Второй курс, осенний семестр 2016/17
Конспект семинаров по программированию под Android

Собрано 17 октября 2016 г. в 06:35

Содержание

1. Настройки **1**
- 1.1. Цели . . . 1
- 1.2. Class Preference . . . 1
- 1.3. Задание Preferences в виде XML-файла . . . 1
- 1.4. Разные наследники Preference . . . 2
- 1.5. dependency . . . 3
- 1.6. PreferenceCategory, дополнительные экраны, открывание ссылок . . . 3
- 1.7. Отображение настроек из активити. PreferenceFragment . . . 5
- 1.8. Чтение настроек . . . 5
- 1.9. Изменение настроек из кода . . . 6

2. SQLite **7**
- 2.1. SQLite . . . 7
- 2.2. Подключение к базе данных . . . 7
- 2.3. Создание таблицы . . . 7
- 2.4. Запрос INSERT . . . 8
- 2.5. Запрос SELECT . . . 9

3. AsyncTask **10**

4. JSON **11**
- 4.1. Формат . . . 11
- 4.2. JSON Scheme . . . 12
- 4.3. JSON в Java . . . 13
- 4.4. Не используйте JSON . . . 14

Настройки

16 октября

1.1. Цели

В любом приложении есть какие-то параметры, которые зависят от предпочтений пользователя (банально, включить/выключить звук, убрать уведомления и подобное). Поэтому всегда нужно иметь некоторые настройки. Дальше пойдет речь о некоторых способах организовать их в своем приложении.

1.2. Class Preference

Все настройки в соответствии с логикой разбиваются на отдельные маленькие пункты (например, одна настройка отвечает за громкость звука, вторая отвечает за имя пользователя и т.п). Каждая такая отдельная маленькая "настройка"будет в итоге представлена экземпляром класса Prefernce или его наследника. У этого класса внутри есть множество XML-аттрибутов, среди них самые основные:

- android:key - это ключ-идентификатор для вашей настройки. Далее в коде, когда вам понадобится доступ к этой настройке, вы будете ее искать по этому ключу.
- android:title - это тот текст, который отображается на экране (то есть это то, как вы хотите, чтобы называлась ваша настройка)
- android:defaultValue - значение по умолчанию. Оно может быть типа Boolean, Integer, String, Float (может быть, еще что-нибудь, можете посмотреть).
- android:summary - какое-то описание этой настройки для вас или для читающего код (например, описание, за что она отвечает)

Также у этого класса есть куча методов. Например, есть set* и get* для каждого аттрибута (например, setKey() и getKey()).

Немного размышлений. В общем, этот класс просто хранит всю информацию про настройку, а также обеспечивает методы для работы с ним. Важно понимать, что этот класс хранит не просто настройку, а именно настройку, которая будет отображена на экране. Другими словами, это не просто абстрактная настройка, у которой есть имя и возможности выбора, а всякая функциональность для взаимодействия с Intent-ами и Fragment-ами т.д. Вам скорее всего не придется об этом думать, вы просто зададите параметры и остальная работа пройдет внутри. Честно говоря, вы даже с самим классом Preference в java-коде работать не будете, но зато примерно понимаете, что внутри.

1.3. Задание Preferences в виде XML-файла

Для создания настроек есть больее удобный способ, а именно объявление их в XML-файле. Создадим документ res/xml/preferences.xml в нашем проекте (такие названия не обязательны, но принято называть так). Там напишем следующий код:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3     <CheckBoxPreference
```

```
4         android:key="sound"
5         android:title="Enable/disable sound"
6         android:summary="this preference is for sound switching-on/off"
7         android:defaultValue="true" />
8 </PreferenceScreen>
```

Давайте разберемся, что тут происходит. Первая строчка задает формат документа, ее, кажется, можно вообще не писать, вообщем, она не очень интересная. Оставшийся блок задает PreferenceScreen. Что это такое? На самом деле, это очередной класс, которым пользуется активити для отображения ваших настроек на экран. Опять же, можно его задать прям в коде, но куда удобнее все делать в xml-файле. Итак, мы все наши настройки обернули в PreferenceScreen, поэтому все настройки, записанные внутри него, будут отображаться последовательными блоками на экране (за это и отвечает PreferenceScreen).

В объявлении Preference screen также есть загадочная строчка "xmlns:android=..". Она нужна, чтобы писать не "http:/../res/android:key а просто "android:key". Очень похоже на using namespace.

Теперь перейдем внутрь тела PreferenceScreen. По формату легко видеть, что дальше идет объявление CheckBoxPreference. Это класс-наследник Preference, который отображается на экране как настройка, у которой бывает два состояния - включена и выключена. Обычно это сделана в виде белого квадратика, в который можно поставить галочку (о других видах настроек чуть позже). А внутри объявления этой настройки мы просто задаем её атрибуты. Легко видеть, что не обязательно задавать все, которые бывают, можете задать только нужные. Можно вообще ничего не задавать, но тогда придется потом делать это в коде, не очень понятно, зачем это вам нужно.

Обратите внимание, что атрибут defaultValue я задаю именно как строчку. Дальше он уже сам разбирается, что вы хотели Boolean, но так как атрибут - это строка, то надо писать именно "true".

1.4. Разные наследники Preference

Обратим внимание еще на несколько стандартных видов настроек. Рассмотрим следующий пример:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3    <CheckBoxPreference
4         android:key="sound"
5         android:title="Enable/disable sound"
6         android:summary="this preference is for sound switching-on/off"
7         android:defaultValue="true" />
8      <ListPreference
9         android:key="level"
10        android:title="difficulty level"
11        android:dialogTitle="@string/listDialogTitle"
12        android:entries="@array/entries"
13        android:entryValues="@array/entry_values"
14        android:dependency="sound"
15     <EditTextPreference
16        android:key="name"
17        android:title="your name"
18        android:dialogTitle="Type your name here" />
19 </PreferenceScreen>
```

Начнем с EditTextPreference, с ним попроще. Как нетрудно догадаться из названия, эта настройка, где вы можете задать какую-то строку. Соответсвенно, по нажатию на нее, у вас вылезает маленькое окошко, где можно набрать текст (например, как когда вы набираете пароль для сети). Из неочевидного - title отвечает за название в списке настроек, а dialogTitle - это заголовок открывающегося окошка.

ListPreference подразумевает собой список вариантов, из которых можно что-нибудь выбрать. Соответсвенно, по нажатию на нее, у вас вылезает маленькое окошко, где виден список доступных выборов. Где-то надо задать список доступных выборов, и какие значения мы хотим, чтобы им соответствовали. Так как это константные списки строк, рекомендуется объявить их в res/values/strings.xml - это файл, в котором рекомендуется объявлять все константные строки. Скорее всего, этот файл уже есть в вашем проекте, там задана константа "app_name". Так выглядит мой файл:

```
<resources>
    <string name="app_name">NextTest</string>
    <string name="listDialogTitle">Choose difficulty level</string>
    <string-array name="entries">
        <item>one</item>
        <item>two</item>
        <item>three</item>
    </string-array>
    <string-array name="entry_values">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </string-array>
</resources>
```

Нетрудно разобраться, что тут происходит. Если вы хотите задать простую строковую константу, это делается в виде «string name = "name_of_const»value of const</string>". Далее в коде, когда вы хотите обратиться к этой константе, можно просто писать "@string/name_of_const". Например, для атрибута dialogTitle я сделал именно так. В похожем синтаксисе объявляются массивы строк. Думаю, тут все и так понятно. entries - это список опций, которые будут доступны на экране, а entry_values - это те значения, которые соответсвуют этим опциям. Соответсвенно, у ListPreference есть метод getValue(), который и возвращает значение, соответствующее выбранной опции.

1.5. dependency

Можете заметить, что у ListPreference я указал атрибут dependency, значение которого равно ключу CheckBoxPreference. Теперь этот лист открывается пользователю, только если в CheckBoxPreference поставлена галочка (то есть он затемнен и на него нельзя нажать, пока галочка не поставлена). Как это работает с другими, я не особо разбирался.

1.6. PreferenceCategory, дополнительные экраны, открывание ссылок

Если вам хочется, чтобы по нажатию на какой-то пункт настроек, открывалось новое полноценное меню с настройками, можете просто сделать еще один PreferenceScreen внутри уже имеющегося PreferenceScreen:

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
4
5      <PreferenceScreen
6          android:key="A"
7          android:title="B">
8
9          <EditTextPreference
10             android:key="PREF_EDITTEXT"
11             android:title="C"
12             android:summary="EditText summary"
13             android:dialogTitle="EditText Dialog" />
14
15         <PreferenceScreen
16             android:key="button_voicemail_setting_key"
17             android:title="XXX">
18             <Preference
19                 android:key="a"
20                 android:title="b"
21                 >
22             <intent android:action="android.intent.action.VIEW"
23                 android:data="http://www.example.com" />
24             </Preference>
25
26         </PreferenceScreen>
27
28     </PreferenceScreen>
29
30 </PreferenceScreen>
```

Сейчас пользователь увидит следующее: изначально ему откроется экран с единтсвенной опцией "B". Когда он нажимает на нее, открываencz новые экран настроек. На нем есть две кнопки "C"(открывает EditTextPref) и "XXX которая снова открывает новый экран. На новом экране есть одна кнопка "b на которую можно нажать и открыть веб-страницу (я не разбирался, как это работает открытие веб-страниц и что за атрибут дата, можете посмотреть, если интересует).

И еще один момент. Некоторые настройки могут быть по смыслу объединены в отдельный блок и вы хотите это подчеркнуть. Для этого существует PreferenceCategory. Синтаксис точно такой же, как и всех классов до этого:

```
1      <PreferenceCategory
2          android:title="Stolovye priborsu"
3          android:key="pribors">
4
5          <Preference
6              android:key="pref_key_sms_delete_limit"
7              android:title="lozki" />
8          <Preference
9              android:key="pref_key_mms_delete_limit"
10             android:title="vilki" />
11     </PreferenceCategory>
```

Если запустите, то увидете, что у вас эти две настройки будут отделены от остальных разделителями.

1.7. Отображение настроек из активити. PreferenceFragment

Вот мы описали экран с настройками. Теперь надо как-то из активити вызвать этот экран. Раньше для этого нужно было создавать наследника класса PreferenceActivity, и дальше там что-то с ним делать. Сейчас, судя по недвусмысленным замечаниям в документации, так делать не рекомендуется. Вообще есть такие объекты, как фрагменты. Это такие мини-актвити или подактивити. Они ведут себя как activity, но по смыслу это отдельные маленькие кусочки активити. Вы можете сделать несколько фрагментов на одном активити, и они будут жить каждый по-своему. Одно из важнейших преимуществ фрагментов в том, что вы можете переиспользовать один и тот же фрагмент в разных актвити. Например, наши настройки - как раз такой пример. Не будем создавать новую activity, а просто сделаем фрагмент в текущей, который будет отвечать за настройки. (Честно говоря, я еще сам не до конца осознал, какой там философский смысл у фрагментов, но раз рекомендуется пользоваться, попытался примерно пояснить). Все, что нам нужно, это создать наследника класса PreferenceFragment:

```
1 public class SettingsFragment extends PreferenceFragment {
2     @Override
3     public void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5 
6         // Load the preferences from an XML resource
7         addPreferencesFromResource(R.xml.preferences);
8     }
9 }
```

Метод addPreferencesFromResource добавляет все настройки из ресурса. Также есть метод addPreferencesFromIntent, который добавляет все настройки из Intent-а (если вам это, в отличие от меня, о чем-то говорит).

1.8. Чтение настроек

Довольно очевидно, что вы хотите в любом месте вашей программы получить доступ к текущим настройкам. Для этого существует два класса: PreferenceManager и SharedPreferences. Далее будут идти наше предположения о том, как это все работает внутри, но методы точно приведут к нужному результату, поэтому если не хотите думать о том, как все устроено, можете просто пользоваться кодом. Итак, SharedPreference - это интерфейс для хранения всех preferenceов (на самом деле где-то внутри есть отдельный файл, который описывает все настройки и SharedPreferences с ним общается). PreferenceManager - это класс для управления preferenceами. Изначально у нас в приложении есть пустой PreferenceManager, то есть в нем нет никаких настроек. Чтобы загрузить в него настройки, нужно вызвать метод setDefaultValues(context, xml-file, bool), где context - это скорее всего просто ваше активити, xml-file - это тот файл, из которого вы хотите забрать настройки (в нашем случае это R.xml.preferences), a bool - это флаг, если он равен false, то нужно добавить (проинициальзировать) настройки только если мы их еще не добавляли. Этот метод берет все настройки из файла и "сохраняет"их в PreferenceManger, причем настроки, для которых были указаны defaultValue, будут проинициализированы этими значениям. Этот метод статичный, вам не нужен экземпляр класса PreferenceManager для его вызова. По всей видимости, внутри себя PreferenceManager хранит статичный экземпляр

класса SharedPreferences, в котором и сохраняются добавленные настройки. Соответсвенно, когда мы хотим обратиться к настройкам, мы вызываем у PreferenceManager статичный метод getDefaultSharedPreferences(context), который возвращает static SharedPreferences. То есть, нам возвращается SharedPreferences, в котором хранятся те настройки, которые мы туда загружали методом setDefaultValues(), причем, если эту настройку пользователь не менял, там будет дефолтное значение, иначе пользовательское. Ну а имея экземпляр SharedPreferences, мы можем спрашивать у него нужные нам величины с помощью getType(String key) (Type - это тип значенея). Например, возвращаясь к первому примеру про звук, для получения информации о нем можно сделать что-то типа:

```
1 PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
2 SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this
    );
3 print(Boolean.toString(sharedPref.contains("sound")))) // prints "YES"
4 print(sharedPref.getString("sound")) // prints true/false
```

1.9. Изменение настроек из кода

Может случиться так, что вы захотите помнять какую-нибудь настройку прямо из кода. Для этого мы точно так же получаем SharedPreferences. У него есть метод edit(), который возвращает объект SharedPreferences.Editor, с помощью которого можно менять настройки, соответсвующие SharedPreferences. Для этого у него есть методы putType(String key, Type value). Также надо не забыть применть наши изменения, для этого можно использовать одну из двух методов.

- apply() - производит запись изменений ассинхронно (смотри AsyncTask) и не уведоит вас об ошибке, если она случится
- commit() - начинает запись тут же и возвращает true, если получилось.

Пример:

```
1 PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
2 SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this
    );
3 SharedPreferences.Editor editor = sharedPref.edit();
4 editor.putBoolean("sound", true);
5 editor.apply();
```

SQLite

16 октября

2.1. SQLite

SQLite - это некоторая система управления базой данных. В Java для работы с ней есть весьма неудобный интерфейс, на который мы сейчас попытаемся посмотреть. Основными сущностями являются Connection - соединение с базой данных, по которому мы с ней общаемся и Statement - сообщения, которые мы ей передаём. Соединение можно получить у DriverManager'а с помощью метода getConnection. Statement можно получить у имеющегося объкта Connection с помощью createStatement. Работать с Connection предлагается через методы commit и close, а с Statement через методы executeUpdate, executeQuery и close. Итак, приступим.

2.2. Подключение к базе данных

```
1  import java.sql.*;
2
3  public class SQLiteJDBC
4  {
5    public static void main( String args[] )
6    {
7      Connection c = null;
8      try {
9        Class.forName("org.sqlite.JDBC");
10       c = DriverManager.getConnection("jdbc:sqlite:test.db");
11     } catch ( Exception e ) {
12       System.err.println( e.getClass().getName() + ": " + e.getMessage() );
13       System.exit(0);
14     }
15     System.out.println("Opened database successfully");
16   }
17 }
```

2.3. Создание таблицы

```
1  import java.sql.*;
2
3  public class SQLiteJDBC
4  {
5    public static void main( String args[] )
6    {
7      Connection c = null;
8      Statement stmt = null;
9      try {
10       Class.forName("org.sqlite.JDBC");
11       c = DriverManager.getConnection("jdbc:sqlite:test.db");
12       System.out.println("Opened database successfully");
13
14       stmt = c.createStatement();
15       String sql = "CREATE TABLE COMPANY " +
16       "(ID INT PRIMARY KEY     NOT NULL," +
```

```
      " NAME            TEXT    NOT NULL, " +
      " AGE             INT     NOT NULL, " +
      " ADDRESS         CHAR(50), " +
      " SALARY          REAL)";
      stmt.executeUpdate(sql);
      stmt.close();
      c.close();
    } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      System.exit(0);
    }
    System.out.println("Table created successfully");
  }
}
```

2.4. Запрос INSERT

```
import java.sql.*;

public class SQLiteJDBC
{
  public static void main( String args[] )
  {
    Connection c = null;
    Statement stmt = null;
    try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      c.setAutoCommit(false);
      System.out.println("Opened database successfully");

      stmt = c.createStatement();
      String sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
      "VALUES (1, 'Paul', 32, 'California', 20000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
      "VALUES (2, 'Allen', 25, 'Texas', 15000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
      "VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
      "VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );";
      stmt.executeUpdate(sql);

      stmt.close();
      c.commit();
      c.close();
    } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      System.exit(0);
    }
```

```
39      System.out.println("Records created successfully");
40    }
41  }
```

2.5. Запрос SELECT

```
1  import java.sql.*;
2
3  public class SQLiteJDBC
4  {
5    public static void main( String args[] )
6    {
7      Connection c = null;
8      Statement stmt = null;
9      try {
10       Class.forName("org.sqlite.JDBC");
11       c = DriverManager.getConnection("jdbc:sqlite:test.db");
12       c.setAutoCommit(false);
13       System.out.println("Opened database successfully");
14
15       stmt = c.createStatement();
16       ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANY;" );
17       while ( rs.next() ) {
18         int id = rs.getInt("id");
19         String  name = rs.getString("name");
20         int age  = rs.getInt("age");
21         String  address = rs.getString("address");
22         float salary = rs.getFloat("salary");
23         System.out.println( "ID = " + id );
24         System.out.println( "NAME = " + name );
25         System.out.println( "AGE = " + age );
26         System.out.println( "ADDRESS = " + address );
27         System.out.println( "SALARY = " + salary );
28         System.out.println();
29       }
30       rs.close();
31       stmt.close();
32       c.close();
33     } catch ( Exception e ) {
34       System.err.println( e.getClass().getName() + ": " + e.getMessage() );
35       System.exit(0);
36     }
37     System.out.println("Operation done successfully");
38   }
39 }
```

AsyncTask

16 октября

AsyncTask - класс, который позволяет выполнять задачу в фоновом режиме не блокируя пользовательский поток. Примерами задач, для которых можно использовать AsyncTask являются всевозможные случаи общения по сети (скачивание/загрузка файлов, обращение к внешней базе данных), открытие/сохранение ёмких файлов и подобное. Этот класс является некой обёрткой над классами Thread и Handler, которые отвечают за многопоточность в общем. Рекомендуется использовать его для коротких задач, а для более тяжёлых задач стоит посмотреть на Executor, ThreadPoolExecutor и FutureTask.

Класс AsyncTask является generic'ом с парметрами Params, Progress и Result. Params - тип параметров, которые передаются задаче, Progress - тип, который используется для отслеживания прогресса, Result - тип результата задачи. Для того, чтобы использовать AsyncTask нужно отнаследоваться от него и переопределить некоторые из следующих методов:

- Result doInBackground(Params... params) - этот метод переопределять обязательно. В нём содержится непосредственно код выполнения задачи.
- void onPreExecute() - метод, который вызывается до doInBackground. В нём можно провести необходимую инициализацию.
- void onProgressUpdate(Progess... values) - для того, чтобы отслеживать прогресс задачи можно вызывать publishProgress(Progess... values) из doInBackground. В свою очередь, publishProgess вызывает onProgressUpdate
- void onPostExecute(Result result) - метод, который вызывается после завершения doInBackground.
- void onCancelled() - метод, который вызывается после завершения doInBackground, если задача была отменена с помощью cancel.

Пример:

```
1  private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
2    protected Long doInBackground(URL... urls) {
3      int count = urls.length;
4      long totalSize = 0;
5      for (int i = 0; i < count; i++) {
6        totalSize += Downloader.downloadFile(urls[i]);
7        publishProgress((int) ((i / (float) count) * 100));
8        if (isCancelled()) break;
9      }
10     return totalSize;
11   }
12   protected void onProgressUpdate(Integer... progress) {
13     setProgressPercent(progress[0]);
14   }
15   protected void onPostExecute(Long result) {
16     showDialog("Downloaded " + result + " bytes");
17   }
18 }
```

JSON

16 октября

4.1. Формат

Disclaimer: обязательно прочитайте последнюю часть этой главы.

JSON (**JavaScript Object Notation**) - это формат человекочитаемой записи объектов. Объект в нём представляется в виде набора пар ключ-значение (то есть имя поля и его значение). Используется он преимущественно для передачи данных по сети, в частности в асинхронных операциях.

Объект класса Person в формате JSON может выглядеть так:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
  {
    "type": "home",
    "number": "212 555-1234"
  },
  {
    "type": "office",
    "number": "646 555-4567"
  },
  {
    "type": "mobile",
    "number": "123 456-7890"
  }
  ],
  "children": [],
  "spouse": null
}
```

Мы видим окружённое в фигурные скобки перечисление полей через запятую. Каждое поле описывается в формате "field_name": value, где value может быть нескольких типов:

- Число: знаковое десятичное число, которое может иметь дробную часть и может быть записать в экспоненциальной записи.
- Строка: последовательность из нуля или более юникод-символов заключённых в двойные кавычки.
- Логичекое значение: true или false

- Массив: упорядоченный список значений, каждое из которых может быть любого типа, заключённый в квадратные скобки.
- Объект: неупорядоченная коллекция пар ключ-значение, где ключи являются строками, заключённый в фигурные скобки. Рекомендуется, но не требуется чтобы ключи внутри одного объекта были уникальными.
- null: пустое значение обозначаемое словом null.

4.2. JSON Scheme

Для любителей строгости есть способ описать требования к передаваемому объекту, который называется JSON Scheme. Она представляет из себя JSON-объект и имеет интуитивный формат.

Пример:

```
{
  "\$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

Эта схема может использоваться для проверки валидности следующего объекта:

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
  "Bar",
  "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

Подробнее можно почитать в спецификации: https://tools.ietf.org/html/draft-zyp-json-schema-04

4.3. JSON в Java

В Java для работы с JSON есть несколько прекрасных классов:

- JsonObject, JsonArray - неизменяемые массивы/объекты, которые можно создавать с помощью соответствующих Builder'ов.
- JsonString, JsonNumber, JsonValue - типы данных JSON.
- JsonReader - класс с методами readObject, readArray, который позволяет читать объекты из потока или любого Reader.
- JsonWriter - класс с методами writeObject, writeArray, который позволяет писать объекты в поток или любой Writer.
- JsonObjectBuilder, JsonArrayBuilder - классы, позволяющие создавать JSON массивы и объекты.
- Json - класс, с набором статических методов для создания Reader'ов, Writer'ов и Builder'ов.

Пример работы с JSON:

```
        JsonObject value = Json.createObjectBuilder()
        .add("firstName", "John")
        .add("lastName", "Smith")
        .add("age", 25)
        .add("address", Json.createObjectBuilder()
        .add("streetAddress", "21 2nd Street")
        .add("city", "New York")
        .add("state", "NY")
        .add("postalCode", "10021"))
        .build();

        JsonWriter jw = Json.createWriter(System.out);
        jw.writeObject(value);
```

4.4. Не используйте JSON

Всё дело в том, что JSON в контексте проекта под Андроид общение по сети будет происходить между клиентской и серверной частью вашего приложения и, как следствие, никакая человекочитаемость интересовать вас не будет. С другой стороны, человекочитаемость требует значительного количества дополнительных данных, а лишняя нагрузка и траффик мало кому понравятся. В качестве альтернативы предлагается использовать продукт Google, который называется Protocol Buffers. Возможно он будет немного покрыт в этом конспекте немного позже :). Почитать о нём можно по ссылке https://developers.google.com/protocol-buffers/