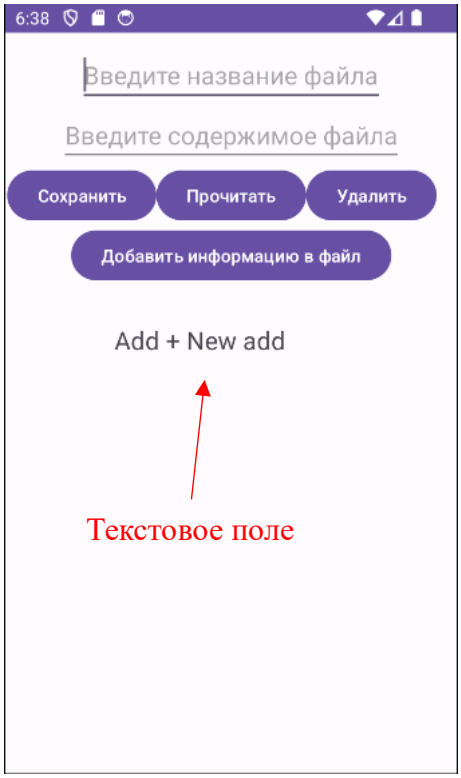
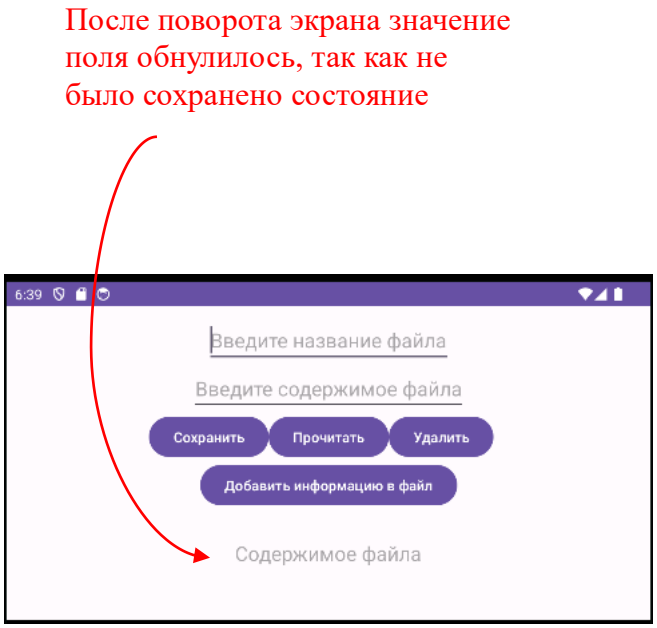
Часть 1. Сохранение состояния приложения

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 активности.





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. В этот объект можно добавлять различные типы данных, такие как строки, числа, сериализуемые объекты и др.

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

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().

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

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

Часть 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 и внешнее.** 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.

Часть 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 следующие методы:

1) Создание и запись файла:

```
    String filename = "ExampleFile.txt"; //Название файла
    String fileContents = "Hello world!"; //Текст внутри файла
    //Открываем поток для записи. Если документ не создан, то он будет создан автоматически
    try (FileOutputStream fos = context.openFileOutput(filename, Context.MODE_PRIVATE)) {
    //Записываем текст в файл, переведя его в массив байт
        fos.write(fileContents.getBytes());
    } catch (IOException e) {
        e.printStackTrace();
    }
```

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.

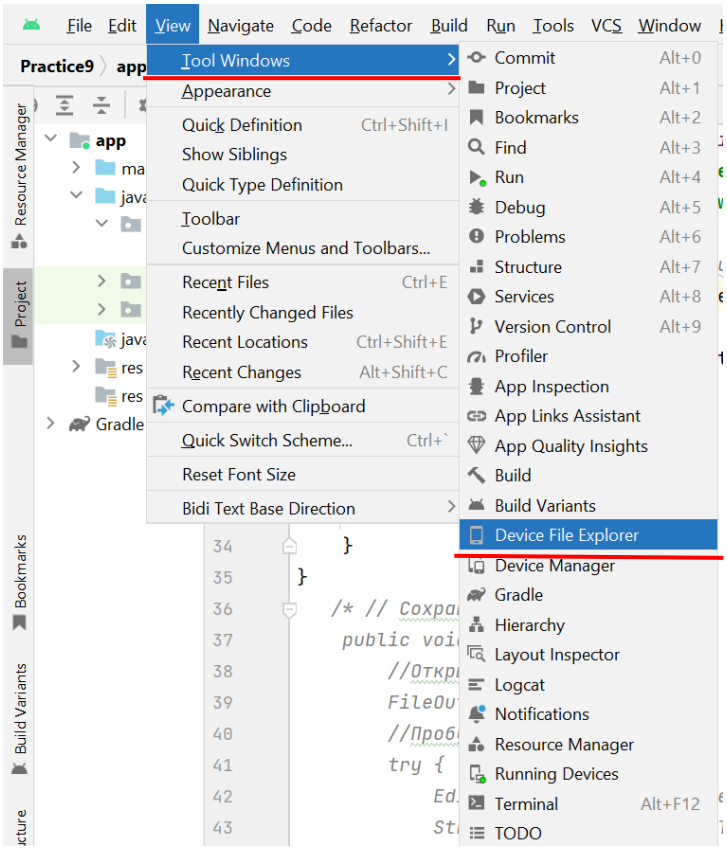
Если файл уже существует, то он будет перезаписан. Если же нам надо дописать файл, тогда надо использовать режим MODE_APPEND.

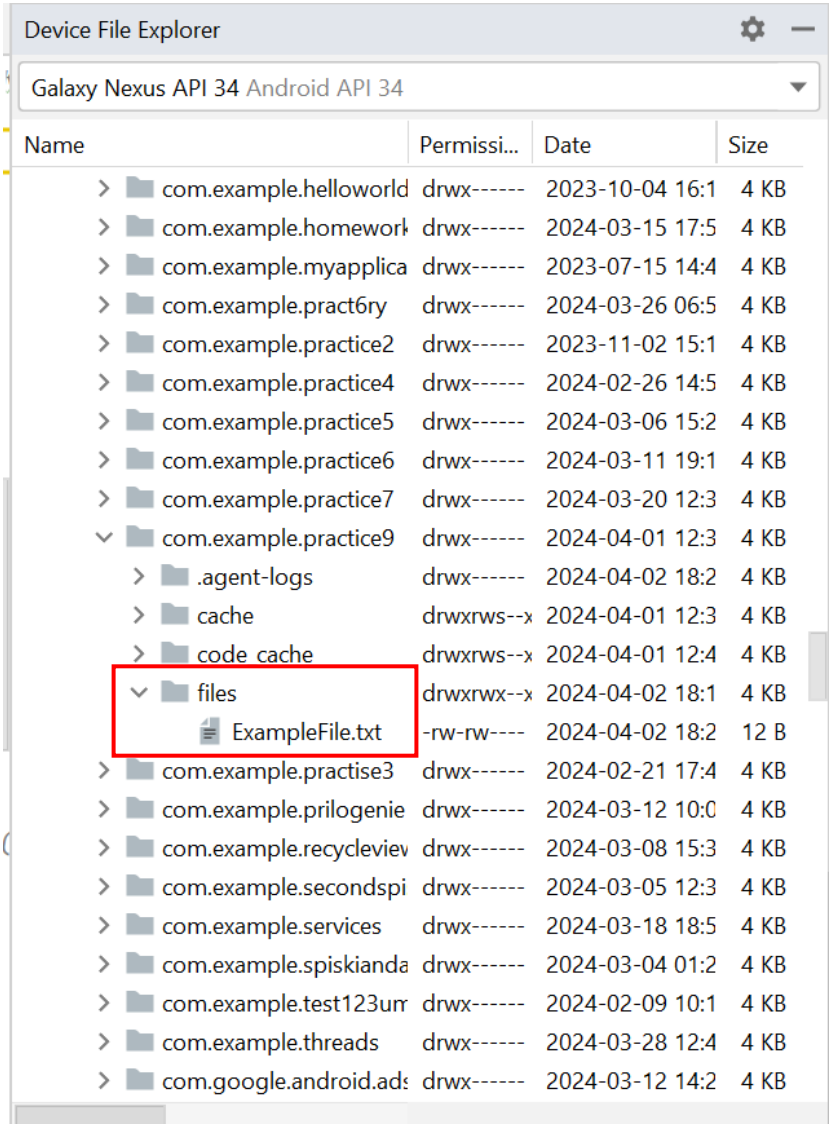
Для автоматического закрытия файла и освобождения ресурса объект FileOutputStream создается с помощью конструктции **try...catch.**

```
String filename = "ExampleFile.txt"; //Название файла
String fileContents = "Hello world!"; //Текст внутри файла
try(
    //Открываем поток для записи и
    FileOutputStream fos = context.openFileOutput(filename, Context.MODE_PRIVATE))
{   //Записываем текст внутрь файла, переведя его в массив байт
    fos.write(fileContents.getBytes());
} catch(
    IOException e)
{
    e.printStackTrace();
}
```

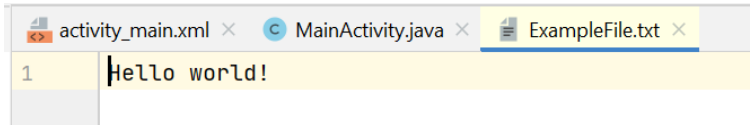
В итоге весь текст будет сохранен в файле **/data/data/название_пакета/files/ExampleFile.txt**

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**





И если дважды нажать на файл, то можно увидеть его содержимое.



2) Чтение файла:

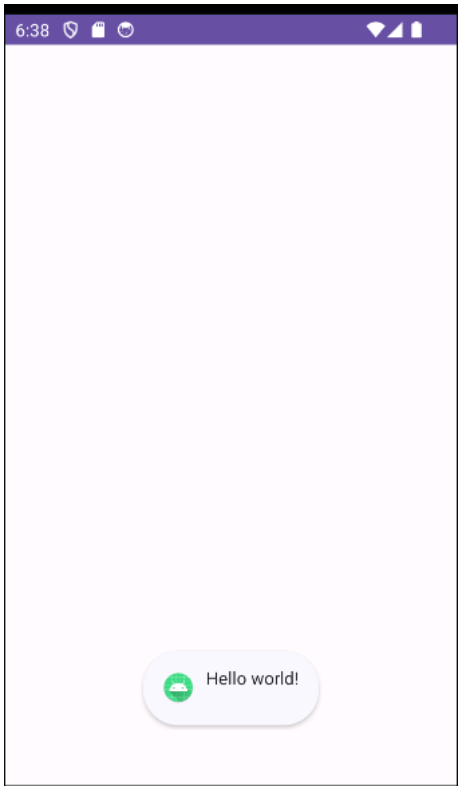
```
    try (FileInputStream fis = context.openFileInput(filename)) {
        InputStreamReader inputStreamReader = new
InputStreamReader(fis, StandardCharsets.UTF_8);
        StringBuilder stringBuilder = new StringBuilder();
        try (BufferedReader reader = new
BufferedReader(inputStreamReader)) {
            String line = reader.readLine();
            while (line != null) {
                stringBuilder.append(line).append('\n');
                line = reader.readLine();
            }
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
        String contents = stringBuilder.toString();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

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.

```
String filename = "ExampleFile.txt"; //Название файла
//Открываем поток для чтение и передаем название файла, который будем читать
try (FileInputStream fis = context.openFileInput(filename)) {
    InputStreamReader inputStreamReader = new InputStreamReader(fis, StandardCharsets.UTF_8);
    //Создаем конструктор строк
    StringBuilder stringBuilder = new StringBuilder();
    try (BufferedReader reader = new BufferedReader(inputStreamReader)) {
        //Считывает строку текста
        String line = reader.readLine();
        //Пока считанная строка не пустая, то добавляем строку в констуктор и читаем следующую
        while (line != null) {
            stringBuilder.append(line).append('\n');
            line = reader.readLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    String contents = stringBuilder.toString();
    Toast.makeText( context: MainActivity.this, contents,Toast.LENGTH_LONG).show();
} catch (IOException e) {
    e.printStackTrace();
}
```

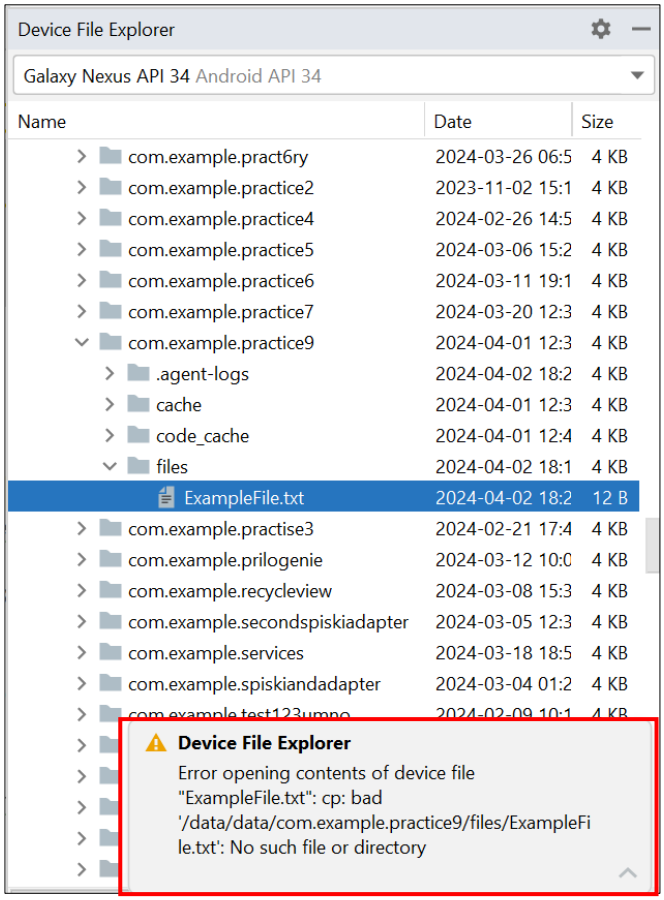


3) Удаление файла:

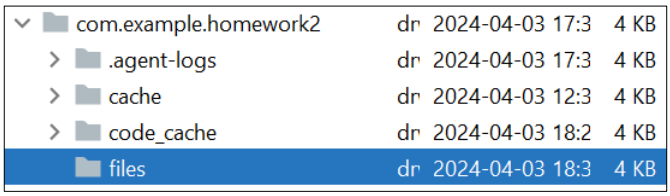
```
// Указываем имя файла, который хотим удалить
String filename = "ExampleFile.txt";
// Получаем файловый объект для файла из внутреннего хранилища
File dir = getFilesDir();
File file = new File(dir, filename);
// Удаляем файл
```

```
boolean deleted = file.delete();
```

Теперь при попытке открыть файл будет выводиться ошибка. После синхронизации папка с файлами будет пуста. Для синхронизации нажмите правой кнопкой мыши на папку **files → Synchronize.**



После синхронизации



Часть 4. Внешнее хранилище

В прошлой теме мы рассмотрели сохранение и чтение файлов из каталога приложения. По умолчанию такие файлы доступны только самому приложения.

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**).

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

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) Создание и запись файла:

```
    // Пример создания текстового файла в публичной директории
"Documents"
    File storageDir =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DO
CUMENTS);

    if (!storageDir.exists()) {
        storageDir.mkdirs(); // Создаем директорию, если она не
существует
    }
    File file = new File(storageDir, "example.txt");
    try {
        if (!file.exists()) {
            boolean created = file.createNewFile(); // Создаем файл,
если он не существует
            if (created) {
                // Записываем данные в файл
                FileWriter writer = new FileWriter(file);
                writer.append("Hello, world!");
                writer.flush();
                writer.close();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

2) Чтение файла:

```
    File storageDir =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DO
CUMENTS);
    File file = new File(storageDir, "example.txt");
```

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

        try {
            BufferedReader br = new BufferedReader(new
FileReader(file));
            String line;

            while ((line = br.readLine()) != null) {
                text.append(line);
                text.append('\n');
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        String fileContent = text.toString();
        // Используйте содержимое файла как необходимо
    }
```

3) Удаление файла:

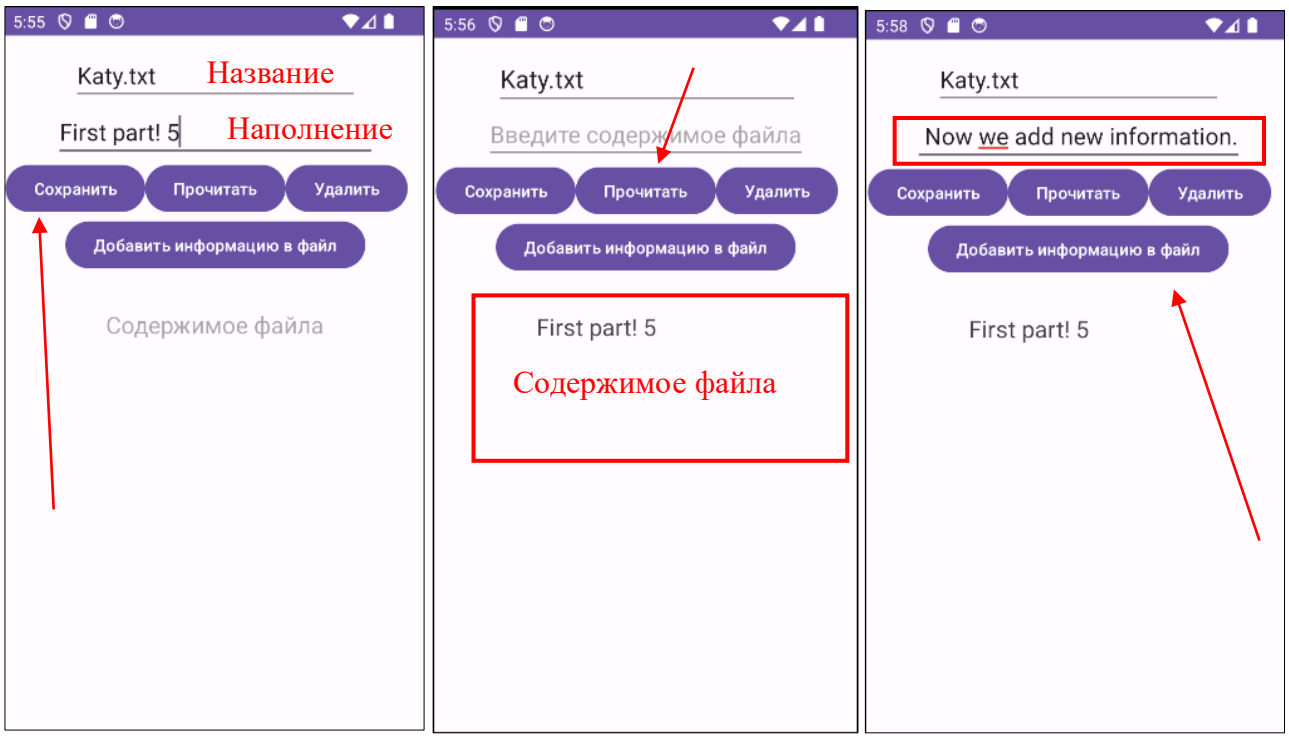
```
    File storageDir =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DO
CUMENTS);
    File file = new File(storageDir, "example.txt");

    if (file.exists()) {
        boolean deleted = file.delete();
        if (deleted) {
            // Файл успешно удален
Toast.makeText(MainActivity.this,"File deleted",
Toast.LENGTH_LONG).show();
        } else {
            // Не удалось удалить файл
        }
    }
```

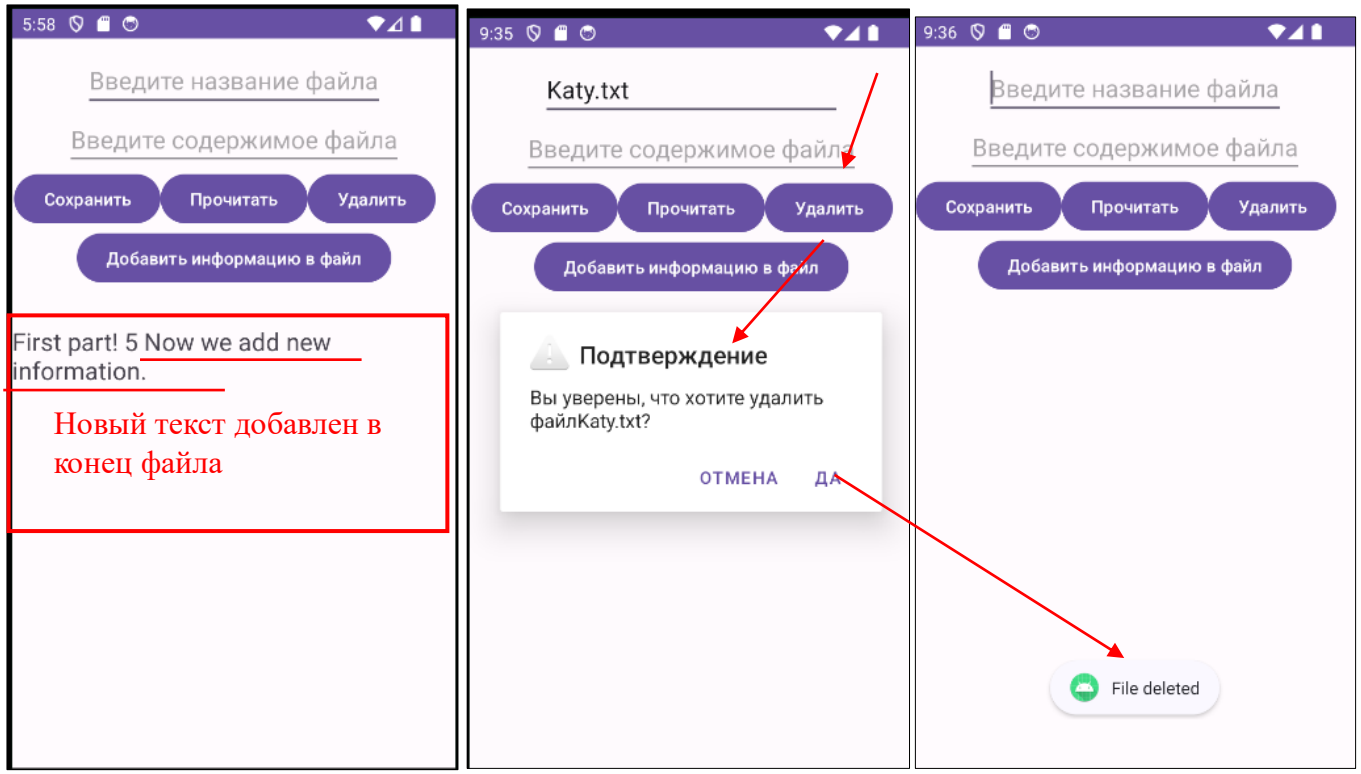
Задание

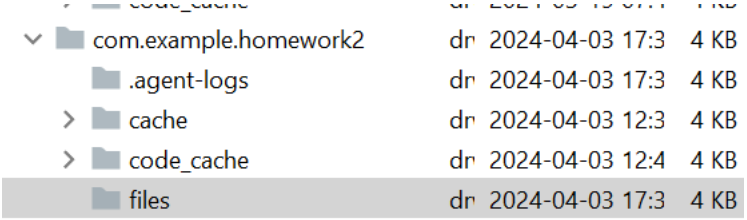
1. На экране сделать несколько полей: поле, в котором будет прописано название создаваемого файла (EditText), поле для ввода содержимого файла (EditText). Также должны быть реализованы кнопки для создания файла, записи дополнительной информации в конец существующего файла, кнопка для чтения файла, а также для удаления файла.

При чтении файла в поле для вывода информации должно выводиться содержимое хранящегося файла. После должно быть выведено сообщение о том, что файл удален. При попытке удалить файл должен быть вызван AlertDialog, который спрашивает, действительно ли я хочу удалить выбранный файл.

ПРИМЕР

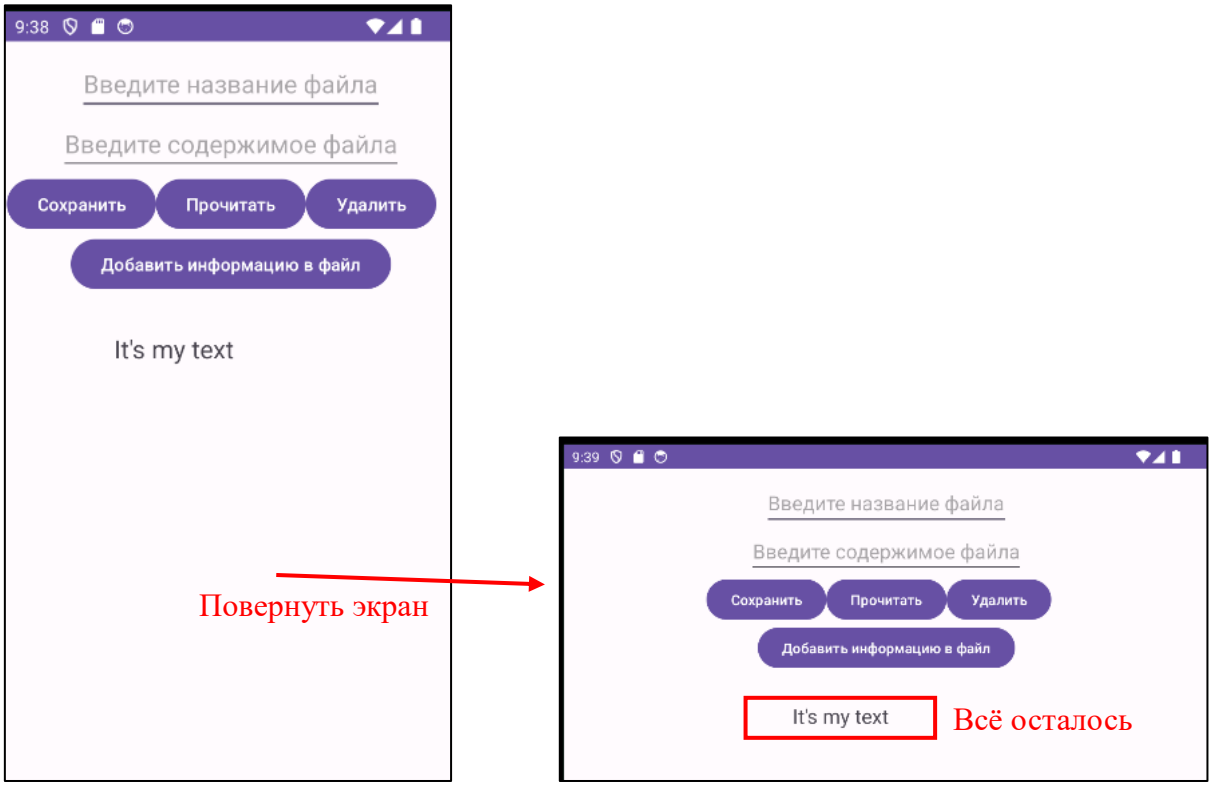






Файл удален из файлов приложения и папка с файлами пустая

2. Реализовать сохранение состояния приложения при повороте экрана.



3. Реализовать создание, запись и удаление во внешней памяти на экране одного приложения и чтение выбранного файла в другом приложении.

Источники

1) https://developer.android.com/reference/android/app/Service
2) https://developer.alexanderklimov.ru/android/theory/data_storage.php
3) https://learntutorials.net/ru/android/topic/150/хранение-файлов-во-внутреннем-и-внешнем-хранилищах