Курс по Android разработке

Яборов Андрей Владимирович

avyaborov@gravity-group.ru

Марквирер Владлена Дмитриевна

vdmarkvirer@hse.ru

НИУ ВШЭ 2022

GRAVITY group

Разработка под Android

Урок 5

- Работа с сетью. Основные библиотеки для работы с сетью
- Списки
- Base проект
- Задания

GRAVITY group

Работа с сетью

Телефоны все дальше и дальше отдаляются от своей основной задачи - просто позвонить. Теперь устройство используется как платформа для сетевых игр, посещения сайтов, общения по Skype, ICQ и т.д. И поэтому телефону требуется обеспечить сетевую поддержку.

Мобильные устройства подключаются к Интернету при помощи 3G, Wi-Fi (в редких случаях и по кабелю), используя стандартные протоколы HTTP, HTTPS, TCP/IP, сокеты.

Основные моменты, на которые должен обратить внимание разработчик при реализации сетевой поддержки - обеспечение конфиденциальности, учет потребления заряда батареи, проверка доступности сети.

Android предоставляет разработчику все необходимые библиотеки для работы с сетью. Android использует для работы с сетью стандартные библиотеки Java , а также ряд своих библиотек.

В таблице приведены некоторые пакеты, относящиеся к сетевым возможностям, которые присутствуют в SDK Android:

Работа с сетью

java.net	Содержит классы, связанные с сетевыми функциями, в том числе сокеты потоков и датаграмм, протокол IP, а также общие средства для работы с HTTP. Это многоцелевой ресурс для работы с сетями.
java.io	Пакет не относится непосредственно к сетям. Его классы используются сокетами и соединениями, содержащимися в других пакетах Java. Они используются также для обмена с локальными файлами (что часто происходит при взаимодействии с сетью).
java.nio	Содержит классы, которые служат буфером для определенных типов данных. Удобен для организации сетевой связи между двумя конечными точками средствами Java.
org.apache.*	Набор пакетов, которые обеспечивают точный контроль и функции для HTTP-коммуникаций на основе Apache - популярного веб-сервера с открытым исходным кодом.
android.net	Содержит дополнительные сокеты доступа к сети в дополнение к основным классам java.net.*. Этот пакет включает в себя класс URI, который часто используется в разработке приложений Android, не связанных с сетью.
android.net.http	Содержит классы для работы с сертификатами SSL.
android.net.wifi	Содержит классы для реализации всех аспектов WiFi (802.11 Wireless Ethernet) на платформе Android.

Важно, чтобы ваше приложение:

- использовало сетевые сервисы только при необходимости (используйте локальное кэширование данных при первой возможности)
- предупреждало пользователя при использовании личной информации
- имело гибкие настройки включения и отключения различных функциональных возможностей без ущерба использования программы. Например, если уровни подгружаются из интернета, то первые несколько уровней должны быть зашиты в память программы
- проверяло наличие сети и обрабатывало эту ситуацию

Работа с сетью

HTTP - широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста»
Этот протокол описывает взаимодействие между двумя компьютерами (клиентом и сервером). Это взаимодействие строится на двух типах сообщений:

Запрос к серверу (**Request**) и Ответ от сервера (**Response**)

Каждое сообщение состоит из трех частей:

1. Стартовая строка запроса. Из всех параметров обязательной является только она. Выглядит она так: *METHOD URI HTTP/VERSION*

- METHOD - это метод HTTP-запроса
- URI - идентификатор ресурса
- VERSION - версия протокола

2. Заголовки (Headers) - это набор пар имя-значение, разделенных двоеточием. В заголовках передается различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент (Referrer) и так далее.
3. Тело сообщения (Body) - это передаваемые данные.

Работа с сетью

Основные Методы HTTP-запроса:

- GET - получить
- POST - создать
- PUT - обновить (не переданные поля считаются null)
- DELETE - удалить
- PATCH - обновить запись (только переданные поля)

Помимо них есть и другие CONNECT, HEAD, OPTIONS, TRACE.

Например:

- GET-запрос /rest/users - получение информации о всех пользователях
- GET-запрос /rest/users/125 - получение информации о пользователе с id=125
- POST-запрос /rest/users - добавление нового пользователя
- PUT-запрос /rest/users/125 - изменение информации о пользователе с id=125
- DELETE-запрос /rest/users/125 - удаление пользователя с id=125

Коды ответа:

Method	Method
200 OK	201 Created
202 Accepted	203 Not authorized
204 No content	205 Reset content
206 Partial content	
300 Multiple choice	301 Moved permanently
302 Found	303 See other
304 Not modified	306 (unused)
307 Temporary redirect	
400 Bad request	401 Unauthorized
402 Payment required	403 Forbidden
404 Not found	405 Method not allowed
406 Not acceptable	407 Proxy auth required
408 Timeout	409 Conflict
410 Gone	411 Length required
412 Preconditions failed	413 Request entity too large
414 Requested URI too long	415 Unsupported media
416 Bad request range	417 Expectation failed
500 Server error	501 Not implemented
502 Bad gateway	503 Service unavailable
504 Gateway timeout	505 Bad HTTP version

GRAVITY group

Работа с сетью

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

Общение с сервером может быть представлено абсолютно различными способами, REST же описывает то, каким конкретно (в рамках http) оно будет. На самом деле помимо REST также существуют RPC, SOAP и другие, но REST является наиболее распространенным.

Особенность REST в том, что сервер не запоминает состояние пользователя между запросами - в каждом запросе передается информация, идентифицирующая пользователя (например, token, полученный через OAuth-авторизацию) и все параметры, необходимые для выполнения операции.

https://habrahabr.ru/post/50147/ https://habrahabr.ru/post/38730/ https://ru.wikipedia.org/wiki/REST

https://habrahabr.ru/post/158605/ https://habrahabr.ru/post/265845/

Работа с сетью

В самом начале в Android имелись только 2 HTTP клиента: **HttpURLConnection** и **Apache HTTP Client**.

HttpURLConnection

Класс java.net.HttpURLConnection является подклассом java.net.URLConnection и позволяет реализовать работу по отправке и получении данных из сети по протоколу HTTP. Данные могут быть любого типа и длины. Данный класс следует использовать для отправки и получения потоковых данных, размеры которых нельзя заранее определить. Используя данный класс, вам не нужно думать о сокетах и реализовывать собственные способы общения между клиентом и сервером.

Алгоритм использования следующий:

- Получить объект **HttpURLConnection** через вызов **URL.openConnection()** и привести результат к **HttpURLConnection**
- Подготовить необходимый запрос. Основное в запросе - сам сетевой адрес. Также в запросе можно указать различные метаданные: учётные данные, тип контента, куки сессии и т.п.
- Опционально загрузить тело запроса. В этом случае используется метод **setDoOutput(true)**. Передача данных, записанных в поток, возвращается через метод **getOutputStream()**
- Прочитать ответ. Заголовок ответа обычно включает метаданные, такие как тип и длина контента, даты изменения, куки сессии. Прочитать данные из потока можно через метод **getInputStream()**. Если у ответа нет тела, то метод возвращает пустой поток.
- Разорвать соединение. После прочтения ответа от сервера **HttpURLConnection** следует закрыть через вызов метода **disconnect()**. Тем самым вы освобождаете ресурсы, занимаемые соединением.

Работа с сетью

Например, для получения страницы по адресу https://developer.android.com/ можно использовать такой код:

```
URL url = new URL("https://developer.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

try {
    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
    finally {
    urlConnection.disconnect();
   }
}
```

По умолчанию **HttpURLConnection** использует метод **GET**. Для использования **POST** вызывайте **setDoOutput(true)** и посылайте данные через **openOutputStream()**. Другие HTTP-методы (OPTIONS, HEAD, PUT, DELETE and TRACE) настраиваются через метод **setRequestMethod(String)**.

Работа с сетью

Apache Http Client

Андроид версии 1.0 был выпущен с предварительной бета-версией библиотеки Apache HttpClient. Чтобы обеспечить совместимость с первым выпуском Android, API Apache HttpClient 4.0 пришлось преждевременно заморозить.

Но при этом разработчики из Apache активно работали над библиотекой, дорабатывали ее, ожидая, что Google включит последние улучшения кода в свое дерево кода. С выходом новых версии Android, появлялись breaking changes, которые ломали совместимость старых библиотек, в том числе и этой. При этом разработчики из Apache активно старались предоставить максимальную совместимость, и было готовы сделать для этого что угодно. Но увы Гугл отказала им в этом, заморозив данные правки и не включая их больше.

Версия Apache HttpClient, поставляемая с Android, фактически стала форком. В конце концов Google решил прекратить дальнейшую разработку своего форка, отказавшись от обновления до стандартной версии Apache HttpClient, сославшись на проблемы совместимости в качестве причины такого решения.

А дальше библиотеку просто без всяких адекватных объяснений выпилили с официальной рекомендацией использовать HttpUrlConnection.

Работа с сетью

Apache Http Client

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet("http://www.google.com");
HttpResponse response = client.execute(request);

// Get the response
BufferedReader rd = new BufferedReader
    (new InputStreamReader(
    response.getEntity().getContent()));

String line = "";
while ((line = rd.readLine()) != null) {
    textView.append(line);
}
```

Android Asynchronous Http Client

Асинхронный Http-клиент на основе callback вызовов для Android, созданный поверх библиотек Apache HttpClient. Все запросы выполняются за пределами основного потока пользовательского интерфейса, но любая логика callback вызова будет выполняться в том же потоке, в котором callback вызов был создан.

```
AsyncHttpClient client = new AsyncHttpClient();
client.get("http://www.google.com", new AsyncHttpResponseHandler() {
    @Override
    public void onSuccess(String response) {
        System.out.println(response);
    }
});
```

GRAVITY group

Работа с сетью

Согласно официальному посту в блоге Google, **HttpURLConnection** имел несколько багов в ранних версиях Android:

Даже сегодня, если вы загляните в исходники **Volley** от Google, вы сможете найти такое наследие:

```
if (stack == null) {
    if (Build.VERSION.SDK_INT >= 9) {
        stack = new HurlStack();
    } else {
        // Prior to Gingerbread, HttpUrlConnection was unreliable.
        // See:
http://android-developers.blogspot.com/2011/09/androids-http-clients.html
        stack = new HttpClientStack(AndroidHttpClient.newInstance(userAgent));
    }
}
```

Это классический пример фрагментации Android, которая заставляет страдать разработчиков. В 2013 году Square обратила внимание на эту проблему, когда выпускала **OkHttp**. OkHttp была создана для прямой работы с верхним уровнем сокетов Java, при этом не используя какие-либо дополнительные зависимости. Она поставляется в виде JAR-файла, так что разработчики могут использовать ее на любых устройствах с JVM (куда мы включаем, конечно, и Android).

Для упрощения перехода на их библиотеку, Square имплементировали OkHttp используя интерфейсы HttpUrlConnection и Apache client.

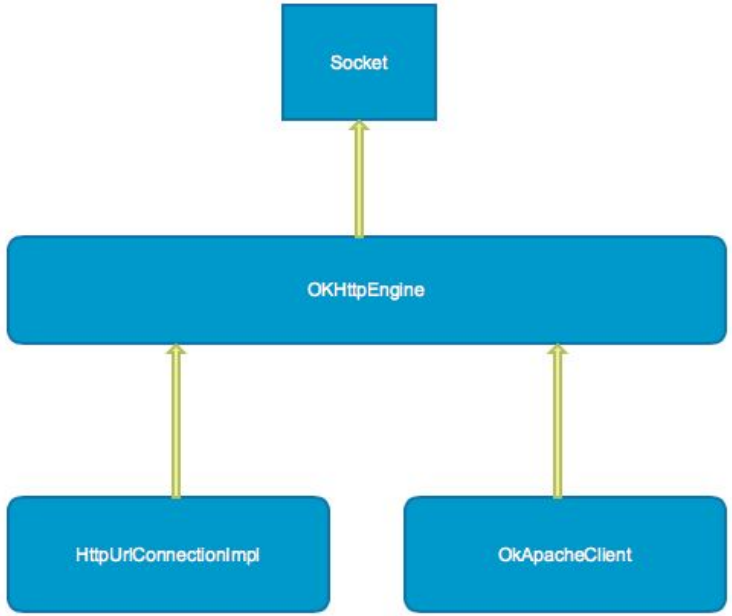
https://square.github.io/okhttp/

Работа с сетью

OkHttp получила большое распространение и поддержку сообществом, и, в конце-концов, Google решили использовать версию 1.5 в Android 4.4 (KitKat). В июле 2015 Google официально признала AndroidHttpClient, основанный на Apache, устаревшим, вместе с выходом Android 5.1 (Lolipop).

Сегодня OkHttp поставляется со следующим огромным набором функций:

1. Поддержка HTTP/2 и SPDY позволяет всем запросам, идущим к одному хосту, делиться сокетом
2. Объединение запросов уменьшает время ожидания (если SPDY не доступен)
3. Прозрачный GZIP позволяет уменьшить вес загружаемой информации
4. Кэширование ответов позволяет избежать работу с сетью при повторных запросах.
5. Поддержка как и синхронизированных блокирующих вызовов, так и асинхронных вызовов с обратным вызовом (callback)



Работа с сетью

Retrofit

Пожалуй самая популярная библиотека когда речь заходит о работы с сетью и REST API. Авторами библиотеки Retrofit являются разработчики из компании "Square", которые написали множество полезных библиотек, например, Picasso, Okhttp, Otto. По умолчанию ретрофит базируется на Okhttp

Библиотекой удобно пользоваться для запроса к различным веб-сервисам с командами GET, POST, PUT, DELETE. Может работать в асинхронном режиме, что избавляет от лишнего кода.

Основной смысл библиотеки состоит в том, чтобы описать методы работы с сетью как обычные функции.

```
public interface GerritAPI {

    @GET("changes/")
    Call<List<Change>> loadChanges(@Query("q") String status);
}
```

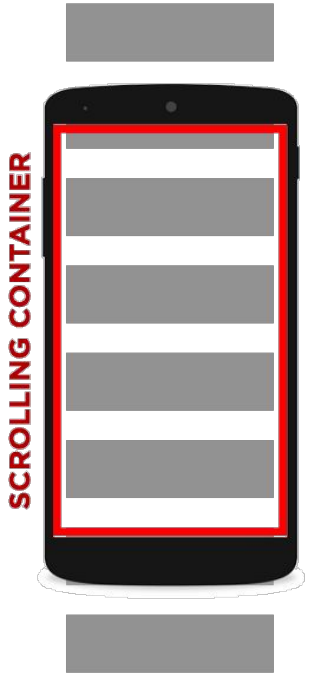
При изначально асинхронном подходе реализации доступа к данным вам становится все равно откуда они приходят. Вы используете стандартный метод языка, а внутри происходит “магия” и данные, которые вы получаете на выходе – это данные от вашего сервера.

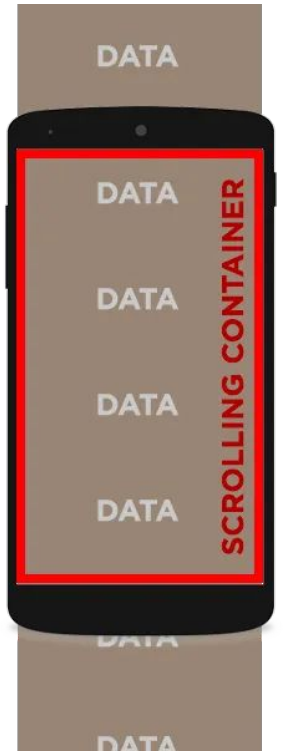
http://square.github.io/retrofit/

Списки

Android представляет широкую палитру элементов,которые представляют списки - ListView, GridView, RecyclerView. Показать длинный или условно бесконечный постраничный список – частая задача для разработчика. В android для этого раньше использовался ListView, а теперь более продвинутая версия RecyclerView

Когда View не умещаются в рамках одного экрана их можно разместить в прокручивающемся контейнере **ScrollView**





Проблема в том, что все элементы, в том числе которые не видны в данный момент, существуют в памяти. Если человек листает уже пятнадцатую страницу ленты в Facebook, ему не нужно чтобы первые записи тормозили прокрутку списка.

https://developer.android.com/reference/android/widget/ScrollView

GRAVITY group

Списки

ListView как оптимизация длинных списков

Часто верстка View для отображения данных одинаковая от элемента к элементу (или же существует ограниченное число разных типов данных).
Можно предположить, что те контейнеры, которые ушли из области видимости пользователя могут быть переиспользованы для отображения следующих данных. Это позволит оптимизировать работу приложения.

Для использования ListView в приложении необходимы:

- ListView. View, контейнер для отображения списка,
- Adapter. Менеджер данных и View – отображения данных в списке. Он отвечает за то, какой именно элемент интерфейса необходимо показать на определенной позиции.

Можно было бы подробнее остановиться на реализации ListView на Android, но Google давно представил переосмысленную версию данного компонента – **RecyclerView**.

https://developer.android.com/guide/topics/ui/layout/recyclerview

Списки

RecyclerView – замена ListView

Основные отличия RecyclerView от ListView

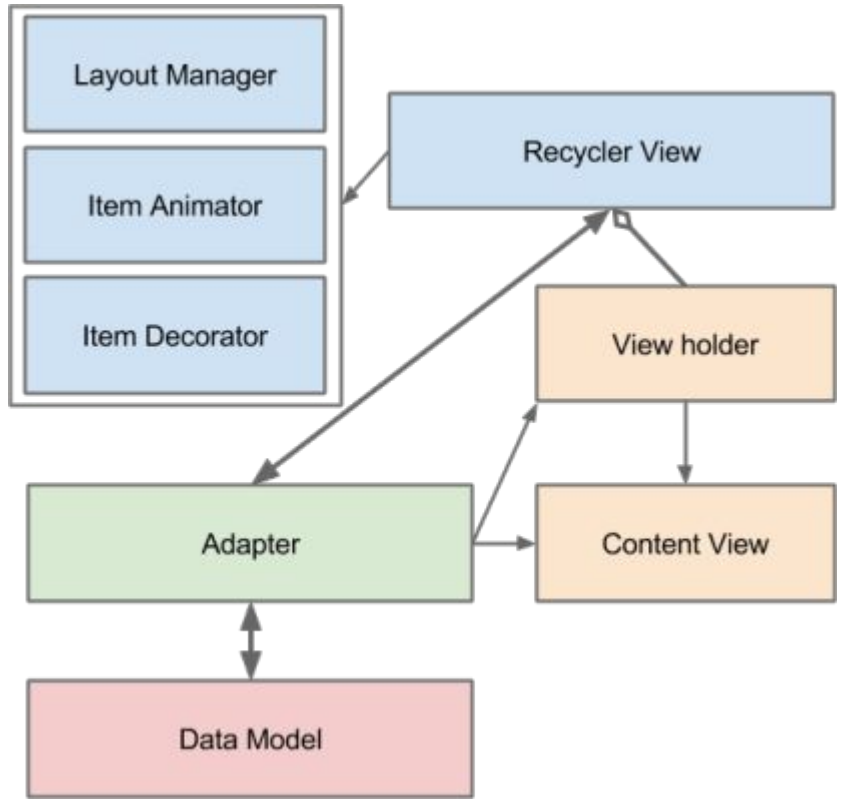
- RecyclerView отвечает только за переиспользование View, способ отображения данных задается с помощью отдельного менеджера – LayoutManager , существует ряд базовых реализаций:
    - LinearLayoutManager – отображение горизонтальных и вертикальных последовательных и инвертированных списков,
    - GridLayoutManager – отображение табличных списков,
    - StaggeredGridLayoutManager – отображение табличных списков с контейнерами динамического размера
- За анимации отдельных элементов при изменении, добавлении, удалении отвечает отдельный компонент ItemAnimator
- Принудительно используется шаблон ViewHolder. Это позволяет оптимизировать производительность за счет сохранения ссылок на View в контейнере.

https://developer.android.com/guide/topics/ui/layout/recyclerview

Списки

Адаптер, который используется в RecyclerView, должен наследоваться от абстрактного класса **RecyclerView.Adapter**. Этот класс определяет три метода:

- **onCreateViewHolder**: возвращает объект ViewHolder, который будет хранить данные.
- **onBindViewHolder**: выполняет привязку объекта ViewHolder к объекту по определенной позиции.
- **getItemCount**: возвращает количество объектов в списке



Списки

Как правильно обновлять данные в списке?

Обновление данных делится на полное и частичное обновление.

Полное обновление данных

1) Передавать новые данные в адаптер и вызывать метод notifyDataSetChanged, чтобы обновить RecyclerView

2) Создавать новый адаптер, давать ему новые данные и передавать этот адаптер в RecyclerView.setAdapter()

Проблема в том, что в обоих случаях весь список будет перерисован. Вернее, для каждой видимой строки будет вызван метод onBindViewHolder. И если у строки тяжелый layout, используется какая-либо анимация и данные адаптера обновляются достаточно часто, то на слабых девайсах вы вполне можете увидеть проблемы в скорости работы вашего списка.

```
List<Product> productList = new LinkedList<>();
productList.add(new Product(1, "Name1", 100));
productList.add(new Product(2, "Name21", 200));
productList.add(new Product(3, "Name3", 300));
productList.add(new Product(4, "Name4", 400));
productList.add(new Product(5, "Name5", 501));

adapter.setData(productList);
adapter.notifyDataSetChanged();
```

Списки

Частичное обновление данных

Вместо notifyDataSetChanged можем использовать

более точечное обновление - метод

notifyItemChanged, notifyItemRangeChanged, notifyItemInserted,

notifyItemMoved, notifyItemRemoved

```
List<Product> productList = new LinkedList<>();
productList.add(new Product(1, "Name1", 100));
productList.add(new Product(2, "Name21", 200));
productList.add(new Product(3, "Name3", 300));
productList.add(new Product(4, "Name4", 400));
productList.add(new Product(5, "Name5", 501));

adapter.setData(productList);
adapter.notifyItemChanged(1);
adapter.notifyItemChanged(4);
```

Эти точечные notify методы удобны, когда мы точно знаем, какие строки были изменены. Но если мы просто получаем новые данные извне, то будет достаточно сложно вручную все сравнивать и определять, что поменялось, а что нет. Эту работу за нас может выполнить **DiffUtil** (Google выпустил Support Library версии 25.1.0, добавив туда DiffUtil).

Он сравнит два набора данных: старый и новый, выяснит, какие произошли изменения, и с помощью notify методов оптимально обновит адаптер. DiffUtil проверяет два списка на различия, используя алгоритм Майерса, который определяет только наличие/отсутствие изменений, но не умеет находить перемещения элементов.

Для этого DiffUtil проходится по созданным алгоритмом Майерса змейкам (об этом дальше), а затем ищет перемещения. DiffResult формируется за если алгоритм не проверяет перемещения элементов и , где P – количество добавленных и удалённых элементов.

$O(N + D^2)$ $O(N + D^2) + O(P^2)$

Результатом работы алгоритма Майерса является скрипт с набором минимального количества действий, которые надо сделать для преобразования одной последовательности в другую.

www.xmailserver.org/diff2.pdf

GRAVITY group

Списки

```
public class ProductDiffUtilCallback extends DiffUtil.Callback {

    private final List<Product> oldList;
    private final List<Product> newList;

    public ProductDiffUtilCallback(List<Product> oldList, List<Product> newList) {
        this.oldList = oldList;
        this.newList = newList;
    }

    @Override
    public int getOldListSize() {
        return oldList.size();
    }

    @Override
    public int getNewListSize() {
        return newList.size();
    }

    @Override
    public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
        Product oldProduct = oldList.get(oldItemPosition);
        Product newProduct = newList.get(newItemPosition);
        return oldProduct.getId() == newProduct.getId();
    }

    @Override
    public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
        Product oldProduct = oldList.get(oldItemPosition);
        Product newProduct = newList.get(newItemPosition);
        return oldProduct.getName().equals(newProduct.getName())
                && oldProduct.getPrice() == newProduct.getPrice();
    }
}
```

А в методах **areItemsTheSame** и **areContentsTheSame** нам дают две позиции: одну из старого списка (oldItemPosition) и одну из нового (newItemPosition). Соответственно, мы из списка oldList берем Product с позицией oldItemPosition, а из newList - Product с позицией newItemPosition, и сравниваем их.

В чем ключевая разница между areItemsTheSame и areContentsTheSame?

areItemsTheSame вы сравниваете поля, чтобы в принципе определить, разные ли это объекты. А в areContentsTheSame уже сравниваете детали, чтобы определить, поменялось ли что-то из того, что вы отображаете на экране.

```
List<Product> productList = new LinkedList<>();
productList.add(new Product(1, "Name1", 100));
productList.add(new Product(2, "Name21", 200));
productList.add(new Product(3, "Name3", 300));
productList.add(new Product(4, "Name4", 400));
productList.add(new Product(5, "Name5", 501));

ProductDiffUtilCallback productDiffUtilCallback =
        new ProductDiffUtilCallback(adapter.getData(), productList);
DiffUtil.DiffResult productDiffResult = DiffUtil.calculateDiff(productDiffUtilCallback);

adapter.setData(productList);
productDiffResult.dispatchUpdatesTo(adapter);
```

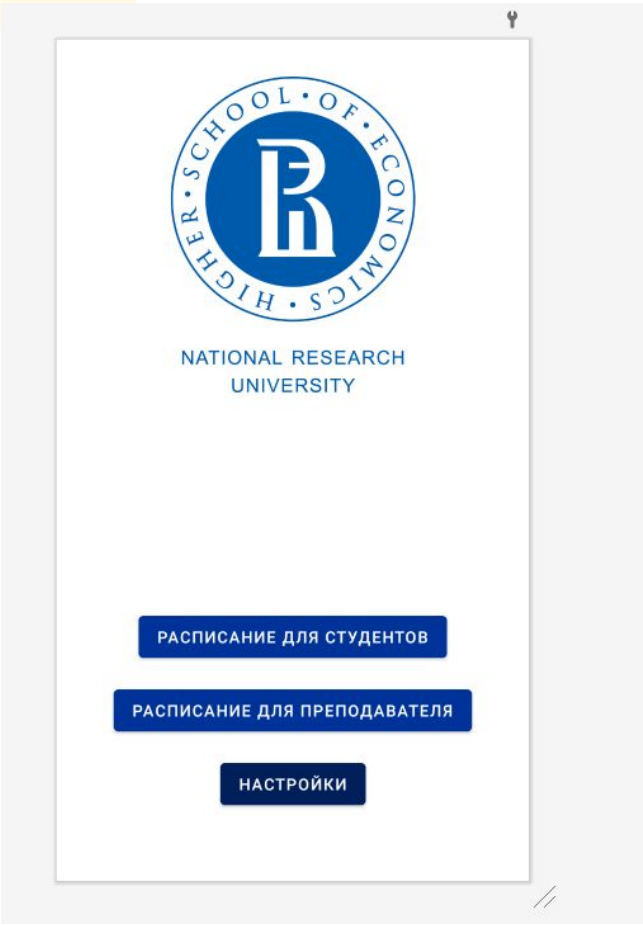
GRAVITY group

Base проект

```
implementation
'androidx.appcompat:appcompat:1.2.0'
implementation
'com.google.android.material:material:1.2.1'
```

```
// Okhttp
implementation "com.squareup.okhttp3:okhttp:4.9.0"

// Gson
implementation 'com.google.code.gson:gson:2.8.6'
```

```
public abstract class BaseActivity extends AppCompatActivity {

    private final static String TAG = "BaseActivity";
    public static final String URL = "https://api.ipgeolocation.io/ipgeo?apiKey=b03018f75ed94023a005637878ec0977";

    protected TextView time;
    protected Date currentTime;

    private OkHttpClient client = new OkHttpClient();

    protected void getTime() {
        Request request = new Request.Builder().url(URL).build();
        Call call = client.newCall(request);
        call.enqueue(new Callback() {
            public void onResponse(Call call, Response response)
                    throws IOException {
                parseResponse(response);
            }

            public void onFailure(Call call, IOException e) {
                Log.e(TAG,  msg: "getTime", e);
            }
        });
    }

    protected void initTime() {
        getTime();
    }
```



https://api.ipgeolocation.io/ipgeo?apiKey=b03018f75ed94023a005637878ec0977

GRAVITY group

Base проект

```
private void showTime(Date dateTime) {
    if (dateTime == null) {
        return;
    }
    currentTime = dateTime;
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "HH:mm, EEEE", Locale.forLanguageTag("ru"));
    time.setText(simpleDateFormat.format(currentTime));
}
```

```
public class TimeResponse {

    @SerializedName("time_zone")
    private TimeZone timeZone;

    public TimeZone getTimeZone() { return timeZone; }

    public void setTimeZone(TimeZone timeZone) { this.timeZone = timeZone; }
}
```

```
private void parseResponse(Response response) {
    Gson gson = new Gson();
    ResponseBody body = response.body();
    try {
        if (body == null) {
            return;
        }
        String string = body.string();
        Log.d(TAG, string);
        TimeResponse timeResponse = gson.fromJson(string, TimeResponse.class);
        String currentTimeVal = timeResponse.getTimeZone().getCurrentTime();
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss.SSS", Locale.getDefault());
        Date dateTime = simpleDateFormat.parse(currentTimeVal);
        // run on UI thread
        runOnUiThread(() -> showTime(dateTime));
    } catch (Exception e) {
        Log.e(TAG,  msg: "", e);
    }
}
```

```
public class TimeZone {

    @SerializedName("current_time")
    private String currentTime;

    public String getCurrentTime() { return currentTime; }

    public void setCurrentTime(String currentTime) { this.currentTime = currentTime; }
}
```

GRAVITY group

Base проект

```
public class TeacherActivity extends BaseActivity {
```

TeacherActivity.java

```
24
25      @Override
26      protected void onCreate(Bundle savedInstanceState) {
27          super.onCreate(savedInstanceState);
28          setContentView(R.layout.activity_teacher);
29
30          spinner = findViewById(R.id.groupList);
31
32          List<StudentActivity.Group> groups = new ArrayList<>();
33          initGroupList(groups);
34
35          ArrayAdapter<?> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, groups);
36          adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
37
38          spinner.setAdapter(adapter);
39
40          spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
41              public void onItemSelected(AdapterView<?> parent,
42                                         View itemSelected, int selectedItemPosition, long selectedId) {
43                  Object item = adapter.getItem(selectedItemPosition);
44                  Log.d(TAG,  msg: "selectedItem: " + item);
45              }
46
47              public void onNothingSelected(AdapterView<?> parent) {
48                  //
49              }
50          });
51
52          time = findViewById(R.id.time);
53          initTime();
54
55          status = findViewById(R.id.status);
56          subject = findViewById(R.id.subject);
57          cabinet = findViewById(R.id.cabinet);
58          corp = findViewById(R.id.corp);
59          teacher = findViewById(R.id.teacher);
60
61          View scheduleDay = findViewById(R.id.schedule_day);
62          scheduleDay.setOnClickListener(v -> showSchedule(ScheduleType.DAY));
63          View scheduleWeek = findViewById(R.id.schedule_week);
64          scheduleWeek.setOnClickListener(v -> showSchedule(ScheduleType.WEEK));
```

GRAVITY group

Base проект

```
public class StudentActivity extends BaseActivity {
```

StudentActivity.java

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

        spinner = findViewById(R.id.groupList);

        List<Group> groups = new ArrayList<>();
        initGroupList(groups);

        ArrayAdapter<?> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, groups);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

        spinner.setAdapter(adapter);

        spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            public void onItemSelected(AdapterView<?> parent,
                                       View itemSelected, int selectedItemPosition, long selectedId) {
                Object item = adapter.getItem(selectedItemPosition);
                Log.d(TAG, msg: "selectedItem: " + item);
            }

            public void onNothingSelected(AdapterView<?> parent) {
                //
            }
        });

        time = findViewById(R.id.time);
        initTime();

        status = findViewById(R.id.status);
        subject = findViewById(R.id.subject);
        cabinet = findViewById(R.id.cabinet);
        corp = findViewById(R.id.corp);
        teacher = findViewById(R.id.teacher);

        View scheduleDay = findViewById(R.id.schedule_day);
        scheduleDay.setOnClickListener(v -> showSchedule(ScheduleType.DAY));
        View scheduleWeek = findViewById(R.id.schedule_week);
        scheduleWeek.setOnClickListener(v -> showSchedule(ScheduleType.WEEK));
```

GRAVITY group

Base проект

```
private void showSchedule(ScheduleType type) {
    Object selectedItem = spinner.getSelectedItem();
    if (!(selectedItem instanceof Group)) {
        return;
    }
    showScheduleImpl(ScheduleMode.STUDENT, type, (Group) selectedItem);
}
```

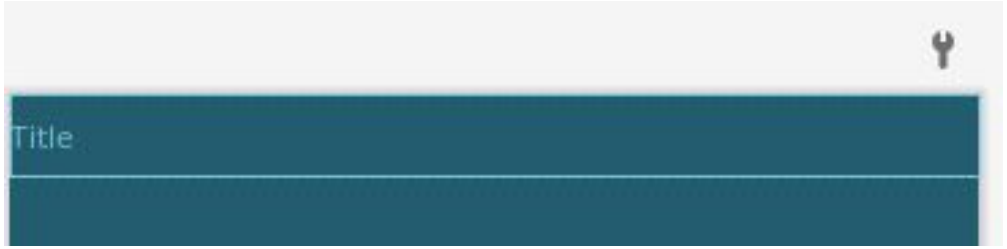
```
enum ScheduleType {
    DAY,
    WEEK
}

enum ScheduleMode {
    STUDENT,
    TEACHER
}
```

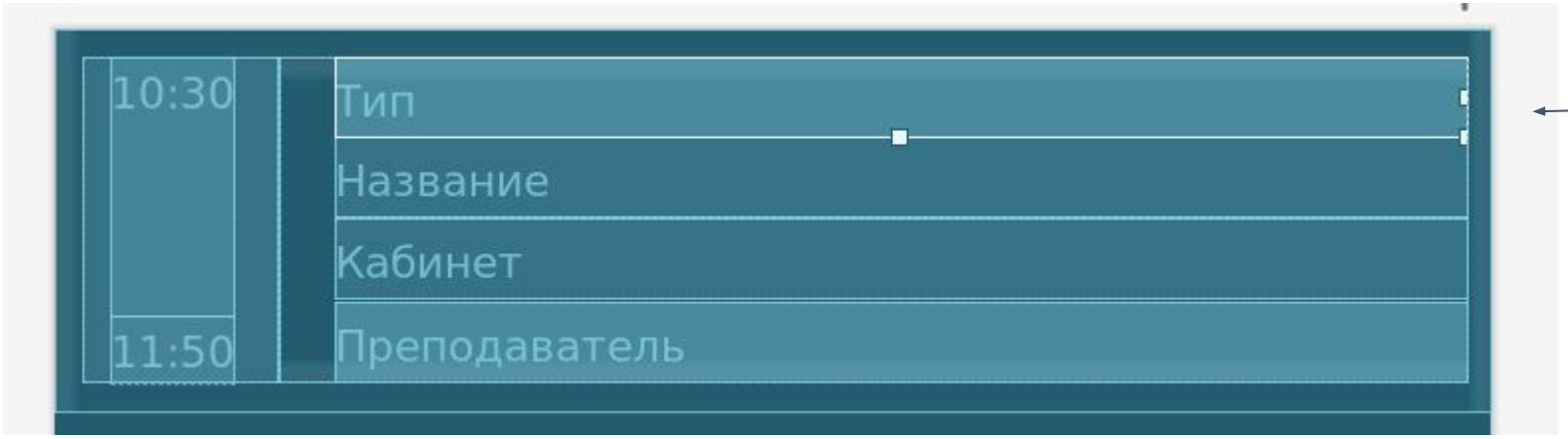
```
protected void showScheduleImpl(ScheduleMode mode, ScheduleType type, Group group) {
    Intent intent = new Intent( packageContext: this, ScheduleActivity.class);
    intent.putExtra(ScheduleActivity.ARG_ID, group.getId());
    intent.putExtra(ScheduleActivity.ARG_TYPE, type);
    intent.putExtra(ScheduleActivity.ARG_MODE, mode);
    startActivity(intent);
}
```

GRAVITY group

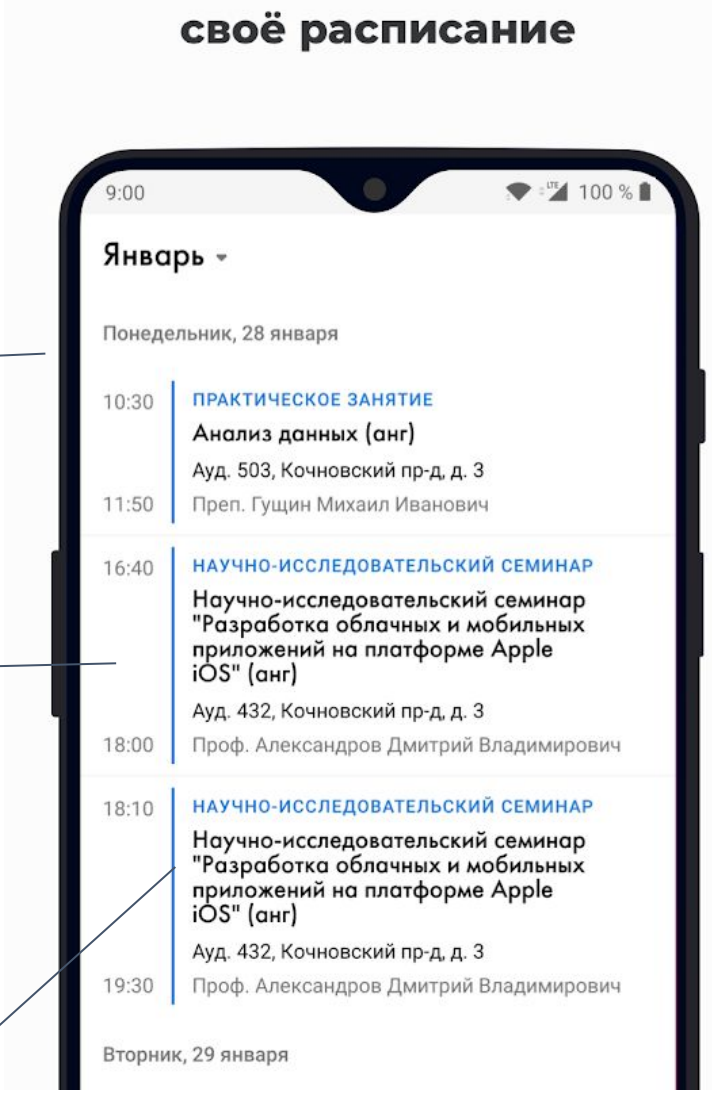
Base проект

Декомпозиция верстки






```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape android:shape="rectangle">
            <solid android:color="@color/color_4080ff" />
        </shape>
    </item>
    <item android:left="4dp">
        <shape android:shape="rectangle">
            <solid android:color="@color/white" />

            <padding android:left="4dp" />
        </shape>
    </item>
</layer-list>
```



GRAVITY group

Base проект

Расписание

```
public class ScheduleActivity extends AppCompatActivity {
```

```
  type = (BaseActivity.ScheduleType) getIntent().getSerializableExtra(ARG_TYPE);
  mode = (BaseActivity.ScheduleMode) getIntent().getSerializableExtra(ARG_MODE);
  id = getIntent().getIntExtra(ARG_ID, DEFAULT_ID);

  TextView title = findViewById(R.id.title);

  recyclerView = findViewById(R.id.listView);
  recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
  recyclerView.addItemDecoration(new DividerItemDecoration( context: this, LinearLayoutManager.VERTICAL));
  adapter = new ItemAdapter(this::onScheduleItemClick);
  recyclerView.setAdapter(adapter);
```

```
////////////////////////////////////////
public final static class ItemAdapter extends
        RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private final static int TYPE_ITEM = 0;
    private final static int TYPE_HEADER = 1;

    private List<ScheduleItem> dataList = new ArrayList<>();
    private OnItemClick onItemClick;

    public ItemAdapter(OnItemClick onItemClick) { this.onItemClick = onItemClick; }
```

GRAVITY group

Base проект

Расписание

```
public static class ViewHolder extends RecyclerView.ViewHolder {

    private Context context;
    private OnItemClick onItemClick;
    private TextView start;
    private TextView end;
    private TextView type;
    private TextView name;
    private TextView place;
    private TextView teacher;

    public ViewHolder(View itemView, Context context, OnItemClick onItemClick) {
        super(itemView);
        this.context = context;
        this.onItemClick = onItemClick;
        start = itemView.findViewById(R.id.start);
        end = itemView.findViewById(R.id.end);
        type = itemView.findViewById(R.id.type);
        name = itemView.findViewById(R.id.name);
        place = itemView.findViewById(R.id.place);
        teacher = itemView.findViewById(R.id.teacher);
    }

    public void bind(final ScheduleItem data) {
        start.setText(data.getStart());
        end.setText(data.getEnd());
        type.setText(data.getType());
        name.setText(data.getName());
        place.setText(data.getPlace());
        teacher.setText(data.getTeacher());
    }
}
```

```
public static class ViewHolderHeader extends RecyclerView.ViewHolder {
    private Context context;
    private OnItemClick onItemClick;
    private TextView title;

    public ViewHolderHeader(View itemView, Context context, OnItemClick onItemClick) {
        super(itemView);
        this.context = context;
        this.onItemClick = onItemClick;
        title = itemView.findViewById(R.id.title);
    }

    public void bind(final ScheduleItemHeader data) { title.setText(data.getTitle()); }
}
```

```
public class ScheduleItem {

    private String start;
    private String end;
    private String type;
    private String name;
    private String place;
    private String teacher;
```

```
public class ScheduleItemHeader extends ScheduleItem {

    private String title;


interface OnItemClick {
    void onClick(ScheduleItem data);
}
```

GRAVITY group

Base проект

Расписание

```
@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    Context context = parent.getContext();
    LayoutInflater inflater = LayoutInflater.from(context);

    if (viewType == TYPE_ITEM) {
        View contactView = inflater.inflate(R.layout.item_schedule, parent, attachToRoot: false);
        return new ViewHolder(contactView, context, onItemClick);
    } else if (viewType == TYPE_HEADER) {
        View contactView = inflater.inflate(R.layout.item_schedule_header, parent, attachToRoot: false);
        return new ViewHolderHeader(contactView, context, onItemClick);
    }
    throw new IllegalArgumentException("Invalid view type");
}

public int getItemViewType(int position) {
    ScheduleItem data = dataList.get(position);
    if (data instanceof ScheduleItemHeader) {
        return TYPE_HEADER;
    }
    return TYPE_ITEM;
}

@Override
public void onBindViewHolder(@NotNull RecyclerView.ViewHolder viewHolder, int position) {
    ScheduleItem data = dataList.get(position);
    if (viewHolder instanceof ViewHolder) {
        ((ViewHolder) viewHolder).bind(data);
    } else if (viewHolder instanceof ViewHolderHeader) {
        ((ViewHolderHeader) viewHolder).bind((ScheduleItemHeader) data);
    }
}
```

GRAVITY group

Base проект

Расписание

```
@Override
protected void onCreate(@Nullable Bundle savedInst
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_schedule);

    type = (BaseActivity.ScheduleType) getIntent()
    mode = (BaseActivity.ScheduleMode) getIntent()
    id = getIntent().getIntExtra(ARG_ID, DEFAULT_I

    TextView title = findViewById(R.id.title);

    recyclerView = findViewById(R.id.listView);
    recyclerView.setLayoutManager(new LinearLayout
    recyclerView.addItemDecoration(new DividerItem
    adapter = new ItemAdapter(this::onScheduleItem
    recyclerView.setAdapter(adapter);

    initData();
}
```

```
private void initData() {
    List<ScheduleItem> list = new ArrayList<>();

    ScheduleItemHeader header = new ScheduleItemHeader();
    header.setTitle("Понедельник, 28 января");
    list.add(header);

    ScheduleItem item = new ScheduleItem();
    item.setStart("10:00");
    item.setEnd("11:00");
    item.setType("Практическое занятие");
    item.setName("Анализ данных (анг)");
    item.setPlace("Ауд. 503, Кочновский пр-д, д.3");
    item.setTeacher("Пред. Гущим Михаил Иванович");
    list.add(item);

    item = new ScheduleItem();
    item.setStart("12:00");
    item.setEnd("13:00");
    item.setType("Практическое занятие");
    item.setName("Анализ данных (анг)");
    item.setPlace("Ауд. 503, Кочновский пр-д, д.3");
    item.setTeacher("Пред. Гущим Михаил Иванович");
    list.add(item);
    adapter.setDataList(list);
}
```

GRAVITY group

Base проект

Задания

1. Доработать base проект согласно презентации
2. На экран с расписанием вывести информацию о выбранной группе или преподавателе над списком с расписанием в поле title
3. Сделать стиль для поля title, чтобы заголовок был более крупного размера и вписывался в дизайн приложения
4. На экран с расписанием вывести информацию о полученном времени от сервера в формате: <день недели>, <число > <месяц>

Литература

https://developer.android.com/reference/android/widget/ScrollView

https://developer.android.com/guide/topics/ui/layout/recyclerview

https://square.github.io/okhttp/

https://habrahabr.ru/post/50147/ https://habrahabr.ru/post/158605/

https://habrahabr.ru/post/38730/ https://habrahabr.ru/post/265845/

https://ru.wikipedia.org/wiki/REST