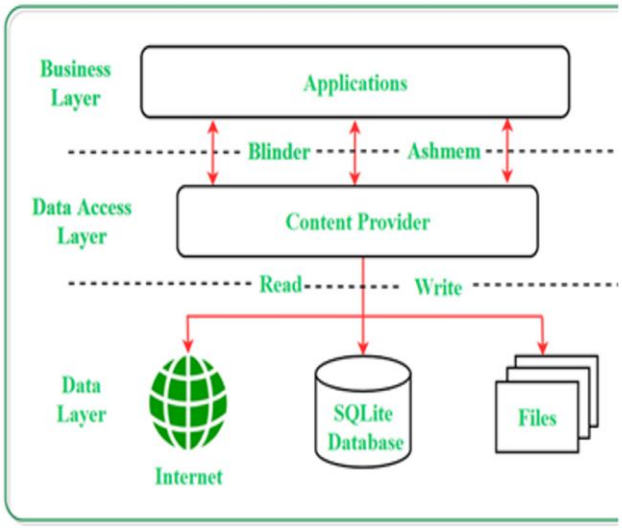
Content Provider



Content URI

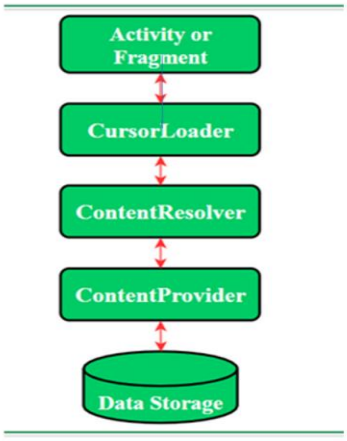
Content URI(Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

Four fundamental operations are possible in Content Provider namely **Create**, **Read**, **Update**, and **Delete**. These operations are often termed as **CRUD operations**.

- **Create:** Operation to create data in a content provider.
- **Read:** Used to fetch data from a content provider.
- **Update:** To modify existing data.
- **Delete:** To remove existing data from the storage.

1. Working of the Content Provider
2. UI components of android applications like Activity and Fragments use an object CursorLoader to send query requests to ContentResolver. The ContentResolver object sends requests (like create, read, update, and delete) to the ContentProvider as a client. After receiving a request, ContentProvider process it and returns the desired result. Below is a diagram to represent these processes in pictorial form.

3.



query()	A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object.
insert()	To insert a new row in the database of the content provider. It returns the content URI of the inserted row.
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method is used to delete the existing rows. It returns the number of rows deleted.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialise the pr

Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.
In this chapter we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.
By default these files are private and are accessed by only your application and get deleted , when user delete your application.

Writing file

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c.
Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file

In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```

After that, you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
  temp = temp + Character.toString((char)c);
}
//string temp contains all the data of the file.
fin.close();
```

Apart from the the methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files. These methods are listed below –

Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream

Apart from the the methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below –

Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

Example

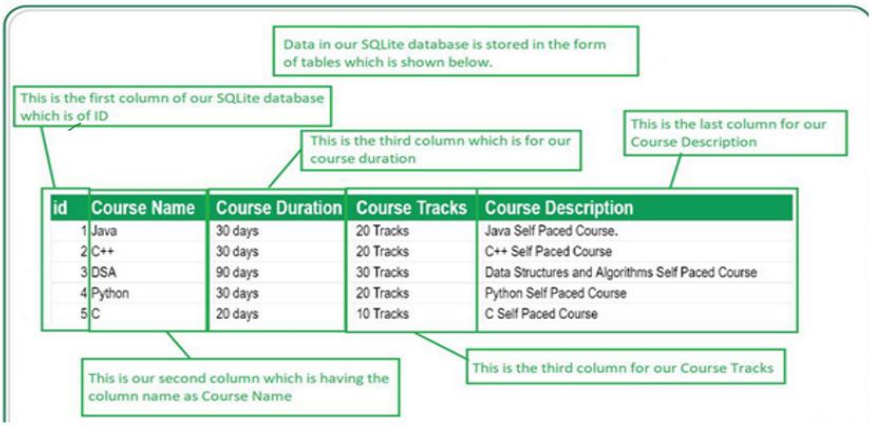
Here is an example demonstrating the use of internal storage to store and read files. It creates a basic storage application that allows you to read and write from internal storage.
To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Android Studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Run the application and choose a running android device and install the application on it and verify the results

What is SQLite Database?

SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file. We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data. SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

Data is stored in the SQLite database in the form of **tables**. When we stored this data in our SQLite database it is arranged in the form of tables that are similar to that of an excel sheet. Below is the representation of our SQLite database which we are storing in our SQLite database.



id	Course Name	Course Duration	Course Tracks	Course Description
1	Java	30 days	20 Tracks	Java Self Paced Course.
2	C++	30 days	20 Tracks	C++ Self Paced Course
3	DSA	90 days	30 Tracks	Data Structures and Algorithms Self Paced Course
4	Python	30 days	20 Tracks	Python Self Paced Course
5	C	20 days	10 Tracks	C Self Paced Course

Bluetooth is a way to exchange data with other devices wirelessly. Android provides Bluetooth API to perform several tasks such as:

scan bluetooth devices
connect and transfer data from and to other devices
manage multiple connections etc.

BluetoothAdapter class
By the help of BluetoothAdapter class, we can perform fundamental tasks such as initiate device discovery, query a list of paired (bonded) devices, create a BluetoothServerSocket instance to listen for connection requests etc.

Constants of BluetoothAdapter class
BluetoothAdapter class provides many constants. Some of them are as follows:

- String ACTION_REQUEST_ENABLE
- String ACTION_REQUEST_DISCOVERABLE
- String ACTION_DISCOVERY_STARTED
- String ACTION_DISCOVERY_FINISHED

Commonly used methods of BluetoothAdapter class are as follows:

- **static synchronized BluetoothAdapter getDefaultAdapter()** returns the instance of BluetoothAdapter.
- **boolean enable()** enables the bluetooth adapter if it is disabled.
- **boolean isEnabled()** returns true if the bluetooth adapter is enabled.
- **boolean disable()** disables the bluetooth adapter if it is enabled.
- **String getName()** returns the name of the bluetooth adapter.
- **boolean setName(String name)** changes the bluetooth name.
- **int getState()** returns the current state of the local bluetooth adapter.
- **Set<BluetoothDevice> getBondedDevices()** returns a set of paired (bonded) BluetoothDevice objects.
- **boolean startDiscovery()** starts the discovery process.

Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

1. **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
2. **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
3. **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
4. **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
5. **None:** This type displays an empty grid with no tiles loaded.

```
1. googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
2. googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
3. googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
4. googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation.
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.
snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

Callback methods in Google Map

1. **OnMapRreadyCallback:** This callback interface invokes when it instance is set on MapFragment object. The onMapReady(GoogleMap) method of OnMapReadyCallback interface is called when the map is ready to used. In the onMapReady(GoogleMap) method we can add markers, listeners and other attributes.
2. **LocationListener:** This interface is used to receive notification when the device location has changed. The abstract method of LocationListener onLocationChanged(Location) is called when the location has changed.
3. **GoogleApiClient.ConnectionCallbacks:** This interface provide callbacks methods onConnected(Bundle) and onConnectionSuspended(int) which are called when the device is to connected and disconnected.
4. **GoogleApiClient.OnConnectionFailedListener:** This interface provide callbacks method onConnectionFailed(ConnectionResult) which is called when there was an error in connecting the device to the service.

Android Google Map Search Location using Geocoder

Now we will implement location search functionality in Google Map.
Searching location in Google Map API is done through **Geocoder** class. Geocoder class is used to handle *geocoding* and *reverse geocoding*.
Geocoding is a process in which street address is converted into a coordinate (latitude,longitude). Reverse geocoding is a process in which a coordinate (latitude,longitude) is converted into an address.