Data Storage

Data Storage Options

- The data storage options in Android are as follows:
  - **Shared preferences** – key-value pairs that store private primitive data. A **key-value pair** (KVP) is a set of two (2) linked data items: a **key**, the unique identifier for some item of data, and the **value**, the data that is identified.
  - **Internal storage** – device memory where private data is stored
  - **External storage** – stores public data
  - **SQLite databases** – private databases used for storing structured data
  - **Room persistence library** – caches an SQLite database locally, and automatically syncs changes to a network database
  - **Cloud backup** – saves user data in the cloud
  - **Firebase realtime database** – stores and syncs data with a NoSQL cloud database. Data is synced across all clients in real time and remains available when your app goes offline.
  - **Custom data store** – stores preferences in a specific storage location using the **Preferences API**. This library manages the user interface and interacts with storage so that you define only the individual settings that the user can configure.

Internal Storage and External Storage

- The table below summarizes the comparison between internal storage and external storage.

Internal Storage	External Storage
Always available	Not always available
The files of an app cannot be accessed by other apps.	Can be accessed by any app
The app's files are removed once the app is uninstalled.	The app's files can be removed using `getExternalFilesDir()`.
It is best to use when you do not want other users or other apps to access your files.	It is best to use if you want to share files with other users or other apps.

- A file can be created in either of the two (2) directories:
  - **Permanent storage**: The **`getFilesDir()`** method returns the absolute path to the directory on the filesystem where files created are stored.
  - **Temporary storage**: The **`getCacheDir()`** method returns the absolute path to the application-specific cache directory on the filesystem. This is recommended for small, temporary files totaling less than 1MB. The system will automatically delete files in the cache directory as disk space is needed elsewhere on the device.
- To create a file:

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

- To write to a file in the internal directory:

```
String filename = "myfile";
String string = "Writing!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

- To access the external storage, indicate a **`uses-permission`** attribute in the Android manifest.
- To write to the external storage:

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

- To read (without writing) the external storage:

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

- To delete a file: **`myFile.delete();`**

Shared Preferences

- Only one (1) shared preferences file is needed for an app.
- The shared preferences file is named with the package name of the app.
- The shared preferences file should be in the `onCreate()` method of the app's main activity and stored in a member variable.
- To create a shared preferences file:

```
private String sharedPrefFile = "com.example.android.myapplication";
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

- To save a shared preferences file:
  1. Get a `SharedPreferences.Editor`, the interface used for modifying values in a `SharedPreferences` object.
  2. Add key/value pairs to the editor using the "put" method appropriate for the data type. Some of the methods are `putInt()`, `putString()`, `putFloat`, and `putBoolean()`.
  3. Call `apply()` to commit the changes. The **`apply()`** method performs the requested modifications, replacing whatever is currently in the shared preferences file.

```
int x = 143;
SharedPreferences.Editor editor = mPreferences.edit();
editor.putInt("num", x);
editor.putString("text", "demo"),
editor.apply();
```

- Use the "get" methods to retrieve the value in the shared preferences file. Some of the methods are `getInt()`, `getString()`, `getFloat`, and `getBoolean()`. These methods take two (2) arguments, one for the key and one for the default value if the key cannot be found.
- To retrieve a shared preferences file:

```
mPreferences = getSharedPreferences(sharedPrefFile,
MODE_PRIVATE);
mNum = mPreferences.getInt("num", 0);
mText = mPreferences.getString("text", null);
```

- To delete a specific key-value pair:

```
SharedPreferences.Editor editor = mPreferences.edit();
editor.remove("num");
editor.apply();
```

- To delete all the key-value pairs:

```
SharedPreferences.Editor editor = mPreferences.edit();
editor.clear();
editor.apply();
```

SQLite Databases

- **SQLite** is one of the most widely deployed database engines in the world. The source code for SQLite is in the public domain. It implements an SQL database engine that has the following characteristics:
  - Self-contained (requires no other components)
  - Serverless (requires no server backend)
  - Zero-configuration (does not need to be configured for your app)
  - Transactional (changes within a single transaction in SQLite either occur completely or not at all)
- The four (4) basic database operations are inserting rows, deleting rows, updating values in rows, and retrieving rows that meet given criteria.
- The **`SQLiteDatabase`** class has methods to execute SQL commands and perform other common database management tasks.
- To create a database:

```
SQLiteDatabase db;
db=openOrCreateDatabase("EmployeeDB", Context.MODE_PRIVATE, null);
```

- To execute an SQL statement:

```
db.execSQL("CREATE TABLE IF NOT EXISTS student(stdnt_id VARCHAR,
stdnt_name VARCHAR, stdnt_prog VARCHAR);");
```

- The **`rawQuery()`** method runs the provided SQL and returns a `Cursor` of the result set. A **cursor** is a pointer to a table row.

```
Cursor c = db.rawQuery("SELECT * FROM student WHERE stdnt_id='"+
etStdntID.getText()+"'", null);
```

References:

DiMarzio, J. (2017). *Beginning Android programming with Android Studio*. Indiana: John Wiley & Sons, Inc.

Google Developers Training Team. (2018). *Android developer fundamentals (version 2)*. Retrieved from https://google-developer-training.github.io