5.1. Хранение файлов приложения

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 5.1. Хранение файлов приложения

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:59

Оглавление

1. 5.1.1 Обзор способов хранения данных и файлов

Android использует файловую систему, аналогичную используемым на других платформах. Хранение файлов и данных можно условно поделить на два способа: во внутреннем или внешнем хранилище. В целом политика Google по отношению к хранению данных с каждой версии Android усложняется и ужесточается.

В Android есть внутреннее Internal Storage и внешнее хранилище External Storage. Исторически это были встроенная память в телефоне и внешняя SD-карта, поэтому объем внешнего хранилища как правило был больше, но медленнее и дешевле. Отсюда пошло разделение — настройки приложения и важные файлы для приложения записывали во внутреннее хранилище, а во внешнем хранили данные и большие файлы, например, медиаконтент. Потом внешнее хранилище также стало встраиваться в телефон, но логическое, сохранилось.

У приложения всегда есть доступ к внутреннему хранилищу. Но эта папка только для конкретного приложения и она ограничена в памяти. К внешнему хранилищу необходимо было получать доступ и, кроме манипуляции со своими данными, можно было получить доступ к файлам и данным других приложений и производить с ними какие-либо действия (редактировать, удалять и т.д.).

Чтобы минимизировать риски для пользователя в Google решили внедрить технологию Scoped Storage во внешнее хранилище . Возможность проникнуть в папки других приложений убрали, а доступ есть только к своим данным — теперь это сугубо личная папка. Папка внутреннего хранилища начиная с 10-й версии по умолчанию является зашифрованной. Если Scoped Storage не поддерживается на устройстве (ниже 10-й версии), то для доступа к данным других приложений потребуется получить доступ к чтению и записи в память. Иначе придётся получать доступ к файлам через Media Content, Storage Access Framework или новый, появившийся в 11-м Android, фреймворк Datasets в зависимости от типа данных. В таком случае придётся получать разрешение доступа к файлу, но по более интересной схеме. Когда общий файл создает само приложение сам, то доступ к нему не нужен. Однако, если переустановить приложение — доступ к нему опять потребуется.

На сегодняшний день в системе есть несколько вариантов хранения данных приложения:

- *Специфичные для приложения файлы.* Здесь хранятся файлы, предназначенные только для использования приложением. Доступ к файлам имеет только приложение, их создавшее. При этом файлы могут находиться во внутреннем и внешнем хранилище. У остальных приложений нет доступа, за исключением тех случаев, когда файлы хранятся на внешнем хранилище. При удалении приложения все файлы удаляются.
- *Общее (разделяемое) хранилище.* Здесь хранятся файлы, которыми ваше приложение намерено поделиться с другими приложениями. Для медифайлов (картинки, видео и т.д.) требуется разрешение READ_EXTERNAL_STORAGE или WRITE_EXTERNAL_STORAGE.
- *Настройки.* Здесь хранятся данные по принципу "ключ-значение". Доступно внутри приложения. Реализовано через Jetpack Preferences. Настройки удаляются, когда приложение удаляется пользователем.
- *Базы данных*. Здесь хранятся структурированные данные в базе данных SQLite. На данный момент хранение реализовано через библиотеку Room. Доступ к данных при этом имеется только у приложения, в котором база была создана.

Решение по выбору одного из вариантов хранения зависит от конкретных потребностей, таких как, должны ли данные быть доступными только для вашего приложения или нет, сколько места требуется для ваших данных и др. Характеристики разнообразных вариантов хранения данных изложены в официальной документации.

2. 5.1.2. Разрешения и доступ к внешнему хранилищу

При работе с внутренним и внешним хранилищем следует соблюдать осторожность, поскольку оно может быть не слишком большим и не подходить по объему для вашего приложения. Вдобавок к внутреннему хранилищу, устройство может иметь внешнее хранилище. В старых моделях им являлась съёмная SD-карта. Если ваше приложение слишком большое, можно указать в файле манифеста о необходимости устанавливать программу во внешнее хранилище:

```
<manifest ...
  android:installLocation="preferExternal">
  ...
</manifest>
```

Требования к разрешению работы с внешним хранилищем постоянно изменялись в зависимости от выхода новых версий системы Android . Сегодня приложение может иметь доступ к собственным и к некоторым общим файлам , которые находятся во внешнем хранилище. Доступ к общим файлам достигается через FileProvider API или контент-провайдеры.

Для просмотра файлов через Android Studio можно воспользоваться инструментом Device File Explorer.

Android определяет следующие разрешения, связанные с хранением файлов во внешней памяти:

- `READ_EXTERNAL_STORAGE` - позволяет приложению считывать данные из внешнего хранилища.
- `WRITE_EXTERNAL_STORAGE` - позволяет приложению записывать данные во внешнее хранилище.
- `MANAGE_EXTERNAL_STORAGE` - позволяет приложению записывать данные во внешнее хранилище. Предназначен для использования несколькими приложениями, которым необходимо управлять файлами от имени пользователей. Если вашему приложению не нужен доступ к разрешению, удалите его из файла манифеста, иначе вы не сможете опубликовать свой продукт.

В более ранних версиях Android в приложениях необходимо было объявить разрешение `READ_EXTERNAL_STORAGE` для доступа к любому файлу за пределами каталогов приложения на внешнем хранилище. Кроме того, в приложениях необходимо было объявить разрешение `WRITE_EXTERNAL_STORAGE` на запись в любой файл за пределами каталога конкретного приложения.

Более поздних версиях Android ситуация по по работе с файлами (чтение или запись) больше зависит от их назначения, а не от его местоположения. Например, если приложение предназначено для Android 11или выше, разрешение `WRITE_EXTERNAL_STORAGE` никак не влияет на доступ приложения к хранилищу. Такой способ хранения данных повышает конфиденциальность пользователей, поскольку приложениям предоставляется доступ только к фактически используемым областям файловой системы устройства.

Чтобы предоставить пользователям больший контроль над собственными файлами и обеспечить их порядок, приложениям для версии Android 11 и выше, по умолчанию предоставляется ограниченный доступ к внешнему хранилищу или хранилищу с областью действия. Так например добавлено принудительное использование хранилища с ограниченной областью видимости (Scoped storage): доступ к каталогам внешних хранилищ ограничен каталогом конкретного приложения и определенными типами носителей, созданных приложением.

Чтобы прочитать/записать данные на внешнем хранилище, требуется добавить в AndroidManifest.xml разрешения:

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

Внешнее хранилище Android - это область памяти, в которой мы выполняем операции чтения и записи. Файлы во внешнем хранилище хранятся в папке /sdcard или /storage и т.д. Файлы, которые сохраняются во внешнем хранилище, доступны для чтения и могут быть изменены пользователем. Прежде чем получить доступ к файлу во внешнем хранилище в приложении, нужно проверить его доступность на устройстве.

Запись чтение в файл во внешнем хранилище

В пакете java.io существует метод `openFileOutput()`, который возвращает экземпляр класса `FileOutputStream` для записи файла во внешнее хранилище устройства. Для того чтобы получить каталог, который используется только вашим приложением, необходимо вызвать метод `getExternalFilesDir()`. Чтобы записать данные в файл, вызовите метод `fileOutputStream.write()`.

```
var myExternalFile:File = File(getExternalFilesDir(filepath),fileName)
 try {
     val fileOutPutStream = FileOutputStream(myExternalFile)
     fileOutPutStream.write(fileContent.toByteArray())
     fileOutPutStream.close()
 } catch (e: IOException) {
     e.printStackTrace()
 }
```

Кроме этого пакет java.io предлагает метод `openFileInput()`, который возвращает экземпляр класса `FileInputStream` и считывает файл из внешнего хранилища устройства. Чтобы прочитать данные из файла, необходимо вызвать метод `BufferedReader().readLine()`:

```
var myExternalFile:File = File(getExternalFilesDir(filepath), fileName)
val filename = fileName.text.toString()
myExternalFile = File(getExternalFilesDir(filepath),filename)
var fileInputStream =FileInputStream(myExternalFile)
var inputStreamReader: InputStreamReader = InputStreamReader(fileInputStream)
val bufferedReader: BufferedReader = BufferedReader(inputStreamReader)
val stringBuilder: StringBuilder = StringBuilder()
var text: String? = null
while ({ text = bufferedReader.readLine(); text }() != null) {
   stringBuilder.append(text)
}
fileInputStream.close()
```

3. 5.1.3. Доступ к файлам приложений

Во многих приложениях создаются файлы. При этом другие приложения зачастую не должны иметь к ним доступ. Система Android предоставляет следующие возможности для хранения таких файлов:

- *Внутренние каталоги*. Они содержат в себе выделенное место для хранения постоянных файлов и данных кэша. Система Android запрещает доступ другим приложениям к этим файлам, и начиная с версии Android 10 (уровень API 29) и выше эти местоположения зашифрованы.
- *Внешние каталоги*. Они также содержат выделенное место для хранения постоянных файлов и данных кэша. Другие приложения могут получить доступ данных только в случае наличия специального разрешения. Когда такой доступ необходим, приложение должно хранить эти файлы в общей части внешнего хранилища.

При удалении приложения, файлы, созданные в хранилище для него также удаляются. Таким образом, не стоит сохранять файлы в подобные хранилища , если материалы могут быть полезны пользователю вне зависимости от того есть приложение на устройстве или его нет. К примеру, если в вашем приложении можно делать снимки с камеры, то пользователь будет ожидать возможность их использования. Необходимо обеспечить доступ к этим фотографиям даже после удаления вашего приложения.

Для каждого приложения система предоставляет два каталога во внутреннем хранилище. Один предназначен для постоянных файлов, а другой содержит кэшированные файлы приложений. При этом, как было сказано ранее, другие приложения не имеют е ним доступ к файлам, что делает внутреннее хранилище хорошим местом для хранения конфиденциальной информации приложений. Данные каталоги имеют относительно небольшой размер памяти, соответственно перед записью файлов необходимо запросить размер свободного пространства на устройстве.

Получить доступ к постоянным файлам приложения можно с использованием свойства `filesDir` объекта `Context`. При этом в Android есть несколько методов, с помощью которых можно получить доступ к файлам и сохранить их.

Первым вариантом для доступа к файлам и их хранения выступает использование File API. Следующий фрагмент кода демонстрирует, как его использовать:

```
val file = File(context.filesDir, filename)
```

В качестве альтернативы использованию файлового API можно задействовать метод `openFileOutput()`. В следующий код демонстрирует как записать текст в файл:

```
val filename = "myfile"
val fileContents = "My IT Academy!"
context.openFileOutput(filename, Context.MODE_PRIVATE).use {
        it.write(fileContents.toByteArray())
}
```

Здесь следует отметить, что на устройствах с Android 7.0 или выше, если не передать режим доступа к файлу `MODE_PRIVATE` в метод `openFileOutput()`, появиться исключительная ситуация `SecurityException`.

Для чтения файла в потоковом режиме необходимо использовать метод `openFileInput()`. Сделать это можно следующим образом:

```
context.openFileInput(filename).bufferedReader().useLines { lines ->
    lines.fold("") { my, message ->
        "$my\n$message "
    }
}
```

Еще одной опцией является получение массива из имен каталоге **filesDir**. Для этого целесообразно вызвать метод fileList(), например следующим образом:

```
var files: Array<String> = context.fileList()
```

Также существует возможность создания вложенных каталогов или открытия внутреннего через вызов метода `getDir()`, например следующим образом:

```
context.getDir(dir, Context.MODE_PRIVATE)
```

Для хранения временных данных, можно использовать каталог кэша. Он предназначен для хранения небольшого количества данных, доступных только внутри разработанного приложения. При этом при удалении приложения все файлы так же будут удалены. В случае недостатка места на устройстве, система Android может удалить файлы кэша. Во избежание ошибок нужно проверить наличие файлов

кэша перед их чтением. Для определения доступного места в каталоге кэша можно вызвать метод `getCacheQuotaBytes()`, а для создания кэшированного файла можно использовать следующий код:

```
File.createTempFile(filename, null, context.cacheDir)
```

В случае недостатка места на устройстве, система Android может удалить файлы кэша. Во избежание ошибок нужно проверить наличие файлов кэша перед их чтением.

Android иногда удаляет файлы кэша самостоятельно, но не очищает их для вас. Для удаления ненужных файлов можно использовать следующий код:

```
cacheFile.delete()
```

Еще один вариант удаления следующий:

```
context.deleteFile(cacheFileName)
```

Вы можете запросить состояние памяти внешнего хранилища, вызвав метод `Environment.getExternalStorageState()`. Если возвращаемое значение соответствует `MEDIA_MOUNTED`, можно читать и записывать файлы во внешнее хранилище. Если метод вернет `MEDIA_MOUNTED_READ_ONLY`, то возможно только чтение. Метод, представленный ниже проверяет размер памяти внешнего хранилища для чтения и записи:

```
fun isExternalStorageWritable(): Boolean {
    return Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED
}
```

Для доступа к файлам внешнего хранилища, требуется вызов метода `getExternalFilesDir()`:

```
val appSpecificExternalDir = File(context.getExternalFilesDir(null), filename)
```

Для добавления файла в кэш внешнего хранилища, можно написать следующий код:

```
val externalCacheFile = File(context.externalCacheDir, filename)
```

Для удаления файла из каталога внешнего кэша, задействуйте метод delete() для объекта File:

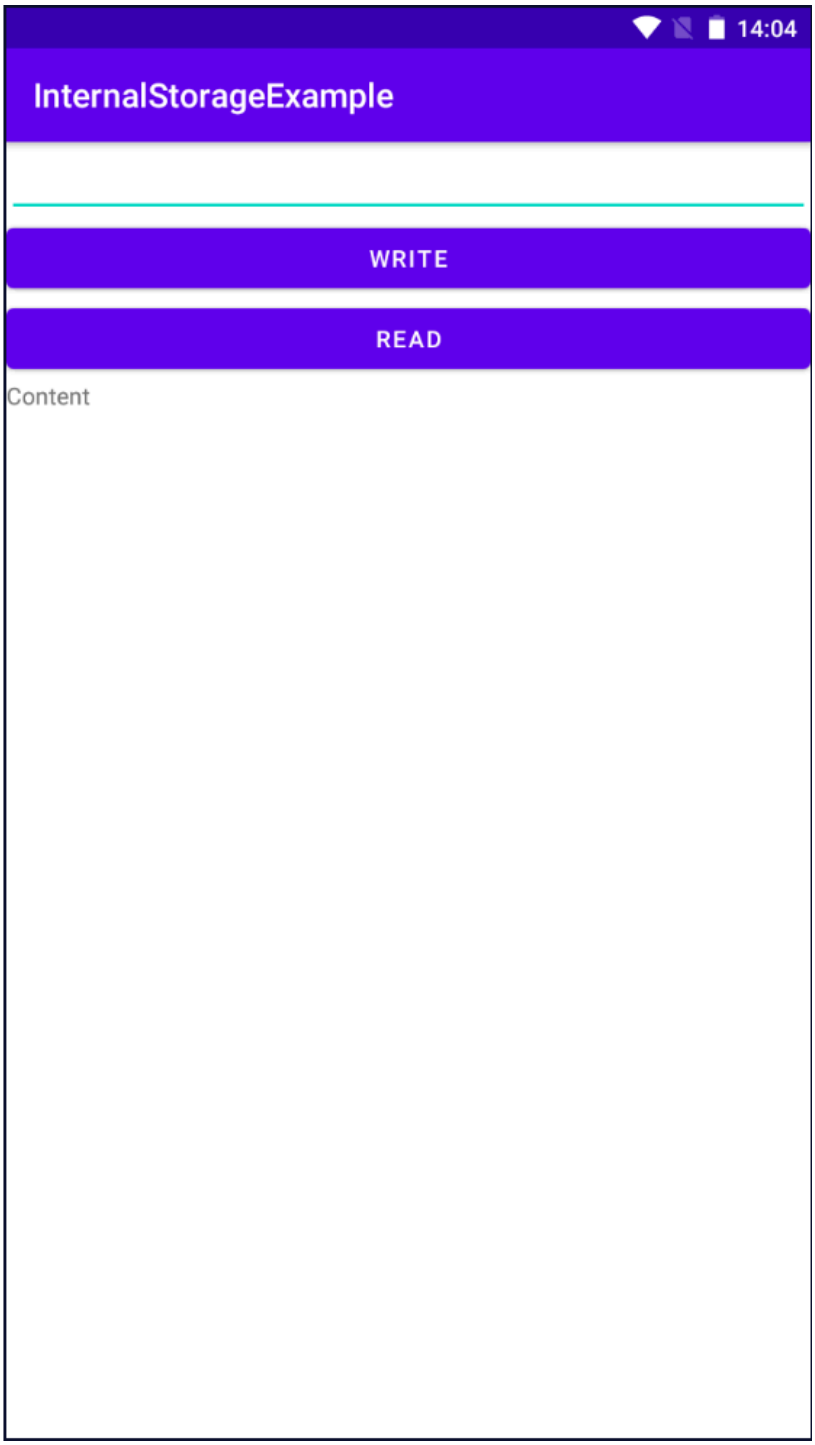
```
externalCacheFile.delete()
```

4. 5.1.4. Упражнение 5.1.

Создадим проект, в котором будем записывать и считывать данные из файла из внутреннего хранилища устройства. Файл activity_main.xml будет иметь следующую структуру: Поле ввода, две кнопки для чтения и записи данных, текстовое представление для отображения содержимого файла. Разметка может выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="wrap_content"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="" />

    <Button
        android:id="@+id/write"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Write" />

    <Button
        android:id="@+id/read"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Read" />

    <TextView
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Content" />

</LinearLayout>
```

При запуске приложение будет выглядеть так:



Внутри MainActivity.kt напишем следующее. Объявим необходимые поля и в методе onCreate проинициализируем их:

```
var read: Button? = null
var write: Button? = null
var userInput: EditText? = null
var fileContent: TextView? = null
private val filename = "file.txt"

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    read = findViewById(R.id.read) as Button
    write = findViewById(R.id.write) as Button
    userInput = findViewById(R.id.input)
    fileContent = findViewById(R.id.content)

    read!!.setOnClickListener(this)
    write!!.setOnClickListener(this)
}
```

Далее реализует для класса MainActivity интерфейс `View.OnClickListener` и его метод onClick():

```
override fun onClick(view: View) {
        val b: Button = view as Button
        val b_text: String = b.getText().toString()
        when (b_text.toLowerCase()) {
            "write" -> {
                writeData()
            }
            "read" -> {
                readData()
            }
        }
    }
```

Методы `writeData()` и `readData()` в методе `onClick()` предназначены для записи данных в файл и их считывания соответственно. При этом будет создан File.txt, который можно найти в файловой системе устройства ***Device File Explorer › data › data › application_package › files ***.

Предлагается следующая реализация методов `writeData()` и `readData()`.

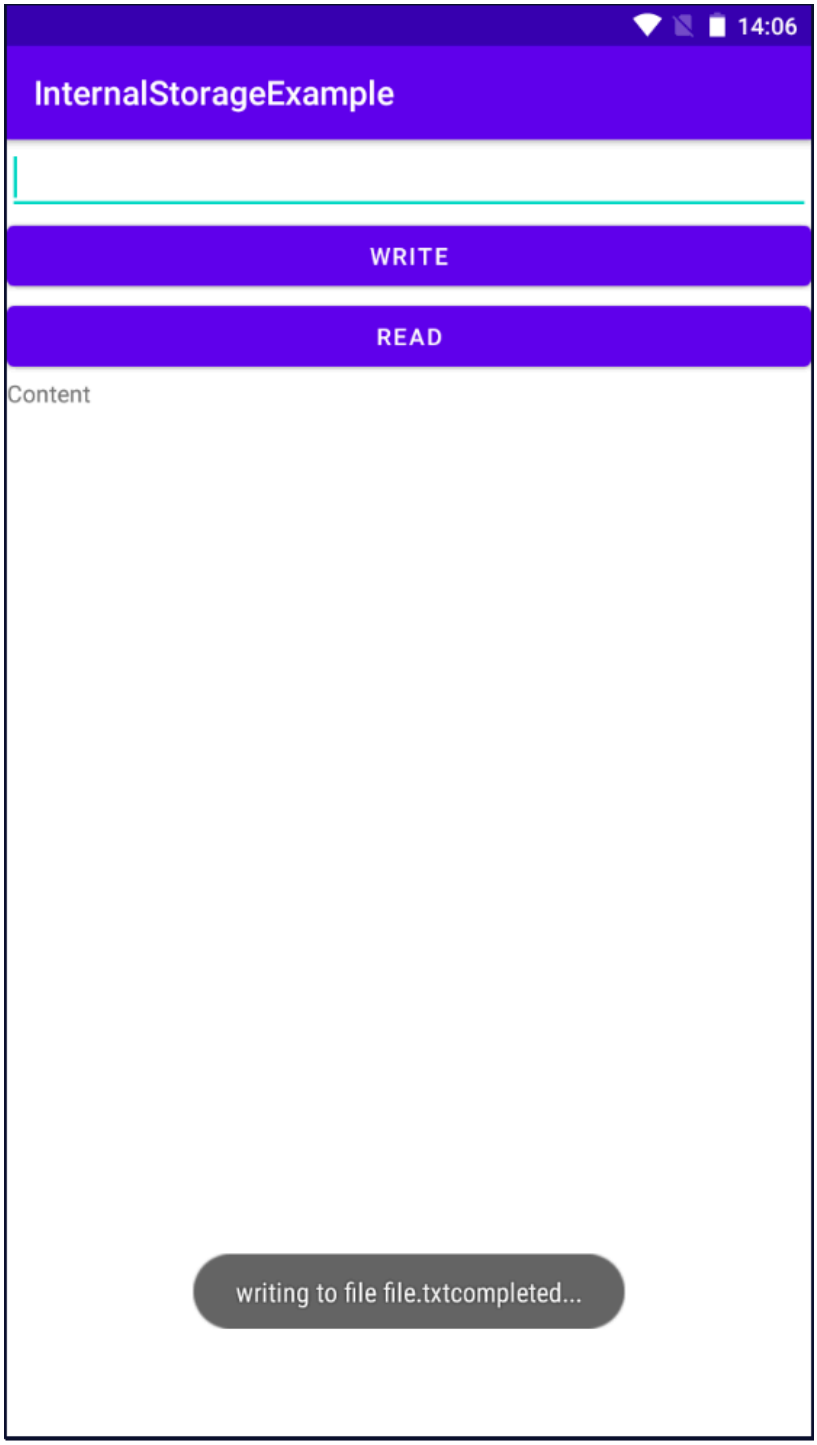
```
private fun writeData() {
        try {
            val fos: FileOutputStream = openFileOutput(filename, Context.MODE_PRIVATE)
            val data = userInput!!.text.toString()
            fos.write(data.toByteArray())
            fos.flush()
            fos.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        userInput!!.setText("")
        printMessage("writing to file " + filename + "completed...")
    }

    private fun readData() {
        try {
            val fin: FileInputStream = openFileInput(filename)
            var a: Char
            val temp = StringBuilder()
            while (fin.read().also { a = it.toChar() } != -1) {
               temp.append(a)

            }

            fileContent!!.text = temp.toString()
            fin.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        printMessage("reading to file $filename completed..")
    }
```

Полностью класс MainActivity будет выглядеть следующим образом:

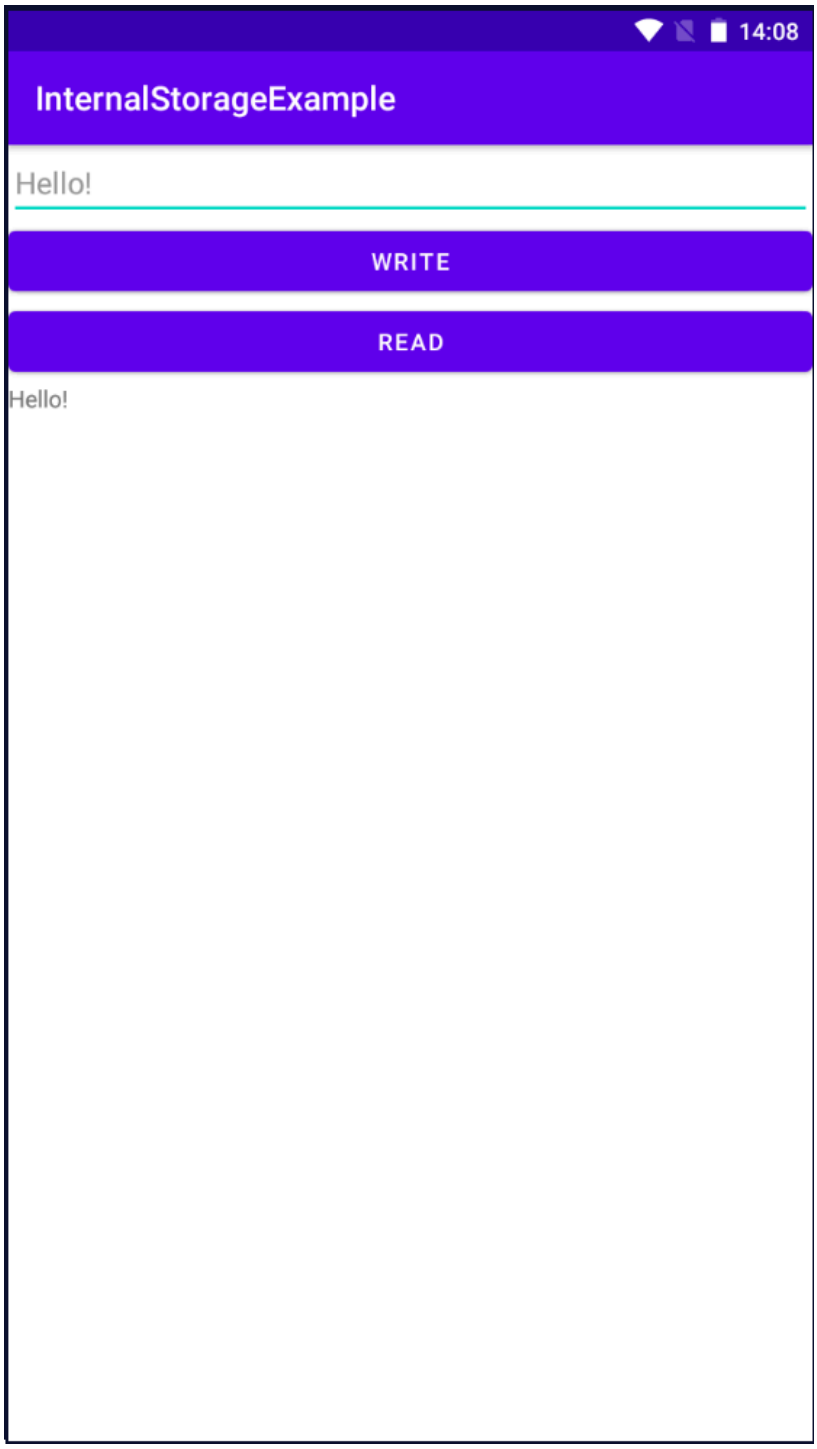
```
package ru.samsung.itacademy.mdev.internalstorageexample

import ru.samsung.itacademy.mdev.internalstorageexample.R
import android.content.Context
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import java.io.FileInputStream
import java.io.FileOutputStream
import java.io.IOException


class MainActivity : AppCompatActivity(), View.OnClickListener {

    var read: Button? = null
    var write: Button? = null
    var userInput: EditText? = null
    var fileContent: TextView? = null
    private val filename = "file.txt"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        read = findViewById(R.id.read) as Button
        write = findViewById(R.id.write) as Button
        userInput = findViewById(R.id.input)
        fileContent = findViewById(R.id.content)

        read!!.setOnClickListener(this)
        write!!.setOnClickListener(this)
    }

    fun printMessage(m: String?) {
        Toast.makeText(this, m, Toast.LENGTH_LONG).show()
    }

    override fun onClick(view: View) {
        val b: Button = view as Button
        val b_text: String = b.getText().toString()
        when (b_text.toLowerCase()) {
            "write" -> {
                writeData()
            }
            "read" -> {
                readData()
            }
        }
    }

    private fun writeData() {
        try {
            val fos: FileOutputStream = openFileOutput(filename, Context.MODE_PRIVATE)
            val data = userInput!!.text.toString()
            fos.write(data.toByteArray())
            fos.flush()
            fos.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        userInput!!.setText("")
        printMessage("writing to file " + filename + "completed...")
    }

    private fun readData() {
        try {
            val fin: FileInputStream = openFileInput(filename)
            var a: Char
            val temp = StringBuilder()
            while (fin.read().also { a = it.toChar() } != -1) {
               temp.append(a)
```

```
            }

            fileContent!!.text = temp.toString()
            fin.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        printMessage("reading to file $filename completed..")
    }
}
```

После запуска приложения, ввода слова "Hello!" в поле для ввода и нажатия кнопки "WRITE" данные из поля ввода запишутся в файл и на экране устройства появиться следующее изображение:



Далее, после нажатия кнопки "READ", информация из файла появиться в текстовом представлении с id content.



Полный код проекта упражнения можно посмотреть по ссылке

Начать тур для пользователя на этой странице