ПРАКТИЧНА РОБОТА №7

Тема роботи: Робота із контент-провайдерами.

Мета роботи: Отримати практичні навички створення власних контент-провайдерів.

Теоретичні відомості

Контент-провайдери надають інтерфейс для публікації та використання структурованих наборів даних, що базуються на URI з використанням простої схеми content: //. Їх використання дозволяє відокремити код додатків від даних, роблячи програму менш чутливою до змін в джерелах даних.

Для взаємодії з контент-провайдером використовується унікальний URI, який зазвичай формується таким чином:

content://<домен-розробника>.provider.<імя додатку>/<шлях до даних>

Класи, що реалізують контент-провайдери, найчастіше мають статичну строкову константу *CONTENT_URI*, яка використовується для звернення до даного контент-провайдеру.

Контент-провайдери є єдиним способом доступу до даних інших додатків і використовуються для отримання результатів запитів, оновлення, додавання і видалення даних. Якщо у програми є потрібні повноваження, вона може запитувати і модифікувати відповідні дані, що належать іншому додатку, у тому числі дані стандартних БД Android. У загальному випадку, контент-провайдери слід створювати тільки тоді, коли потрібно надати іншим програмам доступ до даних застосунку. В інших випадках рекомендується використовувати СУБД (SQLite). Тим не менш, іноді контент-провайдери використовуються всередині однієї програми для пошуку та оброблення специфічних запитів до даних.

Використання контент-провайдерів

Для доступу до даних якого-небудь контент-провайдера використовується об'єкт класу ContentResolver, який можна отримати за допомогою методу getContentResolver контексту програми для зв'язку з постачальником в якості клієнта:

```
ContentResolver cr = getApplicationContext (). GetContentResolver ();
```

Об'єкт ContentResolver взаємодіє з об'єктом контент-провайдера, відправляючи йому запити клієнта. Контент-провайдер обробляє запити і повертає результати обробки.

Контент-провайдери представляють свої дані у вигляді однієї або декількох таблиць подібно таблицями реляційних БД. Кожен рядок при цьому є окремим «об'єктом» з властивостями, зазначеними у відповідних іменованих полях. Як правило, кожен рядок має унікальний цілочисельний індекс із ім'ям «_id», який служить для однозначної ідентифікації необхідного об'єкта.

Контент-провайдери, зазвичай надають мінімум два URI для роботи з даними: один для запитів, що вимагають всі дані відразу, а інший - для звернення до конкретного «рядка». В останньому випадку в кінці URI додається / <номер-рядка> (який збігається з індексом «_id»).

Запити на отримання та зміну даних

Запити на отримання даних схожі на запити до БД, при цьому використовується метод query об'єкта ContentResolver. Відповідь також приходить у вигляді курсору, «націленого» на результуючий набір даних (вибрані рядки таблиці):

```
ContentResolver cr = getContentResolver();
// отримати дані всіх контактів
Cursor c = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,
        null, null);
// отримати всі рядки, де третє поле має конкретне
// значення і впорядкувати за п'ятим  полем
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows = cr.query(MyProvider.CONTENT_URI, null, where,
null, order);
```

Витягання даних-результатів запиту за допомогою курсору було розглянуто раніше.

Для зміни даних застосовуються методи insert, update і delete об'єкта ContentResolver. Для масової вставки також існує метод bulkInsert.

Приклад додавання даних:

```
ContentResolver cr = getContentResolver();
ContentValues newRow = new ContentValues();
// повторюємо для кожного поля в рядку:
newRow.put(COLUMN_NAME, newValue);
Uri myRowUri = cr.insert(SampleProvider.CONTENT_URI, newRow);
// масова вставка:
ContentValues[] valueArray = new ContentValues[5];
// тут заповнюємо масив
// робимо вставку
int count = cr.bulkInsert(MyProvider.CONTENT_URI, valueArray);
```

При вставці одного елемента метод *insert* повертає URI вставленого елемента, а при масовій вставці повертається кількість вставлених елементів.

Приклад видалення:

```
ContentResolver cr = getContentResolver();
// видалення конкретного рядка
cr.delete(myRowUri, null, null);
// видалення декількох рядків
String where = "_id < 5";
cr.delete(MyProvider.CONTENT_URI, where, null);
```

Приклад зміни:

```
ContentValues newValues = new ContentValues();
newValues.put(COLUMN_NAME, newValue);
String where = "_id < 5";
getContentResolver().update(MyProvider.CONTENT_URI, newValues,
where, null);
```

Створення контент-провайдерів

Для створення власного контент-провайдера потрібно розширити клас ContentProvider і перевизначити метод onCreate, щоб проініціалізувати джерело даних, яке потрібно опублікувати. Каркас для класу показаний нижче:

```
public class NewProvider extends ContentProvider {
public final static Uri CONTENT_URI=Uri.parse("URI провайдера");
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
        // видалення даних
        return 0;
}
@Override
public String getType(Uri uri) {
        // Повертає тип MIME для зазначених об'єктів (об'єкта)
        return null;
}
@Override
public Uri insert(Uri uri, ContentValues values) {
        // Додавання даних
        return null;
}
@Override
public boolean onCreate() {
        // Ініціалізація джерела даних
        return false;
}
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
        // Стандартний запит
        return null;
}
@Override
public int update(Uri uri, ContentValues values, String selection,
                String[] selectionArgs) {
        // Обновлення данних
```

```
        return 0;
    }
}
```

Традиційно URI повинні бути представлені двома способами, одним - для доступу до всіх значень певного типу, інший - для вказівки на конкретний екземпляр даних.

Наприклад, URI content: //com.android.provider.metropicker/stations міг би використовуватися для отримання списку всіх станцій (метро), а content: //com.android.provider.metropicker/stations/17 - для станції з індексом 17 .

При створенні контент-провайдера зазвичай застосовується статичний об'єкт класу UriMatcher, який служить для визначення деталей запиту до контент-провайдеру. Використання UriMatcher дозволяє «розвантажити» логіку програми і уникнути множинного порівняння строкових об'єктів за рахунок централізованого «відображення» URI різних видів на цілочисельні константи. Особливо він корисний у випадках, коли створюваний контент-провайдер обслуговує різні URI для доступу до одного і того ж джерела даних. Використовувати UriMatcher дуже просто: всередині класу контент-провайдера можна додати такий код:

```
// Константи для різних типів запитів
private static final int ALL_STATIONS = 1;
private static final int SINGLE_STATION = 2;
private static final UriMatcher uriMatcher;
// заповнення UriMatcher'a
// Якщо URI закінчується на / stations - це запит про всі станції
// Якщо на stations / [ID] - про конкретну станцію
static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("com.example.provider.metropicker", "stations",
            ALL_STATIONS);
    uriMatcher.addURI("com.example.provider.metropicker",
"stations/#",
            SINGLE_STATION);
}
```

У подальшому отримані в запиті до контент-провайдеру URI перевіряється в методах класу наступним чином:

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
    switch (uriMatcher.match(uri)) {
    case ALL_STATIONS:
        // Повернути курсор, який вказує на вибірку з  усіма
    станціями
    case SINGLE_STATION:
        // Витягнути ID станції з URI:
```

```
            String _id = uri.getPathSegments().get(1);
            // Повернути курсор, який вказує на вибірку з однією
        станцією.
        }
        return null;
    }
```

При наповненні об'єкта UriMatcher шаблонами можуть застосовувати в «#» і «*» в якості спеціальних символів: # в шаблоні збігається з будь-яким числом, а * - з будь-яким текстом.

Метод getType класу ContentProvider зазвичай повертає один тип даних для масової вибірки, а інший - для одиночних записів. У нашому випадку це могло б виглядати так:

```
    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
        case ALL_STATIONS:
            return "vnd.com.example.cursor.dir/station";
        case SINGLE_STATION:
            return "vnd.com.example.cursor.item/station";
        default:
            throw new IllegalArgumentException("Unsupported URI: " +
            uri);
        }
    }
```

Для того, щоб Android знав про існування і міг використовувати (через ContentResolver) ваш контент-провайдер, його потрібно описати в маніфесті додатка. Мінімальний опис виглядає так:

```
    <provider
    android:name=".NewProvider"
    android:authorities="com.android.provider.metropicker" >
    </provider>
```

В даному випадку зазначені тільки обов'язкові атрибути елемента <provider>: ім'я класу контент-провайдера і його область відповідальності. Більш повну інформацію про можливі атрибутах можна отримати на сайті developer.android.com:
http://developer.android.com/guide/topics/manifest/provider-element.html

Порядок виконання роботи

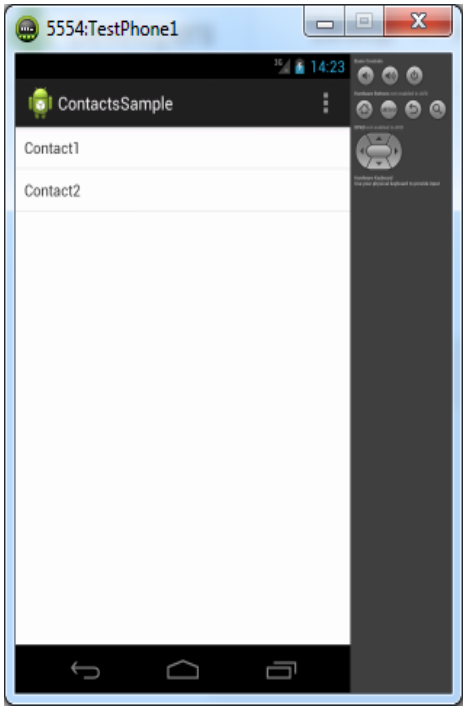
Завдання 1 «Отримання списку контактів»

Для читання інформації про контакти використовується контент-провайдер ContactsContract, точніше, один з його підкласів. Для цієї практичної роботи скористаємося провайдером ContactsContract.Contacts. Для читання контактів додатком потрібні повноваження READ_CONTACTS.

1. Додайте кілька контактів в емуляторі (оскільки потрібно тільки коротке ім'я контакту, інші поля можна не заповнювати :).

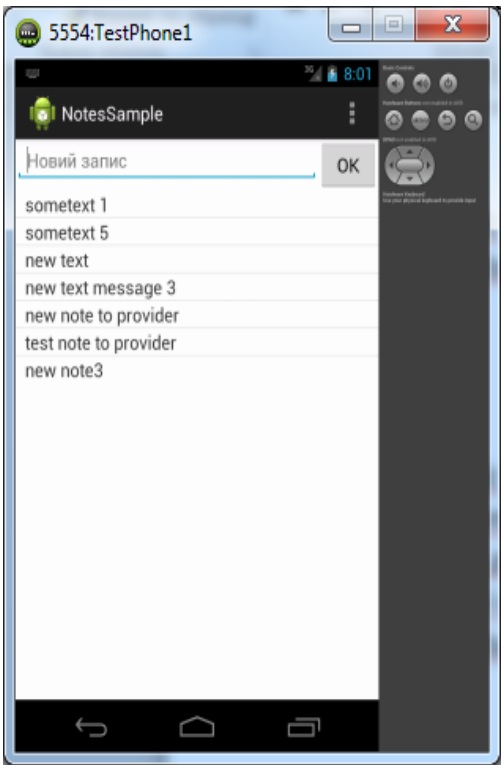
2. Створіть новий проект ContactsSample.

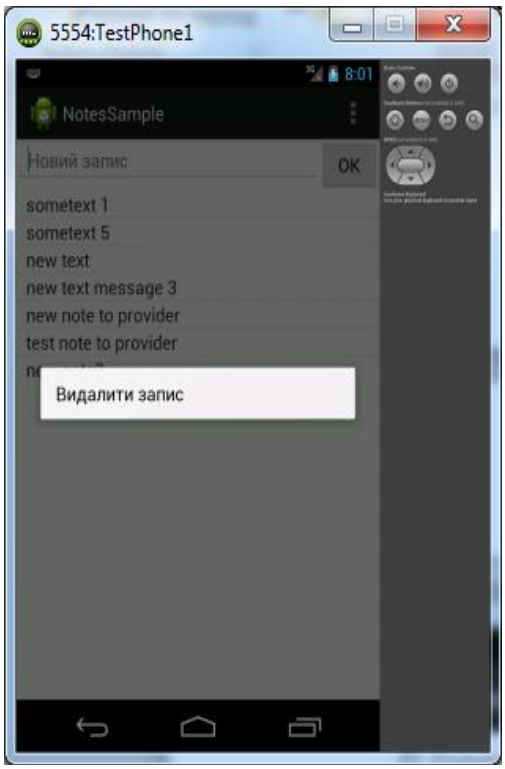
3. Виведіть імена всіх контактів (за допомогою ListView), використовуючи для отримання інформації URI ContactsContract.Contacts.CONTENT_URI. Необхідне ім'я поля для прив'язки адаптера знайдіть серед статичних констант класу ContactsContract.Contacts.



Завдання 2 «Створення контент-провайдера»

В якості основи замініть пряме звернення до СУБД на взаємодію з контент-провайдером, який буде інкапсуллювати всю взаємодію зі «справжніми» даними в окремому класі.





Крім простого відображення web-контенту за допомогою віджета WebView, розробник має можливість «низькорівневої роботи з різноманітними мережевими сервісами. Для цього всього лише потрібно створити мережеве підключення (запит до сервера), отримати, обробити і відобразити дані у потрібному вигляді. Традиційно найбільш зручними форматами для мережевого обміну даними вважаються XML і JSON.

Зрозуміло, будь-який додаток, що використовує мережеві підключення, повинен мати в маніфесті відповідні повноваження:

<uses-permission android:name="android.permission.INTERNET" />

Для створення потоку даних від сервера можна використовувати клас HttpUrlConnection, що є розширенням класу UrlConnection з пакету java.net. Пакети java.net і android.net містять класи, що дозволяють керувати мережевими з'єднаннями. Більш детальну інформацію про класу HttpUrlConnection з прикладами використання можна отримати тут: http://developer.android.com/reference/java/net/HttpURLConnection.html

Простий приклад створення з'єднання:

```
private static final String some_url = "....";
. . . .
try {
        // Створюємо об'єкт типу URL
        URL url = new URL(getString(R.string.rates_url));
        // З'єднуємося
        HttpURLConnection httpConnection = (HttpURLConnection) url
                        .openConnection();
        // Отримуємо код відповіді
        int responseCode = httpConnection.getResponseCode();
```

```
            // Якщо код відповіді хороший, парсер потік (відповідь сервера)
            if (responseCode == HttpURLConnection.HTTP_OK) {
                  // Якщо код відповіді хороший, обробляємо відповідь
                  InputStream in = httpConnection.getInputStream();
            } else {
            // Зробити оповіщення про помилки, якщо код відповіді
            нехороший
                  }
      } catch (MalformedURLException e) {
            e.printStackTrace();
      } catch (IOException e) {
            e.printStackTrace();
      }
```

Метою даного завдання є створення простого додатка, що одержує курси іноземних валют по відношенню до гривні у форматі XML і відображає дані у вигляді списку (*ListView*). Для отримання даних буде використовуватися URL http://bank-ua.com/export/currrate.xml

Відповідь сервера виглядає приблизно так:

```
<ValCurs Date="04.04.2017" name="Foreign Currency Market">
<Value ID="R01010">
      <NumCode>036</NumCode>
      <CharCode>AUD</CharCode>
      <Nominal>1</Nominal>
      <Name>Австралийский доллар</Name>
      <Value>30,4632</Value>
</Value>
. . . . . .
</ValCurs>
```

Для кожної валюти (елемент *Value*) буде потрібно витягти значення дочірніх елементів *CharCode*, *Nominal*, *Name* і *Value*. Значення атрибута *Date* кореневого елемента (*ValCurs*) використовуватиметься для зміни заголовка програми.

Завдання 3 «Робота із віддаленою БД»

1. Створіть новий проект CurrencyRates. Головна (і єдина) активність з таким же ім'ям (CurrencyRates) повинна розширювати клас ListActivity.

2. Відредагуйте Маніфест додатку, додавши в нього необхідні повноваження і тему для активності (android:theme= "@android:style/Theme.Light").

3. Файл строкових ресурсів strings.xml (в каталозі res / values) відредагуйте наступним чином:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name"> Курс гривні НБУ </string>
<string name="rates_url"> http://bank-ua.com/export/currrate.xml
</string>
</resources>
```

4. Для відображення інформації потрібно створити розмітку для елементів списку. У каталозі res / layout створіть файл item_view.xml з наступним вмістом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal" >
<TextView
    android:id="@+id/charCodeView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#FF8"
    android:minWidth="45sp"
    android:padding="4dp"
    android:textColor="#00F"
    android:textStyle="bold"
    android:gravity="center"
    android:shadowDx="8"
    android:shadowDy="8"
    android:shadowColor="#000"
    android:shadowRadius="8"/>
<TextView
    android:id="@+id/valueView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#008"
    android:background="#FFE"
    android:minEms="3"
    android:padding="3dp" />
<TextView
    android:id="@+id/nominalView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="3dp" />
<TextView
```

```
        android:id="@+id/nameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ellipsize="marquee"
        android:singleLine="true" />
</LinearLayout>
```

5. Вся логіка додатка буде зосереджена в класі Currency Rates, тому інші зміни будуть стосуватися тільки цього класу. Додайте необхідні константи:

```
private final static String KEY_CHAR_CODE = "CharCode";
private final static String KEY_VALUE = "Value";
private final static String KEY_NOMINAL = "Nominal";
private final static String KEY_NAME = "Name";
```

6. Тіло методу onCreate повинно містити тільки два рядки:

```
super.onCreate(savedInstanceState);
populate();
```

Оскільки Currency Rates є спадкоємцем List Activity, викликати setContentView не потрібно. Метод populate наповнюватиме ListView вмістом за допомогою адаптера (SimpleAdapter), заповнивши його даними, отриманими від методу getData.

7. Додайте метод populate, в якому створюється і налаштовується адаптер:

```
private void populate() {
    ArrayList<Map<String, String>> data = getData();
    String[] from = { KEY_CHAR_CODE, KEY_VALUE,
    KEY_NOMINAL, KEY_NAME };
    int[] to = { R.id.charCodeView, R.id.valueView, R.id.nominalView,
            R.id.nameView };
    SimpleAdapter sa = new SimpleAdapter(this, data,
    R.layout.item_view,
            from, to);
    setListAdapter(sa);}
```

8. Додайте метод getData. Саме в ньому буде створюватися і оброблятися з'єднання з сервером, а також аналізуватися XML-дані і заповнюватися список, що буде відображатися адаптером. Метод getData об'ємніший інших, але нічого складного в ньому немає (варто відзначити, що інтерфейси Document, Element і NodeList повинні імпортуватися з пакету org.w3c.dom):

```
private ArrayList<Map<String, String>> getData() {
ArrayList<Map<String, String>> list =
```

```
        new ArrayList<Map<String, String>>();
Map<String, String> m;
try {
        // створюємо об'єкт URL
        URL url = new URL(getString(R.string.rates_url));
        // З'єднуємося
        HttpURLConnection httpConnection =
                (HttpURLConnection) url.openConnection();
        // Отримуємо від сервера код відповіді
        int responseCode = httpConnection.getResponseCode();
        // Якщо код відповіді хороший, парсер потік (відповідь сервера),
        // встановлюємо дату в заголовку програми та
        // заповнити list
        if (responseCode == HttpURLConnection.HTTP_OK) {
                InputStream in = httpConnection.getInputStream();
                DocumentBuilderFactory dbf = DocumentBuilderFactory
                                .newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document dom = db.parse(in);
                Element docElement = dom.getDocumentElement();
                String date = docElement.getAttribute("Date");
                setTitle(getTitle() + " на " + date);
                NodeList nodeList = docElement
                        .getElementsByTagName("Valute");
                int count = nodeList.getLength();
                if (nodeList != null && count > 0) {
                        for (int i = 0; i < count; i++) {
                                Element entry = (Element) nodeList.item(i);
                                m = new HashMap<String, String>();
                                String charCode = entry
                                        .getElementsByTagName(KEY_CHAR_CODE)
                                        .item(0).getFirstChild().getNodeValue();
                                String value = entry
                                        .getElementsByTagName(KEY_VALUE)
                                        .item(0).getFirstChild().getNodeValue();
                                String nominal = "за " + entry
                                        .getElementsByTagName(KEY_NOMINAL)
                                        .item(0).getFirstChild().getNodeValue();
                                String name = entry
                                        .getElementsByTagName(KEY_NAME)
                                        .item(0).getFirstChild().getNodeValue();
                                m.put(KEY_CHAR_CODE, charCode);
                                m.put(KEY_VALUE, value);
                                m.put(KEY_NOMINAL, nominal);
```

```
                    m.put(KEY_NAME, name);
                    list.add(m);}}
} else {
// Зробити оповіщення про помилки, якщо код відповіді нехороший
}
} catch (MalformedURLException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
} catch (ParserConfigurationException e) {
        e.printStackTrace();
} catch (SAXException e) {
        e.printStackTrace();
}
return list;};
```

9. Перевірте працездатність програми. У реальній програмі слід відслідковувати наявність підключення пристрою до мережі, відслідковувати помилки з'єднання, а також отримувати дані з мережі в окремому потоці.

Завдання для виконання

Виконайте завдань до виконання у відповідності до теоретичних відомостей поданих вище.

Вимоги до звіту

В звіт по роботі включіть текст програм, що реалізують описані завдання та екранні форми, що відображають їх виконання.