

Практическая работа 13

Методические материалы

Настройки и состояние приложения.

Работа с файловой системой.

Работа с базами данных SQLite

Создание и получение настроек. SharedPreferences

Сохранение состояния приложения

В одной из предыдущих тем был рассмотрен жизненный цикл Activity в приложении на Android, где после создания Activity вызывался метод **onRestoreInstanceState**, который восстанавливал ее состояние, а перед завершением работы вызывался метод **onSaveInstanceState**, который сохранял состояние Activity. Оба этих метода в качестве параметра принимают объект **Bundle**, который как раз и хранит состояние activity:

```
protected void onRestoreInstanceState(Bundle savedInstanceState);
```

```
protected void onSaveInstanceState(Bundle savedInstanceState);
```

В какой ситуации могут быть уместно использование подобных методов? Банальная ситуация - переворот экрана и переход от портретной ориентации к альбомной и наоборот. Если, к примеру, графический интерфейс содержит текстовое поле для вывода TextView, и мы программно изменяем его текст, то после изменения ориентации экрана его текст может исчезнуть. Или если у нас глобальные переменные, то при изменении ориентации экрана их значения могут быть сброшены до значений по умолчанию.

Чтобы точнее понять проблему, с которой мы можем столкнуться, рассмотрим пример. Изменим файл **activity_main** следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent" >
```

```
<EditText
```

```
    android:id="@+id/nameBox"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="Введите имя"
    app:layout_constraintBottom_toTopOf="@id/saveButton"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<Button
```

```
    android:id="@+id/saveButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Сохранить"
    android:onClick="saveName"
    app:layout_constraintBottom_toTopOf="@id/nameView"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/nameBox"/>
```

```
<TextView
```

```
    android:id="@+id/nameView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp"
```

```

        app:layout_constraintBottom_toTopOf="@id/getButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/saveButton"/>
<Button
    android:id="@+id/getButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Получить имя"
    android:onClick="getName"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/nameView"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Здесь определено поле EditText, в которое вводим имя. И также определена кнопка для его сохранения.

Далее для вывода сохраненного имени предназначено поле TextView, а для получения сохраненного имени - вторая кнопка.

Теперь изменим класс **MainActivity**:

```

package com.example.settingsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

```

```
String name ="undefined";
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

```
public void saveName(View view) {
```

```
    // получаем введенное имя
```

```
    EditText nameBox = findViewById(R.id.nameBox);
```

```
    name = nameBox.getText().toString();
```

```
}
```

```
public void getName(View view) {
```

```
    // получаем сохраненное имя
```

```
    TextView nameView = findViewById(R.id.nameView);
```

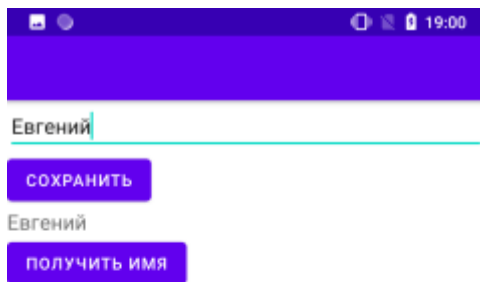
```
    nameView.setText(name);
```

```
}
```

```
}
```

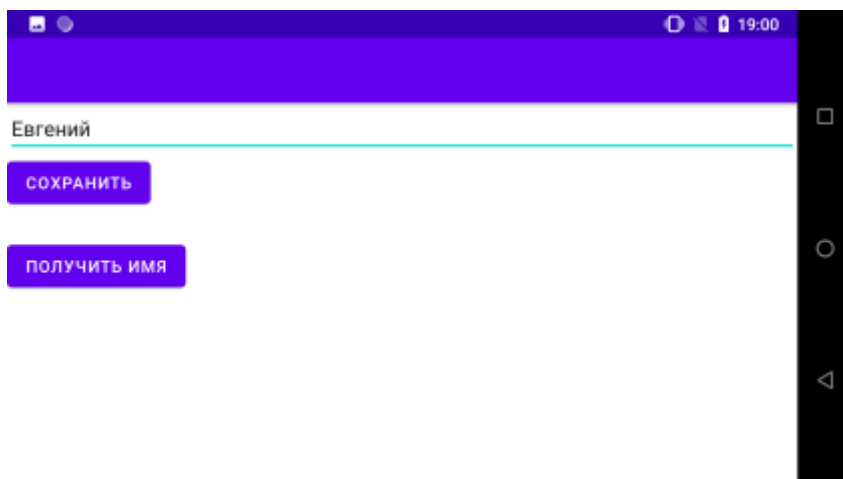
Для хранения имени в программе определена переменная name. При нажатии на первую кнопку сохраняем текст из EditText в переменную name, а при нажатии на вторую кнопку - обратно получаем текст из переменной name в поле TextView.

Запустим приложение введем какое-нибудь имя, сохраним и получим его в TextView:



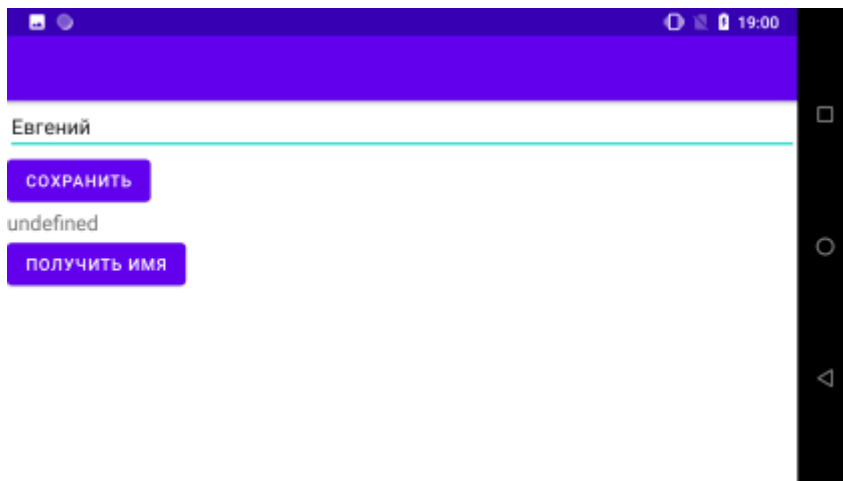
Состояние activity в Android и Java

Но если мы перейдем к альбомному режиму, то `TextView` окажется пустым, несмотря на то, что в него вроде бы уже получили нужное значение:



Состояние Bundle activity в Android и Java

И даже если мы попробуем заново получить значение из переменной `name`, то мы увидим, что она обнулилась:



Сохранение состояния activity в Android и Java

Чтобы избежать подобных ситуаций как раз и следует сохранять и восстанавливать состояние activity. Для этого изменим код MainActivity:

```
package com.example.settingsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.EditText;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    String name ="undefined";
```

```
    final static String nameVariableKey = "NAME_VARIABLE";
```

```
    TextView nameView;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```

        setContentView(R.layout.activity_main);

        nameView = findViewById(R.id.nameView);
    }

    // сохранение состояния
    @Override
    protected void onSaveInstanceState(Bundle outState) {

        outState.putString(nameVariableKey, name);
        super.onSaveInstanceState(outState);
    }

    // получение ранее сохраненного состояния
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        name = savedInstanceState.getString(nameVariableKey);
        nameView.setText(name);
    }

    public void saveName(View view) {

        // получаем введенное имя
        EditText nameBox = findViewById(R.id.nameBox);

        // сохраняем его в переменную name
        name = nameBox.getText().toString();
    }

    public void getName(View view) {

```

```
// выводим сохраненное имя  
nameView.setText(name);  
}  
}
```

В методе `onSaveInstanceState()` сохраняем состояние. Для этого вызываем у параметра `Bundle` метод **`putString(key, value)`**, первый параметр которого - ключ, а второй - значение сохраняемых данных. В данном случае мы сохраняем строку, поэтому вызываем метод `putString()`. Для сохранения объектов других типов данных мы можем вызвать соответствующий метод:

- **`put()`**: универсальный метод, который добавляет значение типа `Object`. Соответственно поле получения данное значение необходимо преобразовать к нужному типу
- **`putString()`**: добавляет объект типа `String`
- **`putInt()`**: добавляет значение типа `int`
- **`putByte()`**: добавляет значение типа `byte`
- **`putChar()`**: добавляет значение типа `char`
- **`putShort()`**: добавляет значение типа `short`
- **`putLong()`**: добавляет значение типа `long`
- **`putFloat()`**: добавляет значение типа `float`
- **`putDouble()`**: добавляет значение типа `double`

- **putBoolean():** добавляет значение типа boolean
- **putCharArray():** добавляет массив объектов char
- **putIntArray():** добавляет массив объектов int
- **putFloatArray():** добавляет массив объектов float
- **putSerializable():** добавляет объект интерфейса Serializable
- **putParcelable():** добавляет объект Parcelable

Каждый такой метод также в качестве первого параметра принимает ключа, а в качестве второго - значение.

В методе **onRestoreInstanceState** происходит обратный процесс - с помощью метода **getString(key)** по ключу получаем из сохраненного состояния строку по ключу. Соответственно для получения данных других типов мы можем использовать аналогичные методы:

- **get():** универсальный метод, который возвращает значение типа Object. Соответственно поле получения данное значение необходимо преобразовать к нужному типу
- **getString():** возвращает объект типа String
- **getInt():** возвращает значение типа int
- **getByte():** возвращает значение типа byte

- **getChar():** возвращает значение типа char
- **getShort():** возвращает значение типа short
- **getLong():** возвращает значение типа long
- **getFloat():** возвращает значение типа float
- **getDouble():** возвращает значение типа double
- **getBoolean():** возвращает значение типа boolean
- **getCharArray():** возвращает массив объектов char
- **getIntArray():** возвращает массив объектов int
- **getFloatArray():** возвращает массив объектов float
- **getSerializable():** возвращает объект интерфейса Serializable
- **getParcelable():** возвращает объект Parcelable

Для примера рассмотрим сохранение-получение более сложных данных. Например, объектов определенного класса. Пусть у нас есть класс User:

```
package com.example.settingsapp;
```

```
import java.io.Serializable;
```

```
public class User implements Serializable {
```

```
    private String name;
```

```
    private int age;
```

```
    public User(String name, int age){
```

```
        this.name = name;
```

```
        this.age = age;
```

```
    }
```

```
    public String getName() {
```

```
        return name;
```

```
    }
```

```
    public void setName(String name) {
```

```
        this.name = name;
```

```
    }
```

```
    public int getAge() {
```

```
        return age;
```

```
    }
```

```
    public void setAge(int age) {
```

```
        this.age = age;
```

```
    }
```

```
}
```

Класс User реализует интерфейс **Serializable**, поэтому мы можем сохранить его объекты с помощью метода putSerializable(), а получить с помощью метода getSerializable().

Пусть у нас будет следующий интерфейс в **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/nameBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите имя"
        app:layout_constraintBottom_toTopOf="@id/yearBox"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/yearBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите возраст"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toTopOf="@id/saveButton"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<Button
    android:id="@+id/saveButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Сохранить"
    android:onClick="saveData"
    app:layout_constraintBottom_toTopOf="@id/dataView"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/yearBox"/>
```

```
<TextView
    android:id="@+id/dataView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    app:layout_constraintBottom_toTopOf="@id/getButton"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/saveButton"/>
```

```
<Button
    android:id="@+id/getButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Получить данные"
    android:onClick="getData"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/dataView"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определены два поля ввода для имени и возраста соответственно.

В классе **MainActivity** пропишем логику сохранения и получения данных:

```
package com.example.settingsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.EditText;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    User user = new User("undefined", 0);
```

```
    final static String userVariableKey = "USER_VARIABLE";
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
    }
```

```
    // сохранение состояния
```

```
    @Override
```

```
    protected void onSaveInstanceState(Bundle outState) {
```

```
        outState.putSerializable(userVariableKey, user);
```

```

        super.onSaveInstanceState(outState);
    }
    // получение ранее сохраненного состояния
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        // получаем объект User в переменную
        user = (User)savedInstanceState.getSerializable(userVariableKey);
        TextView dataView = findViewById(R.id.dataView);
        dataView.setText("Name: " + user.getName() + " Age: " + user.getAge());
    }
    public void saveData(View view) {

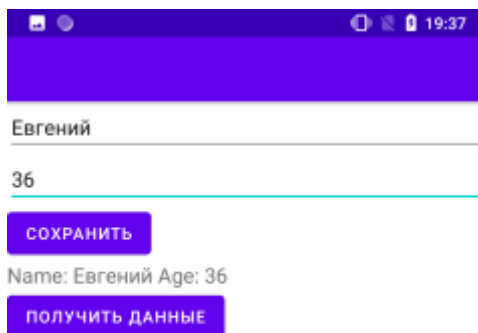
        // получаем введенные данные
        EditText nameBox = findViewById(R.id.nameBox);
        EditText yearBox = findViewById(R.id.yearBox);
        String name = nameBox.getText().toString();
        int age = 0; // значение по умолчанию, если пользователь ввел
некорректные данные
        try{
            age = Integer.parseInt(yearBox.getText().toString());
        }
        catch (NumberFormatException ex){ }
        user = new User(name, age);
    }
    public void getData(View view) {

        // получаем сохраненные данные
        TextView dataView = findViewById(R.id.dataView);

```

```
        textView.setText("Name: " + user.getName() + " Age: " + user.getAge());  
    }  
}
```

Здесь также сохраняем данные в переменную User, которая предварительно инициализированна некоторыми данными по умолчанию. А при нажатии на кнопку получения получем данные из переменной и передаем их для вывода в текстовое поле.



onRestoreInstanceState и onSaveInstanceState в Android и Java

Создание и получение настроек. SharedPreferences

Нередко приложению требуется сохранять небольшие кусочки данных для дальнейшего использования, например, данные о пользователе, настройки конфигурации и т.д. Для этого в Android существует концепция Preferences или настройки. Настройки представляют собой группу пар ключ-значение, которые используются приложением.

В качестве значений могут выступать данные следующих типов: Boolean, Float, Integer, Long, String, набор строк.

Настройки могут быть общими для всех activity в приложении, но также могут быть и настройки непосредственно для отдельных activity

Настройки хранятся в xml-файлах в незашифрованном виде в локальном хранилище. Они невидимы, поэтому для простого пользователя недоступны.

При работе с настройками следует учитывать следующие моменты. Так как они хранятся в незашифрованном виде, то не рекомендуется сохранять в них чувствительные данные типа пароля или номеров кредитных карт. Кроме того, они представляют данные, ассоциированные с приложением, и через панель управления приложением в Настройках ОС пользователь может удалить эти данные.

Общие настройки

Для работы с разделяемыми настройками в классе Activity (точнее в его базовом классе Context) имеется метод **getSharedPreferences()**:

```
import android.content.SharedPreferences;
```

```
//.....
```

```
SharedPreferences settings = getSharedPreferences("PreferencesName",  
MODE_PRIVATE);
```

Первый параметр метода указывает на название настроек. В данном случае название - "**PreferencesName**". Если настроек с подобным названием нет, то они создаются при вызове данного метода. Вторым параметром указывается на режим доступа. В данном случае режим описан константой **MODE_PRIVATE**

Класс `android.content.SharedPreferences` предоставляет ряд методов для управления настройками:

- `contains(String key)`: возвращает `true`, если в настройках сохранено значение с ключом `key`
- `getAll()`: возвращает все сохраненные в настройках значения
- `getBoolean (String key, boolean defValue)`: возвращает из настроек значение типа `Boolean`, которое имеет ключ `key`. Если элемента с таким ключом не окажется, то возвращается значение `defValue`, передаваемое вторым параметром
- `getFloat(String key, float defValue)`: возвращает значение типа `float` с ключом `key`. Если элемента с таким ключом не окажется, то возвращается значение `defValue`
- `getInt(String key, int defValue)`: возвращает значение типа `int` с ключом `key`
- `getLong(String key, long defValue)`: возвращает значение типа `long` с ключом `key`
- `getString(String key, String defValue)`: возвращает строковое значение с ключом `key`
- `getStringSet(String key, Set<String> defValues)`: возвращает массив строк с ключом `key`
- `edit()`: возвращает объект `SharedPreferences.Editor`, который используется для редактирования настроек

Для управления настройками используется объект класса **SharedPreferences.Editor**, возвращаемый метод `edit()`. Он определяет следующие методы:

- `clear()`: удаляет все настройки
- `remove(String key)`: удаляет из настроек значение с ключом `key`
- `putBoolean(String key, boolean value)`: добавляет в настройки значение типа `boolean` с ключом `key`
- `putFloat(String key, float value)`: добавляет в настройки значение типа `float` с ключом `key`
- `putInt(String key, int value)`: добавляет в настройки значение `int` с ключом `key`
- `putLong(String key, long value)`: добавляет в настройки значение типа `long` с ключом `key`
- `putString(String key, String value)`: добавляет в настройки строку с ключом `key`
- `putStringSet(String key, Set<String> values)`: добавляет в настройки строковый массив
- `commit()`: подтверждает все изменения в настройках
- `apply()`: также, как и метод `commit()`, подтверждает все изменения в настройках, однако измененный объект `SharedPreferences` вначале сохраняется во временной памяти, и лишь затем в результате асинхронной операции записывается на мобильное устройство

Рассмотрим пример сохранения и получения настроек в приложении. Определим в файле activity_main.xml следующий пользовательский интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText

        android:id="@+id/nameBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите имя"
        app:layout_constraintBottom_toTopOf="@id/saveButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button

        android:id="@+id/saveButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Сохранить"
        android:onClick="saveName"
        app:layout_constraintBottom_toTopOf="@id/nameView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/nameBox"/>
```

```

<TextView
    android:id="@+id/nameView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    app:layout_constraintBottom_toTopOf="@id/getButton"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/saveButton"/>
<Button
    android:id="@+id/getButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Получить имя"
    android:onClick="getName"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@id/nameView"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

На экране будут две кнопки - для сохранения и для вывода ранее сохраненного значения, а также поле для ввода и текстовое поле для вывода сохраненной настройки.

Определим методы обработчики кнопок в классе **MainActivity**:

```
package com.example.settingsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.SharedPreferences;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private static final String PREFS_FILE = "Account";
    private static final String PREF_NAME = "Name";
    SharedPreferences settings;

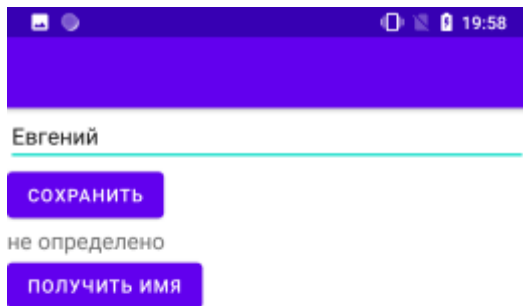
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);
    }

    public void saveName(View view) {
        // получаем введенное имя
        EditText nameBox = findViewById(R.id.nameBox);
        String name = nameBox.getText().toString();
        // сохраняем его в настройках
        SharedPreferences.Editor prefEditor = settings.edit();
        prefEditor.putString(PREF_NAME, name);
        prefEditor.apply();
    }

    public void getName(View view) {
        // получаем сохраненное имя
```

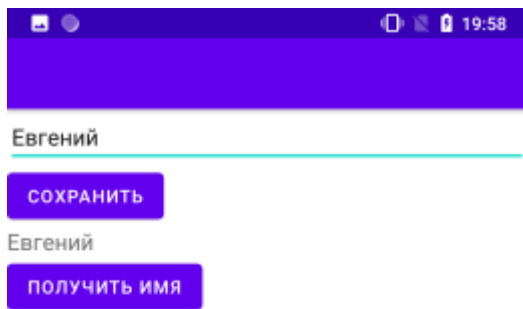
```
TextView nameView = findViewById(R.id.nameView);  
String name = settings.getString(PREF_NAME, "не определено");  
nameView.setText(name);  
}  
}
```

При отсутствии настроек при попытке их получить, приложение выведет значение по умолчанию:



Получение настроек SharedPreferences preferences в Android и Java

Теперь сохраним и выведем заново сохраненное значение:



Сохранение настроек SharedPreferences preferences в Android и Java

Нередко возникает задача автоматически сохранять вводимые данные при выходе пользователя из activity. Для этого мы можем переопределить метод onPause:

```
package com.example.settingsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.SharedPreferences;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.EditText;
```

```
public class MainActivity extends AppCompatActivity {
```



```
private static final String PREFS_FILE = "Account";
private static final String PREF_NAME = "Name";
EditText nameBox;
SharedPreferences settings;
SharedPreferences.Editor prefEditor;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    nameBox = findViewById(R.id.nameBox);
    settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);

    // получаем настройки
    String name = settings.getString(PREF_NAME, "");
    nameBox.setText(name);
}

@Override
protected void onPause(){
    super.onPause();

    String name = nameBox.getText().toString();
    // сохраняем в настройках
    prefEditor = settings.edit();
    prefEditor.putString(PREF_NAME, name);
    prefEditor.apply();
}
```

```
public void saveName(View view) {  
  
}  
  
public void getName(View view) {  
  
}  
}
```

Приватные настройки

Кроме общих настроек каждая activity может использовать приватные, к которым доступ из других activity будет невозможен. Для получения настроек уровня activity используется метод **getPreferences(MODE_PRIVATE)**:

```
import android.content.SharedPreferences;  
  
//.....  
  
SharedPreferences settings = getPreferences(MODE_PRIVATE);
```

То есть в отличие от общих настроек здесь не используется название группы настроек в качестве первого параметра, как в методе **getSharedPreferences()**. Однако вся остальная работа по добавлению, получению и изменению настроек будет аналогична работе с общими настройками.

PreferenceFragmentManager

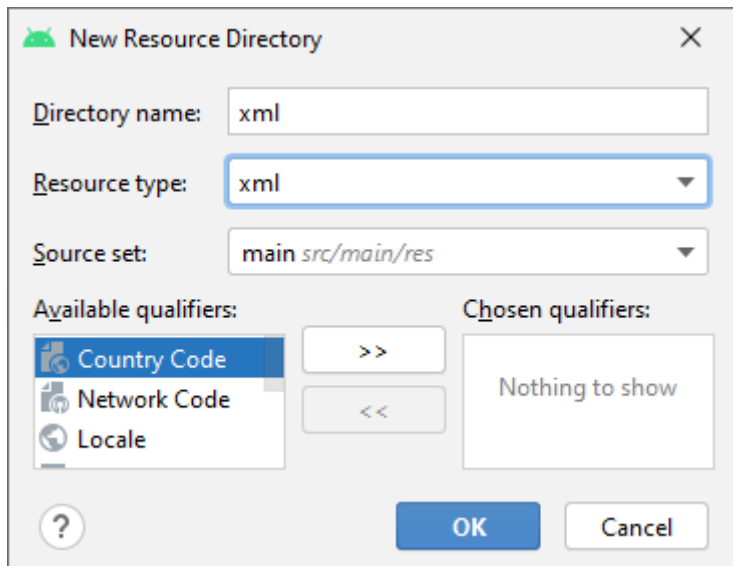
Для упрощения работы с группой настроек Android предоставляет специальный тип фрагмента - PreferenceFragmentManager. Рассмотрим, как ее использовать.

Создадим новый проект и вначале определим в файле build.gradle нужные зависимости для работы с PreferenceFragmentManager:

```
implementation "androidx.fragment:fragment:1.3.6"
```

```
implementation "androidx.preference:preference:1.1.1"
```

Для определения настроек добавим в папку res подпапку xml.



Настройки и PreferenceFragmentManager в Android и Java

Затем в папку **res/xml** добавим новый файл, который назовем **settings.xml**. И изменим его следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <EditTextPreference
```

```
        android:key="login"
```

```
        android:summary="Введите логин"
```

```
        android:title="Логин" />
```

```
<CheckBoxPreference  
    android:key="enabled"  
    android:summary="Отображать логин"  
    android:title="Отображать" />  
</PreferenceScreen>
```

Здесь в корневом элементе **PreferenceScreen** устанавливаются элементы **EditTextPreference** и **CheckBoxPreference**. Через каждый из этих элементов мы можем взаимодействовать с определенной настройкой.

Вобщем в данном случае мы можем использовать ряд различных типов настроек:

- **EditTextPreference**: используется элемент **EditText** для ввода текстового значения
- **CheckBoxPreference**: используется элемент **CheckBox** для установки логических значений **true** или **false**
- **SwitchPreference**: используется элемент **Switch** для установки логических значений **true** или **false** ("on" и "off")
- **RingtonePreference**: использует диалоговое окно для установки рингтона из списка рингтонов для установки логических значений **true** или **false**
- **ListPreference**: использует список для выбора одного из предопределенных значений
- **MultiSelectListPreference**: также использует список для выбора значений, но позволяет выбрать несколько элементов

Для каждого элемента настройки необходимо определить, как минимум, три атрибута:

- `android:key`: устанавливает ключ настройки в `SharedPreferences`
- `android:title`: название настройки для пользователя
- `android:summary`: краткое описание по данной настройке для пользователя

Далее добавим новый класс Java, который назовем `SettingsFragment`:

```
package com.example.settingsapp;
```

```
import android.os.Bundle;
```

```
import androidx.preference.PreferenceFragmentCompat;
```

```
public class SettingsFragment extends PreferenceFragmentCompat {
```

```
    @Override
```

```
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {
```

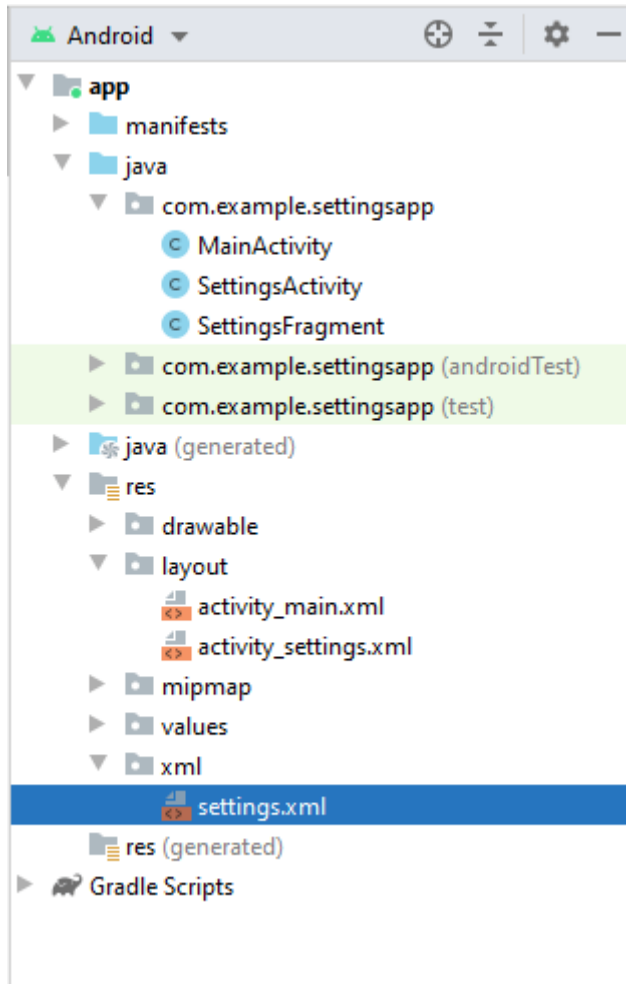
```
        addPreferencesFromResource(R.xml.settings);
```

```
    }
```

```
}
```

Фрагмент **`SettingsFragment`** наследуется от класса `PreferenceFragmentCompat`. В его методе `onCreatePreferences` вызывается метод **`addPreferencesFromResource()`**, в который передается id ресурса xml с настройками (в данном случае ранее определенный ресурс `R.xml.settings`).

И теперь добавим в проект специальную activity для установки настроек. Назовем ее **SettingsActivity**. В итоге проект будет выглядеть следующим образом:



Настройки и PreferenceFragmentCompat в Android

В файле layout для SettingsActivity - **activity_settings.xml** пропишем следующий интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.fragment.app.FragmentContainerView
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:id="@+id/settings_container"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent" />
```

Здесь определен `FragmentManager` с `id = settings_container` - именно тот элемент, в который будет загружаться фрагмент `SettingsFragment`.

В коде `SettingsActivity` определим загрузку фрагмента:

```
package com.example.settingsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
public class SettingsActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_settings);
```

```
        getSupportFragmentManager()
```

```
            .beginTransaction()
```

```
            .replace(R.id.settings_container, new SettingsFragment())
```

```
            .commit();
```

```
    }
```

```
}
```

`SettingsActivity` в качестве разметки интерфейса будет использовать ресурс `R.layout.activity_settings`.

При запуске `SettingsActivity` будет загружать фрагмент `SettingsFragment` в элемент с `id settings_container`.

Далее перейдем к главной activity - MainActivity. В файле **activity_main.xml** определим текстовое поле и кнопку:

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/settingsText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        app:layout_constraintBottom_toTopOf="@id/settingsButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/settingsButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Настройки"
        android:onClick="setPrefs"
        app:layout_constraintTop_toBottomOf="@id/settingsText"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```


И изменим класс **MainActivity**:

```
package com.example.settingsapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.preference.PreferenceManager;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    TextView settingsText;
    boolean enabled;
    String login;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        settingsText = findViewById(R.id.settingsText);
    }

    @Override
    public void onResume() {
```

```
super.onResume();

SharedPreferences prefs=
PreferenceManager.getDefaultSharedPreferences(this);

enabled = prefs.getBoolean("enabled", false);

login = prefs.getString("login", "не установлено");

settingsText.setText(login);

if(enabled)

    settingsText.setVisibility(View.VISIBLE);

else

    settingsText.setVisibility(View.INVISIBLE);

}

public void setPrefs(View view){

    Intent intent = new Intent(this, SettingsActivity.class);

    startActivity(intent);

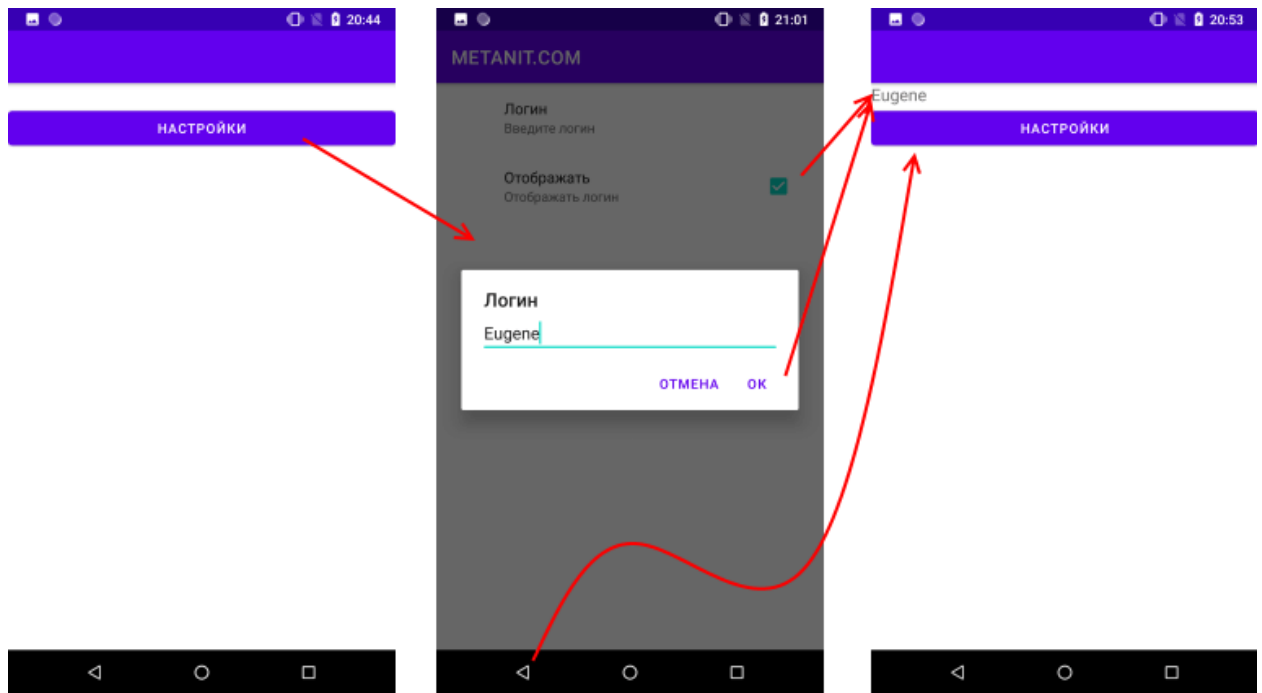
}

}
```

В методе onResume() получаем все настройки. Если настройка enabled равно true, то отображаем текстовое поле с логином.

В методе setPrefs(), который срабатывает при нажатии на кнопку, выполняется переход к SettingsActivity.

При первом запуске настроек не будет, и логин не будет отображаться. Перейдем на страницу настроек и установим там логин и включим его отображение, а затем вернемся на главную activity:



Установка настроек в PreferenceFragmentCompat в Android

При этом вручную нам ничего не надо сохранять, все настройки автоматически сохраняются функционалом PreferenceFragmentCompat.

Работа с файловой системой.

Чтение и сохранение файлов

Работа с настройками уровня activity и приложения позволяет сохранить небольшие данные отдельных типов (string, int), но для работы с большими массивами данных, такими как графически файлы, файлы мультимедиа и т.д., нам придется обращаться к файловой системе.

ОС Android построена на основе Linux. Этот факт находит свое отражение в работе с файлами. Так, в путях к файлам в качестве разграничителя в Linux использует слеш "/", а не обратный слеш "\" (как в Windows). А все названия файлов и каталогов являются регистрозависимыми, то есть "data" это не то же самое, что и "Data".

Приложение Android сохраняет свои данные в каталоге `/data/data/<название_пакета>/` и, как правило, относительно этого каталога будет идти работа.

Для работы с файлами абстрактный класс `android.content.Context` определяет ряд методов:

- **`boolean deleteFile (String name)`**: удаляет определенный файл
- **`String[] fileList ()`**: получает все файлы, которые содержатся в подкаталоге `/files` в каталоге приложения
- **`File getCacheDir()`**: получает ссылку на подкаталог `cache` в каталоге приложения
- **`File getDir(String dirName, int mode)`**: получает ссылку на подкаталог в каталоге приложения, если такого подкаталога нет, то он создается
- **`File getExternalCacheDir()`**: получает ссылку на папку `/cache` внешней файловой системы устройства
- **`File getExternalFilesDir(String type)`**: получает ссылку на каталог `/files` внешней файловой системы устройства
- **`File getFilePath(String filename)`**: возвращает абсолютный путь к файлу в файловой системе
- **`FileInputStream openFileInput(String filename)`**: открывает файл для чтения
- **`FileOutputStream openFileOutput (String name, int mode)`**: открывает файл для записи

Все файлы, которые создаются и редактируются в приложении, как правило, хранятся в подкаталоге /files в каталоге приложения.

Для непосредственного чтения и записи файлов применяются также стандартные классы Java из пакета java.io.

Итак, применим функционал чтения-записи файлов в приложении. Пусть у нас будет следующая примитивная разметка layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/editor"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:textSize="18sp"
        android:gravity="start"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toTopOf="@id/save_text"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/save_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:onClick="saveText"
        android:text="Сохранить"
```

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@id/text"
app:layout_constraintTop_toBottomOf="@id/editor" />
```

<TextView

```
android:id="@+id/text"
android:layout_width="0dp"
android:layout_height="0dp"
android:gravity="start"
android:textSize="18sp"
```

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@+id/open_text"
app:layout_constraintTop_toBottomOf="@+id/save_text" />
```

<Button

```
android:id="@+id/open_text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="openText"
android:text="Открыть"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toBottomOf="@+id/text" />
```

</androidx.constraintlayout.widget.ConstraintLayout>

Поле EditText предназначено для ввода текста, а TextView - для вывода ранее сохраненного текста. Для сохранения и восстановления текста добавлены две кнопки.

Теперь в коде Activity пропишем обработчики кнопок с сохранением и чтением файла:

```
package com.example.filesapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private final static String FILE_NAME = "content.txt";

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

    // сохранение файла
    public void saveText(View view){
```

```

FileOutputStream fos = null;
try {
    EditText textBox = findViewById(R.id.editor);
    String text = textBox.getText().toString();

    fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
    fos.write(text.getBytes());
    Toast.makeText(this, "Файл сохранен", Toast.LENGTH_SHORT).show();
}
catch(IOException ex) {

    Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
}
finally{
    try{
        if(fos!=null)
            fos.close();
    }
    catch(IOException ex){

        Toast.makeText(this, ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
}

// открытие файла
public void openText(View view){

```



```

FileInputStream fin = null;
TextView textView = findViewById(R.id.text);
try {
    fin = openFileInput(FILE_NAME);
    byte[] bytes = new byte[fin.available()];
    fin.read(bytes);
    String text = new String (bytes);
    textView.setText(text);
}
catch(IOException ex) {

    Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
}
finally{

    try{
        if(fin!=null)
            fin.close();
    }
    catch(IOException ex){

        Toast.makeText(this, ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
}
}
}

```

При нажатии на кнопку сохранения будет создаваться поток вывода:
`FileOutputStream fos = openFileOutput(FILE_NAME, MODE_PRIVATE)`

В данном случае введенный текст будет сохраняться в файл "content.txt". При этом будет использоваться режим `MODE_PRIVATE`

Система позволяет создавать файлы с двумя разными режимами:

- **MODE_PRIVATE:** файлы могут быть доступны только владельцу приложения (режим по умолчанию)
- **MODE_APPEND:** данные могут быть добавлены в конец файла

Поэтому в данном случае если файл "content.txt" уже существует, то он будет перезаписан. Если же нам надо было дописать файл, тогда надо было бы использовать режим `MODE_APPEND`:

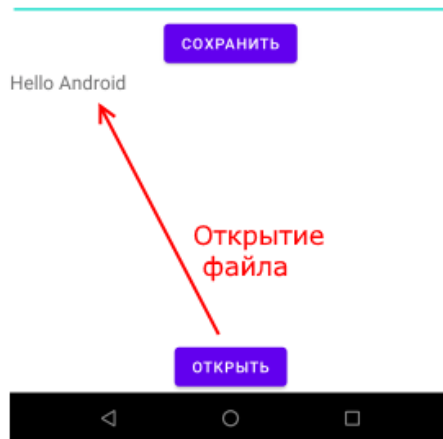
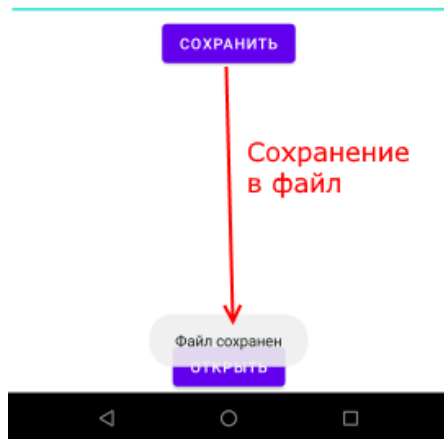
```
FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
```

Для чтения файла применяется поток ввода `FileInputStream`:

```
FileInputStream fin = openFileInput(FILE_NAME);
```

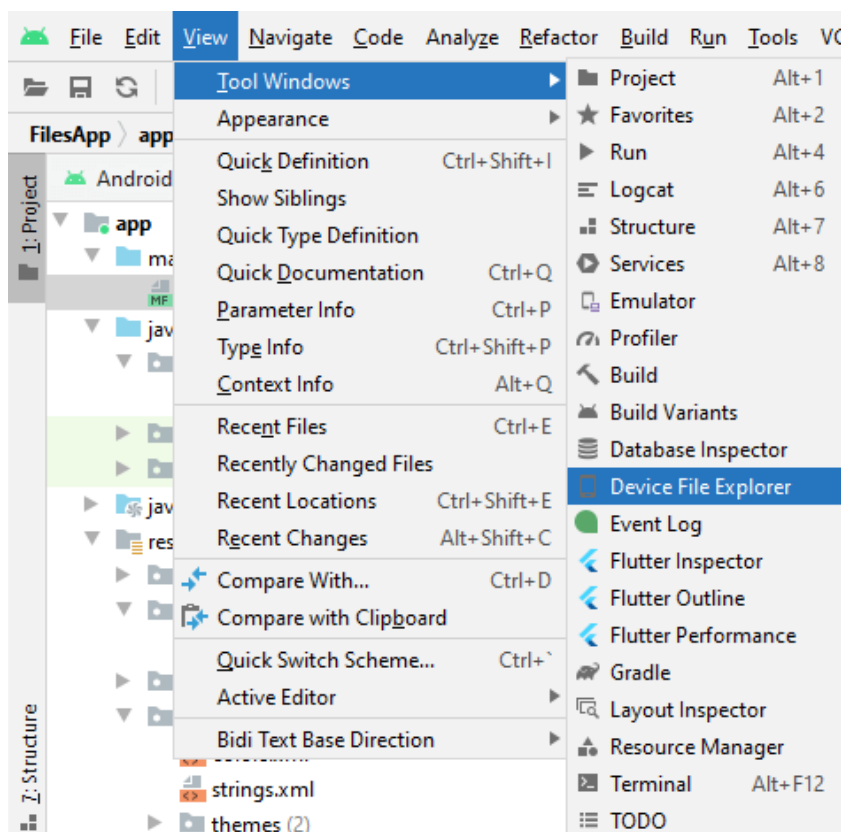
Подробнее про использование потоков ввода-вывода можно прочитать в руководстве по Java.

В итоге после нажатия кнопки сохранения весь текст будет сохранен в файле `/data/data/название_пакета/files/content.txt`



Сохранение и открытие файлов в Android и Java

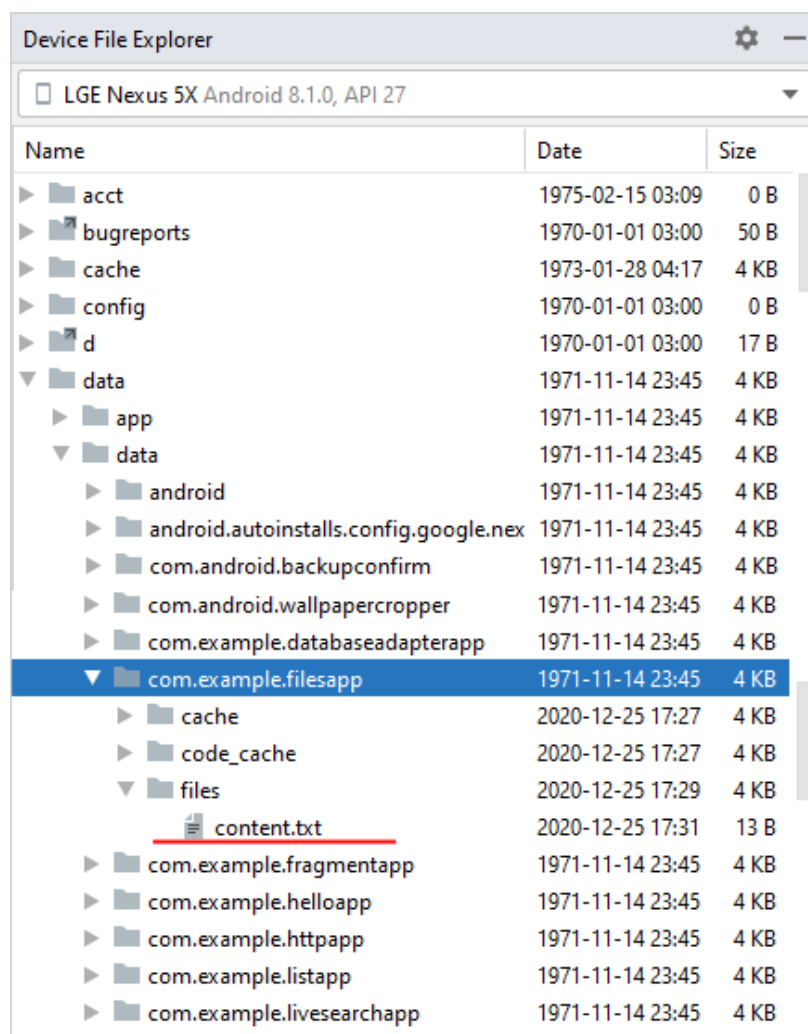
Где физически находится созданный файл? Чтобы увидеть его на подключенном устройстве перейдем в Android Stud в меню к пункту **View -> Tool Windows -> Device File Explorer**



Device File Explorer в Android Studio

После этого откроется окно Device File Explorer для просмотра файловой системы устройства.

И в папке **data/data/[название_пакета_приложения]/files** мы сможем найти сохраненный файл.



Размещение файлов во внешнем хранилище

В прошлом материале мы рассмотрели сохранение и чтение файлов из каталога приложения. По умолчанию такие файлы доступны только самому приложению. Однако мы можем помещать и работать с файлами из внешнего хранилища приложения. Это также позволит другим программам открывать данные файлы и при необходимости изменять.

Весь механизм работы с файлами будет таким же, как и при работе с хранилищем приложения. Ключевым отличием здесь будет получение и использование пути к внешнему хранилищу через метод **getExternalFilesDir()** класса Context.

Итак, пусть в файле activity_main.xml будет такая же разметка интерфейса:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/editor"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:textSize="18sp"
        android:gravity="start"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toTopOf="@id/save_text"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
```

```
android:id="@+id/save_text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

android:onClick="saveText"
android:text="Сохранить"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@id/text"
app:layout_constraintTop_toBottomOf="@id/editor" />
```

<TextView

```
android:id="@+id/text"
android:layout_width="0dp"
android:layout_height="0dp"
android:gravity="start"
android:textSize="18sp"

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toTopOf="@+id/open_text"
app:layout_constraintTop_toBottomOf="@+id/save_text" />
```

<Button

```
android:id="@+id/open_text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="openText"
android:text="Открыть"
app:layout_constraintLeft_toLeftOf="parent"
```

```
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toBottomOf="@+id/text" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

А код класса **MainActivity** будет выглядеть следующим образом:

```
package com.example.filesapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private final static String FILE_NAME = "document.txt";

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

```

private File getExternalPath() {
    return new File(getExternalFilesDir(null), FILE_NAME);
}

// сохранение файла
public void saveText(View view){

    try(FileOutputStream fos = new FileOutputStream(getExternalPath())) {
        EditText textBox = findViewById(R.id.editor);
        String text = textBox.getText().toString();
        fos.write(text.getBytes());
        Toast.makeText(this, "Файл сохранен", Toast.LENGTH_SHORT).show();
    }
    catch(IOException ex) {

        Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
}

// открытие файла
public void openText(View view){

    TextView textView = findViewById(R.id.text);
    File file = getExternalPath();
    // если файл не существует, выход из метода
    if(!file.exists()) return;
    try(FileInputStream fin = new FileInputStream(file)) {
        byte[] bytes = new byte[fin.available()];
        fin.read(bytes);
        String text = new String (bytes);
        textView.setText(text);
    }
}

```



```

    }

    catch(IOException ex) {

        Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();

    }

}

}

```

С помощью выражения `getExternalFilesDir(null)` получаем доступ к папке приложения во внешнем хранилище и устанавливаем объект файла:

```

private File getExternalPath() {

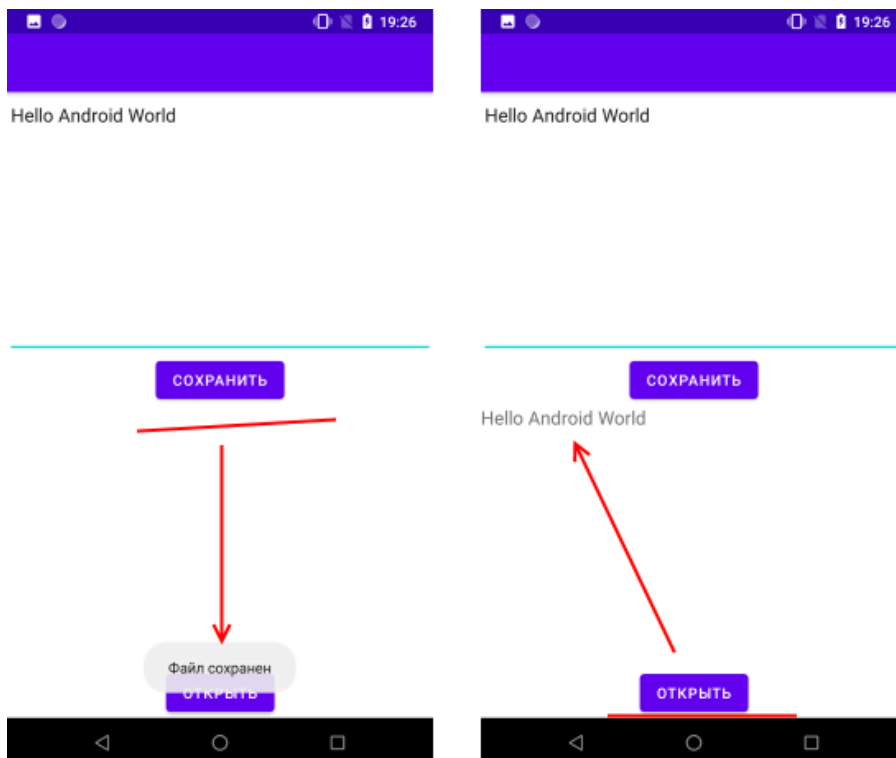
    return new File(getExternalFilesDir(null), FILE_NAME);

}

```

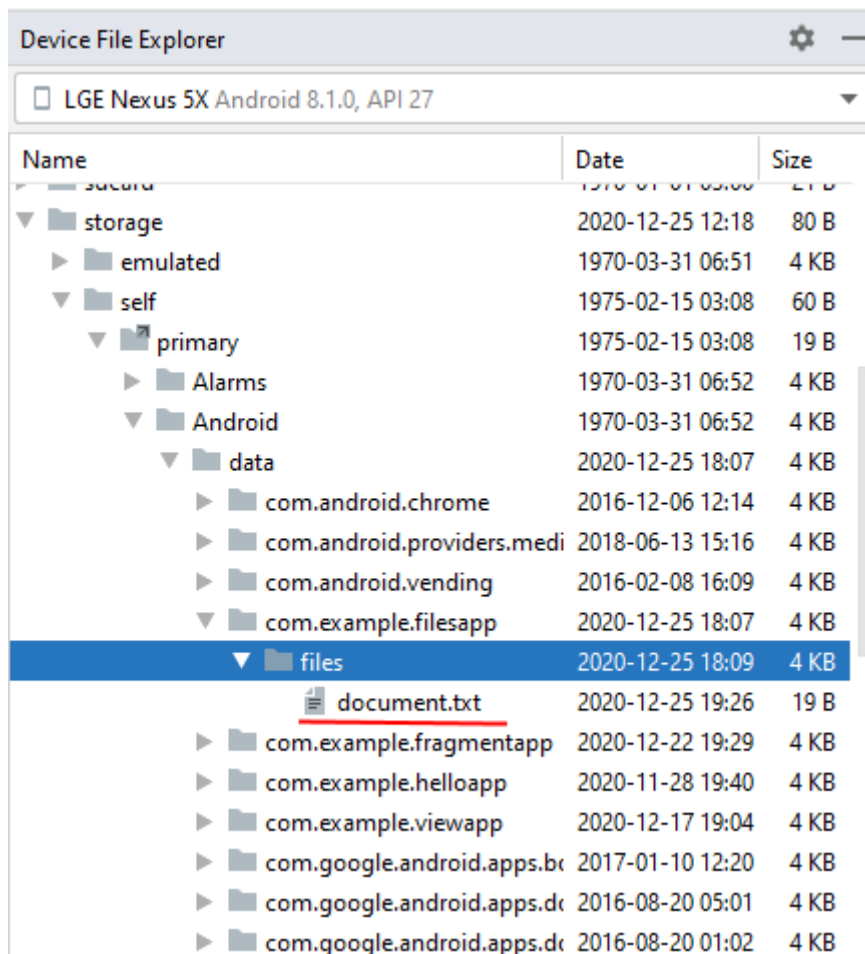
В качестве параметра передается тип папки, но в данном случае он нам не важен, поэтому передается значение `null`

Все остальные действия по записи/чтению файла будут такими же, как и в предыдущей теме в случае с работой с локальной папкой файла.



Запись файла во внешнем хранилище в Android и Java

И после операции записи на смартфоне через Device File Explorer мы сможем увидеть созданный файл в папке **storage/self/primary/Android/data/[название_пакета]/files:**



Name	Date	Size
storage	2020-12-25 12:18	80 B
emulated	1970-03-31 06:51	4 KB
self	1975-02-15 03:08	60 B
primary	1975-02-15 03:08	19 B
Alarms	1970-03-31 06:52	4 KB
Android	1970-03-31 06:52	4 KB
data	2020-12-25 18:07	4 KB
com.android.chrome	2016-12-06 12:14	4 KB
com.android.providers.medi	2018-06-13 15:16	4 KB
com.android.vending	2016-02-08 16:09	4 KB
com.example.filesapp	2020-12-25 18:07	4 KB
files	2020-12-25 18:09	4 KB
document.txt	2020-12-25 19:26	19 B
com.example.fragmentapp	2020-12-22 19:29	4 KB
com.example.helloapp	2020-11-28 19:40	4 KB
com.example.viewapp	2020-12-17 19:04	4 KB
com.google.android.apps.b	2017-01-10 12:20	4 KB
com.google.android.apps.d	2016-08-20 05:01	4 KB
com.google.android.apps.d	2016-08-20 01:02	4 KB

Файл во внешнем хранилище в Device File Explorer в Android Studio

Работа с базами данных SQLite

Подключение к базе данных SQLite

В Android имеется встроенная поддержка одной из распространенных систем управления базами данных - SQLite. Для этого в пакете `android.database.sqlite` определен набор классов, которые позволяют работать с базами данных SQLite. И каждое приложение может создать свою базу данных.

Чтобы использовать SQLite в Android, надо создать базу данных с помощью выражение на языке SQL. После этого база данных будет храниться в каталоге приложения по пути:

DATA/data/[Название_приложения]/databases/[Название_файла_базы_данных]
]

ОС Android по умолчанию уже содержит ряд встроенных баз SQLite, которые используются стандартными программами - для списка контактов, для хранения фотографий с камеры, музыкальных альбомов и т.д.

Основную функциональность по работе с базами данных предоставляет пакет **`android.database`**. Функциональность непосредственно для работы с SQLite находится в пакете **`android.database.sqlite`**.

База данных в SQLite представлена классом **`android.database.sqlite.SQLiteDatabase`**. Он позволяет выполнять запросы к бд, выполнять с ней различные манипуляции.

Класс **`android.database.sqlite.SQLiteCursor`** предоставляет запрос и позволяет возвращать набор строк, которые соответствуют этому запросу.

Класс **`android.database.sqlite.SQLiteQueryBuilder`** позволяет создавать SQL-запросы.

Сами sql-выражения представлены классом **`android.database.sqlite.SQLiteStatement`**, которые позволяют с помощью плейсхолдеров вставлять в выражения динамические данные.

Класс **android.database.sqlite.SQLiteOpenHelper** позволяет создать базу данных со всеми таблицами, если их еще не существует.

В SQLite применяется следующая система типов данных:

- **INTEGER**: представляет целое число, аналог типу `int` в java
- **REAL**: представляет число с плавающей точкой, аналог `float` и `double` в java
- **TEXT**: представляет набор символов, аналог `String` и `char` в java
- **BLOB**: представляет массив бинарных данных, например, изображение, аналог типу `int` в java

Сохраняемые данные должны представлять соответствующие типы в java.

Создание и открытие базы данных

Для создания или открытия новой базы данных из кода Activity в Android мы можем вызвать метод **openOrCreateDatabase()**. Этот метод может принимать три параметра:

- название для базы данных
- числовое значение, которое определяет режим работы (как правило, в виде константы `MODE_PRIVATE`)

- необязательный параметр в виде объекта **SQLiteDatabase.CursorFactory**, который представляет фабрику создания курсора для работы с бд

Например, создание базы данных app.db:

```
SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db",  
MODE_PRIVATE, null);
```

Для выполнения запроса к базе данных можно использовать метод `execSQL` класса **SQLiteDatabase**. В этот метод передается SQL-выражение. Например, создание в базе данных таблицы users:

```
SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db",  
MODE_PRIVATE, null);  
  
db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age  
INTEGER)");
```

Если нам надо не просто выполнить выражение, но и получить из бд какие-либо данные, то используется метод **rawQuery()**. Этот метод в качестве параметра принимает SQL-выражение, а также набор значений для выражения sql. Например, получение всех объектов из базы данных:

```
SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db",  
MODE_PRIVATE, null);  
  
db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age  
INTEGER)");  
  
Cursor query = db.rawQuery("SELECT * FROM users;", null);  
if(query.moveToFirst()){  
  
    String name = query.getString(0);  
    int age = query.getInt(1);  
}
```

Метод **db.rawQuery()** возвращает объект **Cursor**, с помощью которого мы можем извлечь полученные данные.

Возможна ситуация, когда в базе данных не будет объектов, и для этого методом `query.moveToFirst()` пытаемся переместиться к первому объекту, полученному из бд. Если этот метод возвратит значение `false`, значит запрос не получил никаких данных из бд.

Теперь для работы с базой данных сделаем простейшее приложение. Для этого создадим новый проект.

В файле **activity_main.xml** определим простейший графический интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp" >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click"
        android:onClick="onClick"
        app:layout_constraintBottom_toTopOf="@id/textView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

    <TextView
        android:id="@+id/textView"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="22sp"
app:layout_constraintTop_toBottomOf="@id/button"
app:layout_constraintLeft_toLeftOf="parent"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

А в классе MainActivity определим взаимодействие с базой данных:

```
package com.example.sqliteapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
    }
```

```
    public void onClick(View view){
```

```
        SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db",
        MODE_PRIVATE, null);
```

```
db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age  
INTEGER, UNIQUE(name));
```

```
db.execSQL("INSERT OR IGNORE INTO users VALUES ('Tom Smith',  
23), ('John Dow', 31);");
```

```
Cursor query = db.rawQuery("SELECT * FROM users;", null);
```

```
TextView textView = findViewById(R.id.textView);
```

```
textView.setText("");
```

```
while(query.moveToNext()){
```

```
    String name = query.getString(0);
```

```
    int age = query.getInt(1);
```

```
    textView.append("Name: " + name + " Age: " + age + "\n");
```

```
}
```

```
query.close();
```

```
db.close();
```

```
}
```

```
}
```

По нажатию на кнопку здесь вначале создается в базе данных `app.db` новая таблица `users`, а затем в нее добавляются два объекта в базу данных с помощью SQL-выражения **INSERT**.

Далее с помощью выражения **SELECT** получаем всех добавленных пользователей из базы данных в виде курсора **Cursor**.

Вызовом **query.moveToNext()** перемещаемся в цикле **while** последовательно по всем объектам.

Для получения данных из курсора применяются методы **query.getString(0)** и **query.getInt(1)**. В скобках в методы передается номер столбца, из которого мы получаем данные. Например, выше мы добавили вначале имя пользователя в виде строки, а затем возраст в виде числа. Значит, нулевым столбцом будет идти строковое значение, которое получаем с помощью метода **getString()**, а следующим - первым столбцом идет числовое значение, для которого применяется метод **getInt()**.

После завершения работы с курсором и базой данных мы закрываем все связанные объекты:

```
query.close();
```

```
db.close();
```

Если мы не закроем курсор, то можем столкнуться с проблемой утечки памяти.

И если мы обратимся к приложению, то после нажатия на кнопку в текстовое поле будут выведены добавленные данные:



Name: Tom Smith Age: 23
Name: John Dow Age: 31



Работа с классом SQLiteDatabase в Android и Java

SQLiteOpenHelper и SimpleCursorAdapter.

Получение данных из SQLite

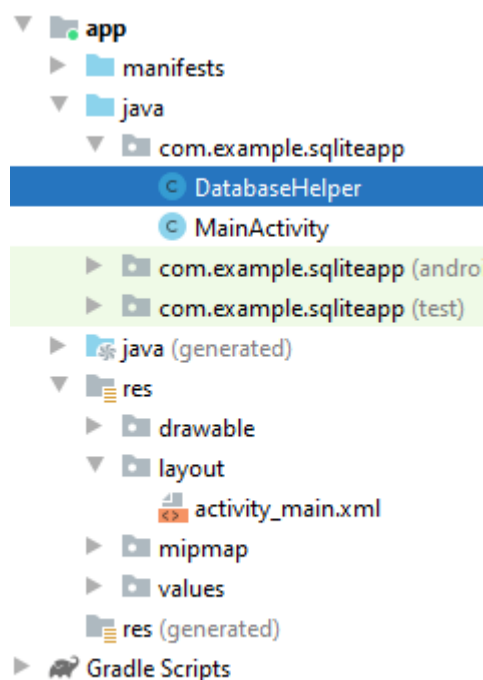
В прошлом материале было рассмотрено, как подключаться к базе данных SQLite и выполнять запросы. Теперь пойдем дальше и создадим полностью интерфейс для работы с базой данных.

Итак, создадим новый проект.

Для упрощения работы с базами данных SQLite в Android нередко применяется класс **SQLiteOpenHelper**. Для использования необходимо создать класса-наследник от SQLiteOpenHelper, переопределив как минимум два его метода:

- **onCreate():** вызывается при попытке доступа к базе данных, но когда еще эта база данных не создана
- **onUpgrade():** вызывается, когда необходимо обновление схемы базы данных. Здесь можно пересоздать ранее созданную базу данных в onCreate(), установив соответствующие правила преобразования от старой бд к новой

Поэтому добавим в проект, в ту же папку, где находится класс MainActivity, новый класс **DatabaseHelper**:



Добавление класса SQLiteOpenHelper в Android и Java

```

package com.example.sqliteapp;

import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "userstore.db"; // название бд
    private static final int SCHEMA = 1; // версия базы данных
    static final String TABLE = "users"; // название таблицы в бд
    // названия столбцов

    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_YEAR = "year";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, SCHEMA);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL("CREATE TABLE users (" + COLUMN_ID
            + " INTEGER PRIMARY KEY AUTOINCREMENT," +
            COLUMN_NAME
            + " TEXT, " + COLUMN_YEAR + " INTEGER);");

        // добавление начальных данных

        db.execSQL("INSERT INTO " + TABLE + " (" + COLUMN_NAME
            + ", " + COLUMN_YEAR + ") VALUES ('Том Смит', 1981);");
    }
}

```

```

    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS "+TABLE);
        onCreate(db);
    }
}

```

Если база данных отсутствует или ее версия (которая задается в переменной SCHEMA) выше текущей, то срабатывает метод onCreate().

Для выполнения запросов к базе данных нам потребуется объект **SQLiteDatabase**, который представляет базу данных. Метод onCreate() получает в качестве параметра базу данных приложения.

Для выполнения запросов к SQLite используется метод **execSQL()**. Он принимает sql-выражение CREATE TABLE, которое создает таблицу. Здесь также при необходимости мы можем выполнить и другие запросы, например, добавить какие-либо начальные данные. Так, в данном случае с помощью того же метода и выражения sql INSERT добавляется один объект в таблицу.

В методе onUpgrade() происходит обновление схемы БД. В данном случае для примера использован примитивный подход с удалением предыдущей базы данных с помощью sql-выражения DROP и последующим ее созданием. Но в реальности если вам будет необходимо сохранить данные, этот метод может включать более сложную логику - добавления новых столбцов, удаление ненужных, добавление дополнительных данных и т.д.

Далее определим в файле **activity_main.xml** следующую разметку:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

```
android:padding="16dp">
```

```
<TextView
```

```
    android:id="@+id/header"
```

```
    android:layout_width="0dp"
```

```
    android:layout_height="wrap_content"
```

```
    android:textSize="18sp"
```

```
    app:layout_constraintBottom_toTopOf="@+id/list"
```

```
    app:layout_constraintTop_toTopOf="parent"
```

```
    app:layout_constraintLeft_toLeftOf="parent"
```

```
    app:layout_constraintRight_toRightOf="parent"
```

```
<ListView
```

```
    android:id="@+id/list"
```

```
    android:layout_width="0dp"
```

```
    android:layout_height="0dp"
```

```
    app:layout_constraintTop_toBottomOf="@+id/header"
```

```
    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintLeft_toLeftOf="parent"
```

```
    app:layout_constraintRight_toRightOf="parent"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определен список `ListView`, для отображения полученных данных, с заголовком, который будет выводить число полученных объектов.

И изменим код класса **MainActivity** следующим образом:

```
package com.example.sqliteapp;
```

```

import androidx.appcompat.app.AppCompatActivity;
import android.widget.SimpleCursorAdapter;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    ListView userList;
    TextView header;
    DatabaseHelper databaseHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;

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

        header = findViewById(R.id.header);
        userList = findViewById(R.id.list);

        databaseHelper = new DatabaseHelper(getApplicationContext());
    }

    @Override
    public void onResume() {

```

```

super.onResume();

// открываем подключение
db = databaseHelper.getReadableDatabase();

//получаем данные из бд в виде курсора
userCursor = db.rawQuery("select * from "+ DatabaseHelper.TABLE, null);
// определяем, какие столбцы из курсора будут выводиться в ListView
String[] headers = new String[] { DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};

// создаем адаптер, передаем в него курсор
userAdapter = new SimpleCursorAdapter(this,
android.R.layout.two_line_list_item,
        userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2},
0);

header.setText("Найдено элементов: " + userCursor.getCount());
userList.setAdapter(userAdapter);
}

@Override
public void onDestroy(){
    super.onDestroy();
    // Закрываем подключение и курсор
    db.close();
    userCursor.close();
}
}

```

В методе onCreate() происходит создание объекта SQLiteOpenHelper. Сама инициализация объектов для работы с базой данных происходит в методе onResume(), который срабатывает после метода onCreate().

Чтобы получить объект базы данных, надо использовать методы:

- `getReadableDatabase()` (получение базы данных для чтения)
- `getWritableDatabase()` (получение базы данных для записи)

Так как в данном случае мы будем только считывать данные из бд, то воспользуемся первым методом:

```
db = sqlHelper.getReadableDatabase();
```

Получение данных и Cursor

Android предоставляет различные способы для осуществления запросов к объекту `SQLiteDatabase`. В большинстве случаев мы можем применять метод **`rawQuery()`**, который принимает два параметра: SQL-выражение `SELECT` и дополнительный параметр, задающий параметры запроса.

После выполнения запроса `rawQuery()` возвращает объект **Cursor**, который хранит результат выполнения SQL-запроса:

```
userCursor = db.rawQuery("select * from "+ DatabaseHelper.TABLE, null);
```

Класс `Cursor` предлагает ряд методов для управления выборкой, в частности:

- **`getCount()`**: получает количество извлеченных из базы данных объектов
- Методы **`moveToFirst()`** и **`moveToNext()`** позволяют переходить к первому и к следующему элементам выборки. Метод **`isAfterLast()`** позволяет проверить, достигнут ли конец выборки.
- Методы **`get*(columnIndex)`** (например, **`getLong()`**, **`getString()`**) позволяют по индексу столбца обратиться к данному столбцу текущей строки

CursorAdapter

Дополнительно для управления курсором в Android имеется класс `CursorAdapter`. Он позволяет адаптировать полученный с помощью курсора набор к отображению в списковых элементах наподобие `ListView`. Как правило, при работе с курсором используется подкласс `CursorAdapter` - **`SimpleCursorAdapter`**. Хотя можно использовать и другие адаптеры, типа `ArrayAdapter`.

```
userAdapter = new SimpleCursorAdapter(this,  
    android.R.layout.two_line_list_item,  
    userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);  
userList.setAdapter(userAdapter);
```

Конструктор класса `SimpleCursorAdapter` принимает шесть параметров:

1. Первым параметром выступает контекст, с которым ассоциируется адаптер, например, текущая `activity`
2. Второй параметр - ресурс разметки интерфейса, который будет использоваться для отображения результатов выборки
3. Третий параметр - курсор
4. Четвертый параметр - список столбцов из выборки, которые будут отображаться в разметке интерфейса
5. Пятый параметр - элементы внутри ресурса разметки, которые будут отображать значения столбцов из четвертого параметра
6. Шестой параметр - флаги, задающие поведения адаптера

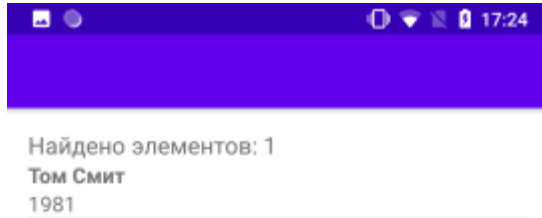
При использовании `CursorAdapter` и его подклассов следует учитывать, что выборка курсора должна включать целочисленный столбец с названием `_id`, который должен быть уникальным для каждого элемента выборки. Значение этого столбца при нажатии на элемент списка затем передается в метод обработки **`onListItemClick()`**, благодаря чему мы можем по `id` идентифицировать нажатый элемент.

В данном случае у нас первый столбец как раз называется `"_id"`.

После завершения работы курсор должен быть закрыт методом `close()`

И также надо учитывать, что если мы используем курсор в SimpleCursorAdapter, то мы не можем использовать метод close(), пока не завершим использование SimpleCursorAdapter. Поэтому метод cursor более предпочтительно вызывать в методе onDestroy() фрагмента или activity.

И если мы запустим приложение, то увидим список из одного добавленного элемента:

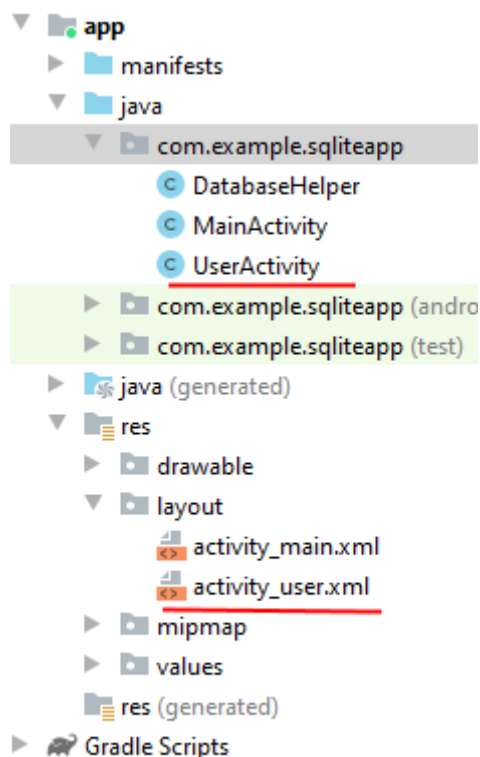


SimpleCursorAdapter и SQLite в Android и Java

Добавление, удаление и обновление данных в SQLite

Продолжим работу с проектом из прошлой темы, где мы получаем данные. Теперь добавим в него стандартную CRUD-логику (создание, обновление, удаление).

Чтобы не нагромождать форму с главной activity, все остальные действия по работе с данными будут происходить на другом экране. Добавим в проект новый класс activity, который назовем UserActivity:



Добавление activity для SQLite в Android и Java

В файле **activity_user.xml** определим универсальную форму для добавления/обновления/удаления данных:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent">
<EditText
    android:id="@+id/name"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="Введите имя"
    app:layout_constraintBottom_toTopOf="@+id/year"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />
<EditText
    android:id="@+id/year"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="Введите год рождения"
    app:layout_constraintTop_toBottomOf="@+id/name"
    app:layout_constraintBottom_toTopOf="@+id/saveButton"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />
<Button
    android:id="@+id/saveButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Сохранить"
    android:onClick="save"
    app:layout_constraintHorizontal_weight="1"
    app:layout_constraintTop_toBottomOf="@+id/year"
    app:layout_constraintLeft_toLeftOf="parent"
```

```

        app:layout_constraintRight_toLeftOf="@+id/deleteButton"
    />
<Button
    android:id="@+id/deleteButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Удалить"
    android:onClick="delete"
    app:layout_constraintHorizontal_weight="1"
    app:layout_constraintTop_toBottomOf="@+id/year"
    app:layout_constraintLeft_toRightOf="@+id/saveButton"
    app:layout_constraintRight_toRightOf="parent"
/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

И также изменим код **UserActivity**:

```

package com.example.sqliteapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

```

```
public class UserActivity extends AppCompatActivity {
```

```
    EditText nameBox;
```

```
    EditText yearBox;
```

```
    Button delButton;
```

```
    Button saveButton;
```

```
    DatabaseHelper sqlHelper;
```

```
    SQLiteDatabase db;
```

```
    Cursor userCursor;
```

```
    long userId=0;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_user);
```

```
        nameBox = findViewById(R.id.name);
```

```
        yearBox = findViewById(R.id.year);
```

```
        delButton = findViewById(R.id.deleteButton);
```

```
        saveButton = findViewById(R.id.saveButton);
```

```
        sqlHelper = new DatabaseHelper(this);
```

```
        db = sqlHelper.getWritableDatabase();
```

```
        Bundle extras = getIntent().getExtras();
```

```
        if (extras != null) {
```

```
            userId = extras.getLong("id");
```

```
        }
```

```
        // если 0, то добавление
```

```

        if (userId > 0) {
            // получаем элемент по id из бд
            userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE + "
where " +
                DatabaseHelper.COLUMN_ID + "=?", new
String[]{String.valueOf(userId)});
            userCursor.moveToFirst();
            nameBox.setText(userCursor.getString(1));
            yearBox.setText(String.valueOf(userCursor.getInt(2)));
            userCursor.close();
        } else {
            // скрываем кнопку удаления
            delButton.setVisibility(View.GONE);
        }
    }
}

```

```

public void save(View view){
    ContentValues cv = new ContentValues();
    cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());
    cv.put(DatabaseHelper.COLUMN_YEAR,
Integer.parseInt(yearBox.getText().toString()));

    if (userId > 0) {
        db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID +
"=" + userId, null);
    } else {
        db.insert(DatabaseHelper.TABLE, null, cv);
    }
    goHome();
}

```

```

public void delete(View view){
    db.delete(DatabaseHelper.TABLE, "_id = ?", new
String[]{String.valueOf(userId)});
    goHome();
}
private void goHome(){
    // закрываем подключение
    db.close();
    // переход к главной activity
    Intent intent = new Intent(this, MainActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
    startActivity(intent);
}
}

```

При обновлении или удалении объекта из списка из главной activity в UserActivity будет передаваться id объекта:

```

long userId=0;
//.....
Bundle extras = getIntent().getExtras();
if (extras != null) {
    userId = extras.getLong("id");
}

```

Если из MainActivity не было передано id, то устанавливаем его значение 0, следовательно, у нас будет добавление, а не редактирование/удаление

Если id определен, то получаем по нему из базы данных объект для редактирования/удаления:


```

if (id < 0) {

    userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE + " where
" +
        DatabaseHelper.COLUMN_ID + "=?", new String[]{String.valueOf(id)});
    userCursor.moveToFirst();
    nameBox.setText(userCursor.getString(1));
    yearBox.setText(String.valueOf(userCursor.getInt(2)));
    userCursor.close();
}

```

Иначе просто скрываем кнопку удаления.

Для выполнения операций по вставке, обновлению и удалению данных SQLiteDatabase имеет методы insert(), update() и delete(). Эти методы вызываются в обработчиках кнопок:

```
db.delete(DatabaseHelper.TABLE, "_id = ?", new String[]{String.valueOf(id)});
```

В метод **delete()** передается название таблицы, а также столбец, по которому происходит удаление, и его значение. В качестве критерия можно выбрать несколько столбцов, поэтому третьим параметром идет массив. Знак вопроса ? обозначает параметр, вместо которого подставляется значение из третьего параметра.

ContentValues

Для добавления или обновления нам надо создать объект ContentValues. Данный объект представляет словарь, который содержит набор пар "ключ-значение". Для добавления в этот словарь нового объекта применяется метод put. Первый параметр метода - это ключ, а второй - значение, например:

```

ContentValues cv = new ContentValues();
cv.put("NAME", "Tom");
cv.put("YEAR", 30);

```

В качестве значений в метод `put` можно передавать строки, целые числа, числа с плавающей точкой

В данном же случае добавляются введенные в текстовое поля значения:

```
ContentValues cv = new ContentValues();  
cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());  
cv.put(DatabaseHelper.COLUMN_YEAR,  
Integer.parseInt(yearBox.getText().toString()));
```

При обновлении в метод **`update()`** передается название таблицы, объект `ContentValues` и критерий, по которому происходит обновление (в данном случае столбец `id`):

```
db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID + "=" +  
userId, null);
```

Метод **`insert()`** принимает название таблицы, объект `ContentValues` с добавляемыми значениями. Второй параметр является необязательным: он передает столбец, в который надо добавить значение `NULL`:

```
db.insert(DatabaseHelper.TABLE, null, cv);
```

Вместо этих методов, как в прошлом материале, можно использовать метод `execSQL()` с точным указанием выполняемого `sql`-выражения. В то же время методы `delete/insert/update` имеют преимущество - они возвращают `id` измененной записи, по которому мы можем узнать об успешности операции, или `-1` в случае неудачной операции:

```
long result = db.insert(DatabaseHelper.TABLE, null, cv);  
if(result>0){  
  
    // действия  
}
```

После каждой операции выполняется метод `goHome()`, который возвращает на главную `activity`.

После этого нам надо исправить код MainActivity, чтобы она инициировала выполнение кода в UserActivity. Для этого изменим код **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/addButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:text="Добавить"
        android:onClick="add"
        app:layout_constraintBottom_toTopOf="@+id/list"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
    />

    <ListView
        android:id="@+id/list"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toBottomOf="@+id/addButton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
app:layout_constraintRight_toRightOf="parent"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

В данном случае была добавлена кнопка для вызова UserActivity.

И также изменим код класса MainActivity:

```
package com.example.sqliteapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.Intent;
```

```
import android.view.View;
```

```
import android.widget.AdapterView;
```

```
import android.widget.SimpleCursorAdapter;
```

```
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.os.Bundle;
```

```
import android.widget.ListView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    ListView userList;
```

```
    DatabaseHelper databaseHelper;
```

```
    SQLiteDatabase db;
```

```
    Cursor userCursor;
```

```
    SimpleCursorAdapter userAdapter;
```

```
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    userList = findViewById(R.id.list);
    userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
            Intent intent = new Intent(getApplicationContext(), UserActivity.class);
            intent.putExtra("id", id);
            startActivity(intent);
        }
    });

    databaseHelper = new DatabaseHelper(getApplicationContext());
}

@Override
public void onResume() {
    super.onResume();
    // открываем подключение
    db = databaseHelper.getReadableDatabase();

    //получаем данные из бд в виде курсора
    userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE, null);
    // определяем, какие столбцы из курсора будут выводиться в ListView
    String[] headers = new String[]{DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};

```

```

        // создаем адаптер, передаем в него курсор
        userAdapter = new SimpleCursorAdapter(this,
        android.R.layout.two_line_list_item,
            userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2},
        0);
        userList.setAdapter(userAdapter);
    }

```

// по нажатию на кнопку запускаем UserActivity для добавления данных

```

public void add(View view) {
    Intent intent = new Intent(this, UserActivity.class);
    startActivity(intent);
}

```

@Override

```

public void onDestroy() {
    super.onDestroy();
    // Закрываем подключение и курсор
    db.close();
    userCursor.close();
}
}

```

При нажатии на кнопку запускается UserActivity, при этом не передается никакого id, то есть в UserActivity id будет равен нулю, значит будет идти добавление данных:

```

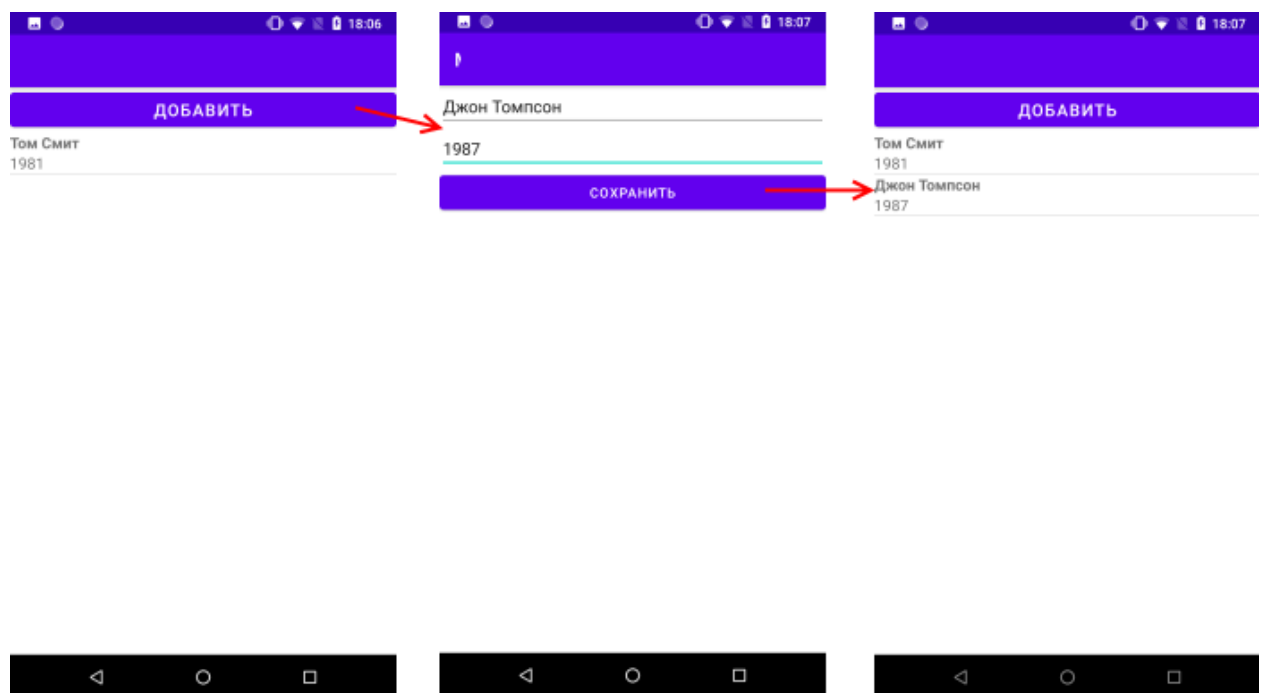
public void add(View view){
    Intent intent = new Intent(this, UserActivity.class);
    startActivity(intent);
}

```

Другую ситуацию представляет обработчик нажатия на элемент списка - при нажатии также будет запускаться `UserActivity`, но теперь будет передаваться `id` выбранной записи:

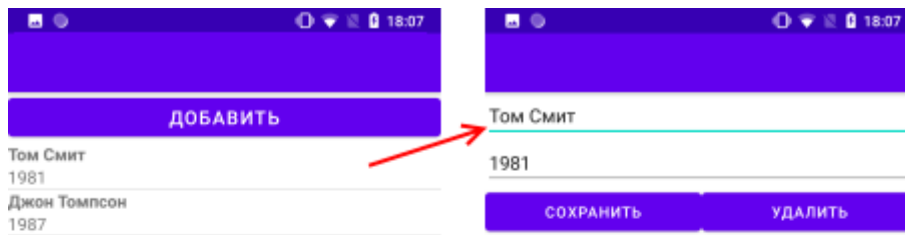
```
public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
    Intent intent = new Intent(getApplicationContext(), UserActivity.class);
    intent.putExtra("id", id);
    startActivity(intent);
}
```

Запустим приложение и нажмем на кнопку, которая должен перенаправлять на `UserActivity`:



Добавление в SQLite в Android и Java

При нажатии в `MainActivity` на элемент списка этот элемент попадет на `UserActivity`, где его можно будет удалить или отредактировать:

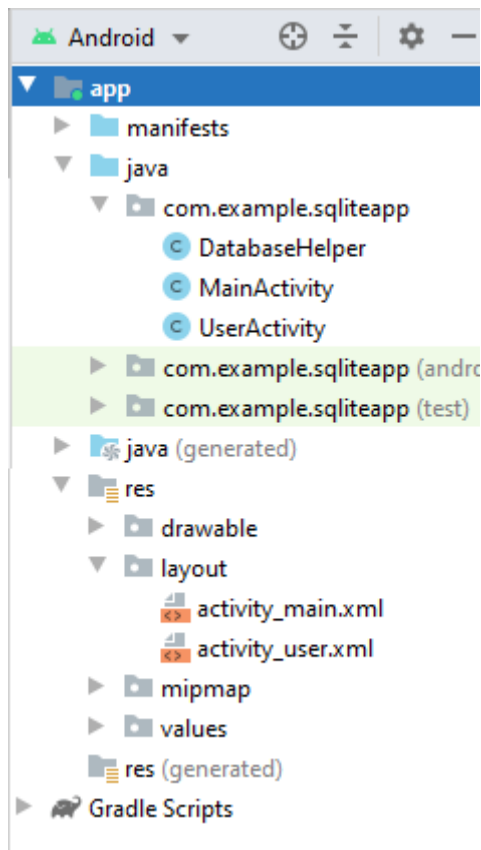


Редактирование в SQLite в Android и Java

Использование существующей БД SQLite

Кроме создания новой базы данных мы также можем использовать уже существующую. Это может быть более предпочтительно, так как в этом случае база данных приложения уже будет содержать всю необходимую информацию.

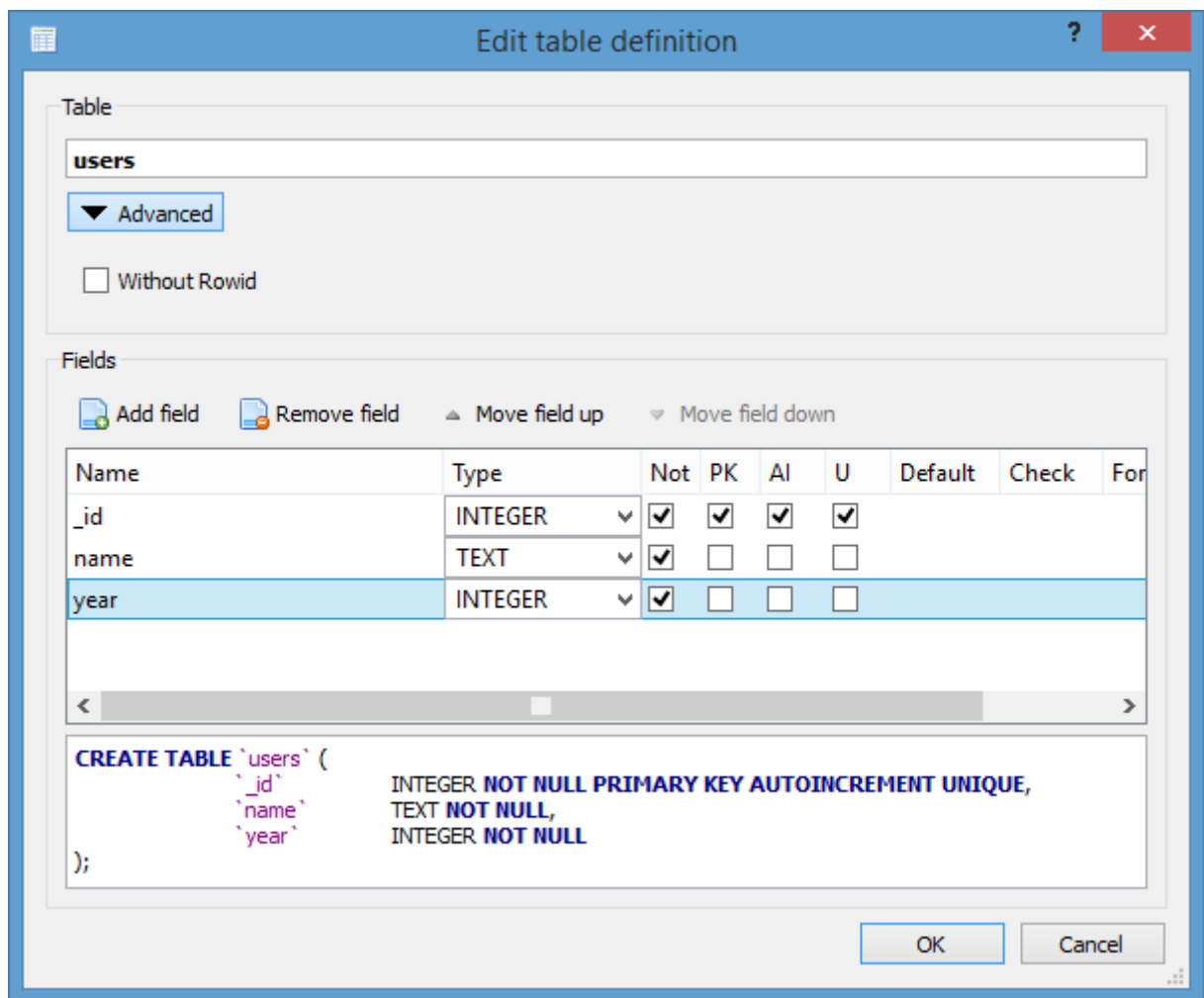
Возьмем проект, созданный в предыдущем материале, где у нас была MainActivity, которая выводила список объектов, и UserActivity, которая позволяла добавлять, редактировать и удалять объекты из БД



existing database SQLite in Android and Java

Для начала создадим базу данных SQLite. В этом нам может помочь такой инструмент как Sqlitebrowser. Он бесплатный и доступен для различных операционных систем по адресу <https://sqlitebrowser.org/>. Хотя можно использовать и другие способы для создания начальной БД.

Sqlitebrowser представляет графический интерфейс для создания базы данных и определения в ней всех необходимых таблиц:

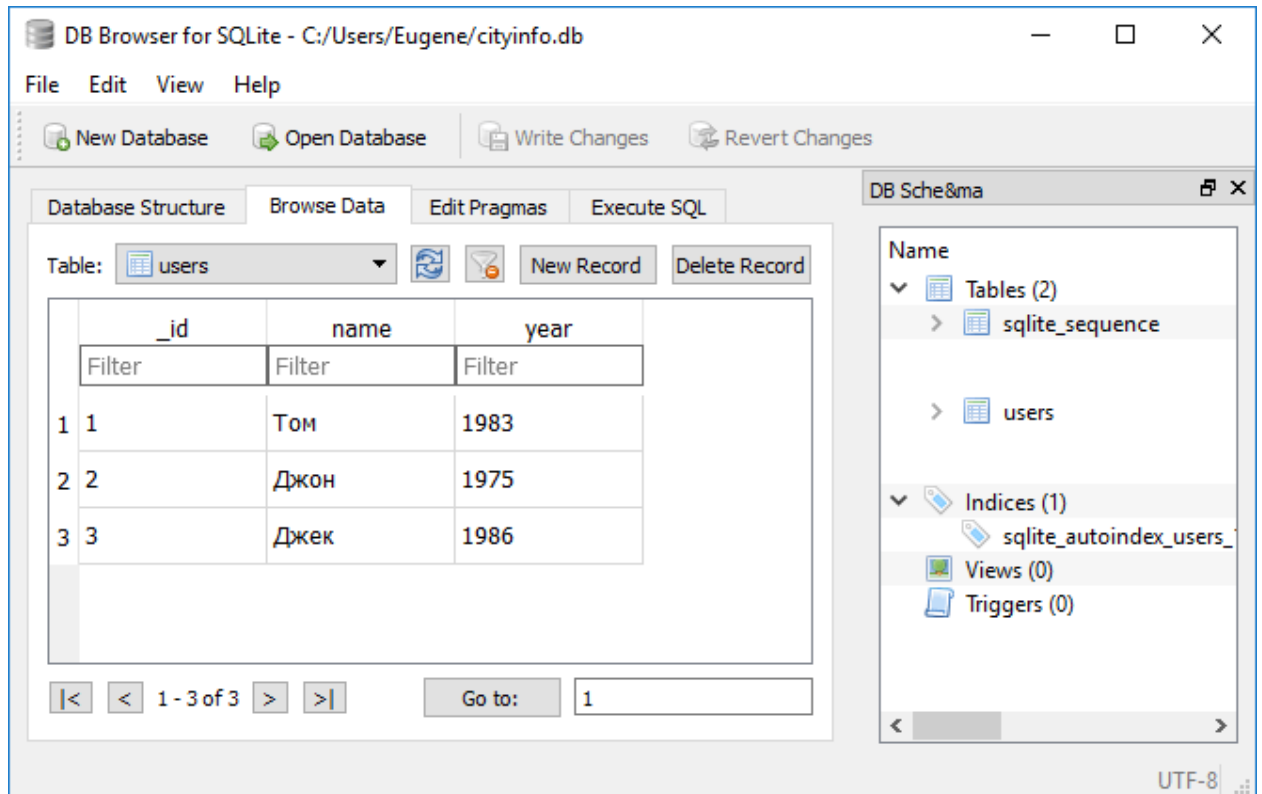


Существующая база данных SQLite в Android

Как видно на скриншоте, я определяю таблицу users с тремя полями: _id, name, age. Общая команда на создание таблицы будет следующей:

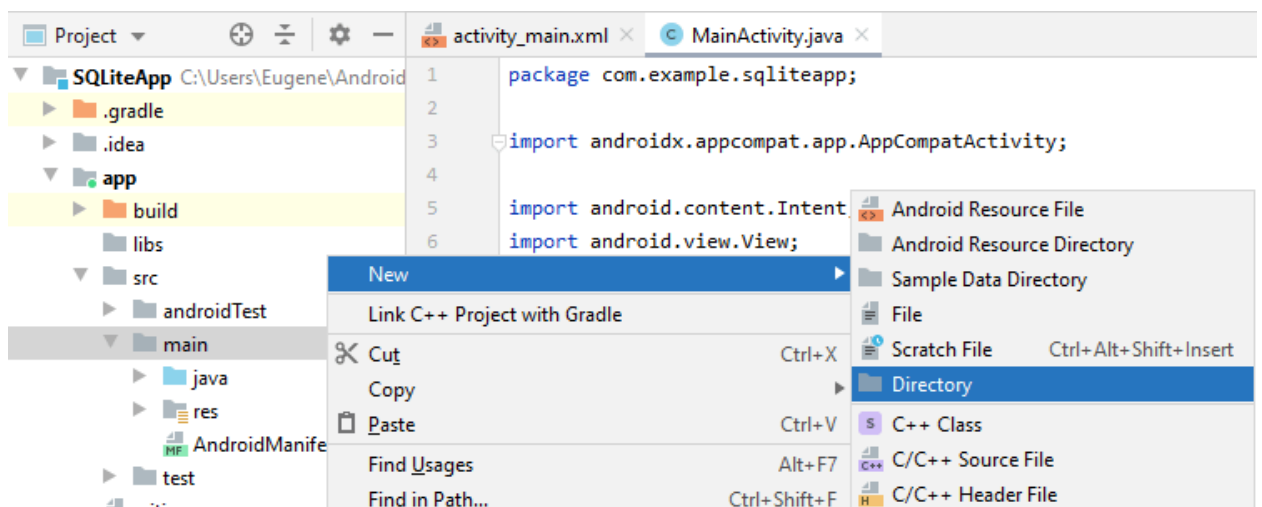
```
CREATE TABLE `users` (
  `_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
  UNIQUE,
  `name` TEXT NOT NULL,
  `year` INTEGER NOT NULL
);
```

Там же в программе добавим несколько элементов в созданную таблицу:



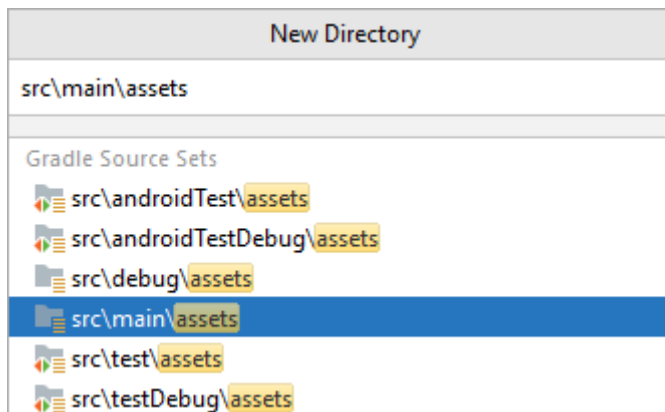
Existing SQLite database in Android

После создания таблицы добавим в проект в Android Studio папку **assets**, а в папку assets - только что созданную базу данных. Для этого перейдем к полному определению проекта, нажмем на папку main правой кнопкой мыши и в меню выберем **New -> Directory**:



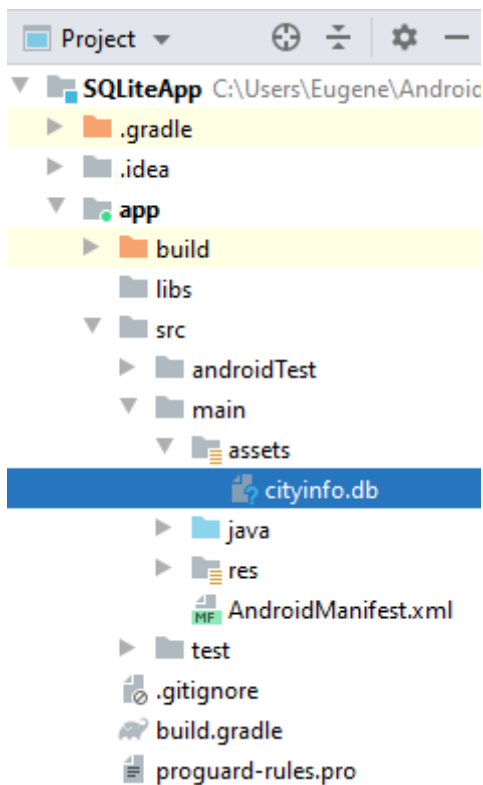
Добавление папки assets в Android Studio

Затем в появившемся окошке выберем пункт **src\main\assets** и нажмем на Enter для ее добавления в проект:



Добавление папки ресурсов assets в Android Studio

И затем скопируем в нее нашу базу данных:



База данных в Android Studio

В моем случае база данных называется "cityinfo.db".

Изменим код DatabaseHelper следующим образом:

```
package com.example.sqliteapp;
```

```

import android.database.SQLException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;
import android.util.Log;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

class DatabaseHelper extends SQLiteOpenHelper {
    private static String DB_PATH; // полный путь к базе данных
    private static String DB_NAME = "cityinfo.db";
    private static final int SCHEMA = 1; // версия базы данных
    static final String TABLE = "users"; // название таблицы в бд
    // названия столбцов
    static final String COLUMN_ID = "_id";
    static final String COLUMN_NAME = "name";
    static final String COLUMN_YEAR = "year";
    private Context myContext;

    DatabaseHelper(Context context) {
        super(context, DB_NAME, null, SCHEMA);
        this.myContext=context;
        DB_PATH =context.getFilesDir().getPath() + DB_NAME;
    }

```

```

@Override

public void onCreate(SQLiteDatabase db) { }

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { }


void create_db(){

    File file = new File(DB_PATH);
    if (!file.exists()) {
        //получаем локальную бд как поток
        try(InputStream myInput = myContext.getAssets().open(DB_NAME);
            // Открываем пустую бд
            OutputStream myOutput = new FileOutputStream(DB_PATH)) {

            // побайтово копируем данные
            byte[] buffer = new byte[1024];
            int length;
            while ((length = myInput.read(buffer)) > 0) {
                myOutput.write(buffer, 0, length);
            }
            myOutput.flush();
        }
        catch(IOException ex){
            Log.d("DatabaseHelper", ex.getMessage());
        }
    }
}

public SQLiteDatabase open()throws SQLException {

```

```

        return SQLiteDatabase.openDatabase(DB_PATH, null,
        SQLiteDatabase.OPEN_READWRITE);
    }
}

```

По умолчанию база данных будет размещаться во внешнем хранилище, выделяемом для приложения в папке **/data/data/[название_пакета]/databases/**, и чтобы получить полный путь к базе данных в конструкторе используется выражение:

```
DB_PATH =context.getFilesDir().getPath() + DB_NAME;
```

Метод `onCreate()` нам не нужен, так как нам не требуется создание встроенной базы данных. Зато здесь определен дополнительный метод `create_db()`, цель которого копирование базы данных из папки `assets` в то место, которое указано в переменной `DB_PATH`.

Кроме этого здесь также определен метод открытия базы данных `open()` с помощью метода `SQLiteDatabase.openDatabase()`

Новый способ организации подключения изменит использование `DatabaseHelper` в `activity`. Так, обновим класс **MainActivity**:

```

package com.example.sqliteapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.view.View;
import android.widget.AdapterView;
import android.widget.SimpleCursorAdapter;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

```

```
import android.os.Bundle;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    ListView userList;
    DatabaseHelper databaseHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;

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

        userList = findViewById(R.id.list);
        userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
                Intent intent = new Intent(getApplicationContext(), UserActivity.class);
                intent.putExtra("id", id);
                startActivity(intent);
            }
        });

        databaseHelper = new DatabaseHelper(getApplicationContext());
        // создаем базу данных
```



```

        databaseHelper.create_db();
    }

    @Override
    public void onResume() {
        super.onResume();
        // открываем подключение
        db = databaseHelper.open();
        //получаем данные из бд в виде курсора
        userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE, null);
        // определяем, какие столбцы из курсора будут выводиться в ListView
        String[] headers = new String[]{DatabaseHelper.COLUMN_NAME,
        DatabaseHelper.COLUMN_YEAR};

        // создаем адаптер, передаем в него курсор
        userAdapter = new SimpleCursorAdapter(this,
        android.R.layout.two_line_list_item,
            userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2},
0);

        userList.setAdapter(userAdapter);
    }

    // по нажатию на кнопку запускаем UserActivity для добавления данных
    public void add(View view) {
        Intent intent = new Intent(this, UserActivity.class);
        startActivity(intent);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }

```

```
        // Закрываем подключение и курсор
        db.close();
        userCursor.close();
    }
}
```

И также изменим класс **UserActivity**:

```
package com.example.sqliteapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class UserActivity extends AppCompatActivity {

    EditText nameBox;
    EditText yearBox;
    Button delButton;
    Button saveButton;

    DatabaseHelper sqlHelper;
    SQLiteDatabase db;
```

```

Cursor userCursor;

long userId=0;

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_user);


    nameBox = findViewById(R.id.name);
    yearBox = findViewById(R.id.year);
    delButton = findViewById(R.id.deleteButton);
    saveButton = findViewById(R.id.saveButton);


    sqlHelper = new DatabaseHelper(this);
    db = sqlHelper.open();


    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        userId = extras.getLong("id");
    }

    // если 0, то добавление
    if (userId > 0) {

        // получаем элемент по id из бд

        userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE + "
where " +

            DatabaseHelper.COLUMN_ID + "=?", new
String[]{String.valueOf(userId)});

        userCursor.moveToFirst();

        nameBox.setText(userCursor.getString(1));
        yearBox.setText(String.valueOf(userCursor.getInt(2)));
    }
}

```

```

        userCursor.close();
    } else {
        // скрываем кнопку удаления
        delButton.setVisibility(View.GONE);
    }
}

public void save(View view){
    ContentValues cv = new ContentValues();
    cv.put(DatabaseHelper.COLUMN_NAME, nameBox.getText().toString());
    cv.put(DatabaseHelper.COLUMN_YEAR,
Integer.parseInt(yearBox.getText().toString()));

    if (userId > 0) {
        db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN_ID +
"=" + userId, null);
    } else {
        db.insert(DatabaseHelper.TABLE, null, cv);
    }
    goHome();
}

public void delete(View view){
    db.delete(DatabaseHelper.TABLE, "_id = ?", new
String[]{String.valueOf(userId)});
    goHome();
}

private void goHome(){
    // закрываем подключение
    db.close();

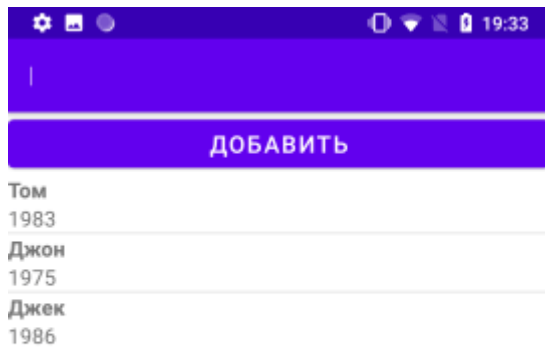
    // переход к главной activity
    Intent intent = new Intent(this, MainActivity.class);

```

```
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);

        startActivity(intent);
    }
}
```

Вся остальная работа с данными будет той же, чтобы и в прошлых темах:



Подключаем свою базу данных SQLite в Android

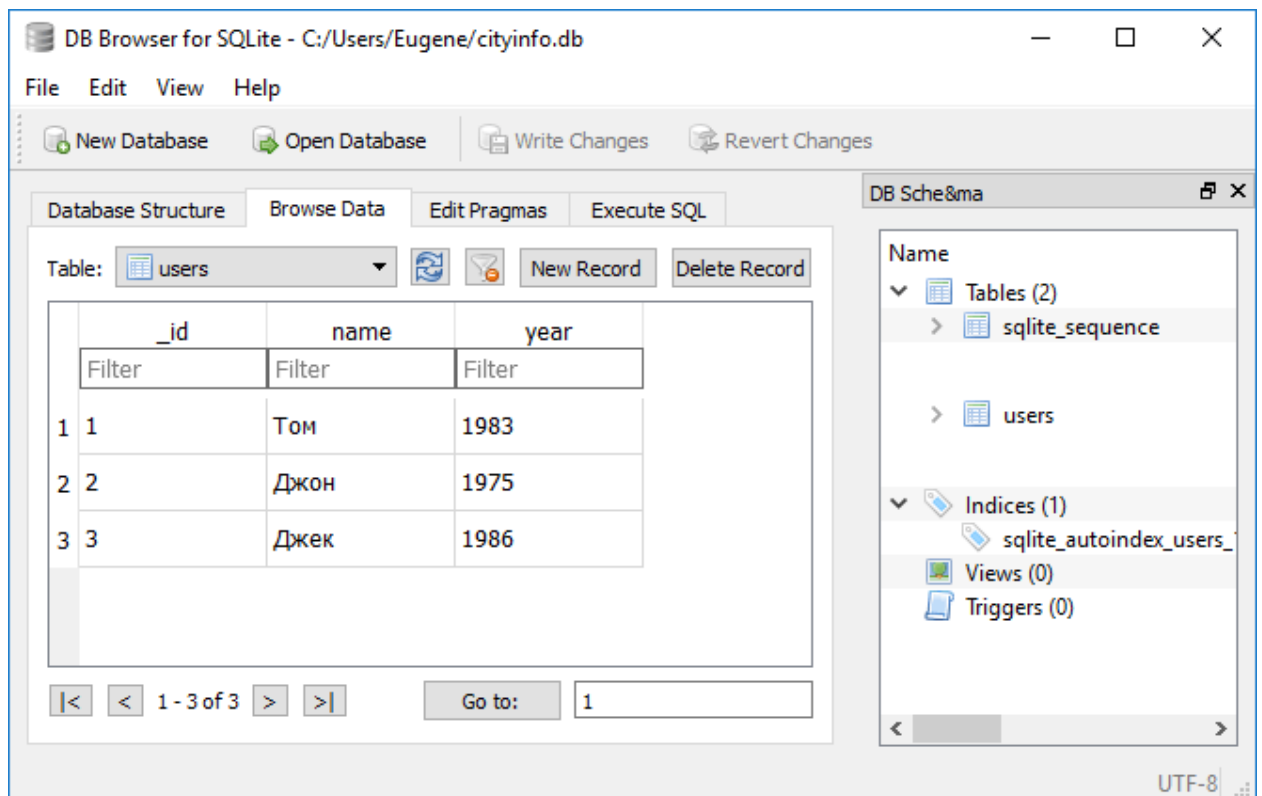
Динамический поиск по базе данных SQLite

Рассмотрим, как мы можем создать в приложении на Android динамический поиск по базе данных SQLite.

Итак, создадим новый проект с пустой MainActivity. Для этого проекта возьмем базу данных из прошлой темы (или создадим новую). Данная база

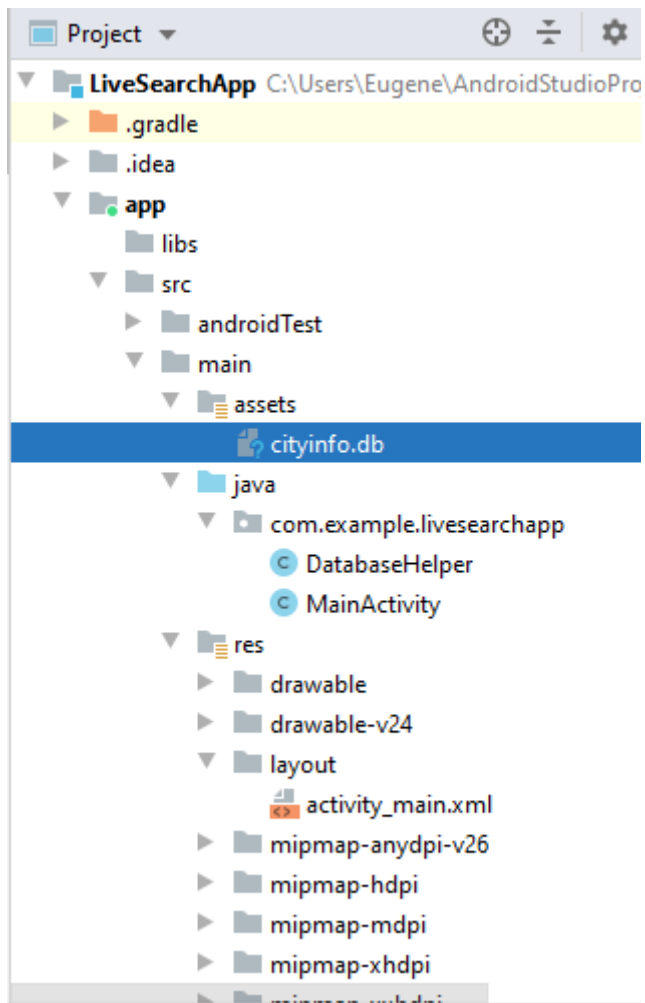
данных называется cityinfo и имеет одну таблицу users с тремя полями _id, name, age:

```
CREATE TABLE `users` (  
    `_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT  
    UNIQUE,  
    `name` TEXT NOT NULL,  
    `year` INTEGER NOT NULL  
);
```



Existing SQLite database in Android

И также добавим в проект в Android Studio папку assets, а в папку assets - только что созданную базу данных:



База данных SQLite в Android Studio

В моем случае база данных называется "cityinfo.db".

Как показано выше на скриншоте, добавив в проект в одну папку с MainActivity новый класс **DatabaseHelper**:

```
package com.example.livesearchapp;
```

```
import android.database.SQLException;
```

```
import android.database.sqlite.SQLiteOpenHelper;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.content.Context;
```

```
import android.util.Log;
```

```
import java.io.File;
```

```

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

class DatabaseHelper extends SQLiteOpenHelper {

    private static String DB_PATH; // полный путь к базе данных
    private static String DB_NAME = "cityinfo.db";
    private static final int SCHEMA = 1; // версия базы данных
    static final String TABLE = "users"; // название таблицы в бд
    // названия столбцов
    static final String COLUMN_ID = "_id";
    static final String COLUMN_NAME = "name";
    static final String COLUMN_YEAR = "year";
    private Context myContext;

    DatabaseHelper(Context context) {
        super(context, DB_NAME, null, SCHEMA);
        this.myContext=context;
        DB_PATH =context.getFilesDir().getPath() + DB_NAME;
    }

    @Override
    public void onCreate(SQLiteDatabase db) { }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { }

    void create_db(){

```



```

File file = new File(DB_PATH);
if (!file.exists()) {
    //получаем локальную бд как поток
    try(InputStream myInput = myContext.getAssets().open(DB_NAME);
        // Открываем пустую бд
        OutputStream myOutput = new FileOutputStream(DB_PATH)) {

        // побайтово копируем данные
        byte[] buffer = new byte[1024];
        int length;
        while ((length = myInput.read(buffer)) > 0) {
            myOutput.write(buffer, 0, length);
        }
        myOutput.flush();
    }
    catch(IOException ex){
        Log.d("DatabaseHelper", ex.getMessage());
    }
}

public SQLiteDatabase open()throws SQLException {

    return SQLiteDatabase.openDatabase(DB_PATH, null,
        SQLiteDatabase.OPEN_READWRITE);
}
}

```

Перейдем к файлу **activity_main.xml**, который определяет визуальный интерфейс, и изменим его следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText android:id="@+id/userFilter"

        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Поиск"
        app:layout_constraintBottom_toTopOf="@+id/userList"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ListView

        android:id="@+id/userList"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/userFilter"
    />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Итак, у нас будет элемент `ListView` для отображения списка и текстовое поле для фильтрации.

Теперь изменим код **MainActivity**:

```
package com.example.livesearchapp;

import androidx.appcompat.app.AppCompatActivity;

import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.EditText;
import android.widget.FilterQueryProvider;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class MainActivity extends AppCompatActivity {

    DatabaseHelper sqlHelper;
    SQLiteDatabase db;
    Cursor userCursor;
    SimpleCursorAdapter userAdapter;
    ListView userList;
    EditText userFilter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

userList = (ListView)findViewById(R.id.userList);
userFilter = (EditText)findViewById(R.id.userFilter);


sqlHelper = new DatabaseHelper(getApplicationContext());
// создаем базу данных
sqlHelper.create_db();
}

@Override

public void onResume() {
    super.onResume();
    try {
        db = sqlHelper.open();

        userCursor = db.rawQuery("select * from " + DatabaseHelper.TABLE,
null);

        String[] headers = new String[]{DatabaseHelper.COLUMN_NAME,
DatabaseHelper.COLUMN_YEAR};

        userAdapter = new SimpleCursorAdapter(this,
android.R.layout.two_line_list_item,

        userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2},
0);


        // если в текстовом поле есть текст, выполняем фильтрацию
        // данная проверка нужна при переходе от одной ориентации экрана к
другой
        if(!userFilter.getText().toString().isEmpty())
            userAdapter.getFilter().filter(userFilter.getText().toString());


        // установка слушателя изменения текста
```

```

userFilter.addTextChangedListener(new TextWatcher() {

    public void afterTextChanged(Editable s) { }

    public void beforeTextChanged(CharSequence s, int start, int count, int
after) { }

    // при изменении текста выполняем фильтрацию
    public void onTextChanged(CharSequence s, int start, int before, int
count) {

        userAdapter.getFilter().filter(s.toString());
    }
});

// устанавливаем провайдер фильтрации
userAdapter.setFilterQueryProvider(new FilterQueryProvider() {
    @Override
    public Cursor runQuery(CharSequence constraint) {

        if (constraint == null || constraint.length() == 0) {

            return db.rawQuery("select * from " + DatabaseHelper.TABLE,
null);
        }
        else {
            return db.rawQuery("select * from " + DatabaseHelper.TABLE + "
where " +
                DatabaseHelper.COLUMN_NAME + " like ?", new
String[]{ "%" + constraint.toString() + "%" });
        }
    }
});

```

```

        }
    });

    userList.setAdapter(userAdapter);
}
catch (SQLException ex){ }
}
@Override
public void onDestroy(){
    super.onDestroy();
    // Закрываем подключение и курсор
    db.close();
    userCursor.close();
}
}

```

Прежде всего надо отметить, что для фильтрации данных в адаптере, нам надо получить фильтр адаптера, а у этого фильтра выполнить метод *filter()*:
`userAdapter.getFilter().filter(s.toString());`

В этот метод `filter()` передается ключ поиска.

Для текстового поля мы можем отслеживать изменения содержимого с помощью слушателя:

```

userFilter.addTextChangedListener(new TextWatcher() {

    public void afterTextChanged(Editable s) {
    }

    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }
}

```

```

    }

    // при изменении текста выполняем фильтрацию
    public void onTextChanged(CharSequence s, int start, int before, int count) {

        userAdapter.getFilter().filter(s.toString());

    }

});

```

В слушателе TextWatcher в методе onTextChanged как раз и вызывается метод filter(), в который передется введенная пользователем в текстовое поле последовательность символов.

Сам вызов метода filter() мало на что влияет. Нам надо еще определить провайдер фильтрации адаптера, которые и будет инкапсулировать реальную логику фильтрации:

```

userAdapter.setFilterQueryProvider(new FilterQueryProvider() {

    @Override

    public Cursor runQuery(CharSequence constraint) {

        if (constraint == null || constraint.length() == 0) {

            return db.rawQuery("select * from " + DatabaseHelper.TABLE, null);

        }

        else {

            return db.rawQuery("select * from " + DatabaseHelper.TABLE + " where "

+

            DatabaseHelper.COLUMN_NAME + " like ?", new String[]{"%" +

constraint.toString() + "%"});

        }

    }

});

```

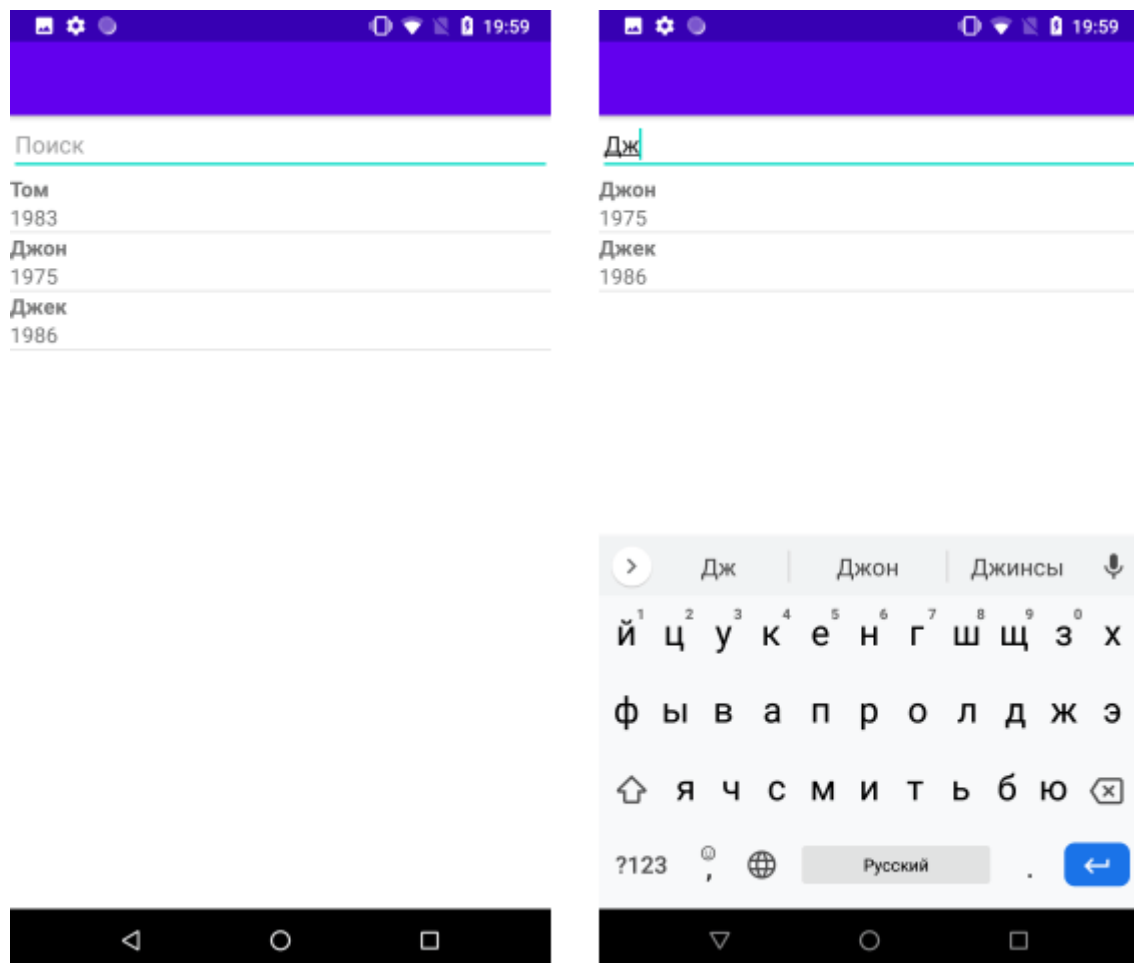
Сущность этого провайдера заключается в выполнении SQL-выражений к бд, а именно конструкций "select from" и "select from where like". Данные простейшие выражения выполняют регистрозависимую фильтрацию. В результате адаптере получает отфильтрованные данные.

Следует также отметить следующий код:

```
if(!userFilter.getText().toString().isEmpty())  
    userAdapter.getFilter().filter(userFilter.getText().toString());
```

Данный код нам нужен при смене ориентации (например, с портретной на альбомную). И если ориентация устройства изменена, но в текстовом поле все же есть некоторые текст-фильтр, то выполняется фильтрация. Иначе бы она не выполнялась.

И после запуска мы сможем насладиться фильтрацией данных:



Live Search Фильтрация данных в Android и Java и SQLite

Модель, репозиторий и работа с базой данных

В прошлых материалах было рассмотрено взаимодействие с базой данных через класс SimpleCursorAdapter. Но есть и другие способы работы с данными, когда мы абстрагируемся от структуры таблицы и работаем через модель, а все взаимодействие с базой данных производится фактически через реализацию паттерна репозиторий.

Так, создадим новый проект с пустой MainActivity и прежде всего добавим в него класс модели, который назовем **User**:

```
package com.example.databaseadapterapp;
```

```
public class User {
```

```
    private long id;
```

```
    private String name;
```

```
    private int year;
```

```
    User(long id, String name, int year){
```

```
        this.id = id;
```

```
        this.name = name;
```

```
        this.year = year;
```

```
    }
```

```
    public long getId() {
```

```
        return id;
```

```
    }
```

```
    public String getName() {
```

```
        return name;
```

```
    }
```

```
    public void setName(String name) {
```

```

        this.name = name;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    @Override
    public String toString() {
        return this.name + " : " + this.year;
    }
}

```

В данном проекте мы будем работать фактически с теми же данными, что и ранее с данными пользователей, у которых есть уникальный идентификатор, имя и год рождения. И модель User как раз описывает эти данные.

Для взаимодействия с базой данных SQLite добавим новый класс **DatabaseHelper**:

```

package com.example.databaseadapterapp;

import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "userstore.db"; // название бд

```

```

private static final int SCHEMA = 1; // версия базы данных
static final String TABLE = "users"; // название таблицы в бд
// названия столбцов
public static final String COLUMN_ID = "_id";
public static final String COLUMN_NAME = "name";
public static final String COLUMN_YEAR = "year";

public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, SCHEMA);
}

@Override
public void onCreate(SQLiteDatabase db) {

    db.execSQL("CREATE TABLE " + TABLE + " (" + COLUMN_ID
        + " INTEGER PRIMARY KEY AUTOINCREMENT," +
COLUMN_NAME
        + " TEXT, " + COLUMN_YEAR + " INTEGER);");
    // добавление начальных данных
    db.execSQL("INSERT INTO " + TABLE + " (" + COLUMN_NAME
        + ", " + COLUMN_YEAR + ") VALUES ('Том Смит', 1981);");
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE);
    onCreate(db);
}
}

```

Также для работы с базой данных добавим в проект класс **DatabaseAdapter**:

```
package com.example.databaseadapterapp;
```

```
import android.content.ContentValues;
```

```
import android.content.Context;
```

```
import android.database.Cursor;
```

```
import android.database.DatabaseUtils;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
public class DatabaseAdapter {
```

```
    private DatabaseHelper dbHelper;
```

```
    private SQLiteDatabase database;
```

```
    public DatabaseAdapter(Context context){
```

```
        dbHelper = new DatabaseHelper(context.getApplicationContext());
```

```
    }
```

```
    public DatabaseAdapter open(){
```

```
        database = dbHelper.getWritableDatabase();
```

```
        return this;
```

```
    }
```

```
    public void close(){
```

```
        dbHelper.close();
```

```
    }
```

```
private Cursor getAllEntries(){  
    String[] columns = new String[] {DatabaseHelper.COLUMN_ID,  
DatabaseHelper.COLUMN_NAME, DatabaseHelper.COLUMN_YEAR};  
    return database.query(DatabaseHelper.TABLE, columns, null, null, null, null,  
null);  
}
```

```
public List<User> getUsers(){  
    ArrayList<User> users = new ArrayList<>();  
    Cursor cursor = getAllEntries();  
    while (cursor.moveToNext()){  
        int id =  
cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN_ID));  
        String name =  
cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN_NAME));  
        int year =  
cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN_YEAR));  
        users.add(new User(id, name, year));  
    }  
    cursor.close();  
    return users;  
}
```

```
public long getCount(){  
    return DatabaseUtils.queryNumEntries(database, DatabaseHelper.TABLE);  
}
```

```
public User getUser(long id){  
    User user = null;
```

```

        String query = String.format("SELECT * FROM %s WHERE
%s=?", DatabaseHelper.TABLE, DatabaseHelper.COLUMN_ID);

        Cursor cursor = database.rawQuery(query, new String[]{
String.valueOf(id)});

        if(cursor.moveToFirst()){

            String name =
cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN_NAME));

            int year =
cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN_YEAR));

            user = new User(id, name, year);
        }

        cursor.close();

        return user;
    }

    public long insert(User user){

        ContentValues cv = new ContentValues();

        cv.put(DatabaseHelper.COLUMN_NAME, user.getName());
        cv.put(DatabaseHelper.COLUMN_YEAR, user.getYear());

        return database.insert(DatabaseHelper.TABLE, null, cv);
    }

    public long delete(long userId){

        String whereClause = "_id = ?";

        String[] whereArgs = new String[]{String.valueOf(userId)};

        return database.delete(DatabaseHelper.TABLE, whereClause, whereArgs);
    }

```

```

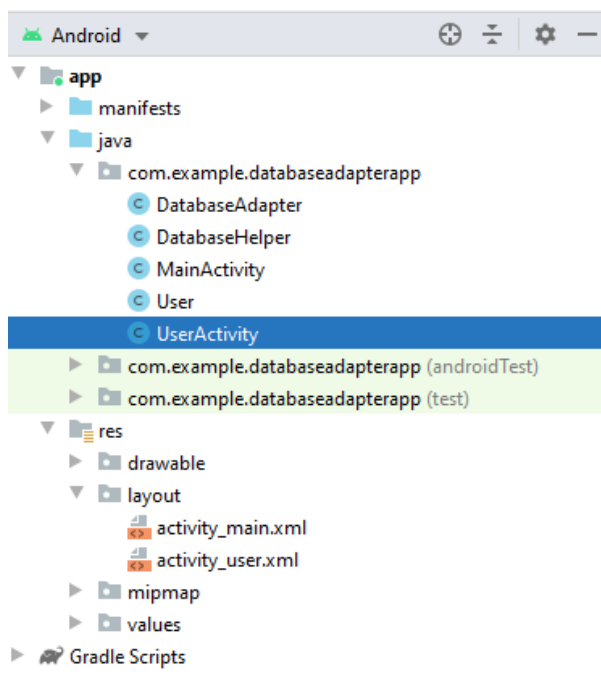
public long update(User user){
    String whereClause = DatabaseHelper.COLUMN_ID + "=" + user.getId();
    ContentValues cv = new ContentValues();
    cv.put(DatabaseHelper.COLUMN_NAME, user.getName());
    cv.put(DatabaseHelper.COLUMN_YEAR, user.getYear());
    return database.update(DatabaseHelper.TABLE, cv, whereClause, null);
}
}

```

Фактически данный класс выполняет роль репозитория данных. Чтобы взаимодействовать с БД он определяет методы `open()` и `close()`, которые соответственно открывают и закрывают подключение к базе данных.

Непосредственно для работы с данными в классе определены методы `insert()` (добавление), `delete()` (удаление), `update()` (обновление), `getUsers()` (получение всех пользователей из таблицы) и `getUser()` (получение одного пользователя по id).

В качестве пользовательского интерфейса будем отталкиваться от того функционала, который использовался в прошлых темах. Так, добавим в проект новый класс Activity - **UserActivity**. В итоге весь проект будет выглядеть следующим образом:



Модель и репозиторий в Android и Java

В файле **activity_user.xml** в папке **res/layout** определим для **UserActivity** простейший интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/name"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите имя"
        app:layout_constraintBottom_toTopOf="@+id/year"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/year"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Введите год рождения"
        app:layout_constraintTop_toBottomOf="@+id/name"
        app:layout_constraintBottom_toTopOf="@+id/saveButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <Button
        android:id="@+id/saveButton"
```



```
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Сохранить"
android:onClick="save"
app:layout_constraintHorizontal_weight="1"
app:layout_constraintTop_toBottomOf="@+id/year"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toLeftOf="@+id/deleteButton"
/>
```

<Button

```
android:id="@+id/deleteButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Удалить"
android:onClick="delete"
app:layout_constraintHorizontal_weight="1"
app:layout_constraintTop_toBottomOf="@+id/year"
app:layout_constraintLeft_toRightOf="@+id/saveButton"
app:layout_constraintRight_toRightOf="parent"
/>
```

</androidx.constraintlayout.widget.ConstraintLayout>

В классе **UserActivity** определим логику добавления/изменения/удаления пользователя:

```
package com.example.databaseadapterapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class UserActivity extends AppCompatActivity {

    private EditText nameBox;
    private EditText yearBox;
    private Button delButton;

    private DatabaseAdapter adapter;
    private long userId=0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);

        nameBox = findViewById(R.id.name);
        yearBox = findViewById(R.id.year);
        delButton = findViewById(R.id.deleteButton);
        adapter = new DatabaseAdapter(this);

        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            userId = extras.getLong("id");
        }
        // если 0, то добавление
        if (userId > 0) {
```

```

        // получаем элемент по id из бд
        adapter.open();
        User user = adapter.getUser(userId);
        nameBox.setText(user.getName());
        yearBox.setText(String.valueOf(user.getYear()));
        adapter.close();
    } else {
        // скрываем кнопку удаления
        delButton.setVisibility(View.GONE);
    }
}

public void save(View view){

    String name = nameBox.getText().toString();
    int year = Integer.parseInt(yearBox.getText().toString());
    User user = new User(userId, name, year);

    adapter.open();
    if (userId > 0) {
        adapter.update(user);
    } else {
        adapter.insert(user);
    }
    adapter.close();
    goHome();
}

public void delete(View view){

```

```

        adapter.open();

        adapter.delete(userId);

        adapter.close();

        goHome();
    }

    private void goHome(){
        // переход к главной activity

        Intent intent = new Intent(this, MainActivity.class);

        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);

        startActivity(intent);
    }
}

```

Эта activity используется для добавления/редактирования/удаления одного объекта User. Если в UserActivity передается параметр id, то значит мы находимся в режиме редактирования пользователя, поэтому обращаемся к методу getUser() класса DatabaseAdapter для получения нужного пользователя.

Для добавления/изменения/удаления пользователя по нажатию на кнопку вызывается соответствующий метод класса DatabaseAdapter.

В файле **activity_main.xml** в папке **res/layout** определим визуальный интерфейс для MainActivity:

```

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"

    android:layout_height="match_parent">

    <Button

```

```
android:id="@+id/addButton"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Добавить"
android:onClick="add"
app:layout_constraintBottom_toTopOf="@+id/list"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
/>
```

<ListView

```
android:id="@+id/list"
android:layout_width="0dp"
android:layout_height="0dp"
app:layout_constraintTop_toBottomOf="@+id/addButton"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"/>
```

</androidx.constraintlayout.widget.ConstraintLayout>

Здесь имеется элемент ListView для вывода объектов из таблицы и кнопка для перехода к UserActivity для добавления пользователя.

И изменим код **MainActivity**:

```
package com.example.databaseadapterapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
```



```

        startActivity(intent);
    }
}
));
}

@Override
public void onResume() {
    super.onResume();

    DatabaseAdapter adapter = new DatabaseAdapter(this);
    adapter.open();

    List<User> users = adapter.getUsers();

    arrayAdapter = new ArrayAdapter<>(this,
    android.R.layout.simple_list_item_1, users);

    userList.setAdapter(arrayAdapter);
    adapter.close();
}

// по нажатию на кнопку запускаем UserActivity для добавления данных
public void add(View view){
    Intent intent = new Intent(this, UserActivity.class);
    startActivity(intent);
}
}

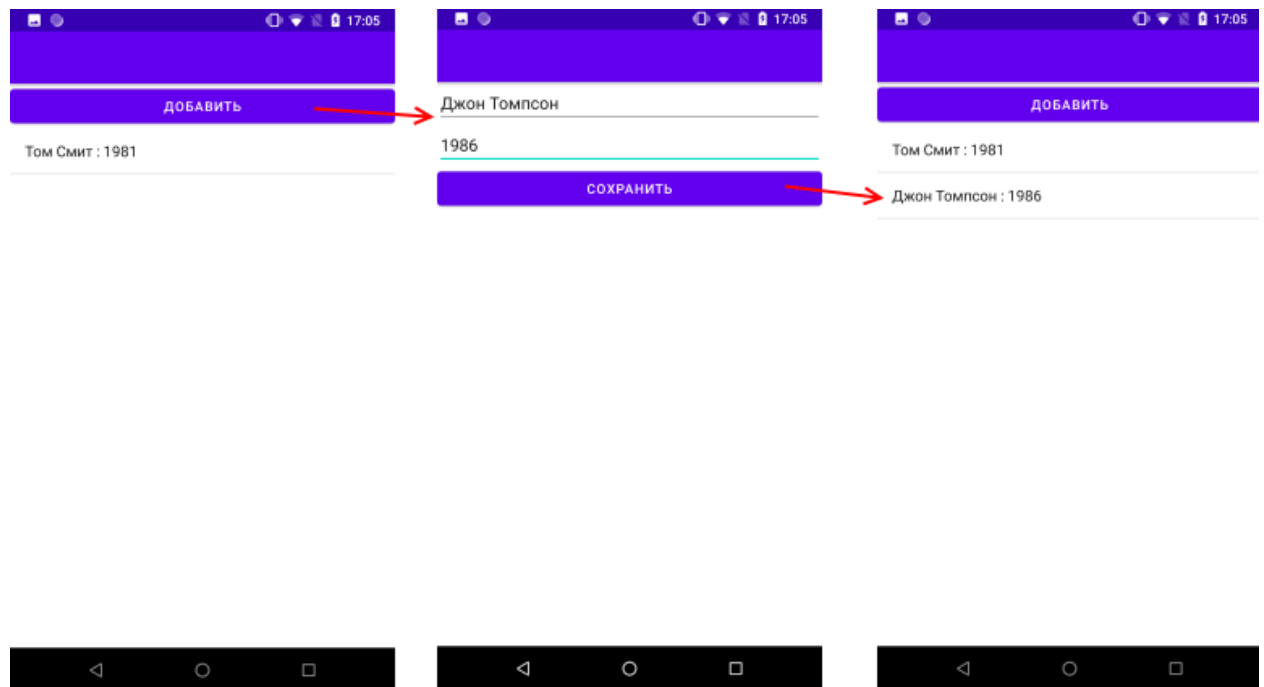
```

В переопределенном методе onResume() через объект DatabaseAdapter получаем всех пользователей из базы данных и через ArrayAdapter выводим их в ListView.

При нажатии на элемент ListView запускаем UserActivity, передавая ей id выделенного пользователя.

При нажатии на кнопку просто вызываем UserActivity.

При запуске MainActivity отобразит список пользователей из базы данных, а при переходе к UserActivity мы сможем отредактировать или добавить пользователей:



Работа с данными в SQLite через репозиторий в Android и Java

Задание

1. Обеспечить сохранение состояния приложения.
2. Реализовать в приложении создание и получение настроек, используя SharedPreferences.
3. Реализовать приватные настройки.
4. Реализовать технологии PreferenceFragmentCompat
5. Реализовать чтение и сохранение файлов.
6. Реализовать размещение файлов во внешнем хранилище
7. Обеспечить работу с локальной СУБД SQLite. Реализовать подключение к базе данных, используя СУБД SQLite, Реализовать создание и открытие базы данных. Обеспечить получение данных из баз данных
8. Для упрощения работы с базами данных SQLite в Android применить класс SQLiteOpenHelper
9. Обеспечить взаимодействие по получению данных, используя класс

Cursor, который предлагает ряд методов для управления выборкой и класс CursorAdapter, позволяющий адаптировать полученный с помощью курсора набор к отображению в списковых элементах наподобие ListView

10. Реализовать добавление, удаление и обновление данных в SQLite.
Применяйте ContentValues
11. Реализуйте использование существующей БД SQLite
12. Организуйте динамический поиск по базе данных SQLite
13. Реализовать работу с базой данных через модель. Все взаимодействие с базой данных производится через реализацию паттерна репозиторий.