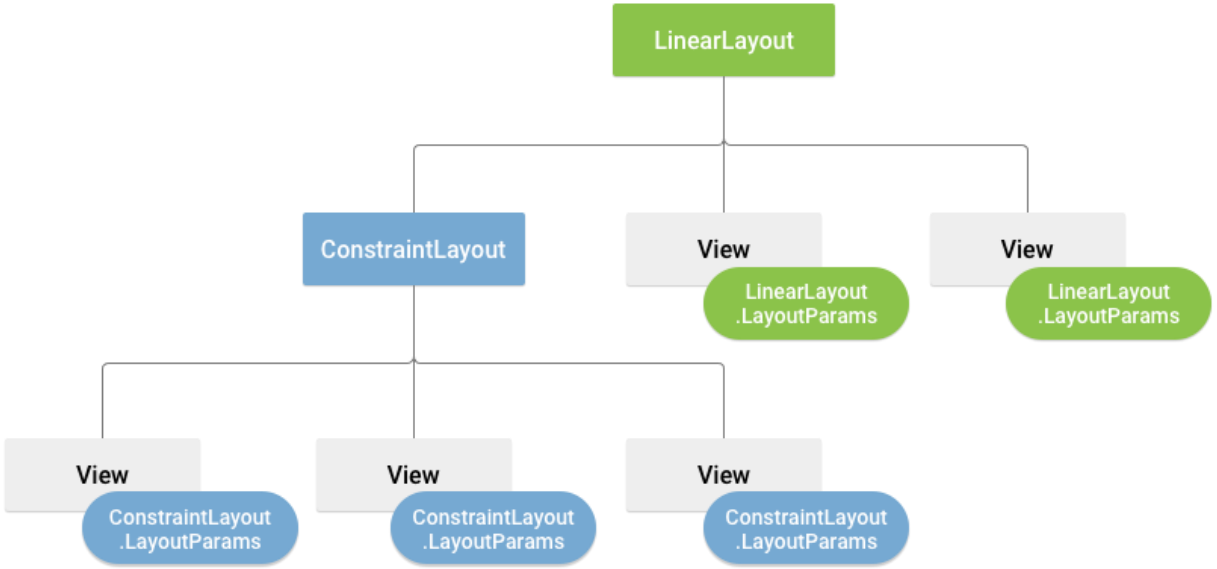
Android UI

- Android provides a variety of **Pre-built UI components**.
- Layout objects and UI controls that allow you to build the graphical user interface for your app.
- Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

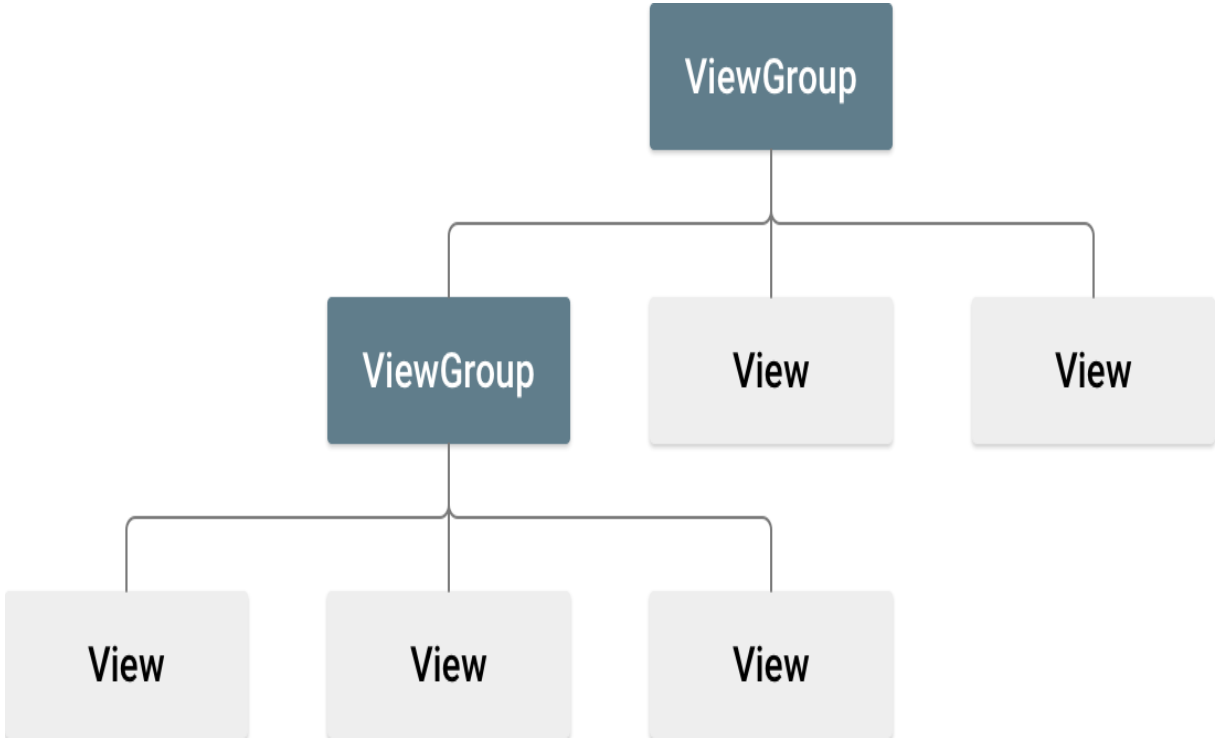
Layouts

- A layout defines the structure for a user interface in your activity.
- All elements in the layout are built using a hierarchy of View and ViewGroup objects.
- A View usually draws something the user can see and interact with.
- **Layout**(View Group) is an invisible container that defines the layout structure for View and other **Layout**(View Group) objects.
- It provide a different layout structure, such as LinearLayout or ConstraintLayout.



View Group

- A ViewGroup is a special view that can contain other views (called children.)
- The view group is the base class for layouts and views containers.
- It's child can be **Views** or **View Group**.




```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

LinearLayout    linearLayout =
findViewById(R.id.linearLayout);

        // Create a Button
        Button button = new Button(this);
        button.setText("Ready for Mid sem. Exam");
```

Views

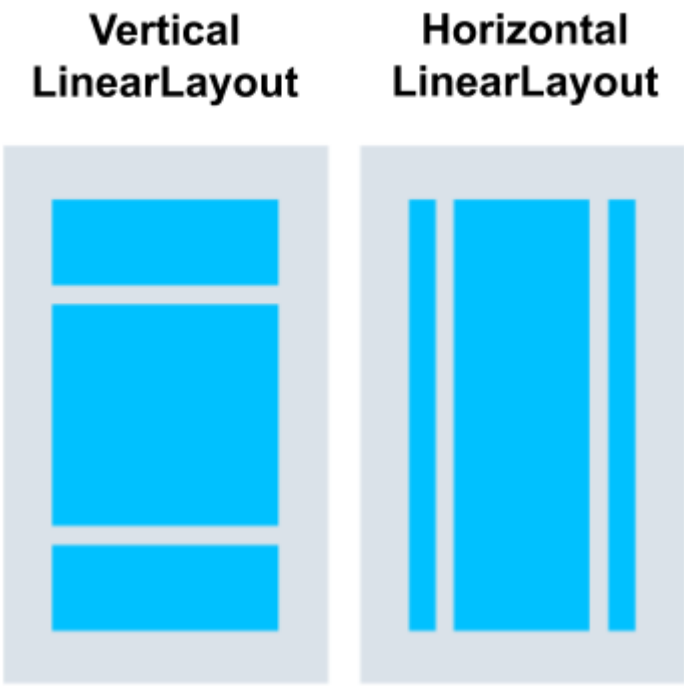
- View class represents the basic building block for user interface components.
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
- All of the views in a window are arranged in a single tree.
- View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.)

```
import android.content.Context;
import android.widget.TextView; // Or the relevant View subclass

public class MySimpleView extends TextView {

    public MySimpleView(Context context) {
        super(context);
        // Optional: Customize view appearance here
    }
}
```

```
setText("This is large text");
        setTextSize(24); // Set larger text size
        setTextColor(Color.BLUE);
```

Linear Layout

- LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
- You can specify the layout direction with the **android:orientation** attribute.
- Linear Layout can be created in two direction: **Horizontal** & **Vertical**.
- LinearLayout also supports assigning a **weight** to individual children with the **android:layout_weight** attribute.
- This attribute assigns an important value to a view in terms of how much space it should occupy on the screen.
- A larger weight value allows it to expand to fill any remaining space in the parent view.



Relative Layout

- RelativeLayout is a view group that displays child views in relative positions.
- The position of each view can be specified as relative to sibling elements or in positions relative to the parent RelativeLayout area.
- As it allows us to position the component anywhere, it is considered as most flexible layout.
- Relative layout is the most used layout after the Linear Layout in Android.
- In Relative Layout, you can use “above, below, left and right” to arrange the component’s position in relation to other component.
- In this view group child views can be layered on top of each other.

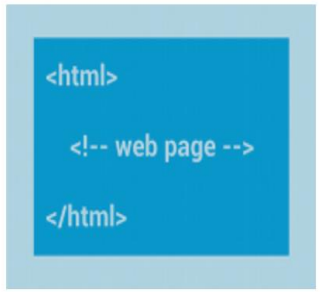
Linear Layout

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.

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).

Web View

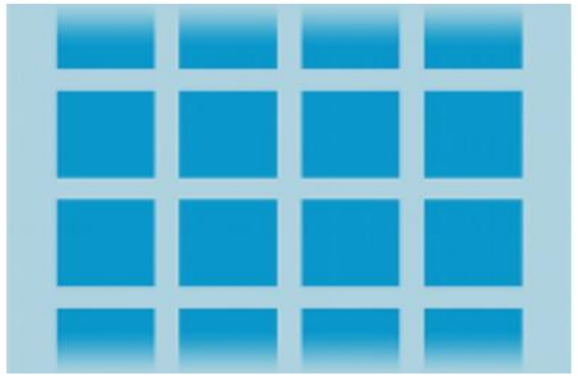


Displays web pages.

List View

Displays a scrolling single column list.

Grid View



Displays a scrolling grid of columns and rows.

Built from data using an Adapter

Frame Layout

- Frame Layout is designed to block out an area on the screen to display a single item.
- It is used to specify the position of Views.
- It contains Views on the top of each other to display only single View inside the FrameLayout.
- You can, add multiple children to a FrameLayout and control their position by using gravity attribute.
- In this the child views are added in a stack and the most recently added child will show on the top.

example

```
public class MainActivity extends AppCompatActivity {
    private TextView textView;
    private int clickCount = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Initialize views
        Button button = findViewById(R.id.button);
        textView = findViewById(R.id.textView);
        // Set click listener for the button
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Increment click count and update text
                clickCount++;
                textView.setText("Button clicked " + clickCount + " times");
```

TextView

- TextView is a UI Component that displays the text to the user on their Display Screen.

Attribution	Description
android:id	This is the ID which uniquely identifies the control.
android:fontFamily	Font family (named by string) for the text.
android:gravity	Specifies how to align the text when the text is smaller than the view.
android:maxHeight	Makes the TextView be at most this many pixels tall.
android:maxWidth	Makes the TextView be at most this many pixels wide.
android:text	Text to display in view.
android:textAllCaps	Make the text in Capital. Possible value either "true" or "false".
android:textColor	Text color. May be a color value, in the form of **"#rgb"**, **"#argb"**, **"#rrggbb"**, or **"#aarrggbb"**.
android:textStyle	Style (bold, italic, bolditalic) for the text. You can use or more of the values separated by '|'. **normal – 0**, **bold – 1**, **italic - 2**
android:typeface	Typeface (normal, sans, serif, monospace) for the text. You can use or more of the values separated by '|'. **normal – 0**, **sans – 1**, **serif – 2**, **monospace – 3**.

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.

Attribute	Description
android:autoText	TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:editable	If set, specifies that this TextView has an input method.
android:background	This is a drawable to use as the background.
android:inputType	The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:hint	Hint text to display when the text is empty.
android:focusable	It specifies that this edittext gets auto focus or not.

Button

- A Button which can be pressed, or clicked, by the user to perform an action.
- To specify an action when the button is pressed, set a click listener on the button object in the corresponding activity code.
- Every button is styled using the system's default button background it can customize.

Attribute	Description
android:text	This is used to display text on button.
android:background	This is a drawable to use as the background.
android:id	This supplies an identifier name for this view.
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.

Card View

- A CardView is child of FrameLayout with a rounded corner background along with a specific elevation.
- The main usage of CardView is that it helps to give a rich feel and look to the UI design.
- These cards have a default elevation above their containing view group, so the system draws shadows below them.
- Information inside cards that have a consistent look across the platform.
- It can be used for creating items in ListView or inside RecyclerView.

Attribute	Description
card_view:cardBackgroundColor	Can set background color.
card_view:cardCornerRadius	To set Radius or card corner
card_view:cardElevation	To set elevation to tha card
card_view:cardUseCompatPadding	Set compatible padding to cardview based on radius and elevation

ListView

- 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 acts as a bridge between UI components and the data source that fill data into UI Component.

Attribute	Description
android:divider	This is drawable or color to draw between list items.
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.

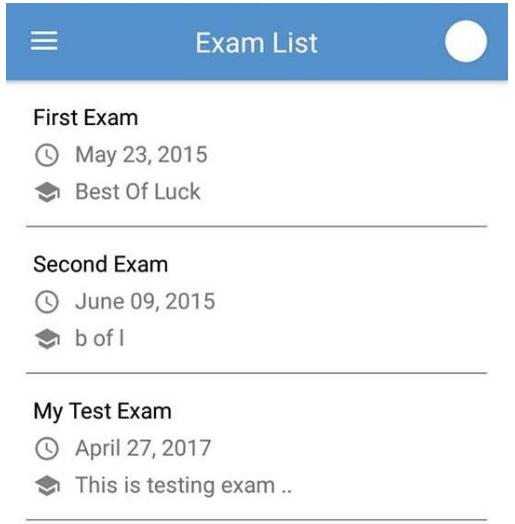
Adapter

- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
- In Android development, any time we want to show a vertical list of scrollable items we will use a ListView which has data populated using an Adapter.
- The simplest adapter to use is called an ArrayAdapter because the adapter converts an ArrayList of objects into View items loaded into the ListView container.
- The ArrayAdapter fits in between an ArrayList (data source) and the ListView (visual representation)
- When your ListView is connected to an adapter, the adapter will instantiate rows until the ListView has been fully populated with enough items to fill the full height of the screen. At that point, no additional row items are created in memory.

```
ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items);
```

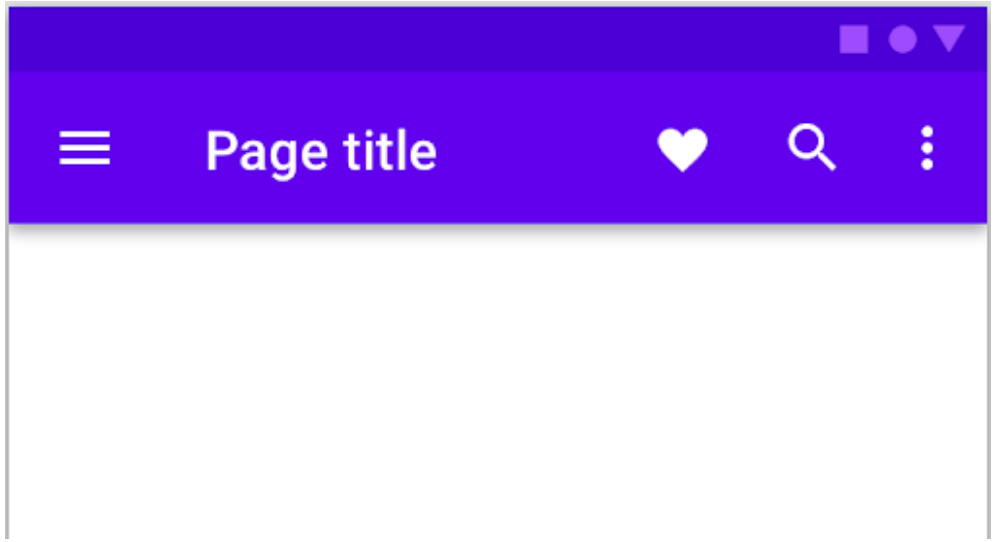
Recycler View

- RecyclerView is a ViewGroup added as a successor of the GridView and ListView.
- RecyclerView is mostly used to design the user interface with the fine-grain control over the lists and grids of android application.
- To implement a basic RecyclerView three sub-parts are needed.
  - 1) The Card Layout : it is an XML layout which will be treated as an item for the list created by the RecyclerView.
  - 2) The ViewHolder: it is a java class that stores the reference to the card layout views that modified during the execution of list.
  - 3) The Data Class: it is a custom java class (getter - setter) that acts as a structure for holding the information for every item of the RecyclerView.
- The adapter is main part for RecyclerView for displaying the list.
  - onCreateViewHolder: which deals with the inflation of the card layout as an item.
  - onBindViewHolder: which deals with the setting of different data and methods related to clicks on particular items.
  - getItemCount: which Returns the length of the RecyclerView.



Material Design Toolbar

- The toolbar gave us so much space for customization and creation for ActionBar.
- MaterialToolbar is a Toolbar that implements certain Material features, such as elevation overlays for Dark Themes and centered titles.
- Toolbars appear a step above the sheet of material affected by their actions.
- Unlike ActionBar, its position is not hardcoded it can be place anywhere according to the need just like any other View in android.
- **Use as an ActionBar:** In an app, the toolbar can be used as an ActionBar in order to provide more customization and a better appearance. All the features of ActionBar such as menu inflation, ActionBarDrawerToogle, etc. are also supported in Toolbar.



Tab Layout

- **TabLayout** is used to implement horizontal tabs.
- Population of the tabs to display is done through TabLayout.Tab instances.
- Create tabs via newTab() From there we can change the tab's label or icon via TabLayout.Tab.setText(int).
- A TabLayout can be setup with a ViewPager
  - Dynamically create TabItems based on the number of pages, their titles, etc.
  - Synchronize the selected tab and tab indicator position with page swipes.
- There are two types of tabs: Fixed tabs, Scrollable tabs.
- 1) Fixed tabs display all tabs on one screen, with each tab at a fixed width.
- 2) Scrollable tabs are displayed without fixed widths. They are scrollable, such that some tabs will remain off-screen until scrolled.

Menus

- To provide a familiar and consistent user experience **menus** are the best.
- It define in separate XML file and use that file in our application based on our requirements.
- We can use menu APIs to represent user actions and other options in our android application activities.
- The Menus in android applications are :
  - 1) Options Menu
  - 2) Context Menu
  - 3) Popup Menu

Options Menu

- The options menu is the primary collection of menu items for an activity.
- Options Menu Generally placed on action bar.
- We can declare items for the options menu from either Activity or a Fragment.

- Re-order the menu items with the android:orderInCategory attribute from xml file menu item.
- To specify the options menu for an activity, override onCreateOptionsMenu().
- In this method, you can inflate your menu resource (defined in XML) into the Menu.
- You can also add menu items using add() and retrieve items with findItem().
- When the user clicks a menu item from the options menu onOptionsItemSelected() method is used to get callback.

Contextual Menus

- You can provide a context menu for any view. but they are most often used for items in a RecycerView, ListView, GridView items.
- There are two ways to provide contextual actions.
  - 1) In a floating context menu appears as a floating list of menu items when the user performs a long-click on a view that declares support for a context menu.
  - 2) In the contextual action mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items ex. Select All Items, Delete Items, etc...
- It can be associate with view using registerForContextMenu() method and pass it the View.
- Menu item inflated to context menu using onCreateContextMenu() method in your Activity or Fragment.
- Menu items click event is handled by onContextItemSelected() in Activity or Fragment.

Popup Menu

- A Popup Menu displays a Menu in a popup window anchored to a View.
- The popup will be shown below the anchored View if there is room(space) otherwise above the View.
- If any Keyboard is visible the popup will not overlap it until the View is touched.
- Touching outside the popup window will dismiss it.