Designing the User Interface

Unit-3

Focus On

1. Android Layout Types
2. Layout Attributes
3. Android Widgets and its attributes
4. Event Handling
5. Working with String, String array and color
6. Working with resources, and drawable
7. Adding icon to the project

Android Layout Types

- ✔ Layout is defined as the structure for a user interface in our app, such as in an activity.
- ✔ In other words, Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an application or activity screen
- ✔ All the elements of the layout are built using a hierarchy of View and ViewGroup objects

View and ViewGroup

- ✔ A View is defined as the user interface which is used to create interactive UI components such as TextView, EditText, ImageView, Button, RadioButton, etc.,and is responsible for event handling and drawing
- ✔ Also called Widgets

android

ViewGroup

- ✔ A ViewGroup is a invisible container that defines the layout structure for View and other ViewGroup objects.
- ✔ Also called layouts such as LinearLayout, RelativeLayout, ConstraintLayout, etc

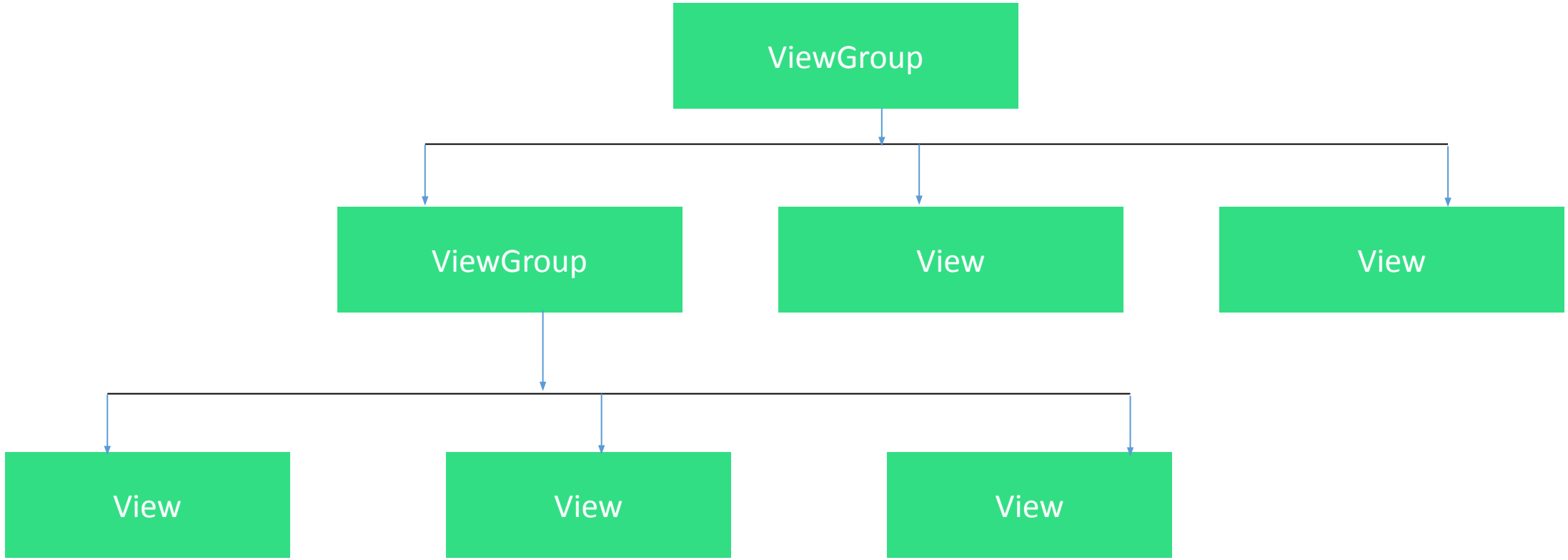

Figure: Illustration of a View hierarchy, which defines a UI layout

android

Declaration of Layout

✔ **We can declare layout in two ways:**

1. **Declare UI elements in XML**
2. **Instantiate layout elements at runtime**

Types of Layout

- ✔ LinearLayout
- ✔ RelativeLayout
- ✔ TableLayout
- ✔ AbsoluteLayout
- ✔ ConstraintLayout

LinearLayout

- ✔ **LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally**
- ✔ **Properties of LinearLayout**
  1. **android:orientation : should be a vertical or horizontal orientation**
  2. **android:weight: defines the maximum weight sum**
  3. **android:gravity: specifies how an object should position its content both the X axis and Y axis, within its bounds**
  4. **android:width: match_parent or wrap_content or a numerical value in dp**
  5. **Android:height: match_parent or wrap_content or a numerical value in dp**

LinearLayout example:

Resource file name: example_activity.xml

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/app_hello"/>

</LinearLayout>
```

RelativeLayout

- ✔ RelativeLayout is a view group that displays child views in relative positions.
- ✔ Each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to both left or center)
- ✔ It is a powerful utility for designing a user interface because it can eliminate nested view groups and keep layout hierarchy flat, which improves performance

✔ Properties of RelativeLayout

1. android:width: match_parent or wrap_content or a numerical value in dp
2. android:height: match_parent or wrap_content or a numerical value in dp
3. android:layout_above: positions the bottom edge of this view above the given anchor view ID
4. android:layout_below: positions the top edge of this view below the given anchor view ID
5. android:layout_toEndOf: positions the start edge of this view to the end of the given anchor view ID
6. android:layout_toLeft: positions the right edge of this view to the left of the given anchor view ID

7. **android:layout_toRightOf: positions the left edge of this view to the right of the given anchor view ID**
8. **android:layout_toStartOf: positions the end edge of this view to the start of the given anchor view ID**
9. **android:layout_centerHorizontal: if true, centers, this child horizontally within its parent**
10. **android:layout_centerVertical: if true, centers, this child vertically within its parent**
11. **android:layout_centerInParent: if true, center this child horizontally and vertically within its parent**

RelativeLayout example:

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

    <Button
        android:id="@+id/buttonOne"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/purple_200"
        android:text="@string/app_title1"
        android:layout_centerInParent="true"/>
    <Button
        android:id="@+id/buttonTwo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/purple_200"
        android:text="@string/app_title2"
        android:layout_margin="16dp"
        android:layout_centerHorizontal="true"/>
    <Button
        android:id="@+id/buttonThree"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:backgroundTint="@color/purple_200"
        android:text="@string/app_title3"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:layout_centerVertical="true"/>

</RelativeLayout>
```

File name: example_activity.xml

TableLayout

- ✔ TableLayout is a layout that arranges its children into rows and columns.
- ✔ It consists of a number of TableRow objects, each defining a row
- ✔ TableLayout containers don’t display border lines for their rows, columns, or cells
- ✔ Each row has zero or more cells; each cell can hold one View object
- ✔ A table can leave cells empty
- ✔ Cells can span columns, as they can in HTML

TableLayout

- ✔ The children of a TableLayout cannot specify the layout_width attribute. Width is always MATCH_PARENT
- ✔ The layout_height attribute can be defined by a child; default value is ViewGroup.LayoutParams.WRAP_CONTENT
- ✔ If the child is a TableRow, then the height is always ViewGroup.LayoutParams.WRAP_CONTENT
- ✔ Cells must be added to a row in increasing column order, both in code and XML
- ✔ Column numbers are zero-based, if we don’t specify a column number for a child cell, then it will auto increment to the next available column

✔ Properties of TableLayout:

- ✔ android:id: this is the ID which uniquely identifies the layout
- ✔ 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. Illegal and duplicate indices are ignored. We can stretch all columns by using the value “*” instead. Note that a column can be marked stretchable and shrinkable at the same time
- ✔ android:shrinkColumns: The zero-based index of the columns to shrink. The column indices must be separated by comma:1,2,5. . Illegal and duplicate indices are ignored. We can shrink all columns by using the value “*” instead. Note that a column can be marked stretchable and shrinkable at the same time
- ✔ android:stretchColumns: The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1,2,5.

TableLayout example

✔ **Will do in real time**

Absolute Layout

- ✔ A layout that lets specify exact locations (x/y coordinates) of its children.
- ✔ Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.
- ✔ AbsoluteLayout was deprecated in API level 3. So, we can use RelativeLayout or LinearLayout or any other layout for better performance
- ✔ Properties of Absolute Layout:
  - ✔ android:layout_x: this specifies the x-coordinate of the view
  - ✔ android:layout_y: this specifies the y-coordinate of the view

Absolute Layout example

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 1"
        android:textSize="16sp"
        android:padding="16dp"
        android:layout_x="100dp"
        android:layout_y="100dp"
        android:textStyle="bold"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 2"
        android:textSize="16sp"
        android:padding="16dp"
        android:layout_x="150dp"
        android:layout_y="120dp"
        android:textStyle="bold"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 3"
        android:textSize="16sp"
        android:padding="16dp"
        android:layout_x="200dp"
        android:layout_y="160dp"
        android:textStyle="bold"/>

</AbsoluteLayout>
```

File name: example_activity.xml

ConstraintLayout

- ✔ Constraint layout is an advanced version of a Relative layout.
- ✔ ConstraintLayout allows us to create large and complex layouts with a flat view hierarchy(no nested view groups).
- ✔ It is used to reduce the child view hierarchies and improve the performance
- ✔ It is used to define a layout by assigning constraints for every child view/widget relative to other views present.
- ✔ A constraint layout is similar to a RelativeLayout, but with more power
- ✔ The aim of ConstraintLayout is to improve the performance of the applications by removing the nested views with a flat and flexible design

- ✔ **Another aim of the ConstraintLayout is to help reduce the number of nested views, which will improve the performance of our layout files**
- ✔ **The layout class also makes it easier for us to define layouts than when using a RelativeLayout as we can now anchor any side of a view with any side of another, rather than having to place a whole view to any side of another**
- ✔ **Dependency: build.gradle**

  implementation ‘com.android.support.constraint:constraint-layout:1.1.3’

S/NO	RelativeLayout	ConstraintLayout
1	android:layout_centerInParent="true"	app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintLeft_toLeftOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintRight_toRightOf="parent" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintTop_toTopOf="parent"
2	android:layout_centerHorizontal="true"	app:layout_constraintLeft_toLeftOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintRight_toRightOf="parent" app:layout_constraintEnd_toEndOf="parent"
3	android:layout_centerVertical="true"	app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintTop_toTopOf="parent"
4	android:layout_alignParentLeft="true"	app:layout_constraintLeft_toLeftOf="parent"
5	android:layout_alignParentStart="true"	app:layout_constraintStart_toStartOf="parent"
6	android:layout_alignParentRight="true	app:layout_constraintRight_toRightOf="parent"
7	android:layout_alignParentEnd="true"	app:layout_constraintEnd_toEndOf="parent"
8	android:layout_alignParentTop="true"	app:layout_constraintTop_toTopOf="parent"
9	android:layout_alignParentBottom="tru	app:layout_constraintBottom_toBottomOf="parent"
10	android:layout_alignStart="@id/view"	app:layout_constraintStart_toStartOf="@id/view"
11	android:layout_alignLeft="@id/view"	app:layout_constraintLeft_toLeftOf="@id/view"
12	android:layout_alignEnd="@id/view"	app:layout_constraintEnd_toEndOf="@id/view"
13	android:layout_alignRight="@id/view"	app:layout_constraintRight_toRightOf="@id/view"
14	android:layout_alignTop="@id/view"	app:layout_constraintTop_toTopOf="@id/view"
15	android:layout_alignBaseline="@id/vie	app:layout_constraintBaseline_toBaselineOf="@id/view"
16	android:layout_alignBottom="@id/view	app:layout_constraintBottom_toBottomOf="@id/view"
17	android:layout_toStartOf="@id/view"	app:layout_constraintEnd_toStartOf="@id/view"
18	android:layout_toLeftOf="@id/view"	app:layout_constraintRight_toLeftOf="@id/view"
19	android:layout_toEndOf="@id/view"	app:layout_constraintStart_toEndOf="@id/view"
20	android:layout_toRightOf="@id/view"	app:layout_constraintLeft_toRightOf="@id/view"
21	android:layout_above="@id/view"	app:layout_constraintBottom_toTopOf="@id/view"
22	android:layout_below="@id/view"	app:layout_constraintTop_toBottomOf="@id/view"

Fig: Properties of ConstraintLayout equivalent to Relative Layout

Assignments

1. What do you mean by Layout? List and explain different types of layout used in android with their major attributes.
2. Differentiate between RelativeLayout and ConstraintLayout with example.
3. Compare AbsoluteLayout with other types of layout.
4. Differentiate between RelativeLayout and LinearLayout with example.
5. What is android platform? Explain view and view group with examples.

Assignment due Date: Jan, 20, 2022

Android Widgets/Views

- ✔ TextView
- ✔ EditText
- ✔ Button
- ✔ ImageView
- ✔ CheckBox
- ✔ RadioButton
- ✔ Spinner

TextView

- ✔ It is a user interface element that displays text to the user
- ✔ Properties of TextView:
  - ✔ android:width:match_parent, wrap_content
  - ✔ android:height:match_parent, wrap_content
  - ✔ android:id: uniques id
  - ✔ android:text: to set text
  - ✔ android:textSize:
  - ✔ android:textStyle:
  - ✔ android:background:
  - ✔ android:padding: top,right,bottom.left
  - ✔ android:margin: top,right,bottom,left

EditText

- ✔ A user interface element for entering and modifying text.
- ✔ inputType attribute must specify
- ✔ Choosing the input type configures the keyboard type that is shown, acceptable characters and appearance of the edit text
- ✔ Properties of EditText:
  - ✔ android:id: uniquely identify
  - ✔ android:gravity: position(center, start, end)
  - ✔ android:width:
  - ✔ android:height:
  - ✔ android:inputType:text,password,phoneNumber,email

Button

- ✔ It is a user interface element the user can tap or click to perform an action
- ✔ Attributes Inherited from TextView – Almost all attributes of TextView can be used for Button.
- ✔ Event: onClick(View view)

CheckBox

- ✔ A checkbox is a specific type of two-states button that can be either checked or unchecked.
- ✔ Attributes Inherited from TextView – Almost all attributes of TextView can be used for checkbox.
- ✔ Properties of CheckBox:

RadioButton

- ✔ A radio button is a two-states button that can be either checked or unchecked
- ✔ Radio buttons are normally used together in a RadioGroup
- ✔ When many radio buttons live inside a radio group, checking one radio button unchecks all the others.
- ✔ Attributes Inherited from TextView – Almost all attributes of TextView can be used for radio button.
- ✔ Properties of RadioButton:

Spinner

- ✔ It is a view that displays one child at a time and lets the user pick among them
- ✔ The items in the Spinner come from the String array or Adapter associated with this view.
- ✔ Properties of Spinner:
  - ✔ android:width:
  - ✔ android:height:
  - ✔ android:id:
  - ✔ android:entries:

Event Handling

- ✔ Events are useful way to collect about a user’s interaction with interactive components of applications
- ✔ Example: button presses or screen touch etc.
- ✔ Android framework maintains an event queue as first-in, first-out (FIFO)basis.

Three concepts to Android Event Management

1. **Event Listeners**
   - An event listener is an interface in the View Class that contains the single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI
2. **Event Listeners Registration**
   1. Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event
4. **Event Handlers**
   1. When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
1. onClick()	✔ OnClickListener(): This is called when the user either clicks or touches or focuses upon any widget like button,text,image, etc. onClick() event handlers is used to handle such event.

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
2. onLongClick()	✔ OnLongClickListener(): This is called when the user either clicks or touches or focuses upon any widget like button,text,image, etc. for one or more seconds. onLongClick() event is used to handle such event.

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
3. onFocusChange()	✔ OnFocusChangeListener(): This is called when the widget looses its focus i.e. user goes away from the view item. onFocusChange() event handler is used to handle such event.

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
4. onKey()	✔ OnKeyListener(): This is called when the user is focused on the item and presses or releases a hardware key on the device. onKey() event is used to handle such event.

android

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
5. onTouch()	✔ OnTouchListener(): This is called when the user presses the key, releases the key or any movement gesture on the screen. onTouch() event is used to handle such event

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
6. onMenuItemClick()	✔ OnMenuItemClickListener(): This is called when the user selects a menu item. onMenuItemClick() event handler is used to handle such event

Event Listeners & Event Handlers

Event Handler	Event Listeners & Description
6. onCreateContextMenu()	✔ OnCreateContextMenuItemListener() : This is called when the context menu is being build(as the result of a sustained ‘long click’)

Assignment

- Design simple UI to calculate simple interest. (P*T*R)
  - 3 EditText
  - Result: Text
  - Button: Calculate

Thank You!!