USER INTERFACES

UNIT 2

LAYOUTS

- Linear layout is a simple layout used in android for layout designing. In the Linear layout all the elements are displayed in linear fashion means all the childs/elements of a linear layout are displayed according to its orientation. The value for orientation property can be either horizontal or vertical.
- There are two types of linear layout orientation:
  - Vertical
  - Horizontal
- As the name specified these two orientations are used to arrange there child one after the other, in a line, either vertically or horizontally.

EXAMPLE

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"> <!-- Vertical Orientation set -->

    <!-- Child Views(In this case 2 Button) are here -->
     <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"
        android:id="@+id/button"
        android:background="#358a32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"
        android:id="@+id/button2"
        android:background="#0058b6" />
  </LinearLayout>
```

Main Attributes In Linear Layout:

- **orientation:** The orientation attribute used to set the childs/views horizontally or vertically. In Linear layout default orientation is vertical.
- **gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the layout like left, right, center, top, bottom etc.
- **layout_weight:** The layout_weight attribute specify each child control's relative importance within the parent linear layout.
- **weightSum:** weightSum is the sum up of all the child attributes weight. This attribute is required if we define weight property of the childs.

Relative Layout

- The Relative Layout 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
- In Relative Layout, you can use "above, below, left and right" to arrange the component's position in relation to other component or relative to the container or another container.

Attributes of Relative layout:

1.above: Position the bottom edge of the view above the given anchor view ID and must be a reference of the another resource in the form of id. Example, android:layout_above=”@+id/textView” .

2. alignBottom: alignBottom is used to makes the bottom edge of the view match the bottom edge of the given anchor view ID and it must be a reference to another resource, in the form of id. Example: android:layout_ alignBottom =”@+id/button1”

3. alignLeft: alignLeft is used to make the left edge of the view match the left edge of the given anchor view ID and must be a reference to another resource, in the form of Example: android:layout_ alignLeft =”@+id/button1”.

4. alignRight: alignRight property is used to make 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 like this example: android:layout_alignRight=”@+id/button1”.

5.alignStart: alignStart property is used to 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 of like this example: android:layout_alignStart=”@+id/button1”.

Attributes of Relative layout:

6. alignTop: alignTop property is used to 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 like this example: android:layout_alignTop=”@+id/button1”.

7.alignParentBottom: If alignParentBottom property is true, makes the bottom edge of this view match the bottom edge of the parent. The value of align parent bottom is either true or false. Example: android:layout_alignParentBottom=”true”

8. alignParentEnd: If alignParentEnd property is true, then it makes the end edge of this view match the end edge of the parent. The value of align parent End is either true or false. Example: android:layout_alignParentEnd=”true”.

9. alignParentLeft: If alignParentLeft property is true, makes the left edge of this view match the left edge of the parent. The value of align parent left is either true or false. Example: android:layout_alignParentLeft=”true”.

10. alignParentRight: If alignParentRight property is true, then it makes the right edge of this view match the right edge of the parent. The value of align parent right is either true or false. Example: android:layout_alignParentRight=”true”.

Attributes of Relative layout:

11.alignParentStart: If alignParentStart is true, then it makes the start edge of this view match the start edge of the parent. The value of align parent start is either true or false. Example: android:layout_alignParentStart=”true”.

12.alignParentTop: If alignParentTop is true, then it makes the top edge of this view match the top edge of the parent. The value of align parent Top is either true or false. Example: android:layout_alignParenTop=”true”.

13.centerInParent: If center in parent is true, makes the view in the center of the screen vertically and horizontally. The value of center in parent is either true or false. Example: android:layout_centerInParent=”true”.

14.centerHorizontal: If centerHorizontal property is true, makes the view horizontally center. The value of centerHorizontal is either true or false.Example: android:layout_centerHorizontal=”true”.

15.centerVertical: If centerVertical property is true, make the view vertically center. The value of align parent bottom is either true or false. Example: android:layout_centerVertical=”true”.

Constraint Layout

- Constraint Layout is a ViewGroup (i.e. a view that holds other views) which allows you to create large and complex layouts with a flat view hierarchy, and also allows you to position and size widgets in a very flexible way. It was created to help reduce the nesting of views and also improve the performance of layout files.
- ConstraintLayout is very similar to RelativeLayout in such a way because, views are laid out according to relationships between sibling views and the parent layout yet it’s a lot more flexible and works better with the Layout Editor of the Android Studio’s. It was released at Google I/O 2016. Since it came into existence , it has become a wildly used viewgroup.

Advantages Of Constraint Layout Over Other Layouts

1. One great advantage of the constraintlayout is that you can perform animations on your ConstraintLayout views with very little code.
2. You can build your complete layout with simple drag-and-drop on the Android Studio design editor.
3. You can control what happens to a group of widgets through a single line of code.
4. Constraint Layout improve performance over other layout.

It is not bundled as part of Android SDK and is available as a support library. Due to this, any update in the future would be compatible with all versions of Android.

Difference between Linear And Relative Layout:

RELATIVE LAYOUT:

- Every element of relative layout arranges itself to the other element or a parent element.
- It is helpful while adding views one next to other etc
- In a relative layout you can give each child a Layout Property that specifies exactly where it should go in relative to the parent or relative to other children.
- Views can be layered on top of each other.

LINEAR LAYOUT:

- In a linear layout, like the name suggests, all the elements are displayed in a linear fashion either vertically or horizontally.
- Either Horizontally or Vertically this behavior is set in android:orientation which is an property of the node Linear Layout.
- Linear layouts put every child, one after the other, in a line, either horizontally or vertically.

Table Layout

- 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.
- A table can also leave the cells empty but cells can't span the columns as they can in HTML(Hypertext markup language).

- For building a row in a table we will use the <TableRow> element. Table row objects are the child views of a table layout.
  - Each row of the table has zero or more cells and each cell can hold only one view object like ImageView, TextView or any other view.
  - Total width of a table is defined by its parent container
  - Column can be both stretchable and shrinkable. If shrinkable then the width of column can be shrunk to fit the table into its parent object and if stretchable then it can expand in width to fit any extra space available.

EXAMPLE

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:collapseColumns="0"> <!-- collapse the first column of the table row-->
     <!-- first row of the table layout-->
    <TableRow
        android:id="@+id/row1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- Add elements/columns in the first row-->

    </TableRow>
</TableLayout>
```

Attributes of TableLayout in Android:

1. **id:** id attribute is used to uniquely identify a Table Layout.
2. **stretchColumns:** Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also stretch the columns to take up available free space by using this attribute. The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, 3…n).
3. **shrinkColumns:** Shrink column attribute is used to shrink or reduce the width of the column‘s. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

   If the value is 0 then the first column’s width shrinks or reduces by word wrapping its content.
4. **collapseColumns:** collapse columns attribute is used to collapse or invisible the column’s of a table layout. These columns are the part of the table information but are invisible. If the values is 0 then the first column appears collapsed, i.e it is the part of table but it is invisible.

Frame Layout

- Frame Layout is one of the simplest layout to organize view controls. They are designed to block an area on the screen. Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other.
- We can add multiple children to a FrameLayout and control their position by assigning gravity to each child, using the **android:layout_gravity** attribute.

Attributes of Frame Layout:

- **android:id -** This is the unique id which identifies the layout in the R.java file.
- **android:foreground -** Foreground defines the drawable to draw over the content and this may be a color value. Possible color values can be in the form of “#rgb”, “#argb”, “#rrggbb”, or “#aarrggbb”. This all are different color code model used.
- **android:foregroundGravity -** This defines the gravity to apply to the foreground drawable. Default value of gravity is fill. We can set values in the form of “top”, ”center_vertical” , ”fill_vertical”, ”center_horizontal”, ”fill_horizontal”, ”center”, ”fill”, ”clip_vertical”, ”clip_horizontal”, ”bottom”, ”left” or ”right” .It is used to set the gravity of foreground. We can also set multiple values by using “|”. Ex: fill_horizontal|top .Both the fill_horizontal and top gravity are set to framelayout.
- **android:visibility -** This determine whether to make the view visible, invisible or gone.

  visible – the view is present and also visible

  invisible – The view is present but not visible

  gone – The view is neither present nor visible
- **android:measureAllChildren -** This determines whether to measure all children including gone state visibility or just those which are in the visible or invisible state of measuring visibility. The default value of measureallchildren is false. We can set values in the form of Boolean i.e. “true” OR “false”.
- This may also be a reference to a resource (in the form “@[*package*:]*type*:*name*“) or theme attribute (in the form “?[*package*:][*type*:]*name*“) containing a value of this type.

View is a basic building block of UI (User Interface) in android. A view is a small rectangular box that responds to user inputs. Eg: EditText, Button, CheckBox, etc. ViewGroup is an invisible container of other views (child views) and other ViewGroup.

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.
- **Adapter:** To fill the data in a ListView we simply use adapters. List items are automatically inserted to a list using an Adapter that pulls the content from a source such as an arraylist, array or database.
- **ListView in Android Studio:** Listview is present inside Containers. From there you can drag and drop on virtual mobile screen to create it. Alternatively you can also XML code to create it

Attributes of ListView

1. **id:** id is used to uniquely identify a ListView.
2. **divider:** This is a drawable or color to draw between different list items
3. **dividerHeight:** This specify the height of the divider between list items. This could be in dp(density pixel),sp(scale independent pixel) or px(pixel).
4. **listSelector:** listSelector property is used to set the selector of the listView. It is generally orange or Sky blue color mostly but you can also define your custom color or an image as a list selector as per your design.

An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner etc.

ListView is a subclass of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry.

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 default scrollable so we don’t need to use ScrollView or anything else with GridView.
- GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.

```
<GridView android:id="@+id/simpleGridView"
android:layout_width="fill_parent" android:layout_height="wrap_content"
android:numColumns="3"/>
```

Attributes of GridView:

1.id: id is used to uniquely identify a GridView.

2.numColumns: numColumn define how many columns to show. It may be a integer value, such as “5” or auto_fit

auto_fit is used to display as many columns as possible to fill the available space on the screen.

3. horizontalSpacing: horizontalSpacing property is used to define the default horizontal spacing between columns. This could be in pixel(px),density pixel(dp) or scale independent pixel(sp).

4.verticalSpacing: verticalSpacing property used to define the default vertical spacing between rows. This should be in px, dp or sp.

5.columnWidth: columnWidth property specifies the fixed width of each column. This could be in px, dp or sp.

ADAPTER

GridView and ListView both are subclasses of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry. In android commonly used adapters which fill data in GridView are:

1. Array Adapter
2. Base Adapter
3. Custom Array Adapter

1. Avoid Array Adapter To Fill Data In GridView:

Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.

By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in grid items, please avoid ArrayAdapter and use custom adapters.

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,R.id.textView,StringArray);
```

2. GridView Using Base Adapter In Android:

Base Adapter is a common base class of a general implementation of an Adapter that can be used in GridView. Whenever you need a customized grid view you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying custom grid items. ArrayAdapter is also an implementation of BaseAdapter

3. GridView Using Custom ArrayAdapter In Android Studio:

ArrayAdapter is also an implementation of BaseAdapter so if we want more customization then we create a custom adapter and extend ArrayAdapter in that.

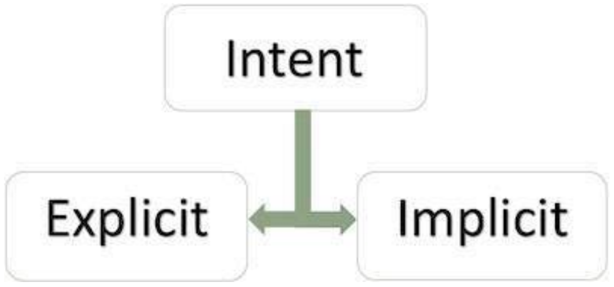
Intent

- An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)** to
- There are separate mechanisms for delivering intents to each type of component – activities, services, and broadcast receivers.communicate with a background Service.

Intents

Sr.No	Method & Description
1	**Context.startActivity()** The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.
2	**Context.startService()** The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	**Context.sendBroadcast()** The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

Types of Intents There are following two types of intents supported by Android



Intent

Explicit Intents

Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.

These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity.

```
Intent i = new Intent(FirstActivity.this, SecondActivity.class);


startActivity(i);
```

Intent

Implicit Intents

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example –

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);

 startActivity(read1);
```

The target component which receives the intent can use the getExtras()method to get the extra data sent by the source component. For example –

```
Bundle extras = getIntent().getExtras();
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Toast

In Android, Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction.
Toast is used when we required to notify user about an operation without expecting any user input. It displays a small popup for message and automatically fades out after timeout.

1. **makeText**(Context context, CharSequence text, int duration): This method is used to initiate the Toast. This method take three parameters First is for the application Context, Second is text message and last one is duration for the Toast. LENGTH_LONG, LENGTH_SHORT are constants of Toast that are used for setting the duration for the Toast.

2. **show**(): This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

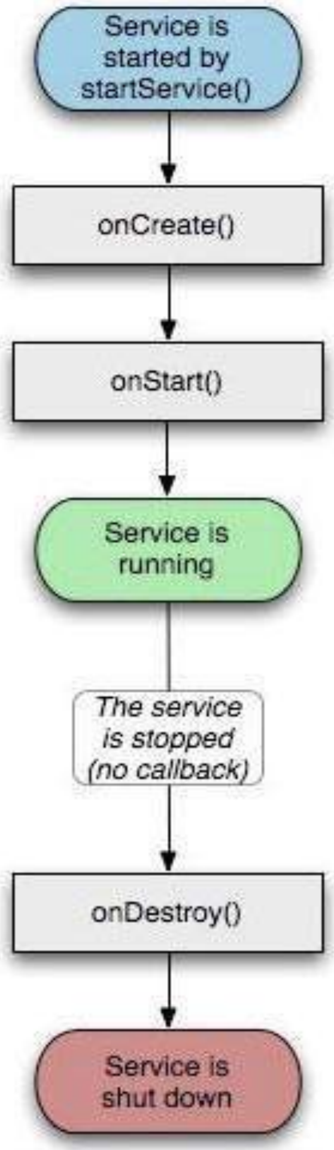
```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast", Toast.LENGTH_LONG);
toast.show();
```

Android - Services

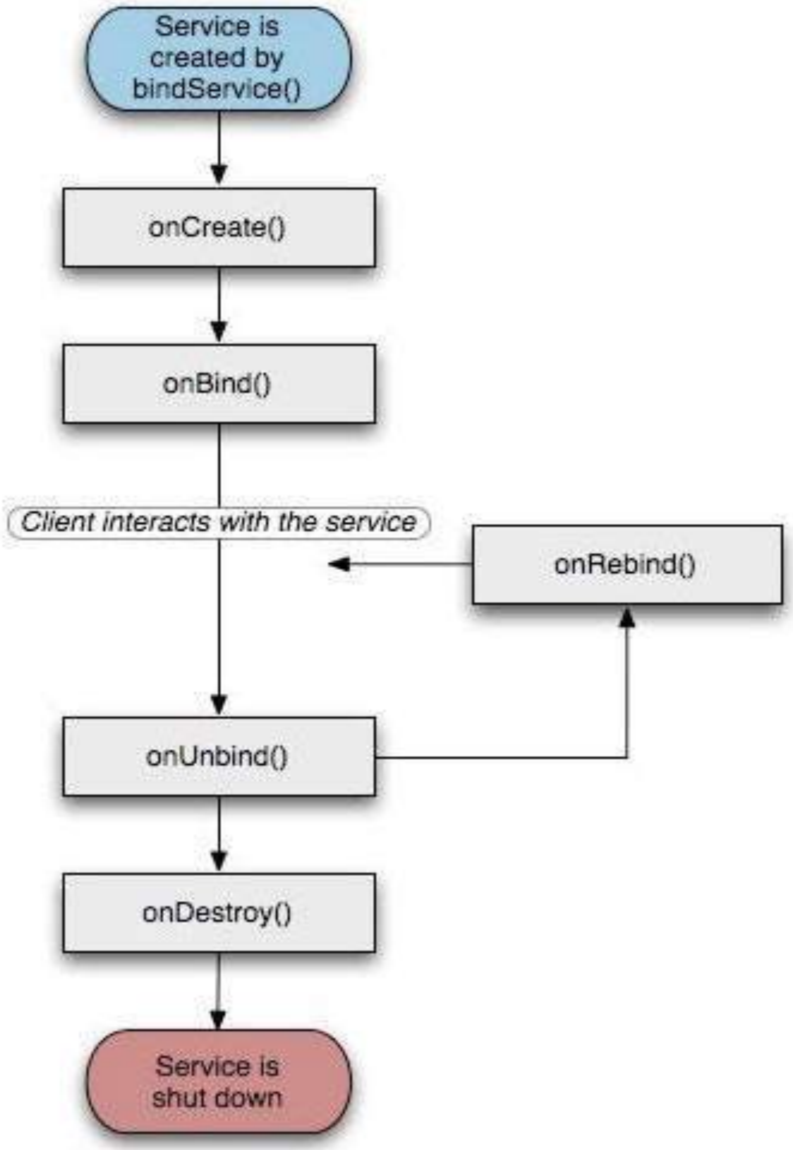
A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

Sr.No.	State & Description
1	**Started** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC)

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage.


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down

UN Bounded Service



Bounded services

Services

To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You can understand each one and implement those that ensure your app behaves the way users expect.

Sr.No.	Callback & Description
1	**onStartCommand()** The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
2	**onBind()** The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
3	**onUnbind()** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind()** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	**onCreate()** The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	**onDestroy()** The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Android - Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver**class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {
 @Override public void onReceive(Context context, Intent intent) {
Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show(); } }
```

Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the *sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

```
public void broadcastIntent(View view) {
 Intent intent = new Intent();
 intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
sendBroadcast(intent); }
```

Using Intent in Android Studio

https://www.youtube.com/watch?v=FYtkui_rn2o&t=16s

https://abhiandroid.com/programming/intent-in-android#gsc.tab=0

Linear Layout :
https://youtu.be/yx3xzuNCJKc?si=VpHIIUyF03x22i-3
Relative Layout : https://youtu.be/gEQCmRq44KU?si=wNK2DQhuw22yvH0k
Table Layout:
https://www.youtube.com/watch?v=2q7R3Pt-NCw
https://www.youtube.com/watch?v=xYhGGIxYZRI
Frame Layout:
https://www.youtube.com/watch?v=_FOORt2FRDQ