МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ
ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ АВТОМАТИКИ ТА ЕНЕРГЕТИКИ

КАФЕДРА ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

Програмування додатків для ОС Android

Методичні вказівки до виконання лабораторних робіт

з елементами кредитно – трансферної
системи організації навчального процесу

для студентів денної форми навчання
за спеціальністю 123 «Комп'ютерна інженерія»
спеціалізацією «Комп'ютерні системи та мережі»

Укладачі:
Доцент Смірнов В.В.
Доцент Смірнова Н.В.

Кропивницький
2019 рік

Програмування додатків для ОС Android: Методичні вказівки до виконання лабораторних робіт для студентів денної форми навчання за спеціальністю 123 «Комп'ютерна інженерія» спеціалізацією «Комп'ютерні системи та мережі» / Укл.: В.В. Смірнов, Н.В. Смірнова. – Кропивницький: ЦНТУ, 2019 р. – 89 с.

Витяг з протоколу № 9
засідання кафедри Програмування комп'ютерних систем і мереж
від 24.04.2019 року

Укладачі:
Смірнова Наталія Володимирівна, к.т.н., доцент кафедри програмування комп'ютерних систем і мереж,
Смірнов Володимир Вікторович, к.т.н., доцент кафедри програмування комп'ютерних систем і мереж.

Для студентів денної форми навчання, що вивчають навчальну дисципліну «Програмування додатків для ОС Android» за спеціальністю 123 «Комп'ютерна інженерія» спеціалізацією «Комп'ютерні системи та мережі».

Лабораторні роботи розроблені на сучасному науково - методичному рівні. Відповідають вимогам до курсу практичного навчання студентів, у доступній формі викладені теоретичні відомості і описані практичні кроки оволодіння основами програмування мобільних пристроїв та систем.

Лабораторна робота № 1

Тема: Побудова програми на ОС Android

Першеий додаток Android

Представлено безліч нових концепцій і складових, необхідних для побудови додатків Android.

Додаток, який ми побудуємо, називається GeoQuiz. Він перевіряє, наскільки добре користувач знає географію. Користувач відповідає на питання, натискаючи кнопку True або False, GeoQuiz миттєво повідомляє йому результат. На рис. 1 показаний результат натискання кнопки False.

Рис. 1. Стамбул, а не Константинополь

Основи побудови програми

Додаток GeoQuiz складається з активності (activity) і макета (layout):

- Активність представлена екземпляром Activity - класу з Android SDK. Вона відповідає за взаємодію користувача з інформацією на екрані.

  Щоб реалізувати функціональність, необхідну вашому додатку, розробник пише субкласи Activity. У простому додатку буває досить одного субкласу; в складному додатку може знадобитися кілька.

  GeoQuiz - простий додаток, тому в ньому використовується всього один субклас Activity з ім'ям QuizActivity. Клас QuizActivity управляє призначеним для користувача інтерфейсом, зображеним на рис. 1.

- Макет визначає набір об'єктів користувальницького інтерфейсу і їх розташування на екрані. Макет формується з визначень, написаних на мові XML. Кожне визначення використовується для створення об'єкта, виведеного на екрані (наприклад, кнопки або тексту).

  Додаток GeoQuiz включає файл макета з ім'ям `activity_quiz.xml`. Розмітка XML в цьому файлі визначає призначений для користувача інтерфейс, який ви бачите на рис. 1.

Відносини між QuizActivity і activity_quiz.xml зображені на рис. 2.

Рис. 2. QuizActivity управляє інтерфейсом, що визначаються у файлі `activity_quiz.xml`

З огляду на все сказане, давайте побудуємо додаток.

Створення проекту Android

Робота починається зі створення проекту Android. Проект Android містить файли, з яких складається програма. Щоб створити новий проект, відкрийте Eclipse і виконайте команду File> New> Android Application Project, щоб запустити майстра нового додатка.

У першому діалоговому вікні введіть ім'я додатку GeoQuiz (рис. 3). Ім'я проекту автоматично оновлюється відповідно до імені програми. В поле Package Name введіть ім'я пакету com.bignerdranch.android.geoquiz.

Зверніть увагу: у введеному імені пакета використовується схема «зворотного DNS», згідно з якою доменне ім'я вашої організації записується в зворотному порядку з приєднанням суфіксів додаткових ідентифікаторів. Ця схема забезпечує унікальність імен пакетів і дозволяє розрізняти додатки на пристрої і в Google Play.

Наступні чотири поля налаштовують ваш додаток для роботи з різними версіями Android. Для додатка GeoQuiz досить налаштувань за замовчуванням.

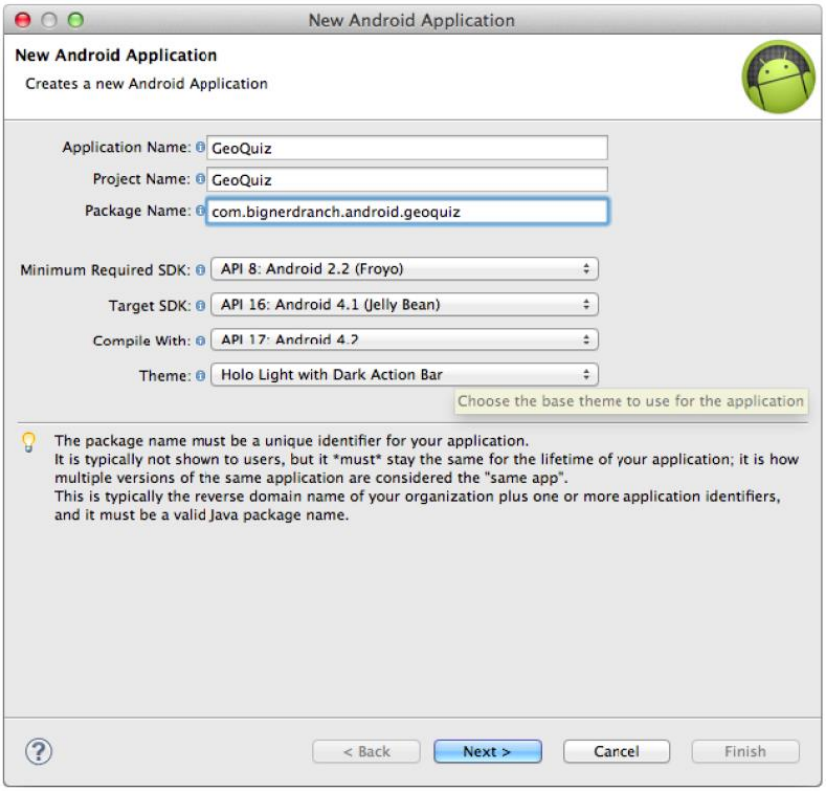

Рис. 3. Створення нової програми

Інструментарій Android оновлюється кілька разів на рік, тому вікно майстра на вашому комп'ютері може дещо відрізнятися. Зазвичай це не створює проблем; прийняті рішення залишаються практично незмінними. (Якщо ваше вікно не має нічого спільного з зображеним, значить, інструментарій змінився більш радикально.) Клацніть на кнопці Next.

У другому діалоговому вікні зніміть прапорець Create custom launcher icon (рис. 4). Додаток GeoQuiz буде використовувати значок запуску за замовчуванням. Простежте за тим, щоб прапорець Create activity був встановлений.

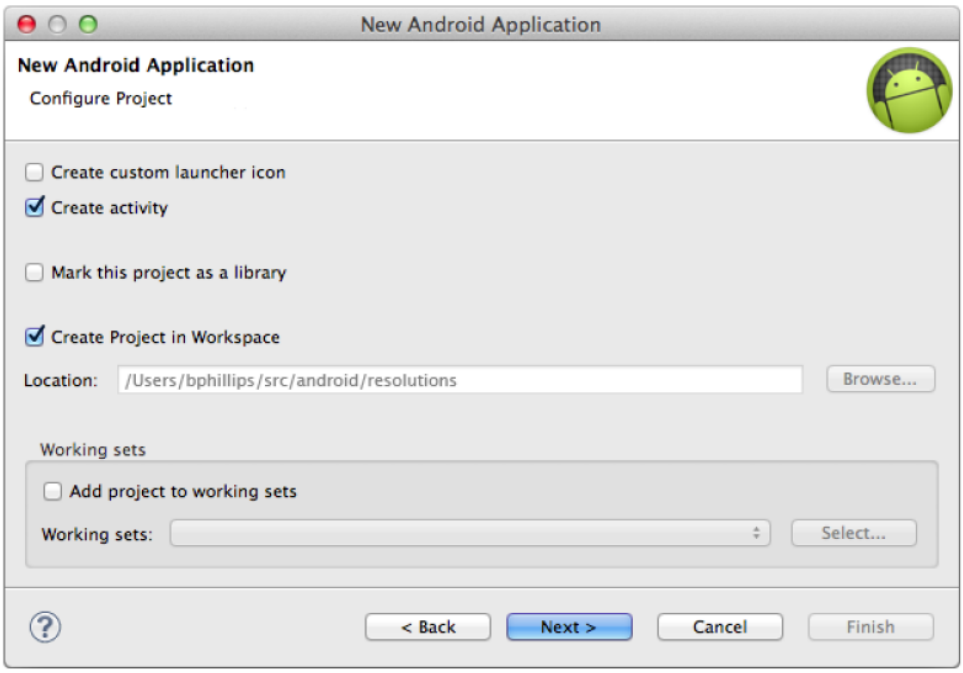

Рис. 4. Налаштування параметрів проекту

Клацніть на кнопці Next.

У наступному діалоговому вікні (рис. 5) вам пропонується вибрати вид створюваної активності. Виберіть в списку рядок Blank Activity (порожня активність).

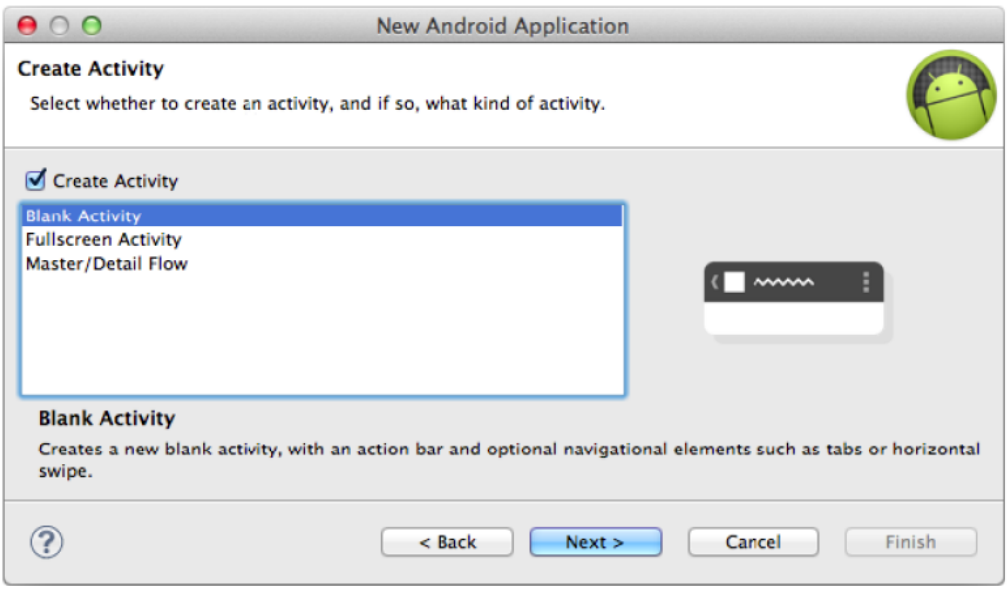

Рис. 5. Створення нової активності

Клацніть на кнопці Next.

В останньому діалоговому вікні майстра введіть ім'я субкласу активності QuizActivity (рис. 6). Зверніть увагу на суфікс Activity в імені класу. Його присутність не обов'язковий, але це дуже корисне погодження, якого варто дотримуватися.

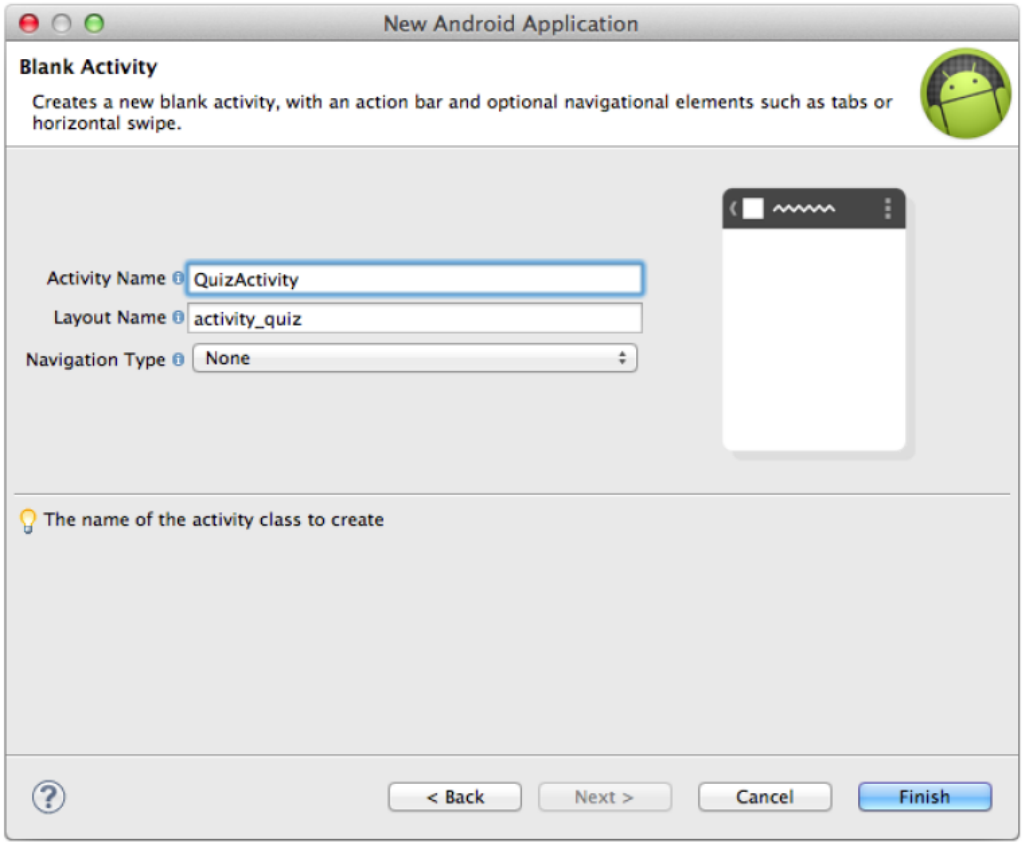

Рис. 6. Налаштування нової активності

Ім'я макета автоматично замінюється на activity_quiz відповідно до перейменуванням активності. Ім'я макета записується в порядку, зворотному імені активності; в ньому використовуються символи нижнього регістра, а слова поділяються символами підкреслення. Цю схему формування імен рекомендується застосовувати як для макетів, так і для інших ресурсів.

Залиште в списку Navigation Type значення None і клацніть на кнопці Finish. Середа Eclipse створює і відкриває новий проект.

Навігація в Eclipse

Eclipse відкриває ваш проект в інструментальному вікні (workbench window), як показано на рис. 7. (Якщо ви тільки що встановили Eclipse, закрийте вікно з привітанням Eclipse, щоб відкрити інструментальне вікно.)

Зліва знаходиться панель Package Explorer. На ній виконуються операції з файлами, що відносяться до вашого проекту.

В середині знаходиться панель редактора. Щоб вам було простіше приступити до роботи, Eclipse відкриває в редакторі activity_quiz.xml.

У правій і нижній частині інструментального вікна також розміщуються панелі. Щоб закрити будь-яку з панелей в правій частині, клацніть на кнопці X поруч з ім'ям панелі (рис. 7). Панелі в нижній частині об'єднані в групу вкладок. Замість того щоб закривати їх, згорніть всю групу за допомогою кнопки в правому верхньому куті панелі.

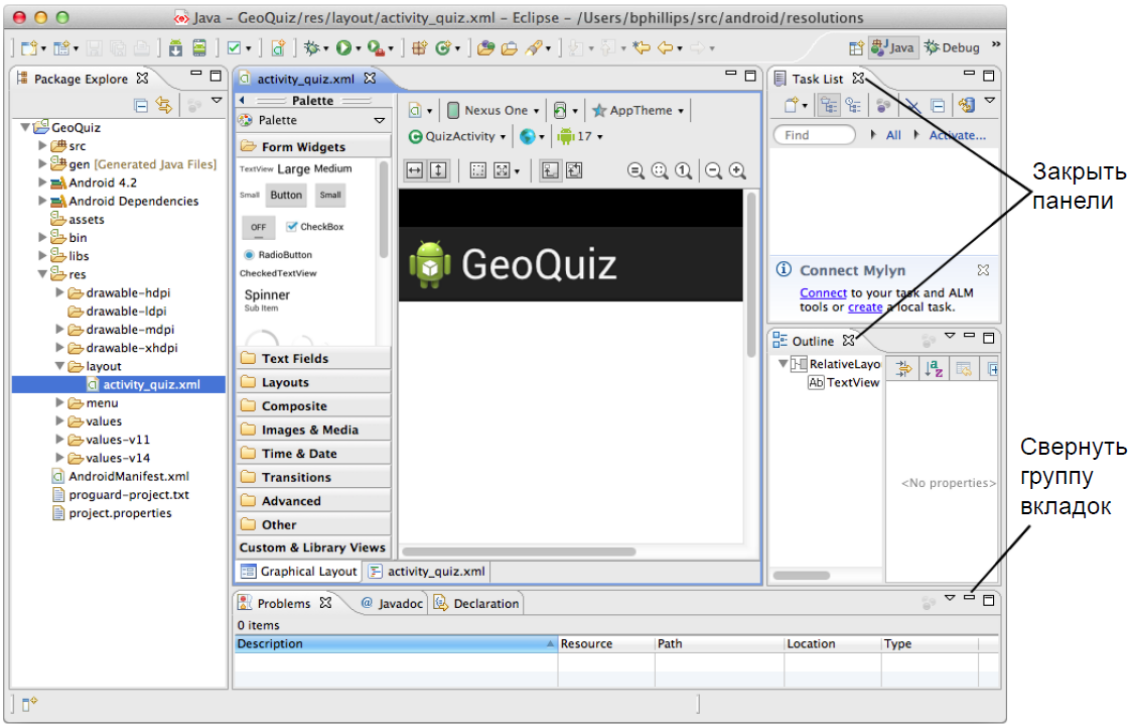

Рис. 7. Звільнення місця в інструментальному вікні

Коли ви звертаєте панель, вона закріплюється на полях інструментального вікна Eclipse. Наведіть покажчик миші на будь-який з маленьких значків на цих панелях інструментів, щоб переглянути імена згорнутих панелей; клацніть кнопкою миші, щоб відновити панель на екрані.

Побудова макета призначеного для користувача інтерфейсу

За замовчуванням Eclipse відкриває activity_quiz.xml в конструкторі макетів Android, в якому відображається графічне представлення макета. Цей режим буває корисний, але зараз ми будемо працювати в редакторі XML, щоб ви краще зрозуміли, як працюють макети.

Щоб перейти безпосередньо до розмітки XML, виберіть вкладку activity_quiz.xml в нижній частині редактора.

В даний час файл activity_quiz.xml визначає розмітку для активності за замовчуванням. Така розмітка часто змінюється, але XML буде виглядати приблизно так, як показано в лістингу 1.

Лістинг 1. Розмітка для активності

```
<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 = ". QuizActivity">

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

Перш за все зверніть увагу на те, що файл activity_quiz.xml не починається з рядку з оголошенням версії і кодування:

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

У версії ADT 21 цей рядок не є обов'язковою в файлах макетів Android. Втім, він все ще досить часто зустрічається в файлах.

Макет активності за замовчуванням визначає два віджети (widgets): RelativeLayout і TextView.

Віджети є структурні елементи, з яких складається призначений для користувача інтерфейс. Міні-програма може виводити текст або графіку, взаємодіяти з користувачем або розміщувати інші віджети на екрані. Кнопки, текстові поля, прапорці - все це різновиди віджетів.

Android SDK включає безліч віджетів, які можна налаштовувати для отримання потрібного оформлення та поведінки. Кожен віджет є екземпляром класу View або одного з його субкласів (наприклад, TextView або Button).

На рис. 8 показано, як виглядають на екрані віджети RelativeLayout і TextView, визначені в лістингу 1.

Але це не віджети, які нам потрібні. В інтерфейсі QuizActivity задіяні віджети:

- вертикальний віджет LinearLayout;
- TextView;
- горизонтальний віджет LinearLayout;
- дві кнопки Button.

На рис. 9 показано, як з цих віджетів утворюється інтерфейс QuizActivity.

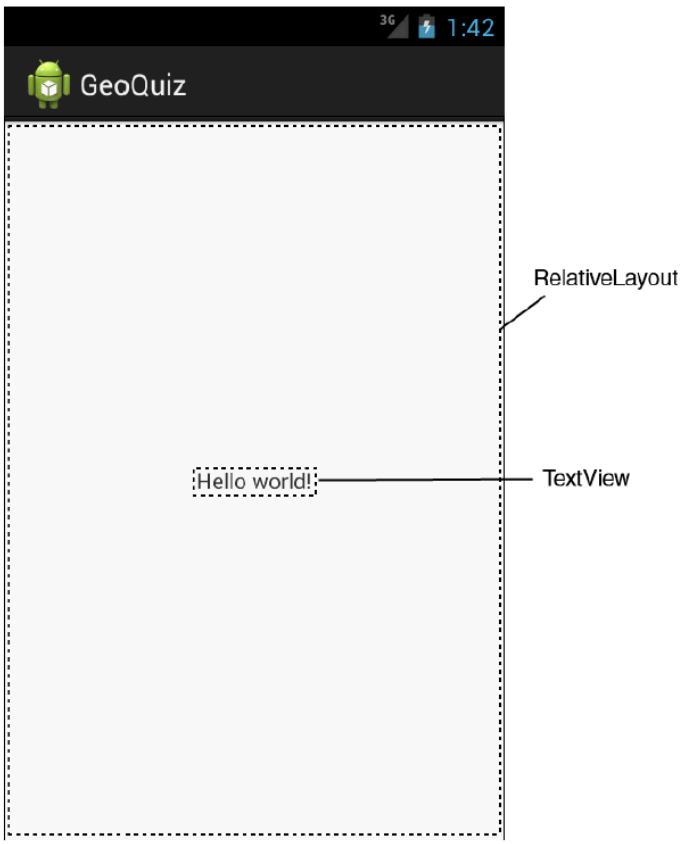

Рис. 8. Віджети за замовчуванням на екрані

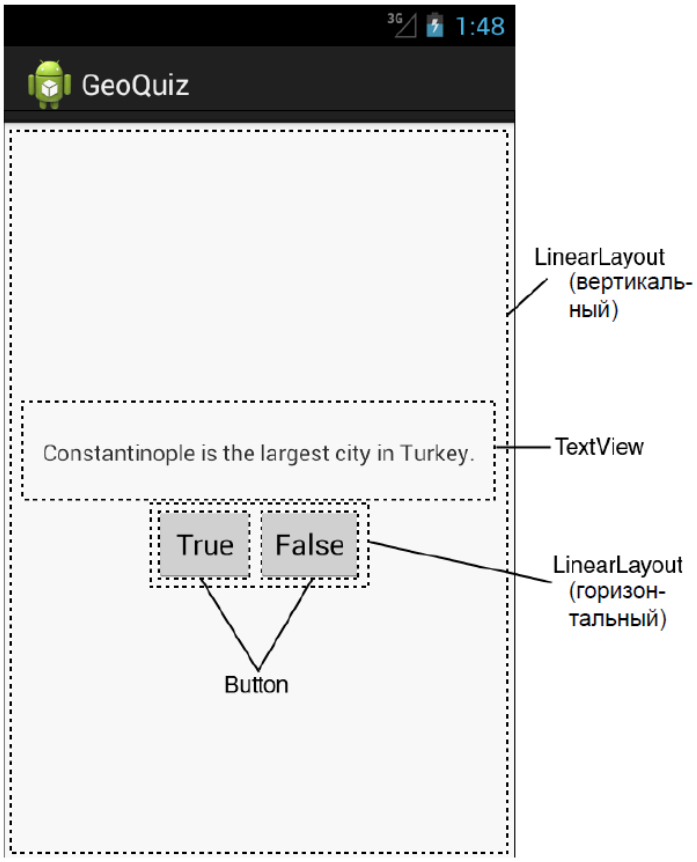

Рис. 9. Заплановане розташування віджетів на екрані

Тепер потрібно визначити ці віджети activity_quiz.xml.

Внесіть в файл activity_quiz.xml зміни, представлені в лістингу 2. Розмітка XML, яку потрібно видалити, виділена перекресленням, і додається розмітка XML - виділена жирним шрифтом.

Лістинг 2. Визначення віджетів в XML (activity_quiz.xml)

```
<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 = ". QuizActivity">

   <TextView
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: layout_centerHorizontal = "true"
      android: layout_centerVertical = "true"
      android: text = "@ string / hello_world" />
</ RelativeLayout>

<LinearLayout xmlns: android = "http://schemas.android.com/apk/res/android"
   android: layout_width = "match_parent"
   android: layout_height = "match_parent"
   android: gravity = "center"
   android: orientation = "vertical">

   <TextView
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: padding = "24dp"
```

```
        android: text = "@ string / question_text" />

    <LinearLayout
        android: layout_width = "wrap_content"
        android: layout_height = "wrap_content"
        android: orientation = "horizontal">

    <Button
        android: layout_width = "wrap_content"
        android: layout_height = "wrap_content"
        android: text = "@ string / true_button" />

    <Button
        android: layout_width = "wrap_content"
        android: layout_height = "wrap_content"
        android: text = "@ string / false_button" />

    </ LinearLayout>
</ LinearLayout>
```

Порівняйте XML з призначеним для користувача інтерфейсом, зображеним на рис. 9. Кожному віджету в розмітці відповідає елемент XML. Ім'я елемента визначає тип віджета.

Кожен елемент має набір атрибутів XML. Атрибути можна розглядати як інструкції по налаштуванню віджетів.

Щоб краще зрозуміти, як працюють елементи і атрибути, корисно поглянути на розмітку з точки зору ієрархії.

Ієрархія представлень

Віджети входять в ієрархію об'єктів View, звану ієрархією представлень.

На рис. 10 зображена ієрархія віджетів для розмітки XML з лістингу 2.

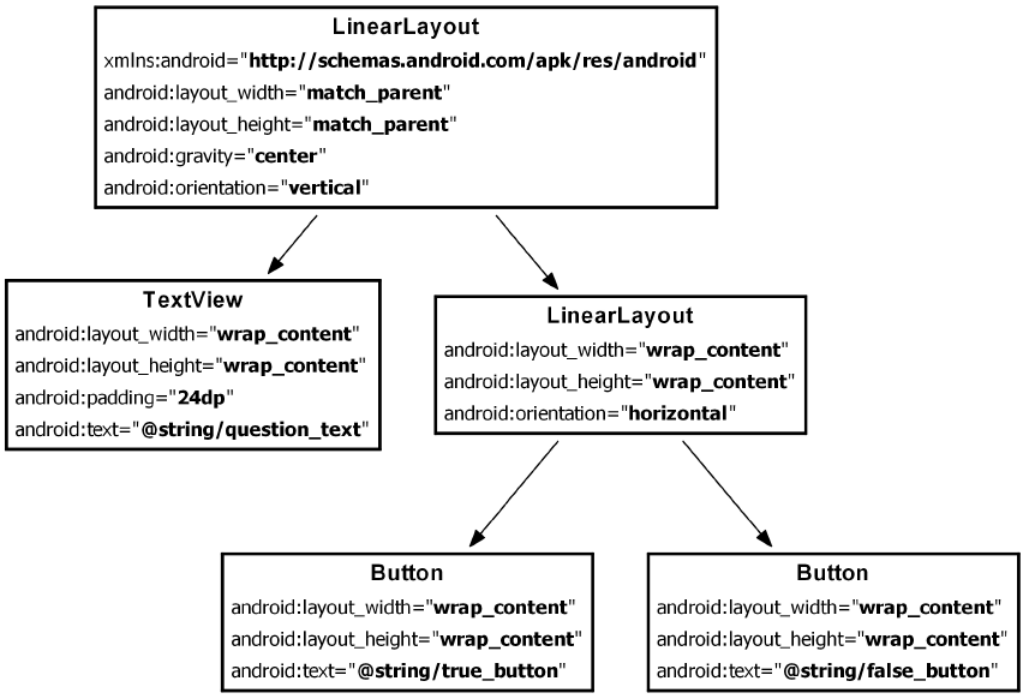

Рис. 10. Ієрархія віджетів і атрибутів

Кореневим елементом ієрархії представлень в цьому макеті є елемент LinearLayout. У ньому має бути зазначений простір імен XML ресурсів Android http://schemas.android.com/ apk/res/android.

LinearLayout успадковує від субкласів View з ім'ям ViewGroup. Віджет ViewGroup призначений для зберігання і розміщення інших віджетів. Він використовується в тих випадках, коли ви хочете побудувати віджети в один стовпець або рядок. Інші субкласи ViewGroup - FrameLayout, TableLayout і RelativeLayout.

Якщо віджет міститься в ViewGroup, він називається нащадком (child) ViewGroup. Кореневий елемент LinearLayout має двох нащадків: TextView і інший елемент LinearLayout. У LinearLayout є два власних нащадка Button.

Атрибути віджетів

Розглянемо деякі атрибути, використовувані для настройки віджетів.

`android:layout_width і android:layout_height`

Атрибути android: layout_width і android: layout_height, що визначають ширину і висоту, необхідні практично для всіх різновидів віджетів. Як правило, їм задаються значення match_parent або wrap_content:

- **`match_parent`** - розміри представлення визначаються розмірами батька.
- **`wrap_content`** - розміри представлення визначаються розмірами вмісту.

(Іноді в розмітці зустрічається значення fill_parent. Це застаріле значення еквівалентно match_parent.)

У кореневому елементі LinearLayout атрибути ширини і висоти рівні match_parent. Елемент LinearLayout є кореневим, але у нього все одно є батько - представлення, яке надає Android для розміщення ієрархії представлень вашого застосування.

У інших віджетів макета ширини і висоті задається значення wrap_content. На рис. 9 показано, як в цьому випадку визначаються їх розміри.

Віджет TextView трохи більше міститься в ньому тексту через атрибута android:padding = "24dp". Цей атрибут наказує віджету додати заданий відступ навколо вмісту при визначенні розміру, щоб текст питання не стикався з кнопкою. (Dp - це пікселі, які не залежать від щільності (density-independent pixels).

`android:orientation`

Атрибут android: orientation двох віджетів LinearLayout визначає, як будуть вибудовуватися нащадки - по вертикалі або горизонталі. Кореневий елемент LinearLayout має вертикальну орієнтацію; у його нащадка LinearLayout горизонтальна орієнтація.

Порядок визначення нащадків визначає порядок їх відображення на екрані.

У вертикальному елементі LinearLayout нащадок, визначений першим, розташовується вище за інших. У горизонтальному елементі LinearLayout перший нащадок є крайнім лівим. (Якщо тільки в ньому не використовується мова з писемністю справа наліво - наприклад, арабська або іврит; в цьому випадку перший нащадок буде знаходитися в крайній правій позиції.)

`android:text`

Віджети TextView і Button містять атрибути android:text. Цей атрибут повідомляє віджету, який текст повинен відображатися

Значення атрибутів являють собою не рядкові літерали, а посилання на строкові ресурси.

Строковий ресурс - рядок, що знаходиться в окремому файлі XML, який називається *строковим файлом*. Віджету можна призначити фіксований рядок (наприклад, android: text = "True"), але зазвичай так робити не варто. Краще розміщувати рядки в окремому файлі, а потім посилатися на них.

Створення строкових ресурсів

Кожен проект включає строковий файл за замовчанням з ім'ям strings.xml. Знайдіть на панелі Package Explorer каталог res / values, розкрийте його і відкрийте файл strings.xml.

Виберіть вкладку strings.xml в нижній частині редактора.

У шаблон вже включено кілька строкових ресурсів. Видаліть рядок з ім'ям hello_world і додайте три нові рядки для вашого макета.

Лістинг 3. Додавання строкових ресурсів (strings.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
    <Resources>
    <String name = "app_name"> GeoQuiz </ string>
    <String name = "hello_world"> Hello, world! </ String>
    <String name = "question_text"> Constantinople is the largest city in
    Turkey.
</ String>
    <String name = "true_button"> True </ string>
    <String name = "false_button"> False </ string>
    <String name = "menu_settings"> Settings </ string>
</ Resources>
```

(Не видаляйте рядок menu_settings - ваш проект містить готове меню. Видалення menu_settings викличе каскадні помилки в інших файлах, що відносяться до меню.)

Тепер за посиланням @ string / false_button в будь-якому файлі XML проекту GeoQuiz ви будете отримувати строковий літерал "False" на стадії виконання.

Збережіть файл strings.xml. Якщо у файлі activity_quiz.xml залишалися помилки, пов'язані з відсутністю строкових ресурсів, вони повинні зникнути.

Строковий файл за умовчанням називається strings.xml, але йому можна присвоїти будь-яке ім'я на ваш розсуд. Проект може містити кілька строкових файлів. Якщо файл знаходиться в каталозі res / values /, містить кореневий елемент resources і дочірні елементи string, ваші рядки будуть знайдені і правильно використані додатком.

Попередній перегляд макета

Макет готовий і його можна переглянути в графічному конструкторі. Перш за все, переконайтесь в тому, що файли збережені і не містять помилок. Потім поверніться до файлу activity_quiz.xml і виберіть вкладку Graphical Layout в нижній частині редактора.

Рис. 1.11. Попередній перегляд в графічному конструкторі макетів (activity_quiz.xml)

Від розмітки XML до об'єктів View

Як елементи XML у файлі activity_quiz.xml перетворюються в об'єкти View? Відповідь на це питання починається з класу QuizActivity.

При створенні проекту GeoQuiz був автоматично створений субклас Activity з ім'ям QuizActivity. Файл класу QuizActivity знаходиться в каталозі src (в якому зберігається Java-код вашого проекту).

На панелі Package Explorer відкрийте каталог src, а потім вміст пакета com.bignerdranch. android.geoquiz. Відкрийте файл QuizActivity.java і перегляньте його вміст (лістинг 4).

Лістинг 4. Файл класу QuizActivity за замовчуванням (QuizActivity.java)

```
package com.bignerdranch.android.geoquiz;

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

public class QuizActivity extends Activity {

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

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

(Якщо ви не бачите всі директиви import, клацніть на знаку ⊕ зліва від першої директиви import, щоб розкрити список.)

Файл містить два методи Activity: onCreate (Bundle) і onCreateOptionsMenu (Menu).

Метод onCreate (Bundle) викликається при створенні екземпляра субкласу активності. Такому класу потрібен призначений для користувача інтерфейс, яким він буде керувати. Щоб надати класу активності призначений його користувальницький інтерфейс, слід викликати наступний метод Activity:

```
public void setContentView (int layoutResID)
```

Цей метод заповнює (inflates) макет і виводить його на екран. При заповненні макета створюються екземпляри всіх віджетів в файлі макета з параметрами, обумовленими його атрибутами. Щоб вказати, який саме макет слід заповнити, ви передаєте ідентифікатор ресурсу макета.

Ресурси і ідентифікатори ресурсів

Макет являє собою ресурс. Ресурсом називається частина програми, яка не є кодом - графічні файли, аудіофайли, файли XML і т. д.

Ресурси проекту знаходяться в підкаталозі каталогу res. На панелі Package Explorer видно, що файл activity_quiz.xml знаходиться в каталозі res / layout /. Строковий файл, який містить рядкові ресурси, знаходиться в res / values /.

Для звернення до ресурсу в коді використовується його ідентифікатор ресурсу. Нашому макету призначений ідентифікатор ресурсу R.layout.activity_quiz.

Щоб переглянути поточні ідентифікатори ресурсів проекту GeoQuiz, відкрийте Package Explorer і розкрийте вміст каталогу gen. Знайдіть і відкрийте файл R.java. Оскільки цей файл генерується процесом складання Android, вам не слід його змінювати, про що делікатно попереджає напис на початку файлу.

Лістинг 5. Поточні ідентифікатори GeoQuiz

```
/ * AUTO-GENERATED FILE. DO NOT MODIFY.
...
* /
package com.bignerdranch.android.geoquiz;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int ic_launcher = 0x7f020000;
   }
   public static final class id {
      public static final int menu_settings = 0x7f070003;
   }
   public static final class layout {
      public static final int activity_quiz = 0x7f030000;
   }
   public static final class menu {
      public static final int activity_quiz = 0x7f060000;
   }
   public static final class string {
      public static final int app_name = 0x7f040000;
      public static final int false_button = 0x7f040003;
      public static final int menu_settings = 0x7f040006;
      public static final int question_text = 0x7f040001;
      public static final int true_button = 0x7f040002;
   }
   ...
}
```

Ім'я R.layout.activity_quiz - це цілочисельна константа з ім'ям activity_quiz з внутрішнього класу layout класу R.

Рядкам також призначаються ідентифікатори ресурсів. ці посилання зазвичай виглядають так:

```
setTitle (R.string.app_name);
```

Android генерує ідентифікатор ресурсу для всього макета і для кожного рядка, але не для окремих віджетів з файлу activity_quiz.xml. Не кожному віджету потрібен ідентифікатор ресурсу. Ми будемо взаємодіяти тільки з двома кнопками, тому ідентифікатори ресурсів потрібні тільки їм.

Щоб згенерувати ідентифікатор ресурсу для віджета, включіть в визначення віджета атрибут android: id. У файлі activity_quiz.xml додайте атрибут android: id для кожної кнопки.

Лістинг 6. Додавання ідентифікаторів кнопок (activity_quiz.xml)

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

   <TextView
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: padding = "24dp"
      android: text = "@ string / question_text" />

   <LinearLayout
```

```
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: orientation = "horizontal">

   <Button
      android: id = "@ + id / true_button"
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: text = "@ string / true_button" />

   <Button
      android: id = "@ + id / false_button"
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: text = "@ string / false_button" />

   </ LinearLayout>
</ LinearLayout>
```

Зверніть увагу: знак + присутній в значеннях android: id, але не в значеннях android: text. Це пов'язано з тим, що ми створюємо ідентифікатори, а на рядки тільки посилаємося.

Збережіть файл activity_quiz.xml. Поверніться до файлу R.java і переконайтеся в тому, що у внутрішньому класі R.id додалися два нових ідентифікатора ресурсів.

Лістинг 7. Нові ідентифікатори ресурсів (R.java)

```
public final class R {
...
   public static final class id {
      public static final int false_button = 0x7f070001;
      public static final int menu_settings = 0x7f070002;
      public static final int true_button = 0x7f070000;
}
...
```

Підключення віджетів до програми

Тепер, коли кнопкам призначені ідентифікатори ресурсів, до них можна звертатися в QuizActivity. Все починається з додавання двох змінних.

Введіть наступний код в QuizActivity.java. (Не використовуйте автозавершення; введіть його самостійно.) Після збереження файлу виводяться два повідомлення про помилки.

Лістинг 8. Додавання полів (QuizActivity.java)

```
public class QuizActivity extends Activity {
   private Button mTrueButton;
   private Button mFalseButton;

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

Зараз ми виправимо помилки, а поки зверніть увагу на префікс m у імен двох полів (змінних екземплярів). Цей префікс відповідає схемі формування імен Android.

Наведіть покажчик миші на прапори помилок зліва від коду. Вони повідомляють про однакову проблему: «Button не вдається зв'язати з типом» (Button can not be resolved to a type).

Щоб позбутися від помилок, слід імпортувати клас android.widget.Button в QuizActivity.java. Введіть наступну директиву імпортування на початку файлу:

```
import android.widget.Button;
```

Організація імпорту

Команда «організації імпорту» наказує середовищу Eclipse проаналізувати код і визначити, що потрібно вашій програмі з Java і Android SDK. Вона імпортує все необхідне і видаляє раніше імпортовані класи, які не використовуються в програмі.

Щоб провести організацію імпорту, натисніть:
- Command + Shift + O на Mac;
- Ctrl + Shift + O в Windows і Linux.

Помилки повинні зникнути. (Якщо вони залишилися, пошукайте помилки в коді і XML.)

Тепер ви можете підключити свої віджети-кнопки. Процедура складається з двох кроків:
- отримання посилань на заповнені об'єкти View;
- призначення для цих об'єктів слухачів, що реагують на дії користувача.

Отримання посилань на віджети

У класі активності можна отримати посилання на заповнений віджет, для чого використовується наступний метод Activity:

```
public View findViewById (int id)
```

Метод отримує ідентифікатор ресурсу віджета і повертає об'єкт View.

У файлі QuizActivity.java по ідентифікаторам ресурсів ваших кнопок можна отримати заповнені об'єкти і привласнити їх полях. Повернений об'єкт View перед привласненням необхідно перетворити в Button.

Лістинг 9. Отримання посилань на віджети (QuizActivity.java)

```
public class QuizActivity extends Activity {
   private Button mTrueButton;
   private Button mFalseButton;

   @Override
   public void onCreate (Bundle savedInstanceState) {
      super.onCreate (savedInstanceState);
      setContentView (R.layout.activity_quiz);
      mTrueButton = (Button) findViewById (R.id.true_button);
      mFalseButton = (Button) findViewById (R.id.false_button);
   }
...
}
```

Призначення слухачів

Додатки Android зазвичай управляються подіями (event-driven). На відміну від програм командного рядка або сценаріїв, такі додатки запускаються і чекають настання деякого події - наприклад, натискання кнопки користувачем. (Події також можуть ініціюватися ОС або іншим додатком, але події, ініційовані користувачем, найбільш очевидні.)

Коли ваш додаток очікує настання конкретної події, ми говоримо, що він «прослуховує» дану подію. Об'єкт, що створюється для відповіді на подію, називається слухачем (listener). Такий об'єкт реалізує інтерфейс слухача даної події.

Android SDK постачається з інтерфейсами слухачів для різних подій, тому вам не доведеться писати власні реалізації. У нашому випадку подією, що прослуховується є «клацання» на кнопці, тому слухач повинен реалізувати інтерфейс View.OnClickListener.

Почнемо з кнопки True. У файлі QuizActivity.java включите наступний фрагмент коду в метод onCreate (...) безпосередньо після присвоювання.

Лістинг 10. Призначення слухача для кнопки True (QuizActivity.java)

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

   mTrueButton = (Button) findViewById (R.id.true_button);
   mTrueButton.setOnClickListener (new View.OnClickListener () {

      @Override
      public void onClick (View v) {
         // Поки нічого не робить, але скоро буде!
      }
   });
   mFalseButton = (Button) findViewById (R.id.false_button);
 }
```

(Якщо ви отримаєте помилку «View can not be res4olved to a type», проведіть організацію імпорту комбінацією Command + Shift + O або Ctrl + Shift + O для імпортування класу View.)

У лістингу 10 призначається слухач, який інформує про натискання віджета Button з ім'ям mTrueButton. Метод setOnClickListener (OnClickListener) отримує в аргументі слухача - а конкретніше об'єкт, який реалізує OnClickListener.

Анонімні внутрішні класи

Слухач реалізований у вигляді анонімного внутрішнього класу. Можливо, синтаксис не очевидний; просто запам'ятайте: все, що укладено в зовнішнє пару круглих дужок, передається setOnClickListener (OnClickListener). В круглих дужках створюється новий безіменний клас, вся реалізація якого передається викликається методу.

```
mTrueButton.setOnClickListener (new View.OnClickListener () {
   @Override
   public void onClick (View v) {
      // Поки нічого не робить, але скоро буде!
   }
});
```

Всі слухачі будуть реалізовані у вигляді анонімних внутрішніх класів. У цьому випадку реалізація методів слухача знаходиться безпосередньо там, де ви хочете її бачити, а ми уникаємо витрат ресурсів на створення іменованого класу, який буде використовуватися тільки в одному місці.

Так як анонімний клас реалізує OnClickListener, він повинен реалізувати єдиний метод цього інтерфейсу onClick (View).

Інтерфейс слухача вимагає, щоб метод onClick (View) був реалізований, але не встановлює ніяких правил щодо того, як саме він буде реалізований.

Призначте аналогічного слухача для кнопки False.

Лістинг 11. Призначення слухача для кнопки False (QuizActivity.java)

```
...
   mTrueButton.setOnClickListener (new View.OnClickListener () {
      @Override
      public void onClick (View v) {
         // Поки нічого не робить, але скоро буде!
      }
   });

   mFalseButton = (Button) findViewById (R.id.false_button);
   mFalseButton.setOnClickListener (new View.OnClickListener () {
      @Override
      public void onClick (View v) {
         // Поки нічого не робить, але скоро буде!
      }
   });
}
```

Повідомлення

У нашому додатку кожна кнопка буде виводити на екран тимчасове повідомлення (toast) - коротке повідомлення, яке містить будь - яку інформацію для користувача, але не вимагає ні введення, ні дій. Наші повідомлення будуть повідомляти користувачу, чи правильно він відповів на питання (рис. 12).

Для початку поверніться до файлу strings.xml і додайте рядкові ресурси, які будуть відображатися в повідомленні.

Лістинг 12. Додавання рядків повідомлень (strings.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Resources>
   <String name = "app_name"> GeoQuiz </ string>
   <String name = "question_text"> Constantinople is the largest city
in Turkey. </ string>
   <String name = "true_button"> True </ string>
   <String name = "false_button"> False </ string>
   <String name = "correct_toast"> Correct! </ String>
   <String name = "incorrect_toast"> Incorrect! </ String>
   <String name = "menu_settings"> Settings </ string>
</ Resources>
```

Рис. 12. Повідомлення з інформацією для користувача

Повідомлення створюється викликом наступного методу класу Toast:

```
public static Toast makeText (Context context, int resId, int duration)
```

Параметр Context зазвичай містить екземпляр Activity (Activity є субкласом Context). У другому параметрі передається ідентифікатор ресурсу рядка, який повинен виводитися в повідомленні. Параметр Context необхідний класу Toast для пошуку і використання ідентифікатора ресурсу рядка. Третій параметр зазвичай містить одну з двох констант Toast, що визначають тривалість перебування повідомлення на екрані.

Після того як об'єкт повідомлення буде створений, викличте Toast.show(), щоб повідомлення з'явилося на екрані.

В класі QuizActivity виклик makeText (...) буде присутній в слухачі кожної кнопки (лістинг 13). Замість того щоб вводити все вручну, спробуйте додати ці виклики з використанням функції автозавершення середовища Eclipse.

Автозавершення

Почніть вводити новий код з лістингу 13. Коли ви дійдете до точки після класу Toast, на екрані з'являється список методів і констант класу Toast.

Щоб вибрати одну з рекомендацій, натисніть клавішу Tab, щоб переключитися на підказку автозавершения. (Якщо ви не збираєтеся використовувати автозавершення, просто продовжуйте друкувати. Без натискання клавіші Tab або клацання на вікні підказки підстановка не виконується.)

Виберіть в списку рекомендацій метод makeText (Context, int, int). Механізм автозавершения додає виклик методу разом з наповнювачами аргументів.

Перший заповнювач виділяється автоматично; введіть реальне значення - QuizActivity.this. Потім знову натисніть Tab, щоб перейти до наступного заповнювача, і так далі, поки не буде введено весь код з лістингу 13.

Лістинг 13. Створення повідомлень (QuizActivity.java)

```
mTrueButton.setOnClickListener (new View.OnClickListener () {
   @Override
   public void onClick (View v) {
      Toast.makeText (QuizActivity.this,
                      R.string.incorrect_toast,
                      Toast.LENGTH_SHORT) .show ();
   }
});

mFalseButton.setOnClickListener (new View.OnClickListener () {
   @Override
   public void onClick (View v) {
      Toast.makeText (QuizActivity.this,
                      R.string.correct_toast,
                      Toast.LENGTH_SHORT) .show ();
   }
});
```

У виклику makeText (...) екземпляр QuizActivity передається в аргументі Context. У цьому місці коду ми визначаємо анонімний клас, де this позначає View.OnClickListener.

Завдяки використанню автозавершения вам не доведеться виконувати організацію імпорту, щоб клас Toast став доступним. Коли ви погоджуєтеся на рекомендацію автозавершения, необхідні класи імпортуються автоматично.

Виконання в емуляторі

Для запуску додатків Android необхідний пристрій - фізичний або віртуальний. Віртуальні пристрої працюють під управлінням емулятора Android, включеного в поставку засобів розробника.

Щоб створити віртуальний пристрій Android (AVD, Android Virtual Device), виконайте команду Window> Android Virtual Device Manager. Коли на екрані з'явиться вікно AVD Manager, клацніть на кнопці New ... в правій частині цього вікна.

Відкривається діалогове вікно з численними параметрами налаштування віртуального пристрою. Виберіть емуляцію пристрою Galaxy Nexus з Google APIs (Google Inc.) - API Level 17, як показано на рис. 13. Якщо ви працюєте в системі Windows, можливо, для правильної роботи AVD вам доведеться зменшити обсяг пам'яті (RAM) з 1024 до 512. Клацніть на кнопці OK.

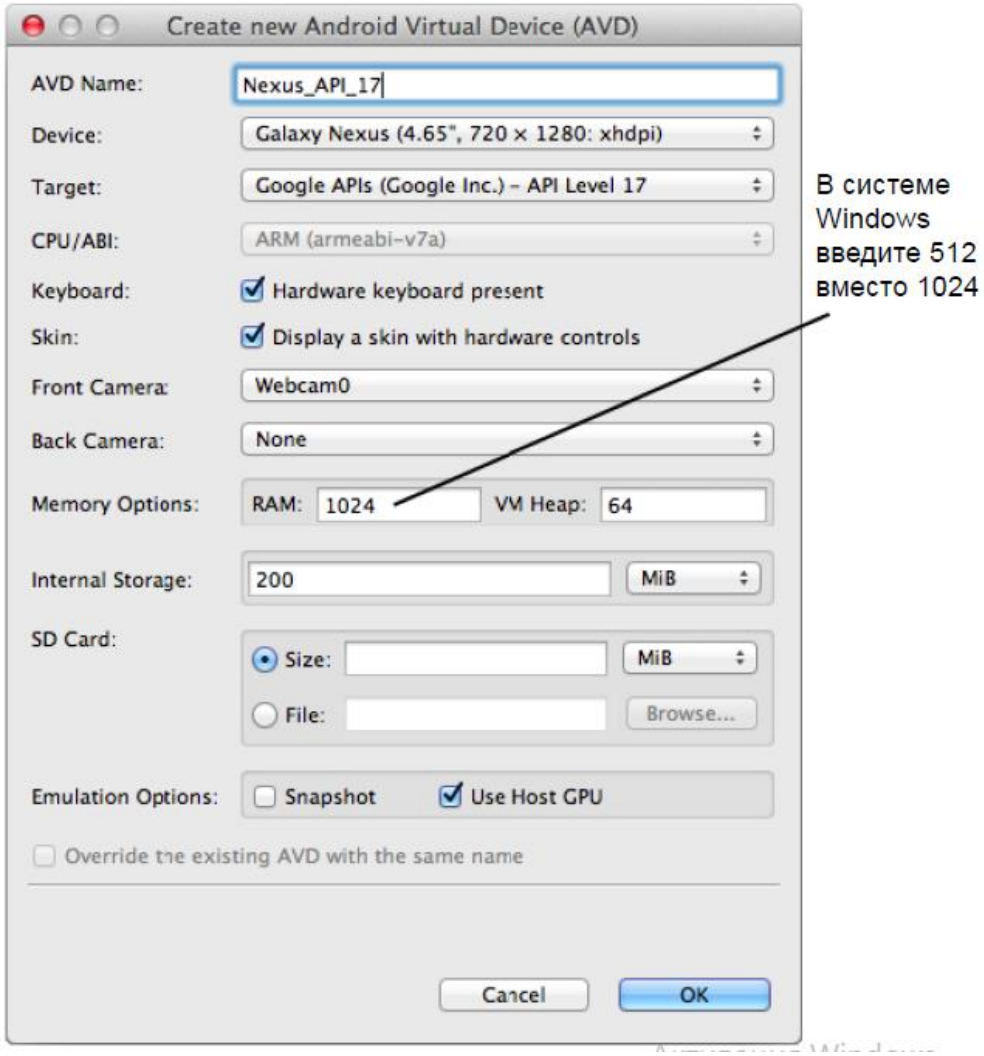

Рис. 13. Створення нового віртуального пристрою Android

Коли віртуальний пристрій буде створено, в ньому можна запустити додаток GeoQuiz. На панелі Package Explorer клацніть правою кнопкою миші на папці проекту GeoQuiz. У контекстному меню виберіть команду Run As > Android Application. Eclipse знаходить створений віртуальний пристрій, встановлює на ньому пакет програми та запускає додаток. Eclipse може запитати, чи хочете ви використовувати автоматичний моніторинг з LogCat - погоджуйтеся.

Якщо при запуску GeoQuiz або натисканні кнопки відбувається збій, в нижній частині інструментального вікна відкривається панель LogCat. Шукайте виключення в журналі; вони будуть виділятися червоним кольором. У стовпці Text вказується ім'я виключення і рядок, в якій виникла проблема.

```
Text
at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21)
at android.app.Activity.performCreate(Activity.java:5008)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
```

Рис. 14. Виняток NullException в рядку 21

Порівняйте свій код з кодом і спробуйте знайти джерело проблеми. Потім знову запустіть додаток.

Не закривайте емулятор; не варто чекати, поки він завантажується, при кожному запуску. Ви можете зупинити додаток кнопкою Back (U-подібна стрілка), а потім знову запустити додаток з Eclipse, щоб протестувати зміни.

Емулятор корисний, але тестування на реальному пристрої дає точніші результати.

Процес побудови додатків Android

Eclipse будує проект автоматично в процесі його зміни, а не по команді. Під час побудови інструментарій Android бере ваші ресурси, код і файл AndroidManifest.xml (що містить метадані додатки) і перетворює їх в файл .apk. Отриманий файл підписується налагоджувальний ключем, що дозволяє запускати його в емуляторі.

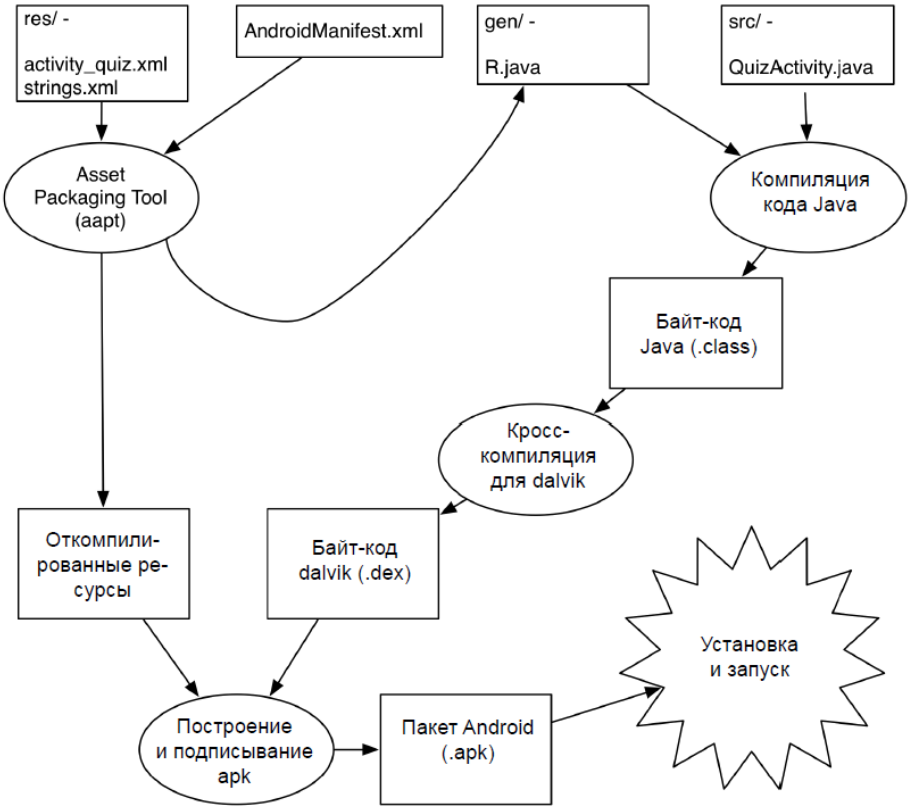

Рис. 15. Побудова GeoQuiz

Як вміст activity_quiz.xml перетворюється в об'єкти View в додатку? В процесі побудови утиліта aapt (Android Asset Packaging Tool) компілює ресурси файлів макетів в більш компактний формат. Відкомпілювалися ресурси упаковуються в файл .apk. Потім, коли метод setContentView (...) викликається в методі onCreate (...) класу QuizActivity, QuizActivity використовує клас LayoutInflater для створення екземплярів всіх об'єктів View, визначених у файлі макета.

(Також класи представлень можна створити на програмному рівні в класі активності замість визначення їх в XML. Однак відділення презентаційної логіки від логіки додатку має свої переваги, головне з яких - використання змін конфігурації, вбудованих в SDK.)

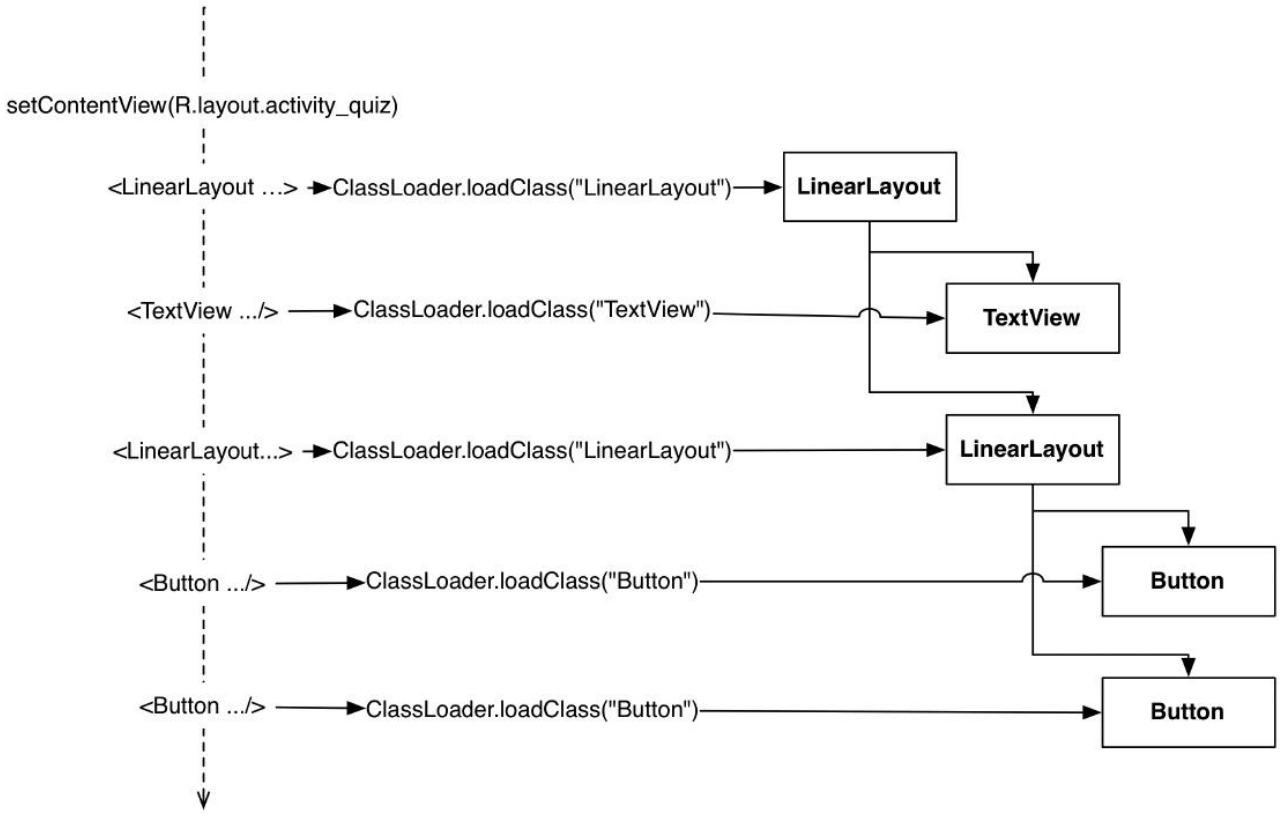

Рис. 16. Заповнення activity_quiz.xml

Засоби побудови програм Android

Всі програми, які ми запускали до теперішнього часу, виконувалися з середовища Eclipse. Цей процес інтегрований в використовуваний плагін ADT - він викликає стандартні засоби побудови програм Android (такі, як aapt), але сам процес побудови проходить під керуванням Eclipse.

Може виявитися, що вам з якихось своїх причин потрібно провести побудову за межами середовища Eclipse. Для цього найпростіше скористатися програмою командного рядка. Дві найпопулярніших утиліти такого роду - maven і ant. Ant поступається по функціональності, але набагато простіше у використанні.

Перш за все виконайте два кроки:

- Переконайтеся в тому, що програма ant встановлена і її можна запустити.
- Переконайтеся в тому, що папки tools / і platform-tools / в Android SDK включені в шляху пошуку виконуваних файлів.

Тепер перейдіть в каталог проекту і виконайте наступну команду:

```
$ Android update project -p.
```

Шаблон генератора проектів Eclipse не включає файл build.xml для ant. Перша команда генерує файл build.xml за вас; вам залишається лише виконати її вдруге.

Побудуйте проект. Щоб побудувати і підписати відлагоджувальний файл .apk, виконайте наступну команду в тій же папці:

```
$ Ant debug
```

Ця команда безпосередньо будує програму. Вона створює файл .apk, що знаходиться в папці bin / ім'я-проекту-debug.apk. Коли файл .apk буде створений, встановіть його за допомогою такої команди:

```
$ Adb install bin / ім'я-проекту-debug.apk
```

Команда встановлює додаток на підключений пристрій, але не запускає його - ви повинні зробити це вручну.

ЗАВДАННЯ

Щоб уникнути випадкового пошкодження поточного проекту, створіть копію в Eclipse і можна практикуватися на ній.

Клацніть правою кнопкою миші на проекті в Package Explorer, виберіть команду Copy, потім знову клацніть правою кнопкою миші і вставте скопійований проект.

Вам буде запропоновано ввести ім'я нового проекту, який з'являється на панелі Package Explorer готовим до роботи.

Завдання 1. Додавання слухача для TextView

Кнопка Next зручна, зробити так, щоб користувач міг перейти до наступного питання простим натисканням на віджеті TextView.

Підказка. Для TextView можна використовувати слухача View.OnClickListener, який використовувався з Button, тому що клас TextView також є похідним від View.

Завдання 2. Додавання кнопки повернення

Додайте кнопку для повернення до попереднього запитання. Інтерфейс повинен виглядати приблизно так, як показано на рис. 2.9.

Рис. 2.9. Тепер з кнопкою повернення!

Завдання 3. Від Button до ImageButton

У призначений для користувача інтерфейс на кнопках відобразити тільки значки

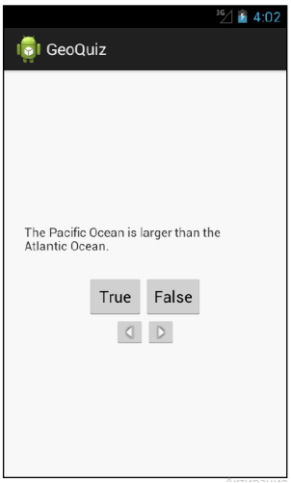

Рис. 2.10. Кнопки тільки зі значками

Для цього обидва віджета повинні відноситися до типу ImageButton (замість звичайного Button).

Віджет ImageButton є похідним від ImageView - на відміну від віджета Button, похідного від TextView.

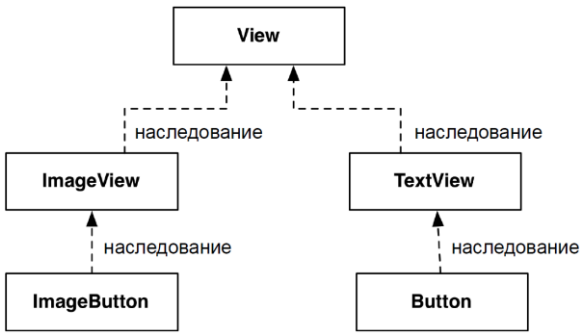

Рис. 2.11. Діаграма успадкування ImageButton і Button

Атрибути text і drawable кнопки Next можна замінити одним атрибутом ImageView:

```
<Button ImageButton
   android: id = "@ + id / next_button"
   android: layout_width = "wrap_content"
   android: layout_height = "wrap_content"
   android: text = "@ string / next_question_button"
   android: drawableRight = "@ drawable / arrow_right"
   android: drawablePadding = "4dp"
   android: src = "@ drawable / arrow_right"
/>
```

Внести зміни в QuizActivity, щоб цей клас працював з ImageButton.

Після того як ви заміните кнопки кнопками ImageButton, Eclipse видасть попередження про відсутність атрибуту android: contentDescription. Цей атрибут забезпечує доступність контенту для читачів з ослабленим зором. Рядок, заданий цьому атрибуту, читається екранним диктором (при включенні відповідних налаштувань в системі користувача).

Додайте атрибут android: contentDescription в кожен елемент Image-Button.

Лабораторна робота № 2

Тема: Створення макетів і віджетів

Оновлення Crime

Відкрийте файл Crime.java і додайте два нових поля. Поле Date представляє дату, а поле mSolved.

Лістинг 1. Додавання полів у клас Crime (Crime.java)

```
public class Crime {
   private UUID mId;
   private String mTitle;
   private Date mDate;
   private boolean mSolved;

   public Crime () {
      mId = UUID.randomUUID ();
      mDate = new Date ();
   }
   ...
}
```

Ініціалізація змінної Date конструктором Date за замовчуванням присвоює mDate поточну дату.

Потім згенеруйте get- і set-методи для своїх нових полів (Source > Generate Getters and Setters ...).

Лістинг 2. Згенеровані get- і set-методи (Crime.java)

```
public class Crime {
   ...
   public void setTitle (String title) {
      mTitle = title;
   }

   public Date getDate () {
      return mDate;
   }
   public void setDate (Date date) {
      mDate = date;
   }
   public boolean isSolved () {
      return mSolved;
   }
   public void setSolved (boolean solved) {
      mSolved = solved;
   }
}
```

Наші наступні дії - оновлення макета в fragment_crime.xml новими віджетами і їх зв'язування з віджетами в CrimeFragment.java.

Оновлення макета

Ось як буде виглядати представлення CrimeFragment.

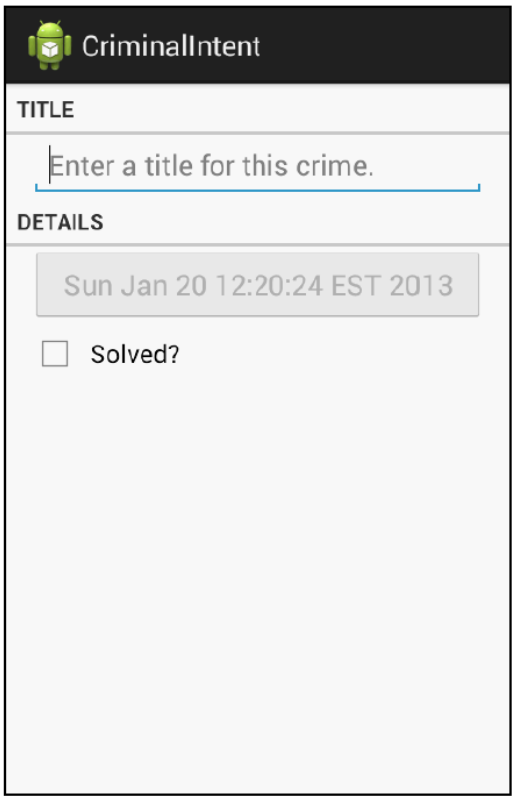

Рис. 1. CrimIntent, епізод 2

Щоб дістатися до цього екрану, ми додамо в макет CrimeFragment чотири віджета: два віджета TextView, Button і CheckBox.

Відкрийте файл fragment_crime.xml і внесіть зміни, представлені в лістингу 3.

Лістинг 3. Додавання нових віджетів (fragment_crime.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns: android = "http://schemas.android.com/apk/res/android"
   android: layout_width = "match_parent"
   android: layout_height = "wrap_content"
   android: orientation = "vertical"
   >
   <TextView
      android: layout_width = "match_parent"
      android: layout_height = "wrap_content"
      android: text = "@ string / crime_title_label"
      style = "? android: listSeparatorTextViewStyle"
   />
   <EditText android: id = "@ + id / crime_title"
      android: layout_width = "match_parent"
      android: layout_height = "wrap_content"
      android: layout_marginLeft = "16dp"
      android: layout_marginRight = "16dp"
      android: hint = "@ string / crime_title_hint"
   />
   <TextView
      android: layout_width = "match_parent"
      android: layout_height = "wrap_content"
      android: text = "@ string / crime_details_label"
      style = "? android: listSeparatorTextViewStyle"
   />
   <Button android: id = "@ + id / crime_date"
      android: layout_width = "match_parent"
      android: layout_height = "wrap_content"
      android: layout_marginLeft = "16dp"
```

```
      android: layout_marginRight = "16dp"
   />
   <CheckBox android: id = "@ + id / crime_solved"
      android: layout_width = "match_parent"
      android: layout_height = "wrap_content"
      android: layout_marginLeft = "16dp"
      android: layout_marginRight = "16dp"
      android: text = "@ string / crime_solved_label"
   />
</ LinearLayout>
```

Для віджета Button ми не вказали атрибут android:text. Кнопка буде виводити дату, що зберігається в Crime, а її текст буде задаватися в коді.

Поверніться до файлу res/values/strings.xml і додайте необхідні строкові ресурси.

Лістинг 4. Додавання строкових ресурсів (strings.xml)

```
<Resources>
   <String name = "app_name"> CrimIntent </ string>
   <String name = "title_activity_crime"> CrimeActivity </ string>
   <String name = "crime_title_hint"> Enter a title for this crime. </ String>
   <String name = "crime_title_label"> Title </ string>
   <String name = "crime_details_label"> Details </ string>
   <String name = "crime_solved_label"> Solved? </ String>
</ Resources>
```

Збережіть файли і перевірте можливі помилки.

Підключення віджетів

Зміна стану CheckBox повинна призводити до оновлення поля mSolved класу Crime.

Від Button поки що потрібно тільки виведення дати з поля mDate.

У файлі CrimeFragment.java додайте дві змінні екземпляра.

Лістинг 5. Додавання змінних екземпляра для віджетів (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;
    @Override
    public void onCreate (Bundle savedInstanceState) {
    ...
```

Виконайте організацію імпорту, щоб дозволити посилання на DatePicker і CheckBox.

Потім в onCreateView (...) отримаєте посилання на нову кнопку, задайте в тексті кнопки дату і заблокуйте її.

Лістинг 6. Призначення тексту Button (CrimeFragment.java)

```
@Override
public View onCreateView (LayoutInflater inflater, ViewGroup parent,
      Bundle savedInstanceState) {
   View v = inflater.inflate (R.layout.fragment_crime, parent, false);
   ...

   mTitleField.addTextChangedListener (new TextWatcher () {
      ...
   });
```

```
   mDateButton = (Button) v.findViewById (R.id.crime_date);
   mDateButton.setText (mCrime.getDate (). toString ());
   mDateButton.setEnabled (false);

   return v;
}
```

Блокування кнопки гарантує, що вона не буде реагувати на натискання. Також при цьому змінюється оформлення кнопки, щоб повідомити користувачеві про заблокований стан.

CheckBox: ми отримуємо посилання і призначаємо слухача, який буде оновлювати поле mSolved об'єкта Crime.

Лістинг 7. Призначення слухача для змін CheckBox (CrimeFragment.java)

```
...
mDateButton = (Button) v.findViewById (R.id.crime_date);
mDateButton.setText (mCrime.getDate (). toString ());
mDateButton.setEnabled (false);

mSolvedCheckBox = (CheckBox) v.findViewById (R.id.crime_solved);
mSolvedCheckBox.setOnCheckedChangeListener (new OnCheckedChangeListener () {
   public void onCheckedChanged (CompoundButton buttonView, boolean isChecked)
{
      // Призначення прапора
      mCrime.setSolved (isChecked);
   }
});

return v;
}
```

При імпортуванні інтерфейсу OnCheckedChangeListener Eclipse запропонує вибрати між інтерфейсом, визначеним у класі CompoundButton, і інтерфейсом, визначеним у класі RadioGroup. Виберіть інтерфейс CompoundButton; CheckBox є субклас CompoundButton.

Якщо ви використовуєте функцію автозаповнення, над методом onCheckedChanged (...) може відображатися анотація @Override, якої немає в лістингу 7. Для методів, визначених в інтерфейсах, анотації @Override не обов'язкові.

Запустіть CrimIntent. Переведіть стан CheckBox і відобразиться заблокована кнопка, на якій відображається поточна дата.

Атрибути макетів XML. Стилі, теми і атрибути тем

Стиль (Style) являє собою ресурс XML, який містить атрибути, що описують зовнішній вигляд і поведінку віджета. Наприклад, нижче наведений ресурс стилю, який налаштовує віджет на використання збільшеного розміру тексту.

```
<Style name = "BigTextStyle">
   <Item name = "android: textSize"> 20sp </ item>
   <Item name = "android: layout_margin"> 3dp </ item>
</ Style>
```

Ви можете створювати власні стилі. Вони додаються в файл стилів з каталогу res/values/, а посилання на них в макетах виглядають так: @ style/my_own_style.

Кожен віджет (віджети TextView з файлу fragment_crime.xml) має атрибут style, який посилається на стиль, створений Android. З цим конкретним стилем віджети TextView виглядають як роздільники списку, а береться він з теми програми. Тема (theme) являє собою набір стилів. Зі структурної точки зору тема сама є ресурсом стилю, атрибути якого посилаються на інші ресурси стилів.

Android надає платформні теми, які можуть використовуватися вашими додатками. При створенні CrimIntent майстер запропонував використовувати тему додатки Holo Light with Dark Action Bar, і ви погодилися.

Стиль з теми програми можна застосувати до віджету за допомогою посилання на атрибут теми (theme attribute reference). Саме це ми робимо в файлі fragment_crime.xml, Використовуючи значення ?Android:listSeparatorTextViewStyle. Посилання на атрибут теми наказує менеджеру ресурсів Android: «Перейди до теми програми та знайди в ній атрибут з ім'ям listSeparatorTextViewStyle. Цей атрибут вказує на інший ресурс стилю. Поклади значення цього ресурсу сюди».

Кожна тема Android включає атрибут з ім'ям listSeparatorTextViewStyle, але його визначення залежить від оформлення конкретної теми. Використання посилання на атрибут теми гарантує, що оформлення віджетів TextView буде відповідати оформленню вашого застосування.

Щільність пікселів, dp і sp

У файлі fragment_crime.xml значення атрибута margin задається в одиницях dp. Іноді значення атрибутів представлення задаються в конкретних розмірах (найчастіше в пікселах, але іноді в пунктах, міліметрах або дюймах). Найчастіше цей спосіб використовується для атрибутів розміру тексту, полів і відступів. Розмір тексту дорівнює висоті тексту в пікселах на екрані пристрою. Поля задають відстані між представленнями, а відступи задають відстань між зовнішньою межею представлення і його вмістом.

Android автоматично масштабує зображення для різної щільності пікселів екрану, використовуючи вміст каталогів drawable-ldpi, drawable-mdpi і drawable-hdpi.

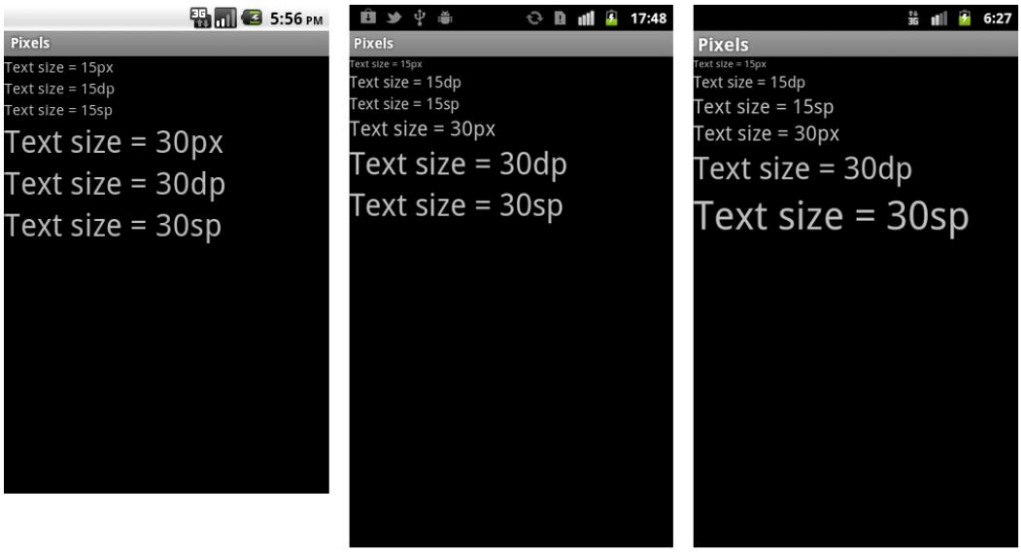

Рис. 2. Використання одиниць пристрою на TextView (зліва: MDPI; середині: HDPI; справа: HDPI з великим текстом)

В Android підтримуються одиниці, які не залежать від щільності пікселів; використовуючи їх, можна отримати однакові розміри на екранах з різною щільністю. Android перетворює ці одиниці в пікселі під час виконання.

– **dp** (Або dip) - скорочення від «density-independent pixel» (пікселі, які не залежать від щільності). Зазвичай ці одиниці використовуються для полів, відступів і всього іншого, для чого зазвичай задаються розміри в пікселах. На екранах з більш високою щільністю одиниці dp розгортаються в більшу кількість екранних пікселів. Одна одиниця dp завжди дорівнює 1/160 дюйма на екрані пристрою. Розмір буде однаковим незалежно від щільності пікселів.

- **sp** - скорочення від «scale-independent pixel» (пікселі, які не залежать від масштабу). Це одиниці, які не залежать від щільності пікселів пристрою, також враховують обраний користувачем розмір шрифту. Одиниці sp майже завжди використовуються для призначення розміру тексту.
- **pt, mm, in**- масштабовані одиниці (як і dp), які дозволяють задавати розміри інтерфейсних елементів в пунктах (1/72 дюйма), міліметрах або дюймах. Але не всі пристрої правильно налаштовані для правильного масштабування цих пристроїв.

На практиці майже виключно використовуються тільки одиниці dp і sp. Android перетворює ці значення в пікселі під час виконання.

Рекомендації з проектування інтерфейсів Android

Для полів в нашому прикладі в лістингу 3 використовується значення 16dp. Воно слідує рекомендації з проектування інтерфейсів Android, відомої як «ритм 48dp».

Сучасні програми Android повинні відповідати цим рекомендаціям, наскільки це можливо. Рекомендації значною мірою залежать від нової функціональності Android SDK, яка не завжди доступна або легко можна реалізувати на старих пристроях. Деякі рекомендації можуть дотримуватися з використанням бібліотеки підтримки, але для багатьох потрібні бібліотеки сторонніх розробників - такі, як ActionBar.

Параметри макета

Деякі імена атрибутів починаються з layout_(android:layout_marginLeft), а у інших атрибутів цього префікса немає (android:text).

Атрибути, імена яких не починаються з layout_, є рекомендаціями для віджетів. При заповненні віджет викликає метод для налаштування своєї конфігурації на підставі цих атрибутів і їх значень.

Якщо ім'я атрибута починається з layout_, то цей атрибут є директивою для батьків цього віджета. Такі атрибути, звані параметрами макета, повідомляють батьківського макету, як слід розташувати дочірній елемент всередині батька.

Навіть якщо об'єкт макета (наприклад, LinearLayout) є кореневим елементом, він все одно залишається віджетом, які мають батьків, і у нього є параметри макета. Визначаючи елемент LinearLayout в файлі fragment_crime.xml, ми задали атрибути android:layout_width і android: layout_height. Ці атрибути будуть використовуватися батьківським макетом LinearLayout при заповненні. В даному випадку параметри макета LinearLayout будуть використовуватися елементом FrameLayout в поданні вмісту CrimeActivity.

Поля і відступи

У файлі fragment_crime.xml віджетів призначаються атрибути margin і padding. Атрибути margin є параметрами макета; вони визначають відстань між віджетами. Так як віджет має інформацію тільки про самого себе, за дотримання полів відповідає батько віджета.

Навпаки, відступ не є параметром макета. Атрибут android:padding повідомляє віджету, з яким перевищенням розміру вмісту він повинен промальовувати себе. Припустимо, ви хочете помітно збільшити розміри кнопки дати без зміни розміру тексту. Додайте в Button наступний атрибут, збережіть макет і запустіть додаток заново.

Лістинг 8. Призначення відступів

```
<Button android: id = "@ + id / crime_date"
   android: layout_width = "match_parent"
   android: layout_height = "wrap_content"
   android: layout_marginLeft = "16dp"
   android: layout_marginRight = "16dp"
```

```
android: padding = "80dp"
/>
```

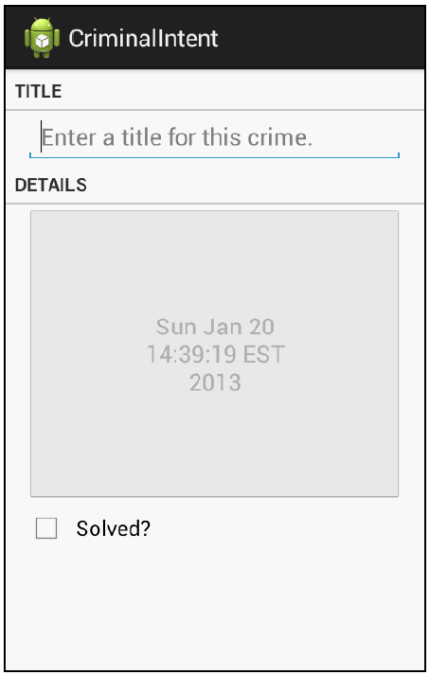

Рис. 3. Велика кнопка

Використання графічного конструктора

До теперішнього моменту ми створювали макети, вводячи розмітку XML. Ми використовуємо графічний конструктор для побудови альтернативного альбомного макета CrimeFragment.

Більшість вбудованих класів макетів - таких, як LinearLayout - автоматично змінюють розміри себе і своїх нащадків при поворотах. Однак в деяких випадках зміна розмірів за замовчуванням недостатньо ефективно використовує вільний простір.

Відкрийте програму CrimIntent і поверніть пристрій, щоб побачити макет CrimeFragment альбомного формату.

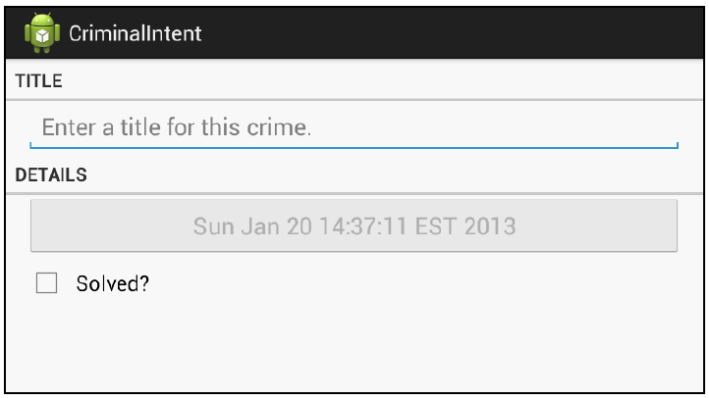

Рис. 4. CrimeFragment альбомного формату

Кнопка дати стає занадто довгою; було б краще, якби в альбомної орієнтації кнопка і прапорець розташовувалися поруч один з одним. Для цього створіть каталог res/layout-land (клацніть правою кнопкою миші на каталозі res/ на панелі Package Explorer і виберіть команду New > Folder), після чого скопіюйте файл fragment_crime.xml в res/layout-land/.

Щоб внести зміни в графічному конструкторі, спочатку закрийте файл res/layout/fragment_crime.xml, якщо він відкритий у редакторі. Відкрийте файл res/layout-land/fragment_crime.xml і виберіть вкладку Graphical Layout.

В середині графічного конструктора макета в області попереднього перегляду відображається вже знайомий альбомний макет. Зліва знаходиться палітра. Панель містить всі необхідні віджети, впорядковані за категоріями.

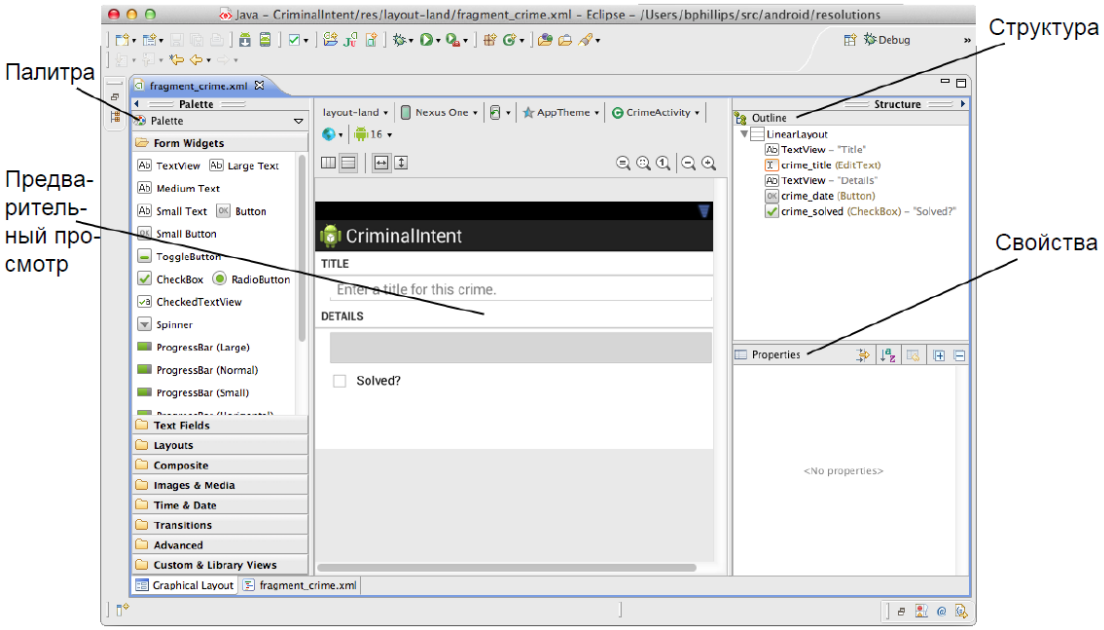

Рис. 5. Графічний конструктор макета

Праворуч від області попереднього перегляду знаходиться *панель структури*. На ній показана ієрархія віджетів в макеті.

Під панеллю структури знаходиться *панель властивостей*. На ній можна переглядати і редагувати атрибути віджета, виділеного на панелі структури.

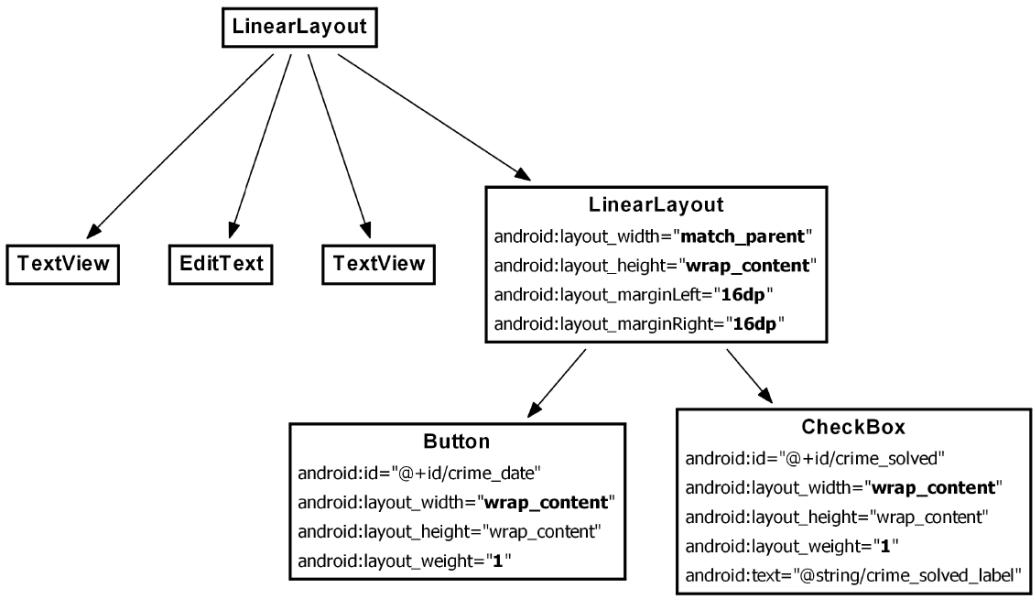

Рис. 6. Альбомний макет для CrimeFragment

Зміни, які слід внести в цей макет можна розділити на чотири групи:

– Додавання міні-LinearLayout в макет.
– Редагування атрибутів LinearLayout.

– Призначення віджетів Button і CheckBox нащадками LinearLayout.
– Оновлення параметрів макета Button і CheckBox.

Додавання нового віджету

Щоб додати віджет, можна виділити його в палітрі і перетягнути на панель структури. Клацніть на категорії Layouts в палітрі. Виберіть пункт LinearLayout (Horizontal) і перетягніть на панель структури. Відпустіть об'єкт, що перетаскується LinearLayout на корінному елементі LinearLayout, щоб зробити його прямим нащадком кореневого елемента LinearLayout.

Віджети також можна додавати перетягуванням з палітри в область попереднього перегляду. Однак віджети Layout часто порожні або закриті іншими представленнями, тому може бути важко зрозуміти, де слід розмістити віджет в області попереднього перегляду для отримання потрібної ієрархії.

Перетягування на панель структури істотно спрощує виконання цієї операції.

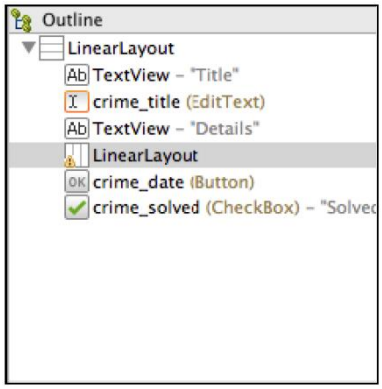

Рис. 7. Додавання міні-LinearLayout в файл fragment_crime.xml

Редагування атрибутів у властивостях

Виберіть новий віджет LinearLayout на панелі структури, щоб відобразити його атрибути на панелі властивостей. Розгорніть категорію Layout Parameters, потім категорію Margins.

Необхідно, щоб поля нового віджету Linear-Layout збігалися з полями інших віджетів. Виділіть поле поруч з Left і введіть значення 16dp. Зробіть те ж саме для правого (Right) поля.

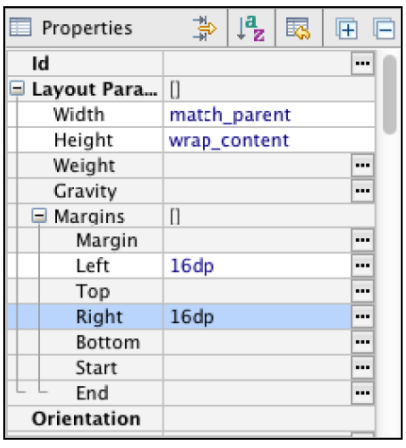

Рис. 8. Призначення полів на панелі властивостей

(На деяких платформах при спробі ввести дані в ці поля на екрані з'являється тимчасове вікно. Це обхідне рішення відомої помилки ... яке, на жаль, не дозволяє ввести значення. Якщо це

станеться, збережіть файл, перейдіть на розмітку XML і скопіюйте два атрибути margin з EditText в LinearLayout).

Збережіть файл макета і переключіться на розмітку XML за допомогою вкладки fragment_crime.xml в нижній частині області попереднього перегляду. Ви побачите тільки що доданий елемент LinearLayout з атрибутами полів.

Реорганізація віджетів на панелі структури

Наступний крок - призначення віджетів Button і CheckBox нащадками нового віджету LinearLayout. Поверніться до графічного конструктору, виберіть на панелі структури віджет Button і перетягніть його на LinearLayout.

На панелі структури відображається той факт, що Button тепер є нащадком нового віджету LinearLayout. Виконайте те ж саме з CheckBox.

Якщо нащадки розміщуються не в тому порядку, їх можна змінити таким чином за допомогою перетягування. Також на панелі структури можна видаляти віджети з макета, але будьте обережні: видалення віджета призводить до видалення його нащадків.

В області попереднього перегляду віджет CheckBox ніхто не бачить - його приховує Button. Віджет LinearLayout перевірив ширину (match_parent) свого першого нащадка (Button) і виділив йому весь простір, нічого не залишивши Check-Box (рис. 10).

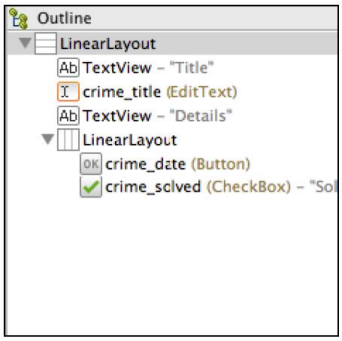

Рис. 9. Віджети Button і CheckBox тепер є нащадками LinearLayout

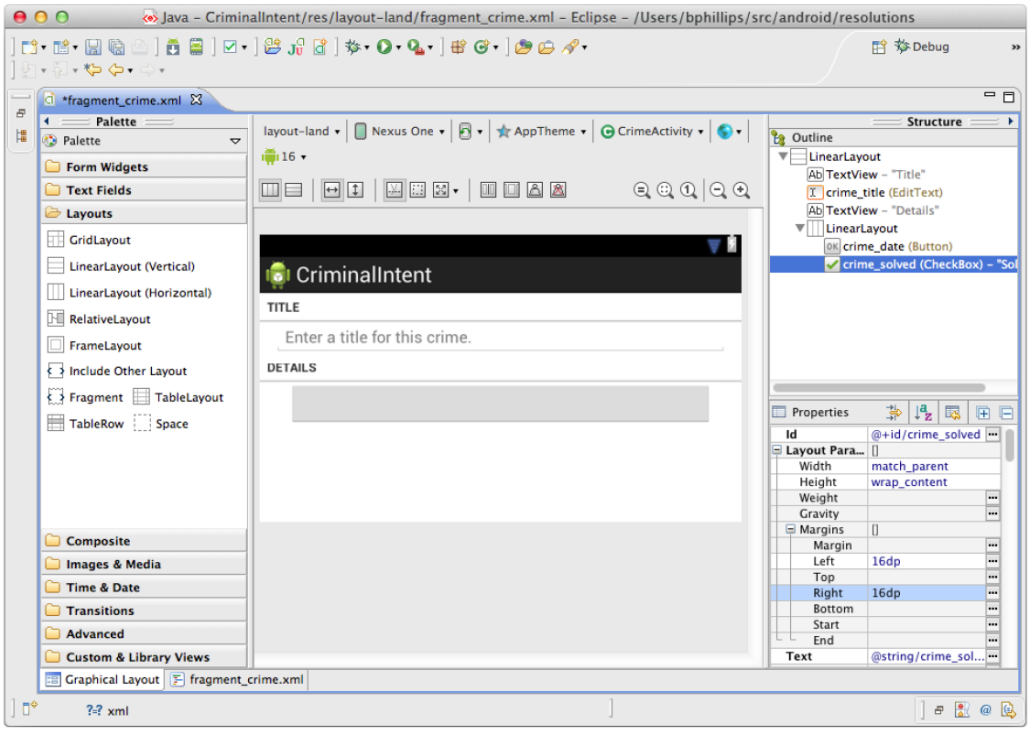

Рис. 10. Віджет Button, який був визначений першим, приховує CheckBox

Щоб відновити рівноправність нащадків LinearLayout, ми змінимо параметри макета нащадків.

Оновлення параметрів макета нащадків

Спочатку виділіть кнопку дати на панелі структури. На панелі властивостей клацніть на поточному значенні Width і замініть його значенням wrap_content.

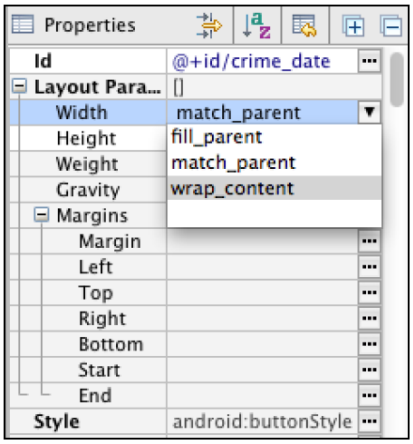

Рис. 11. Ширині кнопки задається значення wrap_content

Видаліть обидва значення 16dp полів кнопки. Тепер, коли кнопка перебуває всередині LinearLayout, поля їй не потрібні.

Знайдіть поле Weight в розділі Layout Parameters і задайте йому значення 1. Це поле відповідає атрибуту android:layout_weight на рис. 6.

Виберіть віджет CheckBox на панелі структури і внесіть ті ж зміни:

– атрибут Width повинен містити wrap_content,
– атрибут Weight - 1,
– а атрибути полів повинні бути порожніми.

В області попереднього перегляду переконайтеся в тому, що обидва віджета тепер видно. Збережіть файл і поверніться до XML, щоб підтвердити зміни. У лістингу 9 приведена відповідна розмітка XML.

Лістинг 9. Розмітка XML макета, створеного в графічному конструкторі (layout-land/ fragment_crime.xml)

```
...
<TextView
   android: layout_width = "match_parent"
   android: layout_height = "wrap_content"
   android: text = "@ string / crime_details_label"
   style = "? android: listSeparatorTextViewStyle"
   />
<LinearLayout
   android: layout_width = "match_parent"
   android: layout_height = "wrap_content"
   android: layout_marginLeft = "16dp"
   android: layout_marginRight = "16dp">
       <Button
           android: id = "@ + id / crime_date"
           android: layout_width = "wrap_content"
           android: layout_height = "wrap_content"
           android: layout_weight = "1" />
       <CheckBox
           android: id = "@ + id / crime_solved"
```

```
            android: layout_width = "wrap_content"
            android: layout_height = "wrap_content"
            android: layout_weight = "1"
            android: text = "@ string / crime_solved_label" />
        </ LinearLayout>
</ LinearLayout>
```

Як працює android: layout_weight

Атрибут ваги android:layout_weight повідомляє віджету LinearLayout, як він повинен розподілити нащадків за розміром контейнера. Обом віджеттам задані однакові значення ширини, але це не гарантує, що вони будуть мати однакову ширину на екрані. Для визначення ширини дочірніх представлень LinearLayout використовує комбінацію параметрів layout_width і layout_weight.

LinearLayout обчислює ширину представлення в два проходи. На першому проході LinearLayout перевіряє значення layout_width (або layout_height для вертикальної орієнтації). Значення layout_width як для Button, так і для CheckBox тепер дорівнює wrap_content, так що кожному представленню виділяється місце, достатнє лише для його промальовування (рис. 12).

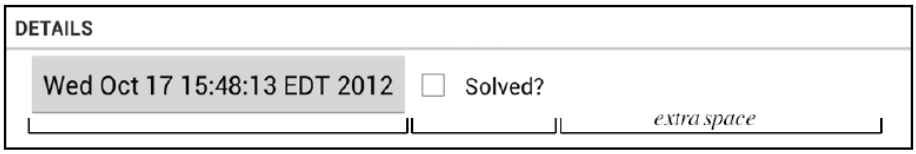

Рис. 12. Прохід 1: розподіл простору на підставі layout_width

На наступному проході LinearLayout розподіляє додатковий простір на підставі значень layout_weight (рис. 13).

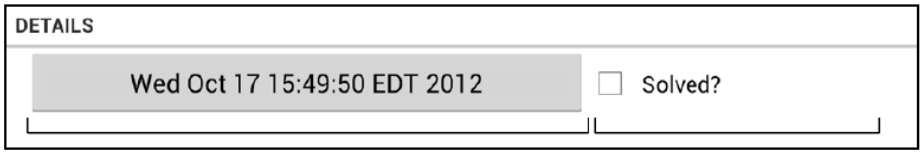

Рис. 13. Прохід 2: розподіл додаткового простору на підставі layout_weight

У нашому макеті Button і CheckBox мають однакові значення layout_weight, тому додатковий простір розподіляється в співвідношенні 50/50. Якщо задати ваговий коефіцієнт Button рівним 2, то їй буде виділено 2/3 додаткового простору, а CheckBox дістанеться всього 1/3 (рис. 14).

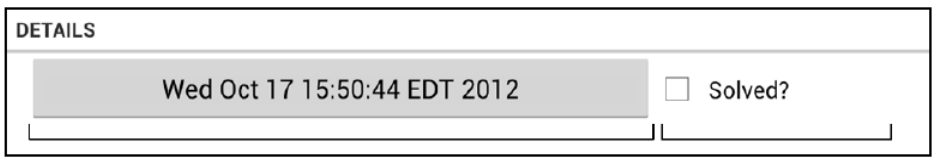

Рис. 14. Нерівномірний розподіл додаткового простору
з пропорцією layout_weight 2:1

В якості вагового коефіцієнта може використовуватися будь-яке дійсне число. Програмісти використовують різні системи позначень ваг. У файлі fragment_crime.xml використовується система «рецепт коктейлю». Також часто застосовуються набори ваг, сума яких становить 1.0 або 100; в цьому випадку вага кнопки в наведеному прикладі складе 0.66 або 66 відповідно.

Що якщо ви хочете, щоб віджет LinearLayout виділяв для кожного представлення рівно 50% своєї ширини? Просто пропустіть перший прохід, задавши атрибуту layout_width кожного віджета значення 0dp замість wrap_content. В цьому випадку LinearLayout приймає рішення тільки на підставі значень layout_weight (рис. 15).

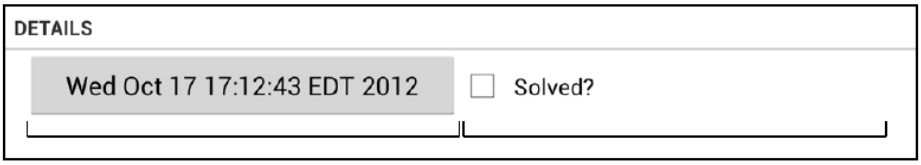

Рис. 15. При layout_width = "0dp" враховуються тільки значення layout_weight

Графічний конструктор макетів

Графічний конструктор макетів зручний, і Android удосконалює його з кожним випуском ADT. Однак часом він працює повільно і ненадійно і тоді простіше використовувати пряме введення розмітки XML. Ви можете перемикатися між внесенням змін в графічному конструкторі і в XML (для надійності не забудьте зберегти файл перед перемиканням).

Надалі, коли буде потрібно створити макет, ми будемо приводити діаграму на зразок рис. 6. Ви самі зможете вирішити, як створити її - у вигляді розмітки XML, в графічному конструкторі або поєднанням цих двох способів.

Ідентифікатори віджетів і множинні макети

Два макета, створені для CrimIntent, відрізняються незначно, але в деяких ситуаціях можливі більш серйозні розбіжності. У таких випадках перед зверненням до віджету в коді необхідно переконатися в тому, що він дійсно існує.

Якщо віджет присутній в одному макеті і відсутній в іншому, то для перевірки його наявності в поточній орієнтації перед викликом методів слід використовувати перевірку null:

```
Button landscapeOnlyButton = (Button) v.findViewById
(R.id.landscapeOnlyButton);
if (landscapeOnlyButton! = null) {
   // Операції
}
```

Для того щоб ваш код міг знайти віджет, останній повинен мати однакові значення атрибута android:id у всіх макетах, в яких він присутній.

Завдання 1. Форматування дати

Об'єкт Date більше нагадує тимчасову мітку (timestamp), ніж традиційну дату. При виклику toString() для Date ви отримуєте саме тимчасову мітку, яка відображається на кнопці. Тимчасові місця добре підходять для звітів. На кнопці виводити дату в форматі (наприклад, «Oct 12, 2018»). Для цього можна скористатися екземпляром класу android.text.format.DateFormat.

Використовуйте методи класу DateFormat для формування рядка в стандартному форматі або ж підготуйте власний форматний рядок.

Створіть форматний рядок для виведення дня тижня («Tuesday, Oct 12, 2018»).

Завдання 2. Виведення версії побудови

Додайте в макет GeoQuiz віджет TextView для виведення рівня API пристрою, на якому працює програма.

Задати текст TextView в макеті неможливо, тому що версія побудови пристрою невідома до моменту виконання. Використовуйте метод TextView для завдання тексту. Вам потрібен метод, який одержує один аргумент - рядок (або CharSequence).

Рис. 6.6. Результат завдання

Для настройки розміру і гарнітури тексту використовуйте атрибути XML, перераховані в описі TextView.

Лабораторна робота № 3

Тема: Робота з діалоговими вікнами

Діалогові вікна вимагають уваги користувача і введення даних. Зазвичай вони використуються для прийняття рішень або відображення важливої інформації. Для прикладу ми додамо діалогове вікно, в якому користувач може змінити дату запису.

При натисканні кнопки дати в CrimeFragment відкривається діалогове вікно, показане на рис. 1.

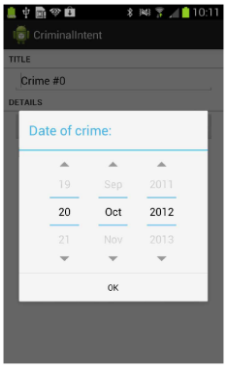

Рис. 1.Діалогове вікно для вибору дати

Діалогове вікно на рис. 1 є екземпляром AlertDialog - субкласа Dialog. Саме цей багатоцільовий субклас Dialog ви будете найчастіше використовувати в своїх програмах.

Екземпляр AlertDialog на рис. 1 упакований в екземпляр DialogFragment - субкласа Fragment. Екземпляр AlertDialog може відображатися без DialogFragment, але Android так чинити не рекомендує. Управління діалоговим вікном з FragmentManager відкриває більше можливостей для його відображення.

Крім того, «мінімальний» екземпляр AlertDialog зникне при повороті пристрою. З іншого боку, якщо екземпляр AlertDialog упакований в фрагмент, після повороту діалогове вікно буде створено заново і з'явиться на екрані.

Для додатка CrimIntent ми створимо субклас DialogFragment з ім'ям DatePickerFragment. У коді DatePickerFragment створюється і налаштовується екземпляр AlertDialog, що відображає віджет DatePicker. В якості хоста DatePickerFragment використовується екземпляр CrimePager Activity.

На рис. 2 зображена схема цих відносин.

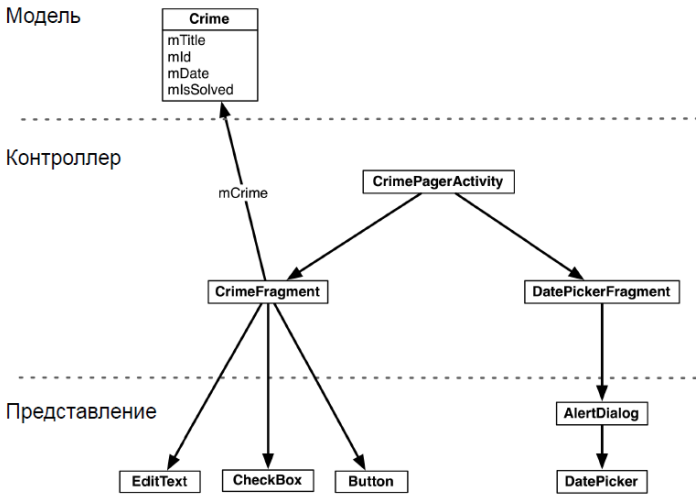

Рис. 2.Діаграма об'єктів для двох фрагментів з хостом CrimePagerActivity

Першочергові завдання:
- створення класу DatePickerFragment;
- побудова AlertDialog;
- виведення діалогового вікна на екран з використанням FragmentManager.

Додайте строковий ресурс (лістинг 1).

Лістинг 1. Додавання рядка для заголовка діалогового вікна (values/strings.xml)

```
<resources>
    ...
    <string name="crime_solved_label">Solved?</string>
    <string name="crimes_title">Crimes</string>
    <string name="date_picker_title">Date of crime:</string>
</resources>
```

Створення DialogFragment

Створіть новий клас з ім'ям DatePickerFragment і призначте його суперкласом версію DialogFragment з бібліотеки підтримки: android.support.v4.app.DialogFragment.

Клас DialogFragment містить наступний метод:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

Екземпляр FragmentManager активності-хоста викликає цей метод в процесі виведення DialogFragment на екран.

Додайте в файл DatePickerFragment.java реалізацію onCreateDialog (...), яка створює AlertDialog з заголовком і однією кнопкою OK.

Лістинг 2. Створення DialogFragment (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

В цій реалізації використовується клас AlertDialog.Builder, що надає динамічний інтерфейс для конструювання екземплярів AlertDialog.

Спочатку ми передаємо об'єкт Context конструктору AlertDialog.Builder, який повертає екземпляр AlertDialog.Builder.

Потім викликаються два методи AlertDialog.Builder для настройки діалогового вікна:

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

Метод setPositiveButton (...) отримує строковий ресурс і об'єкт, який реалізує DialogInterface.OnClickListener. У лістингу 2 передається константа Android для кнопки OK і null замість слухача.

Позитивна кнопка (Positive) натискається користувачем для підтвердження інформації в діалоговому вікні. В AlertDialog також можна додати ще дві кнопки: негативну (Negative) і нейтральну (Neutral). Ці позначення визначають позицію кнопок в діалоговому вікні (якщо їх

декілька). На пристроях Froyo і Gingerbread позитивною є ліва кнопка. На новіших пристроях порядок кнопок змінений, і позитивною є права кнопка).

Побудова діалогового вікна завершується викликом AlertDialog.Builder.create(), Який повертає налаштований екземпляр AlertDialog.

На цьому можливості AlertDialog і AlertDialog.Builderне вичерпані. Перейдемо до механіки виведення діалогового вікна на екран.

Відображення DialogFragment

Як і всі фрагменти, екземпляри DialogFragment знаходяться в цьому екземплярі FragmentManager активності-хоста.

Для додавання екземпляра DialogFragment в FragmentManager і виведення його на екран використовуються наступні методи екземпляра фрагмента:

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

Строковий параметр однозначно ідентифікує DialogFragment в списку FragmentManager. Вибір версії (з FragmentManager або FragmentTransaction) залежить тільки від вас - якщо передати FragmentManager, транзакція буде автоматично створена і закріплена. У нашому прикладі передається FragmentManager.

Додайте в CrimeFragment константу для мітки DatePickerFragment. Потім в методі onCreateView (...) видаліть код, який блокує кнопку дати, і призначте слухача View.OnClickListener, який відображає DatePickerFragment при натисканні кнопки дати.

Лістинг 3. Відображення DialogFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.Crimintent.crime_id";

    private static final String DIALOG_DATE = "date";
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(fm, DIALOG_DATE);
            }
        });
        mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
        ...

        return v;
    }

    ...
}
```

Запустіть додаток CrimIntent і натисніть кнопку дати, щоб діалогове вікно з'явилося на екрані. Кнопка OK закриває діалогове вікно (рис. 3).

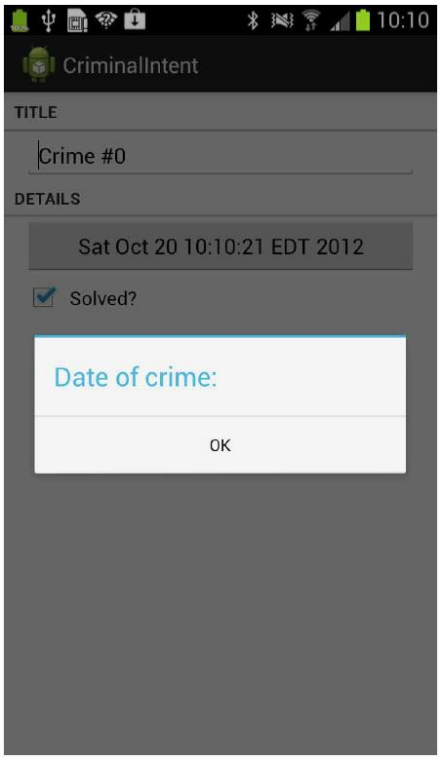

Рис. 3. AlertDialog з заголовком і кнопкою

Призначення вмісту діалогового вікна

Далі ми включимо в AlertDialog віджет DatePicker за допомогою методу AlertDialog.Builder:

```
public AlertDialog.Builder setView(View view)
```

Метод налаштовує діалогове вікно для відображення переданого об'єкта View між заголовком і кнопкою (ами).

В Package Explorer створіть новий файл макета з ім'ям dialog_date.xml і призначте його кореневим елементом DatePicker. Макет буде складатися з одного об'єктаView (DatePicker), Який ми заповнимо і передамо setView (...).

Налаштуйте макет DatePickerтак, як показано на рис. 4.

DatePicker

xmlns:android="**http://schemas.android.com/apk/res/android**"
android:id="**@+id/dialog_date_datePicker**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:calendarViewShown="**false**"

Рис. 4.Макет DatePicker (layout/dialog_date.xml)

В методі DatePickerFragment.onCreateDialog (...) заповніть представлення і призначте його діалоговому вікну.

Лістинг 4. Включення DatePicker в AlertDialog (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

Запустіть додаток CrimIntent. Натисніть кнопку дати і переконайтеся в тому, що в діалоговому вікні тепер відображається DatePicker.

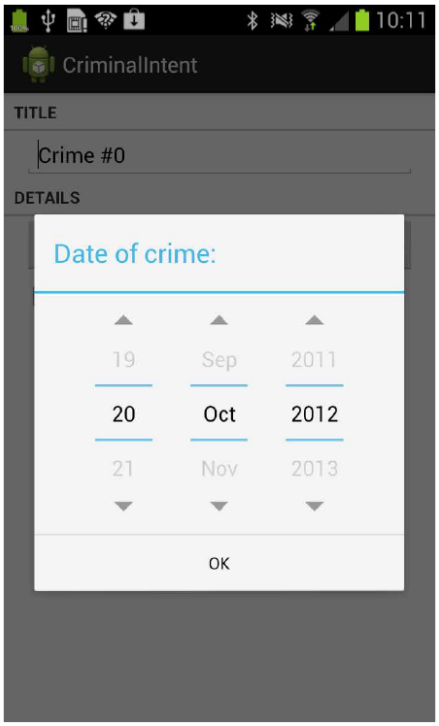

Рис. 5.AlertDialog з DatePicker

Використання макета спрощує зміни в разі зміни його вмісту. Припустимо, ви захотіли, щоб поруч з DatePicker в діалоговому вікні, відображається віджет TimePicker. При використанні заповнення можна просто оновити файл макета, і нове представлення з'явиться на екрані.

Отже, наше діалогове вікно успішно відображається.

Передача даних між фрагментами

Ми передавали дані між двома активностями; ми передавали дані між двома фрагментними активностями. Тепер потрібно передати дані між двома фрагментами, хостом яких є одна активність – CrimeFragment і DatePickerFragment (див. рис. 5.10).

Щоб передати дату запису DatePickerFragment, ми напишемо метод newInstance (Date) і зробимо об'єкт Date аргументом фрагмента.

Щоб повернути нову дату фрагменту CrimeFragment для поновлення рівня моделі і його власного представлення, ми спакуємо її як доповнення об'єкта Intent і передамо цей об'єкт Intent у виклику CrimeFragment.onActivityResult (...).

Виклик Fragment.onActivityResult (...) може здатися дивним - з урахуванням того, що активність-хост не отримує виклику Activity.onActivityResult (...) в цій взаємодії. Використання

onActivityResult (...) для передачі даних від одного фрагмента до іншого не тільки працює, але і покращує гнучкість відображення фрагмента діалогового вікна.

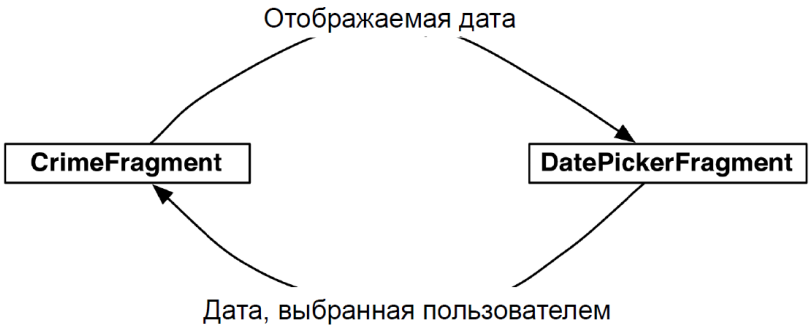

Рис. 6.Взаємодія між CrimeFragment і DatePickerFragment

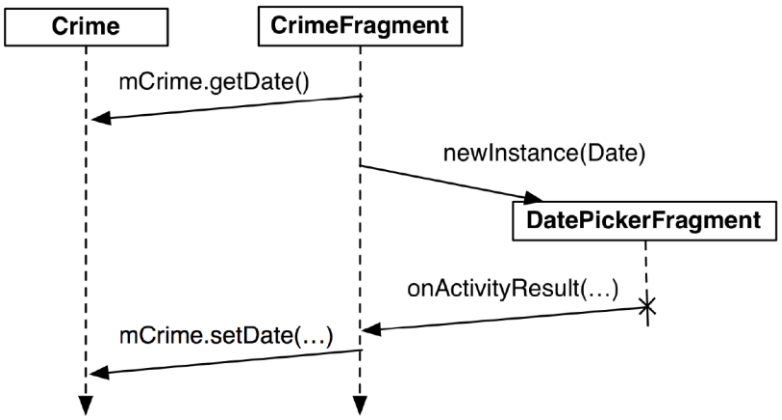

Рис. 7.Послідовність подій взаємодії між CrimeFragment і DatePickerFragment

Передача даних DatePickerFragment

Щоб отримати дані в DatePickerFragment, ми збережемо дату в пакеті аргументів DatePickerFragment, де DatePickerFragment зможе звернутися до неї.

Створення аргументів фрагмента і присвоювання їм значень зазвичай виконується в методі newInstance() конструктор фрагмента, що його замінює. Додайте в файл DatePickerFragment.java метод newInstance (Date).

Лістинг 5. Додавання методу newInstance (Date) (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    public static final String EXTRA_DATE =
        "com.bignerdranch.android.Crimintent.date";

    private Date mDate;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);

        return fragment;
    }
    ...
}
```

В класі CrimeFragment видаліть виклик конструктора DatePickerFragment і замініть його викликом DatePickerFragment.newInstance (Date).

Лістинг 6. Додавання виклику newInstance() (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater,
      ViewGroup parent, Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    updateDate();
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(fm, DIALOG_DATE);
        }
    });
    return v;
}
```

Екземпляр DatePickerFragment повинен ініціалізувати DatePicker по інформації, що зберігається в Date. Однак для ініціалізації DatePicker необхідно мати цілочисельні значення місяця, дня і року. Об'єкт Date більше нагадує тимчасову мітку і не може надати потрібні цілі значення безпосередньо.

Щоб отримати потрібні значення, слід створити об'єкт Calendar і використовувати Date для визначення його конфігурації. Після цього ви зможете отримати потрібну інформацію з Calendar.

В методі onCreateDialog (...) отримаєте об'єкт Date з аргументів і використовуйте його з Calendar для ініціалізації DatePicker.

Лістинг 7. Витяг Date і ініціалізація DatePicker (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mDate = (Date)getArguments().getSerializable(EXTRA_DATE);

    // створення об'єкта Calendar для отримання року, місяця і дня
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(mDate);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    DatePicker datePicker =
            DatePicker)v.findViewById(R.id.dialog_date_datePicker);
    datePicker.init(year, month, day, new OnDateChangedListener() {
        public void onDateChanged(DatePicker view, int year, int month, int day)
        {
            // Перетворення року, місяця і дня в об'єкт Date
            mDate = new GregorianCalendar(year, month, day).getTime();
            // оновлення аргументу для збереження
            // обраного значення при повороті
            getArguments().putSerializable(EXTRA_DATE, mDate);
        }
    });
    ...
}
```

При ініціалізації DatePicker ми також призначаємо йому слухача OnDateChangedListener. Коли користувач змінює дату в DatePicker, ми оновлюємо Date відповідно до внесених змін.

Наприкінці onDateChanged (...) об'єкт Date записується назад в аргументи фрагмента. Це робиться для того, щоб зберегти значення mDate в разі повороту. Якщо в момент повороту пристрою DatePickerFragment знаходиться на екрані, FragmentManager знищує поточний екземпляр і створює новий. При створенні нового екземпляра FragmentManager викликає для нього onCreateDialog (...), і екземпляр отримує збережену дату з аргументів. Це більш простий спосіб збереження значення, ніж збереження стану в onSaveInstanceState (...).

Зараз CrimeFragment успішно повідомляє DatePickerFragment, яку дату слід відобразити. Ви можете запустити додаток CrimIntent і переконатися в тому, що все працює так само, як раніше.

Повернення даних CrimeFragment

Щоб екземпляр CrimeFragment отримував дані від DatePickerFragment, нам необхідно якимось чином відстежувати відносини між двома фрагментами.

З активностями ви викликаєте startActivityForResult (...), а ActivityManager відстежує відносини між батьківсько. та дочірньо. активністю. Коли дочірня активність припиняє існування, ActivityManager знає, яка активність повинна отримати результат.

Призначення цільового фрагмента

Для створення аналогічної зв'язку можна призначити CrimeFragment *цільовим фрагментом* (target fragment) для DatePickerFragment. Для цього викликається наступний методFragment:

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

Метод отримує фрагмент, який стане цільовим, і код запиту, аналогічний переданому startActivityForResult (...). За кодом запиту цільовий фрагмент пізніше може визначити, який фрагмент повертає інформацію.

FragmentManager зберігає цільовий фрагмент і код запиту. Щоб отримати їх, викличте getTargetFragment() і getTargetRequestCode() для фрагмента, який призначив цільової фрагмент.

В файлі CrimeFragment.java створіть константу для коду запиту, а потім призначте Crime-Fragment цільовим фрагментом екземпляра DatePickerFragment.

Лістинг 8. Призначення цільового фрагмента (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
       "com.bignerdranch.android.Crimintent.crime_id";
    private static final String DIALOG_DATE = "date";
    private static final int REQUEST_DATE = 0;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
               FragmentManager fm = getActivity()
                  .getSupportFragmentManager();
               DatePickerFragment dialog = DatePickerFragment
                  .newInstance(mCrime.getDate());
               dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
               dialog.show(fm, DIALOG_DATE);
            }
        });
```

```
        return v;
        }
        ...
    }
```

Передача даних цільовим фрагменту

Зв'язок між CrimeFragment і DatePickerFragment створений, і тепер потрібно повернути дату CrimeFragment. Дата буде включена в об'єкт Intent як доповнення.

Який метод буде використовуватися для передачі інтенту цільовим фрагменту? Як не дивно,DatePickerFragment передасть його при виклику CrimeFragment.onActivityResult (int, int, Intent).

Метод Activity.onActivityResult (...) викликається ActivityManager для батьківської активності при знищенні дочірньої активності. При роботі з активностями ви не викликаєте Activity.onActivityResult (...) самостійно; це робить ActivityManager. Після того як активність отримає виклик, екземпляр FragmentManager активності викликає Fragment.onActivityResult (...) для відповідного фрагмента.

Якщо хостом двох фрагментів є одна активність, то для повернення даних можна скористатися методом Fragment.onActivityResult (...) і викликати його безпосередньо для цільового фрагмента. Він містить все необхідне:

- код запиту, що відповідає коду, переданому setTargetFragment (...), по якому цільовий фрагмент дізнається, хто повертає результат;
- код результату для визначення виконуваної дії;
- екземпляр Intent, який може містити додаткові дані.

В класі DatePickerFragment створіть закритий метод, який створює інтент, поміщає в нього дату як доповнення, а потім викликає CrimeFragment.onActivityResult (...). В onCreateDialog (...) замініть параметр null виклику setPositiveButton (...) реалізацією DialogInterface.OnClickListener, яка викликає закритий метод і передає код результату.

Лістинг 9. Зворотний виклик цільового фрагмента (DatePickerFragment.java)

```
private void sendResult(int resultCode) {
    if (getTargetFragment() == null)
       return;
    Intent i = new Intent();
    i.putExtra(EXTRA_DATE, mDate);

    getTargetFragment()
       .onActivityResult(getTargetRequestCode(), resultCode, i);
}

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...

    return new AlertDialog.Builder(getActivity())
       .setView(v)
       .setTitle(R.string.date_picker_title)
       .setPositiveButton(android.R.string.ok, null)
       .setPositiveButton(
           android.R.string.ok,
           new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int which) {
               sendResult(Activity.RESULT_OK);
}
})
.create();
}
```

У класі CrimeFragment перевизначите метод onActivityResult (...), щоб він повертав доповнення, ставив дату в Crime і оновлював текст кнопки дати.

Лістинг 10. Реакція на отримання даних від діалогового вікна (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        Date date = (Date)data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

Код, що задає текст кнопки, ідентичний коду з onCreateView (...). Щоб уникнути завдання тексту в двох місцях, ми інкапсулюємо цей код в закритому методі u-dateDate(), а потім викличемо його в onCreateView (...) і onActivityResult (...).

Лістинг 11. Виділення коду в метод updateDate() (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
...

    public void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }

    @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        ...
        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        updateDate();
    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if (resultCode != Activity.RESULT_OK) return;
        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
              .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
            updateDate();
        }
    }
}
```

Дані передаються туди і назад.

Запустіть додаток CrimIntent і переконайтеся в тому, що ви дійсно можете управляти датою. Змініть дату Crime; нова дата повинна з'явитися в представлення CrimeFragment. Поверніться до списку записів і перевірте дату Crime і переконайтеся в тому, що рівень моделі дійсно оновлений.

Більше гнучкості в представленні DialogFragment

Використання onActivityResult (...) для повернення даних цільовому фрагменту особливо зручно, коли ваш додаток отримує багато даних від користувача і потребує більшого простору для їх введення. При цьому програма має добре працювати на телефонах і планшетах.

На екрані телефону вільного місця не так багато, тому ви, швидше за все, використовуєте для введення даних активність з повноекранним фрагментом. Дочірня активність буде запускатися викликом startActivityForResult() з фрагмента батьківської активності. При знищенні дочірньої активності батьківська активність буде отримувати виклик onActivityResult (...), який буде перенаправлятися фрагменту, запустившему дочірню активність.

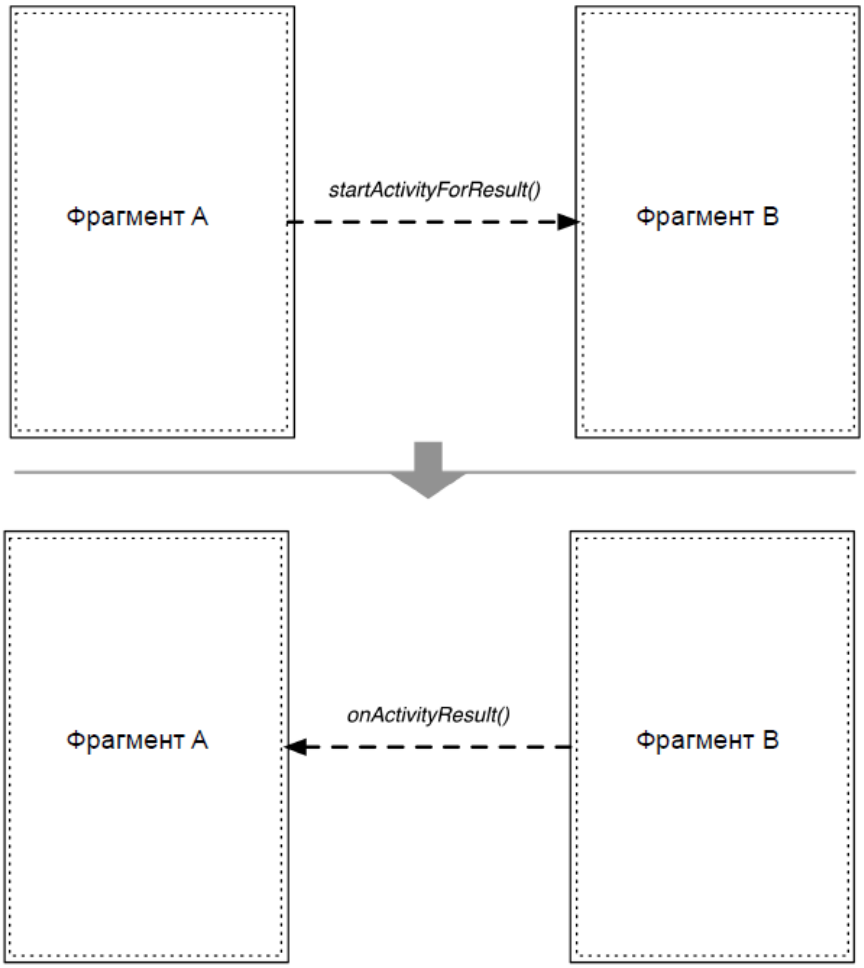

Рис. 8.Взаємодія між активностями на телефонах

На планшетах, де екранного простору більше, часто буває краще відобразити DialogFragment для введення тих же даних. В такому випадку ви задаєте цільовий фрагмент і викликаєте show (...) для фрагмента діалогового вікна. При закритті фрагмент діалогового вікна викликає для своєї мети onActivityResult (...).

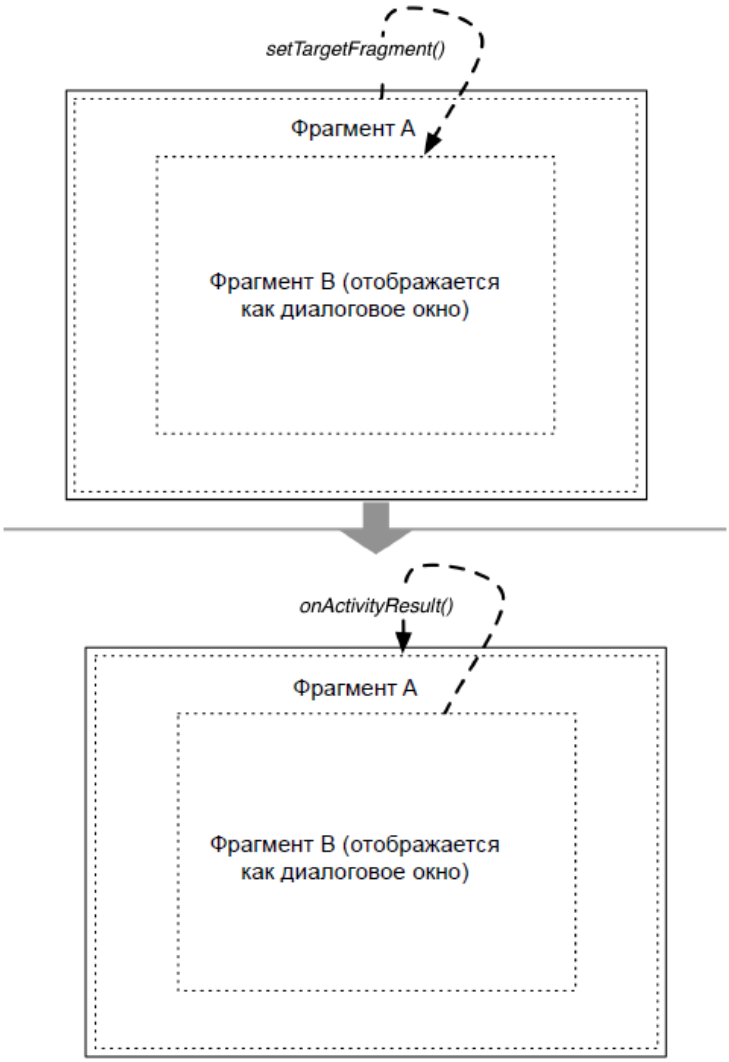

Рис. 9. Взаємодія між фрагментами на планшетах

Метод onActivityResult (...) фрагмента буде викликатися завжди, незалежно від того, чи запустив фрагмент активність або відобразив діалогове вікно. Отже, ми можемо використовувати один код для різних варіантів представлення інформації.

Коли один код використовується і для повноекранного, і для діалогового фрагмента, для підготовки виведення в обох випадках замість onCreateDialog (...) можна перевизначити DialogFragment.onCreateView (...).

Завдання. Нові діалогові вікна

Напишіть ще один діалоговий фрагмент TimePickerFragment для вибору часу запису. Використовуйте виджет TimePicker, додайте в CrimeFragment ще одну кнопку для відображення TimePickerFragment.

Спробуйте обмежитися однокнопочним інтерфейсом. Діалогове вікно, що відкривається кнопкою, має пропонувати користувачу вибрати між зміною часу і зміною дати. Після вибору на екрані повинно з'являтися друге діалогове вікно.

Лабораторна робота № 4

Тема: Відтворення звуку і MediaPlayer

MediaPlayer - клас Android для відтворення аудіо та відео. Він може відтворювати дані з різних джерел (наприклад, локальних файлів або файлів, що завантажуються з Інтернету) і в різних форматах (WAV, MP3, Ogg Vorbis, MPEG-4, 3GPP і т.д.).

Створіть новий проект з ім'ям Hello. У першому діалоговому вікно виберіть тему додатку Holo Dark.

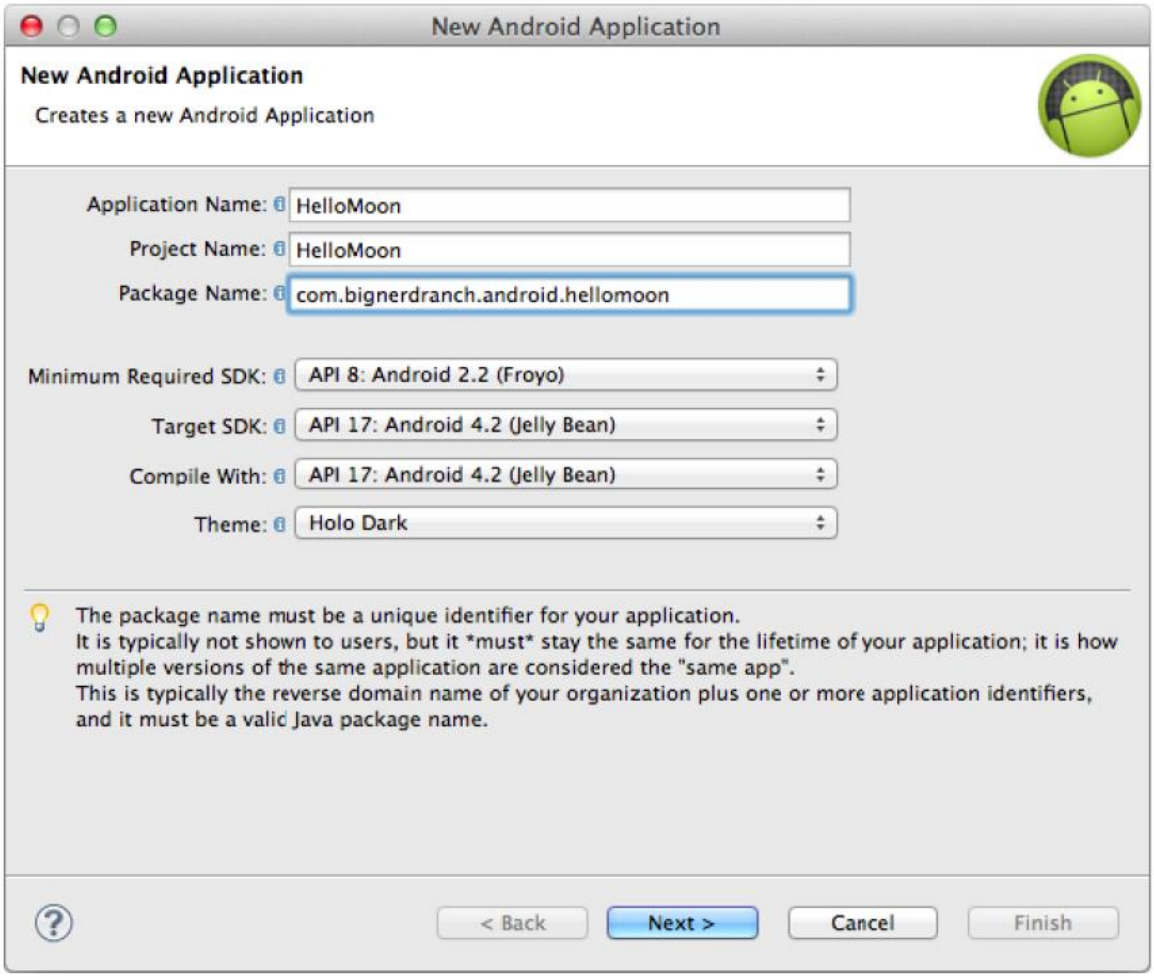

Рис. 2. Створення програми Hello з темою Holo Dark

Клацніть на кнопці Next. У додатку не буде власного значка лаунчер, і він використовує порожній шаблон активності. Накажіть шаблону створити активність з ім'ям HelloActivity.

Додавання ресурсів

Для цього простого додатка ми створили один файл armstrong_on.jpg для екранів середньої щільності (~ 160 dpi), яку Android вважає мінімальною загальною вимогою. Скопіюйте файл armstrong_on.jpg в каталог drawable-mdpi.

Аудіофайл буде знаходитися в каталозі res/raw. Каталог raw призначений для зберігання будь-яких ресурсів, які не потребують спеціальної обробки системою побудови додатків Android.

Каталог res/raw не створюється для проекту за замовчуванням, тому його доведеться додати вручну. (Клацніть правою кнопкою миші на каталозі res і виберіть команду New>Folder.)

Лістинг 1. Додавання рядків (strings.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Resources>
    <String name = "app_name"> Hello </ string>
    <String name = "hello_world"> Hello world! </ String>
    <String name = "menu_settings"> Settings </ string>
    <String name = "hello_play"> Play </ string>
    <String name = "hello_stop"> Stop </ string>
    <String name = "hello_description"> Hello! </ string>

</ Resources>
```

Необхідні ресурси готові; можна переходити до планування загальної архітектури Hello.

У додатку Hello буде використовуватися тільки одна активність HelloActivity, що є хостом для фрагмента HelloFragment.

AudioPlayer - клас, який ми напишемо для інкапсуляції MediaPlayer. Взагалі кажучи, інкапсулювати MediaPlayer не обов'язково; HelloFragment може взаємодіяти з MediaPlayer безпосередньо. Проте така архітектура робить код більш стрункім і послаблює логічні прив'язки.

Але перш ніж створювати клас AudioPlayer, ми спочатку підготуємо інші частини програми:

– визначення макета фрагмента;
– створення класу фрагмента;
– зміна активності та її макета для виконання функцій хоста фрагмента.

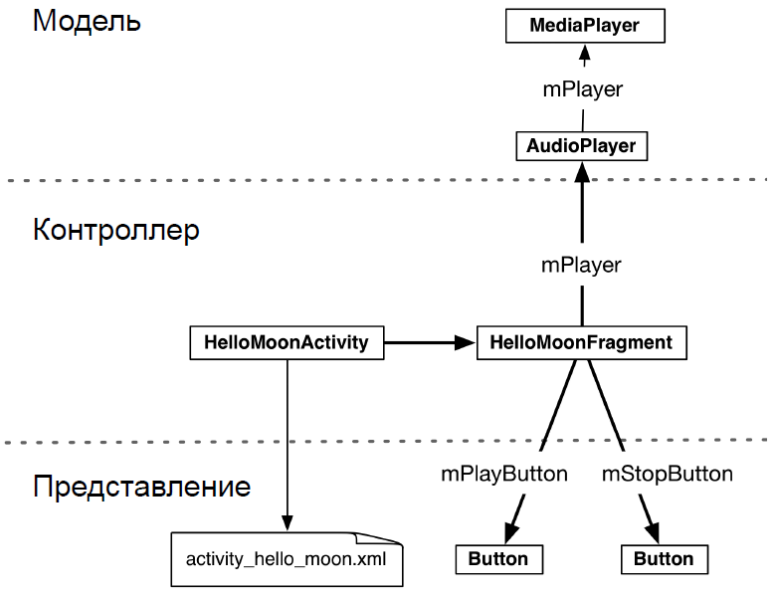

Рис. 3. Діаграма об'єктів Hello

Визначення макета HelloFragment

Створіть новий XML-файл макета Android з ім'ям fragment_hello.xml. Призначте його кореневим елементом TableLayout.

Файл fragment_hello.xml слід заповнювати по рис. 4.

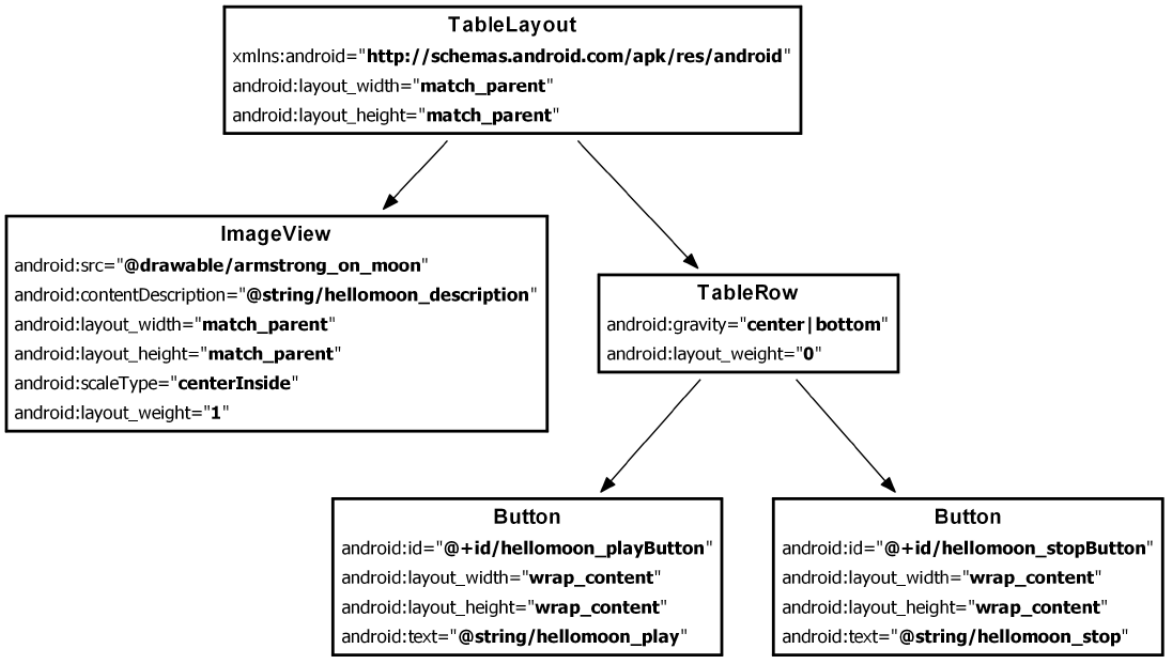

Рис. 4. Діаграма макета додатку Hello

TableLayout працює майже так само, як LinearLayout. Замість вкладення LinearLayout для впорядкування віджетів можна скористатися віджетом TableRow. Комбінація TableLayout і TableRow спрощує організацію представлень в акуратні стовпчики.

Чому ImageView не знаходиться у TableRow? Нащадки TableRow розглядаються як «осередки» таблиці. Ми хочемо, щоб віджет ImageView займав весь екран. Якби він був нащадком TableRow, то віджет TableLayout постарається розгорнути інші осередки в цьому стовпці на весь екран. А якщо зробити його прямим нащадком TableLayout, він зможе робити все, що хоче, тоді як кнопки Button залишаться в своїх стовпцях рівної ширини, розташованих по сусідству один з одним.

Врахуйте, що віджет TableRow не зобов'язаний оголошувати атрибути ширини і висоти. Він використовує ширину і висоту, а також всі інші атрибути TableLayout. З іншого боку, вкладений віджет LinearLayout здатний забезпечити більшу гнучкість у визначенні зовнішнього вигляду ваших віджетів.

Скидання теми додатку

Тема програми оголошується в елементі application маніфесту:

```
...
   <application
      android: allowBackup = "true"
      android: icon = "@ drawable / ic_launcher"
      android: label = "@ string / app_name"
      android: theme = "@ style / AppTheme">
      ...
   </ Application>
</ Manifest>
```

Атрибут android: theme не обов'язковий; якщо тема не оголошена, використовується конфігурація пристрою за умовчанням.

Задане значення є посиланням на ресурс - @style/AppTheme. На панелі Package Explorer знайдіть і відкрийте файл res/values/styles.xml. В елементі style з ім'ям AppBaseTheme замініть значення parent на android:Theme.

Лістинг 2. Зміна файлу стилів за замовчуванням (res/values/styles.xml)

```
<Style name = "AppBaseTheme" parent = "android: Theme.Light">
<Style name = "AppBaseTheme" parent = "android: Theme">
```

В каталозі res також знаходяться два каталогу values з уточненими іменами; кожен містить файл styles.xml. Уточнення в іменах каталогів позначають рівні API. Дані у файлі res/values-11/ styles.xml будуть використовуватися для API рівнів 11-13, а значення в файлі res/values-14/ styles.xml - для API рівня 14 і вище.

Відкрийте файл res/values-11/styles.xml і замініть значення атрибута parent у AppBas-eTheme на android:Theme.Holo. Ця тема повинна використовуватися на всіх пристроях з API 11 і вище, тому каталог res/values-14/ тільки заважає. Видаліть його з проекту Hello.

Збережіть файли і знову перегляньте макет. Тепер він повинен мати темний фон, який добре поєднується з графічним файлом.

Створення класу HelloFragment

Створіть новий клас з ім'ям HelloFragment і призначте його суперкласом android.support.v4.app.Fragment.

Перевизначите метод HelloFragment.onCreateView (...), щоб заповнити тільки що визначений макет і отримати посилання на кнопки.

Лістинг 3. Вихідна версія HelloFragment (HelloFragment.java)

```
public class HelloFragment extends Fragment {
    private Button mPlayButton;
    private Button mStopButton;
    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate (R.layout.fragment_hello, parent, false);
        mPlayButton = (Button) v.findViewById (R.id.hello_playButton);
        mStopButton = (Button) v.findViewById (R.id.hello_stopButton);
        return v;
    }
}
```

Використання фрагмента макету

У додатку CrimIntent хостинг фрагментів здійснювався додаванням їх в код активності. У додатку Hello замість цього буде використовуватися фрагмент макета, при використанні якого розробник задає клас фрагмента в елементі fragment.

Відкрийте файл activity_hello.xml і замініть його вміст елементом fragment, наведеним в лістингу 4.

Лістинг 4. Створення фрагмента макету (activity_hello.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Fragment xmlns: android = "http://schemas.android.com/apk/res/android"
    android: id = "@ + id / helloFragment"
    android: layout_width = "match_parent"
    android: layout_height = "match_parent"
    android: name = "com.bignerdranch.android.hello.HelloFragment">

</ Fragment>
```

Перш ніж ви зможете запустити код, необхідно внести ще одну зміну в код HelloActivity. Змініть суперклас HelloActivity - ним повинен бути клас FragmentActivity:

Лістинг 5. Перетворення HelloActivity в FragmentActivity (HelloActivity.java)

```
public class HelloActivity extends Activity FragmentActivity {
    / ** Викликається при вихідному створенні активності. * /
    @Override
    public void onCreate (Bundle savedInstanceState) {
       super.onCreate (savedInstanceState);
       setContentView (R.layout.activity_hello);
    }
}
```

Відкрийте програму Hello. На цей раз HelloActivity стає хостом представлення HelloFragment.

Коли клас викликав метод .setContentView (...) і заповнив макет activity_hello.xml, він виявив елемент fragment. У цій точці FragmentManager створив екземпляр HelloFragment і додав його до списку. Потім він викликав для HelloFragment метод onCreateView (...) і помістив представлення, повернене цим методом, в місце, підготовлене тегом fragment.

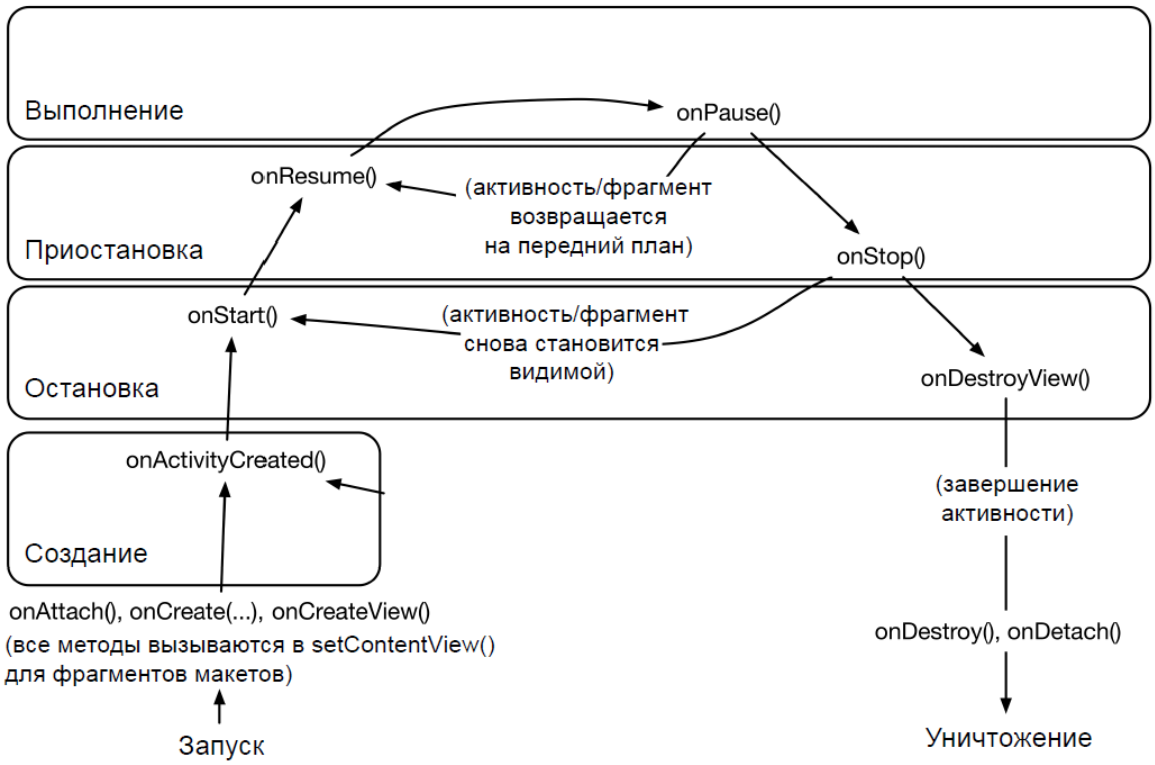

Рис. 5. Життєвий цикл фрагмента макету

Чим доводиться розплачуватися за цю простоту? Ви втрачаєте гнучкість і широту можливостей, характерні для прямої роботи з FragmentManager:

- Ви можете перевизначати методи життєвого циклу фрагмента, щоб реагувати на події, але не можете управляти тим, коли ці методи викликаються.
- Ви не можете закріплювати транзакції, які видаляють, замінюють або від'єднують фрагмент макета. Доводиться задовольнятися тим, що було зроблено при створенні активності.
- До фрагментів макетів можна приєднувати аргументи. Приєднання аргументів повинно здійснюватися після створення фрагмента і до його включення в FragmentManager. З фрагментами макетів всі ці події вам недоступні.

Однак в простому додатку або в статичній частині складного додатка використання фрагмента макету може бути цілком розумним.

Відтворення аудіо

Створіть в пакеті com.bignerdranch.android.hello новий клас з ім'ям AudioPlayer. Залиште його суперкласом java.lang.Object.

У файлі AudioPlayer.java додайте поле для зберігання екземпляру MediaPlayer і методи для зупинки і відтворення цього екземпляру.

Лістинг 6. Простий код відтворення аудіо

```
public class AudioPlayer {
    private MediaPlayer mPlayer;
    public void stop () {
       if (mPlayer! = null) {
           mPlayer.release ();
           mPlayer = null;
    }
}
public void play (Context c) {
    mPlayer = MediaPlayer.create (c, R.raw.one_small_step);
    mPlayer.start ();
}
}
```

У методі play (Context) викликається метод MediaPlayer.create (Context, int). Об'єкт Context потрібен MediaPlayer для впізнання ідентифікатора ресурсу аудіо файлу. (Також існують інші методи MediaPlayer.create (...), що використовуються при отриманні аудіо з інших джерел, наприклад з Інтернету або по локальній URI.)

У методі AudioPlayer.stop () екземпляр MediaPlayer звільняється, а полю mPlayer присвоюється null. Виклик MediaPlayer.release () знищує екземпляр.

Знищення здається занадто агресивною інтерпретацією «зупинки», але для цього є вагомі причини. Клас MediaPlayer утримує аудіообладнання та інші системні ресурси до виклику release (). Ці ресурси спільно використовуються всіма додатками. Клас MediaPlayer включає метод stop () для перекладу екземпляру MediaPlayer в зупинений стан, з якого він може бути перезапущений. Однак при простому відтворенні аудіо коректніше знищити екземпляр викликом release (), а потім створити його заново.

Просте правило: утримуйте рівно один екземпляр MediaPlayer і тільки на той час, в якому він щось відтворює.

Для дотримання цього правила ми внесемо пару змін до play (Context). Додайте вихідний виклик stop () і змусьте слухача викликати stop () при завершенні відтворення.

Лістинг 7. Тільки один екземпляр (AudioPlayer.java)

```
...
public void play (Context c) {
    stop ();
    mPlayer = MediaPlayer.create (c, R.raw.one_small_step);
    mPlayer.setOnCompletionListener (new MediaPlayer.OnCompletionListener () {
        public void onCompletion (MediaPlayer mp) {
             stop ();
        }
    });

    mPlayer.start ();
    }
}
```

Виклик stop () на початку play (Context) запобігає можливе створення декількох екземплярів MediaPlayer, якщо користувач клацне на кнопці Play повторно. Виклик stop () при завершенні відтворення файлу звільняє екземпляр MediaPlayer, як тільки він перестає використовуватися.

Також виклик AudioPlayer.stop () необхідно включити в HelloFragment, щоб екземпляр MediaPlayer не продовжував відтворення після знищення фрагмента. У класі HelloFragment перевизначите метод onDestroy () і включіть в нього виклик AudioPlayer.stop ().

Лістинг 8. Перевизначення onDestroy () (HelloFragment.java)

```
    ...
    @Override
    public void onDestroy () {
        super.onDestroy ();
        mPlayer.stop ();
    }
}
```

Клас MediaPlayer може продовжити відтворення після знищення HelloFragment, тому що MediaPlayer працює в іншому програмному потоці (thread). Зараз ми навмисно ігноруємо цей багатопоточний аспект Hello.

Підключення кнопок відтворення і зупинки

Поверніться до файлу HelloFragment.java. Створіть екземпляр класу AudioPlayer і призначте слухачів для кнопок відтворення і зупинки.

Лістинг 9. Підключення кнопки Play (HelloFragment.java)

```
public class HelloFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer ();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup parent,
          Bundle savedInstanceState) {
       View v = inflater.inflate (R.layout.fragment_hello, parent, false);
       mPlayButton = (Button) v.findViewById (R.id.hello_playButton);
       mPlayButton.setOnClickListener (new View.OnClickListener () {
          public void onClick (View v) {
             mPlayer.play (getActivity ());
          }
    });

    mStopButton = (Button) v.findViewById (R.id.hello_stopButton);
    mStopButton.setOnClickListener (new View.OnClickListener () {
       public void onClick (View v) {
             mPlayer.stop ();
       }
    });
    return v;
    }
}
```

Відкрийте програму Hello, натисніть кнопку Play.

Завдання 1. Призупинення відтворення

Надайте користувачеві можливість призупинити відтворення аудіо.

Відтворення відео

Можна вибирати з декількох варіантів відтворення відео. Можна використовувати клас MediaPlayer.

Часто оновлювані зображення (як відео) в Android відображаються на віджеті SurfaceView. Точніше, вони відображаються на віджеті Surface, хостом якого є SurfaceView. Щоб отримати доступ до Surface, слід отримати примірник SurfaceHolder для SurfaceView.

Підключення SurfaceHolder до MediaPlayer здійснюється викликом MediaPlayer.setDisplay (SurfaceHolder).

Часто для відтворення відео простіше використовувати екземпляр VideoView. Клас VideoView не взаємодіє з MediaPlayer, як SurfaceView. Однак він взаємодіє з MediaController, що дозволяє легко організувати інтерфейс відтворення.

Єдиний нюанс з використанням VideoView полягає в тому, що клас не приймає ідентифікатори ресурсів - тільки шляхи до файлів або об'єкти Uri. Щоб створити об'єкт Uri, що посилається на ресурс Android, використовуйте код такого вигляду:

```
Uri resourceUri = Uri.parse ( "android.resource: //" +
   "Com.bignerdranch.android.hello/raw/apollo_17_stroll");
```

Створіть URI зі схемою android.resource, ім'ям вашого пакета замість хоста, типом і ім'ям вашого ресурсу замість шляху. Результат може бути переданий VideoView.

Завдання 2. Відтворення відео

Змініть програму Hello так, щоб вона також дозволяла відтворювати відеоролики. Розпочніть відтворення відеоролик одним з описаних способів.

Лабораторна робота № 5

Тема: Робота з неявними інтентами

В Android можна запустити активність з іншої програми на пристрої за допомогою *неявного інтенту* (implicit intent). У явному інтенті задається клас активності, що запускається, а ОС запускає його. У неявному інтенті ви описуєте операцію, яку необхідно виконати, а ОС запускає активність відповідного додатку.

У додатку CrimIntent ми будемо використовувати неявні інтенті для вибору записів зі списку контактів користувача і відправки текстових звітів про запис. Користувач вибирає записи в контактному додатку, встановленому на пристрої, і отримує список програм для відправки звіту.

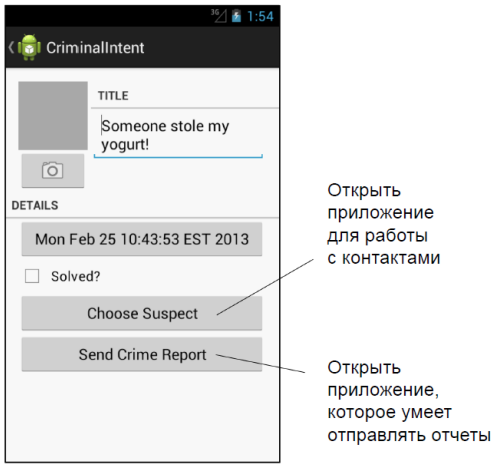

Рис. 1. Відкриття додатків для вибору контактів і відправки звітів

Використовувати функціональність інших додатків за допомогою неявних інтентів набагато простіше, ніж писати власні реалізації стандартних завдань.

Перш ніж створювати неявні інтенти, необхідно виконати з CrimIntent ряд підготовчих дій:

- додати в макети CrimeFragment кнопки вибору запису і відправки звіту;
- додати в клас Crime поле mSuspect, в якому буде зберігатися ім'я запису;
- створити звіт про запис з використанням форматних рядків ресурсів.

Додавання кнопок

Почнемо з включення в макети CrimeFragment нових кнопок. Перш за все додайте рядки, які будуть відображатися на кнопках.

Лістинг 1. Додавання рядків для написів на кнопках (strings.xml)

```
    <string name="take">Take!</string>
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
</resources>
```

Додайте в файл layout/fragment_crime.xml два віджети Button, представлених на рис. 2. На діаграмі не показаний перший віджет LinearLayout.

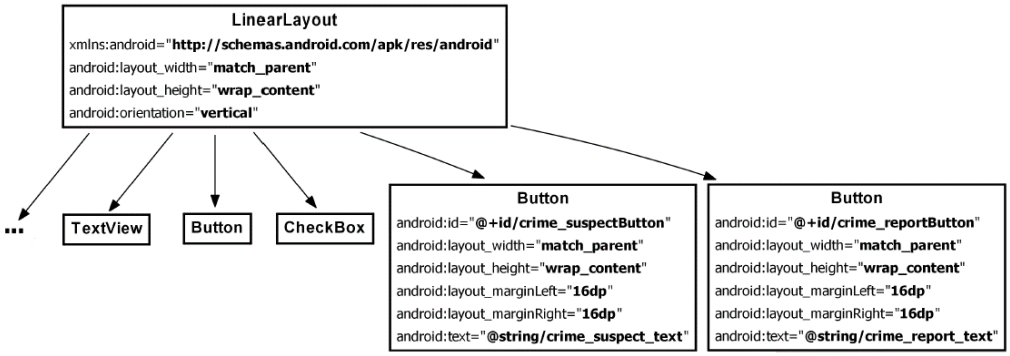

Рис. 2. Додавання кнопок для вибору контактів і відправки звітів

В альбомному макеті ми призначимо нові кнопки нащадками нового горизонтального віджета LinearLayout, розташованого під віджетом з кнопкою дати і прапорцем.

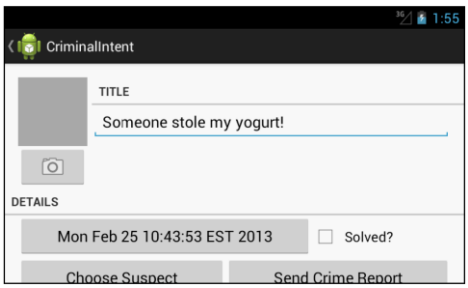

Рис. 3. Нижні кнопки частково приховані в альбомному режимі

Як видно з рис. 3, на малих екранах нові кнопки видно лише частково. Щоб вирішити цю проблему, ми розмістимо весь макет для альбомного режиму вScrollView.

Новий макет представлений на рис. 4. Так як кореневим елементом тепер є ScrollView, Перемістіть простір імен з попереднього кореневого елемента в ScrollView.

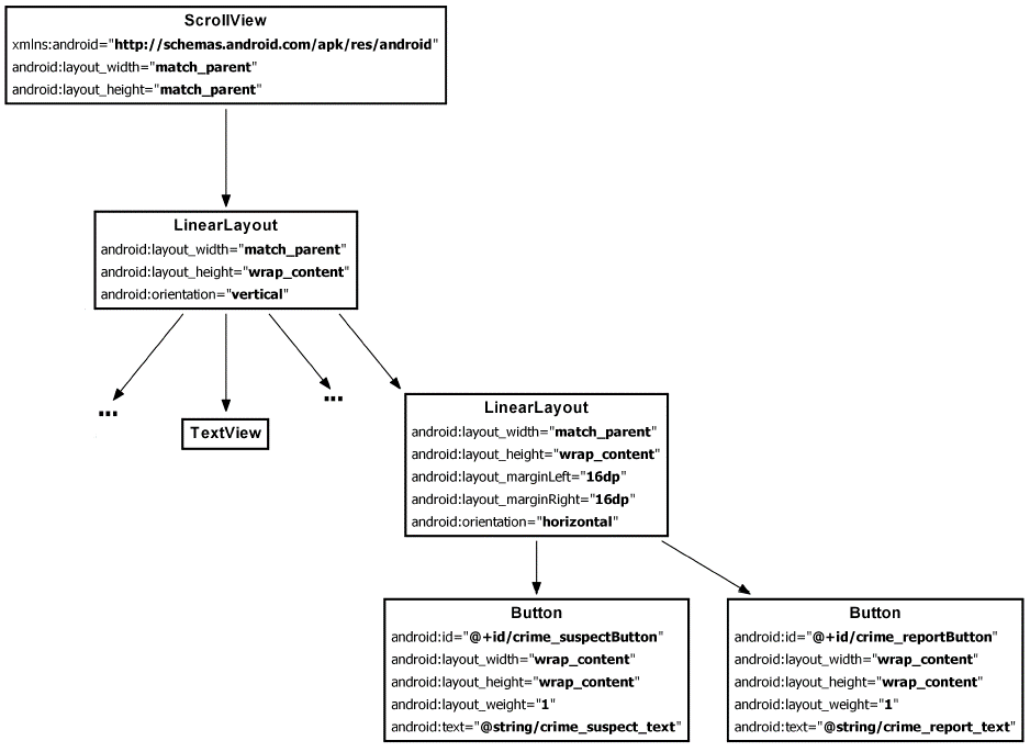

Рис. 4. Додавання кнопок для вибору контактів і відправки звітів (Layout_land/fragment_crime.xml)

На цій стадії ви можете перевірити макети в області попереднього перегляду або запустити додаток CrimIntent, щоб переконатися в правильності розташування нових кнопок.

Додавання імені в рівень моделі

Поверніться до файлу Crime.java, додайте нову константу JSON і поле для зберігання імені запису. Також змініть методи JSON для виконання серіалізації / десеріалізації з коду JSON і додайте нові методи доступу.

Лістинг 2. Додавання поля для імені запису (Crime.java)

```
public class Crime {
    ...
    private static final String JSON_PHOTO = "photo";
    private static final String JSON_SUSPECT = "suspect";

    ...
    private Photo mPhoto;
    private String mSuspect;

    public Crime(JSONObject json) throws JSONException {
      mId = UUID.fromString(json.getString(JSON_ID));
      ...
      if (json.has(JSON_PHOTO))
          mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
      if (json.has(JSON_SUSPECT))
          mSuspect = json.getString(JSON_SUSPECT);
    }

    public JSONObject toJSON() throws JSONException {
      JSONObject json = new JSONObject();
      ...
      if (mPhoto != null)
          json.put(JSON_PHOTO, mPhoto.toJSON());
          json.put(JSON_SUSPECT, mSuspect);
      return json;
    }

    public void setPhoto(Photo p) {
      mPhoto = p;
    }

    public String getSuspect() {
      return mSuspect;
    }

    public void setSuspect(String suspect) {
      mSuspect = suspect;
    }
}
```

Форматні рядки

Створення шаблону звіту про запис, який заповнюється інформацією про конкретний запис. Так як детальна інформація недоступна до стадії виконання, необхідно використовувати форматний рядок з наповнювачами, які будуть замінюватися під час виконання. Форматнийрядок буде виглядати так:

```
    <string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and
%4$s
```

Поля %1$s, %2$s і т.д. - наповнювачі для строкових аргументів. У коді ви викликаєте getString (...) і передаєте форматний рядок і ще чотири рядки в тому порядку, в якому вони повинні замінювати наповнювачі.

Спочатку додайте в strings.xml рядки з лістингу 3.

Лістинг 3. Додавання строкових ресурсів (strings.xml)

```
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
    <string name="crime_report">%1$s!
      The crime was discovered on %2$s. %3$s, and %4$s
    </string>
    <string name="crime_report_solved">The case is solved</string>
    <string name="crime_report_unsolved">The case is not solved</string>
    <string name="crime_report_no_suspect">There is no suspect.</string>
            <string name="crime_report_suspect">The suspect is %s.</string>
    <string name="crime_report_subject">CrimIntent Crime Report</string>
    <string name="send_report">Send crime report via</string>
</resources>
```

У файлі CrimeFragment.java додайте метод, який створює чотири рядки, з'єднує їх і повертає повний звіт.

Лістинг 4. Додавання методу getCrimeReport() (CrimeFragment.java)

```
private String getCrimeReport() {
    String solvedString = null;
    if (mCrime.isSolved()) {
        solvedString = getString(R.string.crime_report_solved);
    } else {
        solvedString = getString(R.string.crime_report_unsolved);
    }

    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat.format(dateFormat,

    mCrime.getDate()).toString();
    String suspect = mCrime.getSuspect();
    if (suspect == null) {
        suspect = getString(R.string.crime_report_no_suspect);
    } else {
        suspect = getString(R.string.crime_report_suspect, suspect);
    }

    String report = getString(R.string.crime_report,
        mCrime.getTitle(), dateString, solvedString, suspect);

    return report;
}
```

Використання неявних інтентів

Об'єкт Intent описує для ОС якусь операцію, яку ви хочете виконати. Для явних інтентів, що використовувалися до цього моменту, розробник явно вказує активність, яку повинна запустити ОС.

```
Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
startActivity(i);
```

Для неявних інтентів розробник описує операцію, що виконується, а ОС запускає активність, яка раніше повідомила про те, що вона здатна виконувати цю операцію. Якщо ОС знаходить кілька таких активностей, користувачеві пропонується вибрати потрібну.

Компоненти неявного інтенту

Нижче перераховані найважливіші складові інтенту, використовувані для визначення виконуваної операції.

Виконувана *дія* (action) - зазвичай визначається константами з класу Intent. Так, для перегляду URL-адреси використовується константа Intent.ACTION_VIEW, а для відправки даних - константа Intent.ACTION_SEND.

Місцезнаходження *даних* - це може бути як посилання на дані, що знаходяться за межами пристрою (URL веб-сторінки), так і URI файлу або URI контенту, який посилається на запис ContentProvider.

Тип даних, з якими працює дія, - тип MIME (наприклад, text/html або audio/mpeg3). Якщо в інтент включено місцезнаходження даних, то тип зазвичай вдається визначити за цими даними.

Необов'язкові *категорії* - якщо дія використовується для опису виконуваної операції, категорія зазвичай описує, де, коли або як ви намагаєтеся використовувати операцію. Android використовує категорію android.intent.category.LAUNCHER для позначення активностей, які повинні відображатися в лаунчер додатків верхнього рівня. З іншого боку, категорія android.intent.category.INFO позначає активність, яка видає користувачеві інформацію про пакет, але не відображається в лаунчері.

Простий неявний інтент для перегляду веб-сайту включає дію Intent.ACTION_VIEW і об'єкт даних Uri з URL-адресою сайту.

На підставі цієї інформації ОС запускає відповідну активність відповідного додатку. (Якщо ОС виявляє більше одного кандидату, користувачеві пропонується прийняти рішення.)

Активність повідомляє про себе як про виконавця для ACTION_VIEW за допомогою фільтра інтенту в маніфесті. Наприклад, якщо ви пишете додаток-браузер, ви включаєте наступний фільтр інтенту в оголошення активності, що реагує на ACTION_VIEW.

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

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" android:host="www.bignerdranch.com" />
    </intent-filter>
</activity>
```

Категорія DEFAULT повинна явно задаватися в фільтрах інтенту. Елемент action в фільтрі інтенту повідомляє ОС, що активність здатна виконувати операцію, а категорія DEFAULT - що вона бажає розглядатися серед кандидатів на виконання операції. Категорія DEFAULT неявно додається до майже будь-якого неявного інтенту. (Єдиний виняток становить категорія LAUNCHER.)

Неявні інтенти, як і явні, також можуть включати доповнення. Однак доповнення неявного інтенту не використовуються ОС для пошуку відповідної активності.

Також слід зазначити, що компоненти дії і даних інтенту можуть використовуватися в поєднанні з явними інтентами. Результат еквівалентний тому, що ви кажете конкретній активності виконати конкретну операцію.

Відправка звіту

Щоб побачити на практиці, як працює ця схема, ми створимо неявний інтент для відправки звіту про запис в додатку CrimIntent. Операція, яку потрібно виконати, - відправка простого тексту;

звіт представляє собою рядок. Таким чином, дія неявного інтенту буде представлено константою ACTION_SEND. Інтент не містить посилань на дані і не має категорій, але визначає тип text / plain.

В методі CrimeFragment.onCreateView (...) отримаєте посилання на кнопку Send Crime Report і призначте для неї слухача. У реалізації слухача створіть неявний інтент і передайте його startActivity (Intent).

Лістинг 5. Відправка звіту про запис (CrimeFragment.java)

```
...
Button reportButton = (Button)v.findViewById(R.id.crime_reportButton);
reportButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_SEND);
        i.setType("text/plain");
        i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
        i.putExtra(Intent.EXTRA_SUBJECT,
            getString(R.string.crime_report_subject));
        startActivity(i);
    }
});
return v;
}
```

Тут ми використовуємо конструктор Intent, Котрий отримує рядок з константою, що описує дію. Також існують інші конструктори, які можуть використовуватися залежно від виду створюваного неявного інтенту. Конструктора, що одержує тип, не існує, тому ми задаємо його явно.

Текст звіту і рядок теми включаються в доповнення. Зверніть увагу на використання у них констант, визначених в класі Intent. Будь-яка активність, що реагує на інтент, знає ці константи і те, що слід робити з асоційованими значеннями.

Запустіть додаток CrimIntent і натисніть кнопку Send Crime Report. Так як цей інтент з великою ймовірністю співпаде з багатьма активностями на пристрої, швидше за все, на екрані з'явиться список активностей:

Якщо на екрані з'явився список, виберіть потрібний варіант. Звіт про записи завантажується в обраному вами додатку. Вам залишається лише ввести адресу і відправити його.

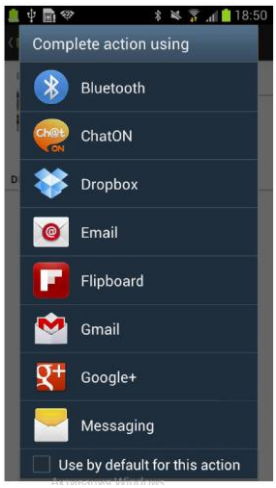

Рис. 5. Активності, які заявили про свою готовність виконати відправку звіту

Якщо список не з'явився, це може означати одне з двох: або ви вже назначили додаток за замовчуванням для ідентичного неявного інтенту, або на вашому пристрої є всього одна активність, здатна реагувати на цей інтент.

Часто кращим варіантом виявляється використання додатка за замовчуванням, обраного користувачем для даної дії. В додатку CrimIntent краще завжди надавати користувачеві вибір.

Ви можете створити список, який буде відображатися кожен раз при використанні неявного інтенту для запуску активності. Після створення неявного інтенту цим способом, ви викликаєте наступний метод Intent і передаєте йому неявний інтент і рядок з заголовком:

```
public static Intent createChooser(Intent target, String title)
```

Потім інтент, повернутий createChooser (...), передається startActivity (...).

У файлі CrimeFragment.java створіть список вибору для відображення активностей, що реагують на неявний інтент.

Лістинг 6. Використання списку вибору (CrimeFragment.java)

```
public void onClick(View v) {
   Intent i = new Intent(Intent.ACTION_SEND);
   i.setType("text/plain");
   i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
   i.putExtra(Intent.EXTRA_SUBJECT,
      getString(R.string.crime_report_subject));
   i = Intent.createChooser(i, getString(R.string.send_report));
   startActivity(i);
}
```

Запустіть додаток CrimIntent і натисніть кнопкуSend Crime Report. Якщо в системі є кілька активностей, здатних обробити ваш інтент, на екрані з'являється список для вибору.

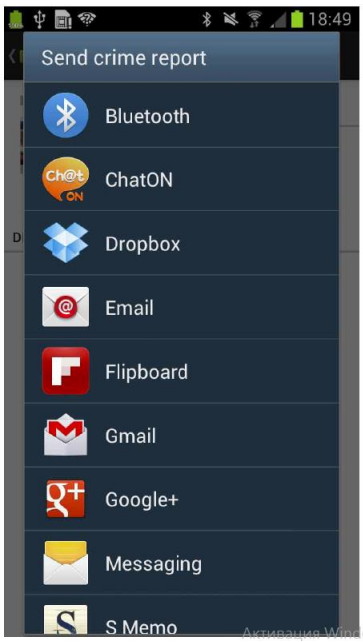

Рис. 6. Відправка тексту з вибором активності

Запит контакту у Android

Тепер ми створимо інший неявний інтент, який пропонує користувачеві вибрати записи зі списку контактів. Для цього неявного інтенту буде визначено дію і місцезнаходження відповідних даних. Дія задається константою Intent.ACTION_PICK, а місцезнаходження даних - ContactsContract.

Contacts.CONTENT_URI. Ви просите Android допомогти з вибором запису з бази даних контактів.

Запущена активність повинна повернути результат, тому ми передаємо інтент через startActivityForResult (...) разом з кодом запиту. Додайте в файл CrimeFragment.java константу для коду запиту і поле для кнопки.

Лістинг 7. Додавання поля для кнопки запис (CrimeFragment.java)

```
...
private static final int REQUEST_PHOTO = 1;
private static final int REQUEST_CONTACT = 2;
...
private ImageButton mPhotoButton;
private Button mSuspectButton;
...
```

В кінці onCreateView (...) отримаєте посилання на кнопку і призначте їй слухача.

В реалізації слухача створіть неявний інтент і передайте його startActivityForResult (...). Також виведіть на кнопці ім'я запису (якщо воно міститься в Crime).

Лістинг 8. Відправка неявного інтенту (CrimeFragment.java)

```
...
mSuspectButton = (Button)v.findViewById(R.id.crime_suspectButton);
mSuspectButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
       Intent i = new Intent(Intent.ACTION_PICK,
          ContactsContract.Contacts.CONTENT_URI);
       startActivityForResult(i, REQUEST_CONTACT);
    }
    });

    if (mCrime.getSuspect() != null) {
       mSuspectButton.setText(mCrime.getSuspect());
    }
    return v;
}
```

Запустіть додаток CrimIntent і натисніть кнопку Choose Suspect. На екрані з'являється список контактів.

Якщо у вас встановлено інше контактний додаток, екран буде виглядати інакше. Це ще одна перевага неявних інтентів: вам не потрібно знати назву контактного додатку, щоб використовувати його зі свого додатка. Відповідно користувач може встановити той додаток, який вважає за потрібним, а ОС знайде і запустить його.

Отримання даних зі списку контактів

Тепер необхідно отримати результат від контактного додатку. Контактна інформація спільно використовується багатьма додатками, тому Android надає розширений API для роботи з контактними даними через ContentProvider. Екземпляри цього класу інкапсулюють бази даних і надають доступ до них іншим програмам. Звернення до ContentProvider здійснюється через ContentResolver.

Так як активність запускалася з поверненням результату з використанням ACTION_PICK, ви можете отримати інтент викликом onActivityResult (...). Інтент включає URI дані - посилання на конкретний контакт, вибраний користувачем.

У файлі CrimeFragment.java додайте наступний код в реалізацію onActivityRe-sult (...) з CrimeFragment.

Лістинг 9. Отримання імені контакту (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        ...
    } else if (requestCode == REQUEST_CONTACT) {
    Uri contactUri = data.getData();

    // Визначення полів, значення яких повинні бути
    // повернуті запитом.
    String[] queryFields = new String[] {
        ContactsContract.Contacts.DISPLAY_NAME
    };

    // Виконання запиту - contactUri тут виконує функції
    // умови "where"
    Cursor c = getActivity().getContentResolver()
        .query(contactUri, queryFields, null, null, null);

    // Перевірка отримання результатів
    if (c.getCount() == 0) {
        c.close();
        return;
    }
    // Витяг першого стовпчика даних - імені запису.
    c.moveToFirst();
    String suspect = c.getString(0);
    mCrime.setSuspect(suspect);
    mSuspectButton.setText(suspect);
    c.close();
    }
}
```

У лістингу 9 створюється запит, який запитує все імена контактів, що відображаються з даних, що повертаються. Потім ми видаємо запит до бази даних контактів і отримуємо об'єкт Cursor для роботи з нею. Так як ми знаємо, що курсор містить всього один елемент, ми переходимо до першого елементу і використовуємо його як рядок. Цей рядок містить ім'я записів, яке ми використовуємо для завдання записів в Crime і тексту кнопки Choose Suspect.

Дозволи контактів

Як отримати дозвіл на читання з бази даних контактів? Контактний додаток поширює свої дозволи на вас. Він володіє повними дозволами на звернення до бази даних. Коли контактний додаток повертає батьківській активності URI даних в інтенті, він також додає прапор Intent. FLAG_GRANT_READ_URI_PERMISSION. Цей прапор повідомляє Android, що батьківській активності в CrimIntent слід дозволити одноразове використання цих даних. Такий підхід працює добре, тому що фактично нам потрібен доступ не до всієї бази даних контактів, а до одного контакту в цій базі.

Перевірка реагуючих активностей

На пристрої Android присутня та чи інша програма для роботи з електронною поштою та контактний додаток. А якщо ви створюєте інший неявний інтент для пристрою, на якому може не виявитися придатних активностей? Якщо ОС не знайде підходящу активність, в додатку відбувається збій.

Проблема вирішується попередньою перевіркою того, від якої частини ОС надійшов виклик PackageManager. Код виглядає приблизно так:

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(yourIntent, 0);
boolean isIntentSafe = activities.size() > 0;
```

Інтент передається методу queryIntentActivities (...) класу PackageManager. Метод повертає список об'єктів, що містять метадані про активність, що прореагувалю на переданий інтент. Залишається переконатися в тому, що список містить хоча б один елемент - тобто на пристрої є хоча б одна активність, що реагує на інтент.

Виконання цієї перевірки в onCreateView (...) дозволяє відключити варіанти, на які неможливо буде відреагувати.

Завдання. Інший неявний інтент

Додати нову кнопку для дзвінка зазначеному запису.

Знадобиться витягти з бази даних контактів телефонний номер, після чого можна створити неявний інтент з URI телефону:

```
Uri number = Uri.parse("tel:5551234");
```

При цьому може використовуватися дію Intent.ACTION_DIAL або Intent.ACTION_CALL. ACTION_CALL запускає телефонний додаток і негайно здійснює дзвінок за номером, відправленому в інтенті; ACTION_DIAL тільки вводить номер та чекає, поки користувач ініціює дзвінок.

Використовуйте ACTION_DIAL. Режим ACTION_CALL може бути обмежений, і для нього можуть будуть потрібні дозволи.

Лабораторна робота № 6

Тема: Робота з інтентами і задачами

У цьому розділі ми використовуємо неявні інтенти для створення програми-лаунчер, що заміняє стандартний лаунчер Android. Щоб додаток працював правильно, нам доведеться поглибити своє розуміння інтенту, фільтрів інтентів і схем взаємодій між додатками в середовищі Android.

Створення програми NerdLauncher

Створіть новий проект (New › Android Application Project) з тими ж параметрами, які використовувалися для CrimIntent (рис. 1). Дайте проекту ім'я NerdLauncher і створіть його в пакеті com.bignerdranch.android.nerdlauncher.

Створіть активність, але без користувальницького значка лаунчер. Виберіть створення нової порожньої активності. Дайте активності ім'я NerdLauncherActivity і клацніть на кнопці Finish.

Клас NerdLauncherActivity повинен бути субкласом SingleFragmentActivity, тому цей клас необхідно додати в проект. На панелі Package Explorer знайдіть файл SingleFragmentActivity.java в пакеті CrimIntent. Скопіюйте його в пакет com.bignerdranch.android.nerdlauncher. При копіюванні файлів Eclipse автоматично оновлює оголошення пакетів.

Нам також знадобиться макет activity_fragment.xml. Скопіюйте res/layout/activity_fragment.xml в каталог res/layout проекту NerdLauncher.

NerdLauncher буде відображати список додатків на пристрої. Користувач натискає елемент списку, щоб запустити відповідну програму. А тепер подивимося, які об'єкти для цього знадобляться.

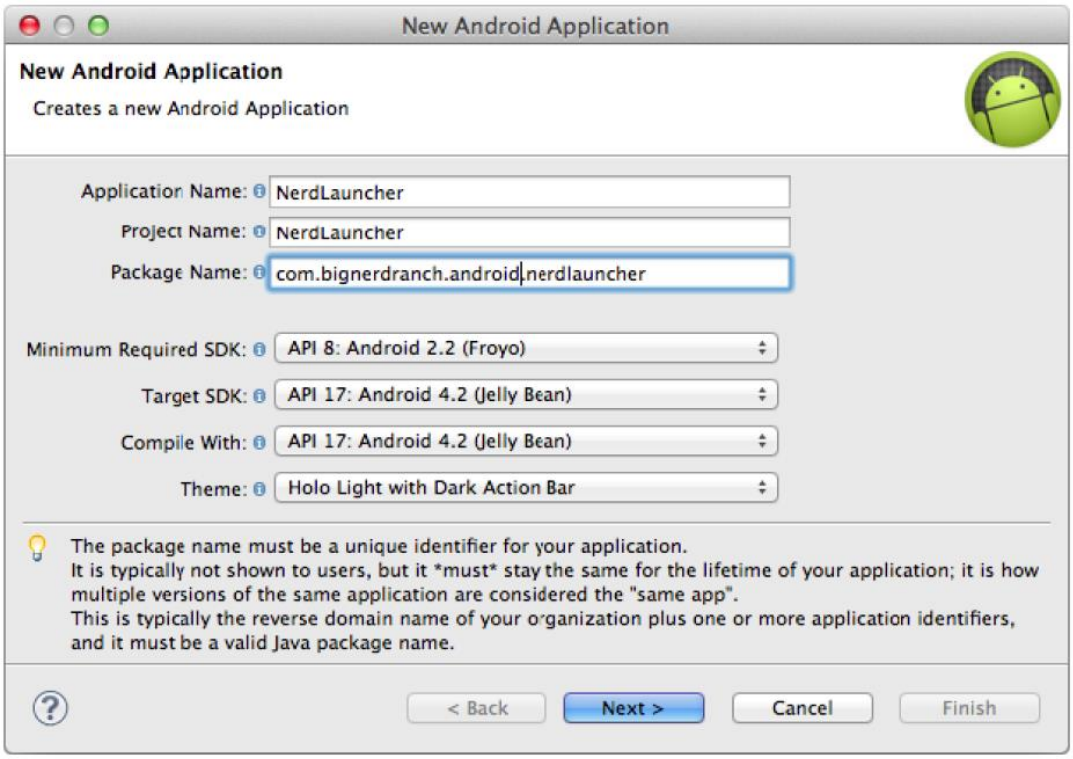

Рис. 1. Створення проекту NerdLauncher

Клас NerdLauncherFragment є субкласом ListFragment, а в якості представлення він буде використовувати стандартний клас ListView, що додається до ListFragment.

Створіть новий клас з ім'ям NerdLauncherFragment і призначте його суперкласом android.support.v4.app.ListFragment. Поки залиште цей клас порожнім.

Відкрийте файл NerdLauncherActivity.java і змініть суперклас NerdLauncherActivity на SingleFragmentActivity. Видаліть код шаблону і перевизначите метод createFragment() так, щоб він повертав NerdLauncherFragment.

Лістинг 1. Субклас SingleFragmentActivity (NerdLauncherActivity.java)

```
public class NerdLauncherActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nerd_launcher);
    }

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

    }
    @Override
    public Fragment createFragment() {
        return new NerdLauncherFragment();
    }
}
```

Обробка неявного інтенту

NerdLauncher відображає список додатків на пристрої. Для цього Nerd-Launcher відправляє неявний інтент, на який повинна відреагувати головна активність кожного додатка. У інтенті включається дія MAIN і категорія LAUNCHER.

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

У файлі NerdLauncherFragment.java перевизначите метод onCreate (Bundle) для створення неявного інтенту. Отримайте від PackageManager список активностей, відповідних інтентів. Поки ми обмежимося простою реєстрацією кількості активностей, повернутих PackageManager.

Лістинг 2. Отримання інформації у PackageManager (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends ListFragment {
    private static final String TAG = "NerdLauncherFragment";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent,
        0);

        Log.i(TAG, "I've found " + activities.size() + " activities.");
    }
}
```

Запустіть додаток NerdLauncher і подивіться в даних LogCat, скільки додатків повернув екземпляр PackageManager.

У CrimIntent для відправки звітів використовувався неявний інтент. Щоб представити на екрані список вибору додатків, ми створили неявний інтент, упакували його в об'єкт вибору і відправили ОС викликом startActivity (Intent):

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Створення і розміщення доповнень інтенту
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

Ми не використовуємо цей підхід тут тому, що фільтр інтенту MAIN / LAUNCHER може відповідати або не відповідати неявному інтенту MAIN / LAUNCHER, відправленому з startActivity (...).

Виклик startActivity (Intent) не означає «Запустити активність, яка відповідає цьому неявному інтенту». Він означає «Запустити активність за замовчуванням відповідає цьому неявному інтенту». Коли ви відправляєте неявний інтент з використанням startActivity (...) (або startActivityForRe-sult (...)), ОС непомітно включає в інтент категорію Intent.CATEGORY_ DEFAULT.

Таким чином, якщо ви хочете, щоб фільтр інтенту відповідав неявним інтентам, відправленим через startActivity (...), ви повинні включити в цей фільтр інтенту категорію DEFAULT.

Активність з фільтром інтенту MAIN / LAUNCHER є головною точкою входу додатки, якому вона належить. Для неї важливо лише те, що вона є головною точкою входу додатки, а чи є вона головною точкою входу «за замовчуванням» - несуттєво, тому вона не повинна мати категорію CATEGORY_DEFAULT.

Так як фільтри інтентів MAIN / LAUNCHER можуть не включати CATEGORY_DEFAULT, Надійність їх відповідності неявним інтентом, відправленим викликом startAc-tivity (...), не гарантована. Тому ми використовуємо інтент для прямого запиту у PackageManager інформації про активність з фільтром інтенту MAIN / LAUNCHER.

Наступний крок - відображення міток цих активностей в списку ListView екземпляру NerdLauncherFragment. Мітка (label) активності являє собою відображуване ім'я - щось, зрозуміле користувачу. Якщо врахувати, що ці активності є активностями лаунчер, такою міткою, швидше за все, має бути назва програми.

Мітки активностей разом з іншими метаданими містяться в об'єктах ResolveInfo, що повертаються PackageManager.

Спочатку додайте наступний код для сортування об'єктів ResolveInfo, PackageManager, що повертаються, в алфавітному порядку міток, одержуваних методом ResolveInfo.loadLabel (...).

Лістинг 3. Алфавітна сортування (NerdLauncherFragment.java)

```
...
Log.i("NerdLauncher", "I've found " + activities.size() + " activities.");

Collections.sort(activities, new Comparator<ResolveInfo>() {
    public int compare(ResolveInfo a, ResolveInfo b) {
        PackageManager pm = getActivity().getPackageManager();
        return String.CASE_INSENSITIVE_ORDER.compare(
            a.loadLabel(pm).toString(),
            b.loadLabel(pm).toString());
    }
});
```

Потім створіть об'єкт ArrayAdapter, який створить прості представлення елементів списку з мітками активностей, і призначте цей адаптер ListView.

Лістинг 4. Створення адаптера (NerdLauncherFragment.java)

```
...
Collections.sort(activities, new Comparator<ResolveInfo></ResolveInfo>() {
    ...
    }
});
ArrayAdapter<ResolveInfo> adapter = new ArrayAdapter<ResolveInfo>(
        getActivity(), android.R.layout.simple_list_item_1, activities) {
    public View getView(int pos, View convertView, ViewGroup parent) {
        View v = super.getView(pos, convertView, parent);
        // У документації сказано, що simple_list_item_1
        // є TextView; перетворимо для завдання текстового значення.
        TextView tv = (TextView)v;
        ResolveInfo ri = getItem(pos);
        tv.setText(ri.loadLabel(pm));
        return v;
    }
};

setListAdapter(adapter);
```

Запустіть додаток NerdLauncher; ви побачите список ListView, заповнений мітками активностей.

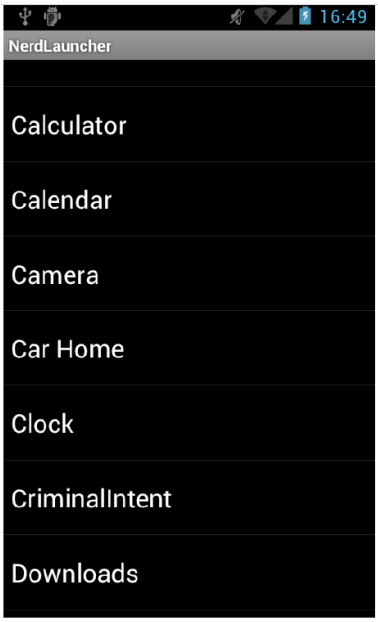

Рис. 2. Список активностей

Створення явних інтентів на стадії виконання

Ми використовували неявний інтент для збору інформації про активність і виведення в форматі списку. Наступним кроком має стати запуск обраної активності при натисканні користувачем на елементі списку. Для запуску активності буде використовуватися явний інтент.

Для створення явного інтенту нам будуть потрібні додаткові дані з ResolveInfo - зокрема, ім'я пакета і ім'я класу активності. Ці дані можна отримати з частини ResolveInfo з ім'ям ActivityInfo. (Про те, які дані доступні в різних частинах ResolveInfo, можна дізнатися з документації.)

В файлі NerdLauncherFragment.java перевизначите метод onListItemClick (...) для отримання об'єкта ActivityInfo для елемента списку. Потім використовуйте його дані для створення явного інтенту, що запускає активність.

Лістинг 5. Реалізація onListItemClick (...) (NerdLauncherFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    ResolveInfo resolveInfo = (ResolveInfo)l.getAdapter().getItem(position);
    ActivityInfo activityInfo = resolveInfo.activityInfo;
    if (activityInfo == null) return;
    Intent i = new Intent(Intent.ACTION_MAIN);
    i.setClassName(activityInfo.applicationInfo.packageName,
    activityInfo.name);
    startActivity(i);
}
```

В цьому інтенті ми відправляємо дію як частина явного интенту. Більшість додатків поводиться однаково незалежно від того, включено дію чи ні, однак деякі додатки можуть змінювати свою поведінку. Одна і та ж активність може відображати різні інтерфейси в залежності від того, як вона була запущена.

У лістингу 5 ми отримуємо ім'я пакета і ім'я класу з метаданих і використовуємо їх для створення явної активності методом Intent:

```
public Intent setClassName(String packageName, String className)
```

Цей спосіб відрізняється від того, який використовувався нами для створення явних інтентів в минулому. Раніше ми використовували конструктор Intent, який одержує об'єкти Context і Class:

```
public Intent(Context packageContext, Class<?> cls)
```

Цей конструктор використовує свої параметри для отримання ComponentName- імені пакета, об'єднаного з ім'ям класу. Коли ви передаєте Activity і Class для створення Intent, Конструктор визначає повне ім'я пакета по Activity.

Також можна самостійно створити ComponentName по іменах пакета і класу і використовувати наступний метод Intent для створення явного інтенту:

```
public Intent setComponent(ComponentName component)
```

Однак рішення з методом setClassName (...), автоматично створює ім'я компонента, виходить більш компактним.

Запустіть NerdLauncher і подивіться, як працює запуск додатків.

Завдання і стек повернення

Android використовує завдання для відстежування поточного стану користувача кожному виконуваному додатку. Завдання (task) являє собою стек активностей, з якими має справу користувач. Активність в нижній позиції стека називається базовою активністю, а активність у верхній позиції видно користувачу. При натисканні кнопки Back верхня активність вилучають із стека. Якщо натиснути кнопку Back при перегляді базової активності, ви повернетеся до домашнього екрану.

Диспетчер завдань дозволяє перемикатися між завданнями без зміни стану кожного завдання. Наприклад, якщо ви починаєте вводити новий контакт і перемикаєтеся на перевірку своєї публікації в Твіттері, будуть запущені два завдання. При перемиканні на редагування контактів зберігається ваша поточна позиція в обох задачах.

Іноді активність, що запускається повинна додаватися до поточної задачі. В інших випадках вона повинна запускатися в новому завданні, незалежною від її активності, яка запустила.

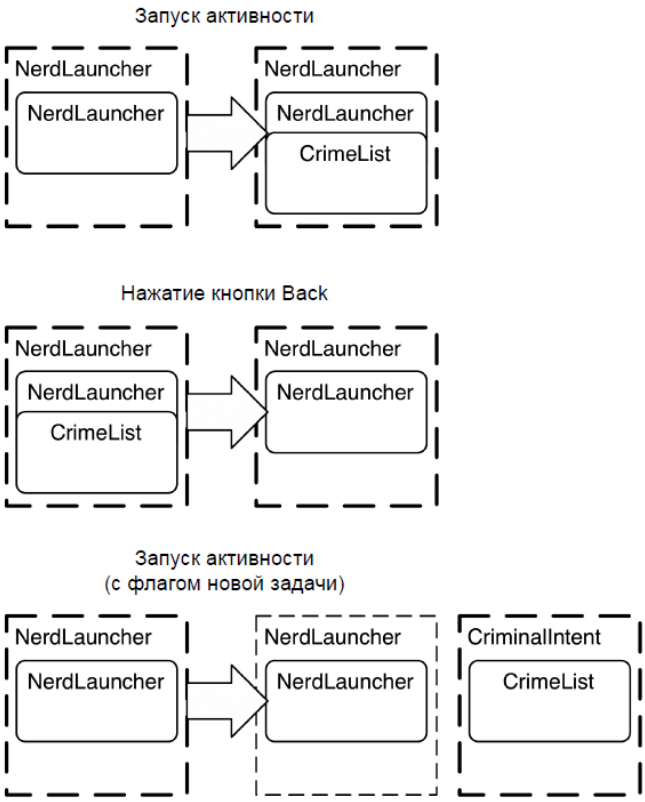

Рис. 3. Завдання і стек повернення

За замовчуванням нові активності запускаються в поточному завданні. У додатку CrimIntent всі активності, що запускалися додавалися до поточної задачі. Це стосувалося навіть до активностей, які не були частиною програми CrimIntent (наприклад, при запуску активності для відправки звіту). Перевага додавання активності до поточної задачі полягає в тому, що користувач може виконувати зворотну навігацію всередині завдання, а не в ієрархії додатків.

В поточній версії всі активності, що запускаються з NerdLauncher, додаються в задачу NerdLaucher. Щоб переконатися в цьому, запустіть CrimIntent з Nerd-Launcher і викличте диспетчер задач. (Натисніть кнопку Recents, якщо вона є на пристрої; в іншому випадку використовуйте довге натискання кнопки Home.) Ви не знайдете в списку CrimIntent. Запущена активність CrimeListActivity була додана в задачу NerdLauncher. Натискання на задачі NerdLauncher поверне вас до екрану CrimIntent, який ви переглядали перед запуском диспетчера задач.

Рис. 4. Додаток CrimIntent не виконується в окремому завданні

Ми хочемо, щоб додаток NerdLauncher запускав активності в нових завданнях. Далі користувач може перемикатися між виконуваними програмами так, як вважає за потрібне. Щоб при запуску нової активності запускалася нова задача, слід додати в інтент відповідний прапор.

Запустіть додаток NerdLauncher і виберіть CrimIntent. На цей раз при виклику диспетчера задач стає видно, що CrimIntent виконується в окремій задачі.

Лістинг 6. Додавання прапора в інтент (NerdLauncherFragment.java)

```
Intent i = new Intent(Intent.ACTION_MAIN);
i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(i);
```

Рис. 5. CrimIntent виконується у власній задачі

Повторний запуск CrimIntent з NerdLauncher не приведе до створення другого завдання CrimIntent. Прапор FLAG_ACTIVITY_NEW_TASK створює тільки одну задачу на активність. У CrimeListActivity вже є працююче завдання, тому Android переключиться на цю задачу замість запуску нової.

Використання NerdLauncher в якості домашнього екрану

Набагато логічніше використовувати NerdLauncher як заміну для домашнього екрану пристрою. Відкрийте файл AndroidManifest.xml в NerdLauncher і додайте наступний фрагмент в головний фільтр інтенту.

Лістинг 7. Зміна категорій NerdLauncher (AndroidManifest.xml)

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

Додавання категорій HOME і DEFAULT означає, що активність NerdLauncher повинна включатися в число варіантів домашнього екрану. Натисніть кнопку Home і вам буде запропоновано використовувати NerdLauncher.

(Якщо ви призначите NerdLauncher домашнім екраном, а потім захочете скасувати свій вибір, виконайте команду Settings › Applications › Manage Applications. Оберіть All, знайдіть NerdLauncher і скиньте режим запуску за замовчуванням Launch by default. При наступному натисканні кнопки Home ви зможете вибрати новий домашній екран за замовчуванням.)

Процеси і завдачи

Для існування будь-якого об'єкта необхідна пам'ять і віртуальна машина. *Процес* являє собою місце, створене ОС, в якому існують об'єкти вашого застосування і в якому виконується сам додаток.

Процесам можуть належати ресурси, що знаходяться під управлінням ОС - пам'яті, мережеві сокети, відкриті файли і т.д. Процес також містить мінімум один (а ймовірно, кілька) програмний *потік* (thread). На платформі Android процес завжди виконується рівно на одній *віртуальній машині Dalvik*.

Як правило, кожен компонент додатка в Android зв'язується рівно з од-им процесом (хоча зустрічаються досить смутні виключення). Додаток створюється з власним процесом, який стає процесом за замовчуванням для всіх компонентів програми.

Окремі компоненти можна призначати різним процесам, але рекомендується дотримуватися процесу за замовчуванням. Якщо ви думаєте, що якийсь код повинен виконуватися в іншому процесі, аналогічного результату зазвичай вдається домогтися використанням *багатопоточності* (multi-threading), яка програмується в Android набагато простіше, ніж багатопроцесність виконання.

Кожен екземпляр активності існує рівно в одному процесі і рівно в одній задачі. Втім, на цьому вся схожість і завершується. Завдання містять тільки активності і часто складаються з активностей різних додатків. З іншого боку, процеси містять тільки виконуваний код і об'єкти додатка.

Процеси і завдання легко сплутати, тому що ці концепції частково перекриваються, а для посилань на них часто використовуються імена додатків. Наприклад, при запуску CrimIntent з NerdLauncher ОС створює процес CrimIntent і нове завдання, для якої CrimeListActivityє базовою активністю. У диспетчері завдань ця задача забезпечується міткоюCrimIntent.

Задача, в якій існує активність, може бути не пов'язана з процесом, в якому вона існує. Коли ми запустили контактний додаток для вибору запису в CrimIntent, він був запущений в завданні CrimIntent - але при цьому виконувався в процесі контактного додатку.

Рис. 6. Завдання і процеси

Це означає, що коли користувач натискає кнопку Back для переходу між різними активностями, він може непомітно для себе перемикатися між процесами.

Android не надає засобів для знищення завдань або заміни стандартного диспетчера задач Android. Довгий на-жатіе на кнопці Home жорстко пов'язано з диспетчером завдань за замовчуванням, а завдання знищуватися не можуть. Процеси, навпаки, можуть знищуватися. Додатки, що рекламуються в магазині Google Play як знищувачі завдань, в дійсності є тими, що знищують процесів.

Завдання 1. Використання ActionBarSherl

Після додавання посилання на проект ActionBarSherl, Все готово до його інтеграції в CrimIntent. Бібліотека ABS працює за тим же принципом, що і бібліотека підтримки - вона надає альтернативні версії таких основоположних класів Android, як Activity, Fragment і ActionBar. Багато (але не всі) імена класів ABS починаються із префікса Sherl, щоб їх було простіше відрізнити від класів бібліотеки підтримки.

Класи фрагментів і активностей мають префікс Sherl-, але у класів меню цього префікса немає.

Базова інтеграція ABS в CrimIntent

Послідовність дій по базовій інтеграції ABS в проект:

- Змініть оголошення класів SingleFragmentActivity і CrimePagerActiv-ity так, щоб вони були похідними від SherlFragmentActivity (замість FragmentActivity).
- Змініть всі оголошення класів фрагментів так, щоб вони були похідними від SherlFragment, SherlDialogFragment або SherlListFragment (Замість версій цих класів з бібліотеки підтримки).
- Замініть посилання на Menu, MenuItem і MenuInflater посиланнями на відповідні реалізації з com.actionbarsher.view.

Перші два кроки прості - ви просто змінюєте імена суперкласів. Коли це буде зроблено, у CrimeFragment і CrimeListFragment з'являться численні помилки. Щоб позбутися від них, необхідно виконати третій крок, який складніше двох попередніх.

Для CrimeFragment проблема вирішується просто: видаліть директиви, що відносяться до меню import на початку файлу, проведіть організацію імпорту комбінацією клавіш Command + Shift + O / Ctrl + Shift + O і виберіть версії com.actionbarsher.view.

Однак якщо ви спробуєте зробити те ж саме з CrimeListFragment, виникнуть проблеми. Справа в тому, що CrimeListFragment використовує контекстні меню і MultiChoiceModeListener, для яких необхідні оригінальні версії класів меню з бібліотеки Android.

Тому, замість обхідного рішення з організацією імпорту повністю уточніть посилання на типи MenuItem, Menu і MenuInflater у onCreateOptionsMenu (...) і onOptionsItemSelected (...).

Візьмемо для прикладу наступний код:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}
```

Його необхідно записати в такому вигляді:

```
@Override
public void onCreateOptionsMenu(com.actionbar.view.Menu menu,
        com.actionbar.view.MenuInflater inflater) {
    ...
}
```

Інтеграція більш високого рівня

Для повноцінного використання бібліотеки необхідно позбутися від коду сумісності. Для цього викличте в код igetSherlActivity(). getSupportActionBar() замість getActivity(). getActionBar(). Панель дій SherlActivity доступна завжди, тому її використання дозволить обійтися без коду

захисної перевірки. Після цього можна перемістити файл res/menu-v11/fragment_crime_list.xml в res/menu, щоб позбутися від перемикання версій ОС на рівні конфігурації.

Інтеграція ще більш високого рівня

Наступним кроком на шляху до ідентичної поведінки CrimIntent в різних версіях є відмова від використання MultiChoiceModeListener і контекстних меню. Видалити код контекстного меню нескладно, але для заміни MultiChoiceModeListener буде потрібна імітація функціональності.

Перш за все можна використовувати класичний режим вибору ListView: ListView.CHOICE_ MODE_MULTIPLE. Режим ListView.CHOICE_MODE_MULTIPLE_MODAL працює тільки в нових версіях Android. Режим ListView.CHOICE_MODE_MULTIPLE не підтримує сучасну поведінку модальних довгих натискань, але у нього є своя перевага: він працює у всіх версіях Android. Вибір режиму виділення ListView.CHOICE_MODE_MULTIPLE для ListView дозволить виділяти кілька елементів списку. Щоб заборонити виділення, поверніть значення ListView. CHOICE_MODE_NONE.

Наступним кроком повинна стати імітація модальної поведінки панелі дій, реалізованого CHOICE_MODE_MULTIPLE_MODAL. Щоб зробити це способом, універсальним для всіх версій, слід викликати getSherlActivity().startActionMode(). Використовуйте версію, яка одержує com.action-barsher.view.ActionMode.Callback, а не звичайну версію Android.

Нарешті, необхідно організувати виявлення довгих натискань. Для цього можна передати слухача OnItemLongClickListener при виклику ListView.setOnItemLongClickListener (...).

Завдання 2. Значки, зміна порядку завдань

Метод ResolveInfo.loadLabel (...) використовувався для відображення змістовних імен в лаунчер. Клас ResolveInfo надає аналогічний метод loadIcon() для отримання значка, який відображатиметься для кожної програми.

Забезпечте кожен додаток в NerdLauncher значком.

Додайте в NerdLauncher іншу активність для перемикання між виконуваними завданнями. Для цього використовуйте системну службу ActivityManager, яка надає інформацію про активність, задачах і додатках, які виконуються в даний час. На відміну від PackageManager клас Activity не надає допоміжного методу getActivityManager() для отримання доступу до цієї системної службі.

Замість цього для отримання об'єкта ActivityManager слід викликати Activity. getSystemService() з передачею в параметрі константи Activity.ACTIVITY_SERVICE. викличте getRunningTasks() для отримання списку виконуваних завдань, упорядкованих за часом запуску (від нових до старих). Щоб вивести одну з таких задач на передній план, викличте методmove TaskToFront(). Обов'язково звіртеся з довідковою документацією Android - для перемикання між завданнями в маніфест додатку необхідно додати додатковий дозвіл.

Лабораторна робота № 7

Тема: Обробка подій TouchScreen

Створення проекту DragAndDraw

Клас BoxDrawingView займає центральне місце в новому проекті DragAndDraw. Виконайте команду New > Android Application Project. Задайте параметри проекту, як показано на рис. 2, і створіть порожню активність з ім'ям DragAndDrawActivity.

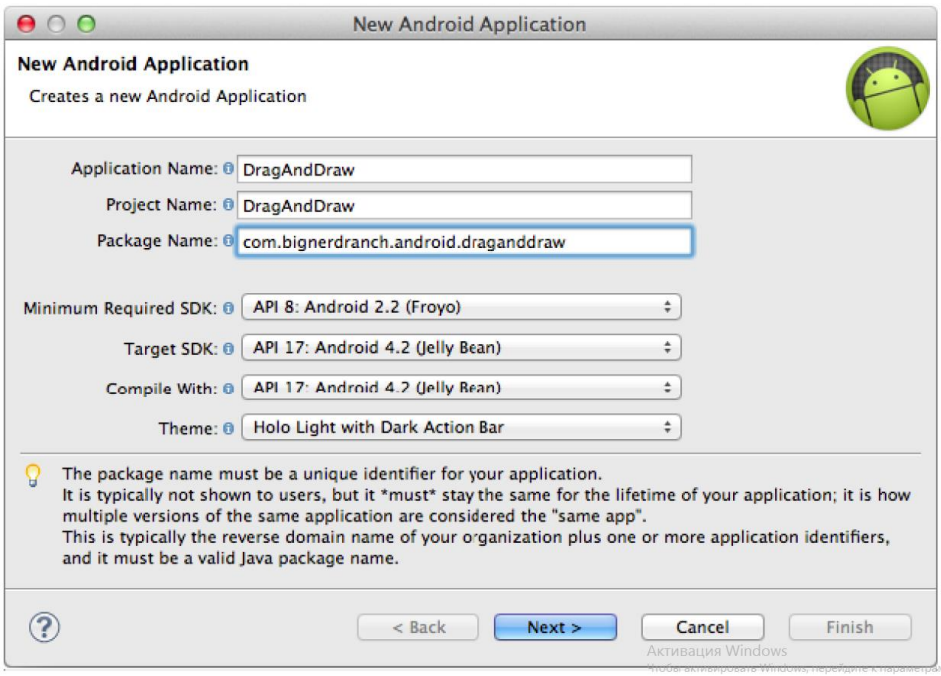

Рис. 2. Створення проекту DragAndDraw

Створення класу DragAndDrawActivity

Клас DragAndDrawActivity є субкласом SingleFragmentActiv-ity, який заповнює звичайний макет з одним фрагментом. На панелі Package Explorer скопіюйте файл SingleFragmentActivity.java в пакет com.bignerdranch.android. draganddraw. Потім скопіюйте файл activity_fragment.xml в каталог res / layout проекту DragAndDraw.

У файлі DragAndDrawActivity.java оголосіть DragAndDrawActivity субкласом Single-FragmentActivity. Цей клас повинен створювати екземпляр DragAndDrawFragment (клас, який буде створений наступним).

Лістинг 1. Зміна активності (DragAndDrawActivity.java)

```
public class DragAndDrawActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setContentView (R.layout.activity_drag_and_draw);
    }

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

```
    @Override
    public Fragment createFragment () {
    return new DragAndDrawFragment ();
    }
}
```

Створення класу DragAndDrawFragment

Щоб підготувати макет DragAndDrawFragment, перейменуйте файл and_draw.xml в fragment_drag_and_draw.xml.

Макет DragAndDrawFragment в кінцевому підсумку буде складатися з BoxDrawingView - користувальницького представлення. Все графічне виведення і обробка подій торкання будуть реалізовані в BoxDrawingView.

Створіть новий клас з ім'ям DragAndDrawFragment і призначте його суперкласом android.support.v4.app.ListFragment. Перевизначите метод onCreateView (...), щоб він заповнював макет fragment_drag_and_draw.xml.

Лістинг 2. Створення фрагмента (DragAndDrawFragment.java)

```
public class DragAndDrawFragment extends Fragment {
    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup parent,
                Bundle savedInstanceState) {
        View v = inflater.inflate (R.layout.fragment_drag_and_draw, parent, false);
        return v;
    }
}
```

Відкрийте програму DragAndDraw і переконайтеся в тому, що настройка була виконана правильно.

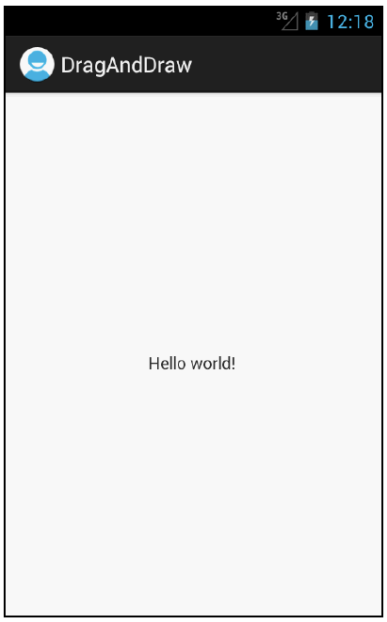

Рис. 3. DragAndDraw з макетом за замовчуванням

Створення нестандартного представлення

Android надає багато чудових стандартних представлень і віджетів, але іноді потрібно створити нестандартне представлення з візуальним оформленням, повністю унікальним для вашого застосування.

Все різноманіття нестандартних представлень можна умовно розділити на дві загальні категорії:

- прості представлення - просте представлення може бути влаштовано досить складно; «Простим» воно називається тільки тому, що не має дочірніх представлень. Просте представлення майже завжди також виконує нестандартну прорисовку;
- складові представлення - складаються з інших об'єктів представлень. Складові представлення зазвичай керують дочірніми представленнями, але не займаються своєю прорисовкою. Замість цього кожному дочірньому представленню делегується своя частина роботи по прорисовці.

Створення нестандартного представлення складається з трьох кроків:

- Вибір суперкласу. Для простого нестандартного представлення View надає порожне «полотно» для малювання, тому цей вибір є найбільш поширеним. Для складових нестандартних представлень виберіть відповідний клас макета.
- Субкласуйте обраний клас і перевизначите як мінімум один конструктор з суперкласу або створіть власний конструктор, що викликає один з конструкторів суперкласу.
- Перевизначите інші ключові методи для налаштування поведінки.

Створення класу BoxDrawingView

Клас BoxDrawingView відноситься до категорії простих представлень і є прямим субкласом View.

Створіть новий клас з ім'ям BoxDrawingView і призначте View його суперкласом.

Додайте в файл BoxDrawingView.java два конструктора.

Лістинг 3. Вихідна реалізація BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    // Використовується при створенні представлення в коді
    public BoxDrawingView (Context context) {
        this (context, null);
    }
    // Використовується при заповненні представлення по розмітці XML
    public BoxDrawingView (Context context, AttributeSet attrs) {
        super (context, attrs);
    }
}
```

Два конструктора потрібні тому, що екземпляр представлення може створюватися як в коді, так і по файлу розмітки. Представлення, створені на базі файлу макета, отримують примірник AttributeSet з атрибутами XML, заданими в XML. Навіть якщо ви не збираєтеся використовувати обидва конструктора, їх рекомендується включити.

Потім поновіть файл макета fragment_drag_and_draw.xml, щоб в ньому використовувалося нове представлення.

Лістинг 4. Включення Add BoxDrawingView в макет (fragment_drag_and_draw.xml)

```
<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"
>

<TextView
    android: layout_width = "wrap_content"
    android: layout_height = "wrap_content"
    android: layout_centerHorizontal = "true"
```

```
    android: layout_centerVertical = "true"
    android: text = "@ string / hello_world" />

</ RelativeLayout>

<com.bignerdranch.android.draganddraw.BoxDrawingView
    xmlns: android = "http://schemas.android.com/apk/res/android"
    android: layout_width = "match_parent"
    android: layout_height = "match_parent"
/>
```

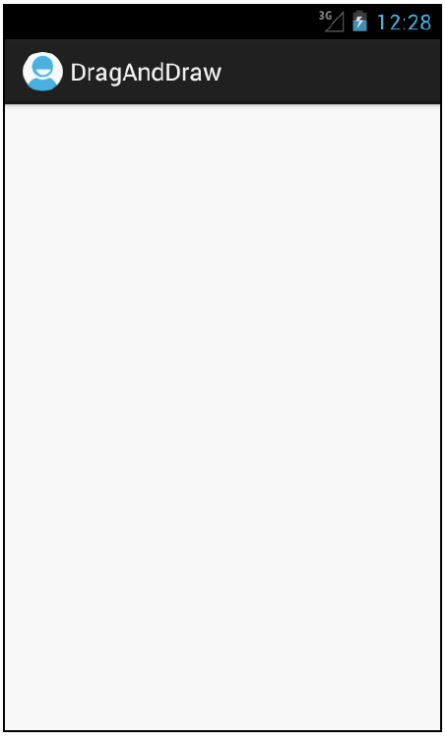

Рис. 4. BoxDrawingView без прямокутників

Щоб заповнювач макетів знайшов клас BoxDrawingView, ви повинні використовувати повністю уточнене ім'я. Заповнювач переглядає файл макета, створюючи екземпляри View. Якщо ім'я елемента буде неповним, то заповнювач шукає клас з вказаним ім'ям в пакетах android.view і android.widget. Якщо клас знаходиться в іншому місці, заповнювач його не знайде, і в додатку відбудеться збій. З цієї причини для класів, що не входять в android.view і android.widget, необхідно завжди ставити повністю уточнене ім'я.

Відкрийте програму DragAndDraw і переконайтеся в тому, що настройка була виконана правильно. Правда, поки на екрані немає нічого, крім порожнього представлення.

Обробка подій торкання

Для прослуховування подій торкання можна призначити слухача події за допомогою наступного методу класу View:

```
public void setOnTouchListener (View.OnTouchListener l)
```

Цей метод працює майже так само, як setOnClickListener (View.OnClickListener). Ви надаєте реалізацію View.OnTouchListener, а слухач викликається кожен раз, коли відбувається подія торкання.

Але оскільки ми субкласуемо View, можна піти за скороченим шляху і перевизначити наступний метод класу View:

```
public boolean onTouchEvent (MotionEvent event)
```

Цей метод отримує екземпляр MotionEvent - класу, що описує подію належним чином, включаючи його позицію і дію. Дія описує стадію події.

Константи дій	Опис
ACTION_DOWN	Користувач доторкнувся до екрану
ACTION_MOVE	Користувач переміщує палець по екрану
ACTION_UP	Користувач відводить палець від екрану
ACTION_CANCEL	Батьківське представлення перехопило подія торкання

У своїй реалізації onTouchEvent (...) для перевірки дії можна скористатися наступним методом класу MotionEvent:

```
public final int getAction ()
```

Додайте в файл BoxDrawingView.java тег для журналу і реалізацію onTouch (...), яка реєструє в журналі інформацію про кожного з чотирьох різних дій.

Лістинг 5. Реалізація BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";
    ...

    public boolean onTouchEvent (MotionEvent event) {
        PointF curr = new PointF (event.getX (), event.getY ());

        Log.i (TAG, "Received event at x =" + curr.x +
        ", Y =" + curr.y + ":");

        switch (event.getAction ()) {
          case MotionEvent.ACTION_DOWN:
              Log.i (TAG, "ACTION_DOWN");
              break;
          case MotionEvent.ACTION_MOVE:
              Log.i (TAG, "ACTION_MOVE");
              break;
          case MotionEvent.ACTION_UP:
              Log.i (TAG, "ACTION_UP");
              break;
          case MotionEvent.ACTION_CANCEL:
              Log.i (TAG, "ACTION_CANCEL");
              break;
        }

        return true;
    }
}
```

Зверніть увагу: координати X і Y упаковуються в об'єкті PointF. PointF - наданий Android клас-контейнер, який вирішує цю задачу за вас.

Відкрийте програму DragAndDraw і відкрийте LogCat. Доторкніться до екрану проведіть пальцем. Ви побачите в журналі повідомлення з координатами X і Y кожної дії торкання, отриманого BoxDrawingView.

Відстеження переміщень між подіями

Клас BoxDrawingView повинен малювати прямокутники, а не реєструвати координати. Для цього необхідно вирішити ряд завдань.

Перш за все для визначення прямокутника нам знадобляться:
- базова точка (в якій було зроблено вихідний дотик);
- поточна точка (в якій знаходиться палець).

Отже, для визначення прямокутника необхідно відстежувати дані від декількох подій MotionEvent. Дані будуть зберігатися в об'єкті Box.

Створіть клас з ім'ям Box для зберігання даних, що визначають прямокутник.

Лістинг 6. Клас Box (Box.java)

```
public class Box {
    private PointF mOrigin;
    private PointF mCurrent;

    public Box (PointF origin) {
        mOrigin = mCurrent = origin;
    }

    public void setCurrent (PointF current) {
        mCurrent = current;
    }

    public PointF getOrigin () {
        return mOrigin;
    }
}
```

Коли користувач торкається до BoxDrawingView, новий об'єкт Box створюється і включається в масив існуючих прямокутників (рис. 5).

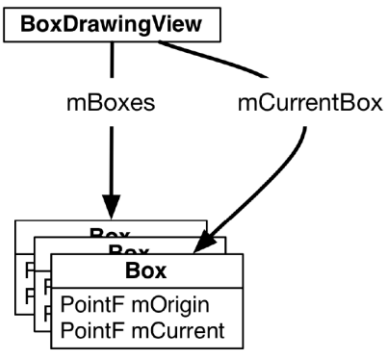

Рис. 5. Об'єкти в DragAndDraw

Додайте в BoxDrawingView код, який використовує новий об'єкт Box для відстежування поточного стану малювання.

Лістинг 7. Додавання методів життєвого циклу подій торкання (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private ArrayList <Box> mBoxes = new ArrayList <Box> ();

    ...
```

```
    public boolean onTouchEvent (MotionEvent event) {
        PointF curr = new PointF (event.getX (), event.getY ());

    switch (event.getAction ()) {
        case MotionEvent.ACTION_DOWN:
            Log.i (TAG, "ACTION_DOWN");
            // Reset drawing state
            mCurrentBox = new Box (curr);
            mBoxes.add (mCurrentBox);
            break;
        case MotionEvent.ACTION_MOVE:
            Log.i (TAG, "ACTION_MOVE");

            if (mCurrentBox! = null) {
                mCurrentBox.setCurrent (curr);
                invalidate ();
            }
            break;

        case MotionEvent.ACTION_UP:
            Log.i (TAG, "ACTION_UP");
            mCurrentBox = null;
            break;

        case MotionEvent.ACTION_CANCEL:
            Log.i (TAG, "ACTION_CANCEL");
            mCurrentBox = null;
            break;
    }
    return true;
    }
}
```

При кожному отриманні події ACTION_DOWN в полі mCurrentBox зберігається новий об'єкт Box з базовою точкою, відповідної позиції події. Цей об'єкт Box додається в масив прямокутників (BoxDrawingView буде виводити кожен об'єкт Box з масиву).

В процесі переміщення пальця по екрану додаток оновлює mCurrentBox, mCurrent. Потім, коли дотик скасовується або палець відходить від екрану, поле mCurrentBox обнуляється для завершення операції. Об'єкт Box завершено; він збережений в масиві і вже не буде оновлюватися подіями переміщення.

У разі ACTION_MOVE виклик invalidate () змушує BoxDrawingView перемалювати себе, щоб користувач бачив прямокутник в процесі перетягування.

Малювання всередині onDraw (...)

При запуску програми всі його представлення недійсні (invalid). Це означає, що вони нічого не вивели на екран. Для виправлення ситуації Android викликає метод draw () об'єкта View верхнього рівня. В результаті представлення перемальовує себе, що змушує його нащадків перемалювати себе. Потім нащадки цих нащадків перемальовували себе і так далі вниз по ієрархії. Коли все представлення в ієрархії перемалюють себе, об'єкт View верхнього рівня перестає бути недійсним.

Щоб втрутитися в процес прорисовкb, слід перевизначити наступний метод View:

```
protected void onDraw (Canvas canvas)
```

Виклик invalidate (), що виконується у відповідь на дію ACTION_MOVE в onTouchEvent (...), знову робить об'єкт BoxDrawingView недійсним. Це змушує його перемалювати себе і призводить до повторного виклику onDraw (...).

Гараметр Canvas.

Canvas і Paint - два основні класи, які використовуються при малюванні в Android.

Клас Canvas містить всі виконувані операції графічного виведення. Методи, що викликаються для об'єкта Canvas, визначають, де і що виводиться - лінія, коло, слово або прямокутник.

Клас Paint визначає, як будуть виконуватися ці операції. Методи, що викликаються для об'єкта Paint, визначають характеристики виведення - чи повинні фігури заповнюватися, яким шрифтом буде подаватись текст, яким кольором повинні виводитися лінії і т.д.

У файлі BoxDrawingView.java створіть два об'єкти Paint в конструкторі BoxDrawingView для XML.

Лістинг 8. Створення об'єктів Paint (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private ArrayList <Box> mBoxen = new ArrayList <Box> ();
    private Box mCurrentBox;
    private Paint mBoxPaint;
    private Paint mBackgroundPaint;

    ...

    // Використовується при заповненні представлення по розмітці XML
    public BoxDrawingView (Context context, AttributeSet attrs) {
        super (context, attrs);

        // Прямокутники малюються напівпрозорим червоним кольором (ARGB)
        mBoxPaint = new Paint ();
        mBoxPaint.setColor (0x22ff0000);

        // Фон зафарбовується сірувато-білим кольором
        mBackgroundPaint = new Paint ();
        mBackgroundPaint.setColor (0xfff8efe0);
    }
}
```

Після створення об'єктів Paint можна переходити до малювання прямокутників на екрані.

Лістинг 9. Перевизначення onDraw (Canvas) (BoxDrawingView.java)

```
@Override
protected void onDraw (Canvas canvas) {
    // Заповнення фону
    canvas.drawPaint (mBackgroundPaint);

    for (Box box: mBoxes) {
        float left = Math.min (box.getOrigin (). x, box.getCurrent (). x);
        float right = Math.max (box.getOrigin (). x, box.getCurrent (). x);
        float top = Math.min (box.getOrigin (). y, box.getCurrent (). y);
        float bottom = Math.max (box.getOrigin (). y, box.getCurrent (). y);
        canvas.drawRect (left, top, right, bottom, mBoxPaint);
    }
}
```

Перша частина коду тривіальна: використовуючи сірувато-білий колір, ми заповнюємо «полотно» заднім фоном для виведення прямокутників.

Потім для кожного прямокутника в списку ми визначаємо значення left, right, top і bottom по двох точках. Значення left і top будуть мінімальними, а bottom і right - максимальними.

Після обчислення параметрів виклик методу Canvas.drawRect (...) малює червоний прямокутник на екрані.

Відкрийте програму DragAndDraw і намалюйте кілька прямокутників.

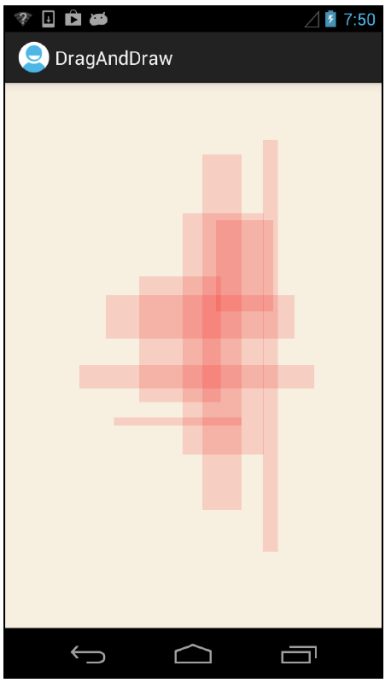

Рис. 6. Додаток з намальованими прямокутниками

Завдання. Повороти

Як зберегти прямокутники з View при зміні орієнтації? Наприклад, це можна зробити за допомогою таких методів View:

```
protected Parcelable onSaveInstanceState ()
protected void onRestoreInstanceState (Parcelable state)
```

Ці методи працюють не так, як метод onSaveInstanceState (Bundle) класів Activity і Fragment. Замість об'єкта Bundle вони повертають і обробляють об'єкт, який реалізує інтерфейс Parcelable. Рекомендується використовувати Bundle як Parcelable замість того, щоб писати реалізацію Parcelable самостійно. (Реалізація інтерфейсу Parcelable вельми складна. Краще уникати її там, де це можливо.)

Реалізуйте можливість обертання прямокутників другим пальцем. Для цього вам буде потрібно відстежувати операції з декількома покажчиками в коді обробки MotionEvent.

При роботі з множинними дотиками вам знадобляться:

- *індекс покажчика* - повідомляє, до якого покажчика в поточному наборі відноситься подія;
- *ідентифікатор покажчика* - забезпечує однозначну ідентифікацію конкретного пальця в жесті.

Індекс покажчика може змінюватися, але ідентифікатор залишається незмінним.

Методи MotionEvent:

```
public final int getActionMasked ()
public final int getActionIndex ()
public final int getPointerId (int pointerIndex)
public final float getX (int pointerIndex)
public final float getY (int pointerIndex)
```

константи `ACTION_POINTER_UP` і `ACTION_POINTER_DOWN`.

Завдання 2. Видалення з CrimeFragment

Користувачам буде зручно мати можливість видаляти запис як зі списку, так і з представлення деталізації. В цьому випадку видалення буде застосовуватися до екрану в цілому, тому така дія повинна розміщуватись в командному меню або на панелі дій.

Реалізуйте можливість видалення записів у CrimeFragment.

Список літератури

1. Харди Б. Программирование под Android. Для профессионалов / Харди Б., Филлипс Б. - СПб.: Питер, 2014. - 592 с.
2. https://uk.wikipedia.org/wiki/Android
3. Дон Гриффитс Head First. Программирование для Android / Дон Гриффитс, Дэвид Гриффитс. - СПб.: Питер, 2018. - 912 с.
4. Ян Ф. Дарвин Android. Сборник рецептов. Задачи и решения для разработчиков приложений / Ян Ф. Дарвин. - М.: Вильямс, 2017. - 768 с.
5. Пол Дейтел Android для разработчиков / Пол Дейтел, Харви Дейтел, Александер Уолд. - СПб.: Питер, 2016. - 512 с.
6. Майк МакГрат Создание приложений на Android для начинающих / Майк МакГрат. - М.: Эксмо, 2016. - 192 с.
7. Сильвен Ретабоуил Android NDK. Руководство для начинающих / Сильвен Ретабоуил. - М.: ДМК Пресс, 2016. - 518 с.
8. Android 3 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов / Сатия Коматинени, Дэйв Маклин, Саид Хашими. - М.: Вильямс, 2012. - 1024 с.
9. Андерс Ёранссон Эффективное использование потоков в операционной системе Android. Технологии асинхронной обработки данных / Андерс Ёранссон - М.: ДМК Пресс, 2017. - 304 с.
10. Тимур Машнин Сборник тестов: 1500 вопросов и ответов на знание Android / Тимур Машнин. - Издательские решения, 2015. - 650 с.