МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ
ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ АВТОМАТИКИ ТА ЕНЕРГЕТИКИ

КАФЕДРА ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

Програмування мобільних пристроїв та систем

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 мобільних пристроїв та систем: 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. – 100 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 мобільних пристроїв та систем» 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: Робота з панеллю дій

Панель дій (Action bar) з'явилася у Honeycomb. Вона замінює класичний рядок заголовка, але її функції не зводяться до простого відображення значка додатку і заголовка. На панелі дій також можуть розміщуватися команди меню, а значок програми може використовуватися як навігаційна кнопка.

Створимо меню для додатка CrimIntent. В меню буде присутньою *команда меню* для додавання нового запису. Крім того, кнопка додатку буде виконувати функції кнопки Up.

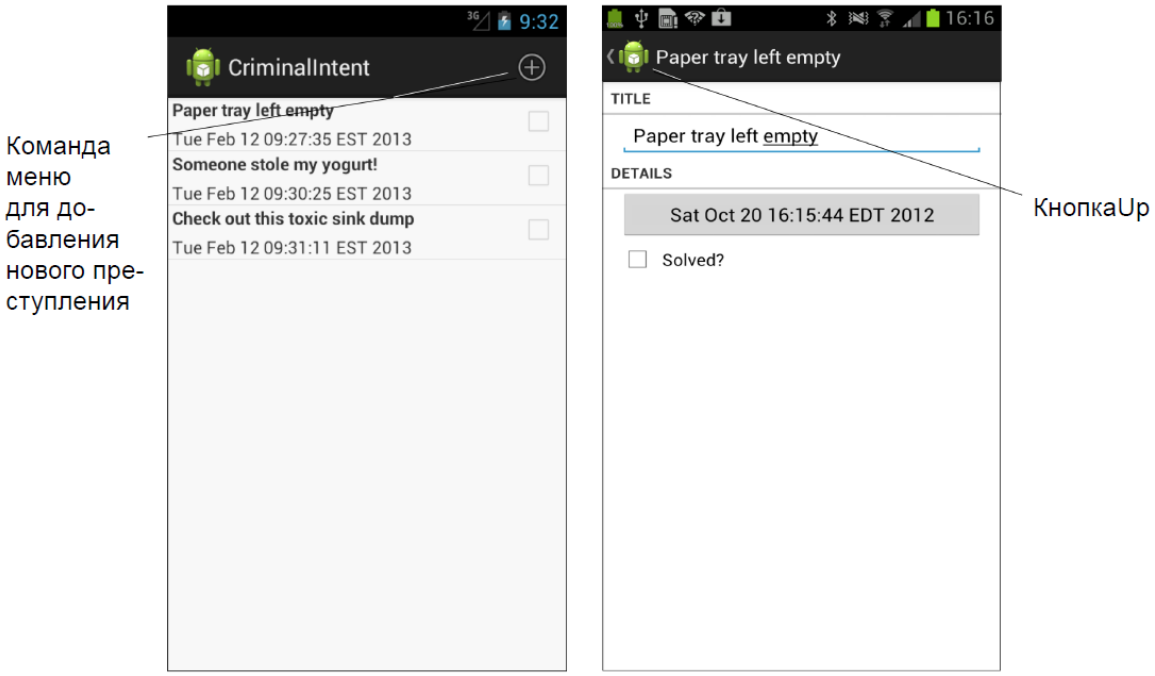

Рис. 1.Розширені можливості панелі дій

Командне меню

Меню на панелі дій називається *командним меню (options menu)*. Пункти цього меню відносяться до екрану або з додатком в цілому. Хорошим прикладом служить додавання нового запису. З іншого боку, операція видалення запису зі списку краще підходить для контекстного меню, оскільки для неї необхідний контекст - інформація про те, який запис потрібно видалити.

Нашому командному меню будуть потрібні нові рядкові ресурси. Один ресурс також знадобиться і для контекстного меню. Додайте рядкові ресурси в файл strings.xml (Лістинг 1).

Лістинг 1. Додавання рядків для меню (res/values/strings.xml)

```
    ...
    <string name="crimes_title">Crimes</string>
    <string name="crime_date_label">Date:</string>
    <string name="date_picker_title">Date of crime:</string>
    <string name="new_crime">New Crime</string>
    <string name="show_subtitle">Show Subtitle</string>
    <string name="hide_subtitle">Hide Subtitle</string>
    <string name="subtitle">If you see something, say something.</string>
    <string name="delete_crime">Delete</string>
</resources>
```

Рис. 2.Командні меню до Honeycomb

Панель дій для розміщення командних меню використовується відносно недавно, але самі командні меню існують з перших версій Android.

З командними меню майже не буває проблем сумісності. Код один і той же, а кожен пристрій вибирає спосіб представлення командних меню в залежності від рівня Android.

Визначення командного меню в XML

Меню визначаються такими ж ресурсами, як і макети. Ви створюєте опис меню в XML і ставите файл у каталог res/menu свого проекту. Android генерує ідентифікатор ресурсу для файлу меню, який потім використовується для заповнення меню в коді.

На панелі Package Explorer створіть в каталозі res/ підкаталог menu. Клацніть правою кнопкою миші на новому каталозі і виберіть команду New › Android XML File. Переконайтеся в тому, що для створюваного файлу обраний тип ресурсу Menu, і надайте файлу ім'я fragment_crime_list.xml.

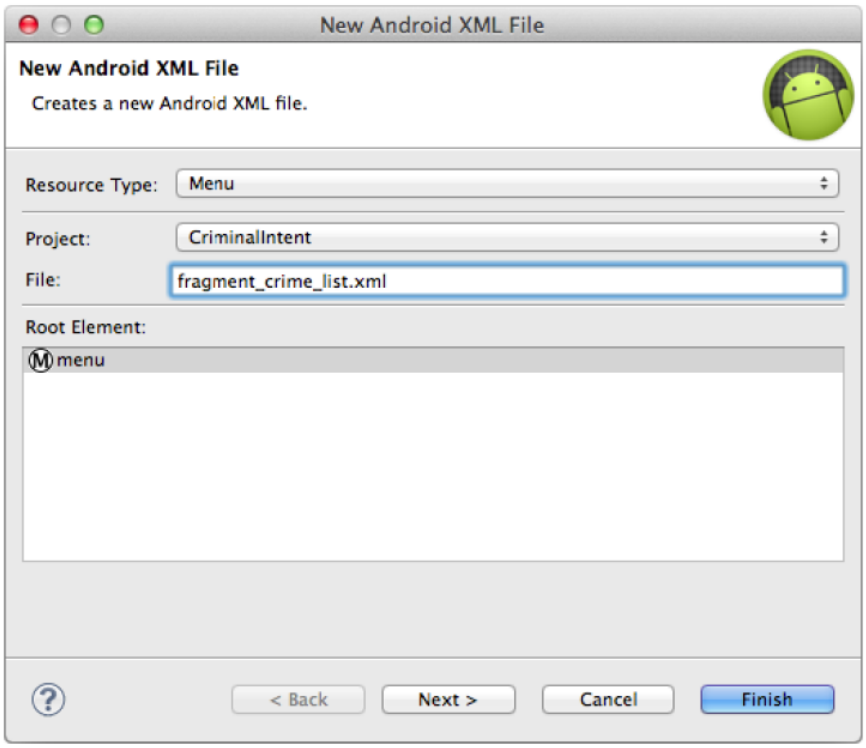

Рис. 3.Створення файлу командного меню

Відкрийте файл fragment_crime_list.xml і перейдіть до XML. Додайте елемент item, Представлений в лістингу 2.

Лістинг 2. Створення ресурсу меню CrimeListFragment (fragment_crime_list.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        android:showAsAction="ifRoom|withText"/>
</menu>
```

Атрибут showAsAction показує, чи повинна команда меню відображатися на самій панелі дій або в *додатковому* меню (overflow menu). Ми об'єднали два значення ifRoom і withText, щоб при наявності вільного місця на панелі дій відображався значок і текст команди. Якщо на панелі не вистачає місця для тексту, то відображається тільки значок. Якщо місця немає ні для того, ні для іншого, команда переміщається в додаткове меню.

Спосіб звернення до додаткового меню залежить від пристрою. Якщо у пристрої є фізична клавіша виклику меню, то для виклику додаткового меню необхідно натиснути її. Більшість нових пристроїв не має фізичної клавіші меню, тому додаткове меню викликається значком у вигляді трьох точок в правій частині панелі дій (рис. 4).

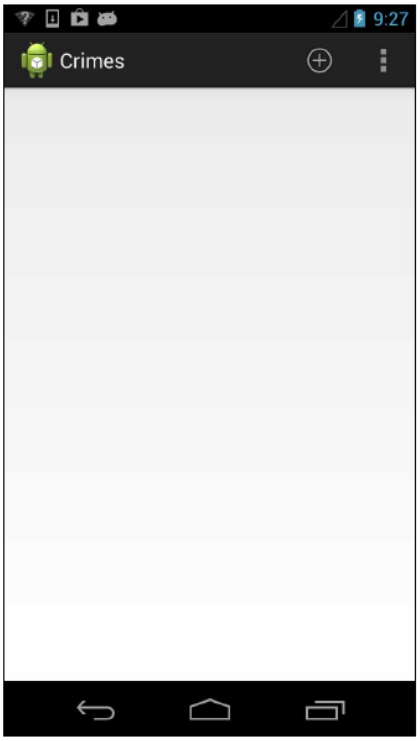

Рис. 4.Додаткове меню на панелі дій

Також атрибут showAsAction може приймати значення always і never. Вибирати always не рекомендується; краще використовувати ifRoom і надати рішення ОС. Варіант never добре підходить для рідко виконуваних дій. Як правило, на панелі дій слід розміщувати тільки часто використовувані команди меню, щоб не захаращувати екран.

Android Lint не скаржиться на атрибут android: showAsAction, що з'явився у API рівня 11. Для атрибутів XML такий захист, як для коду Java, не потрібний. Розмітка XML майбутніх версій просто ігнорується в більш ранніх API.

У атрибуті android:icon значення @android:drawable/ic_menu_add посилається на системний значок (System icon). Системні значки знаходяться на пристрої, а не в ресурсах проекту.

Використання системних значків

У прототипі посилання на системні значки працюють нормально. Однак в додатку, готовому до випуску, краще бути впевненим в тому, що саме користувач побачить на екрані. Системні значки можуть сильно відрізнятися між пристроями і версіями ОС, а на деяких пристроях системні значки можуть не відповідати дизайну програми.

Одне з можливих рішень - створення власних значків. Вам доведеться підготувати версії для кожного дозволу екрану, а можливо, і для інших конфігурацій пристроїв.

Також можна діяти інакше - знайти системні значки, що відповідають потребам вашої програми, і скопіювати їх прямо в графічні ресурси проекту. Це простий спосіб забезпечити додаток добре знайомими і зрозумілими значками.

Щоб знайти системні значки, перейдіть до каталогу установки Android SDK і знайдіть шлях виду *домашній-каталог-android-SDK/platforms/рівень-API/data/res.* Наприклад, каталог Android 4.2 на одному з наших Mac має вигляд /Developer/androidsdkmac_86/platforms/android-17/data/res.

Перегляньте системні значки для різних SDK або проведіть пошук за умовою ic_menu_add. Ви можете скопіювати результати в відповідну папку drawable ресурсів вашого проекту. Використовуйте у файлі макета атрибут icon виду android: icon = "@ drawable/ic_menu_add", щоб значок витягувався з ресурсів проекту, а не з системних значків пристрою.

Створення командного меню

Для управління командними меню в коді використовуються методи зворотного виклику класу Activity. Коли виникає необхідність в командному меню, Android викликає метод Activity з ім'ям onCreateOptionsMenu (Menu).

Однак архітектура нашого застосування вимагає, щоб реалізація знаходилася у фрагменті, а не в активності. Клас Fragment містить власний набір методів зворотного виклику для командних меню, які ми реалізуємо в CrimeListFragment. Для створення командного меню і обробки вибраних команд використовуються наступні методи:

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

У файлі CrimeListFragment.java перевизначіть метод onCreateOptionsMenu (Menu, Menu-Inflater) так, щоб він заповнював меню, визначене в файлі fragment_crime_list.xml.

Лістинг 3. Заповнення командного меню (CrimeListFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
```

У цьому методі ми викликаємо метод MenuInflater.inflate (int, Menu) і передаємо ідентифікатор ресурсу свого файлу меню. Виклик заповнює екземпляр Menu командами, визначеними у файлі.

Виклик реалізації onCreateOptionsMenu (...) суперкласу не обов'язковий, але рекомендується викликати версію суперкласу просто для дотримання загальноприйнятої схеми, щоб працювала вся функціональність командних меню, що визначається в суперкласі. В даному випадку це лише формальність - базова реалізація цього методу з Fragment не робить нічого.

FragmentManager відповідає за виклик Fragment.onCreateOptionsMenu (Menu, MenuIn-flater) при отриманні активністю зворотного виклику onCreateOptionsMenu (...) від ОС. Ви повинні явно

вказати FragmentManager, що фрагмент повинен отримати виклик onCreateOptionsMenu (...). Для цього викликається наступний метод:

```
public void setHasOptionsMenu(boolean hasMenu)
```

В методі CrimeListFragment.onCreate (...) повідомте FragmentManager, що екземпляр CrimeListFragment повинен отримувати зворотні виклики командного меню.

Лістинг 4. Виклик hasOptionsMenu (CrimeListFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);

    getActivity().setTitle(R.string.crimes_title);
    ...
```

У додатку CrimIntent з'являється командне меню (рис. 5).

Де текст командного меню? У більшості телефонів в книжковій орієнтації вистачає місця лише для значка. Текст команди відкривається довгим натисканням на значку на панелі дій.

Рис. 5. Значок команди меню на панелі дій

Мал. 6. Довге натиснення на значку на панелі дій виводить текст команди

У альбомній орієнтації на панелі дій вистачає місця як для значка, так і для тексту (рис. 7).

Рис. 7.Значок і текст на панелі дій

Якщо запустити CrimIntent на пристрої до Honeycomb, для перегляду командного меню слід натиснути фізичну клавішу меню на пристрої. Меню з'являється в нижній частині екрана. На рис. 8 зображений результат виконання тог ж коду на пристрої Gingerbread.

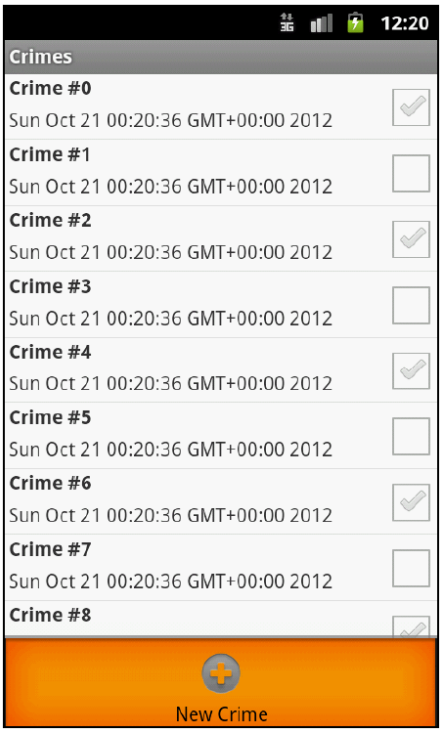

Рис. 8.Командне меню на пристрої Gingerbread

Код командного меню нормально працює і на старих, і на нових пристроях; вам не доведеться займатися упаковкою і захисної перевіркою версій. Однак існує невелика різниця в поведінці меню в нових і старих SDK. на пристроях з версією Honeycomb і вище виклик методу onCreateOptionsMenu (...) і створення командного меню відбуваються при запуску активності. Меню повинно бути готове до початку життєвого циклу активності, щоб команди меню з'явилися на панелі дій. На старих пристроях виклик onCreateOptionsMenu (...) і створення меню виконуються при першому натисканні користувачем клавіші меню.

(На старих пристроях додатки з панеллю дій зазвичай використовують сторонню бібліотеку ActionBarSherl для імітації функціональності панелі дій на більш ранніх рівнях API.)

Реакція на вибір команд

Щоб відреагувати на вибір користувачем команди New Crime, знадобиться механізм додавання нового об'єкта Crime у список. Включіть в файл CrimeLab.java наступний метод.

Лістинг 5. Додавання нового об'єкта Crime (CrimeLab.java)

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}
public ArrayList<Crime> getCrimes() {
    return mCrimes;
}
...
```

Тепер, коли ви зможете вводити опис записів самостійно, програмне генерування 100 об'єктів стає зайвим. У файлі CrimeLab.java видаліть код, що генерує ці записи.

Лістинг 6. Видалення випадкових записів (CrimeLab.java)

```
public CrimeLab(Context appContext) {
    mAppContext = appContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setDate(new Date());
        c.setSolved(i % 2 == 0); // Каждое второе
        mCrimes.add(c);
    }
}
```

Коли користувач вибирає команду в командному меню, фрагмент отримує зворотний виклик методу onOptionsItemSelected (MenuItem). Цей метод отримує екземпляр MenuItem, що описує вибір користувача.

І хоча наше меню складається всього з однієї команди, в реальних меню зазвичай більше. Щоб визначити, яка команда меню була обрана, перевірте ідентифікатор команди меню і відреагуйте відповідним чином. Цей ідентифікатор відповідає ідентифікатору, призначеного команді в файлі меню.

У файлі CrimeListFragment.java реалізуйте метод onOptionsItemSelected (MenuItem), що реагує на вибір команди меню. Реалізація створює новий об'єкт Crime, додає його в CrimeLab і запускає екземпляр CrimePagerActivity для редагування нового об'єкта Crime.

Лістинг 7. Реакція на вибір команди меню (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    Включение иерархической навигации 275
    case R.id.menu_item_new_crime:
        Crime crime = new Crime();
        CrimeLab.get(getActivity()).addCrime(crime);
        Intent i = new Intent(getActivity(), CrimePagerActivity.class);
        i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
        startActivityForResult(i, 0);
```

```
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
    }
```

Метод повертає логічне значення. Після того як команда меню буде оброблена, поверніть true; тим самим ви повідомляєте, що подальша обробка не потрібна. Секція default викликає реалізацію суперкласу, якщо ідентифікатор команди ніхто не знає в вашої реалізації.

Запустіть додаток CrimIntent і випробуйте нову команду. Додайте кілька записів і відредагуйте їх.

Включення ієрархічної навігації

До теперішнього моменту додаток CrimIntent використовує кнопку Back для навігації за додатком. Кнопка Back повертає додаток до попереднього стану. З іншого боку, *ієрархічна навігація* здійснює переміщення по ієрархії додатку.

Android дозволяє легко використовувати значок програми на панелі дій для ієрархічної навігації. Він може здійснювати повернення до початкового вікна служби через всю ієрархію (спочатку значок називався «кнопкою Home» додатка). Однак зараз Android рекомендує реалізувати значок програми так, щоб перехід здійснювався на один рівень «наверх» до батьків поточної активності. При дотриманні цієї рекомендації значок стає «кнопкою Up»).

Реалізуємо функціональність кнопки Up для значка додатка на панелі дій CrimePagerActivity. Натискаючи значок, користувач буде повертатися до списку записів.

Включення значка додатка

Щоб повідомити, що значок програми виконує функції кнопки Up, розробник зазвичай відображає зліва від нього трикутну стрілку, що вказує вліво, як на рис. 9.

Щоб значок програми працював як кнопка, а в представленні фрагмента відображалася стрілка, слід встановити властивість фрагмента викликом наступного методу:

```
public abstract void setDisplayHomeAsUpEnabled(boolean showHomeAsUp)
```

Цей метод відноситься до API рівня 11, тому його слід укласти в конструкцію перевірки, щоб додаток було сумісним з Froyo і Gingerbread. Також метод слід позначити анотацією, щоб скасувати попередження Android Lint.

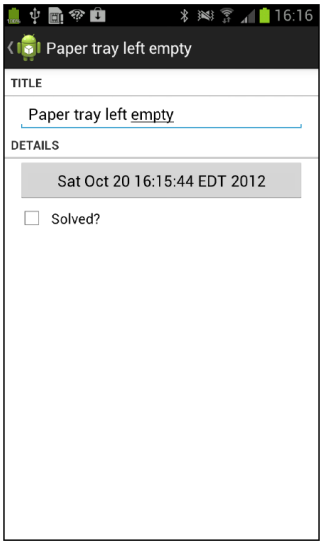

Рис. 9.Панель дій з включеною кнопкою Up

У методі CrimeFragment.onCreateView (...) викличте setDisplayHomeAsUpEnabled (true).

Лістинг 8. Включення кнопки Up (CrimeFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
    }
    ...
}
```

Завдання цієї властивості не активізує виконання функцій кнопки Up. Воно всього лише дозволяє значку діяти як кнопці і відображає стрілку. Підключення доведеться виконувати самостійно. Для додатків, розрахованих на API рівнів 11-13, значок за замовчуванням активізується, але для появи стрілки все одно необхідно викликати setDisplayHomeAsUpEnabled (true).

У лістингу 8 onCreateView (...) забезпечується анотацією @TargetApi. В принципі можна обмежитися анотацією тільки setDisplayHomeAsUpEnabled (true), але метод onCreateView (...) скоро буде містити велику кількість коду, прив'язаного до конкретної версії API, тому ми замість цього помічаємо весь метод.

Запустіть додаток CrimIntent, перейдіть на екран деталізації і переконайтеся в тому, що поруч із міткою додатку з'являється трикутна стрілка.

Обробка кнопки Up

Обробка активного значка додатка проводиться так, як якщо б він був існуючою командою меню - перевизначенням onOptionsItemSelected (MenuIt em). Отже, перш за все слід повідомити FragmentManager, що CrimeFragment реалізує зворотні виклики командного меню від імені активності.

Включіть в метод CrimeFragment.onCreate (...) виклик setHasOptionsMenu (true), Як це було зроблено раніше для CrimeListFragment.

Лістинг 9. Включення обробки меню (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    setHasOptionsMenu(true);
}
```

Вам не доведеться визначати або заповнювати команду значка додатка з файлу XML. У неї є готовий ідентифікатор ресурсу :android.R.id.home. У файлі CrimeFragment.java перевизначіть метод onOptionsItemSelected (MenuItem), щоб він реагував на цю команду.

Лістинг 10. Обробка на команду меню значка додатка (Home)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Кнопка повинна повертати користувача до списку. Звичайно, можна створити інтент і запустити екземпляр CrimePagerActivity:

```
Intent intent = new Intent(getActivity(), CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

Прапор FLAG_ACTIVITY_CLEAR_TOP наказує Android провести пошук існуючого примірника активності в стеці, і якщо він буде знайдений - вивести з стека всі інші активності, щоб активність, що запускається була верхньою.

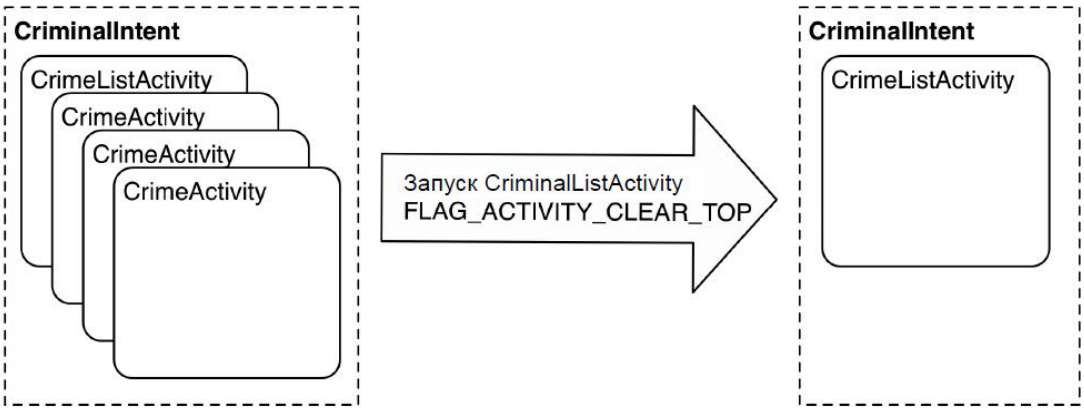

Рис. 10.FLAG_ACTIVITY_CLEAR_TOP в дії

Однак існує інший, більш правильний спосіб реалізації ієрархічної навігації, заснований на використанні допоміжного класу NavUtils і включення метаданих в маніфест.

Почнемо з метаданих. Відкрийте файл AndroidManifest.xml. Додайте в оголошення CrimePagerActivity наступний атрибут, який призначає CrimeListActivity його батьком.

Лістинг 11. Додавання метаданих батьківської активності (AndroidManifest.xml)

```
<activity android:name=".CrimePagerActivity"
    android:label="@string/app_name">
    <meta-data android:name="android.support.PARENT_ACTIVITY"
        android:value=".CrimeListActivity"/>
</activity>
...
```

Тег метаданих - свого роду «листок для нотаток», приклеєний до активності. Такі дописи зберігаються в системному об'єкті PackageManager, і будь-хто може отримати значення з «нотатку», якщо він знає її ім'я. Ви можете створювати власні пари «ім'я-значення» і читати їх дані в міру необхідності. Цю конкретну пару визначає клас NavUtils, щоб він міг впізнати батька заданої активності. Вона особливо корисна в поєднанні з наступним методом класу NavUtils:

```
public static void navigateUpFromSameTask(Activity sourceActivity)
```

У методі CrimeFragment.onOptionsItemSelected (...) спочатку перевірте, чи існує батьківська активність, позначена в метаданих, за допомогою виклику NavUtils.getParentActivityName (Activity). Якщо вона існує, викличте navigateUpFromSameTask (Activity) для переходу до батьківської активності.

Лістинг 12. Використання NavUtils (CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
        switch (item.getItemId()) {
            case android.R.id.home:
                if (NavUtils.getParentActivityName(getActivity()) != null) {
                    NavUtils.navigateUpFromSameTask(getActivity());
                }
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

Якщо інформація про батька в метаданих відсутня, відображати стрілку не потрібно. Поверніться до onCreateView (...) і перевірте наявність батька перед викликом setDisplayHomeAsUpEnabled (true).

Лістинг 13. Немає батька - немає стрілки (CrimeFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
      View v = inflater.inflate(R.layout.fragment_crime, parent, false);

      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
          if (NavUtils.getParentActivityName(getActivity()) != null) {
              getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
          }
      }
      ...
}
```

Чому використання NavUtils краще самостійного запуску активності? По-перше, код NavUtils компактний і зрозумілий. Крім того, використання NavUtils забезпечує централізацію відносин між активностями в маніфесті. Якщо ці відносини зміняться, досить змінити рядок в маніфесті замість того, щоб возитися з кодом Java.

Інша перевага полягає в тому, що ієрархічні відносини відокремлюються від коду фрагмента. CrimeFragment може використовуватися в різних активностях, які можуть мати різних батьків; CrimeFragment все одно буде працювати правильно.

Запустіть додаток CrimIntent. Створіть запис і натисніть на значку додатка, щоб повернутися до списку записів. Може, з дворівневою ієрархії CrimIntent це і не очевидно, але метод navigateUpFromSameTask (Activity) реалізує функціональність «переходу наверх» і піднімає користувача на один рівень до батька CrimePagerActivity.

Альтернативна команда меню

Створення альтернативного файлу меню

Команда меню, що застосовується до панелі дій, не повинна бути видимою користувачам без панелі дій. Отже, першим кроком має бути створення альтернативного ресурсу меню. Створіть в каталозі res проекту папку menu-v11. Скопіюйте та вставте файл fragment_crime_list.xml в цю папку.

У файлі res/menu-v11/fragment_crime_list.xml додайте команду меню Show Subtitle, яка буде відображатися на панелі дій при наявності вільного місця.

Лістинг 14. Додавання команди меню Show Subtitle
(res/menu-v11/ fragment_crime_list.xml)

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

```
        <item android:id="@+id/menu_item_new_crime"
            android:icon="@android:drawable/ic_menu_add"
            android:title="@string/new_crime"
            android:showAsAction="ifRoom|withText"/>
        <item android:id="@+id/menu_item_show_subtitle"
            android:title="@string/show_subtitle"
            android:showAsAction="ifRoom"/>
</menu>
```

Додайте в метод onOptionsItemSelected (...) обробку команди меню - включення підзаголовка на панелі дії.

Лістинг 15. Обробка команди меню Show Subtitle (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Ми тільки скасовуємо попередження Android Lint, а не укладаємо код панелі дій в перевірочну конструкцію. Перевірка не потрібна - цей код не може викликатися на старих пристроях, тому що команда меню R.id. menu_item_show_subtitle на них відображатися не буде.

Запустіть додаток CrimIntent на новому пристрої і включіть підзаголовок. Потім запустіть його на пристрої Froyo або Gingerbread (фізичному або віртуальному). Натисніть кнопку меню і переконайтеся в тому, що команда Show Subtitlene відображається. Додайте новий запис і переконайтеся в тому, що додаток працює так само, як раніше.

Перемикання тексту команди

Тепер підзаголовок відображається, але текст команди меню залишився незмінним: Show Subtitle. Було б краще, якби текст команди і функціональність команди меню змінювалися в залежності від поточного стану підзаголовка.

У методі onOptionsItemSelected (...) перевірте наявність підзаголовка при виборі команди меню і виконайте відповідні дії.

Лістинг 16. Обробка в залежності від наявності підзаголовка (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                item.setTitle(R.string.hide_subtitle);
            } else {
                getActivity().getActionBar().setSubtitle(null);
                item.setTitle(R.string.show_subtitle);
```

```
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Якщо панель дій не містить підзаголовка, ми включаємо підзаголовок і замінюємо текст команди меню на Hide Subtitle. Якщо підзаголовок вже відображається, то він відключається, а команді меню повертається текст Show Subtitle.

Запустіть додаток CrimIntent і переконайтеся в тому, що підзаголовок успішно ховається і відображається.

Повороти

Якщо відобразити підзаголовок, а потім повернути пристрій, підзаголовок зникне при створенні інтерфейсу «з нуля». Для рішення цієї проблеми потрібно визначити поле для ознаки видимості підзаголовка, а також зберегти CrimeListFragment, щоб значення змінної не втрачалося при поворотах.

У файлі CrimeListFragment.java додайте логічну змінну, потім в методі onCreate (...) збережіть CrimeListFragment і ініціалізуйте змінну.

Лістинг 17. Ініціалізація змінних і збереження CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment { private ArrayList<Crime>
mCrimes;
    private boolean mSubtitleVisible;
    private final String TAG = "CrimeListFragment";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        setRetainInstance(true);
        mSubtitleVisible = false;
}
```

Потім в методі onOptionsItemSelected (...) задайте цю змінну при обробці вибору команди меню.

Лістинг 18. Присвоєння змінної subtitleVisible при обробці команди меню (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                mSubtitleVisible = true;
                item.setTitle(R.string.hide_subtitle);
            }
            else {
                getActivity().getActionBar().setSubtitle(null);
                mSubtitleVisible = false;
                item.setTitle(R.string.show_subtitle);
```

```
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Залишилося перевірити, чи може відображатися підзаголовок. У файлі CrimeListFragment.java перевизначіть метод onCreateView (...) і призначте підзаголовок, якщо змінна mSubtitleVisible містить true.

Лістинг 19. Підзаголовок призначається, якщо поле mSubtitleVisible істинно (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (mSubtitleVisible) {
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
        }
    }
    return v;
}
```

Також необхідно перевірити стан підзаголовка в onCreateOptionsMenu (...) і переконатися в тому, що відображається правильний текст команди меню.

Лістинг 20. Призначення тексту команди меню з істинним значенням mSubtitleVisible (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
    MenuItem showSubtitle = menu.findItem(R.id.menu_item_show_subtitle);
    if (mSubtitleVisible && showSubtitle != null) {
        showSubtitle.setTitle(R.string.hide_subtitle);
    }
}
```

Запустіть додаток CrimIntent. Відкрийте підзаголовок, поверніть пристрій. Підзаголовок повинен з'явитися в відтвореному представленні, як і очікувалося.

Завдання 1. Пусте представлення для списку

При запуску CrimIntent відображає порожній список - велику чорну порожнечу. Надайте користувачам щось для взаємодії при відсутності елементів в списку.

Так як клас ListView є субкласом AdapterView, він підтримує спеціальний варіант View, Званий «порожнім представленням» і існуючий саме для таких ситуацій. Якщо задати порожнє представлення, ListView буде автоматично перемикатися між цим представленням за відсутності елементів списку і відображенням списку, якщо в ньому є хоча б один елемент.

Для зміни порожнього представлення в коді використовується наступний метод AdapterView:

```
public void setEmptyView(View emptyView)
```

Також можна створити в розмітці XML макет, в якому задається як ListView, так і порожнє представлення. Якщо призначити їм ідентифікатори ресурсів @android:id/list і @android:id/empty відповідно, буде використовуватися функція автоматичного перемикання.

Поточна реалізація CrimeListFragment не заповнює свій макет у onCreateView (...), але для реалізації порожнього представлення в макеті це необхідно. створіть для CrimeListFragment XML-ресурс макета, який використовує FrameLayout в якості кореневого контейнера, з представленням ListView і іншим представленням View, яке буде використовуватися для порожнього списку.

Нехай в порожньому представленні виводиться повідомлення (наприклад, «Список порожній»). Додайте в представлення кнопку, яка буде ініціювати створення нового запису, щоб користувачеві не довелося звертатися до командного меню або панелі дій.

Завдання 2. Використання зовнішнього сховища

Файли у внутрішньому сховищі добре підходять для більшості додатків, особливо якщо ви турбуєтеся про конфіденційність даних. Тим не менш деякі додатки можуть записувати дані в зовнішнє сховище пристрою, якщо воно встановлене.

Наприклад, зовнішнє сховище зручно для передачі даних (музики, фотографій, завантаженої з Інтернету інформації) іншим програмам або користувачам. Крім того, зовнішнє сховище майже завжди має великий обсяг, і в ньому зазвичай зберігаються великі файли (наприклад, відео).

Для запису у зовнішнє сховище необхідно зробити дві речі. Спочатку переконайтеся у тому, що зовнішнє сховище доступно. Клас android.os.Environment містить кілька корисних методів і констант, які допоможуть вам в цьому. Потім отримайте дескриптор каталогу зовнішніх файлів (знайдіть метод класу Context, який надасть вам цю інформацію).

Лабораторна робота № 2

Тема: Зйомка і обробка зображень

Отримання знімка

Почнемо з поновлення макета CrimeCameraFragment - в нього буде додано індикатор прогресу. Зйомка може зайняти деякий час.

Додайте в файл layout/fragment_crime_camera.xml віджети FrameLayout і ProgressBar, показані на рис. 2.

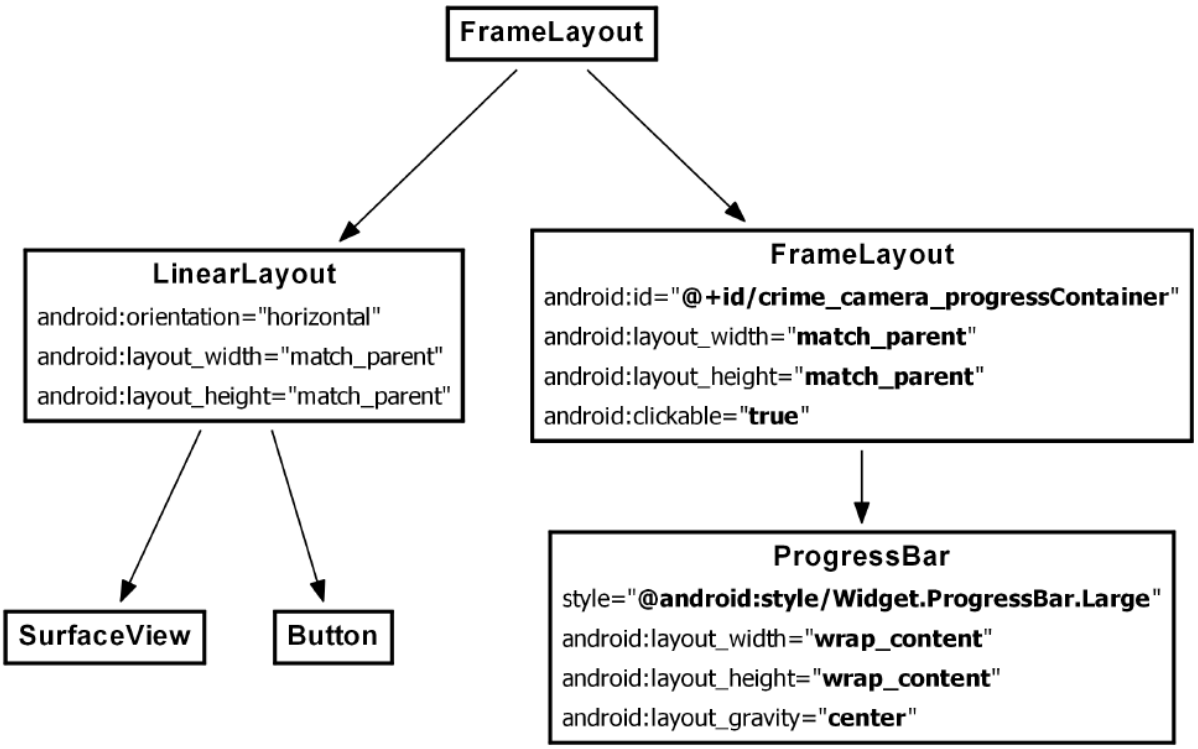

Рис. 2. Додавання віджетів FrameLayout і ProgressBar (fragment_crime_camera.xml)

Стиль @android:style/Widget.ProgressBar.Large створює великий кругової індикатор невизначеної тривалості.

Віджет FrameLayout (і його нащадок ProgressBar) спочатку невидимі. Вони стають видимими тільки після того, як користувач натисне кнопку Take! і почнеться процес зйомки.

Ширина і висота FrameLayout визначаються значенням match_ parent. Кореневий елемент FrameLayout накладає свої дочірні представлення поверх один одного в порядку їх визначення. Таким чином, віджет FrameLayout, що містить ProgressBar, повністю закриє свого сусіда LinearLayout.

Коли віджет FrameLayout стає видимим, користувач як і раніше зможе бачити майже весь вміст LinearLayout. Тільки віджет ProgressBar буде приховувати частину зображення. Однак визначення ширини і висоти FrameLayout у режимі match_parent і завдання властивості android: clickable = "true" гарантує, що FrameLayout буде перехоплювати всі події торкань (і нічого не робити з ними). Тим самим запобігає взаємодію користувача з вмістом LinearLayout, і зокрема повторне натискання кнопки Take !.

Поверніться до файлу CrimeCameraFragment.java. Додайте поле для віджета FrameLayout, отримаєте посилання на віджет і позначте його як невидимий.

Лістинг 1. Підключення FrameLayout (CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {
    ...
    private View mProgressContainer;

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

      View v = inflater.inflate (R.layout.fragment_crime_camera, parent,
    false);
      mProgressContainer = v.findViewById
    (R.id.crime_camera_progressContainer);
      mProgressContainer.setVisibility (View.INVISIBLE);

      ...

      return v;
    }
...
}
```

Реалізація зворотних викликів камери

Індикатор прогресу працює; можна переходити до захоплення кадру з видошукача і збереженню його в форматі JPEG. Для цього використовується наступний метод класу Camera:

```
public final void takePicture (Camera.ShutterCallback shutter,
    Camera.PictureCallback raw,
    Camera.PictureCallback jpeg)
```

Виклик ShutterCallback відбувається при захопленні зображення з камери, але до того, як дані зображення будуть оброблені і стануть доступними. Перший виклик PictureCallback відбувається при появі необроблених графічних даних і зазвичай використовується для попередньої обробки перед збереженням. Другий виклик PictureCallback відбувається при появі доступної JPEG-версії зображення.

Ви можете написати реалізації цих інтерфейсів і передати їх takePicture (...). Якщо який-небудь метод зворотного виклику не використовується, передайте null в відповідному параметрі takePicture (...).

У CrimIntent ми реалізуємо ShutterCallback і метод зворотного виклику для зображення JPEG.

На рис. 4 зображена схема взаємодій між об'єктами.

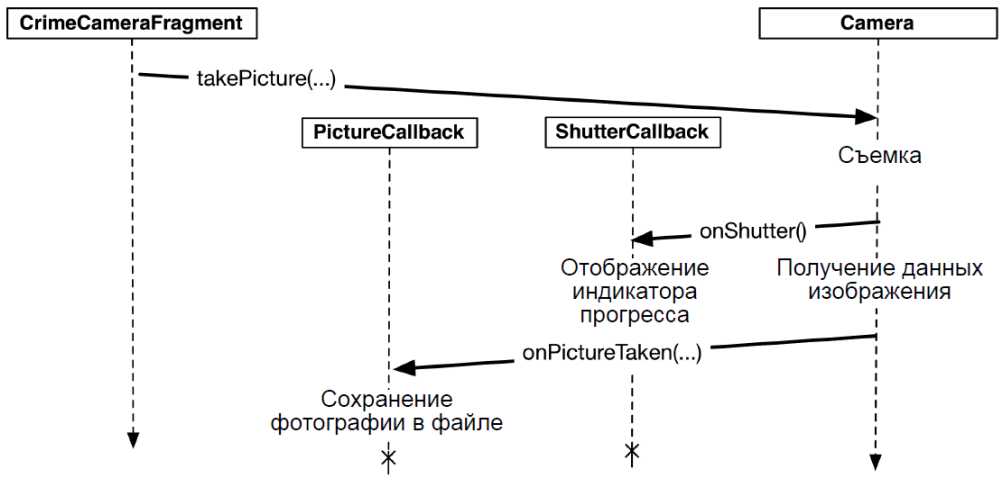

Рис. 4. Отримання фотографії в CrimeCameraFragment

У схемі задіяні два інтерфейси, кожен з яких містить один метод.

```
public static interface Camera.ShutterCallback {
   public abstract void onShutter ();
}
public static interface Camera.PictureCallback {
   public abstract void onPictureTaken (byte [] data, Camera camera);
}
```

Додайте в файл CrimeCameraFragment.java реалізацію Camera.ShutterCallback, яка відображає індикатор прогресу, і реалізацію Camera.PictureCallback, яка обробляє вибір імені і збереження файлу JPEG.

Лістинг 2. Реалізація зворотних викликів в takePicture (...) (CrimeCameraFragment.java)

```
...
private View mProgressContainer;

private Camera.ShutterCallback mShutterCallback = new Camera.ShutterCallback ()
{
    public void onShutter () {
    // Відображення індикатора прогресу
    mProgressContainer.setVisibility (View.VISIBLE);
    }
};

private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback () {
    public void onPictureTaken (byte [] data, Camera camera) {
    // Створення імені файлу
    String filename = UUID.randomUUID (). ToString () + ".jpg";
    // Збереження даних jpeg на диску
    FileOutputStream os = null;
    boolean success = true;
    try {
        os = getActivity (). openFileOutput (filename, Context.MODE_PRIVATE);
        os.write (data);
    } Catch (Exception e) {
        Log.e (TAG, "Error writing to file" + filename, e);
        success = false;
    } Finally {
        try {
          if (os! = null)
             os.close ();
        } Catch (Exception e) {
          Log.e (TAG, "Error closing file" + filename, e);
          success = false;
        }
    }

    if (success) {
        Log.i (TAG, "JPEG saved at" + filename);
    }
    getActivity (). finish ();
    }
};
...
```

У методі onPictureTaken (...) створюється унікальний рядок, який використовується для імені файлу. Потім класи введення-виведення Java використовуються для відкриття вихідного потоку і запису даних JPEG, отриманих від Camera. Якщо операція пройшла нормально, то в протоколі повідомлення про успіх.

Ми не повертаємо mProgressContainer в невидимий стан. Це не потрібно, тому що в зворотному виклику PictureCallback активність завершується, а представлення знищується.

Після підготовки зворотних викликів змініть слухача кнопки Take !, щоб він викликав takePicture (...). Передайте null замість зворотного виклику необроблених даних, який ми не реалізували.

Лістинг 3. Виклик takePicture (...) після клацання на кнопці (CrimeCameraFragment.java)

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

         takePictureButton.setOnClickListener (new View.OnClickListener () {
            public void onClick (View v) {
               getActivity (). finish ();
               if (mCamera! = null) {
                  mCamera.takePicture (mShutterCallback, null, mJpegCallback);
               }
            }
      });
      ...

      return v;
}
```

Призначення розміру зображення

Камера повинна знати розмір створюваного зображення. Розмір фотографії задається так само, як розмір області попереднього перегляду. Для отримання списку допустимих розмірів зображення використовується метод класу Camera.Parameters:

```
      public List <Camera.Size> getSupportedPictureSizes ()
```

У методі surfaceChanged (...) ми використовуємо метод getBestSupportedSize (...) для визначення підтримуваного розміру зображення відповідно до вашому екземпляру Surface. Далі задається розмір зображення камери.

Лістинг 4. Виклик getBestSupportedSize (...) для завдання розміру зображення (CrimeCameraFragment.java)

```
      ...
      public void surfaceChanged (SurfaceHolder holder, int format, int w, int h) {
          if (mCamera == null) return;
          // Розмір поверхні змінився;
          // оновити розмір області попереднього перегляду камери
          Camera.Parameters parameters = mCamera.getParameters ();
          Size s = getBestSupportedSize (parameters.getSupportedPreviewSizes (), w,
                   h);
          parameters.setPreviewSize (s.width, s.height);
          s = getBestSupportedSize (parameters.getSupportedPictureSizes (), w, h);
          parameters.setPictureSize (s.width, s.height);
          mCamera.setParameters (parameters);
          ...
      }
});
```

Відкрийте програму CrimIntent і натисніть кнопку Take !. Щоб побачити, де виявилося зображення, створіть в LogCat фільтр з тегом CrimeCameraFragment.

У цій точці CrimeCameraFragment надає користувачеві можливість зберегти зображення, отримане від камери. Наша робота з API камери закінчена.

Передача даних CrimeFragment

Щоб надати CrimeFragment доступ до фотографій, CrimeCameraFragment буде повертати ім'я файлу. На рис. 5 зображена послідовність подій взаємодії між CrimeFragment і CrimeCameraFragment.

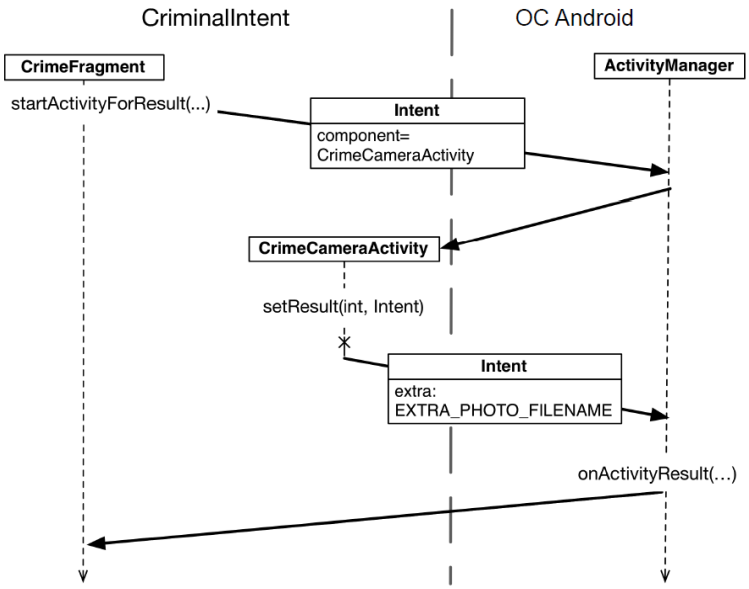

Рис. 5. CrimeCameraActivity призначає результат і доповнення

Спочатку CrimeFragment запускає CrimeCameraActivity з поверненням результату. При отриманні зображення CrimeCameraFragment створює інтент з доповненнями третьому, що містить ім'я файлу, і викликає setResult (...). ActivityManager передає інтент CrimePagerActivity в методі onActivityResult (...). Далі екземпляр FragmentManager, що належить CrimePagerActivity, передає інтент CrimeFragment в CrimeFragment.onActivityResult (...).

Запуск CrimeCameraActivity з поверненням результату

У поточній версії CrimeFragment просто запускає CrimeCameraActivity. У файлі CrimeFragment.java додайте константу для коду запиту, а потім змініть слухача кнопки камери, щоб він запускав CrimeCameraActivity з поверненням результату.

Лістинг 5. Запуск CrimeCameraActivity з поверненням результату (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private static final int REQUEST_DATE = 0;
    private static final int REQUEST_PHOTO = 1;
    ...
    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
    ...
    mPhotoButton.setOnClickListener (new View.OnClickListener () {
       public void onClick (View v) {
           // Запуск активності камери
           Intent i = new Intent (getActivity (), CrimeCameraActivity.class);
           startActivity (i);
           startActivityForResult (i, REQUEST_PHOTO);
       }
    });
    ...
}
```

Призначення результату в CrimeCameraFragment

CrimeCameraFragment поміщає ім'я файлу на додаток ІНТЕНТ і передає його при виклику CrimeCameraActivity.setResult (int, Intent). У файлі CrimeCameraFragment. java додайте константу для доповнення, потім у зворотному виклику зображення створіть інтент і задайте код результату RESULT_OK, якщо зображення було збережено, або RESULT_CANCELED, якщо виникли якісь проблеми.

Лістинг 6. Створення доповнення для імені файлу з фотографією (CrimeCamera Fragment.java)

```
public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    public static final String EXTRA_PHOTO_FILENAME =
      "Com.bignerdranch.android.Crimintent.photo_filename";

    ...

    private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback
    () {
      public void onPictureTaken (byte [] data, Camera camera) {
          ...
          try {
            ...
          } Catch (Exception e) {
            ...
          } Finally {
            ...
          }
          Log.i (TAG, "JPEG saved at" + filename);
          // Файл фотографії записується в інтент результату
          if (success) {
            Intent i = new Intent ();
            i.putExtra (EXTRA_PHOTO_FILENAME, filename);
            getActivity (). setResult (Activity.RESULT_OK, i);
          } Else {
            getActivity (). setResult (Activity.RESULT_CANCELED);
          }
          getActivity (). finish ();
    }
};
```

Отримання імені файлу в CrimeFragment

В майбутньому CrimeFragment буде використовувати ім'я файлу для поновлення рівнів моделі та представлення CrimIntent.

А поки в файлі CrimeFragment.java просто перевизначите метод onActivityResult (...), щоб він перевіряв результат, отримував ім'я файлу і зберігав в журналі повідомлення. Нам також знадобиться мітка TAG для CrimeFragment, що полегшує пошук в журналі.

Лістинг 7. Отримання імені файлу (CrimeFragment.java)

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

```
    @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);
            updateDate ();
        } Else if (requestCode == REQUEST_PHOTO) {
            // Створення нового об'єкта Photo і зв'язування його з Crime
            String filename = data
               .getStringExtra (CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename! = null) {
               Log.i (TAG, "filename:" + filename);
            }
        }
}
```

Відкрийте програму CrimIntent і зробіть знімок в CrimeCameraActivity. Перевірте дані LogCat і переконайтеся в тому, що екземпляр CrimeFragment отримав ім'я файлу.

Тепер, коли CrimeFragment знає ім'я файлу, необхідно виконати:

- Оновлення рівня моделі: ми напишемо клас Photo, що інкапсулює ім'я файлу зображення. Також в клас Crime буде додано властивість mPhoto типу Photo. CrimeFragment використовує ім'я файлу для створення об'єкта Photo і завдання властивості mPhoto класу Crime.
- Оновлення представлення CrimeFragment: ми додамо в макет CrimeFragment віджет ImageView і виведемо на ньому мініатюру фотографії Crime.
- Для виведення збільшеної версії зображення ми створимо субклас DialogFragment з ім'ям ImageFragment і передамо йому шлях до відображуваної фотографії.

Оновлення рівня моделі

На рис. 6 зображена схема відносин між CrimeFragment, Crime і Photo.

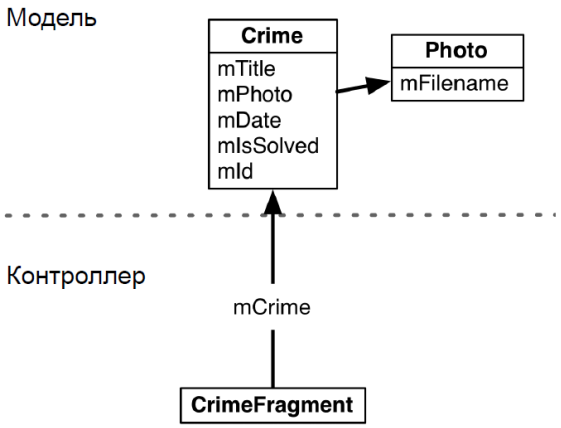

Рис. 6. Об'єкти моделі і CrimeFragment

Додавання класу Photo

Створіть новий клас в пакеті com.bignerdranch.android.Crimintent. Присвойте йому ім'я Photo і залиште йому суперклас java.lang.Object.

Додайте в файл Photo.java поля і методи, наведені в лістингу 8.

Лістинг 8. Клас Photo (Photo.java)

```
...
public class Photo {
    private static final String JSON_FILENAME = "filename";

    private String mFilename;
    / ** Створення об'єкта Photo, що представляє файл на диску * /
    public Photo (String filename) {
      mFilename = filename;
    }

    public Photo (JSONObject json) throws JSONException {
      mFilename = json.getString (JSON_FILENAME);
    }
    public JSONObject toJSON () throws JSONException {
      JSONObject json = new JSONObject ();
      json.put (JSON_FILENAME, mFilename);
      return json;
    }
    public String getFilename () {
      return mFilename;
    }
}
```

У класі Photo два конструктора. Перший конструктор створює об'єкт Photo по заданому імені файлу, а другий являє собою метод серіалізациі JSON, який використовується класом Crime для збереження і завантаження властивості типу Photo.

Включення в Crime властивості для зберігання фотографії

Включіть в клас Crime поле для об'єкта Photo, який буде серіалізований у формат JSON (лістинг 9).

Лістинг 9. Визначення поля Photo в об'єкті Crime (Crime.java)

```
public class Crime {
    ...
    private static final String JSON_DATE = "date";
    private static final String JSON_PHOTO = "photo";

    ...
    private Date mDate = new Date ();
    private Photo mPhoto;

    ...
    public Crime (JSONObject json) throws JSONException {
    ...
        mDate = new Date (json.getLong (JSON_DATE));
        if (json.has (JSON_PHOTO))
            mPhoto = new Photo (json.getJSONObject (JSON_PHOTO));
    }

    public JSONObject toJSON () throws JSONException {
        JSONObject json = new JSONObject ();
        ...
        json.put (JSON_DATE, mDate.getTime ());
        if (mPhoto! = null)
            json.put (JSON_PHOTO, mPhoto.toJSON ());
        return json;
    }
```

```
    ...
    public Photo getPhoto () {
       return mPhoto;
    }
    public void setPhoto (Photo p) {
       mPhoto = p;
    }
}
```

Збереження посилання на фотографію

У файлі CrimeFragment.java внесіть зміни в метод onActivityResult (...), щоб він створював новий екземпляр Photo і пов'язував його з поточним екземпляром Crime.

Лістинг 10. Створення екземпляра Photo

```
@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);
        updateDate ();
    } Else if (requestCode == REQUEST_PHOTO) {
    // Створення нового об'єкта Photo і зв'язування його з Crime
    String filename = data
        .getStringExtra (CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
    if (filename! = null) {
        Log.i (TAG, "filename:" + filename);

        Photo p = new Photo (filename);
        mCrime.setPhoto (p);
        Log.i (TAG, "Crime:" + mCrime.getTitle () + "has a photo");
    }
}
}
```

Відкрийте програму CrimIntent і зробіть знімок. За даними LogCat переконайтеся в тому, що до Crime тепер додається фотографія.

Чому ми створюємо клас Photo замість того, щоб просто додати в Crime властивість для імені файлу? Таке рішення спрацює в нашій ситуації, але не можна виключати, що буде потрібно від об'єкта Photo щось ще - наприклад, виведення заголовка або обробка події торкання. Для таких випадків потрібен окремий клас.

Оновлення представлення CrimeFragment

Від поновлення рівня моделі можна перейти до оновлення рівня представлення.
Клас CrimeFragment буде відображати мініатюру фотографії в віджеті ImageView.

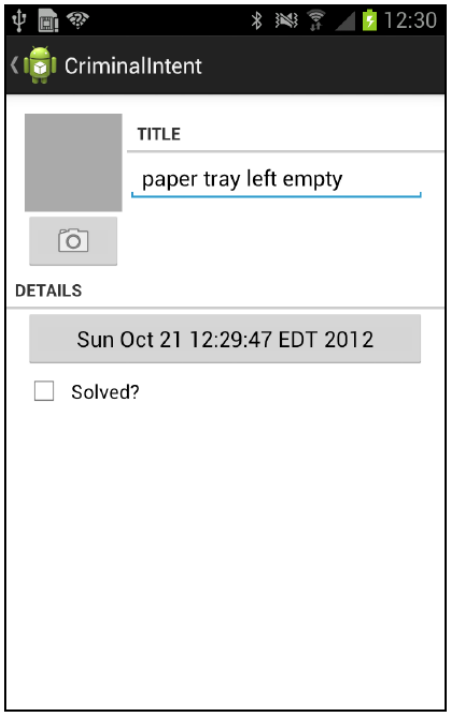

Рис. 7. CrimeFragment з новим віджетом ImageView

Додавання ImageView

Поверніться до файлу layout/fragment_crime.xml і додайте віджет ImageView, показаний на рис. 8.

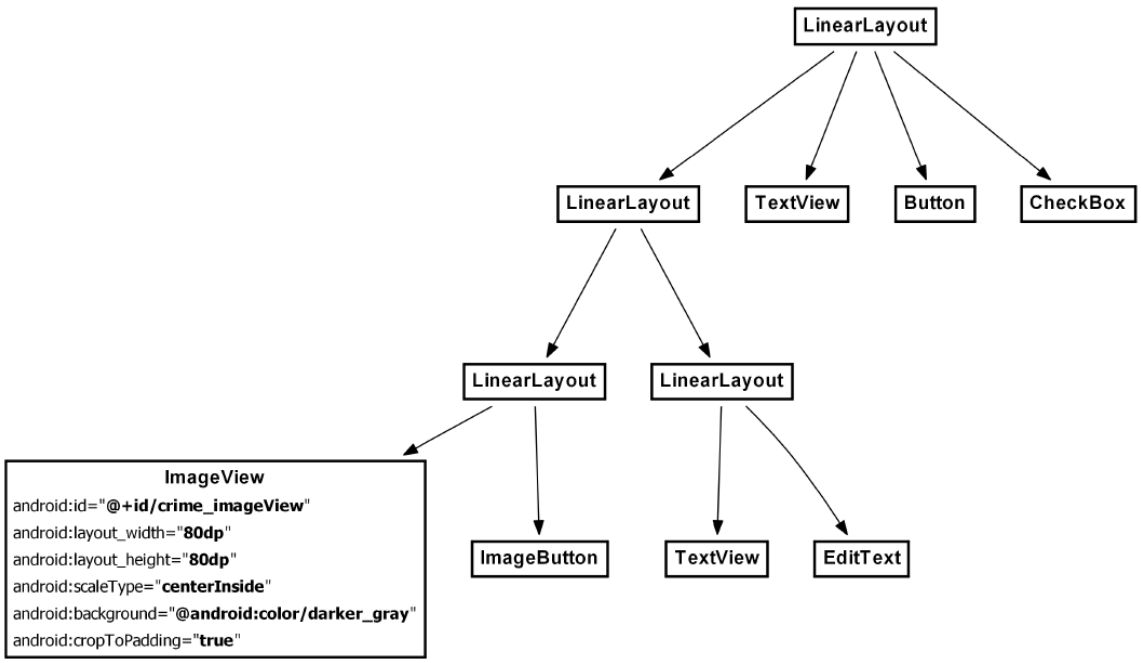

Рис. 8. Макет CrimeFragment з ImageView (layout/fragment_crime.xml)

Також нам знадобиться віджет ImageView в альбомному макеті (рис. 9).

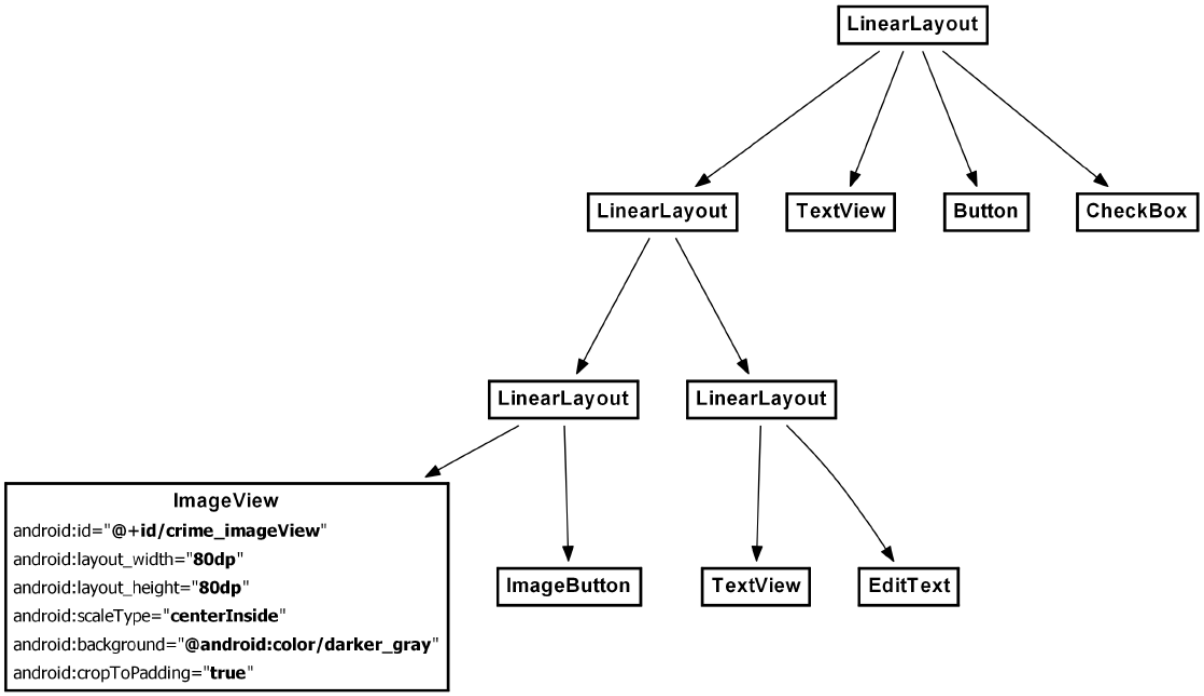

Рис. 9. Альбомний макет з ImageView (layout-land/fragment_crime.xml)

У файлі CrimeFragment.java визначте нове поле і отримаєте посилання на ImageView в методі onCreateView (...).

Лістинг 11. Підготовка ImageView (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
   ...
   private ImageButton mPhotoButton;
   private ImageView mPhotoView;

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

      mPhotoButton.setOnClickListener (new View.OnClickListener () {
         public void onClick (View v) {
            // Launch the camera activity
            Intent i = new Intent (getActivity (), CrimeCameraActivity.class);
            startActivityForResult (i, REQUEST_PHOTO);
         }
   });

   mPhotoView = (ImageView) v.findViewById (R.id.crime_imageView);
   ...
}
```

Щоб переконатися в тому, що віджет ImageView знаходиться на призначеному місці, перегляньте макет або запустіть додаток CrimIntent.

Обробка зображення

Для виведення зображення на віджеті ImageView потрібно деяка попередня обробка, тому що файли, отримані з камери, можуть бути просто величезними. З кожним роком фірми-виробники встановлюють на своїх телефонах все більші і сучасні камери.

Додавання масштабованих фотографій ImageView

У пакеті com.bignerdranch.android.Crimintent створіть новий клас з ім'ям PictureUtils. Додайте в файл PictureUtils.java метод, масштабуючий зображення за розміром стандартного екрану пристрою.

Лістинг 12. Додавання класу PictureUtils (PictureUtils.java)

```
public class PictureUtils {
    / **
    * Отримання об'єкта BitmapDrawable за даними локального файлу,
    * Масштабованого за поточними розмірами вікна.
    * /
    @SuppressWarnings ( "deprecation")
    public static BitmapDrawable getScaledDrawable (Activity a, String path) {
        Display display = a.getWindowManager (). GetDefaultDisplay ();
        float destWidth = display.getWidth ();
        float destHeight = display.getHeight ();

        // Читання розмірів зображення на диску
        BitmapFactory.Options options = new BitmapFactory.Options ();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile (path, options);

        float srcWidth = options.outWidth;
        float srcHeight = options.outHeight;

        int inSampleSize = 1;
        if (srcHeight> destHeight || srcWidth> destWidth) {
            if (srcWidth> srcHeight) {
               inSampleSize = Math.round (srcHeight / destHeight);

            } Else {
               inSampleSize = Math.round (srcWidth / destWidth);
            }
        }

        options = new BitmapFactory.Options ();
        options.inSampleSize = inSampleSize;

        Bitmap bitmap = BitmapFactory.decodeFile (path, options);
        return new BitmapDrawable (a.getResources (), bitmap);
    }
}
```

Методи Display.getWidth () і Display.getHeight () вважаються застарілими (dep-recated).

В ідеалі зображення варто було б масштабувати так, щоб воно точно відповідало розмірам ImageView. Однак розмір представлення, в якому виводиться представлення, часто буває недоступне в потрібний момент. Наприклад, в методі onCreateView (...) ми не можемо отримати розмір ImageView. Для надійності зображення масштабується за розмірами поточного екрану пристрою, який доступний завжди. Представлення, в якому буде виводитися фотографія, може бути менше стандартного розміру екрану, але більше бути не може.

Потім додайте в CrimeFragment закритий метод, який пов'язує зі зміненим розміром версію зображення з ImageView.

Лістинг 13. Додавання showPhoto () (CrimeFragment.java)

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

```
private void showPhoto () {
    // Призначення зображення, отриманого на основі фотографії
    Photo p = mCrime.getPhoto ();
    BitmapDrawable b = null;
    if (p! = null) {
        String path = getActivity ()
            .getFileStreamPath (p.getFilename ()). getAbsolutePath ();
        b = PictureUtils.getScaledDrawable (getActivity (), path);
    }
    mPhotoView.setImageDrawable (b);
}
```

Включіть в файл CrimeFragment.java реалізацію onStart () з викликом showPhoto (), щоб фотографія була готова до того моменту, як користувач побачить CrimeFragment.

Лістинг 14. Завантаження зображення (CrimeFragment.java)

```
...
private void showPhoto () {
    // Призначення зображення, отриманого на основі фотографії
    Photo p = mCrime.getPhoto ();
    BitmapDrawable b = null;
    if (p! = null) {
        String path = getActivity ()
        .getFileStreamPath (p.getFilename ()). getAbsolutePath ();
        b = PictureUtils.getScaledDrawable (getActivity (), path);
    }

    mPhotoButton.setImageDrawable (b);
    }

    @Override
    public void onStart () {
        super.onStart ();
        showPhoto ();
}
```

Включіть в метод CrimeFragment.onActivityResult (...) виклик showPhoto (), щоб зображення було видимим при поверненні користувача з CrimeCameraActivity.

Лістинг 15. Виклик showPhoto () в onActivityResult (...) (CrimeFragment.java)

```
@Override
public void onActivityResult (int requestCode, int resultCode, Intent data) {
    if (resultCode! = Activity.RESULT_OK) return;
    if (requestCode == REQUEST_PHOTO) {
        // Створення нового об'єкта Photo і зв'язування його з Crime
        String filename = data
             .getStringExtra (CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename! = null) {
             Photo p = new Photo (filename);
             mCrime.setPhoto (p);
             showPhoto ();
             Log.i (TAG, "Crime:" + mCrime.getTitle () + "has a photo");
        }
    }
}
```

Вивантаження зображення

Включіть в клас PictureUtils метод для знищення екземпляра BitmapDrawable, пов'язаного з ImageView (якщо він існує).

Лістинг 16. Очищення даних (PictureUtils.java)

```
public class PictureUtils {
    / **
    * ...
    * /
    @SuppressWarnings ( "deprecation")
    public static BitmapDrawable getScaledDrawable (Activity a, String path) {
       ...
    }

    public static void cleanImageView (ImageView imageView) {
        if (! (imageView.getDrawable () instanceof BitmapDrawable))
             return;

        // Видалення зображення для економії пам'яті
        BitmapDrawable b = (BitmapDrawable) imageView.getDrawable ();
        b.getBitmap (). recycle ();
        imageView.setImageDrawable (null);
    }
}
```

Про виклик Bitmap.recycle () варто розповісти докладніше. У документації йдеться, що викликати цей метод не обов'язково, але це не так. Bitmap.recycle () звільняє системну (native) пам'ять, яку займає растрове зображення. Це велика частина вмісту об'єкта растрового зображення. (Системна пам'ять може містити більший або менший обсяг даних в залежності від версії Android. До Honeycomb в ній зберігалися всі дані об'єктів Java Bitmap.)

Якщо не звільнити пам'ять явним викликом recycle (), вона в кінцевому підсумку все одно буде звільнена. Однак звільнення відбудеться колись в майбутньому в *завершувачі* (Finalizer), а не при знищенні самого зображення в ході збирання сміття. Відповідно виникає ймовірність вичерпання вільної пам'яті до виклику завершувача.

Момент виконання завершувача заздалегідь невідомий, тому такі помилки складно відстежувати і відтворювати. Таким чином, при великому розмірі зображень (як в нашому випадку) краще викликати recycle () для запобігання неприємних помилок пам'яті.

Включіть в CrimeFragment реалізацію onStop () з викликом cleanImageView (...).

Лістинг 17. Вивантаження зображення (CrimeFragment.java)

```
@Override
public void onStart () {
    super.onStart ();
    showPhoto ();
}
@Override
public void onStop () {
    super.onStop ();
    PictureUtils.cleanImageView (mPhotoView);
}
```

Завантаження зображень в onStart () з вивантаженням в onStop () - корисна практика. Ці методи відзначають точки, в яких активність може бути видна користувачеві. Якщо ж виконувати завантаження і вивантаження в onResume () і onPause (), результат може виявитися несподіваним для користувача.

Призупинена активність все одно може бути частково видимою - якщо, наприклад, поверх неї відкривається активність, що не займає весь екран. Якщо використовувати onResume () і onPause (), зображення в таких ситуаціях будуть зникати і з'являтися. Краще завантажувати зображення, як тільки ваша активність стає доступною, і відкладати вивантаження до того моменту, коли активність гарантовано не видно на екрані.

Відкрийте програму CrimIntent. Зробіть знімок і переконайтеся в тому, що він з'явився в ImageView. Закрийте CrimIntent і запустіть додаток заново. Переконайтеся в тому, що при поверненні до того ж елементу списку фотографія відображається, як і належить.

Орієнтація CrimeCameraActivity підказує користувачеві, що фотографії слід робити в альбомної орієнтації. Якщо фотографія буде зроблена в книжковій орієнтації, то зображення може бути неправильно розташоване на кнопці.

Відображення повнорозмірних зображень DialogFragment

Створіть новий клас в пакеті com.bignerdranch.android.Crimintent. Дайте класу ім'я ImageFragment; призначте його субкласом DialogFragment.

В аргументах фрагмента ImageFragment повинен передаватися шлях до файлу фотографії для Crime. У файлі ImageFragment.java додайте метод newInstance (String), який отримує шлях до файлу і поміщає його в пакет аргументів, як показано в лістингу 18.

Лістинг 18. Створення ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
       "Com.bignerdranch.android.Crimintent.image_path";
    public static ImageFragment newInstance (String imagePath) {
       Bundle args = new Bundle ();
       args.putSerializable (EXTRA_IMAGE_PATH, imagePath);

       ImageFragment fragment = new ImageFragment ();
       fragment.setArguments (args);
       fragment.setStyle (DialogFragment.STYLE_NO_TITLE, 0);
       return fragment;
    }
}
```

Фрагменту призначається стиль DialogFragment.STYLE_NO_TITLE, з яким фрагмент має мінімалістське оформлення.

ImageFragment не потрібні заголовок і кнопки, що надаються AlertDialog. Якщо ваш фрагмент може обійтися без них, краще використовувати більш елегантне, швидке і гнучке рішення з перевизначенням onCreateView (...) і використанням простого об'єкта View (замість перевизначення onCreateDialog (...) і використання Dialog).

У файлі ImageFragment.java перевизначите метод onCreateView (...), щоб він створював об'єкт ImageView «з нуля» з отриманням шляху до файлу зі своїх аргументів. Потім отримаєте зі зміненим розміром версію зображення і зв'яжіть її з ImageView.

Також перевизначите onDestroyView (), який повинен звільняти пам'ять, коли потреба в зображенні відпадає.

Лістинг 19. Створення ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
       "Com.bignerdranch.android.Crimintent.image_path";

    public static ImageFragment newInstance (String imagePath) {
       ...
    }

    private ImageView mImageView;

    @Override
    public View onCreateView (LayoutInflater inflater,
            ViewGroup parent, Bundle savedInstanceState) {
       mImageView = new ImageView (getActivity ());
```

```
        String path = (String) getArguments (). GetSerializable
        (EXTRA_IMAGE_PATH);
        BitmapDrawable image = PictureUtils.getScaledDrawable (getActivity (),
        path);

        mImageView.setImageDrawable (image);

        return mImageView;
    }

    @Override
    public void onDestroyView () {
        super.onDestroyView ();
        PictureUtils.cleanImageView (mImageView);
    }
}
```

Нарешті, необхідно вивести це діалогове вікно з CrimeFragment. У файлі CrimeFragment.java додайте слухача для mPhotoView. У його реалізації створіть екземпляр ImageFragment і додайте його в екземпляр FragmentManager активності CrimePagerActivity викликом методу show (...) для ImageFragment. Також знадобиться строкова константа для ідентифікації ImageFragment в FragmentManager.

Лістинг 20. Відображення ImageFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private static final String DIALOG_IMAGE = "image";

    ...

    @Override
    @TargetApi (11)
    public View onCreateView (LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...
        mPhotoView = (ImageView) v.findViewById (R.id.crime_imageView);
        mPhotoView.setOnClickListener (new View.OnClickListener () {
            public void onClick (View v) {
                Photo p = mCrime.getPhoto ();
                if (p == null)
                    return;
                FragmentManager fm = getActivity ()
                    .getSupportFragmentManager ();
                String path = getActivity ()
                    .getFileStreamPath (p.getFilename ()). getAbsolutePath ();
                ImageFragment.newInstance (path)
                    .show (fm, DIALOG_IMAGE);
            }
    });
    ...
}
```

Відкрийте програму CrimIntent. Зробіть знімок і переконайтеся в тому, що тепер додаток дозволяє переглянути фотографію сцени у всіх подробицях.

Застарілі конструкції в Android

При призначенні розміру області попереднього перегляду камери ми використовували застарілий метод і застарілу константу. Почнемо з того, що ж розуміти під самим терміном. Якщо деяка частина API вважається застарілою, це означає, що вона більш не є необхідною. Іноді старіння

відбувається через те, що виконувана операція стала непотрібною - як у випадку з методом SurfaceHolder.setType (int) і константою SurfaceHolder. SURFACE_TYPE_PUSH_ BUFFERS.

У старих версіях Android екземпляр SurfaceHolder повинен був налаштовуватися відповідно до передбачуваного використання. Зараз це вже не потрібно, тому метод setType (...) став непотрібним.

В інших випадках це означає, що метод був замінений новим методом, який з якоїсь причини найбільш прийнятний. Наприклад, клас BitmapDrawable містить застарілий конструктор BitmapDrawable (Bitmap), при використанні якого часто допускалися помилки. А може бути, новий метод «чистіше» з точки зору архітектури, як, наприклад, метод View.setBackgroundDrawable (Drawable). Також можна згадати методи Display.getWidth () і Display.getHeight (), які використовувалися раніше. Зараз вони замінені одним методом getSize (Point), який запобігає помилки, що виникали при послідовному виклику getWidth () і getHeight ().

Проблема старіння коду вирішується по-різному в залежності від того, на якій платформі ви працюєте. Дві крайності втілені в двох підходах, які вам, ймовірно, знайомі - підход Microsoft і підход Apple.

У підході Microsoft частини API застарівають, але ніколи не видаляються. Це пов'язано з тим, що в світогляді Microsoft найважливішим фактором є працездатність найбільшої кількості програм в будь-якій версії ОС. Коли Microsoft представляє загальнодоступний API, він завжди підтримується. Часом доходить до збереження помилкової, недокументованої поведінки для забезпечення зворотної сумісності.

З іншого боку, у Apple API видаляються з ОС невдовзі після того, як вони були оголошені застарілими. Керівники Apple прагнуть мати чисту, красиву ОС. Їх не хвилює, скільки API доведеться вбити для досягнення цієї мети. В результаті Apple дуже чистий і акуратний, але старі програми часто перестають працювати без активного оновлення.

У Apple для підтримки як старих, так і нових операційних систем доводиться використовувати конструкції такого вигляду:

```
float destWidth;
float destHeight;

if (Build.VERSION.SDK_INT> Build.VERSION_CODES.HONEYCOMB_MR2) {
    Point size = display.getSize ();
    destWidth = size.x;
    destHeight = size.y;
} Else {
    destWidth = display.getWidth ();
    destHeight = display.getHeight ();
}
```

Справа в тому, що у Apple методи getWidth () і getHeight (), ймовірно, скоро зникнуть. Потрібно діяти обережно, щоб випадково не викликати неіснуючий метод.

Не можна сказати. що ідеологія Android збігається з підходом Microsoft, але принаймні досить близька до нього. Кожна версія Android SDK в основному зберігає зворотну сумісність з попередньою версією, а це означає, що методи API майже ніколи не видаляються. Значить, вам не потрібно уникати виклику старих методів.

Завдання 1. Орієнтація зображення в Crime

Іноді користувач хоче зробити знімок в книжковій орієнтації. Знайдіть в документації API інформацію про те, як виявити поточну орієнтацію. Збережіть правильну орієнтацію в Photo і використовуйте її для повороту зображення і CrimeFragment і ImageFragment.

Завдання 2. Видалення фотографій

У поточній версії програми можна замінити фотографію Crime, але при цьому старий файл залишається і займає місце на диску. Додайте в метод onActivityResult (int, int, Intent) класу CrimeFragment код перевірки існуючої фотографії та видалення відповідного файлу з диска.

Надайте користувачеві можливість видалення фотографії без заміни. Реалізуйте в CrimeFragment контекстне меню і / або режим контекстних дій, що активізується довгим натисканням на мініатюрі зображення. Команда меню Delete Photo видаляє фотографію з диска, з моделі і ImageView.

Лабораторна робота № 3

Тема: Робота з HTTP і фоновими задачами

Для експериментів з мережевими можливостями Android ми створимо новий додаток PhotoGallery. Це клієнт для сайту фотообмена Flickr, який буде завантажувати і відображати останні загальнодоступні фото, надіслані на Flickr.

Створення програми PhotoGallery

Створіть новий проект додатки Android. Задайте його параметри так, як показано на рис. 3.

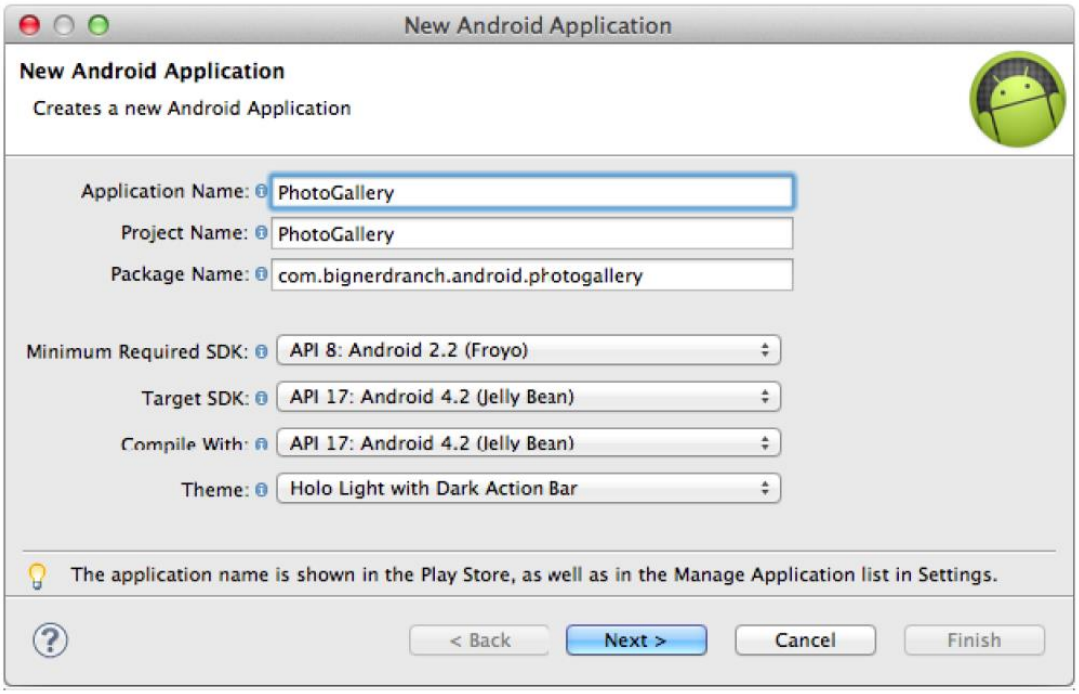

Рис. 3. Створення програми PhotoGallery

Накажіть майстру створити порожню активність з ім'ям PhotoGalleryActivity.

У додатку PhotoGallery використовується така ж архітектура, як у всіх попередніх додатках. Активність PhotoGalleryActivity буде субкласом SingleFragmentActivity, а її представленням буде контейнерне представлення, яке визначається у файлі activity_fragment.xml. Ця активність стане хостом фрагмента, а саме екземпляра PhotoGalleryFragment, який ми невдовзі створимо.

Передайте файли SingleFragmentActivity.java і activity_fragment.xml в свій проект з попереднього проекту.

У файлі PhotoGalleryActivity.java налаштуйте PhotoGalleryActivity як SingleFragment Activity; для цього видаліть код, згенерований шаблоном, і замініть його реалізацією createFragment (). Метод createFragment () повинен повертати екземпляр PhotoGalleryFragment.

Лістинг 1. Налаштування активності (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends Activity {
public class PhotoGalleryActivity extends SingleFragmentActivity {
    / * Код, згенерований шаблоном * /

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

PhotoGallery буде відображати свої результати в віджеті GridView, який стане представленням для PhotoGalleryFragment.

Клас GridView походить від AdapterView, тому він працює за тими ж принципами, що і ListView. Однак на відміну від ListView, GridView не має зручного класу GridFragment, який підключить все за вас. А це означає, що вам доведеться створити файл макета і заповнити його в PhotoGalleryFragment. Підключимо в PhotoGalleryFragment адаптер, який буде постачати GridView назви фотографій, що відображаються.

Щоб створити макет фрагмента, перейменуйте файл layout / activity_photo_gallery.xml в layout / fragment_photo_gallery.xml. Потім замініть його вміст визначенням GridView, наведеними на рис. 4.

GridView

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/gridView"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:columnWidth="120dp"
android:numColumns="auto_fit"
android:stretchMode="columnWidth"
```

Рис. 4. GridView (layout / fragment_photo_gallery.xml)

Тут ми призначаємо стовпцям ширину 120dp і наказуємо GridView створити стільки стовпців, скільки поміститься на екрані. Якщо в виділеній області залишилося місце розміром менше 120dp, атрибут stretchMode наказує GridView рівномірно розподілити його між стовпцями.

Нарешті, створіть клас PhotoGalleryFragment. Збережіть фрагмент, заповніть створений макет і отримаєте посилання на GridView (лістинг 2).

Лістинг 2. Заготівля коду (PhotoGalleryFragment.java)

```
package com.bignerdranch.android.photogallery;
...
public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setRetainInstance (true);
    }
    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {

        View v = inflater.inflate (R.layout.fragment_photo_gallery, container,
        false);
        mGridView = (GridView) v.findViewById (R.id.gridView);
        return v;
    }
}
```

Відкрийте програму PhotoGallery і переконайтеся в тому, що все працює правильно.

Основи мережевої підтримки

У нашому додатку всі мережеві взаємодії PhotoGallery будуть забезпечуватися одним класом. Створіть новий клас Java. Оскільки ми будемо підключатися до Flickr, назвіть клас FlickrFetchr.

Вихідна версія FlickrFetchr буде складатися всього з двох методів:
getUrlBytes (String) і getUrl (String).
Метод getUrlBytes (String) отримує низькорівневі дані по URL і повертає їх у вигляді масиву байтів.
Метод getUrl (String) перетворює результат з getUrlBytes (String) в String.

Додайте в файл FlickrFetchr.java реалізації getUrlBytes (String) і getUrl (String) (лістинг 3).

Лістинг 3. Основний мережевий код (FlickrFetchr.java)

```
package com.bignerdranch.android.photogallery;
...
public class FlickrFetchr {
    byte [] getUrlBytes (String urlSpec) throws IOException {
    URL url = new URL (urlSpec);
    HttpURLConnection connection = (HttpURLConnection) url.openConnection ();

    try {
        ByteArrayOutputStream out = new ByteArrayOutputStream ();
        InputStream in = connection.getInputStream ();
        if (connection.getResponseCode ()! = HttpURLConnection.HTTP_OK) {
        return null;
        }
        int bytesRead = 0;
        byte [] buffer = new byte [1024];
        while ((bytesRead = in.read (buffer))> 0) {
        out.write (buffer, 0, bytesRead);
        }
        out.close ();
        return out.toByteArray ();
    } Finally {
        connection.disconnect ();
    }
    }
    public String getUrl (String urlSpec) throws IOException {
        return new String (getUrlBytes (urlSpec));
    }
}
```

Цей код створює об'єкт URL на базі рядку - наприклад, http://www.google.com. Потім виклик методу openConnection () створює об'єкт підключення до заданого URL-адресу. Виклик URL.openConnection () повертає URLConnection, але оскільки підключення здійснюється по протоколу HTTP, ми можемо перетворити його в HttpURLConnection. Це відкриває доступ до HTTP-інтерфейсів для роботи з методами запитів, кодами відповідей, методами потокової передачі і т.д.

Об'єкт HttpURLConnection представляє підключення, але зв'язок з кінцевою точкою буде встановлений тільки після виклику getInputStream () (або getOutputStream () для POST-викликів). До цього моменту ви не зможете отримати дійсний код відповіді.

Після створення об'єкта URL і відкриття підключення програма багаторазово викликає read (), поки в підключенні не закінчаться дані. Об'єкт InputStream надає байти в міру їх доступності. Коли читання буде завершено, програма закриває його і видає масив байтів з ByteArrayOutputStream.

Хоча всю основну роботу виконує метод getUrlBytes (String), ми будемо використовувати метод getUrl (String). Він перетворює байти, отримані викликом getUrlBytes (String), в String.

Дозвіл на роботу з мережею

Для роботи мережевої підтримки необхідно попросити дозволу. Щоб запросити дозвіл на роботу з мережею, додайте наступний рядок в файл AndroidManifest.xml.

Лістинг 4. Включення дозволу на використання мережі в маніфест (AndroidManifest.xml)

```
<Manifest xmlns: android = "http://schemas.android.com/apk/res/android"
     package = "com.bignerdranch.android.photogallery"
     android: versionCode = "1"
     android: versionName = "1.0">

     <Uses-sdk
         android: minSdkVersion = "8"
         android: targetSdkVersion = "15" />
     <Uses-permission android: name = "android.permission.INTERNET" />
     ...
</ Manifest>
```

Використання AsyncTask для виконання у фоновому потоці

На наступному кроці ми повинні викликати і протестувати тільки що доданий мережевий код. Однак ми не можемо просто викликати FlickrFetchr.getURL (String) прямо з PhotoGallery Fragment. Замість цього необхідно створити фоновий пропрограмний потік і виконати код в ньому.

Для роботи з фоновими потоками найпростіше використовувати допоміжний клас з ім'ям AsyncTask. AsyncTask створює фоновий потік і виконує в ньому код, що міститься в методі doInBackground (...).

У файлі PhotoGalleryFragment.java додайте в кінці PhotoGalleryFragment новий внутрішній клас з ім'ям FetchItemsTask. Перевизначите метод AsyncTask.doIn-Background (...) для отримання даних з сайту і їх реєстрації в журналі. Потім використовуйте новий клас всередині PhotoGalleryFragment.onCreate (...).

Лістинг 5. Реалізація AsyncTask (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ...

    private class FetchItemsTask extends AsyncTask <Void, Void, Void> {
        @Override
        protected Void doInBackground (Void ... params) {
        try {
             String result = new FlickrFetchr (). GetUrl (
             "http://www.google.com");
             Log.i (TAG, "Fetched contents of URL:" + result);
        } Catch (IOException ioe) {
             Log.e (TAG, "Failed to fetch URL:", ioe);
        }
        return null;
        }
    }
}
```

Потім в методі PhotoGalleryFragment.onCreate (...) викличте метод execute () для нового екземпляра FetchItemsTask.

Лістинг 6. Реалізація AsyncTask (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    GridView mGridView;
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setRetainInstance (true);
        new FetchItemsTask (). execute ();
    }
    ...
}
```

Виклик execute () активізує клас AsyncTask, який запускає свій фоновий потік і викликає doInBackground (...). Виконайте свій код і ви побачите, що в LogCat з'являється розмітка HTML домашньої сторінки Google Javascriptlicious - приблизно так, як показано на рис. 5.

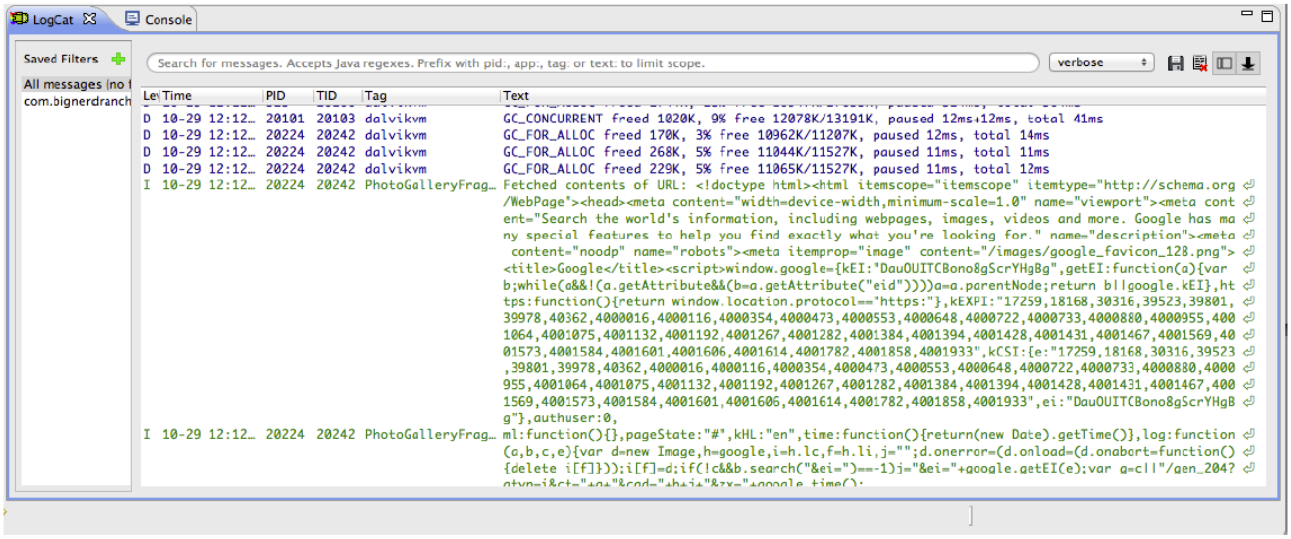

Рис. 5. Розмітка HTML від Google в LogCat

Головний програмний потік

Мережеві взаємодії не відбуваються миттєво. Веб-серверу може знадобитися одна-дві секунди на відповідь, а завантаження файлу може зайняти ще більше часу. Через тривалості мережевих операцій Android забороняє їх виконання в головному програмному потоці, починаючи з Honeycomb і наступних версій Android. Якщо ви спробуєте порушити це обмеження, Android видає виключення NetworkOnMainThreadException.

Програмним потоком(Thread) називається окрема послідовність виконання програми. Життєвий цикл кожної програми Android починається з головного потоку. Однак головний потік не є заздалегідь визначеною послідовністю дій. Він в нескінченному циклі чекає подій, ініційованих користувачем або системою, і виконує код як реакцію на ці події в міру їх виникнення.

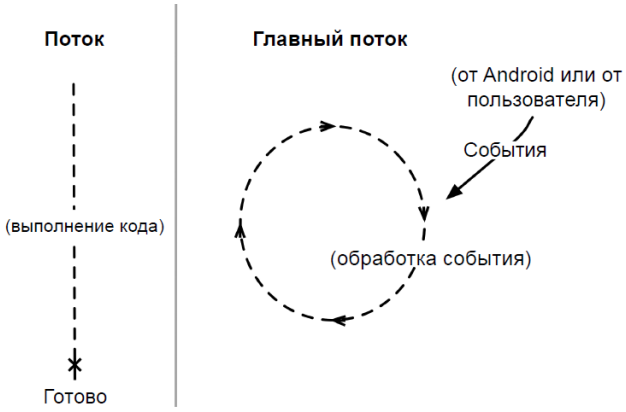

Рис. 6. Звичайні потоки і головний потік

Головний потік вашої програми виконує весь код, відновлювальний користувальницький інтерфейс. Зокрема, сюди відноситься код реакції на різні події користувальницького інтерфейсу, запуск активностей, натискання кнопок і т.д. (Оскільки всі події тим чи іншим чином пов'язані з користувальницьким інтерфейсом, головний потік іноді називається потоком користувальницького інтерфейсу, або UI-потоком.)

Цикл подій забезпечує послідовне виконання коду користувальницького інтерфейсу. Він гарантує, що операції НЕ будуть «перебігати дорогу» один одному, одночасно забезпечуючи своєчасне виконання коду. Таким чином, весь написаний вами код (за винятком коду, написаного для AsyncTask) виконувався в головному потоці.

Крім головного потоку

Мережеві операції займають багато часу в порівнянні з іншими завданнями. В цей час користувальницький інтерфейс буде повністю паралізований, що може привести до помилки ANR (Application Not Responding). Ця помилка відбувається тоді, коли система моніторингу Android виявляє, що головний потік не зреагував на важливу подію - таку, як натискання кнопки Back.

Ось чому в Android, починаючи з версії Honeycomb, заборонено виконання мережевих операцій в головному потоці. В Android - ви створюєте фоновий потік і виконуєте мережеві операції з нього.

Найпростіше працювати з фоновим потоком за допомогою AsyncTask.

Завантаження XML з Flickr

Лістинг 7. Додавання констант (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT =
    "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "yourApiKeyHere";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";

    private static final String EXTRA_SMALL_URL = "url_s";
```

Ці константи визначають кінцеву точку, ім'я методу, ключ API і один додатковий параметр з ім'ям extras і значенням url_s. Завдання доповнення url_s наказує Flickr включити URL зменшеної версії зображення, якщо вона доступна.

Використовуйте ці константи для написання методу, який будує відповідну URL-адресу запиту і завантажує його вміст.

Лістинг 8. Додавання методу fetchItems () (FlickrFetchr.java)

```
public class FlickrFetchr {

    ...

    String getUrl (String urlSpec) throws IOException {
       return new String (getUrlBytes (urlSpec));
    }
    public void fetchItems () {
       try {
          String url = Uri.parse (ENDPOINT) .buildUpon ()
              .appendQueryParameter ( "method", METHOD_GET_RECENT)
              .appendQueryParameter ( "api_key", API_KEY)
              .appendQueryParameter (PARAM_EXTRAS, EXTRA_SMALL_URL)
              .build (). toString ();
         String xmlString = getUrl (url);
         Log.i (TAG, "Received xml:" + xmlString);
      } Catch (IOException ioe) {
         Log.e (TAG, "Failed to fetch items", ioe);
      }
    }
}
```

Тут ми використовуємо клас Uri.Builder для побудови повної URL-адреси для API-запиту до Flickr. Uri.Builder - допоміжний клас для створення параметрезованих URL-адрес з правильним кодуванням символів. Метод Uri.Builder.appendQueryParameter (String, String) автоматично кодує рядки запитів.

Змініть код AsyncTask в PhotoGalleryFragment для виклику нового методу fetchItems ().

Лістинг 9. Виклик fetchItems () (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask <Void, Void, Void> {
    @Override
    protected Void doInBackground (Void ... params) {
        try {
            String result = new FlickrFetchr (). GetUrl (
            "http://www.google.com");
            Log.i (TAG, "Fetched contents of URL:" + result);
        } Catch (IOException ioe) {
            Log.e (TAG, "Failed to fetch URL:", ioe);
        }
        new FlickrFetchr (). fetchItems ();
        return null;
    }
}
```

Відкрийте програму PhotoGallery. У LogCat відображається повноцінна, справжня розмітка XML.

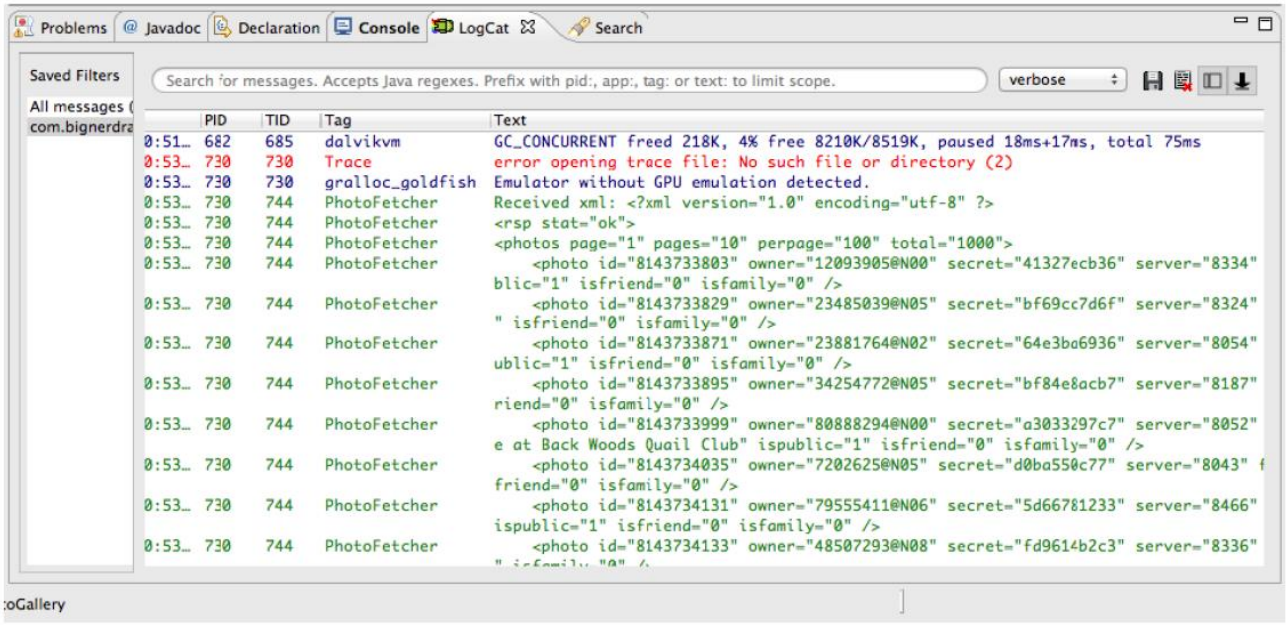

Рис. 8. Розмітка XML в Flickr

Коли, розмітка XML з Flickr отримана, необхідно позначити один або кілька об'єктів моделі. Клас моделі, який ми створимо для PhotoGallery, називається GalleryItem. На рис. 9 зображена діаграма об'єктів PhotoGallery.

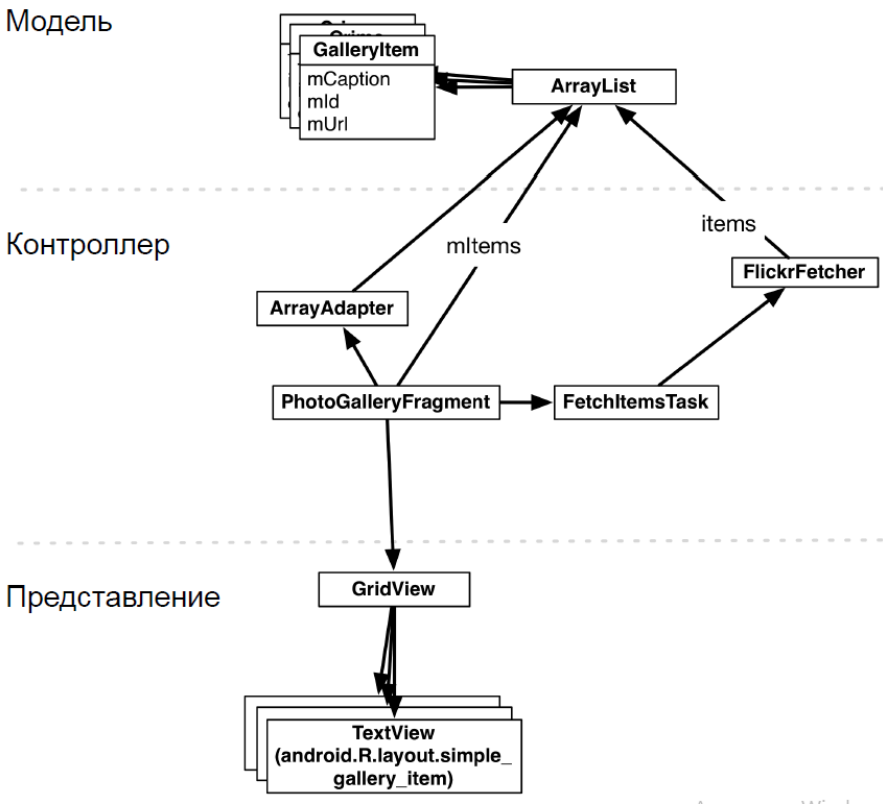

Рис. 9. Розмітка XML в Flickr

На рис. 9 не відображена активність-хост, щоб діаграма була сконцентрована на фрагментах і мережевому коді.

Створіть клас GalleryItem і додайте наступний код.

Лістинг 10. Створення класу об'єкта моделі (GalleryItem.java)

```
package com.bignerdranch.android.photogallery;
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    public String toString () {
       return mCaption;
    }
}
```

Накажіть Eclipse згенерувати get- і set-методи для mId, mCaption і mUrl.

Після того як об'єкти моделі будуть створені, їх слід заповнити даними з розмітки XML, отриманої від Flickr. Для отримання даних з XML використовується інтерфейс XmlPullParser.

Використання XmlPullParser

XmlPullParser - інтерфейс, який може використовуватися для виділення подій розбору в потоці розмітки XML. XmlPullParser використовується у внутрішній реалізації Android для заповнення ваших файлів макетів. Його можна використовувати для розбору об'єктів GalleryItem.

Додайте в FlickrFetchr константу, яка визначає ім'я елемента XML, що містить фотографію. Потім напишіть метод, який використовує XmlPullParser для ідентифікації всіх фотографій в XML. Створіть об'єкт GalleryItem для кожної фотографії і додайте його в ArrayList.

Лістинг 11. Розбір фотографій Flickr (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";
    private static final String ENDPOINT =
    "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "your API key";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";

    private static final String XML_PHOTO = "photo";

    ...

    public void fetchItems () {
       ...
    }

    void parseItems (ArrayList <GalleryItem> items, XmlPullParser parser)
          throws XmlPullParserException, IOException {
       int eventType = parser.next ();
           while (eventType! = XmlPullParser.END_DOCUMENT) {
               if (eventType == XmlPullParser.START_TAG &&
                   XML_PHOTO.equals (parser.getName ())) {
                   String id = parser.getAttributeValue (null, "id");
                   String caption = parser.getAttributeValue (null, "title");
                   String smallUrl = parser.getAttributeValue (null,
                   EXTRA_SMALL_URL);

                   GalleryItem item = new GalleryItem ();
                   item.setId (id);
                   item.setCaption (caption);
                   item.setUrl (smallUrl);
                   items.add (item);
```

```
            }
            eventType = parser.next ();
        }
    }
}
```

XmlPullParser в документі XML послідовно перебирає різні події, такі як START_TAG, END_TAG і END_DOCUMENT. На кожному кроці ви можете викликати такі методи, як getText (), getName () або getAttributeValue (...), для отримання інформації про подію, на яку зараз вказує XmlPullParser. Щоб перемістити XmlPullParser до наступної події в XML, викличте next (). Цей метод також повертає тип події, до якого він тільки що перемістився.

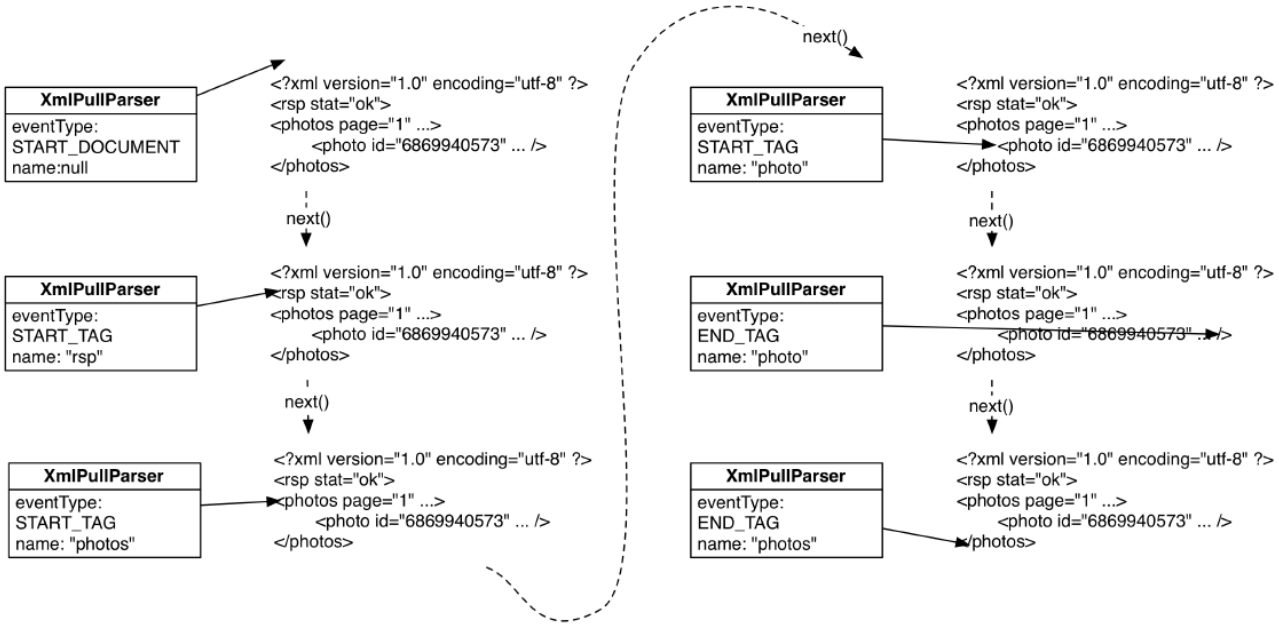

Рис. 10. Як працює XmlPullParser

Методу parseItems (...) передаються XmlPullParser і ArrayList. Отримайте екземпляр парсеру і передайте йому дані xmlString, отримані від Flickr. Потім викличте parseItems (...) з налаштованим парсером і порожнім масивом.

Лістинг 12. Виклик parseItems (...) (FlickrFetchr.java)

```
~~public void fetchItems () {~~
public ArrayList <GalleryItem> fetchItems () {
    ArrayList <GalleryItem> items = new ArrayList <GalleryItem> ();

    try {
        String url = Uri.parse (ENDPOINT) .buildUpon ()
            .appendQueryParameter ( "method", METHOD_GET_RECENT)
            .appendQueryParameter ( "api_key", API_KEY)
            .appendQueryParameter (PARAM_EXTRAS, EXTRA_SMALL_URL)
            .build (). toString ();
        String xmlString = getUrl (url);
        Log.i (TAG, "Received xml:" + xmlString);
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance ();
        XmlPullParser parser = factory.newPullParser ();
        parser.setInput (new StringReader (xmlString));

        parseItems (items, parser);
    } Catch (IOException ioe) {

        Log.e (TAG, "Failed to fetch items", ioe);
    } Catch (XmlPullParserException xppe) {
```

```
            Log.e (TAG, "Failed to parse items", xppe);
        }
        return items;
    }
```

Від AsyncTask до головного потоку

Клас GridView, як і ListView, походить від AdapterView, тому йому потрібен адаптер, що поставляє представлення, які відображаються.

У файлі PhotoGalleryFragment.java додайте оголошення ArrayList з елементами GalleryItems, а потім створіть екземпляр ArrayAdapter для простого макета, наданого Android.

Лістинг 13. Реалізація setupAdapter () (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList <GalleryItem> mItems;

    ...

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

        mGridView = (GridView) v.findViewById (R.id.gridView);

        setupAdapter ();

        return v;
    }
    void setupAdapter () {
        if (getActivity () == null || mGridView == null) return;

        if (mItems! = null) {
            mGridView.setAdapter (new ArrayAdapter <GalleryItem> (getActivity
            (),android.R.layout.simple_gallery_item, mItems));
    } Else {
        mGridView.setAdapter (null);
    }
}
```

Так як у GridView немає зручного класу GridFragment, нам доведеться самостійно побудувати код управління адаптером. Наприклад, для цього можна використовувати метод на зразок тільки що доданого методу setupAdapter (). Цей метод перевіряє поточний стан моделі і відповідним чином налаштовує адаптер GridView. Цей метод повинен викликатися в onCreateView (...), щоб при кожному створенні нового екземпляра GridView при повороті він заново налаштовувався відповідним адаптером. Метод також повинен викликатися при кожній зміні набору об'єктів моделі.

Макет android.R.layout.simple_gallery_item складається з елемента TextView. У GalleryItem ми перевизначили метод toString () для повернення значення mCaption об'єкта. Таким чином, для відображення назв фотографій в GridView досить передати адаптеру масив екземплярів GalleryItem і цей макет.

Перед призначенням адаптера ми перевіряємо getActivity () на null. Фрагменти можуть існувати і автономно, не будучи пов'язаними з будь-якою активністю. До теперішнього моменту ми не стикалися з такою можливістю, тому що виклики методів керувалися зворотними викликами від

інфраструктури. Якщо фрагмент отримує зворотні виклики, він виразно приєднаний до активності. Немає активності - немає зворотних викликів.

Тепер, коли ми використовуємо AsyncTask, деякі дії ініціюються самостійно, і ми вже не можемо припускати, що фрагмент приєднаний до активності. Таким чином, спочатку необхідно переконатися в тому, що фрагмент залишається приєднаним; в іншому випадку спроба виконання операції завершиться невдачею.

Після отримання даних від Flickr слід викликати метод setupAdapter (). Перше, що спадає на думку, - виклик setupAdapter () в кінці методу doInBackground (...) класу FetchItemsTask. Це не найкраща думка.

На комп'ютері плутанина може привести до пошкодження об'єктів в пам'яті. З цієї причини фоновим потокам забороняється оновлювати користувальницький інтерфейс, оскільки такі операції явно небезпечні.

У AsyncTask є метод onPostExecute (...), який можна перевизначити. Метод onPostExecute (...) виконується після завершення doInBack-ground (...). Крім того, onPostExecute (...) виконується в головному, а не в фоновому потоці, тому оновлення користувальницького інтерфейсу в ньому безпечно.

Внесіть зміни в метод FetchItemsTask, щоб він оновлював поле mItems і викликав setupAdapter () після завантаження фотографій.

Лістинг 14. Додавання коду поновлення адаптера (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask <Void, Void, Void> {
private class FetchItemsTask extends AsyncTask <Void, Void, ArrayList
<GalleryItem >> {
    @Override
    protected Void doInBackground (Void ... params) {
    protected ArrayList <GalleryItem> doInBackground (Void ... params) {
    new FlickrFetchr (). fetchItems ();
    return new FlickrFetchr (). fetchItems ();
    return null;
}

@Override
protected void onPostExecute (ArrayList <GalleryItem> items) {
    mItems = items;
    setupAdapter ();
}
}
```

Ми внесли три зміни. По-перше, ми змінили тип третього узагальненого параметра FetchItemsTask. Цей параметр визначає тип результату, виробленого AsyncTask; він задає тип значення, що повертається doInBackground (...), а також тип вхідного параметра onPostExecute (...). По-друге, ми змінили метод doInBackground (...) так, щоб він повертав список елементів GalleryItem. Тим самим ми усунули помилку в коді і забезпечили його нормальну компіляцію. Також при виклику передається список елементів, щоб він міг використовуватися в коді onPostExecute (...). По-третє, була додана реалізація onPostExecute (...). Цей метод отримує список, завантажений в doInBackground (...), поміщає його в mItems і оновлює адаптер GridView.

Відкрийте програму, і ви побачите текст, який з'явиться в кожного завантаженого елемента GalleryItem.

Рис. 11. Заголовки фотографій Flickr

AsyncTask

Перший параметр-тип дозволяє задати тип вхідних параметрів. Він використовується наступним чином:

```
AsyncTask <String, Void, Void> task = new AsyncTask <String, Void, Void> () {
    public Void doInBackground (String ... params) {
        for (String parameter: params) {
             Log.i (TAG, "Received parameter:" + parameter);
        }

        return null;
        }
    };

task.execute ( "First parameter", "Second parameter", "Etc.");
```

Вхідні параметри передаються методу execute (...), який викликається з перемінним числом аргументів. Ці змінні аргументи передаються doInBack-ground (...).

Другий параметр-тип дозволяє задати тип для передачі інформації про хід виконання операції. Ось як це виглядає:

```
final ProgressBar progressBar = / * Індикатор прогресу * /;
progressBar.setMax (100);
AsyncTask <Integer, Integer, Void> task = new AsyncTask <Integer, Integer,
Void> () {
    public Void doInBackground (Integer ... params) {
        for (Integer progress: params) {
             publishProgress (progress);
             Thread.sleep (1000);
        }
    }

    public void onProgressUpdate (Integer ... params) {
        int progress = params [0];
        progressBar.setProgress (progress);
    }
};
task.execute (25, 50, 75, 100);
```

Оновлення зазвичай виконується в тривалому фоновому процесі. Проблема в тому, що необхідні оновлення користувальницького інтерфейсу не можуть виконуватися прямо з фонового процесу, тому AsyncTask надає методи publishProgress (...) і onProgressUpdate (...).

Механізм поновлення працює наступним чином: в методі doInBackground (...) у фоновому потоці викликається publishProgress (...). Це призводить до виклику onProgressUpdate (...) в потоці користувальницького інтерфейсу. Таким чином, користувальницький інтерфейс оновлюється в onProgressUpdate (...), але управління оновленнями здійснюється викликами publishProgress (...) в doInBackground (...).

Знищення AsyncTask

Може виникнути необхідність у відстеженні AsyncTask і навіть їх періодичного скасування і повторний запуск.

У подібних складніших сценаріях використання AsyncTask присвоюється змінній екземпляра, після чого для неї можна викликати AsyncTask.cancel (boolean) для скасування поточної фонової операції AsyncTask.

AsyncTask.cancel (boolean) може працювати в більш жорсткому або менш жорсткому режимі. Якщо викликати cancel (false), метод діє менш жорстко і просто повертає true при виклику isCancelled (). Далі AsyncTask перевіряє isCancelled () всередині doInBackground (...) і приймає рішення про дострокове завершення операції.

Але в разі виклику cancel (true) метод діє жорстко і перериває програмний потік, в якому виконується doInBackground (...). Виклик AsyncTask.cancel (true) є агресивним способом зупинки AsyncTask. Якщо цього можна уникнути - краще так і зробити.

Завдання. Сторінкова навігація

За замовчуванням метод getRecent повертає одну сторінку зі 100 результатами. За допомогою додаткового параметра page отримати другу, третю і так далі сторінку результатів.

Додайте в адаптер код, який виявляє перехід до кінця масиву елементів і замінює поточну сторінку наступною сторінкою результатів.

Організуйте приєднання даних наступних сторінок до результатів.

Лабораторна робота № 4

Тема: Робота з Looper, Handler, HandlerThread

Підготовка GridView до виведення зображень

Поточний адаптер PhotoGalleryFragment просто надає віджети TextView для виведення в GridView. У кожному представленні TextView виводиться вміст заголовка GalleryItem.

Щоб виводити фотографії, вам знадобиться нестандартний адаптер, який замість текстових представлень надає ImageView. В кінцевому підсумку ImageView виведе фотографію, завантажену в поле mUrl екземпляра GalleryItem.

Почнемо зі створення нового файлу макета для елементів фотогалереї в файлі gallery_item.xml. Макет буде складатися з єдиного віджета ImageView (рис. 1).

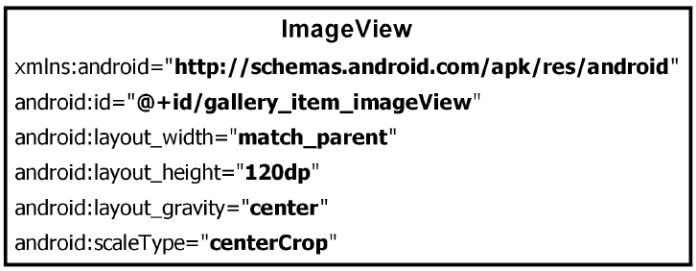

Рис. 1. Макет елемента фотогалереї (res/layout/gallery_item.xml)

Ці віджети ImageView знаходяться під управлінням GridView, це означає, що їх ширина буде величиною змінною. Проте висота буде залишатися фіксованою. Щоб найбільш ефективно використовувати простір віджета ImageView, слід задати його властивості scaleType значення centerCrop. З цим значенням зображення вирівнюється по центру і масштабується, щоб менша сторона була рівною розміру представлення, а велика усікається з обох сторін.

Також знадобиться тимчасове зображення для кожного віджета ImageView, яке буде відображатися до завершення завантаження зображення. Знайдіть файл brian_up_close.jpg в файлі рішень і помістіть його в каталог res/drawable-hdpi.

У класі PhotoGalleryFragment замініть базову реалізацію ArrayAdapter користувальницькою реалізацією, у якій реалізація getView (...) повертає віджет ImageView з тимчасовим зображенням.

Лістинг 1. Створення GalleryItemAdapter (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...

    void setupAdapter () {
        if (getActivity () == null || mGridView == null) return;

        if (mItems! = null) {
            mGridView.setAdapter (new ArrayAdapter <GalleryItem> (getActivity
            (),
               android.R.layout.simple_gallery_item, mItems));
            mGridView.setAdapter (new GalleryItemAdapter (mItems));
        } Else {
            mGridView.setAdapter (null);
        }
    }

    private class FetchItemsTask extends AsyncTask <Void, Void, ArrayList
    <GalleryItem >> {
        ...
    }
```

```
    private class GalleryItemAdapter extends ArrayAdapter <GalleryItem> {
        public GalleryItemAdapter (ArrayList <GalleryItem> items) {
            super (getActivity (), 0, items);
        }

        @Override
        public View getView (int position, View convertView, ViewGroup parent) {
            if (convertView == null) {
                convertView = getActivity (). getLayoutInflater ()
                    .inflate (R.layout.gallery_item, parent, false);
            }

            ImageView imageView = (ImageView) convertView
                .findViewById (R.id.gallery_item_imageView);
            imageView.setImageResource (R.drawable.brian_up_close);

            return convertView;
        }
    }
}
```

AdapterView (GridView в даному випадку) викликає метод getView (...) свого адаптера для кожного потрібного йому окремого представлення.

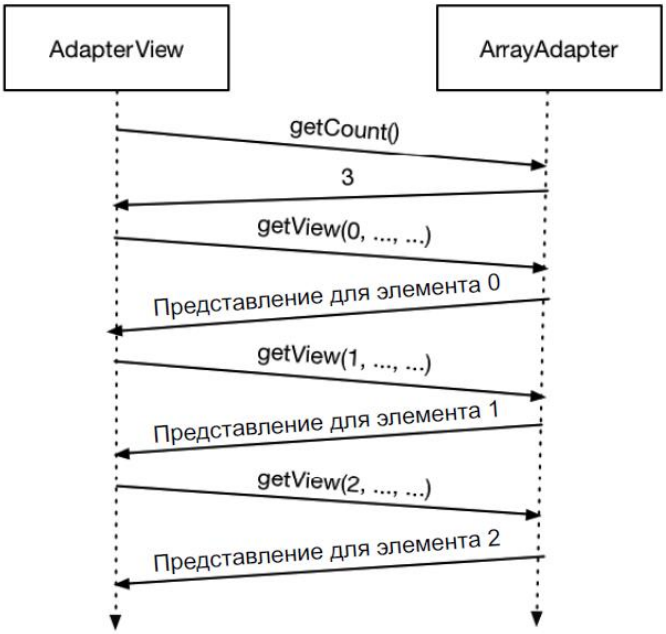

Рис. 2. Взаємодії AdapterView-ArrayAdapter

Множинні завантаження

В даний час мережева частина PhotoGallery працює наступним чином: Photo-GalleryFragment запускає екземпляр AsyncTask, який отримує XML від Flickr у фоновому потоці, і розбирає XML в масив об'єктів GalleryItem. У кожному об'єкті GalleryItem тепер зберігається URL, за яким знаходиться мініатюрна версія фотографії.

Наступним кроком повинне стати завантаження цих мініатюр. Здавалося б, додатковий мережевий код можна просто додати в метод doInBackground () класу FetchItemsTask. Масив об'єктів GalleryItem містить 100 URL- адрес для завантаження. Зображення будуть завантажуватися один за одним, поки у вас не з'являться всі 100. При виконанні onPostExecute (...) вони всі разом з'являться в GridView.

Однак одноразове завантаження всіх мініатюр створює дві проблеми. По-перше, воно займе досить багато часу, а користувальницький інтерфейс не буде оновлюватися до моменту його завершення.

По-друге, зберігання повного набору зображень вимагає ресурсів. Сотня мініатюр легко вміститься в пам'яті. Але що, якщо їх буде 1000? Що, якщо ви захочете реалізувати нескінченну прокрутку? Згодом вільна пам'ять буде вичерпана.

З урахуванням цих проблем реальні програми часто завантажують зображення тільки тоді, коли вони повинні виводитися на екрані. Завантаження в міру потреби висуває додаткові вимоги до GridView і його адаптера. Завантаження зображення ініціюється як частина реалізації getView (...) адаптера.

AsyncTask - найпростіший спосіб отримання фонових потоків, але для багаторазово виконуваних і тривалих операцій цей механізм спочатку малопридатний.

Замість використання AsyncTask ми створимо спеціалізований фоновий потік. Це найпоширеніший спосіб реалізації завантаження у міру потреби.

Взаємодія з головним потоком

Спеціалізований потік буде завантажувати фотографії, але як він буде взаємодіяти з адаптером GridView для їх відображення, якщо він не може безпосередньо звертатися до головного потоку?

Android «поштова скриньку», яка використовується потоками, називається *чергою повідомлень* (Message queue). Потік, що працює з використанням черги повідомлень, називається *циклом повідомлень* (message loop); він знову і знову перевіряє нові повідомлення, які могли з'явитися в черзі (рис. 4).

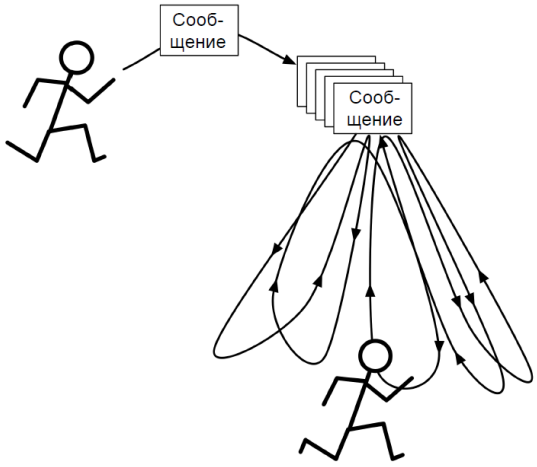

Рис. 4. Цикл повідомлень

Цикл повідомлень складається з потоку і об'єкта Looper, керуючого чергою повідомлень потоку.

Головний потік являє собою цикл повідомлень і у нього є керуючий об'єкт, який виймає повідомлення з черги повідомлень і виконує завдання, описане в повідомленні.

Ми створимо фоновий потік, який теж використовує цикл повідомлень. При цьому буде використовуватися клас HandlerThread, який надає готовий об'єкт Looper.

Створення фонового потоку

Створіть новий клас з ім'ям ThumbnailDownloader, що розширює HandlerThread. Користувачеві ThumbnailDownloader знадобиться об'єкт для ідентифікації кожного завантаження; визначте один узагальнений аргумент Token, присвоївши йому ім'я ThumbnailDownloader <Token>

в діалоговому вікні створення класу. Потім визначте для нього конструктор і заглушку реалізації методу з ім'ям queueThumbnail () (лістинг 2).

Лістинг 2. Вихідна версія коду потоку (ThumbnailDownloader.java)

```
public class ThumbnailDownloader <Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    public ThumbnailDownloader () {
       super (TAG);
    }

    public void queueThumbnail (Token token, String url) {
       Log.i (TAG, "Got an URL:" + url);
    }
}
```

QueueThumbnail () очікує отримати Token і String. Цей метод буде викликатися GalleryItemAdapter в його реалізації getView (...).

Відкрийте файл PhotoGalleryFragment.java. Визначте в PhotoGalleryFragment поле типу ThumbnailDownloader. В якості маркера (token) ThumbnailDownloader можна використовувати будь-який об'єкт. У нашому випадку зручним маркером є об'єкт ImageView, так як в кінцевому підсумку завантажені зображення потраплять саме в нього. У методі onCreate (...) створіть потік і запустіть його. Перевизначите метод onDestroy () для завершення потоку.

Лістинг 3. Створення класу ThumbnailDownloader (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList <GalleryItem> mItems;
    ThumbnailDownloader <ImageView> mThumbnailThread;

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setRetainInstance (true);
        new FetchItemsTask (). execute ();
        mThumbnailThread = new ThumbnailDownloader <ImageView> ();
        mThumbnailThread.start ();
        mThumbnailThread.getLooper ();
        Log.i (TAG, "Background thread started");
    }

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

    @Override
    public void onDestroy () {
        super.onDestroy ();
        mThumbnailThread.quit ();
        Log.i (TAG, "Background thread destroyed");
    }
    ...
}
```

Пара приміток: по-перше, зверніть увагу на те, що виклик getLooper () слідує після виклику start () для ThumbnailDownloader. Тим самим гарантується, що внутрішній стан потоку готовий для продовження

. Подруге, виклик quit () завершує потік всередині onDestroy (). Це дуже важливий момент. Якщо не завершувати потоки HandlerThread, вони ніколи не помруть).

В методі GalleryItemAdapter.getView (...) отримаєте правильний елемент GalleryItem з використанням параметра position, викличте метод queueThumbnail () потоку і передайте ImageView і URL-адресу елемента.

Лістинг 4. Підключення ThumbnailDownloader (PhotoGalleryFragment.java)

```
private class GalleryItemAdapter extends ArrayAdapter <GalleryItem> {
   ...

   @Override
   public View getView (int position, View convertView, ViewGroup parent) {
      ...

      ImageView imageView = (ImageView) convertView
             .findViewById (R.id.gallery_item_imageView);
      imageView.setImageResource (R.drawable.brian_up_close);
      GalleryItem item = getItem (position);
      mThumbnailThread.queueThumbnail (imageView, item.getUrl ());

      return convertView;
   }
}
```

Відкрийте програму PhotoGallery і перевірте дані LogCat. При прокручуванні GridView в LogCat з'являються рядки, які повідомляють про те, що ThumbnailDownloader отримує всі запити на завантаження.

Тепер, реалізація HandlerThread заробила, наступним кроком стає створення повідомлення з інформацією, переданою queueThumbnail (), і його розміщення в черзі повідомлень Thumbnail Downloader.

Повідомлення і обробники повідомлень

Будова повідомлення

Повідомлення є екземпляром класу Message і складається з декількох полів.

Для нашої реалізації важливі три поля:
- what - визначається користувачем значення int, яке описує повідомлення;
- obj - заданий користувачем об'єкт, який передається з повідомленням;
- target - приймач, тобто об'єкт Handler, який буде обробляти повідомлення.

Приймачем повідомлення Message є екземпляр Handler. Коли ви створюєте повідомлення, воно автоматично приєднується до Handler. А коли ваше повідомлення буде готове до обробки, саме Handler стає об'єктом, який відповідає за цю обробку.

Будова обробника

Для виконання реальної роботи з повідомленнями необхідно мати екземпляр Handler. Об'єкт Handler - не просто приймач для обробки повідомлень; він також надає інтерфейс для створення і відправки повідомлень.

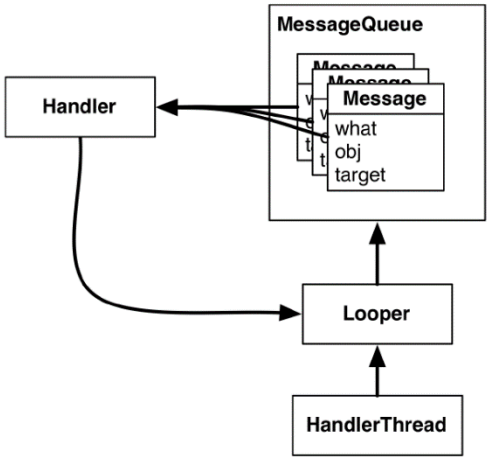

Рис. 5. Looper, Handler, HandlerThread і Message

Повідомлення Message відправляються і споживаються об'єктом Looper, тому що Looper є власником поштової скриньки об'єктів Message. Відповідно Handler завжди містить посилання на свого колегу Looper.

Handler завжди приєднується рівно до одного об'єкту Looper, а Message приєднується рівно до одного об'єкту Handler, званому його наступником. Об'єкт Looper обслуговує цілу чергу повідомлень Message.

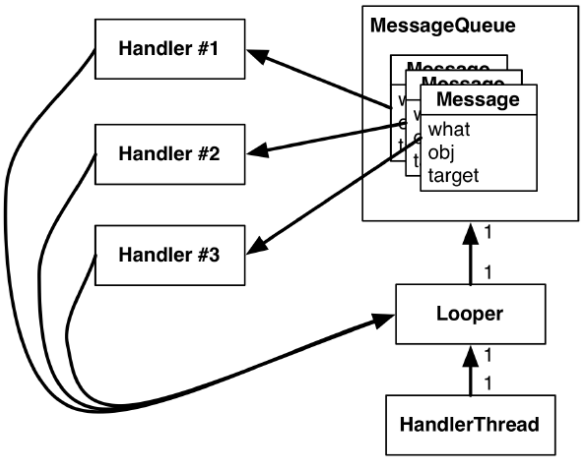

Рис. 6. Кілька об'єктів Handler, один об'єкт Looper

До одного об'єкту Looper можуть бути приєднані кілька об'єктів Handler. Це означає, що повідомлення Message об'єкта Handler можуть співіснувати з повідомленнями іншого об'єкта Handler.

Використання обробників

Зазвичай приймальні об'єкти Handler повідомлень не задаються вручну. Краще побудувати повідомлення викликом Handler.obtainMessage (...). Ви передаєте методу інші поля повідомлення, а він автоматично призначає приймач.

Метод Handler.obtainMessage (...) використовує загальний пул об'єктів, щоб уникнути створення нових об'єктів Message, тому він також працює більш ефективно, ніж просте створення нових екземплярів.

Коли об'єкт Message буде отримано, ви можете викликати метод sendToTarget (), щоб відправити повідомлення його обробнику. Оброблювач поміщає повідомлення в кінець черги повідомлень об'єкта Looper.

Ми збираємося отримати повідомлення і відправити його приймачу в реалізації queueThumbnail (). В полі what повідомлення буде міститися константа, яка визначається під ім'ям MESSAGE_DOWNLOAD. В поле obj міститиметься маркер Token - в нашому випадку екземпляр ImageView, переданий адаптером queueThumbnail ().

Коли об'єкт Looper добирається до конкретного повідомлення в черзі, він передає приймачу повідомлення для обробки. Як правило, повідомлення обробляється в реалізації Handler.handleMessage (...) приймача.

У нашому випадку реалізація handleMessage (...) буде використовувати FlickrFetchr для завантаження байтів за URL-адресою та їх перетворення в растрове зображення. Додайте код, наведений у лістингу 5.

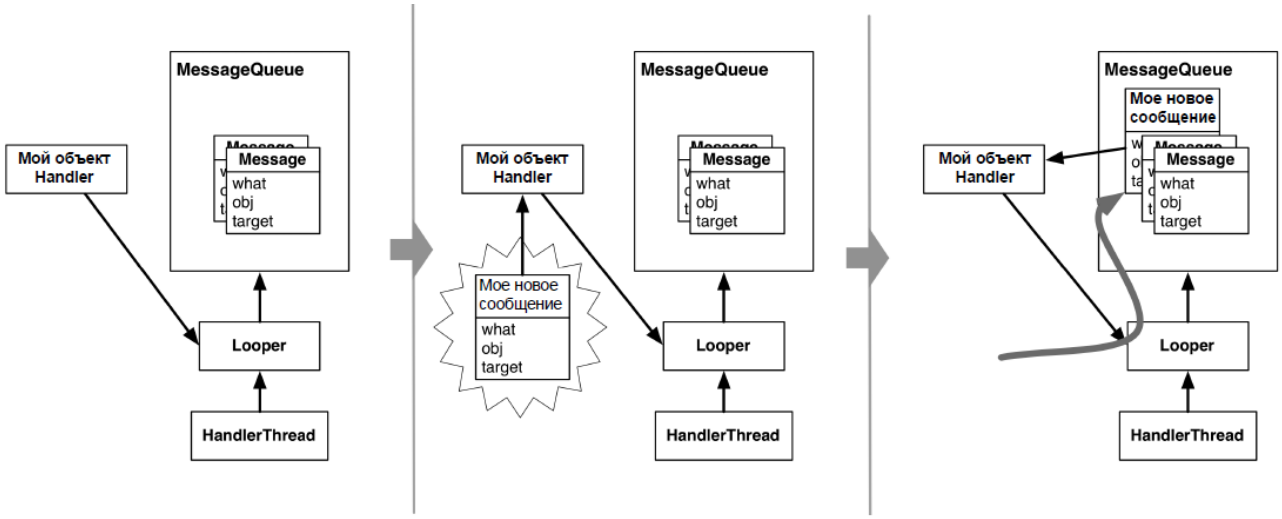

Рис. 7. Створення повідомлення і його відправка

Лістинг 5. Отримання, написання та обробка повідомлення (ThumbnailDownloader.java)

```
public class ThumbnailDownloader <Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map <Token, String> requestMap =
            Collections.synchronizedMap (new HashMap <Token, String> ());

    public ThumbnailDownloader () {
       super (TAG);
    }

    @SuppressLint ( "HandlerLeak")
    @Override
    protected void onLooperPrepared () {
       mHandler = new Handler () {
           @Override
           public void handleMessage (Message msg) {
              if (msg.what == MESSAGE_DOWNLOAD) {
                  @SuppressWarnings ( "unchecked")
                  Token token = (Token) msg.obj;
                  Log.i (TAG, "Got a request for url:" + requestMap.get (token));
                  handleRequest (token);
              }
           }
       };
    }

    public void queueThumbnail (Token token, String url) {
       Log.i (TAG, "Got a URL:" + url ");
```

```
        requestMap.put (token, url);

        mHandler
            .obtainMessage (MESSAGE_DOWNLOAD, token)
            .sendToTarget ();
    }
    private void handleRequest (final Token token) {
        try {
            final String url = requestMap.get (token);
            if (url == null)
            return;
            byte [] bitmapBytes = new FlickrFetchr (). getUrlBytes (url);
            final Bitmap bitmap = BitmapFactory
            .decodeByteArray (bitmapBytes, 0, bitmapBytes.length);
            Log.i (TAG, "Bitmap created");
        } Catch (IOException ioe) {
            Log.e (TAG, "Error downloading image", ioe);
    }
}
```

Для початку зверніть увагу на анотацію на початку onLooperPrepared () - @SuppressLint ("HandlerLeak"). Тут Android Lint видає попередження про субкласування Handler. Життєздатність нашого об'єкта Handler буде забезпечувати його екземпляр Looper. Таким чином, якщо Handler є анонімним внутрішнім класом, може легко виникнути витік пам'яті через неявне посилання на об'єкт. Однак в нашій ситуації все прив'язане до екземпляра HandlerThread, так що витік пам'яті виключений.

Інша анотація @SuppressWarnings("unchecked") більш типова. Вона необхідна, тому що Token є узагальненим аргумент класу, а Message. obj відноситься до типу Object. Через стирання типу пряме перетворення такого роду неможливо.

Змінна requestMap є синхронізований об'єкт HashMap. Тут, використовуючи Token як ключ, можна зберігати і завантажувати URL-адреси, пов'язані з конкретним об'єктом Token.

У методі queueThumbnail () в хеш-карту додається передана пара «Token-URL». Потім ми отримуємо повідомлення, передаємо Token як значення obj і відправляємо на постановку в чергу повідомлень.

Ми реалізуємо Handler.handleMessage (...) в субклас Handler всередині onLooperPrepared (). Метод HandlerThread.onLooperPrepared () викликається до того, як Looper перевірить чергу в перший раз, тому він добре підходить для створення реалізації Handler.

У Handler.handleMessage (...) ми перевіряємо тип повідомлення, отримуємо маркер Token і передаємо його handleRequest (...).

Все завантаження здійснюється в методі handleRequest (). Ми перевіряємо існування URL-адреси, після чого передаємо його новому екземпляру знайомого класу FlickrFetchr. При цьому використовується метод FlickrFetchr.getUrlBytes (...).

Ми використовуємо клас BitmapFactory для побудови растрового зображення з масивом байтів, повернутим getUrlBytes (...).

Запустіть додаток PhotoGallery і перевірте в даних LogCat ваші команди, що підтверджують реєстрації.

Зрозуміло, запит не буде повністю оброблений до моменту призначення зображення в віджеті ImageView, що надійшов від GalleryItemAdapter. Однак ця операція відноситься до користувальницького інтерфейсу, тому вона повинна виконуватися в головному потоці.

До теперішнього моменту ми обмежувалися використанням оброблювачів і повідомлень в одному потоці - приміщенням повідомлень у власній поштовій скриньці.

Передача Handler

Один із способів, яким HandlerThread може виконати роботу в головному потоці, заснований на передачі головним потоком свого об'єкта Handler в HandlerThread.

Головний потік являє собою цикл повідомлень з обробниками і Looper. При створенні екземпляра Handler в головному потоці він асоціюється з екземпляром Looper головного потоку.

Потім цей екземпляр Handler можна передати іншому потоку. Переданий екземпляр Handler зберігає зв'язок з Looper потокустворювача.

Всі повідомлення, за які відповідає Handler, будуть оброблятися в черзі головного потоку. Те, що відбувається дуже схоже на те, як ми планували операції в фоновому потоці з головного потоку з використанням Handler екземпляра ThumbnailDownloader.

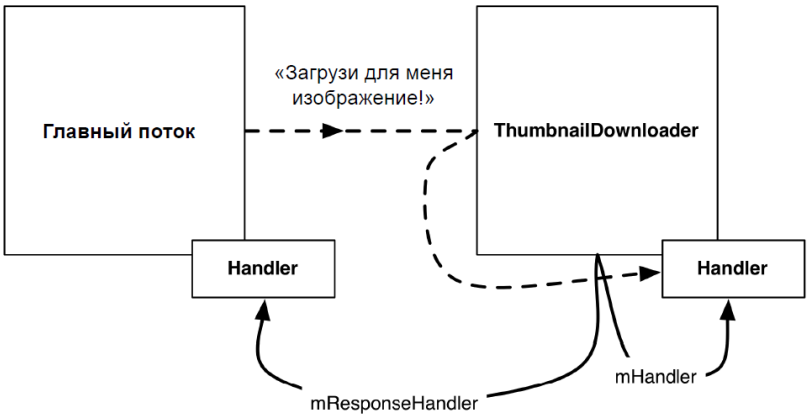

Рис. 8. Планування операцій в ThumbnailDownloader з головного потоку

Аналогічним чином можна планувати операції в головному потоці з фонового потоку з використанням екземпляра Handler, приєднаного до головного потоку.

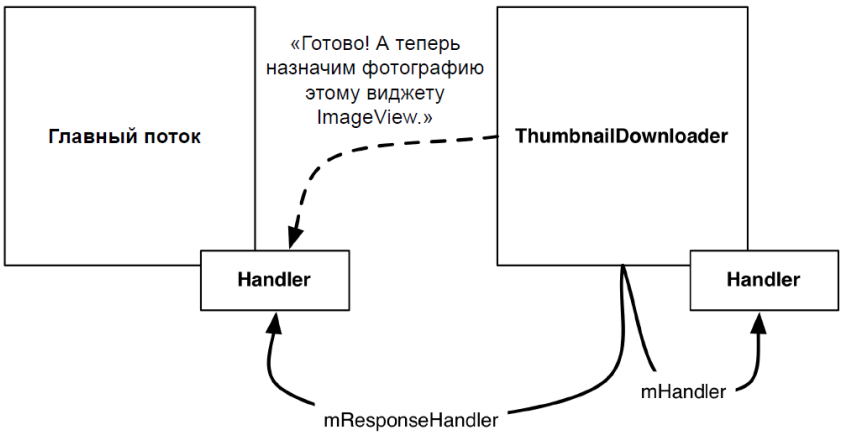

Рис. 9. Планування операцій в головному потоці з ThumbnailDownloader

У файлі ThumbnailDownloader.java додайте змінну mRes ponseHandler для зберігання екземпляра Handler, переданого з головного потоку. Потім замініть конструктор іншим, який отримує Handler і задає змінну, і додайте інтерфейс слухача для передачі відповідей.

Лістинг 6. Додавання обробника відповіді (ThumbnailDownloader.java)

```
public class ThumbnailDownloader extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map <Token, String> requestMap =
            Collections.synchronizedMap (new HashMap <Token, String> ());
    Handler mResponseHandler;
    Listener <Token> mListener;

    public interface Listener <Token> {
        void onThumbnailDownloaded (Token token, Bitmap thumbnail);
    }
```

```
    public void setListener (Listener <Token> listener) {
        mListener = listener;
    }
    public ThumbnailDownloader () {
        super (TAG);
    public ThumbnailDownloader (Handler responseHandler) {
        super (TAG);
        mResponseHandler = responseHandler;
}
```

Потім змініть клас PhotoGalleryFragment так, щоб він передавав Handler класу ThumbnailDownloader, а також Listener для завдання екземплярів Bitmap, що повертаються, по дескрипторамImageView. За замовчуванням Handler приєднує себе до екземпляра Looper поточного потоку. Так як екземпляр Handler створюється в onCreate (...), він буде приєднаний до екземпляра Looper головного потоку.

Лістинг 7. Підключення до обробника відповіді (PhotoGalleryFragment.java)

```
@Override
public void onCreate (Bundle savedInstanceState) {
    super.onCreate (savedInstanceState);

    setRetainInstance (true);
    new FetchItemsTask (). execute ();

    mThumbnailThread = new ThumbnailDownloader ();
    mThumbnailThread = new ThumbnailDownloader (new Handler ());
    mThumbnailThread.setListener (new ThumbnailDownloader.Listener <ImageView>
    () {
    public void onThumbnailDownloaded (ImageView imageView, Bitmap thumbnail) {
        if (isVisible ()) {
             imageView.setImageBitmap (thumbnail);
        }
    }
    });
    mThumbnailThread.start ();
    mThumbnailThread.getLooper ();
    Log.i (TAG, "Background thread started");
}
```

Тепер ThumbnailDownloader має доступ до екземпляра Handler, пов'язаному з екземпляром Looper головного потоку, через поле mResponseHandler. Крім того, він наказує Listener виконувати операції користувальницького інтерфейсу з об'єктами Bitmap, що повертаються. Виклик imageView.setImageBitmap (Bitmap) захищається викликом Fragment.isVisible (). Захист запобігає призначення зображення застарілому віджету ImageView.

Аналогічним чином можна відправити головному потоку нестандартний об'єкт Messsage. Для цього буде потрібен інший субклас Handler з перевизначенням handleMessage (...). Однак замість цього ми використовуємо інший зручний метод

```
Runnable myRunnable = new Runnable () {
    public void run () {
        / * Тут розташовується ваш код * /
    }
};
Message m = mHandler.obtainMessage ();
m.callback = myRunnable;
```

Якщо у Message задано поле callback, то замість приймача Handler виконується об'єкт Runnable з поля callback.

Включіть в ThumbnailDownloader.handleRequest () наступний код.

Лістинг 8. Завантаження і виведення (ThumbnailDownloader.java)

```
...
private void handleRequest (final Token token) {
  try {
    final String url = requestMap.get (token);
    if (url == null)
      return;

    byte [] bitmapBytes = new FlickrFetchr (). getUrlBytes (url);
    final Bitmap bitmap = BitmapFactory
           .decodeByteArray (bitmapBytes, 0, bitmapBytes.length);
    Log.i (TAG, "Bitmap created");

    mResponseHandler.post (new Runnable () {
    public void run () {
      if (requestMap.get (token)! = url)
        return;

      requestMap.remove (token);
      mListener.onThumbnailDownloaded (token, bitmap);
      }
    });
  } Catch (IOException ioe) {
    Log.e (TAG, "Error downloading image", ioe);
  }
}
```

А оскільки mResponseHandler зв'язується з Looper головного потоку, цей код поновлення користувальницького інтерфейсу буде виконаний в головному потоці.

Цей код спочатку він перевіряє requestMap. Така перевірка необхідна, тому що GridView заново використовує свої представлення. На той час, коли ThumbnailDownloader завершить завантаження Bitmap, може виявитися, що віджет Grid-View вже переробив ImageView і запросив для нього зображення з іншої URL-адреси. Ця перевірка гарантує, що кожен об'єкт Token отримає правильне зображення, навіть якщо за минулий час був зроблений інший запит.

Нарешті, ми видаляємо Token з requestMap і призначаємо зображення для Token.

Напишіть наступний метод для видалення всіх запитів з черги.

Лістинг 9. Додавання методу очищення черги (ThumbnailDownloader.java)

```
public void clearQueue () {
    mHandler.removeMessages (MESSAGE_DOWNLOAD);
    requestMap.clear ();
}
```

Потім очистіть завантажувач в PhotoGalleryFragment при знищенні представлення.

Лістинг 10. Виклик методу очищення черги (PhotoGalleryFragment.java)

```
@Override
public void onDestroyView () {
    super.onDestroyView ();
    mThumbnailThread.clearQueue ();
}
```

Відкрийте програму PhotoGallery. Перейдіть та поспостерігайте за тим, як відбувається динамічне завантаження зображень.

Додаток PhotoGallery виконує свою основну функцію - виведення зображень з Flickr. Доповнити його новими функціями: пошуком фотографій і відкриттям сторінки Flickr кожної фотографії в веб-представленні.

AsyncTask і потоки

Клас AsyncTask призначений для короткочасної роботи, яка не повторюється занадто часто.

В Android 3.2 реалізація AsyncTask була серйозно змінена. Починаючи з Android 3.2, AsyncTask не створює потік для кожного екземпляра AsyncTask. Замість цього він створює об'єкт Executor для виконання фонової роботи всіх екземплярів AsyncTask в одному фоновому потоці. Це означає, що екземпляри AsyncTask будуть виконуватися один за одним, а тривала операція AsyncTask не дозволить іншим екземплярам AsyncTask отримати процесорний час.

Організувати безпечне паралельне виконання AsyncTask з використанням пулу потоків можливо. Якщо ви розглядаєте таке рішення, зазвичай краще самостійно організувати багатопоточне виконання, використовуючи об'єкти Handler для взаємодії з головним потоком, коли виникне така необхідність.

Завдання. Попереднє завантаження і кешування

Користувачі розуміють, що не все відбувається миттєво (або принаймні більшість користувачів). Але незважаючи на це, програмісти прагнуть до досконалості.

Для досягнення моментального відгуку в більшості реальних додатків наведений код розширюється в двох напрямках:

- додавання рівня кешування;
- попереднє завантаження зображень.

Кеш являє собою місце для зберігання певної кількості об'єктів Bitmap, щоб вони залишалися в пам'яті навіть після завершення використання. Об'єм кеша обмежений, тому вам знадобиться стратегія вибору об'єктів, зберігаються при вичерпанні вільного місця. Багато кешів використовують стратегію LRU (Least Recently Used): при нестачі вільного місця з кешу видаляється елемент, який найдовше не використовувався.

Бібліотека підтримки Android містить клас з ім'ям LruCache, який реалізує стратегію LRU. У першому завданні використовуйте LruCache для додавання найпростішого кешування ThumbnailDownloader. Кожен раз, коли для URL-адреси завантажується об'єкт Bitmap, ви ставите його в кеш. Потім, коли потрібно завантажити нове зображення, ви спочатку перевіряєте вміст кеша і дивитеся, чи немає його в кеші.

Після того як в програмі буде створено кеш, він може використовуватися для попереднього завантаження, тобто завантаження даних в кеш ще до того, як вони фактично будуть потрібні програмі. Тим самим запобігає затримку для завантаження об'єктів Bitmap до їх виведення.

Якісно реалізувати попереднє завантаження непросто, але вона істотно змінює сприйняття додатку користувачем.

У другому завданні для кожного виведеного елемента GalleryItem виконайте попереднє завантаження 10 попередніх і 10 наступних елементів GalleryItem.

Лабораторна робота №5

Тема: Пошук контенту

Наступним кроком в роботі над додатком PhotoGallery стане пошук фотографій на Flickr.

Пошук був інтегрований Android з самого початку, але він (як і кнопка меню) з тих пір помітно змінився. Як і меню, новий код пошуку будувався на базі існуючих API. Таким чином, будуючи пошук в традиційному стилі, ви насправді готуєтеся реалізувати повноцінний сучасний пошук Jelly Bean.

Пошук в Flickr

Почнемо з того, що потрібно зробити на стороні Flickr. Для виконання пошуку в Flickr слід викликати метод flickr.photos.search.

Метод отримує нові параметри, що визначають умови пошуку, зокрема параметр рядка запиту.

Внесіть зміни, представлені в лістингу 1, щоб додати в FlickrFetchr новий метод пошуку. Оскільки розбір даних результатів search і getRecent для Gal-leryItem відбувається однаково, ми переробимо частину старого коду з fetchItems() новий метод, який буде називатися downloadGalleryItems (String).

Старий код fetchItems() переміщається в нову версію fetchItems (), а не видаляється з програми.

Лістинг 1. Додавання методу пошуку фотографій Flickr (FlickrFetchr.java)

```
public class FlickrFetchr {
  public static final String TAG = "PhotoFetcher";

  private static final String ENDPOINT =
  "http://api.flickr.com/services/rest/";
  private static final String API_KEY = "4f721bbafa75bf6d2cb5af54f937bb70";
  private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
  private static final String METHOD_SEARCH = "flickr.photos.search";
  private static final String PARAM_EXTRAS = "extras";
  private static final String PARAM_TEXT = "text";
  ...

  public ArrayList <GalleryItem> fetchItems () {
  public ArrayList <GalleryItem> downloadGalleryItems (String url) {
     ArrayList <GalleryItem> items = new ArrayList <GalleryItem> ();

      try {
          String url = Uri.parse (ENDPOINT) .buildUpon ()
              .appendQueryParameter ( "method", METHOD_GET_RECENT)
              .appendQueryParameter ( "api_key", API_KEY)
              .appendQueryParameter (PARAM_EXTRAS, EXTRA_SMALL_URL)
              .build (). toString ();
          String xmlString = getUrl (url);
          Log.i (TAG, "Received xml:" + xmlString);
          XmlPullParserFactory factory = XmlPullParserFactory.newInstance ();
          XmlPullParser parser = factory.newPullParser ();
          parser.setInput (new StringReader (xmlString));

          parseItems (items, parser);
      } Catch (IOException ioe) {
          Log.e (TAG, "Failed to fetch items", ioe);
      } Catch (XmlPullParserException xppe) {
          Log.e (TAG, "Failed to parse items", xppe);
      }
      return items;
  }
```

```
    public ArrayList <GalleryItem> fetchItems () {
        // Сюди переміщається код, наведений вище
        String url = Uri.parse (ENDPOINT) .buildUpon ()
            .appendQueryParameter ( "method", METHOD_GET_RECENT)
            .appendQueryParameter ( "api_key", API_KEY)
            .appendQueryParameter (PARAM_EXTRAS, EXTRA_SMALL_URL)
            .build (). toString ();
        return downloadGalleryItems (url);
    }

    public ArrayList <GalleryItem> search (String query) {
        String url = Uri.parse (ENDPOINT) .buildUpon ()
            .appendQueryParameter ( "method", METHOD_SEARCH)
            .appendQueryParameter ( "api_key", API_KEY)
            .appendQueryParameter (PARAM_EXTRAS, EXTRA_SMALL_URL)
            .appendQueryParameter (PARAM_TEXT, query)
            .build (). toString ();
        return downloadGalleryItems (url);
    }
}
```

Метод downloadGalleryItems (String) використовується двічі, тому що код завантаження і розбору XML однаковий для search і getRecent. Пошук зводиться до простого викликом нового методу flickr.photos.search з передачею закодованого рядка запиту в параметрі text.

Тепер підключимо тестовий код для виклику коду пошуку з PhotoGalleryFragment. FetchItemsTask. Поки ми використовуємо жорстко запрограмований запит, щоб переконатися в правильності роботи пошуку.

Лістинг 2. Код з фіксованим запитом пошуку (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask <Void, Void, ArrayList
<GalleryItem >> {
    @Override
    protected ArrayList <GalleryItem> doInBackground (Void ... params) {
        String query = "android"; // Тільки для тестування

        if (query! = null) {
            return new FlickrFetchr (). search (query);
        } Else {
            return new FlickrFetchr (). fetchItems ();
        }
    }
    @Override
    protected void onPostExecute (ArrayList <GalleryItem> items) {
        ...
    }
}
...
}
```

За замовчуванням використовується старий код getRecent. Якщо пошуковий запит відмінний від null (а тепер ця умова виконується завжди), то FetchItemsTask отримує результати пошуку.

Запустіть PhotoGallery і перевірте результати.

Діалогове вікно пошуку

Реалізуємо в PhotoGallery пошуковий інтерфейс Android. Почнемо з традиційного інтерфейсу діалогового вікна.

Створення пошукового інтерфейсу

У Honeycomb творці Android позбулися фізичних кнопок пошуку. Втім, навіть до цього присутність кнопки пошуку не було гарантовано. Сучасні програми Android з функцією пошуку зобов'язані реалізувати кнопку пошуку, якщо вони повинні працювати на пристроях до 3.0.

Реалізація пошуку не так вже складна - досить викликати метод Activity.onSearchRequested(). Цей метод виконує точно таку ж операцію, як і натискання кнопки пошуку.

Додайте XML-файл в меню PhotoGallery res / menu / fragment_photo_gallery.xml. Нашому додатку також знадобиться інтерфейс для очищення умови пошуку, тому додайте кнопку скидання.

Лістинг 3. Додавання команд меню пошуку (res / menu / fragment_photo_gallery.xml)

```
<Menu xmlns: android = "http://schemas.android.com/apk/res/android">
    <Item android: id = "@ + id / menu_item_search"
        android: title = "@ string / search"
        android: icon = "@ android: drawable / ic_menu_search"
        android: showAsAction = "ifRoom"
    />
    <Item android: id = "@ + id / menu_item_clear"
        android: title = "@ string / clear_search"
        android: icon = "@ android: drawable / ic_menu_close_clear_cancel"
        android: showAsAction = "ifRoom"
    />
</ Menu>
```

Зараз нам не вистачає пари рядків; додайте їх в strings.xml. (Пізніше нам знадобиться рядок з підказкою пошуку, заодно додамо і її.)

Лістинг 4. Додавання рядків пошуку (res / values / strings.xml)

```
<Resources>
    ...

    <String name = "title_activity_photo_gallery"> PhotoGalleryActivity </
    string>
    <String name = "search_hint"> Search Flickr </ string>
    <String name = "search"> Search </ string>
    <String name = "clear_search"> Clear Search </ string>
</ Resources>
```

Підключіть зворотні виклики командного меню. Як сказано вище, для кнопки пошуку буде викликатися метод onSearchRequested (). Для кнопки скасування поки не буде робитися нічого.

Лістинг 5. Зворотні виклики командного меню (PhotoGalleryFragment.java)

```
@Override
public void onCreate (Bundle savedInstanceState) {
    super.onCreate (savedInstanceState);

    setRetainInstance (true);
    setHasOptionsMenu (true);
    ...
}

...

@Override
public void onDestroyView () {
    ...
}
```

```
@Override
public void onCreateOptionsMenu (Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu (menu, inflater);
    inflater.inflate (R.menu.fragment_photo_gallery, menu);
}
@Override
public boolean onOptionsItemSelected (MenuItem item) {
    switch (item.getItemId ()) {
        case R.id.menu_item_search:
            getActivity (). onSearchRequested ();
            return true;
        case R.id.menu_item_clear:
            return true;
        default:
            return super.onOptionsItemSelected (item);
    }
}
```

Спробуйте новий інтерфейс меню і переконайтеся в тому, що він правильно відображається.

При натисканні кнопки пошуку зараз нічого не відбувається. Щоб метод onSearchRequested() заробив, необхідно зробити PhotoGalleryActivity пошуковою активністю (searchable activity).

Пошукові активності

Два компонента роблять активність пошуковій. Перший - файл XML. Він містить елемент з ім'ям searchable, який описує, як має відображатися діалогове вікно пошуку. Створіть нову папку з ім'ям res / xml, а в ній - новий файл XML з ім'ям searchable.xml. Заповніть файл простою версією елемента searchable.

Лістинг 6. Конфігурація пошуку (res / xml / searchable.xml)

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Searchable xmlns: android = "http://schemas.android.com/apk/res/android"
    android: label = "@ string / app_name"
    android: hint = "@ string / search_hint"
/>
```

Ця розмітка XML називається конфігурацією пошуку (search configuration). У нашій ситуації конфігурація обмежується рядком підказки і ім'ям програми.

У більш складному додатку цей файл швидко розбухає. Рекомендації з пошуку, голосовий пошук, глобальна конфігурація пошуку, зміни процесів, типи введення - вся ця інформація зберігається в файлі. Базова конфігурація пошуку необхідна навіть в найпростішій реалізації.

Наступний необхідний компонент знаходиться в файлі AndroidManifest.xml. Ви повинні змінити режим запуску додатка, а також оголосити додатковий фільтр інтенту і блок метаданих для PhotoGalleryActivity. Фільтр інтенту повідомляє про прослуховування пошукових інтентів, а метадані пов'язують тільки що написану розмітку XML з активністю.

Всі ці оголошення повідомляють диспетчеру пошуку Android, що ваша активність здатна обробляти пошукові запити, а також визначають її конфігурацію пошуку. Диспетчер пошуку (SearchManager) - служба рівня ОС, що відповідає за виведення діалогового вікна пошуку та управління його взаємодіями.

Відкрийте файл AndroidManifest.xml. Додайте в нього два елементи і атрибут android: launchMode, наведені в лістингу 7.

Лістинг 7. Додавання фільтра інтенту і метаданих (AndroidManifest.xml)

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

    <application
        ...>
        <activity
             android: name = ". PhotoGalleryActivity"
             android: launchMode = "singleTop"
             android: label = "@ string / title_activity_photo_gallery">
             <Intent-filter>
             <Action android: name = "android.intent.action.MAIN" />
             <Category android: name = "android.intent.category.LAUNCHER" />
             </ Intent-filter>
                  <Intent-filter>

             <Action android: name = "android.intent.action.SEARCH" />
             </ Intent-filter>
                  <Meta-data android: name = "android.app.searchable"
             android: resource = "@ xml / searchable" />
        </ Activity>
    </ Application>
</ Manifest>
```

Перше додавання - визначення фільтра інтенту. Результати пошуку передаються за допомогою виклику startActivity (...) з інтенту, якому призначено дію action. intent.action.SEARCH. Пошуковий запит включається в Інтент як доповнення. Таким чином, щоб показати, що активність може обробляти результати пошуку, ви визначаєте фільтр для інтенту action.intent.action.SEARCH.

Друге додавання - тег метаданих. Замість атрибута android:value він використовує android: resource. Наступний приклад демонструє різницю між двома формами. Припустимо, ви посилаєтеся на один строковий ресурс в двох різних тегах метаданих:

```
<Meta-data android: name = "metadata.value"
    android: value = "@ string / app_name" />
<Meta-data android: name = "metadata.resource"
    android: resource = "@ string / app_name" />
```

Звернувшись до значення metadata.value, ми б виявили, що в ньому міститься рядок «PhotoGallery» - значення, яке зберігається в @string/app_name. Однак значенням metadata.resource буде цілочисельний ідентифікатор цього ресурсу. Іншими словами, значення metadata.resource відповідає значенню R.string. app_name в коді.

SearchManager повинен передаватися цілочисельний ідентифікатор searchable.xml, а не строкове значення цього файлу XML; відповідно ми використовуємо android:resource і передаємо SearchManager ідентифікатор ресурсу цього файлу.

Атрибут android:launchMode в тегу activity визначає режим запуску активності.

Після виконання всієї необхідної підготовки ми можемо відкрити діалогове вікно пошуку. Відкрийте програму PhotoGallery і натисніть кнопку пошуку в меню.

Апаратна кнопка пошуку

Такий самий пароль, який виконується при ручному виклику onSearchRequested(), буде виконуватися при натисканні апаратної кнопки пошуку на старих пристроях. Якщо ви хочете самостійно переконатися в цьому, включіть в будь-який емулятор до версії 3.0 апаратну кнопку пошуку; для цього слід налаштувати емулятор на використання апаратної клавіатури, як показано на рис. 3.

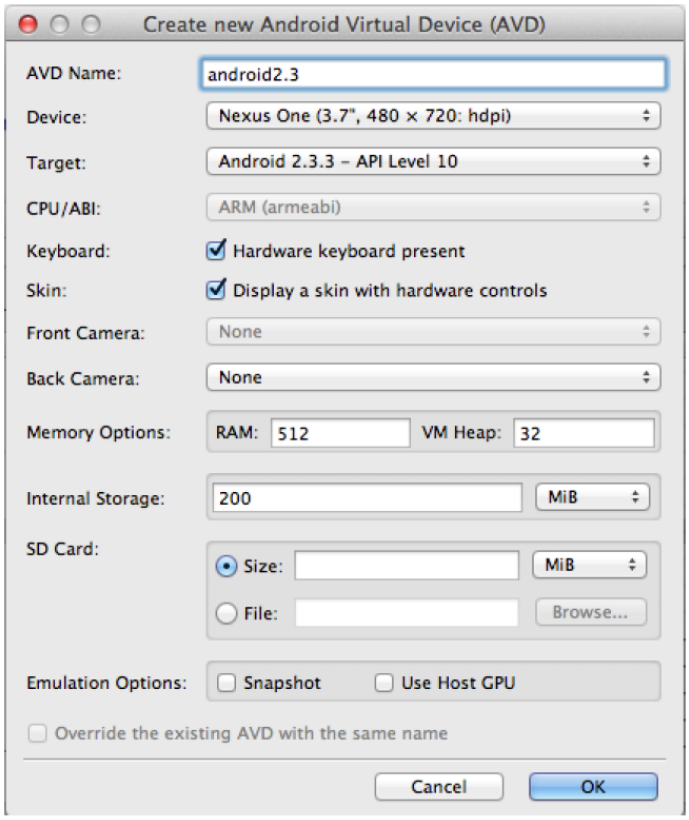

Рис. 3. Додавання підтримки апаратної клавіатури

Як працює пошук

Пошук в Android базується на концепції пошукової активності. Пошукова активність визначається двома компонентами: пошуковим фільтром інтенту і пошуковими метаданими конфігурації пошуку.

З апаратною кнопкою пошуку кожна пошукова взаємодія до самого інтенту пошуку обробляється системою. Вона звертається до файлу AndroidManifest.xml і перевіряє, чи є активність пошуковою. Якщо перевірка дає позитивний результат, то система виводить активність діалогового вікна пошуку поверх вашої активності. Ця активність запускає пошук, відправляючи новий інтент.

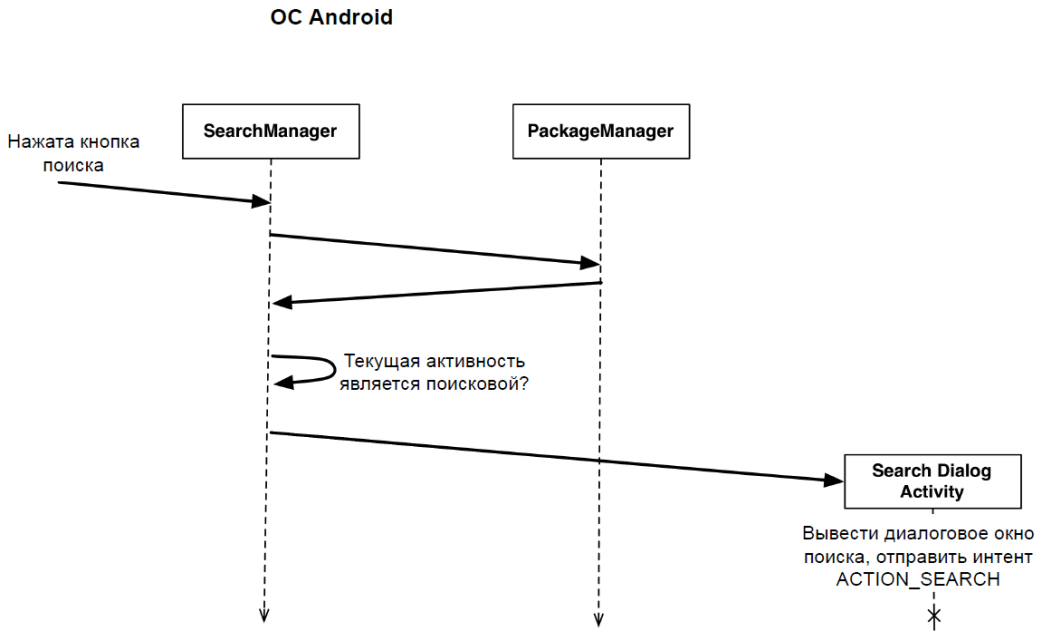

Рис. 4. Системний пошук

Це означає, що при натисканні кнопки пошуку запускається нова активність. Однак в нашому випадку цього не відбувається. Чому? Тому що ми додали атрибут android: launchMode = "singleTop" (лістинг 7), що змінює режим запуску.

Режими запуску і нові інтенти

Режими запуску (launch mode) визначають, як ваша активність повинна запускатися при отриманні нового інтенту, а в деяких ситуаціях - як вона повинна поводитися при запуску інтенту для запуску іншої активності.

Таблиця 1. Режими запуску

Режим запуску	**Поведінка**
standard	Поведінка за замовчуванням для кожного отриманого інтенту запускається нова активність
singleTop	Якщо екземпляр цієї активності вже знаходиться у верхній позиції стека повернення, то новий інтент передається існуючій активності замість створення нової
singleTask	Активність запускається в окремому завданні. Якщо активність вже існує в завданні, то всі активності, що знаходяться вище за неї в стеці повернення, скидаються, а новий інтент передається існуючій активності
singleInstance	Активність запускається в окремому завданні. Це єдина активність свого завдання - якщо з завдання запускаються інші активності, вони будуть запущені в окремих завданнях. Якщо активність вже існує, то новий інтент передається існуючій активності

Всі активності, написані нами до цього моменту, використовували стандартний режим запуску. Коли інтент перетворюється в активність зі стандартним режимом запуску, новий екземпляр цієї активності створюється і додається в стек повернення.

Для PhotoGalleryActivity замість нього доводиться задавати режим запуску singleTop.

Це означає, що замість запуску нової активності отриманий вами пошуковий Інтент передаватиметься вже виконуваної активності PhotoGalleryActivity на вершині стека повернення.

Для отримання нового інтенту ви перевизначаєте метод onNewIntent (Intent) в Activity. При кожному отриманні нового інтенту слід оновлювати елементи PhotoGalleryFragment.

Необхідно переробити PhotoGalleryFragment і включити метод updateItems(), який запускає FetchItemsTask для поновлення поточних елементів.

Лістинг 8. Додавання методу поновлення елементів (PhotoGalleryFragment.java)

```
@Override
public void onCreate (Bundle savedInstanceState) {
    super.onCreate (savedInstanceState);

    setRetainInstance (true);

    ~~new FetchItemsTask (). execute ();~~
    updateItems ();

    mThumbnailThread = new ThumbnailDownloader <ImageView> (new Handler ());
    mThumbnailThread.setListener (new ThumbnailDownloader.Listener <ImageView>
    () {
```

```
            ...
        });
        mThumbnailThread.start ();
        mThumbnailThread.getLooper ();
    }

    public void updateItems () {
        new FetchItemsTask (). execute ();
    }
```

Потім додайте в PhotoGalleryActivity перевизначення onNewIntent (Intent), яке перекриває новий інтент і оновлює елементи PhotoGallery-Fragment:

Лістинг 9. Перевизначення onNewIntent (...) (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends SingleFragmentActivity {
    private static final String TAG = "PhotoGalleryActivity";

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

    @Override
    public void onNewIntent (Intent intent) {
        PhotoGalleryFragment fragment = (PhotoGalleryFragment)
          getSupportFragmentManager (). findFragmentById
          (R.id.fragmentContainer);

        if (Intent.ACTION_SEARCH.equals (intent.getAction ())) {
          String query = intent.getStringExtra (SearchManager.QUERY);
          Log.i (TAG, "Received a new search query:" + query);
        }

        fragment.updateItems ();
    }
}
```

Тепер при проведенні пошуку в LogCat повинна з'являтися інформація про отримання PhotoGalleryActivity нового інтенту. Всі елементи PhotoGallery спочатку повертаються до зображення, а потім оновлюються.

Одне важливе зауваження з приводу onNewIntent (Intent): якщо вам знадобиться значення нового інтенту, збережіть його де-небудь. У значенні, що отримується від getIntent(), буде міститися старий інтент, а не новий. Справа в тому, що метод getIntent() призначений для повернення інтенту, що запустив цю активність, а не останнього отриманого нею інтенту.

Наступним кроком повинна стати інтеграція пошукового запиту в вашу програму. В нашій реалізації пошуку в будь-який момент часу буде існувати тільки один пошуковий запит. Було б зручно реалізувати довгострокове зберігання цього запиту .

Просте збереження з використанням механізму загальних налаштувань

Для збереження даних можна скористатися сериалізацією об'єктів в флеш-пам'ять. Однак для простих значень механізм загальних налаштувань простіше реалізується і зручніше у використанні.

Сховище загальних налаштувань (shared preferences) являє собою файли в файловій системі, для читання і редагування яких використовується клас Shared-Preferences. Примірник SharedPreferences працює як сховище пар «ключ-значення», що має багато спільного з Bundle, але з можливістю довгострокового зберігання. Ключами є рядки, а значеннями - атомарні типи даних. При найближчому розгляді з'ясовується, що файли містять просту розмітку XML.

Щоб приступити до використання простого строкового значення в загальних налаштуваннях, слід визначити константу, яка стане ключем для вашого значення. Увімкніть константу в FlickrFetchr.

Лістинг 10. Константа загальних налаштувань (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";
    public static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String ENDPOINT =
    "http://api.flickr.com/services/rest/";
    ...
```

Для отримання конкретного екземпляра SharedPreferences можна скористатися методом Context.getSharedPreferences (String, int). Однак на практиці часто важливий не конкретний екземпляр, а його спільне використання в межах всієї програми. У таких ситуаціях краще використовувати метод PreferenceManager.ge tDefaultSharedPreferences (Context), який повертає екземпляр з ім'ям за замовчуванням і закритими (private) дозволами.

У класі PhotoGalleryActivity отримаєте об'єкт SharedPreferences за замовчуванням і збережіть запит.

Лістинг 11. Збереження пошукового запиту (PhotoGalleryActivity.java)

```
@Override
public void onNewIntent (Intent intent) {
    PhotoGalleryFragment fragment = (PhotoGalleryFragment)
getSupportFragmentManager ()
         .findFragmentById (R.id.fragmentContainer);
    if (Intent.ACTION_SEARCH.equals (intent.getAction ())) {
         String query = intent.getStringExtra (SearchManager.QUERY);
         Log.i (TAG, "Received a new search query:" + query);
         PreferenceManager.getDefaultSharedPreferences (this)
             .edit ()
             .putString (FlickrFetchr.PREF_SEARCH_QUERY, query)
             .commit ();
    }
    fragment.updateItems ();
}
```

У наведеному вище коді ми викликаємо SharedPreferences.edit() для отримання примірника SharedPreferences.Editor. Ми використовуємо цей клас для збереження значень в SharedPreferences. Він дозволяє групувати зміни в транзакціях, за аналогією з тим, як це робилося з FragmentTransaction: множинні зміни групуються разом для збереження однієї операції запису.

Після того як всі зміни будуть внесені, виклик commit() для об'єкта Editor робить їх видимими для інших користувачів файлу SharedPreferences.

Отримання раніше збереженого значення зводиться до простого виклику SharedPreferences.getString (...), getInt (...) або іншого методу, що відповідає типу даних. Додайте в PhotoGalleryFragment код вибірки пошукового запиту з об'єкта SharedPreferences за замовчуванням.

Лістинг 12. Читання збереженого пошукового запиту (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask <Void, Void, ArrayList
<GalleryItem >> {
    @Override
    protected ArrayList <GalleryItem> doInBackground (Void ... params) {
        String query = "android"; // just for testing
        Activity activity = getActivity ();
        if (activity == null)
            return new ArrayList <GalleryItem> ();
        String query = PreferenceManager.getDefaultSharedPreferences (activity)
            .getString (FlickrFetchr.PREF_SEARCH_QUERY, null);
        if (query! = null) {
            return new FlickrFetchr (). search (query);
```

```
        } Else {
            return new FlickrFetchr (). fetchItems ();
        }
    }
    @Override
    protected void onPostExecute (ArrayList <GalleryItem> items) {
            ...
    }
}
```

Тепер пошук повинен працювати. Відкрийте програму PhotoGallery, спробуйте що-небудь пошукати і подивіться, що з цього вийде.

Щоб реалізувати скасування пошуку, видаліть умову пошуку із загальних налаштувань і знову викличте updateItems ().

Лістинг 13. Реалізація скасування (PhotoGalleryFragment.java)

```
@Override
public boolean onOptionsItemSelected (MenuItem item) {
    switch (item.getItemId ()) {
        ...
        case R.id.menu_item_clear:
            PreferenceManager.getDefaultSharedPreferences (getActivity ())
                .edit ()
                .putString (FlickrFetchr.PREF_SEARCH_QUERY, null)
                .commit ();
                updateItems ();
                return true;
            default:
                return super.onOptionsItemSelected (item);
        }
}
```

Використання SearchView в Android версій вище 3.0

Побудований нами пошуковий інтерфейс працює де завгодно. Проте його робота не відповідає рекомендаціям для Honeycomb.

У Honeycomb додався новий клас з ім'ям SearchView. SearchView є представленням дії (action view) - тобто представленням, яке може включатися на панель дій. SearchView дозволяє перемістити весь пошуковий інтерфейс на панель дій активності (замість використання діалогового вікна, який накладається на активність). В цьому випадку пошуковий інтерфейс використовує те саме стильове оформлення і тему, як ваш додаток.

Використання представлення дії зводиться до простого додаванню атрибута android: actionViewClass в тег елемента меню.

Лістинг 14. Додавання представлення дії в меню (res / menu / fragment_photo_ gallery.xml)

```
<Menu xmlns: android = "http://schemas.android.com/apk/res/android">
    <Item android: id = "@ + id / menu_item_search"
        android: title = "@ string / search"
        android: icon = "@ android: drawable / ic_menu_search"
        android: showAsAction = "ifRoom"
        android: actionViewClass = "android.widget.SearchView"
        />
    <Item android: id = "@ + id / menu_item_clear"
        ...
        />
</ Menu>
```

Визначаючи представлення дії, замість традиційного представлення для цього елемента панелі дії потрібно використовувати цей клас представлення. Зазвичай це також означає зміни в поведінці. Наприклад, SearchView не генерує зворотні виклики onOptionsItemSelected (...).

Ви можете залишити ці зворотні виклики для старих пристроїв, що не підтримують представлення дій.

(SearchViewCompat не є версією SearchView, яка може використовуватися на старих пристроях. Замість цього клас містить пару статичних методів, що спрощують виборчу вставку SearchView там, де ця функціональність доступна.

Запустіть PhotoGallery і подивіться, як виглядає представлення SearchView. Втім, нічого робити воно не буде. Перш ніж SearchView почне відправляти пошукові інтенти, йому необхідно знати вашу конфігурацію пошуку. Необхідно додати в onCreateOptionsMenu (...) код, який витягує конфігурацію пошуку і відправляє її SearchView.

У SearchManager є метод getSearchableInfo (ComponentName), який перевіряє маніфест, упаковує всю актуальну інформацію та повертає її у вигляді об'єкта SearchableInfo. Далі залишається лише передати об'єкт SearchableInfo примірнику SearchView. Для цього потрібно ввести код, наведений у лістингу 15.

Лістинг 15. Налаштування SearchView (PhotoGalleryFragment.java)

```
@Override
@TargetApi (11)
public void onCreateOptionsMenu (Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu (menu, inflater);
    inflater.inflate (R.menu.fragment_photo_gallery, menu);
    if (Build.VERSION.SDK_INT> = Build.VERSION_CODES.HONEYCOMB) {

        // Отримання SearchView
        MenuItem searchItem = menu.findItem (R.id.menu_item_search);
        SearchView searchView = (SearchView) searchItem.getActionView ();

        // Отримання даних з файлу searchable.xml
        // в вигляді об'єкта SearchableInfo
        SearchManager searchManager = (SearchManager) getActivity ()
            .getSystemService (Context.SEARCH_SERVICE);
        ComponentName name = getActivity (). GetComponentName ();
        SearchableInfo searchInfo = searchManager.getSearchableInfo (name);

        searchView.setSearchableInfo (searchInfo);
    }
}
```

Все починається з пошуку SearchView. Для цього ми шукаємо пошуковий елемент MenuItem за ідентифікатором, а потім отримуємо його представлення дії викликом getActionView().

Потім у SearchManager запитується конфігурація пошуку. SearchManager - системна служба, яка відповідає за все, що відноситься до пошуку. Раніше саме об'єкт SearchManager діяв непомітно для нас, витягуючи конфігурацію пошуку і відображаючи пошуковий інтерфейс. Вся інформація про пошук, що включає ім'я активності, яка повинна отримати інтент, і все інше з searchable.xml, зберігається в об'єкті SearchableInfo, який ми отримуємо викликом getSearchabl eInfo (ComponentName).

Після отримання об'єкта SearchableInfo ми повідомляємо про нього SearchView викликом setSearchableInfo (SearchableInfo). В результаті екземпляр SearchView повністю пов'язаний з додатком.

Завдання 1.

Перше полягає в використанні методу Activity.startSearch (...).

У внутрішній реалізації onSearchRequested() викликає Activity.startS-earch (...) - докладний спосіб запуску діалогового вікна пошуку.

Метод startSearch (...) дозволяє задати вихідний запит, який відображається в полі EditText, додати об'єкт Bundle з даними, що відправляються пошуковій активності-одержувачу, в доповненнях інтенту, або запросити глобальне діалогове вікно веб-пошуку (аналогічне тому, яке ви побачите при натисканні кнопки пошуку на домашньому екрані).

У першому завданні використовуйте метод Activity.startSearch (...) для заповнення діалогового вікна пошуку поточним запитом і його виділення.

Друге завдання полягає, щоб при запуску нового пошуку виводилася загальна кількість доступних результатів пошуку в оповіщенні Toast. Для цього вам доведеться звернутися до розмітки XML, отриманої від Flickr. У ній присутній атрибут верхнього рівня з кількістю повернутих результатів.

Лабораторна робота № 6

Тема: Робота з локальними базами даних і SQLite

Додаткам з великими і складними наборами даних часто виявляється недостатньо можливостей простих файлових форматів JSON. Наприклад, в додатку RunTracker користувач може нескінченно довго відстежувати своє місце розташування, в результаті чого буде згенерований великий обсяг даних.

В Android для зберігання таких наборів зазвичай використовуються бази даних SQLite. SQLite - мультиплатформенна бібліотека, яка поширюється з відкритим кодом, яка надає потужний API реляційної бази даних для роботи з одним файлом на диску.

Android Java-інтерфейс до SQLite реалізований в класі SQLiteDatabase, повертає набори даних у вигляді екземплярів Cursor. Створимо для RunTracker механізм зберігання даних, що використовує базу даних для зберігання інформації про серії і їх позицій. також буде створена нова активність зі списком серій і фрагмент для створення і відстеження серій.

Зберігання серій і позицій в базі даних

Щоб зберегти будь-які дані в базі, спочатку необхідно визначити структуру бази даних і відкрити її. Оскільки ця задача досить типова для Android, для неї існує допоміжний клас. SQLiteOpenHelper інкапсулює службові операції по створенню, відкриттю і оновленню баз даних для зберігання даних програми.

У RunTracker ми створимо субклас SQLiteOpenHelper з ім'ям RunDatabaseHelper. Об'єкт RunManager буде містити закритий екземпляр RunDatabaseHelper і надавати додатку API для вставки, обробки запитів і інших операцій з даними в базі. Клас RunDatabaseHelper надає методи, які RunManager викликатиме для реалізації більшої частини свого API.

При проектуванні API зберігання інформації в базі даних розробник зазвичай створює по одному субкласу SQLiteOpenHelper для кожного виду баз даних, якими він збирається керувати. Потім для кожного окремого файлу бази даних SQKite створюється один екземпляр цього субкласу. У більшості додатків, включаючи RunTracker, створюється лише один субклас SQLiteOpenHelper, а його єдиний екземпляр використовується спільно всіма компонентами програми.

В об'єктно-орієнтованому програмуванні при проектуванні баз даних найчастіше для кожного класу в моделі даних програми створюється окрема таблиця. У додатку RunTracker в базі даних повинні зберігатися дані двох класів: Run і Location.

Відповідно в нашому прикладі будуть створені дві таблиці: run (для серій) і location (для даних місця розташування). Об'єкт Run може містити багато об'єктів Location, тому в таблицю location буде включений стовпець зовнішнього ключа run_id, що посилається на столбец_id таблиці run.

Структура таблиць зображена на рис. 1.

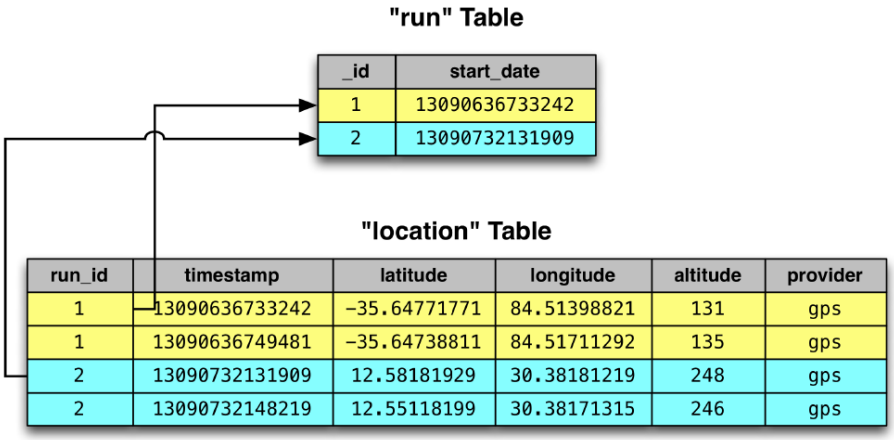

Рис. 1. Схема бази даних RunTracker

Створіть клас RunDatabaseHelper і введіть код, наведений у лістингу 1.

Лістинг 1. Вихідна версія RunDatabaseHelper (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    public RunDatabaseHelper (Context context) {
        super (context, DB_NAME, null, VERSION);
    }

    @Override
    public void onCreate (SQLiteDatabase db) {
        // Створення таблиці "run"
        db.execSQL ( "create table run (" +
        "_Id integer primary key autoincrement, start_date integer)");

        // Створення таблиці "location"
        db.execSQL ( "create table location (" +
            "Timestamp integer, latitude real, longitude real, altitude real," +
            "provider varchar (100), run_id integer references run (_id))");
    }

    @Override
    public void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) {
        // Тут реалізуються зміни схеми і перетворення даних
        // при оновленні схеми
    }

    public long insertRun (Run run) {
        ContentValues cv = new ContentValues ();
        cv.put (COLUMN_RUN_START_DATE, run.getStartDate (). getTime ());
        return getWritableDatabase (). insert (TABLE_RUN, null, cv);
    }
}
```

Реалізація субкласу SQLiteOpenHelper вимагає перевизначення двох методів: onCreate (SQLiteDatabase) і onUpgrade (SQLiteDatabase, int, int). У методі onCreate (...) ми встановлюємо схему створюваної бази даних, а в методі onUpgrade (...) можна розмістити код переходу між версіями схеми бази даних.

Також на практиці часто реалізується спрощений конструктор, який заповнює значення деяких аргументів, необхідних версії з суперкласу. У нашому прикладі передається постійне ім'я файлу бази даних, null замість необов'язкового об'єкта CursorFactory і постійний цілочисельний номер версії.

Хоча в прикладі RunTracker це і не потрібно, SQLiteOpenHelper підтримує можливість управління різними версіями схеми бази даних. Передбачається, що номери версій є цілі числа, починаючи з 1. В реальному додатку при кожній зміні схеми бази даних розробник збільшує константу версії і включає в метод onUpgrade (...) код управління змінами схеми або даних, необхідними між версіями.

В нашій реалізації методу onCreate (...) з щойно створеної базою даних виконуються дві інструкції SQL CREATE TABLE. Також реалізований метод insertRun (Run), який вставляє новий рядок в таблицю run і повертає її ідентифікатор. Запис серії містить всього одне поле даних - початкову дату; її значення в форматі long зберігається в базі даних з використанням об'єкта ContentValues, що представляє відповідність між іменами стовпців і значеннями.

Клас SQLiteOpenHelper містить два методи, що надають доступ до примірника SQLiteDatabase: getWritableDatabase () і getReadableDatabase (). Якщо потрібна база даних з

можливістю запису, використовується метод getWritableDatabase (), а якщо достатньо доступу тільки для читання - getReadableDatabase (). На практиці реалізації цих методів повертають один примірник SQLiteDatabase для заданого примірника SQLiteOpenHelper, але в деяких рідкісних ситуаціях (наприклад, при заповненні диска) можна отримати базу даних, доступну для читання, в ситуації, в якій неможливо отримати базу даних з можливістю запису.

Щоб забезпечити можливість запиту однієї або декількох серій з бази даних і розрізняти їх в додатку, в клас Run необхідно додати властивість-ідентифікатор. Внесіть зміни, представлені в лістингу 2.

Лістинг 2. Включення ідентифікатора в клас Run

```
public class Run {
    private long mId;
    private Date mStartDate;

    public Run () {
        mId = -1;
        mStartDate = new Date ();
    }

    public long getId () {
        return mId;
    }

    public void setId (long id) {
        mId = id;
    }

    public Date getStartDate () {
        return mStartDate;
}
```

Потім необхідно доопрацювати клас RunManager, щоб він використовував нову базу даних. Так буде створений API, який буде використовуватися іншим кодом програми для зберігання і вибірки даних. Для початку обмежимося мінімальним кодом, що забезпечує збереження об'єктів Run.

Лістинг 3. Управління поточною серією (RunManager.java)

```
public class RunManager {
    private static final String TAG = "RunManager";

    private static final String PREFS_FILE = "runs";
    private static final String PREF_CURRENT_RUN_ID =
    "RunManager.currentRunId";

    public static final String ACTION_LOCATION =
        "Com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;
    private RunDatabaseHelper mHelper;
    private SharedPreferences mPrefs;
    private long mCurrentRunId;

    private RunManager (Context appContext) {
        mAppContext = appContext;
        mLocationManager = (LocationManager) mAppContext
        .getSystemService (Context.LOCATION_SERVICE);
```

```
        mHelper = new RunDatabaseHelper (mAppContext);
        mPrefs = mAppContext.getSharedPreferences (PREFS_FILE,
        Context.MODE_PRIVATE);
        mCurrentRunId = mPrefs.getLong (PREF_CURRENT_RUN_ID, -1);
    }

    ...

    private void broadcastLocation (Location location) {
        Intent broadcast = new Intent (ACTION_LOCATION);
        broadcast.putExtra (LocationManager.KEY_LOCATION_CHANGED, location);
        mAppContext.sendBroadcast (broadcast);
    }

    public Run startNewRun () {
        // Вставка об'єкта Run в базу даних
        Run run = insertRun ();
        // Запуск відстеження серії
        startTrackingRun (run);
        return run;
    }

    public void startTrackingRun (Run run) {
        // Отримання ідентифікатора
        mCurrentRunId = run.getId ();
        // Збереження його в загальних налаштуваннях
        mPrefs.edit (). putLong (PREF_CURRENT_RUN_ID, mCurrentRunId) .commit ();
        // Почати оновлення даних місцеположення
        startLocationUpdates ();
    }

    public void stopRun () {
        stopLocationUpdates ();
        mCurrentRunId = -1;
        mPrefs.edit (). remove (PREF_CURRENT_RUN_ID) .commit ();
    }

    private Run insertRun () {
        Run run = new Run ();
        run.setId (mHelper.insertRun (run));
        return run;
    }
}
```

Метод startNewRun () викликає insertRun () для створення і вставки в базу даних нового об'єкта Run, передає його при виклику startTrackingRun (Run) для початку відстеження і, нарешті, повертає його стороні, що викликає. Цей метод буде використовуватися класом RunFragment у відповідь на натискання кнопки Start при відсутності поточної серії.

RunFragment також буде безпосередньо викликати startTrackingRun (Run) при перезапуску відстеження для поточної серії. Цей метод зберігає ідентифікатор переданого йому об'єкта Run в змінній примірника і в загальних налаштуваннях. При такому способі збереження дані можна буде отримати пізніше навіть при повному знищенні додатку; в цьому випадку необхідну роботу виконає конструктор RunManager.

Метод stopRun () зупиняє поновлення і стирає ідентифікатор поточної серії. RunFragment використовує цей метод в реалізації кнопки Stop.

Зараз буде доречно використовувати нові методи RunManager. Внесіть зміни, представлені в лістингу 4.

Лістинг 4. Оновлення коду запуску і зупинки (RunFragment.java)

```
@Override
public View onCreateView (LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
```

```
    View view = inflater.inflate (R.layout.fragment_run, container, false);
    ...

    mStartButton = (Button) view.findViewById (R.id.run_startButton);
    mStartButton.setOnClickListener (new View.OnClickListener () {

    @Override
    public void onClick (View v) {
        mRunManager.startLocationUpdates ();
        mRun = new Run ();
        mRun = mRunManager.startNewRun ();
        updateUI ();
    }
    });

    mStopButton = (Button) view.findViewById (R.id.run_stopButton);
    mStopButton.setOnClickListener (new View.OnClickListener () {
        @Override
        public void onClick (View v) {
            mRunManager.stopLocationUpdates ();
            mRunManager.stopRun ();
            updateUI ();
        }
    });

    updateUI ();
    return view;
}
```

Потім нам знадобиться реалізувати вставку об'єктів Location в базу даних у відповідь на оновлення від LocationManager. За аналогією зі вставкою об'єктів Run, ми додамо в RunDatabaseHelper і RunManager методи для вставки позиції в поточну серію. Однак на відміну від Run клас RunTracker повинен бути здатний вставляти дані позицій у міру надходження оновлень незалежно від того, чи відображається для користувача інтерфейс і працює додаток. Для реалізації цієї вимоги найкраще використовувати автономний об'єкт BroadcastReceiver.

Почнемо з додавання методу insertLocation (long, Location) в RunDatabaseHelper.

Лістинг 5. Вставка позицій в базу даних (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    private static final String TABLE_LOCATION = "location";
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";

    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";
    private static final String COLUMN_LOCATION_PROVIDER = "provider";
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";

    ...

    public long insertLocation (long runId, Location location) {
        ContentValues cv = new ContentValues ();
        cv.put (COLUMN_LOCATION_LATITUDE, location.getLatitude ());
        cv.put (COLUMN_LOCATION_LONGITUDE, location.getLongitude ());
        cv.put (COLUMN_LOCATION_ALTITUDE, location.getAltitude ());
        cv.put (COLUMN_LOCATION_TIMESTAMP, location.getTime ());
        cv.put (COLUMN_LOCATION_PROVIDER, location.getProvider ());
```

```
            cv.put (COLUMN_LOCATION_RUN_ID, runId);
            return getWritableDatabase (). insert (TABLE_LOCATION, null, cv);
        }
```

Тепер додайте в RunManager код вставки позиції для поточної серії.

Лістинг 6. Вставка позиції для поточної серії (RunManager.java)

```
    private Run insertRun () {
        Run run = new Run ();
        run.setId (mHelper.insertRun (run));
        return run;
    }

    public void insertLocation (Location loc) {
    if (mCurrentRunId! = -1) {
        mHelper.insertLocation (mCurrentRunId, loc);
    } Else {
        Log.e (TAG, "Location received with no tracking run; ignoring.");
    }
}
}
```

Ми повинні знайти підходяще місце для виклику нового методу insertLocation (Location). Використання для цієї мети автономного об'єкта BroadcastReceiver гарантує, що інтент позицій будуть оброблені незалежно від працездатності інших компонентів додатка RunTracker. Для цього необхідно створити спеціалізований субклас LocationReceiver з ім'ям TrackingLocationReceiver і зареєструвати його з використанням фільтра інтенту в маніфесті.

Створіть TrackingLocationReceiver як клас верхнього рівня.

Лістинг 7. Клас TrackingLocationReceiver: (TrackingLocationReceiver.java)

```
public class TrackingLocationReceiver extends LocationReceiver {
    @Override
    protected void onLocationReceived (Context c, Location loc) {
        RunManager.get (c) .insertLocation (loc);
    }
}
```

Зареєструйте його в маніфесті, щоб його код виконувався у відповідь на спеціалізовану дію ACTION_LOCATION.

Лістинг 8. Включення TrackingLocationReceiver (AndroidManifest.xml)

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

    ...

    <Receiver android: name = ". LocationReceiver"
    <Receiver android: name = ". TrackingLocationReceiver"
        android: exported = "false">
        <Intent-filter>
            <Action android: name = "com.bignerdranch.android.runtracker.
            ACTION_LOCATION" />
        </ Intent-filter>
    </ Receiver>
</ Application>
```

Після всіх змін запустіть додаток. Додаток RunTracker тепер вміє відслідковувати серії до тих пір, поки ви не накажете йому зупинитися, навіть в разі знищення або завершення програми. Щоб переконатися в тому, що все працює, як задумано, додайте команду реєстрації в «успішну» гілку методу onLocationReceived (...) класу TrackingLocationReceiver. Запустіть серію, потім знищіть або завершите додаток, продовжуючи спостерігати за даними LogCat.

Запит списку серій з бази даних

Додаток RunTracker тепер може зберігати в базі даних нові серії і їх позиції, але у своїй поточній версії RunFragment створює нову серію при кожному натисканні кнопки Start. Додамо нову активність і фрагмент для виведення списку серій. З їх допомогою користувач зможе створювати нові і переглядати існуючі серії. Цей інтерфейс нагадує той, який ми реалізували в CrimIntent для списку, але тільки дані, на основі яких будується список, беруться з бази даних, а не з пам'яті RunTracker сховища файлової системи.

Запит до SQLiteDatabase повертає об'єкт курсора Cursor, що описує результат. API курсорів досить простий і гнучкий, щоб підтримувати будь-які види різновидів результатів від будь-якого запиту. Курсори розглядають свої результати як сукупність рядків і стовпців, а в значеннях можуть використовуватися тільки примітивні типи і String.

Java-програмісти звикли використовувати для інкапсуляції моделі даних об'єкти - наприклад, Run і Location. Оскільки у нас вже є готові таблиці бази даних, що представляють об'єкти, було б ідеально, якби ми могли отримувати від Cursor екземпляри цих об'єктів.

Cursor є вбудований субклас з ім'ям CursorWrapper, який інкапсулює існуючий екземпляр Cursor і передає йому всі виклики методів. Сам по собі він не дуже корисний, але при використанні в якості суперкласу надає хорошу основу для побудови власних реалізацій курсорів для об'єктів моделі.

Обновіть клас RunDatabaseHelper і включіть в нього новий метод queryRuns (), який повертає об'єкт RunCursor зі списком всіх серій, впорядкованих за датою.

Лістинг 9. Запит списку серій (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_ID = "_id";
    private static final String COLUMN_RUN_START_DATE = "start_date";
    ...

    public RunCursor queryRuns () {
        // Еквівалент "select * from run order by start_date asc"
        Cursor wrapped = getReadableDatabase (). Query (TABLE_RUN,
        null, null, null, null, null, COLUMN_RUN_START_DATE + "asc");
        return new RunCursor (wrapped);
    }
    / **
    * Допоміжний клас з курсором, що повертає рядки таблиці "run".
    * Метод {@link getRun ()} повертає екземпляр Run, що представляє
    * Поточний рядок.
    * /
    public static class RunCursor extends CursorWrapper {
        public RunCursor (Cursor c) {
        super (c);
        }
        / **
        * Повертає об'єкт Run, що представляє поточний рядок,
        * Або null, якщо поточний рядок недійсний.
        * /
        public Run getRun () {
            if (isBeforeFirst () || isAfterLast ())
```

```
            return null;
            Run run = new Run ();
            long runId = getLong (getColumnIndex (COLUMN_RUN_ID));
            run.setId (runId);
            long startDate = getLong (getColumnIndex (COLUMN_RUN_START_DATE));
            run.setStartDate (new Date (startDate));
            return run;
        }
    }
}
```

Клас RunCursor визначає всього два методи: простий конструктор і getRun (). Метод getRun () перевіряє, що курсор знаходиться в своїх кордонах, а потім створює і налаштовує екземпляр Run за значеннями в стовпчиках поточного рядка. Користувач RunCursor перебирає рядки отриманого набору і викликає getRun () для кожного рядка, щоб отримати зручний об'єкт замість купи незручних примітивів.

Відповідно головною метою RunCursor є інкапсуляція рутинної роботи по перетворенню рядки таблиці run в екземпляр Run, з усіма необхідними переміщеннями і перетвореннями даних.

Метод queryRuns () виконує всю роботу з виконання запиту SQL і отримання нового примірника RunCursor, який повертається стороні, що викликає. Тепер цей новий метод можна буде використовувати в RunManager, а потім і в RunListFragment.

Лістинг 10. Опосередковане виконання запитів даних серій (RunManager.java)

```
private Run insertRun () {
    Run run = new Run ();
    run.setId (mHelper.insertRun (run));
    return run;
}
public RunCursor queryRuns () {
    return mHelper.queryRuns ();
}
public void insertLocation (Location loc) {
    if (mCurrentRunId! = -1) {
      mHelper.insertLocation (mCurrentRunId, loc);
    } Else {
      Log.e (TAG, "Location received with no tracking run; ignoring.");
    }
}
```

Виведення списку серій з використанням CursorAdapter

Щоб закласти основу для призначеного для користувача інтерфейсу списку серій, створіть нову активність RunListActivity і призначте її активністю за замовчуванням в маніфесті.

Лістинг 11. Клас RunListActivity (RunListActivity.java)

```
public class RunListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment () {
        return new RunListFragment ();
    }
}
<application
    android: allowBackup = "true"
    android: icon = "@ drawable / ic_launcher"
    android: label = "@ string / app_name"
    android: theme = "@ style / AppTheme">
    <Activity android: name = ". RunActivity"
    <Activity android: name = ". RunListActivity"
        android: label = "@ string / app_name">
        <Intent-filter>
            <Action android: name = "android.intent.action.MAIN" />
```

```
            <Category android: name = "android.intent.category.LAUNCHER" />
        </ Intent-filter>
    </ Activity>
    <Activity android: name = ". RunActivity"
              android: label = "@ string / app_name" />
    <Receiver android: name = ". TrackingLocationReceiver"
        android: exported = "false">
        <Intent-filter>
```

Тепер можна переходити до створення заготовки RunListFragment. Поки ми завантажуємо курсор в onCreate (Bundle) і закриваємо його в onDestroy (), але використовувати таку схему не рекомендується, тому що з нею запит до бази даних виконується в головному потоці (UI-потоці), а в деяких випадках це може привести до видачі повідомлення ANR (Application Not Responding).

Лістинг 13. Вихідна версія RunListFragment (RunListFragment.java)

```
public class RunListFragment extends ListFragment {
    private RunCursor mCursor;

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        // Запит на отримання списку серій
        mCursor = RunManager.get (getActivity ()). queryRuns ();
    }

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

Від класу RunCursor не буде ніякої користі без механізму передачі даних віджету ListView, пов'язаного з RunListFragment. В Android API (і в бібліотеку підтримки) входить клас CursorAdapter, який робить саме те, що потрібно. Вам залишається лише субкласувати його і надати реалізації пари методів. Клас CursorAdapter бере на себе логіку створення та повторного використання представлень.

Включіть в клас RunListFragment реалізацію RunCursorAdapter.

Лістинг 14. Реалізація RunCursorAdapter (RunListFragment.java)

```
public class RunListFragment extends ListFragment {

  private RunCursor mCursor;

  @Override
  public void onCreate (Bundle savedInstanceState) {
      super.onCreate (savedInstanceState);
      // Запит на отримання списку серій
      mCursor = RunManager.get (getActivity ()). queryRuns ();
      // Створення адаптера, що посилається на цей курсор
      RunCursorAdapter adapter = new RunCursorAdapter (getActivity (),
      mCursor);
      setListAdapter (adapter);
  }

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

```
    private static class RunCursorAdapter extends CursorAdapter {

        private RunCursor mRunCursor;

        public RunCursorAdapter (Context context, RunCursor cursor) {
            super (context, cursor, 0);
            mRunCursor = cursor;
        }

        @Override
        public View newView (Context context, Cursor cursor, ViewGroup parent) {
            // Використання заповнювача макета для отримання
            // представлення рядку
            LayoutInflater inflater = (LayoutInflater) context
                .getSystemService (Context.LAYOUT_INFLATER_SERVICE);
            return inflater
                .inflate (android.R.layout.simple_list_item_1, parent, false);
        }

        @Override
        public void bindView (View view, Context context, Cursor cursor) {
            // Отримання серії для поточного рядка
            Run run = mRunCursor.getRun ();

            // Створення текстового представлення початковоі дати
            TextView startDateTextView = (TextView) view;
            String cellText =
                context.getString (R.string.cell_text, run.getStartDate ());
            startDateTextView.setText (cellText);
        }
    }
}
```

Конструктор CursorAdapter отримує Context, Cursor і цілочисельний набір прапорів. Багато прапорів зараз вважаються застарілими, або замість них рекомендується використовувати завантажувачі, тому в цьому аргументі передається нуль. Ми також зберігаємо об'єкт RunCursor в змінній примірника, щоб уникнути його подальшого перетворення.

Потім ми реалізуємо метод newView (Context, Cursor, ViewGroup), який повертає об'єкт View для поточного рядка курсору. У нашому прикладі використовується заповнення системного ресурсу android.R.layout.simple_list_item_1, який представляє собою простий віджет TextView. Оскільки всі представлення в списку будуть виглядати однаково, це вся логіка, яка нам зараз знадобиться.

Метод bindView (View, Context, Cursor) буде викликатися кодом CursorAdapter, коли останньому потрібно налаштувати представлення для зберігання даних рядка в курсорі. При виклику завжди передається об'єкт View, повернутий раніше з newView (...).

Реалізація bindView (...) відносно проста. Спочатку ми запитуємо у RunCursor об'єкт Run для поточного рядка (курсор вже буде позиціонується CursorAdapter). Потім ми припускаємо, що передане представлення є Text-View, і налаштовуємо його для виведення простого опису Run.

З цими змінами запустіть додаток RunTracker; на екрані виводиться список раніше створених серій (передбачається, що ви запустили додаток і почали нову серію після реалізації коду вставки в базу даних).

Створення нових серій

Інтерфейс для створення нових серій легко реалізується на панелі дій, як це робилося в додатку CrimIntent. Почніть зі створення ресурсу меню.

Лістинг 15. Меню списку серій (run_list_options.xml)

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

```
    <Item android: id = "@ + id / menu_item_new_run"
        android: showAsAction = "always"
        android: icon = "@ android: drawable / ic_menu_add"
        android: title = "@ string / new_run" />
</ Menu>
```

У меню міститься посилання на рядок; додамо його в файл strings.xml.

Лістинг 16. Додавання рядка New Run (strings.xml)

```
  <String name = "stop"> Stop </ string>
  <String name = "gps_enabled"> GPS Enabled </ string>
  <String name = "gps_disabled"> GPS Disabled </ string>
  <String name = "cell_text"> Run at% 1 $ s </ string>
     <String name = "new_run"> New Run </ string>
</ Resources>
```

Додайте в RunListFragment код створення командного меню і обробки вибору команди, наведений в наступному лістингу.

Лістинг 17. Нові серії в командному меню (RunListFragment.java)

```
public class RunListFragment extends ListFragment {
  private static final int REQUEST_NEW_RUN = 0;

  private RunCursor mCursor;

  @Override
  public void onCreate (Bundle savedInstanceState) {
    super.onCreate (savedInstanceState);
    setHasOptionsMenu (true);
    // Запит на отримання списку серій
    mCursor = RunManager.get (getActivity ()). queryRuns ();
    // Створення адаптера, що посилається на цей курсор
    RunCursorAdapter adapter = new RunCursorAdapter (getActivity (), mCursor);
    setListAdapter (adapter);
  }

  ...

  @Override
  public void onCreateOptionsMenu (Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu (menu, inflater);
    inflater.inflate (R.menu.run_list_options, menu);
  }

  @Override
  public boolean onOptionsItemSelected (MenuItem item) {
    switch (item.getItemId ()) {
    case R.id.menu_item_new_run:
        Intent i = new Intent (getActivity (), RunActivity.class);
        startActivityForResult (i, REQUEST_NEW_RUN);
        return true;
    default:
        return super.onOptionsItemSelected (item);
    }
  }
  @Override
  public void onActivityResult (int requestCode, int resultCode, Intent data)
  {
    if (REQUEST_NEW_RUN == requestCode) {
        mCursor.requery ();
        ((RunCursorAdapter) getListAdapter ()). NotifyDataSetChanged ();
    }
```

```
    }

    private static class RunCursorAdapter extends CursorAdapter {
      private RunCursor mRunCursor;
```

Використання onActivityResult (...) для примусового перезавантаження списку після того, як користувач повернеться до нього в процесі навігації.

Робота з існуючими серіями

Наступний логічний крок - перехід від списку серій до докладної інформації про конкретну серії. Щоб ця можливість заробила, RunFragment знадобиться передавати ідентифікатор серії в аргументі. Оскільки хостом RunFragement є активність RunActivity, їй теж знадобиться додаток для ідентифікатора серії.

Почнемо з додавання аргументу в RunFragment і методу newInstance (long), який спрощує його використання.

Лістинг 18. Додавання аргументу для ідентифікатора серії (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    ...
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    public static RunFragment newInstance (long runId) {
        Bundle args = new Bundle ();
        args.putLong (ARG_RUN_ID, runId);
        RunFragment rf = new RunFragment ();
        rf.setArguments (args);
        return rf;
    }

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
```

Тепер використовуємо нові можливості фрагмента в RunActivity. Якщо інтент містить доповнення RUN_ID, то для створення RunFragment використовується метод newInstance (long). В іншому випадку просто використовується конструктор за замовчуванням, як і раніше.

Лістинг 19. Додавання доповнення для ідентифікатора серії (RunActivity.java)

```
public class RunActivity extends SingleFragmentActivity {
    / ** Ключ для передачі ідентифікатора серії в форматі long * /
    public static final String EXTRA_RUN_ID =
        "Com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment () {
        return new RunFragment ();
        long runId = getIntent (). getLongExtra (EXTRA_RUN_ID, -1);
        if (runId! = -1) {
             return RunFragment.newInstance (runId);
        } Else {
             return new RunFragment ();
        }
    }
}
```

Запрограмувати реакцію на виділення елемента списку в RunListFragment запуском RunActivity з ідентифікатором обраної серії.

Лістинг 20. Відкриття існуючих серій через onListItemClick (...)

```
@Override
public void onListItemClick (ListView l, View v, int position, long id) {
    // Аргумент id містить ідентифікатор серії;
    // CursorAdapter автоматично надає цю інформацію.
    Intent i = new Intent (getActivity (), RunActivity.class);
    i.putExtra (RunActivity.EXTRA_RUN_ID, id);
    startActivity (i);
}
private static class RunCursorAdapter extends CursorAdapter {
```

Так як ми присвоїли одну ідентифікатора в таблиці run ім'я _id, клас CursorAdapter виявив цей факт і передав його в аргументі id методу onListItemClick (...). Відповідно ми можемо передати його безпосередньо RunActivity в доповненні.

Простого запуску RunFragment з аргументом-ідентифікатором недостатньо для виведення корисної інформації про існуючу серії. Необхідно запросити в базі даних детальну інформацію, включаючи останнє записанє місце, для заповнення призначеного для користувача інтерфейсу.

Почати варто з класу RunDatabaseHelper, в якому буде створено новий метод queryRun (long), який повертає об'єкт RunCursor для однієї серії по заданому ідентифікатору.

Лістинг 21. Запит однієї серії (RunDatabaseHelper.java)

```
public RunCursor queryRun (long id) {
    Cursor wrapped = getReadableDatabase (). Query (TABLE_RUN,
        null, // Всі стовпці
        COLUMN_RUN_ID + "=?", // Пошук за ідентифікатором серії
        new String [] {String.valueOf (id)}, // З цим значенням
        null, // group by
        null, // order by
        null, // having
        "1"); // 1 рядок
    return new RunCursor (wrapped);
}
```

Сенс численних аргументів методу query (...) стає зрозумілий. Ми вибираємо всі стовпці з TABLE_RUN і фільтруємо їх за стовпцем ідентифікатора, який передається в аргументі умови WHERE з використанням масиву рядків з одним елементом. Запит обмежується поверненням одного рядка, результат упаковується в RunCursor і повертається.

Потім в клас RunManager додається метод getRun (long), який упаковує результат щойно створеного методу queryRun (long) і витягує об'єкт Run з першого рядка (якщо він є).

Лістинг 22. Реалізація getRun (long) (RunManager.java)

```
public Run getRun (long id) {
   Run run = null;
   RunCursor cursor = mHelper.queryRun (id);
   cursor.moveToFirst ();
   // Якщо рядок присутній, отримати об'єкт серії
   if (! cursor.isAfterLast ())
       run = cursor.getRun ();
   cursor.close ();
   return run;
}
```

Цей метод намагається витягти Run з першого рядка набору RunCursor, отриманого при виклику queryRun (long). Спочатку він наказує RunCursor перейти до першого рядку результату.

Якщо набір містить хоча б один рядок, isAfterLast () поверне false, і ми можемо спокійно запитати об'єкт Run для цього рядка.

Так як для сторони, що викликає, новий метод об'єкт RunCursor недоступний, не забудьте викликати close () перед поверненням, щоб база даних могла якомога швидше звільнити ресурси курсора в пам'яті.

Ця частина роботи завершена, тепер ми можемо оновити RunFragment для роботи з існуючими серіями.

Внесіть зміни, представлені в лістингу 23.

Лістинг 23. Робота з існуючими серіями (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    private BroadcastReceiver mLocationReceiver = new LocationReceiver () {

        @Override
        protected void onLocationReceived (Context context, Location loc) {
            if (! mRunManager.isTrackingRun (mRun))
            return;
               mLastLocation = loc;
            if (isVisible ())
               updateUI ();
        }
        @Override
        protected void onProviderEnabledChanged (boolean enabled) {
            int toastText = enabled? R.string.gps_enabled:
            R.string.gps_disabled;
            Toast.makeText (getActivity (), toastText, Toast.LENGTH_LONG) .show
            ();
        }
    };
    ...

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setRetainInstance (true);
        mRunManager = RunManager.get (getActivity ()); <

        // Перевірити ідентифікатор Run і отримати об'єкт серії
        Bundle args = getArguments ();
        if (args! = null) {
            long runId = args.getLong (ARG_RUN_ID, -1);
            if (runId! = -1) {
                mRun = mRunManager.getRun (runId);
            }
        }
    }

    @Override
    public View onCreateView (LayoutInflater inflater, ViewGroup container,
             Bundle savedInstanceState) {
        View view = inflater.inflate (R.layout.fragment_run, container, false);
        ...
        mStartButton = (Button) view.findViewById (R.id.run_startButton);
        mStartButton.setOnClickListener (new View.OnClickListener () {
            @Override
            public void onClick (View v) {
                mRun = mRunManager.startNewRun ();
                if (mRun == null) {
                    mRun = mRunManager.startNewRun ();
                } Else {
```

```
                    mRunManager.startTrackingRun (mRun);
                }
                updateUI ();
            }
        });
        ...
        return view;
    }

    ...

    private void updateUI () {
        boolean started = mRunManager.isTrackingRun ();
        boolean trackingThisRun = mRunManager.isTrackingRun (mRun);

        if (mRun! = null)
            mStartedTextView.setText (mRun.getStartDate (). toString ());

        int durationSeconds = 0;
        if (mRun! = null && mLastLocation! = null) {
            durationSeconds = mRun.getDurationSeconds (mLastLocation.getTime
        ());
            mLatitudeTextView.setText (Double.toString (mLastLocation.
            getLatitude ()));
            mLongitudeTextView.setText (Double.toString (mLastLocation.
               getLongitude ()))
            mAltitudeTextView.setText (Double.toString (mLastLocation.
               getAltitude ()));
        }

        mDurationTextView.setText (Run.formatDuration (durationSeconds));
        mStartButton.setEnabled (! started);
        mStopButton.setEnabled (started);
        mStopButton.setEnabled (started && trackingThisRun);
    }
}
```

Залишається надати ще одну послугу для користувача програми: змусити екземпляр RunFragment завантажувати останнє місцезнаходження поточної серії з бази даних. Реалізація буде дуже схожа на завантаження Run, але зі створенням нового об'єкта LocationCursor для роботи з об'єктами Location.

Почнемо з включення методу для запиту останнього місця розташування серії і внутрішнього класу LocationCursor в RunDatabaseHelper.

Лістинг 24. Запит останнього місця розташування для серії (RunDatabaseHelper.java)

```
public LocationCursor queryLastLocationForRun (long runId) {
    Cursor wrapped = getReadableDatabase (). Query (TABLE_LOCATION,
        null, // Всі стовпці
        COLUMN_LOCATION_RUN_ID + "=?", // Обмежити заданою серією
        new String [] {String.valueOf (runId)},
        null, // group by
        null, // having
        COLUMN_LOCATION_TIMESTAMP + "desc", // Спочатку найновіші
        "1"); // limit 1
    return new LocationCursor (wrapped);
}

// ... Після RunCursor ...

public static class LocationCursor extends CursorWrapper {
    public LocationCursor (Cursor c) {
       super (c);
    }
```

```
        public Location getLocation () {
        if (isBeforeFirst () || isAfterLast ())
          return null;

        // Спочатку отримуємо постачальника для використання конструктора
        String provider = getString (getColumnIndex (COLUMN_LOCATION_PROVIDER));
        Location loc = new Location (provider);

        // Заповнення інших властивостей
            loc.setLongitude (getDouble (getColumnIndex
            (COLUMN_LOCATION_LONGITUDE)));
            loc.setLatitude (getDouble (getColumnIndex (COLUMN_LOCATION_LATITUDE)));
            loc.setAltitude (getDouble (getColumnIndex (COLUMN_LOCATION_ALTITUDE)));
            loc.setTime (getLong (getColumnIndex (COLUMN_LOCATION_TIMESTAMP)));
        return loc;
        }
    }
```

Об'єкт LocationCursor виконує ті ж функції, що і RunCursor, але в ньому упаковується курсор для повернення рядків таблиці location, а їх різні поля перетворюються в властивості об'єкта Location. У цій реалізації є одна тонкість: конструктору Location слід вказати ім'я постачальника даних, тому ми здобуваємо його з поточного рядка, перш ніж ставити інші властивості.

Метод queryLastLocationForRun (long) дуже схожий на queryRun (long), за винятком того, що він шукає останнє розташування для заданої серії і упаковує результат в LocationCursor.

Як і у випадку з queryRun (long), ми повинні створити в RunManager метод для його виклику і повернути Location з єдиного рядка курсору.

Лістинг 25. Отримання останнього місця розташування для серії (RunManager.java)

```
public Location getLastLocationForRun (long runId) {
    Location location = null;
    LocationCursor cursor = mHelper.queryLastLocationForRun (runId);
    cursor.moveToFirst ();
    // Якщо набір не порожній, отримати місце розташування
    if (! cursor.isAfterLast ())
      location = cursor.getLocation ();
    cursor.close ();
    return location;
}
```

Тепер новий метод RunFragment може використовуватися для вибірки останнього місця розташування поточної серії при створенні фрагмента.

Лістинг 26. Отримання останнього місця розташування для поточної серії (RunFragment.java)

```
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setRetainInstance (true);
        mRunManager = RunManager.get (getActivity ());

        // Перевірити ідентифікатор Run і отримати об'єкт серії
        Bundle args = getArguments ();
        if (args! = null) {
        long runId = args.getLong (ARG_RUN_ID, -1);
        if (runId! = -1) {
            mRun = mRunManager.getRun (runId);
            mLastLocation = mRunManager.getLastLocationForRun (runId);
        }
    }
}
```

В результаті наш об'єкт RunTracker здатний створювати і відстежувати стільки серій, скільки витримає диск (і батарея) вашого пристрою. Всі ці дані виводяться на екран в логічному, послідовному представленні.

Завдання 1. Виділення поточної серії

У поточній реалізації впізнати поточну серію можна тільки одним способом: вручну відкрити її в списку і перевірити стан кнопок запуску і зупинки. Було б зручніше, якби користувач міг швидше і простіше звернутися до поточної серії.

Забезпечте візуальне виділення рядка списку, відповідної поточної серії (наприклад, виведіть поруч з нею значок або змініть її колір).

Використовуйте безперервне оповіщення про включений режим відстеження, при натисканні на якому повинна запускатися активність RunActivity.

Лабораторна робота № 7

Тема: Робота з Maps API

Створити новий клас RunMapFragment, який буде виводити карту переміщень користувача та інтерактивні маркери, які позначають початок і кінець переміщення.

Необхідно налаштувати проект для використання Maps API.

Додавання Maps API в додаток RunTracker

Maps API надається пакетом Google Play services SDK і пред'являє ряд вимог як до середовища розробки, так і до додатка.

Тестування на реальному пристрої

Пакет Google Play services SDK (а отже, і Maps API) вимагає реального пристрою з Android версії не менше 2.2 зі встановленою підтримкою магазину Google Play. Запуск на емуляторі не підтримується.

Установка і використання Google Play services SDK

Щоб підтримка Maps API стала доступною в вашому проекті, спочатку необхідно встановити додаток Google Play services SDK і налаштувати проект бібліотеки для роботи з додатком.

1. В Android SDK Manager встановіть додаток Google Play services з розділу Extras. Доповнення буде встановлено в підкаталог extras / google / google_play_services каталогу Android SDK.
2. В Eclipse імпортуйте копію проекту бібліотеки в робочий простір командою File > Import ...> Existing Android Code Into Workspace. Проект бібліотеки знаходиться в каталозі доповнень Google Play services libproject / google-play-services_lib. Обов'язково виберіть режим копіювання Copy projects into workspace в майстра імпортування, щоб працювати з самостійної копією проекту.
3. Відкрийте вікно властивостей проекту RunTracker і додайте посилання на проект бібліотеки в категорії Android, Library. Клацніть на кнопці Add ... і виберіть проект google-play-services_lib.

Отримання ключа Google Maps API

Щоб використовувати Maps API, необхідно створити ключ API для вашого застосування.

Оновлення маніфесту RunTracker

Для роботи Google Play services і Maps API необхідно включити в маніфест додатка кілька дозволів і вимог (на додаток до щойно отриманому вами ключу API). Додайте виділену розмітку XML в маніфест RunTracker. Дозвіл з закінченням повинен MAPS_RECEIVE починатися з імені пакета RunTracker.

Додайте запис для майбутньої активності RunMapActivity.

Лістинг 1. Вимоги Maps API (AndroidManifest.xml)

```
<Manifest xmlns: android = "http://schemas.android.com/apk/res/android"
    package = "com.bignerdranch.android.runtracker"
    android: versionCode = "1"
    android: versionName = "1.0">

    <Uses-sdk android: minSdkVersion = "9" android: targetSdkVersion = "15" />
```

```
    <permission
        android: name =
        "com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE" android:
        protectionLevel = "signature" />
    <Uses-permission
        android: name =
        "com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE" />
    <Uses-permission android: name = "android.permission.INTERNET" />
    <Uses-permission android: name =
    "android.permission.WRITE_EXTERNAL_STORAGE" />
    <Uses-permission
        android: name =
        "com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <Uses-permission android: name =
    "android.permission.ACCESS_COARSE_LOCATION" />
    <Uses-permission android: name = "android.permission.ACCESS_FINE_LOCATION"
    />
    <Uses-feature android: required = "true"
        android: name = "android.hardware.location.gps" />
    <Uses-feature
        android: required = "true"
        android: glEsVersion = "0x00020000" />
    <application
    android: allowBackup = "true"
    android: icon = "@ drawable / ic_launcher"
    android: label = "@ string / app_name"
    android: theme = "@ style / AppTheme">
        <Activity android: name = ". RunListActivity"
        android: label = "@ string / app_name">
        <Intent-filter>
             <Action android: name = "android.intent.action.MAIN" />
             <Category android: name = "android.intent.category.LAUNCHER" />
        </ Intent-filter>
    </ Activity>

    <Activity android: name = ". RunActivity"
        android: label = "@ string / app_name" />
    <Activity android: name = ". RunMapActivity"
        android: label = "@ string / app_name" />
    <Receiver android: name = ". TrackingLocationReceiver"
        android: exported = "false">
        <Intent-filter>
            <action
            android: name =
            "com.bignerdranch.android.runtracker.ACTION_LOCATION" />
        </ Intent-filter>
    </ Receiver>
    <Meta-data
        android: name = "com.google.android.maps.v2.API_KEY"
        android: value = "your-maps-API-key-here" />
    </ Application>

</ Manifest>
```

Виведення місця розташування користувача на карті

Після виконання всіх вимог прийшов час скористатися плодами роботи і вивести карту. Maps API включає класи MapFragment і SupportMapFragment, які субкласуються для налаштування відображення MapView і пов'язаного з ним об'єкта моделі GoogleMap.

Створіть субклас SupportMapFragment з ім'ям RunMapFragment в новому файлі, код якого приведений в лістингу 2.

Лістинг 2. Базова реалізація RunMapFragment (RunMapFragment.java)

```
public class RunMapFragment extends SupportMapFragment {
    private static final String ARG_RUN_ID = "RUN_ID";
    private GoogleMap mGoogleMap;

    public static RunMapFragment newInstance (long runId) {
        Bundle args = new Bundle ();
        args.putLong (ARG_RUN_ID, runId);
        RunMapFragment rf = new RunMapFragment ();
        rf.setArguments (args);
        return rf;
    }

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

        // Збереження посилання на GoogleMap
        mGoogleMap = getMap ();
        // Виведення місця розташування користувача
        mGoogleMap.setMyLocationEnabled (true);

        return v;
    }
}
```

Метод newInstance (long) класу RunMapFragment отримує ідентифікатор серії і зберігає в аргументах новий екземпляр фрагмента, як це робилося в коді RunFragment.

Реалізація onCreateView (...) викликає реалізацію суперкласу для отримання повернення представлення, але її робота заснована на тому факті, що цей виклик ініціалізує екземпляр GoogleMap фрагмента. *GoogleMap* - об'єкт моделі, прив'язаний до MapView, який буде використовуватися для налаштування різних доповнень до карти. У початковій версії фрагмента просто викличте setMyLocationEnabled (boolean), щоб користувач міг бачити свою позицію на карті і переміщатися до неї.

Для розміщення фрагмента RunMapFragment створіть простий клас RunMapActivity використанням наведеного нижче коду. Посилання на цей клас вже була включене в маніфест.

Лістинг 3. Активність-хост для фрагмента (RunMapActivity.java)

```
public class RunMapActivity extends SingleFragmentActivity {
    / ** Ключ для передачі ідентифікатора серії в форматі long * /
    public static final String EXTRA_RUN_ID =
       "Com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment () {
       long runId = getIntent (). getLongExtra (EXTRA_RUN_ID, -1);
       if (runId! = -1) {
             return RunMapFragment.newInstance (runId);
       } Else {
             return new RunMapFragment ();
       }
    }
}
```

Тепер нам знадобиться код запуску RunMapActivity з RunFragment при доступності серії. Для цього в макет буде додана кнопка, яка буде викликати карту. Але спочатку, як зазвичай, включимо кілька рядків, які будуть використовуватися кнопкою, а також іншими компонентами користувальницького інтерфейсу.

Лістинг 4. Рядки для призначеного для користувача інтерфейсу (res / values / strings.xml)

```
    <String name = "new_run"> New Run </ string>
    <String name = "map"> Map </ string>
    <String name = "run_start"> Run Start </ string>
    <String name = "run_started_at_format"> Run started at% s </ string>
    <String name = "run_finish"> Run Finish </ string>
    <String name = "run_finished_at_format"> Run finished at% s </ string>
</ Resources>
```

Внесіть зміни в макет RunFragment і включіть в нього нову кнопку Map.

Лістинг 5. Додавання кнопки Map (fragment_run.xml)

```
<Button android: id = "@ + id / run_stopButton"
    android: layout_width = "0dp"
    android: layout_height = "wrap_content"
    android: layout_weight = "1"
    android: text = "@ string / stop"
/>
<Button android: id = "@ + id / run_mapButton"
    android: layout_width = "0dp"
    android: layout_height = "wrap_content"
    android: layout_weight = "1"
    android: text = "@ string / map"
/>
</ LinearLayout>
</ TableLayout>
```

Тепер включите в RunFragment код підтримки нової кнопки і управління її доступністю.

Лістинг 6. Підключення кнопки Map (RunFragment.java)

```
public class RunFragment extends Fragment {
    ...

    private RunManager mRunManager;

    private Run mRun;
    private Location mLastLocation;

    private Button mStartButton, mStopButton, mMapButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    ...

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

    ...

      mMapButton = (Button) view.findViewById (R.id.run_mapButton);
      mMapButton.setOnClickListener (new View.OnClickListener () {
          @Override
          public void onClick (View v) {
              Intent i = new Intent (getActivity (), RunMapActivity.class);
              i.putExtra (RunMapActivity.EXTRA_RUN_ID, mRun.getId ());
              startActivity (i);
          }
      });
```

```
        updateUI ();
        return view;

    }
    ...

    private void updateUI () {
        boolean started = mRunManager.isTrackingRun ();
        boolean trackingThisRun = mRunManager.isTrackingRun (mRun);

        if (mRun! = null)
            mStartedTextView.setText (mRun.getStartDate (). toString ());

        int durationSeconds = 0;
        if (mRun! = null && mLastLocation! = null) {
            durationSeconds = mRun.getDurationSeconds (mLastLocation.getTime
            ());
            mLatitudeTextView.setText (Double.toString
            (MLastLocation.getLatitude ()));
            mLongitudeTextView.setText (Double.toString
            (MLastLocation.getLongitude ()));
            mAltitudeTextView.setText (Double.toString
            (MLastLocation.getAltitude ()));
            mMapButton.setEnabled (true);
        } Else {
            mMapButton.setEnabled (false);
        }
        mDurationTextView.setText (Run.formatDuration (durationSeconds));
        mStartButton.setEnabled (! started);
        mStopButton.setEnabled (started && trackingThisRun);
}
```

У новій версії додаток RunTracker зможе вивести карту і ваше місце розташування на ній. Відкрийте програму, завантажте серію і натисніть кнопку Map.

Виведення маршруту

Наше наступне завдання - виведення лінії, яка зображує маршрут. Завдяки Maps API це завдання вирішується тривіально, але спочатку необхідно отримати список позицій для побудови маршруту. Включіть в класи RunDatabaseHelper і RunManager метод, що надає об'єкт LocationCursor з необхідними даними.

Лістинг 7. Запит позицій серії (RunDatabaseHelper.java)

```
public LocationCursor queryLocationsForRun (long runId) {
    Cursor wrapped = getReadableDatabase (). Query (TABLE_LOCATION,
        null,
        COLUMN_LOCATION_RUN_ID + "=?", // Відбір по заданій серії
        new String [] {String.valueOf (runId)},
        null, // group by
        null, // having
        COLUMN_LOCATION_TIMESTAMP + "asc"); // метод
    return new LocationCursor (wrapped); // по тимчасовій мітці
}
```

Метод queryLocationsForRun (long) дуже схожий на метод queryLastLocation ForRun (long) з голови, присвяченій SQLite, але він впорядковує позиції по зростанню, і повертає їх все.

Використовуючи цей метод в RunManager, можна створити зручний фасад (façade) для RunMapFragment.

Лістинг 8. Запит позицій серії, частина II (RunManager.java)

```
public LocationCursor queryLocationsForRun (long runId) {
      return mHelper.queryLocationsForRun (runId);
}
```

RunMapFragment може використовувати цей новий метод для завантаження позицій. Запит до бази даних слід винести з головного потоку за допомогою Loader. Створіть новий клас LocationListCursorLoader для вирішення цього завдання.

Лістинг 9. Завантажувач позицій (LocationListCursorLoader.java)

```
public class LocationListCursorLoader extends SQLiteCursorLoader {
    private long mRunId;

    public LocationListCursorLoader (Context c, long runId) {
        super (c);
        mRunId = runId;
    }

    @Override
    protected Cursor loadCursor () {
        return RunManager.get (getContext ()). queryLocationsForRun (mRunId);
    }
}
```

Використовуйте новий завантажувач в RunMapFragment для завантаження позицій.

Лістинг 10. Завантаження позицій в RunMapFragment (RunMapFragment.java)

```
public class RunMapFragment extends SupportMapFragment
    implements LoaderCallbacks <Cursor> {
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_LOCATIONS = 0;

    private GoogleMap mGoogleMap;
    private LocationCursor mLocationCursor;

    ...

    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);

        // Перевірка ідентифікатора серії в аргументі і пошук серії
        Bundle args = getArguments ();
        if (args! = null) {
            long runId = args.getLong (ARG_RUN_ID, -1);
            if (runId! = -1) {
                LoaderManager lm = getLoaderManager ();
                lm.initLoader (LOAD_LOCATIONS, args, this);
            }
        }
    }
    ...

    @Override
    public Loader <Cursor> onCreateLoader (int id, Bundle args) {
        long runId = args.getLong (ARG_RUN_ID, -1);
        return new LocationListCursorLoader (getActivity (), runId);
    }

    @Override
    public void onLoadFinished (Loader <Cursor> loader, Cursor cursor) {
```

```
        mLocationCursor = (LocationCursor) cursor;
    }
    @Override
    public void onLoaderReset (Loader <Cursor> loader) {
        // Завершення роботи з даними
        mLocationCursor.close ();
        mLocationCursor = null;
    }
}
```

Клас RunMapFragment зберігає об'єкт LocationCursor для списку позицій. Цей список буде використовуватися при побудові карти з маршрутом.

Реалізація onLoaderReset (Loader <Cursor>) закриває і обнуляє посилання на курсор, коли той стає недоступним. У звичайних обставинах цей метод буде викликатися при закритті завантажувача об'єктом LoaderManager при виході користувача з фрагмента, що містить карту. До того ж таке рішення не закриває курсор при виконанні повороту, а це саме те, що нам потрібно.

Виведення інформації на GoogleMap. Додайте метод updateUI () із заповненням даних відображається серії, і викличте цей метод в onLoadFinished (...).

Лістинг 11. Нанесення даних серії на карту (RunMapFragment.java)

```
private void updateUI () {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Створення накладки з позиціями серії.
    // Створюємо ламану з усіма точками.
    PolylineOptions line = new PolylineOptions ();
    // Також створюється об'єкт LatLngBounds для масштабування за розмірами.
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder ();
    // Перебір позицій
    mLocationCursor.moveToFirst ();
    while (! mLocationCursor.isAfterLast ()) {
        Location loc = mLocationCursor.getLocation ();
        LatLng latLng = new LatLng (loc.getLatitude (), loc.getLongitude ());
        line.add (latLng);
        latLngBuilder.include (latLng);
        mLocationCursor.moveToNext ();
    }
    // Додавання маршруту на карту
    mGoogleMap.addPolyline (line);
    // Масштабування карти по маршруту з додатковими відступами
    // Як би прямокутника вибирається
    // розмір поточного екрану.
    Display display = getActivity (). GetWindowManager (). GetDefaultDisplay
    ();
    // Побудова інструкції переміщення для камери карти.
    LatLngBounds latLngBounds = latLngBuilder.build ();
    CameraUpdate movement = CameraUpdateFactory.newLatLngBounds (latLngBounds,
        display.getWidth (), display.getHeight (), 15);
    mGoogleMap.moveCamera (movement);
    }

    @Override
    public Loader <Cursor> onCreateLoader (int id, Bundle args) {
        long runId = args.getLong (ARG_RUN_ID, -1);
        return new LocationListCursorLoader (getActivity (), runId);
    }
    @Override
    public void onLoadFinished (Loader <Cursor> loader, Cursor cursor) {
        mLocationCursor = (LocationCursor) cursor;
        updateUI ();
}
```

Новий метод використовує Maps API в декількох місцях. Спочатку він створює екземпляр PolylineOptions, який буде використовуватися для побудови маршруту, виведеного на екран, і екземпляр LatLngBounds.Builder для створення прямокутника для масштабування карти.

Потім він перебирає дані LocationCursor і для кожного об'єкта Location створює об'єкт LatLng з його координатами. Об'єкт LatLng включається в PolylineOptions, останній включається в LatLngBounds перед переходом до наступного рядка в курсорі.

Після перебору всіх позицій метод викликає addPolyline (PolylineOptions) для GoogleMap, додаючи маршрут на карту.

На наступному кроці слід масштабувати карту так, щоб на ній містилася вся лінія. Переміщення по карті відноситься до відповідальності «камери», а для настройки камери використовуються команди, упаковані в примірниках CameraUpdate і передані moveCamera (CameraUpdate). Створення екземпляра для переміщення камери по межах щойно доданої лінії виконується методом newLatLngBounds (LatLngBounds, int, int, int) класу CameraUpdateFactory.

Ми передаємо розміри поточного екрану як наближене значення розміру карти в пікселах, з додаванням кількох пікселів на відступи, щоб маршрут краще виглядав. Існує спрощена версія цього методу newLatLngBounds (LatLngBounds, int), але вона видає виключення IllegalStateException, якщо метод буде викликаний до того, як MapView завершить визначення своїх розмірів в процесі побудови макета. Так як ми не можемо гарантувати, що це відбудеться до моменту виклику updateUI (), доводиться використовувати приблизну оцінку.

Внесіть зміни та знову запустіть RunTracker - на карті з'являється чорна лінія, що позначає маршрут.

Додавання маркерів початку і кінця маршруту

Тепер, коли ми підготували все необхідне, додати маркери, які позначають початкову і кінцеву позиції маршруту, відносно нескладно. Ми також додамо текст, який буде виводитися в інформаційному вікні при дотику до маркера.

Додайте в метод updateUI () код, виділений жирним шрифтом.

Лістинг 12. Маркери початку і кінця маршруту з інформацією (RunMapFragment.java)

```
private void updateUI () {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Створення накладки з позиціями серії.
    // Створюємо ламану з усіма точками.
    PolylineOptions line = new PolylineOptions ();
    // Також створюється об'єкт LatLngBounds для масштабування за розмірами.
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder ();
    // Перебір позицій
    mLocationCursor.moveToFirst ();
    while (! mLocationCursor.isAfterLast ()) {
        Location loc = mLocationCursor.getLocation ();
        LatLng latLng = new LatLng (loc.getLatitude (), loc.getLongitude ());

        Resources r = getResources ();

        // Якщо це перша позиція, додати маркер
        if (mLocationCursor.isFirst ()) {
            String startDate = new Date (loc.getTime ()). ToString ();
            MarkerOptions startMarkerOptions = new MarkerOptions ()
            .position (latLng)
            .title (r.getString (R.string.run_start))
            .snippet (r.getString (R.string.run_started_at_format, startDate));
            mGoogleMap.addMarker (startMarkerOptions);
        } Else if (mLocationCursor.isLast ()) {
            // Якщо це остання позиція, яка не є
```

```
            // також першою, додати маркер
            String endDate = new Date (loc.getTime ()). ToString ();
            MarkerOptions finishMarkerOptions = new MarkerOptions ()
                .position (latLng)
                .title (r.getString (R.string.run_finish))
                .snippet (r.getString (R.string.run_finished_at_format,
                endDate));
            mGoogleMap.addMarker (finishMarkerOptions);
        }
        line.add (latLng);
        latLngBuilder.include (latLng);
        mLocationCursor.moveToNext ();
    }
    // Додавання маршруту на карту
    mGoogleMap.addPolyline (line);
```

Для першої і останньої точки маршруту код створює екземпляр MarkerOptions для зберігання позиції, заголовка і додаткового тексту. Тема і додатковий текст виводяться в простому інформаційному вікні, коли користувач торкається до маркера.

У цьому коді використовується маркер за замовчуванням, але при бажанні можна створити маркер іншого кольору (і навіть містить задане зображення) за допомогою методу icon (BitmapDescriptor) і BitmapDescriptorFactory. Існує багато стандартних кольорів, з яких ви можете вибрати потрібний.

Відкрийте програму RunTracker і протестуйте її.

Завдання 1. Оперативне оновлення

У поточній версії RunMapFragment може відображати карту з місцями розташування серії, «замороженими» за часом на момент прибуття користувача. Якісний геопозиційний додаток повинен виводити поновлення в режимі «живого» оновлення.

Використовуючи субклас LocationReceiver в RunMapFragment, організуйте обробку надходження нових позицій з перемальовуванням маршруту.

Видаліть попередню накладку (overlay) і маркери, перш ніж додавати нові.

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.