираєтеся використовувати різні зображення для екранів з різною щільністю пікселів. В нашому прикладі одне зображення буде використовуватися для всіх екранів, тому достатньо помістити один екземпляр зображення в одній папці. Якщо ви у своєму додатку вирішіть використовувати різні версії графіки для різних екранів, розмістіть різні варіанти зображень у папках drawable*.

При збереженні зображень у проекті Android призначає їм ідентифікатори у форматі R.drawable.ім'я файлу. Наприклад, зображення з файлу latte.png надається ідентифікатор R.drawable.latte, відповідний значенню ресурсу зображення latte із класу Drink.

Після додавання класу Drink та ресурсів зображень у проект можна переходити до активності. Почнемо з активності верхнього рівня. Під час створення проекту активності за умовчанням було присвоєно ім'я TopLevelActivity.java, а її макет - ім'я activity_top_level.xml. Макет необхідно змінити так, щоб у ньому виводилися зображення та список.


У нашому прикладі знадобиться графічна вистава для логотипу Starbuzz, тому ми створимо уявлення, що використовує starbuzz_logo.png як джерело. Наступна розмітка визначає графічну виставу в макеті:

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

Размеры, которыми должно обладать изображение.

Источником графических данных является файл starbuzz_logo.png, который мы добавили в приложение.

Добавление описания улучшает доступность приложения.

При використанні графічного подання у додатку атрибут

android:contentDescription застосовується для додавання описи; це покращує доступність програми для користувачів з обмеженими можливостями. У нашому прикладі використовується рядок "@string/starbuzz_logo". Додайте її до файлу strings.xml:

```
<resources>
    ...
    <string name="starbuzz_logo">Starbuzz logo</string>
</resources>
```

Ось і все, що необхідно для увімкнення зображення у макет. Тепер можна перейти до списку.

Для додавання подання в вигляді списку до макету використовується елемент <ListView>. Щоб заповнити подання в вигляді списку, використовуйте атрибут android:entries і надайте йому масив рядків. Рядки з масиву будуть відображатися у подання в вигляді списку написів TextView. Приклад додавання до макету подання в вигляді списку, яке отримує значення з масиву рядків options:

```
<ListView
    android:id="@+id/list_options"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/options" />
```

Масив визначається так само, як це вже робилося раніше, - дані включаються до масиву strings.xml:

```
<resources>
    ...
    <string-array name="options">
        <item>Drinks</item>
        <item>Food</item>
        <item>Stores</item>
    </string-array>
</resources>
```

Уявлення в вигляді списку заповнюється трьома значеннями: Drinks, Food та Stores.



Так виглядає повна розмітка нашого макета (не забудьте внести зміни, виділені жирним шрифтом, у свій макет):

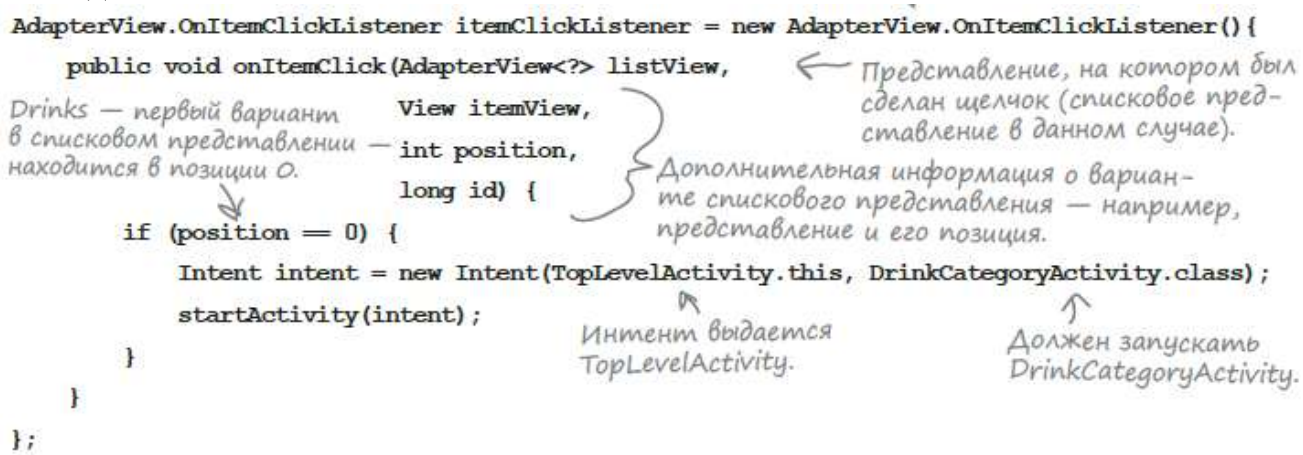
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".TopLevelActivity" >
    <ImageView
        android:layout_width="200dp"
```

```
            android:layout_height="100dp"
            android:src="@drawable/starbuzz_logo"
            android:contentDescription="@string/starbuzz_logo" />
        <ListView
            android:id="@+id/list_options"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:entries="@array/options" />
</LinearLayout>
```

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

Якщо ви бажаєте, щоб варіанти списку реагували на клацання, створіть об'єкт OnItemClickListener та реалізуйте його метод onItemClick(). Слухач OnItemClickListener відстежує клацання на варіантах списку, а метод onItemClick() визначає реакцію активності на клацання. За параметрами, що передаються методом onItemClick(), можна отримати додаткову інформацію про подію — наприклад, отримати посилання на варіант зі списку, дізнатися його позицію у поданні в вигляді списку (починаючи з 0) та ідентифікатор запису використовуваного набору даних.

У нашому прикладі при натисканні на першому варіанті поданні в вигляді списку - варіанті в позиції 0 - повинна запускатися активність DrinkCategoryActivity. Якщо клацання зроблено на варіанті позиції 0, необхідно створити інтент для запуску DrinkCategoryActivity. Код створення слухача виглядає так:



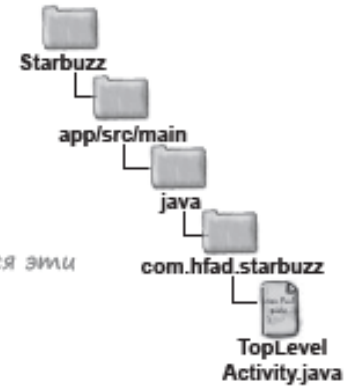

Після того, як об'єкт OnClickItemListener буде створений, його необхідно зв'язати зі поданням в вигляді списку. Це завдання вирішується за допомогою методу setOnItemClickListener() класу ListView. Метод отримує один аргумент - самого слухача:

```
AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener(){
    public void onItemClick(AdapterView<?> listView,
        ...
    }
};
```

```
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);
```

Додавання слухача до подання в вигляді списку вкрай важливо - саме ця операція забезпечує отримання слухачем сповіщень про те, що користувач натискає на подання в вигляді списку. Якщо цього не зробити, варіанти зі подання в вигляді списку не реагуватимуть на клацання. Отже, ви знаєте все необхідне для того, щоб навчити спискову виставку TopLevelActivity реагувати на клацання.

Перед вами повний код TopLevelActivity.java. Замініть код, згенерований майстром, тим, що наведено нижче, та збережіть зміни:




```
package com.hfad.starbuzz;

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

public class TopLevelActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        //Создать OnItemClickListener
        AdapterView.OnItemClickListener itemClickListener =
                                    new AdapterView.OnItemClickListener(){
            public void onItemClick(AdapterView<?> listView,
                                    View v,
                                    int position,
                                    long id) {
                if (position == 0) {
                    Intent intent = new Intent(TopLevelActivity.this,
                                               DrinkCategoryActivity.class);
                    startActivity(intent);
                }
            }
        };
        //Добавить слушателя к списковому представлению
        ListView listView = (ListView) findViewById(R.id.list_options);
        listView.setOnItemClickListener(itemClickListener);
    }
}
```

В коде используются эти внешние классы.

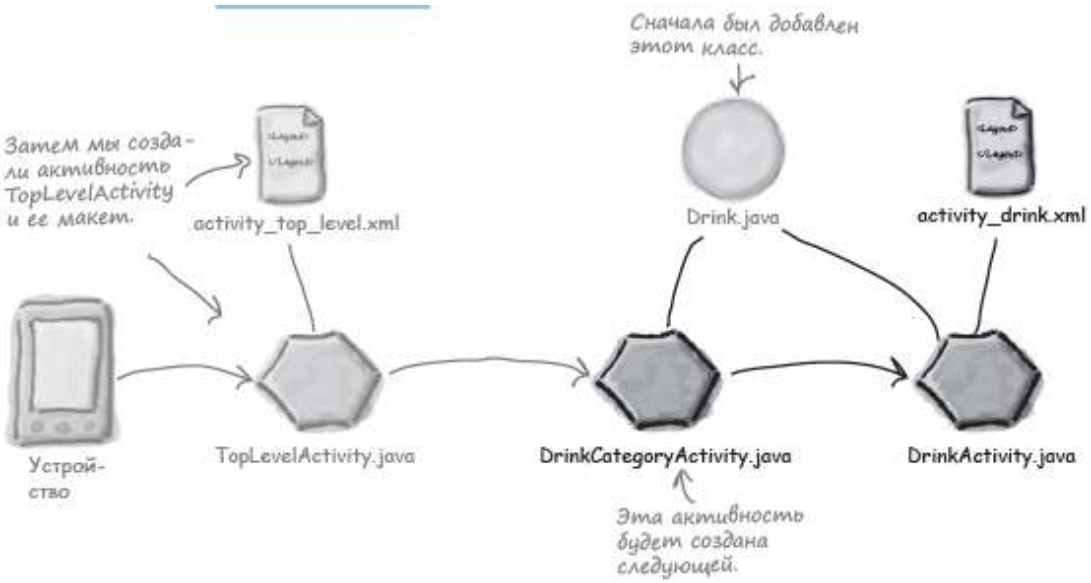
Создание слушателя.

Реализация его метода onItemClick().

Запустить DrinkCategoryActivity, если пользователь щелкнул на варианте Drinks. Даже если Android Studio скажет, что активность не существует, не беспокойтесь — сейчас мы ее создадим.

Добавление слушателя к списковому представлению.

Отже, ми додали додаток клас Drink.java, створили активність TopLevelActivity та її макет.




Наступне, що потрібно зробити, – створити активність DrinkCategoryActivity, яка має запускатися клацанням на команді Drinks активності TopLevelActivity.

Як згадувалося раніше, DrinkCategoryActivity є прикладом активності категорії. Такі активності призначені для виведення даних, що належать до певної категорії або розділу, часто у вигляді списку. Потім активність використовується для початку докладних описів окремих варіантів. У нашому додатку активність DrinkCategoryActivity використовується для виведення списку напоїв. Коли користувач вибирає один із напоїв у списку, на екрані з'являється докладна інформація про цей напій.

Для цього ми створимо активність, яка складається з єдиного списку подання висновку повного переліку напоїв. Так як активність містить лише один список без будь-яких інших компонентів графічного інтерфейсу ми створимо особливий різновид активностей: так звану списову активність.

Спискова активність спеціалізується на роботі зі списком. Вона автоматично зв'язується зі поданням в вигляді списку, тому вам не доведеться створювати таку виставу самостійно. Використання спискової активності для виведення категорій даних має пару переваг:

1) Вам не доведеться робити макет самостійно. Спискові активності визначають свій макет на програмному тому вам не доведеться створювати або займатися супроводом розмітки XML. Макет, що генерується списковою активністю, містить одне уявлення вигляді списку. Для звернення до уявленню в вигляді списку з коду активності використовується метод getListView() спискової активності. Таке звернення необхідне для того, щоб ви могли задати дані, які мають виводитися у поданні в вигляді списку.

2) Вам не потрібно реалізувати свого слухача. Клас ListActivity вже реалізує слухача подій, який виявляє клацання на варіантах уявлення в вигляді списку.

Замість того, щоб створювати власного слухача та прив'язувати його до подання в вигляді списку, розробнику достатньо реалізувати метод onListItemClick() спискової активності. При такому підході вам буде простіше

організувати реакцію активності на вибір варіантів подання в вигляді списку. Обробка клацань у дії буде продемонстрована пізніше, коли ми використовуємо метод onListItemClick() для запуску іншої активності.

Типовий спосіб застосування активностей категорій - виведення одного подання в вигляді списку для переходу до докладних описів, так що спискові активності добре підходять для такої ситуації.

Нижче наведено базовий код створення спискової активності. Як бачите, він дуже схожий на код створення нормальної активності. Скористайтеся майстром New Activity для створення нової активності в проекті з ім'ям DrinkCategoryActivity, після чого замініть вміст DrinkCategoryActivity.java наведеним нижче кодом:

```
package com.hfad.starbuzz;
import android.app.ListActivity;
import android.os.Bundle;
public class DrinkCategoryActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Наведений вище код створює базову списову активність під назвою DrinkCategoryActivity. Оскільки клас представляє саме списову активність, він має розширювати клас ListActivity замість класу Activity. Інша відмінність полягає в тому, що вам не потрібно призначати макет, використовуваний списковий активністю, викликом setContentView(). Справа в тому, що спискові активності визначають свої макети самостійно, тому вам це робити не доведеться - Спискова активність зробить все за вас. Спискові активності, як і звичайні, повинні бути зареєстровані у AndroidManifest.xml. Це необхідно для того, щоб вони могли використовуватись у додатку. При створенні активності Android Studio робить це для вас.

```
<application
    ... >
    <activity
        android:name=".TopLevelActivity"
        android:label="@string/app_name"
        ...
    </activity>
    <activity
        android:name=".DrinkCategoryActivity"
        android:label="@string/title_activity_drink_category" >
    </activity>
</application>
```

Після створення спискової активності її потрібно заповнити даними. Погляньмо, як це робиться.

При створенні першої активності TopLevelActivity ми могли зв'язати дані зі списковим поданням за допомогою атрибуту android:entries у макеті XML. Таке рішення працювало, оскільки дані зберігалися як ресурсу статичного масиву рядків. Масив був описаний у файлі strings.xml, що дозволяло легко

послатися на нього з використанням синтаксису

android:entries="@array/options"

де options – ім'я масиву рядків.

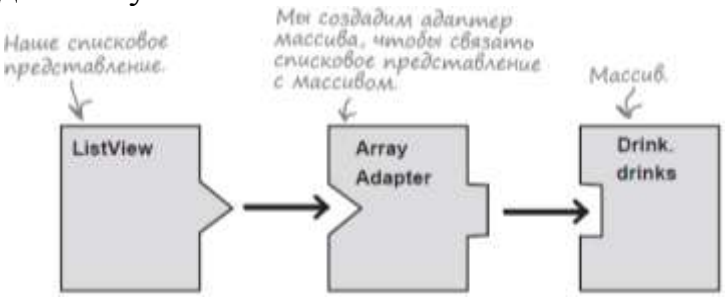
Атрибут android:entries підходить тільки для даних, представлених статичним масивом strings.xml. А якщо в програмі використовується інший спосіб зберігання? Що, якщо дані зберігаються у масиві, створеному на програмному рівні у коді Java, чи базі даних? В цьому випадку атрибут android:entries не працює.

Якщо подання в вигляді списку необхідно пов'язати з даними, що зберігаються в чомусь відмінному від ресурсу масиву рядків, доведеться діяти інакше: необхідно написати код активності для прив'язування даних. У нашому прикладі уявлення в вигляді списку потрібно пов'язати з масивом drinks із класу Drink.

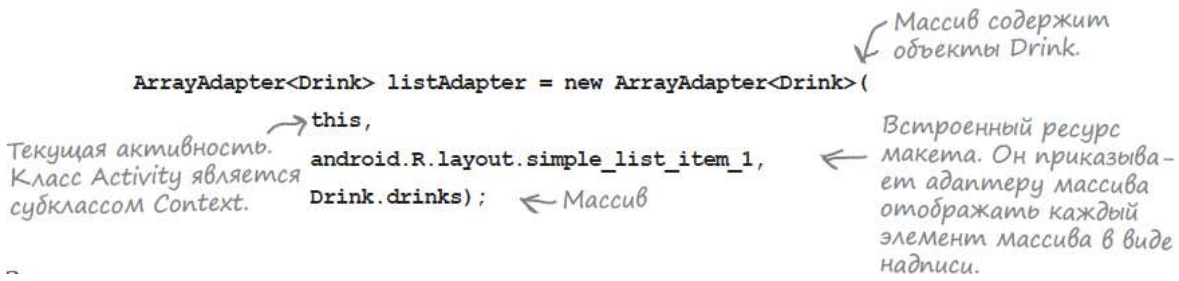
Якщо дані подання в вигляді списку повинні надходити з нестатичного джерела (наприклад, з масиву Java або бази даних), необхідно використовувати адаптер. Адаптер відіграє роль моста між джерелом даних та поданням в вигляді списку:


Існують різні типи адаптерів. Тепер ми займемося адаптерами масивів.

*Адаптер масиву* - різновид адаптерів для зв'язування масивів з уявленнями. Адаптер масиву може використовуватись з будь-яким субкласом класу AdapterView; це означає, що він буде працювати як зі поданням в вигляді списку, так і з списком, що розкривається. У нашому прикладі адаптер масиву використовуватиметься для виведення даних з масиву Drink.drinks у поданні в вигляді списку.




Щоб використовувати адаптер масиву, слід ініціалізувати його та приєднати до подання в вигляді списку. При ініціалізації адаптера масиву ви спочатку вказуєте тип даних масиву, який потрібно зв'язати зі поданням в вигляді списку. Потім адаптеру передаються три параметри: Context (зазвичай поточна активність), ресурс макета, який визначає, як повинен відображатися кожен елемент масиву, і сам масив. Наведений нижче код створює адаптер масиву для відображення даних із масиву Drink.drinks:




Потім адаптер масиву зв'язується зі поданням в вигляді списку за допомогою методу setAdapter() класу ListView:

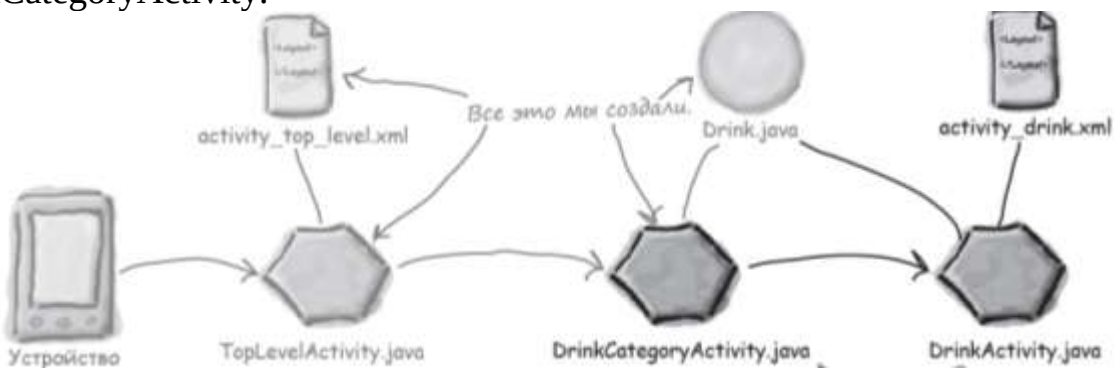
```
ListView listView = getListView();
listView.setAdapter(listAdapter);
```

У внутрішній реалізації адаптер масиву бере кожен елемент масиву, перетворює їх у String методом toString() і поміщає кожен результат на напис. Потім кожен напис виводиться в окремому рядку подання в вигляді списку.

Ми змінимо код DrinkCategoryActivity.java так, щоб списове подання використовувало адаптер масиву для отримання даних напоїв із класу Drink. Код буде включено до методу onCreate(), щоб подання в вигляді списку заповнювалося при створення активності. Нижче наведено повний код активності (внесіть зміни у свою версію коду та збережіть зміни):

```
package com.hfad.starbuzz;
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class DrinkCategoryActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
            this,
            android.R.layout.simple_list_item_1,
            Drink.drinks);
        listDrinks.setAdapter(listAdapter);
    }
}
```

Ось і все, що необхідно зробити для того, щоб у поданні в вигляді списку виводилися напої із класу Drink. На даний момент ми додали додаток клас Drink.java, а також створили активності TopLevelActivity та DrinkCategoryActivity.



Тепер потрібно зробити так, щоб активність DrinkCategoryActivity відкривала DrinkActivity і передавала їй інформацію про те, на якому напої було зроблено клацання.

Для цього ми створювали об'єкт OnItemClickListener, реалізовували метод onItemClick() і призначали його уявленню в вигляді списку:

```
AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener()
    public void onItemClick(AdapterView<?> listView,    <- Списковое представление.
                            View itemView,
                            int position,               Представление варианта, на котором был сделан щелчок, его позиция
                            long id) {                  в списке и идентификатор записи
        //Действия при щелчке на варианте               в используемых данных.
    }
};
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);  <- Назначить слушателя
                                                        для спискового представления.
```

Нам довелося призначати слухача подібним чином, тому що спискові уявлення спочатку не запрограмовані на обробку клацань (на відміну, скажімо, від кнопок). Як змусити DrinkCategoryActivity обробляти клацання?

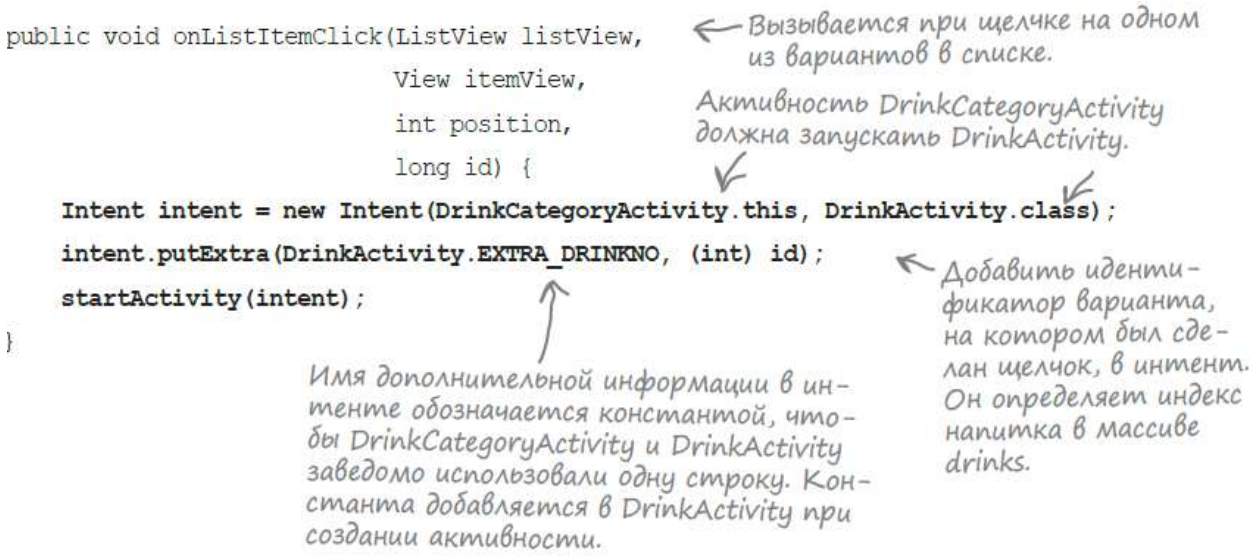
Між TopLevelActivity та DrinkCategoryActivity існує серйозна відмінність. TopLevelActivity є звичайним об'єктом Activity, тоді як DrinkCategoryActivity є ListActivity - особливий різновид активності, призначений для роботи зі уявленнями в вигляді списку.

Ця обставина відіграє у обробці клацань. Принципова відмінність між Activity та ListActivity полягає в тому, що клас ListActivity вже реалізує слухача подій клацання. Замість створення власного слухача подій, при використанні спискової активності достатньо реалізувати метод onListItemClick().

```
public void onListItemClick(ListView listView,
                            View itemView,
                            int position,
                            long id) {
    //Щось відбувається
}
```

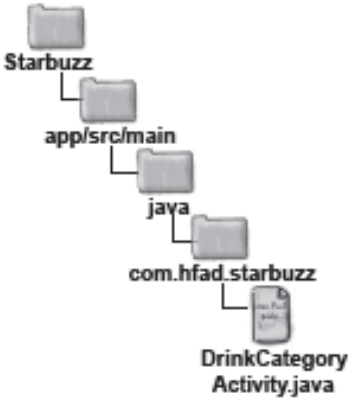
При використанні спискової активності для виведення категорій метод onListItemClick() зазвичай використовується для запуску іншої активності, яка виводить докладний опис варіанта, вибраного користувачем. І тому розробник створює інтент, відкриває другу активність. Ідентифікатор варіанта, вибраного користувачем, включається додаткову інформацію, щоб друга активність могла використовувати його під час запуску.

У нашому випадку потрібно запустити активність DrinkActivity та передати їй ідентифікатор вибраного напою. DrinkActivity використовує цю інформацію для докладного опису напою. Код виглядає так:




Передача ідентифікатора варіанта, на якому було зроблено клацання, - практика дуже поширена, оскільки значення, що передається одночасно є ідентифікатором у використовуваному наборі даних. Якщо набір даних зберігається в масиві, ідентифікатор збігається з індексом елемента масиву. Якщо інформація зберігається у базі даних, то ідентифікатор є індексом запису таблиці. При такому способі передачі ідентифікатора другої активності буде простіше отримати детальну інформацію про дані, а потім вивести її.

Ось і все, що необхідно зробити для того, щоб активність DrinkCategoryActivity запускала активність DrinkActivity і повідомляла, який напій був обраний. Повний код активності наведено на наступній сторінці.




```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.view.View;
import android.content.Intent;

public class DrinkCategoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
                this,
                android.R.layout.simple_list_item_1,
                Drink.drinks);
        listDrinks.setAdapter(listAdapter);
    }

    @Override
    public void onListItemClick(ListView listView,
                                View itemView,
                                int position,
                                long id) {
        Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
        intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int) id);
        startActivity(intent);
    }
}
```

*В коде используются эти внешние классы.*

*Метод onListItemClick() реализуется так, чтобы при щелчке на варианте спискового представления запускалась активность DrinkActivity.*

*Даже если Android Studio сообщит, что активность DrinkActivity не существует, не беспокойтесь — сейчас мы ее создадим.*

Як згадувалося раніше, DrinkActivity є активністю деталізації. Такі активності виводять докладну інформацію про конкретний запис, і зазвичай перехід до них здійснюється з активностей категорій. Ми скористаємося DrinkActivity для виведення докладної інформації про напій, вибраний користувачем. Клас Drink включає назву напою, його опис та ідентифікатор ресурсу зображення; всі ці дані виводитимуться в макеті. Для зображення напою буде створено графічне уявлення, а назви та описи — написи. Нижче наведено розмітку макета. Додайте до проекту нову активність з ім'ям DrinkActivity та макет з ім'ям activity_drink, після чого замініть вміст activity_drink.xml наступною розміткою:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.starbuzz.DrinkActivity" >

    <ImageView
        android:id="@+id/photo"
        android:layout_width="190dp"
        android:layout_height="190dp" />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```



Після того, як макет активності деталізації буде створено, можна переходити до заповнення його уявлень.

**4 Фрагменти**

Одна з найчудовіших особливостей програмування для Android - те, що один і той же додаток може запускатися на пристроях з різними екранами та процесорами і працюватиме ними однаково.

*Фрагменти* – реалізують механізм створення модульних програмних компонентів, які можуть повторно використовувати різні активності. Фрагменти – щось начебто компонентів, призначених для повторного використання або вторинних активностей. Фрагмент керує частиною екранного простору і може використовуватися на різних екрани. Це означає, що ми можемо створити різні фрагменти для списку комплексів вправ та для виведення докладного опису одного комплексу. Після цього створені фрагменти можна використовувати у різних активностях.

Фрагмент, як і активність, пов'язують із макетом. Якщо уважно підійти до його проектування, керувати всіма аспектами інтерфейсу може використовуватися код Java. Якщо код фрагмента містить все необхідне для керування його макетом, ймовірність того, що фрагмент можна буде повторно використовувати в інших частинах програми значно зростає. Процес створення та використання фрагментів буде продемонстровано на прикладі програми Workout.

Workout.java - звичайний файл класу Java, з якого програма отримує інформацію про комплекси вправ. Клас визначає масив із чотирьох елементів; кожен елемент містить назву та опис комплексу. Додайте клас у пакет com.hfad.workout у папці app/src/main/java вашого проекту, надайте йому ім'я

Workout та збережіть зміни.

```
package com.hfad.workout;

public class Workout {
    private String name;
    private String description;

    public static final Workout[] workouts = {
            new Workout("The Limb Loosener",
                    "5 Handstand push-ups\n10 1-legged squats\n15 Pull-ups"),
            new Workout("Core Agony",
                    "100 Pull-ups\n100 Push-ups\n100 Sit-ups\n100 Squats"),
            new Workout("The Wimp Special",
                    "5 Pull-ups\n10 Push-ups\n15 Squats"),
            new Workout("Strength and Length",
                    "500 meter run\n21 x 1.5 pood kettleball swing\n21 x pull-ups")
    };

    // В объекте Workout хранится имя и описание
    private Workout(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return this.name;
    }
}
```

Каждый объект Workout содержит поля названия (name) и описания (description) приложения.

workouts — массив из четырех объектов Workouts.

Get-методы для чтения приватных переменных.

В качестве строкового представления объекта Workout используется его имя.

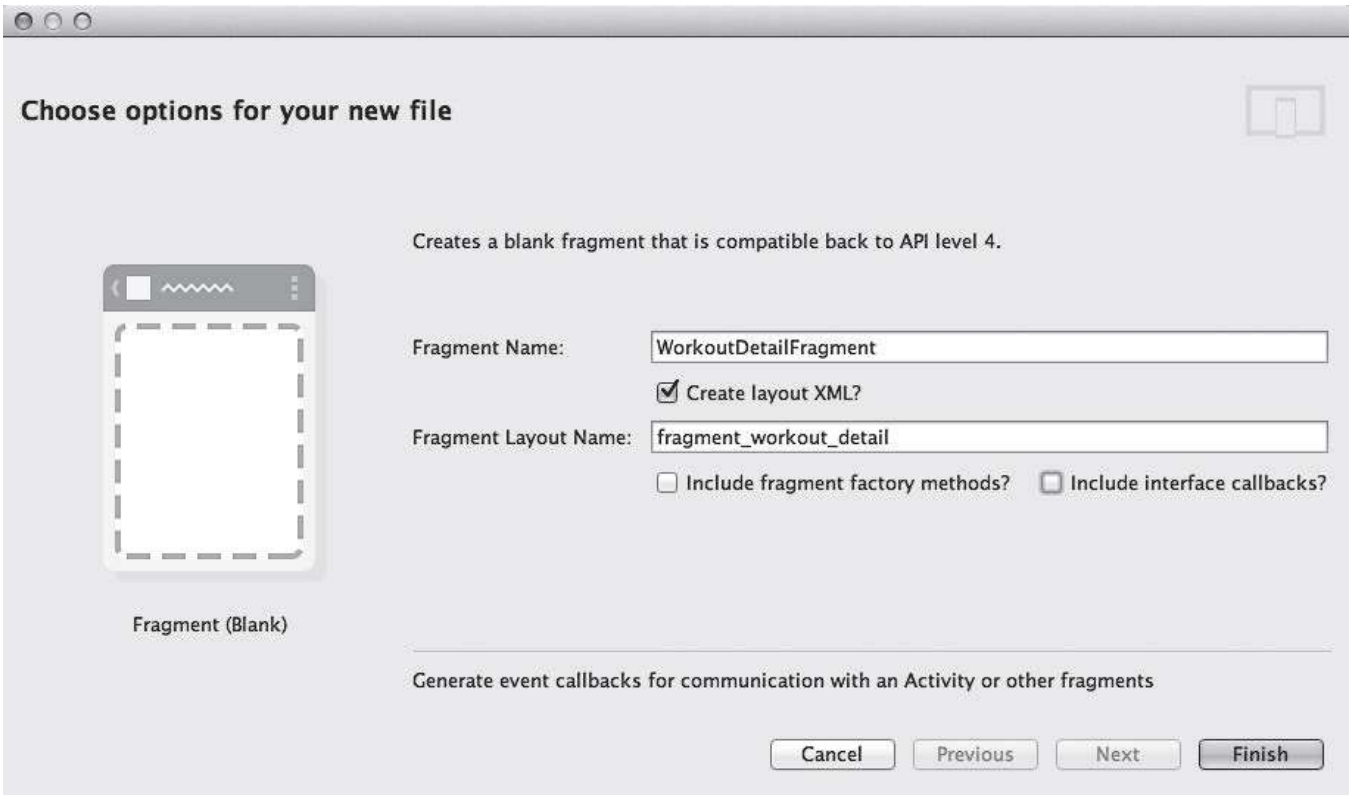

Зараз ми додамо до проекту новий фрагмент з ім'ям WorkoutDetailFragment, призначений для виведення детальної інформації про один комплекс вправ.

Нові фрагменти додаються приблизно так само, як і нові активності: в Android Studio виберіть команду

File→New...→Fragment→Fragment (Blank).

Вам буде запропоновано встановити параметри нового фрагмента. Надайте фрагменту ім'я “WorkoutDetailFragment”, встановіть прапорець створення XML розмітки та надайте макет фрагмента ім'я “fragment_workout_detail”. Зніміть прапорці включення фабричних методів фрагмента та інтерфейсних методів зворотного виклику; вони генерують додатковий код, який нам зараз не потрібний. Коли це буде зроблено, клацніть на кнопці Finish.

Якщо натиснути кнопку Finish, Android Studio створює новий файл фрагмента з ім'ям WorkoutDetailFragment.java у папці app/src/main/java та новий файл макета з ім'ям fragment_workout_detail.xml у папці app/src/res/layout.




Почнемо з оновлення розмітки макету фрагмента. Відкрийте файл fragment_workout_detail.xml з папки app/src/res/layout та замініть його на наступну розмітку:


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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/textDescription" />

</LinearLayout>
```

*Название и описание выводятся в двух разных компонентах TextView.*

*Название.*

*Описание комплекса упражнений.*


Як бачите, розмітка макета фрагмента майже не відрізняється від розмітки макету активності. Макет дуже простий і складається всього з двох написів: в одному (з великим текстом) виводиться назва комплексу вправ, а в іншому (з дрібним) текстом) виводиться опис. Створюючи макети фрагментів для своїх програм, ви можете використовувати в них все уявлення та макети, які вже використовувалися нами для створення макетів активностей.

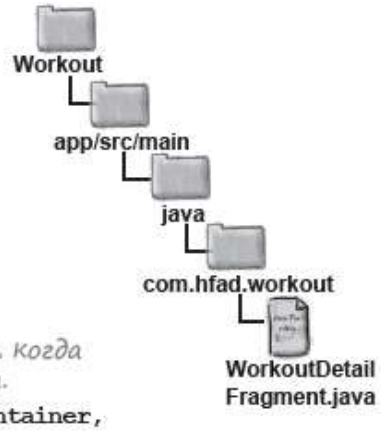
Код фрагменту зберігається у файлі WorkoutDetailFragment.java у папці app/src/main/java. Відкрийте цей файл. Як і слід очікувати, середовище Android Studio згенерувало код Java за вас. Замініть код, згенерований Android Studio, наступним кодом:

```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
}
```

Класс расширяет класс Fragment системы Android.

Метод onCreateView() вызывается тогда, когда Android потребуется макет фрагмента.

Сообщает Android, какой макет используется фрагментом (в данном случае fragment_workout_detail).




Макет фрагмента призначається викликом

inflater.inflate(R.layout.fragment_workout_detail,container, false);

Цей метод є аналогом методу setContentView() активностей у світі фрагментів. Як і setContentView(), він повідомляє, який макет має використовуватись фрагментом. Аргумент container передається активністю, яка використовує фрагмент. У ньому міститься об'єкт ViewGroup активності, який має бути вставлений макет фрагмента.

**Висновки.** Кожна програма за замовчуванням виконується в окремому процесі. Тільки головний програмний потік може оновлювати користувальницький інтерфейс. Об'єкти Handler використовуються для планування виконання чи передачі коду іншому потоку. При зміні конфігурації пристрою активність знищується та створюється заново. Активність успадковує методи життєвого циклу від класу Android Activity.

Усі компоненти графічного інтерфейсу є спеціалізаціями узагальненого уявлення. Всі вони є субкласами класу android.view.View. Усі макети є субкласами класу android.view.ViewGroup. Група уявлень - різновид уявлення, що може містити кілька уявлень. Файл з розміткою XML макета перетворюється на об'єкт ViewGroup, що містить ієрархічне дерево уявлень. У відносному макеті уявлення розміщуються щодо інших уявлень чи щодо батьківського макета. У лінійному макеті вистави розміщуються або по горизонталі, або по вертикалі. Напрямок задається атрибутом android:orientation. У табличному макеті екран ділиться на комірки, а розробник вказує, яку комірку (або комірки) займає кожне уявлення. Кількість стовпців задається атрибутом android:columnCount. Атрибути android:layout_row і android:layout_column визначають, в якому осередку має бути кожне уявлення, а атрибут android:layout_columnSpan - скільки стовпців воно має займати.

При проектуванні додатків потрібно розділяти свої ідеї щодо активностей програми на активності верхнього рівня, активності категорій та активності деталізації/редагування. Активності категорій використовуються для переходу від активностей верхнього рівня до активності деталізації/редагування.

Фрагмент використовується для керування частиною екрана. Фрагменти підходять для повторного використання у кількох активностях. Фрагмент зв'язується з макетом. Клас фрагмента розширює клас android.app.Fragment. Метод onCreateView() викликається щоразу, коли Android знадобиться макет фрагмента. Фрагменти додаються до макету активності за допомогою елемента <fragment> з додаванням атрибута class. Методи життєвого циклу фрагмента пов'язуються зі станами активності, що містить фрагмент.

**Практичне заняття № 5. Робота з базами даних**

**Кількість годин:** 2 год.

**Навчальна мета заняття**:

1. Придбання теоретичних знань з теми «Структура та компоненти мобільного додатку», розвиток здібностей до творчого мислення, формування навичок самостійної роботи з аналізу і узагальнення інформації, вміння проектувати компонентну архітектуру мобільного додатку.

**Рекомендована література:**


1. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.
2. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
3. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
4. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
5. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
6. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.
7. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
8. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.


**Матеріально-технічне забезпечення:** комп'ютерний клас.

**Навчальні питання:**

1. Структура бази даних SQLite
2. Визначення бази даних

3. Оновлення записів бази даних

4. Підключення до баз даних

5. Служби

1. ПОРЯДОК ПРОВЕДЕННЯ ЗАНЯТТЯ:

1.1. Проведення експрес-контролю готовності до заняття.

1.2. Ввести текст підготовленої програми і виконати її відлагодження.

1.3. Підібрати тести і виконати відпрацювання розробленого алгоритму на цих тестах.

1.4. Скласти звіт про виконану роботу і здати роботу викладачу.

**1 Структура бази даних SQLite**

В Android для довгострокового зберігання даних зазвичай використовується база даних SQLite в силу наступних причин:

1) Мінімальні витрати ресурсів.

Для роботи більшості систем управління базами даних потрібний спеціальний процес сервера бази даних. SQLite обходиться без сервера; база даних SQLite є традиційний файл. Коли база даних не використовується, вона не витрачає процесорний час. Це особливо важливо на мобільних пристроях, щоб уникнути розряджання акумулятора.

2) Оптимізація одного користувача. З базою даних взаємодіє тільки наша програма, тому можна обійтися без ідентифікації з ім'ям користувача та паролем.

3) Надійність та швидкість. Бази даних SQLite неймовірно надійні. Вони підтримують транзакції баз даних (іншими словами, якщо при оновленні кількох блоків даних щось піде не так, SQLite зможе повернутися до вихідного стану). Крім того, операції читання та запису даних реалізуються на оптимізованому коді C. Цей код не тільки швидко працює, а й скорочує обсяг необхідних обчислювальних ресурсів.

Додаток Starbuzz отримує інформацію про напої від класу Drink, що містить інформацію про напої з меню Starbuzz. Хоча таке рішення спрощує побудову першої версії програми, існують і більш досконалі способи зберігання та завантаження даних. Змінимо програму Starbuzz і доб'ємося, щоб дані завантажувалися з бази даних SQLite.

Android автоматично створює для кожної програми папку, в якій зберігаються бази даних цієї програми. Коли ми створюємо базу даних для програми Starbuzz, вона зберігатиметься в наступній папці:

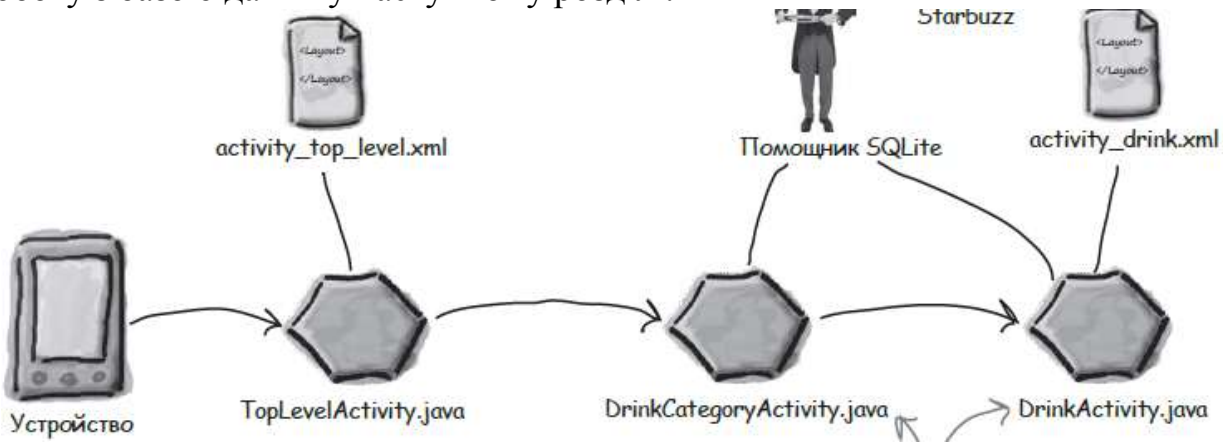
/data/data/com.hfad.starbuzz/databases

У цій папці програма може зберігати кілька баз даних. Кожна база даних складається із двох файлів. Ім'я першого – файлу бази даних – відповідає імені бази даних: наприклад, "starbuzz". Це основний файл баз даних SQLite; у ньому зберігаються всі дані. Другий файл – файл журналу. Його ім'я складається з імені бази даних та суфікса "-journal" - наприклад, "starbuzz-journal". У файлі журналу зберігається інформація про всі зміни, внесених до бази даних. Якщо у роботі з даними виникне проблема, Android використовує дані журналу для скасування останніх змін.

Система Android включає набір класів для керування базою даних SQLite. Основна частина роботи виконується трьома типами об'єктів.

1. Помічник SQLite створюється розширенням класу SQLiteOpenHelper. Він надає засоби для створення та управління базами даних.
2. Клас SQLiteDatabase надає доступ до бази даних. Його можна порівняти з класом SQLConnection у JDBC.
3. Клас Cursor призначений для читання та запису до бази даних. Його можна порівняти з класом ResultSet у JDBC.

Структура програми майже не змінюється, якщо не брати до уваги те, що файл Drink.java замінюється об'єктом помічник aSQLite і базою даних SQLite Starbuzz. Помічник SQLite керуватиме базою даних Starbuzz і забезпечуватиме доступ до неї з інших активностей. Ми займемося переведенням активностей на роботу з базою даних у наступному розділі.

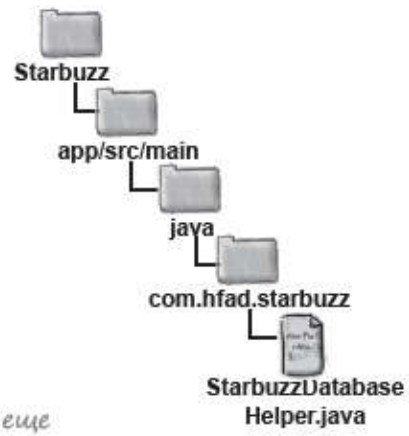



Щоб створити SQLite помічника, напишіть клас, який розширює SQLiteOpenHelper. При цьому ви повинні перевизначити методи onCreate() та onUpgrade(). Ці методи є обов'язковими. Метод onCreate() викликається під час першого створення бази даних на пристрої. Він повинен включати весь код, необхідний створення таблиць, які у додатку. Метод onUpgrade() викликається, коли база даних потребує в оновленні. Наприклад, якщо вам потрібно внести зміни до структури бази, що вже використовується, код оновлення слід розмістити саме у цьому методі. У нашому додатку буде використовуватися SQLite помічник з ім'ям StarbuzzDatabaseHelper. Створіть цей клас у своєму проекті Starbuzz: виділіть папку app/src/main/java/com.hfad.starbuzz на панелі структури проекту та виберіть команду File→New...→Java Class. Надайте класу ім'я “StarbuzzDatabaseHelper” та замініть його на наступний код:

```
package com.hfad.starbuzz;

import android.database.sqlite.SQLiteOpenHelper;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    StarbuzzDatabaseHelper(Context context) {
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
```

Помощники SQLite должны расширять класс SQLiteOpenHelper.

Присутствие методов onCreate() и onUpgrade() обязательно. Пока мы оставим эти методы пустыми, но еще вернемся к ним позднее в этой главе.




**2 Визначення бази даних**

Для створення бази даних помічнику SQLite потрібні два важливі параметри.

По-перше, необхідно задати ім'я бази даних. Надання імені гарантує, що база даних залишиться на пристрій після закриття. Якщо ім'я не задано, то база даних буде існувати тільки у пам'яті, і при закритті інформація пропаде.

По-друге, необхідно вказати версію бази даних. Номер версії являє собою ціле число, починаючи з 1. SQLite помічник використовує номер версії для визначення того, чи потребує база даних оновлення.

Ім'я та версія бази даних передаються конструктору суперкласу SQLiteOpenHelper. У нашому прикладі базі даних присвоюється ім'я "starbuzz", а оскільки це перша версія, їй надається номер версії 1. Код створення бази наведено нижче (замініть їм свою версію StarbuzzDatabaseHelper.java):

```
...
class StarbuzzDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "starbuzz"; // Ім'я бази даних
    private static final int DB_VERSION = 1; // Версія бази даних
    StarbuzzDatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    } Цей параметр використовується для роботи із курсорами.
}
```

Конструктор задає інформацію про базу даних, але сама база даних у цій точці не створюється. SQLite помічник очікує, поки програма звернеться до бази даних, і створює базу даних у цій точці. Після того, як помічник SQLite отримає інформацію про створювану базу даних, можна переходити до визначення таблиць.

Інформація в базах даних SQLite зберігається у таблицях. Таблиця складається з рядків, а рядки поділяються на стовпці. Один стовпець містить один елемент даних, наприклад, число або блок тексту. Ви повинні створити таблицю для всіх видів даних, які повинні зберігатися в базі. Скажімо, у

програмі Starbuzz необхідно створити таблицю для інформації про напої. Така таблиця виглядає приблизно так:

| _id | NAME | DESCRIPTION | IMAGE_RESOURCE_ID |
|---|---|---|---|
| 1 | "Latte" | "Espresso and steamed milk" | 54543543 |
| 2 | "Cappuccino" | "Espresso, hot milk and steamed-milk foam" | 654334453 |
| 3 | "Filter" | "Our best drip coffee" | 44324234 |

Деякі стовпці можуть бути призначені первинними ключами. Первинний ключ однозначно ідентифікує один рядок даних. Якщо певний стовпець є первинним ключем, база даних не дозволить створити рядки з однаковими значеннями цього стовпця.

Ми рекомендуємо створювати в таблицях один цілий стовпець первинного ключа з ім'ям _id. Це пов'язано з тим, що Android код жорстко запрограмований на використання числового стовпця з ім'ям _id, і його відсутність може створити проблеми.

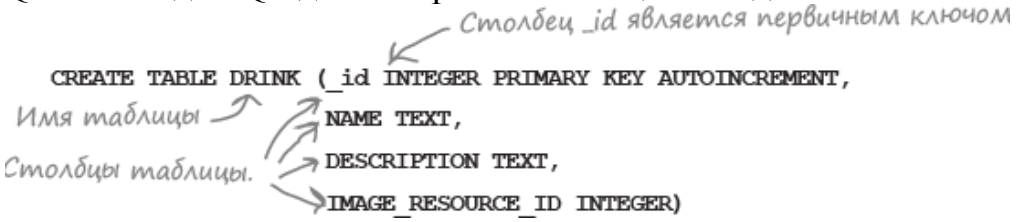
Кожен стовпець у таблиці розрахований зберігання даних певного типу. Наприклад, у нашій таблиці DRINK у стовпці DESCRIPTION можуть зберігатися лише текстові дані. Нижче наведено основні типи стовпців, які використовуються в SQLite, і дані, які можуть зберігатися в них:

INTEGER Будь-яке ціле число
TEXT Будь-які символьні дані
REAL Будь-яке речовинне число
NUMERIC Логічне значення, дата, дата-час
BLOB Двійкові великі об'єкти

На відміну від багатьох систем баз даних, SQLite не потрібно вказувати розмір стовпця. У внутрішній реалізації тип даних перетворюється на набагато більш універсальний клас зберігання. Це означає, що ви можете загалом вказати, які дані збираєтеся зберігати, але не повинні вказувати їх конкретний розмір.

На відміну від багатьох систем баз даних, SQLite не потрібно вказувати розмір стовпця. У внутрішній реалізації тип даних перетворюється на набагато більш універсальний клас зберігання. Це означає, що ви можете загалом вказати, які дані збираєтеся зберігати, але не повинні вказувати їх конкретний розмір.

Щоб створити таблицю DRINK, необхідно видати відповідну команду мовою SQL. Команда SQL для створення таблиці виглядає так:




Команда CREATE TABLE повідомляє, які стовпці повинні бути присутніми в таблиці, і дані якого типу мають у цих шпальтах зберігатися. Стовпець _id є первинним ключем таблиці, а спеціальне ключове слово

AUTOINCREMENT означає, що при занесенні до таблиці нового рядка SQLite автоматично згенерує для неї унікальний цілісний ідентифікатор.

Помічник SQLite відповідає за те, щоб база даних SQLite була створена в момент її першого використання. Спочатку у пристрої створюється порожня база даних, після чого викликається метод onCreate() помічника SQLite.

Під час виклику методу onCreate() передається об'єкт SQLiteDatabase. Ми можемо скористатися цим об'єктом для виконання у методі команди SQL:

```
SQLiteDatabase.execSQL(String sql);
```

Метод отримує один параметр – команду SQL, яку потрібно виконати. Повний код методу onCreate() виглядає так:

```
@Override
public void onCreate(SQLiteDatabase db){
    db.execSQL("CREATE TABLE DRINK ("
            + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "NAME TEXT, "
            + "DESCRIPTION TEXT, "
            + "IMAGE_RESOURCE_ID INTEGER);");
```

Клас SQLiteDatabase містить кілька методів для вставки, оновлення та видалення даних. Якщо вам потрібно заповнити таблицю SQLite даними, використовуйте метод insert() класу SQLiteDatabase. Цей метод вставляє дані до бази і повертає ідентифікатор запису. Якщо метод не зміг створити запис, він повертає значення –1. Щоб використовувати метод insert(), необхідно вказати таблицю та значення, що вставляються. Для визначення значень, що вставляються, створюється об'єкт ContentValues, в якому дані зберігаються у вигляді пар «ім'я/значення»:

```
ContentValues drinkValues = new ContentValues();
```

Для додавання пар «ім'я/значення» до об'єкта ContentValues використовується метод put() цього об'єкта. Щоб вставити рядок даних у таблицю DRINK, ми вказуємо імена стовпців у таблиці DRINK та значення, які зберігаються у кожному стовпці:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("NAME", "Latte");
drinkValues.put("DESCRIPTION", "Espresso and steamed milk");
drinkValues.put("IMAGE_RESOURCE_ID", R.drawable.latte);
```

Нарешті, виклик методу insert() об'єкта SQLiteDatabase вставляє значення таблицю DRINK:

```
db.insert("DRINK", null, drinkValues);
```

В результаті виконання цих рядків коду в таблицю DRINK буде вставлено наступний запис:

| _id | NAME | DESCRIPTION | IMAGE_RESOURCE_ID |
|---|---|---|---|
| 1 | "Latte" | "Espresso and steamed milk" | 54543543 |

Узагальнена форма методу insert() виглядає так:

```
db.insert(String table, String nullColumnHack, ContentValues values);
```

Рядкове значення nullColumnHack вказувати не обов'язково; найчастіше у цьому параметрі передається null, як у наведеному прикладі. Цей параметр є на той випадок, якщо об'єкт ContentValues порожній, і ви хочете вставити в таблицю порожній рядок. SQLite не дозволить вставити порожній рядок без

вказівки імені хоча б одного стовпця; параметр nullColumnHack дозволяє це зробити.

**3 Оновлення записів бази даних**

Для оновлення існуючої інформації SQLite використовується метод update() класу SQLiteDatabase. Цей метод вносить зміни до записів, що зберігаються в базі даних, і повертає кількість оновлених записів. Щоб використовувати метод update(), необхідно вказати таблицю, оновлювані значення та умови їхнього оновлення. Метод отримує такі параметри:

```
public int update (String table,
                   ContentValues values,
                   String whereClause,
                   String[] whereArgs)
```

У наступному прикладі значення стовпця DESCRIPTION замінюється рядком “Tasty”, якщо поле назви містить “Latte”:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("DESCRIPTION", "Tasty");
db.update("DRINK",
          drinkValues,
          "NAME = ?",
          new String[] {"Latte"});
```

Помещает значение "Tasty" в столбец DESCRIPTION.

Обновить столбец DESCRIPTION значением "Tasty" в строках таблицы DRINK, для которых NAME = "Latte".

| _id | NAME | DESCRIPTION | IMAGE_RESOURCE_ID |
|---|---|---|---|
| 1 | "Latte" | "Espresso and steamed milk"<br>"Tasty" | 54543543 |

Перший параметр методу update() містить ім'я таблиці, де оновлюється інформація (у разі таблиця DRINK).

Другий параметр вказує, які значення повинні використовуватися для оновлення. Як і у випадку з методом insert(), для визначення набору значень створюється об'єкт ContentValues для зберігання пар «ім'я/значення» даних:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("DESCRIPTION", "Tasty");
```

Третій параметр визначає умови відбору записів, що оновлюються. У наведеному прикладі "NAME = ?" означає, що стовпець NAME повинен дорівнювати деякому значенню. Символ? позначає значення стовпця, яке визначається вмістом останнього параметра (в даному випадку "Latte"). Якщо у двох останніх параметрах update() передається значення null, буде оновлено всі записи в таблиці.

Наприклад, виклик

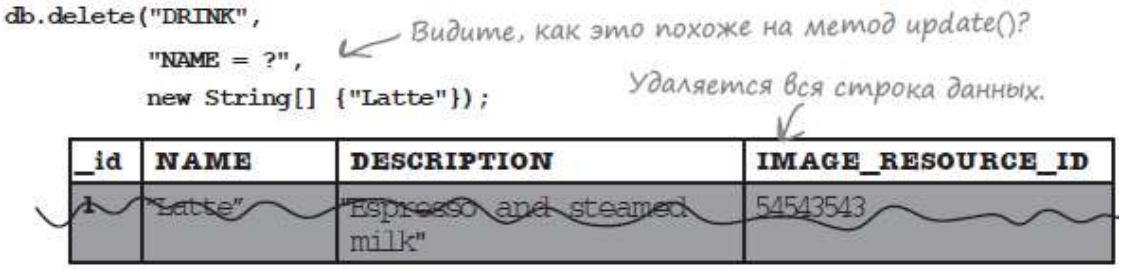
```
db.update("DRINK",
          drinkValues,
          null, null)
```

Метод delete() класу SQLiteDatabase працює за тим самим принципом, як і щойно розглянутий метод update(). Він має таку форму:

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

Наприклад, видалення з бази даних усіх записів, у яких стовпець назви

містить текст Latte, виконується таким чином:




У першому параметрі передається ім'я таблиці, з якої видаляються записи (у даному разі DRINK). Два інші параметри дозволяють точно описати, які записи потрібно видалити (NAME = “Latte”). Отже, ви знаєте, які операції можуть виконуватися під час роботи з даними у таблицях SQLite. Тепер вам відомо все необхідне для створення баз даних SQLite, створення таблиць та їх заповнення даними. На наступній сторінці ми застосуємо ці знання на практиці в коді, який використовує SQLite помічника.

Нижче наведено повний код StarbuzzDatabaseHelper.java (внесіть зміни до свого коду):

app/src/main
java
com.hfad.starbuzz
StarbuzzDatabaseHelper.java

```
package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // Имя базы данных
    private static final int DB_VERSION = 1; // Версия базы данных

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                   + "NAME TEXT, "
                   + "DESCRIPTION TEXT, "
                   + "IMAGE_RESOURCE_ID INTEGER);");
        insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
        insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
                    R.drawable.cappuccino);
        insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

    private static void insertDrink(SQLiteDatabase db, String name,
                                    String description, int resourceId) {
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("NAME", name);
        drinkValues.put("DESCRIPTION", description);
        drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
        db.insert("DRINK", null, drinkValues);
    }
```

Имя базы данных и номер версии.

Метод onCreate() вызывается при создании базы данных; мы используем его для создания таблицы и вставки данных.

Создать таблицу DRINK.

Вставить данные каждого напитка в отдельную строку.

Метод onUpgrade() вызывается тогда, когда возникает необходимость в обновлении базы данных. Сейчас мы познакомимся с ним поближе.

Так как нужно добавить несколько напитков, мы создаем отдельный метод.

При оновленні бази даних зазвичай виконуються два види дій:

Зміна записів бази даних. Раніше в цьому розділі було показано, як виконувати вставку, зміну та видалення записів у базі даних методами SQLiteDatabase insert(), update() та delete(). Оновлення бази даних може супроводжуватися додаванням нових записів, зміною або видаленням існуючих записів.

Зміна структури бази даних. На найближчих сторінках ви дізнаєтеся, як виконуються ці операції. Почнемо зі зміни структури бази даних та додавання стовпців до існуючих таблиць.

Мова SQL також може використовуватися для зміни існуючих таблиць - це завдання вирішується командою ALTER TABLE. Наприклад, команда додавання стовпця до таблиці виглядає так:

ALTER TABLE DRINK

```
ADD COLUMN FAVORITE NUMERIC
```

У цьому прикладі таблицю DRINK додається стовпець з ім'ям FAVORITE, в якому зберігаються числові значення. Команда ALTER TABLE може бути використана для перейменування таблиць. Наприклад, якщо ви захочете перейменувати таблицю DRINK на FOO, це робиться так:

```
ALTER TABLE DRINK
RENAME TO FOO
```

Крім створення та модифікації таблиць, також можливе їх видалення командою DROP TABLE:

```
DROP TABLE DRINK
```

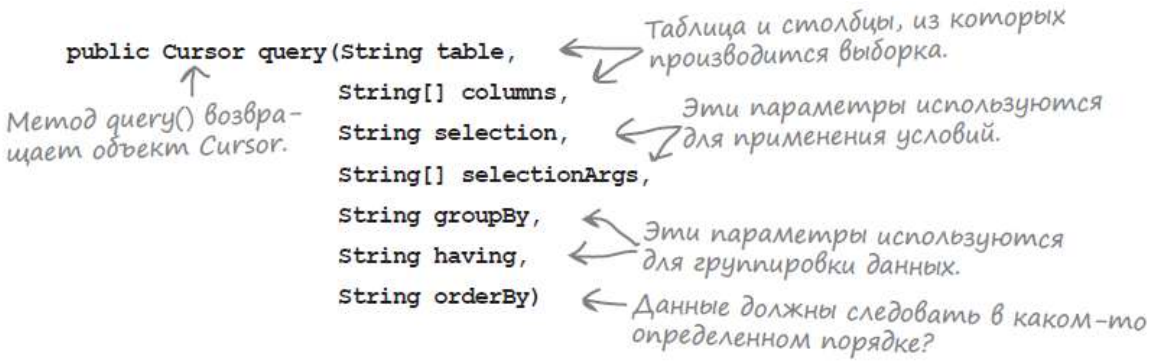
**4 Підключення до баз даних**

Поточний код DrinkActivity отримує інформацію про конкретний напій з Drink. Як змінити його, щоб інформація про напій завантажувалася з бази даних Starbuzz? Як змінити активність, щоб читати інформацію з бази даних? Завдання вирішується за допомогою курсору.

*Курсор* надає доступ до наборів записів бази даних. Ви вказуєте, які дані потрібні, а курсор повертає записи з бази даних. Потім ви перебираєте записи, надані курсором. Щоб створити курсор, ви описуєте цікаві для вас дані у вигляді запиту до бази даних. Запит до бази даних точно визначає набір записів, що вас цікавлять, з бази даних. Наприклад, ви можете вказати, що вам потрібні всі дані з таблиці DRINK або тільки напої, назви яких починаються з літери “L”. Чим жорсткіше обмежуються дані, що повертаються, тим ефективнішим буде запит.

Перше, що необхідно вказати у запиті, — з якої таблиці слід одержати записи та які стовпці вам потрібні. Після вказівки стовпців, що вас цікавлять, можна відфільтрувати результати вибірки; для цього необхідно оголосити умови, яким мають відповідати дані. Наприклад, у нашому додатку потрібно отримати напій, вибраний користувачем; для цього можна обмежити вибірку записами, у яких стовпець _id містить конкретне значення. Якщо запит має повернути кілька рядків даних, можливо, ви захочете вказати порядок, у якому слід виконувати записи. Наприклад, записи напоїв можуть бути впорядковані за назвою. Крім того, запити можуть використовуватися для групування даних за деяким критерієм та застосуванням агрегатних функцій. Наприклад, ви можете отримати кількість записів напоїв та вивести його у своїй програмі.

Для побудови запиту можна скористатися методом query() класу SQLiteDatabase. Метод query() повертає об'єкт типу Cursor, який може використовуватись вашими активностями для звернення до бази даних. Базова форма методу query() виглядає так:




Функції SQL у запитах використовуються для обчислень таких характеристик: AVG() - Середнє значення; COUNT() - Кількість рядків; SUM() – Сума; MAX() - Найбільше значення; MIN() - Найменше значення.

Скажімо, якщо ви хочете підрахувати, скільки напоїв зберігається в таблиці DRINK, скористайтесь функцією SQL COUNT() для підрахунку значень у стовпці _id:

```
Cursor cursor = db.query("DRINK",
                         new String[] {"COUNT(_id) AS count"},
                         null, null, null, null, null);
```

Запит повертає кількість напоїв у стовпці під назвою "count".

Додатку (програмі) потрібно зробити при відкритті бази даних наступі дії. Спочатку він має пройтися флеш-пам'яттю і знайти файл бази даних. Якщо файл бази даних не знайдено, потрібно створити порожню базу даних. Потім програма повинна виконати всі команди SQL для створення таблиць у базі даних та всіх вихідних даних, які йому потрібні. Нарешті, програма має видати запити для вилучення даних із бази.

Все це потребує часу. Для крихітної бази даних на зразок тієї, що використовується в програмі Starbuzz, витрати будуть невеликими. Але зі збільшенням бази даних час також неухильно зростає. Не встигнеш і оком моргнути, як додаток втрачає весь запал і починає працювати повільніше. Зі швидкістю створення бази даних і читання з неї взагалі нічого не поробиш, але запобігти уповільненню роботи інтерфейсу безперечно можливо.

Спільна робота потоків. Основна проблема зі зверненнями до повільної бази даних полягає в тому, що вона може уповільнити реакцію програми на дії користувача. Щоб зрозуміти, чому це відбувається, необхідно подумати над тим, як працюють програмні потоки в Android. Починаючи з версії Lollipop, існують три види потоків, які необхідно враховувати:

*Основний потік подій*. Цей потік - справжня "робоча конячка" Android: він прослуховує інтенти, отримує повідомлення про торкання від екрану і викликає всі методи всередині ваших активностей.

*Потік візуалізації*. Цей потік, з яким зазвичай не взаємодієте, читає список запитів на оновлення екрана, а потім видає команди низькорівневому графічному обладнанню на перемальовку екрана. Завдяки йому ваш додаток знаходить свою красу.

*Всі інші потоки, створені вами*. Якщо не вжити спеціальних заходів, програма виконує майже всю свою роботу в основному потоці подій. Чому?

Тому що основний потік подій виконує методи подій програми. Якщо просто включити код бази даних у метод onCreate() (як це було зроблено у додатку Starbuzz), то основний потік подій буде зайнятий роботою з базою даних замість того, щоб займатися обробкою подій від екрана або інших програм. Якщо виконання коду бази даних займає багато часу, у користувача може виникнути відчуття, що програма забула про його існування. Отже, фокус полягає в тому, щоб винести код бази даних із основного потоку подій та виконати його в окремому потоці у фоновому режимі.

Якщо програма працює з базою даних, цей код корисно винести у потік фону, а оновлювати уявлення даними з бази в основному потоці подій. Ми проаналізуємо код методу onFavoritesClicked() з коду DrinkActivity, щоб ви зрозуміли, як підходити до вирішення таких проблем. Нижче наведено код методу (він складається з кількох частин, які коротко описані нижче):

```
/Оновлення бази даних по клацанню на прапорці
public void onFavoriteClicked(View view){
```

```
int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
ContentValues drinkValues = new ContentValues();
drinkValues.put("FAVORITE", favorite.isChecked());
```

1

```
SQLiteOpenHelper starbuzzDatabaseHelper =
                            new StarbuzzDatabaseHelper(DrinkActivity.this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
    db.update("DRINK", drinkValues,
                            "_id = ?", new String[] {Integer.toString(drinkNo)});
    db.close();
} catch(SQLiteException e) {
```

2

```
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT
    toast.show();
```

3

1 *Код, який виконується до коду бази даних.* У кількох початкових рядках метод отримує поточне значення прапорця і зберігає його в об'єкті ContentValues під назвою drinkValues. Виконання цього коду має передувати виконання коду бази даних.

2 *Код бази даних, який має виконуватися у фоновому потоці.* Оновлення таблиці DRINK.

3 *Код, що виконується після коду бази даних*. Якщо база даних недоступна, слід вивести повідомлення для користувача. Цей код має виконуватися переважно в основному потоці подій.

Клас AsyncTask 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, клас навіть може використовуватися для публікації інформації про прогрес під час виконання завдання.

Щоб створити свою реалізацію AsyncTask, ви розширюєте клас AsyncTask та реалізуєте метод doInBackground(). Код цього методу виконується у фоновому потоці, тому цей метод ідеально підходить для розміщення коду бази даних. Клас AsyncTask також містить метод onPreExecute(), який виконується до doInBackground(), і метод onPostExecute(), який виконується після. Також є метод onProgressUpdate() на той випадок, якщо ви захочете передавати інформацію про хід виконання завдання. Все це виглядає приблизно так:

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>
    protected void onPreExecute() {
    //Код, що передує виконанню завдання
    }
    protected Result doInBackground(Params... params) {
    //Код, що виконується у фоновому потоці
    }
    protected void onProgressUpdate(Progress... values) {
    //Код, що передає інформацію про хід виконання завдання
    }
    protected void onPostExecute(Result result) {
    //Код, що виконується при завершенні завдання
```

AsyncTask визначається трьома узагальненими параметрами: Params, Progress та Results. Params - тип об'єкта, використовуваного передачі довільних параметрів завдання методу doInBackground(), Progress - тип об'єкта, використовуваний передачі інформації про прогрес завдання, і Result - тип результату завдання. Якщо будь-які з цих параметрів не використовуються, можна передати Void.

Зараз ми створимо спеціалізацію AsyncTask з ім'ям UpdateDrinkTask, яка використовуватиметься для фонового оновлення інформації про напої. Пізніше цей код буде додано до DrinkActivity.

Почнемо з методу onPreExecute(). Цей метод викликається до початку фонової задачі та використовується для підготовки її виконання. Метод викликається в основному потоці подій, тому для нього доступні всі уявлення в інтерфейсі користувача. Метод onPreExecute() викликається без параметрів та повертає void.

Ми використовуємо метод onPreExecute() для отримання значення прапорця улюбленого напою та включення його до об'єкту ContentValues з ім'ям drinkValues. Справа в тому, що для виконання цієї операції потрібен доступ до прапорця, а сама операція має бути виконана до виконання будь-якого коду бази даних. Для зберігання об'єкта ContentValues під назвою drinkValues використовується зовнішній атрибут, щоб до нього могли звертатися інші методи класу. Код виглядає так:

```
private class UpdateDrinkTask extends AsyncTask<Params, Progress, Result> {
    ContentValues drinkValues;
    protected void onPreExecute() {
        CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
        drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());
```

```
    }
    ...
}
```

Тепер розглянемо метод doInBackground().

Метод doInBackground() запускається у фоновому режимі відразу після виконання onPreExecute(). Ви визначаєте тип параметрів, які повинні передаватися задачі, і тип значення, що повертається. У нашому додатку метод doInBackground() використовуватиметься для коду роботи з базою даних, щоб він виконувався у фоновому потоці. Метод отримує ідентифікатор напою, інформацію про який потрібно оновити, а логічне значення, що повертається, дозволить перевірити, чи успішно було виконано завдання:

```
private class UpdateDrinkTask extends AsyncTask<Integer, Progress, Boolean> {

    ContentValues drinkValues;

    ...

    protected Boolean doInBackground(Integer... drinks) {
        int drinkNo = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                                new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                                "_id = ?", new String[] {Integer.toString(drinkNo)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
```

Заменяется на Integer в соответствии с параметром метода doInBackground().

Заменяется на Boolean в соответствии с возвращаемым типом метода doInBackground().

Этот код выполняется в фоновом потоке.

Это массив целых чисел, но мы включаем всего один элемент — идентификатор напитка.

Метод update() использует объект drinkValues, созданный методом onPreExecute().

Тепер перейдемо до розгляду методу onProgressUpdate().

Метод onProgressUpdate() викликається в основному потоці подій, тому в ньому доступні уявлення інтерфейсу користувача. Метод може бути використаний для виведення відомостей про хід виконання операції. Ви самі визначаєте тип параметрів, які мають передаватися методом. Метод onProgressUpdate() виконується під час виклику publishProgress() з методу doInBackground():

```
private class UpdateDrinkTask extends AsyncTask<Integer, Progress, Boolean> {

    ContentValues drinkValues;

    ...

    protected Boolean doInBackground(Integer... drinks) {
        int drinkNo = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                                new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                                "_id = ?", new String[] {Integer.toString(drinkNo)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
```

Заменяется на Integer в соответствии с параметром метода doInBackground().

Заменяется на Boolean в соответствии с возвращаемым типом метода doInBackground()

Этот код выполняется в фоновом потоке.

Это массив целых чисел, но мы включаем всего один элемент — идентификатор напитка.

Метод update() использует объект drinkValues, созданный методом onPreExecute().

Метод onProgressUpdate() викликається в основному потоці подій, тому в ньому доступні уявлення інтерфейсу користувача. Метод може бути використаний для виведення відомостей про хід виконання операції. Ви самі визначаєте тип параметрів, які мають передаватися методом. Метод onProgressUpdate() виконується під час виклику publishProgress() з методу doInBackground():

```
protected Boolean doInBackground(Integer... count) {
    for (int i = 0; i < count; i++) {
        publishProgress(i);
    }
}

protected void onProgressUpdate(Integer... progress) {
    setProgress(progress[0]);
}
```

Приводит к вызову метода onProgressUpdate() с передачей значения i.

Щоб запустити завдання виконання, викличте метод execute() об'єкта AsyncTask. Якщо метод doInBackground() отримує параметри, вони додаються до методу execute(). Наприклад, у нашому додатку методу doInBackground() завдання AsyncTask має передаватися напій, вибраний користувачем, тому виклик виглядає так:

```
int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
new UpdateDrinkTask().execute(drinkNo);
```

Тип параметра, який передається методу execute(), повинен відповідати типу параметра, який очікує отримати метод doInBackground() об'єкта AsyncTask. Наш метод doInBackground() отримує параметри типу Integer, тому потрібно передавати цілі числа:

```
protected Boolean doInBackground(Integer... drinks) {
...
}
```

Завдання UpdateDrinkTask буде запускатися з методу onFavoritesClicked() об'єкта DrinkActivity.

Ось і все, що потрібно знати для створення AsyncTask. Коли користувач клацає на прапорці коханого напої в DrinkActivity, інформація в базі даних оновлюється у фоновому режимі.В ідеалі весь код роботиьз базою даних повинен виконуватися у фоновому режимі.

**5 Служби**

Існують операції, які мають виконуватися постійно. Наприклад, якщо ви запустили відтворення музичного файлу в програмі-програвачі, ймовірно, музика не повинна зупинятися при переключення на іншу програму. Служба повідомлень допоможе вам тримати користувачів у курсі справ, а за допомогою служби позиціонування користувач зможе дізнатися, де він знаходиться.

Програма Android складається з активностей та інших компонентів. Основна частина коду забезпечує взаємодію з користувачем, але іноді програмі доводиться виконувати деякі операції у фоновому режимі: наприклад, завантажити великий файл, відтворити музичний файл або очікувати повідомлення від сервера.

Активності таких завдань не пристосовані. У простих випадках можна створити вторинний потік, але при найменшій неуважності код активності стає складним і незручним.

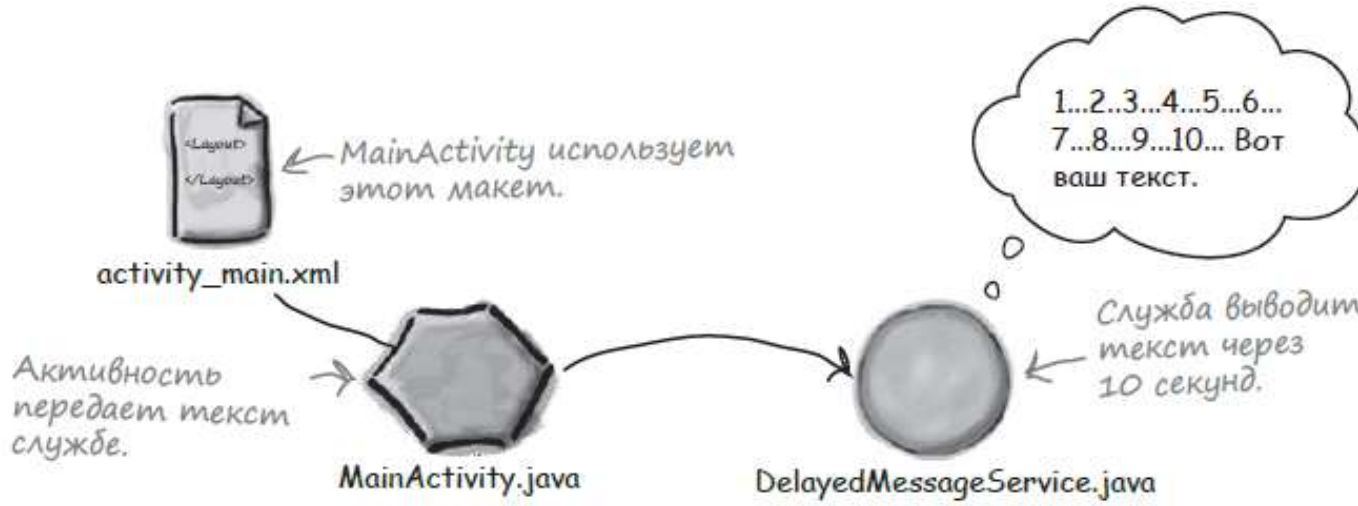
Для таких ситуацій було вигадано служби. Служба (service) є компонентом програми, схожий на активність, але не володіє інтерфейсом користувача. Служби мають більш простий життєвий цикл, ніж активності, а їхня вбудована функціональність спрощує написання коду, що виконується у фоновому режимі, поки користувач займається чимось іншим.

Служби діляться на два різновиди:

*Запускові служби* можуть виконуватися у фоновому режимі як завгодно довго, навіть після знищення активності, що запустила їх. Якщо ви хочете завантажити великий файл з Інтернету, ймовірно, цю операцію слід оформити у вигляді служби, що запускається. Після завершення операції така служба зупиняється.

*Пов'язана служба* прив'язується до іншого компонента, наприклад активності. Активність може взаємодіяти зі службою, надсилати їй запити та отримувати результати. Пов'язана служба працює, поки працюють пов'язані з нею компоненти. Коли зв'язок із компонентом переривається, служба знищується. Наприклад, якщо ви створюєте одометр для вимірювання відстані, пройденого машиною, можливо, слід використовувати пов'язану службу. У цьому випадку будь-які активності, пов'язані зі службою, зможуть звертатися до служби та запитувати пройдену відстань

Наш наступний проект містить активність з ім'ям MainActivity та службу з ім'ям DelayedMessageService. Щоразу, коли MainActivity викликає DelayedMessageService, служба чекає 10 секунд, а потім виводить текст.




Робота проходитиме у три етапи:

*Виведення повідомлення до журналу*. Вміст журналу можна переглянути в Android Studio.

*Виведення повідомлення у повідомленні Toast*. Повідомлення буде виводитися в тимчасовому повідомленні, щоб перевірка працездатності програми не вимагала підключення до Android Studio.

*Виведення повідомлення через службу сповіщень*. DelayedMessageService використовує вбудовану службу повідомлень Android для виведення повідомлення. Це дозволить користувачеві переглянути повідомлення пізніше.

Почнемо із створення проекту. Створіть новий проект Android для програми з ім'ям Joke і ім'ям пакета com.hfad.joke. Мінімальний рівень SDK повинен дорівнювати API 16, щоб програма працювала на більшості пристроїв. Щоб ваш код не відрізнявся від нашого, назвіть порожній активності ім'я "MainActivity", а макету - ім'я "activity_main". Потім потрібно створити службу.

Нова служба створюється розширенням або класу Service, або класу IntentService. Клас Service є базовим всім служб. Він надає основну функціональність служб; як правило, при створенні пов'язаних служб слід розширювати саме цей клас. Клас IntentService є субклас Service, призначений до роботи з інтентами. Зазвичай він розширюється для створення служб, що запускаються.

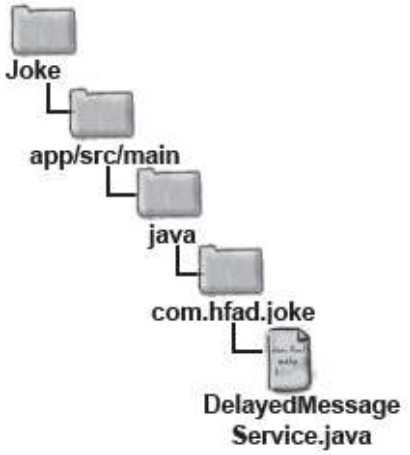
Так як ми збираємося створити службу, що запускається, в проект слід додати нову спеціалізацію IntentService. Виконайте команду File→New... та виберіть Service. Виберіть створення нової спеціалізації IntentService. Надайте службі ім'я DelayedMessageService, зніміть прапорець увімкнення допоміжного методу запуску (оскільки ми збираємося замінити код згенерований Android Studio).

Щоб створити службу за допомогою інтентів, слід розширити клас IntentService і реалізувати його метод наHandleIntent(). Цей метод повинен містити код, який має виконуватися під час виклику служби:

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;

public class DelayedMessageService extends IntentService {

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //...
    }
}
```

Расширить класс IntentService.

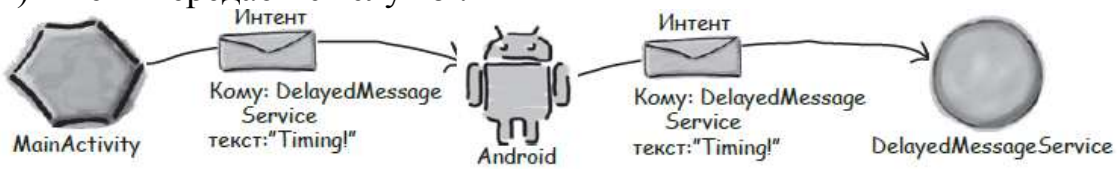
Поместите код, который должен выполняться службой, в метод onHandleIntent().




Загальна схема роботи служб, що запускаються, наведено нижче.

1) Активність показує, яка служба їй потрібна, створюючи явний інтент. Інтент визначає службу, на яку він призначається.

2) Інтент передається службі.




3) Служба запускається та обробляє інтент. Метод onHandleIntent() класу IntentService викликається та виконується в окремому потоці. Якщо служба отримує кілька інтентів, вона обробляє їх послідовно по одному за раз. Після завершення роботи служба зупиняється.

Як бачите, служба запускається тим самим способом, яким запускаються активності: створенням інтенту. Відмінність в тому, що при запуску служби на екрані нічого не змінюється, тому що служба не має інтерфейсу користувача.

Служба DelayedMessageService має вивести повідомлення у журналі Android. Перш ніж братися за зміну коду, необхідно розібратися, як зберігати повідомлення в журналі.


Запис повідомлень у журнал часто допомагає переконатися, що ваш код працює саме так, як треба. Ви повідомляєте Android, яку інформацію потрібно зберегти у своєму коді Java, а потім під час роботи програми переглядаєте результати в журналі Android. Для збереження повідомлень у журналі використовуються такі методи класу Android.util.Log:

Log.v (String tag, String message) Зберігає докладне повідомлення.

Log.d (String tag, String message) Зберігає налагоджувальне повідомлення.

Log.i(String tag, String message) Зберігає інформаційне повідомлення.

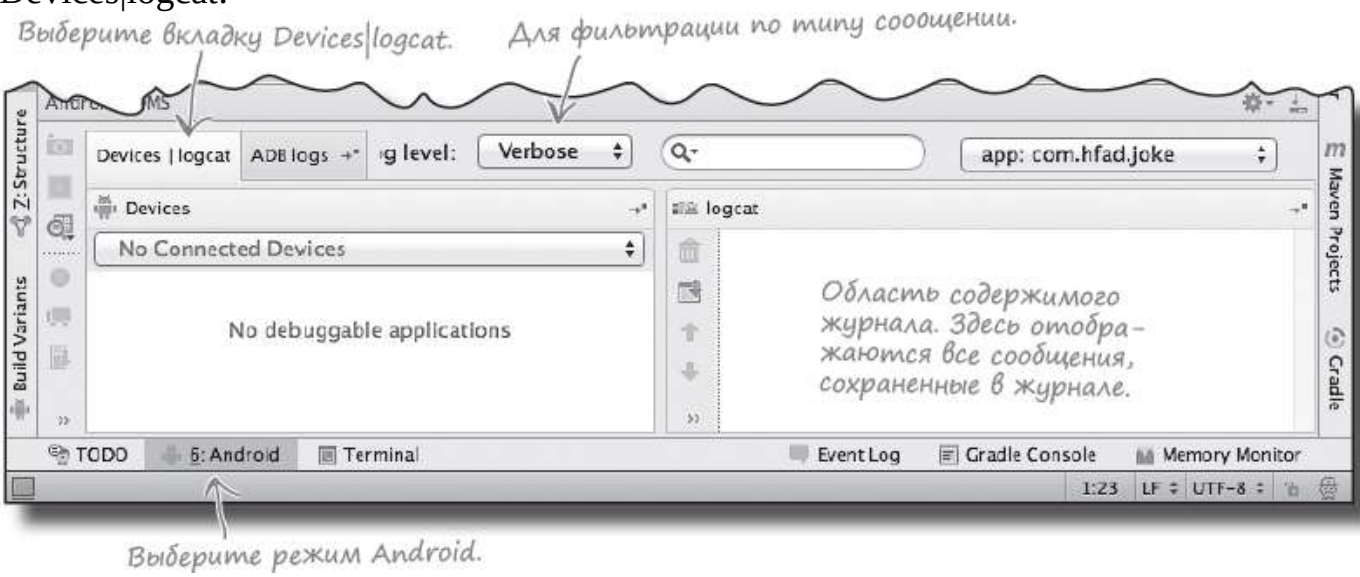
Log.w(String tag, String message) Зберігає попередження.

Log.e(String tag, String message) Зберігає повідомлення про помилку.

Кожне повідомлення складається з рядкової мітки, яка може використовуватися для ідентифікації джерела повідомлення та самого повідомлення. Наприклад, для збереження докладного повідомлення, що надійшло від служби DelayedMessageService, використовується виклик Log.v() такого вигляду:

```
Log.v("DelayedMessageService", "This is a message");
```

Android Studio надає засоби для перегляду журналу та фільтрації даних по типів повідомлень. Щоб переглянути вміст журналу, виберіть режим Android у нижній частині вікна проекту в Android Studio, а потім перейдіть на вкладку Devices|logcat:




Наша служба має отримати текст з інтенту, зачекати 10 секунд, а потім вивести текст до журналу. Ми створимо метод showText() для виведення тексту в журнал, а потім викличемо його з методу onHandleIntent() після закінчення затримки. Нижче наведено повний код DelayedMessageService.java (замініть їм код, згенерований Android Studio):

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);
    }
}
```

Расширить класс IntentService.

Использовать константу для передачи сообщения от активности службе.

Вызвать конструктор суперкласса.

Метод содержит код, который должен выполняться при получении интента службой.

Подождать 10 секунд.

Получить текст из интента.

Вызвать метод showText().

Вывести текст в журнал. В дальнейшем содержимое журнала можно просмотреть в Android Studio.


Служби, як і активності, повинні оголошуватись у файлі AndroidManifest.xml за допомогою елемента <service>. Це необхідно для того, щоб система Android могла викликати службу; якщо служба не оголошена в AndroidManifest.xml, то Android її викликати не зможе. Android Studio автоматично включає оголошення в AndroidManifest.xml під час створення служби. Ось як виглядає розмітка:

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

        <activity
            ...
        </activity>

        <service
            android:name=".DelayedMessageService"
            android:exported="false" >
        </service>
    </application>
</manifest>
```


*Объявление службы в AndroidManifest.xml. Среда Android Studio должна создать его автоматически.*

*Имя службы начинается с префикса «.», чтобы его можно было объединить с именем пакета для получения полного имени класса.*

Елемент <service> містить два атрибути. Атрибут android:name повідомляє Android ім'я служби - у нашому прикладі DelayedMessageService. Атрибут android:exported повідомляє Android, чи повинна служба використовуватись іншими додатками. Якщо привласнити йому false, це означає, що служба буде використовуватися лише у поточній програмі. Отже, ми вдало створили службу. Тепер можна переходити до наступного кроку - виклик цієї служби з активності.

У нашому додатку активність MainActivity буде запускати DelayedMessageService під час натискання кнопки. Почнемо з додавання кнопки до макету MainActivity. Додайте до файлу strings.xml такі значення (вони будуть використовуватися в коді активності та розмітці макета):

```
<string name="button_response">Timing!</string>
<string name="button_text">What is the secret of comedy?</string>
```

Потім внесіть зміни до activity_main.xml, щоб додати кнопку в MainActivity:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_text"
        android:id="@+id/button"
        android:onClick="onClick"
```

```
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Внесемо зміни до коду MainActivity.java, щоб активність запускала службу. Запуск служб активності дуже нагадує запуск інших активностей: потрібно створити явний інтент, призначений для служби, що запускається. Після цього служба запускається викликом методу startService():

```
package com.hfad.joke;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

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

    public void onClick(View view) {
        Intent intent = new Intent(this, DelayedMessageService.class);
        intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,
                        getResources().getString(R.string.button_response));
        startService(intent);
    }
}
```


Мы используем эти классы.

Выполняется при щелчке на кнопке.

Создать интент.

Добавить текст в интент.

Запустить службу.

Ось і весь код, необхідний для запуску служби з активності.

**Висновки.** Android використовує SQLite для організації зберігання даних. Клас SQLiteDatabase надає доступ до бази даних SQLite. Помічник SQLite дозволяє створювати бази даних SQLite та керувати ними. Помічник SQLite створюється розширенням класу SQLiteOpenHelper. Ви повинні реалізувати методи SQLiteOpenHelper onCreate() та onUpgrade().

База даних створюється при першому зверненні до неї. Базі даних необхідно присвоїти ім'я та номер версії. Якщо не присвоїти базі даних ім'я, вона створюється у пам'яті.

Метод onCreate() викликається під час створення бази даних. Метод onUpgrade() викликається при оновлення бази даних. Для виконання команд SQL використовується метод execSQL(String) класу SQLiteDatabase. Додавання записів у таблиці проводиться методом insert(). Оновлення записів

здійснюється методом update(). Видалення записів з таблиць здійснюється методом delete().

Курсори використовуються для читання та запису інформації до баз даних. Курсор створюється методом query() класу SQLiteDatabase. У внутрішній реалізації при цьому будується відповідна команда SQL SELECT.

Метод getWritableDatabase() повертає об'єкт SQLiteDatabase, який може використовуватись для читання та записи до бази даних. Метод getReadableDatabase() повертає об'єкт SQLiteDatabase, який надає доступ до бази даних лише для читання. Він також може надати можливість виконання операцій читання/запису, але це гарантовано. Для переміщення за даними в курсорі використовуються методи moveTo*(). Для читання даних з курсору використовуються методи get*(). Закривайте курсори та підключення до бази даних після завершення роботи з ними. Клас CursorAdapter представляє адаптер для роботи із курсорами. Використовуйте клас SimpleCursorAdapter для заповнення компонента ListView даними, повернутими курсором.

Проектуйте свої програми так, щоб в активності верхнього рівня розміщувався корисний контент. Метод changeCursor() класу CursorAdapter замінює курсор, який зараз використовується адаптером, іншим курсором, вибраним вами. Потім старий курсор закривається. Виконуйте код роботи з базами даних у фоновому потоці за допомогою об'єктів AsyncTask.

Служба є компонентом, виконуючий операції у фоновому режимі. Служба не володіє інтерфейсом користувача. Служба, що запускається, виконується у фоновому режимі невизначено довго, навіть при знищенні активності, що запустила її. Після завершення операції така служба зупиняється.

Служби оголошуються у файлі AndroidManifest.xml елементом <service>. Проста служба, що запускається, створюється розширенням класу IntentService та пере визначенням його методу onHandleIntent(). Клас IntentService розрахований на обробку інтентів. Служба, що запускається, запускається методом startService().

Перевизначаючи метод onStartCommand() класу IntentService, ви повинні викликати реалізацію суперкласу. Повідомлення будуються за допомогою об'єкта Notification.Builder. Щоб сповіщення запускало активність, використовуйте відкладений інтент. Для відображення повідомлень може використовуватись служба повідомлень Android. Пов'язана служба прив'язується до іншого компонента - наприклад, до активності. Активність може взаємодіяти з цим компонентом та отримувати результати.

Пов'язані служби зазвичай створюються розширенням класу Service. Ви повинні визначити свій об'єкт Binder і перевизначити метод onBind(). Цей метод викликається тоді, коли компонент хоче встановити зв'язок зі службою. Метод onCreate() класу Service викликається під час створення служби. Використовуйте його під час створення екземпляра. Метод onDestroy() класу Service викликається безпосередньо перед знищенням служби. Служба позиціонування Android використовується для отримання інформації про текучого місцезнаходження пристрою. Ви створюєте об'єкт LocationListener,

після чого реєструєте його у службі позиціонування. При цьому можна додати критерії частоти оповіщень про зміни. Якщо ви використовуєте модуль GPS пристрої, відповідний дозвіл необхідно додати до AndroidManifest.xml.

Щоб з'єднати активність зі службою, створіть об'єкт ServiceConnection. Перевизначте метод onServiceConnected() для отримання посилання на службу. Зв'язування зі службою здійснюється методом bindService(). Зв'язок розривається методом unbindService()

## ОСНОВНА ЛІТЕРАТУРА З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

### Навчальна та наукова література:


9. Dawn Griffiths, David Griffiths. Head First. Android Development. A Brain-Friendly Guide. O'REILLY. Beijing. Cambridge. Köln. Sebastopol. Tokyo.2015. 704 p.
10. Казимир В., Карпачев І., Усік А. Моделі системи безпеки ос android. URL: https://www.researchgate.net/publication/328775065_MODELI_SISTEMI_BEZPEKI_OS_ANDROID.
11. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв». Укладачі: Готович В. А., Михайлович Т. В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 216 с.
12. Розробка застосувань для мобільних пристроїв. Конспект лекцій. Міністерство освіти і науки України ЗНТУ. Кафедра програмних засобів. Запоріжжя 2016. 62с.
13. Сайко В.Г., Казіміренко В.Я., Літвінов Ю.М. Мережі бездротового широкосмугового доступу. Навчальний посібник. Кив: ДУТ, 2015. 216 с.
14. Опорний конспект лекцій з курсу «Мобільні інформаційні системи». Тернопільський національний економічний університет. Факультет комп'ютерних інформаційних технологій. Тернопіль. 2016. 60с.
15. Соколов В. Ю., Бурячок В. Л., Тадждіні М. М. Безпека безпроводових і мобільних мереж. Київ, КУБГ, 2019. 130 с.
16. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. Харків : НТУ «ХПІ», 2018. 284 с.


## ДОДАТКОВА ЛІТЕРАТУРА З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

### Навчальна та наукова література:


1. Cheng, F. Build Mobile Apps with Ionic 4 and Firebase: Hybrid Mobile App Development. Apress, 2018. 238 p.
2. Heckman R. Designing platform independent mobile apps and services. Hoboken: IEEE Press, 2016. 230 p.
3. John Horton. Android Programming for Beginners: Build in-depth, full-featured Android 9 Pie apps starting from zero programming experience, 2nd Edition.2018. 766 p.
4. Nalwaya, A., Paul, A. React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications. Apress, 2019.119 p.
5. Nolan G., Cinar O., Truxall D. Android best practices. Springer. 2014. 222p.
6. Six J. Application security for the android platform. Sebastopol, CA: O'Reilly, 2011. 97 p.

7. Windmill, E. Flutter in Action. Manning Publications, 2020 .P.310.

**Нормативно-правові акти:**

1. Про інформацію. Закон України від 02.10.1992, № 2657-XII. URL: https://zakon.rada.gov.ua/laws/show/2657-12#Text.
2. Про Державну службу спеціального зв'язку та захисту інформації України. Закон України: від 23.02.2006, № 3475-IV. URL: https://zakon.rada.gov.ua/laws/show/3475-15#Text.
3. Про захист інформації в інформаційно-комунікаційних системах. Закон України: від 05.07.1994, № 1170-VII. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text.
4. Про електронні комунікації: Закон України від 16.12.2020 : [із змінами і доповненнями]. Офіційний вісник України. 2021. № 6 (21.01.2021). Ст. 306.
5. Про основні засади забезпечення кібербезпеки України: Закон України від 05.10.2017 р. № 2163-VIII. URL: https://zakon.rada.gov.ua/laws/show/2163-19#Text.
6. Про захист персональних даних. Закон України від 01.06.2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text.
7. Стратегія кібербезпеки України, затверджена Указом Президента України від 26 серпня 2021 року № 447/2021. URL: https://zakon.rada.gov.ua/laws/show/447/2021#Text (дата звернення: 10.05.2023).
8. Стратегія інформаційної безпеки України, затверджена Указом Президента України від 28 грудня 2021 року № 685/2021. URL: https://zakon.rada.gov.ua/laws/show/685/2021#Text (дата звернення: 10.05.2023).
9. Про створення Центру протидії дезінформації: Рішення Ради національної безпеки і оборони України від 11 березня 2021 року, введено в дію Указом Президента України від 19 березня 2021 року № 106/2021. URL: https://zakon.rada.gov.ua/laws/show/106/2021#Text.
10. ДСТУ ISO/IEC 27000:2019 (ISO/IEC 27000:2018, IDT) Інформаційні технології. Методи захисту. Системи керування інформаційною безпекою. Огляд і словник термінів - На заміну ДСТУ ISO/IEC 27000:2017 (ISO/IEC 27000:2016, IDT).
11. ДСТУ ISO/IEC 27001:2015 (ISO/IEC 27001:2013; Cor 1:2014, IDT) / Поправка № 2:2019.
12. (ISO/IEC 27001:2013/Cor 2:2015, IDT) Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги.
13. ДСТУ ISO/IEC 27002:2015 (ISO/IEC 27002:2013; Cor 1:2014, IDT) / Поправка № 2:2019 (ISO/IEC 27002:2013/Cor 2:2015, IDT). Інформаційні технології. Методи захисту. Звід практик щодо заходів інформаційної безпеки.
14. ДСТУ ISO/IEC 27003:2018 Інформаційні технології. Методи захисту. Системи керування інформаційною безпекою. Настанова (ISO/IEC 27003:2017, IDT).
15. ДСТУ ISO/IEC 27004:2018 Інформаційні технології. Методи захисту. Системи керування інформаційною безпекою. Моніторинг, вимірювання, аналізування та оцінювання (ISO/IEC 27004:2016, IDT).
16. ДСТУ ISO/IEC 27005:2019 (ISO/IEC 27005:2018, IDT) Інформаційні технології. Методи захисту. Управління ризиками інформаційної безпеки - На заміну ДСТУ ISO/IEC 27005:2015 (ISO/IEC 27005:2011, IDT).

**Інформаційні ресурси в Інтернеті:**


1. Офіційний блог компанії Google. URL: http://googleblog.blogspot.com/search/label/Android
2. Онлайн-підтримка StackOverflow URL: http://stackoverflow.com/questions/tagged/android
3. Альянс відкритих мобільних пристроїв. URL: http://www.openhandsetalliance.com/
4. Google Play Hits 1 Million Apps. URL: https://mashable.com/archive/google-play-1-million
5. Android App Stats. URL: http://www.androlib.com/appstats.aspx
6. Java Editor URL: https://play.google.com/store/apps/details?id=air.JavaEditor
7. JavaIDEdroid URL: https://play.google.com/store/apps/details?id=ch.tanapro.JavaIDEdroid
8. The Professional Android IDE. URL: http://www.jetbrains.com/idea/features/android.html
9. NBAndroid. URL: http://plugins.netbeans.org/plugin/19545/nbandroid
10. Android Studio. URL: http://developer.android.com/sdk/index.html
11. Backup & restore Android apps using adb. URL: http://jonwestfall.com/2009/08/backup-restore-android-apps-using-adb/
12. SDK Tools. URL: http://developer.android.com/tools/sdk/tools-notes.html
13. Dalvik Executable format. URL: https://source.android.com/devices/tech/dalvik/dex-format.html
14. Android – Invoke JNI Based Methods (Bridging C/C++ And Java) URL: https://davanum.wordpress.com/2007/12/09/android-invoke-jni-based-methods-bridging-cc-and-java/
15. Native C applications for Android. URL: http://benno.id.au/blog/2007/11/13/android-native-apps
16. Android NDK. URL: https://developer.android.com/tools/sdk/ndk/index.html
17. SKIA graphics library in chrome: first impressions. URL: http://www.atoker.com/blog/2008/09/06/skia-graphics-library-in-chrome-first-283