2 Android User Interface Design

Topics Covered

- User interface screen elements
  Button, EditText, TextView, DatePicker, TimePicker, ProgressBar, ListView, GridView, RadioGroup, ImageButton, Fragement
- Designing user interfaces with Layouts
  Relative Layout, Linear Layout, Table Layout, Grid Layout etc
- Dialogs
- Drawing and working with animation
  - Frame By Frame Animation
  - Twined Animation
    - Fade In
    - Fade Out
    - Cross Fading
    - Blink
    - Zoom In
    - Zoom Out
    - Rotate
    - Move
    - Slide Up
    - Slide Down
    - Bounce
    - Sequential
    - Together

UI OVERVIEW:

- All user 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 it.
- 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 subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

User Interface Layout:

- The user interface for each component of your app is defined using a hierarchy of View and ViewGroup objects.
- Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI.

- This hierarchy tree can be as simple or complex as you need it.
- 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.
- However you can also design your own layout XML file with Graphical layout option and drag and drop controls on it.
- The name of an XML element for a view is respective to the Android class it represents.
- So a <TextView> element creates a TextView widget in your UI, and a <LinearLayout> element creates a LinearLayout view group.

User Interface Components

- Android provides several app components that offer a standard UI layout for which you simply need to define the contents.
- 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:**

S.N.	UI Control & Description
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 ImageButton enables you to put image on a push button.
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 mutually exclusive.
7.	ToggleButton An on/off button with a light indicator.
8.	RadioButton The RadioButton has two states: either checked or unchecked.
9.	RadioGroup The RadioGroup is used to group 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.	TimePicker The TimePicker 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

Designing Layouts

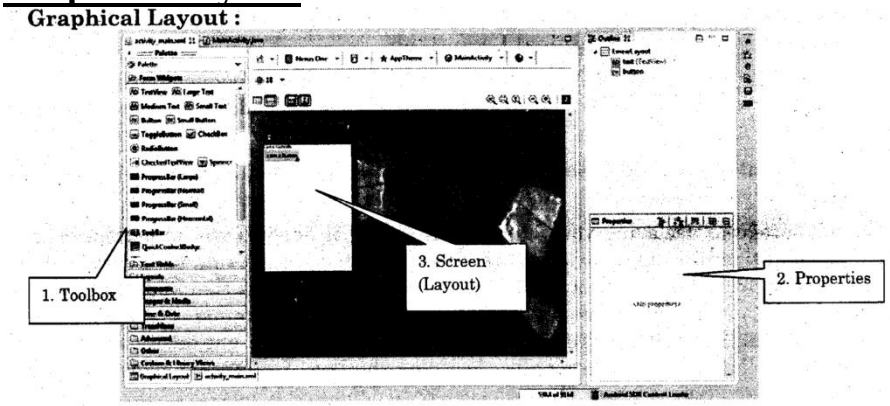
- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.
- You can declare a layout in two ways:

Declare UI elements in XML. Android provides a straightforward XML language that corresponds to the View classes and subclasses, which are use for widgets and layouts.

Instantiate layout elements at runtime. Your application can create View and ViewGroup objects programmatically.

- The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI.
- For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties.
- You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

Graphical Layout:



1. **Toolbox:** In Eclipse you will find Tool box from you can drag and drop your controls on screen.
2. **Properties:** Properties window is used to set different attribute of Control like, (name, color, height, width, etc.)
3. **Screen:** Finally place where you can put your controls and user interacts with it.

Write the XML:

- Using Android's XML file or graphical part, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML- with a series of nested elements.

Load the XML Resources

- When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation.

```
Public void on Create(Bundle savedlnstanceState)
 {
super.onCreate (savedlnstanceState);
setContentView (R.layout.main_layout);
}
```

Attributes

Id

- Any View object may have an integer ID associated with it, to uniquely identify the View
  android: id="@+id/my_button"
- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file).

Layout Prameters

- Note that every LayoutParam subclass has its own syntax for setting values.
- All ViewGroup include width and height (layout_width and layout_height) view is required to define them. Many LayoutParam also include optional margins and borders.
- You can specify width and height width exact measurements.
- wrap_content tells your view to size itself to the dimensions required by content
- fill_parent (renamed match _parent in API Level 8) tells your view to become as big as its parent view group will allow.
- In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using relative measurements such as density- independent pixel units (dp), wrap_content, or fill_parent, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

Layout Position

- A view has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height.

Size,Padding and Margins

- The size of a view is expressed with a width and a height.
- The first pair is known as measured width and measured height.
- The padding is expressed in pixels for the left, top, 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 queried by calling getPaddingLeft(), getPaddingTop(), getPaddingRight() and getPaddingBottom().
- Margins are the spaces outside the border, between the border and the three elements next to this view.
- Margin can be set using the setMargins(int left, int top, int right, int bottom) method.

User Interface Screen Elements

Button

- A button consists of text or an icon (or both text and an icon) that communicate what action occurs when the user touches it.
- Depending on whether you want a button with text an icon or both, you create the button in your layout in three ways:
    - With text, using the Button class:

```
<Button
android: layout_width=”wrap_content”
android: layout_height=”wrap_content”
android: text=”@string/button_text”
…….>
```

    - With an icon, using the Image Button class

```
<ImageButton
android: layout_width=”wrap_content”
android: layout_height=’ wrap_content”
android: src=”@drawable/button_icon”
….>
```

    - With text and an icon, using the Button class with android: drawableLeft attribute:

```
<Button
android: layout_width=”wrap_content”
android: layout_height=”wrap_content”
 android: text=”@string/button_text”
android: drawableLeft=” @drawable/button_icon”
…../>
```

onClick event of Button

- When the user clicks a button, the Button object receives an onClick event. To define the click event handler for a button, add the android:onClick attribute to<Button> element in your XML layout.

Example

```
<Button
android: id=” @+id/button_send”
android: layout_width=”wrap_content”
android: layout_height=”wrap_content”
 android: text=”@string/button_send”
android: onClick=”sendMessage” />
```

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

```
/** Called when the user touches the button
*/public void sendMessage (View view)
{
```

```
// Do something in response to button click
 }
```

The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must be

- Be public
- Return void
- Define a View as its only parameter (this will be the View that was clicked)

Using OnClickListener

- You can also declare the click event handler programmatically rather than in an XML layout.

```
Button button = (Button) findViewByld(R.id.button_send);
button. setOnClickListener (new View. OnClickListener()
{
public void onClick(View v)
{
//Do something in response to button click
}});
```

EditText

- An EditText allows the user to type text into your app.
- It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard.
- You can add a text field to your layout with the EditText object.
- You should usually do so in your XML layout with a <EditText> element.
- Text fields can have different input types, such as number, date, password, or email address.

```
<EditText
android: id=”@+id/email address”
android: layout_width=”fill_parent”
android: layout_height=’wrap_content”
android: hint=” @string/email_hint”
android: inputype=” textEmailAddress” />
```

Controlling other behaviors (TextFields):

- The android:inputType also allows you to specify certain keyboard behaviors, as whether to capitalize all new words or use features like auto-complete spelling suggestions.
- Here are some of the common input type values that define keyboard behaviors:

“textcapsentences” :: Normal text keyboard that capitalizes the first letter each new sentence.
“textcapWords’ :: Normal text keyboard that capitalizes every word.
“textautoCorrect” :: Normal text keyboard that corrects commonly misspelled words.
“textPassword” :: Normal text keyboard, but the characters entered turns into dots
“textmultiline” :: Normal text keyboard that allow users to input long string of text that include line breaks (carriage returns).

Checkboxes

- Checkboxes allow the user to select one or more options from a set.
- 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 separate and you must register a click listener for each one.

Example:

```
<CheckBox
    android : id= "@+id/pizza"
    android : layout_width="wrap_content"
    android : layout_height="wrap_content"
    android : layout_text="Pizza"
    android : onClick="onCheckboxClicked" />
<CheckBox
    android : id= "@+id/gb"
    android : layout_width="wrap_content"
    android : layout_height="wrap_content"
    android : layout_text="Garlic Bread"
    android : onClick="onCheckboxClicked" />
<CheckBox
    android : id= "@+id/tu"
    android : layout_width="wrap_content"
    android : layout_height="wrap_content"
    android : layout_text="Thums Up"
    android : onClick="onCheckboxClicked" />
```

Example:

```
public void onCheckBoxclicked (View view)
{
    Boolean checked=((CheckBox) view) . isChecked();
    String yo= " ";
    switch (view . getId ())

    {
        case R. id .pizza:
            if (checked)
                yo= yo + "pizza";
            break;
        case R. id .gb:
            if (checked)
                yo= yo + "Garlic Bread";
            break;
        case R. id .tu:
            if (checked)
                yo= yo + "Thums Up";
            break;
    }
}
```

RadioGroup:

- Allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.

Responding to Click Event:

- When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event.
- To define the click event handler for a RadioButton, add the android:onClick attribute to the <RadioButton> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event.

Example:

```
<RadioGroup
    xmlns : android=http: //schemas. android. com/apk/res/android
    android : layout_width=”fill_parent”
    android : layout_height=”wrap_content”
    android : orientation=”vertical” >
    <RadioButton
         android : id= “@+id/pen8”
        android : layout_width=”wrap_content”
        android : layout_height=”wrap_content”
        android : text=”8 GB Pendrive”
        android : onClick=”onRadioButtonClicked” />
    <RadioButton
        android : id= “@+id/pen16”
        android : layout_width=”wrap_content”
        android : layout_height=”wrap_content”
        android : text=”16 GB Pendrive”
        android : onClick=”onRadioButtonClicked” />
</RadioGroup>
```

Toggle Buttons:

- A toggle button allows 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 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a Switch object.
- The Toggle Button and Switch controls are subclasses of CompoundButton and function in the same manner, so you can implement .their behavior the same way.

Responding to Click Events:

- When the user selects a ToggleButton and Switch, the object receives an on-click. Event.

```
public void onToggleClicked(View view)
{
    // Is the toggle on?
    boolean on = ((ToggleButton) view) . isChecked();
```

```
        if  (on)
        {
                // Enable vibrate
        }
        else
        {
                // Disable vibrate
        }
}
```

Spinner:

- The choices you provide for the spinner can come from any source, but must be provided through a SpinnerAapter, such as an ArrayAdapter if the choices are available in an array or a CursorAdapter if the choices are available from a database query.
- For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<resources>
<string-array name="cityname">
        < item >Rajkot</item>
        < item >Gondal</ item>
        < item >Virpur</item>
        < item >Jetpur</ item>
        < item >Dhoraji</ item>
</string-array>
</resources>
```

DatePicker

- Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface.
- For this functionality android provides DatePicker and DatePickerDialog components.
- To show DatePickerDialog , you have to pass the DatePickerDialog id to showDialog(id_of_dialog) method. Its syntax is given below
    - showDialog(999);
- On calling this showDialog method, another method onCreateDialog() gets automatically called. So we have to override that method too. Its syntax is given below

```
@Override
protected Dialog onCreateDialog(int id) {
    // TODO Auto-generated method stub
    if (id == 999) {
        return new DatePickerDialog(this, myDateListener, year,
month, day);
    }
    return null;
}
```

- In the last step, you have to register the DatePickerDialog listener and override its onDateSet method. This onDateSet method contains the updated day, month and year. Its syntax is given below.

```
private DatePickerDialog.OnDateSetListener myDateListener = new
DatePickerDialog.OnDateSetListener() {
   @Override
   public void onDateSet(DatePicker arg0, int arg1, int arg2, int
arg3) {
      // arg1 = year
      // arg2 = month
      // arg3 = day
   }
};
```

TimePicker

- Android TimePicker allows you to select the time of day in either 24 hour or AM/PM mode.
- The time consists of hours, minutes and clock format.
- Android provides this functionality through TimePicker class.
- In order to use TimePicker class, you have to first define the TimePicker component in your activity.xml. It is define as below

```
<TimePicker
   android:id="@+id/timePicker1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
```

- After that you have to create an object of TimePicker class and get a reference of the above defined xml component. Its syntax is given below.

```
import android.widget.TimePicker;
private TimePicker timePicker1;
timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
```

In order to get the time selected by the user on the screen, you will use getCurrentHour() and getCurrentMinute() method of the TimePicker Class. Their syntax is given below.

```
int hour = timePicker1.getCurrentHour();
int min = timePicker1.getCurrentMinute();
```

ProgressBar

- Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.
- In android there is a class called ProgressDialog that allows you to create progress bar. In order to do this, you need to instantiate an object of this class. Its syntax is.

  ProgressDialog progress = new ProgressDialog(this);

- Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :) ");
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.setIndeterminate(true);
```

ListView

- Android ListView is a view which groups several items and display them in vertical scrollable list.
- The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.
- An adapter actually bridges between UI components and the data source that fill data into UI Component.
- Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.
- The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.
- Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adaptersare ArrayAdapter,Basedapter, CursorAdapter, SimpleCursorAdapter,SpinnerAdapter and WrapperListAdapter.
- Once you have array adapter created, then simply call setAdapter() on your ListView object as follows

```
ListView listView = (ListView)
findViewById(R.id.listview);
listView.setAdapter(adapter);
```

GridView

- Android GridView shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a ListAdapter.
- An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.
- The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Fragment

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.
- You create fragments by extending Fragment class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element.

- So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time.

How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the Fragment class. The Fragment class has different callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Types of Fragments

Basically fragments are divided as three stages as shown below.

- Single frame fragments − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
- List fragments − fragments having special list view is called as list fragment
- Fragments transaction − Using with fragment transaction. We can move one fragment to another fragment.

Layouts

- It describes how to display different types of data in the application.

Linear Layout

- A layout that organizes its children into a single horizontal or vertical row.
- It creates a scrollbar if the length of the window exceeds the length of the screen.
- All children of a LinearLayout are arranged one after the other.

Layout Weight

→ **Equally weighted children:**

- To create a linear layout in which, each child use the same amount of space on the screen, set the aridroid:layoutheight of each view to “0dp” (for a vertical layout) or the android:layoutwidth of each view to “0dp” (for a horizontal layout).
- Then set the android:layout_weight of each view to “1”.
- LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute.
- This attribute assigns a “value” to a view in terms of how much space is should occupy on the screen.
- Default weight is zero.

Relative Layout

- Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
- A RelativeLayout is a very powerful utility for designing a user interface

➢ **Positioning Views:**

- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID).
- So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.
- By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from RelativeLayout.LayoutParams.
- Some of the many layout properties available to views in a RelativeLayout include:

→ **android: layout_alignParentTop**
If "true", makes the top edge of this view match the top edge of the parent.

→ **android: layout centerVertical**
If "true", centers this child vertically within its parent.

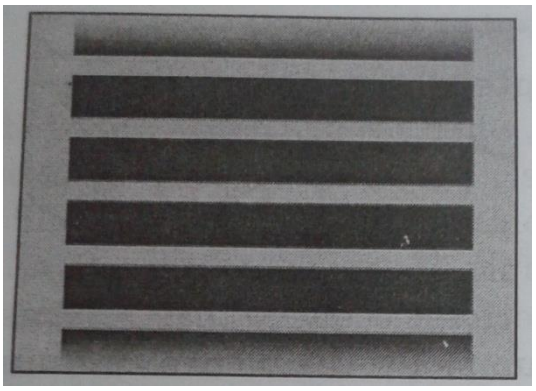
→ **android: layout_below**
Positions the top edge of this view below the view specified with a resource ID.

→ **android:layout_toRightOf**
Positions the left edge of this view to the right of the view specified with a resource ID.

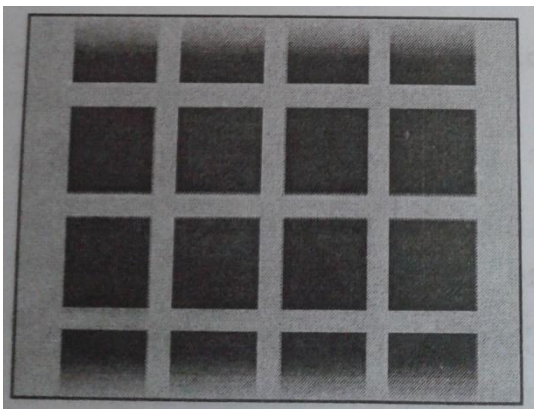
Building Layouts with an Adapter:

- When the content for your layout is dynamic or not pre-determined, you can see layout that subclasses AdapterView to populate the layout with views at runtime.
- A subclass of the AdapterView class uses an Adapter to bind data to its layout.
- Common layouts backed by an adapter include:

❖ **List View:**



❖ **Grid View:**



➢ **ArrayAdapter:**

- Use this adapter when your data source is an array.
- By default., ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView.

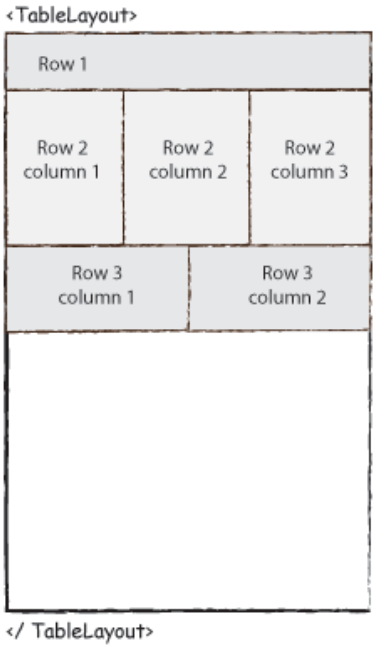
➢ **SimpleCursorAdapter :**

- Use this adapter when your data comes from a Database.
- When using SimpleCursorAdapter, you must specify a layout to use for each row in the Cursor and which columns in the Cursor should be inserted into which views the layout.

TableLayout

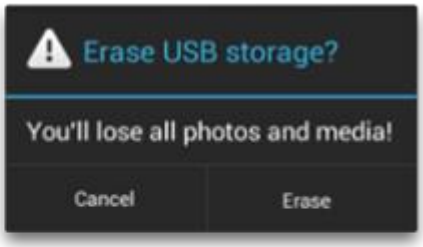
- Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

- TableLayout containers do not display border lines for their rows, columns, or cells.



Dialogs

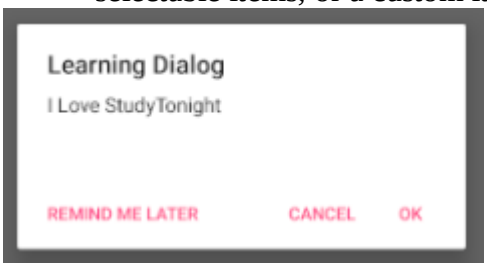
- A dialog is a small window that prompts the user to make a decision or enter additional information.
- A dialog does not fill the screen and is normally used for events that require users to take an action before they can proceed.



- In android, you can create following types of Dialogs:
  - Alert Dialog
  - DatePicker Dialog
  - TimePicker Dialog
  - Custom Dialog

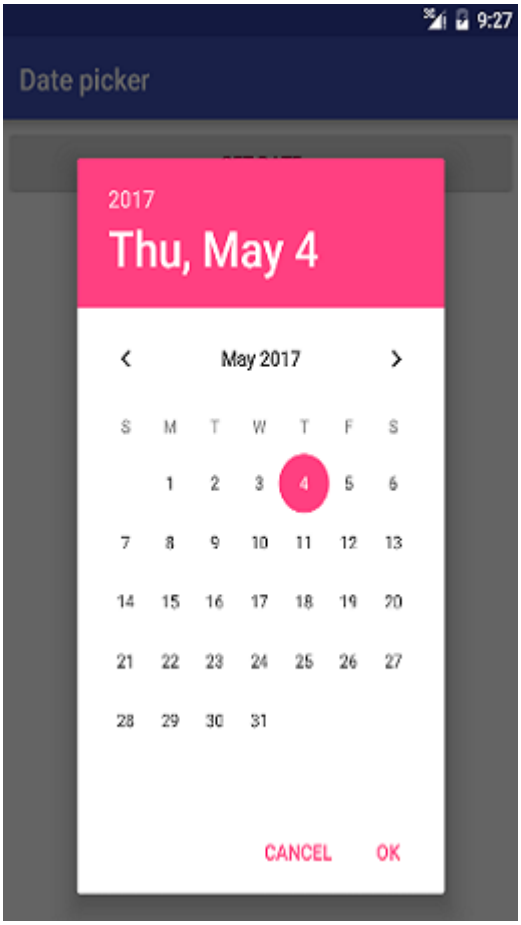
Alert Dialog

- This Dialog is used to show a title, buttons(maximum 3 buttons allowed), a list of selectable items, or a custom layout.



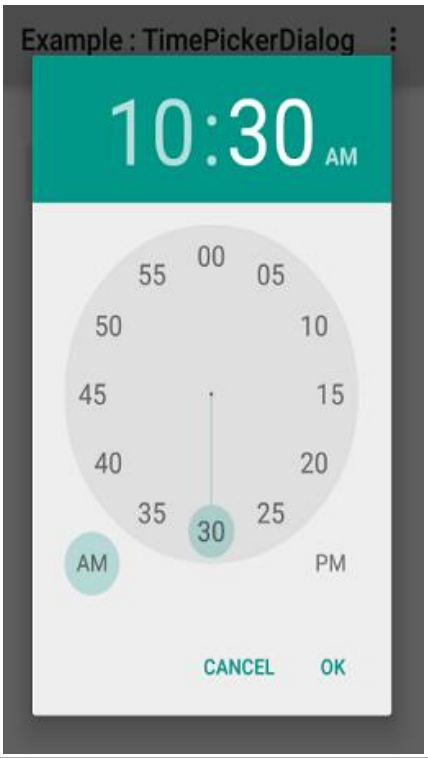
DatePicker Dialog

- This dialog provides us with a pre-defined UI that allows the user to select a date.



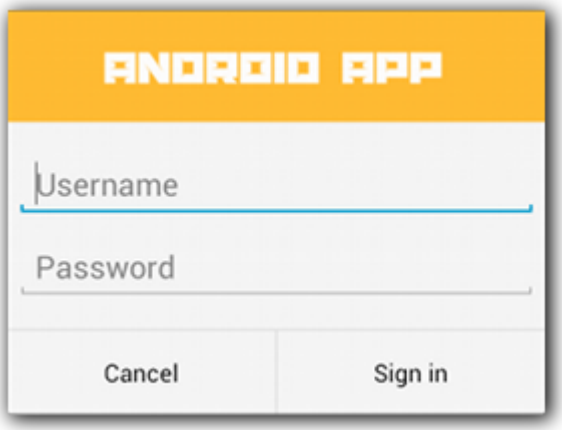
TimePicker Dialog

- This dialog provides us with a pre-defined UI that allows the user to select suitable time.



Custom Dialog

- You can create your own custom dialog with custom characteristics.



Working With Animation:

- Animation is the process of adding a motion effect to any view, image, or text.
- With the help of an animation, you can add motion or can change the shape of a specific view.
- Animation in Android is generally used to give your UI a rich look and feel.
- The animations are basically of three types as follows:

1. **Property Animation**
2. **View Animation**
3. **Drawable Animation**

1. Property Animation

- Property Animation is one of the robust frameworks which allows animating almost everything.
- This is one of the powerful and flexible animations which was introduced in Android 3.0. Property animation can be used to add any animation in the CheckBox, RadioButtons, and widgets other than any view.

2. View Animation

- View Animation can be used to add animation to a specific view to perform tweened animation on views.
- Tweened animation calculates animation information such as size, rotation, start point, and endpoint.
- These animations are slower and less flexible.
- An example of View animation can be used if we want to expand a specific layout in that place we can use View Animation.
- The example of View Animation can be seen in Expandable RecyclerView.

3. Drawable Animation

- Drawable Animation is used if you want to animate one image over another. The simple way to understand is to animate drawable is to load the series of drawable one after another to create an animation.

- A simple example of drawable animation can be seen in many apps Splash screen on apps logo animation.

Important Methods of Animation

Methods	Description
startAnimation()	This method will start the animation.
clearAnimation()	This method will clear the animation running on a specific view.

Frame by Frame animation

- An object used to create frame-by-frame animations, defined by a series of Drawable objects, which can be used as a View object's background.
- The simplest way to create a frame-by-frame animation is to define the animation in an XML file, placed in the res/drawable/ folder, and set it as the background to a View object. Then, call start() to run the animation.
- An AnimationDrawable defined in XML consists of a single <animation-list> element and a series of nested <item> tags. Each item defines a frame of the animation.
- spin_animation.xml file in res/drawable/ folder:

```
<!-- Animation frames are wheel0.png through wheel5.png
     files inside the res/drawable/ folder -->
<animation-list android:id="@+id/selected" android:oneshot="false">
    <item android:drawable="@drawable/wheel0" android:duration="50" />
    <item android:drawable="@drawable/wheel1" android:duration="50" />
    <item android:drawable="@drawable/wheel2" android:duration="50" />
    <item android:drawable="@drawable/wheel3" android:duration="50" />
    <item android:drawable="@drawable/wheel4" android:duration="50" />
    <item android:drawable="@drawable/wheel5" android:duration="50" />
</animation-list>
```

- Here is the code to load and play this animation.

```
// Load the ImageView that will host the animation and
// set its background to our AnimationDrawable XML resource.
ImageView img = (ImageView)findViewById(R.id.spinning_wheel_image);
img.setBackgroundResource(R.drawable.spin_animation);

// Get the background, which has been compiled to an AnimationDrawable object.
AnimationDrawable frameAnimation = (AnimationDrawable) img.getBackground();

// Start the animation (looped playback by default).
frameAnimation.start();
```

Tween Animation

- Animation is the process of creating motion and shape change.
- Tween Animation is defined as an animation which is used to Translate, Rotate, Scale and Alpha any type of view in Android.
- All the Tween Animations are coded in Android xml file which are placed together in folder name “anim” under “res” folder in Project directory.
- Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle etc and perform the required animation on that object.
- It can be applied to any type of object. So in order to use this , android has provided us a class called Animation.
- In order to perform animation in android , we are going to call a static function loadAnimation() of the class AnimationUtils. We are going to receive the result in an instance of Animation Object. Its syntax is as follows −

```
Animation animation =
AnimationUtils.loadAnimation(getApplicationContext(),
   R.anim.myanimation);
```

- Note the second parameter. It is the name of the our animation xml file. You have to create a new folder called anim under res directory and make an xml file under anim folder.
- This animation class has many useful functions which are listed below −

Sr.No	Method & Description
1	**start()** This method starts the animation.
2	**setDuration(long duration)** This method sets the duration of an animation.
3	**getDuration()** This method gets the duration which is set by above method
4	**end()** This method ends the animation.
5	**cancel()** This method cancels the animation.

- In order to apply this animation to an object , we will just call the startAnimation() method of the object. Its syntax is –

```
ImageView image1 = (ImageView)findViewById(R.id.imageView1);
image.startAnimation(animation);
```

Fade In Animation

fade_in.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"

android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0" />

</set>
```

- Here **alpha** references the opacity of an object. An object with lower alpha values is more transparent, while an object with higher alpha values is less transparent, more opaque. Fade in animation is nothing but increasing alpha value from 0 to 1.

Fade Out Animation

fade_out.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
        android:fromAlpha="1.0"

android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="0.0" />

</set>
```

- Fade out android animation is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0.

Cross Fading Animation

- Cross fading is performing fade in animation on one TextView while other TextView is fading out. This can be done by using fade_in.xml and fade_out.xml on the two TextViews.

Blink Animation

blink.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android">
    <alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"

android:interpolator="@android:anim/accelerate_interpolator"
        android:duration="600"
        android:repeatMode="reverse"
        android:repeatCount="infinite"/>
</set>
```

- Here fade in and fade out are performed infinitely in reverse mode each time.

Zoom In Animation

zoom_in.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <scale

xmlns:android="https://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="3"
        android:toYScale="3" >
    </scale>

</set>
```

- We use pivotX="50%" and pivotY="50%" to perform zoom from the center of the element.

Zoom Out Animation

zoom_out.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <scale

xmlns:android="https://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="0.5"
        android:toYScale="0.5" >
    </scale>

</set>
```

- Notice that **android:from** and **android:to** are opposite in zoom_in.xml and zoom_out.xml.

Rotate Animation

rotate.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android">
    <rotate android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="600"
        android:repeatMode="restart"
        android:repeatCount="infinite"

android:interpolator="@android:anim/cycle_interpolator"/>

</set>
```

- A **from/toDegrees** tag is used here to specify the degrees and a cyclic interpolator is used.

Move Animation

move.xml

```
<set

xmlns:android="https://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">

   <translate
        android:fromXDelta="0%p"
        android:toXDelta="75%p"
        android:duration="800" />
</set>
```

Slide Up Animation

slide_up.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <scale
        android:duration="500"
        android:fromXScale="1.0"
```

```
        android:fromYScale="1.0"

android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="0.0" />

</set>
```

- It's achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"** inside the **scale** tag.

Slide Down Animation

slide_down.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true">

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />

</set>
```

- This is just the opposite of slide_up.xml.

Bounce Animation

bounce.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />

</set>
```

- Here bounce interpolator is used to complete the animation in bouncing fashion.

Sequential Animation

sequential.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >


    <!-- Move -->
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="300"
        android:toXDelta="75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="1100"
        android:toYDelta="70%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="1900"
        android:toXDelta="-75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="2700"
        android:toYDelta="-70%p" />

    <!-- Rotate 360 degrees -->
    <rotate
        android:duration="1000"
        android:fromDegrees="0"

android:interpolator="@android:anim/cycle_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="3800"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:toDegrees="360" />
```

```
</set>
```

- Here a different **android:startOffset** is used from the transitions to keep them sequential.

Together Animation

together.xml

```
<set
xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >


    <!-- Move -->
    <scale

xmlns:android="https://schemas.android.com/apk/res/android"
        android:duration="4000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="4"
        android:toYScale="4" >
    </scale>

    <!-- Rotate 180 degrees -->
    <rotate
        android:duration="500"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:toDegrees="360" />

</set>
```

- Here android:startOffset is removed to let them happen simultaneously.