

ЗАНЯТИЕ 2.12

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Тема: Работа с SQLite в Android.

Упражнение №1

В данном упражнении будет реализовано добавление сведений о людях (Person) в базу данных SQLite. Хранению в базе данных подлежат: имя, фамилия, возраст человека.

Создайте новое Android-приложение для телефона с использованием шаблона Empty Activity (или Empty View Activity, в зависимости от версии Android Studio).

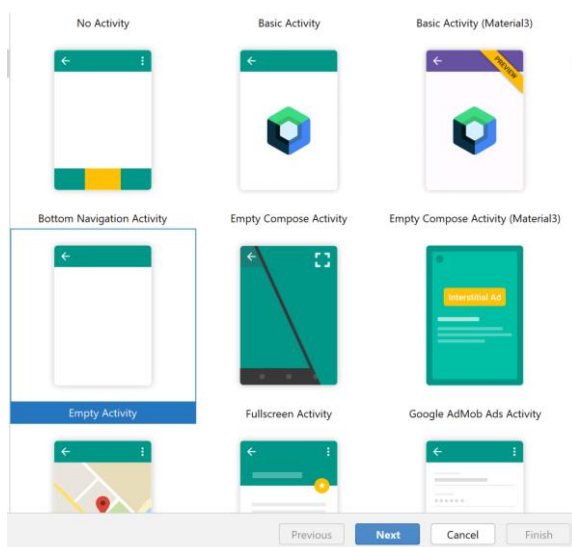


Рисунок 1

Установите минимальную версию API 21. Язык разработки – **Java** (не **Kotlin**). Имя приложения – **DataStorageApplication**.

Empty Activity

Creates a new empty activity

Name:

Package name:

Save location:

Language:

Minimum SDK:

Your app will run on approximately **99,3%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries

Рисунок 2

Дождитесь создания приложения и корректного обновления плагина Gradle. Перейдите в файл макета (в режим Code) **activity_main.xml**. Замените контейнер **ConstraintLayout** на **RelativeLayout** и поместите внутри него еще один контейнер – **LinearLayout** с вертикальной ориентацией (существующий виджет **TextView** нужно удалить):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_margin="8dp">
    </LinearLayout>
</RelativeLayout>
```

Рисунок 3

Атрибут **layout_margin** устанавливает отступ (сверху, снизу, справа, слева) величиной 8 dp. Созданный **LinearLayout** будет содержать поля для ввода значений с клавиатуры (имя, фамилия, возраст). Чтобы виджеты **EditText** смотрелись раздельно друг от друга будет правильным реализовать для них контур (**border**). Так как стандартные атрибуты не позволяют это реализовать, необходимо создать отдельный xml файл **drawable-ресурса**, который будет использоваться как **background** для **EditText** (да, **drawable** может содержать не только файлы картинок, но и графику, описанную с помощью **xml**).

Нажмите правой кнопкой мыши по каталогу **drawable** и добавьте в него новый файл ресурсов.

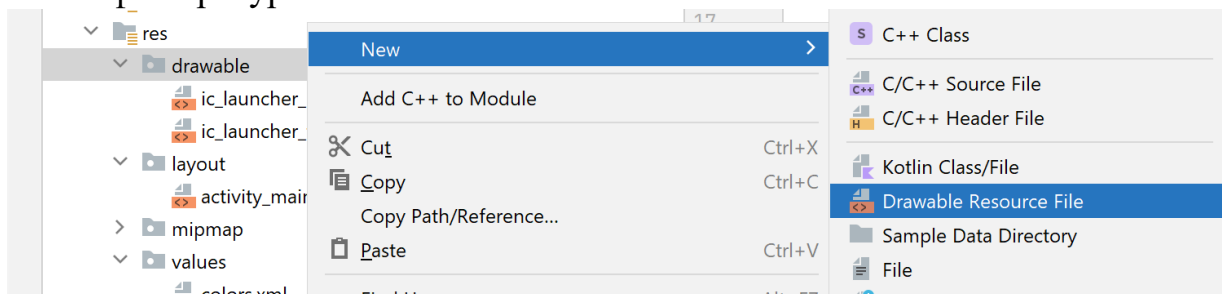
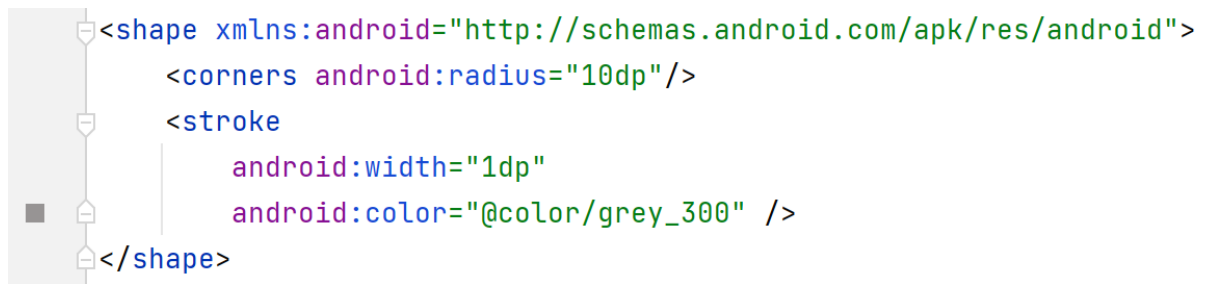


Рисунок 4

Назовите создаваемый файл **border_edit_text**.

XML-код файла:



```

<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <corners android:radius="10dp"/>
    <stroke
        android:width="1dp"
        android:color="@color/grey_300" />
</shape>

```

Рисунок 5

Убедитесь, что в файле colors.xml у вас описан серый цвет (если нет, необходимо его добавить).

С помощью тега `<shape>` можно описывать простые геометрические фигуры, указывая их размеры, фон и контур.

Фигура `<shape>` является корневым элементом в XML.

Атрибуты:

android:shape - задаёт тип фигуры: `rectangle` (прямоугольник, заполняющий элемент, является фигурой по умолчанию), `oval` (овал), `line` (линия, требуется также наличие элемента `<stroke>` для задания ширины линии), `ring` (окружность, для данной фигуры можно использовать атрибуты `android:innerRadius`, `android:innerRadiusRatio`, `android:thickness`, `android:thicknessRatio`, `android:useLevel`).

<corners> - создаёт закругленные углы для фигуры. Только для прямоугольника. Возможные атрибуты: `android:radius`, `android:topLeftRadius`, `android:topRightRadius`, `android:bottomLeftRadius`, `android:bottomRightRadius`.

<gradient> - задаёт градиентную заливку для фигуры. Возможные атрибуты: `android:angle`, `android:centerX`, `android:centerY`, `android:centerColor`, `android:endColor`, `android:gradientRadius`, `android:startColor`, `android:type`, `android:useLevel`.

<padding> - отступы. Возможные атрибуты: `android:left`, `android:top`, `android:right`, `android:bottom`.

<size> - размеры фигуры. Возможные атрибуты: `android:height`, `android:width`.

<solid> - сплошной цвет для фигуры. Возможные атрибуты: `android:color`.

<stroke> - контур фигуры. Возможные атрибуты: `android:width`, `android:color`, `android:dashGap` (расстояние между черточками), `android:dashWidth` (длина пунктирной черточки).

Вернитесь в файл макета `main_activity.xml` и добавьте три виджета `EditText` внутрь контейнера `LinearLayout`. Обратите внимание, что для атрибута `android:background` в каждом `EditText` указана ссылка на разработанный на предыдущем шаге макет, что позволяет реализовать для виджета закругленный контур.

```

        android:layout_margin="0dp"
    <EditText
        android:id="@+id/etFirstName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:paddingStart="4dp"
        android:paddingEnd="4dp"
        android:hint="@string/name"
        android:background="@drawable/border_edit_text"/>
    <EditText
        android:id="@+id/etLastName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:layout_marginTop="4dp"
        android:paddingStart="4dp"
        android:paddingEnd="4dp"
        android:hint="@string/lname"
        android:background="@drawable/border_edit_text"/>
    <EditText
        android:id="@+id/etAge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:layout_marginTop="4dp"
        android:paddingStart="4dp"
        android:paddingEnd="4dp"
        android:inputType="number"
        android:hint="@string/age"
        android:background="@drawable/border_edit_text"/>
</LinearLayout>

```

Рисунок 6

Обратите внимание, что ссылки на строки используются из файла ресурсов strings.xml:

```

<resources>
    <string name="app_name">DataStorageApplication</string>
    <string name="name">Имя</string>
    <string name="lname">Фамилия</string>
    <string name="age">Возраст</string>
</resources>

```

Рисунок 7

Переключитесь в режим дизайна и убедитесь, что виджеты EditText имеют контур и располагаются друг под другом:

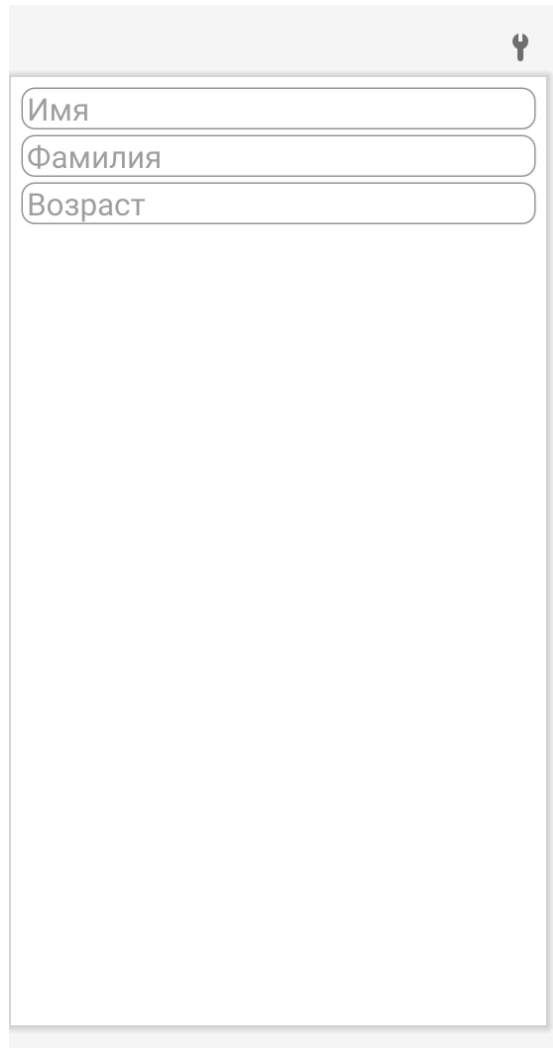


Рисунок 8

Добавьте еще две кнопки в LinearLayout под полями ввода (описание виджетов EditText можно свернуть). Текст в кнопках: «Добавить» и «Продолжить». Обратите внимание, что ссылки на строки используются из файла ресурсов strings.xml:

```
<string name="add">Добавить</string>  
<string name="next">Продолжить</string>
```

Рисунок 9

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_margin="8dp">
        <EditText...>
        <EditText...>
        <EditText...>
        <Button
            android:id="@+id/btnAdd"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="32dp"
            android:layout_gravity="center"
            android:text="@string/add"/>
        <Button
            android:id="@+id/btnNext"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dp"
            android:layout_gravity="center"
            android:text="@string/next"/>
    </LinearLayout>
</RelativeLayout>

```

Рисунок 10

Итоговый макет:

The image shows a mobile application interface. It features a light gray background with a white rounded rectangle containing the form. At the top right of the gray area is a small gray wrench icon. The form consists of three vertically stacked input fields with rounded corners and thin gray borders. The first field is labeled 'Имя' (Name), the second 'Фамилия' (Surname), and the third 'Возраст' (Age). Below these fields, there are two blue buttons with white text. The top button is labeled 'ДОБАВИТЬ' (Add) and the bottom button is labeled 'ПРОДОЛЖИТЬ' (Continue).

Рисунок 11

Далее необходимо подготовить код для работы с базой данных SQLite. Создайте в проекте новый класс – MyAppSQLiteHelper. Класс должен расширять стандартный (входящий в состав Android SDK) абстрактный класс SQLiteOpenHelper. Так как класс абстрактный, необходимо переопределить все его абстрактные методы: onCreate и onUpgrade. Также родительский класс SQLiteOpenHelper требует создания конструктора. Создать конструктор и переопределяемые методы можно с помощью средств генерации кода (**не нужно писать все вручную!**).

onCreate() – вызывается при первом создании базы данных.

onUpgrade() – вызывается при модификации базы данных.

В методе onCreate необходимо реализовать выполнение SQL-команды для создания таблиц, которые будут находиться в базе данных. В данном случае таблица в базе данных будет одна (TABLE_PERSONS). Выполнение SQL-кода в данном случае возможно с помощью метода execSQL объекта класса SQLiteDatabase.

```
public class MyAppSQLiteHelper extends SQLiteOpenHelper {

    public MyAppSQLiteHelper(Context c, String name, SQLiteDatabase.CursorFactory f, int version) {
        super(c, name, f, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table " + "TABLE_PERSONS" + " ("
            + "_id integer primary key autoincrement,"
            + "first_name text,"
            + "last_name text,"
            + "age integer" + ");");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // если версия изменилась, то вносим нужные нам изменения
    }

}
```

Рисунок 12

Не нужно пугаться количеству строк и знаков «+». Такой синтаксис используется исключительно для удобства записи команды:

create table TABLE_PERSONS (_id integer primary key autoincrement,first_name text,last_name text,age integer);

Немного теории.

Оператор SQLite **CREATE TABLE** используется для создания новой таблицы в БД. Создание базовой таблицы включает в себя именование таблицы и определение ее столбцов и типа данных каждого столбца. Если в проекте одна база данных, то ее имя, при обращении к таблице, указывать необязательно.

```
CREATE TABLE database_name.table_name(  
    column1 datatype PRIMARY KEY(one or more columns),  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype  
);
```

Рисунок 13

AUTOINCREMENT - это ключевое слово, используемое для автоматического увеличения значения поля в таблице. Можно автоматически увеличивать значение поля, используя ключевое слово **AUTOINCREMENT** при создании таблицы с определенным именем столбца для автоматического увеличения. Ключевое слово **AUTOINCREMENT** может использоваться только с полем **INTEGER**.

Основное использование ключевого слова **AUTOINCREMENT** заключается в следующем:

```
CREATE TABLE table_name(  
    column1 INTEGER AUTOINCREMENT,  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype,  
);
```

Рисунок 14

Первичный ключ или **PRIMARY KEY** в базах данных SQLite – это отдельное поле или комбинация полей, однозначно определяющая запись. Таблица может иметь только один первичный ключ.

Создайте в проекте еще один класс – **Person**. В нем необходимо создать переменные класса: **firstName**, **lastName**, **age**, а также конструктор, сеттеры и геттеры (**используйте средства генерации кода!**).


```

public class Person {
    private String firstName;
    private String lastName;
    private int age;

    public Person(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}

```

Рисунок 15

Перейдите в класс MainActivity и инициализируйте ссылки виджетов EditText и Button. Также для кнопок необходимо зарегистрировать слушателя. В данном случае слушателем будет сама activity, а это значит класс MainActivity должен реализовывать интерфейс View.OnClickListener.

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editTextFirstName, editTextLastName, editTextAge;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editTextFirstName = findViewById(R.id.etFirstName);
        editTextLastName = findViewById(R.id.etLastName);
        editTextAge = findViewById(R.id.etAge);
        Button btnAdd = findViewById(R.id.btnAdd);
        Button btnNext = findViewById(R.id.btnNext);
        btnAdd.setOnClickListener(this);
        btnNext.setOnClickListener(this);
        //...
    }

    @Override
    public void onClick(View v) {

    }
}

```

Рисунок 16

Для удобства работы с базой данных хорошей практикой считается создание класса, в котором будут содержаться методы для манипуляции данными (добавление, удаление, выборка).

Создайте в приложении новый класс – DBManager. В классе необходимо определить одну переменную – ссылку на SQLiteOpenHelper. Создайте конструктор для ее инициализации.

```

public class DBManager {

    private SQLiteOpenHelper sqLiteHelper;

    public DBManager(SQLiteOpenHelper sqLiteHelper) {
        this.sqLiteHelper = sqLiteHelper;
    }

}

```

Рисунок 17

Создайте в классе MainActivity переменную типа DBManager, через которую будет осуществляться доступ к базе данных и непосредственно, работа с данными.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editTextFirstName, editTextLastName, editTextAge;
    private DBManager dbManager; ←
```

Рисунок 1

Инициализацию этой переменной необходимо поместить в метод onCreate класса MainActivity, чтобы ссылка была доступна сразу при создании главной activity. Конструктор класса DBManager в качестве аргумента принимает объект созданного ранее MyAppSQLiteHelper.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.dbManager = new DBManager(new MyAppSQLiteHelper(c: this, name: "my_database.db", f: null, version: 1));
```

Рисунок 19

Параметры конструктора для MyAppSQLiteHelper в данном случае: ссылка на Context, имя файла базы данных, фабрика для создания «кастомного» курсора (необязательный параметр, поэтому null), версия базы данных.

Вернитесь в класс DBManager. Создайте метод для добавления данных объекта Person в таблицу базы данных:

```
public class DBManager {

    private SQLiteOpenHelper sqliteHelper;
    public DBManager(SQLiteOpenHelper sqliteHelper) {
        this.sqliteHelper = sqliteHelper;
    }

    public boolean savePersonToDatabase(Person person){
        SQLiteDatabase db = this.sqliteHelper.getWritableDatabase(); // получили ссылку на базу данных
        // создание переменной для хранения данных в формате "ключ-значение"
        // по сути формируем запись для вставки в БД
        ContentValues cv = new ContentValues();
        cv.put("first_name", person.getFirstName());
        cv.put("last_name", person.getLastName());
        cv.put("age", person.getAge());
        //вставка записи в таблицу базы данных
        //метод возвращает номер строки в случае успешной вставки или -1, если произошла ошибка
        long rowId = db.insert( table: "TABLE_PERSONS", nullColumnHack: null,cv);
        cv.clear(); // очистка
        db.close(); // закрытие базы данных
        return rowId != -1; // возвращаем результат вставки
    }
}
```

Рисунок 20

Для добавления записи в таблицу используется метод insert класса SQLiteDatabase.

Также есть следующие методы:

- ❑ **query** – выбор из база данных (select)
- ❑ **delete** – удаление записи (записей) из базы данных
- ❑ **update** – обновление существующих значений в базе данных
- ❑ **replace** – вставка (или обновление, если запись была)
- ❑ **execSQL/rawSQL** – выполнение произвольного запроса

Создайте метод в этом же классе для выборки из таблицы TABLE_PERSONS всех записей:

```
public ArrayList<Person> loadAllPersonsFromDatabase(){
    ArrayList<Person> persons = new ArrayList<>();
    SQLiteDatabase db = this.sqliteHelper.getWritableDatabase();
    Cursor dbCursor = db.query( table: "TABLE_PERSONS",
                                columns: null, selection: null, selectionArgs: null,
                                groupBy: null, having: null, orderBy: null);
    if(dbCursor.moveToFirst()){
        do{
            String fName = dbCursor.getString(dbCursor.getColumnIndexOrThrow( columnName: "first_name"));
            String lName = dbCursor.getString(dbCursor.getColumnIndexOrThrow( columnName: "last_name"));
            int age = dbCursor.getInt(dbCursor.getColumnIndexOrThrow( columnName: "age"));
            persons.add(new Person(fName,lName,age));
        }while (dbCursor.moveToNext());
    }
    dbCursor.close();
    db.close();
    return persons;
}
```

Рисунок 21

Вернитесь в MainActivity и реализуйте код, который сохраняет значения из виджетов EditText в базу данных при нажатии на кнопку «Добавить». Также необходимо предусмотреть защиту от аномальных значений в полях ввода.

```
@Override
public void onClick(View v) {
    if(v.getId() == R.id.btnAdd){
        if(this.dbManager != null){
            String fName = editTextFirstName.getText().toString();
            String lName = editTextLastName.getText().toString();
            String age = editTextAge.getText().toString();
            if(!fName.isEmpty() && !lName.isEmpty() && !age.isEmpty()){
                boolean result = dbManager.savePersonToDatabase(new Person(fName,lName,Integer.parseInt(age)));
                if(result){
                    Toast.makeText( context: this, R.string.insert_success,Toast.LENGTH_LONG).show();
                }else{
                    Toast.makeText( context: this, R.string.insert_error,Toast.LENGTH_LONG).show();
                }
            }else{
                Toast.makeText( context: this, R.string.incorrect_value,Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

Рисунок 22

Обратите внимание, что ссылки на строки используются из файла ресурсов strings.xml:

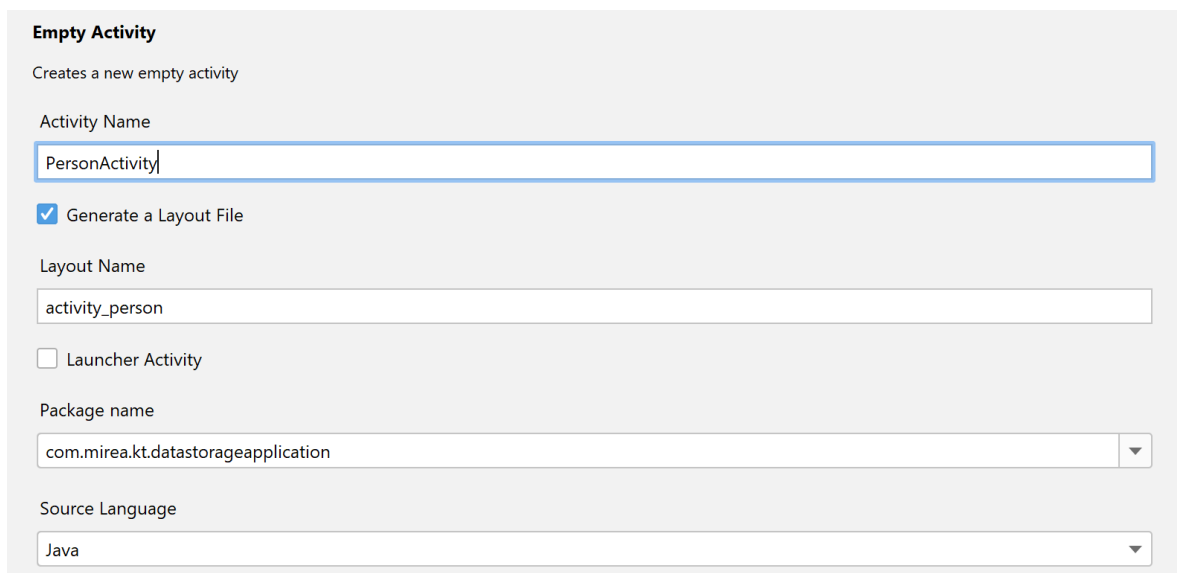
```
<string name="incorrect_value">Вы ввели некорректное значение</string>
<string name="insert_error">При добавлении произошла ошибка</string>
<string name="insert_success">Запись успешно добавлена</string>
```

Рисунок 23

Запустите приложение и убедитесь, что данные успешно добавляются в базу данных.

Упражнение №2

Продолжайте работу в текущем приложении.
Создайте еще одну activity – PersonActivity.



Empty Activity
Creates a new empty activity

Activity Name
PersonActivity

☒ Generate a Layout File

Layout Name
activity_person

☐ Launcher Activity

Package name
com.mirea.kt.datastorageapplication

Source Language
Java

Рисунок 24

В данной activity будет отображаться список людей, информация о которых хранится в базе данных, которая была заполнена в предыдущем упражнении. Реализуйте запуск второй activity по нажатию на кнопку «Продолжить» в MainActivity (необходимо дописать метод onClick).

```
    }
    }else{
        Toast.makeText(context: this, R.string.incorrect_value, Toast.LENGTH_LONG).show();
    }
}
}else if(v.getId() == R.id.btnNext){
    startActivity(new Intent(packageContext: this, PersonActivity.class));
}
}
```

САМОСТОЯТЕЛЬНО реализуйте в классе PersonActivity вызов метода получения всех записей из таблицы базы данных (сам метод был написан в упражнении №1). После получения данных необходимо отобразить их в виде списка в **RecyclerView** с помощью адаптера (см. упражнение №2 предыдущего практического занятия: <https://goo.su/kVkNj>).

Индивидуальное задание

Требования (читаем обязательно ВСЕ пункты):

- 1) На сдачу практического задания отводится **14 дней**.
- 2) Вариант определяется согласно порядковому номеру студента в журнале.
- 3) В случае дистанционного выполнения практического задания код программы необходимо выложить на Github и предоставить ссылку на него.
- 4) **Помимо исходного кода необходимо выкладывать файл apk (release-сборка, подписанная личной ЦП студента)**
- 5) Итоговое приложение должно компилироваться без ошибок, полностью выполнять требуемый функционал и на одном из экранов содержать информацию о номере варианта и ФИО студента.
- 6) Названия переменных и методов должны отражать суть и нести смысловую нагрузку.
- 7) **Обязательно использование логирования с приоритетом DEBUG или INFO (как минимум в методах жизненного цикла activity).**
- 8) Необходимо придерживаться стилистике по написанию Java-кода.
- 9) Необходимо предусмотреть защиту от ввода «аномальных» (ошибочных) значений.

Задание: разработать Android-приложение в соответствии с вариантом.

Вариант 1. Разработать Android-приложение для сохранения в базу данных SQLite информации об **АВТОМОБИЛЯХ**. Параметры автомобиля: модель (String), номер (String), год выпуска (int) необходимо считывать из виджетов EditText после запуска приложения. На втором экране (activity) необходимо загрузить из базы данных список автомобилей и **отобразить** его в RecyclerView с помощью адаптера. Элемент списка должен отображать все параметры объекта.

Вариант 2. Разработать Android-приложение для сохранения в базу данных SQLite информации о **ТЕЛЕФОННЫХ АППАРАТАХ**. Параметры телефона: модель (String), серийный номер (String), цена (int) необходимо считывать из виджетов EditText после запуска приложения. На втором экране (activity) необходимо загрузить из базы данных список телефонных аппаратов и

отобразить его в RecyclerView с помощью адаптера. Элемент списка должен отображать все параметры объекта.

Вариант 3. Разработать Android-приложение для сохранения в базу данных SQLite информации о **ДОКТОРАХ**. Параметры сущности ДОКТОР: имя и фамилия (String), специальность (String), флаг об аттестации (boolean) необходимо считывать из виджетов EditText после запуска приложения. На втором экране (activity) необходимо загрузить из базы данных список докторов и **отобразить** его в RecyclerView с помощью адаптера. Для считывания boolean параметры рекомендуется использования виджет Switch. Элемент списка должен отображать все параметры объекта.

Вариант 4. Разработать Android-приложение для сохранения в базу данных SQLite информации о **РАСТЕНИЯХ**. Параметры растения: название (String), разновидность (String), отметка тепличное растение или нет (boolean) необходимо считывать из виджетов EditText после запуска приложения. На втором экране (activity) необходимо загрузить из базы данных список растений и **отобразить** его в RecyclerView с помощью адаптера. Для считывания boolean параметры рекомендуется использования виджет Switch. Элемент списка должен отображать все параметры объекта.

Вариант 5. Разработать Android-приложение для сохранения в базу данных SQLite информации о **МАГАЗИНАХ**. Параметры магазина: название (String), адрес (String), время открытия, время закрытия необходимо считывать из виджетов EditText после запуска приложения. На втором экране (activity) необходимо загрузить из базы данных список магазинов и **отобразить** его в RecyclerView с помощью адаптера. Элемент списка должен отображать все параметры объекта.