Data Storage: Internal/External Storage

Sensor Based Mobile Applications

Patrick Ausderau, Ulla Sederlöf, Jarkko Vuori

Helsinki Metropolia University of Applied Science

2022

Outline

Doc: `https://developer.android.com/guide/topics/data`

Data Storage on Android

- Shared Preferences
  - Store private primitive data in key-value pairs
- Internal Storage
  - Store private data on the device memory
- External Storage
  - Store (public (deprecated)) or half-private data on the shared external storage
- SQLite Databases / Room
  - Store structured data in a private database
- Media
  - Shareable media files (images, audio files, videos)

Data Storage on Android

- Storage Access Framework
  - allows users to interact with a system picker to choose a documents provider and select specific files and directories for your app to create, open, or modify
- Content providers
  - Manages access to a central repository of data
- Network Connection
  - Store data on the web with your own network server
- Data Backup
  - Store data on user's Google Drive or Android Backup Service in order to restore user data on new devices
- FileProvider API
  - extension of content provider to securely share files from your app to another app using content URIs (with temporary permissions granted only to the receiving app)

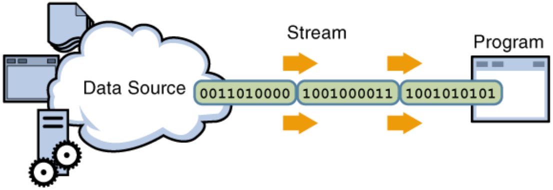
File

- You can store files directly on the mobile device
- To read data from a file, Kotlin inherit several standard Java classes like `java.io.File` or `BufferedInputStream`
- And similar i/o methods, like `byteInputStream()` and `readLine()` simplify reading and writing of streams - these methods provide access only to current folder (package dependent)

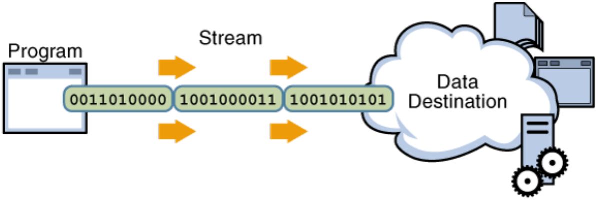
Streams

A stream is a sequence of data.

- A program uses an input stream to read data from a source, one item at a time:



- A program uses an output stream to write data to a destination, one item at time:

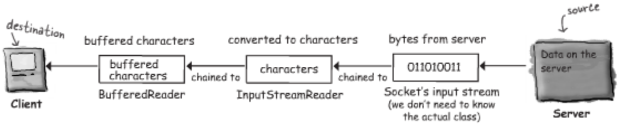


Byte and Character Streams

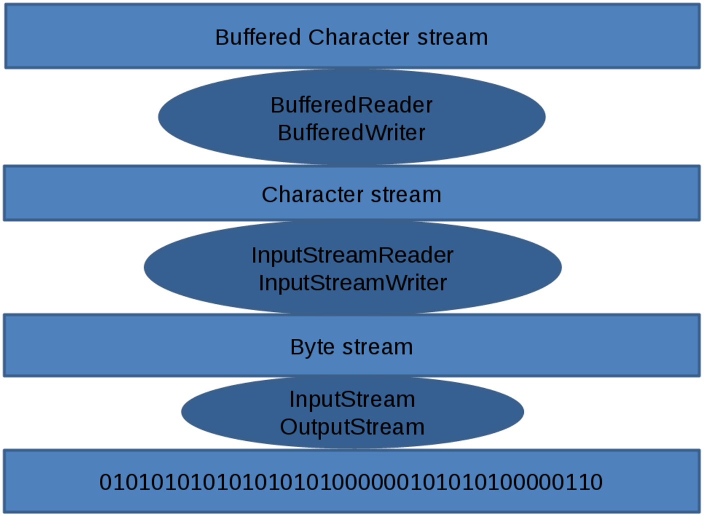
- Unless you are working with binary data, such as image and audio files, you should use readers and writers (character streams) to read and write information for the following reasons:
  - They can handle any character in the Unicode character set (while the byte streams are limited to ISO-Latin-1 8-bit bytes).
  - They are easier to internationalize because they are not dependent upon a specific character encoding.
  - They use buffering techniques internally and are therefore potentially much more efficient than byte streams.
- There are typically two almost identical classes for manipulating byte-streams and character-streams - character-stream classes end with the suffix Reader or Writer and byte-stream classes end with the suffix InputStream and OutputStream.

Stream Chaining

- One of the most convenient features of the I/O stream classes is that they are designed to work together via stream chaining.
- Stream chaining is a way of connecting several stream classes together to get the data in the form required. Each class performs a specific task on the data and forwards it to the next class in the chain.



Stream Chaining



Stream Chaining Related Classes

- `FileReader` and `FileWriter`
  - For reading from or writing to files.
- `BufferedReader` and `BufferedWriter`
  - For buffered reading/writing to reduce disk access for more efficiency.
- `FileInputStream` and `FileOutputStream`
  - For reading streams from or writing streams to files.
- `InputStreamReader` and `OutputStreamWriter`
  - Provide a bridge between byte and character streams.
  - The only purpose of these classes is to convert byte data into character-based data according to a specified (or the platform default) encoding.

Buffering

- Use buffers as they are much more efficient than working without them.
- You can write to a file using `OutputStreamWriter` alone, by calling `write(someString)`, but `OutputStreamWriter` writes each and every thing you pass to the file each and every time.
- By chaining a `BufferedWriter` onto a `OutputStreamWriter`, the `BufferedWriter` will hold all the stuff you write to it until it's full. Only when the buffer is full will the `OutputStreamWriter` actually be told to write to the file.
  - If you do want to send data before the buffer is full call `writer.flush()`

Internal Storage

- Each application has its own private internal storage to save files.
  - It's always available.
  - Files saved here are accessible by only your app.
  - When the user uninstalls your app, the system removes all your app's files from internal storage.
  - Internal storage is best when you want to be sure that neither the user nor other apps can access your files.
  - on Android 10 (API level 29) and higher, this location is encrypted
- Write to file: `openFileOutput()`
- Read from file: `openFileInput()`

OpenFileOutput Example

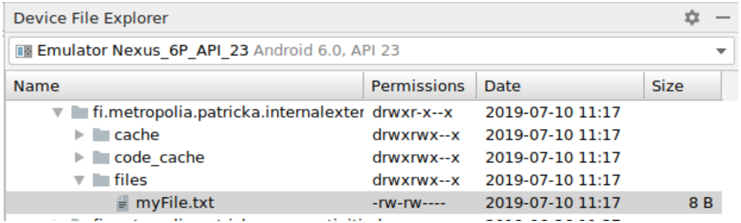
```kotlin
internal const val FILENAME = "myFile.txt"
//...
@Composable
fun WriteFile(app: Application) {
  //text by remember, Column, TextField,...
  Button(onClick = {
    app.openFileOutput(FILENAME, Context.MODE_APPEND).use {
      it.write("$text\n".toByteArray())
    }
    text = ""
  }) { Text(stringResource(R.string.write_file)) }
}
```

Note:

- Destination folder predefined by Android OS (based on app id (package name))!
- `MODE_APPEND` will add at the end of the file (or create the file before writing if it does not exist). To overwrite existing file, use `Context.MODE_PRIVATE`

OpenFileOutput Example

Check internal file – from emulator or rooted device or debugging app from Android Studio



Device File Explorer

Emulator Nexus_6P_API_23 Android 6.0, API 23

Name	Permissions	Date	Size
fi.metropolia.patricka.internalexter	drwxr-x--x	2019-07-10 11:17	
cache	drwxrwx--x	2019-07-10 11:17	
code_cache	drwxrwx--x	2019-07-10 11:17	
files	drwxrwx--x	2019-07-10 11:17	
myFile.txt	-rw-rw----	2019-07-10 11:17	8 B

- Use Device File Explorer (menu View ⇒ Tool Windows ⇒ Device File Explorer)
- file can be find from `/data/data/<your-package>/files` subfolder
- Select your file and Pull a file from the emulator/device

OpenFileOutput Example

Check internal file – from non-rooted device

- Through Android Device Monitor File Explorer you may not have access to private folders. In that case, you have to use Android Debug Bridge (ADB).
- ADB can be find from the SDK's platform-tools subfolder
- Check the device id

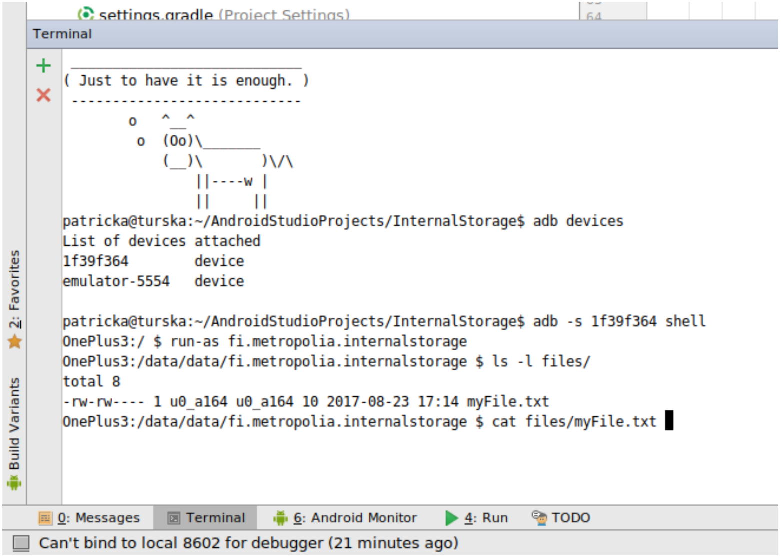
  ```
  $ adb devices
  ```

- Use ADB remote shell to check if file exists:

  ```
  $ adb -s <device id> shell
  $ run-as <your app package name>
  $ ls -l files/
  $ cat files/<your file name>
  ```

OpenFileOutput Example

Check internal file – from non-rooted device



```
 ____________________________
( Just to have it is enough. )
 ----------------------------
       o   ^__^
        o  (Oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||
patricka@turska:~/AndroidStudioProjects/InternalStorage$ adb devices
List of devices attached
1f39f364        device
emulator-5554   device

patricka@turska:~/AndroidStudioProjects/InternalStorage$ adb -s 1f39f364 shell
OnePlus3:/ $ run-as fi.metropolia.internalstorage
OnePlus3:/data/data/fi.metropolia.internalstorage $ ls -l files/
total 8
-rw-rw---- 1 u0_a164 u0_a164 10 2017-08-23 17:14 myFile.txt
OnePlus3:/data/data/fi.metropolia.internalstorage $ cat files/myFile.txt
```

Can't bind to local 8602 for debugger (21 minutes ago)

OpenFileInput (with bufferedReader) Example

```kotlin
internal const val FILENAME = "myFile.txt"
//...
// consider try/catch for FileNotFoundException
fun readFile(app: Application): String =
  app.openFileInput(FILENAME)?.bufferedReader().use {
    it?.readText() ?: app.getString(R.string.read_file_failed)
  }

// @Composable fun...
  var readText by remember { mutableStateOf(readFile(app)) }
      // ... write file onClick
      readText = readFile(app)
  //...
  Text(readText)
```

External Storage

- External storage space shared by all the applications
  - It's not always available, because the user can mount the external storage as USB storage (e.g. with a PC) and in some cases remove it from the device (e.g. SD card).
  - files saved here may be read/modified/moved/deleted outside of your control.
  - files are kept when your application is uninstalled.
  - External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

External Storage

- There may be more than one external storage if the device has a built-in external storage which is a partition on the internal disk and/or a SD card: in that case, the built-in storage is the primary external storage.
- Reading files from the external storage requires the `READ_EXTERNAL_STORAGE` permission and writing (inc. reading) files requires the `WRITE_EXTERNAL_STORAGE` permission (in `AndroidManifest.xml`).
    - These permissions are not required if you're reading or writing only files that are in your app-specific external storage directory.

External Storage

- On device with many users (starting with Android 4.4), the external storage is specific to the current user and files for other users can't be accessed.
- Always verify that external storage is available: `getExternalStorageState()`

External Storage

- Public external storage
  - **Deprecated in Android >= 10 (API 29)**
  - `Environment.getExternalStorageDirectory()` - the root directory of the primary external storage of the device that is shared by all applications.
  - `Environment.getExternalStoragePublicDirectory( type: String!)` - public directory for files of a particular type on the primary external storage of the device e.g.
    - `Environment.DIRECTORY_DOCUMENTS`
    - `Environment.DIRECTORY_DOWNLOADS`
    - `Environment.DIRECTORY_PICTURES`
    - `Environment.DIRECTORY_MUSIC`
    - etc.

External Storage

- App specific external storage
  - `Context.getExternalFilesDir()` - the root directory of the primary external storage specific to your application
  - unlike the other directories of the external storage, the files you store in that folder **will be deleted** when your application is uninstalled
  - if you need to store files (of big size) that are only needed by your application you should use this folder
  - On devices that run Android 9 (API 28) or lower, any app can access app-specific files within external storage, provided that the other app has the appropriate storage permissions
  - there is no specific permission needed for the application to read or write to its own external storage starting with Android $>=$ 4.4 (API 19)

File.appendText() - Example

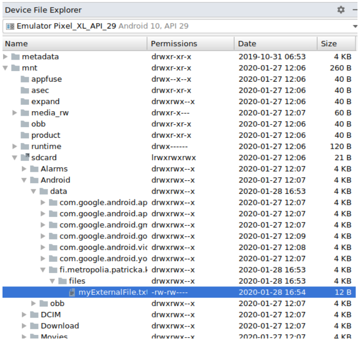
```kotlin
internal const val EXTERNAL_FILENAME = "myExternalFile.txt"
//...
if (Environment.getExternalStorageState() ==
    Environment.MEDIA_MOUNTED) {
  val file = File(app.getExternalFilesDir(null),
      EXTERNAL_FILENAME)
  file.appendText("$text\n")
  text = ""
}
```

Note:

- Destination folder defined by OS
- `appendText(text: String)` will add the text at the end of the file if already exist (create the file before writing otherwise). To overwrite the content of the file if exist, use `writeText(text: String)` instead

External Storage

If using the `getExternalFilesDir()`, then the file will go in a subfolder of the `/mnt` folder. E.g. with USB debugging connected phone:



Device File Explorer

Emulator Pixel_XL_API_29 Android 10, API 29

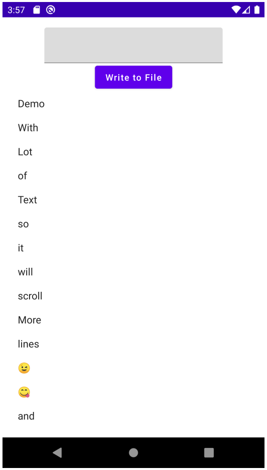
Name	Permissions	Date	Size
metadata	drwxr-xr-x	2019-10-31 06:53	4 KB
mnt	drwxr-xr-x	2020-01-27 12:06	260 B
appfuse	drwx--x--x	2020-01-27 12:06	40 B
asec	drwxr-xr-x	2020-01-27 12:06	40 B
expand	drwxrwx--x	2020-01-27 12:06	40 B
media_rw	drwxr-x---	2020-01-27 12:07	60 B
obb	drwxr-xr-x	2020-01-27 12:06	40 B
product	drwxr-xr-x	2020-01-27 12:06	40 B
runtime	drwx------	2020-01-27 12:06	120 B
sdcard	lrwxrwxrwx	2020-01-27 12:06	21 B
Alarms	drwxrwx--x	2020-01-27 12:07	4 KB
Android	drwxrwx--x	2020-01-27 12:07	4 KB
data	drwxrwx--x	2020-01-28 16:53	4 KB
com.google.android.ap	drwxrwx--x	2020-01-27 12:07	4 KB
com.google.android.ap	drwxrwx--x	2020-01-27 12:07	4 KB
com.google.android.gn	drwxrwx--x	2020-01-27 12:07	4 KB
com.google.android.go	drwxrwx--x	2020-01-27 12:09	4 KB
com.google.android.vic	drwxrwx--x	2020-01-27 12:08	4 KB
com.google.android.yo	drwxrwx--x	2020-01-27 12:07	4 KB
fi.metropolia.patricka.l	drwxrwx--x	2020-01-28 16:53	4 KB
files	drwxrwx--x	2020-01-28 16:53	4 KB
myExternalFile.txt	-rw-rw----	2020-01-28 16:54	12 B
obb	drwxrwx--x	2020-01-27 12:07	4 KB
DCIM	drwxrwx--x	2020-01-27 12:07	4 KB
Download	drwxrwx--x	2020-01-27 12:07	4 KB
Movies	drwxrwx--x	2020-01-27 12:07	4 KB

File.readText() Example

```kotlin
internal const val EXTERNAL_FILENAME = "myExternalFile.txt"
//...
fun readExternalFile(app: Application): String =
  if (Environment.getExternalStorageState() in setOf(
        Environment.MEDIA_MOUNTED,
        Environment.MEDIA_MOUNTED_READ_ONLY
    )
  ) {
    val file = File(app.getExternalFilesDir(null),
        EXTERNAL_FILENAME)
    try {
      file.readText()
    } catch (e: FileNotFoundException) {
      app.getString(R.string.read_file_failed)
    }
  } else app.getString(R.string.storage_not_available)
```

Lab_w3_d3 Internal Storage

- Create an application to read/write to/from file:
  - Have a text field and a button. When the user press the Button, the content of the text field is saved in a file in internal or external storage.
  - Use a lazy list to show the content of the file (so it will automatically scroll if content goes out of screen). Hint: check `readLines()` method from `File` or `BufferedReader`.



Demo
With
Lot
of
Text
so
it
will
scroll
More
lines
😉
😋
and