User Interface Programming

Amit Gulati, amit.gulati@gmail.com

1

Android User Interface

- Mixed Method
  - Android supports both Declarative & Procedural forms of user interface development.
  - You can declare UI controls (Button, Label etc.) in the XML file and then refer to these elements in java code.
  - Any changes required during runtime can be made using Java.
  - Lot less code to write as compared to procedural UI development.
- Procedural Method
  - The entire UI is created using Java code.

2

Android User Interface

- Creating a simple Activity Screen

3

3

Procedural Method

- Write Java code to create View for the Activity.

```
public class SimpleProcActivity extends Activity {

    private LinearLayout mainLayout;
    private TextView helloTextView;
    private ImageView androidLogoImageView;
```

  - Create and configure (with the same properties as in the XML file)
    - Linear Layout
    - TextView
    - ImageView
  - Add TextView and ImageView to the LinearLayout.
  - Set LinearLayout as the content view.

4

4

Procedural Method

- Create and configure the LinearLayout
  - LinearLayout created using XML

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

  - LinearLayout created using Java

```
//Create the main layout
mainLayout = new LinearLayout(this);
mainLayout.setLayoutParams(
                new LayoutParams(LayoutParams.FILL_PARENT,
                                 LayoutParams.FILL_PARENT));
mainLayout.setOrientation(LinearLayout.VERTICAL);
```

5

5

Procedural Method

- Create and Configure the TextView
  - TextView created using XML

```
<TextView
  android:id="@+id/textview_hello"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Hello Android"
  android:textSize="20dp"
  android:gravity="center"
  android:paddingBottom="30px"/>
```

  - TextView created using Java

```
//create and initialize the text view control
helloTextView = new TextView(this);
helloTextView.setLayoutParams(
                new LayoutParams(LayoutParams.FILL_PARENT,
                                 LayoutParams.WRAP_CONTENT));
helloTextView.setText("Hello Android");
helloTextView.setTextSize(20);
helloTextView.setGravity(Gravity.CENTER);
helloTextView.setPadding(0, 0, 0, 30);
```

6

6

Procedural Method

- Create and Configure the ImageView
  - ImageView created using XML

```
<ImageView
  android:id="@+id/imageview_androidlogo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:scaleType="fitXY"
  android:src="@drawable/androidlogo"/>
```

  - ImageView created using Java

```
//create and initialize the
androidLogoImageView = new ImageView(this);
androidLogoImageView.setLayoutParams(
                new LayoutParams(LayoutParams.FILL_PARENT,
                                 LayoutParams.WRAP_CONTENT));
androidLogoImageView.setScaleType(ImageView.ScaleType.FIT_XY);
androidLogoImageView.setImageResource(R.drawable.androidlogo);
```

7

7

Procedural Method

- Add User Controls to LinearLayout

```
//add the controls to the layout
mainLayout.addView(helloTextView);
mainLayout.addView(androidLogoImageView);
```

- Set LinearLayout as the content view

```
//Set the content view
setContentView(mainLayout);
```

8

8

Mixed Method

- Define an XML layout file

```
<?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="fill_parent"
  android:orientation="vertical"
  >

  <TextView
    android:id="@+id/textview_hello"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello Android"
    android:textSize="20dp"
    android:gravity="center"
    android:paddingBottom="30px"/>

  <ImageView
    android:id="@+id/imageview_androidlogo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitXY"
    android:src="@drawable/androidlogo"/>

</LinearLayout>
```

9

9

Mixed Method

- Load the XML layout file in the Activity class.

```
public class SimpleActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

- Get access to the controls defined in the layout
  - For each control that is placed in the layout, an ID has been declared.

    android:id="@+id/textview_hello"

    android:id="@+id/imageview_androidlogo"

10

10

Mixed Method

- Get access to the controls defined in the layout

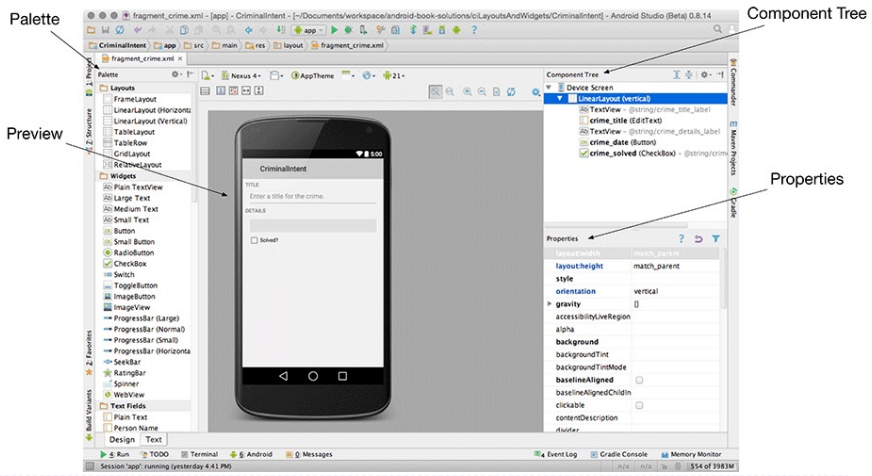
```
public class SimpleActivity extends Activity {

    private TextView helloTextView;
    private ImageView androidLogoImageView;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        initControls();
        helloTextView.setText("Android Here!!");
    }

    private void initControls() {
        helloTextView = (TextView) findViewById(R.id.textview_hello);
        androidLogoImageView = (ImageView)findViewById(R.id.imageview_androidlogo);

    }
}
```

11

11

Android User Interface

- Android Studio Graphical Layout Tool



12

12

Android User Interface

- View
  - Basic building block of Android User Interfaces.
  - View class provides the basic framework required for drawing a rectangular area on screen.
  - Rectangular area which is responsible for drawing and event handling.
  - Examples:

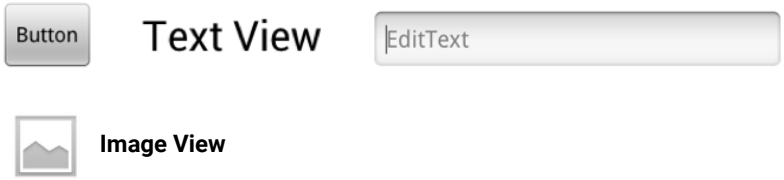


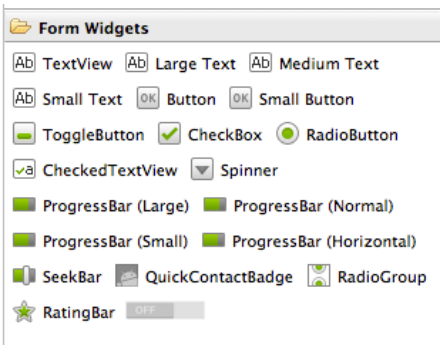
Image View

13

13

Android User Interface

- Common Views (pre-defined widgets in android)

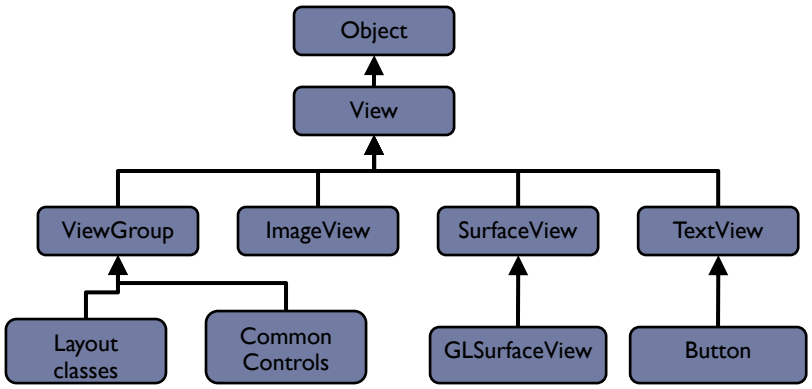


14

14

Android User Interface

- **android.view.View** class is used to encapsulate a View.



15

15

Android User Interface

- Custom View

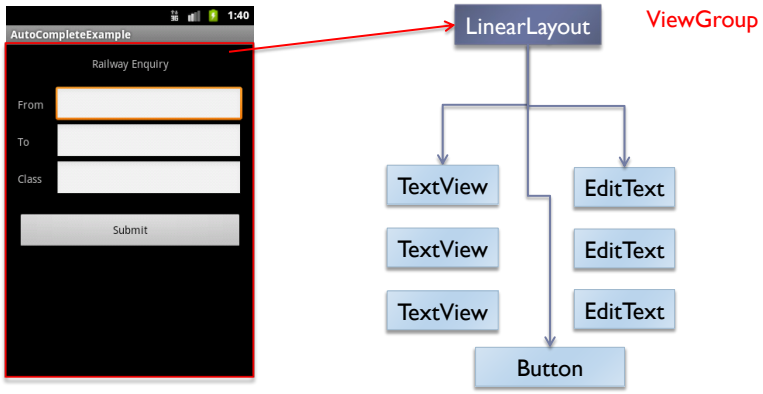
```
public class MyView extends View {

    public MyView(Context context) {
        super(context);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        canvas.drawColor(Color.CYAN);
        canvas.drawCircle(100, 100, 10, new Paint());
    }

}
```

16

16

Android User Interface

- View Group
  - A collection of View objects.



17

17

Android User Interface

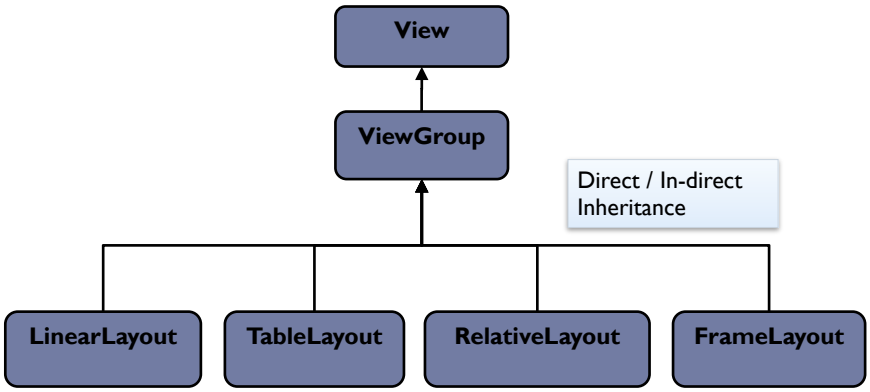
- View Group
  - Special type of View that can contain multiple child views.
  - Provides the interface for sizing and positioning children
- Layouts
  - In Android we don't use absolute location and size for a View.
  - Layout classes calculate the actual size and position of a view.
  - Layouts are the concrete sub-classes of View Groups that provide the algorithms to size and position child controls.
  - Layouts are a flexible way of arranging UI elements on the screen without worrying about their absolute locations.

18

18

Android User Interface

- Layouts

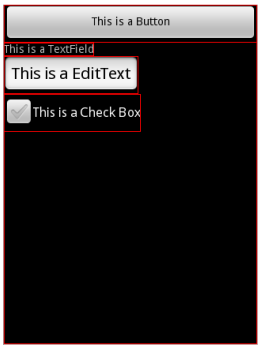


19

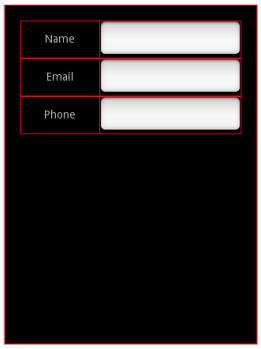
19

Layouts

- **Linear Layout**
  Layout children sequentially, either horizontally or vertically



- **Table Layout**
  Layout children in a tabular format.

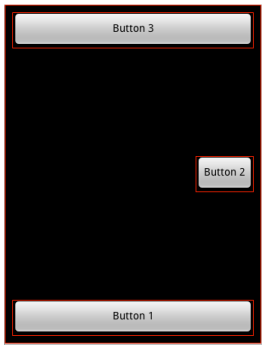


20

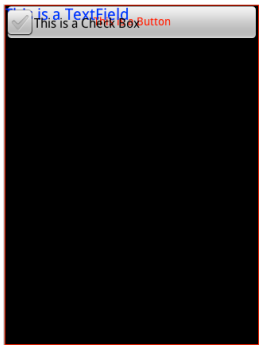
20

Layouts

- **Relative Layout**
  Layout children relative to each others location.



- **Frame Layout**
  Layout children one on top of other.

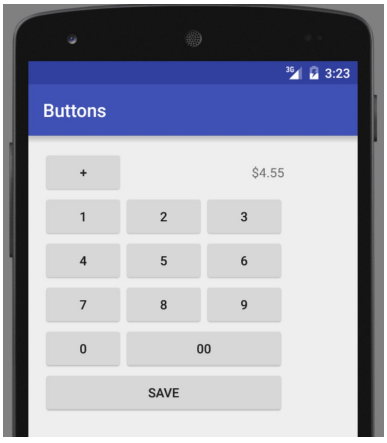


21

21

Layouts

- Grid Layout

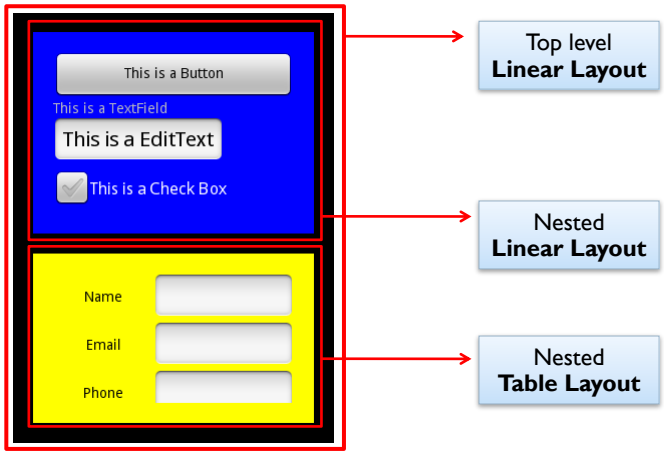


22

22

Layouts

- Nested Layouts



23

23

Dimensions

- Dimension Resource
  - Dimension units that are supported by Android

Unit	Description	postfix
Pixel	Actual pixels on screen.	px
Inches	Physical Measurement	in
Millimeter	Physical Measurement	mm
Density Independent Pixel (Preferred)	Pixels relative to 160 dpi screen	dp
Scale Independent Pixels	Generally used for fonts.	Sp

24

24

Layout Params & View Params

- Layout XML file elements and attributes
  - To build a UI, we add XML Elements related to the UI component that we need.
  - Each XML Element has attributes that are used for controlling the properties of the UI component.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:padding="16dp"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:text="New Button" />
```

25

25

Layout Params & View Params

- Layout XML file elements and attributes
  - Some attribute names begin with "layout_" and some don't
  - Layout Parameters
    - Attribute name begins with layout_
    - Direction to the widget's *parent*.
    - Tell the parent layout how to arrange the child element within the parent.
  - View Parameters
    - Attribute name does *not* begin with layout_
    - Direction to the View.
    - When it is inflated, the View calls a method to configure itself based on each of these attributes and their values.

26

26

Layout Params

- Size of View/ViewGroup
  - XML attributes

android:layout_width

android:layout_height

Possible values for these attributes

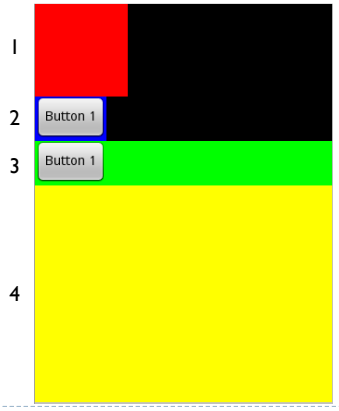
"wrap_content",	Make the View dimension (height/width) as large as the contents inside the View.
"fill_parent"/ "match_parent"	Make the View dimension (height/width) as large as the parent.
integer value, allowed.	Along with the integer the size format also needs to be specified. Specifying only and integer value is not Example :"100dp", or "100px".

27

27

Layout Params

- Size of ViewGroup
  - Specifying Sizing Layout Parameter for ViewGroup



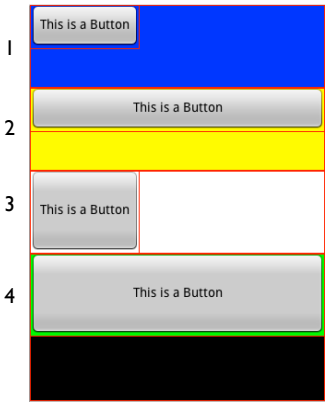
1. layout_width ="100dp"
   layout_height ="100dp"
2. layout_width ="wrap_content"
   layout_height ="wrap_content"
3. layout_width ="fill_parent"
   layout_height ="wrap_content"
4. layout_width ="fill_parent"
   layout_height ="fill_parent"

28

28

Layout Params

- Size of View
  - Specifying Layout Parameters for View/Widget



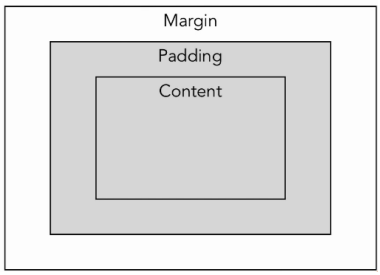
1. layout_width =“wrap_content”
   layout_height =“wrap_content”
2. layout_width =“fill_parent”
   layout_height =“wrap_content”
3. layout_width =“wrap_content”
   layout_height =“fill_parent”
4. layout_width =“fill_parent”
   layout_height =“fill_parent”

29

29

Margin and Padding

- Margin
  - Layout Parameter
  - Adds space outside the View.
- Padding
  - View Parameter
  - Adds space within the View (space between View and its children)



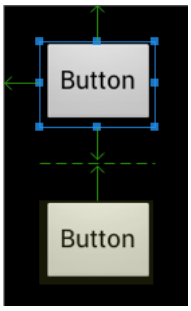
30

30

Layout Param

- Margin
  - Give instructions to the parent, for how to add spacing between children.
  - Adds space outside the View

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_margin="20dp"
    android:text="@string/button" />
```



31

31

Layout Param

- Margin
  - Following XML attribute tags are available to set the padding values

android:layout_margin, margin value to be used for all sides.

android:layout_marginLeft, margin value to be used for left.

android:layout_marginTop, margin value to be used for top.

android:layout_marginRight, margin value to be used for right.

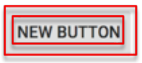
android:layout_marginBottom, margin value to be used for bottom.

32

32

View Param

- Padding
  - Add space between content of the View and the rectangle of the View.
  - Example:
    - Layout (ViewGroup)
    - Button (View)

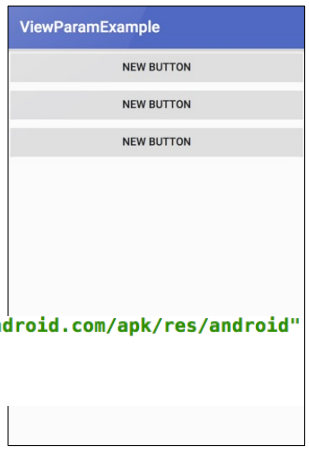


33

33

View Param

- Padding
  - Size of the View is almost the same as the content inside it.
  - Visually that sometimes does not look good especially in the case of Layouts
    - On the right, there is no space between the content of the layout i.e. Buttons and the layout itself.




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

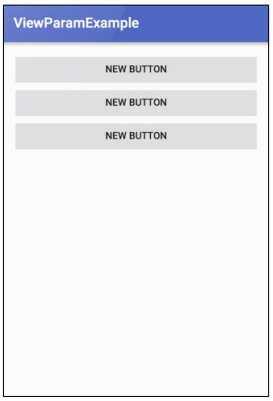
34

34

View Param

- Padding
  - Padding specifies the extra space inside the View, around the content inside the view

```
<LinearLayout xmlns:android="http://schemas.
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="15dp">
```



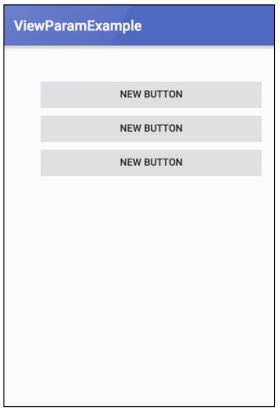
35

35

View Param

- Padding
  - Different Padding value can be specified for left, right, top, and bottom.

```
<LinearLayout xmlns:android="http://schemas.
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingRight="15dp"
    android:paddingTop="45dp"
    android:paddingLeft="50dp">
```



36

36

View Param

- Padding
  - Following XML attribute tags are available to set the padding values

android:padding, padding value (in pixels) to be used for all sides.

android:paddingLeft, padding value (in pixels) to be used for left.

android:paddingTop, padding value (in pixels) to be used for top.

android:paddingRight, padding value (in pixels) to be used for right.

android:paddingBottom, padding value (in pixels) to be used for bottom.

37

37

View Param

- Gravity
  - Gravity is used to specify the alignment of child Views, Layouts and UI controls.
  - By default Gravity fis Top Left.
    - Any children added will be placed at the top left corner of parent.

XML attribute	Java method
android:gravity	**setGravity**(int gravity)

38

38

View Param

- Gravity
  - XML attribute used for setting gravity

    android:gravity

    Potential values for this attribute

Value	
"bottom"	Place object to the bottom of its container.
"top"	Place object to the top of its container.
"center"	Place the object in the center of its container.
"right"	Place object to the right of its container.
"left"	Place object to the left of its container.
"center_horizontal"	Place object in the horizontal center of its container.
"center_vertical"	Place object in the vertical center of its container.

39

39

View Param

- Layout Gravity
  - Following XML attribute is used to specify layout gravity

    android:layout_gravity

    Possible values for this attribute

Value	
"bottom"	Place object to the bottom of its container.
"top"	Place object to the top of its container.
"center"	Place the object in the center of its container.
"right"	Place object to the right of its container.
"left"	Place object to the left of its container.
"center_horizontal"	Place object in the horizontal center of its container.
"center_vertical"	Place object in the vertical center of its container.

40

40

Layouts

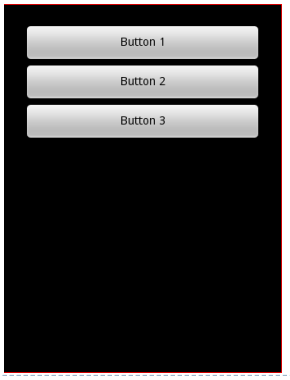
Amit Gulati, amit.gulati@gmail.com

41

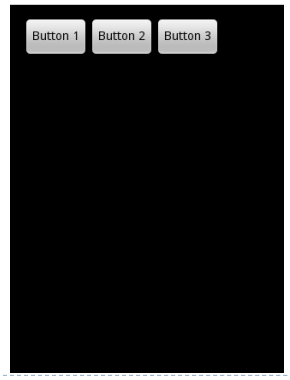
Linear Layout Basics

- One of the most basic layout.
- Lays out UI elements sequentially one after the other in a single row or column.

Vertical



Horizontal



42

42

Linear Layout

- **LinearLayout** XML tag is used to specify Linear Layout in the layout XML file.
- Example:

```
<LinearLayout   xmlns:android=http://schemas.android.com/apk/res/android
                android:orientation="vertical"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:padding="10dp">
//Add other User Interface Controls and other Layouts
……
……
</LinearLayout>
```

43

43

Linear Layout – Properties

- Orientation
  - XML attribute used to specify orientation for the a Linear layout or its descendants.

android:orientation

Possible values for the attribute
"vertical" specifies Vertical orientation for the layout.
"horizontal" specifies Horizontal orientation for the layout.

44

44

LinearLayout Layout Params

- Weight
  - Following XML attribute is used to specify layout weight

android:layout_weight

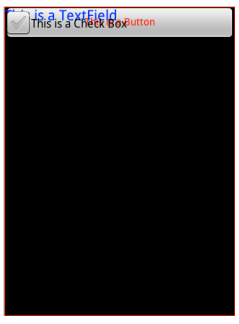
A floating point number is specified for weight.
By default weight for all Views is 0.

45

45

Frame Layout

- Frame Layout is one of the Simplest Layouts
  - Align children relative to corners of screen or center of screen
  - Great for creating a stack of items and showing only one item.



Frame Layout Containing:
1. Button (Red Text).
2. TextView (Blue Text)
3. CheckBox (Black Text)

46

46

Frame Layout

- **FrameLayout** XML tag is used to specify Frame Layout in the layout XML file.
- Example:

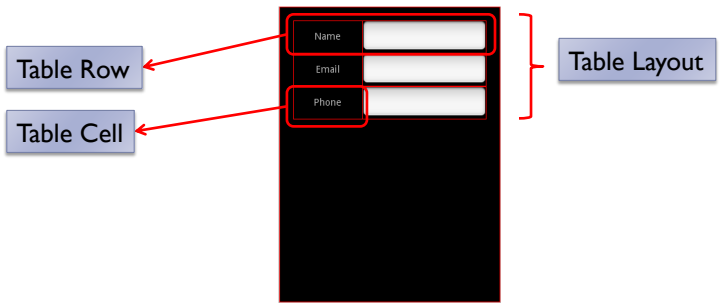
```
<FrameLayout   xmlns:android=http://schemas.android.com/apk/res/android
               android:layout_width="fill_parent"
               android:layout_height="fill_parent">
//Add other User Interface Controls and other Layouts
……
……
</FrameLayout>
```

47

47

Table Layout - Basics

- Layout child elements in tabular form.
- Each Table Layout may contain Sub-Views and Table Rows.
- Table Rows have Table Cells and a single Table Cell can contain only a single View.



48

48

Table Layout

- Creating a Table Layout using layout XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TableRow>
        <TextView android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="Name"/>

        <EditText android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="Amit Gulati"/>
    </TableRow>
    <TableRow>
        <TextView android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="email"/>
        <EditText android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="amit.gulati@gmail.com"/>
    </TableRow>
</TableLayout>
```

49

49

Table Layout – Properties

- <TableLayout> tag attributes
  - Hide columns

    android:collapseColumns

    Specify a list of column indexes (beginning from 0) that will be collapsed.

    android:collapseColumns="1, 3, 5"
  - Shrink or expand columns

    android:shrinkColumns

    Specify a list of columns that will shrink to fit inside the parent.

    android:shrinkColumns="1, 3, 5"

    andriod:stretchColumns

    Specify a list of columns that will stretch to take extra space if available.

    android:stretchColumns="1, 3, 5"

    android:stretchColumns=""*, Stretch all columns

50

50

Table Row – Child View Properties

- TableRow child view attributes
  - Specify column in which view will be placed

    android:layout_column

    Specify a column index in which the view will be placed.

    android:layout_column=" 3"
  - Span a view to multiple columns

    android:layout_span

    Specify number of columns this view can expand to.

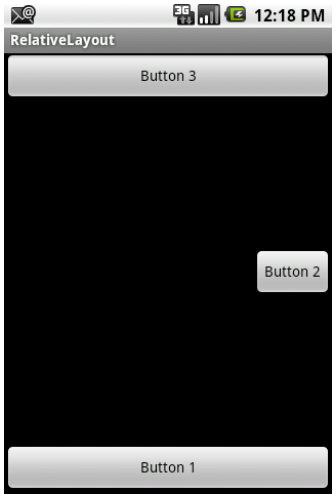
    android:layout_span="2"

51

51

Relative Layout

- Relative Layout is the most flexible layout.
- It enables you to specify where the child view controls are in relation to each other
  - You can specify view objects to be left, right, top, bottom relative to another view or parent view.
- Relative Layout allows us to create complex interfaces with ease.



52

52

Relative Layout

- **android.widget.RelativeLayout** class is used to encapsulate the Relative Layout.
- <RelativeLayout> tag is used to add Relative Layout to a XML layout resource file.

Relative Layout – Child View Properties

- Relative Layout child view properties
  - Align View relative to the parent

    android:layout_centerInParent
    android:layout_centerHorizontal
    android:layout_centerVertical
    android:layout_alignParentTop
    android:layout_alignParentBottom
    android:layout_alignParentLeft
    android:layout_alignParentRight

    Specify a "true" or false value for these attributes.
    <Button android:layout_CenterInParent="true"/>

Relative Layout

- Relative Layout child view properties
  - Align View relative to another view

android:layout_alignRight
android:layout_alignLeft
android:layout_alignTop
android:layout_alignBottom
android:layout_above
android:layout_below
android:layout_toLeftOf
android:layout_toRightOf

Specify the ID of the view relative to which you want to place the view.

<Button android:layout_below="@id/textview_id"/>

55

55

Constraint Layout

Amit Gulati, amit.gulati@gmail.com

64

Constraint Layout

- Basic Concepts
  - Constraints
  - Position / Edge Constraint
  - Opposing Constraint
  - Constraint Bias
  - Chain

65

65

Constraint Layout

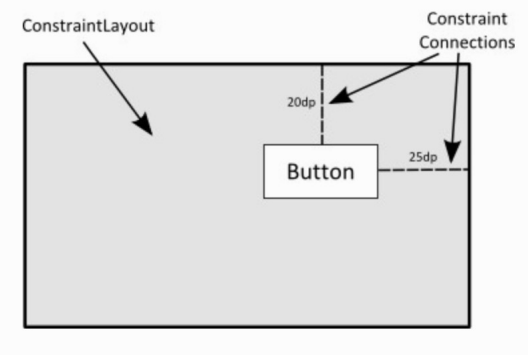
- Constraint
  - Rules that define the way in which a widget is aligned and distanced in relation to
    - Another widgets
    - Edges of Parent (Containing ConstraintLayout)
    - Special elements called Guidelines
  - Widget must have sufficient constraint connections
    - Position can be resolved by the ConstraintLayout layout engine in both the horizontal and vertical planes

66

66

Constraint Layout

- Edge/Position Constraint
  - Form of constraint that specifies a fixed distance.

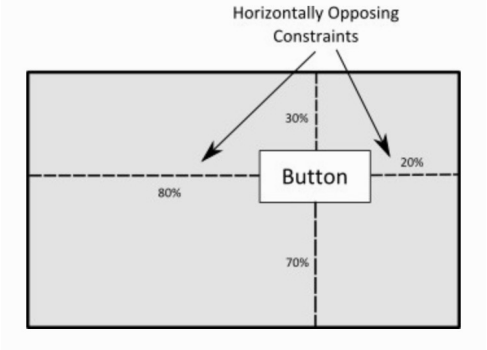


67

67

Constraint Layout

- Opposing Constraint
  - Two constraints operating along the same axis on a single widget are referred to as opposing constraints.

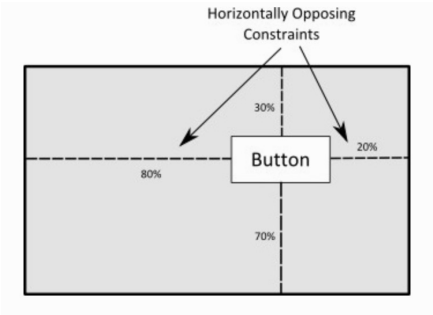


68

68

Constraint Layout

- Constraint Bias
  - If a Widget has opposing constraints, its positioning becomes percentage rather than coordinate based.
  - Constraint bias allows the positioning of a widget to be biased by a specified percentage.
  - .

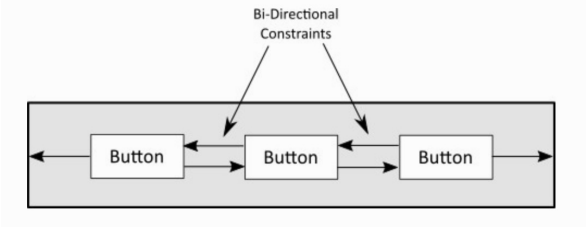


69

69

Constraint Layout

- Chains
  - Widgets in a ConstraingLayout can be grouped using chain.
  - Vertical or Horizontal
  - Spaced and Sized

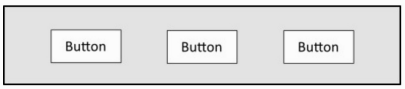


70

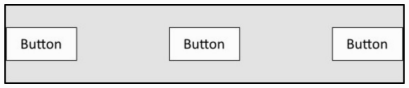
70

Constraint Layout

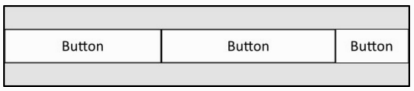
- Chains
  - Chain Styles
    - Spread Chain



    - Spread inside chain



    - Weighted Chain

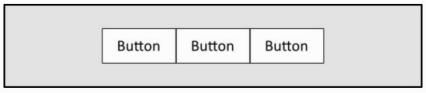


71

71

Constraint Layout

- Chains
  - Chain Styles
    - Packed Chain

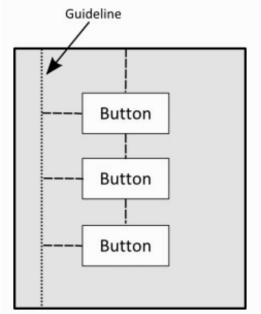


72

72

Constraint Layout

- Guidelines
  - Special elements provide an additional target for adding constraints.
  - Multiple guidelines may be added which may, in turn, be configured in horizontal or vertical orientations.

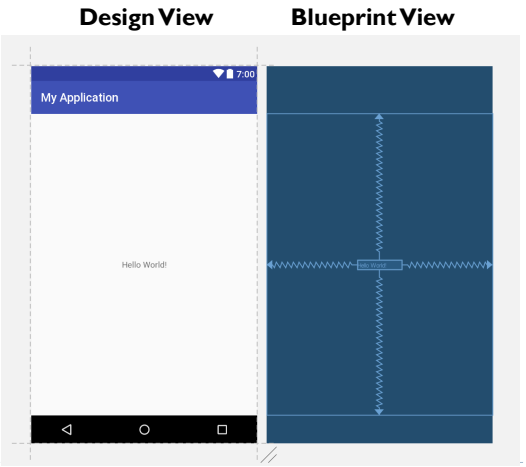


73

73

Constraint Layout

- Android Studio Designer



74

74