

Практическая работа 14

Методические материалы

Провайдеры контента. Работа с json. Работа с XML.

Наше приложение может сохранять разнообразную информацию о пользователе, какие-то связанные данные в файлах или настройках. Однако ОС Android уже хранит ряд важной информации, связанной с пользователем, к которой имеем доступ и которую мы можем использовать. Это и списки контактов, и файлы сохраненных изображений и видеоматериалов, и какие-то отметки о звонках и т.д., то есть некоторый контент. А для доступа к этому контенту в ОС Android определены **провайдеры контента (content provider)**

В Android имеются следующие встроенные провайдеры, определенные в пакете android.content:

- **AlarmClock:** управление будильником
- **Browser:** история браузера и закладки
- **CalendarContract:** календарь и информации о событиях
- **CallLog:** информация о звонках
- **ContactsContract:** контакты
- **MediaStore:** медиа-файлы
- **SearchRecentSuggestions:** подсказки по поиску

- **Settings:** системные настройки
- **UserDictionary:** словарь слов, которые используются для быстрого набора
- **VoicemailContract:** записи голосовой почты

Работа с контактами

Контакты в Android обладают встроенным API, который позволяет получать и изменять список контактов. Все контакты хранятся в базе данных SQLite, однако они не представляют единой таблицы. Для контактов отведено три таблицы, связанных отношением один-ко-многим: таблица для хранения информации о людях, таблица их телефонов и таблица адресов их электронных почт. Но благодаря Android API мы можем абстрагироваться от связей между таблицами.

Общая форма получения контактов выглядит следующим образом:

```
ArrayList<String> contacts = new ArrayList<String>();

ContentResolver contentResolver = getContentResolver();

Cursor cursor =
contentResolver.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);

if(cursor!=null){
    while (cursor.moveToNext()) {
        // получаем каждый контакт

        String contact =
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_N
AME_PRIMARY));

        // добавляем контакт в список

        contacts.add(contact);
    }
}
```

```
cursor.close();  
}
```

Все контакты и сопутствующий функционал хранятся в специальных базах данных SQLite. Но нам не надо напрямую работать с ними. Мы можем воспользоваться объектом класса **Cursor**. Чтобы его получить, сначала вызывается метод `getContentResolver()`, который возвращает объект **ContentResolver**. Затем по цепочке вызывается метод `query()`. В этот метод передается ряд параметров, первый из которых представляет URI - ресурс, который мы хотим получить. Для обращения к базе данных контактов используется константа **ContactsContract.Contacts.CONTENT_URI**

Метод **contactsCursor.moveToNext()** позволяет последовательно перемещаться по записям контактов, считывая по одному контакту через вызов **contactsCursor.getString()**.

Таким образом, получать контакты не сложно. Главная сложность в работе с контактами, да и с любыми другими провайдерами контента, заключается в установке разрешений. До Android API 23 достаточно было установить соответствующее разрешение в файле манифеста приложения. Начиная же с API 23 (Android Marshmallow) Google изменил схему работы с разрешениями. И теперь пользователь сам должен решить, будет ли он давать разрешения приложению. В связи с чем разработчики должны добавлять дополнительный код.

Итак, для доступа к контактам нам надо установить разрешение **android.permission.READ_CONTACTS** в файле манифеста приложения:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.contactsapp">  
  
    <uses-permission android:name="android.permission.READ_CONTACTS" />  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="@string/app_name"
```

```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContactsApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Для вывода списка контактов в файле **activity_main.xml** определим следующую разметку интерфейса:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/header"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Контакты"
        android:textSize="18sp"
```

```
app:layout_constraintBottom_toTopOf="@id/contactList"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

```
<ListView
```

```
    android:id="@+id/contactList"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/header" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Для вывода списка контактов воспользуемся элементом `ListView`. И в классе **MainActivity** получим контакты:

```
package com.example.contactsapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.content.ContentResolver;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.os.Bundle;
```

```
import android.provider.ContactsContract;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_CODE_READ_CONTACTS=1;
    private static boolean READ_CONTACTS_GRANTED =false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // получаем разрешения

        int hasReadContactPermission = ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS);

        // если устройство до API 23, устанавливаем разрешение

        if(hasReadContactPermission ==
PackageManager.PERMISSION_GRANTED){
            READ_CONTACTS_GRANTED = true;
        }
        else{
            // вызываем диалоговое окно для установки разрешений

            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_CONTACTS},
REQUEST_CODE_READ_CONTACTS);
        }
    }
}
```

```

        // если разрешение установлено, загружаем контакты
        if (READ_CONTACTS_GRANTED){
            loadContacts();
        }
    }

    @Override

    public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults){

        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        if (requestCode == REQUEST_CODE_READ_CONTACTS) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                READ_CONTACTS_GRANTED = true;
            }
        }

        if(READ_CONTACTS_GRANTED){
            loadContacts();
        }
        else{
            Toast.makeText(this, "Требуется установить разрешения",
Toast.LENGTH_LONG).show();
        }
    }

    private void loadContacts(){
        ContentResolver contentResolver = getContentResolver();

```

```
Cursor cursor =
contentResolver.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);

ArrayList<String> contacts = new ArrayList<String>();

if(cursor!=null){
    while (cursor.moveToNext()) {

        // получаем каждый контакт
        String contact = cursor.getString(

cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME_PRIMAR
Y));

        // добавляем контакт в список
        contacts.add(contact);
    }
    cursor.close();
}

// создаем адаптер
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
    android.R.layout.simple_list_item_1, contacts);

ListView contactList = findViewById(R.id.contactList);
// устанавливаем для списка адаптер
contactList.setAdapter(adapter);
}
}
```


Кроме собственно загрузки контактов и передачи их через адаптер **ArrayAdapter** в список **ListView** здесь добавлено много кода по управлению разрешениями. Вначале определена переменная **READ_CONTACTS_GRANTED**, которая указывает, было ли выдано разрешение. И здесь есть два варианта действий.

Первый вариант предполагает, что устройство имеет версию Android ниже Marshmallow (ниже API 23). Для этого мы просто узнаем, есть ли разрешение для **READ_CONTACTS** и если оно есть, то устанавливаем для переменной **READ_CONTACTS_GRANTED** значение **true**:

```
int hasReadContactPermission = ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS);

if(hasReadContactPermission == PackageManager.PERMISSION_GRANTED){
    READ_CONTACTS_GRANTED = true;
}
```

Иначе нам надо отобразить пользователю диалоговое окно, где он решить, надо ли дать приложению разрешение:

```
ActivityCompat.requestPermissions(this,
    new String[]{Manifest.permission.READ_CONTACTS},
    REQUEST_CODE_READ_CONTACTS);
```

В этот метод передаются три параметра. Первый - текущий контекст, то есть текущий объект **Activity**.

Второй параметр представляет набор разрешений, которые надо получить, в виде массива строк. Нам надо получить в данном случае только одно разрешение - **Manifest.permission.READ_CONTACTS**.

Третий параметр представляет код запроса, через который мы сможем получить ответ пользователя.

Если мы хотим получить выбор пользователя при установке разрешений, то нам надо переопределить в классе **Activity** метод **onRequestPermissionsResult**:

```
public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults){
```

```

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

if (requestCode == REQUEST_CODE_READ_CONTACTS) {
    if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        READ_CONTACTS_GRANTED = true;
    }
}

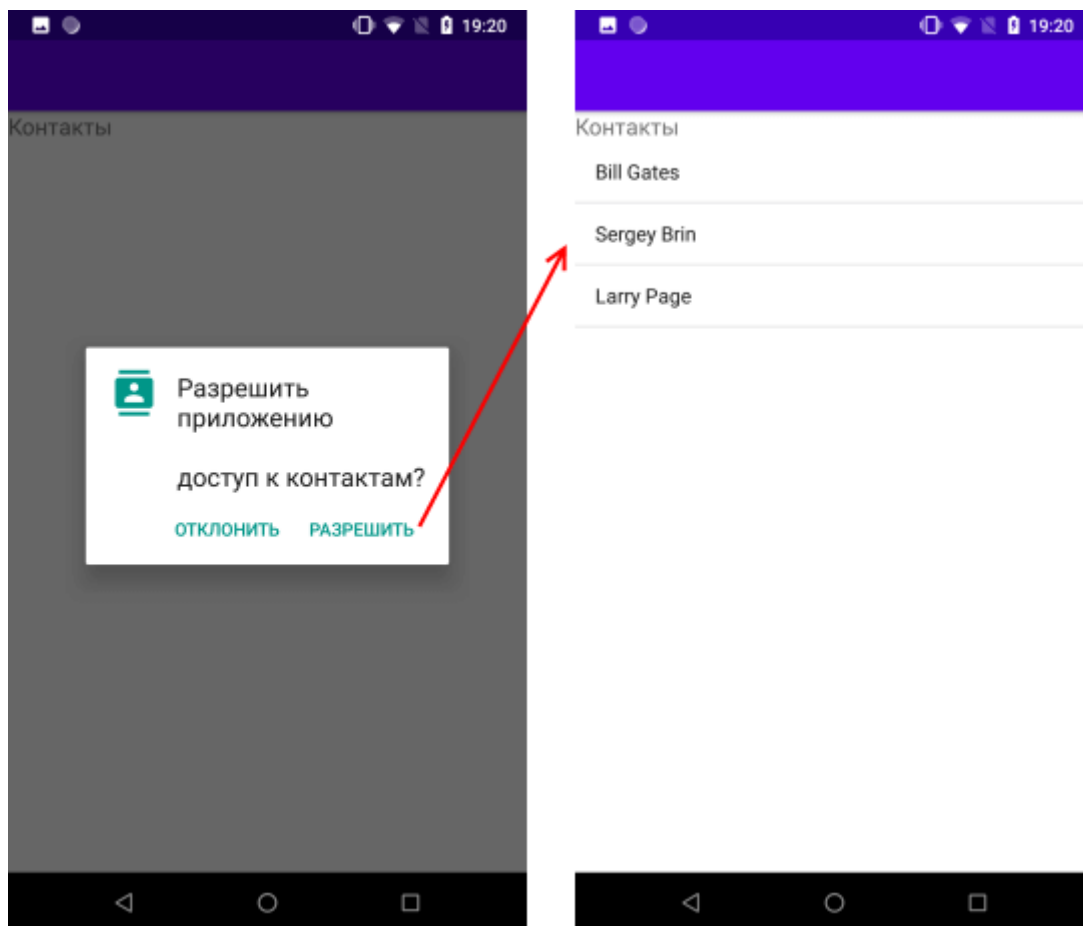
if(READ_CONTACTS_GRANTED){
    loadContacts();
}
else{
    Toast.makeText(this, "Требуется установить разрешения",
Toast.LENGTH_LONG).show();
}
}

```

Первый параметр метода **requestCode** - это тот код запроса, который передавался в качестве третьего параметра в **ActivityCompat.requestPermissions()**. Второй параметр - массив строк, для которых устанавливались разрешения. То есть одновременно мы можем устанавливать сразу несколько разрешений.

Третий параметр, собственно, хранит числовые коды разрешений. Так мы запрашиваем только одно разрешение, то первый элемент массива будет хранить его код. Через условное выражение мы можем проверить этот код: `grantResults[0] == PackageManager.PERMISSION_GRANTED`. И в зависимости от результата проверки изменить переменную `READ_CONTACTS_GRANTED`.

И при запуске приложения нам сначала отобразится окно для выдачи разрешения, а после выдачи подтверждений список контактов:



Чтение контактов в Android

После выдачи разрешения при повторных запусках приложения повторять разрешение не нужно, поэтому метод **onRequestPermissionsResult()** в таком случае будет срабатывать только один раз. А переменная **READ_CONTACTS_GRANTED** в этом случае уже будет иметь значение **true**.

Другая ситуация - если мы отклоним разрешение. В этом случае при повторном запуске приложения повторно будет отображаться данное окно.

Добавление контактов

Продолжим работу с проектом из прошлой темы и добавим в него возможность добавления новых контактов. Добавление контактов представляет собой запрос на изменение списка контактов, то есть его запись. Поэтому нам надо установить соответствующее разрешение в файле манифеста. Возьмем проект из прошлого материала и добавим в него в файл **AndroidManifest.xml** разрешение **android.permission.WRITE_CONTACTS**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.contactsapp">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContactsApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Для добавления контакта добавим изменим файл **activity_main.xml**, определив в нем текстовое поле для ввода данных:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout

```

```
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:app="http://schemas.android.com/apk/res-auto"  
android:layout_width="match_parent"  
android:layout_height="match_parent">
```

```
<EditText
```

```
    android:id="@+id/newContact"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    app:layout_constraintBottom_toTopOf="@id/header"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toLeftOf="@id/addBtn"  
    app:layout_constraintTop_toTopOf="parent" />
```

```
<Button
```

```
    android:id="@+id/addBtn"  
    android:text="Add"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onAddContact"  
    app:layout_constraintBottom_toTopOf="@id/header"  
    app:layout_constraintLeft_toRightOf="@id/newContact"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

```
<TextView
```

```
    android:id="@+id/header"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:text="Контакты"
```

```
android:textSize="18sp"
app:layout_constraintBottom_toTopOf="@id/contactList"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/newContact" />
```

```
<ListView
```

```
    android:id="@+id/contactList"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/header" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

В коде **MainActivity** пропишем обработчик **onAddContact** с добавлением контакта:

```
package com.example.contactsapp;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.content.ContentResolver;
import android.content.ContentUris;
```

```
import android.content.ContentValues;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;
```

```
import java.util.ArrayList;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    private static final int REQUEST_CODE_READ_CONTACTS=1;
```

```
    private static boolean READ_CONTACTS_GRANTED =false;
```

```
    ArrayList<String> contacts = new ArrayList<>();
```

```
    Button addBtn;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

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

```
        setContentView(R.layout.activity_main);
```

```
        addBtn = findViewById(R.id.addBtn);
```

```
        // получаем разрешения
```

```
int hasReadContactPermission = ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS);
```

```
// если устройство до API 23, устанавливаем разрешение
```

```
if(hasReadContactPermission ==
PackageManager.PERMISSION_GRANTED){

    READ_CONTACTS_GRANTED = true;

}
```

```
else{
```

```
    // вызываем диалоговое окно для установки разрешений
```

```
    ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_CONTACTS,
Manifest.permission.WRITE_CONTACTS},
REQUEST_CODE_READ_CONTACTS);

}
```

```
// если разрешение установлено, загружаем контакты
```

```
if (READ_CONTACTS_GRANTED){

    loadContacts();

}
```

```
addBtn.setEnabled(READ_CONTACTS_GRANTED);
```

```
}
```

```
@Override
```

```
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults){
```

```
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
```

```
    if (requestCode == REQUEST_CODE_READ_CONTACTS) {
```

```
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
```



```

        READ_CONTACTS_GRANTED = true;
    }

    addBtn.setEnabled(READ_CONTACTS_GRANTED);
}

if(READ_CONTACTS_GRANTED){
    loadContacts();
}
else{
    Toast.makeText(this, "Требуется установить разрешения",
Toast.LENGTH_LONG).show();
}
}

public void onAddContact(View v) {
    ContentValues contactValues = new ContentValues();
    EditText contactText = findViewById(R.id.newContact);
    String newContact = contactText.getText().toString();
    contactText.setText("");

    contactValues.put(ContactsContract.RawContacts.ACCOUNT_NAME,
newContact);

    contactValues.put(ContactsContract.RawContacts.ACCOUNT_TYPE,
newContact);

    Uri newUri =
getContentResolver().insert(ContactsContract.RawContacts.CONTENT_URI,
contactValues);

    long rawContactsId = ContentUris.parseId(newUri);

    contactValues.clear();

    contactValues.put(ContactsContract.Data.RAW_CONTACT_ID,
rawContactsId);

    contactValues.put(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
;
}

```

```

contactValues.put(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME, newContact);

    getResolver().insert(ContactsContract.Data.CONTENT_URI,
contactValues);

    Toast.makeText(getApplicationContext(), newContact + " добавлен в список
контактов", Toast.LENGTH_LONG).show();

    loadContacts();
}

private void loadContacts(){
    contacts.clear();

    ContentResolver contentResolver = getResolver();

    Cursor cursor =
contentResolver.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);

    if(cursor!=null){
        while (cursor.moveToNext()) {

            // получаем каждый контакт

            String contact = cursor.getString(

cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY));

            // добавляем контакт в список

            contacts.add(contact);
        }

        cursor.close();
    }

    // создаем адаптер

    ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, contacts);

```

```

        // устанавливаем для списка адаптер
        ListView contactList = findViewById(R.id.contactList);
        contactList.setAdapter(adapter);
    }
}

```

Сразу стоит отметить, что для работы с контактами не надо отдельно получать разрешения на чтение и отдельно на изменение контактов. Пользователь один раз согласие для установки сразу двух разрешений. Однако на уровне кода нам необходимо перечислить через запятую устанавливаемые разрешения:

```

// вызываем диалоговое окно для установки разрешений
ActivityCompat.requestPermissions(this, new String[]{
    Manifest.permission.READ_CONTACTS,
    Manifest.permission.WRITE_CONTACTS
},
    REQUEST_CODE_READ_CONTACTS);

```

Однако мы опять же можем управлять разрешением, например, установить доступность кнопки:

```
addBtn.setEnabled(READ_CONTACTS_GRANTED);
```

Если разрешение не получено, то переменная **READ_CONTACTS_GRANTED** будет иметь значение false, и соответственно кнопка будет недоступна, и мы не сможем добавить новый контакт.

Весь код добавления находится в обработчике нажатия кнопки onAddContact. В Android контакты распределяются по трем таблицам: contacts, raw contacts и data. И нам надо добавить новый контакт в две последние таблицы. В таблицу contact в силу настроек мы добавить не можем, но это и не нужно.

Данные контакта представляют объект **ContentValues**, который состоит из ключей и их значений, то есть объект словаря. После его создания происходит добавление в него пары элементов:

```
contactValues.put(RawContacts.ACCOUNT_NAME, newContact);  
contactValues.put(RawContacts.ACCOUNT_TYPE, newContact);
```

Здесь устанавливается название и тип контакта. В качестве ключей выставляются значения **RawContacts.ACCOUNT_NAME** и **RawContacts.ACCOUNT_TYPE**, а в качестве их значения - текст из текстового поля.

Далее этот объект добавляется в таблицу **RawContacts** с помощью метода **insert()**:

```
Uri newUri = getContentResolver().insert(RawContacts.CONTENT_URI,  
contactValues);
```

Метод **insert()** возвращает URI - ссылку на добавленный объект в таблице, у которого мы можем получить id. Затем после очистки мы подготавливаем объект для добавления в таблицу Data, вновь наполняя его данными:

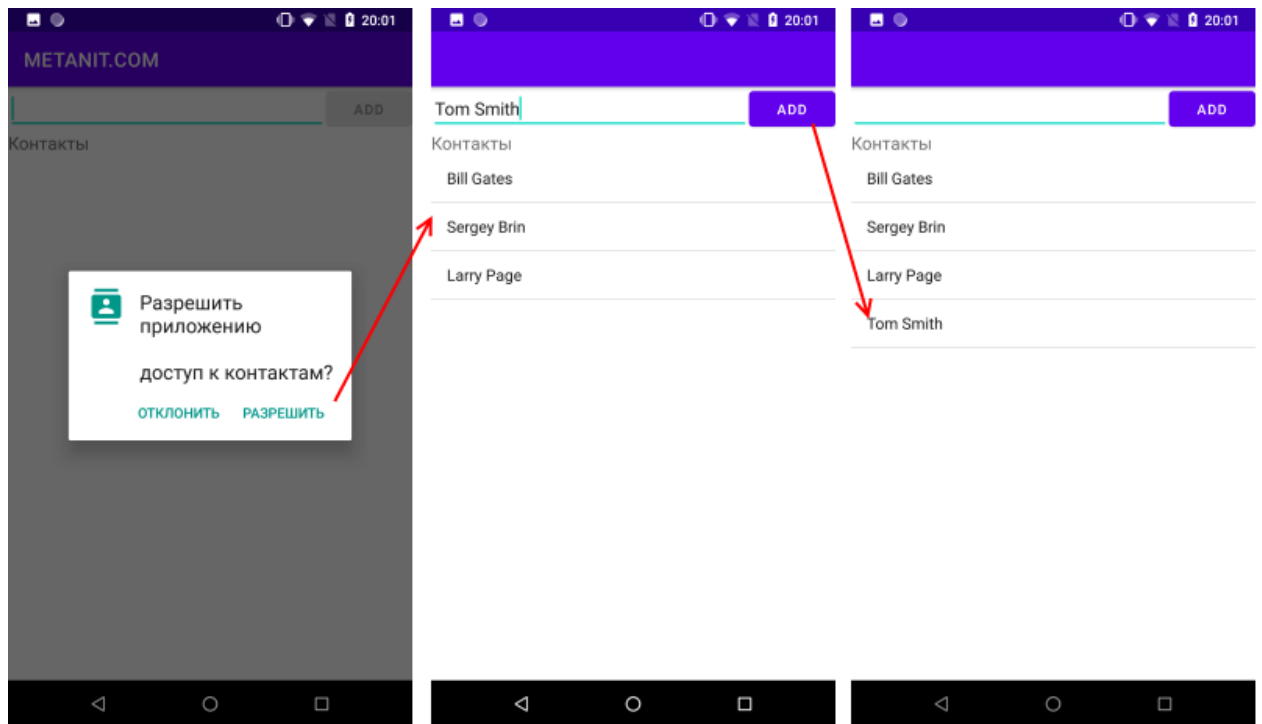
```
contactValues.put(Data.RAW_CONTACT_ID, rawContactsId);  
contactValues.put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);  
contactValues.put(StructuredName.DISPLAY_NAME, newContact);
```

И опять добавление производит метод **insert()**:

```
getContentResolver().insert(Data.CONTENT_URI, contactValues);
```

Перед запуском, если ранее (в прошлой теме) приложение было установлено, то его необходимо удалить, чтобы установить для приложения новые разрешения (разрешение на запись контактов).

Запустим приложение и добавим новый контакт:



Новый контакт в Android

Создание провайдера контента. (Часть 1. - определение контракта)

Контент-провайдеры (content providers) позволяют обращаться одним приложениям к данным других приложений. И мы тоже можем сделать, чтобы другие приложения могли обращаться к данным нашего приложения через некоторый API. Для этого нам надо создать свой контент-провайдер. Рассмотрим как это сделать.

Вначале добавим в проект класс **FriendsContract**, который будет описывать основные значения, столбцы, адреса uri, используемые в контент-провайдере.

```
package com.example.friendsproviderapp;
```

```
import android.content.ContentUris;
```

```
import android.net.Uri;
```

```

public class FriendsContract {

    static final String TABLE_NAME = "friends";

    static final String CONTENT_AUTHORITY = "com.example.friendsprovider";

    static final Uri CONTENT_AUTHORITY_URI = Uri.parse("content://" +
CONTENT_AUTHORITY);

    static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd." +
CONTENT_AUTHORITY + "." + TABLE_NAME;

    static final String CONTENT_ITEM_TYPE= "vnd.android.cursor.item/vnd." +
CONTENT_AUTHORITY + "." + TABLE_NAME;

    public static class Columns{

        public static final String _ID = "_id";

        public static final String NAME = "Name";

        public static final String EMAIL = "Email";

        public static final String PHONE = "Phone";

        private Columns(){

        }

    }

    static final Uri CONTENT_URI =
Uri.withAppendedPath(CONTENT_AUTHORITY_URI, TABLE_NAME);

    // создает uri с помощью id

    static Uri buildFriendUri(long taskId){

        return ContentUris.withAppendedId(CONTENT_URI, taskId);

    }

    // получает id из uri

    static long getFriendId(Uri uri){

```

```

        return ContentUris.parseId(uri);
    }

}

```

С помощью константы `TABLE_NAME` определяется имя таблицы, к которой будет происходить обращение. А вложенный статический класс `Columns` описывает столбцы этой таблицы. То есть таблица будет называться "friends", а столбцы - "_id", "Name", "Email", "Phone". То есть условно говоря в таблице будут храниться данные о друзьях - имя, электронный адрес и номер телефона.

Константа `CONTENT_AUTHORITY` описывает название контент-провайдера. То есть в моем случае провайдер будет называться "com.example.friendsprovider". С помощью имени провайдера создается константа `CONTENT_AUTHORITY_URI` - универсальный локатор или своего рода путь, через который мы будем обращаться к провайдеру при выполнении с ним различных операций.

Также класс определяет две константы `CONTENT_TYPE` и `CONTENT_ITEM_TYPE`, которые определяют тип возвращаемого содержимого. Здесь есть два варианта: возвращение набора данных и возвращение одного объекта. Значение, определяющее набор данных, строится по принципу "vnd.android.cursor.dir/vnd.[name].[table]", где в качестве [name] обычно выступает глобально уникальный идентификатор, например, название провайдера или имя пакета провайдера. А в качестве [type], как правило, используется имя таблицы. По похожей схеме строится второе значение, только вместо "dir" ставится "item".

Также в классе определяется вспомогательная константа `CONTENT_URI`, которая описывает путь для доступа к таблице friends. И также определяем два вспомогательных метода: `buildFriendUri()` (возвращает uri для доступа к объекту по определенному id) и `getFriendId` (для извлечения id из переданного пути uri).

Далее добавим в проект новый класс **AppDatabase**:

```
package com.example.friendsproviderapp;
```

```
import android.content.Context;
```

```

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

public class AppDatabase extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "friends.db";
    public static final int DATABASE_VERSION = 1;

    private static AppDatabase instance = null;

    private AppDatabase(Context context){
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    static AppDatabase getInstance(Context context){
        if(instance == null){
            instance = new AppDatabase(context);
        }
        return instance;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        String sql = "CREATE TABLE " + FriendsContract.TABLE_NAME + "(" +
            FriendsContract.Columns._ID + " INTEGER PRIMARY KEY NOT
NULL, " +
            FriendsContract.Columns.NAME + " TEXT NOT NULL, " +
            FriendsContract.Columns.EMAIL + " TEXT, " +
            FriendsContract.Columns.PHONE + " TEXT NOT NULL)";
    }
}

```



```

db.execSQL(sql);

// добавление начальных данных

db.execSQL("INSERT INTO "+ FriendsContract.TABLE_NAME +" (" +
FriendsContract.Columns.NAME
        + ", " + FriendsContract.Columns.PHONE + ") VALUES ('Tom',
'+12345678990');");

db.execSQL("INSERT INTO "+ FriendsContract.TABLE_NAME +" (" +
FriendsContract.Columns.NAME
        + ", " + FriendsContract.Columns.EMAIL + ", " +
FriendsContract.Columns.PHONE +
        " ) VALUES ('Bob', 'bob@gmail.com', '+13456789102');");
}

@Override

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

}

}

```

Данный класс по принципу синглтона организует доступ к базе данных и, кроме того, создает саму базу данных и добавляет в нее начальные данные.

Создание провайдера контента. (Часть 2- создание провайдера)

Продолжим работу с проектом из прошлого материала и добавим в него класс **AppProvider**, который, собственно, и будет представлять провайдер контента:

```
package com.example.friendsproviderapp;
```

```
import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

public class AppProvider extends ContentProvider {

    private AppDatabase mOpenHelper;
    private static final UriMatcher sUriMatcher = buildUriMatcher();

    public static final int FRIENDS = 100;
    public static final int FRIENDS_ID = 101;

    private static UriMatcher buildUriMatcher(){
        final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
        // content://com.example.friendsprovider/FRIENDS
        matcher.addURI(FriendsContract.CONTENT_AUTHORITY,
            FriendsContract.TABLE_NAME, FRIENDS);

        // content://com.example.friendsprovider/FRIENDS/8
        matcher.addURI(FriendsContract.CONTENT_AUTHORITY,
            FriendsContract.TABLE_NAME + "/" + "#", FRIENDS_ID);

        return matcher;
    }
}
```

```
}
```

```
@Override
```

```
public boolean onCreate() {
```

```
    mOpenHelper = AppDatabase.getInstance(getContext());
```

```
    return true;
```

```
}
```

```
@Nullable
```

```
@Override
```

```
public Cursor query(@NonNull Uri uri, @Nullable String[] projection,  
@Nullable String selection, @Nullable String[] selectionArgs, @Nullable String  
sortOrder) {
```

```
    final int match = sUriMatcher.match(uri);
```

```
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
```

```
    switch(match){
```

```
        case FRIENDS:
```

```
            queryBuilder.setTables(FriendsContract.TABLE_NAME);
```

```
            break;
```

```
        case FRIENDS_ID:
```

```
            queryBuilder.setTables(FriendsContract.TABLE_NAME);
```

```
            long taskId = FriendsContract.getFriendId(uri);
```

```
            queryBuilder.appendWhere(FriendsContract.Columns._ID + " = " +  
taskId);
```

```
            break;
```

```
        default:
```

```
            throw new IllegalArgumentException("Unknown URI: " + uri);
```

```
    }
```

```
    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
```

```
        return queryBuilder.query(db, projection, selection, selectionArgs, null, null,
sortOrder);
    }
}
```

@Nullable

@Override

```
public String getType(@NonNull Uri uri) {
```

```
    final int match = sUriMatcher.match(uri);
```

```
    switch(match){
```

```
        case FRIENDS:
```

```
            return FriendsContract.CONTENT_TYPE;
```

```
        case FRIENDS_ID:
```

```
            return FriendsContract.CONTENT_ITEM_TYPE;
```

```
        default:
```

```
            throw new IllegalArgumentException("Unknown URI: "+ uri);
```

```
    }
```

```
}
```

@Nullable

@Override

```
public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) {
```

```
    final int match = sUriMatcher.match(uri);
```

```
    final SQLiteDatabase db;
```

```
    Uri returnUri;
```

```
    long recordId;
```

```
    if (match == FRIENDS) {
```

```

db = mOpenHelper.getWritableDatabase();
recordId = db.insert(FriendsContract.TABLE_NAME, null, values);
if (recordId > 0) {
    returnUri = FriendsContract.buildFriendUri(recordId);
} else {
    throw new SQLException("Failed to insert: " + uri.toString());
}
} else {
    throw new IllegalArgumentException("Unknown URI: " + uri);
}
return returnUri;
}

```

@Override

```

public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable
String[] selectionArgs) {

```

```

    final int match = sUriMatcher.match(uri);

```

```

    final SQLiteDatabase db = mOpenHelper.getWritableDatabase();

```

```

    String selectionCriteria = selection;

```

```

    if (match != FRIENDS && match != FRIENDS_ID)

```

```

        throw new IllegalArgumentException("Unknown URI: " + uri);

```

```

    if (match == FRIENDS_ID) {

```

```

        long taskId = FriendsContract.getFriendId(uri);

```

```

        selectionCriteria = FriendsContract.Columns._ID + " = " + taskId;

```

```

        if ((selection != null) && (selection.length() > 0)) {

```

```

            selectionCriteria += " AND (" + selection + ")";

```

```

    }
}

return db.delete(FriendsContract.TABLE_NAME, selectionCriteria,
selectionArgs);
}

@Override

public int update(@NonNull Uri uri, @Nullable ContentValues values,
@Nullable String selection, @Nullable String[] selectionArgs) {

    final int match = sUriMatcher.match(uri);

    final SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    String selectionCriteria = selection;

    if(match != FRIENDS && match != FRIENDS_ID)

        throw new IllegalArgumentException("Unknown URI: " + uri);

    if(match==FRIENDS_ID) {

        long taskId = FriendsContract.getFriendId(uri);

        selectionCriteria = FriendsContract.Columns._ID + " = " + taskId;

        if ((selection != null) && (selection.length() > 0)) {

            selectionCriteria += " AND (" + selection + ")";

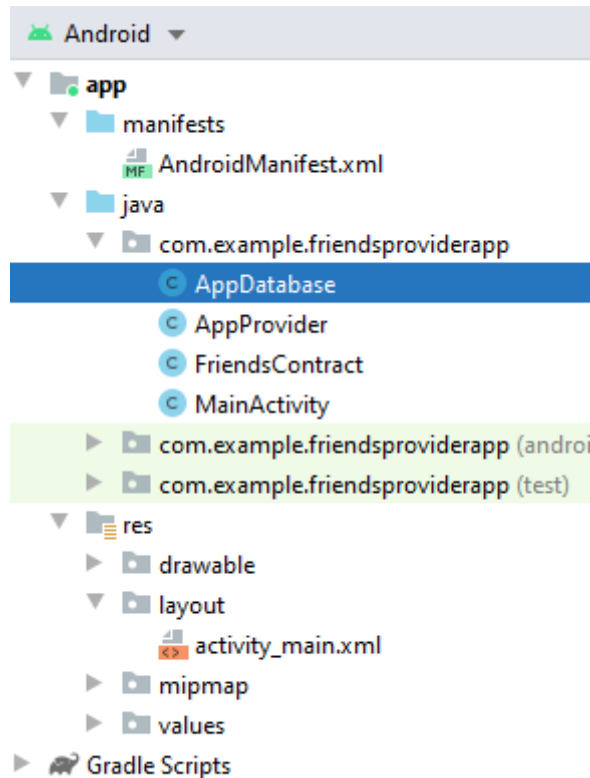
        }

    }

    return db.update(FriendsContract.TABLE_NAME, values, selectionCriteria,
selectionArgs);
}
}

```

В итоге получится следующий проект:



Создание Content Provider в Android и Java

Класс провайдера контента должен наследоваться от абстрактного класса **ContentProvider**, который определяет ряд методов для работы с данными, в частности, методы onCreate, query, insert, update, delete, getType.

Для построения путей uri для запросов к источнику данных определен объект sUriMatcher, который представляет тип **UriMatcher**. Для его создания применяется метод buildUriMatcher:

```
private static UriMatcher buildUriMatcher(){  
    final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);  
    // content://com.example.friendsprovider/FRIENDS  
    matcher.addURI(FriendsContract.CONTENT_AUTHORITY,  
FriendsContract.TABLE_NAME, FRIENDS);  
    // content://com.example.friendsprovider/FRIENDS/8  
    matcher.addURI(FriendsContract.CONTENT_AUTHORITY,  
FriendsContract.TABLE_NAME + "/#", FRIENDS_ID);  
}
```

```
    return matcher;
}
```

С помощью метода `addURI` в объект `UriMatcher` добавляется определенный путь `uri`, используемый для отправки запроса. В качестве первого параметра `addUri` принимает название провайдера, который описывается константой `CONTENT_AUTHORITY`. Второй параметр - путь к данным в рамках источника данных - в данном случае это таблица `friends`. Третий параметр - числовой код, который позволяет разграничить характер операции. В данном случае у нас возможны два типа запросов - для обращения ко всей таблице, либо для обращения к отдельному объекту, вне зависимости идет ли речь о добавлении, получении, обновлении или удалении данных. Поэтому добавляются два `uri`. И для каждого используется один из двух числовых кодов - `FRIENDS` или `FRIENDS_ID`. Это могут быть абсолютно любые числовые коды. Но они позволят затем узнать, идет запрос ко всей таблице в целом или к какому-то одному определенному объекту.

Метод `oncreate()` выполняет начальную инициализацию провайдера при его создании. В данном случае просто устанавливается используемая база данных:

```
public boolean onCreate() {
    mOpenHelper = AppDatabase.getInstance(getContext());
    return true;
}
```

Получение данных

Для получения данных в провайдере определен метод `query()`.

```
public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable
String selection,
                    @Nullable String[] selectionArgs, @Nullable String sortOrder) {
    final int match = sUriMatcher.match(uri);
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
    switch(match){
```



```

    case FRIENDS:
        queryBuilder.setTables(FriendsContract.TABLE_NAME);
        break;
    case FRIENDS_ID:
        queryBuilder.setTables(FriendsContract.TABLE_NAME);
        long taskId = FriendsContract.getFriendId(uri);
        queryBuilder.appendWhere(FriendsContract.Columns._ID + " = " +
taskId);
        break;
    default:
        throw new IllegalArgumentException("Unknown URI: " + uri);
}
SQLiteDatabase db = mOpenHelper.getReadableDatabase();
return queryBuilder.query(db, projection, selection, selectionArgs, null, null,
sortOrder);
}

```

Данный метод должен принимать пять параметров:

uri: путь запроса

projection: набор столбцов, данные для которых надо получить

selection: выражение для выборки типа "WHERE Name = ?"

selectionArgs: набор значений для параметров из selection (вставляются вместо знаков вопроса)

sortOrder: критерий сортировки, в качестве которого выступает имя столбца

С помощью объекта **SQLiteQueryBuilder** создаем запрос sql, который будет выполняться. Для этого вначале получаем числовой код операции с помощью выражения `sUriMatcher.match(uri)`. То есть здесь мы узнаем, обращен запрос ко всей таблице (код `FRIENDS`) или к одному объекту (код `FRIENDS_ID`). Если запрос обращен ко всей таблице, то вызываем метод `queryBuilder.setTables(FriendsContract.TABLE_NAME)`.

Если запрос идет к одному объекту, то в этом случае получаем собственно идентификатор объекта и с помощью метода `appendWhere()` добавляем условие для выборки по данному идентификатору.

В конце собственно выполняем запрос с помощью метода `queryBuilder.query()` и возвращаем объект `Cursor`.

Далее мы рассмотрим использование этого метода и возвращаемого им курсора.

Добавление данных

Для добавления данных применяется метод **insert()**:

`@Nullable`

`@Override`

```
public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) {
```

```
    final int match = sUriMatcher.match(uri);
```

```
    final SQLiteDatabase db;
```

```
    Uri returnUri;
```

```
    long recordId;
```

```
    if (match == FRIENDS) {
```

```

db = mOpenHelper.getWritableDatabase();
recordId = db.insert(FriendsContract.TABLE_NAME, null, values);
if (recordId > 0) {
    returnUri = FriendsContract.buildFriendUri(recordId);
} else {
    throw new SQLException("Failed to insert: " + uri.toString());
}
} else {
    throw new IllegalArgumentException("Unknown URI: " + uri);
}
return returnUri;
}

```

Метод принимает два параметра:

- **uri:** путь запроса
- **values:** объект ContentValues, через который передаются добавляемые данные

Для выполнения добавления выполняется метод `db.insert`, который возвращает идентификатор добавленного объекта:

```
recordId = db.insert(TasksContract.TABLE_NAME, null, values);
```

С помощью этого идентификатора создается и возвращается путь Uri к созданному объекту.

Удаление данных

```
public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable
String[] selectionArgs) {

    final int match = sUriMatcher.match(uri);

    final SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    String selectionCriteria = selection;

    if(match != FRIENDS && match != FRIENDS_ID)

        throw new IllegalArgumentException("Unknown URI: "+ uri);

    if(match==FRIENDS_ID) {

        long taskId = FriendsContract.getFriendId(uri);

        selectionCriteria = FriendsContract.Columns._ID + " = " + taskId;

        if ((selection != null) && (selection.length() > 0)) {

            selectionCriteria += " AND (" + selection + ")";

        }

    }

    return db.delete(FriendsContract.TABLE_NAME, selectionCriteria,
selectionArgs);

}
```

Данный метод должен принимать три параметра:

- **uri:** путь запроса
- **selection:** выражение для выборки типа "WHERE Name = ?"

- **selectionArgs**: набор значений для параметров из selection (вставляются вместо знаков вопроса)

При удалении мы можем реализовать один из двух сценариев: либо удалить из таблицы набор данных (например, друзей, у которых имя Том), либо удалить один объект по определенному идентификатору. В случае если идет удаление по идентификатору, то к выражению выборки удаляемых данных в selection добавляется условие удаления по id:

```
long taskId = FriendsContract.getFriendId(uri);
selectionCriteria = FriendsContract.Columns._ID + " = " + taskId;
if((selection != null) && (selection.length() > 0)){
    selectionCriteria += " AND (" + selection + ")";
}
count = db.delete(FriendsContract.TABLE_NAME, selectionCriteria,
selectionArgs);
```

Результатом удаления является количество удаленных строк в таблице.

Обновление данных

Для обновления данных применяется метод **update()**:

```
public int update(@NonNull Uri uri, @Nullable ContentValues values, @Nullable
String selection, @Nullable String[] selectionArgs) {

    final int match = sUriMatcher.match(uri);

    final SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    String selectionCriteria = selection;

    if(match != FRIENDS && match != FRIENDS_ID)
        throw new IllegalArgumentException("Unknown URI: "+ uri);
```

```

if(match==FRIENDS_ID) {
    long taskId = FriendsContract.getFriendId(uri);
    selectionCriteria = FriendsContract.Columns._ID + " = " + taskId;
    if ((selection != null) && (selection.length() > 0)) {
        selectionCriteria += " AND (" + selection + ")";
    }
}

return db.update(FriendsContract.TABLE_NAME, values, selectionCriteria,
selectionArgs);
}

```

Данный метод должен принимать четыре параметра:

uri: путь запроса

values: объект ContentValues, который определяет новые значения

selection: выражение для выборки типа "WHERE Name = ?"

selectionArgs: набор значений для параметров из selection (вставляются вместо знаков вопроса)

Метод update во многом аналогичен методу delete за тем исключением, что в метод передаются данные типа ContentValues, которые передаются в метод db.update().

AndroidManifest

Но чтобы провайдер контента заработал, необходимо внести изменения в файл **AndroidManifest.xml**. К примеру, по умолчанию данный файл выглядит примерно следующим образом:

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.friendsproviderapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.FriendsProviderApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

И в конец элемента <application> добавим определение провайдера:

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

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.friendsproviderapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.FriendsProviderApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <provider
            android:authorities="com.example.friendsprovider"
            android:name="com.example.friendsproviderapp.AppProvider"
            android:exported="false"/>
    </application>

</manifest>
```

В элементе provider атрибут android:authorities указывает на название провайдера - в данном случае это название, которое определено в прошлой теме в константе CONTENT_AUTHORITY в классе FriendsContract, то есть **com.example.friendsprovider**. А атрибут android:name указывает на полное

название класса провайдера с учетом его пакета. В моем случае пакет **com.example.friendsproviderapp**, а класс провайдера - **AppProvider**, поэтому в итоге получается **com.example.friendsproviderapp.AppProvider**.

Создание провайдера контента. (Часть 3-применение)

В прошлом материале был определен провайдер контента. Рассмотрим, как его использовать. Сначала определим простейший визуальный интерфейс для тестирования возможностей провайдера в файле **activity_main.xml**:

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

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button

        android:id="@+id/getButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Get"
        android:onClick="getAll"
        app:layout_constraintBottom_toTopOf="@id/addButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button

        android:id="@+id/addButton"
```

```
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Add"
android:onClick="add"
app:layout_constraintBottom_toTopOf="@id/updateButton"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/getButton" />
```

<Button

```
android:id="@+id/updateButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Update"
android:onClick="update"
app:layout_constraintBottom_toTopOf="@id/deleteButton"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/addButton" />
```

<Button

```
android:id="@+id/deleteButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Delete"
android:onClick="delete"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/updateButton" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определен набор кнопок для вывода списка друзей, а также добавления, обновления и удаления. Каждая кнопка будет вызывать соответствующий метод в классе MainActivity.

Теперь изменим код класса MainActivity. Для упрощения результаты будем выводить в окне Logcat с помощью метода Log.d():

```
package com.example.friendsproviderapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.ContentResolver;
```

```
import android.content.ContentValues;
```

```
import android.database.Cursor;
```

```
import android.net.Uri;
```

```
import android.os.Bundle;
```

```
import android.util.Log;
```

```
import android.view.View;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    private static final String TAG = "MainActivity";
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

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

```
        setContentView(R.layout.activity_main);
```

```
    }
```

```

// получение всех
public void getAll(View view){
    String[] projection = {
        FriendsContract.Columns._ID,
        FriendsContract.Columns.NAME,
        FriendsContract.Columns.EMAIL,
        FriendsContract.Columns.PHONE
    };
    ContentResolver contentResolver = getContentResolver();
    Cursor cursor = contentResolver.query(FriendsContract.CONTENT_URI,
        projection,
        null,
        null,
        FriendsContract.Columns.NAME);
    if(cursor != null){
        Log.d(TAG, "count: " + cursor.getCount());
        // перебор элементов
        while(cursor.moveToNext()){
            for(int i=0; i < cursor.getColumnCount(); i++){
                Log.d(TAG, cursor洗getColumn洗Name(i) + " : " + cursor.getString(i));
            }
            Log.d(TAG, "=====");
        }
        cursor.close();
    }
    else{
        Log.d(TAG, "Cursor is null");
    }
}

```

// Добавление

```
public void add(View view){  
    ContentResolver contentResolver = getContentResolver();  
    ContentValues values = new ContentValues();  
    values.put(FriendsContract.Columns.NAME, "Sam");  
    values.put(FriendsContract.Columns.EMAIL, "sam@gmail.com");  
    values.put(FriendsContract.Columns.PHONE, "+13676254985");  
    Uri uri = contentResolver.insert(FriendsContract.CONTENT_URI, values);  
    Log.d(TAG, "Friend added");  
}
```

// Обновление

```
public void update(View view){  
    ContentResolver contentResolver = getContentResolver();  
    ContentValues values = new ContentValues();  
    values.put(FriendsContract.Columns.EMAIL, "sammy@gmail.com");  
    values.put(FriendsContract.Columns.PHONE, "+55555555555");  
    String selection = FriendsContract.Columns.NAME + " = 'Sam'";  
    int count = contentResolver.update(FriendsContract.CONTENT_URI, values,  
selection, null);  
    Log.d(TAG, "Friend updated");  
}
```

// Удаление

```
public void delete(View view){  
    ContentResolver contentResolver = getContentResolver();  
    String selection = FriendsContract.Columns.NAME + " = ?";  
    String[] args = {"Sam"};  
    int count = contentResolver.delete(FriendsContract.CONTENT_URI,  
selection, args);
```

```
        Log.d(TAG, "Friend deleted");
    }
}
```

Разберем отдельные действия, выполняемые в данном коде.

Получение данных

```
public void getAll(View view){
    String[] projection = {
        FriendsContract.Columns._ID,
        FriendsContract.Columns.NAME,
        FriendsContract.Columns.EMAIL,
        FriendsContract.Columns.PHONE
    };
    ContentResolver contentResolver = getContentResolver();
    Cursor cursor = contentResolver.query(FriendsContract.CONTENT_URI,
        projection,
        null,
        null,
        FriendsContract.Columns.NAME);
    if(cursor != null){
        Log.d(TAG, "count: " + cursor.getCount());
        // перебор элементов
        while(cursor.moveToNext()){
            for(int i=0; i < cursor.getColumnCount(); i++){
                Log.d(TAG, cursor洗getColumnName(i) + " : " + cursor.getString(i));
            }
        }
        Log.d(TAG, "=====");
    }
}
```

```

    }

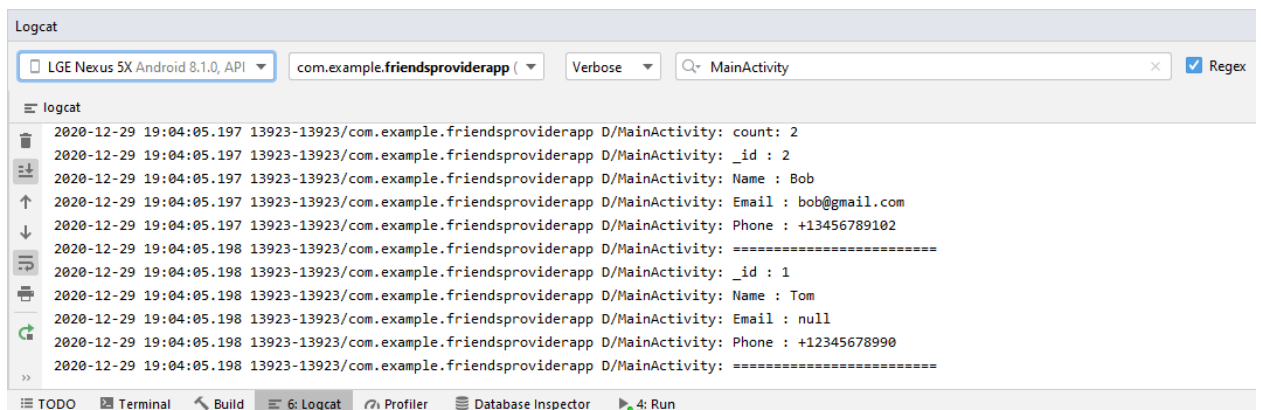
    cursor.close();
}

else{
    Log.d(TAG, "Cursor is null");
}
}

```

Взаимодействие с провайдером контента осуществляется через объект **ContentResolver**. Для получения данных вызывается метод `query()`, по сути он представляет вызов метод `query` провайдера контента. В метод `query` передается `uri` - путь к данным, `projection` - набор столбцов для извлечения, выражение выборки и параметры для него и название столбца, по которому проводится сортировка.

Метод возвращает курсор `Cursor`, который с помощью метода `moveToNext()` можно перебрать и получить отдельные данные. Метод `getColumnName()` возвращает название столбца, а `getString()` - собственно значение этого столбца:



Custom Content Provider in Android and Java

Получение одного объекта по id:

```

String[] projection = {
    FriendsContract.Columns._ID,
    FriendsContract.Columns.NAME,

```

```

        FriendsContract.Columns.EMAIL,
        FriendsContract.Columns.PHONE
    };
    ContentResolver contentResolver = getContentResolver();
    Cursor cursor = contentResolver.query(FriendsContract.buildFriendUri(2),
        projection, null, null, FriendsContract.Columns.NAME);
    if(cursor != null){
        while(cursor.moveToNext()){
            for(int i=0; i < cursor.getColumnCount(); i++){
                Log.d(TAG, cursor洗getColumnName(i) + " : " + cursor.getString(i));
            }
        }
        cursor.close();
    }

```

В данном случае получаем объект с `_id=2`.

Добавление данных

Добавление данных:

```

ContentResolver contentResolver = getContentResolver();
ContentValues values = new ContentValues();
values.put(FriendsContract.Columns.NAME, "Sam");
values.put(FriendsContract.Columns.EMAIL, "sam@gmail.com");
values.put(FriendsContract.Columns.PHONE, "+13676254985");
Uri uri = contentResolver.insert(FriendsContract.CONTENT_URI, values);

```


Для добавления применяется метод insert, который принимает путь URI и добавляемые данные в виде ContentValues.

Обновление данных

Обновление данных:

```
ContentResolver contentResolver = getContentResolver();
ContentValues values = new ContentValues();
values.put(FriendsContract.Columns.EMAIL, "sammy@gmail.com");
values.put(FriendsContract.Columns.PHONE, "+555555555555");
String selection = FriendsContract.Columns.NAME + " = 'Sam'";
int count = contentResolver.update(FriendsContract.CONTENT_URI, values,
selection, null);
```

В данном случае обновляются данные у всех объектов, у которых "Name=Sam". Критерий обновления передается через третий параметр.

Естественно с помощью выражения SQL можно задать любую логику выборки объектов для обновления. И для большего удобства мы можем вводить в него данные с помощью параметров, которые задаются знаком вопроса:

```
ContentResolver contentResolver = getContentResolver();
ContentValues values = new ContentValues();
values.put(FriendsContract.Columns.NAME, "Sam");
String selection = FriendsContract.Columns.NAME + " = ?";
String args[] = {"Sam Scromby"};
int count = contentResolver.update(FriendsContract.CONTENT_URI, values,
selection, args);
```

В этом случае с помощью четвертого параметра передается массив значений для параметров выражения выборки.

Но в примерах выше обновлялись все строки в бд, которые имели, например, имя "Sam". Но также можно обновлять и один объект по id. Например, обновим строку с `_id=3`:

```
ContentResolver contentResolver = getContentResolver();  
ContentValues values = new ContentValues();  
values.put(FriendsContract.Columns.NAME, "Sam");  
values.put(FriendsContract.Columns.EMAIL, "sam@gmail.com");  
int count = contentResolver.update(FriendsContract.buildFriendUri(3), values,  
null, null);
```

Удаление данных

Удаление данных по общему условию:

```
ContentResolver contentResolver = getContentResolver();  
String selection = FriendsContract.Columns.NAME + " = ?";  
String[] args = {"Sam"};  
int count = contentResolver.delete(FriendsContract.CONTENT_URI, selection,  
args);
```

В данном случае удаляются все строки, у которых `Name=Sam`.

Удаление по id:

```
ContentResolver contentResolver = getContentResolver();  
int count = contentResolver.delete(FriendsContract.buildFriendUri(2), null, null);
```

В данном случае удаляется строка с `_id=2`.

Асинхронная загрузка данных

В базе данных может быть много данных, и их загрузка может занять некоторое время. В этом случае можно воспользоваться асинхронной загрузкой данных. Для этого класс activity или фрагмента должен реализовать интерфейс **LoaderManager.LoaderCallbacks<Cursor>**.

Возьмем проект из прошлого материала, где реализован провайдер контента **AppProvider**, и изменим класс **MainActivity**:

```
package com.example.friendsproviderapp;
```

```
import androidx.annotation.NonNull;
```

```
import androidx.annotation.Nullable;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import androidx.loader.app.LoaderManager;
```

```
import androidx.loader.content.CursorLoader;
```

```
import androidx.loader.content.Loader;
```

```
import android.database.Cursor;
```

```
import android.os.Bundle;
```

```
import android.util.Log;
```

```
import java.security.InvalidParameterException;
```

```
public class MainActivity extends AppCompatActivity implements  
LoaderManager.LoaderCallbacks<Cursor> {
```

```
    private static final String TAG = "MainActivity";
```

```
    private static final int LOADER_ID = 225;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

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

```
        // setContentView(R.layout.activity_main);
```

```
// запускаем загрузку данных через провайдер контента
LoaderManager.getInstance(this).initLoader(LOADER_ID, null, this);
}
```

```
@NonNull
```

```
@Override
```

```
public Loader<Cursor> onCreateLoader(int id, @Nullable Bundle args) {
```

```
    String[] projection = {
```

```
        FriendsContract.Columns._ID,
```

```
        FriendsContract.Columns.NAME,
```

```
        FriendsContract.Columns.EMAIL,
```

```
        FriendsContract.Columns.PHONE
```

```
    };
```

```
    if(id == LOADER_ID)
```

```
        return new CursorLoader(this, FriendsContract.CONTENT_URI,
```

```
            projection,
```

```
            null,
```

```
            null,
```

```
            FriendsContract.Columns.NAME);
```

```
    else
```

```
        throw new InvalidParameterException("Invalid loader id");
```

```
}
```

```
@Override
```

```
public void onLoadFinished(@NonNull Loader<Cursor> loader, Cursor data) {
```

```
    if(data != null){
```

```
        Log.d(TAG, "count: " + data.getCount());
```

```
        // перебор элементов
```

```
        while(data.moveToNext()){
```

```

        for(int i=0; i < data.getColumnCount(); i++){
            Log.d(TAG, data洗getColumn洗Name(i) + " : " + data.getString(i));
        }
        Log.d(TAG, "=====");
    }
    data.close();
}
else{
    Log.d(TAG, "Cursor is null");
}
}

@Override
public void onLoaderReset(@NonNull Loader<Cursor> loader) {
    Log.d(TAG, "onLoaderReset...");
}
}

```

Интерфейс `LoaderManager.LoaderCallbacks<Cursor>` предполагает реализацию трех методов. Метод `onCreateLoader()` загружает курсор. В этот метод в качестве параметров передаются числовой код операции и объект `Bundle`. Числовой код передается при запуске загрузки курсора. В данном случае в качестве такого кода использует константа `LOADER_ID`.

В самом методе создается объект `CursorLoader`. В его конструктор передается несколько параметров:

- объект `Context` (текущая activity)
- набор столбцов, которые надо получить

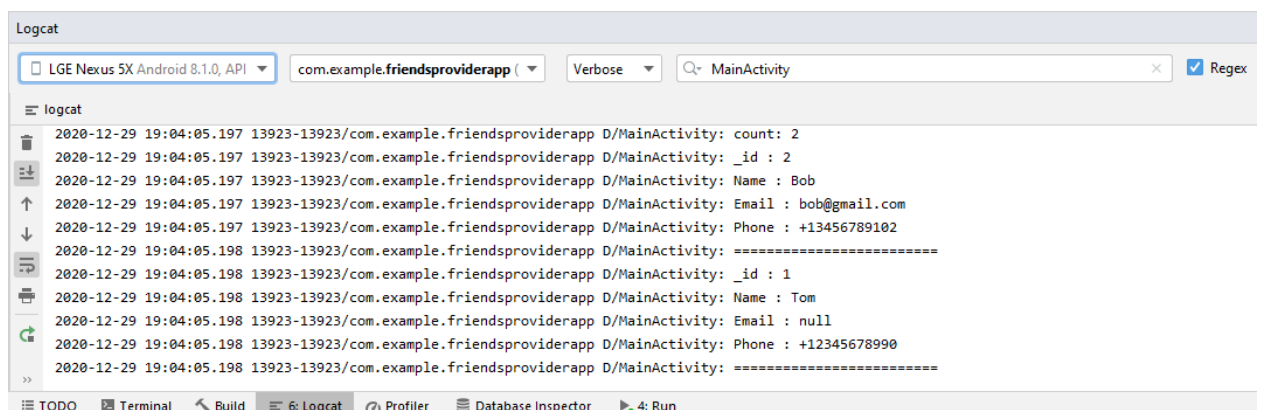
- выражение для выборки данных
- значения для параметров для выражения выборки
- столбец, по которому идет сортировка

Метод **onLoadFinished** вызывается при загрузке курсора. Через второй параметр мы можем собственно получить курсор и через него загруженные данные. И соответственно в этом методе мы можем перебрать курсор, получить данные и вывести их в элементах графического интерфейса или на консоль.

И метод **onLoaderReset** предназначен для сброса загрузчика.

Чтобы запустить загрузку данных, в методе **onCreate** вызывается метод **LoaderManager.getInstance(this).initLoader(LOADER_ID, null, this);**. Первый параметр - числовой код, а второй - объект **Bundle**. Это те значения, которые мы можем получить в методе **onCreateLoader**. И третий - объект **Context**.

В итоге при запуске **MainActivity** данные асинхронно будут загружены из базы данных:



```

Logcat
LGE Nexus 5X Android 8.1.0, API 26
com.example.friendsproviderapp (Verbose) MainActivity
2020-12-29 19:04:05.197 13923-13923/com.example.friendsproviderapp D/MainActivity: count: 2
2020-12-29 19:04:05.197 13923-13923/com.example.friendsproviderapp D/MainActivity: _id : 2
2020-12-29 19:04:05.197 13923-13923/com.example.friendsproviderapp D/MainActivity: Name : Bob
2020-12-29 19:04:05.197 13923-13923/com.example.friendsproviderapp D/MainActivity: Email : bob@gmail.com
2020-12-29 19:04:05.197 13923-13923/com.example.friendsproviderapp D/MainActivity: Phone : +13456789102
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: =====
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: _id : 1
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: Name : Tom
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: Email : null
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: Phone : +12345678990
2020-12-29 19:04:05.198 13923-13923/com.example.friendsproviderapp D/MainActivity: =====
  
```

Content Provider in Android and Java

Работа с json

Для работы с форматом json нет встроенных средств, но есть куча библиотек и пакетов, которые можно использовать для данной цели. Одним из наиболее популярных из них является пакет `com.google.code.gson`.

Для его использования в проекте Android в файл `build.gradle`, который относится к модулю `app`, в секцию `dependencies` необходимо добавить соответствующую зависимость:

```
implementation 'com.google.code.gson:gson:2.8.8'
```

То есть после добавления секция зависимостей `dependencies` в файле **build.gradle** может выглядеть следующим образом:

```
dependencies {  
  
    implementation 'com.google.code.gson:gson:2.8.6'  
  
    implementation 'androidx.appcompat:appcompat:1.2.0'  
    implementation 'com.google.android.material:material:1.2.1'  
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  
}
```

После добавления пакета в проект добавим новый класс `User`, который будет представлять данные:

```
package com.example.filesapp;
```

```
public class User {
```

```
private String name;
```

```
private int age;
```

```
User(String name, int age){
```

```
    this.name = name;
```

```
    this.age = age;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
public int getAge() {
```

```
    return age;
```

```
}
```

```
public void setAge(int age) {
```

```
    this.age = age;
```

```
}
```

```
@Override
```

```
public String toString(){
```

```
    return "Имя: " + name + " Возраст: " + age;
```

```
}
```

```
}
```


Объекты этого класса мы будем сериализовать в формат json и наоборот десериализовать из файла.

Для работы с json добавим следующий класс **JSONHelper**:

```
package com.example.filesapp;
```

```
import android.content.Context;
```

```
import com.google.gson.Gson;
```

```
import java.io.FileInputStream;
```

```
import java.io.FileOutputStream;
```

```
import java.io.IOException;
```

```
import java.io.InputStreamReader;
```

```
import java.util.List;
```

```
class JSONHelper {
```

```
    private static final String FILE_NAME = "data.json";
```

```
    static boolean exportToJSON(Context context, List<User> dataList) {
```

```
        Gson gson = new Gson();
```

```
        DataItems dataItems = new DataItems();
```

```
        dataItems.setUsers(dataList);
```

```
        String jsonString = gson.toJson(dataItems);
```

```
        try(FileOutputStream fileOutputStream =
```

```
            context.openFileOutput(FILE_NAME, Context.MODE_PRIVATE)) {
```

```
            fileOutputStream.write(jsonString.getBytes());
```

```
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
        return false;
    }
```

```
static List<User> importFromJSON(Context context) {

    try(FileInputStream fileInputStream = context.openFileInput(FILE_NAME);
        InputStreamReader streamReader = new
        InputStreamReader(fileInputStream)){

        Gson gson = new Gson();
        DataItems dataItems = gson.fromJson(streamReader, DataItems.class);
        return dataItems.getUsers();
    }
    catch (IOException ex){
        ex.printStackTrace();
    }

    return null;
}
```

```
private static class DataItems {
    private List<User> users;

    List<User> getUsers() {
```

```

        return users;
    }

    void setUsers(List<User> users) {

        this.users = users;
    }
}
}

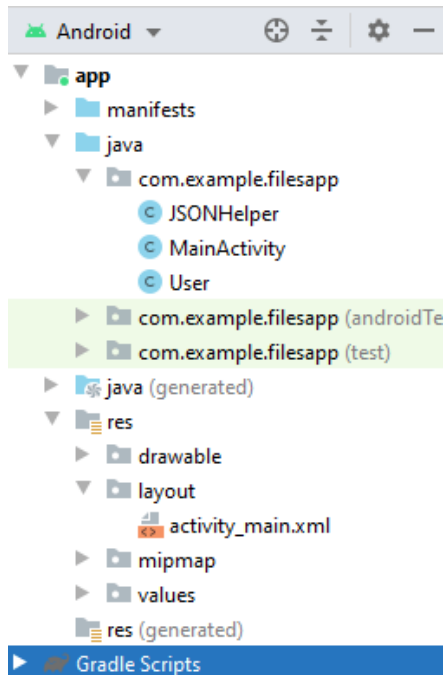
```

Для работы с json создается объект **Gson**. Для сериализации данных в формат json у этого объекта вызывается метод **toJson()**, в который передаются сериализуемые данные.

Для упрощения работы с данными применяется вспомогательный класс **DataItems**. На выходе метод **toJson()** возвращает строку, которая затем сохраняется в текстовый файл.

Для десериализации выполняется метод **fromJson()**, в который передается объект **Reader** с сериализованными данными и тип, к которому надо десериализовать данные.

В итоге проект будет выглядеть так:



com.google.code.gson и toJson и fromJson в Android и Java

Теперь определим основной функционал для взаимодействия с пользователем. Изменим файл **activity_main.xml** следующим образом:

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

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/nameText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите имя"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toTopOf="@id/ageText"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/ageText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите возраст"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toTopOf="@id/addButton"
        app:layout_constraintTop_toBottomOf="@id/nameText" />

    <Button
```

```
android:id="@+id/addButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Добавить"
android:onClick="addUser"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@id/saveButton"
app:layout_constraintTop_toBottomOf="@id/ageText" />
```

<Button

```
android:id="@+id/saveButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Сохранить"
android:onClick="save"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toLeftOf="@id/openButton"
app:layout_constraintBottom_toTopOf="@id/list"
app:layout_constraintTop_toBottomOf="@id/addButton"/>
```

<Button

```
android:id="@+id/openButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Открыть"
android:onClick="open"
app:layout_constraintLeft_toRightOf="@id/saveButton"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@id/list"
app:layout_constraintTop_toBottomOf="@id/addButton"/>
```

```

<ListView
    android:id="@+id/list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toBottomOf="@id/openButton" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Здесь определены два текстовых поля для ввода названия модели и цены объекта User и одна кнопка для добавления данных в список. Еще одна кнопка выполняет сериализацию данных из списка в файл, а третья кнопка - восстановление данных из файла.

Для вывода сами данных определен элемент ListView.

И изменим класс **MainActivity**:

```

package com.example.filesapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;

```

```
import java.util.List;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    private ArrayAdapter<User> adapter;
```

```
    private EditText nameText, ageText;
```

```
    private List<User> users;
```

```
    ListView listView;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

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

```
        setContentView(R.layout.activity_main);
```

```
        nameText = findViewById(R.id.nameText);
```

```
        ageText = findViewById(R.id.ageText);
```

```
        listView = findViewById(R.id.list);
```

```
        users = new ArrayList<>();
```

```
        adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,  
users);
```

```
        listView.setAdapter(adapter);
```

```
    }
```

```
    public void addUser(View view){
```

```
        String name = nameText.getText().toString();
```

```
        int age = Integer.parseInt(ageText.getText().toString());
```

```
        User user = new User(name, age);
```

```
        users.add(user);
```

```
        adapter.notifyDataSetChanged();
```

```
}

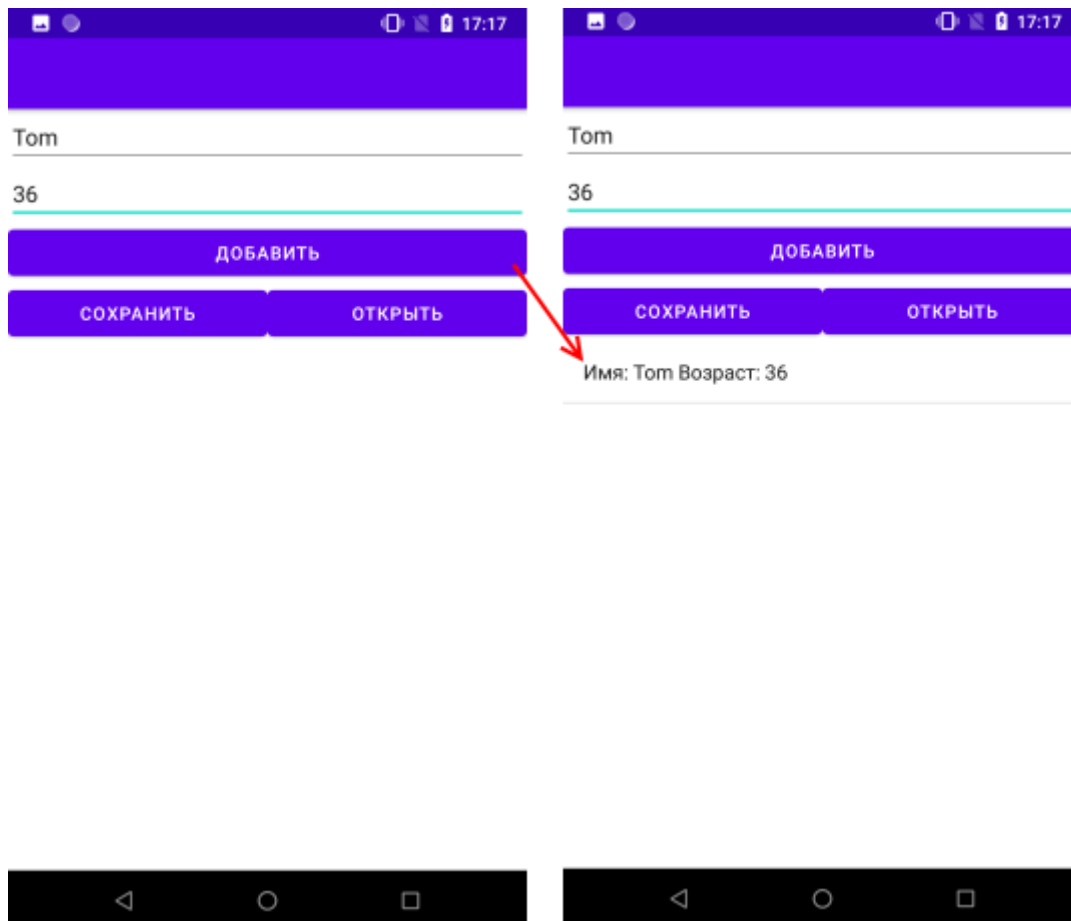
public void save(View view){

    boolean result = JSONHelper.exportToJSON(this, users);
    if(result){
        Toast.makeText(this, "Данные сохранены",
Toast.LENGTH_LONG).show();
    }
    else{
        Toast.makeText(this, "Не удалось сохранить данные",
Toast.LENGTH_LONG).show();
    }
}

public void open(View view){
    users = JSONHelper.importFromJSON(this);
    if(users!=null){
        adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
users);
        listView.setAdapter(adapter);
        Toast.makeText(this, "Данные восстановлены",
Toast.LENGTH_LONG).show();
    }
    else{
        Toast.makeText(this, "Не удалось открыть данные",
Toast.LENGTH_LONG).show();
    }
}
}
```


Все данные находятся в списке users, который представляет объект List<User>. Через адаптер этот список связывается с ListView.

Для сохранения и восстановления данных вызываются ранее определенные методы в классе JSONHelper. Кнопка добавления добавляет данные в список user, и они сразу же отображаются в ListView. При нажатии на кнопку сохранения данные из списка users сохраняются в локальный файл. Затем с помощью кнопки открытия мы сможем открыть ранее сохраненный файл.



Сериализация и десериализация в JSON в Android и Java

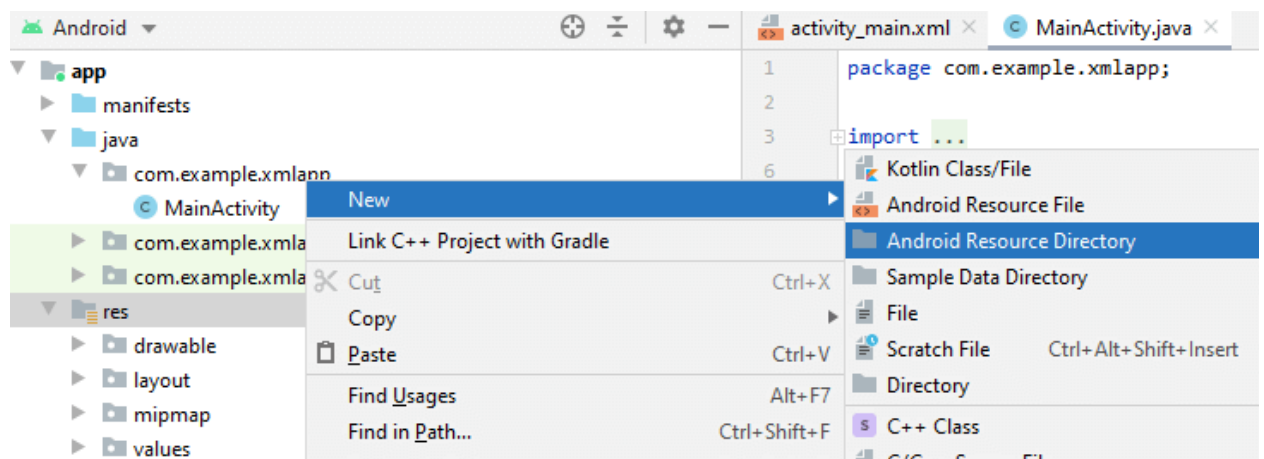
Работа с XML

Ресурсы XML и их парсинг

Одним из распространенных форматов хранения и передачи данных является xml. Рассмотрим, как с ним работать в приложении на Android.

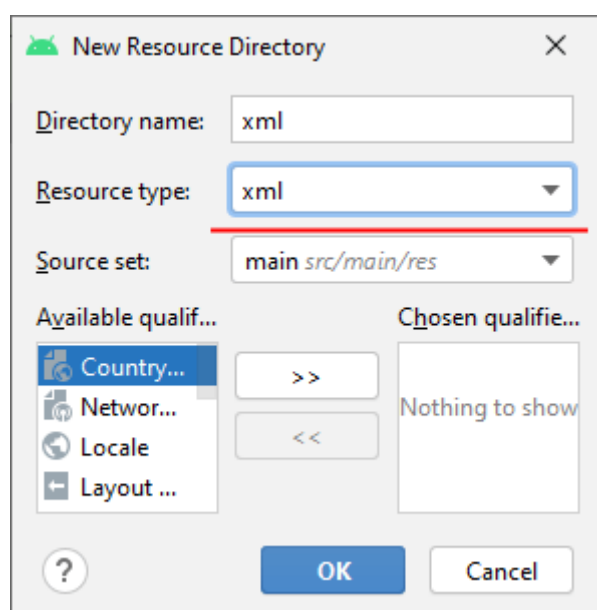
Приложение может получать данные в формате xml различными способами - из ресурсов, из сети и т.д. В данном случае рассмотрим ситуацию, когда файл xml хранится в ресурсах.

Возьмем стандартный проект Android по умолчанию и в папке **res** создадим каталог **xml**. Для этого нажмем на каталог res правой кнопкой мыши и в контекстном меню выберем **New -> Android Resource Directory**:



Добавление папки xml в Android Studio

В появившемся окне в качестве типа ресурсов укажем **xml**:



Добавление папки ресурсов xml в Android Studio

В этот каталог добавим новый файл, который назовем users.xml и который будет иметь следующее содержимое:

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

<users>

    <user>

        <name>Tom</name>

        <age>36</age>

    </user>

    <user>

        <name>Alice</name>

        <age>32</age>

    </user>

    <user>

        <name>Bob</name>

        <age>28</age>

    </user>

</users>
```

Это обычный файл xml, который хранит набор элементов user. Каждый элемент характеризуется наличием двух под элементов - name и age. Условно говоря, каждый элемент описывает пользователя, у которого есть имя и возраст.

В папку, где находится основной класс MainActivity, добавим новый класс, который назовем **User**:

```
package com.example.xmlapp;
```

```
public class User {

    private String name;

    private String age;
```

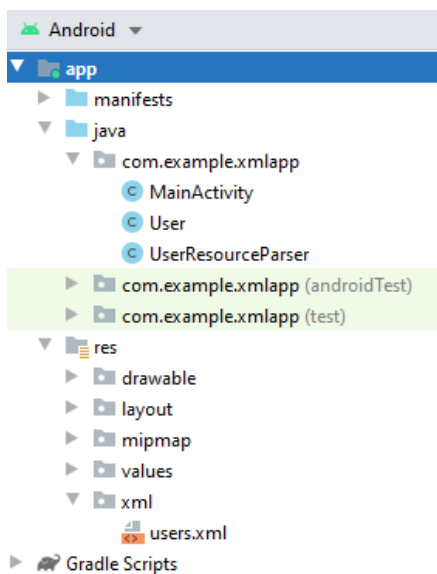
```

public String getName(){
    return name;
}
public String getAge(){
    return age;
}
public void setName(String name){
    this.name = name;
}
public void setAge(String age){
    this.age = age;
}
public String toString(){
    return "User: " + name + " - " + age;
}
}

```

Этот класс описывает товар, информация о котором будет извлекаться из xml-файла.

И в ту же папку добавим новый класс `UserResourceParser`:



XML-ресурсы в Android Studio и их парсинг с помощью Java

Определим для класса UserResourceParser следующий код:

```
package com.example.xmlapp;
```

```
import org.xmlpull.v1.XmlPullParser;
```

```
import java.util.ArrayList;
```

```
public class UserResourceParser {
```

```
    private ArrayList<User> users;
```

```
    public UserResourceParser(){
```

```
        users = new ArrayList<>();
```

```
    }
```

```
    public ArrayList<User> getUsers(){
```

```
        return users;
```

```
    }
```

```
    public boolean parse(XmlPullParser xpp){
```

```
        boolean status = true;
```

```
        User currentUser = null;
```

```
        boolean inEntry = false;
```

```
        String textValue = "";
```

```
        try{
```

```
            int eventType = xpp.getEventType();
```

```
            while(eventType != XmlPullParser.END_DOCUMENT){
```

```

String tagName = xpp.getName();
switch (eventType){
    case XmlPullParser.START_TAG:
        if("user".equalsIgnoreCase(tagName)){
            inEntry = true;
            currentUser = new User();
        }
        break;
    case XmlPullParser.TEXT:
        textValue = xpp.getText();
        break;
    case XmlPullParser.END_TAG:
        if(inEntry){
            if("user".equalsIgnoreCase(tagName)){
                users.add(currentUser);
                inEntry = false;
            } else if("name".equalsIgnoreCase(tagName)){
                currentUser.setName(textValue);
            } else if("age".equalsIgnoreCase(tagName)){
                currentUser.setAge(textValue);
            }
        }
        break;
    default:
}
eventType = xpp.next();
}
}
catch (Exception e){

```

```

        status = false;

        e.printStackTrace();
    }

    return status;
}
}

```

Данный класс выполняет функции парсинга xml. Распарсенные данные будут храниться в переменной users. Непосредственно сам парсинг осуществляется с помощью функции parse. Основную работу выполняет передаваемый в качестве параметра объект **XmlPullParser**. Этот класс позволяет пробежаться по всему документу xml и получить его содержимое.

Когда данный объект проходит по документу xml, при обнаружении определенного тега он генерирует некоторое событие. Есть четыре события, которые описываются следующими константами:

START_TAG: открывающий тег элемента

TEXT: прочитан текст элемента

END_TAG: закрывающий тег элемента

END_DOCUMENT: конец документа

С помощью метода `getEventType()` можно получить первое событие и потом последовательно считывать документ, пока не дойдем до его конца. Когда будет достигнут конец документа, то событие будет представлять константу **END_DOCUMENT**:

```

int eventType = xpp.getEventType();
while(eventType != XmlPullParser.END_DOCUMENT){

```

```
//.....  
eventType = xpp.next();  
}
```

Для перехода к следующему событию применяется метод `next()`.

При чтении документа с помощью метода `getName()` можно получить название считываемого элемента.

```
String tagName = xpp.getName();
```

И в зависимости от названия тега и события мы можем выполнить определенные действия. Например, если это открывающий тег элемента `user`, то создаем новый объект `User` и устанавливаем, что мы находимся внутри элемента `user`:

```
case XmlPullParser.START_TAG:  
    if("user".equalsIgnoreCase(tagName)){  
        inEntry = true;  
        currentUser = new User();  
    }  
break;
```

Если событие `TEXT`, то считано содержимое элемента, которое мы можем прочитать с помощью метода **`getText()`**:

```
case XmlPullParser.TEXT:  
    textValue = xpp.getText();  
    break;
```

Если закрывающий тег, то все зависит от того, какой элемент прочитан. Если прочитан элемент `user`, то добавляем объект `User` в коллекцию `ArrayList` и сбрасываем переменную `inEntry`, указывая, что мы вышли из элемента **`user`**:

```
case XmlPullParser.END_TAG:  
    if(inEntry){
```



```
if("user".equalsIgnoreCase(tagName)){  
    users.add(currentUser);  
    inEntry = false;
```

Если прочитаны элементы name и age, то передаем их значения переменным name и age объекта User:

```
else if("name".equalsIgnoreCase(tagName)){  
    currentUser.setName(textValue);  
} else if("age".equalsIgnoreCase(tagName)){  
    currentUser.setAge(textValue);  
}
```

Теперь изменим класс MainActivity, который будет загружать ресурс xml:

```
package com.example.xmlapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.util.Log;
```

```
import org.xmlpull.v1.XmlPullParser;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

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

```
        setContentView(R.layout.activity_main);
```

```

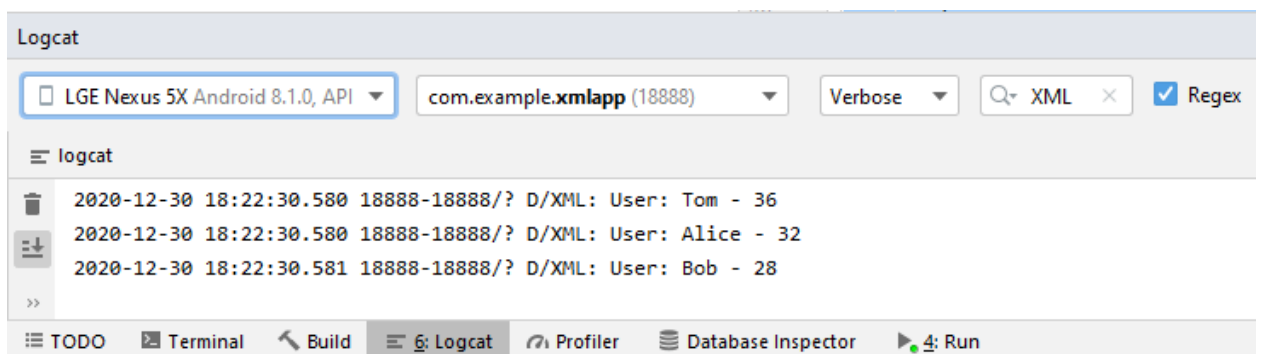
XmlPullParser xpp = getResources().getXml(R.xml.users);

UserResourceParser parser = new UserResourceParser();

if(parser.parse(xpp))
{
    for(User prod: parser.getUsers()){
        Log.d("XML", prod.toString());
    }
}
}
}

```

Вначале получаем ресурс xml с помощью метода **getXml()**, в который передается название ресурса. Данный метод возвращает объект **XmlPullParser**, который затем используется для парсинга. Для простоты выводим данные в окне **Logcat**:



Получение xml по сети

Рассмотрим получение данных в формате xml по сети. Допустим, на некотором сайте <https://example.com> находится файл **users.xml** со следующим содержимым:

```

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

<users>

  <user>

    <name>Tom</name>

```

```
        <age>36</age>
    </user>
    <user>
        <name>Alice</name>
        <age>32</age>
    </user>
    <user>
        <name>Bob</name>
        <age>28</age>
    </user>
</users>
```

То есть сам файл доступен по адресу <https://example.com/users.xml>. Но это необязательно должен быть именно файл, это может быть любой ресурс, который динамически генерирует данные в xml.

Возьмем стандартный проект Android и вначале определим в нем класс User, который будет представлять загружаемые данные:

```
package com.example.xmlapp;

public class User {
    private String name;
    private String age;

    public String getName(){
        return name;
    }
    public String getAge(){
        return age;
    }
    public void setName(String name){
```

```

        this.name = name;
    }
    public void setAge(String age){
        this.age = age;
    }
    public String toString(){
        return "User: " + name + " - " + age;
    }
}

```

Далее определим класс UserXmlParser:

```

package com.example.xmlapp;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;
import java.util.ArrayList;
import java.io.StringReader;

public class UserXmlParser {

    private ArrayList<User> users;

    public UserXmlParser(){
        users = new ArrayList<>();
    }

    public ArrayList<User> getUsers(){
        return users;
    }
}

```

```
public boolean parse(String xmlData){
    boolean status = true;
    User currentUser = null;
    boolean inEntry = false;
    String textValue = "";

    try{
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
        factory.setNamespaceAware(true);
        XmlPullParser xpp = factory.newPullParser();

        xpp.setInput(new StringReader(xmlData));
        int eventType = xpp.getEventType();
        while(eventType != XmlPullParser.END_DOCUMENT){

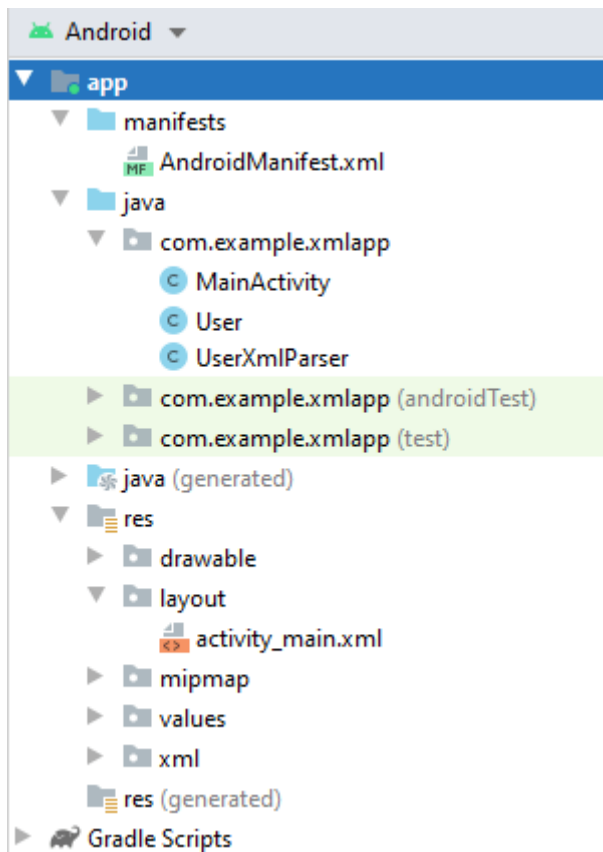
            String tagName = xpp.getName();
            switch (eventType){
                case XmlPullParser.START_TAG:
                    if("user".equalsIgnoreCase(tagName)){
                        inEntry = true;
                        currentUser = new User();
                    }
                    break;
                case XmlPullParser.TEXT:
                    textValue = xpp.getText();
                    break;
                case XmlPullParser.END_TAG:
                    if(inEntry){
```

```

        if("user".equalsIgnoreCase(tagName)){
            users.add(currentUser);
            inEntry = false;
        } else if("name".equalsIgnoreCase(tagName)){
            currentUser.setName(textValue);
        } else if("age".equalsIgnoreCase(tagName)){
            currentUser.setAge(textValue);
        }
    }
    break;
default:
}
eventType = xpp.next();
}
}
catch (Exception e){
    status = false;
    e.printStackTrace();
}
return status;
}
}

```

То есть в итоге получится следующий проект:



Парсинг xml из сети в Android

Для парсинга xml здесь используется класс **XmlPullParser**, который уже рассматривался в прошлой теме. Единственное отличие заключается в том, что для создания объекта этого класса применяется класс **XmlPullParserFactory**:

```
XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
```

```
XmlPullParser xpp = factory.newPullParser();
```

Для работы определим простейший визуальный интерфейс в файле **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<androidx.constraintlayout.widget.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >
```

```
<TextView
    android:id="@+id/contentView"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toTopOf="@id/usersList"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<ListView
    android:id="@+id/usersList"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/contentView" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определен элемент `TextView` для отображения некоторой дополнительной информации о состоянии загрузки файла и элемент `ListView` для отображения загруженных объектов.

Далее изменим класс **MainActivity**:

```
package com.example.xmlapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
```



```

import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;

import javax.net.ssl.HttpURLConnection;

public class MainActivity extends AppCompatActivity {

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

        ListView usersList = findViewById(R.id.usersList);
        TextView contentView = findViewById(R.id.contentView);

        contentView.setText("Загрузка...");

        new Thread(new Runnable() {
            public void run() {
                try{
                    String content = download("https://example.com/users.xml");
                    usersList.post(new Runnable() {
                        public void run() {

```

```

        UserXmlParser parser = new UserXmlParser();
        if(parser.parse(content))
        {
            ArrayAdapter<User> adapter = new
ArrayAdapter(getBaseContext(),
                android.R.layout.simple_list_item_1, parser.getUsers());
            usersList.setAdapter(adapter);
            contentView.setText("Загружено объектов: " +
adapter.getCount());
        }
    });
}

catch (IOException ex){
    contentView.post(new Runnable() {
        public void run() {
            contentView.setText("Ошибка: " + ex.getMessage());
        }
    });
}

}).start();
}

```

```

private String download(String urlPath) throws IOException{
    StringBuilder xmlResult = new StringBuilder();
    BufferedReader reader = null;
    InputStream stream = null;
    HttpURLConnection connection = null;

```

```

try {
    URL url = new URL(urlPath);
    connection = (HttpsURLConnection) url.openConnection();
    stream = connection.getInputStream();
    reader = new BufferedReader(new InputStreamReader(stream));
    String line;
    while ((line=reader.readLine()) != null) {
        xmlResult.append(line);
    }
    return xmlResult.toString();
} finally {
    if (reader != null) {
        reader.close();
    }
    if (stream != null) {
        stream.close();
    }
    if (connection != null) {
        connection.disconnect();
    }
}
}
}

```

При создании MainActivity будет запускаться дополнительный поток, который вызывает метод download(). Этот метод с помощью класса HttpsURLConnection загружает файл users.xml и возвращает его содержимое в виде строки (Если необходимо загрузить файл xml по протоколу http, то вместо применяется класса HttpsURLConnection класс java.net.HttpURLConnection).

```
String content = download("https://example.com/users.xml");
```

Затем загруженное содержимое передается в метод `parse()`, класса `UserXmlParser`, который формирует список объектов.

```
UserXmlParser parser = new UserXmlParser();  
if(parser.parse(content)){  
    //.....
```

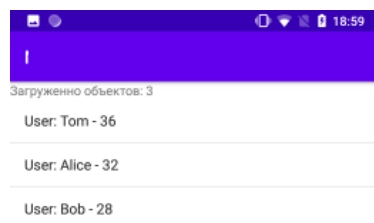
Затем загруженный список передается в адаптер `ArrayAdapter`, а через него в `ListView` для отображения на экране устройства:

```
ArrayAdapter<User> adapter = new ArrayAdapter(getApplicationContext(),  
android.R.layout.simple_list_item_1, parser.getUsers());  
usersList.setAdapter(adapter);
```

В завершении надо добавить в файл манифеста **AndroidManifest.xml** разрешения на взаимодействие с сетью:

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

И после запуска приложения в окне Logcat мы увидим полученные с сервера данные:



`XmlPullParser` и парсинг xml-файла и `HttpsURLConnection` в Android и Java

Задание

1. Реализовать небольшой проект работы с контактами. Добавление контактов.
2. Реализовать создание провайдера контента, чтобы другие приложения могли обращаться к данным нашего приложения через некоторый API. Определение контракта через класс `FriendsContract`.
3. Реализовать получение данных в провайдере используя метод `query()`.
4. Реализовать добавление данных, применяя метод `insert()`
5. Реализовать удаление данных, применяя метод `delete()`
6. Реализовать обновление данных, применяя метод `update()`
7. Для работы провайдера контента, реализовать внесение соответствующих изменений в файл **AndroidManifest.xml**
8. Реализовать применение провайдера контента. Получение данных. Добавление данных. Обновление данных. Удаление данных.
9. Реализовать асинхронную загрузку данных
10. Реализовать пример работы с json.
11. Реализовать пример работы с XML