



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных  
технологий

## **Отчет по практической работе №9**

по дисциплине «Разработка мобильных приложений»

**Выполнил:**

Студент группы ИКБО-20-23

Комисарик М.А.

**Проверил:**

Старший преподаватель кафедры  
МОСИТ

Шешуков Л.С.

Москва 2025 г.

# СОДЕРЖАНИЕ

1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ .....	3
1.1 Сохранение состояния приложения .....	3
1.2 Хранение данных в Android .....	4
1.3 Внутреннее хранилище.....	6
1.3.1 Создание и запись файла .....	6
1.3.2 Чтение файла .....	7
1.3.3 Удаление файла .....	8
1.4 Внешнее хранилище .....	9
1.4.1 Создание и запись файла .....	10
1.4.2 Чтение файла .....	11
1.4.3 Удаление файла .....	11
2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ .....	12
2.1 Внутреннее хранилище.....	12
2.1.1 Разметка.....	12
2.1.2 Реализация работы с файлами .....	14
2.1.3 Сохранение состояния приложения .....	17
2.1.4 Тестирование .....	17
2.2 Работа с внешней памятью.....	20
2.2.1 Разрешения.....	20
2.2.2 Разметка.....	20
2.2.3 Реализация.....	21
2.2.4 Тестирование .....	23
ЗАКЛЮЧЕНИЕ .....	25

# 1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

## 1.1 Сохранение состояния приложения

Сохранение и восстановление состояния приложения являются важными аспектами разработки Android-приложений, особенно при обработке изменений конфигурации, таких как поворот экрана, изменение языка и других сценариев, которые приводят к пересозданию Activity (Рисунок 1).

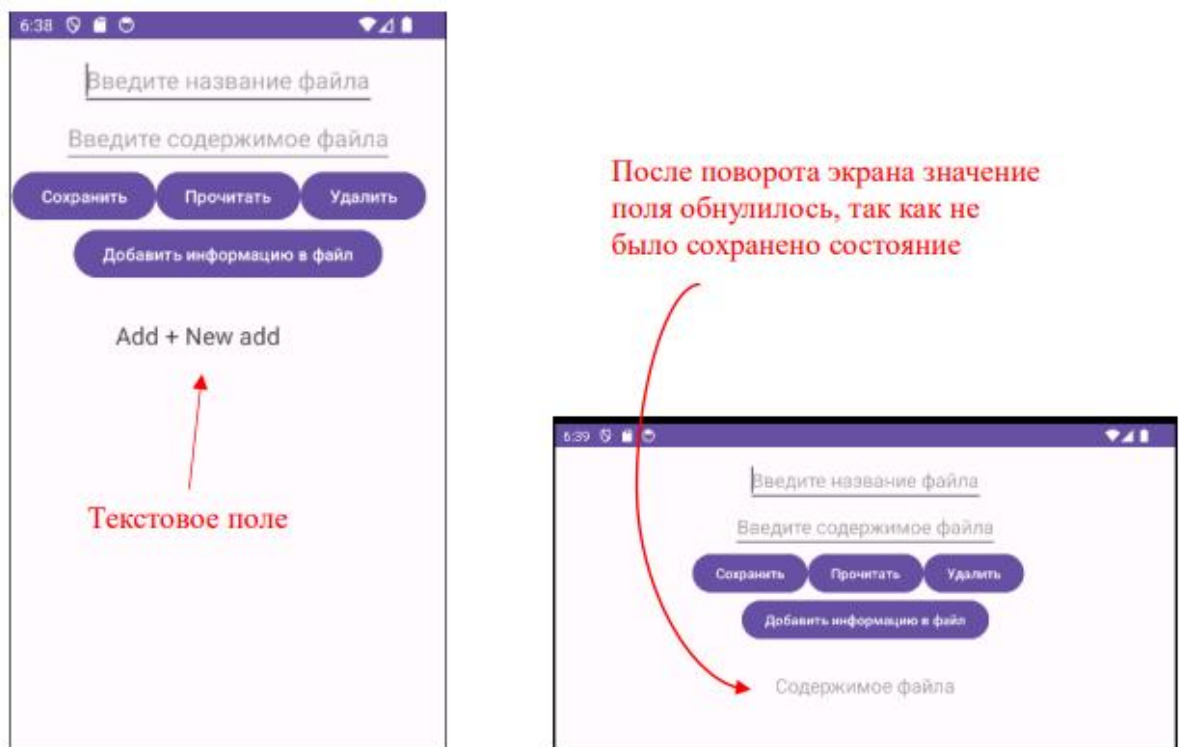


Рисунок 1 – Потеря введенных данных при повороте экрана

Android предоставляет несколько механизмов для управления состоянием приложения, включая использование методов `onSaveInstanceState()` и `onRestoreInstanceState()`. Эти методы позволяют сохранять и восстанавливать данные о состоянии пользовательского интерфейса, обеспечивая бесперебойное взаимодействие пользователя с приложением.

Метод `onSaveInstanceState()` вызывается системой перед тем, как активность будет уничтожена, чтобы дать возможность сохранить состояние пользовательского интерфейса в объект `Bundle` (Рисунок 2). В этот объект можно

добавлять различные типы данных, такие как строки, числа, сериализуемые объекты и другие.

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // Сохраняем значение строковой переменной
    outState.putString("KEY_STATE", "some state");
}
```

Рисунок 2 – Переопределение метода onSaveInstanceState()

В методе onSaveInstanceState() сохраняем состояние. Для этого вызываем у параметра Bundle метод putString(key, value), первый параметр которого - ключ, а второй - значение сохраняемых данных.

Метод onRestoreInstanceState() вызывается после метода onStart(), когда активность воссоздается после пересоздания. Этот метод получает объект Bundle, содержащий данные о состоянии, которые были сохранены в onSaveInstanceState() (Рисунок 3).

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    // Восстанавливаем сохраненное состояние
    String state = savedInstanceState.getString(key: "KEY_STATE");
    // Используем сохраненное значение для восстановления состояния UI или других компонентов
}
```

Рисунок 3 – Переопределение метода onRestoreInstanceState()

Таким образом, например, при смене ориентации экрана все записанные данные не пропадут.

## 1.2 Хранение данных в Android

Более удобным способом сохранять все состояния приложения является их запись в отдельный файл, который не будет затронут при смене метода жизненного цикла активности.

Работа с файловой системой в Android является ключевым элементом в процессе разработки мобильных приложений. Это необходимость обусловлена разнообразными задачами, с которыми сталкиваются разработчики: от простого сохранения настроек пользователя и состояний приложения до сложных операций с медиафайлами, документами и другими данными, требующими постоянного хранения между сессиями. Кроме того, эффективное использование файловой системы позволяет реализовывать функции загрузки ресурсов из сети и их последующего кэширования, что существенно улучшает производительность приложения и удобство его использования.

В Android предусмотрены два основных типа хранилища для работы с файлами: внутреннее (Рисунок 4) и внешнее.

**Внутреннее хранилище:** это хранилище, доступное пользователю только через установленные приложения (или при наличии root-доступа на устройстве). Пример: `data/data/app_packageName`.

Внешнее хранилище бывает двух типов:

- **основное внешнее хранилище:** встроенное общее хранилище, к которому «пользователь может получить доступ, подключив USB-кабель и установив его как накопитель на хост-компьютере». Пример: когда мы говорим о Nexus 5 с 32 ГБ памяти;
- **вторичное внешнее хранилище:** съемное хранилище. Пример: SD-карта.

Внутреннее хранилище гарантирует безопасность данных, делая их доступными только для вашего приложения, и обеспечивает их удаление при деинсталляции приложения. Внешнее хранилище, в свою очередь, предлагает больший объем памяти и возможность обмена файлами между приложениями и даже передачу данных между устройствами.

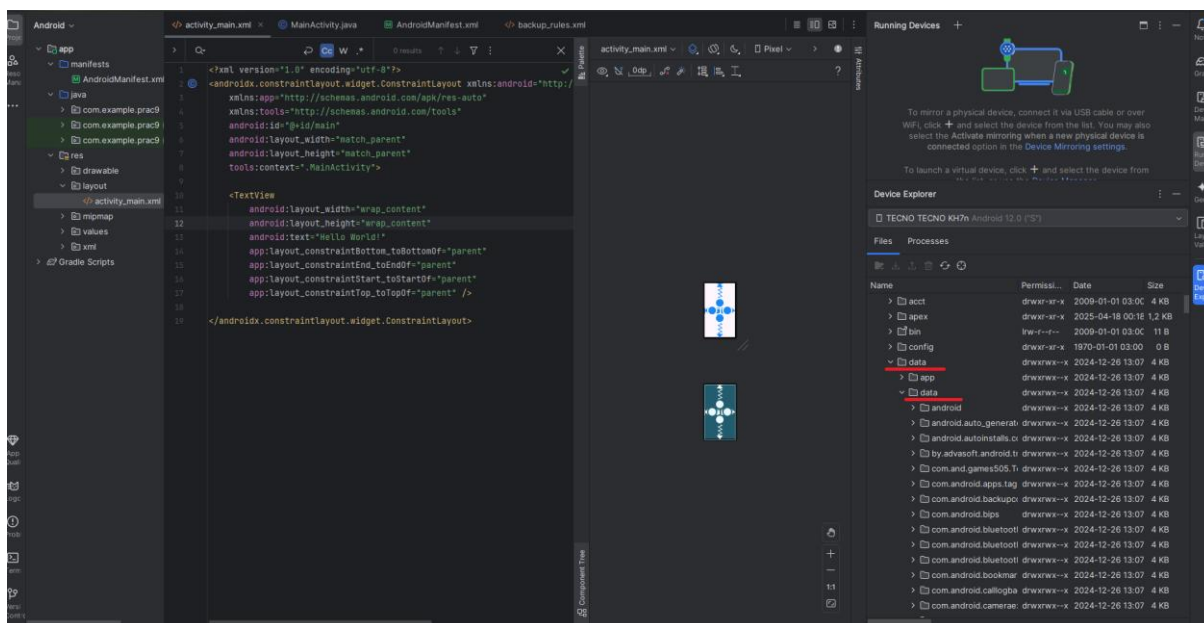


Рисунок 4 – Расположение внутренней памяти в Android

## 1.3 Внутреннее хранилище

Внутреннее хранилище предназначено для индивидуальных данных приложения, к которым не предполагается доступ из других приложений. Данные, сохраненные во внутреннем хранилище, удаляются при удалении приложения. Для работы с внутренним хранилищем используются различные методы.

### 1.3.1 Создание и запись файла

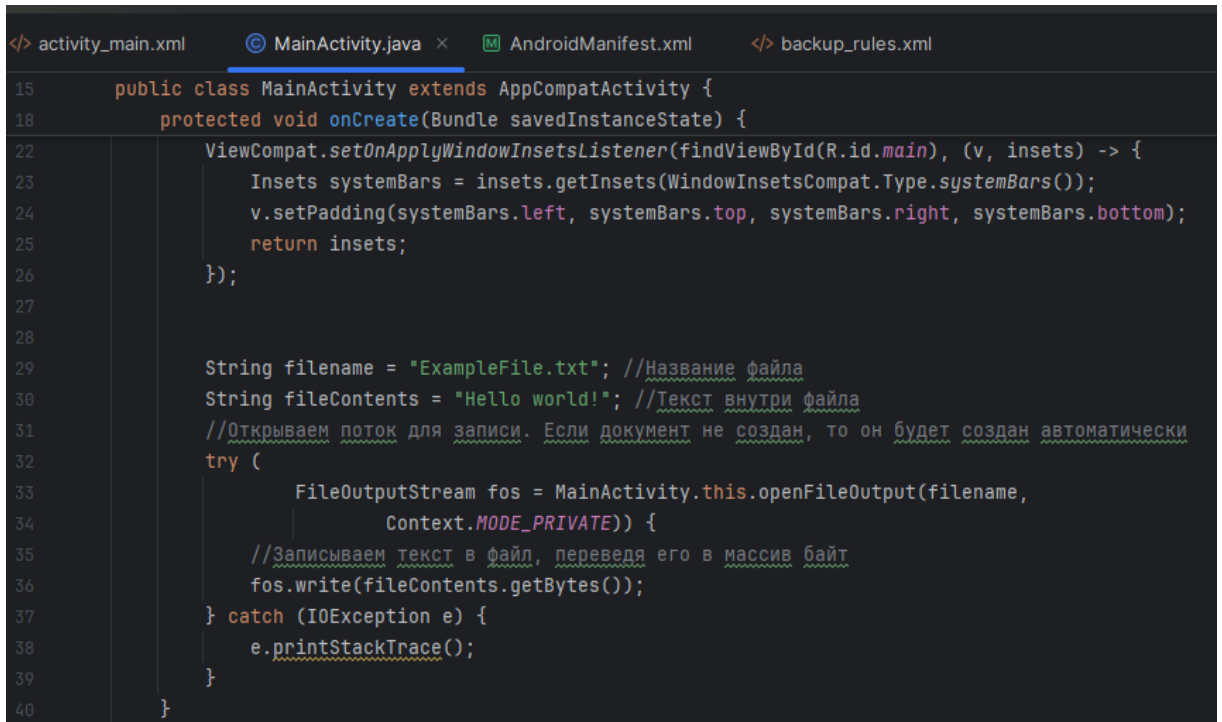
Для создания файла требуется открыть поток для его записи/чтения.

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.

Если файл уже существует, то он будет перезаписан. Если же нам надо дописать файл, тогда надо использовать режим **MODE\_APPEND**. Для

автоматического закрытия файла и освобождения ресурса объект `FileOutputStream` создается с помощью конструкции `try...catch` (Рисунок 5).



```
15 public class MainActivity extends AppCompatActivity {
18     protected void onCreate(Bundle savedInstanceState) {
22         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
23             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
24             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
25             return insets;
26         });
27
28
29         String filename = "ExampleFile.txt"; //Название файла
30         String fileContents = "Hello world!"; //Текст внутри файла
31         //Открываем поток для записи. Если документ не создан, то он будет создан автоматически
32         try {
33             FileOutputStream fos = MainActivity.this.openFileOutput(filename,
34                 Context.MODE_PRIVATE);
35             //Записываем текст в файл, переводя его в массив байт
36             fos.write(fileContents.getBytes());
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40     }
```

Рисунок 5 – Создание текстового файла

В итоге весь текст будет сохранен в файле `/data/data/название_пакета/files/ExampleFile.txt`

И если дважды нажать на файл, то можно увидеть его содержимое (Рисунок 6).

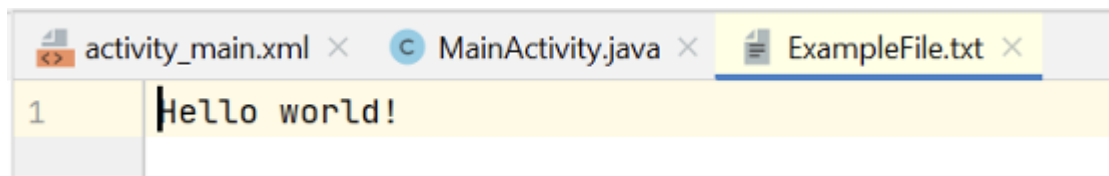


Рисунок 6 – Содержимое файла `ExampleFile.txt`

### 1.3.2 Чтение файла

Класс `FileInputStream` создаёт объект класса `InputStream`, который можно использовать для чтения байтов из файла.

`InputStreamReader` – это переход от потоков байтов к потокам символов: он считывает байты и декодирует их в символы, используя указанный `Charset`.

Используемая им кодировка может быть указана по имени или может быть задана явно, или может быть принята кодировка платформы default charset (Рисунок 7).

```
1 usage
public void read_file(View v){
    String file_name = file_name_text.getText().toString();
    try (FileInputStream fis = MainActivity.this.openFileInput(file_name)){
        InputStreamReader inputStreamReader = new InputStreamReader(fis,
            StandardCharsets.UTF_8);
        StringBuilder stringBuilder = new StringBuilder();
        try (BufferedReader reader = new BufferedReader(inputStreamReader)){
            String line = reader.readLine();
            while (line != null){
                stringBuilder.append(line).append('\n');
                line = reader.readLine();
            }
        } catch (IOException e){
            e.printStackTrace();
        }
        String contents = stringBuilder.toString();
        file_read_data.setText(contents);
    } catch (IOException e){
        e.printStackTrace();
    }
}
```

Рисунок 7 – Реализация чтения текста из файла

### 1.3.3 Удаление файла

Для того чтобы удалить файл необходимо обратиться к его директории, а после провести удаление самого файла (Рисунок 8).

```
1 usage
public void delete_file(){
    String file_name = file_name_text.getText().toString();
    File dir = getFilesDir();
    File file = new File(dir, file_name);
    boolean deleted = file.delete();
    Toast.makeText(context: this, text: deleted + "", Toast.LENGTH_LONG).show();
}
```

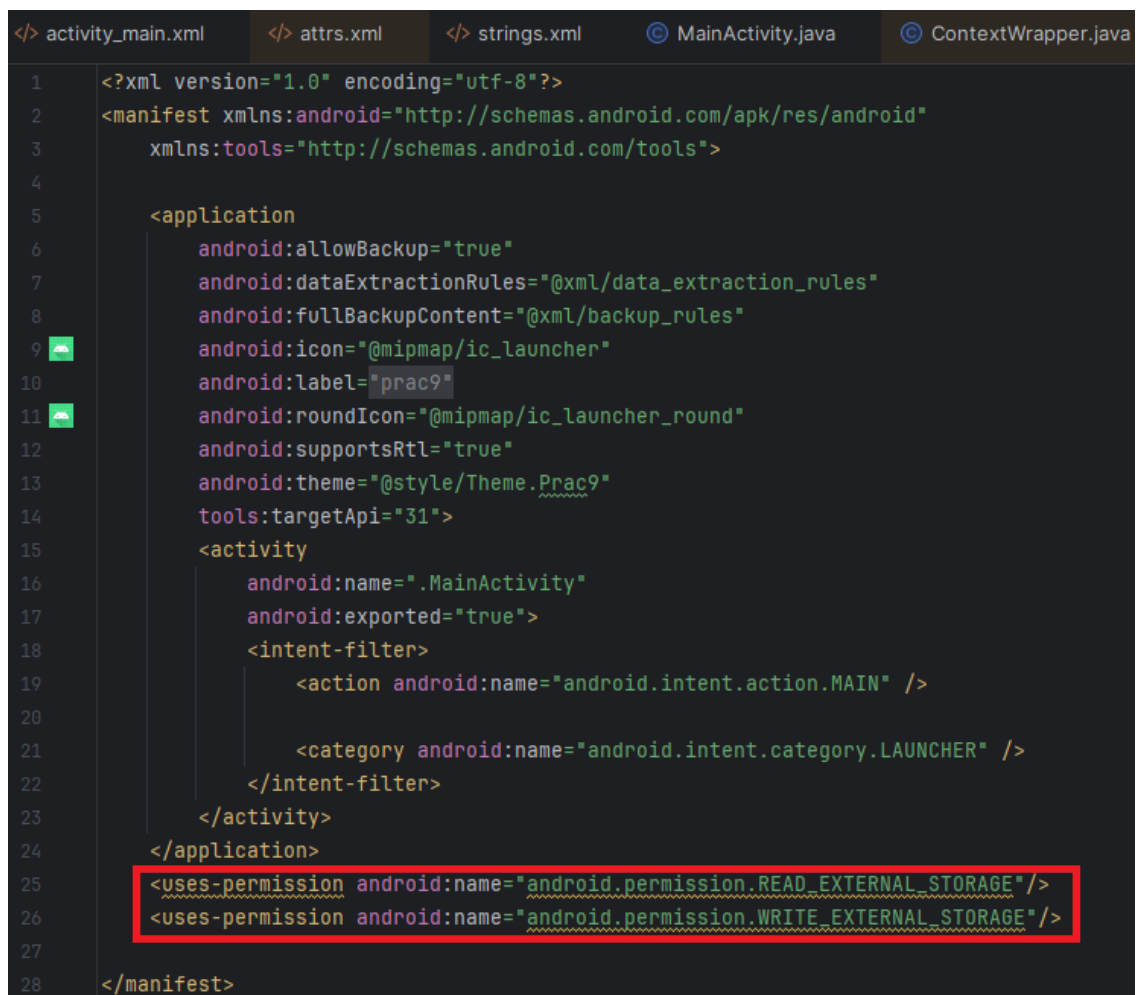
Рисунок 8 – Реализация удаления файла



## 1.4 Внешнее хранилище

В прошлой теме были рассмотрены сохранение и чтение файлов из каталога приложения. По умолчанию такие файлы доступны только самому приложению.

Внешнее хранилище используется для сохранения файлов, которые могут быть общими для нескольких приложений или требуют сохранения даже после удаления вашего приложения. Прежде чем создавать файлы, убедитесь, что у вашего приложения есть необходимые разрешения для работы с внешним хранилищем. Начиная с Android 6.0 (API уровня 23), требуется запросить эти разрешения во время выполнения (для этого добавьте в файл Манифеста разрешения READ\_EXTERNAL\_STORAGE и WRITE\_EXTERNAL\_STORAGE) (Рисунок 9).



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/prac9"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Prac9"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

</manifest>
```

Рисунок 9 – Добавление разрешений на чтение и запись файлов внешнего хранилища

Однако способ, представленный на рисунке 9 не предоставляет доступ к файлам внешнего хранилища, начиная с версии API 29, что довольно печально.

Для Android 10 и выше рекомендуется использовать механизм Scoped Storage, который не требует явных разрешений для доступа к определенным типам файлов, таким как фотографии и видео, через MediaStore API.

### 1.4.1 Создание и запись файла

Для создания файла в первую очередь необходимо удостовериться в существовании заданного репозитория, в котором хранится выбранный файл, после чего происходит создание файла с записью или без (Рисунок 10).

```
public void save_file(View v){
    String file_name = file_name_text.getText().toString();
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    if (!dir.exists()){
        dir.mkdirs();
    }
    File file = new File(dir, file_name);
    try{
        if (!file.exists()){
            boolean created = file.createNewFile();
            if (created){
                FileWriter writer = new FileWriter(file);
                writer.append(file_data_text.getText().toString());
                writer.flush();
                writer.close();
            }
        }
        else{
            boolean readable = file.canRead();
            boolean writeable = file.canWrite();
        }
    }catch (IOException e){
        e.printStackTrace();
    }
}

1 usage
public void add_data(View v) throws IOException {
    String file_name = file_name_text.getText().toString();
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);

    if (!dir.exists()){
        dir.mkdirs();
    }

    File file = new File(dir, file_name);
    FileOutputStream fos = new FileOutputStream(file);
    fos.write(file_data_text.getText().toString().getBytes());
    fos.close();
}
```

Рисунок 10 – Реализация создания файла и записи в него текста во внешнем хранилище

### 1.4.2 Чтение файла

Для чтения файла необходимо убедиться в его существовании, после чего провести попытку получения доступа к чтению файла и в случае удовлетворительного результата воспроизвести чтение файла (Рисунок 11).

```
public void read_file(View v){
    String file_name = file_name_text.getText().toString();
    File dir = getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    File file = new File(dir, file_name);

    if (file.exists()) {
        StringBuilder text = new StringBuilder();

        try {
            boolean readable = file.canRead();
            BufferedReader br = new BufferedReader(new FileReader(file));
            String line;
            while ((line = br.readLine()) != null) {
                text.append(line).append('\n');
            }
            file_read_data.setText(text);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Рисунок 11 – Реализация чтения файла из внешнего хранилища

### 1.4.3 Удаление файла

Для удаления файла необходимо обратиться к его родительской директории, а после к нему самому для его непосредственного удаления (Рисунок 12).

```
1 usage
public void delete_file(){
    String file_name = file_name_text.getText().toString();
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    File file = new File(dir, file_name);
    boolean deleted = file.delete();
    Toast.makeText(context: this, text: deleted + "", Toast.LENGTH_LONG).show();
}
```

Рисунок 12 – Реализация удаления файла из внешнего хранилища

## 2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

### 2.1 Внутреннее хранилище

#### 2.1.1 Разметка

После создания проекта подготовим файл разметки `activity_main.xml` для работы с внутренней памятью.

Добавим 2 поля для ввода текста, первое – для ввода названия файла, а второе – для ввода содержимого (Рисунок 13).

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="vertical">
    <EditText
        android:id="@+id/edit_main_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Название файла" />
    <EditText
        android:id="@+id/edit_main_contents"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Записать в файл" />
</LinearLayout>
```

Рисунок 13 – Текстовые поля ввода в файле разметки `activity_main.xml`

Добавим 4 кнопки для работы с файлами, первая – для создания файла, вторая – для чтения файла, третья – для удаления файла, четвертая – для записи текста в конец файла (Рисунок 14).

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center">

    <Button
        android:id="@+id/button_main_create"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Создать"/>

    <Button
        android:id="@+id/button_main_read"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Прочитать"/>

    <Button
        android:id="@+id/button_main_delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Удалить"/>

</LinearLayout>

<Button
    android:id="@+id/button_main_append"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:text="Записать"/>

```

Рисунок 14 – Кнопки в файле разметки activity\_main.txt

Добавим текстовое поле для вывода содержимого файлов (Рисунок 15).

```

<TextView
    android:id="@+id/text_main_contents"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:text="Содержимое файла" />

```

Рисунок 15 – Текстовое поле в файле разметки activity\_main.xml

Отображение файла разметки представлено на рисунке 16.

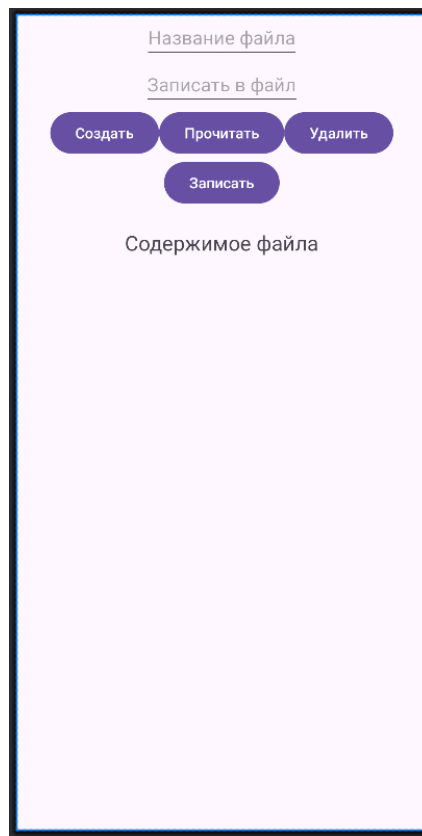


Рисунок 16 – Отображение файла разметки activity\_main.xml

### 2.1.2 Реализация работы с файлами

Для создания файла был написан метод `createFile()`, принимающий название файла и содержимое при создании и создает файл. В случае ошибки при создании выводится диалоговое окно об ошибке (Рисунок 17).

```
private void createFile(String fileName, byte[] contents)
{
    try (FileOutputStream out = MainActivity.this.openFileOutput(fileName, MODE_PRIVATE))
    {
        out.write(contents);
    }
    catch (IOException e)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Не удалось создать файл")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
    }
}
```

Рисунок 17 – Описание метода `createFile()` класса `MainActivity`

Для чтения файла был создан метод `readFile()`, принимающий название файла и возвращающий `String` представление содержимого файла. В случае ошибки при чтении или при отсутствии файла выводит диалоговое окно об ошибке (Рисунок 18).

```
private String readFile(String fileName)
{
    try (FileInputStream in = MainActivity.this.openFileInput(fileName))
    {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(in, StandardCharsets.UTF_8));
        return reader.lines().collect(Collectors.joining(delimiter: "\n"));
    }
    catch (IOException e)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Не удалось прочитать файл")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
        return "";
    }
}
```

Рисунок 18 – Описание метода `readFile()` класса `MainActivity`

Для удаления файла был создан метод `deleteFileAlert()`, принимающий название файла. Перед удалением файла на экран выводится диалоговое окно, спрашивающее пользователя, точно ли он хочет удалить файл (Рисунок 19).

```
private void deleteFileAlert(String fileName)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
    builder
        .setTitle("Вы точно хотите удалить файл " + fileName + "?")
        .setPositiveButton(text: "Да", (dialog, which) -> deleteFile(fileName))
        .setNegativeButton(text: "Отмена", (dialog, which) -> { })
        .show();
}
```

Рисунок 19 – Описание метода `deleteFileAlert()` класса `MainActivity`

Для записи текста в файл был создан метод `appendToFile()`, принимающий название файла и текст для записи. При отсутствии файла или возникновении ошибки при записи на экран выводится диалоговое окно об ошибке (Рисунок 20).

```

private void appendToFile(String fileName, byte[] contents)
{
    File file = MainActivity.this.getFileStreamPath(fileName);
    if (!file.exists())
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Файл не найден")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
        return;
    }
    try (FileOutputStream out = MainActivity.this.openFileOutput(fileName,
        mode: MODE_PRIVATE | MODE_APPEND))
    {
        out.write(contents);
    }
    catch (IOException e)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Не удалось изменить файл")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
    }
}

```

Рисунок 20 – Описание метода appendToFile() класса MainActivity

Переопределим метод onCreate() класса MainActivity для привязки нажатия кнопок к описанным методам (Рисунок 21).

```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) ->
    {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });

    EditText editFileName = findViewById(R.id.edit_main_name);
    EditText editFileContents = findViewById(R.id.edit_main_contents);

    Button buttonCreate = findViewById(R.id.button_main_create);
    buttonCreate.setOnClickListener(v -> createFile(editFileName.getText().toString(),
        editFileContents.getText().toString().getBytes(StandardCharsets.UTF_8)));

    TextView textContents = findViewById(R.id.text_main_contents);
    Button buttonRead = findViewById(R.id.button_main_read);
    buttonRead.setOnClickListener(v -> textContents.setText(
        readFile(editFileName.getText().toString())));

    Button buttonDelete = findViewById(R.id.button_main_delete);
    buttonDelete.setOnClickListener(v -> deleteFileAlert(editFileName.getText().toString()));

    Button buttonAppend = findViewById(R.id.button_main_append);
    buttonAppend.setOnClickListener(v -> appendToFile(editFileName.getText().toString(),
        editFileContents.getText().toString().getBytes(StandardCharsets.UTF_8)));
}

```

Рисунок 21 – Описание метода onCreate() класса MainActivity



### 2.1.3 Сохранение состояния приложения

Для того, чтобы сохранить состояние приложения при повороте, воспользуемся методами `onSaveInstanceState()` и `onRestoreInstanceState()` (Рисунок 22).

```
@Override
public void onSaveInstanceState(@NonNull Bundle outState)
{
    super.onSaveInstanceState(outState);
    TextView textFileContents = findViewById(R.id.text_main_contents);
    outState.putString("ARGUMENT_TEXT_FILE_CONTENTS", textFileContents.getText().toString());
    EditText editFileName = findViewById(R.id.edit_main_name);
    outState.putString("ARGUMENT_EDIT_FILE_NAME", editFileName.getText().toString());
    EditText editFileContents = findViewById(R.id.edit_main_contents);
    outState.putString("ARGUMENT_EDIT_FILE_CONTENTS", editFileContents.getText().toString());
}

@Override
protected void onRestoreInstanceState(@NonNull Bundle savedInstanceState)
{
    super.onRestoreInstanceState(savedInstanceState);
    TextView textFileContents = findViewById(R.id.text_main_contents);
    textFileContents.setText(savedInstanceState.getString(key: "ARGUMENT_TEXT_FILE_CONTENTS"));
    EditText editFileName = findViewById(R.id.edit_main_name);
    editFileName.setText(savedInstanceState.getString(key: "ARGUMENT_EDIT_FILE_NAME"));
    EditText editFileContents = findViewById(R.id.edit_main_contents);
    editFileContents.setText(savedInstanceState.getString(key: "ARGUMENT_EDIT_FILE_CONTENTS"));
}
```

Рисунок 22 – Описание методов `onSaveInstanceState()` и `onRestoreInstanceState()` класса `MainActivity`

### 2.1.4 Тестирование

Проверим работу приложения. Сначала откроем приложение (Рисунок 23).



Рисунок 23 – Начальный экран приложения

Создадим файл с названием file.txt и прочитаем его (Рисунок 24).



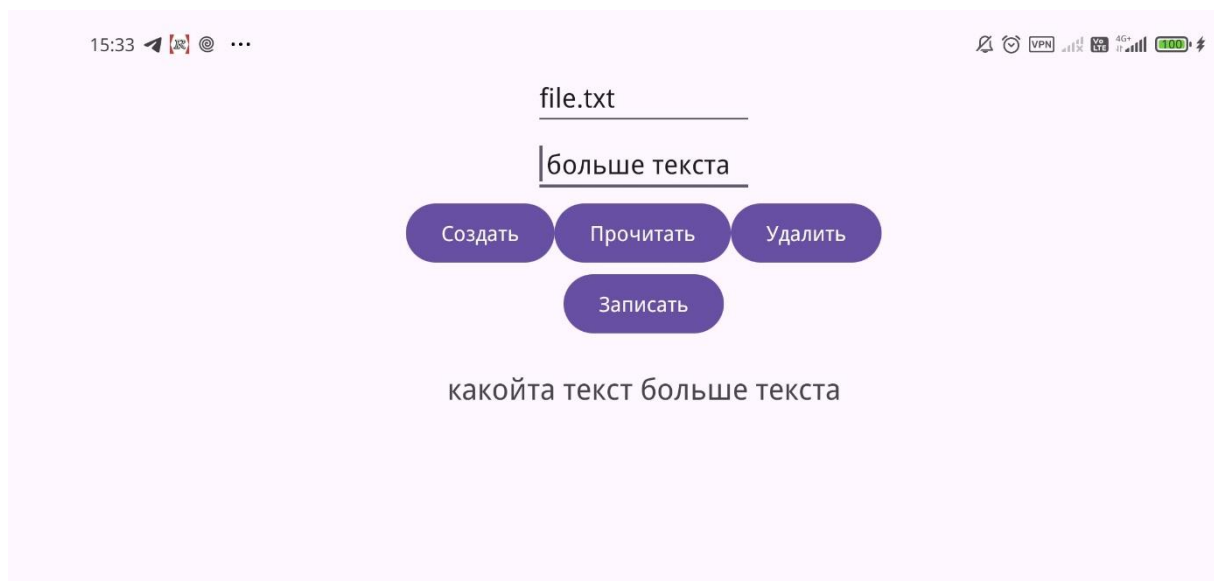
**Рисунок 24 – Создание файла внутреннего хранилища**

Запишем текст в файл и опять прочитаем его содержимое (Рисунок 25).



**Рисунок 25 – Запись текста в файл внутреннего хранилища**

Попробуем повернуть приложение для проверки, что все данные сохраняются (Рисунок 26).



**Рисунок 26 – Сохранение данных при повороте телефона**

Попробуем удалить файл. При удалении появляется диалоговое окно с подтверждения (Рисунок 27). Файл удаляется успешно.



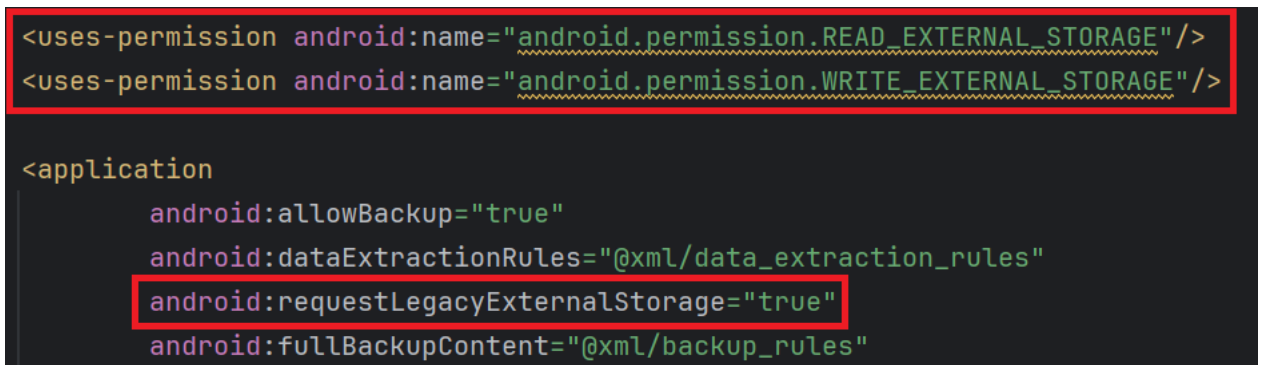
**Рисунок 27 – Диалоговое окно подтверждения при попытке удаления файла внутреннего хранилища**

## 2.2 Работа с внешней памятью

Создадим второй проект для второго приложения. В нем мы будем считывать созданный в первом приложении файл внешнего хранилища.

### 2.2.1 Разрешения

Перед выполнением задания предоставим обоим приложениям разрешения на чтение и запись во внешнюю память. Так как приложение будет тестироваться на устройстве Android версии 10, необходимо также указать атрибут `requestLegacyExternalStorage="true"` (Рисунок 28).



```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:requestLegacyExternalStorage="true"
    android:fullBackupContent="@xml/backup_rules"
```

Рисунок 28 – Предоставление необходимых разрешений

### 2.2.2 Разметка

Оставим файл разметки первого приложения неизменным.

Заполним файл разметки `activity_main.xml` второго приложения текстовым полем ввода для ввода названия файла, кнопкой для чтения файла и текстовым полем для вывода содержимого файла (Рисунок 29).

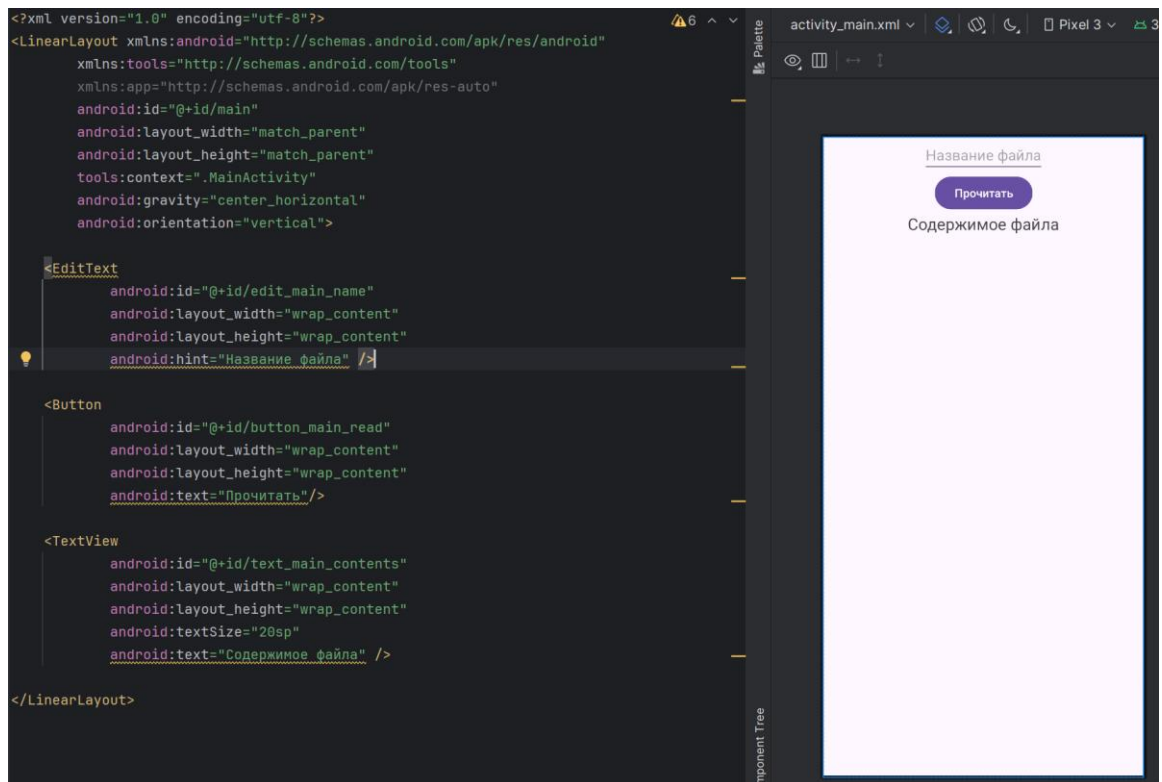


Рисунок 29 – Файл разметки activity\_main.xml второго приложения

### 2.2.3 Реализация

Оставим метод onCreate() класса MainActivity первого приложения неизменным, однако немного изменим каждый из методов работы с файлами.

На рисунках 30-32 изображено описание методов createFile(), readFile() и deleteFileAlert() соответственно.

```
private void createFile(String fileName, byte[] contents)
{
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    File file = new File(dir, fileName);
    try (FileOutputStream out = new FileOutputStream(file))
    {
        out.write(contents);
    }
    catch (IOException e)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Не удалось создать файл")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
    }
}
```

Рисунок 30 – Описание метода createFile() класса MainActivity для внешнего хранилища

```

private String readFile(String fileName)
{
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    File file = new File(dir, fileName);
    if (!file.exists())
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Файл не найден")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
        return "";
    }
    try (FileInputStream in = new FileInputStream(file))
    {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(in, StandardCharsets.UTF_8));
        return reader.lines().collect(Collectors.joining(delimiter: "\n"));
    }
    catch (IOException e)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
        builder
            .setTitle("Не удалось прочитать файл")
            .setPositiveButton(text: "Ок", (dialog, which) -> { })
            .show();
        return "";
    }
}

```

**Рисунок 31 – Описание метода readFile() класса MainActivity для внешнего хранилища**

```

private void deleteFileAlert(String fileName)
{
    File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
    File file = new File(dir, fileName);
    AlertDialog.Builder builder = new AlertDialog.Builder(context: this);
    builder
        .setTitle("Вы точно хотите удалить файл " + fileName + "?")
        .setPositiveButton(text: "Да", (dialog, which) -> file.delete())
        .setNegativeButton(text: "Отмена", (dialog, which) -> { })
        .show();
}

```

**Рисунок 32 – Описание метода deleteFileAlert() класса MainActivity для внешнего хранилища**

В классе MainActivity второго приложения определим такой же как и в первом приложении метод readFile(), а также переопределим метод onCreate() для привязки нажатия кнопки к чтению файла (Рисунок 33).

```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) ->
    {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });

    EditText editFileName = findViewById(R.id.edit_main_name);

    TextView textContents = findViewById(R.id.text_main_contents);
    Button buttonRead = findViewById(R.id.button_main_read);
    buttonRead.setOnClickListener(v -> textContents.setText(
        readFile(editFileName.getText().toString())));
}

```

Рисунок 33 – Описание метода onCreate() класса MainActivity второго приложения

## 2.2.4 Тестирование

Функционал первого приложения визуально не отличается от того, что было протестировано в разделе 2.1.4, но файлы записываются во внешнее хранилище, а не во внутреннее.

В первом приложении создадим файл "pract9.txt" и запишем в него текст "данные для чтения" (Рисунок 34).



Рисунок 34 – Создание файла во внешнем хранилище из первого приложения

Откроем второе приложение (Рисунок 35).



**Рисунок 35 – Начальный экран второго приложения**

Введем название файла "pract9.txt" и нажмем кнопку (Рисунок 36).



**Рисунок 36 – Чтение файла внешнего хранилища из второго приложения**

Было считано ранее записанное содержимое.



## **ЗАКЛЮЧЕНИЕ**

В ходе данной практической работы были получены знания по обработке файлов во внешнем и внутреннем хранилищах, а также способ сохранения состояния приложения при смене его ориентации. Полученные знания были закреплены путём выполнения практического задания.