

Программа Android-разработчик

Конспект

Быстрый старт
в Android-разработку

Многопоточность и сетевое
взаимодействие

Часть 2

Архитектура
Android-приложений

Тестирование и работа
с библиотеками

Дизайн и анимации

Облачные сервисы
и периферия

Оглавление

1 НЕДЕЛЯ 2	3
1.1 Списки	3
1.1.1 Экраны со списками. Обзор ListView, GridView	3
1.1.2 Обзор RecyclerView, Adapter, Holder, LayoutManager	13
1.1.3 Работа с RecyclerView (заглушечные данные)	19
1.1.4 Работа с RecyclerView (заглушечные данные), часть 2	22
1.1.5 Добавление SwipeRefreshLayout	27
1.1.6 Добавление ContentProvider, CursorLoader, показ контактов в RecyclerView	37
1.1.7 Обработка нажатий на элементы списка	42
1.1.8 Добавление декораторов	48
1.2 Работа с файлами	52
1.2.1 Способы хранения данных в Android (Preferences, Sqlite+Room, Файлы) .	52
1.2.2 Чтение данных из assets/raw	55
1.2.3 Runtime Permissions	56
1.2.4 Запрос Runtime Permissions	61

1.2.5	Запись данных в файловую систему	66
1.2.6	Материалы для самостоятельного изучения	72
1.3	Работа с БД	72
1.3.1	Проектирование БД на бумаге	72
1.3.2	Room. Знакомство	74
1.3.3	Создание Room базы	76
1.3.4	Сохранение и извлечение данных с Room	80
1.3.5	Добавление контент провайдера над Room	84

Глава 1

НЕДЕЛЯ 2

1.1. Списки

1.1.1. Экраны со списками. Обзор ListView, GridView

Предисловие

Практически в каждом приложении существует необходимость в том, чтобы отобразить огромное количество данных одного и того же и даже не одного и того же типа. И чаще всего отображают такие данные в виде списка. Лента в ВК – список. Лента в инстаграме – список. Чат? Тоже список!

Вот так может выглядеть список постов:

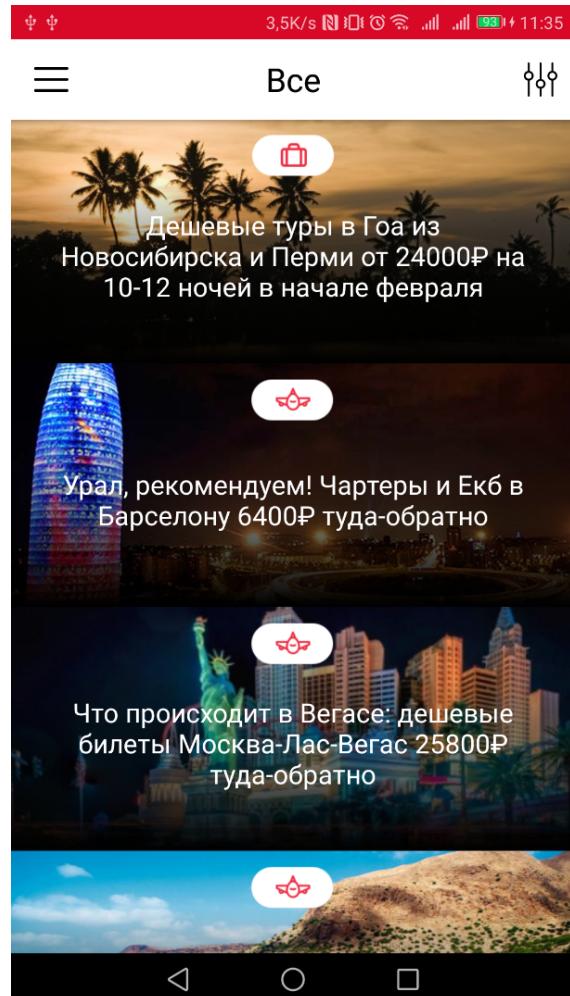


Рис. 1.1

Список помимо того, что содержит различные данные, может также обновляться и дополняться новыми данными.

Так выглядит обновление списка для пользователя:

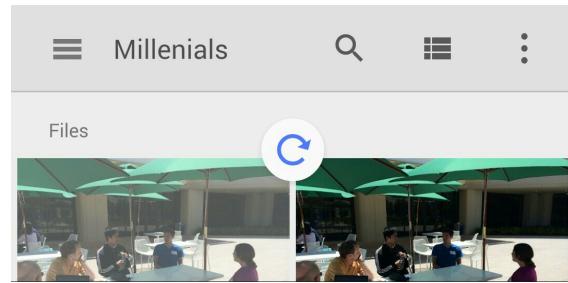


Рис. 1.2

Список не обязательно должен состоять только из текстовых строк. Он может состоять из картинок, текста, аудиозаписей, видеозаписей и вообще содержать абсолютно любые различные данные. Причем все это может содержаться в одном списке.

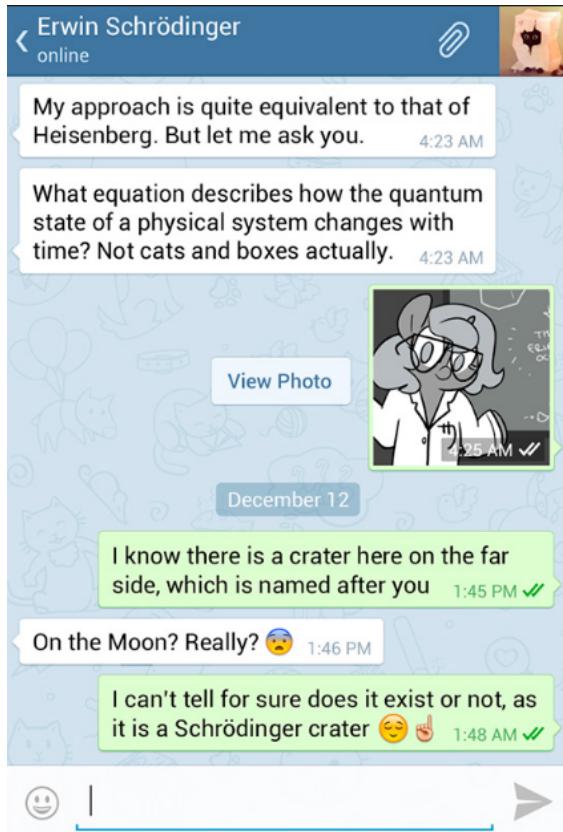


Рис. 1.3

Также часто требуется обработать нажатие по элементу списка или по элементу, который содержится внутри него, чтобы, например, открыть элемент на весь экран или произвести какое-либо другое действие.

List View. Для чего?

Чтобы вывести на экран список, состоящий из множества элементов, нам как раз поможет **ListView**.

ListView – view, которая отображает на экране список элементов с помощью адаптера.

Чтобы **ListView** понимал, как именно должны отображаться данные, мы должны передать в

него Adapter, а внутри Adapter описать то, в каком view какие данные должны быть.

ListView. Как работает?

Давайте попробуем с вами создать список, который будет выводить элементы с иконкой, текстом и кнопкой. При нажатии на кнопку показывается Toast-уведомление с текстом элемента.

Создадим xml layout для нашего элемента списка:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="wrap_content"
6     android:gravity="center"
7     android:orientation="horizontal"
8     android:padding="16dp"
9     tools:context="com.e_legion.coursera.MainActivity">
10
11     <ImageView
12         android:id="@+id/ivIcon"
13         android:layout_width="24dp"
14         android:layout_height="24dp" />
15
16     <TextView
17         android:id="@+id/tvText"
18         android:layout_width="wrap_content"
19         android:layout_height="wrap_content"
20         android:layout_weight="1"
21         android:maxLines="1"
22         android:paddingLeft="12dp" />
23
24     <Button
25         android:id="@+id/button"
26         android:layout_width="wrap_content"
27         android:layout_height="wrap_content"
28         android:maxLines="1"
29         android:paddingLeft="12dp"
30         android:text="Нажми меня!" />
```

```
31    </LinearLayout>
```

Добавим ListView в xml layout:

```
1  <ListView
2      android:id="@+id/listView"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent" />
5
6  </LinearLayout>
```

Инициализируем ListView и адаптер в Activity:

```
1  public class MainActivity extends AppCompatActivity {
2
3      private ListView mListview;
4      private ListViewAdapter mAdapter = new ListViewAdapter(new ArrayList<ListObject>()
5          {{
6              add(new ListObject(R.drawable.ic_android_black_24dp, "Android"));
7              add(new ListObject(R.drawable.ic_add_location_black_24dp, "Местоположение"));
8              add(new ListObject(R.drawable.ic_arrow_downward_black_24dp, "Безумный
9                  текст"));
10             add(new ListObject(R.drawable.ic_audiotrack_black_24dp, "Лалака"));
11             add(new ListObject(R.drawable.ic_launcher_background, "Сеточка"));
12         }});
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18
19         mListview = findViewById(R.id.listView);
20         mListview.setAdapter(mAdapter);
21     }
22 }
```

Код адаптера:

```
1  public class ListViewAdapter extends BaseAdapter {
2
3      private List<ListObject> mListObjects;
4
5      public ListViewAdapter(List<ListObject> listObjects) {
6          mListObjects = listObjects;
7      }
8
9      @Override
10     public View getView(int position, View view, ViewGroup viewGroup) {
11         LayoutInflator inflater = (LayoutInflator)
12             → viewGroup.getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
13         View v = inflater.inflate(R.layout.li_object, viewGroup, false);
14         TextView tvText = v.findViewById(R.id.tvText);
15         ImageView ivIcon = v.findViewById(R.id.ivIcon);
16         Button button = v.findViewById(R.id.button);
17
18         final ListObject item = mListObjects.get(position);
19
20         tvText.setText(item.getText());
21         ivIcon.setImageDrawable(ContextCompat.getDrawable(viewGroup.getContext(),
22             → item.getIcon()));
23         button.setOnClickListener(new View.OnClickListener() {
24             @Override
25             public void onClick(View view) {
26                 Toast.makeText(view.getContext(), item.getText(),
27                     → Toast.LENGTH_SHORT).show();
28             }
29         });
30
31         return v;
32     }
33
34     @Override
35     public int getCount() {
36         return mListObjects.size();
37     }
38
39     @Override
40     public ListObject getItem(int position) {
```

```
38         return mListObjects.get(position);
39     }
40
41     @Override
42     public long getItemId(int position) {
43         return position;
44     }
45 }
```

Вообще можно наследоваться не только от BaseAdapter, но также и от ListAdapter, ArrayAdapter и других.

На экране это все выглядит вот так:

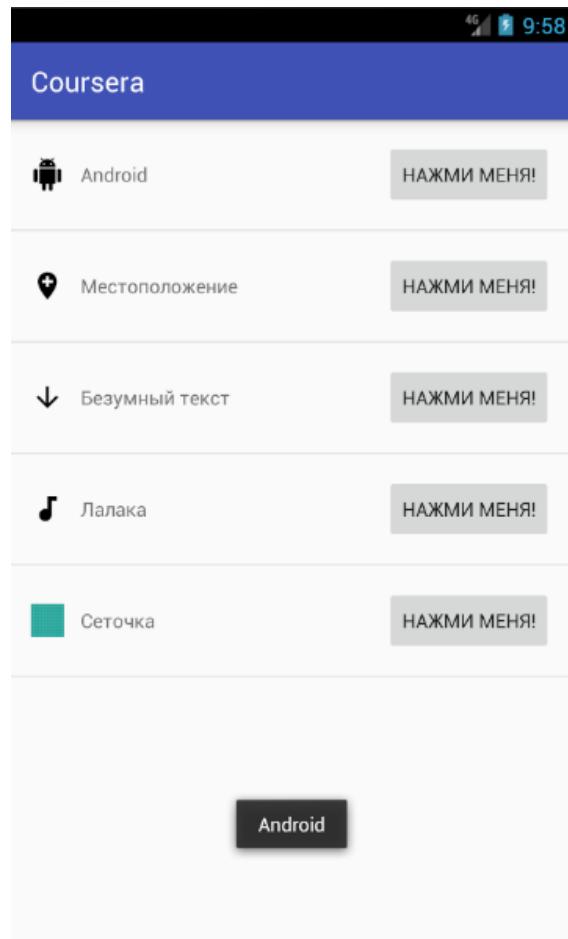


Рис. 1.4

GridView. Для чего?

GridView – это почти то же самое, что и ListView, но в GridView можно размещать элементы в несколько столбцов.

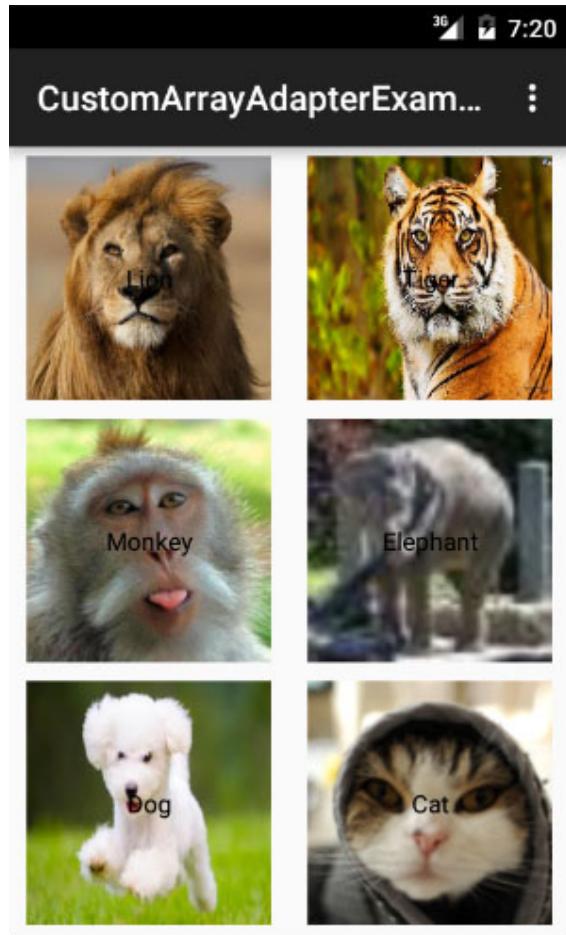


Рис. 1.5

Соответственно у него есть атрибуты, которые отвечают за настройку его уникальных свойств.

- android:numColumns – определяет количество столбцов. Если поставлено значение auto_fit, то система вычислит количество столбцов, основанное на доступном пространстве.
- android:verticalSpacing – устанавливает размер пустого пространства между ячейками таблицы.
- android:columnWidth – устанавливает ширину столбцов.
- android:stretchMode – указывает, куда распределяется остаток свободного пространства для таблицы с установленным значением.

- android:numColumns="auto_fit". Принимает значения columnWidth для распределения остатка свободного пространства между ячейками столбцов для их увеличения или spacingWidth – для увеличения пространства между ячейками.

Если упростить, то во всем остальном GridView идентичен ListView.

Отличные гайды по работе с ListView:

1. <https://www.raywenderlich.com/124438/android-listview-tutorial>
2. <https://developer.android.com/guide/topics/ui/layout/listview.html>
3. <https://guides.codepath.com/android/implementing-a-horizontal-listview-guide>

Дополнительная информация и гайды по GridView:

1. <http://developer.alexanderklimov.ru/android/views/gridview.php>
2. <https://www.raywenderlich.com/127544/android-gridview-getting-started>
3. <https://developer.android.com/guide/topics/ui/layout/gridview.html>

1.1.2. Обзор RecyclerView, Adapter, Holder, LayoutManager

Предисловие

Мы с вами уже познакомились с ListView и GridView. Но у этих View есть некоторые проблемы. При достаточно большом количестве данных они будут работать очень медленно, и поэтому ребята из Google придумали такую штуку как RecyclerView. Можно сказать, что это универсальная View для отображения «списковых» данных. Она настраиваема так, как ваша душа пожелает и даже больше.

ListView + GridView vs RecyclerView. Основные отличия

Но в чем же существенные отличия? А в том, что RecyclerView использует паттерн ViewHolder. Этот паттерн нужен для того, чтобы постоянно не создавать с нуля каждый элемент списка, который отображается на экране. Т.е. у ListView каждый элемент создавался каждый раз с нуля и плодил огромную кучу ненужных объектов в памяти, а в RecyclerView из-за использования паттерна заново используются уже созданные ViewHolder (элементы списка, если по-простому) и удаляются уже неиспользуемые, что благоприятно оказывается на быстродействии и размере используемой памяти.

В ListView, конечно, можно реализовать этот паттерн самому, как раньше и делали, но RecyclerView обязывает вас использовать этот паттерн, к тому же, зачем писать свой велосипед?

Также в RecyclerView есть такая штука, как LayoutManager. LayoutManager нужен для того, чтобы указывать, каким образом будут позиционироваться элементы внутри нашего списка, при желании вы можете написать свой собственный. Вот основные из них:

- LinearLayoutManager – горизонтальный (1 столбец) или вертикальный (1 строка).
- GridLayoutManager – менеджер, по аналогии с GridView.
- StaggeredGridLayoutManager – проще один раз показать, чем рассказать...

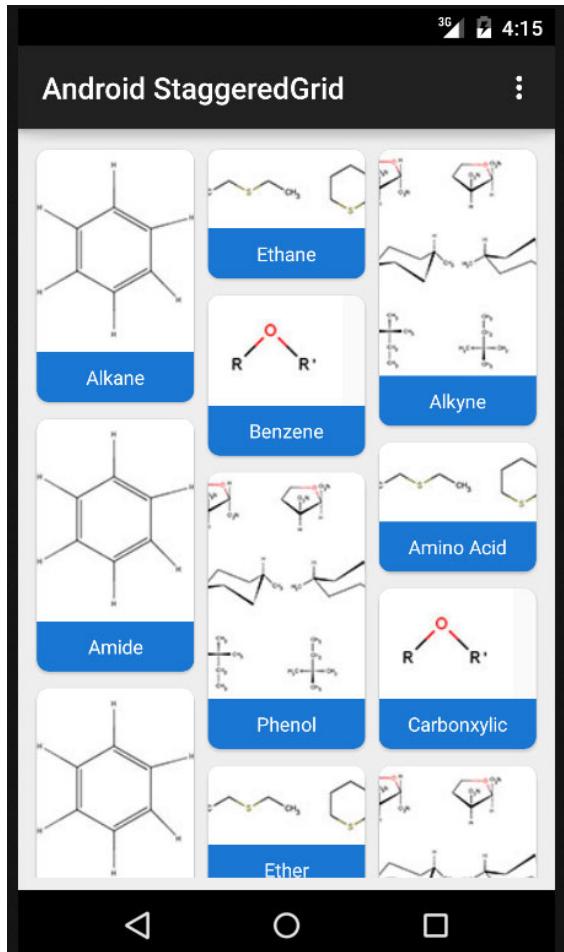


Рис. 1.6

RecyclerView. Практика

Давайте попробуем создать точно такой же список, который мы создавали, когда изучали ListView, но уже с использованием ViewHolder. Напомню, список состоит из элементов с иконкой, текстом и кнопкой, по нажатию на которую показывается Toast-уведомление. По факту, реализация работы с ними практически не будет отличаться, но вот реализация адаптеров будет абсолютно разная, и также нам будет нужно указать LinearLayoutManager. Давайте посмотрим на различия.

Для начала подключим RecyclerView в Gradle:

```
1 implementation 'com.android.support:recyclerview-v7:26.1.0'
```

Да, он подключается отдельно, в отличие от ListView.

View и наши layout остались абсолютно такими же, но после инициализации RecyclerView нужно указать ему LayoutManager:

```
1 mRecyclerView = findViewById(R.id.recyclerView);
2 mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
3 mRecyclerView.setAdapter(mAdapter);
```

Теперь перейдем к адаптеру. Раньше мы наследовались от BaseAdapter, теперь же мы наследуемся от RecyclerView.Adapter:

```
1 public class ListViewAdapter extends RecyclerView.Adapter<ListObjectViewHolder>
```

В адаптере, по умолчанию, нам нужно переопределить всего 3 метода:

- onCreateViewHolder – метод, который вызывается, когда нужно создать ViewHolder.
- onBindViewHolder – метод, который вызывается, когда отображается item на определенной позиции. В нем нужно обновлять отображаемые данные.
- getItemCount – текущее количество элементов в списке.

Переопределим onCreateViewHolder:

```
1 @Override
2 public ListObjectViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
3     return new ListObjectViewHolder(LayoutInflater.from(parent.getContext()).
4         inflate(R.layout.li_object, parent, false));
5 }
```

Переопределим onBindViewHolder, мы рекомендуем делать это в методе bind самого ViewHolder, так код будет выглядеть чище и правильнее:

```
1  @Override
2  public void onBindViewHolder(ListObjectViewHolder holder, int position) {
3      holder.bind(mListObjects.get(position));
4  }
```

И метод getItemCount:

```
1  @Override
2  public int getItemCount() {
3      return mListObjects.size();
4  }
```

Так выглядит сам ViewHolder:

```
1  public class ListObjectViewHolder extends RecyclerView.ViewHolder {
2      TextView tvText;
3      ImageView ivIcon;
4      Button button;
5      public ListObjectViewHolder(View itemView) {
6          super(itemView);
7          tvText = itemView.findViewById(R.id.tvText);
8          ivIcon = itemView.findViewById(R.id.ivIcon);
9          button = itemView.findViewById(R.id.button);
10     }
11
12     public void bind(final ListObject item) {
13         tvText.setText(item.getText());
14         ivIcon.setImageDrawable(ContextCompat.getDrawable(itemView.getContext(),
15             item.getIcon()));
16         button.setOnClickListener(new View.OnClickListener() {
17             @Override
18             public void onClick(View view) {
19                 Toast.makeText(view.getContext(), item.getText(),
20                     Toast.LENGTH_SHORT).show();
21             }
22         });
23     }
24 }
```

Так он выглядит для пользователя:

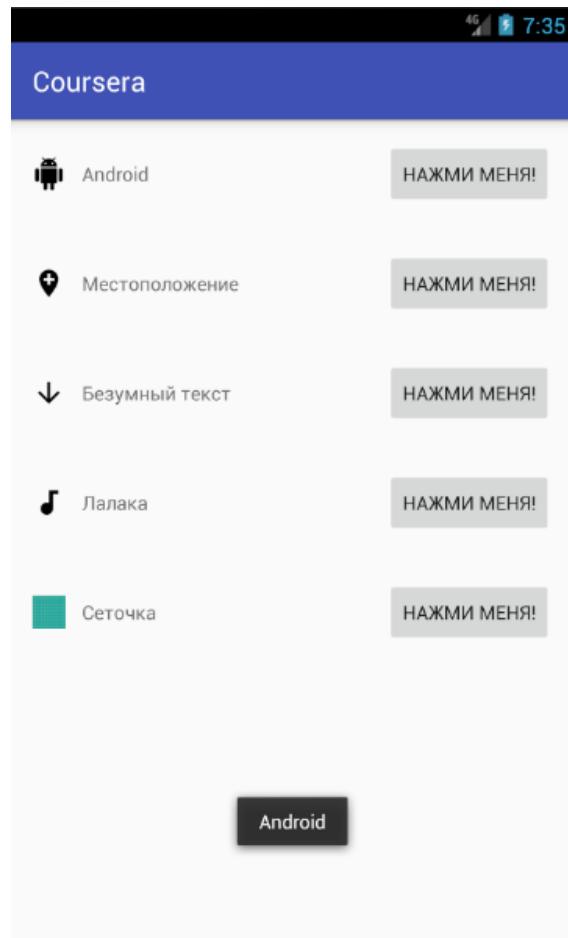


Рис. 1.7

Также наш RecyclerView может содержать в себе разные ViewHolder. Например, нам нужно отобразить заголовок или просто каким-то образом отличающиеся ViewHolder. Его возможности ограничены только вашей фантазией. С другими более сложными возможностями RecyclerView и его адаптера мы встретимся в последующих уроках специализации.

Дополнительная информация:

1. <https://habrahabr.ru/post/258195/>

2. <https://developer.android.com/training/material/lists-cards.html?hl=ru>
3. <https://developer.android.com/guide/topics/ui/layout/recyclerview.html>
4. <https://guides.codepath.com/android/using-the-recyclerview>
5. <https://willowtreeapps.com/ideas/android-fundamentals-working-with-the-recyclerview-adapter-and-viewholder-pattern/>

1.1.3. Работа с RecyclerView (заглушечные данные)

В этом видео мы разберем, как работать со списками, причем я разбил план на несколько пунктов, и их видно на экране. Во-первых, мы добавим фрагмент с recyclerView, дальше создадим адаптер, холдер для хранения данных и генератор заглушечных данных, дальше мы добавим обновление данных по swipe to refresh и состояние ошибки, после этого сделаем загрузку данных из телефонной книги, добавим обработку нажатий, допустим, звонок по контакту, и добавим декораторы. Что ж, начнем.

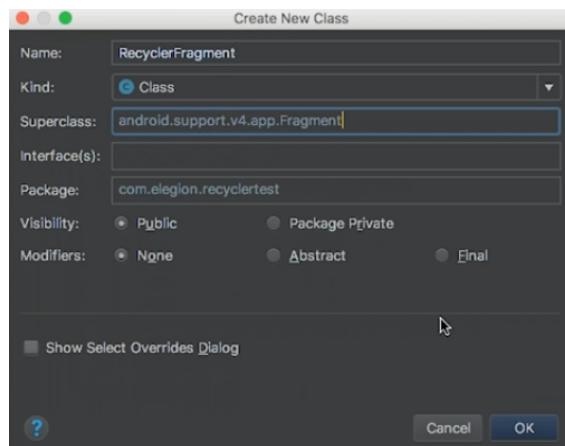


Рис. 1.8

Для начала щелкаем на Java, создаем новый Java-класс и называем его RecyclerFragment – создаем фрагмент. Суперкласс – Fragment из библиотеки поддержки. Щелкаем OK.

Переопределяем onCreateView():

```
1 public class RecyclerFragment extends Fragment {  
2  
3     @Nullable  
4     @Override  
5     public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,  
6         @Nullable Bundle savedInstanceState) {  
7         return inflater.inflate(R.layout.fr_recycler, container, attachToRoot: false);  
8     }  
}
```

Layout'a такого у нас нет, давайте создадим. Alt+Enter -> Create layout resource file -> Enter. Корневой элемент – LinearLayout. Щелкаем OK.

Теперь нам нужно добавить сюда Recycler, и самый простой способ – это перейти во вкладку Design, найти Recycler среди компонентов в палитре и просто перетащить. Studio спросит, стоит ли качать, щелкаем OK. Сейчас в Gradle добавилась зависимость с recycler'ом, можем даже проверить build.gradle (Module: app), видим:

```
1 implementation 'com.android.support:recyclerview-v7:26.1.0'
```

Окей. Возвращаемся в RecyclerFragment – ошибки больше нет. Возвращаемся во Fragment Recycler, открываем текст, видим RecyclerView. Размеры нас устраивают, нам нужно только добавить id для этого элемента.

```
1 android:id="@+id/recycler"
```

Снова возвращаемся в RecyclerFragment. Теперь нам неплохо было бы добавить переменную для этого recycler'a. Переопределяем метод onViewCreated():

```
1 @Override  
2 public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {  
3     RecyclerView recycler = view.findViewById(R.id.recycler);  
4 }
```

Наводимся на переменную recycler, зажимаем Cmd+Alt+F, ну или Ctrl+Alt+F, если у вас Windows. Это приведет к тому, что создастся поле с этой переменной, то есть вместо локальной

переменной у нас теперь переменная поля. Теперь у нас есть фрагмент, но неплохо было бы добавить его в activity. Создадим метод newInstance для фрагмента.

```
1 public static RecyclerFragment newInstance() {  
2     return new RecyclerFragment();  
3 }
```

Переменные нам не нужны, просто возвращаем new RecyclerFragment. Переходим в MainActivity и в методе onCreate() прописываем:

```
1 getSupportFragmentManager().beginTransaction()  
2     .replace(R.id.container, RecyclerFragment.newInstance())  
3     .commit();
```

У нас нет контейнера, давайте его добавим. Переходим в activity_main и удаляем ConstraintLayout, вместо него у нас будет LinearLayout. orientation="vertical". Удаляем TextView. Вместо нее добавляем FrameLayout.

```
1 <FrameLayout  
2     android:id="@+id/container"  
3         android:layout_width="match_parent"  
4         android:layout_height="match_parent"/>
```

Переходим в MainActivity, видим, что ошибка исчезла. Что мы можем сделать дальше? Так как после переворота экрана State фрагментов сохраняется, мы добавляем фрагмент только в случае, если savedInstanceState равен нулю, то есть это первое создание экрана. Иначе фрагмент сам убивается.

```
1 if (savedInstanceState == null) {  
2     getSupportFragmentManager().beginTransaction()  
3         .replace(R.id.container, RecyclerFragment.newInstance())  
4         .commit();  
5 }
```

Так, можем попробовать запустить. Сейчас в эмуляторе у меня старая разметка с textView Hello World. По идеи после перезапуска я должен стать обладателем пустого экрана, если все пошло как надо. Да, пустой экран готов. Я думаю, что с первым пунктом мы разобрались, теперь

нам нужно добавить адаптер, холдер и генератор заглушечных данных. Этим мы займемся на следующем уроке.

1.1.4. Работа с RecyclerView (заглушечные данные), часть 2

В этом видео нам нужно добавить адаптер, чтобы показывать элементы, холдер, чтобы было где их показывать и генератор заглушечных данных, то есть имитацию запроса в сеть либо к базе данных – ничего сложного.

Переходим в пакет recyclertest, щелкаем правой: New → Package, называем наш новый пакет mock – здесь будут храниться адаптер, холдер, класс заглушки и генератор заглушки. Давайте начнем с самой заглушки, я назову ее Mock. Добавим две переменные – String и int. Ctrl+N → Constructor → оба поля, Ctrl+N → Getter and Setter. Пусть также getValue() возвращает нам не int, а строку. То есть мы получаем то же самое, но в виде строки.

```
1  public class Mock {  
2  
3      private String mName;  
4  
5      private int mValue;  
6  
7      public Mock(String name, int value) {  
8          mName = name;  
9          mValue = value;  
10     }  
11  
12     public String getName() {  
13         return mName;  
14     }  
15  
16     public void setName(String name) {  
17         mName = name;  
18     }  
19  
20     public String getValue() {  
21         return String.valueOf(mValue);  
22     }  
23 }
```

```
24     public void setValue(int value) {
25         mValue = value;
26     }
27 }
```

С классом определились, теперь давайте создадим генератор. Щелкаем по тому же пакету, New → Java Class. Имя – MockGenerator.

Создадим статический метод generate. Он должен возвращать List<Mock>. На вход будет принимать количество элементов, которые нужно генерировать – int count. Прописываем mocks сразу нужного размера). Нужно передать название и value, давайте их тоже будем генерировать.

```
1  public class MockGenerator {
2
3      public static List<Mock> generate(int count) {
4          List<Mock> mocks = new ArrayList<>(count);
5          Random random = new Random();
6
7          for(int i = 0; i < count; i++) {
8              mocks.add(new Mock(UUID.randomUUID().toString(), random.nextInt( bound:
9                  ↪ 200) ));
10         }
11
12         return mocks;
13     }
14 }
```

Генератор готов. Теперь создаем адаптер. New → Java Class. Имя – MockAdapter.

Адаптер нужен, чтобы снабжать RecyclerView данными. Он – своего рода посредник между объектами, о которых RecyclerView ничего не знает, и самим RecyclerView. В качестве суперкласса указываем Adapter от RecyclerView.

Alt+Enter → Implement methods.

Что из себя представляет классический адаптер? Во-первых, в нем хранится либо массив, либо List, либо что-то еще из данных, в нашем случае это List<Mock>. getItemCount() возвращает размер этого List'a, onCreateViewHolder() должен возвращать viewHolder, в котором будут

находиться элементы списка. У нас его еще нет, создадим его.

Переходим обратно в наш пакет mock, щелкаем правой, New → Java Class, имя – MockHolder, в качестве родителя указываем ViewHolder, который имеет отношение к RecyclerView. Опять же, Alt+Enter – создаем конструктор, который соответствует родительскому. Еще одно дополнение – мой MockAdapter будет работать только с Mockholder'ом, так что я его передам в качестве generic-параметра.

```
1 public class MockAdapter extends RecyclerView.Adapter<MockHolder> {  
2     // содержание  
3 }
```

MockHolder'у требуется разметка, создадим ее. Переходим в res → layout → New → Layout resource file. Родителем будет опять LinearLayout, имя файла – li_mock. Щелкаем OK, переходим в текстовый редактор и создаем две textView.

```
1 <TextView  
2     android:id="@+id/tv_name"  
3     android:layout_width="match_parent"  
4     android:layout_height="wrap_content"  
5     android:textColor="@color/colorAccent"  
6     android:textStyle="bold"  
7     android:layout_marginBottom="8dp"  
8     tools:text="Sample name" />  
9  
10 <TextView  
11     android:id="@+id/tv_value"  
12     android:layout_width="match_parent"  
13     android:layout_height="wrap_content"  
14     tools:text="12342" />
```

Добавим padding. Также высоту контейнера нужно сделать wrap_content, чтобы она не открывалась на весь экран. В этом контейнере и будут храниться элементы Mock.

```
1 android:layout_height="wrap_content"  
2 android:padding="8dp"
```

Возвращаемся к MockHolder'у.

```
1 public class MockHolder extends RecyclerView.ViewHolder {  
2  
3     private TextView mName;  
4     private TextView mValue;  
5  
6     public MockHolder(View itemView) {  
7         super(itemView);  
8         mName = itemView.findViewById(R.id.tv_name);  
9         mValue = itemView.findViewById(R.id.tv_value);  
10    }  
11}
```

Отлично. Обратно к MockAdapter'у и имплементим onCreateViewHolder(). Для начала нам нужно получить LayoutInflater. Его мы можем получить с помощью контекста, который хранится в ViewGroup parent.

```
1 LayoutInflater inflater = LayoutInflater.from(parent.getContext());
```

Теперь у нас есть Inflater. Inflater нужен для того, чтобы из xml-разметки сделать view. Воспользуемся этим:

```
1 inflater.inflate(R.layout.li_mock, parent, attachToRoot: false);
```

Cmd+Alt+V (Ctrl+Alt+V) сделает из этой строки переменную:

```
1 View view = inflater.inflate(R.layout.li_mock, parent, attachToRoot: false);
```

И возвращаем наш ViewHolder, которому передаем на вход созданную View – в нашем случае это view.

```
1 return new MockHolder(view);
```

Теперь onBindViewHolder(). На вход он получает holder, либо созданный, либо переиспользованный, и position, то есть позицию объекта из адаптера. И в классической реализации binding выглядит следующим образом:

```
1 holder.bind(mMockList.get(position));
```

У нас нет метода bind, но это не страшно. Щелкаем курсором на bind, Alt+Enter → Create method, переходим в MockHolder.

```
1 public void bind(Mock mock) {  
2     mName.setText(mock.getName());  
3     mValue.setText(mock.getValue());  
4 }
```

Теперь, я надеюсь, вам понятно, почему я переделал getValue() так, чтобы он возвращал строку – чтобы не преобразовывать int к строке в методе bind.

Адаптер готов... ну, почти. Он работает с данными, только он их ниоткуда не берет. Сделаем следующим образом. private final List<Mock> mMocklist инициализируем прямо в теле класса:

```
1 private final List<Mock> mMocklist = new ArrayList<>();
```

И создадим метод addData, который на вход получает тоже List Mock-элементов. В реализации просто добавляем полученные элементы в наш текущий mMocklist. Чтобы система знала, что адаптер обновился, неплохо было бы вызвать notifyDataSetChanged(). Если у вас точечные изменения, то лучше использовать другие вариации этого метода.

```
1 public void addData(List<Mock> mocks) {  
2     mMockList.addAll(mocks);  
3     notifyDataSetChanged();  
4 }
```

Осталось добавить MockAdapter к нашему Fragment'у:

```
1 private final MockAdapter mMockAdapter = new MockAdapter();
```

И в onActivityCreated, когда у нас есть доступ к контексту, вызываем у Recycler'a следующие методы:

```
1 mRecycler.setLayoutManager(new LinearLayoutManager(getActivity()));
2 mRecycler.setAdapter(mMockAdapter);
```

Мы добавили Adapter к Recycler'у, но еще не вызвали метод, который добавляет данные в адаптер. Поступаем следующим образом:

```
1 mMockAdapter.addData(MockGenerator.generate(count: 20));
```

И это все. После этих телодвижений мы должны получить экран со списком. Вуаля – список есть, и он, как видите, рандомный, наверху выделен UID, а внизу просто рандомные числа до 200, как и ожидалось.

На следующем занятии мы рассмотрим обновление данных и состояние ошибки. До встречи в следующем видео.

1.1.5. Добавление SwipeRefreshLayout

Итак, в прошлом видео мы добавили адаптер, холдер и генератор заглушечных данных в наш RecyclerView. В этом видео мы добавим обновление данных и состояние ошибки. Что ж, меньше слов, больше дела, давайте этим займемся. Переходим в RecyclerFragment, прямо на разметку и обрамляем наш RecyclerView в так называемый SwipeRefreshLayout. Причем это мы можем сделать двумя способами. Либо мы можем добавить сюда SwipeRefreshLayout, либо мы поступим по-другому, мы поменяем LinearLayout на SwipeRefreshLayout. Зададим ему id.

```
1 android:="@+id/refresher"
```

Переходим во Fragment, добавляем refresher в качестве поля.

```
1 private SwipeRefreshLayout mSwipeRefreshLayout;
```

В onViewCreated() создаем ссылку:

```
1 mSwipeRefreshLayout = view.findViewById(R.id.refresher);
```

Отлично. Мы добавили SwipeRefreshLayout на разметку. Давайте теперь посмотрим, как он работает. Такой подход: вначале добавляем, потом смотрим, как работает. На самом деле SwipeRefreshLayout реализует стандартный механизм обновления данных в Android, скорее всего, вы с ним сталкивались. Если будучи на экране, который я могу обновить, я потяну пальцем вниз – в нашем случае я просто зажму левую кнопку мыши – то я увижу, как из-под тулбара вот таким образом появляется индикатор прогресса или же, как его еще называют, thumb. Если я сейчас, дотянув до упора, отпущу палец (левую кнопку мыши), то я увижу, как thumb крутится, говоря пользователю о том, что идет процесс загрузки данных, обновление экрана либо другой процесс, смотря что происходит на экране. Так как я просто добавил SwipeRefreshLayout на разметку, никак его не обрабатывая, этот кругляш будет крутиться бесконечно. Давайте добавим обработку экрана, точнее, обработку обновления экрана. Что нам нужно для этого сделать? Во-первых, наш фрагмент должен имплементити

```
1 public class RecyclerFragment extends Fragment implements
  ↳ SwipeRefreshLayout.OnRefreshListener {
2     // содержание
3 }
```

Alt+Enter -> Implement methods.

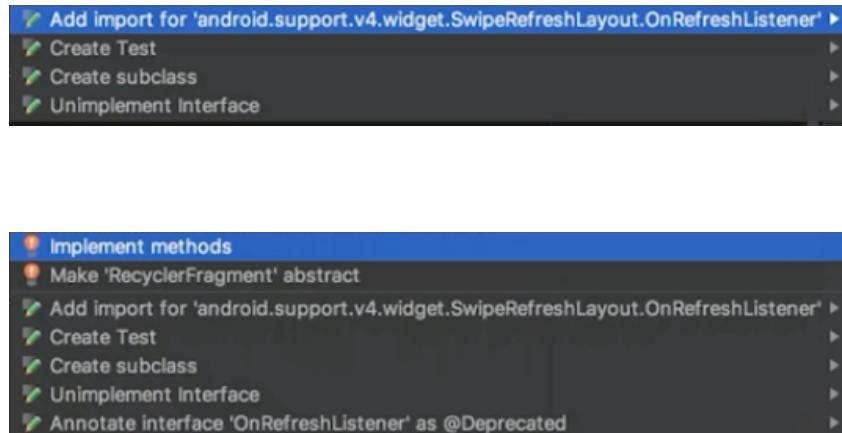


Рис. 1.9

Нам нужно реализовать метод `onRefresh()`. Соответственно, `onRefresh()` вызывается, когда пользователь дергает пальцем по экрану и отпускает, тем самым вызывая загрузку. Мы хотели обновить данные, давайте сымитируем доступ в сеть, то есть обновление данных будет происходить с некоторой задержкой.

```
1 public void onRefresh() {
2     mSwipeRefreshLayout.post(new Runnable() {
3         @Override
4         public void run() {
5
6
7     })
8 }
```

.post на любом view-элементе работает как любой хэндлер на main thread'e, то есть просто постит runnable-сообщение в message queue. То, о чем я рассказывал, что у нас будет в методе `run()`. Во-первых, мы добавим данные в адаптер так, как мы делали это раньше. И для того, чтобы это было нагляднее, я удалю код добавления элементов в `onActivityCreated()` и добавлю его сюда полностью. Я не хочу, чтобы было сгенерировано 20 элементов, пусть будет 5. Так будет гораздо заметнее, что данные изменились.

```
1 public void onRefresh() {
2     mSwipeRefreshLayout.post(new Runnable() {
3         @Override
4         public void run() {
5             mMockAdapter.addData(MockGenerator.generate(count: 5));
6
7     })
8 }
```

Помимо добавления данных, нам нужно спрятать индикатор прогресса. Как мы это делаем? Во-первых, мы проверяем, что индикатор прогресса виден:

```
1 if(mSwipeRefreshLayout.isRefreshing()) {
2     mSwipeRefreshLayout.setRefreshing(false);
3 }
```

Давайте вместо post сделаем postDelayed, добавив тем самым задержку.

```
1 mSwipeRefreshLayout.postDelayed(new Runnable() {...}, delay Millis: 2000);
```

И, самое главное, нам нужно связать реализацию onRefreshListener() с нашим layout'ом. Делается это следующим образом:

```
1 mSwipeRefreshLayout.setOnRefreshListener(this);
```

Запускаем.

Проводим пальцем, в моем случае мышкой. Две секунды. Данные обновились. Проводим пальцем еще раз. Бам. Ошибка. Что происходит? Вместо обновления данных мы получили добавление данных к текущему. Это несколько неправильно, потому что SwipeRefreshLayout подразумевает именно обновление данных. Что нам нужно сделать? Нам нужно перейти в адаптер, в метод addData, и перед добавлением новых данных очистить старые.

```
1 mMockList.clear();
```

И после этого addAll. Запустим. Обновляем. Так, данные получили. Обновляем еще раз. Данные поменялись. Это то, что нам нужно. Но иногда нам нужно и добавлять данные, и обновлять их. Как же поступить в этом случае? Самый простой способ – это в addData помимо самих новых данных передавать еще и bool значение.

```
1 public void addData(List<Mock> mocks, boolean refresh) {
2     if(refresh) {
3         mMockList.clear();
4     }
5 }
```

Соответственно, если мы вызываем вместе с данными еще и true, то обновляем наш список. Очищаем список, добавляем новые данные. Если же в addData() передать false, то будет только добавление данных. Рекомендую взять на вооружение, кстати. Возвращаемся во фрагмент и, так как в нашем случае нам нужно обновить данные, я передаю вместе с данными еще и true, то есть обновить.

```
1 mMockAdapter.addData(MockGenerator.generate(count: 5), refresh: true);
```

Так, реализацию обновления данных мы сделали. Теперь нам нужно сделать состояние ошибки, то есть если вдруг запрос в сеть вернулся неудачным. Давайте добавим состояние ошибки для этого случая. Для начала создадим layout, который будет показываться при ошибке.

Layout → New → Layout resource file.

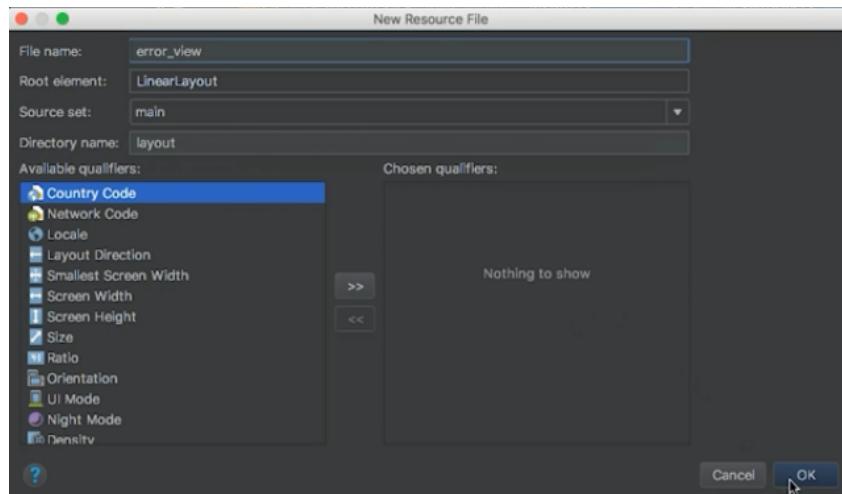


Рис. 1.10

Перейдем во вкладку Text. Добавим ImageView размерами 60 на 60

```
1 <ImageView  
2     android:layout_width="60dp"  
3     android:layout_height="60dp" />
```

и TextView:

```
1 <TextView  
2     android:text="Что-то пошло не так. Попробуйте еще раз."  
3     android:layout_width="match_parent"  
4     android:layout_height="wrap_content" />
```

Текст простенький, например «Что-то пошло не так. Попробуйте еще раз.» Давайте загоним эту строку в ресурсы. Нажмем Alt+Enter, перейдем в меню Extract Resource.

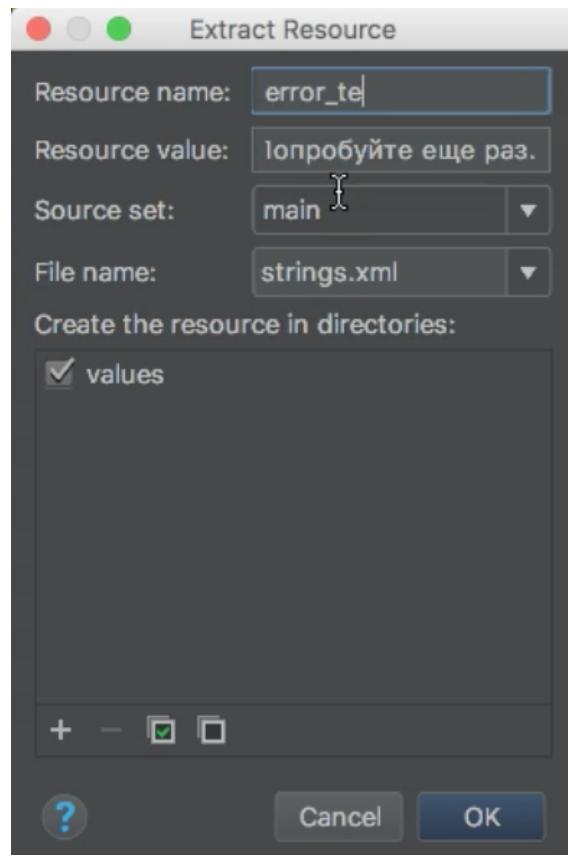


Рис. 1.11

В ImageView добавим иконку ошибки. Давайте попробуем это сделать. Добавим одну из системных иконок. Правой кнопкой по drawable. New -> Vector Asset. Щелкаем по дроиду, жмем alert, затем error. Название поменяем на ic_error. Щелкаем Next. Щелкаем Finish. Щелкаем Yes. Открываем drawable и меняем fillColor на термоядерный красный.

```
1 android:fillColor="#FF0000"
```

Перейдем в меню Extract Resource (Alt+Enter).

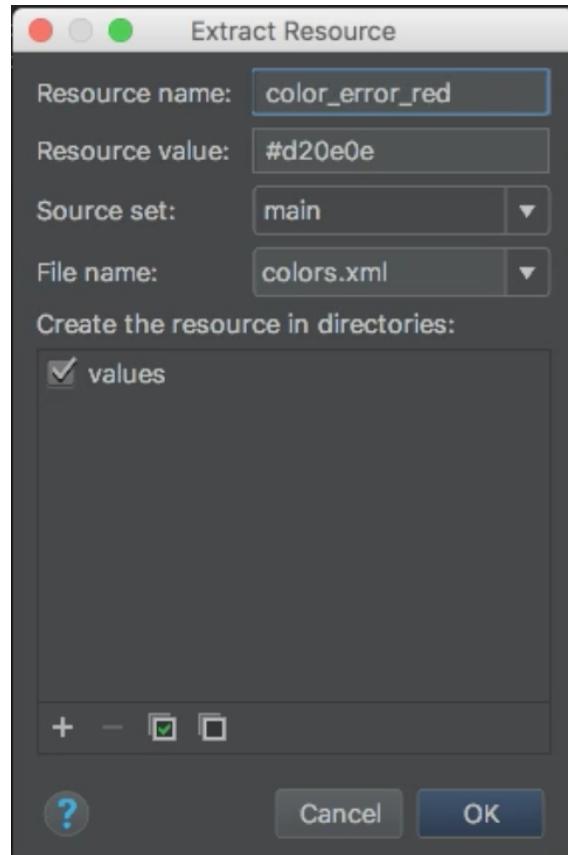


Рис. 1.12

OK.

Возвращаемся в error view. Добавляем source.

```
1 <ImageView  
2     android:src="@drawable/ic_error"  
3     android:layout_width="60dp"  
4     android:layout_height="60dp" />
```

Сделаем цвет текста таким же красным.

```
1 <TextView  
2     android:textColor="@color/color_error_red"  
3     android:text="@string/error_text"  
4     android:layout_width="match_parent"  
5     android:layout_height="wrap_content" />
```

Сделаем его жирным, добавив:

```
1 android:textStyle="bold"
```

Сделаем всю эту красоту по центру:

```
1 android:gravity="center"
```

Поменяем width:

```
1 android:layout_width="wrap_content"
```

Добавим расстояние между текстом и иконкой, добавив в TextView:

```
1 android:layout_marginTop="8dp"
```

Хорошо. Теперь мы создали экран ошибки. Нам нужно добавить его в экран с recycler'ом. Нужно добавить id.

```
1 android:id="@+id/error_view"
```

Возвращаемся в fragment_recycler и оборачиваем наш RecyclerView в frame layout:

```
1 <FrameLayout  
2     android:layout_width="match_parent"  
3     android:layout_height="match_parent">  
4     <android.support.v7.widget.RecyclerView  
5         android:id="@+id/recycler"  
6         android:layout_width="match_parent"
```

```
7         android:visibility="visible"
8         android:layout_height="match_parent"/>
```

Recycler внутри. Ну а теперь помимо recycler'a в framelayout будет еще и state ошибки. Как добавить другой layout в наш layout? Можно воспользоваться командой, точнее, нодой include.

```
1 <FrameLayout
2     include_layout="@layout/error_view" />
```

Добавили. Что мы делаем дальше? Чтобы не было одновременного показа, мы даем recyclerView

```
1 android:visibility="visible"
```

...то есть ничего не поменяется, а errorview

```
1 android:visibility="gone"
```

...то есть state ошибки на этом экране как бы есть, но его как бы нет. Забавно. Переходим в recycler fragment и инициализируем view с ошибкой.

```
1 private View mErrorView;
```

В onViewCreated():

```
1 mErrorView = view.findViewById(R.id.error_view);
```

Дальше нам нужно добавить case ошибки. Мне кажется, проще это сделать следующим образом. Мы создадим новую переменную для получения случайного количества элементов.

```
1 private Random mRandom= new Random();
```

Дальше

```
1 int count = mRandom.nextInt(bound: 4);
```

4, так как будет больше шансов, что выпадет 0.

Дальше проверяем как раз-таки на ноль, если ноль, то показываем ошибку. Можем даже оформить это в виде отдельного метода.

```
1 if(count==0) {
2     showError();
3 } else {
4     showData();
5 }
```

И вне зависимости от того, случилась ошибка или нет, мы набираем индикаторы прогресса. Нужно создать методы showError и showData.

Alt+Enter → Create method → 'showError' в RecyclerFragment'.

Что он из себя представляет:

```
1 private void showError() {
2     mErrorView.setVisibility(View.VISIBLE);
3     mRecycler.setVisibility(View.GONE);
4 }
```

ErrorView мы делаем видимым, а Recycler невидимым. Соответственно с showData то же самое, но наоборот. ErrorView мы делаем невидимым, а Recycler делаем видимым.

```
1 private void showData() {
2     mErrorView.setVisibility(View.GONE);
3     mRecycler.setVisibility(View.VISIBLE);
4 }
```

Помимо всего прочего, нужно добавить данные на экран Recycler'a. Вместо count'a мы уже передаем нашу переменную, и переменную неплохо было бы передавать в сам метод showData.

```

1  private void showData (int count) {
2      mMockAdapter.addData(MockGenerator.generate(count), refresh: true);
3      // содержимое
4  }

```

Как-то так. Давайте протестируем нашу логику. Ждем, пока загрузится. Куча конфликтов. Красный цвет в иконке – плохая идея. Один. Ошибка. Да, все работает как надо, за исключением момента, где я недосмотрел, что нельзя передавать ресурс цвет в сам вектор. В целом логика работает, в принципе можете пользоваться. Это один из способов. Можно также менять фрагмент на другой фрагмент. Можно state ошибки добавлять в виде нового ViewHolder'a в адаптер, ну или просто показывать диалог с ошибкой. Это уже на ваше усмотрение. Что у нас по плану? Добавить обновление данных и состояние ошибки. Сделано.

Хорошо. В следующем видео мы обсудим добавление данных с телефонной книги. Это будет происходить с помощью системного ContentProvider'a и с помощью Loader'ов. Встретимся в следующем видео.

1.1.6. Добавление ContentProvider, CursorLoader, показ контактов в RecyclerView

В этом видео рассмотрим загрузку данных из телефонной книги. Загружать мы будем с помощью ContentProvider'a – до телефонной книги иначе не достучаться, и при помощи Loader, посмотрим, как этот механизм вообще работает. Переходим к RecyclerFragment'у и добавляем LoaderCallbacks в самом начале, передавая курсор в качестве generic-параметра. К курсору стоит относиться как к таблице.

```

1  class RecyclerFragment extends Fragment implements
   → SwipeRefreshLayout.onRefreshListener, LoaderManager.LoaderCallbacks<Cursor> {
2      // содержание
3  }

```

В onCreateLoader() мы создаем Loader и передаем его в систему. Она уже сама его запустит, обработает запрос и передаст его в onLoadFinished().

Пишем:

```
1  return new CursorLoader(getActivity(), ContactsContract.Contacts.CONTENT_URI, new
2      → String[]{ContactsContract.Contacts._ID, ContactsContract.Contacts.DISPLAY_NAME},
3  selection: null, selectionArgs: null, ContactsContract.Contacts._ID
4  };
```

Окей, мы создали курсор и передали его в систему. Соответственно, она его в фоновом потоке обработает и вернет нам курсор и данные в onLoadFinished(). Теперь, так как мы хотим показать контакты из телефонной книги в таблице, нам неплохо было бы создать Adapter, который будет выдергивать эти данные из курсора и скармливать их Holder'у. Давайте создадим такой адаптер. New → Java Class, name: ContactsAdapter, суперкласс – Adapter, который имеет отношение к RecyclerView. Щелкаем OK, добавляем в Git, передаем generic-параметр MockHolder, потому что нам надо показать два поля, и MockHolder как раз-таки хранит эти два поля: String и int.

```
1  public class ContactsAdapter extends RecyclerView.Adapter<MockHolder> {
2  }
```

Alt+Enter → Implement methods → onCreateViewHolder(). Дальше, если посмотреть на MockAdapter, то мы хранили mMockList в качестве поля и добавляли данные в него. В ContactsAdapter мы будем хранить курсор. Вообще, onCreateViewHolder можно просто скопировать из MockAdapter.

```
1  private Cursor mCursor;
```

В getItemCount() возвращаем getCount() из курсора с условием, что курсор не равен null.

```
1  return mCursor != null ? mCursor.getCount() : 0;
```

И добавим метод, который добавляет курсор в этот адаптер. Традиционно метод, добавляющий курсор, называется swapCursor().

```
1  public void swapCursor(Cursor cursor) {
2      if (cursor != null && cursor != mCursor) {
3          if (mCursor != null) mCursor.close();
4          mCursor = cursor;
5          notifyDataSetChanged();
6      }
7  }
```

Избавляемся от MockAdapter'a, добавляем ContactsAdapter.

```
1 private final ContactsAdapter mContactsAdapter = new ContactsAdapter();  
2 ...  
3 mRecycler.setAdapter(mContactsAdapter);
```

В onLoadFinished():

```
1 mContactsAdapter.swapCursor(data);
```

Мы добавили логику создания курсора, создание Loader и добавление курсора после загрузки, но не создали сам Loader. Создавать будем в методе onRefresh(). Все, что сейчас в нем находится, мы просто комментим. Инициализируем Loader:

```
1 getLoaderManager().restartLoader( id: 0, args: null, callback: this);
```

Что еще нужно сделать? После того, как данные загружены, мы еще и убираем индикатор загрузки в SwipeRefreshLayout'e. ShowData, showError тоже можем закомментить, они нам не нужны в текущей реализации, onLoaderReset() также оставляем пустым, потому что я не планирую перезапускать Loader во время загрузки, но это не значит, что вы оставите его пустым... Готовьтесь к тестовому.

Идем в ContactsAdapter. Окей, мы запустили Loader, создали Loader, получили данные и передали их курсору. Теперь нам нужно обработать onBindViewHolder(), то есть нам нужно получить данные из курсора и скормить его холдеру, чтобы он уже их показал. Пишем:

```
1 if(mCursor.moveToPosition(position)){  
2     String name =  
3         mCursor.getString(mCursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));  
4     int id = mCursor.getInt(mCursor.getColumnIndex(ContactsContract.Contacts._ID));  
5 }
```

В принципе можно оверрайднуть метод Holder'a и передать ему строку и число, либо создать объект Mock и передать в метод bind его – по старой логике.

```
1  if(mCursor.moveToPosition(position)){  
2      String name =  
3          mCursor.getString(mCursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_  
4          NAME));  
5      int id = mCursor.getInt(mCursor.getColumnIndex(ContactsContract.Contacts._ID));  
6      holder.bind(new Mock(name, id));  
7  }
```

Дальше нам нужно попросить разрешения у пользователя заглянуть в его телефонную книгу. В манифесте пишем:

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

Если мы запустим наш проект, он обвалится с ошибкой. Почему? Потому что прямо сейчас у меня эмулятор 26 уровня, и это означает, что мне нужно запросить Runtime Permission для контактов, так как запрос контактов – это чувствительная, опасная, компрометирующая информация для пользователя. По идеи надо показать всплывающее окно с просьбой дать доступ к контактам. Чтобы не усложнять проект, я этого делать не буду, а воспользуюсь лайфхаком: перейду в настройки устройства и сам вручную включу разрешение.

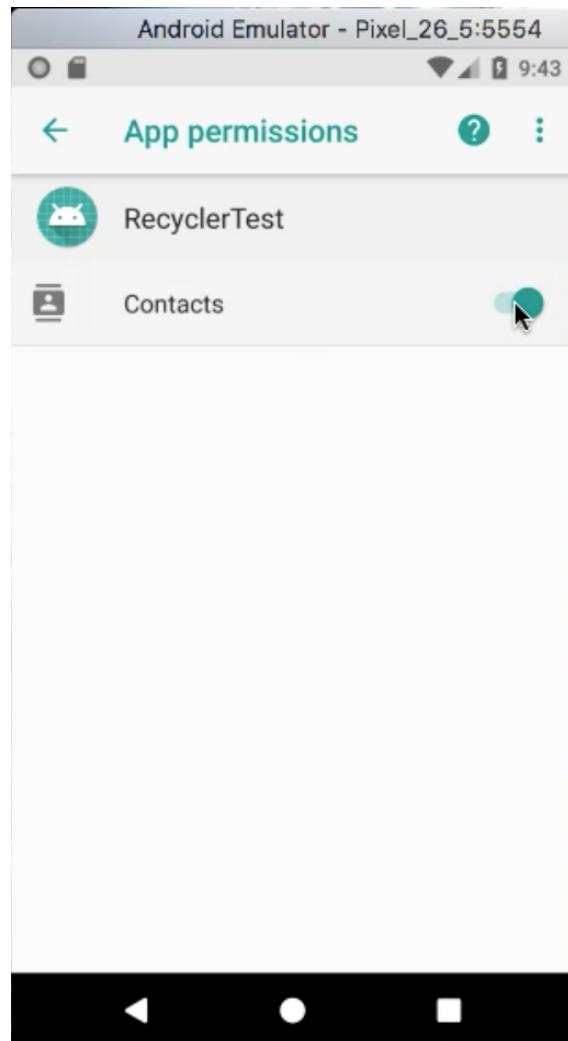


Рис. 1.13

Запустим приложение. Видим данные из телефонной книги. Добавив новый контакт, видим, что в приложении данные автоматически обновились. Насчет Runtime Permission не переживайте, чуть дальше по курсу будет материал по добавлению Runtime Permissions и по проверке Permissions в самом приложении, будем проверять, как это все работает.

В следующем видео добавим обработку нажатий. Не переключайтесь.

1.1.7. Обработка нажатий на элементы списка

В этом видео разберем, как обрабатывать нажатия на элементы списка. Меньше слов, больше дела, давайте начнем. Во-первых, перейдем в наш адаптер ContactsAdapter и определим там интерфейс:

```
1 public interface OnItemClickListener {}
```

В этом интерфейсе зададим метод.

```
1 void onItemClick();
```

Логика какая? При нажатии элемента списка вызывается метод OnItemClick, который определен в этом интерфейсе. Получается, во-первых, что этот интерфейс должен реализовывать какой-то элемент нашего кода. Во-вторых, реализация этого интерфейса должна быть передана в holder, чтобы именно нажатый элемент был обработан в месте ее реализации. Давайте вначале определимся, кто будет реализовывать этот интерфейс. Вообще желательно, чтобы реализацией занималась именно activity, потому что чем выше реализация – тем лучше, ведь реализовав интерфейс в activity, я могу менять логику, впоследствии не затрагивая остальные элементы. Например, я могу нажать на кнопку и запустить звонок. Или я могу нажать на элемент списка и поменять один фрагмент на другой. Если реализация интерфейса будет ниже во фрагменте, либо в адаптере, либо в холдере, то при изменении логики мне придется поменять очень много зависимых от этой логики классов. Поэтому реализацией интерфейса займется activity. Я надеюсь, я понятно объяснил, почему.

```
1 public class MainActivity extends AppCompatActivity implements
  → ContactsAdapter.OnItemClickListener {
2     // содержание
3 }
```

Alt+Enter → Implement Methods → onItemClick().

Хорошо. Пока что реализация ничего не будет делать. Подумаем над тем, как передать реализацию в holder. Перейдем в RecyclerFragment. Удалим код, который остался с прошлого раза. RecyclerFragment является своего рода прослойкой между activity и адаптером. В прошлом курсе мы разбирали, как передавать данные и логику из фрагмента в activity. Посредством каста в методе attach и зануления в методе onDetach.

Точно так же создадим поле с listener'ом и будем инициализировать его в методе onAttach.

```
1 private ContactsAdapter.OnItemClickListener mListener;
2 public void onAttach(Context context) {
3     super.onAttach(context);
4     if(context instanceof ContactsAdapter.OnItemClickListener) {
5         mListener = (ContactsAdapter.OnItemClickListener) context;
6     }
7 }
```

В onAttach'е мы записали значения в поле, соответственно, в методе onDetach мы это поле зануляем.

```
1 mListener = null;
```

Это чтобы не мешать сборщику мусора. Listener теперь передан из activity во fragment, нам нужно по цепочке передать его в адаптер.

```
1 mContactsAdapter.setListener(mListener);
```

Щелкаем по выделенному методу, Alt+Enter и выбираем Create Setter. Он помимо самого метода создаст еще и поле listener – одна из фишек Android Studio.

Раз listener передан в adapter, теперь его нужно передать в holder. Как мы можем это сделать? Мы можем переопределить метод bind, либо задать его отдельным методом:

```
1 holder.setOnClickListener(mListener);
```

Alt+Enter. В этом случае мы выбираем Create Method, потому что я не хочу, чтобы в Holder'e хранилась ссылка на listener. Что мы делаем дальше? Будучи в Holder'e, мы можем обратиться к самому элементу списка, не к ее части, а ко всему элементу целиком. Чтобы это сделать, нам нужно вызвать поле itemView. Если посмотреть на конструктор Holder'a, то видно, что оно задавалась в конструкторе. Итак, itemView. И так же как с любым view элементом, задаем ему onClickListener стандартный:

```
1 itemView.setOnClickListener (new View.OnClickListener() {  
2     @Override  
3     public void onClick(View v) {  
4         //  
5     }  
6 }) ;
```

И в методе onClick пишем:

```
1 listener.OnItemClick();
```

То есть при нажатии на элемент списка вызовется метод onItemClick переданного onItemClick listener'a. Давайте перейдем в main activity и зададим логику.

```
1 public void onItemClick() {  
2     Toast.makeText (context: this, text "clicked",Toast.LENGTH_SHORT).show();  
3 }
```

Запускаем. Обновили список, щелкнули на элемент из списка и видим, показывается toast. Хорошо. Теперь что нам нужно сделать? Нам нужно передать id щелкнутого элемента, чтобы я мог отличить, что я щелкнул либо на первый элемент, либо на четвертый, либо на шестой. Сейчас это не сделано. Переходим к определению нашего интерфейса. Наводимся на метод. Нажимаем Shift+F6 и сигнатуру. Вообще я думал, что сработает автозамена, но она не сработала. Переходим в MockHolder и видим, что onItemClick теперь на вход должен получать id. id мы можем получить либо из поля TextView mValue, либо сохранить его в отдельное поле.

```
1 private String mId;  
2 mId = mock.getValue();
```

Соответственно, в onItemClick передаем mId. RecyclerFragment у нас без эффектов, а вот в main activity нужно добавить аргумент метода onItemClick и попробовать проверить, как оно работает теперь. Добавляем теперь в сообщение, которое должно вывестись при toast'e, id, которое мы передали из холдера. Щелкаем run, обновляем, щелкаем на первый элемент – видим clicked 1, на второй – clicked 2, на шестой – clicked 6. Все работает как надо.

Возвращаемся к нашей activity. Теперь добавим какую-нибудь логику обработки нажатия на элемент списка, которая отличается от показа toast'a, сделаем так, чтобы при нажатии на имя

контакта запускался звонок этому контакту. Проблема в том, что у нас нет номера телефона. Дело в том, что ContentProvider, который мы использовали в прошлом уроке, не содержит номера телефона. Чтобы получить номер телефона, нам нужно обратиться по другому content URI. И в этом видео для сравнения я не буду добавлять Loader'ы, а просто обращусь к content-провайдеру напрямую.

Как мне это сделать? Я просто пишу getContentResolver и получаю контент провайдер. Вызываю метод query. Если посмотреть на сигнатуру метода, то она кажется подозрительно знакомой. Неудивительно, потому что query возвращает курсор и создание CursorLoader'a похоже на запуск метода query. Тот же URI, тот же projection, selection, аргументы, sort order. Я думаю, понятно, почему. Давайте заполнять.

```
1   getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI);
```

Первый аргумент у нас есть. Дальше projection, то есть какие столбцы у нас есть. Мне нужен только один столбец, это столбец с номерами телефонов, но мне нужно обернуть этот столбец в массив, потому что аргумент метода этого требует. Поэтому я пишу new String и создаю массив из одного элемента.

```
1   new String[]{ContactsContract.CommonDataKinds.Phone.NUMBER}
```

Дальше selection. Мне не нужны все номера из этой таблицы, мне нужен только номер конкретно того человека, на которого я щелкнул, и для этого мне нужно сравнить id из текущей таблицы, из текущего ContentProvider'a с тем id, который мне пришел в метод onItemClick.

```
1   ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " =?"
```

Вместо вопросительного знака при запуске метода подставится элемент из следующего аргумента из selectionArgs. Мне нужен именно мобильный телефон, поэтому я добавлю еще один атрибут для сравнения в строку selection.

```
1   ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ? AND " +
    ↳ ContactsContract.CommonDataKinds.Phone.TYPE + " = ?"
```

С selection'ом определились. Теперь selectionArgs, то есть значения, которые потом подставятся вместо вопросительных знаков в строку selection. Это тоже массив, пишем new String. Первым

элементом будет id, который мы получили на входе, вторым элементом – mobile, который можно найти в том же CommonDataKinds.

```
1 new String[]{id, ContantContract.CommonDataKinds.Phone.TYPE_MOBILE}
```

TYPE_MOBILE – это число, нужно кастануть его к строке.

```
1 new String[]{id, String.valueOf (ContantContract.CommonDataKinds.Phone.TYPE_MOBILE)}
```

И последним аргументом в этот метод является sort order, передаю null.

```
1 new String[]{id, String.valueOf (ContantContract.CommonDataKinds.Phone.TYPE_MOBILE)},  
2 sortOrder: null);
```

Метод готов. Что мы имеем в конечном итоге? Мы создали ContentProvider. Обращаемся к нему, получаем курсор, который будет хранить один столбец, номер телефона, в котором мы сравниваем id, который находится в этом столбце, с id, который мы ему передали. И сравниваем тип мобильного телефона с TYPE_MOBILE. Звучит сложно, выглядит еще сложнее, но я думаю, вы разберетесь.

Кстати, query возвращает нам курсор, загоним его в переменную. Наводимся на query. Cmd+Alt+V. Либо, если вы работаете на Windows, Ctrl+Alt+V. Получили переменную, дальше query назовем ее cursor. Проверяем.

Кстати, у курсора есть method moveToFirst, который работает как moveToPosition, только переносит его к первому элементу таблицы. Я полагаю, что у меня будет только один номер такого типа, и перемещаюсь к самому началу.

```
1 if(cursor!=null &&_cursor.moveToFirst()){  
2     String number =  
3         ↗ cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds  
4             .Phone.NUMBER));  
5 }
```

Получили номер, закрыли курсор, и теперь нам нужно создать intent для запуска приложения, для звонков.

```
1  startActivity(new Intent(Intent.ACTION_CALL).setData(Uri.parse("tel:"+number)));
```

Этого кода должно быть достаточно для запуска приложения с полученным номером телефона. На самом деле нет. ACTION_CALL требует Permission, так же как и READ_CONTACTS. Запрашиваем Permission. Скорее всего, приложение снова обвалится, так как я не проверяю, что у меня есть этот Permission. Воспользуемся лайфхаком и запросим этот Permission прямо в настройках. Обновили, щелкаем по первому элементу, и запускается приложение для совершения звонков.

Итак, что мы сделали, чего мы добились в этом уроке. Мы передали данные из Holder в activity путем реализации интерфейса, который определен в адаптере и передаче listener'a из activity во фрагмент, из фрагмента в адаптер, и из адаптера в holder. Прямая связь. Получается, вызов этого интерфейса приводит к тому, что срабатывает код в main activity, где этот интерфейс реализован, и это обратная связь.

В методе onItemClick реализованного интерфейса мы обращаемся к ContentProvider'у, ищем номер телефона для контакта, id которого мы передали в этот метод, следим за тем, чтобы номер телефона был мобильным, проверяем, что полученный курсор непустой и в нем есть данные. Выдергиваем оттуда номер телефона. Создаем неявный intent для звонков, указываем в нем данные, номер телефона, по которому мы хотим позвонить, и запускаем activity.

В чем проблема этого кода, если не акцентироваться на том, что я не проверяю Permission в Runtime'e? Проблема в том, что я обращаюсь к ContentProvider'у, делаю запрос в базу, проверяю, выдергиваю данные, закрываю курсор, и все это я делаю в main thread'e. Это неправильно. Я должен был обратиться к ContentProvider'у в фоновом потоке, провести все эти манипуляции, и когда все будет готово, просто возвращать номер телефона в главный поток, с которого я мог бы уже позвонить. И все это из-за того, что я отказался использовать Loader'ы. В принципе это можно сейчас исправить, создать сервер-executor, например, выполнить в нем этот task, получить future, из future выдернуть строку с номером, передать ее в intent, в который мы передаем start activity. Но я не буду этого делать, потому что это будет одним из ваших тестовых заданий. Так, обработку нажатий мы тоже реализовали. В следующем уроке мы поговорим о декораторах. Что это такое, с чем это едят и вообще зачем это нужно. До встречи.

1.1.8. Добавление декораторов

Приветствуя на очередном уроке, посвященном RecyclerView и работе со списками. В этом видео мы обсудим декораторы. Прежде чем я объясню, что такое декораторы, давайте посмотрим на наш список. Мы видим, что список есть, его можно обновлять, на элементы можно тыкать, вызывать всякого рода звонки. При всем этом наш список не выглядит как список. Он какой-то безликий. Если человек не знает заранее, что это список, то он даже не поймет, что это список. Давайте мы сделаем его красивее, добавив разделительную линию между элементами списка.

Добавить разделительную линию мы можем двумя способами: легким и через декораторы. Я сначала покажу легкий способ. Легкий способ заключается в том, что мы добавляем разделительную линию на сам элемент списка, то есть мы открываем наш элемент списка listitem_mock и просто добавляем разделительную линию на саму разметку.

```
1 <View  
2     android:background="#4444"  
3     android:layout_width="match_parent"  
4     android:layout_height="1dp" />
```

Background color сделаем темно-серым.

Видим разделитель на экране. У TextView второй добавим паддинг, чтобы разделитель не был привязан к элементу.

```
1 android:layout_marginBottom="8dp"
```

Дальше убираем паддинг у LinearLayout, у самого холста. Добавляем margin к TextView.

```
1 android:layout_marginTop="8dp"  
2 android:layout_marginStarts="8dp"
```

И у второго элемента тоже:

```
1 android:layout_marginStarts="8dp"
```

Получили элемент списка, в самой верстке которого находится разделительный элемент. Давайте попробуем запустить приложение и посмотрим, что будет. Бам. Теперь этот список больше похож

на список.

Давайте теперь рассмотрим работу с CardView. Сделаем наш список еще красивее. Переходим во вкладку Design, находим в палитре в категории AppCompat CardView и просто перетаскиваем на наше дерево компонентов. Я делаю это для того, чтобы зависимость CardView добавилась в проект. На самом деле CardView не будет добавлена в таком виде. Из CardView мы сделаем обложку, холст. Удаляем. Создаем. Обернули наш элемент списка в CardView. Зачем? CardView – это компонент из библиотеки поддержки, который делает списки более красивыми. Возможно, вы видели, сейчас даже покажу.

```
1 android:layout_margin="8dp"
2 app:cardElevation="4dp"
```

И удалим наш divider. Он нам не нужен. Что еще мы можем сделать? У CardView задать background белый:

```
1 android:background="#FFF"
```

А у элемента списка задать background selectableItemBackground из атрибутов:

```
1 android:background="?attr/selectableItemBackground"
```

Что получится? Давайте посмотрим.

xmlns нам здесь не нужен, удалим. Запускаем. Обновляем список. Видим, что наш список теперь в виде карточек. Нажимаем на список, видим красивую анимацию. Слишком много звонков, не успевает обрабатывать. Это следствие того, что CursorLoader работает в главном потоке.

Что я только что сделал? Я только что показал, как в разметке самого элемента списка добавлять декорацию, то есть как-то изменять его. Я добавил в начале разделительную линию через ViewDivider, указал размеры, фоновый цвет, затем обернул элемент списка в карточку, в ней указал margin'ы, что привело к тому, что у карточки появились красивые отступы по бокам и указал elevation, что добавило тень под нашей карточкой. Дальше я указал цвет у карточки отдельно и цвет у LinearLayout'a отдельно. Это был простой способ декорирования списков.

Сложный способ в добавлении конкретного класса, который наследуется от RecyclerView.ItemDecorator. Мы добавляем реализацию класса в RecyclerView, что приводит к тому, что наш список преображается. Само создание декораторов – сложный процесс. Дело в том, что рисование в

нем происходит на более низком уровне, чем в том, чем мы только что занимались. Поэтому я сейчас скопирую готовый класс декоратора и мы просто пройдемся по его коду. Я скопировал класс CardDecoration. Прежде чем мы начнем его разбирать, давайте отменим изменения в элементе списка. Все мы отменять не будем, давайте только уберем отступы по бокам главной карточки. Я думаю, этого должно быть достаточно, чтобы работа декоратора была заметна. Так, layout_margin у карточки убрали. Хорошо. Возвращаемся в CardDecoration и рассмотрим этот класс. Во-первых,

```
1 CardDecoration extends RecyclerView.ItemDecoration
```

то есть собираемся добавить декорацию в RecyclerView, логично предположить, что RecyclerView должен знать про этот класс. Давайте, кстати, сделаем это прямо сейчас. Переходим во фрагмент с Recycler'ом. Находим объявление Recycler'a, пишем

```
1 mRecycler.addItemDecoration(new CardDecoration());
```

Все, возвращаемся в наш класс. Что мы здесь видим в переменных? Видим два поля типа paint, то есть краска. Желтая и красная. Соответственно в конструкторе у нас идет настройка этих объектов. Задается стиль. fill, что значит полностью закрашен. Задается сглаживание, цвет. То же самое для красной краски. Дальше у нас есть три метода: onDraw, onDrawOver и getItemOffsets. onDraw и onDrawOver закомментированы, их разберем позже.

Разберем сначала getItemOffsets. Это низкоуровневый код, суть которого заключается в том, что прямоугольник, который представляет собой элемент списка, получает offset'ы, то есть отступы. То же самое, как если бы мы задавали margin в самом элементе списка, но при этом низкоуровнево.

R.dimen.li_margin ссылается на resource, это 8dp. Так как здесь измерение идет в пикселях, я беру не просто dp, я беру dp, переведенную в пиксели getDimensionPixelOffset. Считаю левый край прямоугольника, outRectangle, вот он. Правый, верхний, нижний. Сейчас запускаем заново. Должна сработать декорация, которую мы добавляли в программу. Студия ругается, требует пересборки проекта, скорее всего из-за того, что я копировал класс.

Давайте сделаем Rebuild и запускаем. Обновляем. Да, видим картину. Изменения незаметны, но они все-таки есть. Если приглядеться, то отступы между элементами списка чуть меньше, чем были до этого, при этом если сейчас убрать декорацию, закомментировав эту строку, то мы получим просто список.

Запустим. Одна большая карточка, сейчас без декорации. Возвращаем декорацию, получаем отступы, то есть для каждого элемента был высчитан свой отступ и применен. Вернемся к коду декорации и рассмотрим два других метода – `onDraw` и `onDrawOver`. Раскомментим.

`onDraw` вызывается перед добавлением списка в `Recycler`, то есть вначале выполнится код в декорации в `onDraw`. На самом деле у `Recycler`'а может быть несколько декораций, и у каждой выполнится метод `onDraw`. Затем сверху добавится элемент списка, после этого выполнится `onDrawOver`.

Сейчас на самом деле будет адовый психодел, потому что по коду мы берем границы, нижнюю и верхнюю, и закрашиваем прямоугольник, где хранится элемент списка, в желтый или красный цвет в зависимости от четности или нечетности. Щелкаем ОК. Обновляем. Видим, с начала было закрашено поле, потом был добавлен элемент списка. Вот такой вот психодел.

Теперь рассмотрим метод `onDrawOver`. Он вызывается уже после того, как элемент списка был добавлен. Это приводит к тому, что рисунок будет нанесен на элемент списка, поверх него. Мы рисуем прямоугольник примерно по центру карточки, причем если подложка была желтой, то рисунок сверху будет красным и наоборот.

Давайте запустим и посмотрим. Вот, получили. Подложка желтая, рисунок сверху красный. Подложка красная, рисунок сверху желтый. Это и есть декораторы. Если посмотреть на этот код, то если кто-то спросит вас, что лучше, декораторы или разделители на самой разметке, то на первый взгляд ответом является второе. Проще, быстрее, понятнее добавить разделители на саму разметку. Добавить отступы на саму разметку элемента списка. В чем же преимущество декораторов? Преимущество в том, что, во-первых, у вас может быть несколько списков на экране, которым нужно приделать одно и то же изменение, один и тот же декоратор. Например у вас везде отступы одинаковые, либо сверху закрашивается что-то одинаково, бейджик внизу справа от элемента списка. Чтобы не городить кучу разных элементов списка с одинаковыми разделителями, с одинаковой декорацией, мы придумали декораторы. В этом их преимущество, преимущество в переиспользовании.

Другое преимущество. Если в `Recycler`'е я захочу в `Runtime` удалить элемент списка, то вместе с ним удалится и разделитель, который был на нем. И при `Swipe`'ах это будет заметно, и при удалении, и при добавлении. В то время как декоратор никуда не денется. Он просто изменится.

Снова откроем код декоратора. Смотрите. `onDraw` получает на вход `canvas`. То есть сам холст. Это очень низкоуровневый код. Я сейчас не углубляюсь в это, потому что в одном из следующих курсов, когда мы будем рассматривать `material design`, анимации и кастомные `view`-элементы, будет уделено достаточно много внимания низкоуровневому рисованию – рисованию собственных

элементов, где тоже вызываются методы `onDraw`, `onMeasure` и тому подобное.

Сейчас код вы получите, посмотрите, как это работает, поковыряйтесь, рассмотрите код встроенного декоратора, сейчас я его покажу. Он просто рисует разделительную линию, вертикальную или горизонтальную. Это необязательно, но попробуйте разобраться, как это работает. Дальше, если у вас приложение маленькое, никто не будет ругать за то, что вы добавили разделитель в самой разметке. Ничего страшного в этом нет. Но если планируется, что у вас будет несколько элементов списка, большое приложение, одинаковые декораторы, какие-либо рисунки поверх элементов списка – те же бейджики или сердечки, кто знает – тогда лучше углубиться в тему декораторов и использовать именно их. В маленьких проектах добавление разделителей или `offset'`ов в самом элементе списка это нормально, не страшно, но как только речь заходит о серьезных проектах с анимацией, жесткими изменениями в Runtime, с несколькими списками, то все-таки лучше использовать именно декораторы.

Итак, на этом уроке работа со списками завершена. Впереди вас ждет несколько тестовых заданий на RecyclerView, на адаптер, на Loader'ы. Что еще мы не рассмотрели? Мы не рассмотрели анимацию списков. Это понятно, потому что введен специальный курс. Еще мы не рассмотрели гетерогенные списки, то есть списки, в которых находится несколько различных элементоов, несколько ViewHolder'ов. Будет теория и будет практическое задание по нему. Также мы не рассмотрели добавление и удаление элементов в Runtime. Тоже будет теория по этому и тоже будет практическое задание, так что не рекомендую вам расслабляться. Что ж, тогда до встречи.

1.2. Работа с файлами

1.2.1. Способы хранения данных в Android (Preferences, Sqlite+Room, Файлы)

В каждом более-менее серьезном приложении должна быть предусмотрена возможность сохранять данные между запусками. Эту возможность сохранения, процесс хранения, а также место хранения можно выразить одним емким английским словом – **persistence**.

Вариантов у нас 3:

- SharedPreferences
- Файлы

- БД/ORM

Выбор способа хранения зависит от решаемой задачи. Какие данные мы хотим сохранить?

SharedPreferences

Если речь идет о **примитивных данных – булевых, численных или строках**, то очевидным вариантом является использование SharedPreferences. Мы уже разбирали SharedPreferences в первом курсе, так что задерживаться не будем.

1. <https://developer.android.com/reference/android/content/SharedPreferences.html>
2. <https://developer.android.com/training/data-storage/shared-preferences.html#java>

Файлы

Если нам нужно сохранить **неструктурированные данные** – такие как изображения, музыка, большой текст, книга, то, очевидно, хранить нужно в виде файлов.

Android достаточно гибок в этом плане и позволяет создавать и хранить файлы с некоторыми различиями в доступе.

Давайте разберемся с понятиями Internal Storage и External Storage.

Internal Storage – внутреннее хранилище данных. Это подпапка, в которую ваше приложение может записывать файлы. Другие приложения к этой папке не имеют доступа (до 7.0 была возможность указать открытый доступ internal файлам, но сейчас это делается через специальный класс FileProvider). Пользователь эту папку не видит. При удалении приложения папка тоже удаляется. Внутреннее хранилище всегда доступно. Для чтения/записи во внутреннее хранилище никаких разрешений не требуется.

И так как приложения по умолчанию устанавливаются в Internal Storage, не следует злоупотреблять хранилищем и записывать туда большие объемы данных. Вместо этого следует воспользоваться External Storage.

External Storage – внешнее хранилище данных. И это не обязательно SD-карта. Внешнее хранилище данных может быть несъемным. Закрепим.

Всякая съемная SD-карта является внешним хранилищем, но не всякое внешнее хранилище является съемной SD-картой.

Внешнее хранилище может быть извлечено, и тогда к файлам не будет доступа. Файлы, записанные в external storage доступны всем приложениям, пользователю, а при подключении устройства к компьютеру в режиме USB-накопителя – и компьютеру тоже. Их можно переносить, переименовывать, удалять. Защитного механизма нет.

Далее в курсе мы рассмотрим запись и чтение файлов оба хранилища, а также чтение файла из res/raw и assets папок проекта.

Базы данных и ORM-обертки

Если нам нужно сохранить **структурированные данные**, то лучшим выбором станет база данных.

Android полностью поддерживает **SQLite** – встраиваемую реляционную базу данных. И работа с SQLite идентична работе с любой другой SQL базой: есть таблицы, состоящие из столбцов/полей и строки – непосредственно сама запись. Есть уникальные идентификаторы, внешние ключи, поддержка запросов и тому подобное. Скорость работы с голым SQLite, понятное дело, эталонная.

Другое дело, что работа с SQLite происходит на низком уровне – нет проверки валидности SQL-запросов во время компиляции, запросы нужно писать вручную, опечататься или ошибиться очень легко. Все это в конце концов переходит огромное количество рутинного кода и человекочасов. Разработчики это понимали и теперь в нашем распоряжении огромное количество оберток, упрощающих создание и обработку данных в базе. Эти обертки называются ORM, что расшифровывается как Object-Relational Mapping и переводится как объектно-реляционное отображение. То есть ORM – это своего рода прослойка между нашим приложением (вызывающей стороной) и базой данных. Работа с ORM происходит как с обычными объектами. Мы не вникаем в тонкости устройства БД. ORM-библиотек достаточно много, все они отличаются реализацией БД и скоростью доступа к данным и временем выполнения типичных операций – чтения, записи, удаления и т.д.

В этом курсе мы разберем Room – ORM, представленную Google. Но большинство современных

ORM работает по тому же принципу – через создание обычных java-классов и маркировку их аннотациями.

1.2.2. Чтение данных из assets/raw

Мы уже знаем, что ресурсы, которые не подходят ни под один из стандартных типов можно хранить в папках res/raw/ либо в asset'ах. Давайте же теперь чуть подробнее разберем, в чем разница между двумя этими вариантами хранения и научимся читать файл оттуда.

Плюс assets в том, что файлы можно группировать по папкам. Если в проекте их много, то наверняка имеет смысл добавлять директории для наших файлов. raw, напротив, как и любая другая директория res, хранит все свои файлы на одном уровне, но, как и для других ресурсов, к raw-файлам можно обращаться по id, например: r.raw.sample. То есть уже на этапе компиляции система знает, что есть такой или иной файл в raw директории и генерирует для него id. Файлы в asset'ах, в свою очередь, не снабжаются id, обращение к ним происходит по строковым названиям, что приводит к возможности опечататься и далее к ошибке.

С теорией разобрались. Теперь давайте разберем кейсы, когда нам может пригодиться хранить данные в raw или в asset'ах. Допустим, у нас есть огромное полотно текста, например, лицензионное соглашение. Конечно, в большинстве нормальных приложений лицензионное соглашение открывается в браузере. Предположим, что мы садисты и хотим показать соглашение прямо в приложении. Из-за огромного количества текста просто неблагодарным было бы хранить его в string'ах. Мы поступим проще. Мы создадим raw-директорию и в ней текстовый файл. Вставим туда лицензионное соглашение. Вот оно.

Теперь попытаемся считать данные из этого файла и вывести их в TextView. На разметке у меня определен TextView, который обернут в ScrollView, что позволит нам его скроллить. Для того, чтобы считать данные из raw-директории нам нужно вызвать метод openRawResource, в который мы передаем id нашего raw-файла, который мы хотим открыть. Видим: getRawResource, openRawResource, на вход принимает rawId и на выходе InputStream. Вызов происходит в этом месте. Полученный InputStream мы передаем в метод getStringFromRawResource. Дальше, стандартно, как и в большой Java InputStream оборачивается в InputStreamReader, который обрамляется в BufferedReader, создается StringWriter и далее, строчка за строчкой, считывается строка из Reader'a и записывается в Writer. В конце концов из StringWriter'a собирается строка с помощью метода toString и возвращается в вызывающую сторону. Мы получаем строку license. Далее эту строку мы передаем в качестве текста в TextView.

Давайте запустим приложение и посмотрим, как оно сработает. Итак, мы получили огромное полотно текста, мы на это и рассчитывали. Все скроллится, все нормально, то есть файл из res/raw/ благополучно считался и отобразился в activity. Profit.

Теперь давайте рассмотрим еще один кейс использования таких файлов. Допустим, мы хотим замокать результат сетевого запроса, сделать что-то вроде тестового сервера прямо в нашем приложении. Тогда мы храним результат в виде json-файла и во время имитации запроса мы не лезем в сеть, а просто считываем данные из нашего файла. В этот раз мы расположим наш файл в asset'ах. Asset response. Я скопировал стандартный json-файл из ISO-документа. Для открытия файла из asset'ов нужно вызывать метод getAssets и на нем вызвать метод open. Open получает на вход строковое название файла и бросает IOException, соответственно на выходе из этого метода мы точно так же получаем InputStream, который мы передаем в знакомый нам метод getStringFromRawResource. Точно так же, все абсолютно то же самое, считаются данные из файла. Так как json response и вообще мы хотим сымитировать работу сервера, то я подключу библиотеку Gson и стандартным образом собираю из строки response class. Он расположен тут и состоит из нескольких типов, которые я сгенерировал с помощью сайта json2pojo. А далее я просто вызываю toast, в котором показываю

```
1 response.getGlossary().getTitle()
```

Давайте перезапустим и посмотрим, что теперь произойдет. По идеи должен показаться toast с данными из модельки. Example glossary, toast у нас есть. Идем в наш response и видим, что есть такая строка, можем даже ее тут поменять, написать что-то вроде интеллектуального набора рандомных букв.

Перезапустим. Все работает. Итак, мы обработали два кейса, которые могут возникнуть у вас в работе. Это mock-серверы и считывание большого текстового файла. Возможно, есть еще какие-то кейсы. Работа с файлами из asset'ов и raw ничем не отличается от работы с большой Java. Если вам это дополнительно интересно, просто найдите какую-нибудь книгу по Java Core и прочитайте про работу с файлами. И на этом у меня все.

1.2.3. Runtime Permissions

Вплоть до появления Android M все системные разрешения (на использование интернет-трафика, камеры, GPS и т.д.) запрашивались при установке приложения. Начиная с 6-ой версии появилась возможность запросить их в тот момент, когда они действительно потребовались, напри-

мер, перед запросом на работу с камерой. То есть теперь пользователь может сам выбирать, что разрешать, а что нет, не принимая все разрешения скопом, и лишь лишаясь некоторой функциональности, запретить приложению доступ к чему-либо (звонкам, смс камере, и т. д.).

Стоит отметить, что разрешения делятся на 2 типа:

- обычные (normal);
- опасные (dangerous).

Давайте разберемся, что это значит.

Обычные разрешения будут получены приложением при установке (то есть работают как раньше). В дальнейшем отозвать их у приложения будет невозможно.

Опасные же должны быть запрошены в процессе работы приложения и в любой момент могут быть отозваны, поэтому нужно каждый раз проверять, что у приложения есть доступы, вызывая метод

```
1 public static int checkSelfPermission(@NonNull Context context, @NonNull String  
→ permission)
```

который возвращает нам одно из int значений: **PackageManager.PERMISSION_GRANTED** в случае, если разрешение есть или **PackageManager.PERMISSION_DENIED**, если его нет.

Иначе, если не выполнить проверку, то при отсутствии разрешения приложение может просто упасть с `java.lang.SecurityException`.

Для того, чтобы вызвать системный диалог с запросом разрешений нужно вызвать метод. Если мы хотим, например, запросить доступ к чтению/записи во внешнее хранилище, то вызовем метод `requestPermissions()` с этими параметрами:

```
1 ActivityCompat.requestPermissions(this, new String[]  
2 {Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE);
```

`this` – `Activity`, на которой показывается диалог. `new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}` – список разрешений, не может быть пустым или null.

При повторном запросе разрешения, после того, как пользователь уже отказал ранее, появляется checkbox «Never Ask Again»:

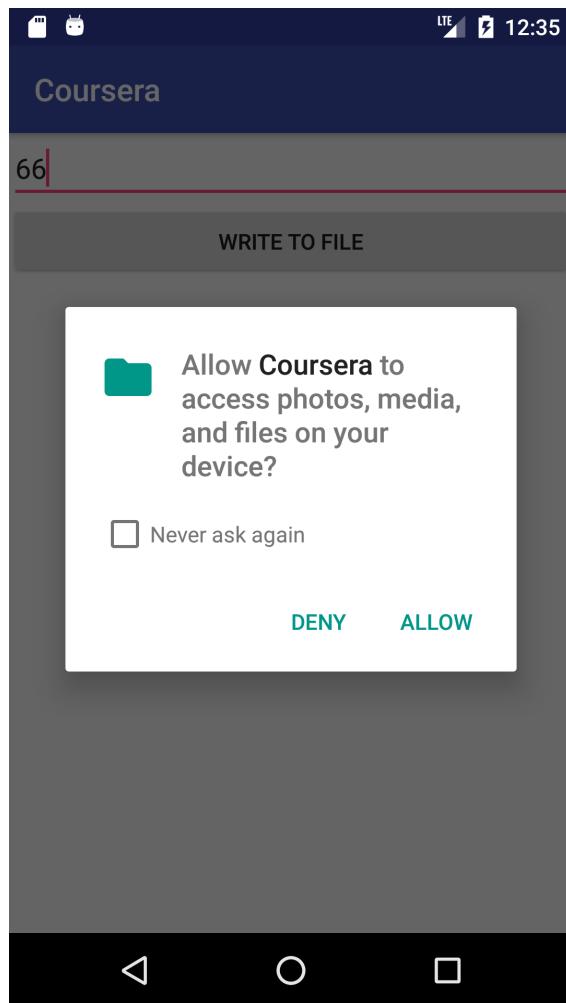


Рис. 1.14

В случае, если пользователь выберет эту опцию, диалог более показываться не будет, `shouldShowRequestPermissionRationale()` будет выдавать `false`, а в `onRequestPermissionsResult()` будет получен результат `PERMISSION_DENIED`. Теперь получить разрешение можно только самостоятельно, зайдя в системные настройки устройства и выбрав нужное приложение. Если функ-

циональность приложения страдает из-за отсутствия разрешения, то об этом также следует сообщить пользователю.

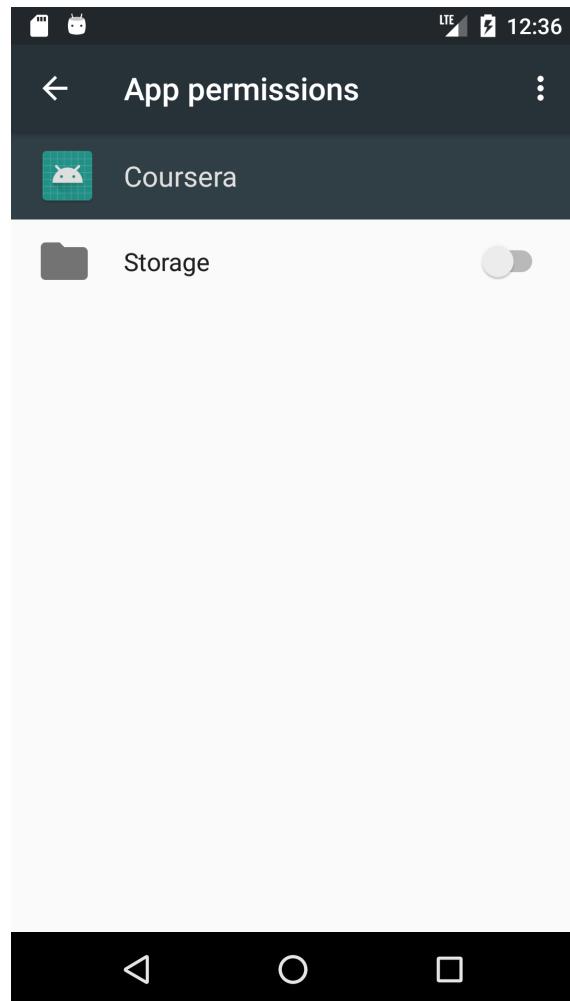


Рис. 1.15

Также хотелось бы осветить метод `ActivityCompat.shouldShowRequestPermissionRationale()`, который возвращает boolean, стоит ли нам показать UI с объяснением, почему приложение требует это разрешение. Ниже приводится небольшая блок-схема, демонстрирующая принцип его работы.

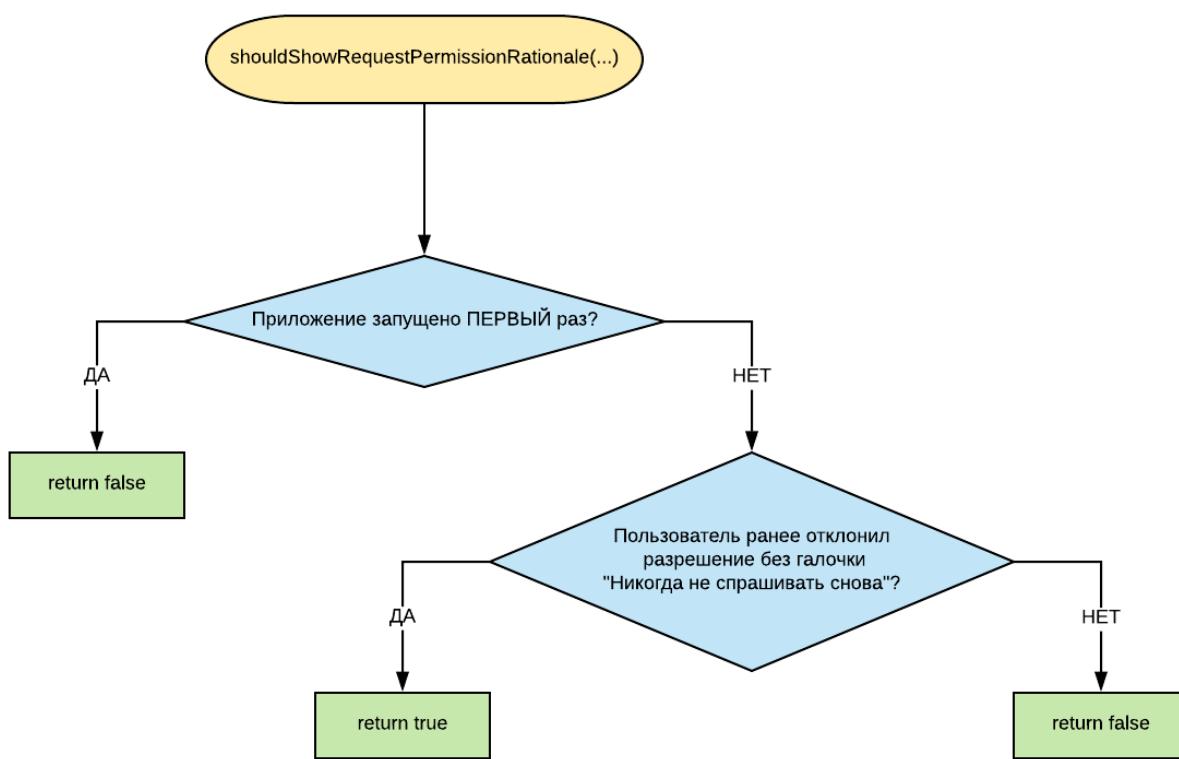


Рис. 1.16

Обязательно к прочтению, если вы хотите стать тем, кто будет знать больше нас:

1. <https://developer.android.com/training/permissions/requesting.htmljava>
2. <https://www.kaspersky.ru/blog/android-permissions-guide/14099/>
3. <https://xakep.ru/2016/01/26/android-permissions-2/>
4. <https://habrahabr.ru/post/278945/>

1.2.4. Запрос Runtime Permissions

В этом видео мы рассмотрим Runtime Permission'ы. И давайте я создам план. Первым делом мы проверяем, что у нас есть Permission. Проверка состояния разрешения. Второе. Запрос непосредственно разрешения, если его нет. Показ объяснения, зачем нам вообще нужно разрешение, если оно нужно. И четвёртое – это обработка ответа на разрешение, то есть мы показываем пользователю запрос, он либо принимает его, либо отклоняет. Обработать ответ нужно соответственно в своём методе. Обработка ответа на запрос разрешения.

Хорошо. План у нас есть. Давайте зайдем в activity_main. Посмотрим, что тут вообще происходит. ConstraintLayout и Hello World, то есть стандартное приложение, стандартная разметка, которую мы получаем при первом запуске приложения. ConstraintLayout я меняю на LinearLayout для удобства, чтобы не маяться с constraint'ами. Надо закрыть. Вместо TextView у меня будет EditText.

```

1 <EditText
2     android:id="@+id/et_input"
3     android:layout_width="match_parent"
4     android:layout_height="wrap_content" />
5 <Button
6     android:id="@+id/btn_write"
7     android:layout_width="match_parent"
8     android:layout_height="wrap_content" />
```

Ориентация вертикальная. Давайте запустим. Что я хочу написать? Я хочу следующую функциональность. Я что-то вбиваю в TextView, нажимаю кнопку и это что-то записывается в файл. Вбиваю, щёлкаю на кнопку. Кнопка у меня безликая получилась, надо было текст добавить. Давайте добавлю.

```
1 android:text="Write to file"
```

Так возвращаемся к activity. При чём тут разрешения? Если я записываю файл во внутреннее хранилище, то есть internal storage, то мне разрешение не нужно. Допустим я хочу записать файл во внешнее хранилище, то есть external storage. Для этого на Android 6.0 и выше – Marshmallow и выше – мне нужно, чтобы пользователь дал разрешение на это, то есть заходим в manifest и пишем

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

Отлично. Возвращаемся в наше активити. Для начала давайте определим переменные для EditText'a и для кнопки.

```
1 private EditText mInput;
2 private Button mWrite;
```

Так, mInput это

```
1 mInput = findViewById(R.id.et_input);
2 mWrite = findViewById(r.id.btn_write);
```

Добавляем функциональность:

```
1 mWrite.setOnClickListener (new View.OnClickListener(){
2     // содержание
3});
```

Мы берём текст из EditText'a и передаём его в метод, который впоследствии запишет этот текст в файл. Добавим ещё проверку на то, что текст непустой. Пишем название метода writeToFileIfNotEmpty() и передаём

```
1 String textToWrite = mInput.getText().toString();
```

и передаём textToWrite

```
1 writeToFileIfNotEmpty(textToWrite);
```

Alt+Enter по методу, создаём метод, создаём его в main activity. Мы проверяем, что текст непустой. Если у нас текст пустой, то мы покажем toast.

```
1 if(TextUtils.isEmpty(textToWrite)) {
2     Toast.makeText( context: this, text: "text is empty",
3         → Toast.LENGTH_SHORT).show();
```

```
3 } else {
4     writeToFileWithPermissionRequestIfNeeded(textToWrite);
5 }
```

Иначе пытаемся записать в файл. Если у нас 6.0 и выше, то нужно запросить разрешение. Создаём метод. Мы сейчас находимся в состоянии «проверка состояния разрешения».

```
1 private void writeToFileWithPermissionRequestIfNeeded(string textToWrite) {
2     if(isWritePermissionGranted()) {
3         writeToFile(textToWrite);
4     } else {
5         requestWritePermission();
6     }
7 }
```

Достаточно прямой подход. Если у нас есть разрешение, записываем. Если нет, то просим разрешения. Alt+Enter на WritePermissionGranted, Create Method. Что мы возвращаем?

```
1 return ContextCompat.checkSelfPermission( context: this,
    → Manifest.permission.WRITE_EXTERNAL_STORAGE ) == PackageManager.PERMISSION_GRANTED;
```

В качестве контекста мы передаём this, мы в активити находимся. В качестве строки на разрешение у нас Manifest.permission.WRITE_EXTERNAL_STORAGE. Такое разрешение нам дано, отлично. Создадим метод writeToFile() и в данном конкретном уроке это будет просто toast, а то урок затянемся.

```
1 Toast.makeText( context: this, text: textToWrite + " is written to file",
    → Toast.LENGTH_SHORT).show();
```

Alt+Enter по requestWritePermission. Проверку состояния разрешения мы уже сделали. Теперь запрос разрешения, если его нет. Как мы запрашиваем разрешение?

```
1 ActivityCompat.requestPermissions( activity: this, new
    → String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE}, requestCode:
    → WRITE_PERMISSION_RC);
```

Permission'ы можно передавать пачками, поэтому этот метод принимает на вход массив. Нам на этом уроке нужно только одно разрешение, но надо обернуть его в массив. Так как это request, то нам надо передать ещё и request code. Cmd+Alt+C, создаём константу и назовем ее WRITE_PERMISSION_RC.

Запрос разрешения, если его нет, мы сделали. Теперь показ объяснения, если оно нужно. Для того, чтобы проверить, нужно ли нам показывать объяснение, нам нужно вызвать метод

```
1 ActivityCompat.shouldShowRequestPermissionRationale( activity: this,  
→ Manifest.permission.WRITE_EXTERNAL_STORAGE)
```

Опять же, вместо activity this, вместо Permission'a копирую Manifest.permission.WRITE_EXTERNAL_STORAGE. Этот метод возвращает boolean, то есть мы оборачиваем его в if, и если нам нужно показать объяснение, то показываем его. Иначе просто запрашиваем разрешение. Вроде как всё прямолинейно.

Как мы можем показать объяснение? В принципе, мы можем сделать это в виде диалога. Почему бы и нет?

```
1 new AlertDialog.Builder( context: this)  
2 .setMessage("Без разрешения невозможно записать текст в файл")
```

И setPositiveButton, то есть человек соглашается. Пишем «понятно» и в качестве action'a передаем тот же запрос разрешения, то есть просто копируем

```
1 setPositiveButton( text: "Понятно", new DialogInterface.OnClickListener() {  
2     @Override  
3     public void onClick(DialogInterface dialog, int which) {  
4         ActivityCompat.requestPermissions( MainActivity.this, new  
→ String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},  
→ requestCode: WRITE_PERMISSION_RC);  
5     }  
6 })  
7 .show();
```

Показ объяснения, если оно нам нужно, реализовали. Теперь обрабатываем ответ на запрос разрешения. Обработка ответа происходит в методе onRequestPermissionsResult. На вход получаем requestCode, массив разрешений и массив ответов. Что мы делаем в первую очередь? В первую

очередь мы проверяем, что request code совпадает с нашим request code'ом. В данном случае я пишу, что если он не совпадает, то return, нам в методе больше делать нечего. Дальше мы проверяем, что в результатах у нас что-то есть. Так как у меня один запрос на разрешение, то длина массива grantResults должна быть 1.

```
1 if (grantResults.length !=1) return;
```

Если ничего из этого не произошло, то есть мы остались в методе, то проверяем

```
1 if (grantResult[0]==PackageManager.PERMISSION_GRANTED) {  
2  
3 }
```

По идеи в этот момент мы должны записать текст в файл, то есть весь этот процесс мы должны пройти заново, но потому как у нас теперь есть разрешение, то мы попадём в эту ветку, то есть я вызываю метод writeToFileWithPermissionRequestIfNeeded в случае, если пользователь дал разрешение на запись файла.

Если пользователь нам не дал разрешения, то нам нужно сказать ему, что он в любой момент может дать разрешение в настройках приложения, в настройках устройства. Снова создадим диалог. Ctrl+C. Ctrl+V. Меняем диалог. В качестве текста пишем «Вы можете дать разрешение в настройках устройства». И передаём null в качестве обработчика нажатия на кнопку, потому что сейчас тут нам нечего делать.

План выполнен, давайте запускать и проверять, что у нас получилось. Щелкаем OK.

Комментарии излишни. Собралось. Во-первых, пустой текст, text is empty. Пишем текст, write to file. Системный диалог, щёлкаю allow – и текст записан в файл.

Теперь давайте я перейду в настройки устройства и отклоню разрешение. Приложение у меня называется RequestPermissionsDemo, вот оно. Заходим в permissions и я отклоняю разрешение на storage. Возвращаемся к приложению, оно у меня перезапустилось, щелкаю write to file, и, так как я отозвал разрешение, теперь shouldShowRequestPermissionRationale возвращает true, а в первый раз оно возвращало false, поэтому я вижу объяснение, зачем мне нужно разрешение. Без разрешения невозможно записать текст в файл. OK, щелкает пользователь и видит разрешение.

Видим, так как я уже отклонял разрешение, точнее, в данном случае я убрал разрешение из настроек приложения, у меня появилась галочка «Don't ask again» и, если я выберу её, то я

уже не могу дать разрешение, у меня остаётся только deny. Щелкаю deny, вижу: «Вы можете дать разрешения в настройках устройства». Окей, понятно, щелкаю еще раз. «Без разрешения невозможно записать текст в файл». Понятно, don't ask again, deny. «Вы можете дать разрешение в настройках устройства». Мучительный круг.

Смотрите, я в последний раз щёлкнул на don't ask again. Щелкаю на write to file и я уже сразу попадаю в ветку без запроса permission, я сразу попадаю в ветку permission not granted, у меня нет разрешения. Я щёлкнул «не показывать снова» и, действительно, я больше этот диалог не вижу.

Кстати, в следующем занятии вы найдете приложение, которое пишет текст в файл, во внутреннее хранилище, во внешнее хранилище, удаляет файл, ну то есть работа с файлами. А этот урок мы заканчиваем, чтобы дольше его не затягивать.

1.2.5. Запись данных в файловую систему

В манифесте

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

На Android 6+ сделайте запрос Runtime Permission или дайте доступ в настройках устройства.

Код MainActivity.class

```
1 public class MainActivity extends AppCompatActivity {  
2  
3     public static final String FILENAME = "myfile";  
4     private EditText mInput;  
5     private TextView mFromInternal;  
6     private TextView mFromExternal;  
7     private Switch mIsExternalSwitch;  
8     private Button mDelete;  
9 }
```

```
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14
15         initUI();
16
17         mIsExternalSwitch.setEnabled(isExternalStorageAvailable());
18
19         mInput.setOnEditorActionListener(new TextView.OnEditorActionListener() {
20             @Override
21             public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
22                 if (actionId == EditorInfo.IME_ACTION_DONE) {
23                     String text = v.getText().toString();
24                     saveToFile(text, mIsExternalSwitch.isChecked());
25                     updateTextViews();
26                 }
27                 return false;
28             }
29         });
30
31         mDelete.setOnClickListener(new View.OnClickListener() {
32             @Override
33             public void onClick(View v) {
34                 deleteFile(mIsExternalSwitch.isChecked());
35                 updateTextViews();
36             }
37         });
38
39         updateTextViews();
40
41     }
42
43     private void deleteFile(boolean isExternal) {
44         if (isExternal) {
45             File file = new
46                 File(Environment.getExternalStorageDirectory().getAbsolutePath(),
47                 FILENAME);
48             if (file.delete()) {
49                 Toast.makeText(this, "deleted from external",
50
```

```
48             Toast.LENGTH_SHORT).show();
49     }
50
51 } else {
52     deleteFile(FILENAME);
53     Toast.makeText(this, "deleted from internal", Toast.LENGTH_SHORT).show();
54 }
55
56 }
57 }
58
59 private void updateTextViews() {
60     mFromInternal.setText(readFromInternalFileIfOption());
61     mFromExternal.setText(readFromExternalFileIfOption());
62 }
63
64 private String readFromExternalFileIfOption() {
65     File file = new
66         File(Environment.getExternalStorageDirectory().getAbsolutePath(),
67         FILENAME);
68     try (BufferedReader reader = new BufferedReader(new InputStreamReader(new
69         FileInputStream(file)))) {
70         StringBuilder stringBuilder = new StringBuilder();
71         String string;
72         while ((string = reader.readLine()) != null) {
73             stringBuilder.append(string).append("\n");
74         }
75
76         return stringBuilder.toString();
77     } catch (FileNotFoundException e) {
78         e.printStackTrace();
79         Toast.makeText(this, "File not found", Toast.LENGTH_SHORT).show();
80     } catch (IOException e) {
81         e.printStackTrace();
82         Toast.makeText(this, "Can't read from file", Toast.LENGTH_SHORT).show();
83     }
84     return "";
85 }
```

```
85     private void initUI() {
86         mFromInternal = findViewById(R.id.tv_from_internal_file);
87         mFromExternal = findViewById(R.id.tv_from_external_file);
88         mIsExternalSwitch = findViewById(R.id.switch_is_external);
89         mInput = findViewById(R.id.et_some_text);
90         mDelete = findViewById(R.id.btn_delete_file);
91     }
92
93     private void saveToFile(String text, boolean isInExternal) {
94         if (isInExternal) {
95             saveToExternalFile(text);
96         } else {
97             saveToInternalFile(text);
98         }
99     }
100
101    private boolean isExternalStorageAvailable() {
102        String state = Environment.getExternalStorageState();
103        return state.equals(Environment.MEDIA_MOUNTED);
104    }
105
106    private String readFromInternalFileIfOption() {
107        try (BufferedReader reader = new BufferedReader(new
108            InputStreamReader(openFileInput(FILENAME)))) {
109            StringBuilder stringBuilder = new StringBuilder();
110            String string;
111            while ((string = reader.readLine()) != null) {
112                stringBuilder.append(string).append("\n");
113            }
114            return stringBuilder.toString();
115        } catch (FileNotFoundException e) {
116            e.printStackTrace();
117            Toast.makeText(this, "File not found", Toast.LENGTH_SHORT).show();
118        } catch (IOException e) {
119            e.printStackTrace();
120            Toast.makeText(this, "Can't read from file", Toast.LENGTH_SHORT).show();
121        }
122        return "";
123    }
```

```
124
125     private void saveToInternalFile(String text) {
126         try {
127             String textToWrite = text + "\n";
128             FileOutputStream outputStream = openFileOutput(FILENAME,
129                 Context.MODE_APPEND);
130             outputStream.write(textToWrite.getBytes());
131             outputStream.close();
132             Toast.makeText(this, "written to internal", Toast.LENGTH_SHORT).show();
133         } catch (Exception e) {
134             e.printStackTrace();
135         }
136     }
137
138     private void saveToExternalFile(String text) {
139         try {
140             String textToWrite = text + "\n";
141             File file = new File(Environment.getExternalStorageDirectory(), FILENAME);
142             FileOutputStream outputStream = new FileOutputStream(file, true);
143             outputStream.write(textToWrite.getBytes());
144             outputStream.close();
145             Toast.makeText(this, "written to external", Toast.LENGTH_SHORT).show();
146         } catch (FileNotFoundException e) {
147             e.printStackTrace();
148         } catch (IOException e) {
149             e.printStackTrace();
150         }
151     }
```

Код activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
```

```
6         android:orientation="vertical"
7         android:padding="16dp">
8
9     <EditText
10        android:id="@+id/et_some_text"
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content"
13        android:imeOptions="actionDone"
14        android:inputType="text" />
15
16     <Switch
17        android:id="@+id/switch_is_external"
18        android:layout_width="wrap_content"
19        android:layout_height="wrap_content"
20        android:text="external storage?" />
21
22     <Button
23        android:text="delete"
24        android:id="@+id/btn_delete_file"
25        android:layout_width="wrap_content"
26        android:layout_height="wrap_content" />
27
28     <TextView
29        android:id="@+id/tv_from_internal_file"
30        android:layout_width="match_parent"
31        android:layout_height="wrap_content"
32        android:textColor="#FF0000" />
33
34     <TextView
35        android:id="@+id/tv_from_external_file"
36        android:layout_width="match_parent"
37        android:layout_height="wrap_content"
38        android:textColor="#00FF00" />
39
40 </LinearLayout>
```

1.2.6. Материалы для самостоятельного изучения

Работа с файлами в java

1. <https://docs.oracle.com/javase/tutorial/essential/io/file.html>
2. <https://metanit.com/java/tutorial/6.11.php>

Работа с файлами в Android

1. <https://developer.android.com/training/data-storage/files.html>

1.3. Работа с БД

1.3.1. Проектирование БД на бумаге

Давайте смиримся с той мыслью, что мы добавляем в приложение базу данных. Предметная область – музыка (спойлер: эта БД пригодится вам в работе с курсовым проектом).

Определимся со структурой, таблицами и связями.

Вырисовывается первая таблица – **Songs**, что значит «песни».

№	Название	Тип переменной
1	id	численный
2	name	строковый
3	duration	строковый

С id и name все понятно, надеюсь. duration – строковый, для упрощения картины, нам сейчас не нужно возиться с кастом Date в String.

Неплохо. Песни должны храниться в альбомах. Создадим таблицу **Albums**.

№	Название	Тип переменной
1	id	численный
2	name	строковый
3	release_date	строковый

release_date – дата выпуска альбома, тоже строка, чтобы не возиться с кастом дат.

В альбоме несколько песен, логично. А одна песня может позднее быть перепета, соответственно, у песни может быть несколько альбомов. Напрямую создать связь многие ко многим мы не можем, но зато можем создать таблицу, которая будет хранить связи – **AlbumsSongs**.

№	Название	Тип переменной
1	id	численный
2	album_id	численный
3	song_id	численный

Тут все просто – каждая запись показывает, какому альбому принадлежит песня.

Возможно, я хочу выразить свое восхищение тем или иным альбомом. Возможно, я такой не один. Нам нужна таблица, которая будет хранить комментарии к альбомам – **Comments**.

№	Название	Тип переменной
1	id	численный
2	album_id	численный
3	text	строковый
4	author	строковый
5	timestamp	дата

Получаем, опять же, id комментария, id альбома, к которому относится комментарий, текст и автора, и таймштамп в виде даты. В этот раз не будем халтурить.

Теперь у нас есть тематика, таблицы и связи.

1.3.2. Room. Знакомство

Про Room есть замечательный цикл статей в официальной документации. Здесь же будет краткий обзор сущностей и аннотаций, которые нам нужны.

Итак, с чего начать? С зависимости в gradle.

Скорее всего у вас уже будет указанный гугловский репозиторий, но лучше все же проверить.

В build.gradle уровня project в блоке buildscript.repositories должен быть указан google(). Если его там нет, то добавьте в новой строке.

В build.gradle уровня app добавляем следующие строки в блок dependencies (версию ставьте актуальную):

```
1 implementation "android.arch.persistence.room:runtime:1.0.0"
2 annotationProcessor "android.arch.persistence.room:compiler:1.0.0"
```

Добавление БД

БД – это прежде всего таблицы. Используя Room, мы можем просто создать POJO-класс и снабдить его аннотациями.

```
1 @Entity // обозначаем таблицу
2 public class Song {
3
4     @PrimaryKey // первичный ключ, идентификатор строки
5     @ColumnInfo(name = "id") // обозначаем столбец, задаем имя
6     private int mId;
7
8     @ColumnInfo(name = "name")
9     private String mName;
10
11    @ColumnInfo(name = "duration")
12    private String mDuration;
13
14    //тут обычные геттеры и сеттеры
15 }
```

После создания одной или нескольких таблиц, нам нужно создать DAO – data access object – то есть сущность, с помощью которой и будет производиться обращение к таблицам для чтения и записи. Он может выглядеть, например, так:

```
1  @Dao //обозначаем класс для работы с таблицами
2  public interface MusicDao {
3
4      //добавить песни в таблицу
5      @Insert
6      public void insertSongs(List<Song> songs);
7
8      //получить список всех песен из таблицы
9      @Query("SELECT * from song")
10     public List<Song> getSongs();
11
12     //удалить песню по id
13     @Query("DELETE FROM song where id = :songId")
14     void deleteSongById(int songId);
15 }
```

Теперь нам нужно создать класс самой базы данных. Точно так же, через аннотации. Из важного тут – класс должен быть абстрактным, наследоваться от RoomDatabase, содержать абстрактный метод, который возвращает DAO. В аннотации указываем таблицы и версию БД.

```
1  @Database(entities = {Song.class}, version = 1)
2  public abstract class DataBase extends RoomDatabase {
3      public abstract MusicDao getMusicDao();
4  }
```

Подготовка завершена.

Доступ к БД

Для получения экземпляра базы данных воспользуемся фабричным методом. Если базы на этот момент не существует, то она создастся с заданным именем.

```
1 DataBase mDatabase = Room.databaseBuilder(getApplicationContext(), DataBase.class,
2     "music_database") // название файла
3         .fallbackToDestructiveMigration() // дешевый способ миграции на новую версию
4         .build();
```

Также можно создать inMemoryDatabase – базу, которая существует до тех пор, пока процесс не будет уничтожен. Работает как кеш.

Ссылки на документацию

1. <https://developer.android.com/training/data-storage/room/index.html>

А теперь давайте перейдем к практике.

1.3.3. Создание Room базы

В этом и в последующих двух видео мы разберемся, как работать с базами данных и ContentProvider'ом. Давайте я, как я люблю обычно делать, намечу план. Первый пункт нашего плана. «Добавить базу данных Room». Второй пункт. «Вставить данные/извлечь данные», то есть чуть поработать с этой базой. Третий пункт. «Добавить ContentProvider над Room».

Первым делом нам нужно добавить зависимости в gradle. И будет неправдой, если я скажу, что знаю эти зависимости наизусть. Вот в гугле вбили «room database dependency», нашли страничку и копируем. Ctrl+C. Возвращаемся в наш проект, вставляем. Щелкаем Sync Now. Теперь в нашем проекте есть Room, магия Gradle'a.

Давайте проверим это. Room, есть такое. Отлично. Теперь давайте создадим классы и сущности, из которых будет состоять наша база данных. Не хочу, чтобы они лежали кучей, я создам новый пакет, назову его database, щелкну OK. New → Java Class. Album. Альбом. Щелкаю OK. Преимущество Room в том, что мы создаем классы именно как java-объекты вместо того, чтобы создавать их вручную, как раньше. Аннотация Entity. Альбом у нас состоит из трех сущностей. Это id:

```
1 private int mId;
```

название альбома:

```
1 private String mName;
```

дата релиза, она тоже будет храниться в строке:

```
1 private String mReleaseDate;
```

Сгенерим конструктор без параметров select none и Getter'ы, Setter'ы. То есть создаем обычный POJO-класс. Теперь id у нас будет primary (@PrimaryKey). Давайте сделаем еще аннотацию @ColumnInfo(name = "id").

Есть два подхода. Либо указывать id вместо mId, либо явно указыватьColumnInfo, name. Я хочу, чтобы это поле хранилось вот в таком столбце. Так:

```
1 @ColumnInfo(name = "release")
```

Хорошо, создали сущность-альбом, теперь создадим сущность-песню. New -> Java class -> Song -> OK.

```
1 private int mId;
2 private String mName;
3 private String mDuration;
```

У нас очень хороший сервер, он возвращает уже отформатированные данные, во всяком случае мне так сказали. Так, тут у нас @Entity, здесь у нас @ColumnInfo(name = "name") и @ColumnInfo(name = "duration"). Ctrl+N, конструктор без параметров, Ctrl+N, конструктор со всеми параметрами, Ctrl+N, Getter'ы и Setter'ы для всех трех полей. Хорошо, у нас есть альбомы, в которых есть песни, но я не могу в базу данных загнать список песен в одну запись альбома, потому что это база данных, так нельзя. Чтобы это обойти, мне нужно сделать еще одну сущность, которая будет связующим звеном, еще одну таблицу, которая будет связывать песни и альбомы. Назовем ее AlbumSong. Щелкаем OK. Что тут будет храниться?

```
1  private int mId;
2  private int mAlbumId;
3  private int mSongId;
```

То есть связь между альбомом и песней, какая песня относится к какому альбому. По-моему, это очень прямолинейно. Это у нас PrimaryKey и я не могу заполнять эту таблицу сам, она будет заполняться автоматически.

```
1  @PrimaryKey (autoGenerate = true)
```

Ну и стандартно:

```
1  @ColumnInfo (name = "id")
2  @ColumnInfo (name = "album_id")
```

И здесь:

```
1  @ColumnInfo (name = "song_id")
```

Хорошо. Ctrl+N, конструктор с параметрами, Ctrl+N, конструктор без параметров, Ctrl+N, Getter'ы и Setter'ы. И здесь нам нужно сделать Entity, мне нужно чуть-чуть повозиться, объявить, что это не просто таблица, а именно связующее звено. Делается это с помощью директивы foreignKeys, которая принимает на вход массив, который мы передаем соответственно в ForeignKey.

```
1  @ForeignKey (entity = Album.class, parentColumns = "id", childColumns = "album_id")
```

И еще один ForeignKey:

```
1  @ForeignKey (entity = Song.class, parentColumns = "id", childColumns = "song_id")
```

Все хорошо. Сущность создали. Теперь нам нужно создать объект DAO. Объект с помощью которого мы будем обращаться к этим сущностям и манипулировать ими.

New -> Java Class. MusicDao пусть называется. Щелкаем OK. И в этом DAO – data access object, если что – будут храниться наши методы, с помощью которых мы будем обращаться к нашим

данным. Во-первых, это не класс, это интерфейс. Щелкну Sync Now. Какие методы нам нужно добавить? Во-первых, добавим методы, которые заполняют нашу базу, соответственно это:

```
1 @Insert(onConflict = onConflictStrategy.REPLACE)
2 void invertAlbums(List<Album> albums);
```

Если в базе уже есть такой альбом, то новый его заменит. Сравнение идет по PrimaryKey, по ключу. Дальше:

```
1 @Insert(onConflict = onConflictStrategy.REPLACE)
2 void insertSongs(List<Song> songs);
```

И нам нужно заполнить еще и связку:

```
1 @Insert(onConflict = onConflictStrategy.REPLACE)
2 void setLinksAlbumSongs(List<AlbumSong> linkAlbumSongs);
```

Следующее, что мы можем сюда дописать. Мы данные вставили, мы можем их еще потребовать обратно:

```
1 @Query("select * from album")
2 List<Album> getAlbums();
```

И то же самое для списка песен:

```
1 @Query("select * from song")
2 List<Song> getSong();
```

Что нам еще можно сделать? Можно попытаться удалить альбом.

```
1 @Delete
2 void deleteAlbum(Album album);
```

И так далее. Ничего сложного в принципе в составлении этих запросов нет. Разве что можно создать еще один запрос, комплексный, который связывает альбомы и песни через album song. Пускай будет:

```
1 List<Song> getSongsFromAlbum(int albumId);
```

Пишем query-запрос:

```
1 @Query("select * from song inner join albumsong on song.id = albumsong.song_id where  
        ↴ album_id = :albumId")
```

Написали мы query-запрос. Получить все из таблицы песен, связанной с таблицей AlbumSong по id песен и в этой новой таблице, где album_id равен albumId, который мы ему передали. Вроде как все просто. Мы, кстати, можем щелкнуть Run, чтобы посмотреть, правильно ли мы все это заполнили, потому что Room проверяет query-запросы в момент компиляции, а не в момент самой работы. Судя по всему, все хорошо.

И последнее, что нам осталось сделать – это создать объект базы данных. MusicDatabase, суперкласс: roomdatabase. И он должен быть абстрактным. Соответственно наша база данных должна возвращать наш DAO:

```
1 abstract MusicDao getMusicDao();
```

И мы должны указать в Database все entity, все сущности, которые находятся в этой базе:

```
1 @Database(entities = {Album.class, Song.class, AlbumSong.class}, version = 1)
```

Итак, мы написали три таблицы в нашей базе данных, один DAO-объект для работы с этими таблицами и один класс musicDatabase, чтобы получить доступ к нашей базе. В следующем видео мы будем добавлять, извлекать данные из таблицы.

1.3.4. Сохранение и извлечение данных с Room

Продолжаем работать с нашей базой данных. Пункт «добавить базу данных Room» я считаю выполненным. Теперь мы начинаем изучать, как вставлять данные и как извлекать данные из базы данных.

Итак, я нахожусь в activity, и как мне получить доступ к моей базе данных? Google рекомендует,

а мы следуем – создавать базы данных в классе application. Так что давайте мы создадим такой класс. New → Java Class. Назовем его AppDelegate. Суперкласс: Application. OK. onCreate.

Application нужно добавить в Manifest:

```
1 android:name=".AppDelegate"
```

Теперь наше приложение использует наш AppDelegate. Здесь что мы делаем? Здесь мы создаем поле database:

```
1 private MusicDatabase mMusiceDatabase;
```

и дальше в onCreate'е инициализируем его:

```
1 mMusiceDatabase = Room.DatabaseBuilder(getApplicationContext(), MusicDatabase.class,  
    ↴ name: "music_database").build()
```

Создали базу данных. Теперь давайте создадим Getter для этой базы. Переключаемся в наше activity и пишем:

```
1 ((AppDelegate) getApplicationContext()).getMusicDatabase()
```

Теперь у нас есть экземпляр MusicDatabase в нашем activity. На самом деле мне не нужна база данных, мне нужен объект DAO. getMusicDao. «getMusicDao is not public». Раз он не public, я его сделаю public. Мы получили доступ к DAO и теперь мы можем делать с ним все, что захотим.

Давайте сначала добавим в нашу таблицу данные. Для этого я создам какую-нибудь кнопку. Constraint Layout. LinearLayout.

```
1 <Button  
2     android:text="Add"  
3     android:id="@+id/add"  
4     android:layout_width="match_parent"  
5     android:layout_height="wrap_content" />
```

И еще одну кнопку:

```
1 <Button  
2     android:text="Add"  
3     android:id="@+id/get"  
4     android:layout_width="match_parent"  
5     android:layout_height="wrap_content" />
```

В этот раз не add, а get. Можно было бы, конечно, написать insert query, но не буду. Так, main activity:

```
1 mAddBtn = (findViewById(R.id.add).add);  
2 mAddBtn.setOnClickListener(new View.OnClickListener() {  
3     @Override  
4     public void onClick(View v) {  
5         ...  
6     }  
7 });
```

Вообще, Room по умолчанию работает в фоновом потоке, то есть извлечение и добавление должно происходить в фоновом потоке. Я в ознакомительных целях исключительно поменяю это поведение.

```
1 .allowMainThreadQueries()
```

С помощью этого метода. И теперь я смогу добавлять данные в базу данных в main thread'e, но, как вы уже знаете, это плохая идея, не самая лучшая.

```
1 public void onClick(View v) {  
2     musicDao.insertAlbums(createAlbums());  
3 }
```

Alt+Enter → Create method → main activity.

```
1 private List<Album> createAlbums() {  
2     List<Album> albums = new ArrayList<>(InitialCapacity: 10);  
3     for (int i = 0; i < 10; i++) {  
4         albums.add( new ALbum (i, name: "album "+ i, releaseDate "release" +  
5             System.currentTimeMillis()));
```

```
5         }
6     return albums;
7 }
```

Такой несчастный, прямой, простой, как палка, способ создать несколько альбомов. При нажатии на кнопку я добавлю список альбомов в базу данных. Так, вторая кнопка:

```
1 mGetBtn = FindViewById(R.id.get);
2 mGetBtn.setOnClickListener(new View.OnClickListener() {
3     @Override
4     public void onClick(View v) {
5         showToast(musicDao.getAlbums());
6     }
7});
```

Получим список альбомов. И что мне с этими альбомами делать? Я их выведу в toast – самый простой способ показать, что они там есть. Alt+Enter, Create method. Ctrl+X, Ctrl+V.

```
1 private void showToast(List<Album> albums) {
2     StringBuilder builder = new StringBuilder();
3     for(int i = 0, size = albums.size(); i < size; i++){
4         builder.append(albums.get(i).ToString()).append("\n");
5     }
6 }
```

Давайте переключимся в альбомы и здесь переопределим метод `toString`. Нам нужно указать еще `toast`.

```
1 Toast.makeText ( context: this, builder.toString(), Toast.LENGTH_SHORT).show();
```

В принципе все должно быть нормально, щелкаем на Run. Добавили список альбомов и проверить это можно через метод. Так, OK, видим, что в подавляющем большинстве это 52, 52, 52, 52 и в конце 53. Щелкнем еще раз ADD, если вы помните, там была стратегия – при конфликтах перезаписывать.

Теперь щелкаем на GET, видим совсем другие миллисекунды, то есть старые данные были заменены на новые. Вот таким простым, нехитрым образом мы получили базу данных, добавили

данных – я показывал один метод – и получили данные, я показал другой метод. Это было сделано в main thread'е только в ознакомительных целях. В нормальных базах, нормальных приложениях, всегда работайте с базами данных в фоновом потоке, чтобы не аффектить UI. Хорошо. Видео на этом закончено, а в следующем мы добавим ContentProvider над нашим Room.

1.3.5. Добавление контент провайдера над Room

Последнее видео, посвященное базам данных – добавление ContentProvider'a. В уроке со списками я много рассказывал про ContentProvider, про то, как с ним работать, но мы упустили одну маленькую деталь. Мы не затронули создание самого ContentProvider'a, то есть мы научились работать с системным ContentProvider'ом, но не научились создавать свои. Давайте этот пробел утрясем.

New → Java Class → Other → Content Provider. ClassName: MusicProvider. URI Authorities: com.elegion.roomdatabase.musicprovider. Ну, в принципе, окей. URI – это уникальное название вашего провайдера, по которому к нему будет обращаться. Соответственно, когда я обращался к системным ContentProvider'ам, я вбивал константу, но по факту за этой константой скрывается вот такой адрес. Не точно такой же, это мой адрес, но системный какой-нибудь. Щелкаем finish. Создался класс ContentProvider'a с кучей методов, которые нам нужно переопределить.

Хорошо, давайте начнем переопределять, и начнем мы чуть-чуть издалека, давайте для начала создадим TAG:

```
1 private static final String TAG = MusicProvider.class.getSimpleName();
```

Дальше нам нужно создать строковую константу для AUTHORITY, то, которое мы скормили при создании wizard'у классу:

```
1 public static final String AUTHORITY = "com.elegion.roomdatabase.musicprovider"
```

Будем точно так же создавать таблицу альбомов в текущем уроке ознакомительно, так что загоним название таблицы в строковое поле:

```
1 private static final String TABLE_ALBUM = "album";
```

Заметьте, что таблица и здесь и тут называется одинаково. Это одна таблица, одно название. Что мы делаем дальше? Дальше нам нужно различить два состояния: когда мы обращаемся к таблице и когда мы обращаемся к элементу в этой таблице. И это можно сделать с помощью специального класса, который называется UriMatcher. Давайте его создадим:

```
1 private static final UriMatcher URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
```

Дальше добавляем два разных события: обращение к таблице и обращение к альбому соответственно:

```
1 static {
2     URI_MATCHER.addURI(AUTHORITY, TABLE_ALBUM, ALBUM_TABLE_CODE);
3 }
```

Если путь заканчивается просто на название таблицы, мы возвращаем код, который говорит о том, что это таблица. Пускай это будет 100. Другой случай, если путь выглядит следующим образом:

```
1 URI_MATCHER.addURI(AUTHORITY, path: TABLE_ALBUM+"/*", ALBUM_ROW_CODE);
```

То есть это может быть AUTHORITY, альбом первый, то есть это какой-то определенный элемент в этой таблице. * – это любое числовое значение. Соответственно в нашем случае мы возвращаем ALBUM_ROW_CODE. Alt+Enter → Create constant.

```
1 private static final int ALBUM_ROW_CODE = 101;
```

Что мы получили? Так как у нас только одна таблица, два состояния: либо сама таблица, либо какое-то значение в этой таблице, соответственно, чем больше таблиц, тем больше состояний.

Дальше давайте отсортируем по – я не знаю... по уму. onCreate, getType, дальше query, delete пускай будет в конце, insert и update, пусть будет так. В onCreate мы должны получить доступ к нашей базе данных. Давайте это сделаем:

```
1 if(getContext()!=null){
2     Room.databaseBuilder(getContext().getApplicationContext(), MusicDatabase.class,
3             "music_database").build().getMusicDao();
4     return true;
5 }
```

```
4 }
```

И это все загоним в переменную, точнее, в поле. Ctrl+Alt+F → mMUSICDAO. Если у нас все хорошо прошло, то возвращаем true. Базу данных создали.

Дальше implementим метод getType. Читаем: «Implement this to handle requests for the MIME type of the data». OK, допустим. Зажимаем Cmd, щелкаем на getType, щелкаем на Yes → Content Provider → getType. Да, и видим, что getType должен возвращать либо cursor.item в начале для одной записи таблицы, либо directory для нескольких записей. Немного расплывчатое объявление, но ничего страшного. В этом методе мы пользуемся switcher'ом для URI_MATCHER'a. На вход у нас будут коды, которые возвращал matcher.

```
1 switch (URI_MATCHER.match(uri)){
2     case ALBUM_TABLE_CODE:
3         return "vnd.android.cursor.dir/" + AUTHORITY + ". " + TABLE_ALBUM;
4     case ALBUM_ROW_CODE:
5         return "vnd.android.cursor.item/" + AUTHORITY + ". " + TABLE_ALBUM;
6     default:
7         throw new UnsupportedOperationException("not yet implemented");
8 }
```

Все, наверное, да? Идем дальше. На вход метода получает URI, по которому мы сможем отличить, нам нужна таблица либо запись в этой таблице, и projection, selection, selectionArgs и sort order. Как вы помните из урока по Loader'ам в списке и ContentProvider'у в списке, эти аргументы должны вам напоминать те аргументы, которые используются при создании cursor'a и cursorloader'a. Этот конкретный метод ожидает на выходе cursor, что означает, что мы должны в нашем musicDAO добавить методы, которые возвращают курсор. Room это умеет из коробки, я просто скопирую:

```
1 @Query ("select * from album")
2 Cursor getAlbumsCursor();
```

Это в случае таблицы. В случае одного элемента:

```
1 @Query ("select * from album where id = :albumId")
2 Cursor getAlbumsWithIdCursor(int album id);
```

Возвращаемся к нашему Provider'у. Что делаем в query? Во-первых, давайте достанем код:

```
1 int code = URI_MATCHER.match(uri);
```

Дальше защитное программирование:

```
1 if(code !=ALBUM_ROW_CODE && code !=ALBUM_TABLE_CODE) return null;
```

В другом случае создаем курсор:

```
1 Cursor cursor;
2 if(code ==ALBUM_TABLE_CODE){
3     cursor = mMusicDao.getAlbumsCursor();
4 } else {
5     cursor = mMusicDao.getAlbumWithIdCursor((int)ContentUris.parseId(uri));
6 }
7 return cursor;
```

Нам нужно выдернуть id, id мы можем выдернуть из URI с помощью класса.

И, в принципе, по такому же принципу имплементятся все остальные методы. Мы проверяем код в URI_MATCHER и, в соответствии с этим, вызываем нужные нам методы Dao.

Как показать, что это все работает правильно? Для этого нам нужно создать другое приложение. А, вообще-то сначала нужно все запустить. Так, запустили, add, get. Все так же работает, так же, как и было. Что мы делаем дальше? Создаем новое приложение. File -> New -> New Project -> SecondAppForContentProviderTest, щелкаем Next. Next -> finish.

Во втором приложении мы постараемся достучаться до ContentProvider'a и делается это легко, с помощью Loader'ов. Пишем:

```
1 implements LoaderManager.LoaderCallbacks
```

Alt+Enter -> Implement method -> onCreate.

```
1 return new CursorLoader(context: this,
2     Uri.parse("content://com.elegion.roomdatabase.musicprovider/album") ,
3     projection: null,
```

```
3     selection: null,  
4     selectionArgs: null,  
5     sortOrder: null);
```

Видите, он принимает на вход URI, мы передаем ему тот URI, который создали. Selection, projection – передаем туда нули, так как у меня нет задачи делать что-то крутое, моя задача показать, что это работает. Так, и на onLoadFinished:

```
1  public void onLoadFinished(Loader<Cursor> loader, Cursor data) {  
2      if (data != null && data.moveToFirst()) {  
3          StringBuilder builder = new StringBuilder();  
4          do  
5          {  
6              builder.append(data.getString(data.getColumnIndex( columnName  
7                  ← = "name"))).append("\n");  
8          } while (data.moveToNext());  
9          Toast.makeText( context: this, builder.toString(),  
10             ← Toast.LENGTH_LONG).show()  
11      }  
12  }
```

Вроде все хорошо. И покажем мой любимый toast, который я использую по случаю и без.

```
1  getSupportLoaderManager().initLoader( id: 123, args: null, callback: this);
```

Запускаем и теперь при запуске этого приложения мы должны увидеть toast с таблицей, которая была в нашем другом приложении.

Когда это может пригодиться? Допустим, у вас есть несколько приложений, у которых большая аудитория, и скорее больше одного приложения от этой компании, соответственно, эти приложения могут делиться данными друг с другом, что очень удобно.

На этом у меня все, мы закончили работать с базами данных, закончили работать с ContentProvider'ом. Я думаю, что данная информационная база для начала сойдет, а вам придется учиться и развиваться дальше самим. Поэкспериментируйте с content provider'ом, заимплементьте другие методы, соответственно другие методы в Dao, посмотрите на результаты.

О проекте

Академия e-Legion – это образовательная платформа для повышения квалификации в мобильной разработке. Слушайте лекции топовых разработчиков, выполняйте практические задания и прокачивайте свои скиллы. Получите высокооплачиваемую профессию – разработчик мобильных приложений.

Программа “Многопоточность и сетевое взаимодействие”

Блок 1. Обзор средств для обеспечения многопоточности

- Знакомство с курсом
- Многопоточность и параллельное программирование
- Обзор инструментов для обеспечения многопоточности в Java (Thread, Runnable, Callable, Future, Executors)
- Обзор инструментов для обеспечения многопоточности в Android (IntentService + BroadcastReceiver, HaMeR, AsyncTask, Loaders)
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.

Блок 2. Service + BroadcastReceiver

- Знакомство с Service, IntentService
- Создание Service
- Бродкастресивер, знакомство
- Создание BroadcastReceiver
- Связка Activity-Service-BroadcastReceiver-Activity
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.

Блок 3. Многопоточность в Android

- AsyncTask, знакомство
- AsyncTask, работа
- HaMeR
- Пример работы HaMeR
- Loader, знакомство

- ContentProvider, знакомство
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.