

Android Developer Fundamentals V2

# Preferences and settings

## Lesson 9



# 9.0 Data Storage

# Contents

- Android File System
- Internal Storage
- External Storage
- SQLite Database
- Other Storage Options

# Storage Options

# Storing data

- [Shared Preferences](#)—Private primitive data in key-value pairs
- [Internal Storage](#)—Private data on device memory
- [External Storage](#)—Public data on device or external storage
- [SQLite Databases](#)—Structured data in a private database
- [Content Providers](#)—Store privately and make available publicly

# Storing data beyond Android

- [Network Connection](#)—On the web with your own server
- [Cloud Backup](#)—Back up app and user data in the cloud
- [Firebase Realtime Database](#)—Store and sync data with NoSQL cloud database across clients in realtime

# Files

# Android File System

- External storage -- Public directories
- Internal storage -- Private directories for just your app

Apps can browse the directory structure

Structure and operations similar to Linux and java.io



# Internal storage

- Always available
- Uses device's filesystem
- Only your app can access files, unless explicitly set to be readable or writable
- On app uninstall, system removes all app's files from internal storage

# External storage

- Not always available, can be removed
- Uses device's file system or physically external storage like SD card
- World-readable, so any app can read
- On uninstall, system does not remove files private to app

# When to use internal/external storage

## Internal is best when

- you want to be sure that neither the user nor other apps can access your files

## External is best for files that

- don't require access restrictions and for
- you want to share with other apps
- you allow the user to access with a computer

# Save user's file in shared storage

- Save new files that the user acquires through your app to a public directory where other apps can access them and the user can easily copy them from the device
- Save external files in public directories



# Internal Storage

# Internal Storage

- Uses private directories just for your app
- App always has permission to read/write
- Permanent storage directory—[getFilesDir\(\)](#)
- Temporary storage directory—[getCacheDir\(\)](#)



# Creating a file

```
File file = new File(  
    context.getFilesDir(), filename);
```

Use standard [java.io](https://java.io) file operators or streams to interact with files

# External Storage



# External Storage

- On device or SD card
- Set permissions in Android Manifest
  - Write permission includes read permission

```
<uses-permission
```

```
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

```
<uses-permission
```

```
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

# Always check availability of storage

```
public boolean isExternalStorageWritable() {  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state)) {  
        return true;  
    }  
    return false;  
}
```

# Example external public directories

- [DIRECTORY\\_ALARMS](#) and [DIRECTORY\\_RINGTONES](#)  
For audio files to use as alarms and ringtones
- [DIRECTORY\\_DOCUMENTS](#)  
For documents that have been created by the user
- [DIRECTORY\\_DOWNLOADS](#)  
For files that have been downloaded by the user

[developer.android.com/reference/android/os/Environment.html](https://developer.android.com/reference/android/os/Environment.html)



# Accessing public external directories

1. Get a path [getExternalStoragePublicDirectory\(\)](#)
2. Create file

```
File path = Environment.getExternalStoragePublicDirectory(  
    Environment.DIRECTORY_PICTURES);  
  
File file = new File(path, "DemoPicture.jpg");
```

# How much storage left?

- If there is not enough space, throws [IOException](#)
- If you know the size of the file, check against space
  - [getFreeSpace\(\)](#)
  - [getTotalSpace\(\)](#).
- If you do not know how much space is needed
  - try/catch [IOException](#)

# Delete files no longer needed

- External storage

```
myFile.delete();
```

- Internal storage

```
myContext.deleteFile(fileName);
```



# Do not delete the user's files!

When the user uninstalls your app, your app's private storage directory and all its contents are deleted

***Do not use private storage for content that belongs to the user!***

For example

- Photos captured or edited with your app
- Music the user has purchased with your app



# Shared Preferences & SQLite Database



# SQLite Database

- Ideal for repeating or structured data, such as contacts
- Android provides SQL-like database
- Covered in following chapters and practicals
  - SQLite Primer
  - Introduction to SQLite Databases
  - SQLite Data Storage Practical
  - Searching an SQLite Database Practical



# Shared Preferences

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- Covered in later chapter and practical
  - Shared Preferences



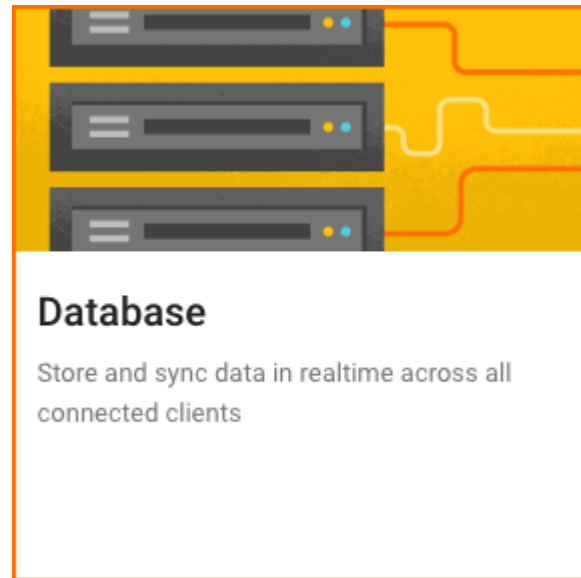
# Other Storage Options

# Use Firebase to store and share data

Store and sync data with the Firebase cloud database

Data is synced across all clients, and remains available when your app goes offline

[firebase.google.com/docs/database/](https://firebase.google.com/docs/database/)



# Firestore Realtime Database

- Connected apps share data
- Hosted in the cloud
- Data is stored as JSON
- Data is synchronized in realtime to every connected client



# Network Connection

- You can use the network (when it's available) to store and retrieve data on your own web-based services
- Use classes in the following packages
  - [java.net.\\*](#)
  - [android.net.\\*](#)

# Backing up data

- Auto Backup for 6.0 (API level 23) and higher
- Automatically back up app data to the cloud
- No code required and free
- Customize and configure auto backup for your app
- See [Configuring Auto Backup for Apps](#)

# Backup API for Android 5.1 (API level 22)

1. Register for Android Backup Service to get a Backup Service Key
2. Configure Manifest to use Backup Service
3. Create a backup agent by extending the BackupAgentHelper class
4. Request backups when data has changed

More info and sample code: [Using the Backup AP](#) and [Data Backup](#)





# Learn more about files

- [Saving Files](#)
- [getExternalFilesDir\(\) documentation and code samples](#)
- [getExternalStoragePublicDirectory\(\) documentation and code samples](#)
- [java.io.File class](#)
- [Oracle's Java I/O Tutorial](#)

# Learn more about backups

- [Configuring Auto Backup for Apps](#)
- [Using the Backup API](#)
- [Data Backup](#)

# What's next?

- Concept Chapter: [9.0 Data Storage](#)
- No practical, this was an overview lecture

# END