Android Developer Fundamentals

Storing Data

Lesson 4

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

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 file operators or streams to interact with files

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

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");
```

Request All files access

An app can request All files access from the user by doing the following:

1. Declare the MANAGE_EXTERNAL_STORAGE permission in the manifest.
2. Use ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION intent action to direct users to a system settings page to enable your app: **access to manage all files**.

https://developer.android.com/training/data-storage/manage-all-files#all-files-access

Request All files access

1. The MANAGE_EXTERNAL_STORAGE permission has some restrictions
2. If your app includes a use case that's similar to the following examples, it's likely to be allowed to request the MANAGE_EXTERNAL_STORAGE permission:
   File managers apps, Backup and restore apps, Document management apps, Anti-virus apps, ...

https://developer.android.com/training/data-storage/manage-all-files#all-files-access

Enable MANAGE_EXTERNAL_STORAGE for testing

1. You can enable the permission for testing purposes
2. Run the following command on the machine that's connected to your test device

```
adb shell appops set --uid com.example.myapplication
MANAGE_EXTERNAL_STORAGE allow
```

https://developer.android.com/training/data-storage/manage-all-files#all-files-access

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

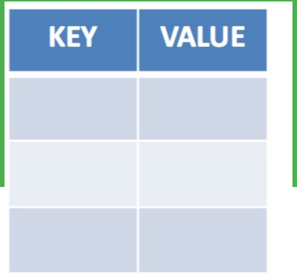
- Photos captured or edited with your app
- Music the user has purchased with your app

Shared Preferences

What is Shared Preferences?

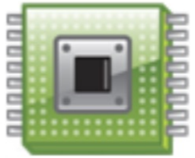
- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- SharedPreference class provides APIs for reading, writing, and managing this data
- Save data in onPause()
  restore in onCreate()

Shared Preferences



KEY	VALUE

1. Data in this type of container is saved as **<Key, Value>** pairs where
   - The key is a string and
   - its associated value must be a primitive data type.
2. Data is stored in the device's internal main memory.
3. PREFERENCES are typically used to keep state information and shared data among several activities of an application.



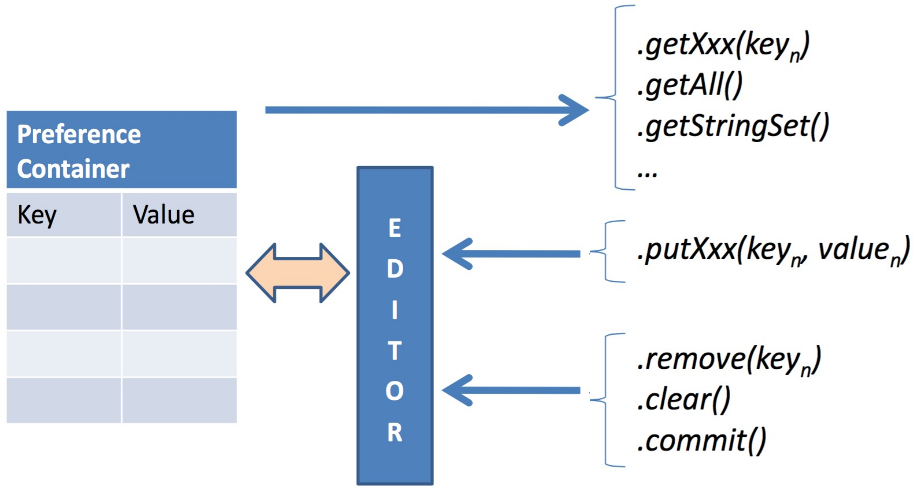
getSharedPreferences()

```
String sharedPrefFile ="hellosharedprefs";
mPreferences = getSharedPreferences(
                                sharedPrefFile,
                                MODE_PRIVATE);
```

Shared Preferences

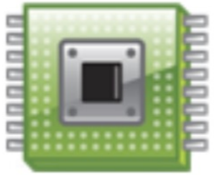
Each of the Preference mutator methods carries a typed-value content that can be manipulated by an **editor** that allows **putXxx**... and **getXxx**... commands to place data in and out of the Preference container.

Xxx = { `Long, Int, Double, Boolean, String` }



Shared Preferences

Key	Value
chosenColor	RED
chosenNumber	7



```
private void usingPreferences(){
  // Save data in a SharedPreferences container
  // We need an Editor object to make preference changes.

1 SharedPreferences myPrefs = getSharedPreferences("my_preferred_choices",
                                                    Activity.MODE_PRIVATE);
  SharedPreferences.Editor editor = myPrefs.edit();
2         editor.putString("chosenColor", "RED");
          editor.putInt("chosenNumber", 7 );
  editor.commit();

  // retrieving data from SharedPreferences container (apply default if needed)
3 String favoriteColor = myPrefs.getString("chosenColor", "BLACK");
  int favoriteNumber = myPrefs.getInt("chosenNumber", 11 );

}
```

Other Storage Options

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

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/

Database

Store and sync data in realtime across all connected clients

Firebase 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 C Storing Data]
- No practical, this was an overview lecture

END