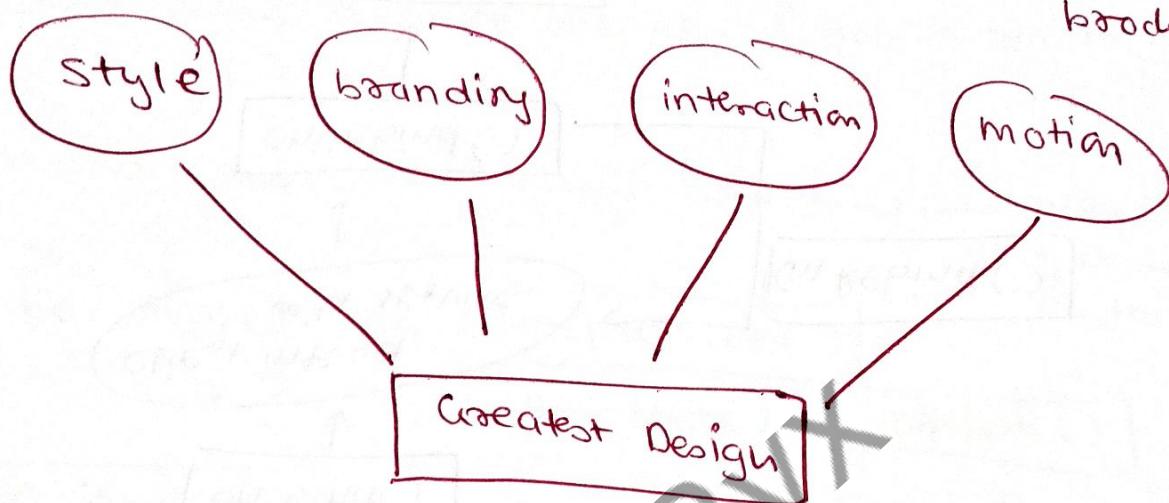


## Unit - 3

### Material Design

↳ system for building bold & beautiful digital products.



For Android:

TMDC

Material

Design

Component

↳ updates library over time

available on web, ios & flutter.



Annexure No :

Linear Layout:



A view group

that aligns all children in  
either vertical & horizontally

① android:id → identify layout

② android:baselineAligned

↳ true or false.

③ android:divider

use vertical divider betn  
buttons.

④ android:weightSum

sum of child weight

⑤ orientation

↳ horizontal

, vertical

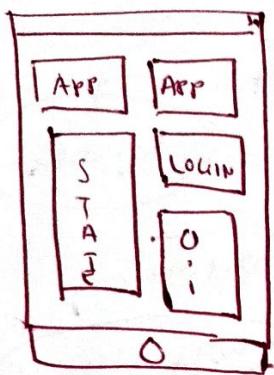
→ default → horizontal

⑥ gravity

top, bottom, center, etc.

## Relative layout:

C → enables you to specify how child views are positioned to each other.



- id
- gravity
- ignore gravity
- layout\_alignBottom
- layout\_above

layout\_alignBottom

" left

" parent bottom

→ makes bottom edge of this view  
make left edge of this view match  
True / false.

## TABLE LAYOUT : Rows & column

Eg. calculator

collapse columns

shrink columns

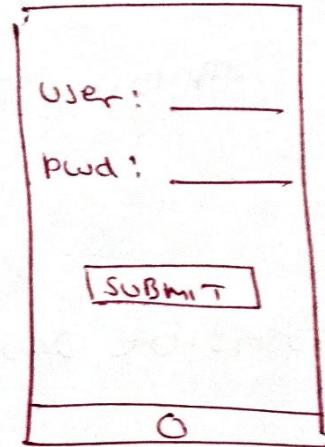
stretch columns

Annexure No :

## Absolute Layout



- exact location of its children.
- flexibility is less
- harder maintenance.



layout-x

layout-y

## Frame layout :

- designed to block out an area on the screen to display a single item.

- difficult to organize.
- scalable w/o overlapping

foreground

↳ drawable

foreground gravity

↳ define gravity to apply to drawable.

## Event Handling:

Events are used to collect data about a user's interactions with APP Components.

- 3 types of Event management :

### ① Event Listener:

an interface in the view class.

Called for viewing which listener is registered when user triggered the interaction.

Event registration  $\Rightarrow$  Process by which an Event handler gets registered w/ Event Listener.

### ② Event Handler:

When an Event happens & we have registered a listener for that event,

The event listener calls the Event Handler

→ Actually handles the event.

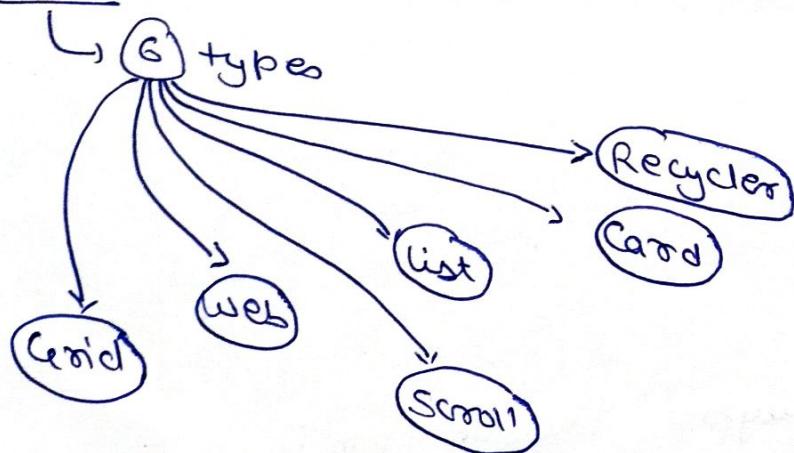
## Widgets :

- small gadget → Placed on homescreen.
- handy
- allows you to put your favourite apps on the home screen.
  - ↓
  - easy access to them
- e.g. Clocks,  
weather, music

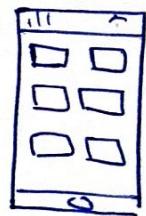
Cyph3rRyx

Chapter 4

## MAD - Working with views &amp; fragment.

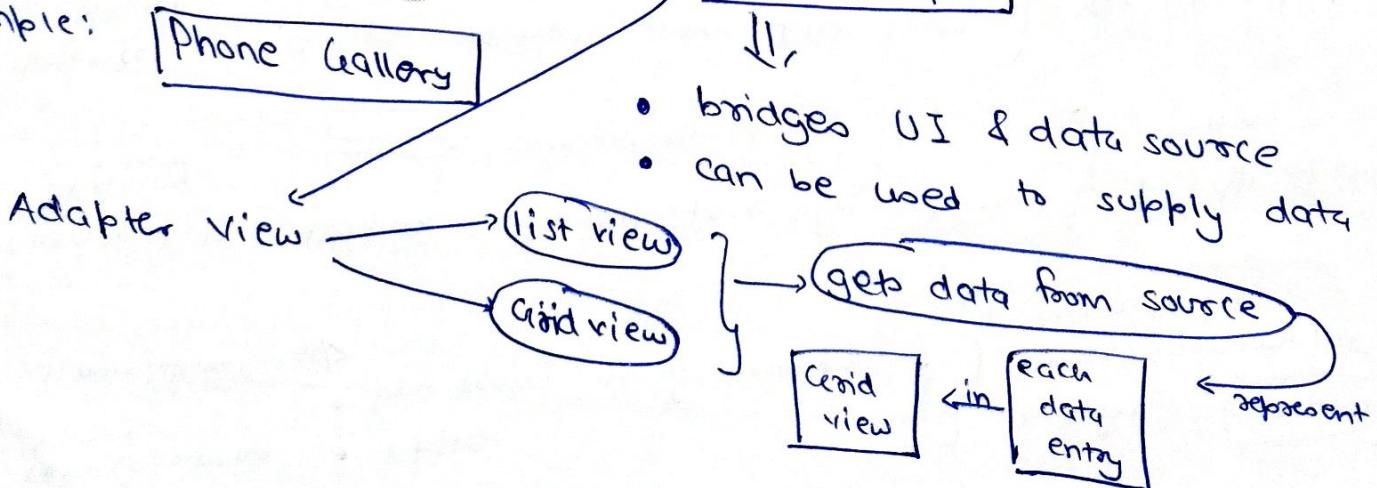
ViewsGrid View:

- display in 2-D scrolling grid (rows & columns)
- The grid items → automatically inserted to layout



Note: ✓ Default scrollable

## Example:



Components: ① Id : <Grid View

android: id = "@+id / simpleGrid"

/ >

② numColumns : • integer

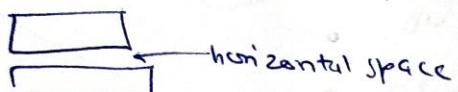
• how many no. of columns

< android: numColumns = "4" />

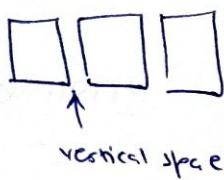
Autofit = auto-fit

display as many possible column to fit

③ Horizontal Spacing : Defines horizontal space b/w columns } px, dp, sp, in, or mm



④ Vertical Spacing : Defines vertical space b/w rows } px, dp, sp, in, or mm.



⑤ Stretch Mode:

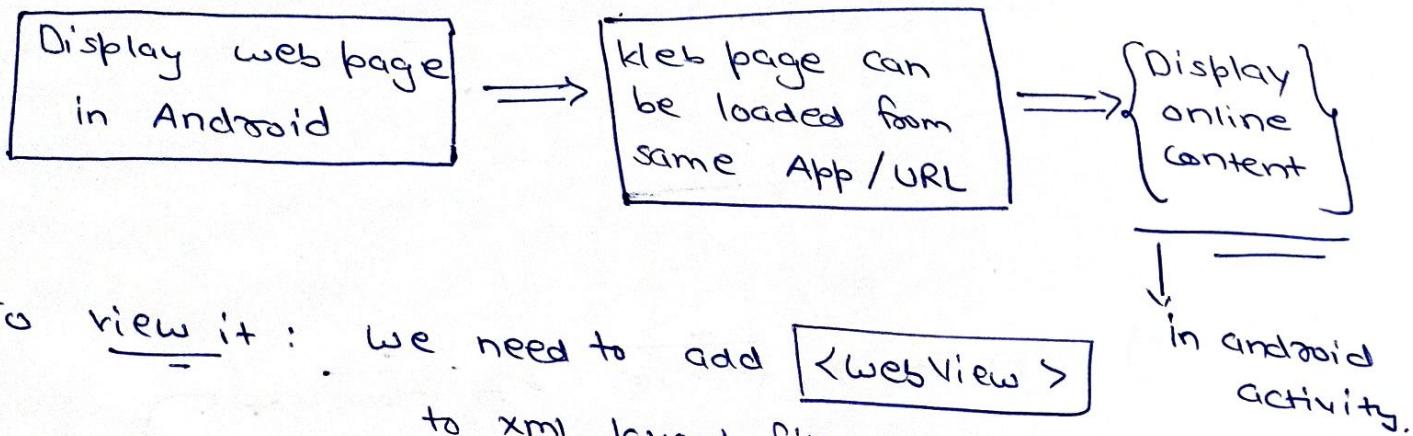
if => none: stretching disabled

spacingWidth: space b/w each column is stretched.

Column Width: columns are stretched equally.

spacingWidthUniform: uniformly stretched columns.

(2) Web View:



- To view it : we need to add `<WebView>` to xml layout file

Uses : `WebKit` engine to display web page.

→ `android.webkit.WebView` → subclass of `AbsoluteLayout` class.  
`{loadUrl()  
loadData()}` → methods → for adding & displaying web page

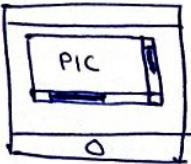
Create an object in java file:

`WebView browser = (WebView) findViewById(R.id.webview);`

To load a web url into webview :

`browser.loadUrl ("http://www.ayx.com");`

### ③ ScrollView:

- Useful to add a horizontal / vertical scroll bar  
  
A very large picture or content
- The android ScrollView can hold any one direct child.
- Enable ScrollView  $\xrightarrow{\text{for}}$  scrolling
- disable ScrollView  $\xrightarrow[\text{using}]{\text{for}}$  List View & Grid View  
because they take care of their own scrolling.
- Default: vertical view of ScrollView  
(only support)
- for horizontal scrolling : Horizontal ScrollView is called
- Android: fillViewPort  
↳ to stretch the content to fill view port.

## (4) List View:

- List of scrollable items  $\Rightarrow$  displayed via ListView.

• Helps in displaying data in form of a scrollable list.

• selection click on any item.

• Example Phone Contact Book

Contacts display in List  
Select and call

items inserted  
via

ADAPTER

pulls data from source

< ListView

    android: id = "@+id/list view"

    android: layout - width = "match - parent"

    android: layout - height = "match - parent"

    android: divider = "@color / teal - 70"

    android: divider - height = "1dp"

    android: list selector = "#000000" >

< /ListView >

SOURCE SAMPLE

uniquely identify ID

draw b/w list items

specify height

set the selector of  
list view

(generally orange/  
blue)

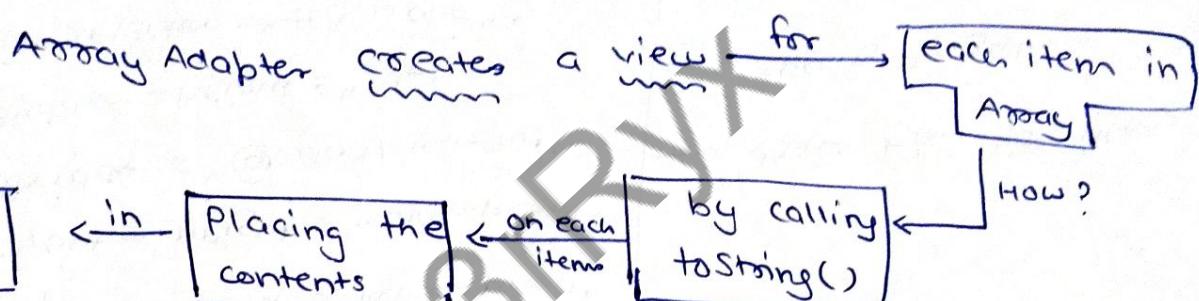
## ADAPTER's In List View:

Array adapter : to extract data items from the blocks of array data structure we use

For instance:

- list of phone contacts,
- " " countries,
- " " names

Default:



Example:

Let's say you have

Array of Strings

LIST  
VIEW

you want to  
DISPLAY IT IN

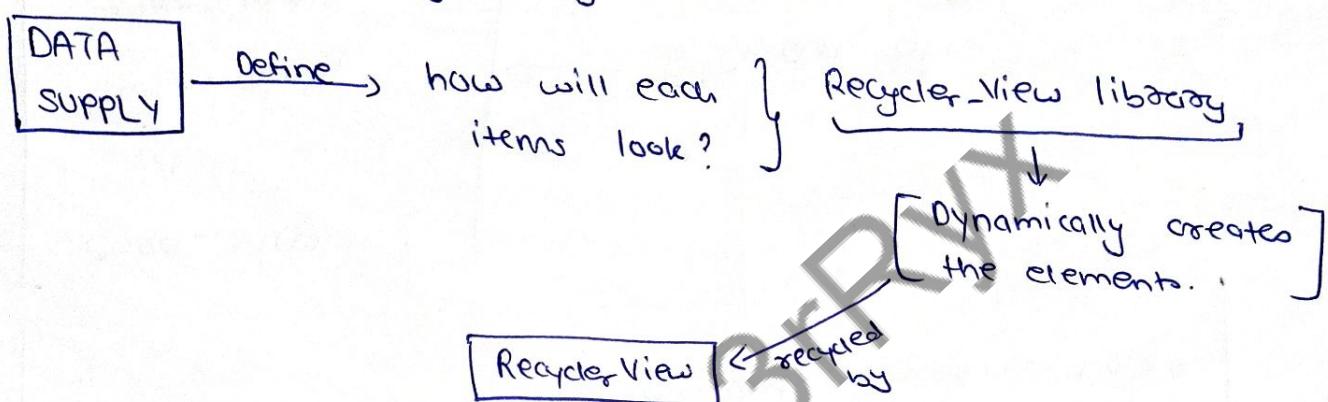
St initialize "Array Adapter" using constructor to specify the layout of each string

Array Adapter = new

Array Adapter<string> (this, R.layout.ListView, String Array);

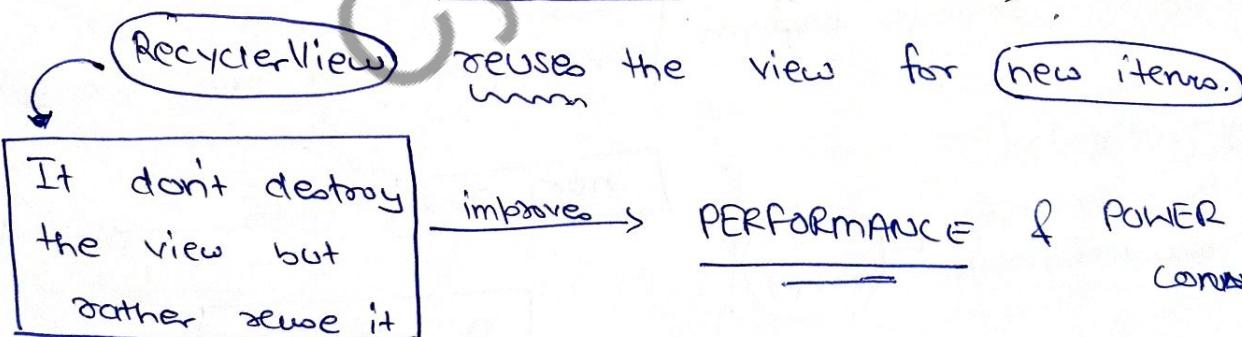
## ⑤ Recycler View:

- Advanced version of Listview → improved performance
- mostly used to design UI w/ fine-grain control over the lists and grids of android app.
- easy → Display large sets of data.



### Explanation:

- ① When item is scrolled off the screen,



## ⑥ Card View:

display: Styled containers

↳ often used in "lists" to hold each items information.

introduced

in : Android 5.0

helps in : giving rich look

} Added via → ViewGroup manner in our Activity using layout XML file.

→ Represent info in Card & shadow manner

w/ drop shadow

called elevation

dependencies :

implementation "androidx.cardview:cardview:1.0.0"

Card\_view

id

layout\_gravity

layout\_width

layout\_height

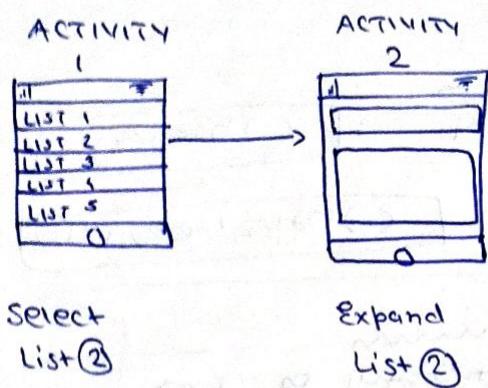
Card corner radius

→ Elements of

Card View

FRAGMENT :

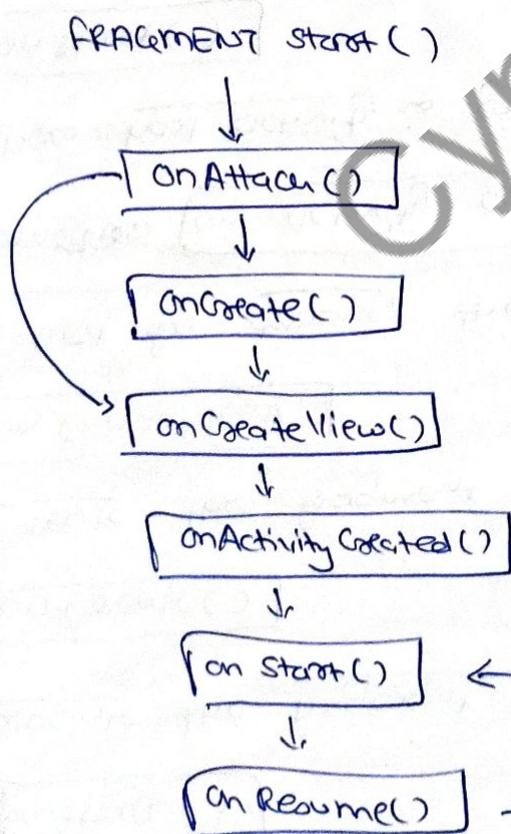
TLDR:

A - sub activity

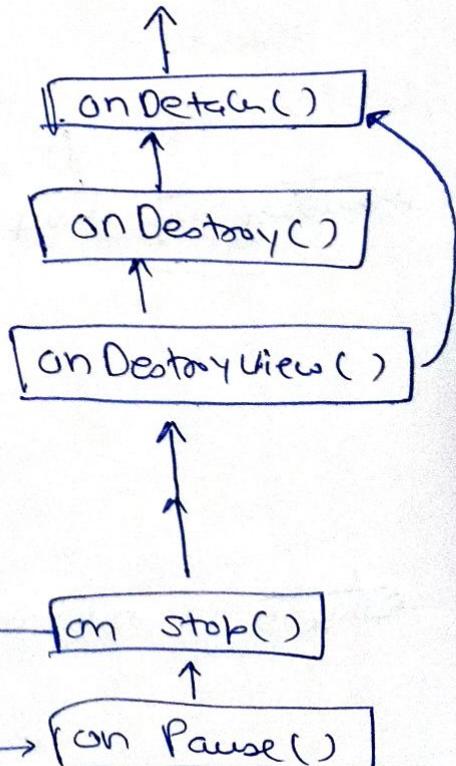
Piece of activity which enables  
more modular activity design

Here we selected a "list 2" from  
activity ① & now we have a  
whole space to write the "list 2"  
info in activity ②

Here "List-2" & "Expanded List-2"  
are FRAGMENTS

Life Cycle:

FRAGMENT ENDS()



## fragment Start

① on Attach()

↳ associate the fragment instance (w) an instance of activity

② on Create()

↳ create the fragment

③ on Create View()

↳ called for drawing the UI for the first time for fragment

④ on Activity created()

↳ new host activity is created

⑤ on Start()

↳ fragment is visible

⑥ on Resume()

↳ fragment is interactable

⑦ on Pause()

↳ fragment no longer interactable.

↳ user is leaving the fragment

⑧ on Stop()

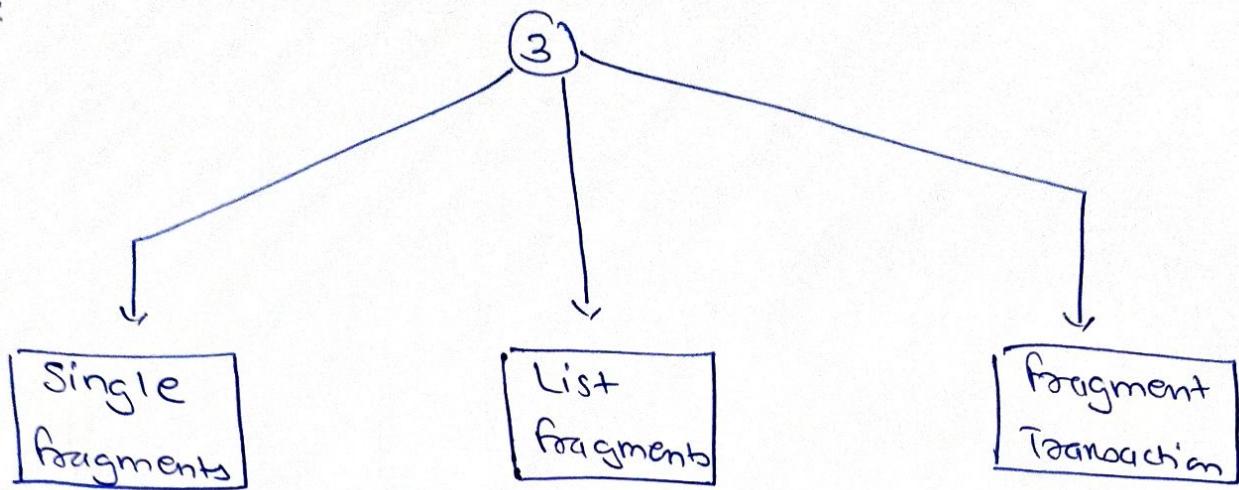
↳ fragment is not visible

⑨ on Destroy View()

: resources in onCreateView() is removed

⑩ on Destroy() : destroy the fragment,

Types :



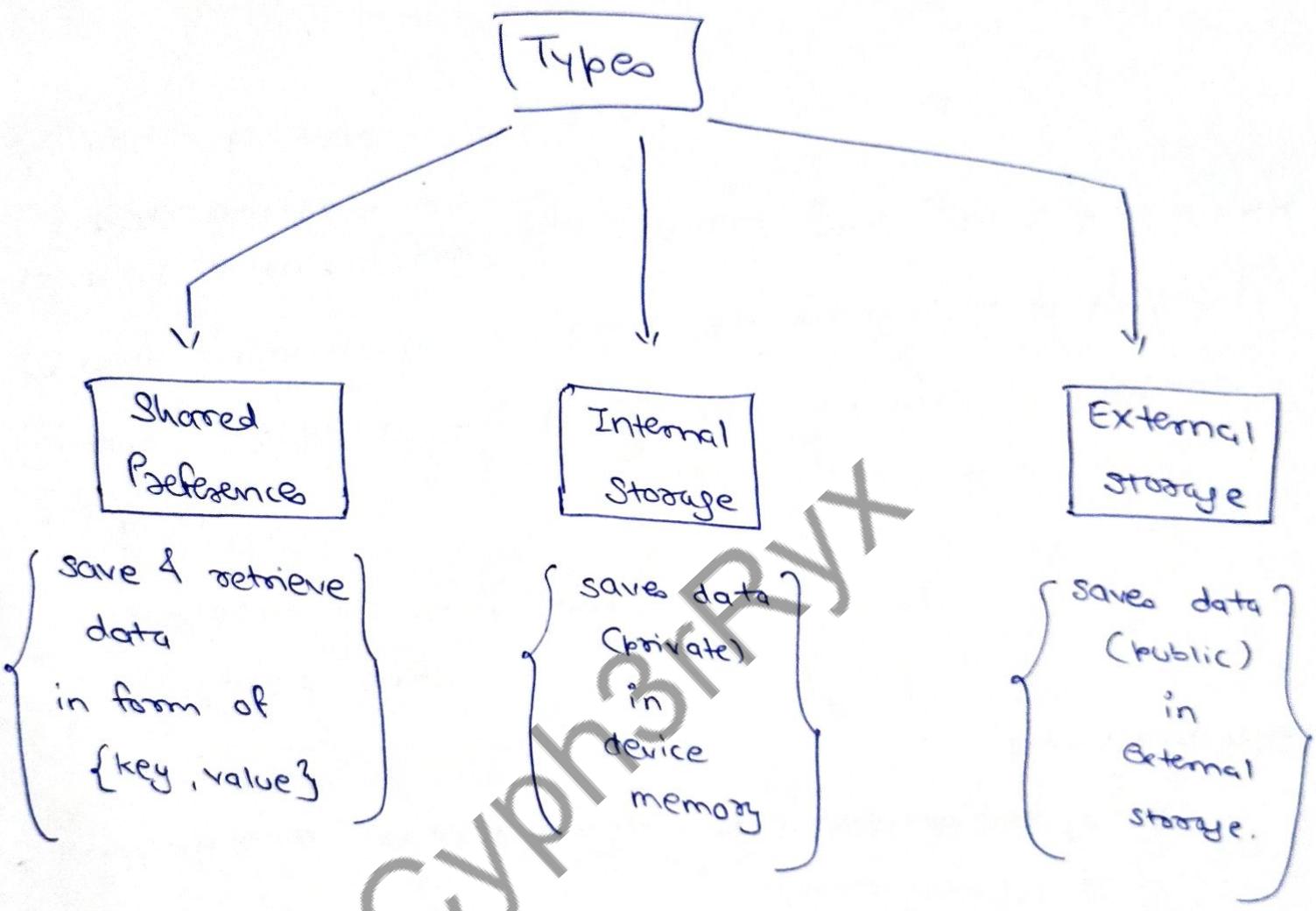
- only single view
- mobile phones.
- they have (special) listview
- there is a "list" & user can choose to see a "sub-activity"
- Transitioning from one fragment to another.
- Support switching b/w 2 fragments

NOTES :

- (i) Fragment is independent
- own layout
  - own behavior
  - own lifecycle.
- TATAKE!!!
- (ii) Add & Remove can be done while activity is running.
- (iii) One fragment → multiple activities
- (iv) multiple fragments → one activity
- (v) Was added in Android 11
- (vi) Fragment can also have no UI (in specific cases)

Unit - 5

## Data Storage Techniques

SHARED PREFERENCE :

{key, value} → e.g. String, int, boolean

in XML file

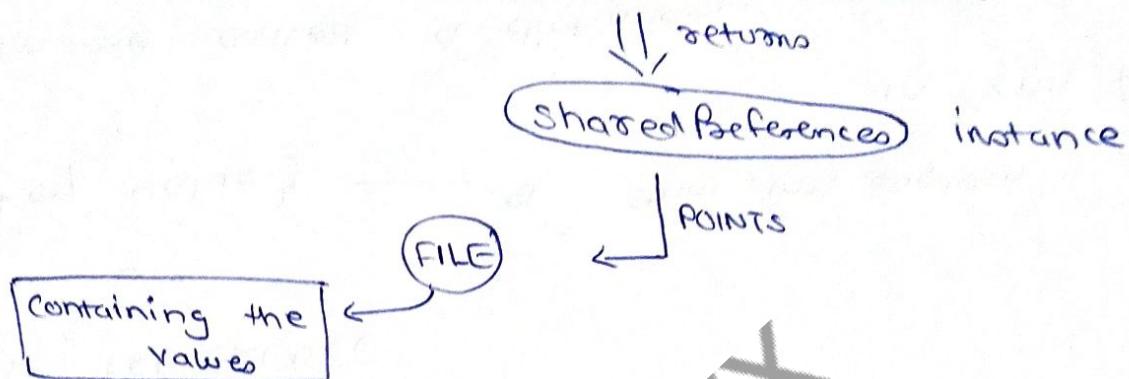
→ Can be called a dictionary.

e.g. You give key = "username" & value = "user's username"  
to validate the login .

## Usage:

You need to call a method

get Shared Preferences()



Eg.

Shared Preferences      `sharedpreferences = getSharedPreferences("MyPref", Context.MODE_PRIVATE)`

here, My Pref. = key

`Context.MODE = Value / MODE`

## Diff. Modes:

- ① MODE\_APPEND → joins new w/ existing preference.
- ② MODE\_MULTI\_PROCESS → checks for modification of preference.
- ③ MODE\_WORLD\_READABLE  
MODE\_WORLD\_WRITEABLE } → Allows other app to read & write
- ④ MODE\_PRIVATE → only accessed using application

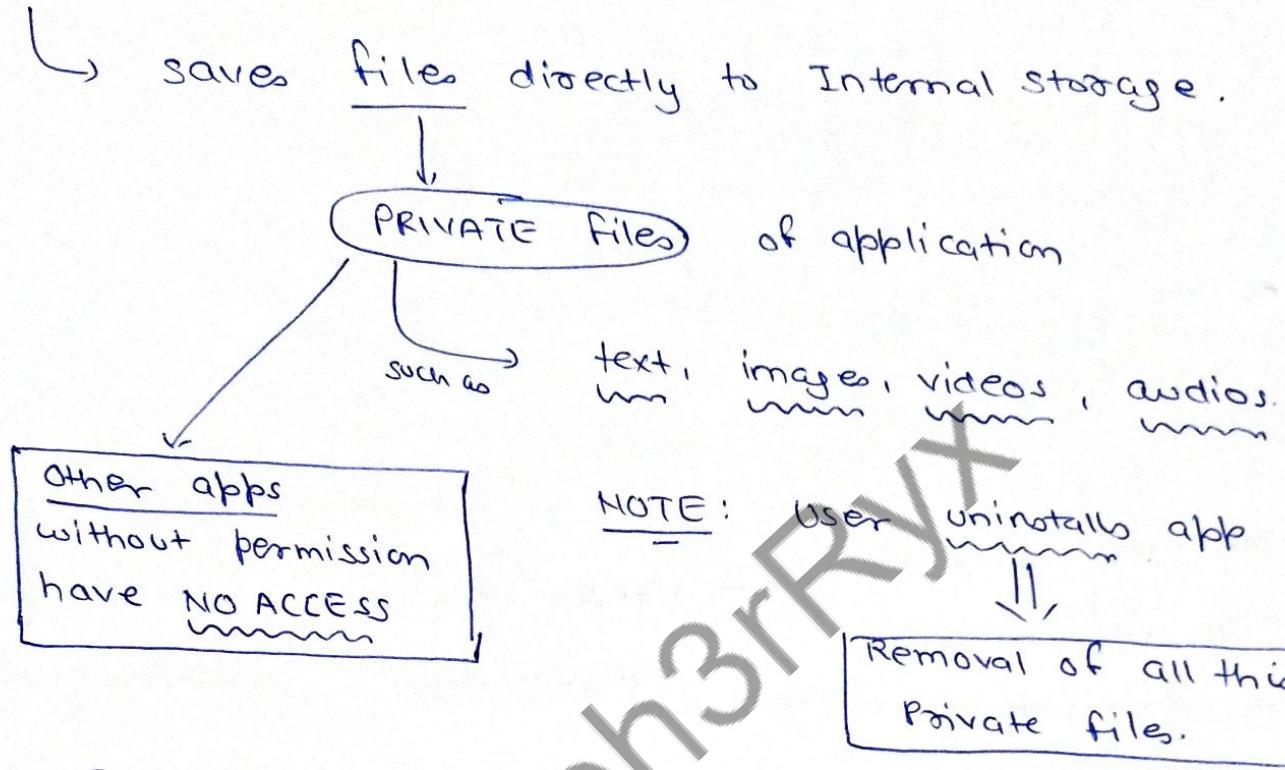
You can also save something in Shared Preference

by .Editor class:

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

- ① apply() → commit changes back from editor.
- ② clear() → remove all values.
- ③ remove(String key) → remove a given key value
- ④ putLong(String key, long value) → save a long value in a preference editor.
- ⑤ " " , int value) → save a int value
- ⑥ " " , float value) → save a float value

## Internal Storage:



## HOW TO USE IT?

To create & write Private file to Internal storage.

**Call** → `openfileOutput(String filename, int mode)`

**mode** → `MODE_PRIVATE` → only your app have access

`MODE_APPEND` → append data to existing content

`write` → `write()`

`read` → `read()`

`close` → `close()`

Ex.

```

String FILENAME = "hello_file";
String string = "hello world!";
fileOutputStream fos = openfileOutput(FILENAME, context.mode -
private);
fos.write(string.getBytes());
fos.close();

```

for reading also same, but use

openfile Input()

Ex.

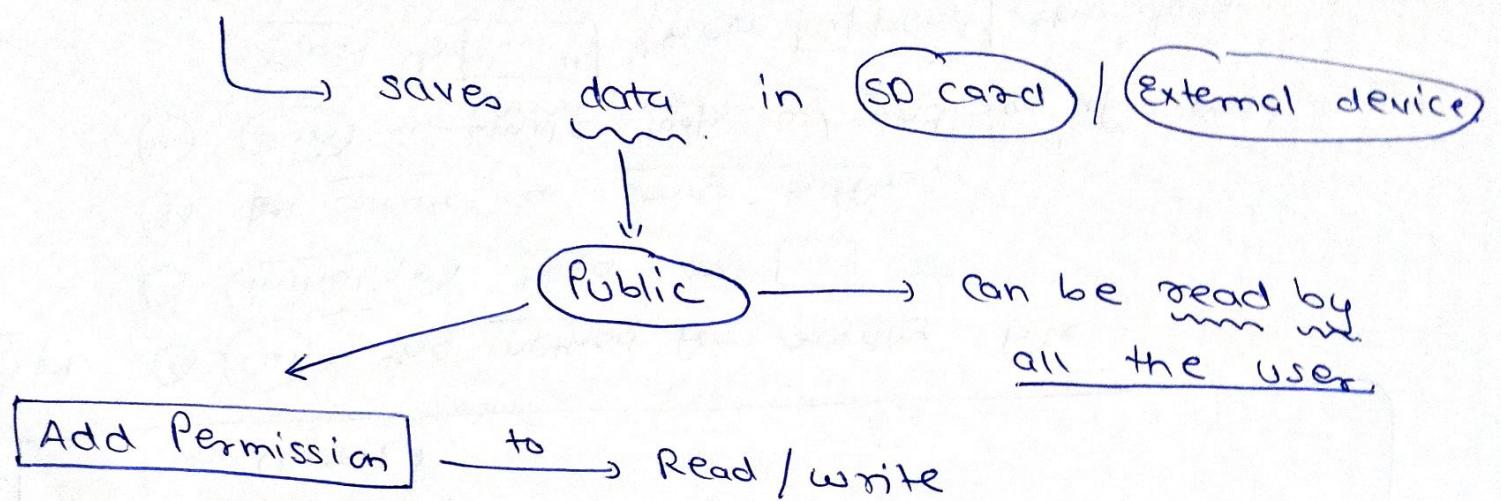
```

FileInputStream fis = openfileInput(FILENAME);
int read = -1;
StringBuffer buffer = new StringBuffer();
while (read = fis.read() != -1) {
    buffer.append((char)read);
}
fis.close();

```

- Expl.
- ① call the method for reading file
  - ② initialize read on -1
  - ③ do create a string buffer
  - ④ run a while loop w/ condition if read is not equal to "-1" then append the data
  - ⑤ fis.close() → closes the input read.

## External Storage:



< uses-permission android:name = "android.permission.READ\_EXTERNAL\_STORAGE" />  
< " " " " . WRITE\_EXTERNAL\_STORAGE" />

But to read / write we atleast need to check if SD card is available or not.

∴ to check its availability we use

getExternalStorageState()

- if it returns true ⇒ Yes its available
- " " " false ⇒ Nope its not!

CODE :

```

/*
Availability Check */

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

```

Exp. ① initialize "public" mode of "boolean" type  
for External Storage

- ② Define "state" w/ getExternalStorageState()
- ③ Run "if" loop
- ④ Check result.

for checking if External Storage is Readable or not:

Just change boolean data to isExternalStorageReadable()

& Environment.MEDIA\_MOUNTED\_READ\_ONLY

To write the DATA:

- ① Create a file specific to APP, [removed when uninstalled]

```
file folder = getExternalStorageDir("MyFolder");
file myfile = new File(folder, "mydata.txt");
```

Here

getExternalStorageDir (String type)

→ used to create temporary files just for app

but

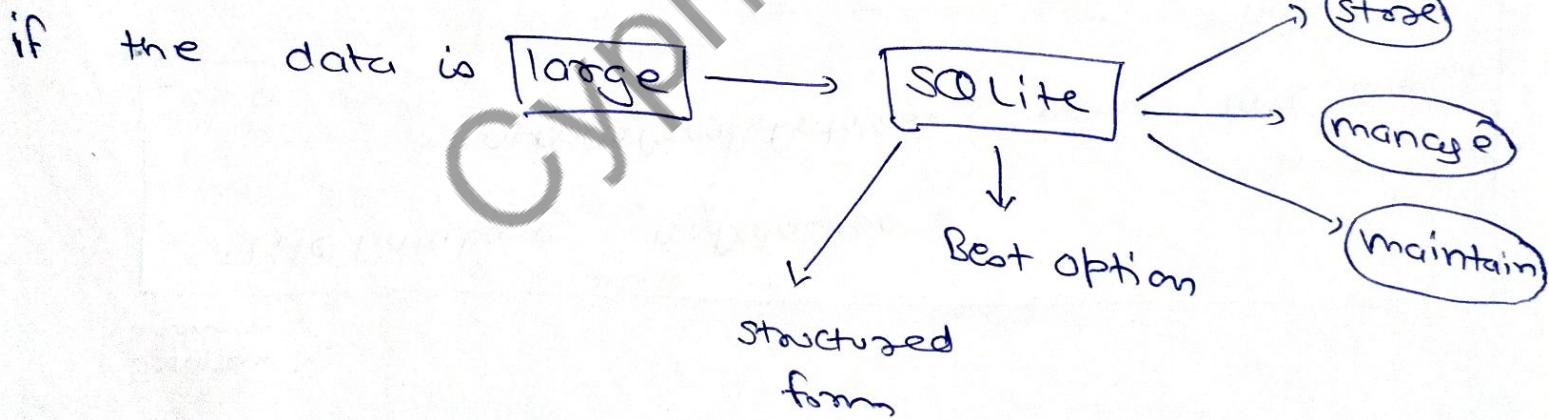
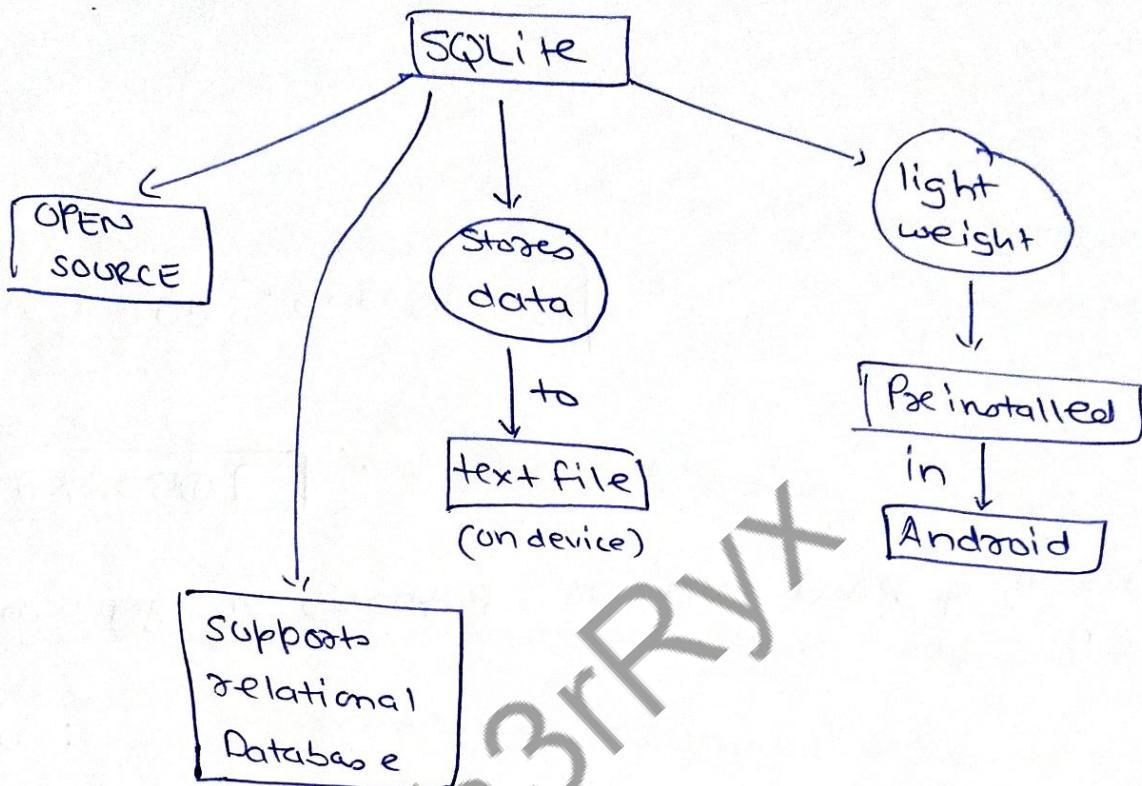
getExternalStoragePublicDirectory (String type)

→ creates a permanent file in External storage.

Now To read the DATA:

Same code as the internal Storage

## SQLite

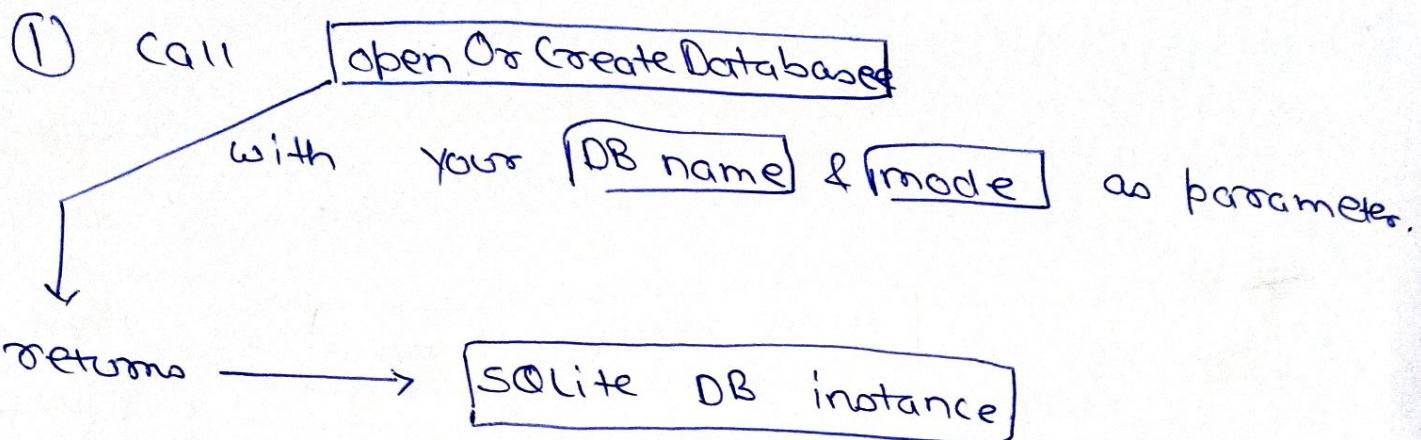


Android  
Package

=

`android.database.sqlite`

## Create Database:



Syntax:  
~~~~~

```
SQLiteDatabase myDatabase =  
    openOrCreateDatabase("name", MODE_PRIVATE,  
    null);
```

## Insertion:

Now once DB is created, so we need to insert data.

Using `execSQL`

→ `"name of DB", execSQL`

Syntax:

myDatabase.execSQL

( "CREATE TABLE IF NOT EXISTS Ryx ( Username VARCHAR,  
Password VARCHAR);");

myDatabase.execSQL

( "INSERT INTO Ryx VALUES ('admin','Imaded');");

Exp. • first syntax will create a table named "Ryx"  
in which username is assigned Varchar datatype  
and password is assigned Varchar datatype.

• second syntax will insert the values in table "Ryx"  
which are  
username = admin  
password = Imaded

NOTE : execSQL is also used to { update  
modify  
insert the data }

← How { to an already  
existing data  
in DB }

Using  
bind  
argument

Enrollment No :

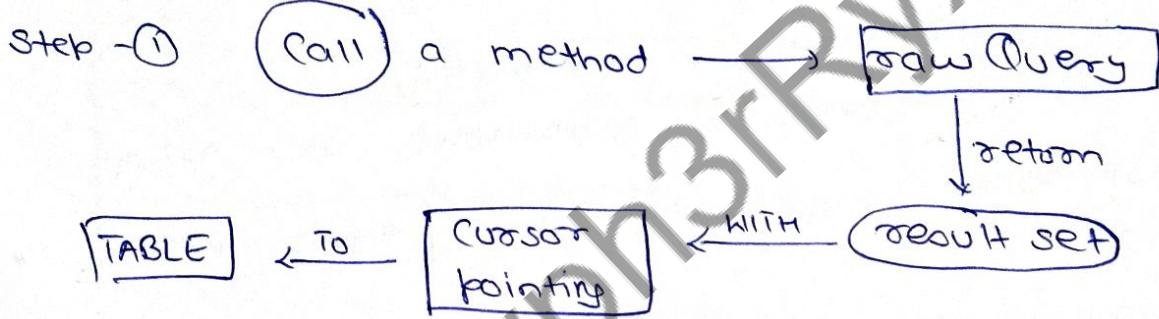
Page No :

## Fetching Database:

Once Creation is checked ✓  
Insertion

we move to "fetching" phase . ,

Using → Cursor class



S-② we can move cursor → forward  
and retrieve data.

## SYNTAX:

```
Cursor resultSet = myDatabase.executeQuery("SELECT * from Ryx ", null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

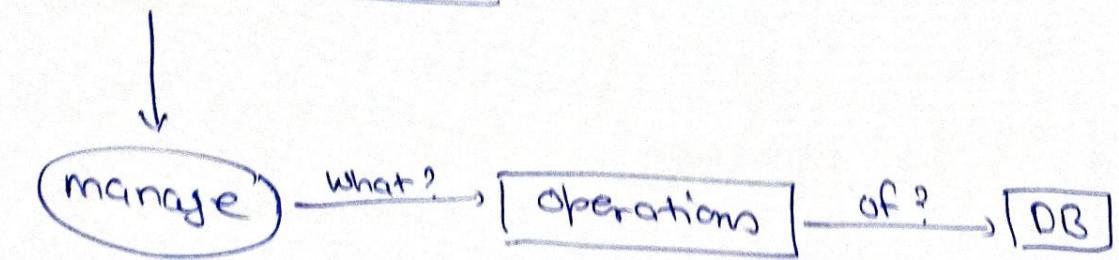
Exp.

- First: We define cursor for result set  
we define (call) rawQuery argument as null for Table "Ryx" f
- Second: we move the cursor forward for (retrieval) of Data
- Third: we get "username" as String type.
- Fourth: we get "password" as String type.

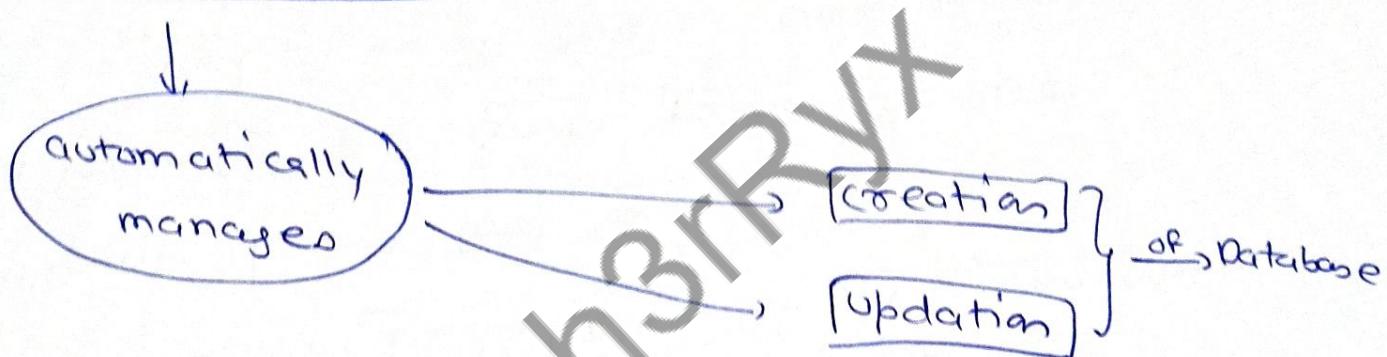
### Cursor class methods:

- ① [get Column Count()] → Total columns
- ② [getColumnName()] → Array of all column names
- ③ [getCount()] → Total rows
- ④ [get Position()] → Current position of cursor.
- ⑤ [isClosed()] → if true ⇒ cursor is closed  
if false ⇒ " " not closed.

## Database Helper Class :



`SQLiteOpenHelper` → Helper Class



### SYNTAX :

```
public class DBHelper extends SQLiteOpenHelper {  
    public DBHelper() {  
        Context context;  
        super(context, DATABASE_NAME, null, 1);  
    }
```

Ex: We define a public class DB helper which uses SQLiteOpenHelper

- We give then commands for DBhelper()
  - Via a "Constructor"
- The Constructor takes Context as a parameter
  - (to) initialize SuperClass (`SQLiteOpenHelper`)
- "Super" calls constructor & passes

|                            |                  |
|----------------------------|------------------|
| Context                    | → object         |
| <code>DATABASE_NAME</code> | → DB name        |
| null                       | → cursor factory |
| 1                          | → version 1      |

## CONTENT PROVIDER :

- Manages → access to a structured set of data.
- Allows → other apps to access & modify
  - ↓
  - (DATA) in a secure manner
- Used → To retrieve data from → Central repository
  - e.g. SQLite

## Content Resolver:

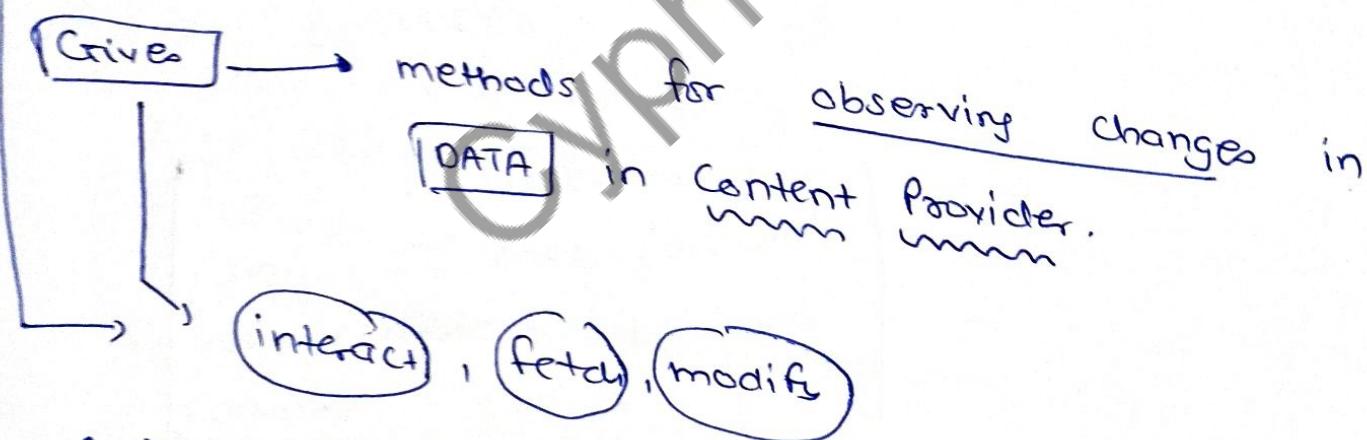
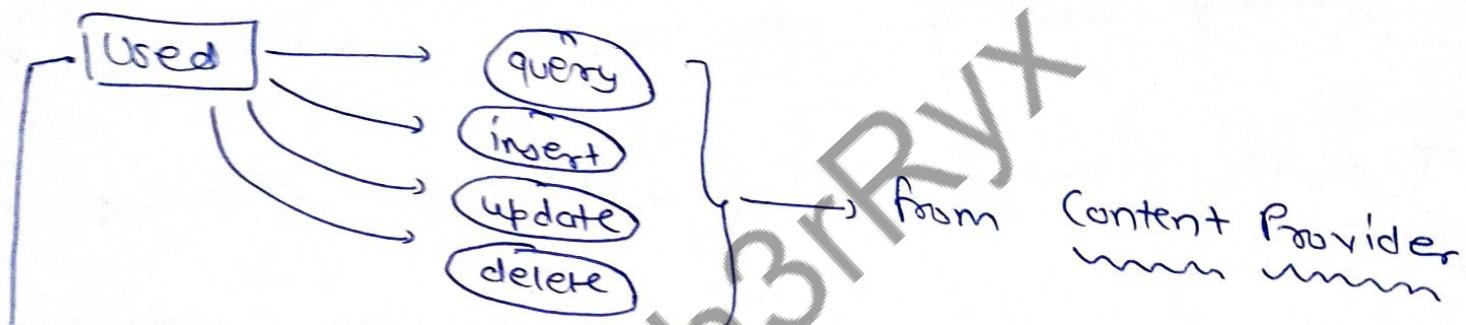
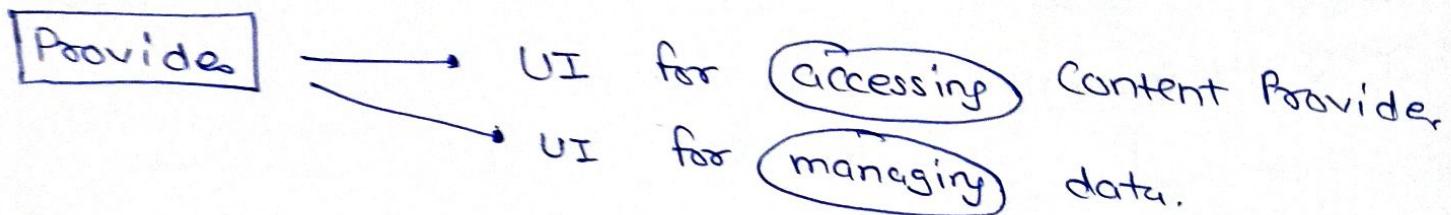
v

A mediator  
~~~~~

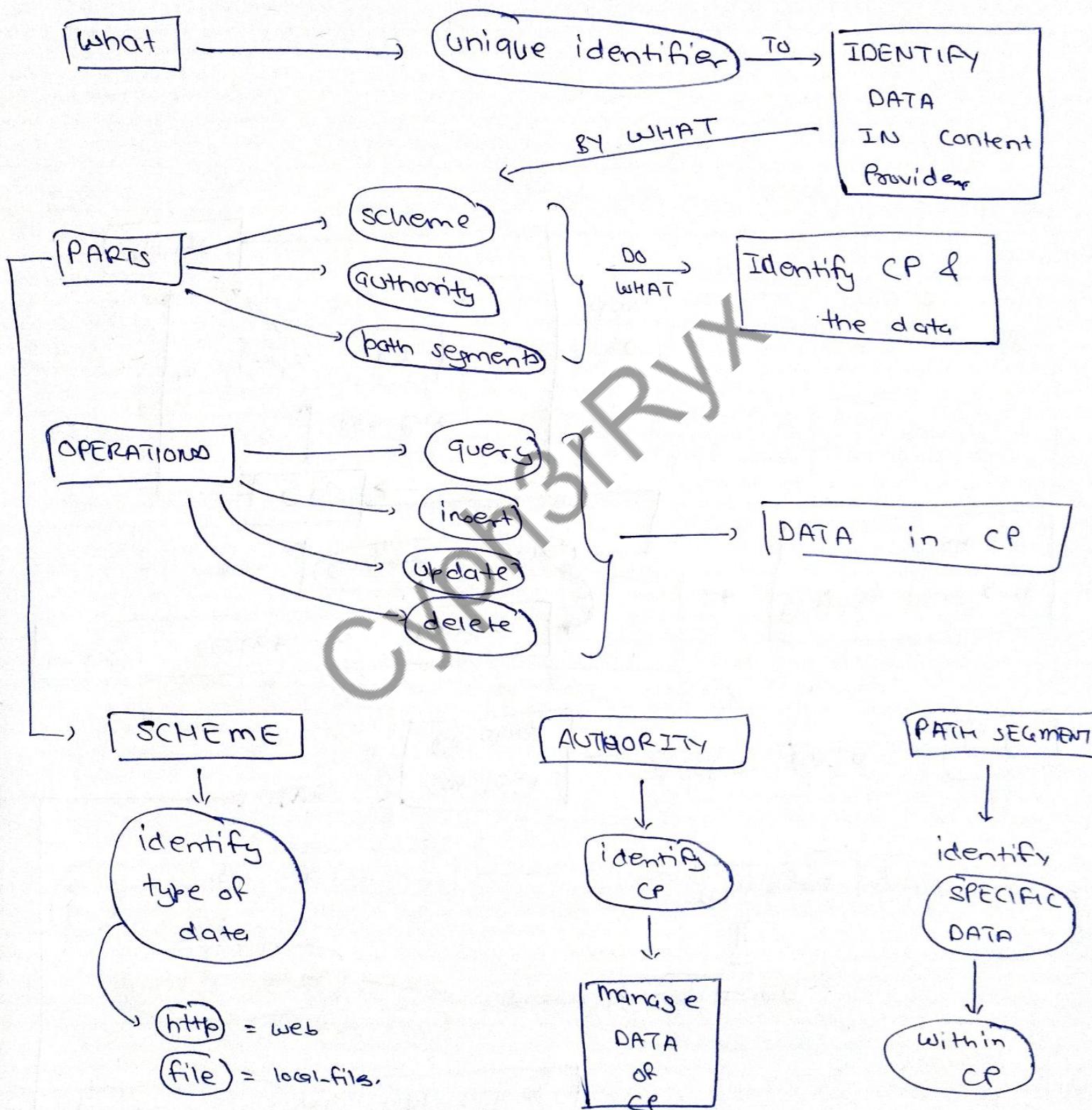
between

Content  
Provider

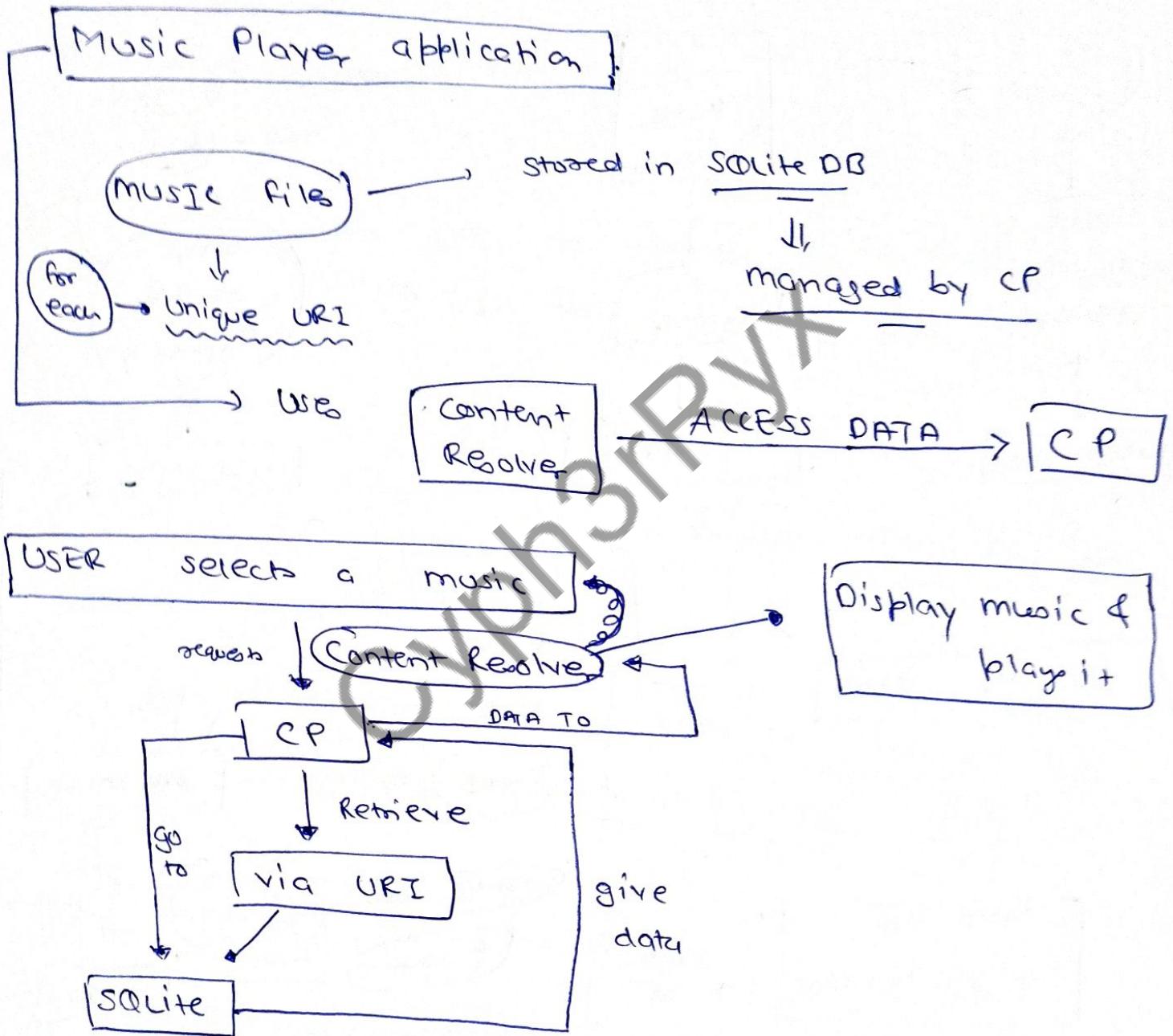
& Application



- Installed APPs → Content:// URI scheme
- File Systems → file:// URI scheme

Content URI's :

Example:

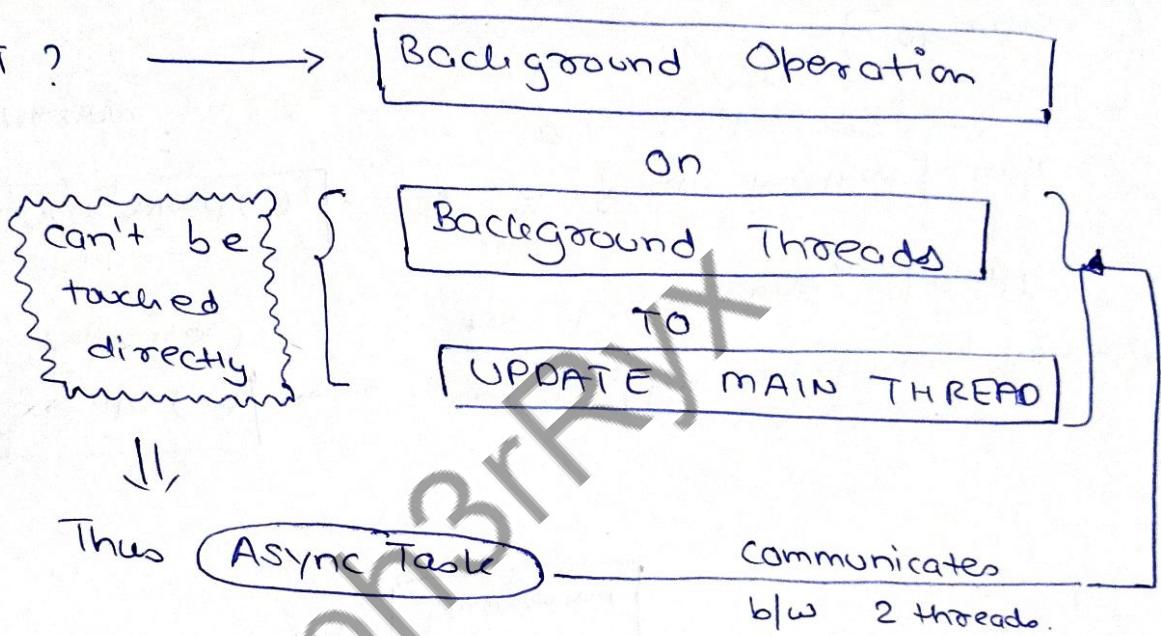


UNIT - 6

## Web App Integration,

Android Async Task

- DO WHAT ? → **Background Operation**



- USED for → **short duration Task**

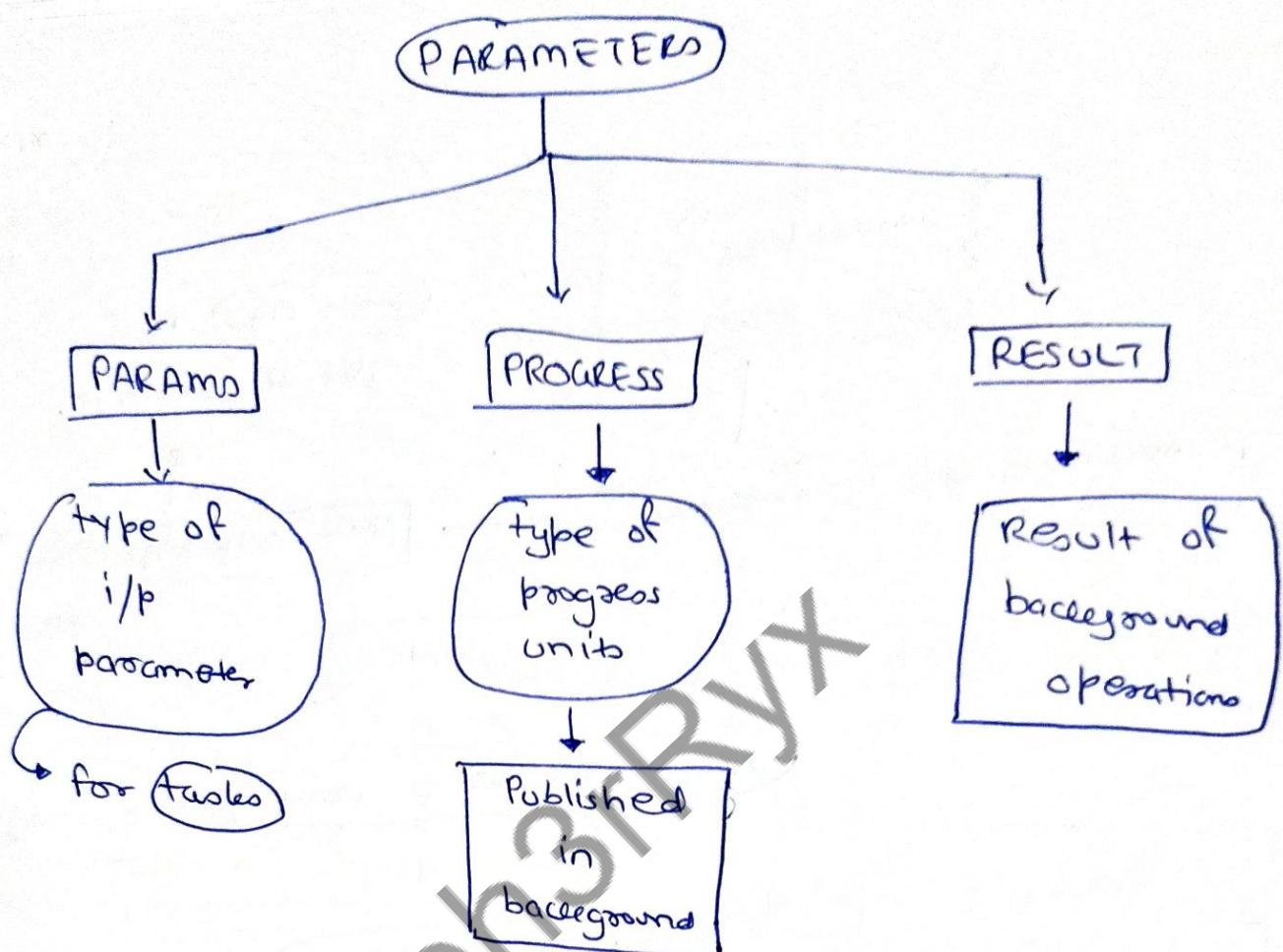
↓  
also can

do **long duration task**

requires a lot  
of **CPU time**

→ **asynchronously** in background

keeps UI  
responsive



### Methods :

① `onPreExecute()`  
(e.g., shows Progressbar)

Setup pre-conditions required  
for background operation.

② `doInBackground()`  
(PARAMS)

A main method where you  
can perform { background  
operations. }

- ③ `onProgressUpdate()` → Publish progress update in UI thread about Background Operations  
(Progress)
- ④ `onPostUpdate()` → called when Background operations are completed.  
(Result)

Start Sync Task

via `@ execute()`

Cancel Sync Task

via `onCancelled()`

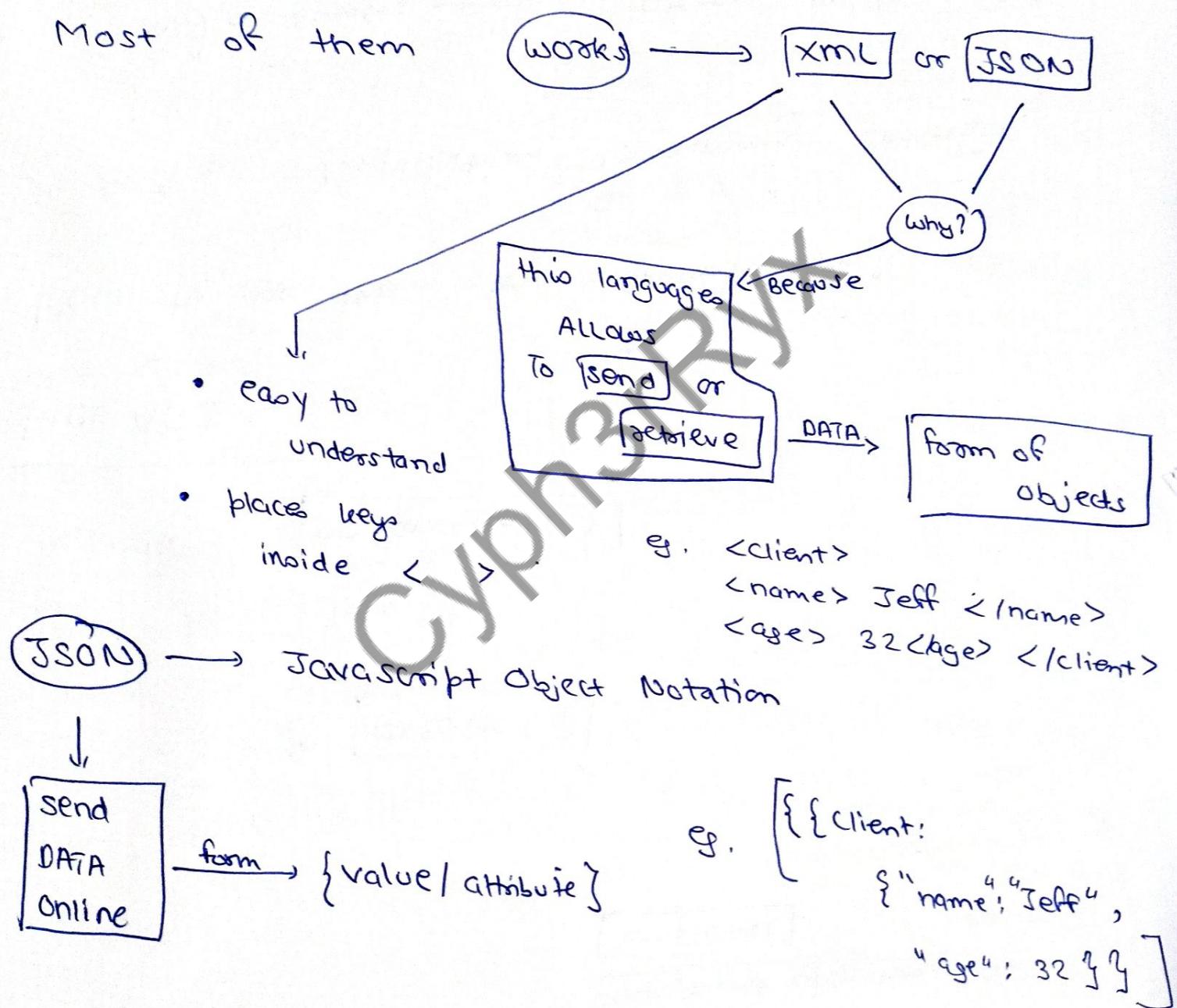
Web API

How to use them from Android App?

API = Application Programming Interface.

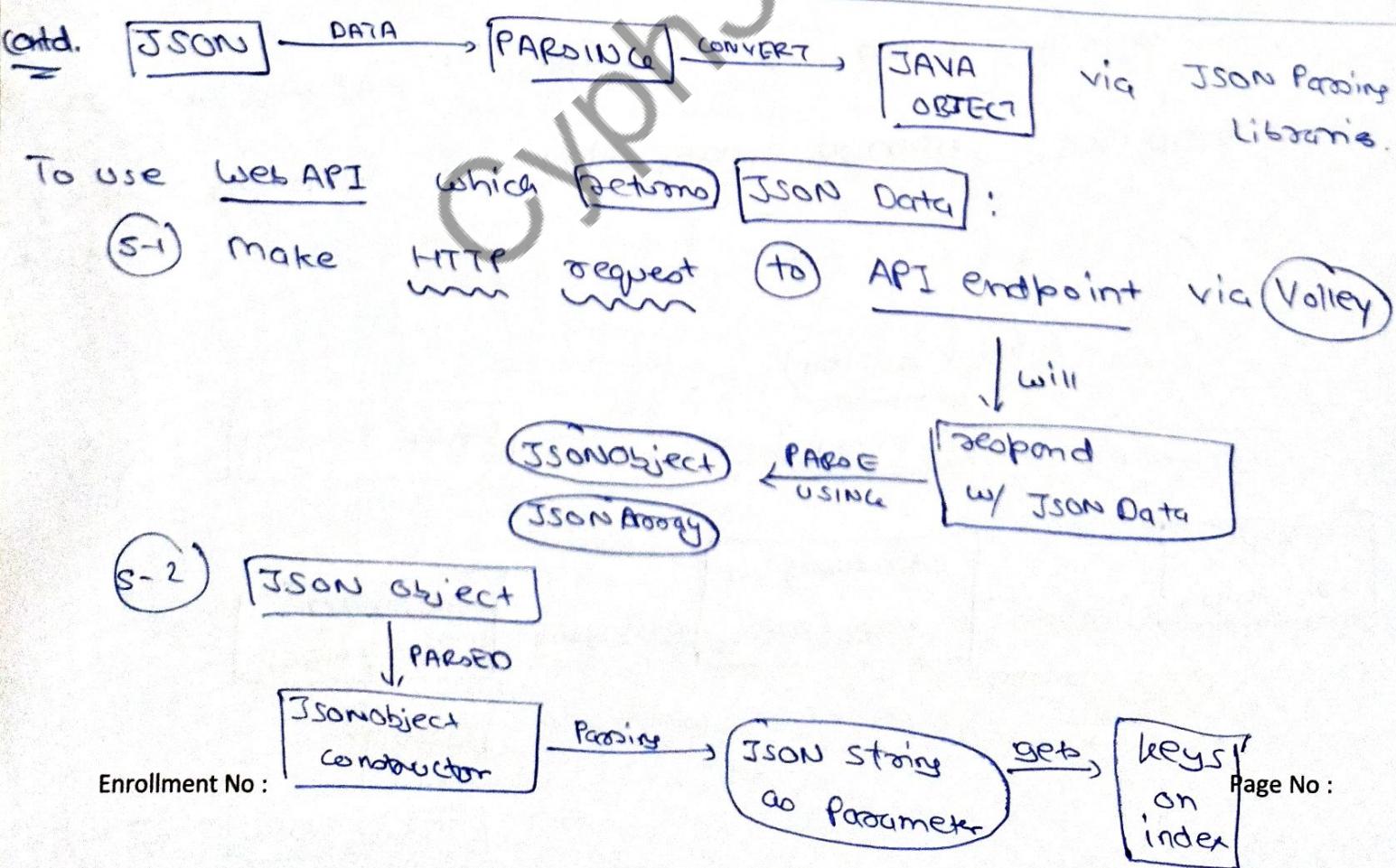
↓  
allows developers → interact w/ `external services`.

- API are commands set by developer to use some of features of their program.
- Most of them



Q) But how to use them?

- S-1 **Determine** API endpoint → provided by API provider.
- S-2 **Select** Network library → volley API
- S-3 **Create** Network request → GET, POST, OPTIONS
- S-4 **Send** the request → send request w/ headers
- S-5 **Parse** the response → format of response  
JSON / XML
- S-6 **Use** the data in your App → Hurray!!! You did it.



S-3

JSON Array

Passed

JSON Array  
constructor

Passing

String (JSON)  
as Parameter

retrieve

values on  
index

S-4

Passed Data

USED  
FOR

Update UI

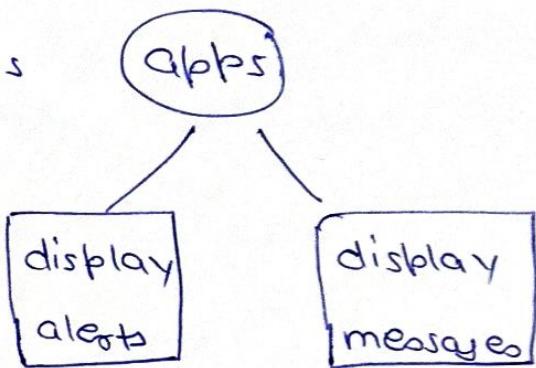
Populate views

NOTE:

Error Handling is important because JSON data can be malformed, triggering an error situation.

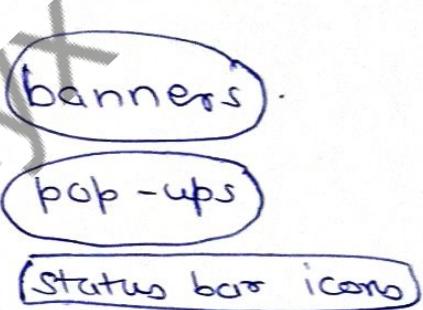
## NOTIFICATION :

It allows



from the App's main UI

→ Can be displayed via



## HOW TO USE THEM?

S-1 Create a Notification Object by **NotificationCompat.Builder** class

S-2 It can be customized with **setSmallIcon()**, **setContentTitle()**, **setContentText()**

S-3 Display notification object via **NotificationManager** class

For displaying :

Obtain an instance via `getSystemService()`  
passing in `NOTIFICATION_SERVICE` constant

- ③ Set up an `intent` to specify action  $\xrightarrow{\text{when}}$  user clicks it

- ④ Show notification

via calling `NotificationManagerCompat.notify()`

## Telephony API:



Set of fx<sup>n</sup> & protocols

to interact with a telephone system / device

→ can be used while creating a system

that can make call, receive call,

send & receive sms,

access phone book

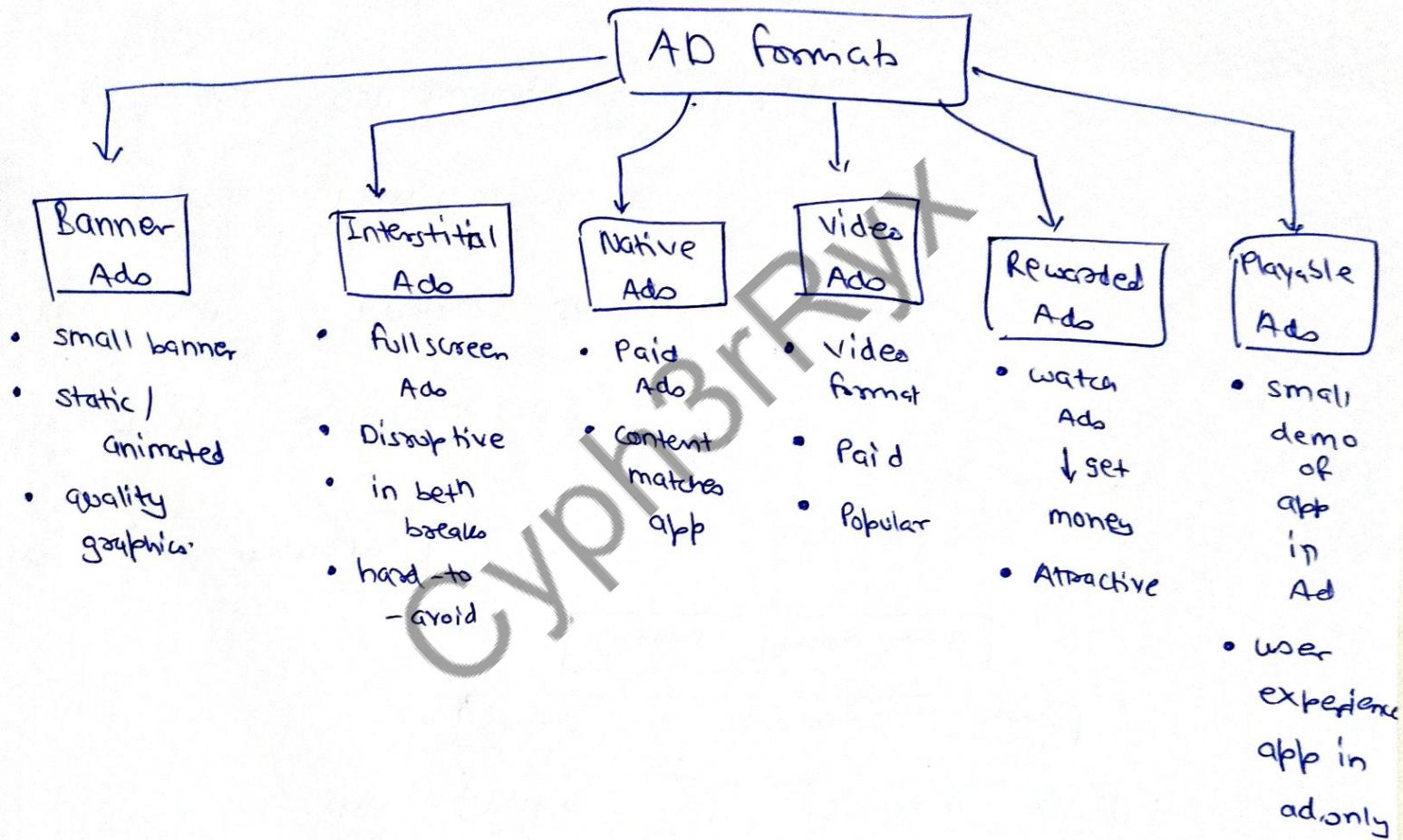
eg. call manager

- ① Make & Receive calls
- ② send & receive sms
- ③ Access Phone book
- ④ Access call history & call log.

## Chapter - 7

### PUBLISH APP

#### Monetize



## Steps to publish an App:

- ① Create a Google Play Console Account.
- ② Prepare your APP for release (testing & optimizing)
- ③ Create a signed APK or APP BUNDLE.
- ④ Create Title, description, screenshots.
- ⑤ Set pricing
- ⑥ Submit your APP for review
- ⑦ Approval acceptance → PUBLISH APP!

Hurray !!!

Syllabus completed

