

ANDROID

CH - 2

Android User Interface Design

Prepared By :

Ms. Kakadiya Jainam A.

POINTS OF LEARN :


- **3.1** UI Overview.
- **3.2** Designing Layouts.

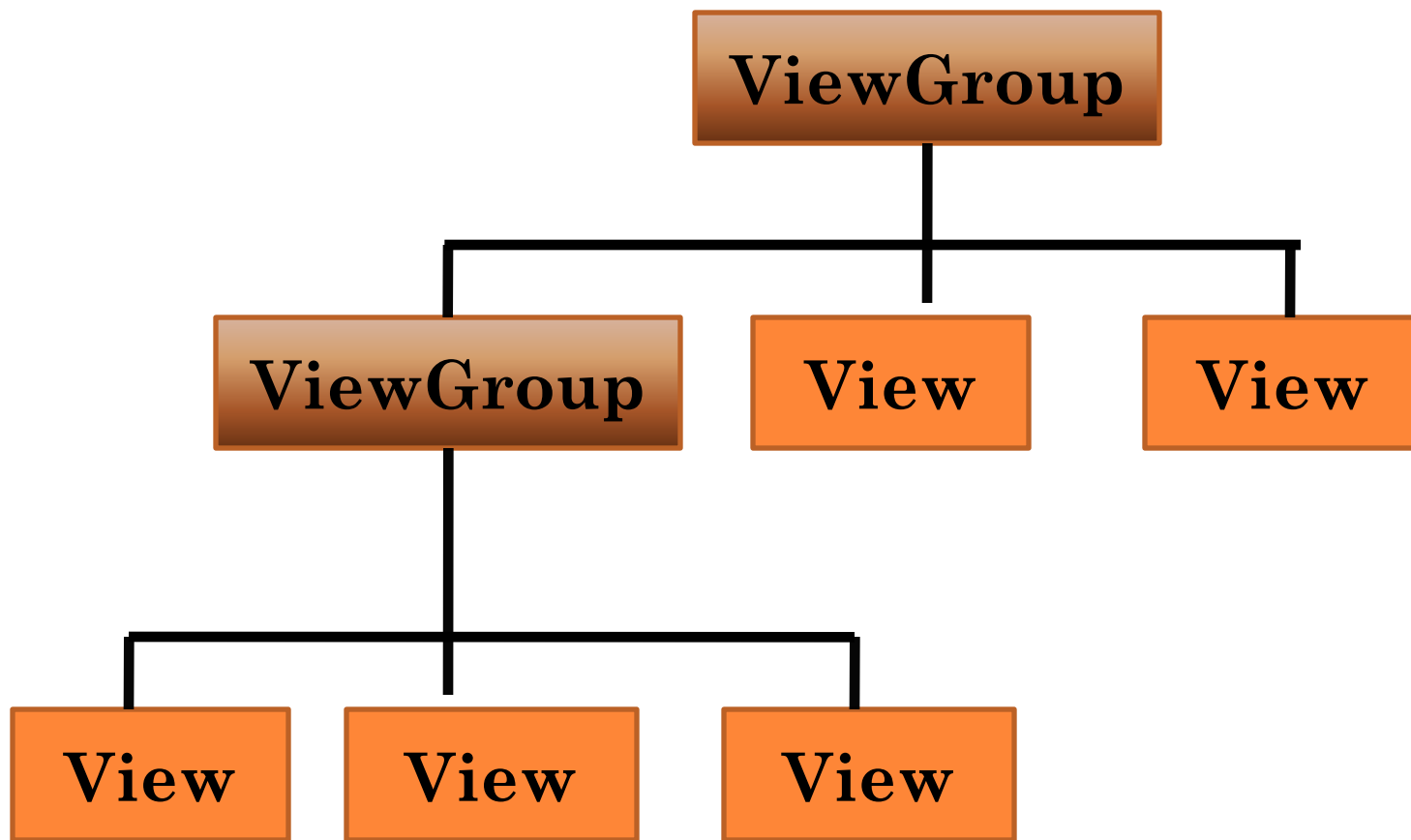


UI OVERVIEW

- All interface elements in an Android app are built using View and ViewGroup objects.
- A view is an object that draws something on the screen that the user can interact with other.
- A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.
- Android provides a collection of both View and ViewGroup subclass that offer you common input controls (such as buttons and text fields) and various layout modules.

USER INTERFACE LAYOUT

- The user elements in an Android app are built using View and ViewGroup objects.
 - Each view group is an invisible container that organizes child view while child view may be input controls or other widgets that draw some part of UI.
 - This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).
- 



- To declare your layout, you can instantiate View objects in code and start building a tree, but the easiest and most effective way to define your layout is with an XML file. XML offers a human-readable structure for the layout, similar to HTML.

USER INTERFACE COMPONENTS

- You don't have to build all of your UI using View and ViewGroup objects. Android provides several app components that offer a standard UI layout for which you simply need to define the content.
- These UI components each have a unique set of APIs that are described in their respective documents, such as Action Bar, Dialogs, and Status Notifications.



- **List Of Basic UIs** : There are number of UI controls provides by Android that allowed you to build the graphical user interface for your app.
 - **1) TextView** : This control is used to display text to the user.
 - **2) EditText** : EditText is a predefined subclass of TextView that includes rich editing capabilities.
 - **3) AutoCompleteTextView** : The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.

- **4) Button** : A-push-button that can be pressed, or clicked by the user to perform an action.
- **5) ImageButton** : AbsoluteLayout enables you to specify the exact location of its children.
- **6) CheckBox** : An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not manually exclusive.
- **7) ToggleButton** : An on/off button with light indicator.



- **8) RadioButton** : The RadioButton has two states : either checked or unchecked.
- **9) RadioGroup** : The RadioGroup is used to group together one or more RadioButtons.
- **10) ProgressBar** : The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
- **11) Spinner** : A drop-down list that allows users to select one value from a set.



- **12) TimerPicker** : The TimerPicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
- **13) DatePicker** : The DatePicker view enables users to select a date of the day.

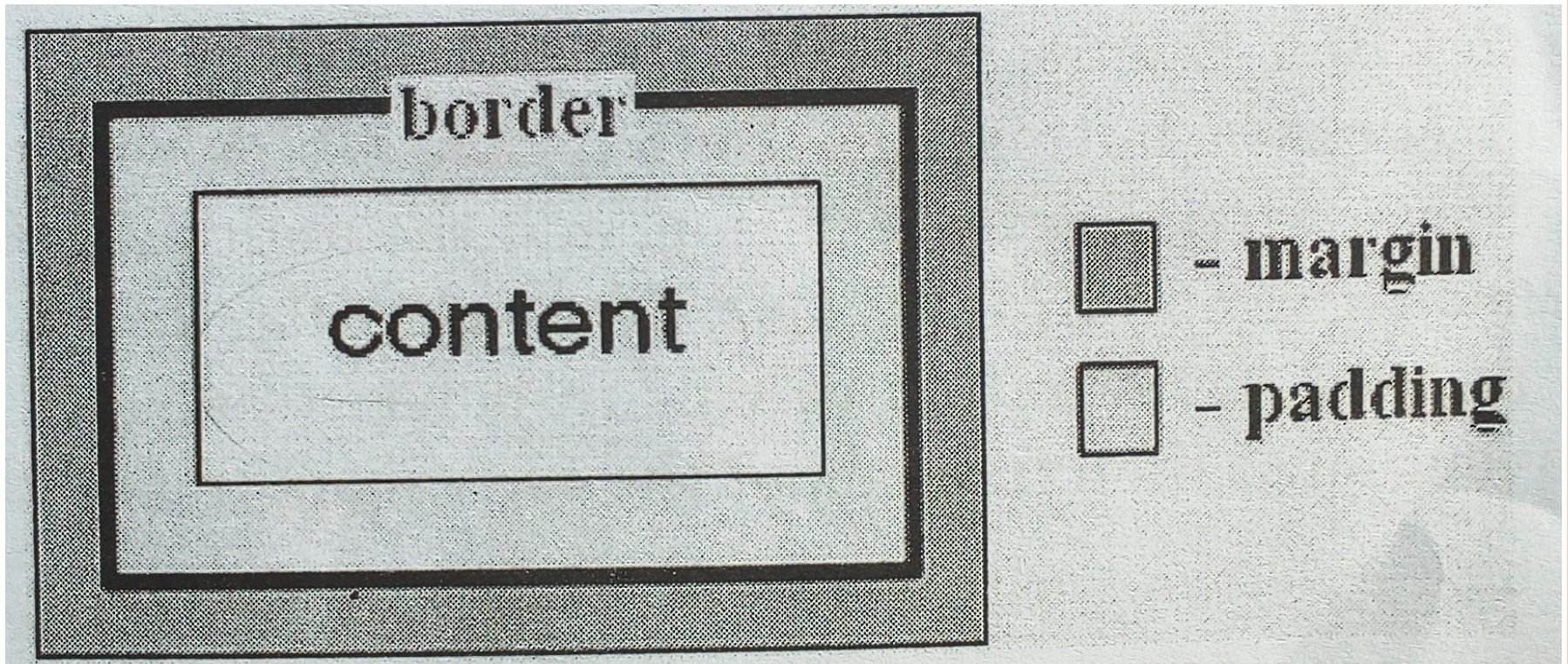


SIZE, PADDING AND MARGIN

- The **Size** of a view is expressed with a width and a height. A view actually possesses two pairs of width and height values.
- The **first pairs** is known as measured width and measured height. These dimensions define how big a view wants to be within its parent. The measured dimensions can be obtained by calling `getMeasuredWidth()` and `getMeasuredHeight()`.
- The **second pairs** is simply known as width and height, can be obtained by calling `getWidth()` and `getHeight()`.

- The **Padding** is expressed in pixels for the left, right and bottom parts of the view. Padding can be used to offset the content of the view by a specific amount of pixels.
- Padding can be set using the `setPadding(int,int,int,int)` method and required by calling `getPaddingLeft()` , `getPaddingTop()` , `getPaddingRight()`, and `getPaddingBottom()`.
- Padding and margin works almost same to leave space. But, difference is location of space consider the following figure and different between this two :

- **Margins** are the space outside the border, between the border and the other elements next to this view.
- Margin can be set using the `setMargins(int left,int top,int right,int bottom)` method.



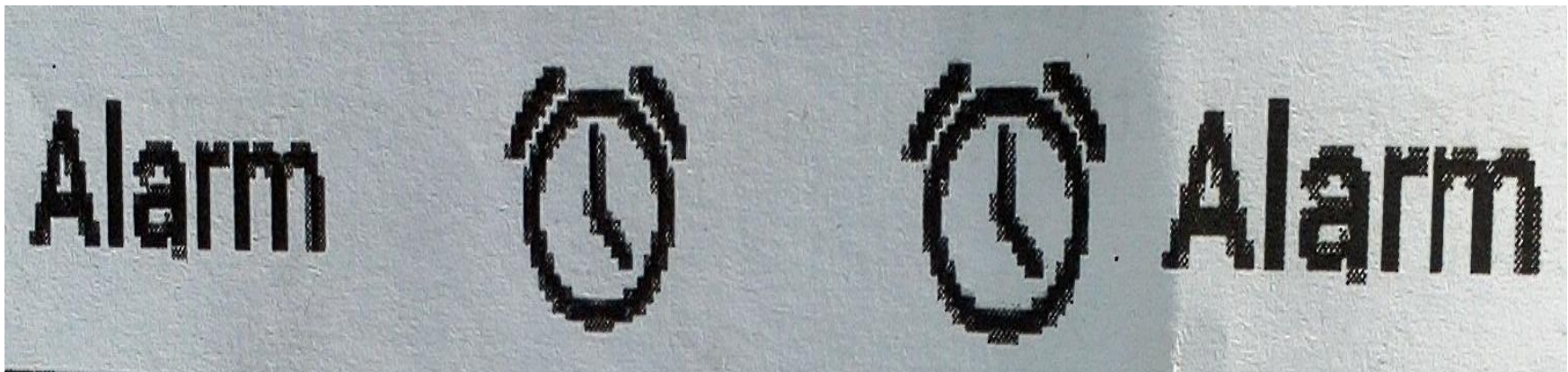
LET'S CHECK SOME OF CONTROL IN DETAIL

- **1) Buttons** : A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

[1]

[2]

[3]



- Depending on whether you want a button with text, an icon, or both, you can create the button your location in **three ways** :

- **1) with text, using the Button class :**

```
<Button
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text" />
```

- **2) with icon, using ImageButton class :**

```
<ImageButton
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon" />
```



- **3) with text and icon, using Button class with the android:drawableLeft attribute:**

```
<ImageButton
```

```
    android:layout_width= "wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="@string/button_text"
```

```
    android:src="@drawable/button_icon" />
```



○ **Responding to Click Events** : when the user click on Button, the button object receives an on-Click event.

- **For example here's a layout with a button using android:onClick :**

```
<xml version="1.0" encoding="utf-8" ?>
```

```
<Button xmlns:android=http://schemas.android.com  
/apk/res/android android:id="@+id/button_send"  
android:layout_width="wrap_content"  
andriod:layout_height="wrap_content"  
android:text="@string/button_send"  
android:onclick="sendMessage" />
```



- **Using an OnClickListener** : you can also declare the click event handler programmatically rather than in an XML layout.
- This might be necessary if you instantiate the Button at runtime.
- To declare the event handler programmatically, create an View.
- OnClickListener object and assign it to the button by calling `setOnClickListener (view.OnClickListener)`.

- **For example :**

```
Button button=(Button) findViewById  
    (R.id.button_send);
```

```
Button.setOnClickListener(new  
    view.OnClickListener()
```

```
{
```

```
    public void onClick(view v)
```

```
{
```

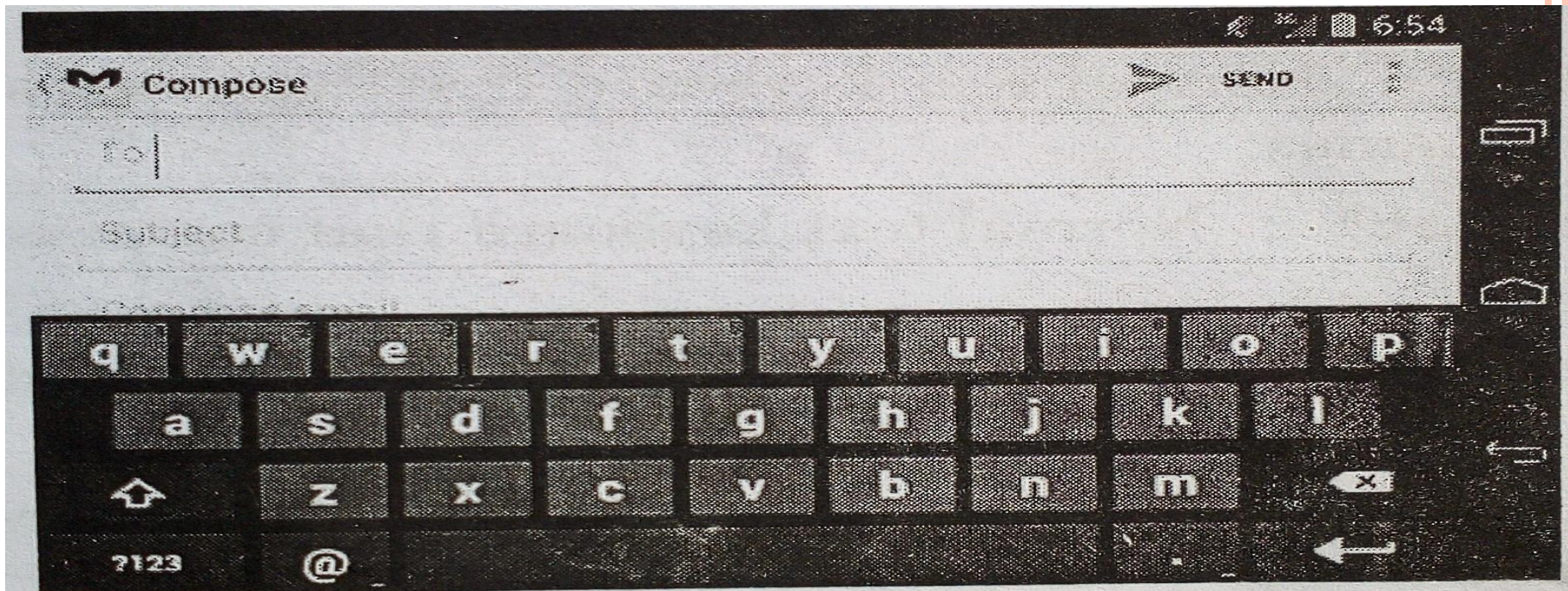
```
    //do something in response to button click
```

```
}
```

```
});
```



- **2) Text Fields :** A Text Field allows the user to type text into your app. Touching a text field place the cursor and automatically display the keyboard.
- You can add text field to your layout with the EditText object. You should usually do so in your XML layout with a <EditText> element.



- Textfield can have different input types, such as number, date, password, or email address. The type determines what kinds of characters are allowed inside the field, and many prompt the virtual keyboard to optimize its layout for frequently used characters.

- **For example :**

```
<EditText android:id="@+id/email_address"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/email_hint"  
    android:inputType="textEmailAddress" />
```




- **3) Auto-complete Suggestions** : If you want to provide suggestions to users as they type, you can use a subclass of EditText called `AutoCompleteTextView`.
- To implement auto-complete, you must specify an (`@link android.widget.Adapter`) that provides the text suggestions.
- There are several kinds of adapters available, depending on where the data is coming from, such as from a database or an array.



- The following procedure describes how to set up an `AutoCompleteTextView` that provides suggestions from an array, using `ArrayAdapter`.
- **[1] Add the `AutoCompleteTextView` to your layout. Here's a layout with only the text field :**

```
<?xml version="1.0" encoding="utf-8" ?>  
<AutoCompleteTextView xmlns:android="http://  
schemas.android.com/apk/res/android"  
android:id="@+id/autocomplete_country"  
android:layout_width="fill_parent"  
android:layout_height="wrap_content" />
```



- [2] Define the array that contains all text suggestions for example, here's an array of country names that's defined in an XML resource file :

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<resources>
```

```
    <string-array name="cityname">
```

```
        <item>Rajkot</item>
```

```
        <item>Amreli</item>
```

```
        <item>Surat</item>
```

```
    </string-array>
```

```
</resources>
```



- **[3] In your Activity or Fragment, use the following code to specify the adapter that supplies the suggestions:**

```
Public class MainActivity extends Activity
{ @Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    autoCompleteTextView actv=
    (AutoCompleteTextView) findViewById
    (id.autoCompleteTextView1);
```



```
string[] cities=getResources().getStringArray  
(R.array.cityname);  
ArrayAdapter<String> adpt= new ArrayAdapter  
<string>(getApplicationContext(), android.R.layout.  
Simple_list_item_1,cities);  
actv.setAdapter(adpt);  
}  
}
```



- **4) Checkboxes** : If Checkbox allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.
- To create each checkbox option, create a Checkbox in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.
- **Responding to Click Events** : When the user selects a checkbox, the Checkbox object receives an on-click event.

- To define the click event handler for a checkbox, add the `android:onClick` attribute to the `CheckBox` element in your XML layout.

- **For example of CheckBox :**

<CheckBox

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

`android:layout_width="fill_parent"`

`android:layout_height="fill_parent"`

`android:text="Pizza"`


`android:onClick="onCheckboxClicked" />`



- Within the Activity that holds this layout, the following method handles the click events checkboxes :

```
public void onCheckboxClicked(View view)
{ booleanchecked=((checkBox) view).isChecked();
  String yo=""; // to store list of your order
  // Check which checkbox was clicked
  switch(view.getId())
  { case R.id.pizza:
      if (checked)
          yo=yo+"pizza";
      break; } }
```



- **5) Radio Buttons** : Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are manually exclusive if you think that the user needs to see all available options side-by-side.
 - To create each radio option, create a `RadioButton` in your layout.
 - **Responding to Click Events** : When the user selects one of the radio buttons, the corresponding `RadioButton` object receives an `on-clicked` event.
- 

○ Example of Radio button :

```
<RadioButton
```

```
    android:id="@+id/r1"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_width="wrap_content"
```

```
    android:text="Radio 1"
```

```
    android:onClick="onRadioButtonClicked">
```

Following method handles the click event for radiobutton :

```
public void onRadioButtonClicked(View view)
```

```
{ boolean checked=((RadioButton)view).isChecked();
```

Following method handles the click event for radiobutton :

```
public void onRadioButtonClicked(View view)
{
    boolean checked=((RadioButton)view).isChecked();
    String yc= "";
    // ck=hecked which radio button was clicked
    switch(view.getId())
    {
        case R.id.r1:
            if(checked)
                yc="Radio 1"; break;
    } }
```



- **6) Toggle Buttons** : A toggle buttons allow the user to change a setting between two states.
- You can add a basic toggle button to your layout with the `ToggleButton` object. Android 4.0 introduces another kinds of toggle button kind of toggle button called a switch that provides a slider control, which you can add with `Switch` object.
- **Responding to Click Event** : When the user a `ToggleButton` and `Switch`, the object receives an on-click event. To define the click event handler, add the `android:onClick` attribute to the `<ToggleButton>` in your XML layout.

- **For example, here's ToggleButton with the android:onClick attribute :**

```
<ToggleButton  
    android:id="@+id/tb"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:textOn="Vibrate on"  
    android:textOff="Vibrate off"  
    android:onClick="onToggleClicked">
```



- **Within the Activity that hosts this layout, the following method handles the click event:**

```
Public void onToggleClicked(View view)
{ //is the toggle on?
Boolean on=((ToggleButton)view).isChecked();

If(on)
{ //enable vibrate
}
Else
{ //disable vibrate
} }
```



- **7) Spinner** : A Spinner provides a quick way to select one value from a set. In the default state, a spinner shows its currently selected value.
- Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.
- You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a `<Spinner>` element.



- **For example :**

<Spinner


```
android:id="@+id/planet_spinner"  
android:layout_height="wrap_content"  
android:layout_width="fill_parent" />
```

- **You can provide them with a string array defined in a string resource file :**

```
<resources>  
  <string-array name="" city>  
    <item> Rajkot</item>  
    <item>Amrelli</item>  
  </string-array> </resources>
```



BUILDING LAYOUTS WITH AN ADAPTER

- When the content for your layout is dynamic or not pre-determined, you can use a layout that subclass AdapterView to populate the layout with views at runtime.
 - The Adapter behaves as a middle-man between the data source and the AdapterView layout the Adapter retrieves the data and converts each entry into a view that can be added into the AdapterView layout.
 - **Main two types of layouts :**
 - 1) List View
 - 2) Grid View
- 

FILLING AN ADAPTER VIEW WITH DATA

- You can populate an AdapterView such as ListView or GridView by binding the AdapterView instance to an Adapter, which retrieves data from an external source and create a view that response each data entry.
- Android provides several subclass of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView.
- **The two most common adapter are :**
 - 1) ArrayAdapter**
 - 2) SimpleCursorAdapter**

○ 1) **ArrayAdapter** :

- Use this adapter when you data source in array. By default, ArrayAdapter create view for each array item by calling toString() on each item and placing the contents in a TextView.
- For example, if you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array



○ Example:

- `Array adapter=new ArrayAdapter<String>(this, android.R.layout.simple_list_item1, myStringArray);`

```
ListView listview=(ListView) findViewById  
(R.id.listview);
```

```
listview.setAdapter(adapter);
```




○ 2) SimpleCursorAdapter :

- Use this adapter when you data comes from a Cursor. When using SimpleCursorAdapter, you must specify a layout to use for each row in the cursor and which columns in the cursor should be intrested into which views of the layout.
- For example, if you want to create a list of people's names and phone number, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers.




○ Example :

```
string[] columns=  
{contactsContract.Data.DISPLAY_NAME,contacts  
Contract.commonDataKinds.phone.NUMBER};  
int[] toViews=  
{R.id.display_name,R.id.phone_number};  
SimpleCursorAdapter adapter=new  
SimpleCursorAdapter(this,  
R.layout_person_name_and_number, cursor,  
fromColumns, toViews, 0);  
ListView listview=getListView();  
listview.setAdapter(adapter);
```



DESIGNING LAYOUTS

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget, You can declare a layout in two ways :
 - **Declare UI elements in XML** : Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widget and layouts.
 - **Instantiate layout elements at runtime** : Your application can create View and ViewGroup objects (and manipulate their properties) programmatically. 

- The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your Application's UI.

- **Android supports the following ViewGroups:**

- LinearLayout
- AbsoluteLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView



○ **LinearLayout :**

- The LinearLayout arranges views in a single column or single row. Child views can either be arranged vertically or horizontally.
- To see how LinearLayout works, let's modify the main.xml file in the project:
- ```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/
res/android" >
```

```
<TextView android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="@string/hello" />
```

```
<Button android:layout_width="100px"
 android:layout_height="wrap_content"
 android:text="Button" />
```

```
</LinearLayout>
```



## ○ AbsoluteLayout

- The AbsoluteLayout lets you specify the exact location of its children.
- Consider the following UI defined in main.xml:
- ```
<?xml version="1.0" encoding="utf-8"?>  
<AbsoluteLayout  
  android:layout_width="fill_parent"  
  android:layout_height="fill_parent"  
  xmlns:android="http://schemas.android.com/apk  
/res/android" >
```



```
<Button android:layout_width="188px"  
android:layout_height="wrap_content"  
android:text="Button" android:layout_x="126px"  
android:layout_y="361px" />
```

```
<Button android:layout_width="113px"  
android:layout_height="wrap_content"  
android:text="Button" android:layout_x="12px"  
android:layout_y="361px" />
```

```
</AbsoluteLayout>
```



- the two Button views located at their specified positions using the `android_layout_x` and `android_layout_y` attributes.



○ **TableLayout**

- The TableLayout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views.
- Each view you place within a row forms a cell. The width for each column is determined by the largest width of each cell in that column.
- Populate main.xml with the following elements :

- ```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk
/res/android" android:layout_height="fill_parent"
android:layout_width="fill_parent"
android:background="#000044"> <TableRow>
<TextView android:text="User Name:"
android:width="120px" /> <EditText
android:id="@+id/txtUserName"
android:width="200px" /> </TableRow>
<TableRow> <TextView
android:text="Password:" /> <EditText
android:id="@+id/txtPassword"
android:password="true"/>
```

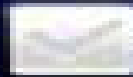


```
</TableRow> <TableRow> <TextView />
 <CheckBox
 android:id="@+id/chkRememberPassword"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Remember Password" />
</TableRow> <TableRow> <Button
 android:id="@+id/buttonSignIn"
 android:text="Log In" /> </TableRow>
</TableLayout>
```



User Name:

Password:



Remember Password

Log In



## ○ RelativeLayout :

- The RelativeLayout lets you specify how child views are positioned relative to each other. Consider the following main.xml file:

- ```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:id="@+id/RLLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Comments"
```



```
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true" /> <EditText
android:id="@+id/txtComments"
android:layout_width="fill_parent"
android:layout_alignLeft="@+id/lblComments"
android:layout_centerHorizontal="true" /> <Button
android:id="@+id/btnSave"
android:layout_height="wrap_content"
android:text="Save"
android:layout_alignRight="@+id/txtComments" />
<Button android:id="@+id/btnCancel"
android:layout_height="wrap_content"
android:text="Cancel"
android:layout_below="@+id/txtComments"
android:layout_alignLeft="@+id/txtComments" />
</RelativeLayout>
```



Comments

Cancel

Save




○ **FrameLayout**

- The FrameLayout is a placeholder on screen that you can use to display a single view.
- Views that you add to a FrameLayout is always anchored to the top left of the layout.

○ **Consider the following content in main.xml:**

```
<?xml version="1.0" encoding="utf-8"?>  
<AbsoluteLayout android:id="@+id/widget68"  
  android:layout_width="fill_parent"  
  android:layout_height="fill_parent"  
  xmlns:android="http://schemas.android.com/apk/res/android" >
```



```
<FrameLayout android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:layout_x="40px" android:layout_y="35px" >  
<ImageView android:src = "@drawable/androidlogo"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content" />  
</FrameLayout> </AbsoluteLayout>
```

Note: This example assumes that the res/drawable folder has an image named androidlogo.png.



○ **ScrollView :**

- A ScrollView is a special type of FrameLayout in that it allows users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.
- The following main.xml content shows a ScrollView containing a LinearLayout, which in turn contains some Button and EditText views:
- ```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView android:id="@+id/widget54"
 android:layout_width="fill_parent"
```



```
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res
/android" > <LinearLayout
android:layout_width="310px"
android:layout_height="wrap_content"
android:orientation="vertical" > <Button
android:id="@+id/button1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 1" /> <Button
android:id="@+id/button2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 2" />
```



```
<Button android:id="@+id/button3"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 3" /> <EditText
 android:id="@+id/txt"
 android:layout_width="fill_parent"
 android:layout_height="300px" /> <Button
 android:id="@+id/button4"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 4" /> <Button
 android:id="@+id/button5"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 5" /> </LinearLayout>
</ScrollView>
```



```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
```

```
xmlns:android=http://schemas.android.com/apk/res/
```

```
android android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
```

```
android:orientation="vertical">
```

```
<TextView android:id="@+id/text"
```

```
android:layout_width="wrap_content">
```







# WORKING WITH CANVAS

- When you're writing an application in which you would like to perform specialized drawing and control the animation of graphics, you should do so by drawing through a Canvas.
- A canvas works for you as a interface, to the actual surface upon which your graphics will be draw it holds all of your draw calls.

- **You can set up a new Canvas like this :**

Bitmap

```
b=Bitmap.createBitmap(100,100,Bitmap.config.ARGB
B_8888); Canvas c=new Canvas(b);
```

# WORKING WITH ANIMATION

- Android provides a variety of powerful APIs for applying animation UI elements:
  - **Animation : The** Android framework provides two animations: property animation and view animation.
  - In addition to these two systems, you can utilize Drawable animation, which allows you to load drawable resources and display them one after another.



- **1) Property Animation:** Your The property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well.
- **2) View Animation:** View Animation is older system and can only be used for Views. It is relatively easy to setup and offers enough capabilities to meet many application's needs.
- **3) Drawable Animation :** Drawable animation involves displaying Drawable resources one after another, like a roll of film.

- This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.
- **4) Property Animation** : The property animation system is a robust framework that allows you to animate almost anything.
- Property animation changes a property's value over a specified length of time.

