структура папок:

* com.example.taxometr - файлы, отвечающие за содержимое окон, с которыми работает пользователь, обработку нажатий кнопок на них. Каждый класс из этой папки должен быть прописан следующим образом: **public** **class** <имя\_класса> **extends** Activity **implements** OnClickListener, messageController.OnDismissListener. а также внутри самого класса должны быть методы

@Override

**public** **void** OnDismissListener(**int** res) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onClick(View v) {

}

* com.example.taxometr.Adapter - файлы, для адаптера. он выводит необходимую нам информацию в необходимом для нас виде
* com.example.taxometr.Controller - контроллеры

структура файлов:

* com.example.taxometr

1. startActivity - с этого окна начинает свою работу приложение. Если в системе зарегистрирован только один пользователь и стоит галочка "работать монопольно", то входим под этим пользователем и переходим на mainActivity. Иначе loginActivity. В этом ничего не показывается.
2. loginActivity - показывается форма авторизации. При успешной авторизации человек отправляется на mainActivity.
3. mainActivity - главное меню для пользователя.
4. workActivity - пока ничего нет, в будущем будет логика работы водителя
5. settingActivity - каждая из кнопок, расположенной в этом окне открывает окно listViewActivity, с указанием типа (driver, mode, polygon, tarif или другие).
6. listViewActivity - т.к. все дочерние элементы меню "настройки" очень похожи внешне (имеется в виду метод и форма вывода информации пользователю), то используется класс адаптер (adapter Про него чуть ниже). В данном окне выводится необходимый список данных (например список водителей, или список настроек для данного пользователя). Есть три кнопки:  
   - добавить - открывает пустой addOrChangeListViewActivity  
    - изменить - открывает addOrChangeListViewActivity и заполняет необходимой информацией  
    - удалить - соответственно, удаляет выбранную запись.
7. addOrChangeListViewActivity - используется для добавления или редактирования данных. Выводит либо пустую форму для добавления новой записи, либо выводит заполненную форму для редактирования существующей записи ("инфо пользователя", "режим работы", "полигон" и т.д.)

* com.example.taxometr.Adapter

adapter - необходим для вывода списка информации в необходимом для нас виде.   
рассмотрим его чуть подробнее.  
  
вызывается следующим образом:  
adapter = new adapter(this, form.db.select("selectAllModes"), typeListView);  
как видно вторым параметром передается набор данных из базы данных. Третий параметр используется для передачи типа выводимой информации ("пользователь", "полигон", "сервера" и др.).  
  
затем внутри самого класса в методе   
**public** View getView(**int** position, View convertView, ViewGroup parent)  
идет перебор типа выводимой информации и соответственно ее вывод.  
  
  
**if**(convertView==**null**){

vi = *inflater*.inflate(R.layout.*item\_list\_view*, **null**);//выбираем layout для парсинга

holder = **new** ViewHolder();

holder.serialCar = (TextView) vi.findViewById(R.id.*serialCar*);//находим текстовое поле по идентификатору

holder.fio = (TextView) vi.findViewById(R.id.*fio*);//находим текстовое поле по идентификатору

holder.driverId = (TextView) vi.findViewById(R.id.*textView2*);//находим текстовое поле по идентификатору

vi.setTag( holder );

}

**else** holder=(ViewHolder) vi.getTag();

//берет один элемент из набора данных бд, который вернул котроллер бд

tempValues = (varsForAdapter) data.get( position );

//устанавливаем текстовому полю serialCar значение элемента driver.get(1) (позывной)

holder.serialCar.setText( tempValues.driver.get(1) );

//устанавливаем текстовому полю serialCar значение элемента (фио), формируется из трех элементов

holder.fio.setText( tempValues.driver.get(2)+" "+tempValues.driver.get(3)+" "+tempValues.driver.get(4));

//для скрытого элемента устанавливаем значение элемента (id)

holder.driverId.setText( tempValues.driver.get(0));

* com.example.taxometr.dbController

**public** ArrayList<varsForAdapter> select(String type){//делаем запрос из бд

/\* возвращает массив, состоящий из классов varsForAdapter

\* (внутри этого класса есть еще один массив из строк, который будет заполняться

\* информацией из бд)

\* !!!такое извращение необходимо для адаптера!!!

\*

\* например - делаем запрос

\* query = "select id, serialCar, fName, sName, tName from driver;";

\* c = database.rawQuery(query, null);

\* находим первый элемент для начала перебора данных

if(!c.moveToFirst()) {

Log.w(tag, "нет ни одной записи, удовл данному запросу");

return data;

}

do {

temp = new varsForAdapter();//создаем новую переменную

for(String str : c.getColumnNames()){

temp.driver.add(c.getString(c.getColumnIndex(str)));

}

полученная структура для нашего запроса, после этого блока будет иметь вид

temp.driver = {<id>, <serialCar>, <fName>, <sName>, <tName>}

data.add(temp);полученную переменную добавляем в переменную для возврата

} while (c.moveToNext());

c.close();

return data;

\*/  
  
**public** **void** update(String type, ArrayList<String> strArr){//фунция обновления информации

/\*

\* для обновления информации о каком-либо типе данных принимает массив из данных (строк)

\* этот массив формируется и сортируется еще до отправки в эту функцию таким образом, чтобы

\* соответствовать структуре бд необходимого типа данных.

\*

\* например, структура бд mode следующая

\* поля : id, userId, name, val, work

\*

\* поэтому массив будет формироваться следующим образом:

\* <массив> = {name, val, work } - эти данные будут обновляться

\* поле id узнаем из переменной tempId, которую установили из вызываемой формы

\* поле userId узнаем из переменной userId, которая устанавливается при авторизации пользователя

\*

\* поэтому при переборе мы опускаем эти поля id, userId.

\*/

//Log.w(tag, "type = "+strArr.size());  
  
**public** **void** insert(String type, ArrayList<String> strArr){

/\*

\* массив данных формируется таким же образом, как и для функции update() см. выше

\*/