Unit - 1

OS → Operating System

↓

initially loaded via (Boot program)

interaction via (Command line) or (GUI)

API → Application Program Interface

↳ used for making requests for services by Appl. Program.

Types :

Android	Bada	Symbian
Apple	Blackberry	Web

Android

↳ (mobile OS)

↳ developed via (Google)

↳ based on (Linux) kernel

Founder : Andy Rubin & Palo Alto

Date : Oct, 2003

& (17th Aug. 2005)

↑

Google acquired android

Ryx

Versions of Android:

1.0 → Alpha

2.0 → Beta

3.0 → Cupcake → 1.5

4.0 → Donut → 1.6

5.0 → Eclair → 2.0

6.0 → Froyo → 2.2

7.0 → Gingerbread → 2.3

Honeycomb → 3.0

Icecream Sandwich → 4.0

Jelly bean → 4.1

kitkat → 4.4

Lolipop → 5.0

Marshmallow → 6.0

Nought → 7.0

Oreo → 8.0

Pie → 9.0

Q → 10.0

R → 11.0

SAOWcom → 12.0

Tiramishu → 13.0

Features:

- memory management
- Process management
- Device management
- file management
- Security

Annexure No :

OHA : Open Handset Alliance

→ organization of (84) companies

→ led by Google.

→ develop open Standard for mobile

For promoting Android.,

members of OHA

↓ Prohibited

producing devices that are → incompatible forks of Android

Android architecture :

Applications : HOME, CONTACTS, PHONE, BROWSER

App. frame work : Activity manager, Window manager, Content Provider, Resource manager, Notification manager

Libraries : SGL, SSL, SQLite, OpenGL, libc

Android runtime : DVM, Core libraries

Linux kernel : USB Driver, Camera Driver, Bluetooth driver, Audio driver, Power management

Android components:

① Activities

- always used in App
- multiple activity in app
- each activity has various UI component
- Activity class — all work is done there.

② Services

- background processes — not displayed
- can be started & managed from activities
- higher priority
- eg. playing music, change it automatically, etc.

③ Intent

- provides a developer with the option to move from one screen to other
- consists of the operation to be performed & the activity class name has to be declared.

(4) Intent Reciever:

↳ when a fxn has to be executed in response to an event.

↳ alert via notification.

(5) Content Provider:

↳ Sharing data within the app.

↳ Apps can store data in shared preferences.

eg. SQLite, files, etc

(6) Widgets & notification:

→ Widgets for information : only info is displayed | Dynamic | Home screen manipulation.

→ widgets for collection : Collection of info. of same kind | Email App

→ Control panel widget : display most used features | Pause, Play

Hybrid widget → Combines above all 3's

Broadcast reciever:

↳ responds to Broadcast Messages from other applications.

↳ eg. App. can initiate brodcast saying / stating the info. about some data getting downloaded.

Broadcast Reciever will intercept this communication & will initiate appropriate action

↳ implemented as subclass of Broadcast Reciever class

ex. public class MyReciever extends BroadcastReciever {
public void onRecieve (Context, intent) { }
}

Drawable Resources:

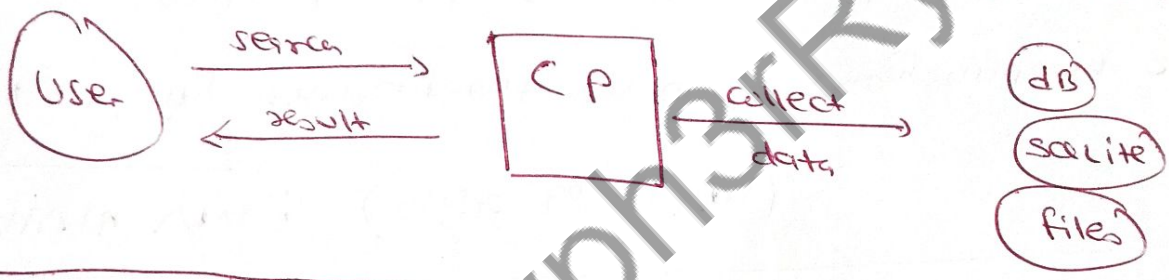
↳ graphic

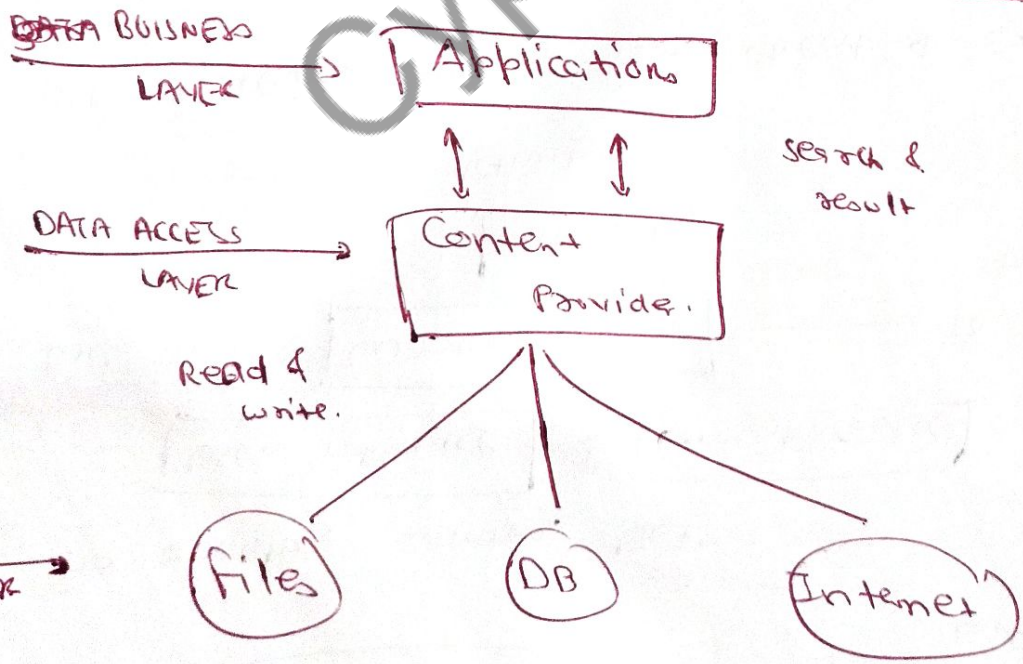
↳ Bitmap file represented via a BitmapDrawable class.

↳ Every drawable is stored as individual files in one of the res/drawable folder.

Content Provider

- share data bet^n Apps
- supply data on basis of request.
  - handled by "ContentResolverClass"
- implemented as → subclass of ContentProvider } must implement a standard set of API's that enable other apps to perform transactions.





Access data with Content Provider:

① Use (Content Resolver) → to communicate with CP for DATA ACCESS

② for enabling comm. beth [User interface] & [Content Resolver]

we use [Cursor Loader] → to run query asynchronously.

↓ called by Activity

③ CP recieves query from client & execute result.

DALVIK VM : (Register based VM)

↳ providing environment in which every android app. runs.

↳ each android app runs its own process.

↳ in own instance of

[multiple VMs] ← can run ← (Dalvik VM)

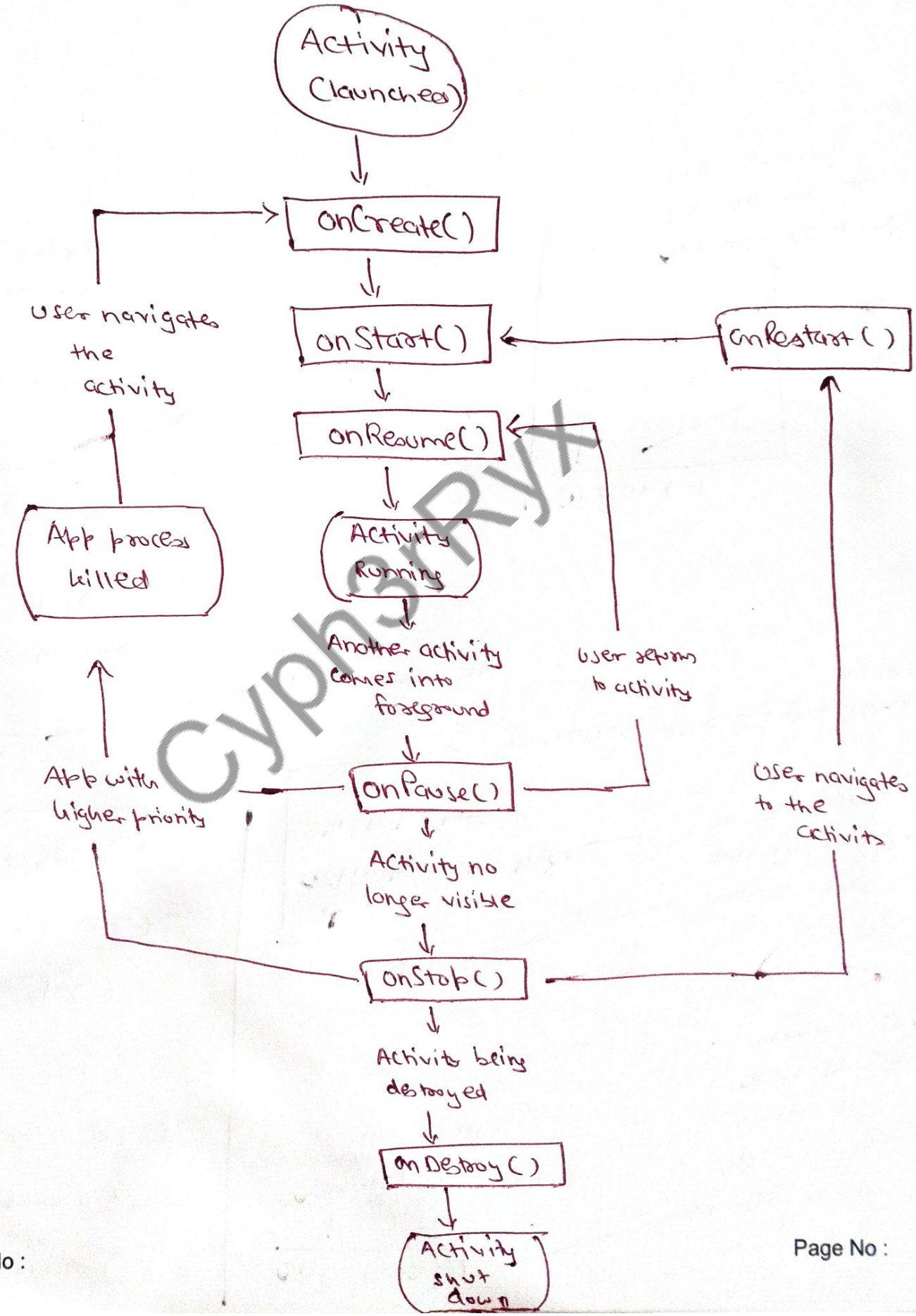
[.dex] format

↳ optimized for minimal memory footprint

Compilation:

[.java] → [.class] → [.dex]

Activity Life cycle :



Ryx

1. onCreate()
2. onStart()
3. onResume()
4. onPause()
5. onStop()
6. onRestart()
7. onDestroy()

onCreate()

↓

activity is created in the state.

→ OS calls this method when an activity gain memory.

onStart()

↓

activity enters the started state.

→ makes the activity visible to the user

→ initializes the UI maintaining code.

onResume()

↓

activity enters the foreground state

→ app communicates with user

onPause()

↓

user leaving your activity (no longer in foreground)

→ still be visible in multi window mode.

onStop()

↓

Any activity is on halt

→ hidden from user

→ App will cease to provide the user with any useful info.

onRestart()

- ↓ After stopped state, activity is called in this stage
- → User returns to the screen or resumes the activity that was stopped.
- → Once activity is stopped then & only then it will be restart.

onDestroy()

- ↓ App not in background then it reaches this stage.
- → User clicks back button & the activity will end after it completes the pause & stop lifecycle.

Intent:

- message passed b/w components
- used with startActivity() to involve activity, broadcast reciever..

→ used for

1. start service
2. launch an activity
3. Display a webpage
4. Dial on phone call
5. Broadcast a message
6. Display contacts.

(I) Implicit Intent: (doesn't specify the component)

↳ intent provides info of available components provided by the system to be involved.

eg. call, mail, phone, see any website.

```
Intent i = new Intent();
i.setAction(Intent.ACTION_VIEW);
i.setData(uri.parse("www.pu.ac.in"));
startActivity(i);
```

Ryx

Explicit Intent:

- specifies the component
- provides the external class to be invoked.

```
Intent i = new Intent (getApplicationContext(), Activity2.class);
startActivity(i);
```

Manifest Attributes of Android Activity:

- To implement activities in our App.
- we need to register them in the manifest file.
- for activities to work properly, we must manage their lifecycle properly.
  - Declare Activities
  - Declare Intent filters
  - Declare Permission

Annexure No :

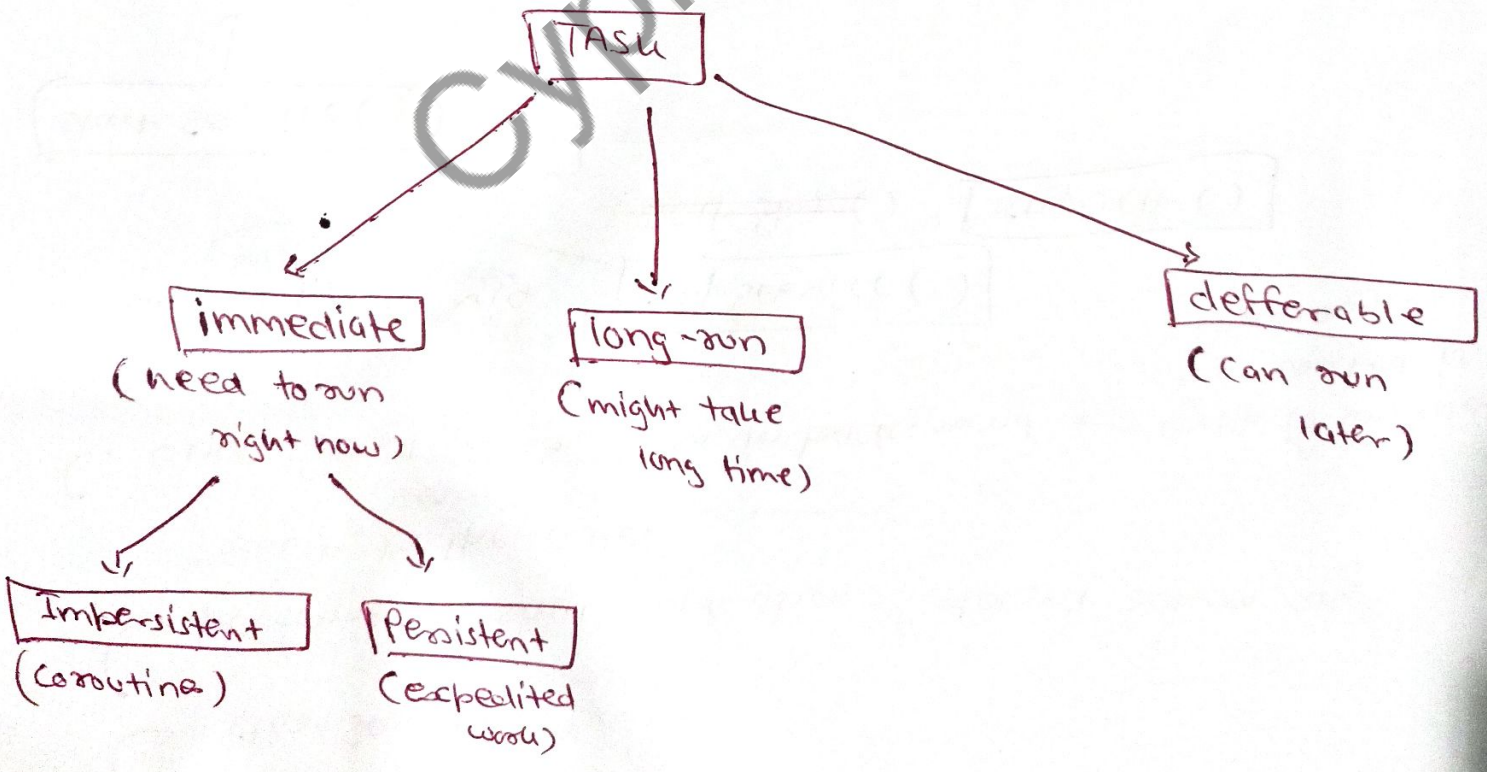
Types of services :

① Foreground services :

- service that notify user about its ongoing operations.
- visible service to users.
- eg. Music Player & Downloads.

② Background service :

- can't see them on screen.
- don't need the user to know them.
- eg. Syncing data & storing data



Lifecycle of Android Services:

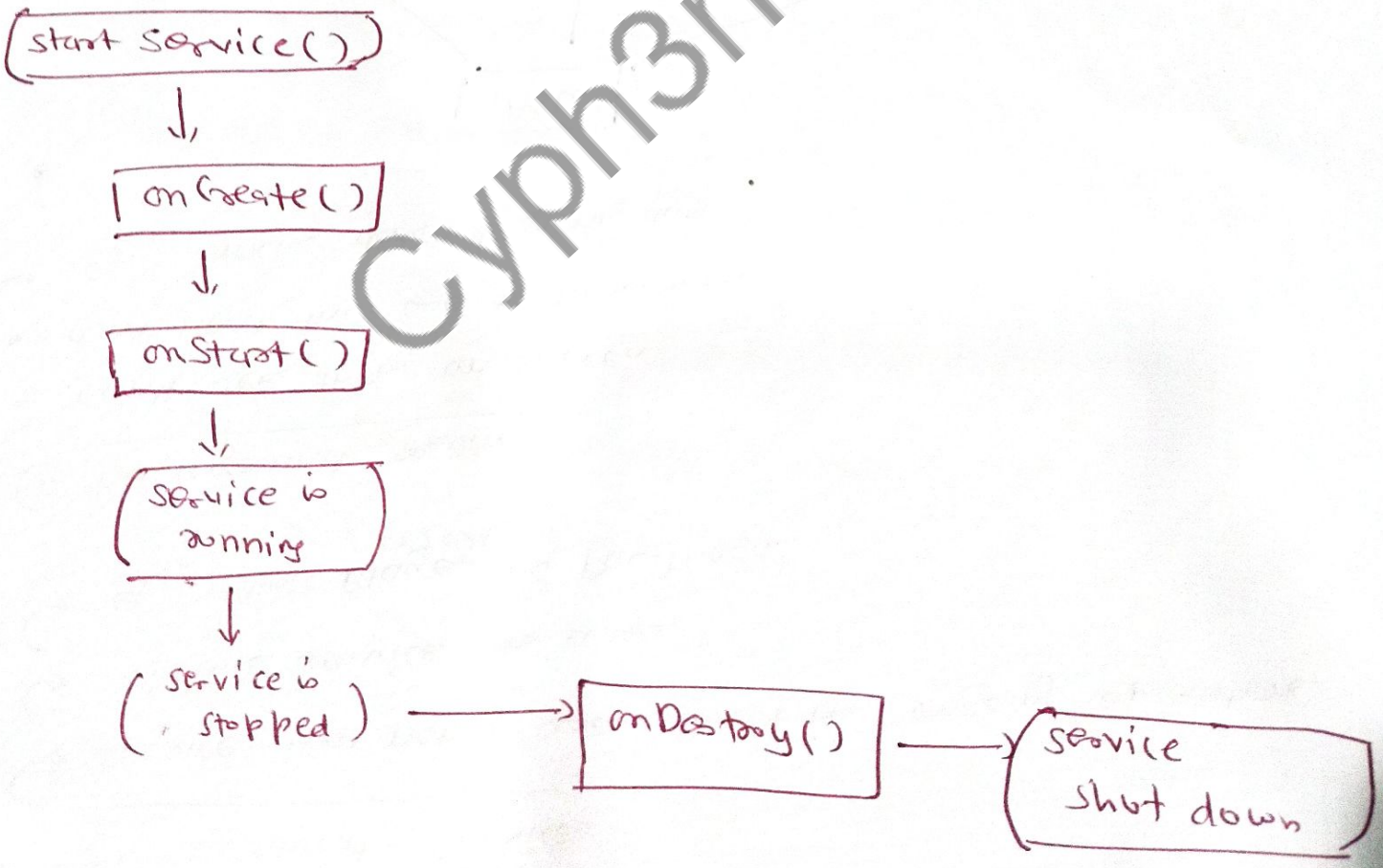
① Started service:

↳ startService()

↳ performs a single operation & doesn't return any result to the caller.

↳ once start → runs in background → even if component that created is destroyed.

can be stopped via stopService() / ~~self stop()~~ stopSelf()

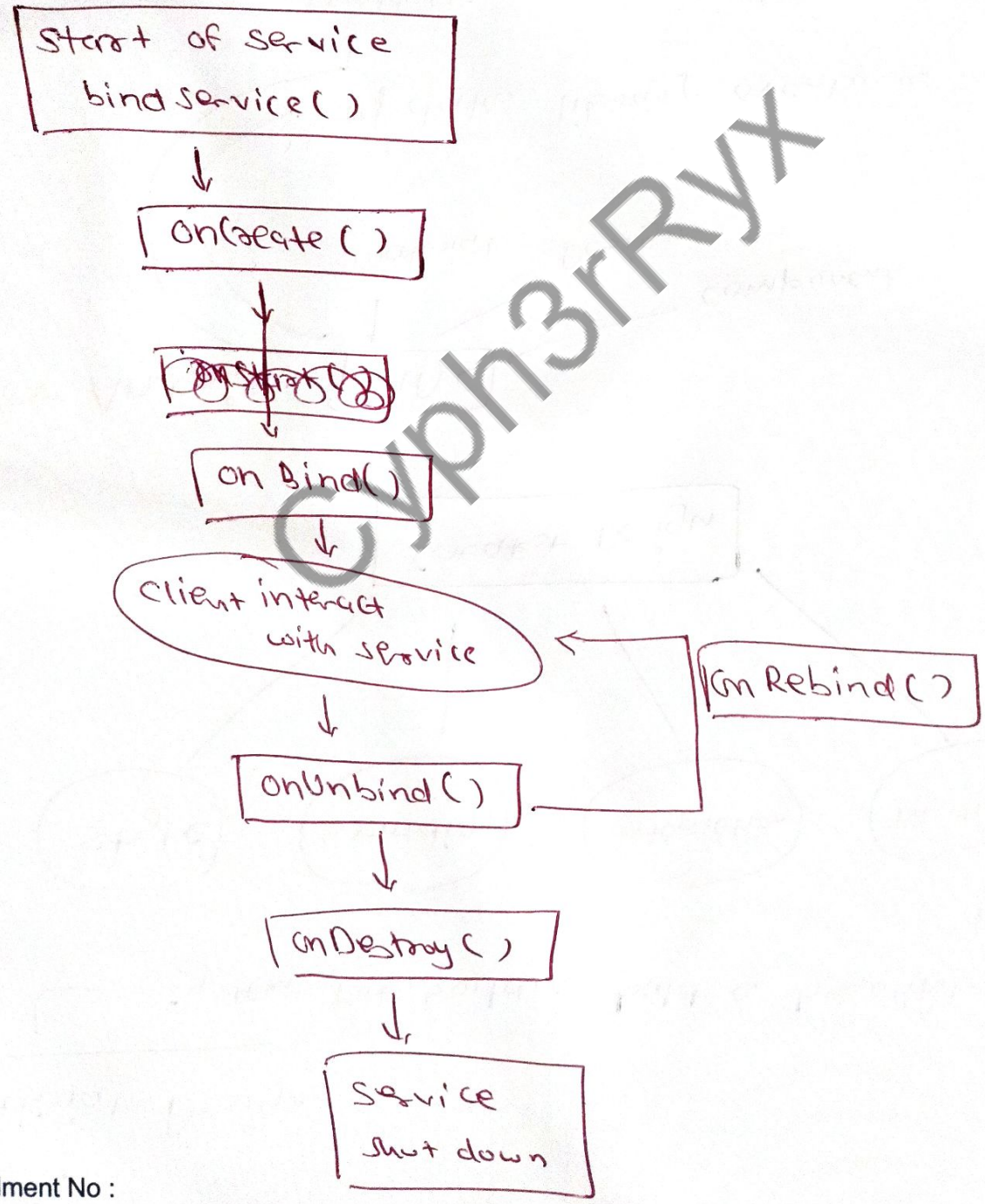


② Bound Service :

↳ service runs in background

bindService ( ) ⟶ to bound the application component

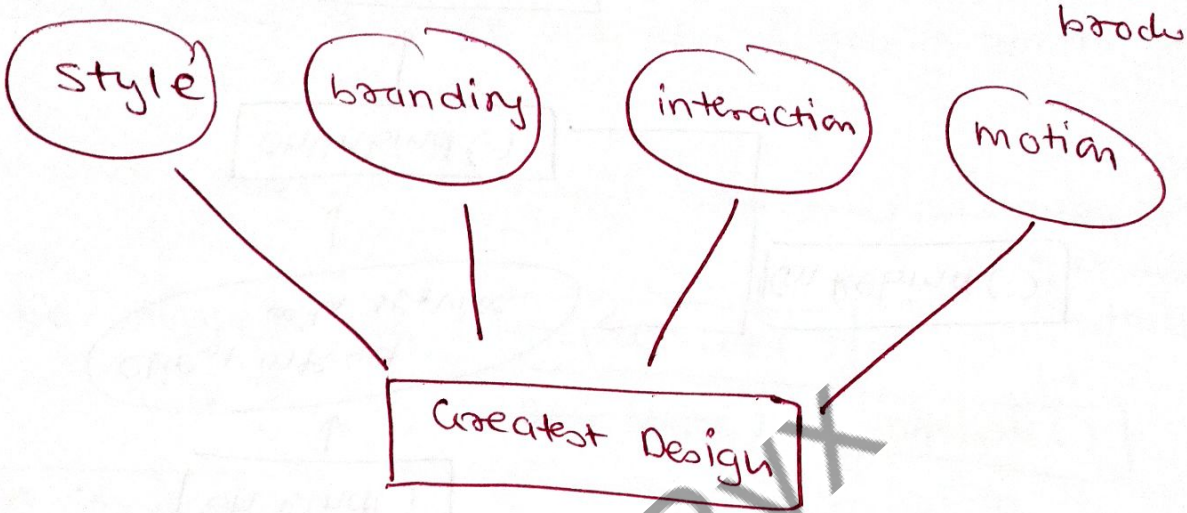
unbind service ( ) ⟶ to unbind the



Unit - 3

Material Design

↳ system for building bold & beautiful digital products.



For Android : MDC

- Material Design
- Components

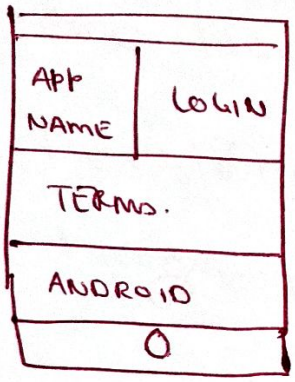
→ updates library overtimes

↓

available on web, ios & flutter.

Linear Layout :

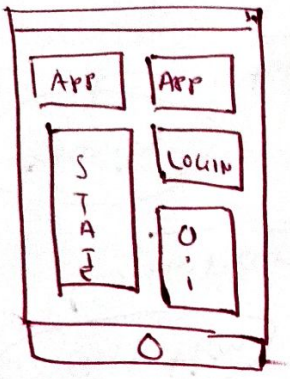
A view group that aligns all children in either vertical & horizontally



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:

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



id

gravity

ignore gravity

layout_above

layout_allignBottom → makes bottom edge of this view

" left → makes left edge of this view match

" Parent bottom → true / false.

TABLE LAYOUT : Rows & column

eg. calculator.

Collapse columns

shrink columns

Stretch columns.

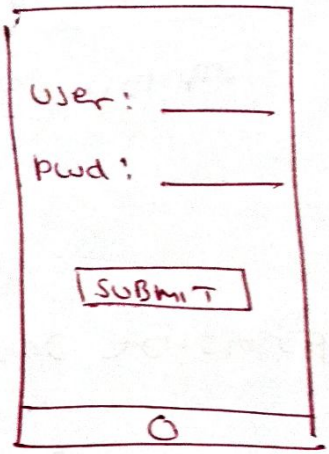
Absolute Layout

↓

→ 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 Listeners.
     - ↳ an interface in the view class
     - ↳ called for viewing which listener is registered when user triggered the interaction.

Event registeration ⟹ 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 Handlers

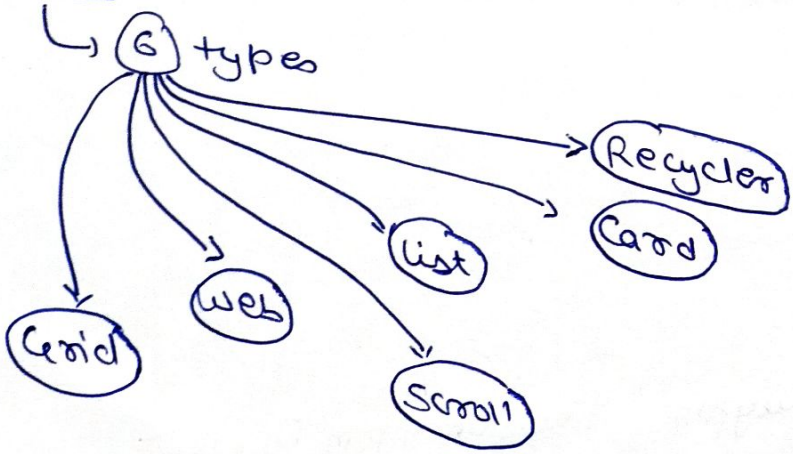
actually handles the event.

Widgets :

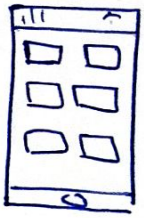
- small gadget ⟶ Placed on homescreen.
- handy
- allows you to put your favourite apps on the home screen.
  - ↓
  - easy access to them
- eg. Clocks, weather, music

Chapter 4 — MAD – Working with views & fragment.

Views



Grid View:



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

Note: ✓ Default scrollable

Example: Phone Gallery

- bridges UI & data source
- can be used to supply data

Adapter view → list view, Grid view → gets data from source → represent each data entry in Grid view

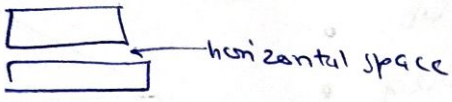
Components: ① Id : <Grid view
android:id="@+id/simpleGrid"
/>

② numColumns : • integer
• how many no. of columns

<android:numColumns="4"/>

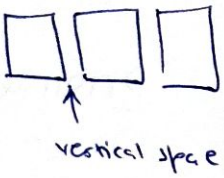
} Autofit = (auto_fit)
display as many possible column to fit

③ Horizontal Spacing : Define horizontal space b/w columns



} px, dp, sp, in or mm

④ Vertical Spacing : Define vertical space b/w rows



} px, dp, sp, in, mm.

⑤ Stretch Mode:

if => none: stretching disabled

spacingWidth: space b/w each column is stretched.

Colum width: columns are stretched equally.

spacing width Uniform: uniformly stretched columns.

(2) Web View :

⇓

Display web page in Android ⟹ Web page can be loaded from same App / URL ⟹ { Display online content }

- To view it : we need to add <WebView> to xml layout file in android activity.

Uses : Webkit engine to display web page.

↳ android.webkit.WebView → subclass of AbsoluteLayout class.

{ loadUrl() / loadData() } → methods for → loading & displaying web page

Create an object in .java file :

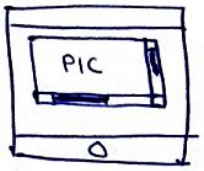
```
WebView browser = (WebView) findViewById (R.id.webview);
```

To load a web url into webview :

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

③ Scroll view :

- Useful to add a horizontal / vertical scroll bar to



A very large picture or content

- The android ScrollView can hold any one direct child.
- Enable scrollview for → scrolling
- disable scroll view for using → ListView & Grid view

because they can take care of their own scrolling.

- Default : vertical ~~direction~~ of scroll view (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 => displayed via ListView.
- helps in → displaying data in form of a scrollable list.
- selection → click on any items.
- 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_700"
    android:dividerHeight="1dp"
    android:listSelector="#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 block of array data structure we use

For instance:

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

Default : Array Adapter creates a view → for → each item in Array

How? → by calling toString() → on each item → Placing the contents → in → Text View

Example : Lets say you have Array of Strings → YOU WANT TO DISPLAY IT IN → LIST VIEW

St initialize "ArrayAdapter" using → constructor to specify the layout of each string

```
ArrayAdapter = new
ArrayAdapter<string> (this, R.layout.ListView, StringArray);
```

(5) Recycler View :

↳ Advanced version of **ListView** → with → 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.

DATA SUPPLY → Define → how will each items look? } Recycler_View library

↓

[Dynamically creates the elements.]

RecyclerView ← recycled by

Explanation:

(1) When item is scrolled off the screen,

RecyclerView reuses the view for (new items.)

It don't destroy the view but rather reuse it → improves → PERFORMANCE & POWER CONSUMPTION

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**

```
dependcies {
    implementation "androidx.cardview:cardview:1.0.0"
}
```

- card_view
  - id
  - layout_gravity
  - layout_width
  - layout_height
  - card_corner_radius

→ Elements of **Card View**

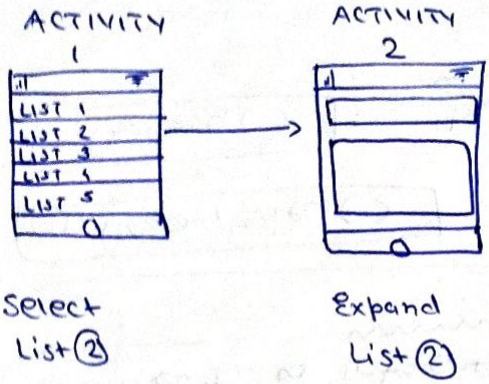
Annexure No :

FRAGMENT :

TLDR;

A - sub activity

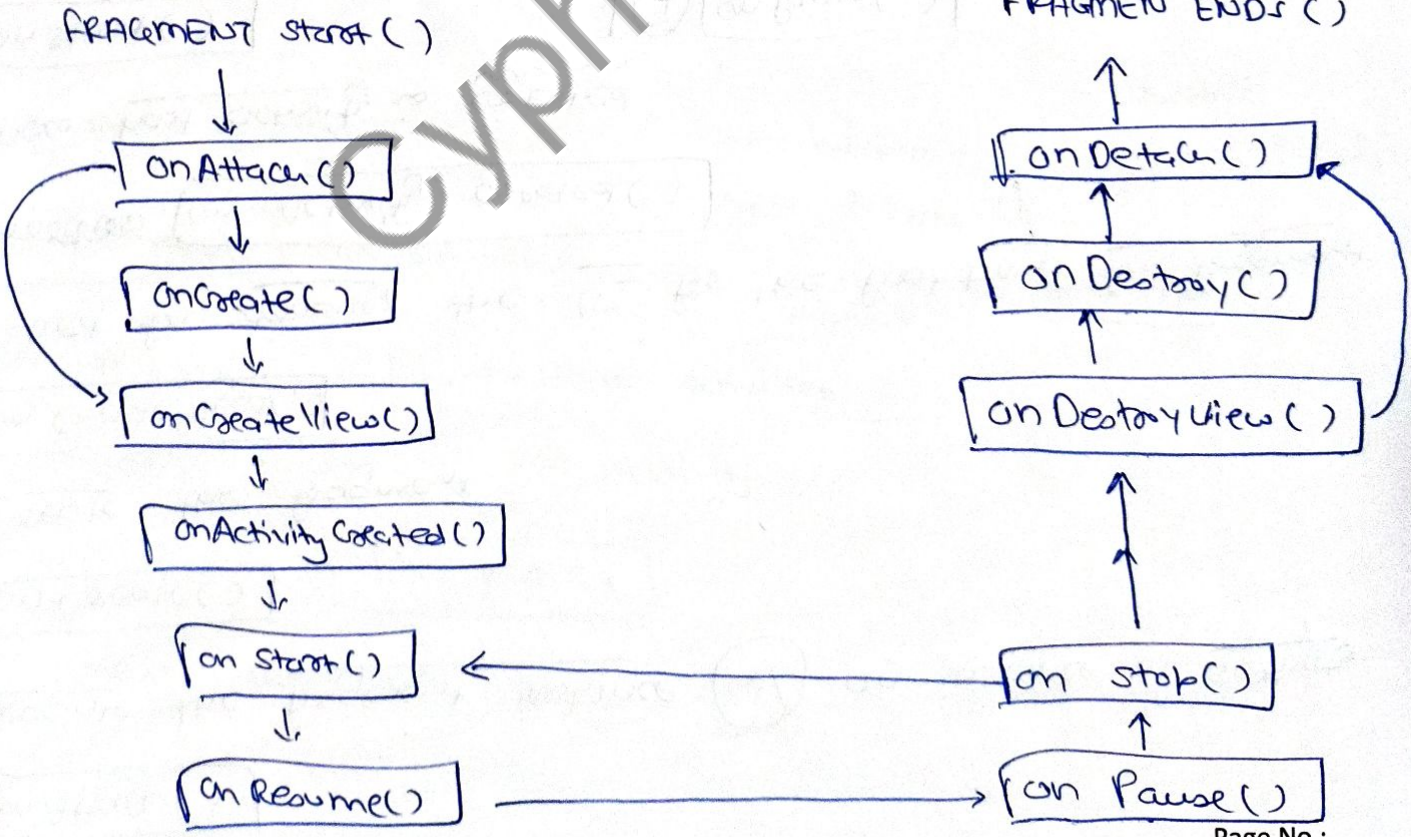
Piece of activity which enables more modular activity design



[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 Start

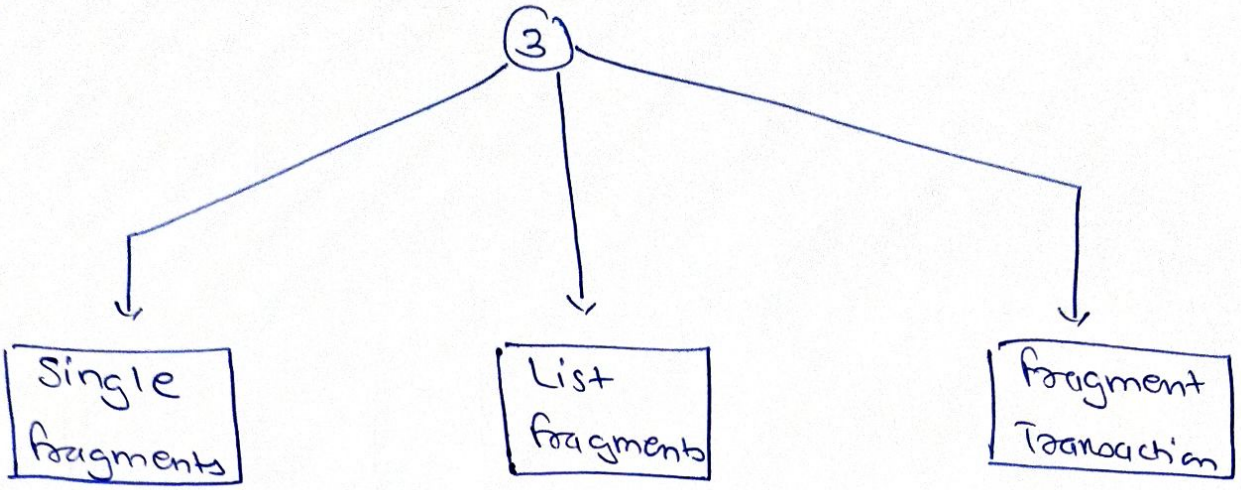
1. onAttach()
   - associate the fragment instance w/ an instance of activity
2. onCreate()
   - create the fragment
3. onCreateView()
   - called for drawing the UI for the first time for fragment
4. onActivityCreated()
   - new host activity is created
5. onStart()
   - fragment is visible.
6. onResume()
   - fragment is interactable

7. onPause()
   - fragment no longer interactable.
   - user is leaving the fragment
8. onStop()
   - fragment is not visible

9. onDestroyView() : resources in onCreateView() is removed
10. onDestroy() : destroy the fragment.

Annexure No :

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 life cycle.

TATAKAE!!!

(ii) Add & Remove can be done while activity is running.

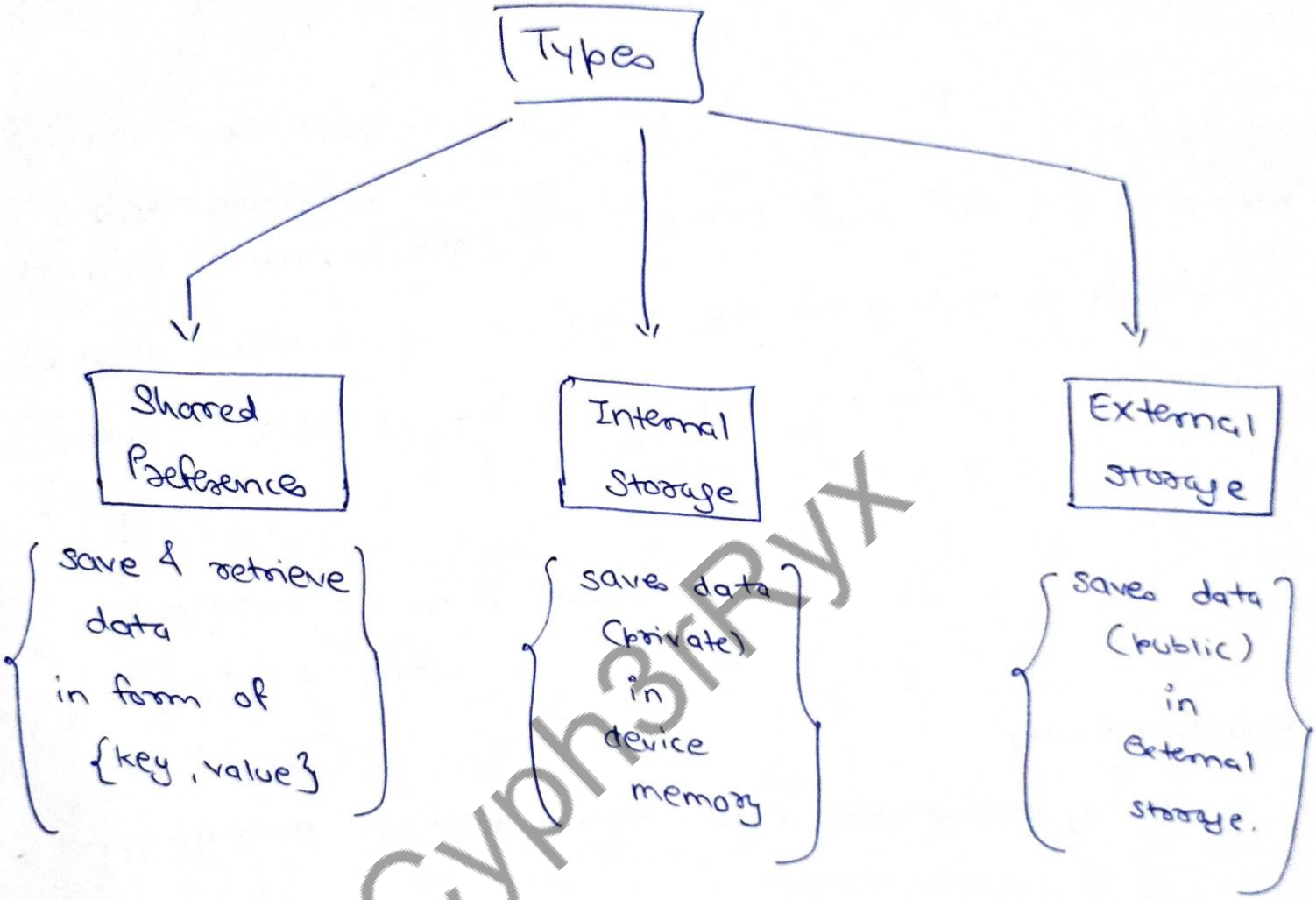
(iii) One fragment ⟶ multiple activities

(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



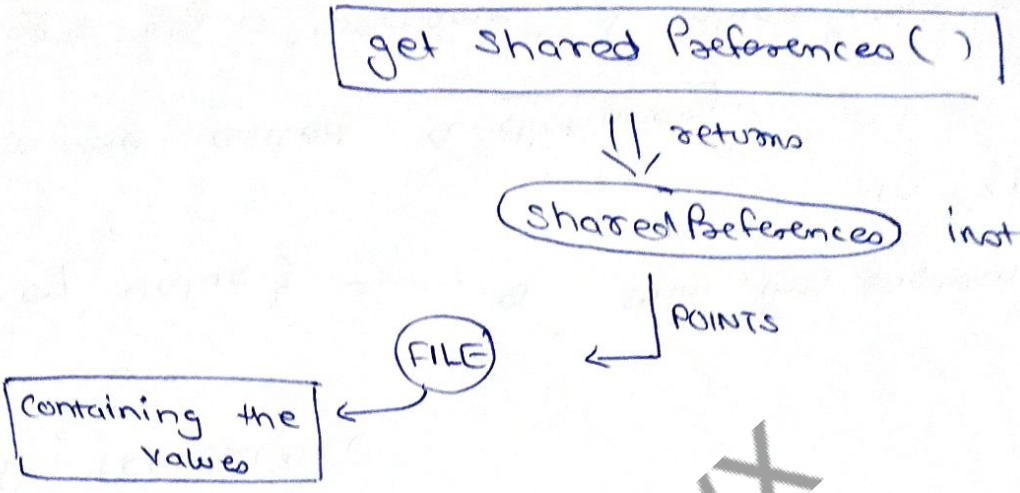
SHARED PREFERENCE :

- {key, value} ——→ eg String, int, boolean
  in XML file
- → can be called a dictionary.

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

Usage:

You need to call a method



eg.

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

here, MyPref. = key

Context.mode = value / MODE

Diff. Modes:

1. MODE_APPEND → joins new w/ existing preference.
2. MODE_MULTIPROCESS → checks for modification of preference.
3. MODE_WORLD_READABLE, MODE_WORLD_WRITABLE → Allows other app to read & write
4. MODE_PRIVATE → only accessed Using application

Annexure No :

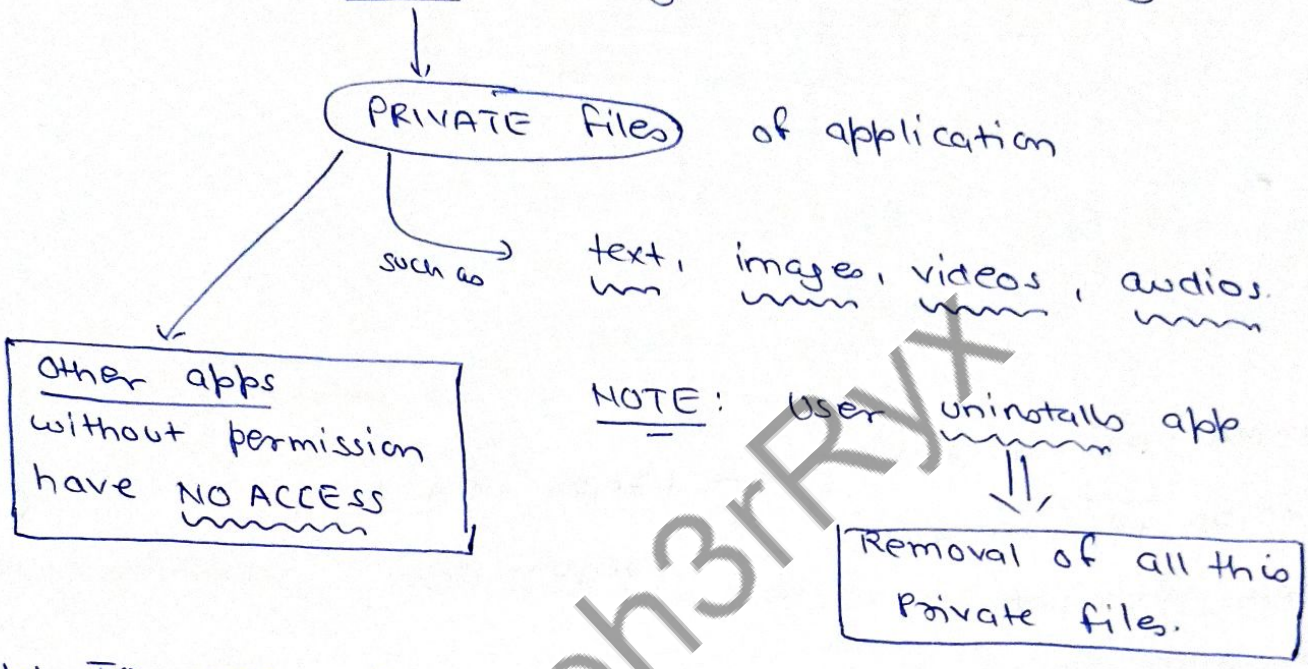
You can also save something in SharedPreference by .Editor class:

```
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:

↳ saves files directly to 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()

eg.

```
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

openFileInput()

eg. ↳ for reading file data

```
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 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:

↳ saves data in SD card / External device

data → Public → can be read by all the user

Add Permission → to → Read / write

< 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 isExternalStorageWritable() {
    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/ `getExternalStorage.State()`

(3) Run "if" loop

(4) Check result.

for checking if `External Storage` is Readable or not:

Just change boolean data to `isExternal Storage Readable()`

& `Environment.MEDIA_MOUNTED_READ_ONLY`

To write the DATA:

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

```
File folder = getExternalStorageFilesDir("myfolder");
File myfile = new File(folder, "mydata.txt");
```

Here `getExternalStorageFilesDir(String type)`

→ used to create temporary files just for app

but `getExternalStoragePublicDirectory(String type)`

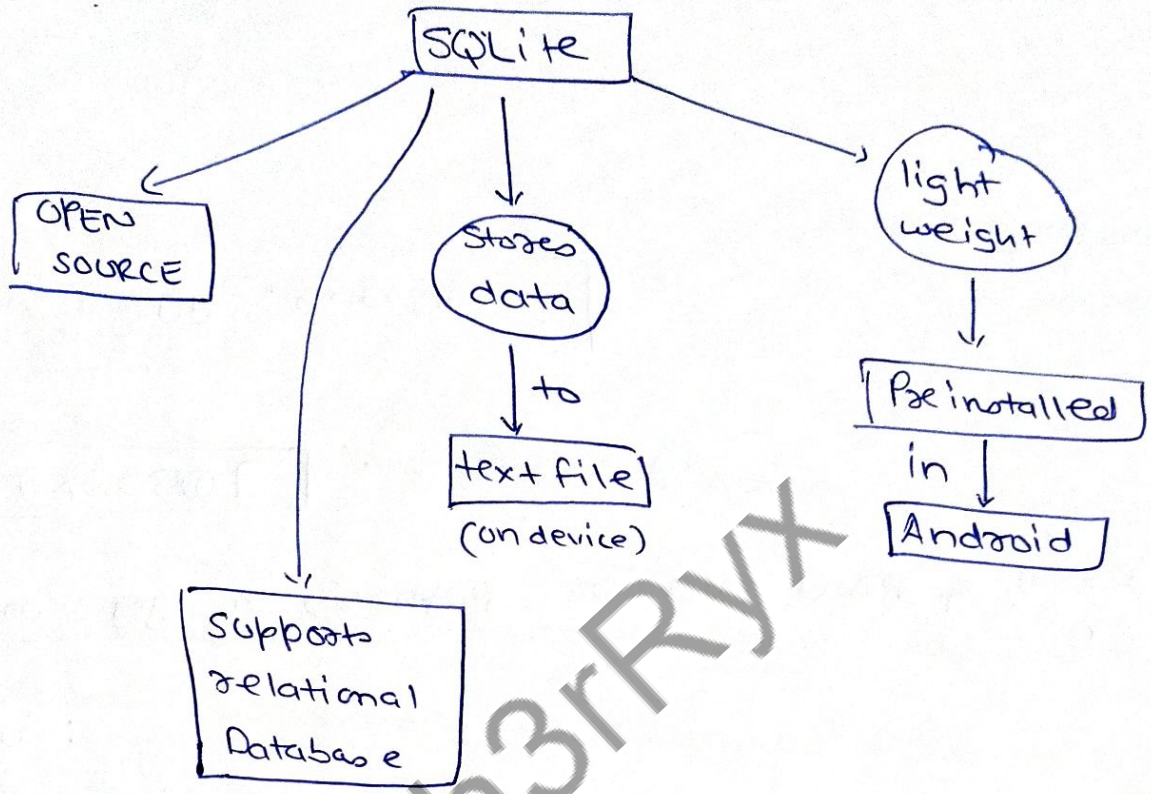
→ creates a permanent file in External Storage.

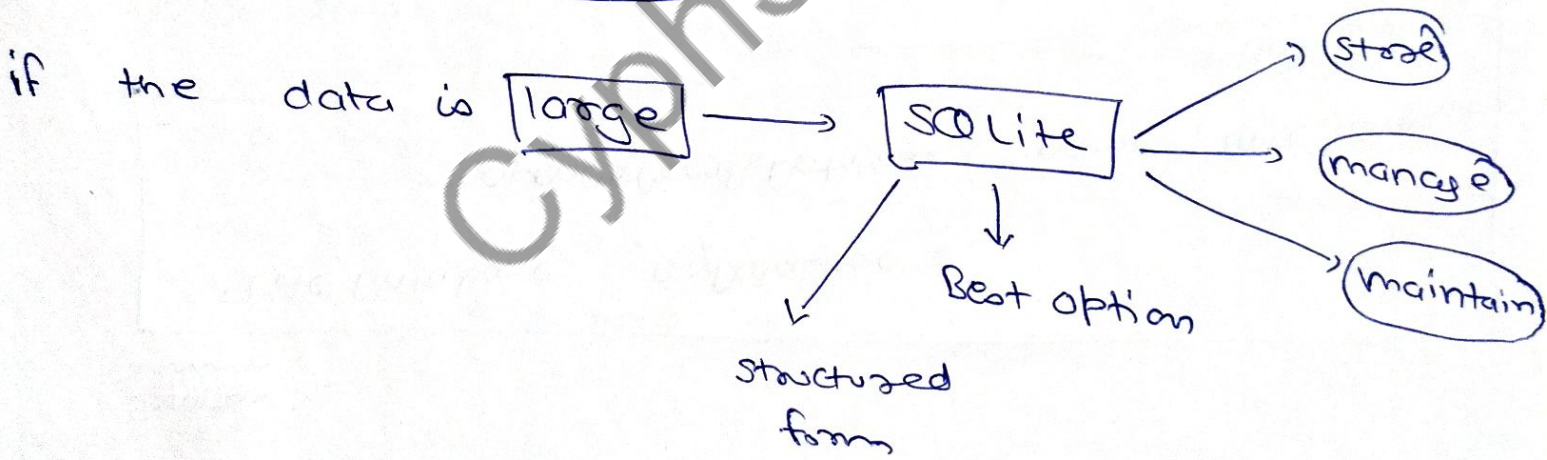
Now To read the DATA:

Same code as the internal Storage

Annexure No :

SQLite :





Android Package = `android.database.sqlite`

Create Database:

① Call `openOrCreateDatabase` with your `DB name` & `mode` as parameter.

returns ⟶ `SQLite DB instance`

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','lmaoded');");
```

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 = lmaoded

NOTE : execSQL is also used to { update, modify, insert the data } to an already existing data in DB

HOW ← Using bind argument

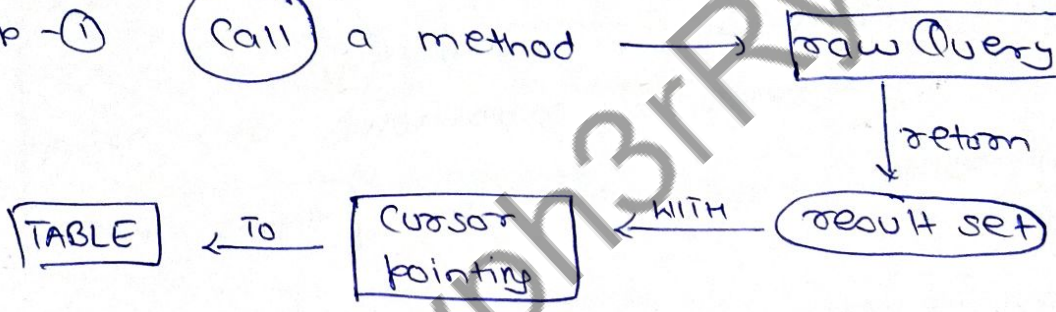
Fetching Database:

Once Creation / Insertion is checked ✓

we move to "fetching" phase.,

Using → Cursor class

Step-① Call a method → rawQuery



S-② we can move Cursor → forward & retrieve data.

SYNTAX:

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

Annexure No :

Exp.

First: We define [Cursor] for result set
we define/(call) rawQuery for Table "Ryx" & arguement as null

Second: We [move] the cursor forward for (retrieval) of data

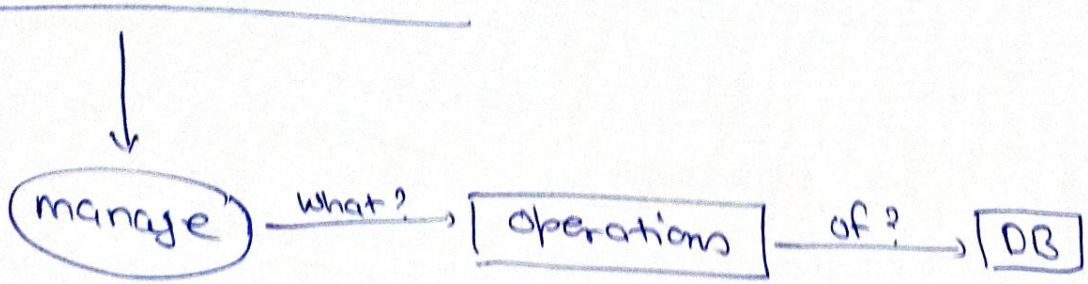
Third: We get "username" as String type.

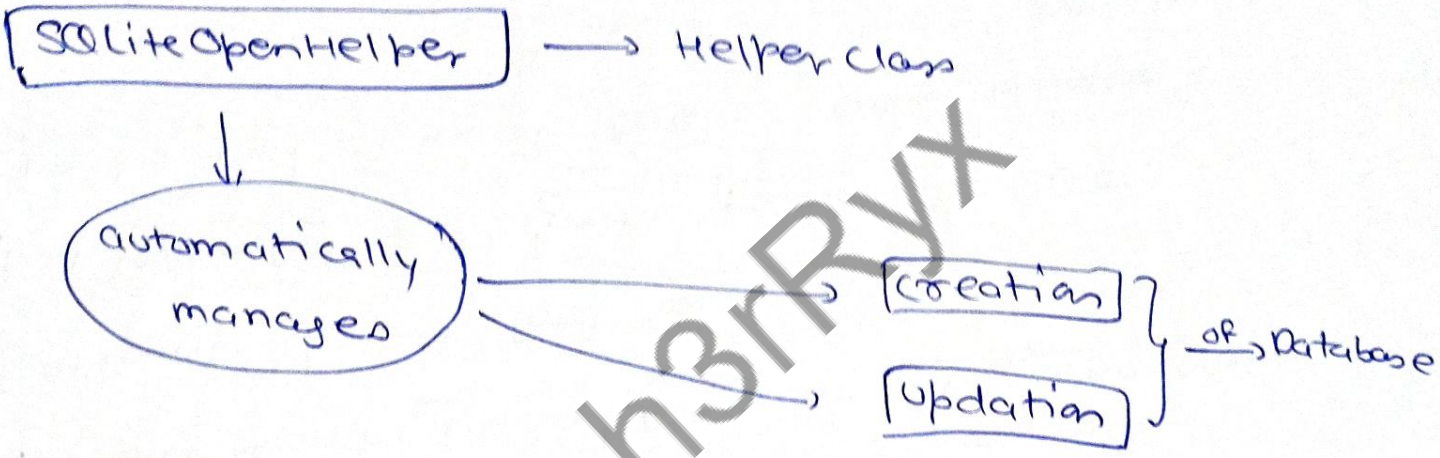
Fourth: We get "password" as String type.

Cursor class methods:

1. [getColumnCount()] → Total columns
2. [getColumnName()] → Array of all column names
3. [getCount()] → Total rows
4. [getPosition()] → current position of cursor.
5. [isClosed()] → if (true) => cursor is closed
   if (false) => " " not closed.

Database Helper class :





SYNTAX :

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

Exp. We define a public class DB helper which uses SQLiteOpenHelper

- We give then command for DBhelper() via a "Constructor"
- The Constructor takes Context as a parameter (to) initialize superClass (SQLiteOpenHelper)
- "super" calls constructor & passes

Context	→	object
DATABASE_NAME	→	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
  - eg. SQLite

Content Resolver:

A mediator between (Content Provider) & (Application)

Provides → UI for (accessing) Content Provider
→ UI for (managing) data.

Used → (query, insert, update, delete) → from Content Provider

Gives → methods for observing changes in DATA in Content Provider.

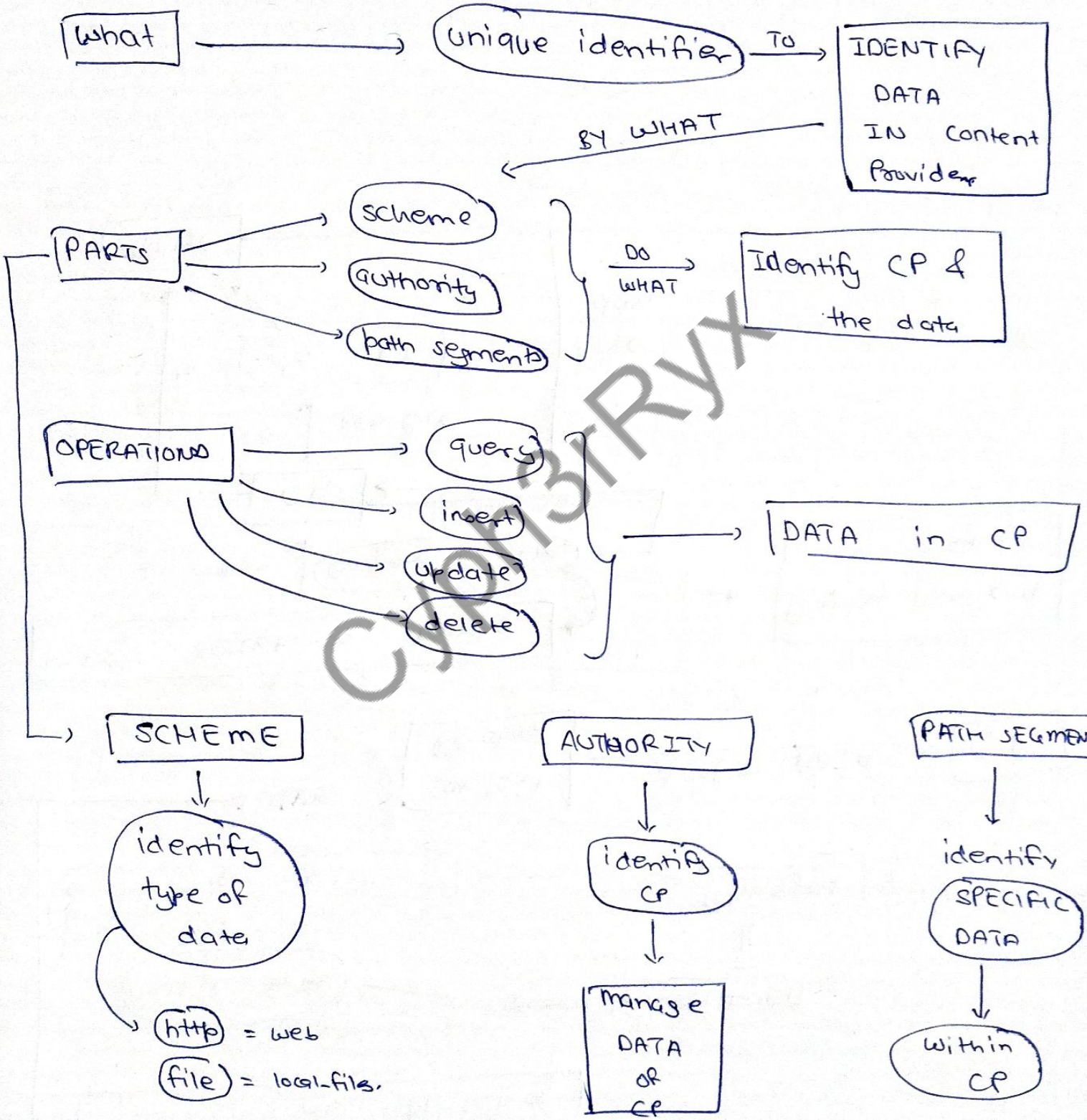
→ (interact, fetch, modify)

- Installed Apps → (content:// URI scheme)
- File systems → (file:// URI scheme)

Annexure No :

CP * = Content Provider.

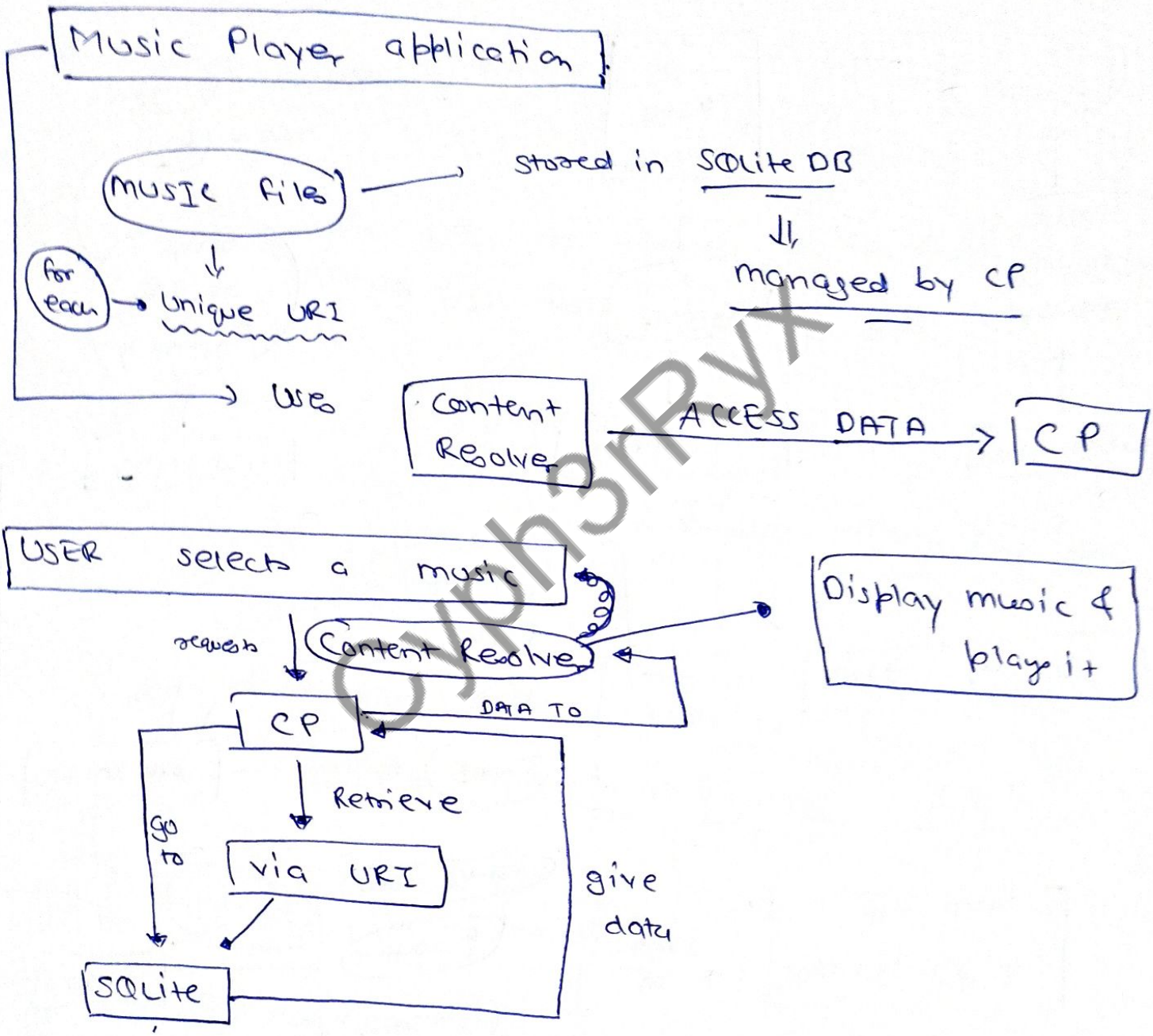
Content URI's :



Page No :

Enrollment No :

Example:



UNIT – 6 Web App Integration.

Android Async Task

- DO WHAT ? ⟶ Background Operation

on

Background Threads

TO

UPDATE MAIN THREAD

can't be touched directly

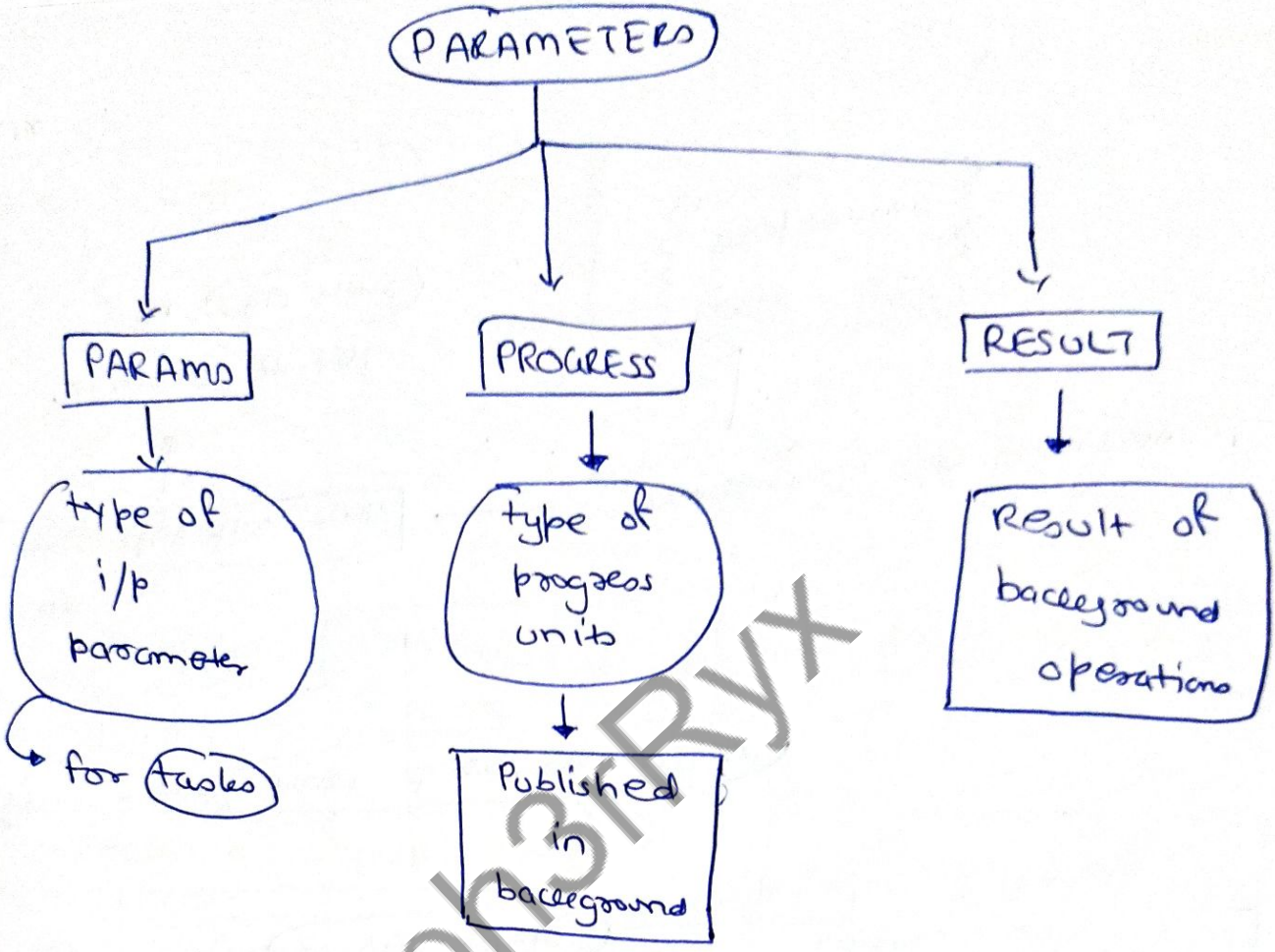
Thus Async Task ⟶ communicates b/w 2 threads.

- USED for ⟶ short duration Task

also can do long duration task ⟶ asynchronously in background

long duration task ⟶ requires a lot of CPU time

asynchronously in background ⟶ keeps UI responsive



Methods :

1. **onPreExecute()** → setup **pre-conditions** required for background operation.
   (e.g. shows progress bar)
2. **doInBackground()** → A **main method** where you can perform **background operations**.
   (PARAMS)

3. `onProgressUpdate()` (Progress) → Publish progress update (in) UI thread about Background operations.

4. `onPostUpdate()` (Result) → called when Background operations are completed.

Start Async Task via `execute()`

Cancel Async Task via `onCancelled()`

Web API

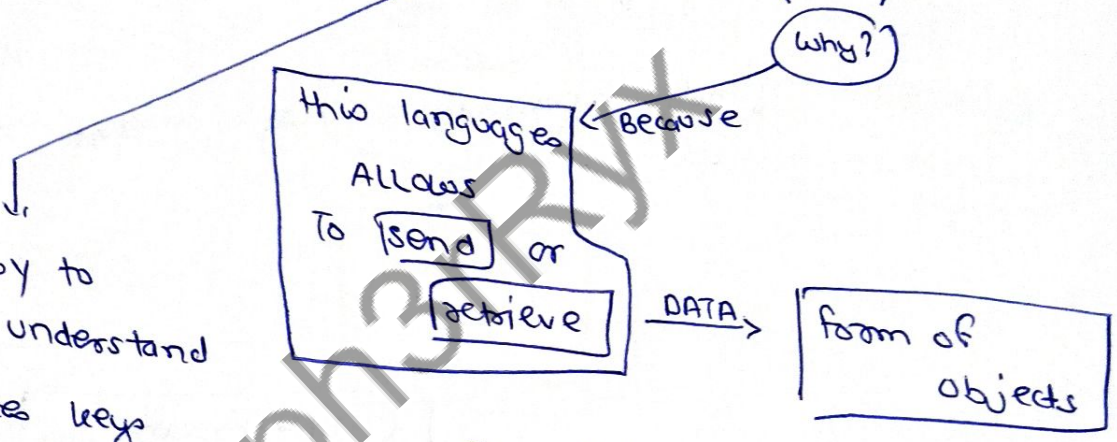
How to use them from Android app.?

API = Application Programming Interface.

↓

allows developers → To → interact w/ external services.

- API are commands set by developer to use some of features of their program.
- Most of them works → XML or JSON



- easy to understand
- places keys inside < >

eg. <client>
<name> Jeff </name>
<age> 32 </age> </client>

JSON → JavaScript Object Notation

send DATA Online → form → {value / attribute}

eg. [{{client:
{"name": "Jeff",
"age": 32}}]

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.

---

contd. JSON —DATA→ PARSING —CONVERT→ JAVA OBJECT via JSON Parsing Libraries.

To use Web API which returns JSON Data:

S-1 make HTTP request to API endpoint via Volley

↓ will

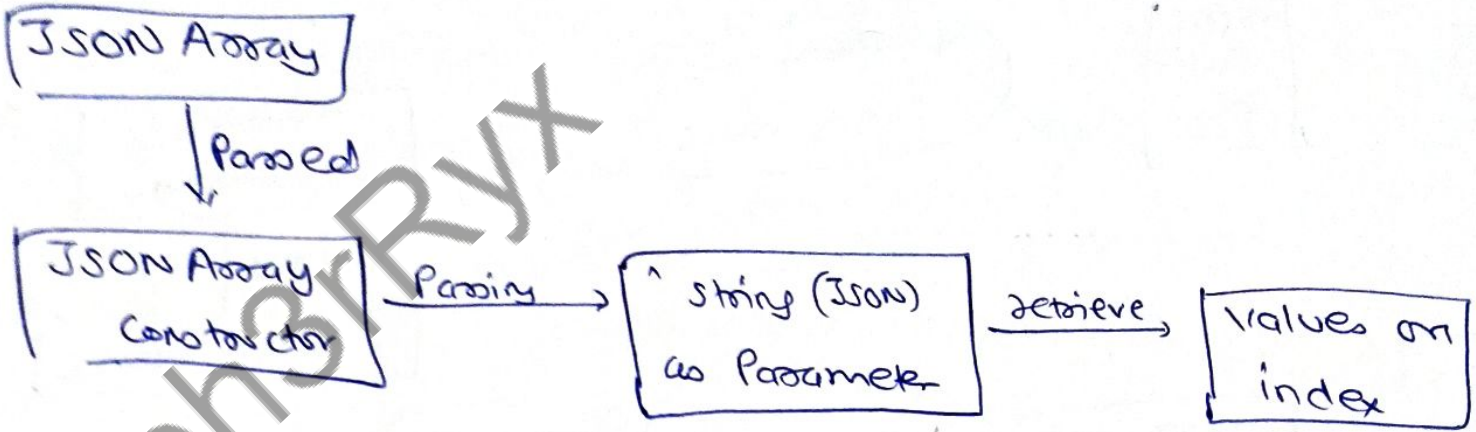
respond w/ JSON Data —PARSE USING→ JSONObject / JSONArray

S-2 JSON Object

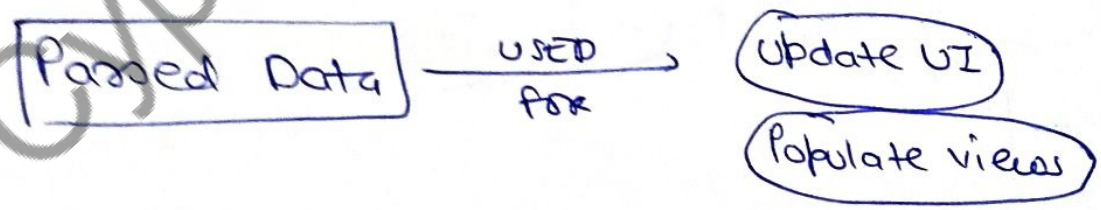
↓ PARSED

JSONObject constructor —Passing→ JSON string as Parameter —gets→ keys or index

S-3



S-4

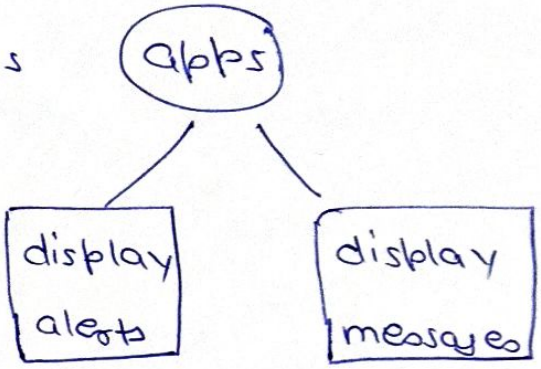


NOTE:

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

NOTIFICATION :

It allows apps



from the App's main UI

→ Can be displayed via banners.

pop-ups

Status bar icons

HOW TO USE THEM?

S-1 Create a Notification object by NotificationCompact.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 —when→ user clicks it

(4) Show notification

via calling `NotificationManagerCompact.notify()`

Telephony API :

↓

Set of fx^n & protocols

to interact with a telephone system / device

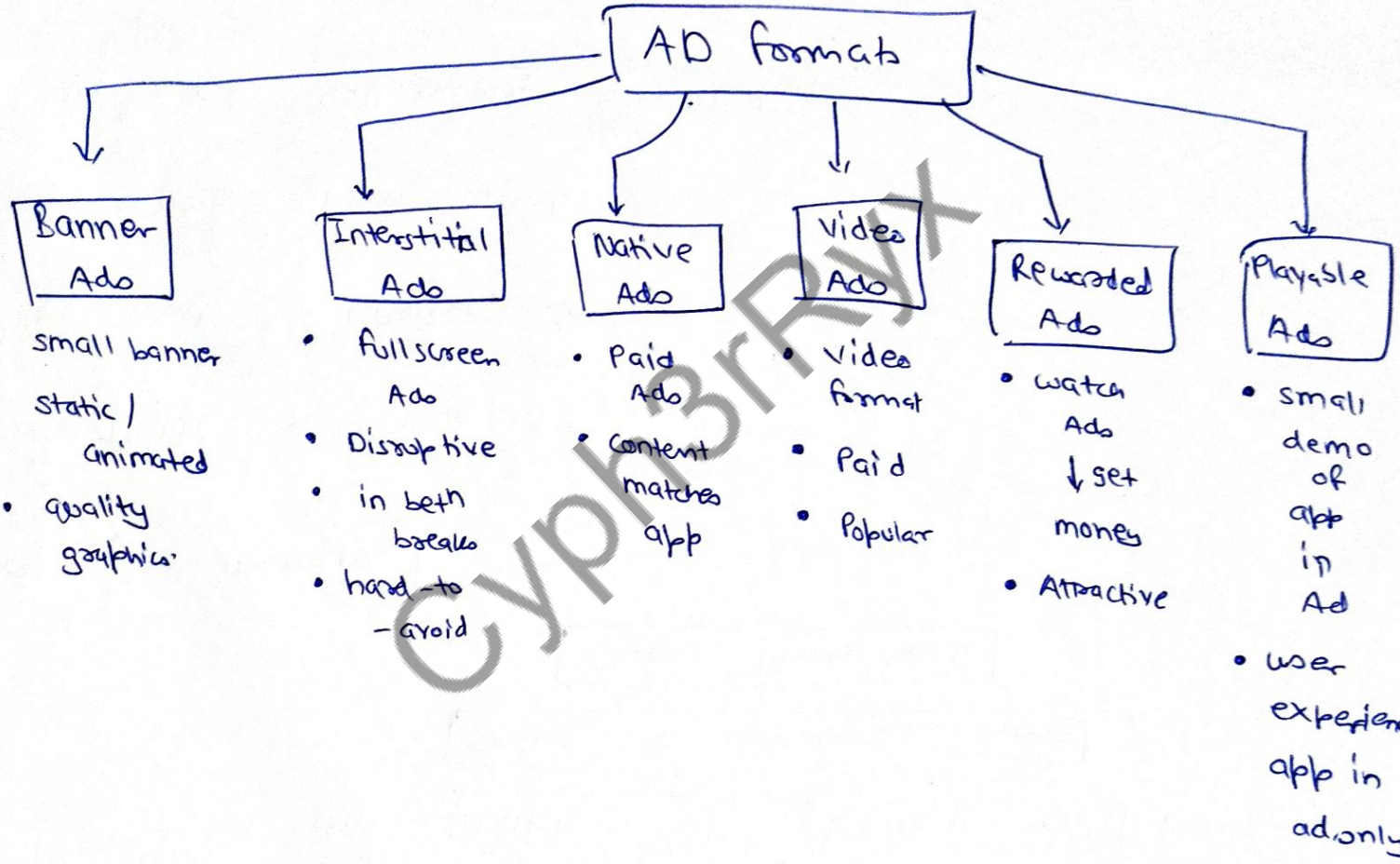
→ can be used while creating a system that can [make call], [recieve call], [send & recieve SMS], [access Phone book]

eg. [Call manager]

1. Make & Recieve calls
2. Send & Recieve sms
3. Access Phone book
4. Access call history & call logs.

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