Practical No. 01

- **Create "HelloWorld" message in android studio.**

Steps to follow:

1) Create new project in android studio ,select File → New Project
2) In the new project screen enter the following information:
    a) Application name: "HelloworldDemo"
    b) Company domain: if u want u can change or **keep as it is.**
    c) Project location : change as u want the project at desired location.
    d) Keep package name as it is.
    e) Click **NEXT** button.
3) In the **target android devices** screen, keep the defaults values and click **Next.**
4) In the **add an activity to mobile** screen, select **Empty Activity** and click **Next.**
5) In the **configure activity** screen, keep the default vales as they are otherwise if u want u can change the activity name. Make sure that the check boxes are checked.
6) From the project pan u can open activity_main.xml file which resides under app → res→ layout→activity_main.xml

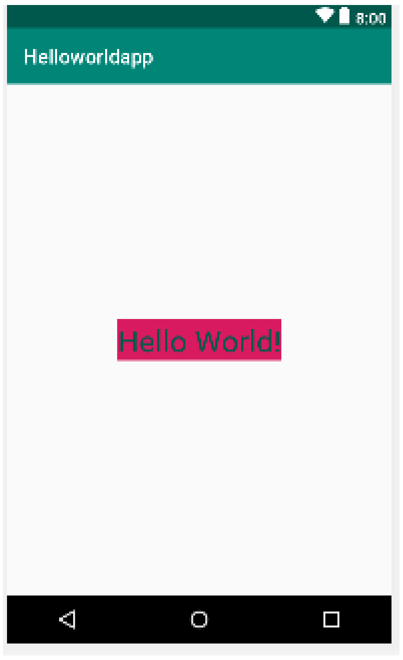
Modify Activity_Main.xml file as follow→

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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"
  tools:context=".MainActivity">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="30dp"
    android:textColor="@color/colorPrimaryDark"
    android:background="@color/colorAccent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

7) **There is no change in MainActivity.java →**

```
package com.anu.helloworldapp;
import android.support.v7.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);
 }
 }
```

Output:



PRACTICAL NO. 02
RESOURCES IN ANDROID

Color resource :

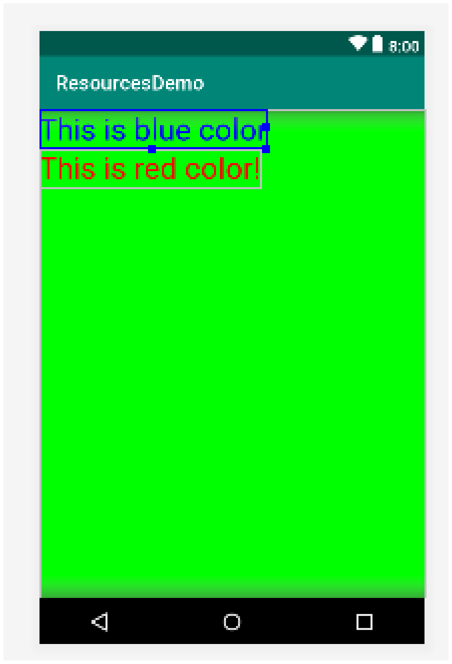
1) Create a new project named ResourceDemo
2) In the project pan go to app→res→values→color.xml
3) write the following code:

colors.xml as follow→

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="colorPrimary">#008577</color>
<color name="colorPrimaryDark">#00574B</color>
<color name="colorAccent">#D81B60</color>
<color name="red">#FF0000</color>
<color name="blue">#0000FF</color>
<color name="green">#00FF00</color>
</resources>
```

Activity_Main.xml→

```
<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"
android:background="@color/green"
tools:context=".MainActivity">
<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30dp"
android:text="This is blue color"
android:textColor="@color/blue"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```



This is blue color
This is red color!

```
        android:text="This is red color!"
        android:textSize="30dp"
        android:textColor="@color/red"
        />
</LinearLayout>
```

Theory of Color Resource:

A color value defined in XML. The color is specified with an RGB value and alpha channel. You can use a color resource any place that accepts a hexadecimal color value. You can also use a color resource when a drawable resource is expected in XML (for example, android:drawable="@color/green").

The value always begins with a pound (#) character and then followed by the Alpha-Red-Green-Blue information in one of the following formats:

- *#RGB*
- *#ARGB*
- *#RRGGBB*
- *#AARRGGBB*

file location:

res/values/colors.xml

The filename is arbitrary. The <color> element's name will be used as the resource ID.

resource reference:

In Java: R.color.*color_name*

In XML: @[*package*:]color/*color_name*

syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color
        name="color_name"
        >hex_color</color>
</resources>
```

elements:

<resources>

Required. This must be the root node.

No attributes.

<color>

A color expressed in hexadecimal, as described above.

attributes:

name

String. A name for the color. This will be used as the resource ID.

example:

XML file saved at res/values/colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <colorname="opaque_red">#f00</color>
  <colorname="translucent_red">#80ff0000</color>
</resources>
```

Dimension Resource:

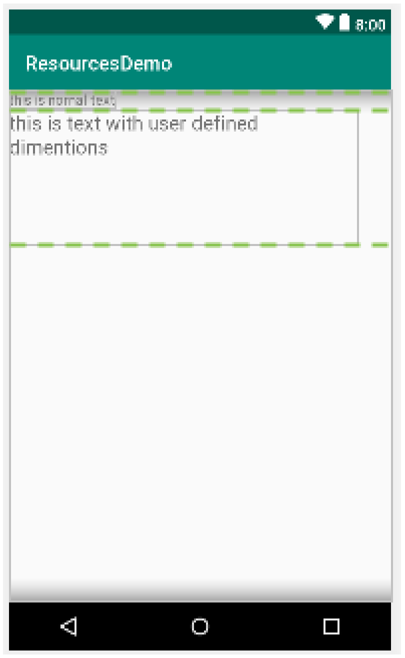
1) **Create a new file dimens.xml**
   a) **Right click on app→res→values→ then add new Resource file named above.**
2) **Write the code as follow :**

dimens.xml →

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<dimen name="height">35dp</dimen>
<dimen name="width">150dp</dimen>
<dimen name="size">20dp</dimen>
</resources>
```

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">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="this is nornal text"/>

<TextView
android:layout_width="@dimen/width"
android:layout_height="@dimen/height"
android:textSize="@dimen/size"
android:text="this is text with user defined dimentions"/>
</LinearLayout>
```



Theory of Dimension Resource:

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android:

dp

Density-independent Pixels - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

sp

Scale-independent Pixels - This is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.

pt

Points - 1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

px

Pixels - Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

mm

Millimeters - Based on the physical size of the screen.

in

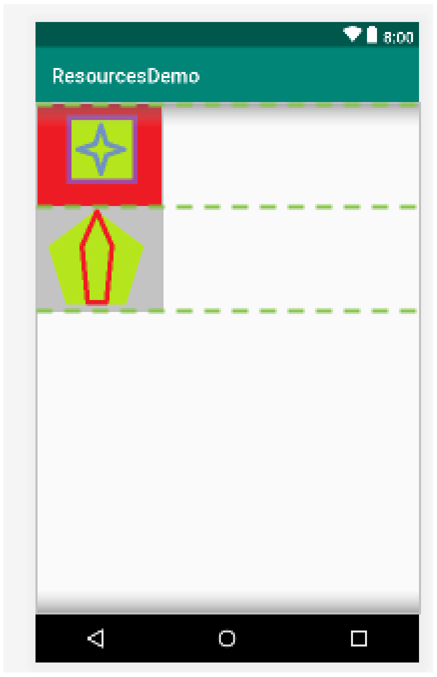
Inches - Based on the physical size of the screen.

Drawable Resource:

1) First create minimum 2images in paint brush. Save at desired location.
2) Open file explorer ,go to the desired location, copy that images and paste the images by right click on drawable directory under res. [i.e. app→res→drawable] in android studio.

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">

<ImageView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:src="@drawable/Untitled"/>
<ImageView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:src="@drawable/Untitled1"/>
</LinearLayout>
```



String Resources:

To open strings.xml file :

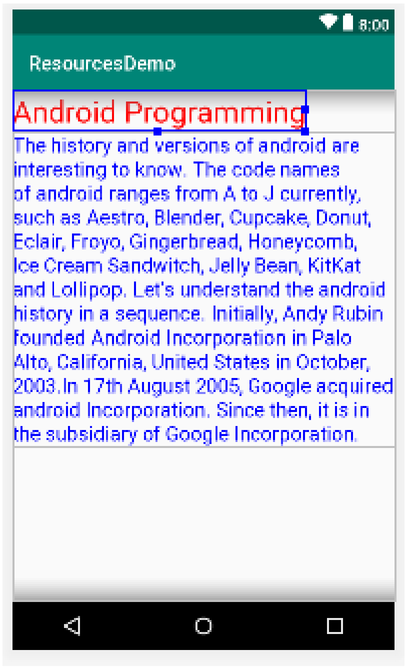
app→res→values→strings.xml

write code as follow in strings.xml →

```
<resources>
<string name="app_name">ResourcesDemo</string>
<string name="Heading">Android Programming</string>
<string name="History">
    The history and versions of android are interesting to know. The code names of androidranges from A
to J currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo,Gingerbread, Honeycomb, Ice Cream
Sandwitch, Jelly Bean, KitKat and Lollipop.
    Let's understand the android history in a sequence.
    Initially, Andy Rubin founded Android Incorporation in Palo Alto,
    California, United States in October, 2003.In 17th August 2005,
    Google acquired android Incorporation. Since then,
    it is in the subsidiary of Google Incorporation.
</string>
</resources>
```

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">
<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="30dp"
android:text="@string/Heading"
android:textColor="@color/red"/>
<TextView
    android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/History"
android:textSize="20dp"
android:textColor="@color/blue"
```



Android Programming

The history and versions of android are interesting to know. The code names of android ranges from A to J currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat and Lollipop. Let's understand the android history in a sequence. Initially, Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.

```
        />
</LinearLayout>
```

Style Resource:

To open style.xml file right click app→res→values→style.xml

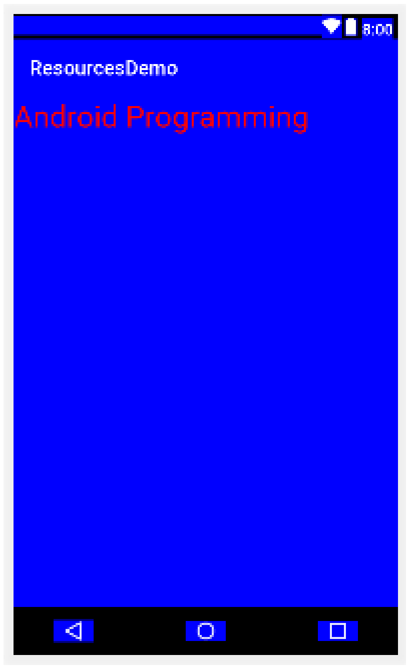
Style.xml→

```
<resources>
<style name="MyTheme" parent="Theme.AppCompat.DayNight.DarkActionBar">
<item name="android:textColor">@color/red</item>
<item name="android:background">@color/blue</item>
</style>
</resources>
```

Manifest.xml→

To open manifest file app->>manifest→manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.anu.resourcesdemo">

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
android:theme="@style/MyTheme">
<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```



Android Programming

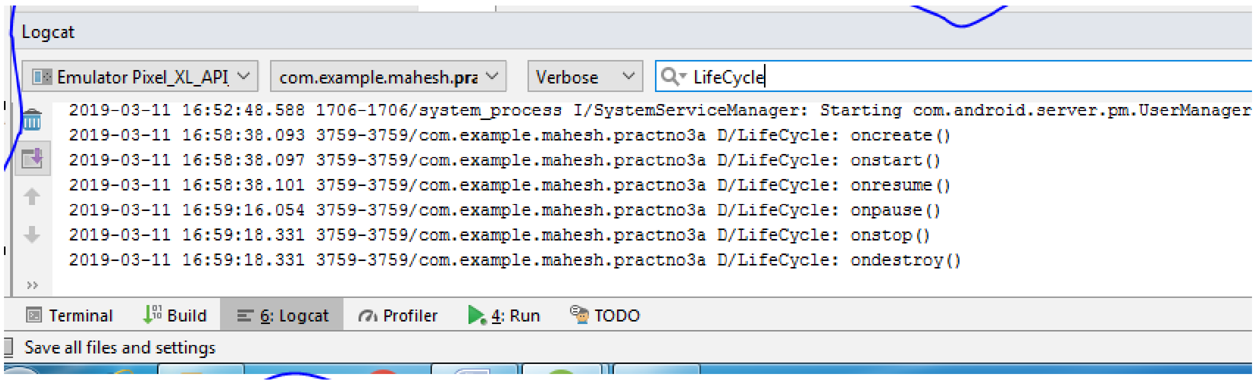
Practical No. 03 [A]
Android Activity Lifecycle

Write code in MainActivity.java file as follow→

Note: To add following lifecycle methods , right click in mainactivity.java file and select Generate→override methods OR shortcut key is “CTRL+O”.

```
package com.example.mahesh.practno3a;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;


public class MainActivity extends AppCompatActivity
{
String tag="LifeCycle";

   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(tag,"oncreate()");

   }
   public void onStart()
   {
      super.onStart();
      Log.d(tag,"onstart()");
   }
   public void onRestart()
   {
      super.onRestart();
      Log.d(tag,"onrestart()");
   }
   public void onResume()
   {
      super.onResume();
      Log.d(tag,"onresume()");
   }
   public void onPause()
   {
      super.onPause();
      Log.d(tag,"onpause()");
   }
```

```
    public void onStop()
    {
       super.onStop();
       Log.d(tag,"onstop()");
    }
    public void onDestroy()
    {
       super.onDestroy();
       Log.d(tag,"ondestroy()");
    }
}
```



```
2019-03-11 16:52:48.588 1706-1706/system_process I/SystemServiceManager: Starting com.android.server.pm.UserManager
2019-03-11 16:58:38.093 3759-3759/com.example.mahesh.practno3a D/LifeCycle: oncreate()
2019-03-11 16:58:38.097 3759-3759/com.example.mahesh.practno3a D/LifeCycle: onstart()
2019-03-11 16:58:38.101 3759-3759/com.example.mahesh.practno3a D/LifeCycle: onresume()
2019-03-11 16:59:16.054 3759-3759/com.example.mahesh.practno3a D/LifeCycle: onpause()
2019-03-11 16:59:18.331 3759-3759/com.example.mahesh.practno3a D/LifeCycle: onstop()
2019-03-11 16:59:18.331 3759-3759/com.example.mahesh.practno3a D/LifeCycle: ondestroy()
```

Save all files and settings

Note: When your app is getting loaded in to the emulator then ,then only the methods onCreate(),onStart() and onResume() are going to be executed. To show all methods of activity lifecycle in Logcat we need to close the app from the emulator then it shows all life cycle methods ,as shown in the above output window.[i.e. onPause(),onStop(),onDestroy()]

To display the log messages for an app:

1. Build and run your app on a device.
2. Click **View > Tool Windows > Logcat** (or click **Logcat** in the tool window bar).

The Logcat window shows the log messages for the selected app

Theory→

Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

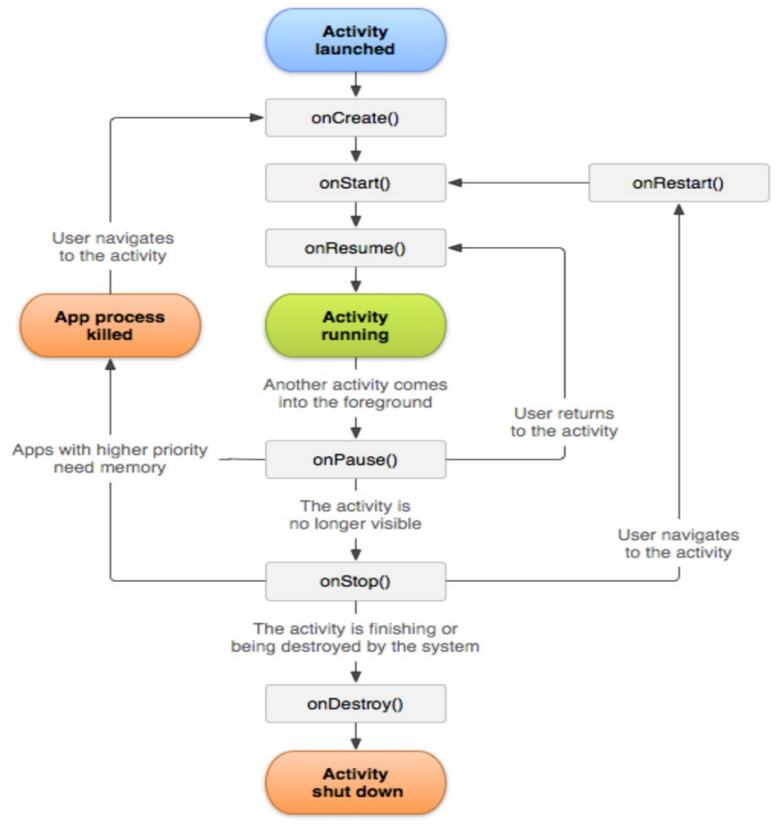
An activity is the single screen in android. It is like window or frame of Java.

By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.

Android Activity Lifecycle methods

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.


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

Logcat →

The **Logcat** window in Android Studio displays system messages, such as when a garbage collection occurs, and messages that you added to your app with the `Log` class. It displays messages in real time and keeps a history so you can view older messages.

To display just the information of interest, you can create filters, modify how much information is displayed in messages, set priority levels, display messages produced by app code only, and search the log. By default, logcat shows the log output related to the most recently run app only.

When an app throws an exception, logcat shows a message followed by the associated stack trace containing links to the line of code.

Write log messages

The `Log` class allows you to create log messages that appear in logcat. Generally, you should use the following log methods, listed in order from the highest to lowest priority (or, least to most verbose):

- `Log.e(String, String)` (error)
- `Log.w(String, String)` (warning)
- `Log.i(String, String)` (information)
- `Log.d(String, String)` (debug)
- `Log.v(String, String)` (verbose)

- **Verbose:** Show all log messages (the default).
- **Debug:** Show debug log messages that are useful during development only, as well as the message levels lower in this list.
- **Info:** Show expected log messages for regular usage, as well as the message levels lower in this list.
- **Warn:** Show possible issues that are not yet errors, as well as the message levels lower in this list.
- **Error:** Show issues that have caused errors, as well as the message level lower in this list.
- **Assert:** Show issues that the developer expects should never happen.

Practical No. 03[B]
Android Fragments LifeCycle

Create a new file as follow:

Right click app→new→Android Resource File→Name it as Lifeoffragment→click ok.

Lifeoffragment.java file as follow:-

```
package com.example.mahesh.practno3b;
import android.content.Context;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.util.Log;

public class LifeOfFragment extends Fragment
{
   String tag="LifeCycle";

public LifeOfFragment()
 {
// Required empty public constructor
}

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

   @Override
public void onAttach(Context context)
 {
super.onAttach(context);
   Log.d(tag,"on attach");
   }

   @Override
public void onCreate(@Nullable Bundle savedInstanceState)
```

```
{
super.onCreate(savedInstanceState);
        Log.d(tag,"on create");
    }

    @Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState)
{
super.onViewCreated(view, savedInstanceState);
        Log.d(tag,"on viewcreate");
    }

    @Override
public void onPause()
{
super.onPause();
        Log.d(tag,"on pause");
    }

    @Override
public void onStop()
 {
super.onStop();
        Log.d(tag,"on stop");
    }

    @Override
public void onDestroyView()
 {
super.onDestroyView();
        Log.d(tag,"on destroyview");
    }

    @Override
public void onDestroy()
 {
super.onDestroy();
        Log.d(tag,"on destory");
    }

    @Override
public void onActivityCreated(@Nullable Bundle savedInstanceState)
 {
super.onActivityCreated(savedInstanceState);
        Log.d(tag,"on act creation");
    }

    @Override
public void onStart()
```

```
 {
super.onStart();
      Log.d(tag,"on start");
   }

   @Override
public void onResume()
 {
super.onResume();
      Log.d(tag,"on resume");
   }
}
```

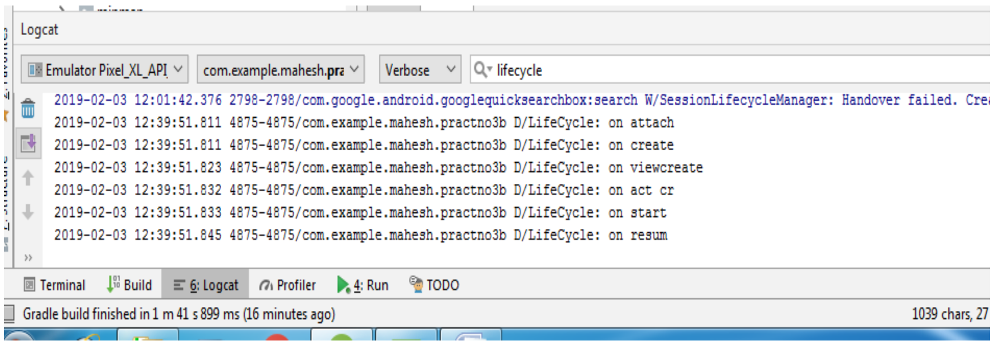
fragment_life_of.xml file as follow:-

```
<?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=".LifeOfFragment">

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

</FrameLayout>
```

Activity_main.xml file as follow:-

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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"
   tools:context=".MainActivity">


<fragment
      android:id="@+id/test_fragment"
      class="com.example.mahesh.practno3b.LifeOfFragment"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      tools:layout="@layout/fragment_life_of"
```

```
/>

</android.support.constraint.ConstraintLayout>
```



```
2019-02-03 12:01:42.376 2798-2798/com.google.android.googlequicksearchbox:search W/SessionLifecycleManager: Handover failed. Cre
2019-02-03 12:39:51.811 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on attach
2019-02-03 12:39:51.811 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on create
2019-02-03 12:39:51.823 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on viewcreate
2019-02-03 12:39:51.832 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on act cr
2019-02-03 12:39:51.833 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on start
2019-02-03 12:39:51.845 4875-4875/com.example.mahesh.practno3b D/LifeCycle: on resum
```

```
Gradle build finished in 1 m 41 s 899 ms (16 minutes ago)          1039 chars, 27
```

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.

The **FragmentManager** class is responsible to make interaction between fragment objects.

Difference between activity and fragments:-

1. An Activity is an application component that provides a screen, with which users can interact in order to do something (such as take a picture , send sms , dial a number, send an email, view a map, etc).Fragment represents a behavior or a portion of user interface in an Activity. A single Activity can have one or more fragments visible. Fragments were designed to be modular, self-contained UI components that can be reused many times within the same app or between apps. A fragment can't exist independently. It should be always part of an activity whereas activity can exist without any fragment in it.

   Fragments were introduced into the framework to handle the difficulty of designing an app that scales and responds the same across many different devices of varying sizes and types.

2. Activity is an application component which gives user interface where user can interact. Fragment is a part of an activity, which contribute its own UI to that activity.
3. For tablet or if mobile is in landscape then using fragment we can show two list like one list for show the state name and other list will show the state description in single activity; but using activity, we can't do the same thing.
4. Activity is not dependent on fragment, but fragment is dependent on activity. It can't exist independently.

5. Without using fragment in activity, we can't create multi-pane UI; but by using multiple fragments in a single activity, we can create multi-pane UI.
6. If we create a project using only activities, it's difficult to manage; but if we use fragments, the project structure will be good and we can handle it easily.
7. An activity may contain 0 or more fragments. A fragment can be reused in multiple activities, so it acts like a reusable component in activities.
8. Activity has own life cycle, but fragment has their own life cycle.
9. For activity, we need to mention it in the manifest; for fragment, it's not required.

Practical No. 04

TYPES OF LAYOUTS

Note: we have already seen how to use ConstraintLayout and LinearLayout in above mentioned practicals so here I am discussing the other Layouts like TableLayout,GridLayout,ListView,FrameLayout.

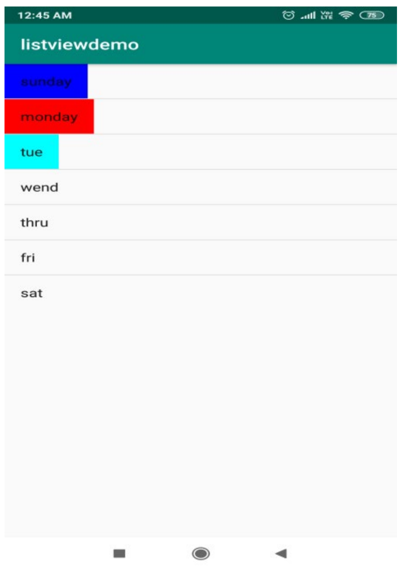
1) ListView Demo

Activity_Main.xml →

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayoutxmlns:android="http://schemas.android.com/apk/res/andro
id"
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/listview1"
android:layout_width="wrap_content "
android:layout_height="wrap_content">
</ListView>
</android.support.constraint.ConstraintLayout>
```

MainActivity.java →

```
package com.anu.listviewdemo;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener
 {
ListView l;
   String[] days={"sunday","monday","tue","wend","thru","fri","sat"};

   @Override
   protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```


12:45 AM
listviewdemo
sunday
monday
tue
wend
thru
fri
sat

```
setContentView(R.layout.activity_main);
        l=(ListView) findViewById(R.id.listview1);
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,days);
l.setAdapter(adapter);
l.setOnItemClickListener(this);

    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id)
    {
TextView temp=(TextView) view;
Toast.makeText(this,temp.getText()+"  "+position,Toast.LENGTH_SHORT).show();
        if(position ==0)
        {
temp.setBackgroundColor(Color.BLUE);
        }
        if(position ==1)
        {
temp.setBackgroundColor(Color.RED);
        }
        if(position ==2)
        {
temp.setBackgroundColor(Color.CYAN);
        }
    }
}
```

OUTPUT: onclick of list item the background color of that list item row, is getting changed.

Note: As you can see in the above code events are created only for 1st three list items. If you want you can have for all list items mentioned in array "days".

2 TableLayout Demo

Activity_Main.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"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

<TableLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">

<TableRow>

<Button
            android:id="@+id/btnclk"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="click" />

<Button
      android:id="@+id/button2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Button" />
</TableRow>

<TableRow>
<TextView
      android:id="@+id/textView3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
     android:textSize="20sp"
      android:textColor="@color/colorAccent"
      android:text="TextView" />

<TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:textSize="20sp"
        android:textColor="@color/colorAccent"
        android:text="TextView" />

<EditText
        android:id="@+id/txtmsg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:text="Name" />

</TableRow>

<ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/image2"
            android:id="@+id/img1"
            />
</TableLayout>

</LinearLayout>
```

MainActivity.java :-

```
package com.anu.framelayouttest;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
   Button btn;
  EditText txt;

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

txt=(EditText) findViewById(R.id.txtmsg);
      btn=(Button) findViewById(R.id.btnclk);
      btn.setOnClickListener(this);
    }

    @Override
    public void onClick(View v)
    {
if(v.getId()==R.id.btnclk)
  {
      txt.setText("hello");
  }
    }

    }
```

OUTPUT: As per above code event(onClick) is fired just on one button ,which displays “hello” ,in the targeted textview.

3 FrameLayout

MainActivity.java file:

```
package com.anu.framelayouttest;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
   ImageView i1,i2;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        i1=(ImageView) findViewById(R.id.imageview1);
        i2=(ImageView) findViewById(R.id.imageview2);
        i1.setOnClickListener(this);
        i2.setOnClickListener(this);
    }
    @Override
    public void onClick(View v)
    {
        if ((v.getId()==R.id.imageview1))
        {
            i2.setVisibility(View.VISIBLE);
            i1.setVisibility(View.INVISIBLE);
        }
        else
        {
            i1.setVisibility(View.VISIBLE);
            i2.setVisibility(View.INVISIBLE);

        }
    }
}
```

Activity_Main.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"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

<ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/image1"
      android:id="@+id/imageview1"
   />

<ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/image2"
      android:id="@+id/imageview2"
      android:visibility="invisible"
      />

</FrameLayout>
```

OUTPUT:

on click of 1st image 2nd image appears and on click on 2nd again 1st image appears.

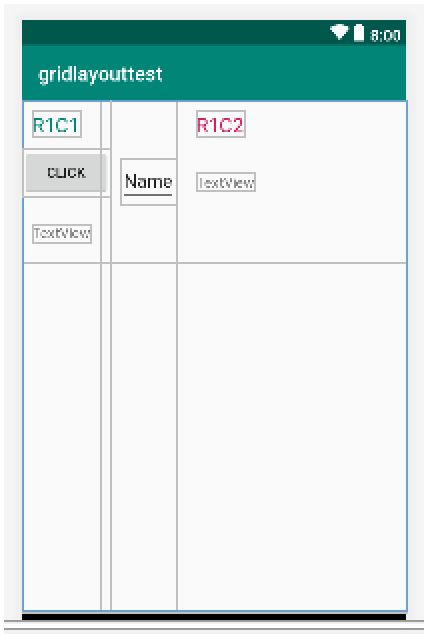
[*Visibility property is focused here.]*

4 Use of **GridLayout**

Activity_Main.xml→

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
   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:rowCount="2"
   android:columnCount="3"
   tools:context=".MainActivity">

<TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="R1C1"
      android:textColor="@color/colorPrimary"
      android:textSize="20dp"
      android:layout_margin="10dp"
      />

<android.support.v4.widget.Space
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

<TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
     android:textSize="20dp"
      android:textColor="@color/colorAccent"
      android:text="R1C2"
      android:layout_margin="10dp"
      />

<Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"

      android:text="click" />

<EditText
      android:id="@+id/editText"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:text="Name"
        android:layout_margin="10dp"/>

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:layout_margin="10dp"/>

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:layout_margin="10dp"/>

    </GridLayout>
```



Layout Introduction:

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup**objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	**Linear Layout** LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	**Relative Layout** RelativeLayout is a view group that displays child views in relative positions.
3	**Table Layout** TableLayout is a view that groups views into rows and columns.
4	**Absolute Layout** AbsoluteLayout enables you to specify the exact location of its children.
5	**Frame Layout** The FrameLayout is a placeholder on screen that you can use to display a single view.
6	**List View** ListView is a view group that displays a list of scrollable items.
7	**Grid View** GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id**This is the ID which uniquely identifies the view.
2	**android:layout_width**This is the width of the layout.

3	**android:layout_height**This is the height of the layout
4	**android:layout_marginTop**This is the extra space on the top side of the layout.
5	**android:layout_marginBottom**This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft**This is the extra space on the left side of the layout.
7	**android:layout_marginRight**This is the extra space on the right side of the layout.
8	**android:layout_gravity**This specifies how child Views are positioned.
9	**android:layout_weight**This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:layout_x**This specifies the x-coordinate of the layout.
11	**android:layout_y**This specifies the y-coordinate of the layout.
12	**android:layout_width**This is the width of the layout.
13	**android:layout_width**This is the width of the layout.
14	**android:paddingLeft**This is the left padding filled for the layout.
15	**android:paddingRight**This is the right padding filled for the layout.
16	**android:paddingTop**This is the top padding filled for the layout.
17	**android:paddingBottom**This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

Practical No. 05[A]

AppBar

The *app bar*, also known as the *action bar*, is one of the most important design elements in your app's activities, because it provides a visual structure and interactive elements that are familiar to users. Using the app bar makes your app consistent with other Android apps, allowing users to quickly understand how to operate your app and have a great experience. The key functions of the app bar are as follows:

- A dedicated space for giving your app an identity and indicating the user's location in the app.
- Access to important actions in a predictable way, such as search.
- Support for navigation and view switching (with tabs or drop-down lists).

Creating appbar:

Styles.xml →

```
<resources>
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
<!-- Customize your theme here. -->
<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
</resources>
```

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">

<include
      layout="@layout/toolbar_layout" />

</LinearLayout>
```

MainAcivity.java→

```
package com.anu.appbar;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.*;

public class MainActivity extends AppCompatActivity
{
  Toolbar tbr;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tbr=(Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(tbr);

    }
}
```

toolbar_layout.xml →

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    android:id="@+id/toolbar"  >

</android.support.v7.widget.Toolbar>
```

Adding menu to appbar/toolbar→

1) create new resource file by right on res→new→new resource file

Name: appbarmenu **Resource type: menu**

Appbarmenu.xml file →

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

<item
      android:id="@+id/a1"
      android:title="File"
      android:icon="@mipmap/ic_launcher"
      app:showAsAction="ifRoom"/>
```

```
<item
        android:id="@+id/a2"
        android:title="New"
        app:showAsAction="never"
        />
</menu>
```

MainActivity.java →

```
package com.anu.appbar;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.*;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
{
  Toolbar tbr;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tbr=(Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(tbr);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater mi=getMenuInflater();
        mi.inflate(R.menu.app_bar_menu,menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            case R.id.a1:
                Toast.makeText(this, "File menu clkd", Toast.LENGTH_SHORT).show();
                return true;
            case R.id.a2:
```

```
            Toast.makeText(this, "New menu clkd", Toast.LENGTH_SHORT).show();
            return true;
            default:return super.onOptionsItemSelected(item);
      }
   }
}
```

OUTPUT: here one icon as a tool is getting added into the actionbar and 2menu items like File ,New. OnClick of these items Toast appears .

Practical on Fragments 5[B]

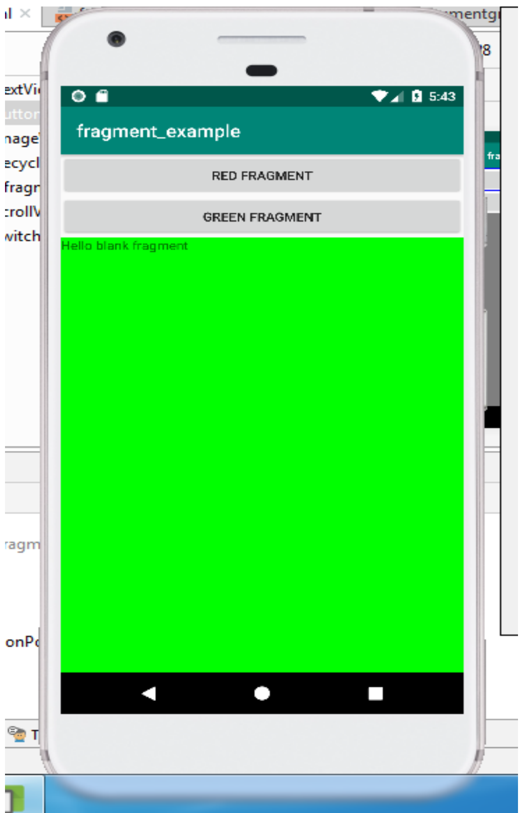
(activity to fragment communication)

Activity_Main.xml as follow→

```
<?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">

<Button
      android:id="@+id/btnred"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="changefragment"
      android:text="red fragment" />

<Button
      android:id="@+id/btngreen"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="changefragment"
      android:text="green fragment" />
<fragment
      android:name="com.example.mahesh.fragment_example.fragmentred"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:id="@+id/fragmentplace"
>

</fragment>
</LinearLayout>
```

MainActivity.java as follow→

```
package com.example.mahesh.fragment_example;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.*;
import android.view.View;
import android.widget.Button;


public class MainActivity extends AppCompatActivity
{
Button btnred,btngreen;

    @Override
protected void onCreate(Bundle savedInstanceState)
    {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
}
public void changefragment(View v1)
    {
        Fragment fg;
if(v1==findViewById(R.id.btnred))
        {
            fg= new fragmentred();
            FragmentManager fm = getSupportFragmentManager();
            FragmentTransaction ft= fm.beginTransaction();
            ft.replace(R.id.fragmentplace,fg);
            ft.commit();

        }
if(v1==findViewById(R.id.btngreen))
        {
            fg = new fragmentgreen();
            FragmentManager fm = getSupportFragmentManager();
            FragmentTransaction ft = fm.beginTransaction();
            ft.replace(R.id.fragmentplace, fg);
            ft.commit();
        }
        }
    }
```



Create 2 fragments named as fragmentred and fragmentgreen.

Right click app→res→layout→fragment→blanlfragment

fragmentgreen.xml as follow:

```
<?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"
   android:background="#00FF00"
   tools:context=".fragmentgreen">

<!-- TODO: Update blank frament layout -->
<TextView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

Fragmentred.xml as follow:

```
<?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"
   android:background="#FF0000"
   tools:context=".fragmentred">

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

</FrameLayout>
```

Practical no. 5C
UI Components

Toast class

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime. It doesn't block the user interaction.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image.

Constants of Toast class

There are only 2 constants of Toast class which are given below.

Constant	Description
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

Methods of Toast class

The widely used methods of Toast class are given below.

Methods	Description
public static Toast makeText(Context context, CharSequence text, int duration)	makes the toast containing text and duration.
public void show()	displays toast.
ublic void setMargin (float horizontalMargin, float verticalMargin)	changes the horizontal and vertical margin difference.

Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.Since Android 4.0, there is another type of toggle button called *switch* that provides slider control.Android ToggleButton and Switch both are the subclasses of CompoundButton class.

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">


<Button
      android:id="@+id/btn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Button" />

<ImageButton
      android:id="@+id/imgbtn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      app:srcCompat="@mipmap/ic_launcher" />

<EditText
      android:id="@+id/txt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="20dp"
      android:hint="edit text"
      android:inputType="textPersonName" />

<CheckBox
      android:id="@+id/cb1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="20dp"
      android:text="music" />

<CheckBox
      android:id="@+id/cb2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="playing"
      android:textSize="20dp"
      style="?android:attr/starStyle"       />

<RadioGroup
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/rbg"
>

<RadioButton
         android:id="@+id/rb1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
        android:textSize="20dp"
        android:text="male" />

<RadioButton
        android:id="@+id/rb2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:text="female" />
</RadioGroup>

<ToggleButton
      android:id="@+id/tb"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="20dp"
      android:text="ToggleButton" />


</LinearLayout>
```

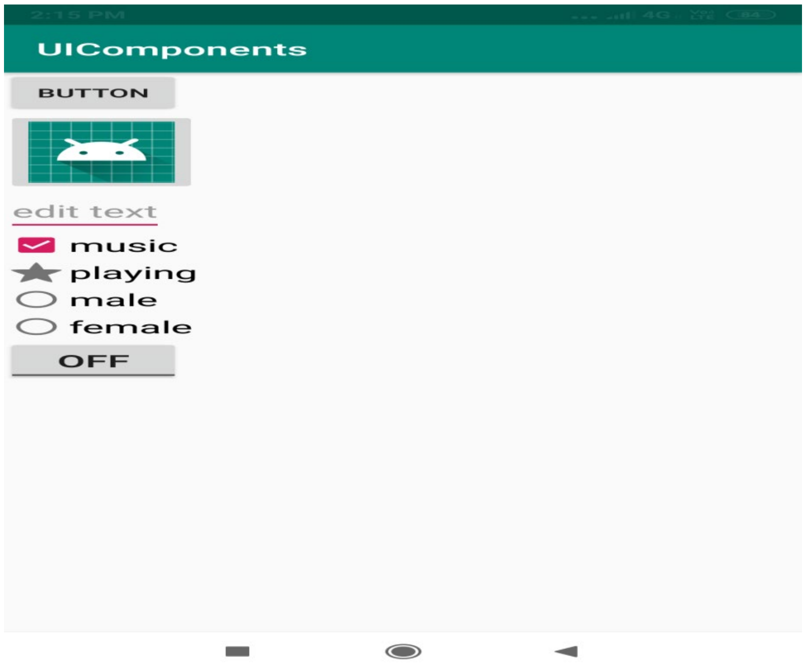
MainActivity.java→

```
package com.anu.uicomponents;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Display;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.ToggleButton;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
{
   Button btn;
   ImageButton imgbtn;
   CheckBox cb1,cb2;
   RadioGroup rgb;
   RadioButton rb1,rb2;
   ToggleButton tb;
   EditText txt;

   @Override
```

```
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  btn=(Button) findViewById(R.id.btn);
  imgbtn=(ImageButton) findViewById(R.id.imgbtn);
  cb1=(CheckBox) findViewById(R.id.cb1);
  cb2=(CheckBox) findViewById(R.id.cb2);
  rgb=(RadioGroup) findViewById(R.id.rbg);
  rb1=(RadioButton) findViewById(R.id.rb1);
  rb2=(RadioButton) findViewById(R.id.rb2);
  tb=(ToggleButton) findViewById(R.id.tb);
  txt=(EditText) findViewById(R.id.txt);

  btn.setOnClickListener(new View.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
     Disp("button clkd");

    }
  });

  imgbtn.setOnClickListener(new View.OnClickListener()
    {
     @Override
     public void onClick(View v)
     {
        Disp("image button clkd");
     }
    });

  cb1.setOnClickListener(new View.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      if(cb1.isChecked())
      {
        Disp("music clkd");
      }
      else
      {
        Disp("not clkd");
      }
    }
  });
  cb2.setOnClickListener(new View.OnClickListener()
  {
    @Override
```

```
            public void onClick(View v)
            {
                if(cb2.isChecked())
                {
                    Disp("playing clkd");
                }
                else
                {
                    Disp("not clkd");
                }
            }
        });

        rb1.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
            {
                if(rb1.isChecked())
                {
                    Disp("male radio button clkd");
                }
                else
                {
                    Disp("female radio button clkd");
                }
            }
        });
        tb.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                if(tb.isChecked())
                {
                    Disp("toogle button is on");
                }
                else
                {
                    Disp("toogle button is off");
                }
            }
        });
    }
    public void Disp(String s)
    {
        Toast.makeText(this,s,Toast.LENGTH_SHORT).show();
    }
}
```

OUTPUT: respected events are fired on each UI component.



Practical no. 6A

Menu

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu*button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

Options menu and app bar

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

Context menu and contextual action mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

MainActivity.java →

```
package com.anu.menu_example;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;


public class MainActivity extends AppCompatActivity
{

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

  }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater mi=getMenuInflater();
        mi.inflate(R.menu.mymenu,menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch(item.getItemId())
        {
            case R.id.new1:
                Toast.makeText(this, "new selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.save:
                Toast.makeText(this, "save selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.search:
                Toast.makeText(this, "search selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.share:
                Toast.makeText(this, "share selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.exit:
                Toast.makeText(this, "exit selected", Toast.LENGTH_LONG).show();
                return true;
            default:
                Toast.makeText(this, "default", Toast.LENGTH_LONG).show();
        return super.onOptionsItemSelected(item);
        }
    }
}
```

create new resource file named as (mymenu)

res→new→new resource file

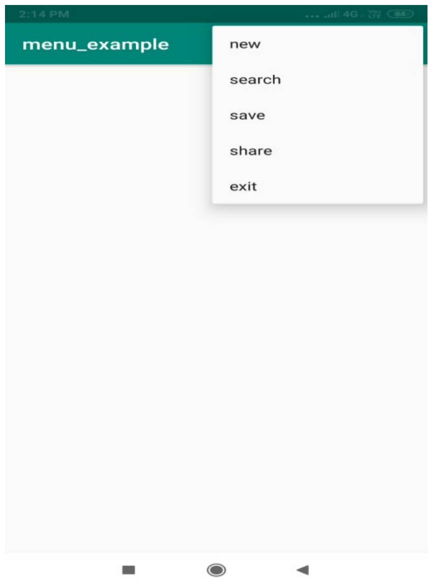
name:mymenu

resource type:menu

root elemtn:menu

mymenu.xml→

```
<?xml version="1.0" encoding="utf-8"?>
<menu
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
>
<item
   android:id="@+id/new1"
   android:title="new"/>
<item
      android:id="@+id/search"
      android:title="search"/>
<item
      android:id="@+id/save"
      android:title="save"/>
<item
      android:id="@+id/share"
      android:title="share"/>
<item
      android:id="@+id/exit"
      android:title="exit"/>

</menu>
```



Practical no. 7
Activity to Activity communication

About Intent:

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.An intent can contain data via a Bundle. This data can be used by the receiving component.

Program no.01:

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">

<Button
      android:id="@+id/btnact"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="call2ndact"       />
</LinearLayout>
```

To create new activity rightclick app→res→layout→new→activity→Empty Activity

[name it -- eg. Newactivity]

Create Newacitvity.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=".LinearLayout">

<Button
        android:onClick="goRed"
        android:id="@+id/btnred"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RED"
        android:textSize="30dp"
        />

<Button
        android:onClick="goGreen"
        android:id="@+id/btngreen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="GREEN"

        android:textSize="30dp"
        />

</LinearLayout>
```

MainActivity.java→

```
package com.anu.backgoundcolor;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity
{


    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btna=(Button) findViewById(R.id.btnact);
        btna.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent i=new Intent(getApplicationContext(),LinearLayout.class);
                startActivity(i);
        }
      });
    }
}
```

To create new activity rightclick app→res→layout→new→activity→Empty Activity
[name it -- eg. Linearlayout]

LinearLayout.java →

```
package com.anu.backgoundcolor;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class LinearLayout extends AppCompatActivity
{
    View view;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_linear_layout);
        view=this.getWindow().getDecorView();
// view.setBackgroundResource(R.color.colorAccent);
}
    public void goRed(View v)
    {
        view.setBackgroundResource(R.color.colorAccent);
    }
    public void goGreen(View v)
    {
        view.setBackgroundResource(R.color.colorPrimary);          }
}
```