Pre – Requirements for ANDROID

Basic Introduction

Android – Open Source & Linux Based OS for Mobile Devices, Smartphones, Tablets, Computers et…

Android apps can be developed using Kotlin Programming Language or Java Programming Language.

Java is a simple, powerful, and robust object-oriented programming language suited for various purposes like Android apps, web apps, server apps, embedded systems, big data and more.

Kotlin is a programming language that runs on a Java virtual machine (JVM), can be compiled into JavaScript, and run-in browsers. An Android developer can code on Kotlin/Native and use IDE to build cross-platform apps.

Kotlin and Java provide almost the same speed for coding.

As both Kotlin and Java compile to ByteCode (which runs on JVM) it is a daunting task to compare them when it comes to memory usage. Hence, measuring, tracking, and comparing their performance is hard.

Android applications are usually developed in Java language using android software development kit.

Basic Understanding of Java is Required for Android

1. JAVA OOPS Concepts
2. Control Statements
3. Java Basics
4. Interface
5. Inheritance
6. Packages
7. Type Casting
8. Multi-Threading
9. Exception Handling
10. Java Annotations
11. IO
12. Collections

Syntax :

1. Variables :
   int rollnumber = 3;
   char name = 'X';
   float income = 500.36;
   Local Variables: These can be defined inside method, constructor or also inside block. The scope or life time of local variable destroyed with end of method completion.

   Instance variables: These are associated with the object creation. As the object get created instance variable also get created

   Class Variable/static variables: These are loaded and initialized when class is loaded in JVM and there exists only one copy of class variable

```
public class TypeOfVariable{
  public static int staticvariable;
  int instancevariable;
  public void printValue(){
        int localvariable;
        System.out.println("the value of staticvariable \t"+staticvariable);
        System.out.println("the value of instancevariable \t"+instancevariable);
        System.out.println("the value of localvariable \t"+localvariable);
  }
  public static void main(String args[]){
        TypeOfVariable object=new TypeOfVariable();
        object.printValue();
  }
}
```

2. Data Type
   Byte
   Short
   Int
   Long
   Char
   Float
   Double
   Boolean

3. String
   String s1= "Welcome to Android Lecture Series";
   String s2 = new String("This is example of String");
   Functions : compare(), concat(), equals(), split(), length(), replace(), compareTo(), charAt(), endsWith(), indexOf(), trim(), toLowerCase(),startsWith()

4. Operators
   4.1 Arithmetic Operator : +, -, /, *, %
   4.2 Unary Operator : +2, -2, x++, --x,
   4.3 Equality & Relational Opeator : = =, !=, <, >, <=, >=
   4.4 Conditional Operator : &&, ||, ?:
   4.5 Assignment : =, +=, -=, *=,
   4.6 Bitwise Operator : &, ^, |,

5. Keywords : Abstract, Continue, For, New, switch, assert, default, goto, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum****, instanceof, return, transient, catch extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp**, volatile, const*, float, native, super and while

6. Class & Objects

```
class LearnAndroid{
        int sub_code;
        String file_path;

}

LearnAndoroid la  = new LearnAndroid();
```

7. Arrays
   Array Declaration in JAVA

```
int[] arr;
```

```
arr = new int [5];
int [] arr = {10, 20, 30, 40, 50};

String fullname  = new String[3];
Fullname[0] = "Jenis"
Fullname[1] = "Jayesh"
Fullname[2] = "Shah"
```

Multidimensional Array :

```
Int [][] learnArray = new int[2][3];

int[][] learn2DArray= new int[2][];//right way to write
int[][] learn2DArray= new int[][3];//wrong way to write

int[][] learn2DArray=new int[2][];
learn2DArray[0]=new int[2];
learn2DArray[1]=new int[2];
```

8. Inheritance

```
class Base
{
//Code of Base class
}

Class Child extends Base

{
//extends the properties of base class
}
```

Multiple inheritance is not supported in java.
Types:
Single, Multi Level and Hierarchical

9. Abstraction
It is a process of hiding internal working and showing only necessary details.

```
abstract class LearnAndroid
{
        abstract void test();
        abstract void messaging();
        {
                System.out.println("Messaging");
        }
}

class TestAndroid extends LearnAndroid
{
        Void test()
        {
                System.out.println("Start Testing");
        }
}

psvm(String args[])
{
        TestAndroid ta = new TestAndroid();
        ta.test();
}
```

10. Interface
    - 100% Abstraction
    - Contains methods which has no implementation
    - Uses implement keyword

```
interface vehicle {
        void inition();
        void breaking();
}


class TwoWheeler implements vehicle {
        public void ignition() {
```

```
        }
    }
```

One interface can extends another interface. Remember interface cannot implements other interface, only class can do that.

all the methods of Interface does not have body at all. So these methods must be implemented in some other class before accessing them.

Class which is implementing Interface must override/implement all the declared methods inside interface

11. Encapsulation

    Encapsulation is a process of hiding the data from the users or in other words we can say it protects the code by preventing access from the outside class.

```
class Test
{
        int a;
        int b;

        public Test(int a, int b)
        {
                this.a= a; this.b= b;
        }
        public void setA(int a){     this.a =a    }
        public int getA() {  return a; }
}
```

12. Polymorphism

    - Run time polymorphism: Dynamic Binding at Run time : Method Overriding
    - Compile time polymorphism : Method Overloading

```
class Test
{

        public int sum(int a, int b)
        {
                return (a+b);
        }
        public int sum(int a, int b, int c)
        {
                        return(a+b+c);
        }
}
```

Runtime

```
class Base {
        public void read()
        {
                System.out.println("Hello Base Class");
        }
}
class Child{
        public void read()
        {
                System.out.println("Hello Child Class");
        }
```

```
}

class TestRunTimePolymorphism
{
        public static void main (String args[])
        {
                Base b = new Child();
                b.read(); // call child class
                Base c = new  Base();
                b.read(); // call base class
        }

}
```

13. Constructor :
    - A constructor always has a same name as the class whose instance members they initialize.
    - A constructor does not have a return type,
    - not even void.
    - It is because the constructor is automatically called by the compile whenever an object of the class is created.

```
class Test
{
        int a,b;
        Test()
        {
                a = 3; b = 5;
        }
        Test(int c, int d)
        {
                a = c; b = d;
        }
```

}

Default Constructor : public Test() {}

Parameterized Constructor: `` public Test(int c, int d) {}

Copy Constructor : Test(Test obj) { a = obj.a; b = obj.b }

How to call constructors:

Default constructor: Test t = new Test();

Parameterized Constructor : Test t1 = new Test(3,4);

Copy Constructor : Test t2 = new Test(t1);

14. Multi Threading
    - Multithreading in Java is a process of executing multiple threads simultaneously.
    - Thread = Light Weight Sub Process
    - Non Blocking Paradigm, Multiple Operations at a Same time

Creating Thread

- By Extending Thread Class
- Implementing Runnable Interface

Ex:

- Thread()
- Thread (String Name)
- Thread(Runnable r)
- Thread (Runnable r, String Name)

Methods of Thread :

Run(), start(), sleep(), join(), getPriority(), setPriority(),suspend(), resume(), stop(), interrupt(), getId() etc…..

By extending Thread Class

```
class MultiThreadEx extends Thread {
        public void run(){
          System.out.println("Thread is running");
        }



        public stactic void main(String args[])
        {
          MultiThreadEx mlt1 = new MultiThreadEx();
          mlt1.start();
        }
}
```

Implementing Runnable Interface

```
class MultiThreadEx2 implements Runnable{
        public void run(){
        System.out.println("thread is running...");
        }

        public static void main(String args[]){
        MultiThreadEx2 m1=new MultiThreadEx2 ();
        Thread t1 =new Thread(m1);
        t1.start();
```

```
    }
}
```

Multithreading in Android

Multi-Threading in Android is a unique feature through which more than one threads execute together without hindering the execution of other threads.

Multi-Threading in Android is not different from conventional multi-Threading.

15. Exception handling in Java

- Checked Exception : IOException, SQLException etc. Checked exceptions are checked at compile-time.
- Unchecked Exception : ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException
  they are checked at runtime
- Errors
  OutOfMemoryError, VirtualMachineError, AssertionError etc…

```
public class ExceptionExample{

  public static void main(String[] args) {
    try
    {
    int data=50/0; //may throw exception
    }
      // handling the exception by using Exception class
    catch(Exception e)
    {
      System.out.println(e);
    }
    System.out.println("rest of the code");
  }
```

```
}
```

16. Collections

Lists

- ArrayLists : ArrayList is a dynamic data structure in which you can add or remove any number of elements and those elements are stored in ordered sequence. It may also contain duplicate values.

```
ArrayList<Integer> aList = new ArrayList<Integer>();

aList.add(5);
aList.add(11);
aList.add(17);
```

Other methods : clear(), clone(),

- LinkList : Linked List is a type of Linear Data Structure that is second mostly used data structure after array, which allocates memory dynamically at run time that is it doesn't require any size initialization as in case of array.

```
LinkedList<String> linkedList=new LinkedList<String>();
```

- Vector:

Vector is type of data structure that implements List Interface. It is very much similar to ArrayList as it also maintains insertion order, that is elements are retrieved in same order as they are added into it.

Like Arrays it has index value that is automatically allocated to it, but main think which distinguishes it is, it can grow or shrink its capacity according to the need after it is created, as it implements growable array.

```
Vector<Integer> vectorObject = new Vector<Integer>(4);

 vectorObject.add(3);
 vectorObject.add(5);
```

Map

- HashMaps

HashMap is a type of Collection, that stores our data in a pair such that each element has a key associated with it. The pair of key and value is often known as Entry and these entries can have only unique keys.
HashMap does not allow Entry with duplicate key, it overlaps old value with new one. Below program helps you understand this.

```
HashMap<Integer,String> HashMap=new
HashMap<Integer,String>();

//Step 2: Adding Key Value pair

HashMap.put(1001,"India");
```

- LikedHashMap,

In Addition to all the functionalities of HashMap Class, the functionality of maintaining the insertion is added into LinkedHashMap and to attain this functionality all the entries(key and value) are linked to each other using doubly-linked list. This doubly-linked list maintains the iteration ordering, which is in general the order in which keys were added in the map.

```
Map<String, Integer> linkedHashMapobject = new
LinkedHashMap<String, Integer>();

linkedHashMapobject.put("Samsung Grand quattro price ", new
Integer(10000));
```

TreeMap

Others are iterator, Set, HashSet, TreeSet

SDLC

Requirements Gathering → Planning → Designing → Developing → Testing → Deployment

Various SDLC Models

Waterfall Model

Iterative Model

Spiral Model

Flowchart

Improved Communication. Flowchart software empowers entire teams to collaborate as they create, edit, and analyze flowcharts. ...

Benefit 2: Visual Clarity. ...

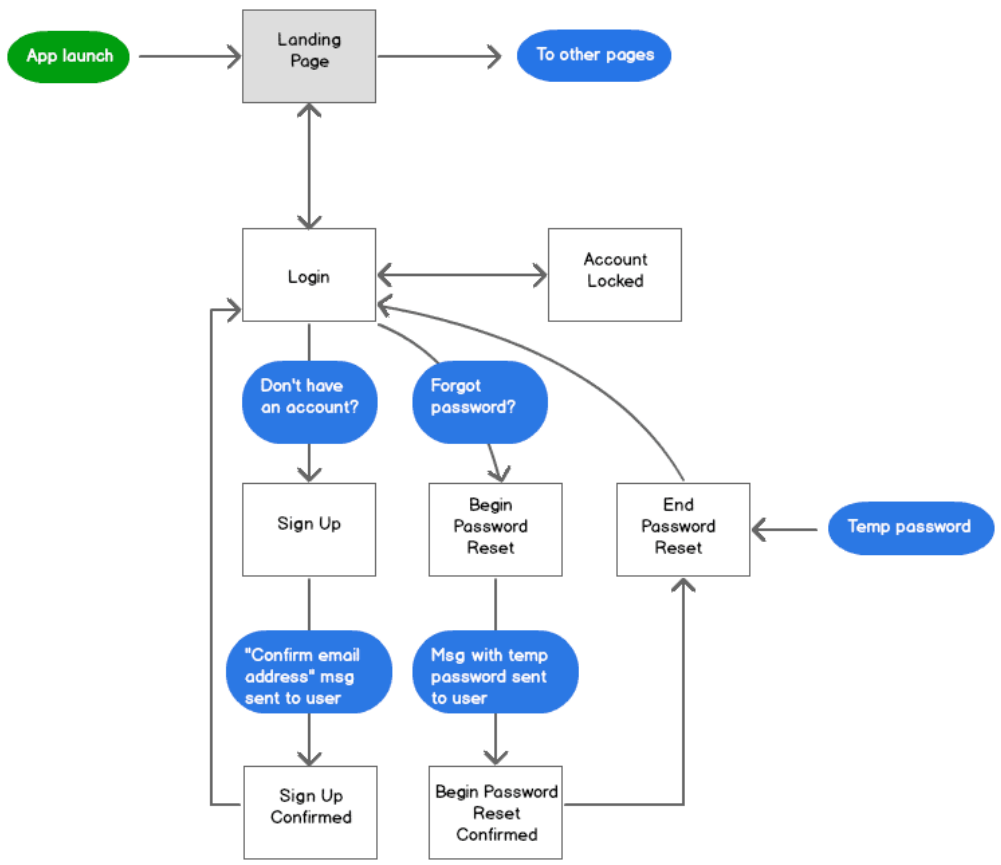
Benefit 3: Effective Analysis. ...

Benefit 4: Problem Solving. ...

Benefit 5: Documentation.

	Terminal/terminator	Indicates start/end of the flowchart or process
	Decision	Represents different decisions emerging from different points
	Action/Process	Represents an action or process
	Input/output	Holds the input/output information
	Connector	Indicates the flow connection to the next symbol
	Document	Indicates a report or a document
	Multiple document	Indicates multiple documents or reports
	Alternate	Indicates an alternate process to take place
	Preparation	Indicates preparation taken for the following step

Flowchart for basic User Login Registration and Forgot Password



Data Flow Diagram

A data flow diagram is a graphical depiction of flow of data through intended software system and is used as 1st step to create an overview of system.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

- All names should be unique. This makes it easier to refer to elements in the DFD.
- Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
- A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
- Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

External Entity (User)

Source of System Input

Or Sink of System Outputs

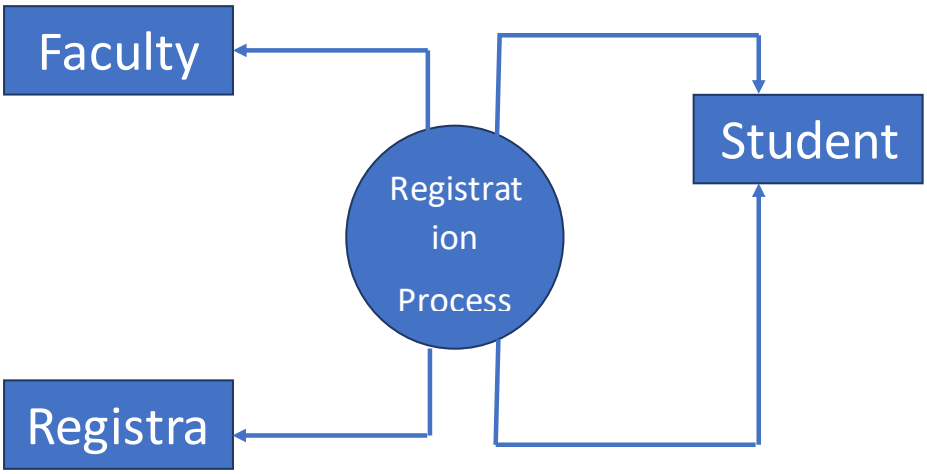
Process – Circle :

Perform some transformation of input data to get output data

Data Flow

Ex:

Context Time :



Level 1 DFD Diagram

SQL

SQL is to insert data into specific tables and to extract that data when required using a range of different filters.

Insert

Delete

Update

Read

Introduction To Android

- Open Source Linux based Operating System
- Developed by Open Handset Alliance led by Google
- Anyone can download the source code of android, change it as per their requirements, add their own features
- Java language is mainly used to write the android code even though other languages can be used.
- Goal of android project is to create a successful real-world product that improves the mobile experience for end users.

Why Android ?

- Open Source
- Largest development community
- Increased marketing
- Inter app integration
- Reduced cost of development
- Higher success ratio
- Rich development environment

Android Versions Year Name API Level

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:

- A core set of packages and classes
- A set of XML elements and attributes for declaring a manifest file
- A set of XML elements and attributes for declaring and accessing resources
- A set of Intents
- A set of permissions that applications can request, as well as permission enforcements included in the system

1.0	23 Sep, 2008	N/A	1
1.1	9 Feb, 2009	N/A	2
1.5	30 Apr, 2009	Cupcake	3
1.6	15 Sep, 2009	Donut	4
2.0/2.1	26 Oct, 2009	Éclair	5-7
2.2	20 May, 2010	Froyo	8
2.3	6 Dec, 2010	Gingerbread	9-10

3.0/3.1/3.2	22 Feb, 2011	Honeycomb	11-13
4.0	18 Oct, 2011	Ice Cream Sandwich	14-15
4.1/4.2/4.3	9 Jul, 2012	Jelly Bean	16-18
4.4	31 Oct, 2013	KitKat	19-20
5.0/5.1	12 Nov, 2014	Lollipop	21-22
6.0	5 Oct, 2015	Marshmallow	23
7.0	2016 End	Nougat	24
8.0	21 Aug, 2017	Oreo	26
9.0	16 Aug 2018	Pie	27
10	Android 10(Queen Cake)	September 7 2019	29
11	Android 11(Red Velvet Cake)	September 8 2020	30
12	Android 12 (Snow cone)	Launching Soon	

Features Of Android

1. NTC (Near Field Communication) : Easily interact across short distance
2. Alternate Keyboards
3. Infrared Transmission (Use phone or tablet as remote control)
4. No-Touch Control (We can say it as gesture control)
5. Automation : Control app permission like location, music etc.
6. Storage: SQLite for storage purpose in smartphones
7. Media Support : Audio, Images & Video in various formats
8. Messaging : SMS & MMS
9. Web Browser
10. Connectivity: networks like: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, WiFi, LTE and WiMAX.
11. Hardware Support : Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor & GPS and a lot more.
12. Tethering : Supports sharing of Internet as wired or wireless hotspots.
13. Multi Touch and Multi Tasking
14. Video calling
15. Screen capture
16. External storage
17. Streaming media support
18. Optimized graphics
19. Widgets
20. Custom ROMs

Where you can publish your app

- Google play store
- SlideME
- Amazon App Store
- Aptoide
- AppsZoom
- Opera Mobile Store
- Mobango
- 1Mobile

Categories For Android Applications

- Entertainment
- Music
- News
- Multimedia
- Sports
- Lifestyle
- Food & Drink
- Travel
- Weather
- Books
- Business
- Reference
- Navigation
- On demand delivery (E-commerce)
- Social Media
- Utilities
- Finance

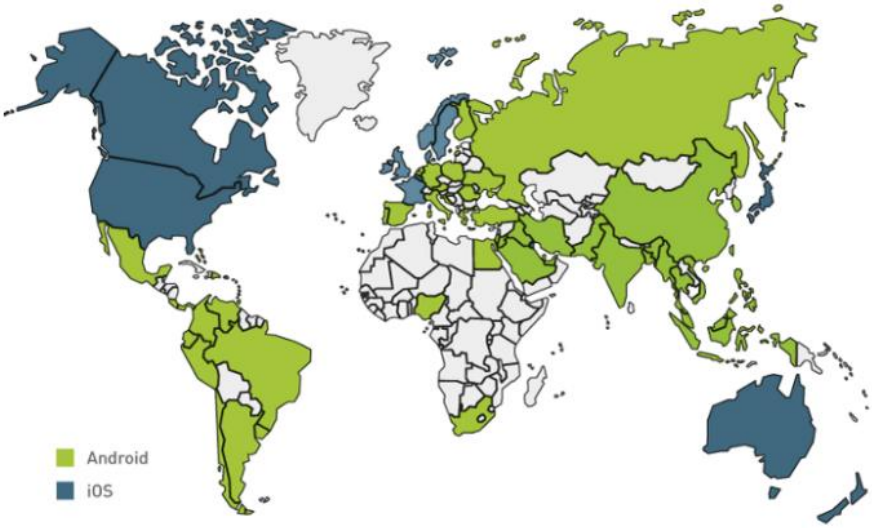
Advantages of Android Development For Developers

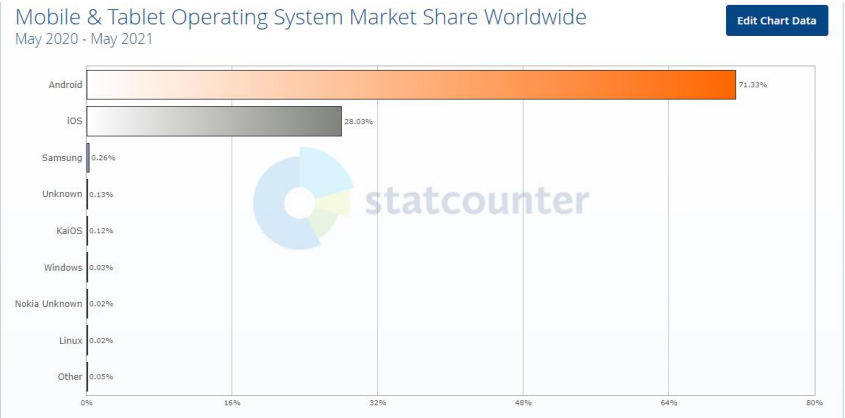
- Open Source
- Source code is freely available
- Customizable UI
- Faster Deployment
- High ROI with Low cost of investment
- Versatility & Scalability
- More 80% users are Android Device users
- Multiple Sales Channels

Disadvantages of Android Development For Developers

- Need multiple devices & time for testing
- Multitude of device screen size makes UI development challenging
- Launching new feature may malfunction on other devices
- Cost of testing my increase
- Device issue : Storage limit, Drains Battery, Over heating

Global Users of Android & iOS





Tools Required to start Android Development

- Android Studio
- JDK
- Genymotion (Emulator For Developers)

Other Tools That You Can Use

- Eclipse
- IntelliJ Idea
- Unity 3D
- Many More…

Android Architecture

Android Architecture stack

1. Linux kernel
2. Native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications

Linux Kernel

Android was created on the open source kernel of Linux. One main reason for choosing this kernel was that it provided proven core features on which to develop the Android operating system

This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc.

It is responsible for device drivers, power management, memory management, device management and resource access.

HAL

The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Libraries:

Running on the top of the kernel, the Android framework was developed with various features. It consists of various C/C++ core libraries with numerous of open source tools. Some of these are:

On the top of linux kernel, their are Native libraries such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc.

The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.
- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.
- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

The Android runtime:

The Android runtime consist of core libraries of Java and ART (the Android RunTime). Older versions of Android (4.x and earlier) had Dalvik runtime.

Open GL(graphics library):

This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.

WebKit:

This open source web browser engine provides all the functionality to display web content and to simplify page loading.

Media frameworks:

These libraries allow you to play and record audio and video.

Secure Socket Layer (SSL):

These libraries are there for Internet security.

Android Run Time

It is the third section of the architecture. It provides one of the key components which is called Dalvik Virtual Machine. It acts like Java Virtual Machine which is designed specially for Android. Android uses it's own custom VM designed to ensure that multiple instances run efficiently on a single device.

The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality ,including security, threading and memory management.

It consumes less memory and provides fast performance.

Application Framework

The Android team has built on a known set proven libraries, built in the background, and all of it these is exposed through Android interfaces. These interfaces warp up all the various libraries and make them useful for the Developer.

1. Activity Manager:
   It manages the activity lifecycle and the activity stack.

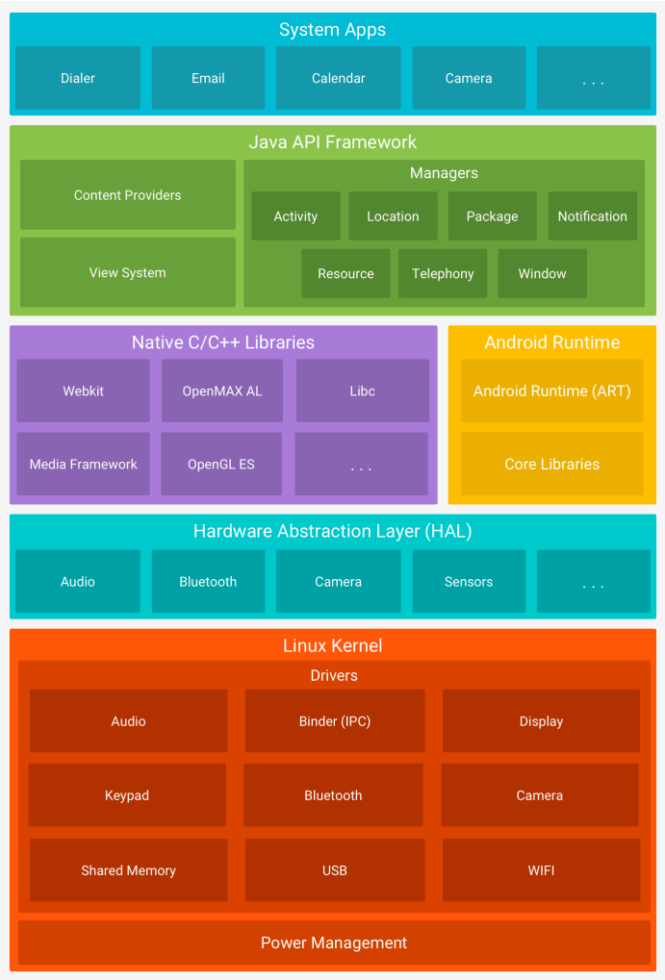
2. Telephony Manager:
   It provides access to telephony services as related subscriber information, such as phone numbers.

3. View System:
   It builds the user interface by handling the views and layouts.

4. Location manager:
   It finds the device's geographic location.
5. **Content Providers** – Allows applications to publish and share data with other applications.
6. **Resource Manager** – Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
7. **Notifications Manager** – Allows applications to display alerts and notifications to the user.

Applications:

Android applications can be found at the topmost layer. At application layer we write our application to be installed on this layer only. Examples of applications are Games, Messages, Contacts etc.



Speech : Linux Kernal :

Identify the various Drivers Like Wifi, Pendrive, Mouse, Keyboard

Kernal is used to work between Hardware devices and OS

Kernal is Heart of OS

HAL : Work as Interface between Application & Kernal (Camera app request to HAL and send back to Kernal)

Native Library Layers:

Libraries that is used to work for Android Apps

Mainly programmed in C/C++

Kernal is also written in C/C++

OPEN GL: Used to produce 2D and 3D Graphics

WebKit: WebBroser Engine to show Data from Website

Media Framework :

SQl Library: to store

Android Runtime : TO Run Apps & DVM is there User less power and memory specially designed for

JVM: /java-- .class --- machine code

DVM: .java—classs--.dex—machine code

AFTER lollipop DVM is replaced by ART

Framework: Managers like Alarm, Notification,

App development is there in Framework

Application layer: In which the apps are working

XMPP " Extensible Messaging and Presence Protocol.

Resourses: Non Programmable things

ELF " Execulable link formate " Dex and Resourse code

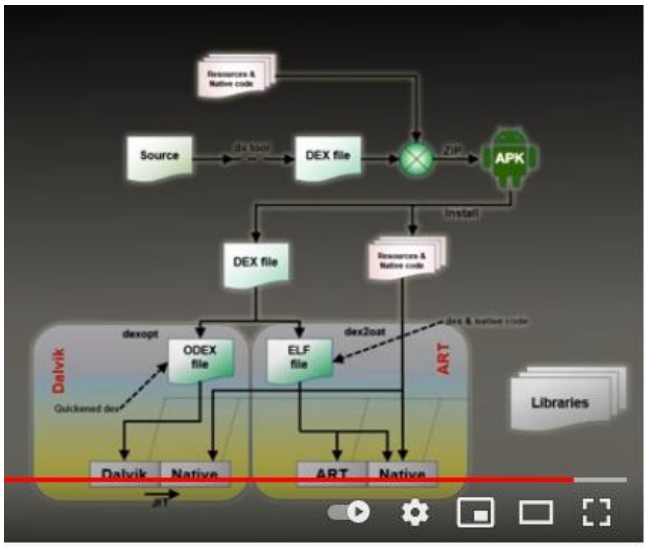
DEX : dalvik executable

ODex" Optimize dalvik Executable files

Dex2Oat "

DexOpt

Ahead of time Compilation



Android Building Blocks or App Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

- Activities: They dictate the UI and handle the user interaction to the smart phone screen.:
  An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.

- Services: They handle background processing associated with an application.
  A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

- Broadcast Receivers: They handle communication between Android OS and applications.
  Broadcast Receivers simply respond to broadcast messages from other applications or from the system. 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.

- Content Providers: They handle data and database management issues.

  A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely.

- Fragments: Represents a portion of user interface in an Activity.
- Views: UI elements that are drawn on-screen including buttons, lists forms etc.
- Layouts: View hierarchies that control screen format and appearance of the views.
- Intents: Messages wiring components together.
- Resources: External elements, such as strings, constants and drawable pictures.
- Manifest: Configuration file for the application.

Android Advance Components

SQLite

Webservice

XML

JSON

Retrofit

Material Design

GSON

Firebase

Google Map

Google Places

Google Cloud Messaging Services

Graphics Programming

SQLite: Use to store the data

App maintain data in one device

DisAdvantage: Data limited – we can not access data of other device or store data over there

In real time data should be accessible from anywhere from world

Apart from storing data in SQL db

We may use SQL Server and MySQL

From APP we can not communicate with these servers directly

How to store data inthat ?

This communication possible using these technologies

: java php .net

App → Technology → DB Servers

.net cannot understand the java object or vice versa

Hindi – Telugu

English as a co

- Need Common Language As well as Same Grammar rules and medium to communicate
  Android " java code ==== MySQL Server has .net

1. Common language: XML or JSON (Transfer the data bet technologies)
2. Grammar rules : Protocols
3. Medium : Local Network or Internet

Web Services : Providing a communication bet two different technologies

Heavyweight webservices and restful service

Retrofit: For Powerful Performance for Restful Services

How to call web services (XML + JSON knowledge)

GSON ----- Library to convert your object data into JSON and JSON to Object

Material Design: For Rich UI – New advance Ui Componenets like Recycler view – toolbar- etc..

Fire Base: if you don't know web development then you have firebase

Not required to knowledge of Webservives

Firebase Notificaiton – database storage – authentication – modifications – remote configurations – crash report – Push Notificaitons

No server side prog is needed.

Android Activity Life Cycle

- Each activity goes through various stages or a lifecycle and is managed by activity stacks.
- when a new activity starts, the previous one always remains below it. There are four stages of an activity.

Activity Stage

1. An activity is in the foreground of the screen
   - Must be On Top of The Stack
   - Said to be Active / Running
   - It's the activity that user is currently interacting with.
2. Activity has lost focus and a non-full-sized or transparent activity on top of your activity.
   - Another activity has a higher position in multi-window mode
   - The activity itself is not focusable in the current window mode.
   - Such activity is completely Alive.
3. An activity is completely hidden by another activity,
   - it is stopped or hidden
   - It still retains all the information,
   - As its window is hidden thus it will often be killed by the system when memory is needed elsewhere.
4. The system can destroy the activity from memory
   - Destroy Activity either asking it to finish or simply killing its process.
   - When it is displayed again to the user, it must be completely restarted and restored to its previous state.

Activity – Call Backs

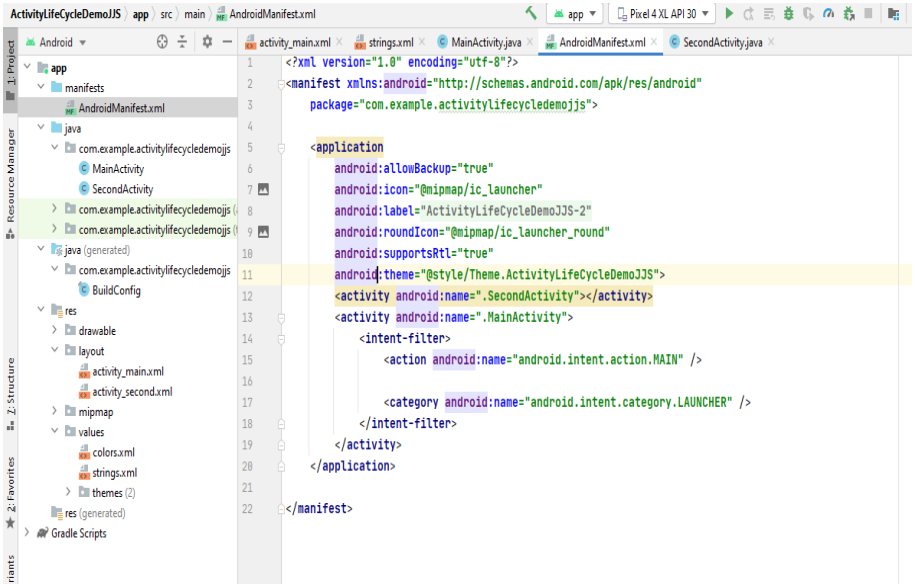
- Activity class provides a number of callbacks that allow the activity to know that a state has changed
- Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.
- Ex: Video Streaming App – Internet - Switch To Another App – Back to Same App

Good Implementation of Lifecyle Call Backs Avoids

- Crashing When User Receives Call or Switch to another app
- Consume Valuable Resources when User not actively using it
- Losing user's progress if user leaves app & return to it later
- App Crashing or Losing Progress while Changing Screen – Portrait / Landscape Orientation

Activity Call Back Methods

- To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks:
- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()
- System invokes each callbacks when Activity Enters New State



Main Activity.java

```
package com.example.activitylifecycledemojjs;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast;
        Toast.makeText(getApplicationContext(), "onCreate Called",
Toast.LENGTH_LONG).show();

        TextView textView = findViewById(R.id.firstActivity);
        textView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent;
                intent = new Intent( MainActivity.this,
SecondActivity.class );
                startActivity(intent);
            }
        });
    }
```

```
    protected void onStart() {
        super.onStart();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onStart Called",
Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onRestart Called",
Toast.LENGTH_LONG).show();
    }

    protected void onPause() {
        super.onPause();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onPause Called",
Toast.LENGTH_LONG).show();
    }

    protected void onResume() {
        super.onResume();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onResume Called",
Toast.LENGTH_LONG).show();
    }

    protected void onStop() {
        super.onStop();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onStop Called",
Toast.LENGTH_LONG).show();
    }

    protected void onDestroy() {
        super.onDestroy();
        Toast toast;
        Toast.makeText(getApplicationContext(), "onDestroy Called",
Toast.LENGTH_LONG).show();
    }

}
```

SecondActivity.java

```
package com.example.activitylifecycledemojjs;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.example.activitylifecycledemojjs.R.layout;
```

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(layout.activity_second);
        Toast toast;
        toast = Toast.makeText(this, "Second On Crreate",
Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

        TextView textView = findViewById(R.id.firstActivity);
        textView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent;
                intent = new Intent( SecondActivity.this,
MainActivity.class );
                startActivity(intent);
            }
        });
    }

    protected void onStart() {
        super.onStart();
        Toast toast;
        toast = Toast.makeText(this, "Second On Start", Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast toast;
        toast = Toast.makeText(this, "Second On Restart",
Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

    }

    protected void onPause() {
        super.onPause();
        Toast toast;
        toast = Toast.makeText(this, "Second On Pause", Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

    }

    protected void onResume() {
        super.onResume();
        Toast toast;
        toast = Toast.makeText(this, "Second On Resume",
Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
```

```
    }

    protected void onStop() {
        super.onStop();
        Toast toast;
        toast = Toast.makeText(this, "Second On Stop", Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

    }

    protected void onDestroy() {
        super.onDestroy();
        Toast toast;
        toast = Toast.makeText(this, "Second On Destroy",
Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

    }


}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/teal_200"
    tools:context=".MainActivity">

    <TextView

        android:id="@+id/firstActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="serif-monospace"
        android:text="First Activity"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.499" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Activity_second.xml

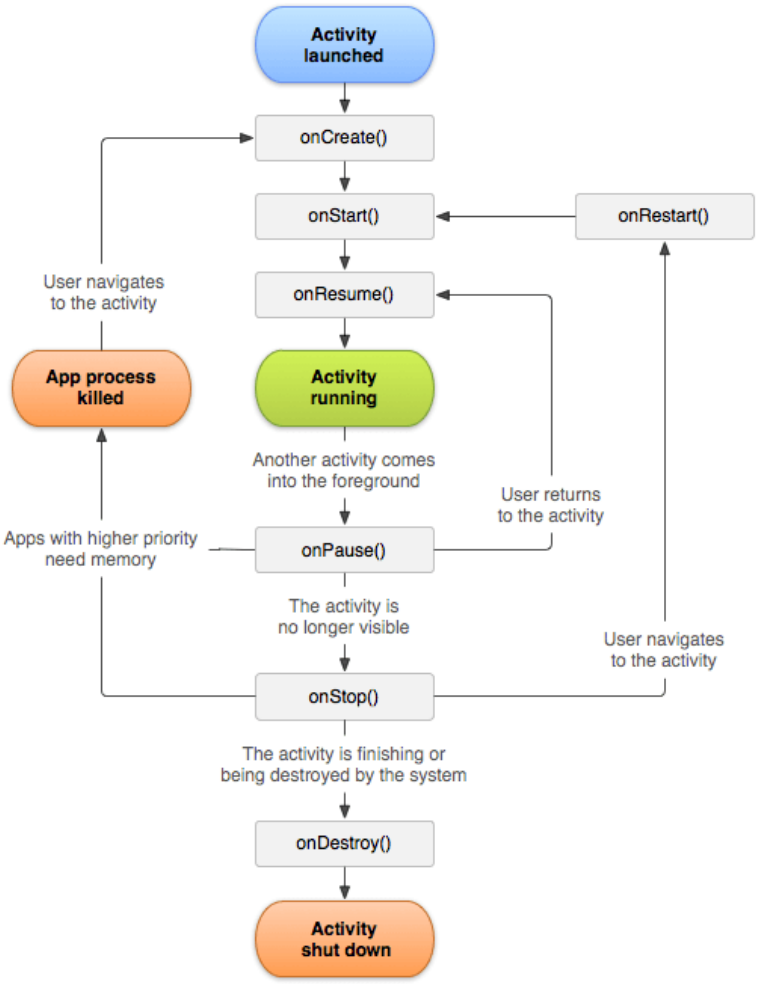
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#689FF1"
    tools:context=".SecondActivity">

    <EditText
        android:id="@+id/firstActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="serif-monospace"
        android:text="Second Activity"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

String.xml

```
<resources>
    <string name="app_name">ActivityLifeCycleDemoJJS-2</string>
</resources>
```

Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Fragments

History

- New concept of Android Honeycomb 3.0 – Android API version 11
- Provides modularity & reusability
-

Introduction

- A Fragment represents a reusable portion of your app's UI.
- Fragments cannot live on their own--they must be hosted by an activity or another fragment.
- A Fragment is a piece of an activity which enable more modular activity design.
- Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity.
- Fragments represent multiple screen inside one activity.
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
- Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity

Points to Note Down For Fragments

- 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.

Problem Before Fragment Introduction in Honeycomb 3.0 & – API 11

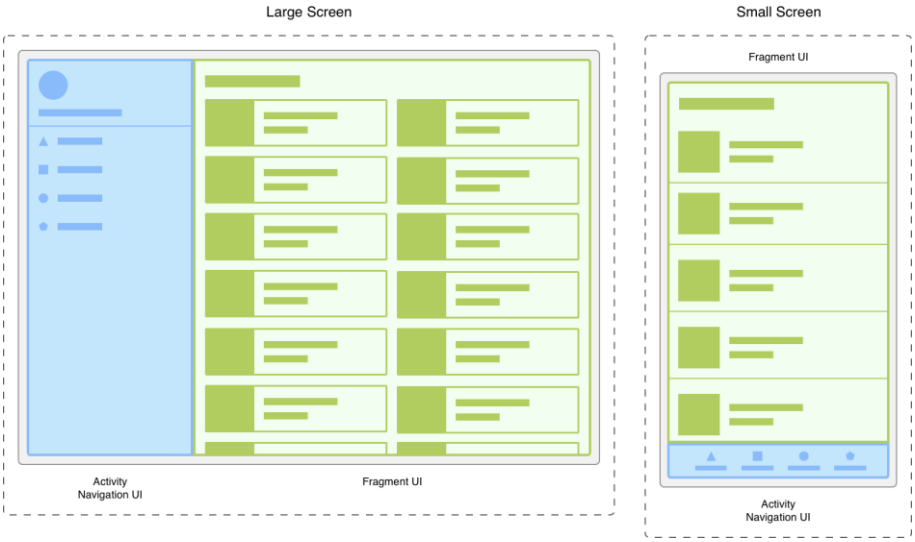
- Before fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time.
- So we were not able to divide device screen and control different parts separately.

Solution After Fragments

- After Fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time.
- Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.
- Fragments introduce modularity and reusability into your activity's UI by allowing you to divide the UI into discrete chunks.
- Fragments are better suited to define and manage the UI of a single screen or portion of a screen.

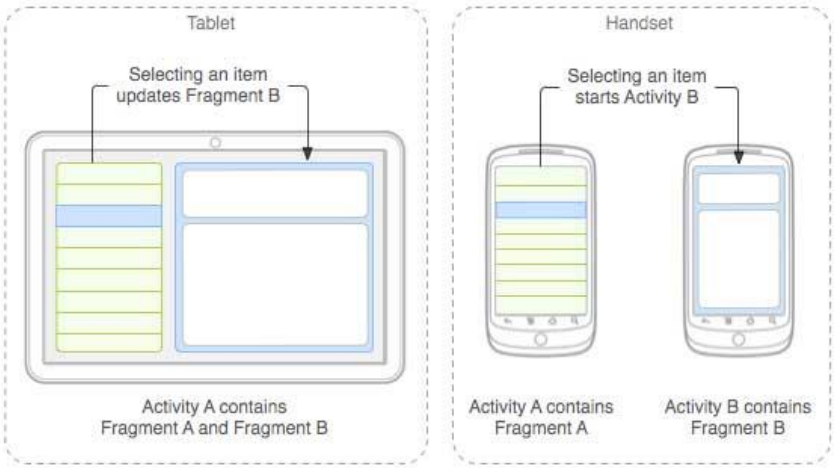
Examples For Fragment Concept

- Let's take an application that On larger screens, the app should display a static navigation drawer and a list in a grid layout.
- On smaller screens, the app should display a bottom navigation bar and a list in a linear layout.
- Separating the navigation elements from the content can make this process more manageable.
- The activity is then responsible for displaying the correct navigation UI while the fragment displays the list with the proper layout.



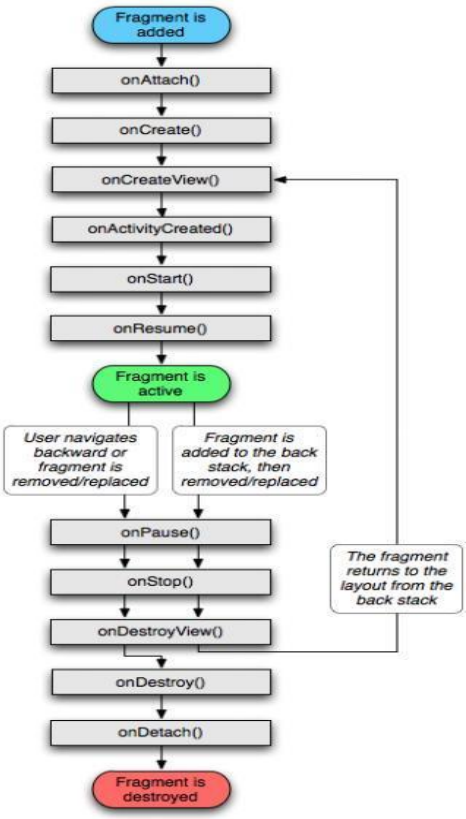
- In above figure On the left, a large screen contains a navigation drawer that is controlled by the activity and a grid list that is controlled by the fragment.
- On the right, a small screen contains a bottom navigation bar that is controlled by the activity and a linear list that is controlled by the fragment.

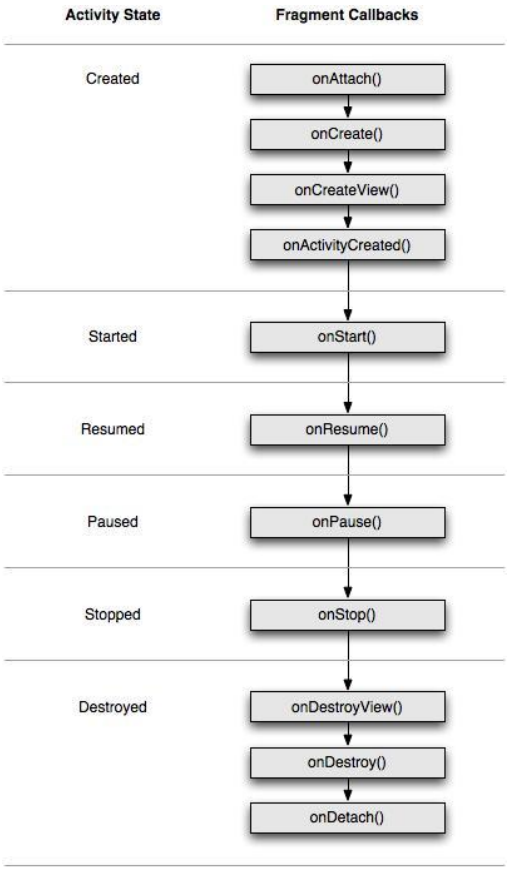
Ex 2 for Concept:



- Ex: The application can embed two fragments in Activity A, when running on a tablet-sized device.
- In Smartphone size device, no room for both fragments.
- So Activity A includes only the fragment for the list of articles,
- When the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Fragment Lifecycle





Add Fragment Statically in Activity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal">

<fragment
    android:id="@+id/frg1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:layout_weight="1"
    android:name="com.example.fragmentexample.Frg1"/>

</LinearLayout>
```

Steps

1. Create another layout xml file for fragment
2. Create fragment (By Extending the fragment class)
3. Set the layout xml file to fragment
4. Use fragment tag to include fragment in xml layout

1. Create Sample Blank fragment in xml layout: Fragment_sample.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".sample"
    android:background="@color/fragment_color">

</LinearLayout>
```

2. Sample.java

```
package com.example.simplefragmentdemo;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class sample extends Fragment {



    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_sample, container,
false);
    }
}
```

3. AcivityMain.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/activity_color">

    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/sampleFragment"
        android:name="com.example.simplefragmentdemo.sample"
        android:layout_margin="15dp"/>

</LinearLayout>
```

4. MainActivity.java

```
package com.example.simplefragmentdemo;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Not uSed Much for the developer

Example 1 Activity – 3 Fragments

1. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal">


    <fragment
        android:id="@+id/frg1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:name="com.example.fragmentexample.Frg1"/>

    <fragment
        android:id="@+id/frg2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:name="com.example.fragmentexample.Frg2"/>

    <fragment
        android:id="@+id/frg3"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:name="com.example.fragmentexample.Frg3"/>

</LinearLayout>
```

2. Fragment_Frg1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Frg1">

    <!-- TODO: Update blank fragment layout -->

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFC6C6"
        android:text="Learn" />

</FrameLayout>
```

3. Fragment_frg2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Frg2">

    <!-- TODO: Update blank fragment layout -->
```

```
        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="#AAC6C6"
            android:text="With" />

    </FrameLayout>
```

4. Fragment_frg3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Frg3">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Jenis Shah"
        android:background="#FAe606"/>
</FrameLayout>
```

5. Mainactivity.java

```
package com.example.fragmentexample;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    Button actbtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        actbtn = (Button) findViewById(R.id.actbtn);
        actbtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(),"On Click In
Activity",Toast.LENGTH_LONG).show();
            }
        });
    }


}
```

6. Frg1.java

```
package com.example.fragmentexample;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;


public class Frg1 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_frg1, container,
false);
    }
}
```

7. Frg2.java

```
package com.example.fragmentexample;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Frg2 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_frg2, container,
false);
    }
}
```

8. Frg3.java

```
package com.example.fragmentexample;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Frg3 extends Fragment {
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_frg3, container,
false);
    }
}
```

ADD – REMOVE – REPLACE FRAGMENTS DYNAMICALLY

Add Fragments

1. Create another layout xml file for fragment
2. Create fragment (By Extending the fragment class)
3. Set the layout xml file to fragment
4. Host a layout that will host a Fragment
5. Writing a code to add fragment to Activity in Activity.xml File

1. Fragment_Sample2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SampleFragment2"
    android:background="@color/teal_200"
    >

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/FragmentExample2" />

</FrameLayout>
```

2. SampleFragment2.java

```
package com.example.fragmentdemoexm;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class SampleFragment2 extends Fragment {
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_sample2,
container, false);
    }
}
```

3. ActivityMain.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/purple_200">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/fragmentContainer1"
        android:layout_margin="5dp"/>

</LinearLayout>
```

4. MainActivity.java

```
package com.example.fragmentdemoexm;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;
import android.os.Bundle;


public class MainActivity extends AppCompatActivity {

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

protected void addFragment(){

FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
SampleFragment2 sampleFragment2 = new SampleFragment2();
fragmentTransaction.add(R.id.fragmentContainer1,sampleFragment2);
fragmentTransaction.commit();

    }
}
```

ADD – Fragment Method Steps

Writing a code to add fragment to Activity in Activity - Java File

- Make a Method to Add Fragment
- Get Fragment Manager To Manage Fragment Transactions
- Create Fragment Transaction Objects
- Create Object of Sample Fragment Class
- Add Fragment Layout to Fragment Transaction
- Commit Transaction

```
protected void addFragment() {
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
SampleFragment2 sampleFragment2 = new SampleFragment2();
fragmentTransaction.add(R.id.fragmentContainer1 , sampleFragment2);
fragmentTransaction.commit();
}
```

Logs in Android

API for sending log output.

Generally, you should use the Log.v(), Log.d(), Log.i(), Log.w(), and Log.e() methods to write logs. You can then view the logs in logcat.

- The Log.e() method is used to log errors.
- The Log.w() method is used to log warnings.
- The Log.i() method is used to log informational messages.
- The Log.d() method is used to log debug messages.
- The Log.v() method is used to log verbose messages. : Involves more information than the standard or typical logging process.
- The Log.wtf() method is used to log terrible failures that should never happen. ("WTF" stands for "What a Terrible Failure!" of course.)

```
private static final String TAG = "MyApp";
Log.i(TAG, "I am logging something informational!");
```

Logging for exception :

```
try {
// ...
} catch (Exception exception) {
   Log.e(TAG, "Received an exception", exception);
}
```

Logging for Activity

```
public class MySimpleAppActivity extends Activity {
   private static final String TAG= "MySimpleAppLogging";
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Log.i(TAG, "Info about MySimpleAppActivity.");
   }
}
```

Activity Life Cycle & Fragment Lifecycle – Relation

To check the relation between activity & fragment lifecycle we will implement all override methods of both. But for that just add the fragment by using any method static or dynamic.

We will do that with dynamic add fragment method by implementing add Fragment method in mainactivity.java class.

So the code should be as below.

1. Fragment_sample1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SampleFragment1"
    android:background="@color/teal_700">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Fragment1-1" />


</FrameLayout>
```

2. SampleFragment1.java

```
package com.example.activityfragmentlifecyclerelation;

import android.content.Context;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SampleFragment1 extends Fragment {

    private static final String Fragment_Name =
SampleFragment1.class.getSimpleName();
    private static final String TAG = Fragment_Name;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i(TAG,"ON Create Called");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        Log.i(TAG,"ON CreateView Called");
        return inflater.inflate(R.layout.fragment_sample1, container,
false);
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.i(TAG,"ON Start Called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.i(TAG,"ON Pause Called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.i(TAG,"ON Resume Called");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.i(TAG,"ON Detach Called");
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.i(TAG,"ON DestroyView Called");
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        Log.i(TAG,"ON Attach Called");
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        Log.i(TAG,"ON ActivityCreated Called");
    }

    @Override
    public void onStop() {
        super.onStop();
```

```
            Log.i(TAG,"ON Stop Called");
        }

        @Override
        public void onDestroy() {
            super.onDestroy();
            Log.i(TAG,"ON Destroy Called");
        }

    }
```

3. Activity_main.xml

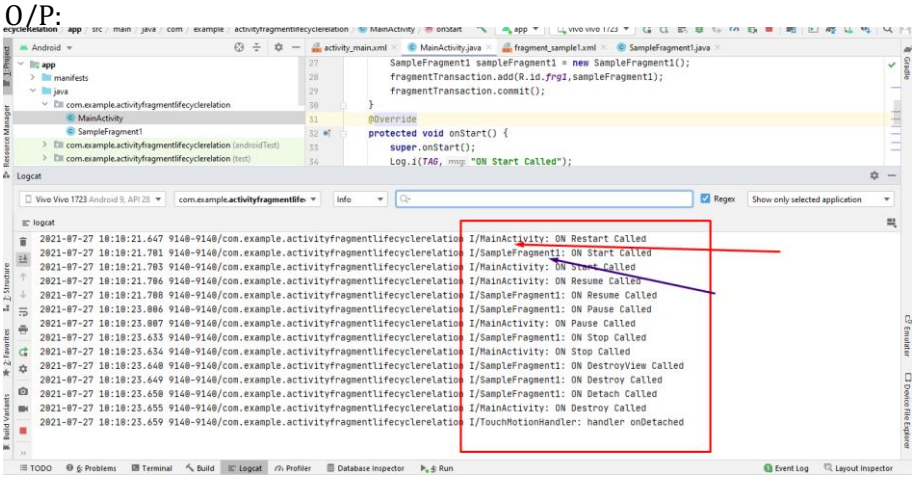
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/purple_700">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="15dp"
        android:id="@+id/frg1"/>

</LinearLayout>
```

4. MainActivity.java

```
package com.example.activityfragmentlifecyclerelation;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private static final String Activity_Name =
MainActivity.class.getSimpleName();
    private static final String TAG = Activity_Name;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i(TAG,"ON Create Called");
        addFragment();
    }

    protected  void addFragment()
    {
        FragmentManager fragmentManager = getSupportFragmentManager();
```

```
        FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
        SampleFragment1 sampleFragment1 = new SampleFragment1();
        fragmentTransaction.add(R.id.frg1,sampleFragment1);
        fragmentTransaction.commit();
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.i(TAG,"ON Start Called");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.i(TAG,"ON Resume Called");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.i(TAG,"ON Pause Called");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.i(TAG,"ON Stop Called");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.i(TAG,"ON Restart Called");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i(TAG,"ON Destroy Called");
    }
}
```

O/P:



Add Fragment Using Button IN Activity

Code of above program is as it is. We just need to do following changes in files.

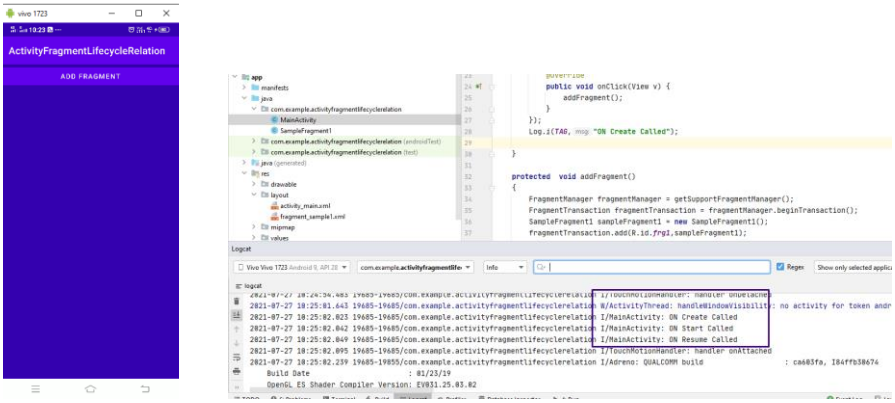
1. In MainActivity.Java (Changes in On Create Method and add one button variable in the beginning)

```
private Button buttonAddFragment;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    buttonAddFragment = findViewById(R.id.buttonAddFrg);
    buttonAddFragment.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            addFragment();
        }
    });
    Log.i(TAG,"ON Create Called");

}
```

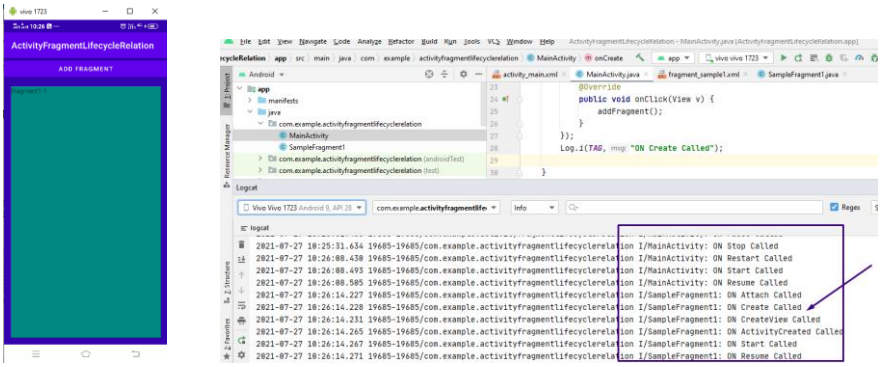
2. In Activity Main. Xml – add one button over there

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:background="@color/purple_700">

      <Button
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:id="@+id/buttonAddFrg"
          android:text="Add Fragment"/>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="15dp"
        android:id="@+id/frg1"/>

</LinearLayout>
```

O/P



(IN the beginning Before clicking on Add Fragment Button)



(After clicking on Add Fragment Button – Methods of Fragment have been also called)

BackStack in Fragments

```
protected  void addFragment()
{
    FragmentManager fragmentManager = getSupportFragmentManager();
    FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
    SampleFragment1 sampleFragment1 = new SampleFragment1();
    fragmentTransaction.add(R.id.frg1,sampleFragment1);
    fragmentTransaction.addToBackStack("frg"); //code
to add fragment to the back stack
    fragmentTransaction.commit();
}
```

With this : when click on Back button – first fragment will remove then by clicking on Back button again – then and only then Activity will remove

Motto : We will create 3 – fragments then add them into backstack and try to check how backstack works.

1. Fragment_Sample1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SampleFragment1"
    android:orientation="vertical"
    android:gravity="center"
    android:background="@color/frg1_color">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="80sp"
        android:layout_gravity="center_vertical|center_horizontal"
        android:text="1" />

</LinearLayout>
```

2. Fragment_Sample2.xml

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

```
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SampleFragment2"
    android:orientation="vertical"
    android:gravity="center"
    android:background="@color/frg2_color">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="80sp"
        android:layout_gravity="center_vertical|center_horizontal"
        android:text="2" />


</LinearLayout>
```

3. Fragment_Sample3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SampleFragment3"
    android:orientation="vertical"
    android:gravity="center"
    android:background="@color/frg3_color">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="80sp"
        android:layout_gravity="center_vertical|center_horizontal"
        android:text="3" />


</LinearLayout>
```

4. SampleFragment1.java

```
package com.example.fragmentwithbackstack;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
public class SampleFragment1 extends Fragment {



    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_sample1, container,
false);
    }
}
```

5. SampleFragment2.java

```
package com.example.fragmentwithbackstack;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SampleFragment2 extends Fragment {



    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_sample2, container,
false);
    }
}
```

6. SampleFragment3.java

```
package com.example.fragmentwithbackstack;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;


public class SampleFragment3 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
```

```
                                Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_sample3, container,
false);
    }
}
```

7. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/teal_700"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/textCount"
        android:text="Count is = "/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/buttonAddFrg"
        android:text="Add Fragment"/>
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="15dp"
        android:id="@+id/fragment_container"/>
</LinearLayout>
```

8. Mainactivity.java

```
package com.example.fragmentwithbackstack;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private static final String Activity_Name =
MainActivity.class.getSimpleName();
    private static final String TAG = Activity_Name;
    private Button buttonAddFragment;
```

```
    private TextView textViewCount;
    FragmentManager fragmentManager;
    FragmentTransaction fragmentTransaction;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        buttonAddFragment = findViewById(R.id.buttonAddFrg);
        textViewCount = findViewById(R.id.textCount);
        fragmentManager = getSupportFragmentManager();
        textViewCount.setText("Fragment count in back stack
"+fragmentManager.getBackStackEntryCount());
        fragmentManager.addOnBackStackChangedListener(new
FragmentManager.OnBackStackChangedListener() {
            @Override
            public void onBackStackChanged() {
                textViewCount.setText("Fragment Count in back stack
"+fragmentManager.getBackStackEntryCount());
            }
        });
        buttonAddFragment.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addFragment();
            }
        });
        Log.i(TAG,"ON Create Called");

    }


    @Override
    protected void onStart() {
        super.onStart();
        Log.i(TAG,"ON Start Called");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.i(TAG,"ON Resume Called");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.i(TAG,"ON Pause Called");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.i(TAG,"ON Stop Called");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.i(TAG,"ON Restart Called");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
```

```
            Log.i(TAG,"ON Destroy Called");
        }

        protected  void addFragment()
        {
            Fragment fragment;
            switch (fragmentManager.getBackStackEntryCount()){
                case 0: fragment = new SampleFragment1(); break;
                case 1: fragment = new SampleFragment2(); break;
                case 2: fragment = new SampleFragment3(); break;
                default: fragment = new SampleFragment1(); break;
            }
            fragmentTransaction = fragmentManager.beginTransaction();
            fragmentTransaction.add(R.id.fragment_container,fragment);
            fragmentTransaction.addToBackStack(null);
            fragmentTransaction.commit();
        }
    }
```

CODE FOR Backstack:

```
fragmentManager.addOnBackStackChangedListener(new
FragmentManager.OnBackStackChangedListener() {
    @Override
    public void onBackStackChanged() {
        textViewCount.setText("Fragment Count in back stack
"+fragmentManager.getBackStackEntryCount());
    }
});
```

Code for Add Fragment

```
protected  void addFragment()
{
    Fragment fragment;
    switch (fragmentManager.getBackStackEntryCount()){
        case 0: fragment = new SampleFragment1(); break;
        case 1: fragment = new SampleFragment2(); break;
        case 2: fragment = new SampleFragment3(); break;
        default: fragment = new SampleFragment1(); break;
    }
    fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.add(R.id.fragment_container,fragment);
    fragmentTransaction.addToBackStack(null);
    fragmentTransaction.commit();
}
```

REMOVE FRAGMENT

1. In above code – Comment the addToBackstack() line
   // fragmentTransaction.addToBackStack(null); & Run the code…

   So now : When you press back button from any fragment : App will directly get closed and No Back Stack Count will be maintained as no fragment will be added in back stack
2. Add A Method : onBackPressed() // TO Remove Fragment

```
@Override
public void onBackPressed(){
    Fragment fragment =
fragmentManager.findFragmentById(R.id.fragment_container);
    if(fragment!=null)
    {
        fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.remove(fragment);
        fragmentTransaction.commit();
    }
    else {
            super.onBackPressed();
        }
}
```

Now run the program and check the output by clicking on the add fragment and then back button…

As you can click on the Add Fragment Button : Only Fragment 1 will be added one above another

So, to get Fragment 1 – 2 – 3 – By clicking on add button we need to edit code of the previous addFragment Method()

PREVIOUS addFragment () Code :

```
protected  void addFragment()
{
    Fragment fragment;
    switch (fragmentManager.getBackStackEntryCount()){
        case 0: fragment = new SampleFragment1(); break;
        case 1: fragment = new SampleFragment2(); break;
        case 2: fragment = new SampleFragment3(); break;
        default: fragment = new SampleFragment1(); break;
    }
    fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.add(R.id.fragment_container,fragment);
```

```
//    fragmentTransaction.addToBackStack(null);
    fragmentTransaction.commit();
}
```

NEW-EDITED addFragment() Method Code

```
protected  void addFragment()
{
    Fragment fragment;
    fragment = fragmentManager.findFragmentById(R.id.fragment_container);

      fragment = fragmentManager.findFragmentById(R.id.fragment_container);
      if(fragment instanceof  SampleFragment1){
          fragment = new SampleFragment2();
      }
      else if(fragment instanceof  SampleFragment2)
      {
          fragment = new SampleFragment3();
      }
      else if(fragment instanceof  SampleFragment3)
      {
          fragment = new SampleFragment1();
      }
      else{
          fragment = new SampleFragment1();
      }

     fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.add(R.id.fragment_container,fragment);
    fragmentTransaction.commit();
}
```

asdasdasā

Code For Replacement of Fragment Instead of Removal

```
protected  void addFragment()
{
    Fragment fragment;
     fragment = fragmentManager.findFragmentById(R.id.fragment_container);
    if(fragment instanceof  SampleFragment1){
        fragment = new SampleFragment2();
    }
    else if(fragment instanceof  SampleFragment2)
    {
        fragment = new SampleFragment3();
    }
    else if(fragment instanceof  SampleFragment3)
    {
        fragment = new SampleFragment1();
    }
    else{
        fragment = new SampleFragment1();
    }
```

```
    fragmentTransaction = fragmentManager.beginTransaction();
   fragmentTransaction.replace(R.id.fragment_container,fragment);

    fragmentTransaction.commit();
}
```

Intents

Formatted: Right: 0 cm, Tabstops: Not at 1.25 cm + 1.5 cm

Formatted: Font: Bold

Intents in Android

- Have you ever do jump from one app to another?
- Like : Searching for a location on browser & direct jump to Map application
- Like: Receiving payment link in SMS and by clicking on it Jump to PayTM or Google Pay App
- This process from Jumping one app to another is done by Passing Intent.

- Android Intent is the message that is passed between components such as activities, content providers, broadcast receivers, services etc.
- Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also.
- Intent are used for communicating between the Application components and it also provides the connectivity between two apps.

Intents are mainly Used for

- Start the service
- Sending user to another app
- Getting Result from the Activity
- Allowing apps to start Activity
- Dial a call
- Take user to camera to take Picture
- Take user to URL within android web browser
- Broad cast a message
- Display the list of contacts etc…

IMP Usages of Intent

- Intent for Activity :
  To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

- Intent for Services:
  Intents could be used to start a Service that performs one-time task like download some file. or starting a Service you need to pass Intent to startService() method.

- Intent For Broadcast Receiver :
  Intents can be used to send broadcast messages into the Android system. A broadcast receiver can register to an event and is notified if such an event is sent.

Structure of Intent

Two important things are there in intent :

1. Action: The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
2. Data: The data to operate on, such as a person record in the contacts database, expressed as a Uri

Examples:

- ACTION_VIEW content://contacts/people/1 -- Display information about the person whose identifier is "1".
- ACTION_DIAL content://contacts/people/1 -- Display the phone dialer with the person filled in.
- ACTION_VIEW tel:123 -- Display the phone dialer with the given number filled in
- ACTION_DIAL tel:123 -- Display the phone dialer with the given number filled in.
- ACTION_EDIT content://contacts/people/1 -- Edit information about the person whose identifier is "1".
- ACTION_VIEW content://contacts/people/ -- Display a list of people, which the user can browse through.
-

With this The Secondary Attributes are as below:

- Category -- Gives additional information about the action to execute. Ex: CATEGORY_LAUNCHER : appear in launcher
- Type: - Specifies an explicit type (a MIME type) of the intent data.

- component -- Specifies an explicit name of a component class to use for the intent.
- Extras -- This is a Bundle of any additional information. This can be used to provide extended information to the component

Types of Intent

1. Implicit Intent :
   Implicit Intent doesn't specify the component. In such a case, intent provides information on available components provided by the system that is to be invoked.
   (With following code you can view the webpage)

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.ljku.edu.in/"));
startActivity(intentObj)
```

2. Explicit intents:
   - specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name
   - In Explicit we use the name of component which will be affected by Intent.
   - Explicit Intent work internally within an application to perform navigation and data transfer.

   (How you can navigate from one activity to another activity)

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

Program for Implicit and Explicit Intent

We are going to create a Screen – 1 with EditText & 2 Buttons. By clicking on 1st button user will redirect to the entered URL. (This is Implicit Intent)

2nd button – By clicking on this button User will redirect to the second activity.

As well as Screen – 2 with Button and

1. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <EditText
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/editTextData"
        android:layout_marginTop="100dp"
        android:layout_marginLeft="20dp"
        android:ems="10"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/implicit_button"
        android:text="Click Here"
        android:layout_marginTop="120dp"
        android:layout_marginLeft="50dp"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/explicit_button"
        android:text="Click To Go Second"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="50dp"
        />
</LinearLayout>
```

2. Activiy_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity"
    android:background="@color/purple_200">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is Second Activity"
        android:id="@+id/second_text_view"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="50dp"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/second_button"
        android:text="Click To Go First"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="20dp"
        />
</LinearLayout>
```

3. MainActivity.java

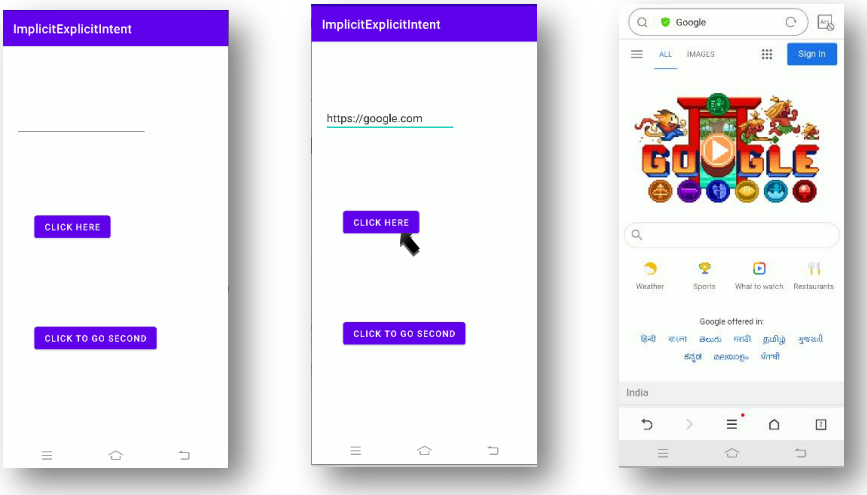
```
package com.example.implicitexplicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
    private Button implicitButton, explicitButton;
    private EditText URLtext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        URLtext = findViewById(R.id.editTextData);
        implicitButton = findViewById(R.id.implicit_button);
        explicitButton = findViewById(R.id.explicit_button);
        // code for implicit intent
        implicitButton.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                String url = URLtext.getText().toString();
                Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse(url));
```

```
                startActivity(intent);
            }
        });
        // code for explicit intent
        explicitButton.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
SecondActivity.class);
                startActivity(intent);
            }
        });

    }
}
```

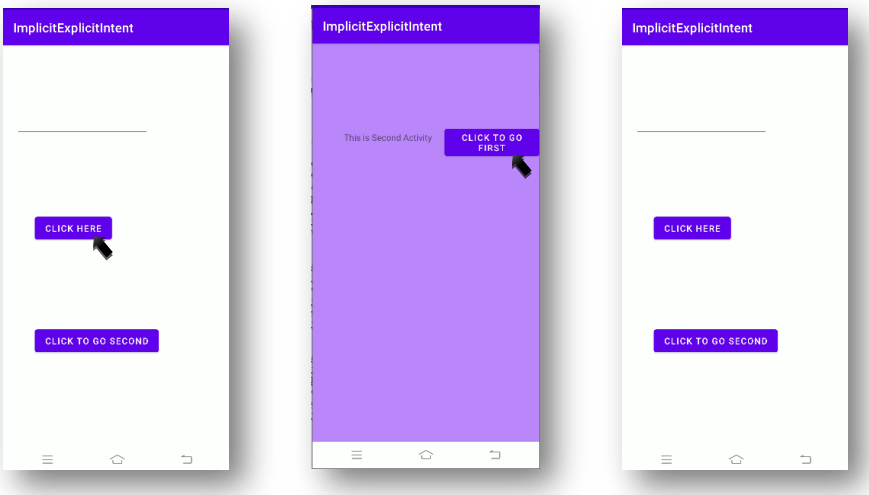
4. SecondActivity.java

```
package com.example.implicitexplicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class SecondActivity extends AppCompatActivity {

    private Button secondBtn; ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        secondBtn = findViewById(R.id.second_button);
        secondBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
MainActivity.class);
                startActivity(intent);
            }
        });

    }
}
```

Implicit intent Example O/P



Explicit Intent Example O/P



2nd Example of Intents

MainActivity.java

```
package com.example.implicitexplicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
    private Button implicitButton, explicitButton;
    private EditText URLtext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        URLtext = findViewById(R.id.editTextData);
        implicitButton = findViewById(R.id.implicit_button);
        explicitButton = findViewById(R.id.explicit_button);
        // code for implicit intent
        implicitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String url = URLtext.getText().toString();
                Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse(url));
                startActivity(intent);
            }
        });
        // code for explicit intent
        explicitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
SecondActivity.class);
                startActivity(intent);
            }
        });

    }
}
```

Activity_Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
    <EditText
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/editTextData"
        android:layout_marginTop="100dp"
        android:layout_marginLeft="20dp"
        android:ems="10"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/implicit_button"
        android:text="Click Here"
        android:layout_marginTop="120dp"
        android:layout_marginLeft="50dp"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/explicit_button"
        android:text="Click To Go Second"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="50dp"
        />
</LinearLayout>
```

Second Activity.java

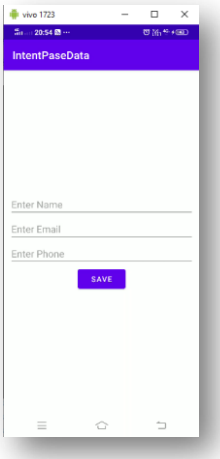
```
package com.example.implicitexplicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class SecondActivity extends AppCompatActivity {

    private Button secondBtn; ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        secondBtn = findViewById(R.id.second_button);
        secondBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
MainActivity.class);
                startActivity(intent);
            }
        });
```

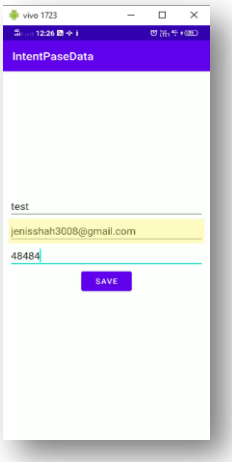
```
    }
}
```

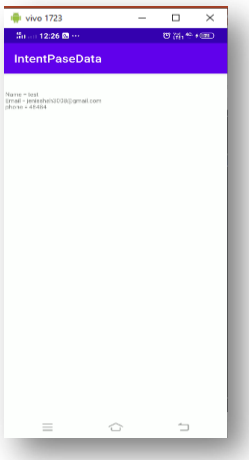
Activity_Second.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity"
    android:background="@color/purple_200">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is Second Activity"
        android:id="@+id/second_text_view"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="50dp"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/second_button"
        android:text="Click To Go First"
        android:layout_marginTop="130dp"
        android:layout_marginLeft="20dp"
        />
</LinearLayout>
```

O/P







LAYOUTS IN ANDROID

- A layout defines the structure for a user interface in your app, such as in an activity.
- Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen.
- 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.
- Whereas a **ViewGroup** is an invisible container that defines the layout structure for View and other **ViewGroup**

View

- A View is a simple building block of a user interface.
- It is a small rectangular box that can be TextView, EditText, or even a button.
- **Usage :** It occupies the area on the screen in a rectangular area and is responsible for drawing and event handling.
- The use of a view is to draw content on the screen of the user's Android device.
- All of the views in a window are arranged in a single tree.
- **Implementation :** You can add views either from code or by specifying a tree of views in one or more XML layout files.

Types of Views

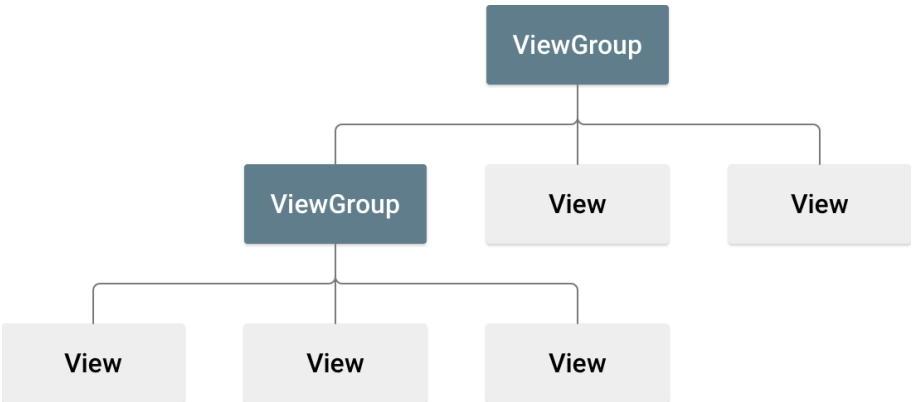
1. TextView
2. EditText
3. Button
4. Image Button
5. Date Picker
6. RadioButton
7. CheckBox buttons
8. Image View

View Group

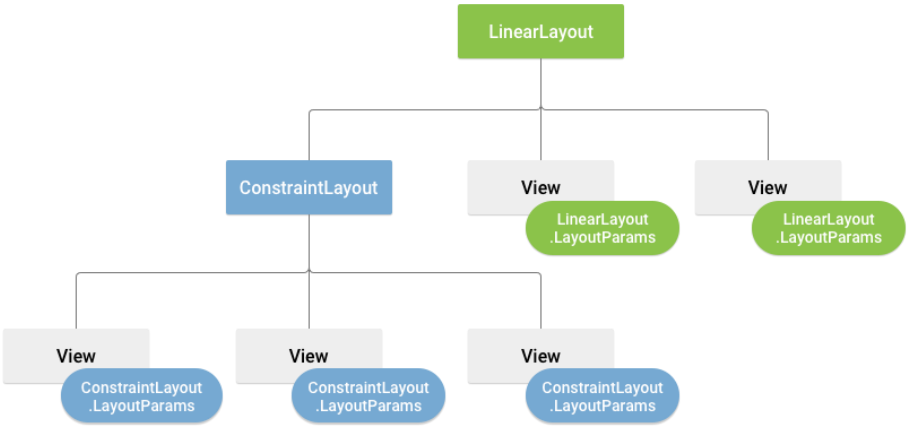
- A View Group is a subclass of the ViewClass and can be considered as a superclass of Layouts.
- It provides an invisible container to hold the views or layouts.
- ViewGroup instances and views work together as a container for Layouts
- The subclasses of the ViewGroup:
  1. LinearLayout
  2. RelativeLayout
  3. FrameLayout
  4. GridView
  5. ListView

Note: The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView.

The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.



(View and ViewGroup Hierarchy)



(Example of View & ViewGroup Hierarchy)

Layout Declaration

- Declare UI elements in XML
  - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.
- Instantiate Layout Element at Runtime
  - Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Types of Layouts

- Linear Layout
- Relative Layout
- Constraint Layout
- Table Layout
- Frame Layout
- List View
- Grid View
- Absolute Layout
- WebView
- ScrollView

Ex of Code

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a Button" />
</LinearLayout>
```

Attributes of Layout in Android:

- android:id: It uniquely identifies the Android Layout.
- android:hint: It shows the hint of what to fill inside the EditText.
- android:layout_height: It sets the height of the layout.
- android:layout_width: It sets the width of the layout.
- android:layout_gravity: It sets the position of the child view.
- android:layout_marginTop: It sets the margin of the from the top of the layout.
- android:layout_marginBottom: It sets the margin of the from the bottom of the layout.
- android:layout_marginLeft: It sets the margin of the from the left of the layout.
- android:layout_marginRight: It sets the margin of the from the right of the layout.
- android:layout_x: It specifies the x coordinates of the layout.
- android:layout_y: It specifies the y coordinates of the layout.

1. LinearLayout:

- We use this layout to place the elements in a linear manner.
- A Linear manner means one element per line.
- A LinearLayout aligns each of the child View in either a vertical or a horizontal line.
- A vertical layout has a column of Views, whereas in a horizontal layout there is a row of Views.
- It supports a weight attribute for each child View that can control the relative size of each child View within the available space.
- This layout creates various kinds of forms on Android.

a. Vertical
In this all the child are arranged vertically in a line one after the other.

b. Horizontal
In this all the child are arranged horizontally in a line one after the other.

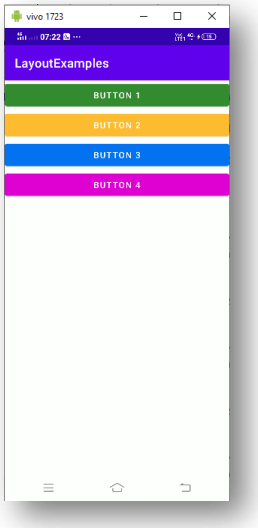
Example of Linear Layout Vertical

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#FFBA32" />

    <Button
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:id="@+id/btn3"
        android:backgroundTint="#0472f2" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 4"
        android:id="@+id/btn4"
        android:backgroundTint="#e100d5" />

</LinearLayout>
```

O/P



Example of Linear Layout Horizontal

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#FFBA32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:id="@+id/btn3"
        android:backgroundTint="#0472f2" />

</LinearLayout>
```

O/P

Main Attributes of Linear Layout

1. Orientation: The orientation attribute used to set the childs/views horizontally or vertically. In Linear layout default orientation is vertical.
   Ex : Orientation Vertical :

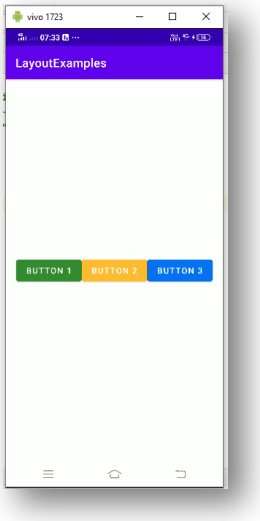
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"> <!-- Vertical Orientation set -->
    <!-- Put Child Views like Button here -->
</LinearLayout>
```

Ex: Orientation Horizontal

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"> <!-- Horizontal Orientation set -->
    <!-- Child Views are here -->
</LinearLayout>
```

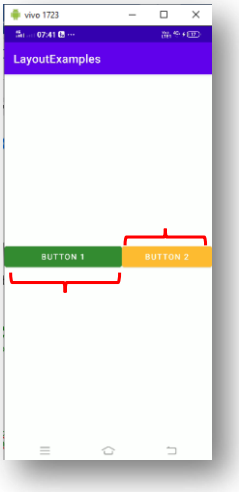
2. Gravity: The gravity attribute is an optional attribute which is used to control the alignment of the layout like left, right, center, top, bottom etc.
   Android:gravity = center

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal"
    android:gravity="center">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#FFBA32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:id="@+id/btn3"
        android:backgroundTint="#0472f2" />

</LinearLayout>
```



3. layout_weight: specifies how much of the extra space in the layout to be allocated to the View

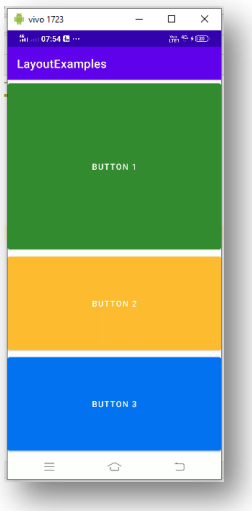
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal"
    android:gravity="center">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32"
        android:layout_weight="2"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#FFBA32"
        android:layout_weight="1"/>

  </LinearLayout>
```



4. weightSum: weightSum is the sum up of all the child attributes weight. This attribute is required if we define weight property of the childs.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:weightSum="4"
    >

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32"
        android:layout_weight="2"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#FFBA32"
        android:layout_weight="1"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
```

```
        android:id="@+id/btn3"
        android:layout_weight="1"
        android:backgroundTint="#0472f2" />

</LinearLayout>
```



Gravity : android:gravity attribute is used to arrange the position of the content inside a View (Ex: text inside the button widget)

Layout Gravity : android:layout_gravity It is used to arrange the position of the entire View relative to it's container(Parent).

2. Relative Layout:
   - The Relative Layout is very flexible layout used in android for custom layout designing.
   - It allow its child view to position relative to each other or relative to the container or another container.
   - This layout is for specifying the position of the elements in relation to the other elements that are present there.
   - 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 (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).

- A RelativeLayout is a very powerful utility for designing a user interface.
- Because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance.

Attributes of Relative Layout

Attribute	Description
layout_alignParentTop	If it specified "true", the top edge of view will match the top edge of the parent.
layout_alignParentBottom	If it specified "true", the bottom edge of view will match the bottom edge of parent.
layout_alignParentLeft	If it specified "true", the left edge of view will match the left edge of parent.
layout_alignParentRight	If it specified "true", the right edge of view will match the right edge of the parent.
layout_centerInParent	If it specified "true", the view will be aligned to the centre of parent.
layout_centerHorizontal	If it specified "true", the view will be horizontally centre aligned within its parent.
layout_centerVertical	If it specified "true", the view will be vertically centre aligned within its parent.
layout_above	It accepts another sibling view id and places the view above the specified view id.
layout_below	It accepts another sibling view id and places the view below the specified view id.
layout_toLeftOf	It accepts another sibling view id and places the view left of the specified

	view id.
layout_toRightOf	It accepts another sibling view id and places the view right of the specified view id.
layout_toStartOf	It accepts another sibling view id and places the view to start of the specified view id.
layout_toEndOf	It accepts another sibling view id and places the view to the end of the specified view id.

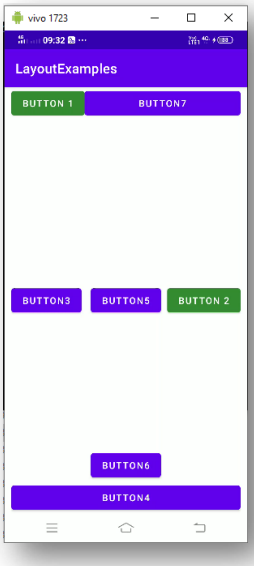
Example:

Relative_Layout_Example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RelativeLayoutExample"
    android:paddingLeft="10dp"
    android:paddingRight="10dp">


    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:id="@+id/btn1"
        android:backgroundTint="#358a32"
       android:layout_alignParentLeft="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:id="@+id/btn2"
        android:backgroundTint="#358a32"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"/>
    <Button
        android:id="@+id/btn3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:text="Button3" />

    <Button
```

```
        android:id="@+id/btn4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Button4" />
    <Button
        android:id="@+id/btn5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/btn2"
        android:layout_centerHorizontal="true"
        android:text="Button5" />
    <Button
        android:id="@+id/btn6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/btn4"
        android:layout_centerHorizontal="true"
        android:text="Button6" />
    <Button
        android:id="@+id/btn7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toEndOf="@+id/btn1"
        android:layout_toRightOf="@+id/btn1"
        android:layout_alignParentRight="true"
        android:text="Button7" />

</RelativeLayout>
```

(There is no change in default java file of this layout file)



Constrain Layout

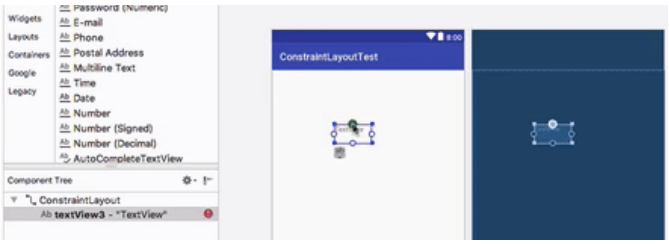
- Similar to Relative Layout
- Constraint Layout is a ViewGroup (i.e. a view that holds other views) which allows you to create large and complex layouts with a flat view hierarchy.
- It allows you to position and size widgets in a very flexible way.
- It was created to help reduce the nesting of views and also improve the performance of layout files.
- All the power of ConstraintLayout is available directly from the Layout Editor's visual tools.
- So you can build your layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.
- To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.
- Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
- Each constraint defines the view's position along either the vertical or horizontal axis.
- When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. if a view has no constraints when you run your layout on a device, it is drawn at position [0,0] (the top-left corner).

Advantages of Constrain Layout

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

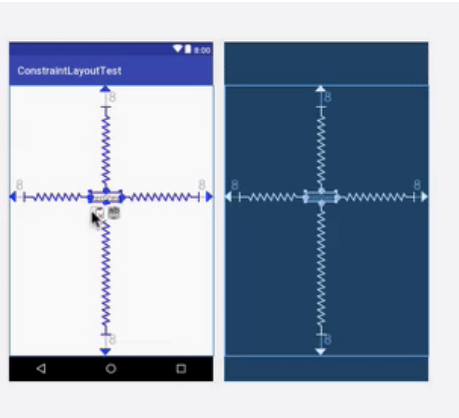
Handle Anchor Points in Constrain View

Suppose you drag a TextView element in ConstraintLayout visual editor of Android Studio. Immediately after dragging you will notice a error with a message, "This view is not constrained…" So this simply means the view we created is not Constrained and we need to fix it.

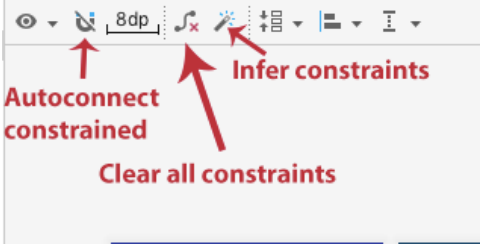


Bias decides view placement between its constraints on an axis. By default it is set 50% and can be changed easily by dragging.

Delete a constrain



Tools in Constrain layout



Relative Positioning in constrain view

- Relative Positioning is the most important type of Constraint Layout and considered as the basic block building in it.
- The different constraint option it offers works in relation/relative to position of one another.
- Those relative positioning works only in vertical and horizontal axis only.

- Using horizontal axis, you can set positioning of one widget in right, left, end and start sides of other widget.

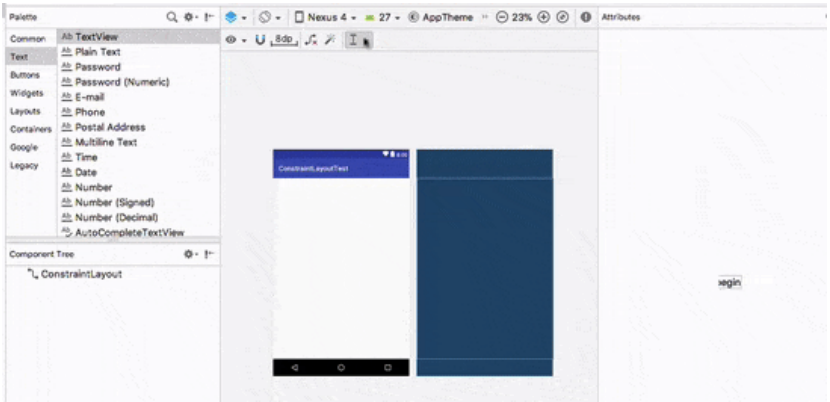
- While using vertical axis you can set bottom, top sides and text baseline.

Chain in Constrain Layout

Chains allow us to control the space between elements and chains all the selected elements to another.

To create a chain, select the elements that you want to form part of the chain, and then right click – "Chain" – "Create Horizontal or Vertical Chain".

Use Guidelines



Group in Constrain Layout

Group in android helps to carry out some actions on a set of widgets with the most common case being to control the visibility of a collection of widget.

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.
- 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

Imp Points

- 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.
- We cannot specify the width of the children's of the Table layout.
- Here, width always match parent width.
- The height attribute can be defined by a child; default value of height attribute is wrap content.

Basic Code

```
<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="match_parent"
        android:layout_height="wrap_content">

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

    </TableRow>
</TableLayout>
```

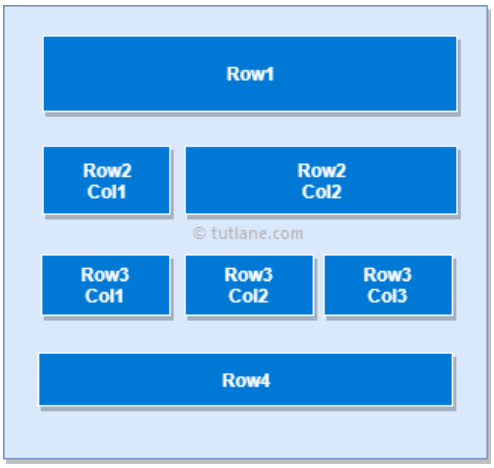


Table Layout Example

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Constrain2">

    <TableRow android:background="#0079D6" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="UserId" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="User Name" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Location" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="1" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Suresh Dasari" />
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Hyderabad" />
    </TableRow>

    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="2" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Rohini Alavala" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Guntur" />
    </TableRow>

    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="3" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Trishika Dasari" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Guntur" />
    </TableRow>

</TableLayout>
```

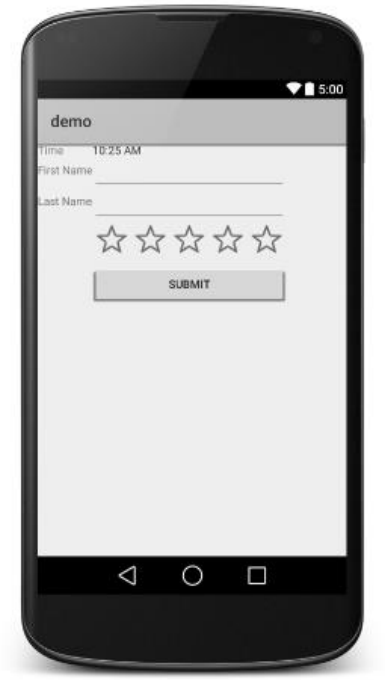
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:collapseColumns** This specifies the zero-based index of the columns to collapse. The

	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

Table Layout Example 2

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <TextView
         android:text="Time"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />

      <TextClock
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/textClock"
         android:layout_column="2" />

   </TableRow>

   <TableRow>

      <TextView
         android:text="First Name"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />

      <EditText
         android:width="200px"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
```

```
    </TableRow>

    <TableRow>

       <TextView
          android:text="Last Name"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_column="1" />

       <EditText
          android:width="100px"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />
    </TableRow>

    <TableRow
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">

       <RatingBar
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/ratingBar"
          android:layout_column="2" />
    </TableRow>

    <TableRow
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"/>

    <TableRow
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">

       <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Submit"
          android:id="@+id/button"
          android:layout_column="2" />
    </TableRow>

</TableLayout>
```



ShrinkColumn Example

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="0"> <!-- shrink the first column of the layout-->


    <!-- first row of the table layout-->
    <TableRow
        android:id="@+id/firstRow"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- first element of the first row-->
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#b0b0b0"
            android:padding="18dip"
            android:text="Shrink Column Example"
            android:textColor="#000"
            android:textSize="18dp" />

    </TableRow>
</TableLayout>
```

Collapse Column Example

```
<?xml version="1.0" encoding="utf-8"?>
<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/simpleTableLayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- first element of the row that is the part of table but it is
invisible-->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#b0b0b0"
            android:padding="18dip"
            android:text="Columns 1"
            android:textColor="#000"
```

```
            android:textSize="18dp" />

        <!-- second element of the row that is shown in the screenshot-->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#b0b0b0"
            android:padding="18dip"
            android:text="Columns 2"
            android:textColor="#000"
            android:textSize="18dp" />
    </TableRow>
</TableLayout>
```

Another Examples : Login Screen

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000"
    android:orientation="vertical"
    android:stretchColumns="1">

    <TableRow android:padding="5dip">

        <TextView
            android:layout_height="wrap_content"
            android:layout_marginBottom="20dp"
            android:layout_span="2"
            android:gravity="center_horizontal"
            android:text="@string/loginForm"
            android:textColor="#0ff"
            android:textSize="25sp"
            android:textStyle="bold" />
    </TableRow>
```

```
    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_marginLeft="10dp"
            android:text="@string/userName"
            android:textColor="#fff"
            android:textSize="16sp" />

        <EditText
            android:id="@+id/userName"
            android:layout_height="wrap_content"
            android:layout_column="1"
            android:layout_marginLeft="10dp"
            android:background="#fff"
            android:hint="@string/userName"
            android:padding="5dp"
            android:textColor="#000" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_marginLeft="10dp"
            android:layout_marginTop="20dp"
            android:text="@string/password"
            android:textColor="#fff"
            android:textSize="16sp" />

        <EditText
            android:id="@+id/password"
            android:layout_height="wrap_content"
            android:layout_column="1"
```

```
            android:layout_marginLeft="10dp"
            android:layout_marginTop="20dp"
            android:background="#fff"
            android:hint="@string/password"
            android:padding="5dp"
            android:textColor="#000" />
    </TableRow>


    <TableRow android:layout_marginTop="20dp">

        <Button
            android:id="@+id/loginBtn"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_span="2"
            android:background="#0ff"
            android:text="@string/login"
            android:textColor="#000"
            android:textSize="20sp"
            android:textStyle="bold" />
    </TableRow>
</TableLayout>
```

Main Activity.java

n button and display the message by using a Toast.

```
package example.abhiandriod.tablelayoutexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
```

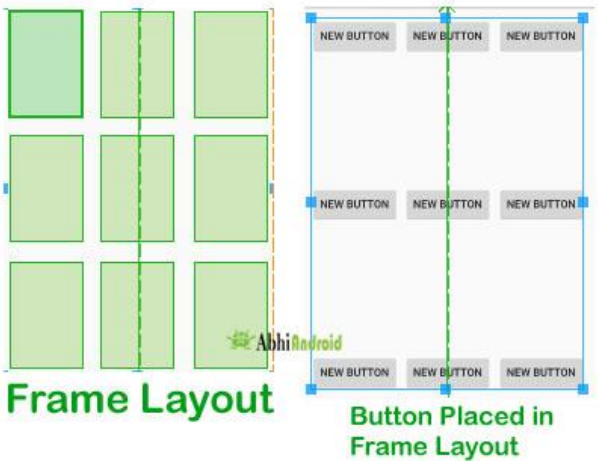
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate a button
        Button loginButton = (Button) findViewById(R.id.loginBtn);
        // perform click event on the button
        loginButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Hello AbhiAndroid..!!!",
Toast.LENGTH_LONG).show();  // display a toast message
            }
        });
    }



}
```

String.xml

```
<resources>
    <string name="app_name">TableLayoutExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="loginForm">Login Form</string>
    <string name="userName">UserName</string>
    <string name="password">Password</string>
    <string name="login">LogIn</string>
</resources>
```

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

1. 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

2. 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"

3. 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

4. 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.

Example of Frame Layout

2: Now Open res -> layout -> activity_main.xml and add the following code. Here we are putting different TextView in Frame Layout.

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


<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    >
    <TextView android:text="LeftTop"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightTop"
        android:layout_gravity="top|right" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CentreTop"
        android:layout_gravity="top|center_horizontal" />
    <TextView android:text="Left"
        android:layout_gravity="left|center_vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Right"
        android:layout_gravity="right|center_vertical" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Centre"
        android:layout_gravity="center" />
    <TextView android:text="LeftBottom"
        android:layout_gravity="left|bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightBottom"
        android:layout_gravity="right|bottom" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CenterBottom"
        android:layout_gravity="center|bottom" />
</FrameLayout>
```

Foreground

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

android:id="@+id/framelayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_gravity="center"
android:foregroundGravity="fill"
android:foreground="#0f0"><!--foreground color for a FrameLayout-->

<LinearLayout
android:orientation="vertical"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
>

<!-- Imageview will not be shown because of foreground color which is drawn over
it-->

<ImageView
android:layout_width="200dp"
android:layout_height="200dp"
android:layout_marginBottom="10dp"
android:src="@mipmap/ic_launcher"
android:scaleType="centerCrop"
/>

<!--Textview will not be shown because of foreground color is drawn over it-->

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
android:text="abhiAndroid"/>
```

```
</LinearLayout>


</FrameLayout>
```

Example of measure all child

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/frame"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:measureAllChildren="true"
    >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:src="@drawable/ic_launcher"/>

</FrameLayout>
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.demo);
        FrameLayout frame=(FrameLayout)findViewById(R.id.frame);
        frame.measure(View.MeasureSpec.UNSPECIFIED, View.MeasureSpec.UNSPECIFIED);
        int width = frame.getMeasuredWidth();
        int height = frame.getMeasuredHeight();
        Toast.makeText(getApplicationContext(),"width="+width+"
height="+height,Toast.LENGTH_SHORT).show();


    }
```

```
}
```

Example 3 :

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frameLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:src="@drawable/img_name"  /><!--Change image as per your name of
image saved in drawable-->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:text="abhiAndroid"
        android:textSize="30sp"
        android:textColor="#f3f3f3"
        android:textStyle="bold" />
</FrameLayout>
```

Jaba

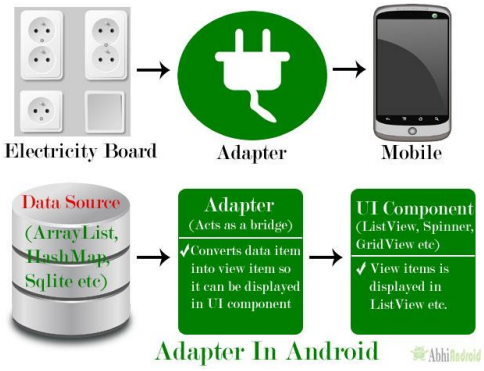
```
package abhiandroid.com.farmelayoutjava;


import android.graphics.Color;
```

```
import android.graphics.Typeface;
import android.os.Bundle;
import android.app.Activity;
import android.view.Gravity;
import android.view.Menu;
import android.widget.AbsListView;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ImageView imageView = new ImageView(this);
        imageView.setImageResource(R.drawable.img_name);
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);
        imageView.setLayoutParams(new
FrameLayout.LayoutParams(FrameLayout.LayoutParams.MATCH_PARENT,
                RelativeLayout.LayoutParams.MATCH_PARENT));
        TextView textView1 = new TextView(this);
        textView1.setText("abhiAndroid");
        textView1.setTextSize(30);
        textView1.setGravity(Gravity.CENTER);
        textView1.setTextColor(Color.parseColor("#f3f3f3"));
        textView1.setTypeface(null, Typeface.BOLD);
        textView1.setLayoutParams(new
FrameLayout.LayoutParams(FrameLayout.LayoutParams.MATCH_PARENT,
                RelativeLayout.LayoutParams.MATCH_PARENT));
        FrameLayout.LayoutParams lp1 = new
FrameLayout.LayoutParams(FrameLayout.LayoutParams.MATCH_PARENT,
FrameLayout.LayoutParams.WRAP_CONTENT);
        lp1.setMargins(0,20,0,0);
        textView1.setLayoutParams(lp1);

//Initializing frame layout
```

```
        FrameLayout framelayout = new FrameLayout(this);
        framelayout.setLayoutParams(new
AbsListView.LayoutParams(FrameLayout.LayoutParams.MATCH_PARENT,
                FrameLayout.LayoutParams.MATCH_PARENT));

//Adding views to FrameLayout

        framelayout.addView(imageView);
        framelayout.addView(textView1);
        setContentView(framelayout);
    }

}
```

ADAPTER

- 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 an Adapter view then view can takes the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc.
- For more customization in Views we uses the base adapter or custom adapters.
- To fill data in a list or a grid we need to implement Adapter.
- Adapters acts like a bridge between UI component and data source.
- Here data source is the source from where we get the data and UI components are list or grid items in which we want to display that data.



Adapters in Android

1. Base Adapter
   - BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView, GridView, Spinner etc.
   - Whenever we need a customized list in a ListView or customized grids in a GridView we create our own adapter and extend base adapter in that.
   - Base Adapter can be extended to create a custom Adapter for displaying a custom list item.
   - ArrayAdapter is also an implementation of BaseAdapter.

Code That shows usage of base Adapter

```
public class CustomAdapter extends BaseAdapter {

@Override
public int getCount() {
return 0;
}

@Override
public Object getItem(int i) {
return null;
}
```

```
@Override
public long getItemId(int i) {
return 0;
}

@Override
public View getView(int i, View view, ViewGroup viewGroup) {

return null;
}
```

we see the overrided functions of BaseAdapter which are used to set the data in a list, grid or a spinner.

2. Array Adapter
   - Whenever we have a list of single items which is backed by an Array, we can use ArrayAdapter.
   - For instance, list of phone contacts, countries or names.

```
- ArrayAdapter(Context context, int resource, int textViewResourceId,
  T[] objects)
```

3. Custom Array Adapter
   - ArrayAdapter is also an implementation of BaseAdapter, so if we want more customization then we can create a custom adapter and extend ArrayAdapter in that.
   - Since array adapter is an implementation of BaseAdapter, so we can override all the function's of BaseAdapter in our custom adapter.

Code :

```
public class MyAdapter extends ArrayAdapter {

public MyAdapter(Context context, int resource, int textViewResourceId, List
objects) {
super(context, resource, textViewResourceId, objects);
```

```
}
@Override
public int getCount() {
return super.getCount();
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
return super.getView(position, convertView, parent);
}
}
```

4. Simple Adapter in Android
   - In Android SimpleAdapter is an easy Adapter to map static data to views defined in an XML file(layout).
   - Android we can specify the data backing to a list as an ArrayList of Maps(
   - Each entry in a ArrayList is corresponding to one row of a list.
   - The Map contains the data for each row.
   - Here we also specify an XML file(custom list items file) that defines the views which is used to display the row, and a mapping from keys in the Map to specific views.
   - Whenever we have to create a custom list we need to implement custom adapter.
   - As we discuss earlier ArrayAdapter is used when we have a list of single item's backed by an Array.
   - So if we need more customization in a ListView or a GridView we need to implement simple adapter.

```
- SimpleAdapter (Context context, List<? extends Map<String, ?>>  data,
  int resource, String[] from, int[] to)
```

5. Custom Simple Adapter
   - Whenever we have to create a custom list we need to implement custom adapter.

- when we need more customization in list or grid items where we have many view's in a list item and then we have to perform any event like click or any other event to a particular view then we need to implement a custom adapter who fulfills our requirement's and quite easy to be implemented.
- BaseAdapter is the parent adapter for all other adapters so if we extends a SimpleAdapter then we can also override the base adapter's function in that class.

Code :

```
public class CustomAdapter extends SimpleAdapter {
public CustomAdapter(Context context, List<? extends Map<String, ?>> data, int
resource, String[] from, int[] to) {
super(context, data, resource, from, to);

}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
return super.getView(position, convertView, parent);

}

@Override
public int getCount() {
return super.getCount();
}
}
```

List View :

- 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.

- 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.

We 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 xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/simpleListView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="abhiandroid.com.listexample.MainActivity">
</ListView>
```

Attributes:

- Id : id is used to uniquely identify a ListView.
- divider: This is a drawable or color to draw between different list items.
- 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).
- 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.

```
-   <!-- List Selector Code in ListView -->
-   <ListView
-   android:id="@+id/simpleListView"
-   android:layout_width="fill_parent"
-   android:layout_height="wrap_content"
-   android:divider="#f00"
-   android:dividerHeight="1dp"
-   android:listSelector="#0f0"/> <!--list selector in green color-->
```

Adapters in List View

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.

In android commonly used adapters are:

- Array Adapter
- Base Adapter

1.Array Adapter:

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.

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

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

Example:

MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
android:divider="@color/material_blue_grey_800"
android:dividerHeight="1dp" />
</LinearLayout>
```

LIstView.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<TextView
android:id="@+id/textView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:padding="@dimen/activity_horizontal_margin"
android:textColor="@color/black" />
</LinearLayout>
```

Main Activity.Java

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;import android.widget.ListView;

public class MainActivity extends Activity
{
    // Array of strings...
    ListView simpleList;
    String countryList[] = {"India", "China", "australia", "Portugle", "America",
"NewZealand"};

    @Override   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

        simpleList = (ListView)findViewById(R.id.simpleListView);

        ArrayAdapter<String>   arrayAdapter   =   new   ArrayAdapter<String>(this,
R.layout.activity_listview, R.id.textView, countryList);

        simpleList.setAdapter(arrayAdapter);

    }

}
```

Example 2 :

ActivityMain.xml

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

<LinearLayout

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

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">


    <ListView

        android:id="@+id/list"

        android:layout_width="match_parent"

        android:layout_height="match_parent"/>

</LinearLayout>
```

Activiymain.Java

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    ListView l;
```

```
    String tutorials[]
        = { "Algorithms", "Data Structures",
            "Languages", "Interview Corner",
            "GATE", "ISRO CS",
            "UGC NET CS", "CS Subjects",
            "Web Technologies" };

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        l = findViewById(R.id.list);
        ArrayAdapter<String> arr;
        arr
            = new ArrayAdapter<String>(
                this,
                R.layout.support_simple_spinner_dropdown_item,
                tutorials);
        l.setAdapter(arr);
    }
}
```

List View with Base Adapter:

Main Activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="@color/material_blue_grey_800"
android:dividerHeight="1dp"
android:footerDividersEnabled="false" />
</LinearLayout>
```

Activity_listview.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal">

<ImageView
android:id="@+id/icon"
android:layout_width="50dp"
android:layout_height="50dp"
android:src="@drawable/ic_launcher" />

<TextView
android:id="@+id/textView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:padding="@dimen/activity_horizontal_margin"
android:textColor="@color/black" />
</LinearLayout>
```

Main Activity.Java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;

public class MainActivity extends Activity {

ListView simpleList;
String countryList[] = {"India", "China", "australia", "Portugle", "America",
"NewZealand"};
int flags[] = {R.drawable.india, R.drawable.china, R.drawable.australia,
R.drawable.portugle, R.drawable.america, R.drawable.new_zealand};
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
simpleList = (ListView) findViewById(R.id.simpleListView);
CustomAdapter customAdapter = new CustomAdapter(getApplicationContext(),
countryList, flags);
simpleList.setAdapter(customAdapter);
}
}
```

CustomAdapter.java

```
import android.content.Context;
import android.media.Image;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.zip.Inflater;

public class CustomAdapter extends BaseAdapter {
Context context;
String countryList[];
int flags[];
LayoutInflater inflter;

public CustomAdapter(Context applicationContext, String[] countryList, int[]
flags) {
this.context = context;
this.countryList = countryList;
this.flags = flags;
inflter = (LayoutInflater.from(applicationContext));
}
```

```
@Override
public int getCount() {
return countryList.length;
}

@Override
public Object getItem(int i) {
return null;
}

@Override
public long getItemId(int i) {
return 0;
}

@Override
public View getView(int i, View view, ViewGroup viewGroup) {
view = inflter.inflate(R.layout.activity_listview, null);
TextView country = (TextView)           view.findViewById(R.id.textView);
ImageView icon = (ImageView) view.findViewById(R.id.icon);
country.setText(countryList[i]);
icon.setImageResource(flags[i]);
return view;
}
```

Example 3 For List view

1. **String.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="teams">
    <item>India</item>
    <item>South Africa</item>
```

```
        <item>Australia</item>
        <item>England</item>
        <item>New Zealand</item>
        <item>Sri Lanka</item>
        <item>Pakistan</item>
        <item>West Indies</item>
        <item>Bangladesh</item>
        <item>Ireland</item>
    </string-array>
</resources>
```

2. **List_View.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<!--  Single List Item Design -->
<TextView
xmlns:android="https://schemas.android.com/apk/res/android"
   android:id="@+id/textview"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:padding="10dip"
      android:textSize="16dip"
      android:textStyle="bold" >
</TextView>

// storing string resources into Array
String[] teams = getResources().getStringArray(R.array.teams);

// Binding resources Array to ListAdapter
this.setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item,
R.id.textview, teams));
```

3. **MainActivity.Java**

```
package journaldev.com.listview;


import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends ListActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // storing string resources into Array
      String[] teams = getResources().getStringArray(R.array.teams);

      // Binding resources Array to ListAdapter
      this.setListAdapter(new ArrayAdapter<String>(this,
R.layout.list_item, R.id.textview, teams));

      ListView lv = getListView();

      // listening to single list item on click
      lv.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
         public void onItemClick(AdapterView<?> parent, View view,
                        int position, long id) {

            // selected item
            String team = ((TextView) view).getText().toString();

            // Launching new Activity on selecting single List Item
            Intent i = new Intent(getApplicationContext(),
SecondActivity.class);
            // sending data to new activity
            i.putExtra("team", team);
            startActivity(i);

         }
      });
   }
}
```

4. **SecondActivity.java**

```
package journaldev.com.listview;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
      TextView txtProduct = (TextView) findViewById(R.id.team_label);

      Intent i = getIntent();
      // getting attached intent data
      String product = i.getStringExtra("team");
      // displaying selected product name
      txtProduct.setText(product);
   }

}
```

List View Implementation Using Base Adapter

FINAL EXAMPLES oF LIST VIEW WITH BASIC DATA

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listView2"
        android:divider="#ff4432"
        android:dividerHeight="2dp"/>


</LinearLayout>
```

Acitivity_list_view.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ListView">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listTextView"
        android:layout_gravity="center"
        android:padding="5dp"
        android:textColor="@color/black"/>


</LinearLayout>
```

Listview.java

```
package com.example.finallistviewex2;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class ListView extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view);
    }
}
```

Main Activity.java

```
package com.example.finallistviewex2;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.ArrayAdapter;

public class MainActivity extends AppCompatActivity {

   ListView lv;
    String data [] = {"AI", "MAD", "ML", "MCWC", "CD"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lv = findViewById(R.id.listView2);
        ArrayAdapter<String> arrayAdapter = new
ArrayAdapter<String>(this,R.layout.activity_list_view,R.id.listTextView,dat
a);
        lv.setAdapter(arrayAdapter);

    }
}
```

Example2

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:padding="15dp"
    android:orientation="vertical">

    <TextView
       android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="SampleList"
        android:textSize="20sp"
        android:textColor="@color/black"
        />
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listInstitutes"
        android:divider="#AAf002"
        android:dividerHeight="1dp"
        android:listSelector="#FFA200"/>

</LinearLayout>
```

String.xml

```
<resources>
    <string name="app_name">FinalListViewEx</string>
    <string-array name="LJU">
        <item>LJIET</item>
        <item>LJMCA</item>
        <item>LJ Diploma</item>
        <item>LJ Pharma</item>
        <item>LJ Arch</item>
    </string-array>
</resources>
```

MainActivity.Java

```
package com.example.finallistviewex;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    ListView lvInstitutes;
    String [] institutes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lvInstitutes = findViewById(R.id.listInstitutes);
        institutes =  getResources().getStringArray(R.array.LJU);
        ArrayAdapter<String> instituteAdapter = new
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,institutes);
        lvInstitutes.setAdapter(instituteAdapter);
    }
}
```

TAB LAYOUT

- In Android TabLayout is a new element introduced in Design Support library.
- A TabLayout provides a way to display tabs horizontally. When used together with a ViewPager , a TabLayout can provide a familiar interface for navigating between pages in a swipe view.
- We can quickly swipe between the tabs. TabLayout is basically view class required to be added into our layout(xml) for creating Sliding Tabs.
- We use different methods of TabLayout to create, add and manage the tabs.
- If we need simple tabs without sliding then we replace the layout with the fragment on tab selected listener event and if we need sliding tabs then we use Viewpager.

Ex:

Create Basic Tab Layout

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabMode="fixed"
app:tabGravity="fill"/>
```

Methods of Tab Layout

1. newTab(): This method is used to create and return a new TabLayout.Tab.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); //
get the reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names
firstTab.setText("First Tab"); // set the Text for the first Tab
```

2. addTab(Tab tab): This method is used to add a tab in the TabLayout. By using this method we add the tab which we created using newTab() method in the TabLayout.
   The tab will be added at the end of the list and If it is the first tab to be added then it will become the selected tab.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the
reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
tabLayout.addTab(firstTab); // add  the tab to the TabLayout
```

3. addTab(Tab tab, boolean setSelected): This method is used to add a tab in the TabLayout and set the state for the tab. By using this method we add the tab which we created using newTab() method in the TabLayout. In this method we also set the state of the tab whether it is selected or not.

   Below we firstly create a new tab and then add it in the TabLayout and set the true value for setSelected parameter that makes it selectable.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the
reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
tabLayout.addTab(firstTab,true); // add  the tab in the TabLayout and makes it
selectable
```

4. addTab(Tab tab, int position): This method is used to add a tab in the TabLayout and set the state for the tab. By using this method we add the tab which we created using newTab() method in the TabLayout. The tab will be inserted at the given position. If it is the first tab to be added then it will become the selected tab.

Below we firstly create a new tab and then add it in the TabLayout at a specific position.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the
reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab,2);  //  add  the  tab  in  the  TabLayout  at  specific
position
```

5. addTab(Tab tab, int position, boolean setSelected): This method is used to add a tab at a specific position and set the state of the tab. By using this method we add the tab which we created using newTab() method in the TabLayout. The tab will be inserted at the defined position and a Boolean value used to set the state of the tab. True value is used to make the tab selectable.

   Below we firstly create a tab and then add it in the TabLayout at a specific position and we also set true value to make the tab selectable.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the
reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
```

```
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab,2,true); // add  the tab at specified position in the TabLayout and makes it selectable
```

6. getSelectedTabPosition(): This method is used to get the position of the current selected tab. This method returns an int type value for the position of the selected tab. It returns -1 if there isn't a selected tab.

   Below we get the current selected tab position.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
int selectedTabPosition = tabLayout.getSelectedTabPosition(); // get the position for the current selected tab
```

7. getTabAt(int index): This method is used to get the tab at the specified index. This method returns TabLayout.Tab.

   Below we get the tab at 1th index.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
TabLayout.Tab tab = tabLayout.getTabAt(1); //  get the tab at 1th in index
```

8. getTabCount(): This method is used to get the number of tabs currently registered with the action bar. This method returns an int type value for the number of total tabs.

   Below we get the total number of tabs currently registered with the action bar.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
int tabCount= tabLayout.getTabCount(); // get the total number of tabs currently registered with the action bar.
```

9. setTabGravity(int gravity): This method is used to set the gravity to use when laying out the tabs.

   Below we set the gravity for the tabs.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER); // set the gravity to use when laying out the tabs
```

10. getTabGravity(): This method is used to get the current gravity used for laying out tabs. This method returns the gravity which we set using setTabGravity(int gravity) method.

    Below we firstly set the gravity and then get the current gravity used for laying out tabs.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER); // set the gravity to use when laying out the tabs.
int gravity=tabLayout.getTabGravity(); // get the current gravity used for laying out tabs
```

11. setTabMode(int mode): This method is used to set the behavior mode for the Tabs in this layout. The valid input options are:
    MODE_FIXED: Fixed tabs display all tabs concurrently and are best used with content that benefits from quick pivots between tabs.

    MODE_SCROLLABLE: Scrollable tabs display a subset of tabs at any given moment and it can contain longer tab labels and a larger number of tabs. They are best used for browsing contexts in touch interfaces when users don't need to directly compare the tab labels. This mode is commonly used with a ViewPager.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE); // set the behaviour mode for the tabs
```

12. getTabMode(): This method is used to get the current mode of TabLayout. This method returns an int type value which we set using setTabMode(int mode) method.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE); // set the behaviour mode for the tabs
int mode=tabLayout.getTabMode(); // get the current mode of the TabLayout
```

13. setTabTextColors(int normalColor, int selectedColor): This method is used to set the text colors for the different states (normal, selected) of the tabs.

    Below we set the tab text colors for the both states of the tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabTextColors(Color.RED,Color.WHITE); // set the tab text colors for the both states of the tab.
```

14. getTabTextColors(): This method is used to get the text colors for the different states (normal, selected) of the tabs. This method returns the text color which we set using setTabTextColors(int normalColor, int selectedColor) method.

    Below we firstly set the text colors and then get the text colors for the both states of the tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setTabTextColors(Color.RED,Color.WHITE); // set the tab text colors for the both states of the tab.
ColorStateList colorStateList=tabLayout.getTabTextColors(); // get the text colors for the both states of the tab
```

15. removeAllTabs(): This method is used to remove all tabs from the action bar and deselect the current tab.

    Below we remove all the tabs from the action bar and deselect the current tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
```

```
tabLayout.removeAllTabs(); // remove all the tabs from the action bar and deselect
the current tab
```

16. setOnTabSelectedListener(OnTabSelectedListener listener): This method is used to add a listener that will be invoked when tab selection changes.

    Below we show how to use addOnTabSelectedListener of TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the
reference of TabLayout
tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
@Override
public void onTabSelected(TabLayout.Tab tab) {
// called when tab selected
}

@Override
public void onTabUnselected(TabLayout.Tab tab) {
// called when tab unselected
}

@Override
public void onTabReselected(TabLayout.Tab tab) {
// called when a tab is reselected
}
});
```

17. removeTab(Tab tab): This method is used to remove a tab from the layout. In this method we pass the TabLayout.Tab object to remove the tab from the layout. If the removed tab was selected then it will be automatically deselected and another tab will be selected if present in the TabLayout.

    Below we firstly create and add a tab and then remove it from the TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
tabLayout.addTab(firstTab); // add  the tab at in the TabLayout
tabLayout.removeTab(firstTab); // remove the tab from the TabLayout
```

18. removeTabAt(int position):
    This method is used to remove a tab from the layout. In this method we pass the position of the tab that we want to remove from the layout. If the removed tab was selected then it will be automatically deselected and another tab will be selected if present in the TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.removeTabAt(1); // remove the tab from a specified position of TabLayout
```

19. setSelectedTabIndicatorColor(int color): This method is used to set the tab indicator's color for the currently selected tab.

    Below we set the red color for the selected tab indicator.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setSelectedTabIndicatorColor(Color.RED); // set the red color for the selected tab indicator.
```

20. setSelectedTabIndicatorHeight(int height): This method is used to set the tab indicator's height for the currently selected tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference of TabLayout
tabLayout.setSelectedTabIndicatorHeight(2); // set the height for the selected tab's indicator
```

21. setupWithViewPager(ViewPager viewPager): This method is used for setting up the TabLayout with ViewPager. ViewPager is mainly used for creating Sliding tabs.

```
ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager); // get the
reference of ViewPager
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); //
get the reference of TabLayout
tabLayout.setupWithViewPager(viewPager); // set the TabLayout with the
ViewPager.
```

ViewPager in Android

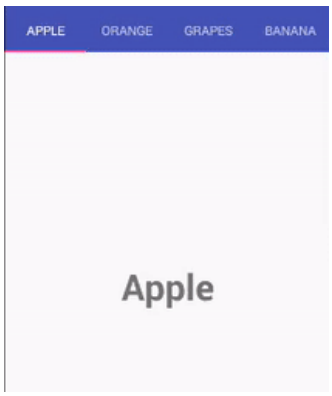
- Layout manager that allows the user to flip left and right through pages of data. You supply an implementation of a PagerAdapter to generate the pages that the view shows.

- ViewPager is most often used in conjunction with Fragment, which is a convenient way to supply and manage the lifecycle of each page.

- There are standard adapters implemented for using fragments with the ViewPager, which cover the most common use cases.

- These are FragmentPagerAdapter and FragmentStatePagerAdapter; each of these classes have simple code showing how to build a full user interface with them.

- ViewPager is most often used along with fragment which is convenient way to manage the life cycle of each page. ViewPager class works with PagerAdapter which provides pages.



Code to implement View Pager:

```
<android.support.v4.view.ViewPager
android:id="@+id/viewpager"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

Using FragmentPagerAdapter To Implement PagerAdapter:

1. Usually to display sliding fragments two methods of FragmentPagerAdapter are used, getCount() and getItem(). The first one returns the number of fragments to be displayed, and the second one is used to display the actual fragment & it should be used when we need to store the whole fragment in memory.

2. It is an implementation of PagerAdapter class that represents each page as Fragment that is persistently kept in the fragment manager as long as the user can return to the page, hence best to use when there's a fixed small set of screens to be navigated through.

3. It works even better when the content of the fragments are static than something that constantly changes or gets updated.

4. When using FragmentPagerAdapter the host ViewPager must have a valid ID set. It stores the previous data which is fetched from the adapter. It stores the whole fragment in memory and could increase a memory overhead if large amount of fragments are used in the ViewPager.

Example of Tab With ViewPager

1. Create activity main.xml layout with tablayout(to implement tab layout) and one ViewPager2 (to implement swipe navigation)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <com.google.android.material.tabs.TabLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tab_layout"
        />
    <androidx.viewpager2.widget.ViewPager2
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/view_pager2"
        />
</LinearLayout>
```

2. Create 3 – Fragments with Text View

Fragment-1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="First Fragment"
        android:textSize="50sp"
        android:gravity="center"/>

</FrameLayout>
```

Fragment-2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SeconfFragment">
```

```
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Second Fragment"
        android:textSize="50sp"
        android:gravity="center"/>

</FrameLayout>
```

Fragment-3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ThirdFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Third Fragment"
        android:textSize="50sp"
        android:gravity="center"/>

</FrameLayout>
```

3. No change in auto generated Java files for First Fragment , Second Fragment and Third Fragment

   FirstFragment.java

```
package com.example.fragmentex3;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FirstFragment extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first, container, false);
```

```
    }
}
```

SecondFragment.java

```
package com.example.fragmentex3;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SeconfFragment extends Fragment {



    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_seconf, container,
false);
    }
}
```

ThirdFragment.java

```
package com.example.fragmentex3;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;


public class ThirdFragment extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_third, container, false);
    }
}
```

4. Add a Adapter class to manage the view of Fragments. I am going to create a class FragmentAdapter which is going to extend FragmentStateAdapter

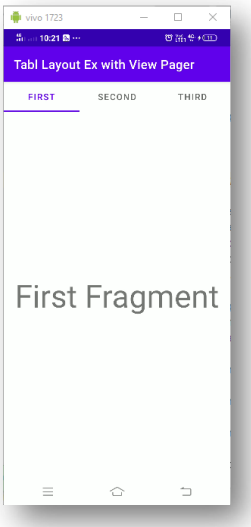
```
package com.example.fragmentex3;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.lifecycle.Lifecycle;
import androidx.viewpager2.adapter.FragmentStateAdapter;

public class FragmentAdapter extends FragmentStateAdapter {


    public FragmentAdapter(FragmentManager fragmentManager,  Lifecycle
lifecycle) {
        super(fragmentManager, lifecycle);
    }

    @Override
    public Fragment createFragment(int position) {

        switch (position)
        {
            case 1: return new SeconfFragment();// if position is one then
call second fragment
            case 2: return  new ThirdFragment(); // if position is two then
call third fragment

        }
        return new FirstFragment(); // default cases it will call first
fragment object and bind to the viewpager / layout
    }

    @Override
    public int getItemCount() {
        return 3;  // as we have only 3 Fragments to view so we can set it
to 3 as static value
     }
}
```

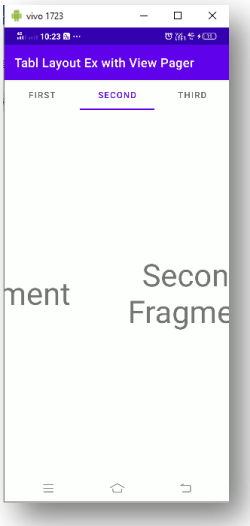
MainActivity.Java Code

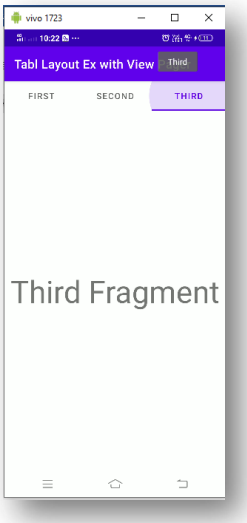
```
package com.example.fragmentex3;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.viewpager2.widget.ViewPager2;

import android.os.Bundle;

import com.google.android.material.tabs.TabLayout;
```

```
public class MainActivity extends AppCompatActivity {

    TabLayout tabLayout;
    ViewPager2 viewPager2;
    FragmentAdapter fragmentAdapter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tabLayout = findViewById(R.id.tab_layout);
        viewPager2 = findViewById(R.id.view_pager2);

        FragmentManager fm = getSupportFragmentManager();
        fragmentAdapter  = new FragmentAdapter(fm, getLifecycle());
        viewPager2.setAdapter(fragmentAdapter);

        tabLayout.addTab(tabLayout.newTab().setText("First"));

        tabLayout.addTab(tabLayout.newTab().setText("Second"));

        tabLayout.addTab(tabLayout.newTab().setText("Third"));

        tabLayout.addOnTabSelectedListener(new
TabLayout.OnTabSelectedListener() {
            @Override
            public void onTabSelected(TabLayout.Tab tab) {
                viewPager2.setCurrentItem(tab.getPosition());
            }

            @Override
            public void onTabUnselected(TabLayout.Tab tab) {

            }

            @Override
            public void onTabReselected(TabLayout.Tab tab) {

            }
        });


        viewPager2.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
            @Override
            public void onPageSelected(int position) {
                tabLayout.selectTab(tabLayout.getTabAt(position));
            }
        });
    }
}
```

o/p


vivo 1723
10:21
Tabl Layout Ex with View Pager
FIRST
SECOND
THIRD
First Fragment


vivo 1723
10:23
Tabl Layout Ex with View Pager
FIRST
SECOND
THIRD
ment
Secon
Fragme


vivo 1723
10:22
Tabl Layout Ex with View
Third
FIRST
SECOND
THIRD
Third Fragment