|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 группыИКБО-20-23 | Комисарик М.А. |
| **Проверил:**  Старший преподаватель кафедры МОСИТ | Шешуков Л.С. |

Москва 2025 г.

СОДЕРЖАНИЕ

[**РТУ МИРЭА** 1](#_Toc197129705)

[СОДЕРЖАНИЕ 2](#_Toc197129706)

[1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ 3](#_Toc197129707)

[1.1 Сохранение состояния приложения 3](#_Toc197129708)

[1.2 Хранение данных в Android 4](#_Toc197129709)

[1.3 Внутреннее хранилище 6](#_Toc197129710)

[1.3.1 Создание и запись файла 6](#_Toc197129711)

[1.3.2 Чтение файла 7](#_Toc197129712)

[1.3.3 Удаление файла 8](#_Toc197129713)

[1.4 Внешнее хранилище 9](#_Toc197129714)

[1.4.1 Создание и запись файла 10](#_Toc197129715)

[1.4.2 Чтение файла 11](#_Toc197129716)

[1.4.3 Удаление файла 11](#_Toc197129717)

[2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ 12](#_Toc197129718)

[2.1 Внутреннее хранилище 12](#_Toc197129719)

[2.1.1 Разметка 12](#_Toc197129720)

[2.1.2 Реализация работы с файлами 14](#_Toc197129721)

[2.1.3 Сохранение состояния приложения 17](#_Toc197129722)

[2.1.4 Тестирование 17](#_Toc197129723)

[2.2 Работа с внешней памятью 20](#_Toc197129724)

[2.2.1 Разрешения 20](#_Toc197129725)

[2.2.2 Разметка 20](#_Toc197129726)

[2.2.3 Реализация 21](#_Toc197129727)

[2.2.4 Тестирование 23](#_Toc197129728)

[ЗАКЛЮЧЕНИЕ 25](#_Toc197129729)

# ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

## 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).

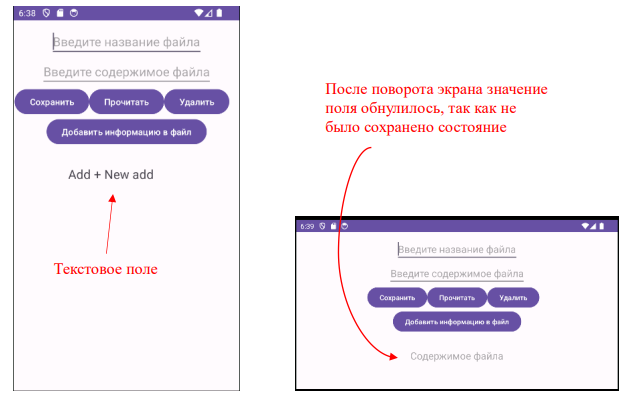


Рисунок 1 – Потеря введенных данных при повороте экрана

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.

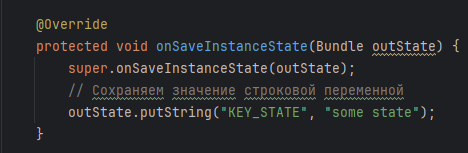


Рисунок 2 – Переопределение метода onSaveInstanceState()

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).

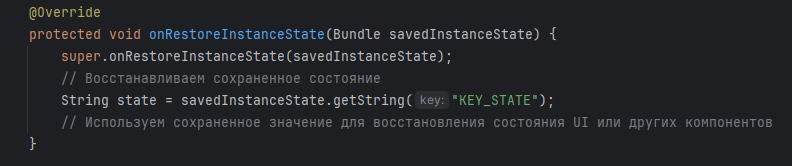


Рисунок 3 – Переопределение метода onRestoreInstanceState()

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

## 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.

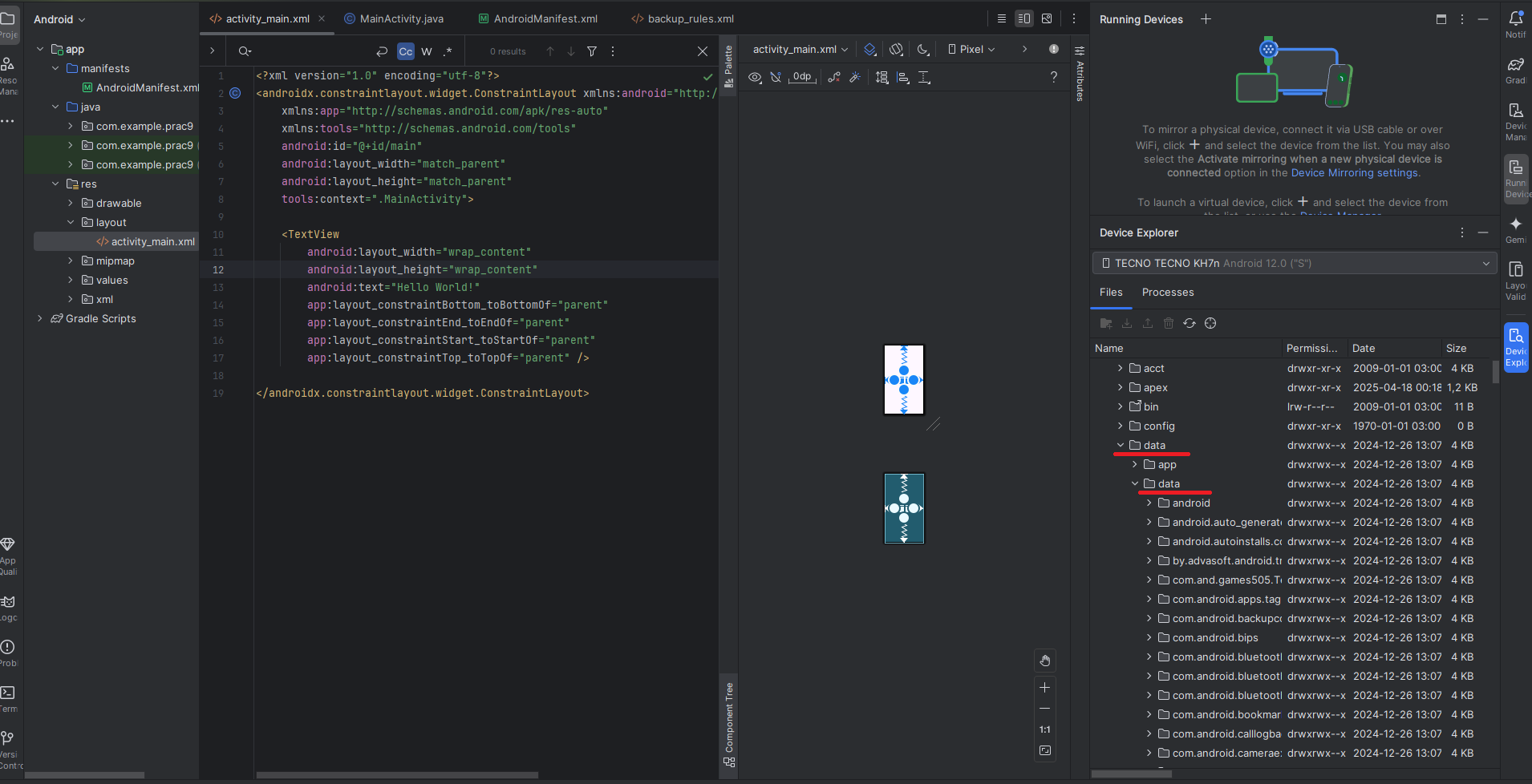


Рисунок 4 – Расположение внутренней памяти в Android

## 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.

### 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 конструктции try...catch (Рисунок 5).

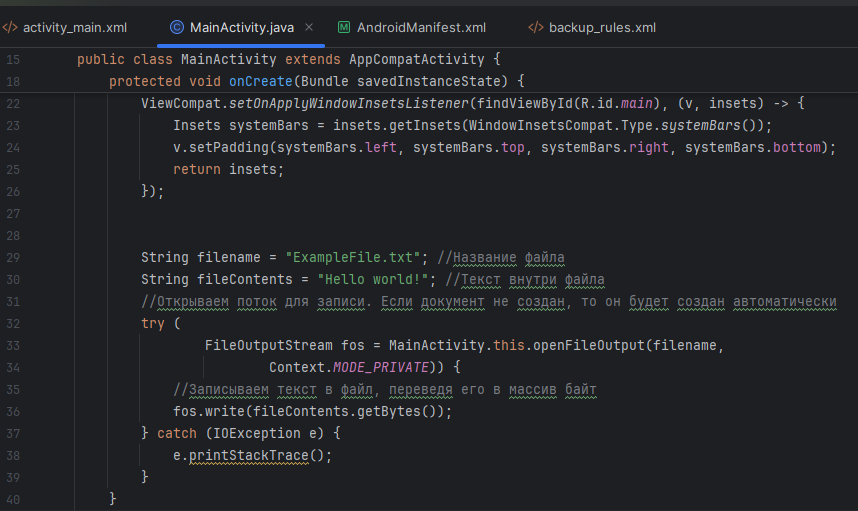


Рисунок 5 – Создание текстового файла

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).

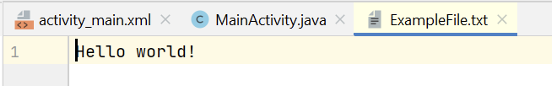


Рисунок 6 – Содержимое файла ExampleFile.txt

### 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).

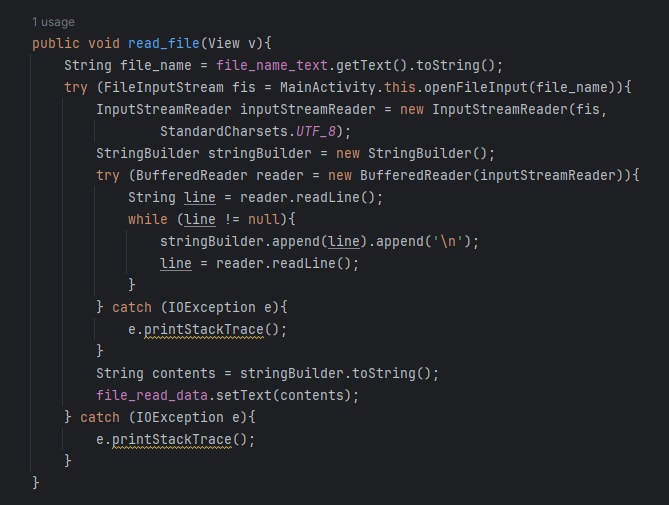


Рисунок 7 – Реализация чтения текста из файла

### Удаление файла

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

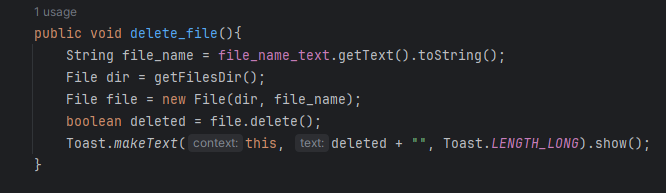


Рисунок 8 – Реализация удаления файла

## Внешнее хранилище

В прошлой теме были рассмотрены 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) (Рисунок 9).

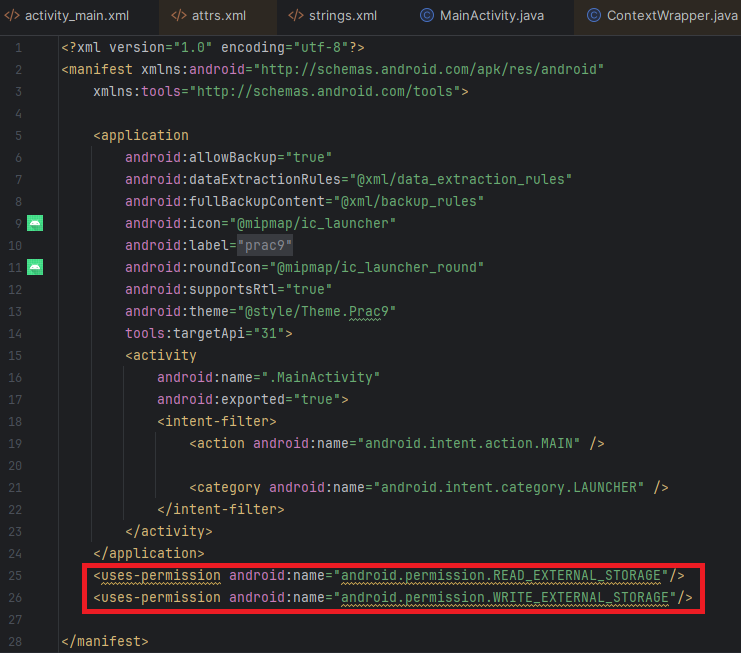


Рисунок 9 – Добавление разрешений на чтение и запись файлов внешнего хранилища

Однако способ, представленный на рисунке Рисунок 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.

### 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).



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

### 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).



Рисунок 11 – Реализация чтения файла из внешнего хранилища

### Удаление файла

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

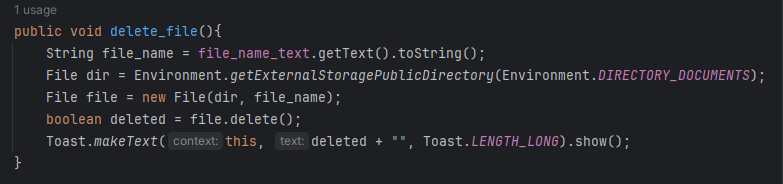


Рисунок 12 – Реализация удаления файла из внешнего хранилища

# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## Внутреннее хранилище

### 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).

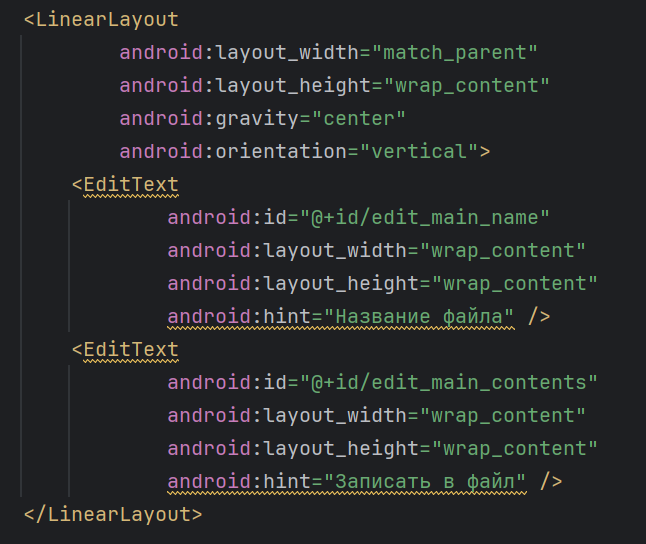


Рисунок 13 – Текстовые поля ввода в файле разметки activity\_main.xml

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).



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

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

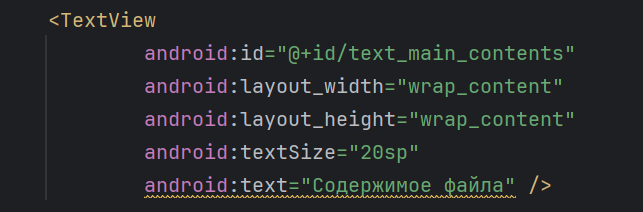


Рисунок 15 – Текстовое поле в файле разметки activity\_main.xml

Отображение файла разметки представлено на рисунке Рисунок 16.

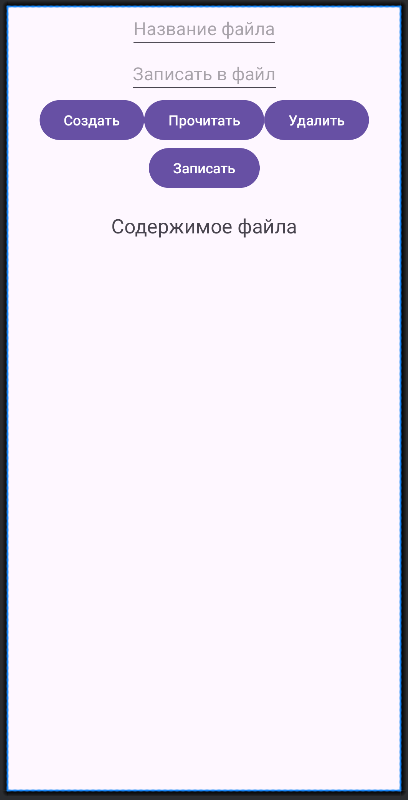


Рисунок 16 – Отображение файла разметки activity\_main.xml

### 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).

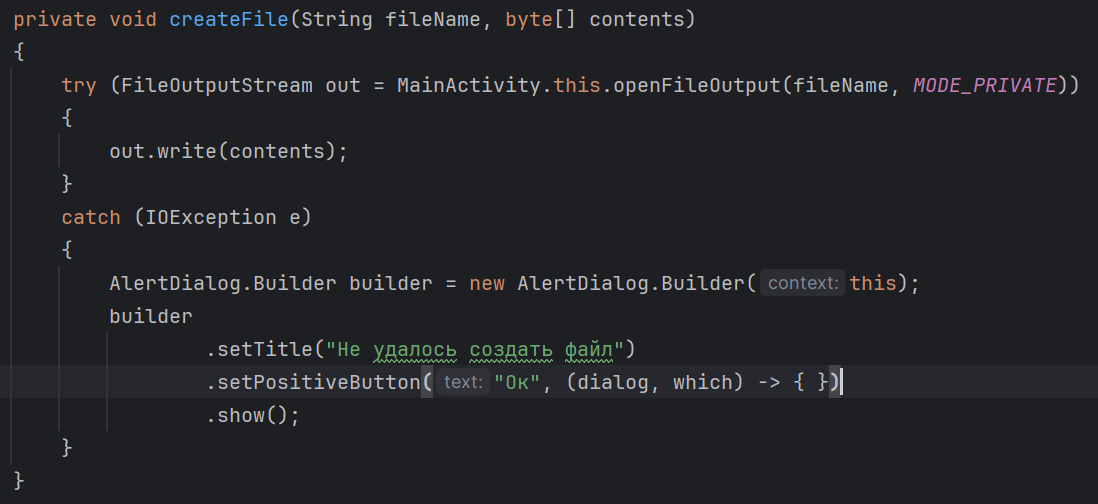


Рисунок 17 – Описание метода createFile() класса MainActivity

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).

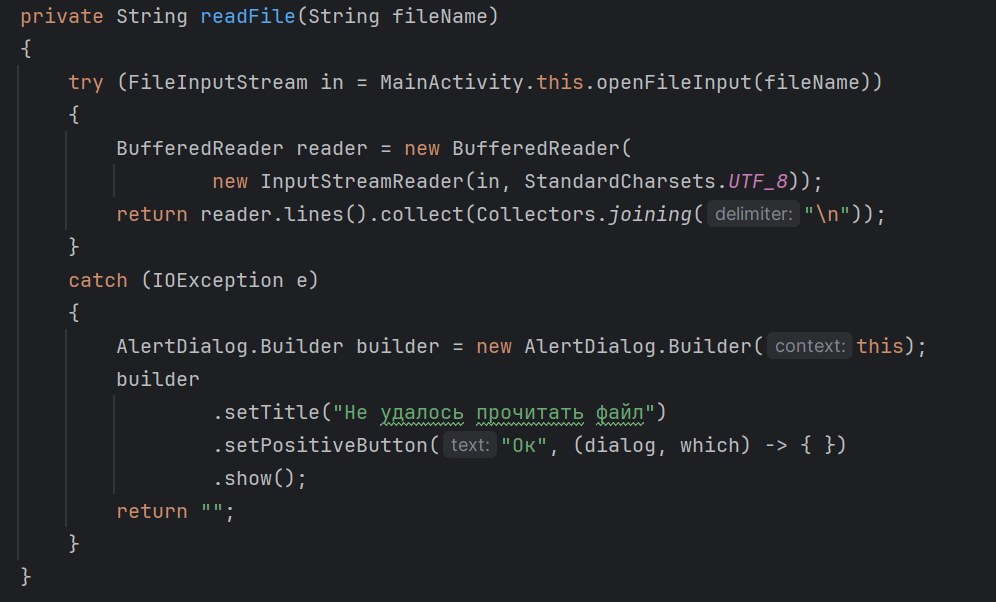


Рисунок 18 – Описание метода readFile() класса MainActivity

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).

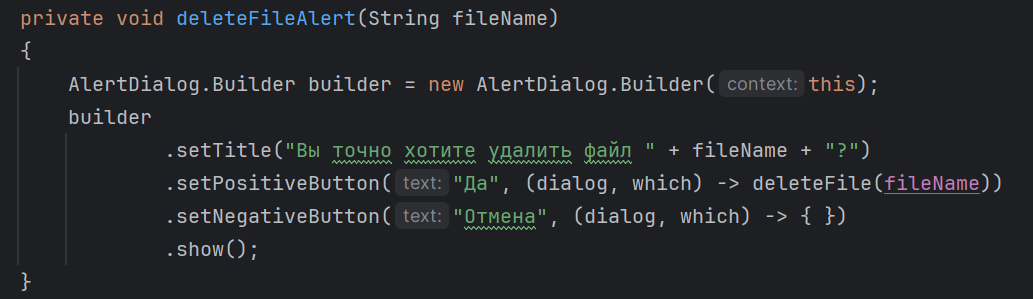


Рисунок 19 – Описание метода deleteFileAlert() класса MainActivity

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).

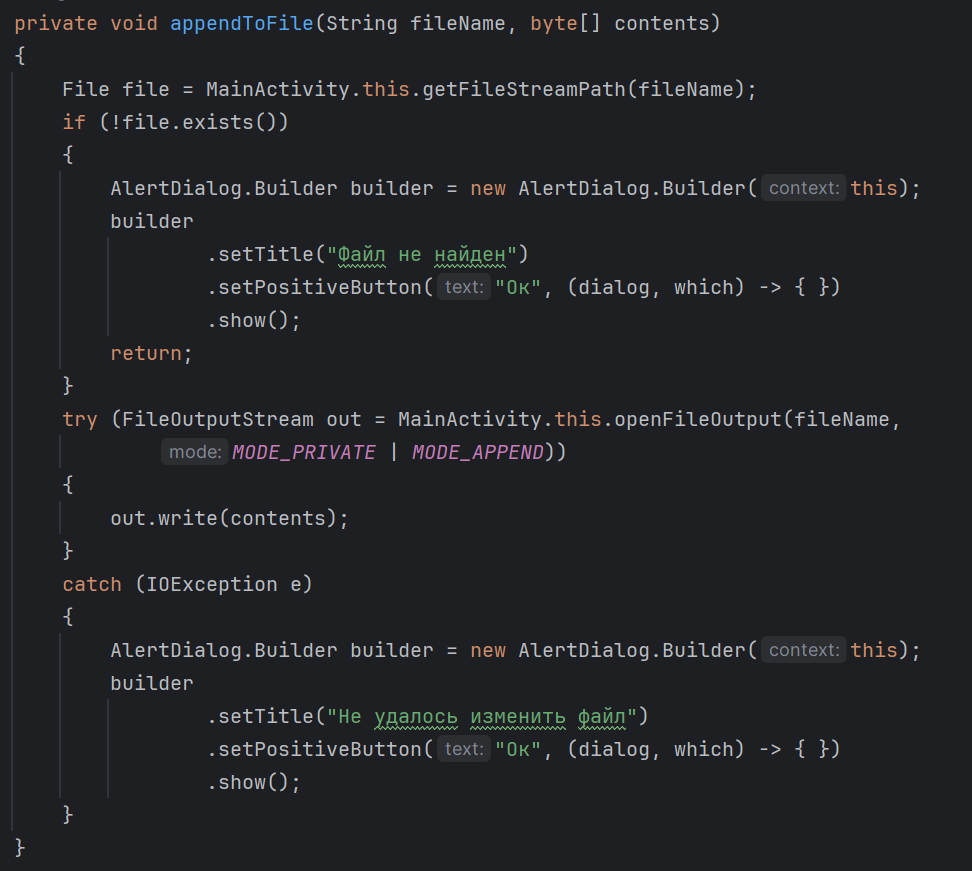


Рисунок 20 – Описание метода appendToFile() класса MainActivity

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

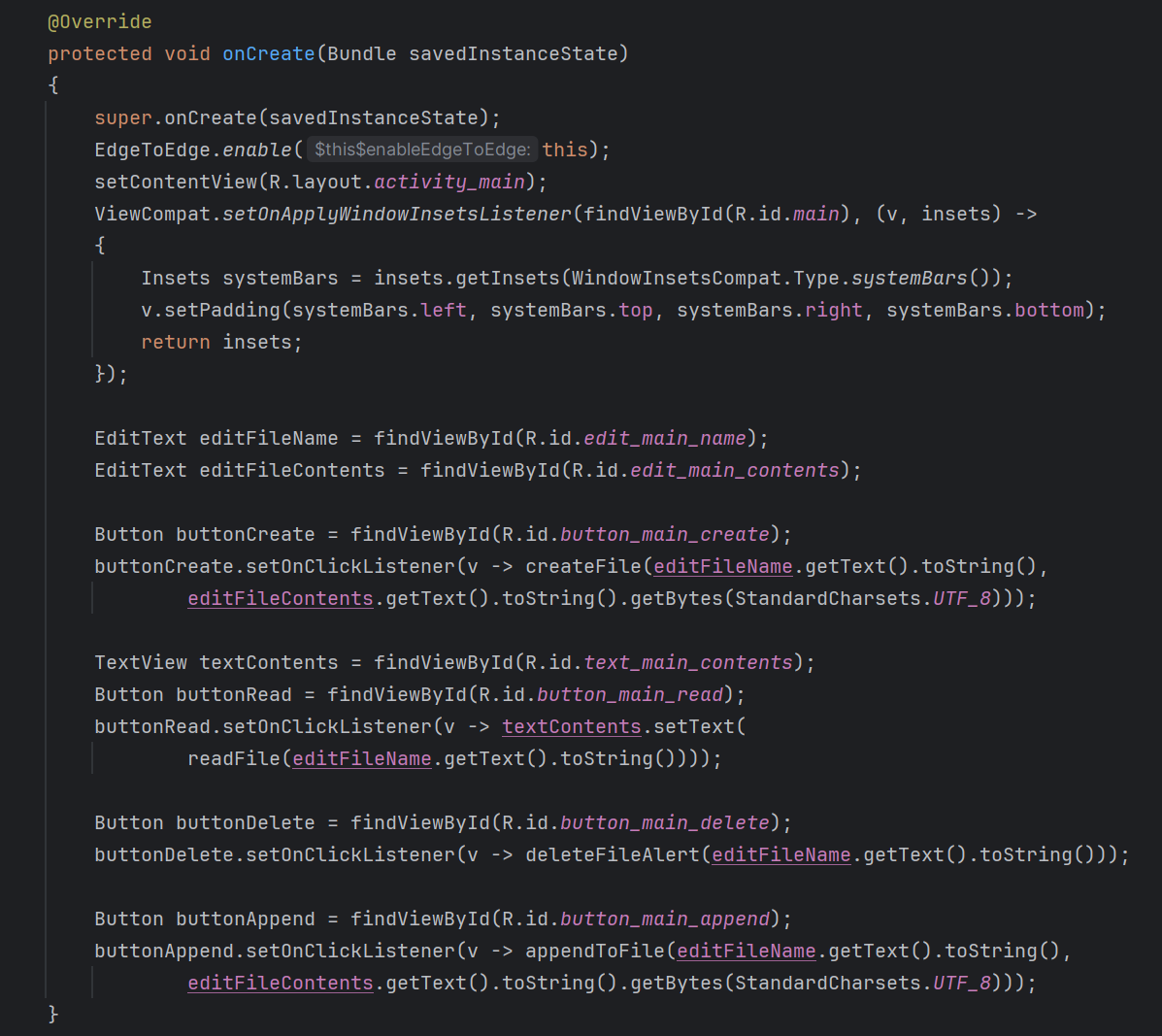


Рисунок 21 – Описание метода onCreate() класса MainActivity

### Сохранение состояния приложения

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



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

### 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).

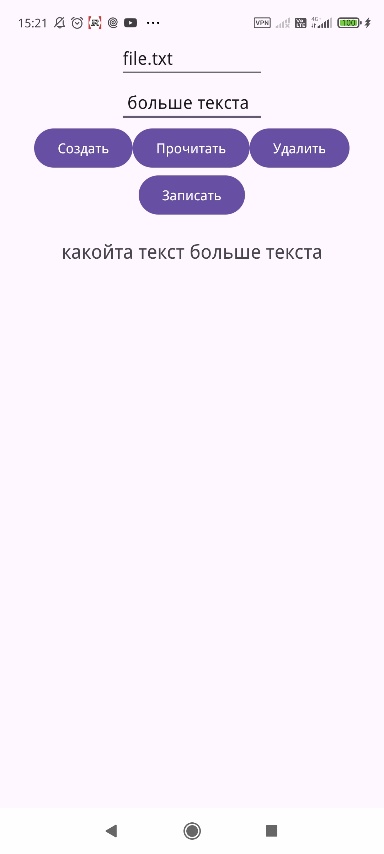


Рисунок 25 – Запись текста в файл внутреннего хранилища

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.

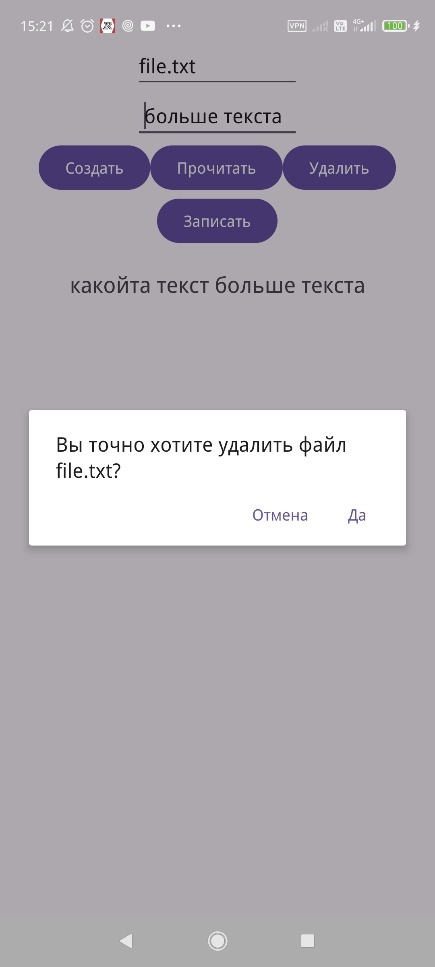


Рисунок 27 – Диалоговое окно подтверждения при попытке удаления файла внутреннего хранилища

## Работа с внешней памятью

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

### 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).

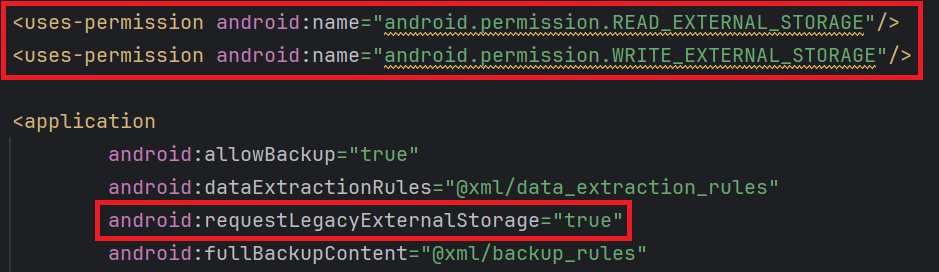


Рисунок 28 – Предоставление необходимых разрешений

### 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).

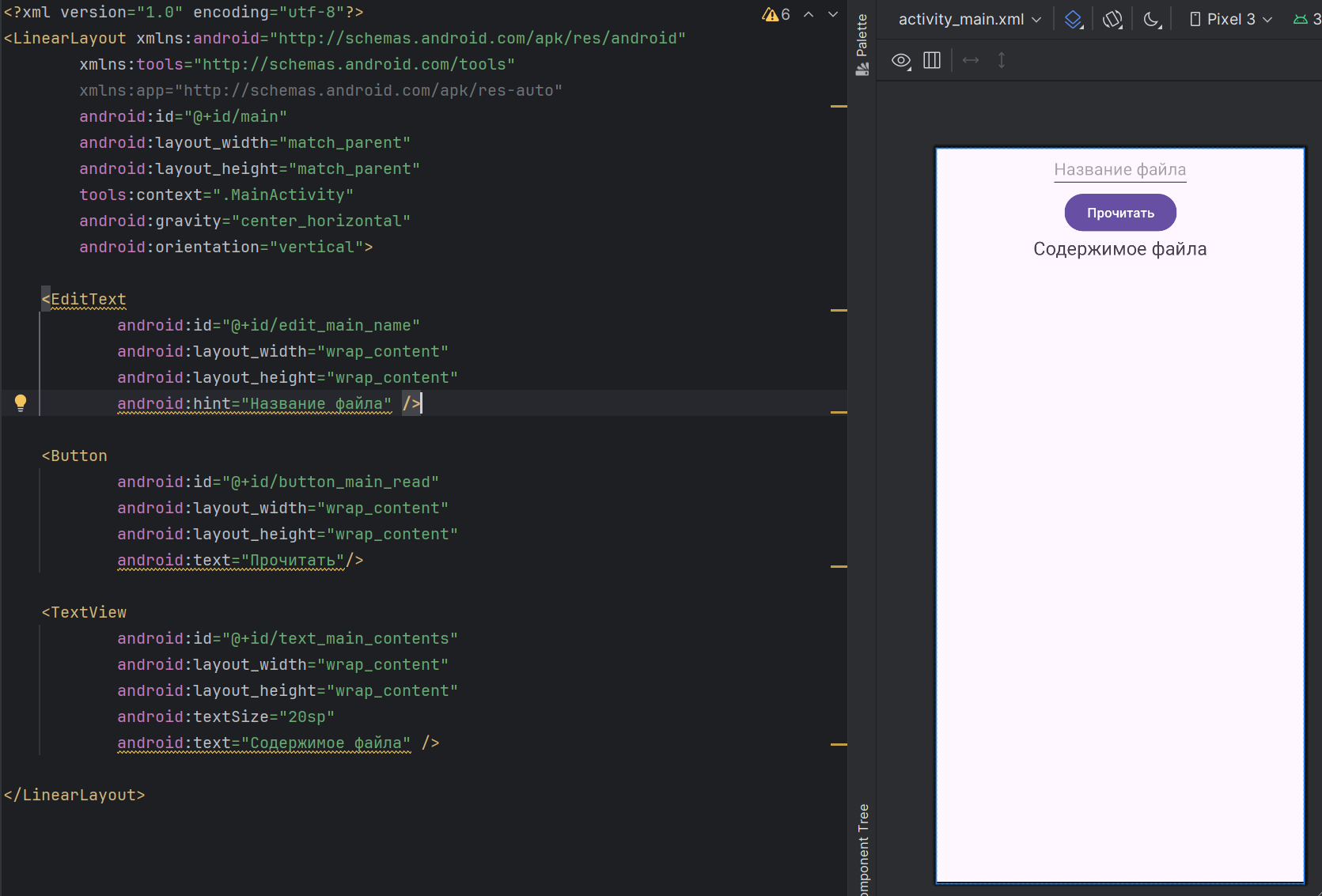


Рисунок 29 – Файл разметки activity\_main.xml второго приложения

### Реализация

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

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

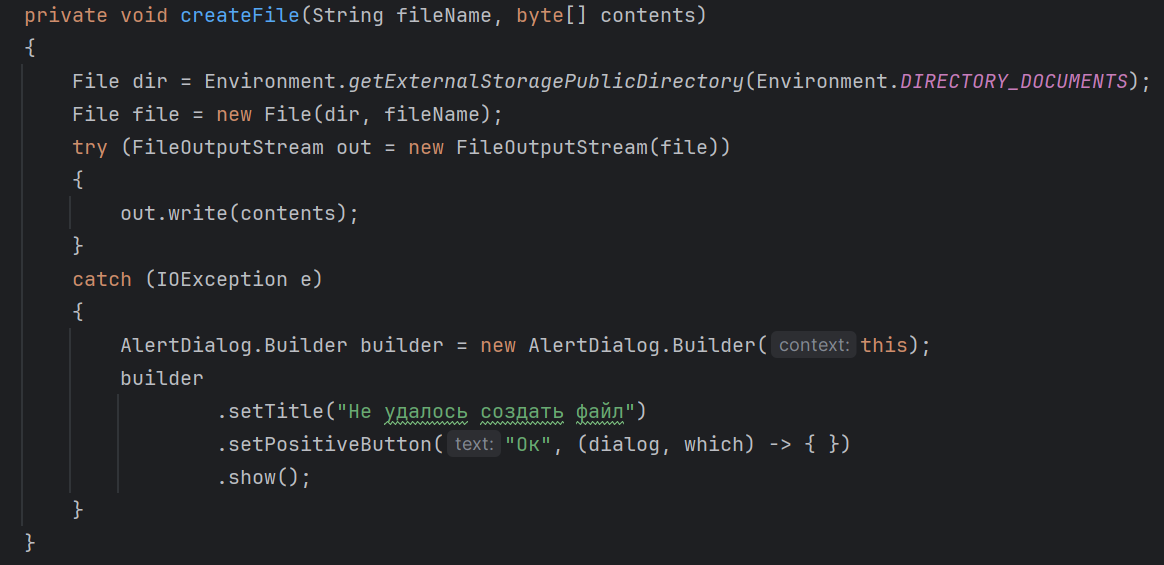


Рисунок 30 – Описание метода createFile() класса MainActivity для внешнего хранилища

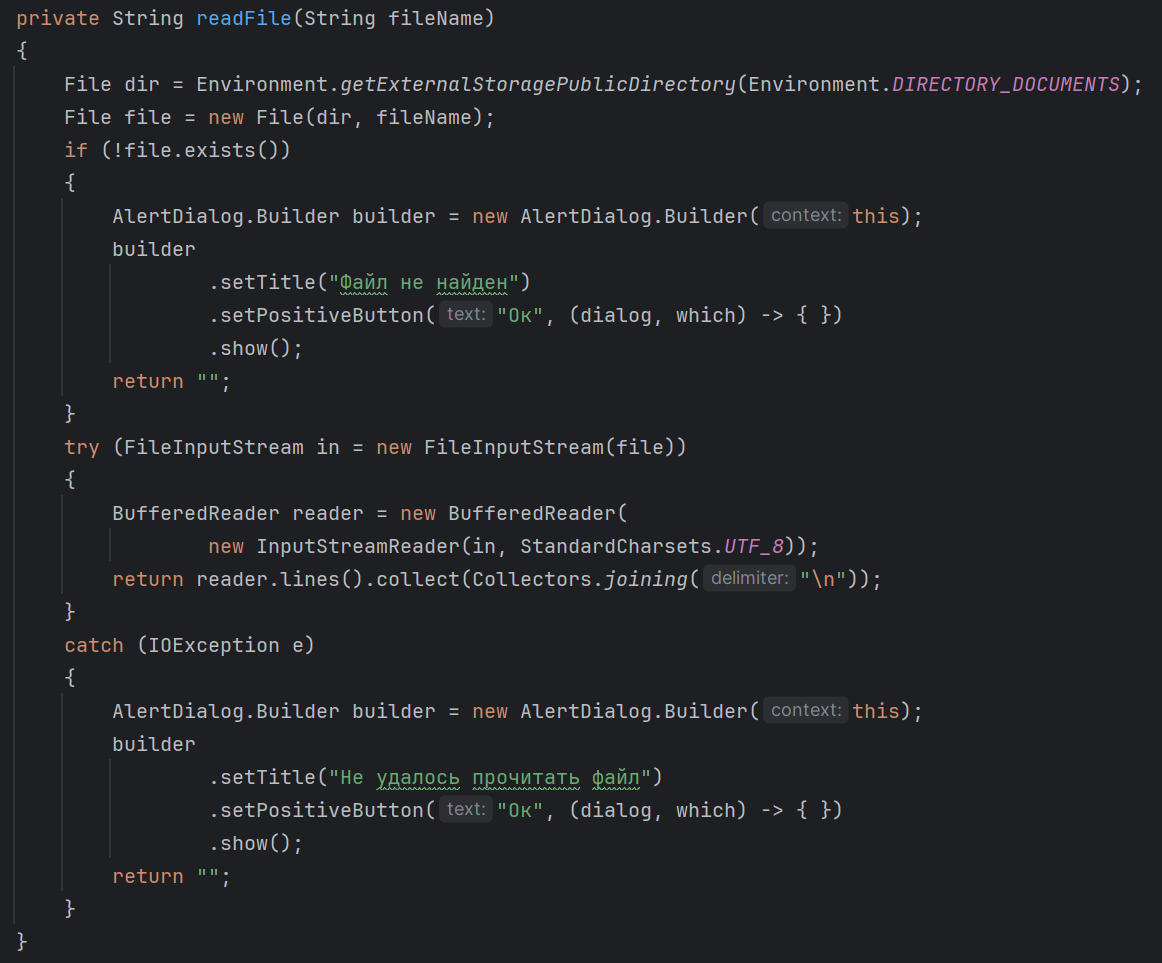


Рисунок 31 – Описание метода readFile() класса MainActivity для внешнего хранилища

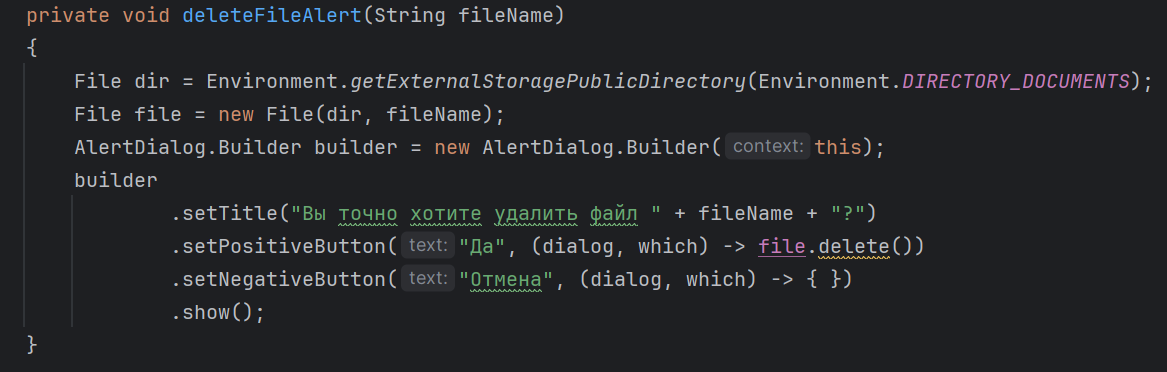


Рисунок 32 – Описание метода deleteFileAlert() класса MainActivity для внешнего хранилища

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).

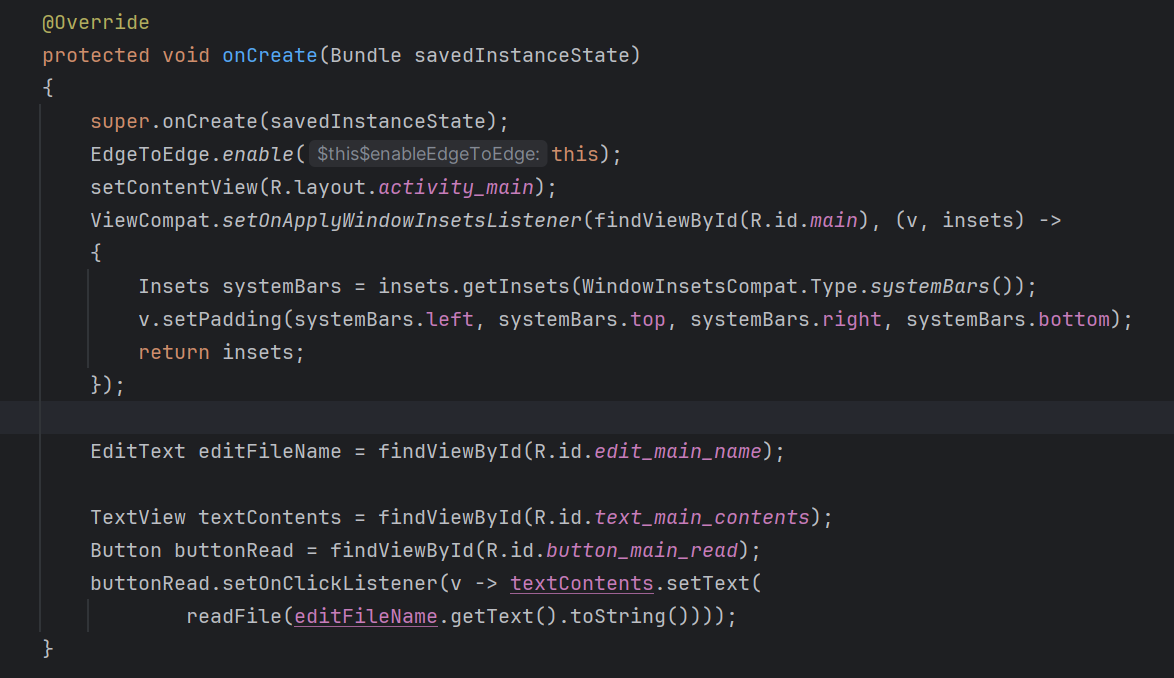


Рисунок 33 – Описание метода onCreate() класса MainActivity второго приложения

### 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 приложение при смене его ориентации. Полученные знания были закреплены путём выполнения практического задания.