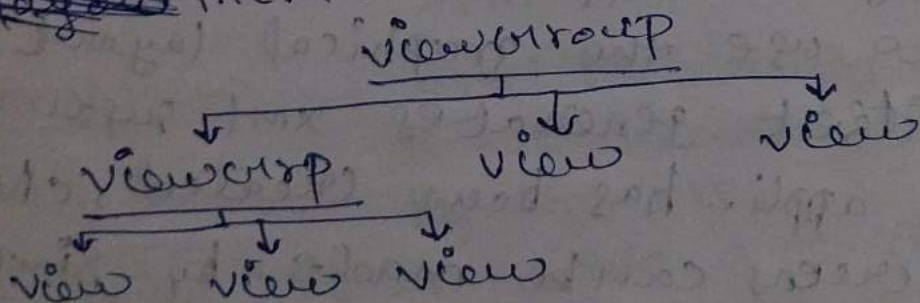


## Module - III

### Android User Interfaces

- \* (An) UI is everything that the user can see & interact with.
- \* (An) SDK provides text fields, btns, lists, grids & so on. In addition, (An) provides a collection of controls that are appropriate for mob devices.
- \* At the heart of the common controls are 2 cses:  
android.view.view & android.view.viewgroup.
- \* view is an obj that draws smthg on the screen that the user can interact with.
- \* view group is an obj that holds other view obj in order to define the layout of the ~~target~~ interface.



view, Target Control } each of these represents a UI element.



2) Containers → views used to contain other views.

3) Layout → visual arrangement of containers  
eg views & can include other layout.

## I Building a UI in code:

- You can choose from several approaches to build UIs in (Android).
- You can construct UIs entirely in code. You can also define UIs in XML.
- You can even combine the 2 — define the UI in XML & then refer to it, modify it in code.
- Java code is ideal for creating UI dynamically at runtime.

## II Building a UI in XML:

- Key adv of XML approach includes the ability to use the graphical layout tool, which itself generates XML resources.
- Once an app has been created, changes to UI screens can be made by simply modifying XML file without recompiling app.

## III UI in XML with code:

eg → <LinearLayout>

android:orientation = "vertical"

" : layout\_width = "match\_parent"

" : layout\_height = "match\_parent" >

<EditText>

a : id = "@+id/edit\_msg"

a : l\_w = "wrap"

a : l\_h = "wrap"

a : hint = "Enter a msg" />

<Button>

a : l\_w = "wrap"

a : l\_h = "wrap"

a : text = "Send" />

</LinearLayout>

→ Enter a msg

• EditText → used to accept text input from the user.

• TextView → used to display text that is not intended to be edited by the user.

• android:hint → provide a placeholder text that is displayed when the field is empty.



### III UI in XML with code:

- we should design our UIs in XML
- then reference the controls from code.
- This approach enables us to bind dynamic data to the controls defined at design time.
- This is the recommended approach.

### IV (An) Controls:

#### 1) Text controls:-

- \* These control provide an Editable text field
- \* You should always declare the `id` method for `xml` text fields by adding the `android:id` attribute to the `EditText` element.

~~eg~~ `<EditText`

eg → `<EditText`

`android:id="@+id/phone"`

`android:layout_width="fill_parent"`

`android:layout_height="wrap_content"`

`android:hint="Enter phone no."`

`android:inputType="textPhone"/>`

→ display `Phone` non keypad.

for password → `android:inputType="textPassword"`

`textPassword`

for enabling auto spelling correction

`android:inputType="text|textAutoCorrect"`

`textAutoCorrect`

→ Hey buddy. Where are you? `text`

#### \* 3 categories —

#### 1) TextView:

most widely used view used to show pre-defined text on display screen. It is a complete text editor, however the basic `cls` is configured to not allow editing.

#### Attributes

`android:text`

`android:textSize`

`android:textColor`

`android:textAllCaps="true"` [make the text appear in upper case]

`android:hint`

`android:letterSpacing` [spacing b/w letters of the text]

`android:editable="true"` [specifies this `TextView` has an `id` method]

`android:specifies this TextView has an id method]`

eg →

`<TextView`

`android:id="@+id/text-id"`

`android:layout_width="wrap_content"`

`android:layout_height="wrap_content"`



"text = "I am a textView" />

Java -

TextView myText = (TextView) findViewById(R.id.textView);

## b) EditText()

- \* It is an overlay on TextView that configures itself to be editable.
- \* It is the predefined subset of TextView that includes rich editing capabilities.
- \* attributes -

- inputType : none, textEmail, Phone, textAutoCorrect, gravity → control alignment of text like L, R, C, T, B

- textStyle → add bold, italic & normal.

android:padding = "15dp"

" : textColorHint = "#ffff" (hint color, text color)

" : textStyle = "bold | italic"

" : background : "#0000" />

Java -

EditText editText = (EditText) findViewById(R.id.simpleEditText);

String editTextValue = editText.getText().toString();

## c) AutoCompleteTextView

- ~~textView~~ is a textView with auto-complete (-ity).

To implement auto complete, we have to specify an adapter that provides text suggestions.

eg → <AutoCompleteTextView

android:layout\_width="match\_parent",  
android:layout\_height="wrap\_content",  
android:layout\_margin="10dp" />

~~define~~ the array that contains all text suggestions

<resources>

<string-array name="c\_array">

<item> chives </item>

<item> indies </item>

</string-array>

</resources>

Java -

AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id....);

// get string array

String[] countries = getResources().getStringArray(R.array.c\_array);

## 2) Button controls :

- \* consist of text / an icon / both text & icon that communicates wth actions occurs when the user touches it.



## \* Types -

### a) Button control (basic btn):

basic btn cls in (An) is `android.widget.Button`

xml code -

```
<Button
    android:id="@+id/button1"
    android:text="@string/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

btn controls -

\* `onClick` - when user clicks a btn.

add `android:onClick` attribute to btn in xml

called when user touches a btn.

```
public void onClick(View view) {
    // btn actions
}
```

### b) ImageButton control:

(An) provides a img btn via `android.widget.ImageButton`

xml code -

```
<ImageButton
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

an:onClick = "myClick"  
an:src = "@drawable/icon" (save img in haw)

java code -

`ImageButton imgbtn = (ImageButton) this.findViewById(R.id.imgbtn2);`

`imgbtn.setImageResource(R.drawable.icon);`

### c) Btn with text & icon:

xml code -

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="yes"
    android:drawableLeft="@drawable/icon" />
```



### d) ToggleButton control:

It is a 2-state btn & can be in either the ON or OFF state.

Its default behaviour is to show a green bar when in the ON state & a grayed-out bar when in the off state.

xml code -

```
<ToggleButton
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```



an: textOn = "Stop" />  
 an: textOff = "Run" />  
 → toggle btn on program's stop screen that  
 any btn-on will remain display on the

### 3) Check box Control:

- \* Allow user to select 1 or more options from a set. It is a type of 2 state btn either checked / unchecked.
- \* c-box cly in (An) is android.widget.CheckBox.
- \* methods—
- 1) public boolean isChecked() → returns T if it is checked.

2) public void setChecked (boolean state)  
 → changes state of c-box.

\* eg → <CheckBox

```

  an:id="@+id/chicken"
  an:text="chicken"
  an:checked="true"
  an:layout_w="w_c"
  an:layout_h="h_c" />
  (default true android:state_checked="true")
  </CheckBox>
  
```

Java code—

CheckBox box = (CheckBox) findViewById(R.id.chicken);  
 // check current state of c-box (T/F)

Boolean checkboxState = box.isChecked().  
 // returning T/F to c-box state.

### 4) Radio button:

- \* Gives the users several choices & forces them to select a single item.
- \* It generally belong to a grp, and each grp is forced to have only 1 item selected at a time.
- \* To create a grp of R-btn in (An), 1st create a RadioGroup & then populate the grp with R-btns.
- \* Implementing R-grp using an.widget.RadioGroup.

R-btn using an.widget.RBtn.

\* eg → <RadioGroup

```

  an:id="@+id/btngroup"
  an:layout_w="w_c"
  an:layout_h="h_c"
  an:orientation="vertical"
  
```

```

  <RadioButton
    an:id="@+id/bt1"
    an:text="chicken"
    an:layout_w="w_c"
    an:layout_h="h_c" />
  
```

```

  <RadioButton
    an:id="@+id/fishbtn"
    an:text="fish"
  
```

</RadioGroup

[R-grp-on 22222  
 means 5 btns are  
 using / displaying]



Java code -

RadioButton fd = (RadioButton) findViewById(R.id.checkbox);

Boolean state = checkbox.isChecked();

### 5) ImageView:

- \* used to display an img, where the img can come from a file, a content provider, a resource such as a drawable.
- \* we can even specify a color, & the ImageView will display that color.
- \* Implemented using an widget - ImageView
- \* eg -> <ImageView

1>

Java -

ImageView imageView = (ImageView) findViewById(R.id.imageView1);

imageView.setImageResource(R.drawable.capture);

(XML -> an image -> drawable resource file and imageView -> img)

### 6) Date & Time Controls:

- \* (And) provides controls for the user to pick a time / pick a date as ready-to-use dialogs.

\* Each Picker provides controls for selecting each part of time (hr, min, AM/PM) or date (month, day, yr).

\* (And) Date Picker: Allows to select the date consisting of day, month & yr in yr

user can use interfaces.

For this (And) provides DatePicker


Attributes -

- 1) id - used to uniquely identify a date picker.
- 2) datePickerMode -> set the date picker in mode either spinner / calendar.

3) background -> used to set the background of a date picker.

4) padding -> to set padding.

Methods -

1) setSpinnerShown (true) -> 

" (false) ->  only.

2) getMonth(), getYear, getDayOfMonth.

eg -> <DatePicker

android

" id

" min

" max

android:calendarMode = "spinner" />

Java -



DatePicker SimpleDateFormat = (d.p) find view by id  
(R.id.  $\rightarrow$ );

for getting day = "Day = " +  $\rightarrow$  .getDayOfMonth();

### \* Time Picker :

- used for selecting the time of the day in either AM/PM mode 12-hrs mode.
- displayed time consist of hr, min & clock format.

Implement ~~using~~ using an widget. TimePicker

Methods -

1) setTime, setTime

2) setIs24HourView (true)  $\rightarrow$  show view as 24-hr not AM/PM.

3) is24HourView ()

an : timePickerMode = "Spinner"  $\rightarrow$   $\frac{08}{10} \frac{12}{09} \frac{AM}{13} \frac{PM}{14}$

Java  $\rightarrow$

as DatePicker

singlePicker, setIs24HourView (false);  
 $\rightarrow$  to display AM/PM mode.

(elegent - clock)

### 7) Map View control :

\* (An) always to integrate google maps in an app.

\* You can show any loc on the map & also customize the map according to yr choices.

\* main interfaces & class for handling map -

- GoogleMap
- MapView
- SupportMapFragment.

\* GoogleMap automatically performs these operations -

- connecting to Google map service
- downloading map files
- displaying files on the device screen.
- displaying various controls like Pan & zoom

\* add a <fragment> element to the activity's layout file to define a fragment obj.

### IV Adapter = (ad)

\* An (ad) act like a bridge b/w a data source & the user interface.

\* It reads data from various data sources convert it into view obj's & provide it to the linked (ad) view to create UI components.

\* (An) SDK also provide some ready-to-use (ad) class like ArrayAdapter, Cursor (ad), Simple (ad), SimpleCursor (ad).

view

viewmodel

AdapterView

ListView

GridView

Spinner

RecyclerView



\* eg → let's use an array as the data set

```
String[] cheese = {
    "ABC",
    "cde",
    "fgh"
};

ArrayAdapter<String> cheeseAdapter =
    (type of data the adapter will handle, here it is a String array)
    new ArrayAdapter<String>(this,
        R.layout.item,
        R.id.checkbox_name,
        cheese);
```

## Adapter View :

- \* can be used to display large sets of data efficiently in form of list / grid, etc provided by an adapter to it.
- \* capable of displaying millions of items on the UI
- \* It only renders those view obj which are cntly on screen, hence saving mly.
- \* list controls are class that extend an widget. Adapter view to include listview, GridView, Spinner & Gallery.

## a) List View :

- \* displays a list of items vertically
- \* we generally use listview by extending a new activity that extends an app. Legtactivity

\* eg → <LinearLayout>

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:content="LegtActivity"
```

<listview>

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

<listview> (LinearLayout) by mobile instead of LinearLayout

## b) Grid View :

- \* A type of view that displays items in a 2D scrolling grid.
- \* items are inserted into this grid layout from a DB / from array.
- \* Adapter (used for displaying this data, set Adapter()) is used to join the adapter with GridView.

Gridview	item1	item2
item1	item1	item2
item1	item1	item2
item1	item1	item2
item1	item1	item2

Listview
Android
Java
Python
Ruby
C++
PHP
JS



\* eg → main.xml

← vertical view

an:id =

an:lw =

an:lh =

new column

in the grid

stretch the column

if there is extra space available

specifies the gravity of the layout fitting the grid

center → items are centered both horizontally & vertically

an:gravity = "center"

an:columnWidth = "wrap"

1 >

c) Gravity control:

in a vertical view is a view used to show items

\* you can allow the user to select items from their device

The gravity is a view that shows items in a center-locked, horizontal scrolling list.

\* belongs to android.widget.HorizontalScrollView

\* eg → <gravity>

an:id =

an:lw =

an:lh =

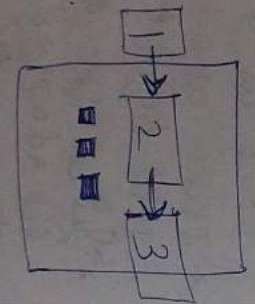
1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"



d) Spinner control:

\* It is like a drop-down menu.

\* Typically used to select from a relatively short list of choices.

\* only 1 value will be displayed when the spinner is at rest.

\* eg → <spinner>

an:id =

an:lw =

an:lh =

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

an:gravity = "center"

an:columnWidth = "wrap"

1 >

Home
Home
work
offices

VI (An) Styles =

\* A style is a collection of attributes that specify the appearance for a single view

\* A style can specify attributes such as font color, font size, background color, etc

\* A style is defined in a XML resource that is separate from the XML that specifies the layout.

\* This XML file resides under res/values/dimens.

\* Multiple styles per file can be defined using <style> tag but each style will have its name that uniquely identifies the style

\* (An) style attributes are set using <dimens> tag.

\* eg →



## style.xml

<?xml -

<resources>

<style name = "Custom font">

<item name = "an:lw"> F-P </item>

<item name = "an:lh"> w-E </item>

" = "an:capitalize"> Character </item>

" = "textsize"> 12pt </item>

</style> </resources>

So, style define as above, now we want to apply this style

using style:

once style is defined, it can be used in xml layout file using style attribute

<TextView

an:id

style = "@style/customfont"

an:text = "hello world"/>

(An) style inheritance:

In (An) by using parent attribute in <style> we can inherit (pro) from an existing style & define only the attributes that we want to change/add.

eg -> <style name = "TextViewStyle" parent =

"@android:style/Widget.TextView">

<item name = "an:textColor"> #888 </item>  
<item name = "an:textStyle"> bold </item>

</style>

To inherit uses defined styles after add style name add. then add new name -

<style name = "TextViewStyle.Blue">

oldname newname.

<item name = "an:textColor"> #0000ff </item>

<item name = "an:textStyle"> static

</style>

(An) Themes:

\* It is a collection of named resources which can be referred later by style, layout, etc.

\* They provide semantic names to (An) resources. So you can refer to them later. eg -> colorPrimary.

\* A theme can be created in same way as creating styles.

\* There is applied with the an:theme attribute on either <activity> tag or <activity> tag in (An) manifest.xml file.

\* eg -> <manifest>

<application an:theme = "@style/Theme.AppCompat">

</application>

</manifest>

apply dark theme to whole app.



Apply 'light' theme to 1st activity -

<manifest ~>

<activity ~>

activity attribute = "android:theme="@style/Theme.AppCompat.Light" ~>

<activity>

<activity ~>

</manifest>

### Layout Managers =

\* To organize our components, we use specialised viewable obj  $\rightarrow$  L.M.

\* Basic building block for UI is a view obj which is created from the view class & occupies a  $\square$  area on the screen is responsible for drawing & event handling.

\* View is the base class for widgets, which are used to create interactive UI components like btn, textfield, etc.

\* ViewGroup is a subclass of View & provide viewable container.

\* we have different layouts which are subclasses of ViewGroup & a layout defining the visual structure for an (any) UI.

\* Each L.M implements a specific strategy to manage the size & position of its children.

\* Located in res/layout / main-layout.xml.

\* ~~5 types~~ Attributes -

android

android

android

android

android

android

android

android

android

\* 5 types -

1) LinearLayout manages:

\* It is ViewGroup that aligns all children in either vertically / horizontally.

\* Layout (also) can be specified with the android:orientation attribute

\* android.widget.LinearLayout  $\rightarrow$  implements

\* eg  $\rightarrow$  <LinearLayout>

android

android

android

android

android

android

android

android

android

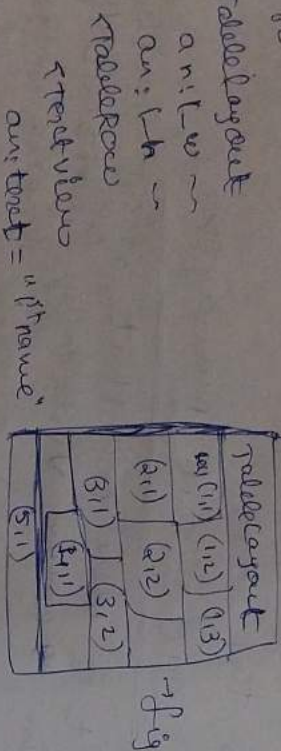
android

</LinearLayout>



## b) Table Layout:

- \* used to arrange the grp of views into rows & columns.
- \* TableLayout contains layout display a border line for their columns, rows / cells.
- \* `<TableLayout>` → to build a row in table.
- \* Each row has 0 or more cells.



defines the column position of a child view within a grid-based layout

```

</TableLayout>
<TableRow>
<TextView>
<EditText>
</TableLayout>
    
```

## c) Relative Layout:

- \* It is a ViewGroup that displays child views in relative positions.
- \* The position of each view can be specified as relative to sibling elements

eg → `<RelativeLayout>`

```

ani:lw ~
ani:lh ~
ani:pl = "16dp"
ani:pr = "16dp" >
<EditText>
    ani:id ~
    ani:lw ~
    ani:lh ~
    ani:hint = "remainder" / >
</RelativeLayout>
    
```

## d) Frame Layout:

- \* mainly used to display a single item.
- \* used to dynamically display a single view, but it can be populated with many items.
- \* setting it to visible while others are invisible.
- \* child views are drawn in a stack, with most recently added child on top.

eg → `<FrameLayout>`

```

ani:id ~
ani:lw ~
ani:lh ~
ani:foreground = "@color/white" >
    
```



<ImageView

an: L\_h ~

an: L\_w ~

an: src = "@drawable/..."

</FrameLayout>

o) Grid Layout:

- \* A Layout that places its children in a grid.
- \* A grid is composed of a set of infinitely thin lines that separate the viewing area into cells.
- \* A view in a grid can occupy 1 or more cells.
- \* The grid consists of rows & columns.
- \* eg → <GridLayout

an: L\_w ~

an: L\_h ~

an: rowCount = "4"

an: rowCount = "3" >

<TextView

an: L\_row = "5"

an: L\_column = "3"

!

<Button

an: L\_rowspan = "2"

</GridLayout>