Лабораторная работа 6.

Хранение данных.

Создадим дневную и ночную темы для работы с приложением (рис.1).

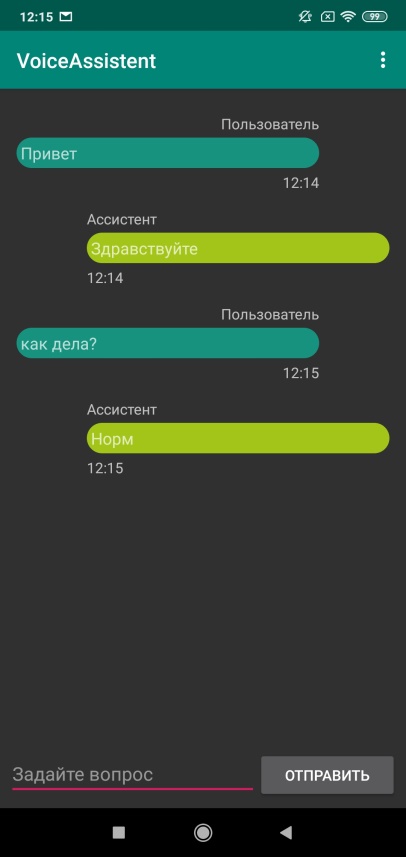
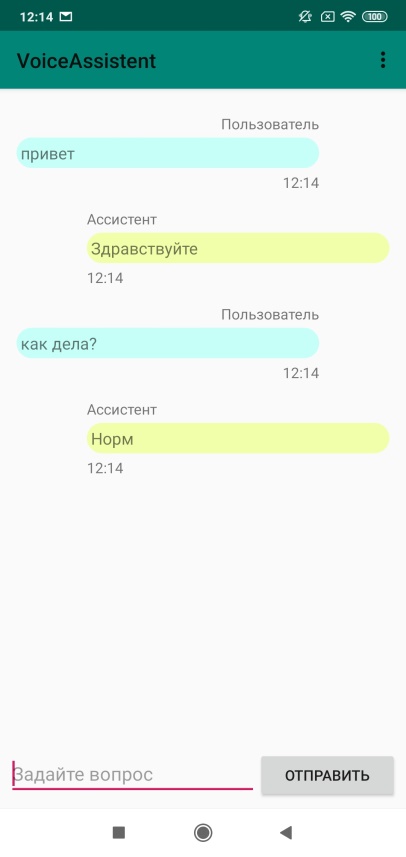


Рисунок 1. Дневная и ночная темы приложения.

Создайте в каталоге res папку menu, а в этой папке – файл разметки menu\_main.xml (рис. 2).

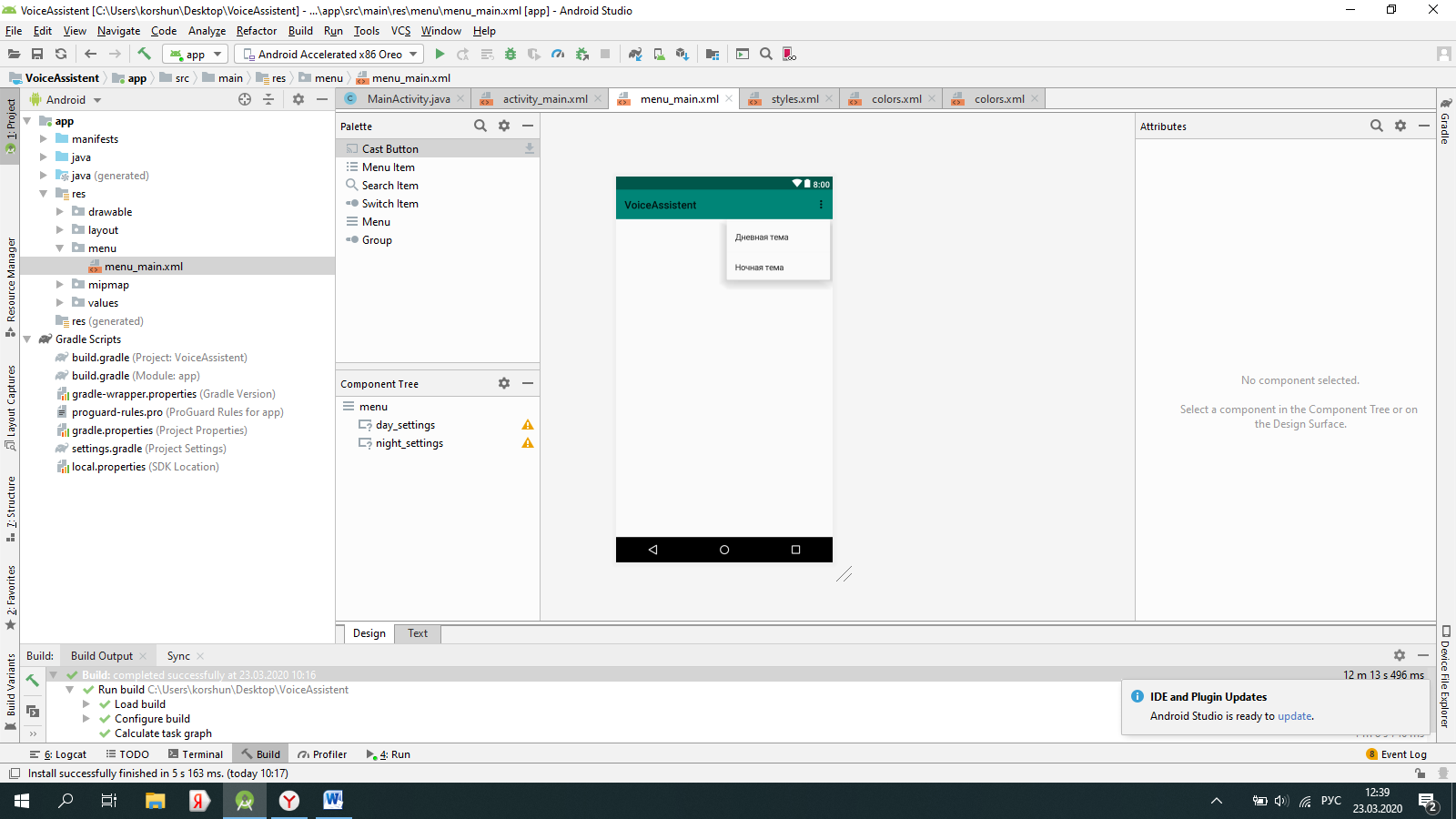


Рисунок 2. Файл res/ menu/ menu\_main.xml

В файл разметки добавьте элемент menu. Внутри него два элемента item:

*<?***xml version="1.0" encoding="utf-8"***?>*<**menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"**>  
  
 <**item  
 android:orderInCategory="100"  
 app:showAsAction="never"**/>  
  
 <**item  
 android:orderInCategory="100"  
 app:showAsAction="never"**/>  
</**menu**>

Дополните описание элементов меню.

Значение **android:id**:

первый item – **day\_settings**,

второй item – **night\_settings**.

Значение **android:title**:

первый item – **"Дневная тема"**,

второй item – **"Ночная тема"**.

В режиме дизайнера должно получиться следующее меню (рис. 3):

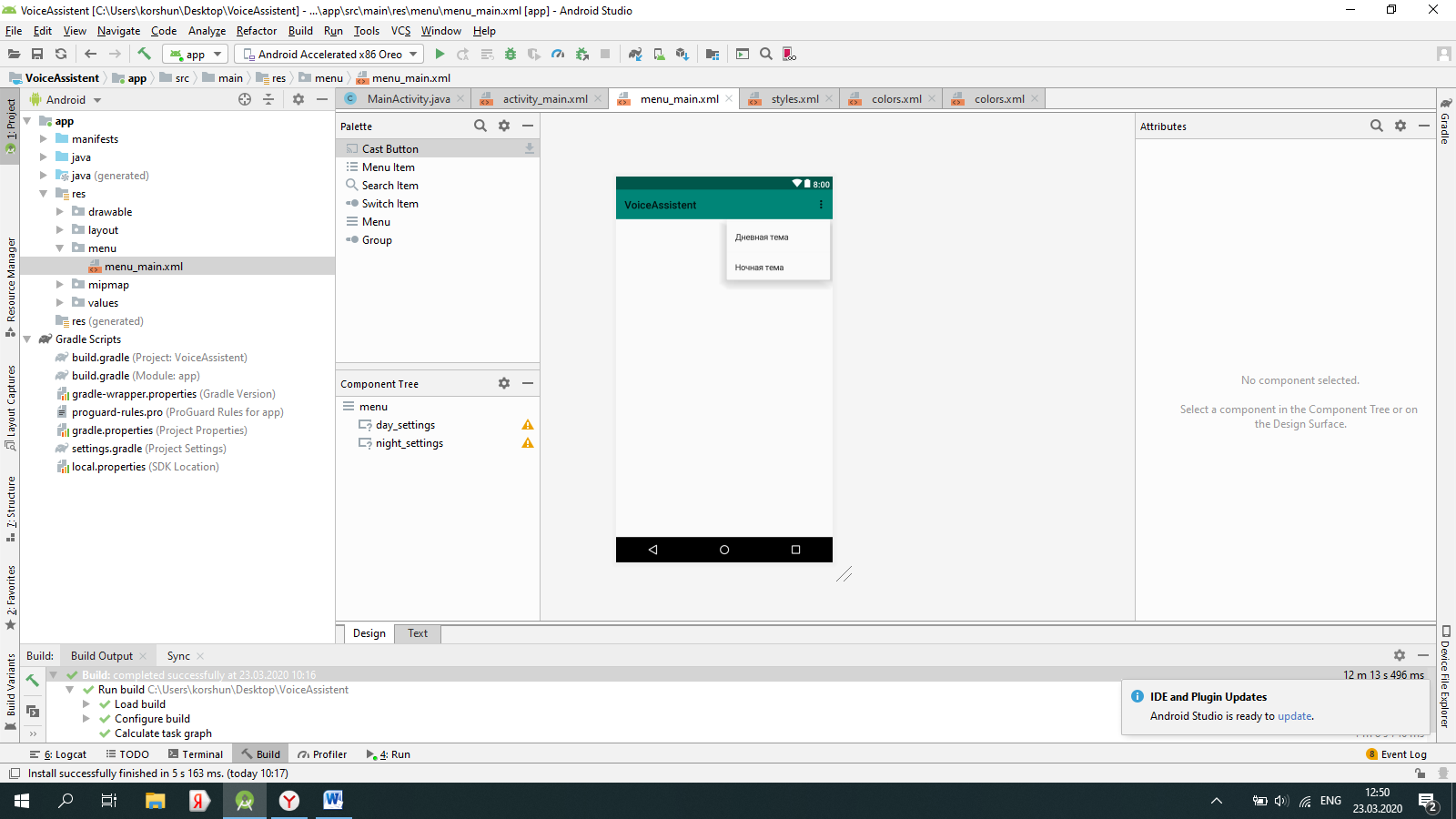


Рисунок 3. Меню в режиме Design

Чтобы разместить меню в activity нужно в теле класса MainActivity создать метод onCreateOptionsMenu. Начните набирать название метода, и студия сама подскажет, какой метод нужно выбрать. В метод добавьте одну строчку – метод, который берёт данные из ресурсов меню и преобразует их в пункты меню на экране.

@Override  
**public boolean** onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.***menu\_main***, menu);**return super**.onCreateOptionsMenu(menu);  
}

В переводе с английского "inflate" переводится как "надувать", таким образом, получается, что мы как бы надуваем данными объект, в данном случае, меню. Есть другая точка зрения: слово "inflate" происходит от словосочетания in flat – в квартиру. Таким образом, мы запускаем данные из XML-файла в объект MenuInflater, получаемый с помощью метода getMenuInflater().

Меню создано, но оно пока бесполезно, так как его пункты никак не реагируют на наши нажатия. Для обработки нажатий пунктов меню служит метод onOptionsItemSelected. Добавьте его по такому же принципу в тело класса MainActivity.

Для выбора темы будем использовать оператор выбора.

@Override  
**public boolean** onOptionsItemSelected(@NonNull MenuItem item) {  
 **switch** (item.getItemId()) {  
 **case** R.id.***day\_settings***:  
 *//установка дневной темы* **break**;  
 **case** R.id.***night\_settings***:  
 *//установка ночной темы* **break**;  
 **default**:  
 **break**;  
 }  
 **return super**.onOptionsItemSelected(item);  
}

Для смены темы будем использовать новую тему DayNight. Она поддерживает автоматическое переключение между светлой и темной темой оформления в зависимости от времени суток.

Перейдите в файл res/values/styles.xml. В нем описана тема приложения – **AppTheme**. Унаследуйте ее от темы DayNight:

<**style name="AppTheme" parent="Theme.AppCompat.DayNight"**>

Сейчас нужно сделать установку нужной темы в методе onOptionsItemSelected. Метод setLocalNightMode() позволяет переопределить значение темы по умолчанию в каждом компоненте. В случае дневной темы запишите:

getDelegate().setLocalNightMode(AppCompatDelegate.***MODE\_NIGHT\_NO***);

Установку ночной темы запишите самостоятельно.

**Запуск**. Запустите приложение и протестируйте смену тем. Есть один недостаток – цвета овалов, в которых выводится текст сообщения, остается одинаковым в обеих темах.

Исправить этот недостаток очень легко. Зайдите в файл values/colors.xml добавьте в него два цветовых ресурса:

<**color name="assistantBubble"**>#F1FFAA</**color**>  
<**color name="userBubble"**>#C6FFF7</**color**>

Цвета МОЖНО выбрать свои!

Затем в файлах assistant\_bubble.xml и user\_bubble.xml изменить значение атрибута **android:color** на **"@color/assistantBubble"** и **"@color/userBubble"** соответственно.

Пока остаются все те же два цвета для дневной темы. Чтобы создать еще два для ночной темы, вызовите контекстное меню у папки values. Выберите в нем New->Values resource file. В открывшемся меню задайте имя файла (рис.4). Выберите Night Mode и перенесите его из левого окна в правое.

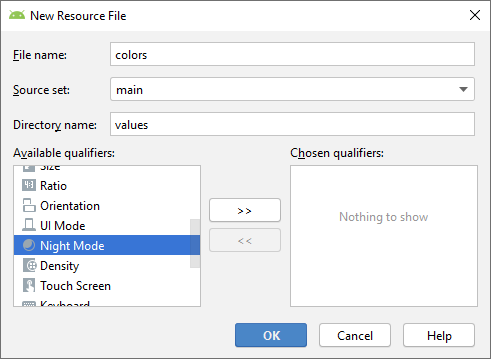


Рисунок 4. Окно создания ресурсов.

Значение Night mode установите равным Night. Убедитесь, что значение Directory name изменилось на values-night. Нажимайте OK.

Должна появиться папка colors а в ней два файла (рис. 5). Скопируйте в файл night/colors.xml цветовые ресурсы, измените цвет на более темный.

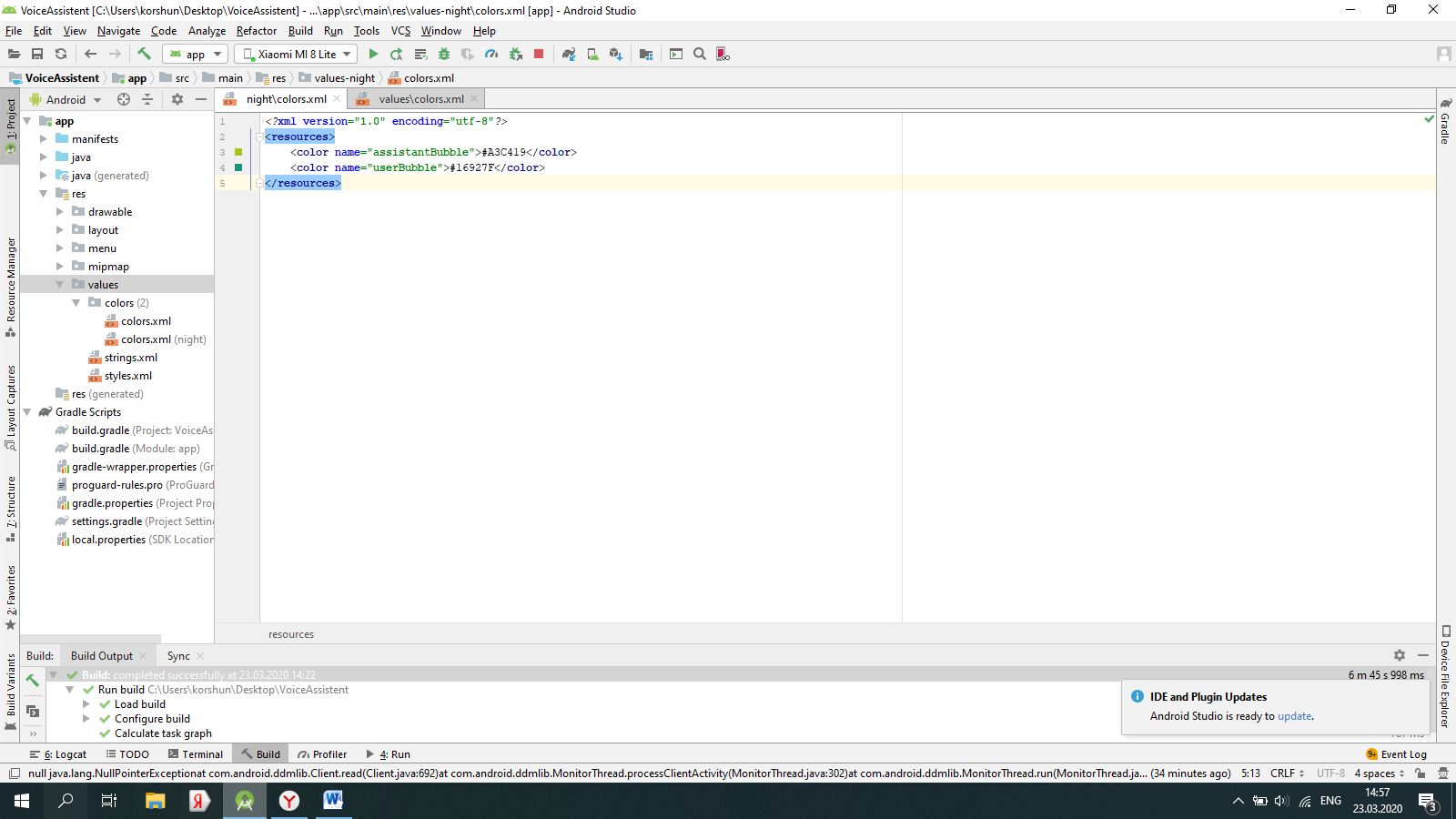


Рисунок 5. Файл night/colors.xml

**Запуск**. Запустите приложение и протестируйте смену тем. Сейчас с цветом все должно быть в прядке. Но остаются другие недочеты.

Как вы, наверное, заметили, при смене темы пропадает весь диалог. Это нужно исправить с помощью Bundle. Вы умеете это делать. Сделайте это самостоятельно. Подсказка: для добавления списка в объект Bundle используйте метод putSerializable. Для этого придется изменить тип messageList с List на ArrayList. Класс ArrayList обладает необходимым свойством.

Чтобы проследить за работой приложения, за последовательностью вызовов методов жизненного цикла Activity, реализуйте их и добавьте в них запись в Log, например, для метода onCreate:

Log.*i*(**"LOG"**, **"onCreate"**);

Проследить за тем, что выводится в Log и какие методы вызываются, можно в окне Logcat (внизу окна программы). Добавьте фильтры Info и LOG (рис. 6) чтобы лишняя информация не выводилась.

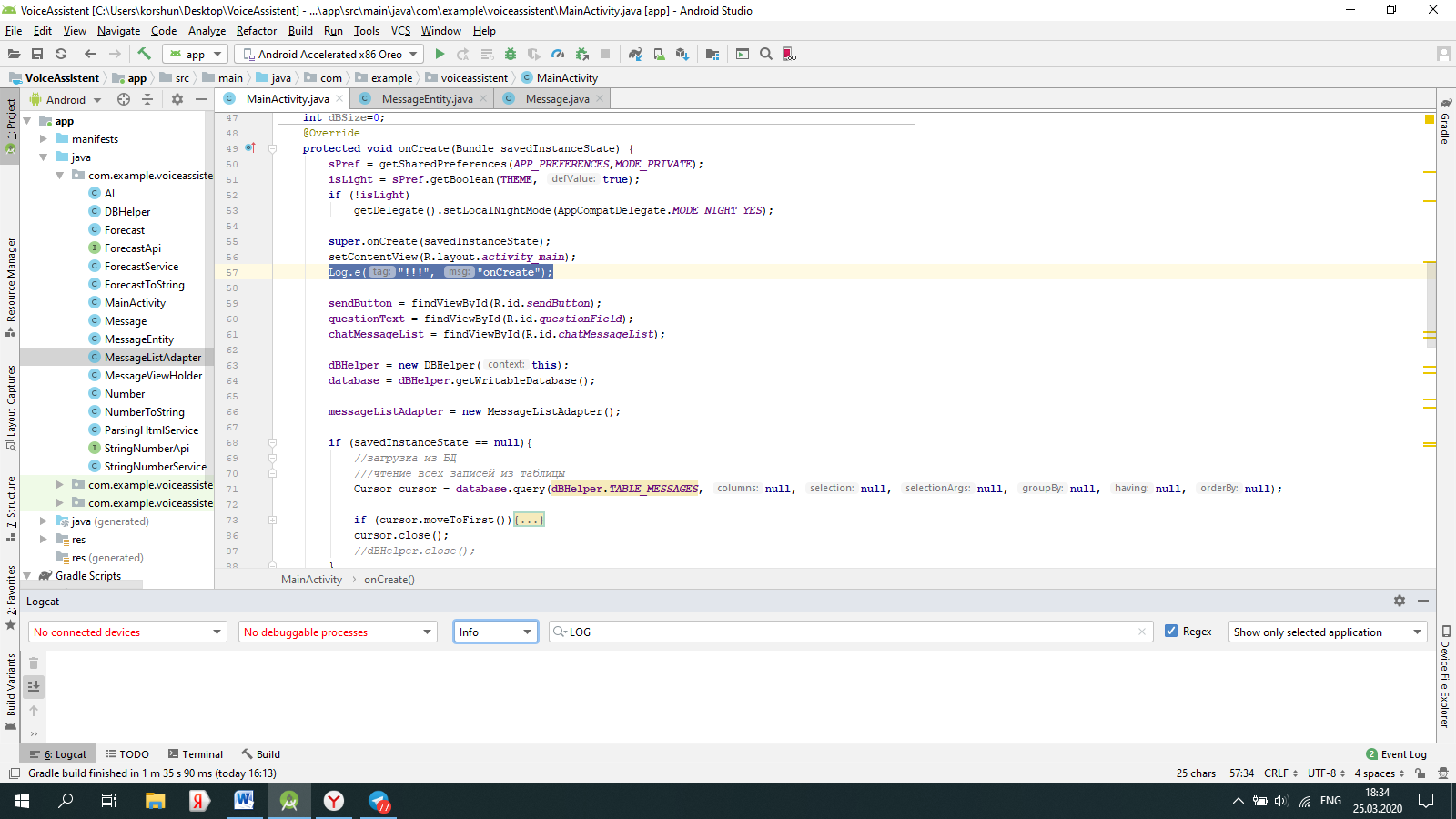


Рисунок 6. Фильтры Logcat

Еще один недостаток. Если ваше приложение было "совсем" закрыто в темной теме, а затем запущено заново. Тема будет светлой. Настройка не сохранится и Bundle здесь не поможет. Почему?

Нужно куда-то сохранить значение темы Activity. Один из способов хранения данных в Android – SharedPreferences. Значения в нем сохраняются в виде пары: имя, значение.

Опишите в классе activity поле:

SharedPreferences **sPref**;

Чтобы получить экземпляр класса SharedPreferences для получения доступа к настройкам в коде приложения будем использовать метод getSharedPreferences, передав имя общих настроек (имя файла):

В методе onCreate перед super.onCreate запишите:

**sPref** = getSharedPreferences(***APP\_PREFERENCES***,***MODE\_PRIVATE***);

Добавьте к списку полей константу:

**public static final** String ***APP\_PREFERENCES*** = **"mysettings"**;

Здесь **mysettings** – это имя файла настроек. Файлы настроек хранятся в каталоге /data/data/имя\_пакета/shared\_prefs/имя\_файла\_настроек.xml.

Параметр ***MODE\_PRIVATE*** используется для приватного доступа к данным приложением-владельцем.

Сохранение данных. Для того чтобы запомнить, какая тема была применена, вернее даже была ли светлая (дневная) тема, введем два новых поля:

**private boolean isLight** = **true**;  
**private** String **THEME** = **"THEME"**;

Эта пара переменных будет парой ключ-значение для сохранения ресурса. Логическая переменная – значение, которое отвечает за выбор темы, строковая переменная – ключ, по которому можно сохранить и изъять значение из ресурсов.

Соответствующее значение переменной isLight задайте в методе onOptionsItemSelected.

Сохранять это значение будем в методе onStop. Создайте его.

Чтобы записать значение переменной необходимо:

1. Получить объект SharedPreferences.Editor, выполнив метод edit() объекта класса SharedPreferences

Editor editor = **sPref**.edit();

1. Записать значение с помощью метода putBoolean(String key, boolean value):

editor.putBoolean(**THEME**, **isLight**);

1. Выполнить метод commit() или apply()

editor.apply();

Значение сохраниться. Осталось его изъять при создании приложения. В методе onCreate после создания sPref выполните:

**isLight** = **sPref**.getBoolean(**THEME**, **true**);

И если значение переменной ложно, примените темную тему.

Подробнее прочитать про SharedPreferences можно в <https://www.fandroid.info/sharedpreferences-sohranenie-dannyh-v-postoyannoe-hranilishhe-android/> или в <http://developer.alexanderklimov.ru/android/theory/sharedpreferences.php>

**Запуск**. Запустите и протестируйте приложение.

Каждый раз приложение стартует с пустым диалогом (если только оно не было оставлено в состоянии Pause). Предыдущие записи не сохраняются. Дополним приложение возможностью хранить данные предыдущих диалогов.

Сделать это с помощью класса SharedPreferences нельзя, так как он может хранить только простые типы (Float, Boolean, Int, Long, String) и небольшие данные.

Нужно использовать базу данных.

Создайте класс DBHelper, унаследуйте его от класса SQLiteOpenHelper. Наследование обязывает реализовать два обязательных абстрактных метода: onCreate и onUpgrade. Метод onCreate вызывается при создании базы данных (при первом обращении к базе данных), метод onUpgrade вызывается при изменении базы данных (когда указанный в программе номер базы данных выше, чем в самой базе данных). Так же обязательным для реализации является конструктор.

Добавьте в класс следующие константы: версия и название базы данных, название таблицы и ее полей.

**public static final int *DATABASE\_VERSION*** = 1;  
**public static final** String ***DATABASE\_NAME***=**"messageDb"**;  
**public static final** String ***TABLE\_MESSAGES***=**"messages"**;  
  
**public static final** String ***FIELD\_ID*** = **"id"**;  
**public static final** String ***FIELD\_MESSAGE*** = **"message"**;  
**public static final** String ***FIELD\_SEND*** = **"send"**;  
**public static final** String ***FIELD\_DATE*** = **"date"**;

Вернитесь к конструктору. Удалите из заголовка все параметры, кроме context. В вызове конструктора суперкласса запишите следующие значения:

**public** DBHelper(@Nullable Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
}

В методе onCreate выполните запрос на создание таблицы:

db.execSQL(**"create table "** + ***TABLE\_MESSAGES*** + **"("** +  
 ***FIELD\_ID*** + **" integer primary key,"** +  
 ***FIELD\_MESSAGE*** + **" text,"** +  
 ***FIELD\_SEND*** + **" integer,"** +  
 ***FIELD\_DATE*** + **" text"** + **")"**);

Метод execSQL позволяет выполнить любой запрос к базе данных, написанный на языке SQL.

В методе onUpgrade можно сначала удалить старую версию базы данных, а затем создать новую:

db.execSQL(**"drop table if exists "** + ***TABLE\_MESSAGES***);  
onCreate(db);

Перейдите в класс MainActivity и создайте поля для работы с базой данных:

DBHelper **dBHelper**;  
SQLiteDatabase **database**;

В методе onCreate создайте экземпляры классов:

**dBHelper** = **new** DBHelper(**this**);  
**database** = **dBHelper**.getWritableDatabase();

Класс SQLiteDatabase предназначен для управления базой данных SQLite. Метод getWritableDatabase откроет для чтения и записи, и вернет экземпляр базы данных, с которой нужно работать.

В классе SQLiteDatabase определены следующие методы:

query() – чтение данных из БД

insert() – добавление данных в БД

delete() – удаление данных из БД

update() – изменение данных в БД

Сначала реализуем сохранение сообщений в таблицу. Данные, которые нужно записать в базу данных, хранятся в списке messageListAdapter.messageList. Элемент списка – это экземпляр класса Message, имеющий поля следующих типов: строка, тала и логическое поле.

Каждое значение, хранящееся в базе данных SQLite, имеет один из следующих классов хранения: TEXT, NUMERIC, INTEGER, REAL, NONE. Значения типа boolean рекомендуют хранить как 1 или 0. Для хранения даты рекомендуют использовать строку.

Поэтому для работы с базой данных создайте специальный класс MessageEntity. В нем опишите следующие поля:

**public** String **text**;  
**public** String **date**;  
**public int isSend**;

Опишите в нем два конструктора. Первый конструктор будет использоваться для создания элемента из базы данных, второй для преобразования элемента Message в удобный для хранения в базе данных элемент:

**public** MessageEntity(String text, String date, **int** isSend){  
}  
  
**public** MessageEntity(Message message){  
}

В класс Message добавьте конструктор для преобразования элемента MessageEntity в Message:

**public** Message(MessageEntity entity) {  
}

Итак, для работы с приложение удобен класс Message, для работы с базой данных удобен класс MessageEntity. Конструкторы устанавливают между ними взаимосвязь, преобразуют данные из одного вида в другой. Если не хотите конструкторы, придумайте методы.

В классе MainActivity сохранение данных будет реализуйте в методе onStop. Чтобы данные не повторялись, удалите все содержимое в таблице:

**database**.delete(**dBHelper**.***TABLE\_MESSAGES***, **null**, **null**);

Дальше нужно пробежать по всему списку messageListAdapter.messageList и выполнить следующее:

1. преобразовать объект класса Message, хранящийся в messageListAdapter.messageList в объект класса MessageEntity:

MessageEntity entity = **new** MessageEntity(**messageListAdapter**.**messageList**.get(i));

1. добавить новые данные в таблицу. Для этого создайте объект класса ContentValues и положите в него значения необходимые значения:

ContentValues contentValues = **new** ContentValues();  
contentValues.put(DBHelper.***FIELD\_MESSAGE***, entity.**text**);  
contentValues.put(DBHelper.***FIELD\_SEND***, entity.**isSend**);  
contentValues.put(DBHelper.***FIELD\_DATE***, entity.**date**);

Значения добавляются в contentValues по ключу. Далее добавьте получившийся объект в базу:

**database**.insert(**dBHelper**.***TABLE\_MESSAGES***,**null**,contentValues);

Считывание данных из базы выполним в методе onCreate. Когда нужно читать данные из базы? Когда приложение было полностью закрыто, то есть activity "пережило" метод onDestroy. В этом случае параметр savedInstanceState будет равен null.

Для чтения данных используется экземпляр класса Cursor:

Cursor cursor = **database**.query(**dBHelper**.***TABLE\_MESSAGES***, **null**, **null**, **null**, **null**, **null**, **null**);

При его создании указана таблица, по которой курсор будет "передвигаться". Подробнее о классе Cursor можно почитать здесь <http://developer.alexanderklimov.ru/android/sqlite/cursor.php>.

Метод moveToFirst передвигает курсор к первой записи в таблице и одновременно возвращает возможность сделать это. Для каждого поля создается свой индекс.

**if** (cursor.moveToFirst()){  
 **int** messageIndex = cursor.getColumnIndex(**dBHelper**.***FIELD\_MESSAGE***);  
 **int** dateIndex = cursor.getColumnIndex(**dBHelper**.***FIELD\_DATE***);  
 **int** sendIndex = cursor.getColumnIndex(**dBHelper**.***FIELD\_SEND***);  
  
 **do**{  
 MessageEntity entity = **new** MessageEntity(cursor.getString(messageIndex),  
 cursor.getString(dateIndex), cursor.getInt(sendIndex));  
 Message message = **new** Message(entity);  
 **messageListAdapter**.**messageList**.add(message);  
 }**while** (cursor.moveToNext());  
}

Просматриваем все запись друг за другом. Сначала получаем из таблицы объект класса MessageEntity, затем переводим его в объект класса Message и добавляем в список messageListAdapter.messageList.

После того, как таблица просмотрена, курсор необходимо закрыть:

cursor.close();

Закройте базу данных в методе onDestroy:

cursor.close();

**Запуск**! Запустите и убедитесь в том, что диалог сохранился.

На самом деле, базу данных SQLite используют в чистом виде только ученики. Обычно для облегчения работы с ней используется библиотека Room. Именно она работает с SQLite, а вы работаете с ней.

SQLite – это реляционная база данных. Кроме нее используется нереляционая база данных – Realm. Можно подключаться к Firebase.