Answers **(5)**	Coding Efficiency **(5)**	Viva **(5)**	Timely Completion **(5)**	Total **(20)**	Dated Sign of Subject Teacher

Expected Date of Completion:-------------------------

Actual Date of Completion:----------------------------

Experiment No: Group B-3

Problem Definition:

Write an application that draws basic graphical primitives on the screen.

3.1 Prerequisite:

Basics concepts of Java

3.2 Learning Objective:

1. Introduction to Android Platform.
2. Android Versions and Installing Android SDK and updating SDK components.

3.3 Theory:

3.3.1 Introduction

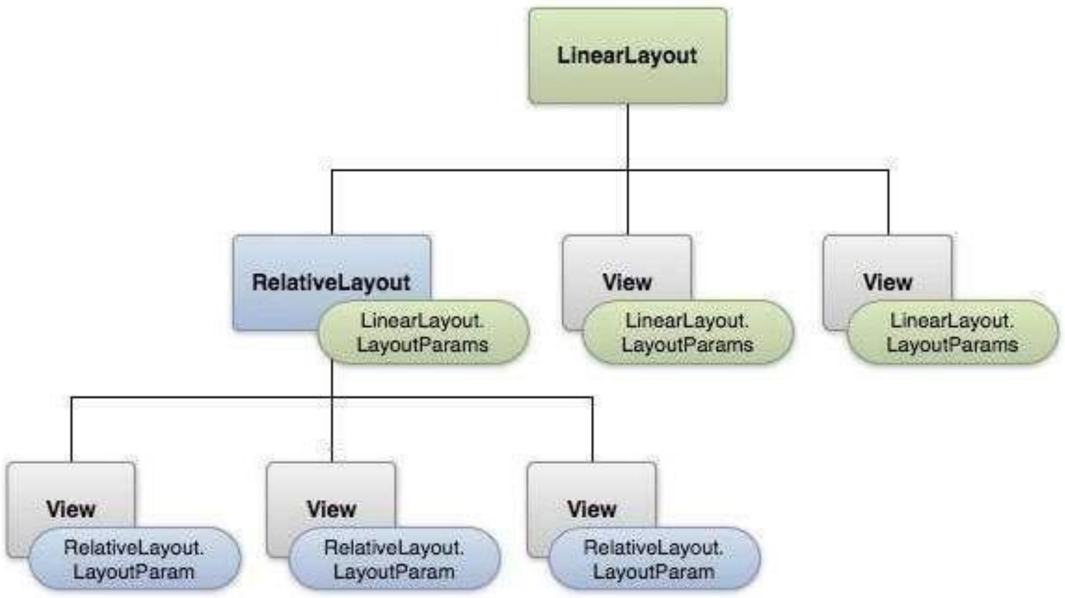
The basic building block for user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text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

Following is a simple example of XML file having Linear Layout –

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="f
  ill_parent"
  android:orientation="ve
  rtical" >
```


LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

```
    <TextView
    android:id="@+id/text"
    android:layout_width="wra
    p_content"
    android:layout_height="wra
    p_content"
    android:text="This is a
    TextView" />

  <Button
```

```
    android:id="@+id/button"
    android:layout_width="wra
    p_content"
    android:layout_height="wra
    p_content"
    android:text="This is a
    Button" />

  <!-- More GUI components go here -->

</LinearLayout>
```

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below –

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

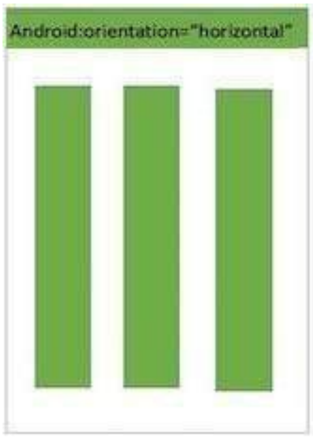
Android Layout Types:

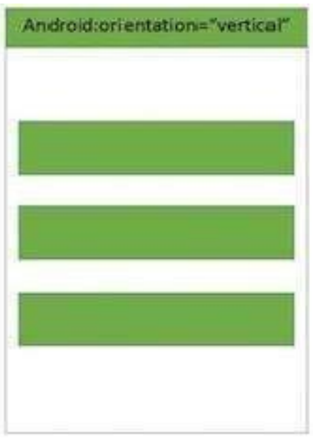
There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

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

Linear Layout:

Android Linear Layout is a view group that aligns all children in either vertically or horizontally.





LinearLayout Attributes:-

Following are the important attributes specific to LinearLayout –

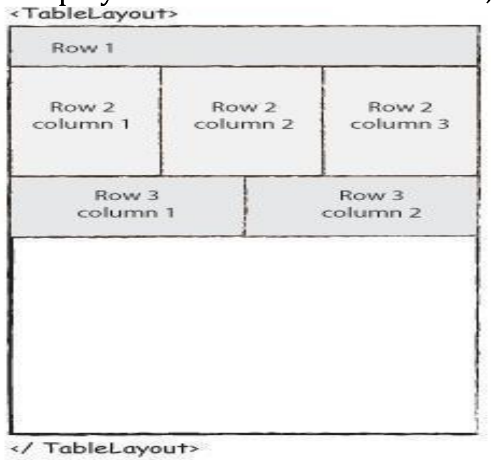
Sr. No.	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the layout.
2	**android:baselineAligned :** This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	**android:baselineAlignedChildIndex :** When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

Table Layout:

Android TableLayout going to be arranged groups of views into rows and columns.

You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

TableLayout containers do not display border lines for their rows, columns, or cells.



TableLayout Attributes

Following are the important attributes specific to TableLayout –

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

Widgets:

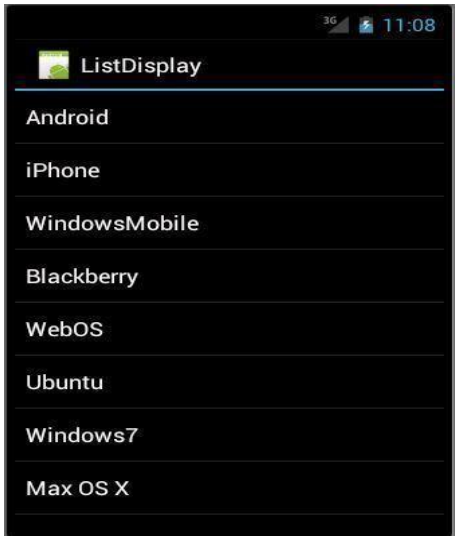
Widgets are an essential aspect of home screen customization. You can imagine them as "at-a- glance" views of an app's most important data and functionality that is accessible right from the user's home screen. Users can move widgets across their home screen panels, and, if supported, resize them to tailor the amount of information within a widget to their preference.

List view:

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **List View** and **Grid View** are subclasses of **Adapter View** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.



List View

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are **ArrayAdapter**, **BaseAdapter**, **CursorAdapter**,**SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

List View Attributes:

Following are the important attributes specific to GridView –

Sr. No	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the layout.

2	**android:divider :**This is drawable or color to draw between list items.
3	**android:dividerHeight :**This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	**android:entries :**Specifies the reference to an array resource that will populate the ListView.
5	**android:footerDividersEnabled :**When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	**android:headerDividersEnabled :**When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter:

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **Text View**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array –

ArrayAdapter adapter = new

ArrayAdapter<String>(this,R.layout.ListView,StringArray); Here are

arguments for this constructor –

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows –

```
ListView listView = (ListView)
findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

Option Menu :

Menu items are a very old and famous user interface entity. Almost all users are comfortable using menus in their apps. Android provides an easy and flexible infrastructure to add menus in your app. It lets you create menus through XML resources or directly via code. Based on the menu item clicked, a certain action can be performed by your app.

Code for Activity_main.xml:

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

  android:layout_width="match_parent"

 android:layout_height="match_parent">

<ImageView

 android:layout_width="match_parent"

 android:layout_height="match_parent"

  android:id="@+id/imageView" /> </RelativeLayout>
```

Code for MainActivity.java:

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

    android:layout_width="match_parent"

   android:layout_height="match_parent">

  <ImageView        android:layout_width="match_parent"

    android:layout_height="match_parent"

 android:id="@+id/imageView" />

 </RelativeLayout>

 {    @Override

   public void onCreate(Bundle savedInstanceState)

{

  super.onCreate(savedInstanceState);
```

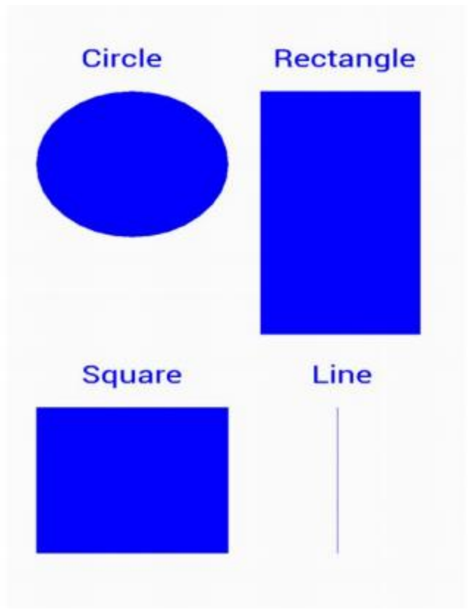
```
setContentView(R.layout.activity_main);
 //Creating a Bitmap
  Bitmap bg = Bitmap.createBitmap(720, 1280, Bitmap.Config.ARGB_8888);
 //Setting the Bitmap as background for the ImageView
 ImageView i = (ImageView) findViewById(R.id.imageView);
  i.setBackgroundDrawable(new BitmapDrawable(bg));
   //Creating the Canvas Object
  Canvas canvas = new Canvas(bg);
  //Creating the Paint Object and set its color & TextSize
Paint paint = new Paint();
paint.setColor(Color.BLUE);
paint.setTextSize(50);
//To draw a Rectangle
canvas.drawText("Rectangle", 420, 150, paint);
canvas.drawRect(400, 200, 650, 700, paint);
//To draw a Circle
canvas.drawText("Circle", 120, 150, paint);
  canvas.drawCircle(200, 350, 150, paint);
//To draw a Square
 canvas.drawText("Square", 120, 800, paint);
 canvas.drawRect(50, 850, 350, 1150, paint);
//To draw a Line
canvas.drawText("Line", 480, 800, paint);
```

```
    canvas.drawLine(520, 850, 520, 1150, paint);

   }

}
```

Output:



3.3.2 Questions:

1. Does android support other languages than Java?
2. What are the advantages of Android?
3. Define Android Toast.
4. What is Innet?
5. What are Android Layout Types.

Conclusion: Thus a Simple Android Application that draws basic Graphical Primitives on the screen is developed and executed successfully.