Sql lite

SQLite is a opensource SQL database that stores data to a text file on a device Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database
name",MODE_PRIVATE,null);
```

Apart from this , there are other functions available in the database package , that does this job. They are listed below

Sr.No	Method & Description
1	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)

	This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags) It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory) It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory) This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column

3	**getColumnName(int columnIndex)** **This method returns the name of the column by specifying the index of the column**
4	**getColumnNames()** **This method returns the array of all the column names of the table.**
5	**getCount()** **This method returns the total number of rows in the cursor**
6	**getPosition()** **This method returns the current position of the cursor in the table**
7	**isClosed()** **This method returns true if the cursor is closed and return false otherwise**

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

Advantages of SQLite

SQLite database only contains a single file on the disk which makes them portable to any other database. Some organisations use more than one databases in their applications, so it's really important the database to be portable or not. SQLite can be used for Native and Cross-platform applications as well. So, either you're developing your application on React Native or Java, you can use SQLite.

Below are some more reasons on why should you use SQLite:

- **SQLite uses SQL, so it has all the features of a standard SQL database.**
- **Some developers require databases which can scale and provide support for concurrency. SQLite, with its rich feature, can be linked with any application in production.**
- **Often developers find it hard to perform testing when the application database is messy. SQLite is very good for testing.**
- **Zero-configuration: SQLite doesn't need any complex set up to store the data. When you build Native applications with Java, it comes integrated with the platform.**
- **Developers call SQLite, a serverless database and it really lives up to the expectation. You don't need to set up any API or install any library to access data from SQLite.**
- **SQLite is cross-platform which means that it can be used on Android application built on Java, and as well as cross-platform application built on React Native.**

Disadvantages of SQLite

The main drawback of using SQLite is there is no user management. Any user can read/write the data without any special access. Any activity or process in your application can have direct access to the stored data. Security is a big concern in SQLite. The stored data can be injected easily at any time.

Furthermore, most of the advance databases can be configured in a way to achieve expected query performance. SQLite, on the other side, offers limited scope to tune the performance in a complex scenario.

When To Use SQLite

SQLite, with its rich set of features, can be used in multiple use-cases.

When your mobile application needs portability

All applications that need portability, that do not require expansion, e.g. single-user local applications, mobile applications or games.

When your application needs direct access to the server

In many cases, applications that need to read/write files to the server directly can benefit from switching to SQLite for additional functionality and simplicity that comes from using the Structured Query Language.

When your application goes through heavy testing

It is an overkill for a large portion of applications to use an additional process for testing the business-logic and the functionality.

When Not To Use SQLite

Multi-user applications

If you are working on an application whereby multiple clients need to access and use the same database, a fully-featured RDBM is probably better to choose over SQLite.

Applications requiring high write volumes

One of the limitations of SQLite is the write operations. This DBMS allows only one single write operating to take place at any given time, hence allowing a

limited throughput. So now, you know the pros and cons of SQLite using in mobile app development. Do you have any other issue to share? Let me know in comments

Canvas

Diving into using the Android Canvas class can unlock magical super powers you never knew you had . Imagine being able to draw *anything your heart desires just with some basic shapes, paths and bitmaps? Well, the Android Canvas gives you just that ability.**

What is a Canvas?

Canvas is a class in Android that performs 2D drawing of different objects onto the screen. The saying "a blank canvas" is very similar to what a Canvas object is on Android. It is basically, an empty space to draw onto.

The Canvas class is not a new concept, this class is actually wrapping a SKCanvas under the hood. The SKCanvas comes from SKIA, which is a 2D Graphics Library that is used on many different platforms. SKIA is used on platforms such as Google Chrome, Firefox OS, Flutter, Fuschia etc. Once you understand how the Canvas works on Android, the same drawing concepts apply to many other different platforms.

It is useful to know that SKIA is used in the underlying code for Android, so when you get stuck trying to understand how a certain API works, you can look at the source for SKIA to gain a deeper understanding.

Canvas Coordinate System

The coordinate system of the Android canvas starts in the top left corner, where [0,0] represents that point. The `y` axis is positive downwards, and `x` axis positive towards the right.

All elements drawn on a canvas are placed relative to the [0,0] point.

When working with the Canvas, you are working with px and not dp, so any methods such as translating, or resizing will be done in pixel sizes. This means you need to translate any dp values into px before calling any canvas operations. This will ensure that your drawing looks consistent across devices with different pixel densities.

Canvas draw commands will draw over previously drawn items. The last draw command will be the topmost item drawn onto your canvas. It is up to you to ensure that your items are laid out correctly (Alternatively, you might want to use some of the built-in layout mechanisms for this — such as `LinearLayout`).

How do I use a Canvas?

To draw onto a canvas in Android, you will need four things:

1. **A bitmap or a view — to hold the pixels where the canvas will be drawn.**
2. **Canvas — to run the drawing commands on.**
3. **Drawing commands — to indicate to the canvas what to draw.**
4. **Paint — to describe how to draw the commands.**

Get access to a Canvas instance

In order to get access to a `Canvas` instance, you will need to create a class that extends from `View`. This will then allow you to override the `onDraw` method, which has a `Canvas` as a parameter.

```
class CustomView @JvmOverloads constructor(context: Context,

   attrs: AttributeSet? = null, defStyleAttr: Int = 0)

   : View(context, attrs, defStyleAttr) {
```

```
   // Called when the view should render its content.

   override fun onDraw(canvas: Canvas?) {

       super.onDraw(canvas)

       // DRAW STUFF HERE

   }

}
```

You can then include this view inside your layout XML and this will then automatically invoke the `onDraw` method of the Custom View.

```
<za.co.riggaroo.customviews.CustomView

   android:layout_width="200dp"

   android:layout_height="300dp"

   app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintLeft_toLeftOf="parent"

    app:layout_constraintRight_toRightOf="parent"

    app:layout_constraintTop_toTopOf="parent"/>
```

You can also get access to a Canvas object by programatically creating one in code, like this:

```
val bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)

val canvas = Canvas(bitmap)
```

It’s worth noting at this point that any Canvas created programmatically without using a View, will be software rendered and not hardware rendered. This can affect the appearance of some of the drawing commands. For instance, some commands are just not supported with hardware rendering, or only supported from a certain API level. For more information about the differences between hardware rendering and software rendering, read this post.

What can I draw on a Canvas?

There are many different things you can draw onto a `Canvas`. One of the most common drawing operations is to draw a bitmap (image) onto the canvas. The method for doing this is just called `drawBitmap` and it takes in the bitmap object that is loaded up either with Android's built-in mechanisms, or with Glide.

```
canvas.drawBitmap(bitmap, null, rect, paint)
```

The second parameter here allows us to pass in the portion of the bitmap that you want to render, when passing in `null` the whole bitmap will be rendered. The third parameter is a `RectF` object which represents the scale and translation of the bitmap that you want to draw on screen.

Tip: Make sure your `RectF` object that you pass into the `drawBitmap` function is scaled with the correct aspect ratio otherwise your output may be stretched

You need to be careful with this, since it can quite easily stretch the bitmap, as the Canvas calls don't take into account the aspect ratio of the provided image. You need to ensure the `rect` that is passed in is properly scaled. The fourth parameter is the `paint` object, we will cover the purpose of this parameter soon.

There are many other `Canvas` drawing methods that can give you some great looking views. We won't be covering them all here, but here are two other examples of drawing methods on the `Canvas` class:

To draw a circle onto the view, give it a center point `x,y` , its `size` and a`paint` object:

```
canvas.drawCircle(x, y, size, paint)
```

Another one is the `drawRect()` method. This draws a rectangle on screen:

```
canvas.drawRect(rect, paint)
```

This is not the full list of drawing methods but just a small highlight of some of them to get you more comfortable with the concepts, feel free to browse the Canvas documentation for a comprehensive list of all of them.

Paint □

In my opinion, the `Paint` class is possibly the most interesting graphics class and it is also my favourite, for that very reason. There is so much that a `Paint` object can do that

can really make your drawing operations shine. ✨

The `Paint` class typically holds colour and style information. The `Paint` object is then used for drawing objects (i.e. bitmap, text, paths etc) onto a `Canvas`.

To create a `Paint` object:

```
private val textPaint =

  Paint().apply {

      isAntiAlias = true

      color = Color.RED

      style = Paint.Style.STROKE

  }
```

This object should be created before using it in Canvas#onDraw(). It is not recommended to create it in onDraw() since you shouldn't be doing object allocations in that method.

The isAntiAlias flag is quite an important one. If you are drawing objects to your canvas and you notice that the edges of your objects have jagged edges, it is likely that you haven't set this flag to true. This flag indicates to the paint to smooth out the edges of the object you are drawing to the canvas.

The Paint class has more than just those three properties, there are many more things you can do with it. For instance, you can also set properties related to text rendering, such as the typeface, letterSpacing(kerning) and textSize.

```
private val textPaint =

  Paint().apply {

     isAntiAlias = true

     textSize = fontSize

     letterSpacing = letterSpace

     typeface = newTypeface

     setShadowLayer(blurValue, x, y, Color.BLACK)

  }
```

It is worth noting that the `Paint#setShadowlayer()` method doesn't work consistently across API levels and drawing commands. It works when drawing text on a Canvas, but applying the shadow to other commands such as `drawBitmap` doesn't yield the same results across API levels.

The reason for the inconsistency between API levels is because the Canvas APIs are bundled with the Android Platform and therefore are not updated until the OS is updated. See the list on this page for more information about which APIs work on which Android versions.

Once you've created the Paint object, pass that object into your Canvas#draw*() calls and the drawing will then take on the properties you've specified in the paint.

Shared Preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedpreferences =
getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
```

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes available that are listed below −

Sr.No	Mode & description
1	**MODE_APPEND** **This will append the new preferences with the already existing preferences**
2	**MODE_ENABLE_WRITE_AHEAD_LOGGING**

	Database open flag. When it is set , it would enable write ahead logging by default
3	**MODE_MULTI_PROCESS** **This method will check for modification of preferences even if the sharedpreference instance has already been loaded**
4	**MODE_PRIVATE** **By setting this mode, the file can only be accessed using calling application**
5	**MODE_WORLD_READABLE** **This mode allow other application to read the preferences**
6	**MODE_WORLD_WRITEABLE** **This mode allow other application to write the preferences**

You can save something in the sharedpreferences by using SharedPreferences.Editor class. You will call the edit method of SharedPreference instance and will receive it in an editor object. Its syntax is −

```
Editor editor = sharedpreferences.edit();
```

```
editor.putString("key", "value");
editor.commit();
```

Apart from the putString method , there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows −

Sr. NO	Mode & description
1	**apply()** **It is an abstract method. It will commit your changes back from editor to the sharedPreference object you are calling**
2	**clear()** **It will remove all values from the editor**
3	**remove(String key)** **It will remove the value whose key has been passed as a parameter**
4	**putLong(String key, long value)** **It will save a long value in a preference editor**

5	**putInt(String key, int value)** **It will save a integer value in a preference editor**
6	**putFloat(String key, float value)** **It will save a float value in a preference editor**

File i/o

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**

Multimedia – Audio/Video playback and record

Playing Audio

In terms of audio playback, most implementations of Android support AAC LC/LTP, HE-AACv1 (AAC+), HE-AACv2 (enhanced AAC+), AMR-NB, AMR-WB, MP3, MIDI, Ogg Vorbis, and PCM/WAVE formats.

Audio playback can be performed using either the MediaPlayer or the AudioTrack classes. AudioTrack is a more advanced option that uses streaming audio buffers and provides greater control over the audio. The MediaPlayer class, on the other hand, provides an easier programming interface for implementing audio playback and will meet the needs of most audio requirements.

The MediaPlayer class has associated with it a range of methods that can be called by an application to perform certain tasks. A subset of some of the key methods of this class is as follows:

- **create() – Called to create a new instance of the class, passing through the Uri of the audio to be played.**
- **setDataSource() – Sets the source from which the audio is to play.**
- **prepare() – Instructs the player to prepare to begin playback.**
- **start() – Starts the playback.**
- **pause() – Pauses the playback. Playback may be resumed via a call to the resume() method.**
- **stop() – Stops playback.**
- **setVolume() – Takes two floating-point arguments specifying the playback volume for the left and right channels.**
- **resume() – Resumes a previously paused playback session.**
- **reset() – Resets the state of the media player instance. Essentially sets the instance back to the uninitialized state. At a minimum, a reset player will need to have the data source set again and the prepare() method called.**
- **release() – To be called when the player instance is no longer needed. This method ensures that any resources held by the player are released.**

In a typical implementation, an application will instantiate an instance of the MediaPlayer class, set the source of the audio to be played and then call prepare() followed by start(). For example:

```
MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setDataSource("https://www.yourcompany.com/myaudio
.mp3");
mediaPlayer.prepare();
mediaPlayer.start();
```

Recording Audio and Video using the MediaRecorder Class

As with audio playback, recording can be performed using a number of different techniques. One option is to use the MediaRecorder class, which, as with the MediaPlayer class, provides a number of methods that are used to record audio:

- **setAudioSource() – Specifies the source of the audio to be recorded (typically this will be MediaRecorder.AudioSource.MIC for the device microphone).**

- setVideoSource() – Specifies the source of the video to be recorded (for example MediaRecorder.VideoSource.CAMERA).
- setOutputFormat() – Specifies the format into which the recorded audio or video is to be stored (for example MediaRecorder.OutputFormat.AAC_ADTS).
- setAudioEncoder() – Specifies the audio encoder to be used for the recorded audio (for example MediaRecorder.AudioEncoder.AAC).
- setOutputFile() – Configures the path to the file into which the recorded audio or video is to be stored.
- prepare() – Prepares the MediaRecorder instance to begin recording.
- start() - Begins the recording process.
- stop() – Stops the recording process. Once a recorder has been stopped, it will need to be completely reconfigured and prepared before being restarted.
- reset() – Resets the recorder. The instance will need to be completely reconfigured and prepared before being restarted.
- release() – Should be called when the recorder instance is no longer needed. This method ensures all resources held by the instance are released.

A typical implementation using this class will set the source, output and encoding format and output file. Calls will then be made to the prepare() and start() methods. The stop() method will then be called when recording is to end, followed by the reset() method. When the application no longer needs the recorder instance, a call to the release() method is recommended:

```
MediaRecorder mediaRecorder = new MediaRecorder();

mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.AAC_A
DTS);
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
mediaRecorder.setOutputFile(audioFilePath);

mediaRecorder.prepare();
mediaRecorder.start();
.
.
mediaRecorder.stop();
mediaRecorder.reset();
mediaRecorder.release();
```

In order to record audio, the manifest file for the application must include the android.permission.RECORD_AUDIO permission:

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

LOCATION AWARNESS

This becomes possible with the help of Google Play services, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The Location object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information −

Sr.No.	Method & Description
1	**float distanceTo(Location dest)** **Returns the approximate distance in meters between this location and the given location.**

2	**float getAccuracy()** **Get the estimated accuracy of this location, in meters.**
3	**double getAltitude()** **Get the altitude if available, in meters above sea level.**
4	**float getBearing()** **Get the bearing, in degrees.**
5	**double getLatitude()** **Get the latitude, in degrees.**
6	**double getLongitude()** **Get the longitude, in degrees.**
7	**float getSpeed()** **Get the speed if it is available, in meters/second over ground.**

8	**boolean hasAccuracy()** **True if this location has an accuracy.**
9	**boolean hasAltitude()** **True if this location has an altitude.**
10	**boolean hasBearing()** **True if this location has a bearing.**
11	**boolean hasSpeed()** **True if this location has a speed.**
12	**void reset()** **Clears the contents of the location.**
13	**void setAccuracy(float accuracy)** **Set the estimated accuracy of this location, meters.**

14	**void setAltitude(double altitude)** **Set the altitude, in meters above sea level.**
15	**void setBearing(float bearing)** **Set the bearing, in degrees.**
16	**void setLatitude(double latitude)** **Set the latitude, in degrees.**
17	**void setLongitude(double longitude)** **Set the longitude, in degrees.**
18	**void setSpeed(float speed)** **Set the speed, in meters/second over ground.**
19	**String toString()** **Returns a string containing a concise, human-readable description of this object.**

Get the Current Location

To get the current location, create a location client which is LocationClient object, connect it to Location Services using connect() method, and then call its getLastLocation() method. This method returns the most recent location in the form of Location object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces −

- **GooglePlayServicesClient.ConnectionCallbacks**
- **GooglePlayServicesClient.OnConnectionFailedListener**

These interfaces provide following important callback methods, which you need to implement in your activity class −

Sr.No.	Callback Methods & Description
1	**abstract void onConnected(Bundle connectionHint)** **This callback method is called when location service is connected to the location client successfully. You will use connect() method to connect to the location client.**
2	**abstract void onDisconnected()** **This callback method is called when the client is disconnected. You will use disconnect() method to disconnect from the location client.**

3	**abstract void onConnectionFailed(ConnectionResult result)** **This callback method is called when there was an error connecting the client to the service.**

You should create the location client in onCreate() method of your activity class, then connect it in onStart(), so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in onStop() method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement LocationListener interface as well. This interface provide following callback method, which you need to implement in your activity class −

Sr.No.	Callback Method & Description
1	**abstract void onLocationChanged(Location location)** **This callback method is used for receiving notifications from the LocationClient when the location has changed.**

Location Quality of Service

The LocationRequest object is used to request a quality of service (QoS) for location updates from the LocationClient. There are following useful setter

methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

Sr.No.	Method & Description
1	setExpirationDuration(long millis) Set the duration of this request, in milliseconds.
2	setExpirationTime(long millis) Set the request expiration time, in millisecond since boot.
3	setFastestInterval(long millis) Explicitly set the fastest interval for location updates, in milliseconds.
4	setInterval(long millis) Set the desired interval for active location updates, in milliseconds.
5	setNumUpdates(int numUpdates) Set the number of location updates.

6	**setPriority(int priority)** **Set the priority of the request.**

Now for example, if your application wants high accuracy location it should create a location request with setPriority(int) set to PRIORITY_HIGH_ACCURACY and setInterval(long) to 5 seconds. You can also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.

Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have Location object, you can use Geocoder.getFromLocation() method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the doInBackground() method of an AsyncTask class.

The AsyncTask must be subclassed to be used and the subclass will override doInBackground(Params...) method to perform a task in the background and onPostExecute(Result) method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is execute(Params... params), this method executes the task with the specified parameters.