UNIT: 3 ANDROID USER INTERFACE DESIGN

CS-31 Mobile Application Development in Android using Kotlin

- Android architecture
- User interface screen elements
- Button, edittext, textview, datepicker, timepicker, progressbar, listview, gridview, radiogroup, imagebutton
- Designing user interfaces with layouts
- Relative layout, linear layout, tablelayout
- Dialogs
- Drawing and working with animation
- Frame by frame animation
- Twined animation

-: Assignment 3 :-

1. Explain list view in brief
2. Write steps to create alert dialog with ok and cancel buttons.
3. Differentiate between linear layout and relative layout.
4. What is toggle button.
5. List types of layouts and explain any one in detail.
6. Explain grid view with its attributes.
7. Explain linear layout with its attributes.
8. Explain dialog in android with a suitable example.
9. Explain frame by frame animation with steps and suitable example.
10. Explain tween animation with steps and suitable example.
11. Explain androidmanifist.xml
12. Explain Relative layout with its attributes
13. Text View *v/s* Edit Text
14. Image View v/s Image Button
15. Develop and android app to pass data to another activity
16. _ file is used to store string resource.
17. _ view is similar to dropdown list
18. _ widget is used to display image in android.
19. _allows the user to type text into your app.
20. _Animation by performing a senes of transformation on a single image with an animation.
21. When the user selects a checkbox, the Check Box object receives an ____ event.
22. What is listview ?

PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI

Android Architecture

Android architecture contains different number of components to support any android device needs. Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are following:-

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

Applications

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.

Application framework

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation. It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

Application runtime

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries. Like Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM) is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

Platform libraries

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

- Media library provides support to play and record an audio and video formats.
- Surface manager responsible for managing access to the display subsystem.
- SGL and OpenGL both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- SQLite provides database support and FreeType provides font support.
- Web-Kit This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- SSL (Secure Sockets Layer) is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.
The features of Linux kernel are:

- Security: The Linux kernel handles the security between the application and the system.
- Memory Management: It efficiently handles the memory management thereby providing the freedom to develop our apps.
- Process Management: It manages the process well, allocates resources to processes whenever they need them.
- Network Stack: It effectively handles the network communication.
- Driver Model: It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

User Interface Screen Element

- Android UI Controls are those components of Android that are used to design the UI in a more interactive way. It helps us to develop an application that makes user interaction better with the view components. Android provides us a huge range of UI controls of many types such as buttons, text views, etc.
- As we know UI is the only thing that a user interacts with within an Application. This is the reason that we make our Application look aesthetic and, more and more connective. To do so, we need to add the UI controls or we say Input controls in the respective application.

Moving forth we will see some important Android UI controls for our applications:

1. **TextView**
2. **EditText**
3. **Button**
4. **ImageButton**
5. **ToggleButton**
6. **RadioButton**
7. **RadioGroup**
8. **CheckBox**
9. AutoCompleteTextView
10. ProgressBar
11. Spinner
12. TimePicker
13. DatePicker
14. SeekBar
15. AlertDialog
16. Switch
17. RatingBar

1. TextView

- A TextView displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.
- **XML file:**

```
<TextView
        //attributes to describe it
/>
```

Sr. No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the control.
2	**android:capitalize** If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
5	**android:fontFamily** Font family (named by string) for the text.
6	**android:gravity** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
7	**android:hint** Hint text to display when the text is empty.
8	**android:inputType** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
13	**android:password** Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
14	**android:phoneNumber** If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
15	**android:text** Text to display.
16	**android:textAllCaps** Present the text in ALL CAPS. Possible value either "true" or "false".
17	**android:textColor** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
18	**android:textColorHighlight** Color of the text selection highlight.
19	**android:textColorHint** Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
20	**android:textIsSelectable** Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
21	**android:textSize** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).

22	**android:textStyle** Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'.

2. EditText

- A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.**XML file:**

<EditText
//attributes
>

Sr. No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

3. Button

- A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.XML file:

<Button
//attributes
/>

Sr. No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.
6	**android:background** This is a drawable to use as the background.
7	**android:contentDescription** This defines text that briefly describes content of the view.
8	**android:id** This supplies an identifier name for this view.
9	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.

10	**android:visibility** This controls the initial visibility of the view.

4. ImageButton

- An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.**XML file:**

<ImageButton
//other attributes...
android:src= "@drawable/add_icon"/>

Sr.No	Attribute & Description
1	**android:adjustViewBounds** Set this to true if you want the ImageView to adjust its bounds to preserve the aspect ratio of its drawable.
2	**android:baseline** This is the offset of the baseline within this view.
3	**android:baselineAlignBottom** If true, the image view will be baseline aligned with based on its bottom edge.
4	**android:cropToPadding** If true, the image will be cropped to fit within its padding.
5	**android:src** This sets a drawable as the content of this ImageView.

5. ToggleButton

- The toggle button displays the ON/OFF states of a button with a light indicator. A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.

XML file:

<ToggleButton
//attributes
android:checked="true"
android:textOff="OFF"
android:textOn="ON"/>

Sr.No.	Attribute & Description
1	**android:disabledAlpha** This is the alpha to apply to the indicator when disabled.
2	**android:textOff** This is the text for the button when it is not checked.
3	**android:textOn** This is the text for the button when it is checked.
4	**android:background** This is a drawable to use as the background.
5	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
6	**android:drawableBottom** This is the drawable to be drawn below the text.
7	**android:drawableRight** This is the drawable to be drawn to the right of the text.
8	**android:editable** If set, specifies that this TextView has an input method.

9	**android:text** This is the Text to display.

6. RadioButton

- Radio button in Android is the one that has only two possible states, that are either checked or unchecked. Initially, it is in the unchecked state, once it's checked it can't be unchecked. A RadioButton has two states: either checked or unchecked.This allows the user to select one option from a set.

XML file:

```
<RadioButton
android:text="Radio button"
android:checked="true"/>
```

7. RadioGroup

- It's a group of Radio buttons that are alike. In this, only one of all the buttons can be chosen. A RadioGroup class is used for set of radio buttons.
- If we check one radio button that belongs to a radio group, it automatically unchecks any previously checked radio button within the same group.

XML file:

```
<RadioGroup android:orientation="vertical">
  <RadioButton android:text="Radio Button 1"/>
  <RadioButton android:text="Radio Button 2"/>
  <RadioButton android:text="Radio Button 3"/>
</RadioGroup>
```

Attribute	Description
android:checkedButton	This is the id of child radio button that should be checked by default within this radio group.

8. CheckBox

- A CheckBox is the UI control that has two states that are either checked or unchecked. If we have a group of CheckBox, we can select as many as we want, unlike RadioGroup.
- A CheckBox is an on/off switch that can be toggled by the user. You should use check-boxes when presenting users with a group of selectable options that are not mutually exclusive.

XML file:

```
<CheckBox
android:checked="true"
android:text="CheckBox"
// other attributes
/>
```

9. ProgressBar

- In Android, we have a progress bar that shows the progress of some action that is happening like pasting a file to some location. A progress bar can be in two modes:
- **Determinate Mode:**In this, the progress is shown with the percent of action completed. Also, the time to be taken is already determined.
- **Indeterminate Mode:**In this, there is no idea of when the task would be completed, therefore, it functions continuously.

XML file:

```
<ProgressBar
// attributes, here we define the speed, layout, id, etc.
/>
```

- 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.

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

Sr. No	Title & description
1	**getMax()** This method returns the maximum value of the progress.
2	**incrementProgressBy(int diff)** This method increments the progress bar by the difference of value passed as a parameter.
3	**setIndeterminate(boolean indeterminate)** This method sets the progress indicator as determinate or indeterminate.
4	**setMax(int max)** This method sets the maximum value of the progress dialog.
5	**setProgress(int value)** This method is used to update the progress dialog with some specific value.
6	**show(Context context, CharSequence title, CharSequence message)** This is a static method, used to display progress dialog.

The above were some of the very important UI controls, now we will also read about some more UI controls :

1. Spinner

- The Spinner in Android is a User Interface that is used to select a particular option from a list of given options. Spinner in Android is the same as dropdown in HTML. It provides us with a faster way to select an option of our choice. When we click on the down arrow, it shows a list of values from which we can select one. By default, the first value would be shown as the currently selected Value.

2. TimePicker

- Time picker is a UI component that works as an intermediate to select a time of the day. The time chosen by it is shown either in 24 hrs format or in 12hrs format using AM and PM.
- It gives a virtual Clock/watch to select it. This virtual clock makes it easy to choose the time.

3. DatePicker

- Like we have time picker, we have a date picker as UI control too. In this, the System shows a virtual calendar to the users to choose the day.
- This enables the user to choose a particular date using either a calendar or a dropdown. These both are made to make it easier for the user to pick up a date and a time.

4. SeekBar

- In Android, Seekbar is an extended Progress bar. A seekbar comes with a pointer that is draggable throughout the bar either in left or right. This pointer helps to set the progress as well. This helps the user to choose a particular range of values.

5. RatingBar

- A rating bar in Android is an extended version of a seekbar. It is used to give the rating by touching it. In the rating bar, a user can rate at a scale of 5 with a difference of 0.5.
- Its rating is in Stars. The user needs to tap/click the stars.

6. AlertDialog

- Alert Dialog Box is a UI that gives the users an Alert or Warning of something. It appears on the screen in a small window. Once it comes, the user needs to decide or choose an option that it shows.

- For example, when you enter the wrong password for email id.

7. Switch

- In Android, a switch is a two-state UI element that holds either ON or OFF state. ON generally means Yes and OFF means No. By default, a switch is in the OFF state. A user can change its state many times.

8. AutoCompleteTextView

- AutoCompleteTextView is an extension of EditText. In this UI element, the user is provided with a few suggestions of some values/texts. The value can be chosen by the user while filling AutoCompleteTextView.

UI Layouts

- The basic building block for user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets(UI controls), which are used to create interactive UI components like buttons, text fields, etc.
- Types of Layouts
- **LinearLayout**
- **RelativeLayout**
- FrameLayout
- **GridView**
- **ListView**
- **Table Layout**
- Absolute Layout
- ConstraintLayout

LinearLayout:

- This is the most basic layout in android studio, that aligns all the children sequentially either in a horizontal manner or a vertical manner by specifying the android:orientation attribute.
- If one applies android:orientation=”vertical” then elements will be arranged one after another in a vertical manner and If you apply android:orientation=”horizontal” then elements will be arranged one after another in a horizontal manner.

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:baselineAligned** This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	**android:divider** This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
4	**android:gravity** This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
5	**android:orientation** This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
6	**android:weightSum**

	Sum up of child weight

Relative Layout:

- This is very flexible layout used in android for custom layout designing. It gives us the flexibility to position our component/view based on the relative or sibling component's position. Just because it allows us to position the component anywhere we want so it is considered as most flexible layout. For the same reason Relative layout is the most used layout after the Linear Layout in Android. It allow its child view to position relative to each other or relative to the container or another container.
- In Relative Layout, you can use "above, below, left and right" to arrange the component's position in relation to other component. For example, in the below image you can see content is placed in related to Heading.

Sr. No.	Attribute & Description
1	**android:layout_above** Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"
2	**android:layout_alignBottom** Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
3	**android:layout_alignLeft** Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
4	**android:layout_alignParentBottom** If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".
5	**android:layout_alignParentEnd** If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
6	**android:layout_alignParentLeft** If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".
7	**android:layout_alignParentRight** If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
8	**android:layout_alignParentStart** If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
9	**android:layout_alignParentTop** If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".
10	**android:layout_alignRight** Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
11	android:layout_alignStart Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
12	**android:layout_alignTop**

	Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
13	**android:layout_below** Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
14	**android:layout_centerHorizontal** If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
15	**android:layout_centerInParent** If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
16	**android:layout_centerVertical** If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
17	**android:layout_toEndOf** Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
18	**android:layout_toLeftOf** Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
19	**android:layout_toRightOf** Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
20	**android:layout_toStartOf** Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

GridView:

- In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter. Users can then select any grid item by clicking on it.
- GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.
- Adapter Is Used To Fill Data In Gridview: To fill the data in a GridView we simply use adapter and grid items are automatically inserted to a GridView using an Adapter which pulls the content from a source such as an arraylist, array or database.

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:columnWidth** This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	**android:gravity** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:horizontalSpacing** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	**android:numColumns** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.
6	**android:stretchMode** Defines how columns should stretch to fill the available empty space, if any.

7	**android:verticalSpacing** Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

ListView:

- List of scrollable items can be displayed in Android using ListView. It helps you to displaying the data in the form of a scrollable list. Users can then select any list item by clicking on it. ListView is default scrollable so we do not need to use scroll View or anything else with ListView.
- ListView is widely used in android applications. A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:divider** This is drawable or color to draw between list items.
3	**android:dividerHeight** This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	**android:entries** Specifies the reference to an array resource that will populate the ListView.
5	**android:footerDividersEnabled** When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	**android:headerDividersEnabled** When set to false, the ListView will not draw the divider after each header view. The default value is true.

TableLayout:

- In Android, Table Layout is used to arrange the group of views into rows and columns. Table Layout containers do not display a border line for their columns, rows or cells. A Table will have as many columns as the row with the most cells.
- 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.

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:collapseColumns** This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	**android:shrinkColumns** The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	**android:stretchColumns** The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

There are following three concepts related to Android Event Management −

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Common Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc which may be needed for your application. So I recommend to refer official documentation for Android application development in case you are going to develop a sophisticated apps.

Dialogs:

- A Dialog is small window that prompts the user to a decision or enter additional information.
- Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.
- In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog.

Syntax:

```
val alertDialogBuilder = AlertDialog.Builder(this)
```

Sr.No	Method type & description
1	**setIcon(Drawable icon)** This method set the icon of the alert dialog box.
2	**setCancelable(boolean cancel able)** This method sets the property that the dialog can be cancelled or not
3	**setMessage(CharSequence message)** This method sets the message to be displayed in the alert dialog
4	**setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener)** This method sets list of items to be displayed in the dialog as the content. The selected option will be notified by the listener
5	**setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)** This method Sets the callback that will be called if the dialog is cancelled.
6	**setTitle(CharSequence title)** This method set the title to be appear in the dialog

Example:

```
alertDialogBuilder.setMessage("This is Trial")
    alertDialogBuilder.setPositiveButton("Yes"){ _, _ ->
       Toast.makeText(this,"Yes!",Toast.LENGTH_LONG).show()
    }
    alertDialogBuilder.setNegativeButton("No"){ _, _        p[]=\      ->
       Toast.makeText(this,"Nope",Toast.LENGTH_LONG).show()
    }
    alertDialogBuilder.setNeutralButton("Maybe"){ _, _ ->
       Toast.makeText(this,"Maybe",Toast.LENGTH_LONG).show()
    }
    alertDialogBuilder.create()
    alertDialogBuilder.show()
```

Drawing and 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.

Frame By Frame Animation/Animate drawable graphics

- Frame-by-frame animation is where a series of images are shown in succession at quick intervals so that the final effect is that of an object moving or changing.
- In Android, frame-by-frame animation is implemented through the class AnimationDrawable. This class is a Drawable. These objects are commonly used as backgrounds for views. AnimationDrawable,

in addition to being a Drawable, can take a list of other Drawable resources (like images) and render them at specified intervals.

- To use this AnimationDrawable class, start with a set of Drawable resources (for example, a set of images) placed in the /res/drawable subdirectory. You will then construct an XML file that defines the AnimationDrawable using a list of these images. This XML file needs to be placed in the /res/drawable/frame_animation.xml subdirectory as well.

Twined Animation/View Animation

- Tween Animation is one kind of animation that we only need to set the start and the end. It will supply the lack between the start and the end. It is why we called tween here.
- We have four kinds tween animation: **Alpha, Rotate, Translate, Scale**.
- Tween animation could be set in both code and xml.
- By AnimationSet, we could mix these animations together.
- **By AnimationUtils.loadAnimation, we could load these animations defined in xml.**
- By Animation.AnimationListener, we could add listner to listen to changes of animations.