User Interface Basics

Dilman A. Salih

University of Halabja

Slides: 3,4,7,8,9,12,13,14,1,15,17,19,22,23,24 and 25

Outline

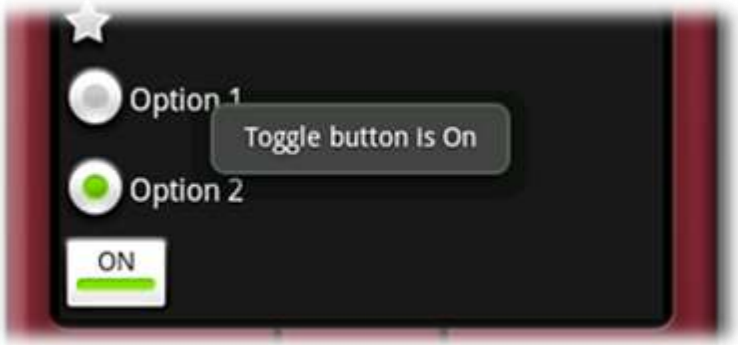
- ✓ View
- ✓ View Classes
- ✓ User Interface Elements
- ✓ Understanding Layout
  - ✓ Types of Layout
- ✓ Complex View
- ✓ View and ViewGroup Attributes
- ✓ TextView
- ✓ EditText
- ✓ Button Types
  - ✓ Regular Button
  - ✓ ImageButton
  - ✓ ToggleButton

View

- Views are the building blocks of Android application's UI.
- Activities contains Views.
- View classes represent elements on the screen and are responsible for interacting with users through events.
- Views are defined in XML files.

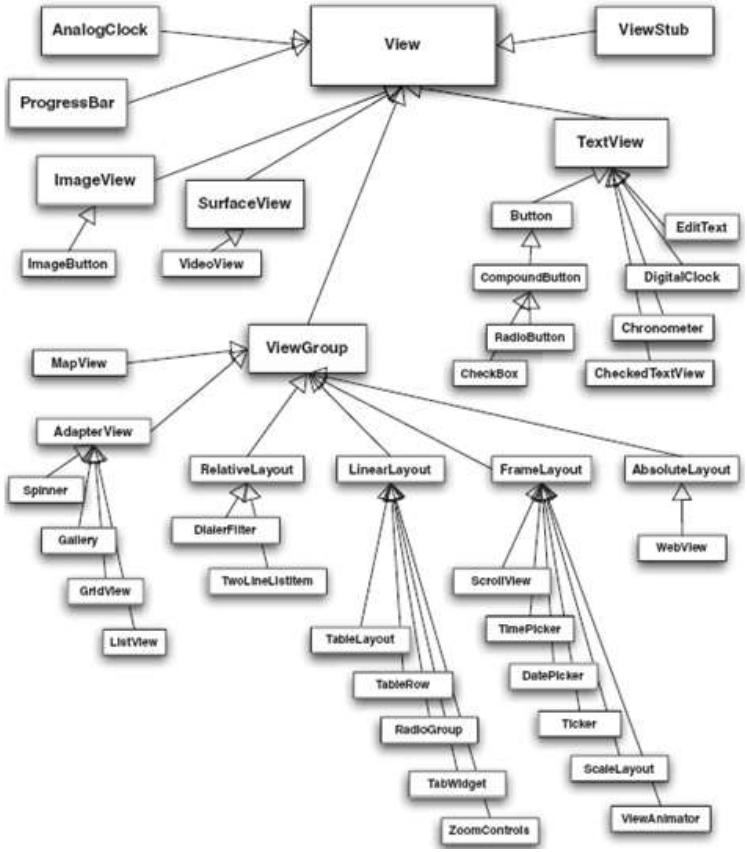
View (2)

- Every Android screen contains a hierarchical tree of view elements, these views come in a variety of shapes and sizes.
- Many of the views that you will need on a day – to – day basis are provided as part of the platform, such as text elements, input elements, images, buttons, …etc.



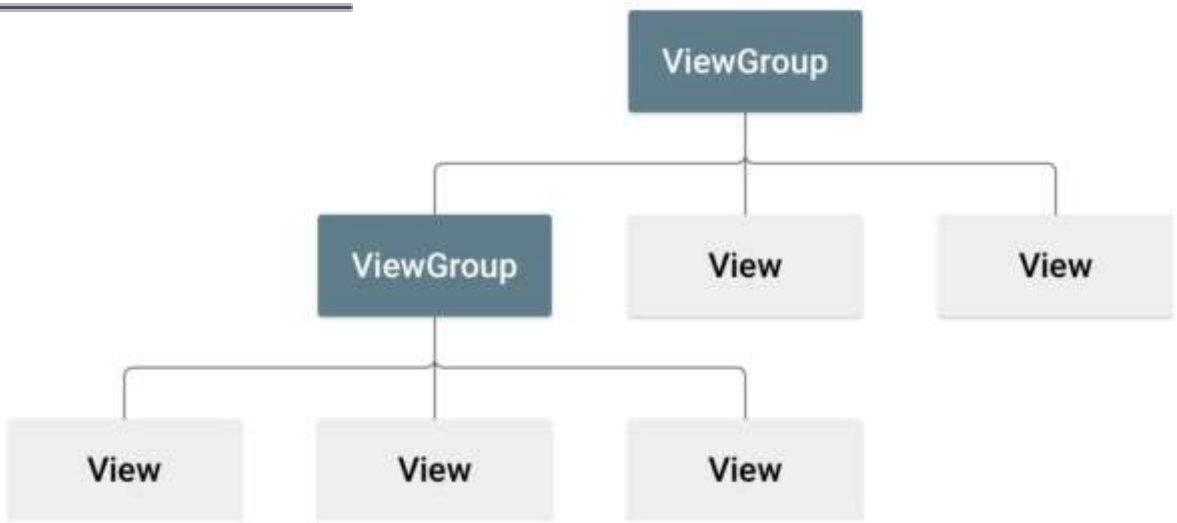
View Classes

Android provides a generous set of view classes in the **android.view** package, these classes range from familiar constructs, namely, EditText, Spinner, TextView, … etc.



User Interface Elements

- View
  - Control
  - View Group
    - Layout
    - Widget (Compound Control)



- Many pre built views
  - Button, CheckBox, RadioButton, …etc
  - TextView, EditText, ListView, …etc
  - Can be customized by extending and overriding *onDraw()*

Understanding Layout

- One of the most significant aspects of creating your UI and designing your screen is understanding Layout
- Layouts are subclasses of ViewGroup
- Android manages layouts through ***ViewGroup*** and ***Layout – Params*** objects
- ***ViewGroup*** is a view that contains other views and also provides access to the layout
- Layout can be nested (*Layout inside Layout*)

Types of Layout

There are several types of layout:

① FrameLayout

- The simplest type of the Layout object
- The Frame layout pins each child view to the **top left corner**, so the default position is the top – left corner, through you can use ***gravity*** attribute to alter its location
- Adding multiple children stacks each new child on top of the previous, with each new View obscuring the last

Types of Layout (2)

② Linear Layout

- Linear Layout is one of the simplest layout classes
- It allows you to create simple UI elements that align a sequence of child Views in either a ***vertical*** or a ***horizontal*** line
- A *vertical* layout has a column of Views, whereas a *horizontal* layout has a row of Views
- The Linear Layout supports a "***weight***" attribute for each child View which can control the relative size of each child View within the available space
- Aligns child elements, such as Buttons, EditText boxes, Pictures, …etc in a single direction
- ***orientation*** attribute defines direction:
  - android:orientation=*"vertical" or "horizontal"*

Types of Layout (3)

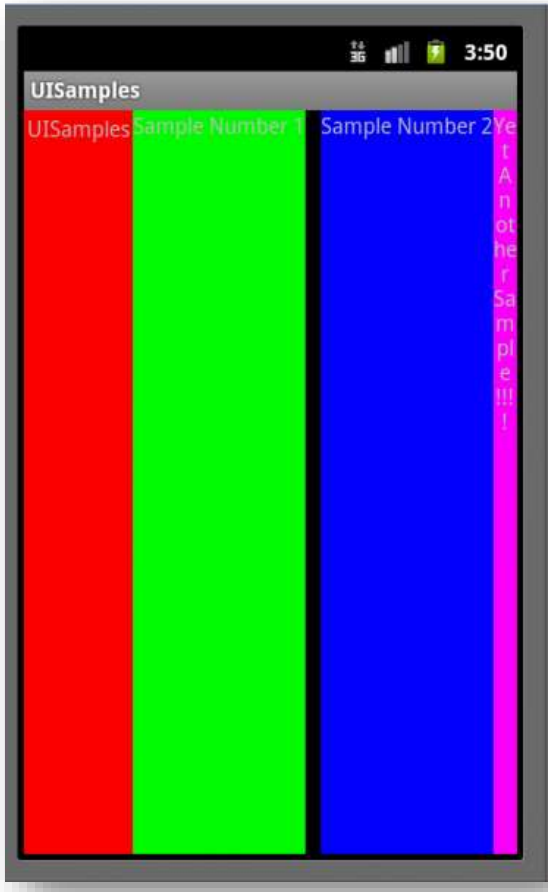
2. A. Linear Layout – *Vertical Orientation*

- a vertical layout has a column of Views

Types of Layout (4)

2. B. Linear Layout – *Horizontal Orientation*

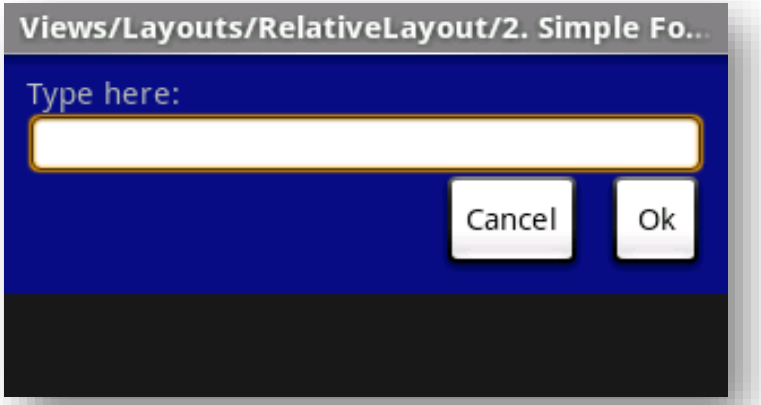
- a horizontal layout has a row of Views



Types of Layout (5)

③ Relative Layout

- One of the most flexible of the native layouts
- It lets you to define the positions of each child View relative to the others and to the screen boundaries
- Children specify position relative to parents or to each others (specified by ***ID***)
- First element listed is placed in “center”. However, other elements placed based on position to other elements



Types of Layout (6)

④ Grid Layout

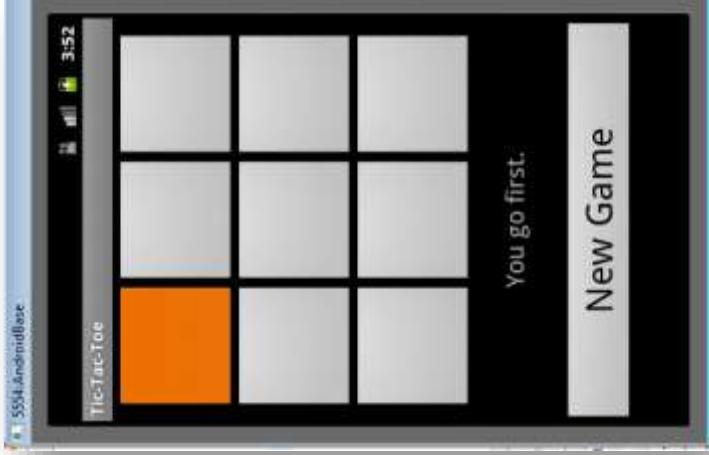
- GridLayout uses a rectangular grid of infinitely thin lines to layout Views in a series of *rows* and *columns*
- It is incredibly flexible and can be used to greatly simplify layouts and reduce or eliminate the complex nesting often required to construct UIs.
- Two Dimensional Scrollable Grid
- Items inserted into layout via a ***ListAdapter***
- GridLayout introduced in Android **4.0** (**API level 14**)

Types of Layout (7)

- Adapter-based view, for example GridLayout, should be used for any situation where you have a significant amount of data and the user wants scrolling view.
- It is a lot more efficient than having to create the entire view hierarchy up-front to display your data.
- For the same UI, a GridLayout will generally be faster and take less memory than a TableLayout.

Credit: https://bit.ly/3k8I6es

Types of Layout (8)



⑤ Table Layout

- The TableLayout lets you layout Views using a grid of rows and columns. Its can span multiple rows and columns, and columns could be set to shrink or grow.
- A Table will have as many columns as the row with the most cells. A table can also leave the cells empty.
- Table Layout containers do not display a border line for their columns, rows or cells.
- Rows normally are ***TableRows***. TableRows contain other elements, such as Buttons, Text, RadioButton, …etc

Types of Layout (9)

- TableLayout is just a layout manager, somewhat like a table in HTML. It does not itself do any scrolling, to have something that scrolls you must put the TableLayout in a ScrollView. This implies that all of the data you are displaying must be populated into the TableLayout up-front, so the ScrollView knows the total space it is to scroll in.

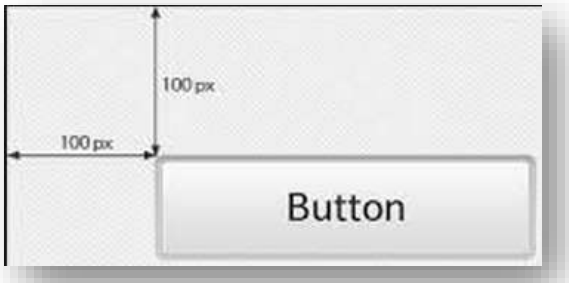
- Table layout is useful if someone have low amount of data to display because every row of table layout will have to be instantiated and it won't be **recycled**.

Credit: https://bit.ly/3k8I6es

Types of Layout (10)

⑥ Absolute Layout

- Each child View's position is defined in absolute *coordinates (x, y)*
- In this layout your layout cannot dynamically adjust for different screen resolutions and orientation
- Positions are defined for elements based on coordinates (in pixel)



Types of Layout (11)

⑦ Tab Layout

- There are two types of TabLayout:
  1. TabHost
  2. TabWidget
- Swap between views in same activity or switch between different activities

Complex View

There are some Views which can be used to display UI components:

① **ListView:** (Like JList in Java swing)

- A scrollable list displayed vertically
- Items added via a **ListAdapter** as in GridView

- Attributes:
  - android:divider drawable or color to draw between list items
  - android:dividerHeight height of the divider
  - android:entries reference to an array resource that will populate the ListView

Complex View (2)

② **GalleryView**

- Horizontally scrollable list focusing on the center of the list
- The default values for the Gallery assume you will be using ***Them_galleryItemBackground*** as the background for each View given to the Gallery from the Adapter. If you are not doing this, you may need to adjust some Gallery properties, such as the spacing
- Attributes:
  - android:animationDuration setAnimationDuration(int) sets how long a transition animation should run (in milliseconds) when layout has changed
  - android:gravity setGravity(int) specifies how to place the content of an object, both on the x- and y-axis, within the object itself
  - android:spacing setSpacing(int)
  - android:unselectedAlpha setUnselectedAlpha(float) sets the alpha on the items that are not selected

Complex View (3)

③ **GridView**

- A GridView is a ViewGroup that displays items in two – dimensional scrolling grid
- The items in the grid come from the ***ListAdapter***
- Attributes:
  - android:columnWidth setColumnWidth(int) specifies the fixed width for each column
  - android:horizontalSpacing setHorizontalSpacing(int) defines the default horizontal spacing between columns
  - android:numColumns setNumColumns(int) defines how many columns to show
  - android:stretchMode setStretchMode(int) defines how columns should stretch to fill the available empty space
  - android:verticalSpacing setVerticalSpacing(int) defines the default vertical spacing between rows

Complex View (4)

④ **SpinnerView:** (Like ComboBox in Java swing)

- Spinners provides a quick way to select one value from a set
- A scrollable drop down menu of choices
- Attribute:
  - android:entries tells the Spinner to use a particular array resource

View and ViewGroup Attributes

Attribute	Description
layout_width	specifies width of View or ViewGroup
layout_height	specifies height of View or ViewGroup
layout_marginTop	extra space on top
layout_marginBottom	extra space on bottom side
layout_marginLeft	extra space on left side
layout_marginRight	extra space on right side
layout_gravity	how child views are positioned
layout_weight	how much extra space in layout should be allocated to View (only when in LinearLayout or TableView)
layout_x	x-coordinate
layout_y	y-coordinate

Weight and *Gravity* Attributes

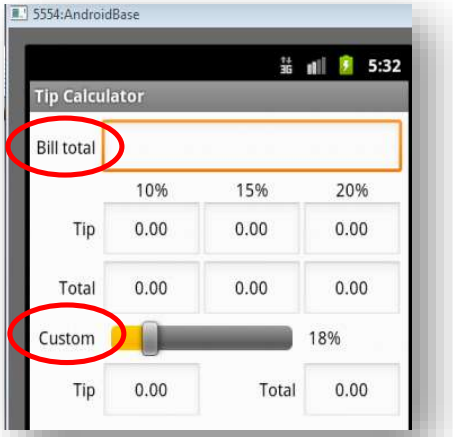
- **Weight**
  - sizing a component in a view
  - controls size relative to other components
    - *android:layout_weight*
  - relative values between 0.0 and 1.0

- **Gravity**
  - component alignment
  - *android:gravity* applies to the text within a view
  - *android:layout_gravity* applies to the view alignment
  - Possible values: "right", "left", "center", "top", "bottom“, …etc

wrap_content & match_parent

- For each of the layout elements, the constant *wrap_content* and *match_parent* (*fill_parent*) are used rather than an exact height or width in pixels
- These constants are the most powerful, technique for ensuring your layouts are screen – size and resolution independent
- The ***wrap_content*** constant sets the size of View to the minimum required to contain the contents it displays
  - *wrap_content*: use only the amount of space necessary
- The ***match_parent*** constant expands the View to match the available space within the parent View, Fragment and Activity
  - *fill_pant* renamed to *match_parent* in API level 8 and higher

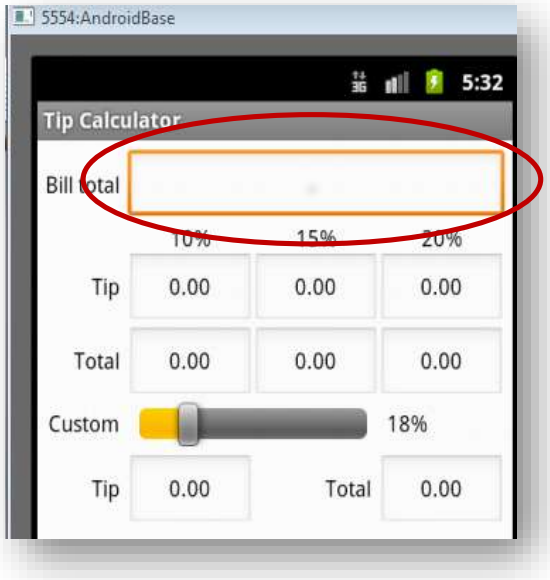
TextView



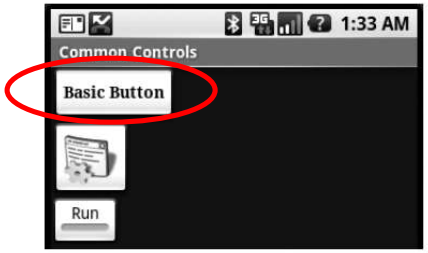
- TextView like a smarter Java Jlabel
- TextViews are used to display static texts
- Attributes:
  - android:text setText(CharSequence,TextView.BufferType) text to display
  - android:textColor setTextColor(int) text color
  - android:textSize setTextSize(int,float) size of the text
  - android:textStyle setTypeface(Typeface) style (bold, italic, bold, italic) for the text
  - android:autoText setKeyListener(KeyListener) if set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors
  - android:capitalize setKeyListener(KeyListener) if set, specifies that this TextView has a textual input method and should automatically capitalize what the user types

EditText

- *EditText* like Java *JTextField* with additional capabilities
- An EditText allows the user to type text into your application
- It can be either single line or multiple – lines
- Able to validate syntax for phone numbers, email, password, ...etc
- Attribute:
  - getText() return the text from the EditText
  - selectAll() convenience for select All



Button



- *Button* like Java *JButton*
- A Button consists of text or an icon (or both text and an icon) that communicates what action occurs when the users touches it
- Button can be pressed, or clicked by the user to perform an action

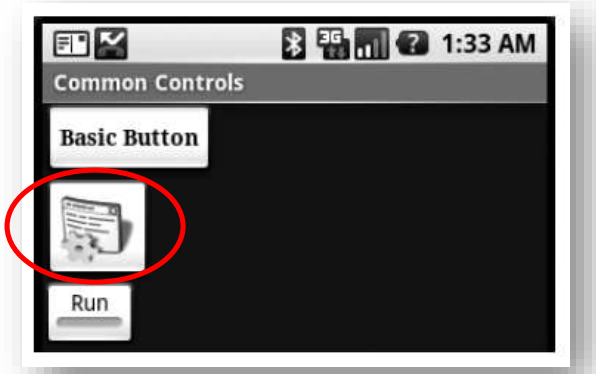
```
<Button
   android:layout_height="wrap_content"
   android:layout_width="wrap_content"
   android:text="@string/self_destruct“
   android:id="@+id/bADD"
 />
```

Event Handler for Button:

```
bnm= (Button)findViewById(R.id.bADD(in XML));
bnm.setOnClickListener(new View.OnClickListener(){
@Override
    public void onClick(View v) {
          counter ++;
          tvDisplay.setText(""+counter);
    }
});
```

ImageButton

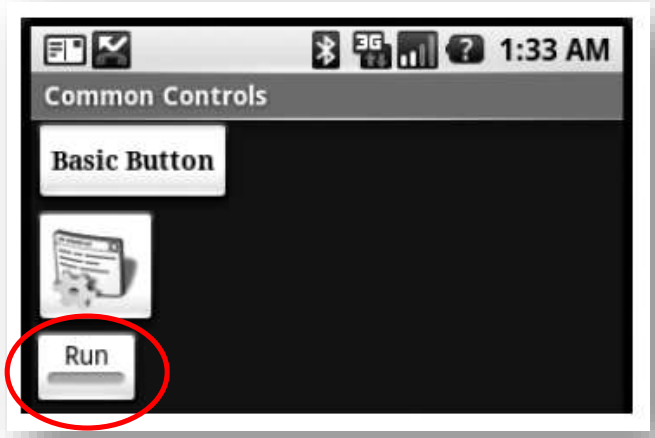
- Display a button with an image (instead of text) that can be pressed or clicked by the user
- By default, an ImageButton looks like a regular Button, with the standard button background that changes color during different Button states
- The Image on the surface of the Button is defined either by the ***android:src*** attribute in the **XML** element or by the ***setImageResource(int)*** method



ToggleButton

- A *ToggleButton* allows the user to change a setting between two states
- Attribute:
  - android:disabledAlpha The alpha to apply to the indicator when disabled
  - android:textOff The text for the button when it is not checked
  - android:textOn The text for the button when it is checked

```
<ToggleButton
    android:id="@+id/cctglBtn“
    android:layout_width="wrap_content“
    android:layout_height="wrap_content“
    android:textOn="Stop“
    android:textOff="Run“
    android:text="Toggle Button“
 />
```




Q & A