**Здесь описаны правила написания кода, начиная от названия методов и переменных, заканчивая конструированием класса.**

**Название методов**

Метод следует называть следующим образом:

* Начинается название с маленькой буквы. (**g**oToService)
* Все последующие слова с большой буквы.(go**T**o**S**ervice)
* Первым в названии идет глагол (что делает метод). (**go**ToService, **check**Status)
* Затем идет то над чем выполняется действие или куда ведет метод. (go**ToService**, check**Status**, check**FreeLogin**).
* Имя метода должно отражать то, что он выполняет внутри.
* Методы, отвечающие за передвижения между activity, имеют следующую структуру: goTo + имя activity
* …

**Название переменных**

Переменные следует называть следующим образом:

* Начинается название с маленькой буквы. (**p**assword)
* Все последующие слова с большой буквы.(user**P**hone)
* Переменная это не действие, так что никаких глаголов (**~~go, sent …~~**)
* Булевскую переменную следует называть так, чтобы она отражала состояние TRUE (if(**freeLogin**), if(**status**), if(**~~noStatus~~**))
* Все переменные типа Boolean изначально имеют значение FALSE
* Константы следует писать в верхнем регистре (**TAG, ERROR …**)
* Если константа составная, разделять их нижним подчеркиванием (ERROR**\_**DEL**\_**ZERO)
* Переменная должна нести смысловой характер! (**~~X~~**, coordinateX)
* Имена не должны содержать цифры (**~~userPhone1~~**)
* Следует использовать антонимы (begin/end, first/last)
* Используй каждую переменную только с 1-ой целью
* Переменная объекта называется также, как id объекта
* …

**Название объектов на layout**

Объекты следует называть следующим образом:

* Button имя кнопки + имя layout + Btn (**findServiceSearchServiceBtn**)
* Button имя кнопки не содержит больше 2-х слов
* EditText (TextBox) имя поля + имя layout + Input (**passwordRegisrationInput**)
* EditText (TextBox) имя поля не содержит больше 2-х слов
* …

**Название класса / activity**

Классы следует называть следующим образом:

* Название класса должно отражать реальный объект (**profile, service, authorization**).
* Название класса не должно быть действием!
* Начинается название с маленькой буквы. (**p**rofile)
* Все последующие слова с большой буквы.(server**S**earch)

**Структура класса**

Классы следует конструировать следующим образом:

* Константы
* Переменные
* View
* База данных
* Работа с файлами
* Работа с xml-файлами
* Метод onCreate()
* Init()
* Метод onClick()
* Методы выполняющие изменения в классе
* Методы перемещения (**goTo**…)

**Структура класса DBHelper**

Класс отвечающий за базу данных стоит конструировать так:

* Версия базы данных
* Имя базы данных
* Имена таблиц
* С комментарием (пометкой) перечисления названия строк, который хранятся в базе данных ( // users **public static final** String ***KEY\_PHONE*** = **"phone"**; )
* Метод onCreate() содержит String переменные, в которые положен текст, создающий таблицу.

String users = **"create table "**+ ***TABLE\_CONTACTS\_USERS*** + **"("** + ***KEY\_ID*** + **" integer primary key,"** + ***KEY\_PHONE*** + **" text,"** + ***KEY\_PASS*** + **" text"** + **")"**;

* …

**Правила переноса при написании кода**

Так следует переносить код:

* При формировании большой строки она имеет следующий формат   
  msg += **"msg" + value  
  + "msg1" + value1**То есть, на каждой новой строке сообщение и ее значение, потом перенос по знаку «+»  
  msg += **" Name = "** + cursor.getString(indexName)  
   + **" Cost = "** + cursor.getString(indexMinCost)  
   + **" Descr = "** + cursor.getString(indexDescription)  
   + **" "**;
* Если передаваемых параметров больше 4 пишем «в стобик»  
  Cursor cursor = database.query(  
  DBHelper.***TABLE\_CONTACTS\_SERVICES***,  
   **null**,  
   **null**,  
   **null**,  
   **null**,  
   **null**,  
   **null**,  
   **null**);  
  Но  
  registration(database,name,cost,description);

**Правила работы со строками**

В нашем приложении при сравнении все строки мы опускаем в нижний регистр!

* Чтобы не сравнивались РАЗНЫЕ строки **N**ame && **n**ame

**Папка fragments**

Папка fragments хранит классы, которые отвечают за создание динамических view объектов на странице

**Написание запросов и Cursor**

Всегда комментируй сначала, что должен делать тот или иной запрос или курсор!!!

**Если пишешь свое API**

Все наши доп API находятся в папке **helpApi.**

Называются по принципу "что делает" + Api

В нем мы описываем методы, которые используются больше 1 раза в разных activity.

**Метод init()**

Метод init() идет сразу после onCreate и подгружает, инициализирует данные, создает объекты.

**Метод подгрузки данных из Firebase**

Метод должен называться **load**+данные, которые скачивает

**Метод подгрузки данных из Sqlite**

Метод должен называться **get**+данные, которые скачивает

**Метод подгрузки данных в Firebase**

Метод должен называться **upload**+данные, которые загружает

**Метод подгрузки данных в Sqlite**

Метод должен называться **add**+данные, которые загружает + **InLocalStorage**

**Основный принципы работы с загрузкой и обработкой данных**

Есть статический массив (если на эту страницу можно зайти с разным Id) или статическая переменная.

При заходе на активити, мы проверяем состояние массива или переменной.

Если не заходили на эту страницу, то выполняется метод LOAD и кладется Id этой активи(например, сервиса или юзера) в массив или меняется переменная (isFirst = false).

Если id уже есть или переменная isFirst = false, тогда выполняем метод GET.   
Внутри метода LOAD происходит загрузка из внешней БД и устанавливаются значения методом SET, а также методами ADD, данные добавляются в локальную БД.

В методе GET также вызывается метод SET, он точно такой же, как и в LOAD, разница только в том, что мы берем из локальной БД, не нужно загружать еще раз из внешней БД что-либо.

Также при проходе по методу LOAD, вешаются листенеры, которые постоянно слушают изменения в БД, если они происходят, то срабатывают методы ADD.  
