CS 473 – MDP
Mobile Device Programming

Maharishi International University

CS 473 – MDP
Mobile Device Programming

MS.CS Program

Department of Computer Science

Renuka Mohanraj, Ph.D.,

Maharishi International University

CS 473 – MDP
Mobile Device Programming

Lesson 4

Views, Layouts, Resources and Activity Life Cycle

Maharishi International
University

Wholeness

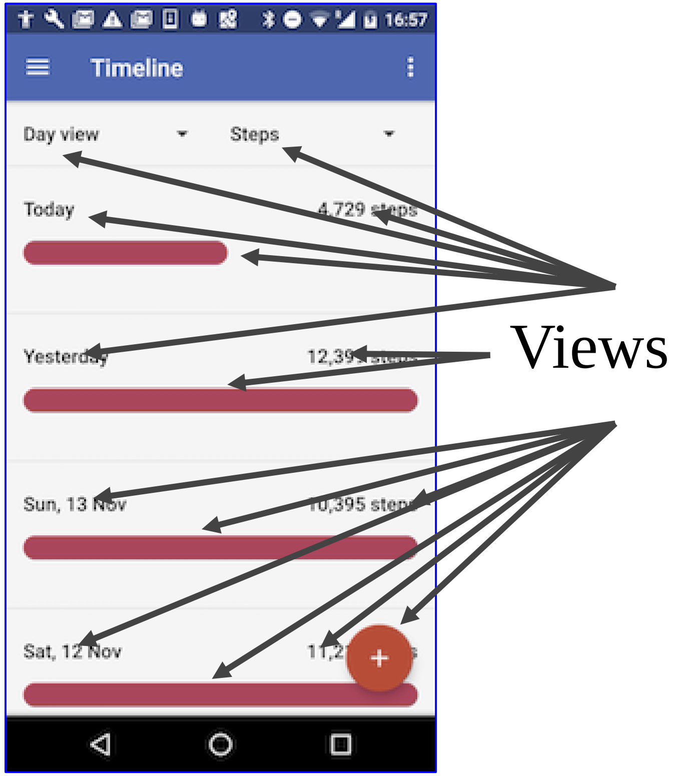
- In this lecture we examine additional fundamental building blocks of Android apps including the layouts, input/output controls, and buttons. We also explore resources and activity life cycle. App can build effectively with this fundamental knowledge. *The most fundamental knowledge is the most important knowledge, since everything else is built upon it. The reason TM can provide such a wide range of benefits to life in general is because it works at such a truly fundamental level.*

Contents

- Views, view groups, and view hierarchy
- Layouts in XML and Kotlin code
- Resources
- Activity Life Cycle
- Save State Information call backs
- Hands on Examples – Basic UI, Simple Calculator, Working with ScrollView, Lifecycle Activity and Save state

Everything you see is a view

If you look at your mobile device, every user interface element that you see is a **View**.



What is a view

Views are Android's basic user interface building blocks.

- display text (TextView class), edit text (EditText class)
- buttons (Button class), menus, other controls
- scrollable (ScrollView, RecyclerView)
- show images (ImageView)
- subclass of View class

Views have properties

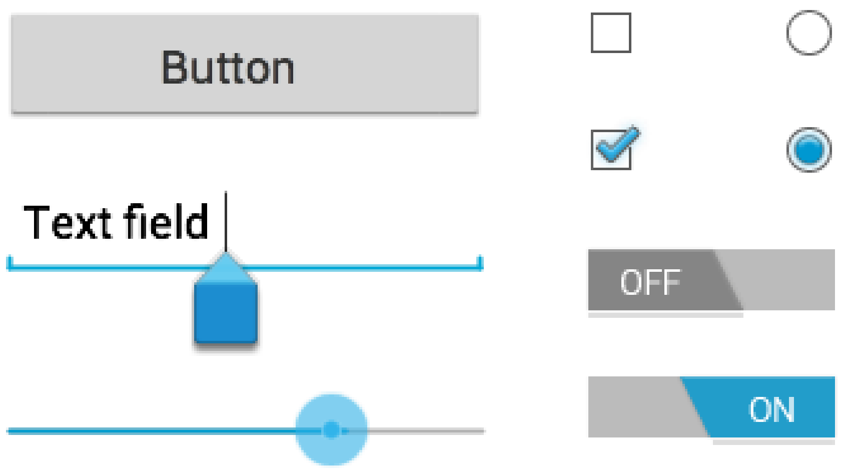
- Have properties (e.g., color, dimensions, positioning)
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Have relationships to other views

Examples of views

Button

EditText

SeekBar

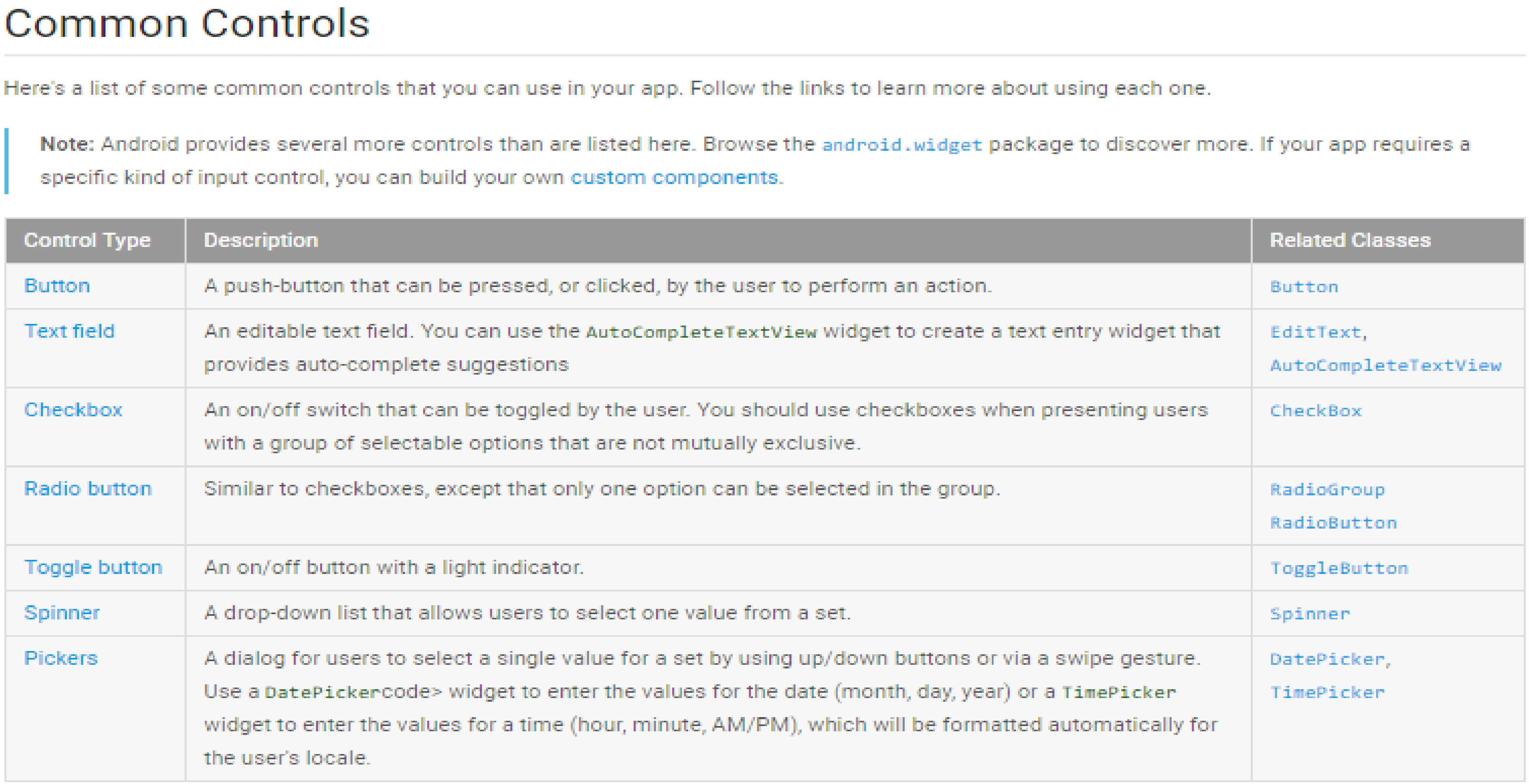


CheckBox

RadioButton

Toggle

Refer for more Info : https://developer.android.com/guide/topics/ui/index.html



Common Controls

Here's a list of some common controls that you can use in your app. Follow the links to learn more about using each one.

> **Note:** Android provides several more controls than are listed here. Browse the `android.widget` package to discover more. If your app requires a specific kind of input control, you can build your own custom components.

Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	`Button`
Text field	An editable text field. You can use the `AutoCompleteTextView` widget to create a text entry widget that provides auto-complete suggestions	`EditText`, `AutoCompleteTextView`
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	`CheckBox`
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	`RadioGroup` `RadioButton`
Toggle button	An on/off button with a light indicator.	`ToggleButton`
Spinner	A drop-down list that allows users to select one value from a set.	`Spinner`
Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a `DatePicker`code> widget to enter the values for the date (month, day, year) or a `TimePicker` widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	`DatePicker`, `TimePicker`

Creating and laying out views

- Graphically using XML Design Editor
- XML Code Editor
- Programmatically

Views defined in XML

```
<TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/myBackgroundColor"
        android:text="@string/count_initial_value"
        android:textColor="@color/colorPrimary"
        android:textSize="@dimen/count_text_size"
        android:textStyle="bold"
/>
```

View properties in XML

android:<property_name>="<property_value>"

Example: `android:layout_width="match_parent"`

android:<property_name>="@<resource_type>/resource_id"

Example: `android:text="@string/button_label_next"`

android:<property_name>="@+id/view_id"

Example: `android:id="@+id/show_count"`

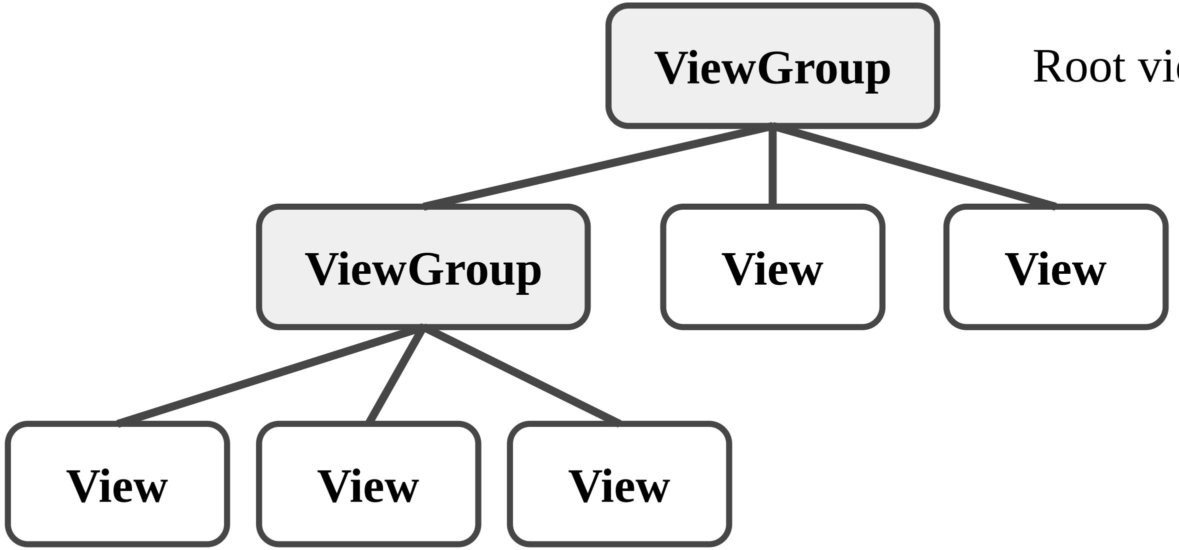
ViewGroup views

A ViewGroup (parent) is a type of view that can contain other views (children)

ViewGroup is the base class for layouts and view containers

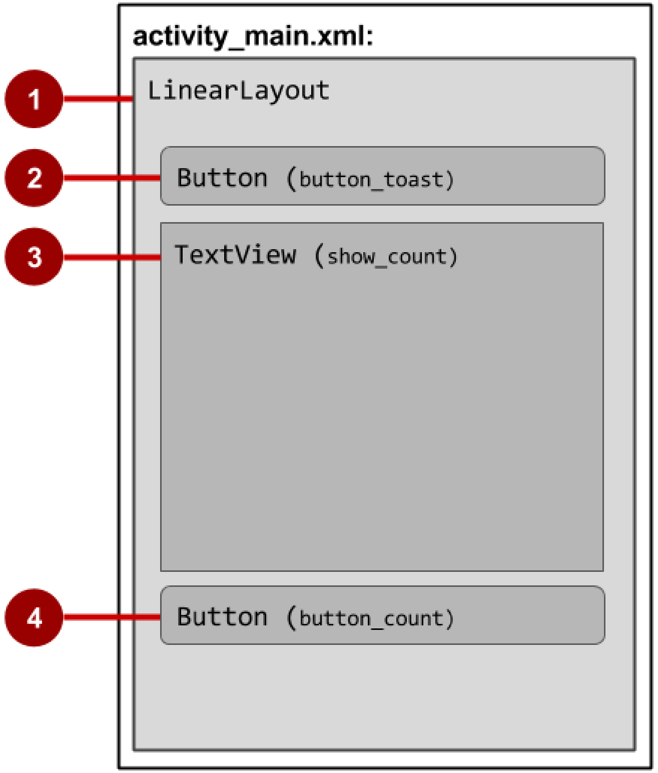
- ScrollView—scrollable view that contains one child view
- LinearLayout—arrange views in horizontal/vertical row
- RecyclerView—scrollable "list" of views or view groups

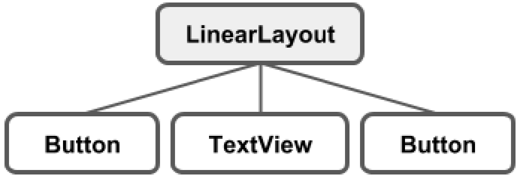
Hierarchy of view groups and views

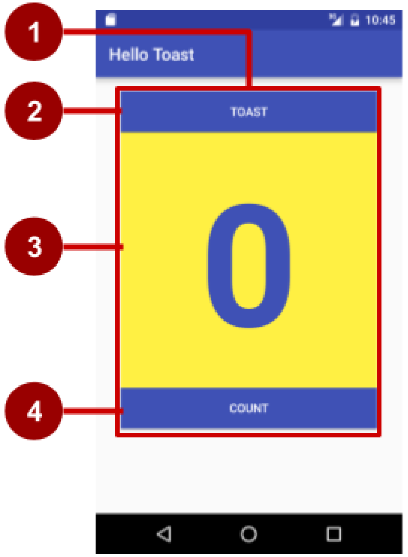


Root view is always a view group

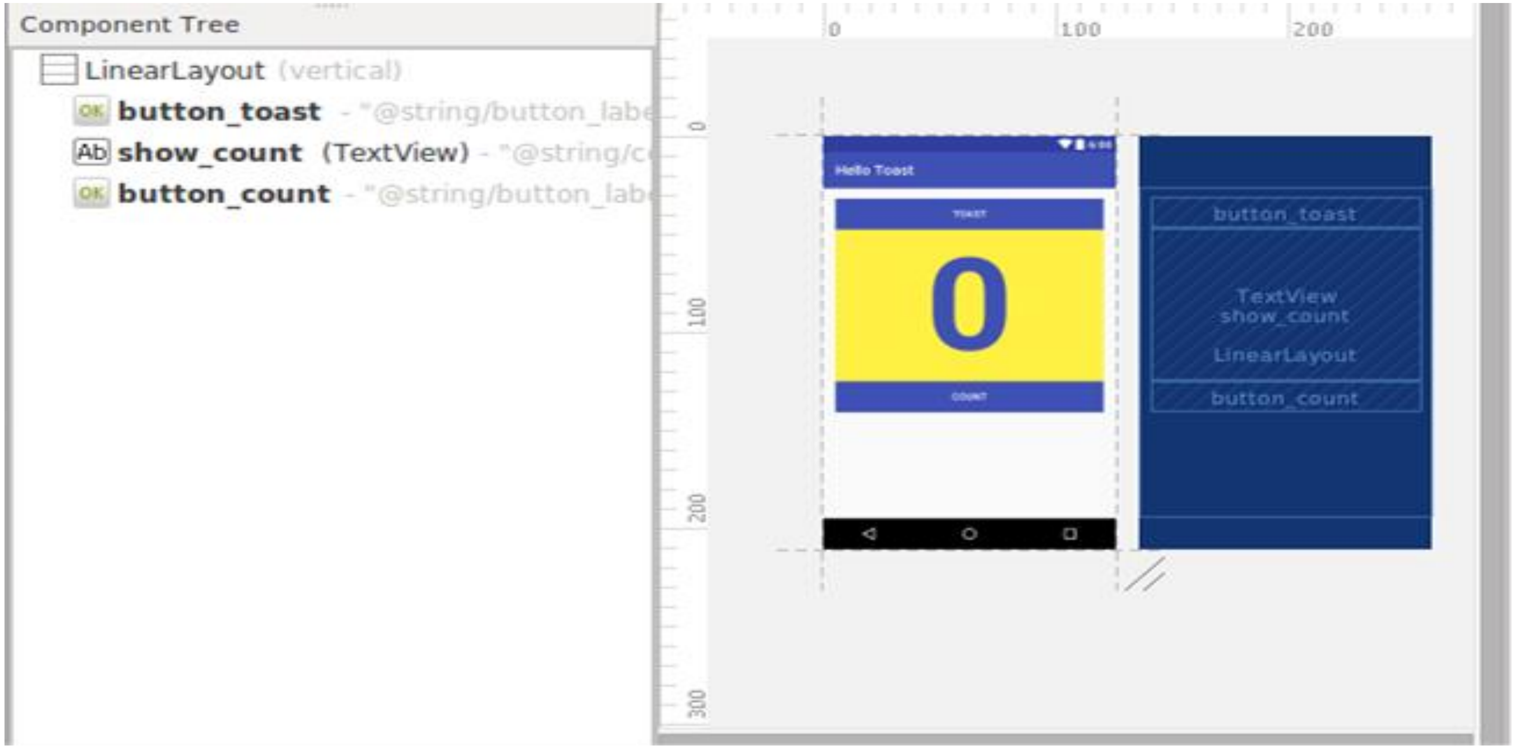
View hierarchy and screen layout







View hierarchy in the component tree



Component Tree

LinearLayout (vertical)

button_toast - "@string/button_labe

show_count (TextView) - "@string/c

button_count - "@string/button_labe

Best practices for view hierarchies

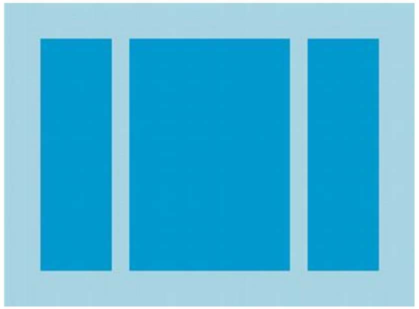
- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

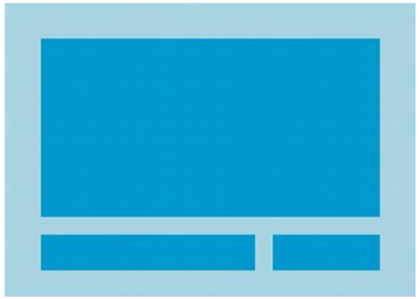
Layout Views

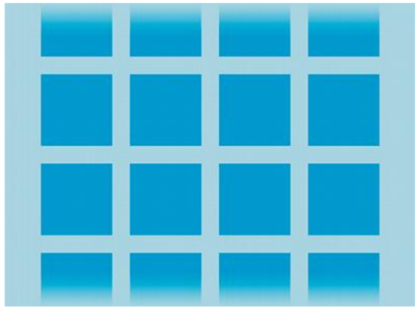
Layouts

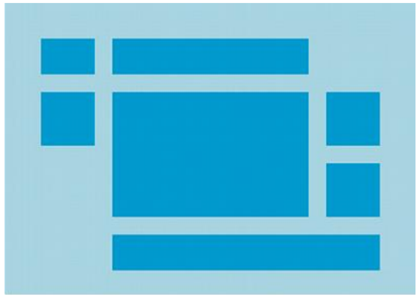
- are specific types of view groups
- are subclasses of ViewGroup
- contain child views
- can be in a row, column, grid, table, absolute

Common Layout Classes


LinearLayout


RelativeLayout


GridLayout


TableLayout

Common Layout Classes

- **LinearLayout** - horizontal or vertical row
- **RelativeLayout** - child views relative to each other
- **TableLayout** - rows and columns
- **ConstraintLayout** - connect views with constraints
- **FrameLayout** - shows one child of a stack of children
- **GridView** - 2D scrollable grid
- **In our course we are going to discuss the first three layouts.**

Class Hierarchy vs. Layout Hierarchy

- View class-hierarchy is standard object-oriented class inheritance
  - For example, Button is-a TextView is-a View is-a Object
  - Superclass-subclass relationship

Kotlin class-hierarchy

```
open class Button : TextView
```

```
kotlin.Any
  ↳ android.view.View
      ↳ android.widget.TextView
          ↳ android.widget.Button
```

Java class-hierarchy

```
public class Button
extends TextView
```

```
java.lang.Object
  ↳ android.view.View
      ↳ android.widget.TextView
          ↳ android.widget.Button
```

- Layout hierarchy is how Views are visually arranged
  - For example, LinearLayout can contain Buttons arranged in a row
  - Parent-child relationship

Layout created in XML

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <EditText
       ... />
    <Button
       ... />
</LinearLayout
```

Layout created in Kotlin Activity code

```
val linearL = LinearLayout(this)
linearL.setOrientation(LinearLayout.VERTICAL)
val myText = TextView(this)
myText.setText("Display this text!")
linearL.addView(myText)
setContentView(linearL)
```

Main Point 1

Views are the basic building block for user interface components. Layouts are used to organize Views. ***Science of Consciousness: :*** *In a similar way, creation itself is structured in layers; the activity at each layer has its own unique set of governing laws; laws that pertain to one level or layer may longer be applicable at another level.*

Linear Layout Example

```
<LinearLayout xmlns:android= "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/TextView01"
        android:text="Text View"
          android:layout_height="wrap_content"
          android:layout_width="match_parent" />
    <Button
          android:id="@+id/Button01"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
        android:text="Press Me" />
</LinearLayout>
```

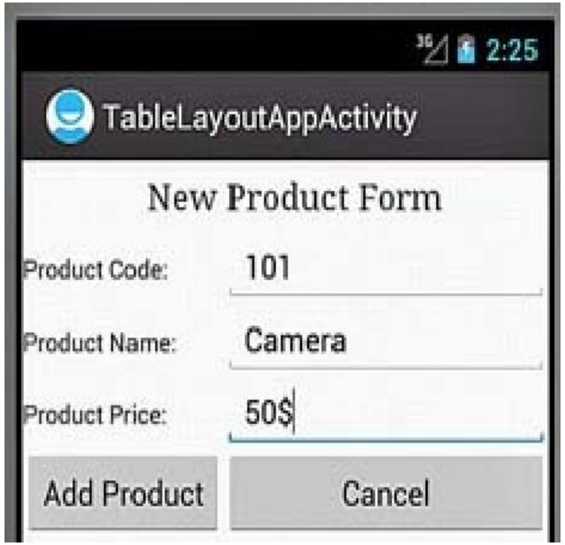
Here’s an XML layout resource example of a `LinearLayout` set to the size of the screen, containing one `TextView` that is set to its height and the width of the `LinearLayout`.

`android:id="@+id/Button01"`

If you want to access the component from xml to Kotlin code, you must configure the id for the component using the above line of code

Hands on Example 1 - Table Layout

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
    <TableRow android:padding="5dip">
        <TextView
            android:layout_height="wrap_content"
            android:text="New Product Form"
            android:typeface="serif"
            android:layout_span="2"
            android:gravity="center_horizontal"
            android:textSize="20sp" />
    </TableRow>
```



Hands on Example 1 - Table Layout

```
<TableRow>
      <TextView
         android:layout_height="wrap_content"
         android:text="Product Code:"/>
      <EditText
         android:id="@+id/prod_code"
         android:layout_height="wrap_content"/>
   </TableRow>
   <TableRow>
      <TextView
         android:layout_height="wrap_content"
         android:text="Product Name:" />
      <EditText
         android:id="@+id/prod_name"
         android:layout_height="wrap_content"/>
   </TableRow>
<TableRow>
      <TextView
         android:layout_height="wrap_content"
         android:text="Product Price:" />
      <EditText
         android:id="@+id/prod_price"
         android:layout_height="wrap_content" />
   </TableRow>
   <TableRow>
      <Button
         android:id="@+id/add_button"
         android:text="Add Product"
         android:layout_height="wrap_content" />
      <Button
         android:id="@+id/cancel_button"
         android:text="Cancel"
         android:layout_height="wrap_content" />
   </TableRow>
</TableLayout>
```

MainActivity.kt

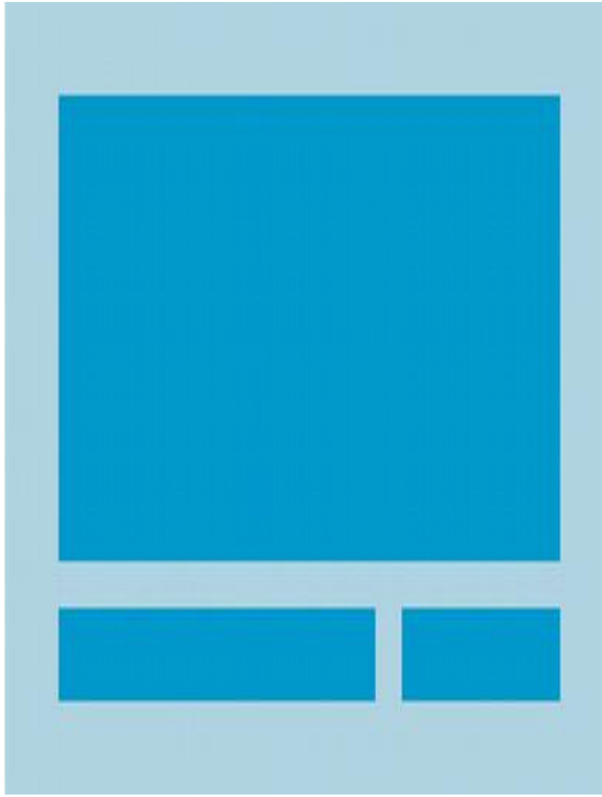
```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.Toast

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val badd = findViewById<Button>(R.id.add_button)
        val bcan = findViewById<Button>(R.id.cancel_button)
        // Anonymous Implementation of Button click event listener to give a toast Message
        badd.setOnClickListener {
            Toast.makeText(this, "Add Product Button Pressed", Toast.LENGTH_LONG).show()
        }
        bcan.setOnClickListener {
            Toast.makeText(this, "Cancel Button Pressed", Toast.LENGTH_LONG).show()
        }
    }
}
```

Refer Demo Code: Lesson4\TableLayoutApp

Relative Layout

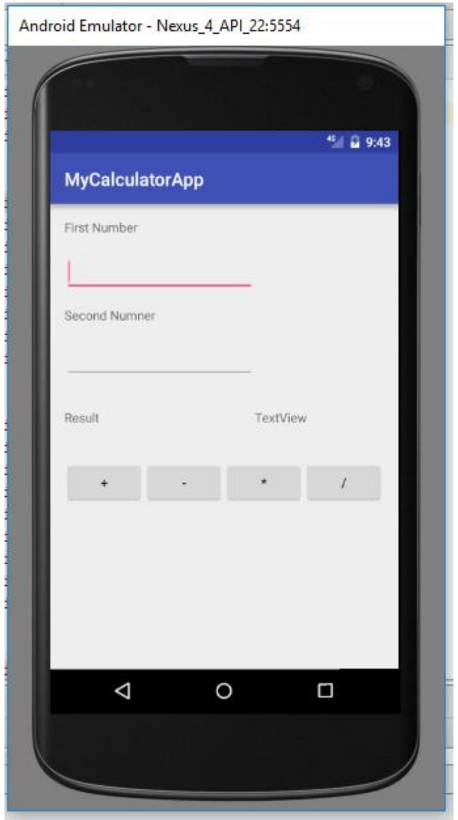
- The RelativeLayout view enables you to specify where the child View controls are in relation to each other.
  - For instance, you can set a child View to be positioned “above” or “below” or “to the left of” or “to the right of” another View.
  - You can also align child View controls relative to one another or the parent layout edges.
- Combining RelativeLayout attributes can simplify the creation of interesting user interfaces without resorting to multiple layout groups to achieve a desired effect.



Hands on Example 2 – Simple Calculator using Relative and Linear Layouts

The requirement of this problem is to design a screen as per the screen shot, using nested layouts and performing click action on the operator buttons to display the Result. This screen uses 4 TextView, 2 EditText and 4 Buttons. The toper layout is Relative Layout and all buttons are combined using Linear Layout.

Refer Demo Code: Lesson4\CalculatorApp



XML Design(activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">
  <TextView
    android:text="First Number"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tv1"
    android:layout_alignLeft="@+id/et1"
    android:layout_alignStart="@+id/et1"
    android:layout_alignParentTop="true" />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:layout_marginTop="17dp"
    android:id="@+id/et1"
    android:layout_below="@+id/tv1"
    android:layout_alignLeft="@+id/et2"
    android:layout_alignStart="@+id/et2" />
```

```
<TextView
  android:text="Second Numner"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="15dp"
  android:id="@+id/tv2"
  android:layout_below="@+id/et1"
  android:layout_alignLeft="@+id/et1"
  android:layout_alignStart="@+id/et1" />

<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:inputType="number"
  android:ems="10"
  android:layout_marginTop="16dp"
  android:id="@+id/et2"
  android:layout_below="@+id/tv2"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true" />

<TextView
  android:text="Result"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="32dp"
  android:id="@+id/tv3"
  android:layout_below="@+id/et2"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true" />

<TextView
  android:text="TextView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/tv4"
  android:layout_alignBaseline="@+id/tv3"
  android:layout_alignBottom="@+id/tv3"
  android:layout_toRightOf="@+id/et2"
  android:layout_toEndOf="@+id/et2" />
```

```
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="38dp"
        android:layout_below="@+id/tv3"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true">

        <Button
            android:text="+“
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/add“
            android:onClick="click"/>

        <Button
            android:text="-"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/sub"
            android:onClick="click"/>

<Button
  android:text="*"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/mul“
  android:onClick="click"/>

<Button
    android:text="/"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content“
    android:id="@+id/div“
    android:onClick="click"/>
</LinearLayout>
</RelativeLayout>
```

Main Activity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View;
// To get the ids directly from XML without using findViewById() need to do below import
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
```

Main Activity.kt

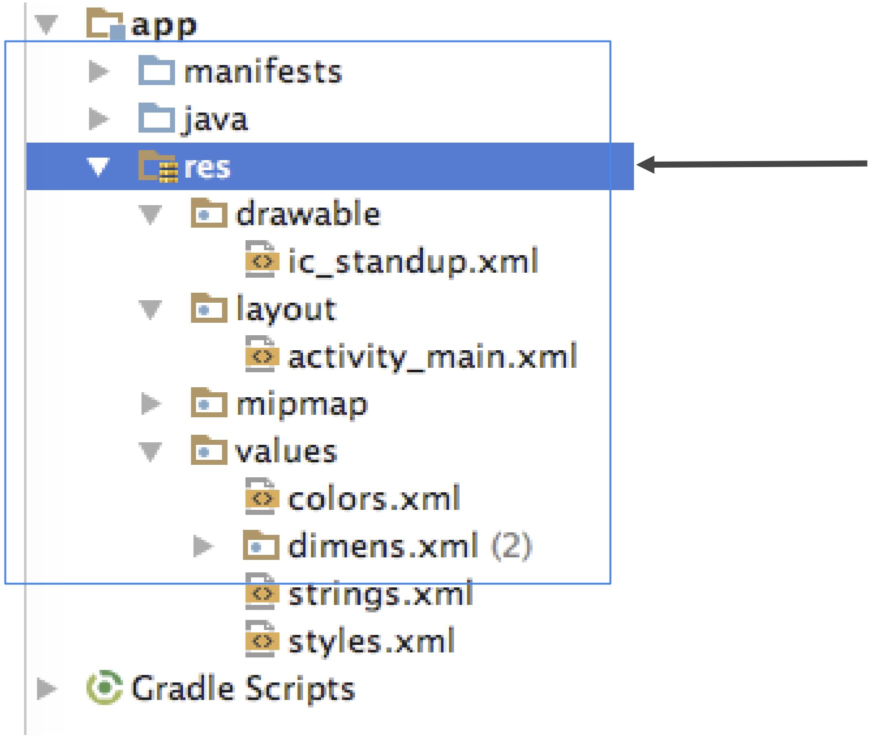
```
// click event implementation from xml  android:onClick="click"
 fun click(view:View) {
    val num1 = et1.text.toString() // to retrieve a text like getter – 1st Input
    val num2 = et1.text.toString() // 2nd Input
    when (view.getId()) { // Read the clicked component
        R.id.add -> {
            val addition = Integer.parseInt(num1) + Integer.parseInt(num2)
            tv4.text = addition.toString() // Setting value to the text view like setter
        }
        R.id.sub -> {
            val minus = Integer.parseInt(num1) - Integer.parseInt(num2)
            tv4.text = minus.toString()
        }
        R.id.mul -> {
            val mult = Integer.parseInt(num1) * Integer.parseInt(num2)
            tv4. text = mult.toString()
        }
```

```
R.id.div -> try{
val dvd = Integer.parseInt(num1) /
                Integer.parseInt(num2)
  tv4.text = dvd.toString()
         }
 catch (e:Exception) {
   tv4.text = "Division be Zero"
         }
      }
   }
}
```

Resources

- Separate static data from code in your layouts.
- Strings, dimensions, images, menus, colors, styles
- Useful for localization

Where are the resources in your project?



```
▼ app
    ▶ manifests
    ▶ java
    ▼ res
        ▼ drawable
              ic_standup.xml
        ▼ layout
              activity_main.xml
        ▶ mipmap
        ▼ values
              colors.xml
            ▶ dimens.xml (2)
              strings.xml
              styles.xml
▶ Gradle Scripts
```

resources and resource files stored in **res** folder

Hands on Example 3
TextView, ScrollView & Resourses

- TextView is a view for displaying single and multi-line text
- EditText is a subclass of TextView with editable text
- Controlled with layout attributes
- Set text statically from a string resource in XML, or dynamically from Kotlin code and any source
- The user may need to scroll.
  - News stories, articles, ...

- To allow users to scroll a TextView, embed it in a ScrollView.
- Other Views can be embedded in a ScrollView.
  - LinearLayout, TextView, Button, ...
- ScrollView is a subclass of FrameLayout
  - Can only hold one view
  - Do not nest multiple scrolling views
  - Use HorizontalScrollView for horizontal scrolling
  - Use a RecyclerView for lists(Will discuss later)

Common TextView attributes

android:text—text to display

android:textColor—color of text

android:textAppearance—predefined style or theme

android:textSize—text size in `sp`

android:textStyle—`normal`, `bold`, `italic`, or `bold|italic`

android:typeface—`normal`, `sans`, `serif`, or `monospace`

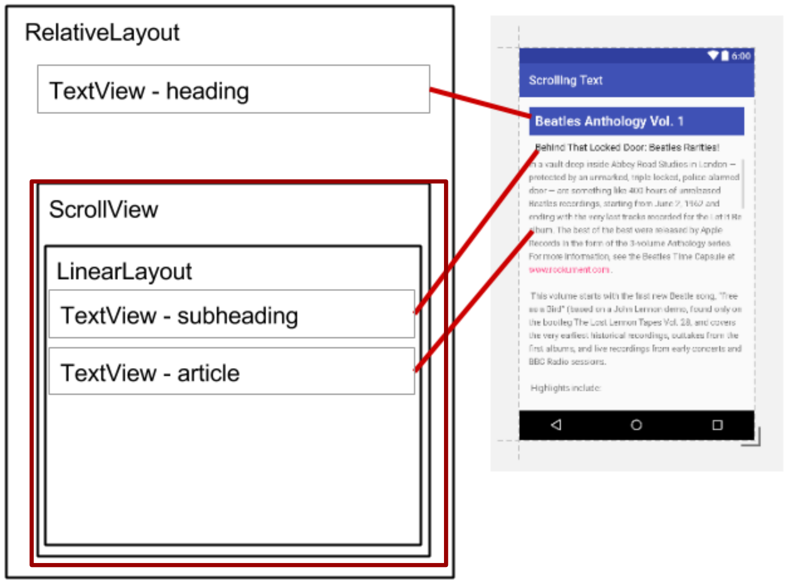
android:lineSpacingExtra—extra space between lines in `sp`

ScrollView layout with a view group

```
<ScrollView ...
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">

       <TextView
           android:id="@+id/article_subheading"
           .../>

       <TextView
           android:id="@+id/article" ... />
   </LinearLayout>
</ScrollView>
```



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London – protected by an unmarked, triple locked, police alarmed door – are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the Let It Be album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on the bootleg The Lost Lennon Tapes Vol. 28, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

Highlights include:

Main Point 2

Different layouts in Android helps to place and arrange the UI components and other layouts. These self-referral dynamics support a much broader range of possibilities in the design of UIs.

Activity Life Cycle

- The activity base class defines a series of events that govern the life cycle of an activity. The set of states an activity can be in during its lifetime, from when it is created until it is destroyed.

Activity states and app visibility

- Created (not visible yet)
- Started (visible)
- Resume (visible)
- Paused(partially invisible)
- Stopped (hidden)
- Destroyed (gone from memory)

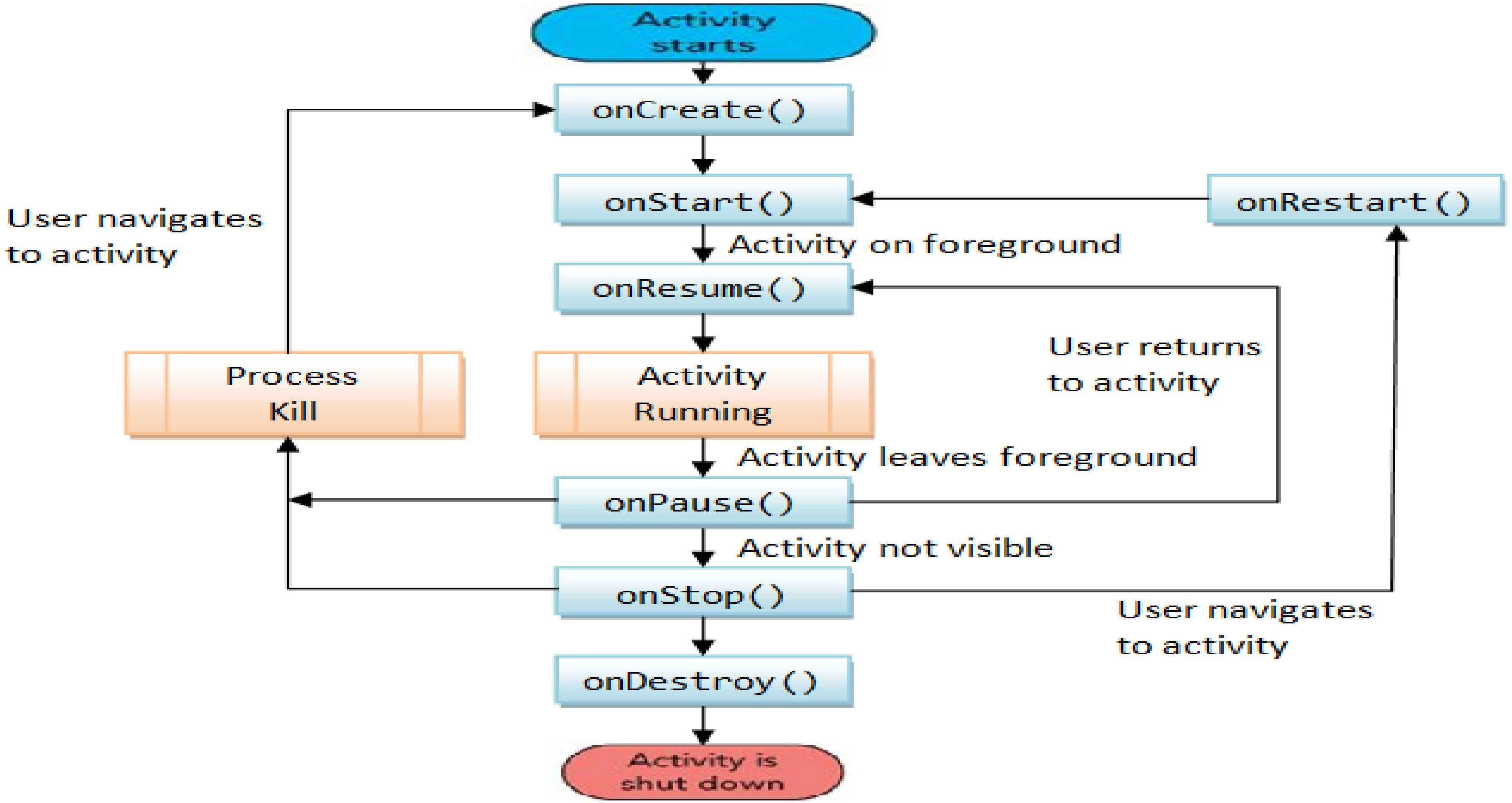
State changes are triggered by user action, configuration changes such as device rotation, or system action like changing language settings.

Callbacks and when they are called

Only onCreate() is required, Override the other callbacks to change default behavior

- **`onCreate(Bundle savedInstanceState)`**—member fields initialization
  - **`onStart()`**—when activity (screen) is becoming visible
  - **`onRestart()`**—called if activity was stopped (calls onStart())
    - **`onResume()`**—start to interact with user
    - **`onPause()`**—about to resume PREVIOUS activity
  - **`onStop()`**—no longer visible, but still exists and all state info preserved
- **`onDestroy()`**—final call before Android system destroys activity

Activity Life Cycle



States of Activity Life cycle

1. Active State: [Activity in the Foreground/Running]

- When an Activity is present at the top of the stack, it is the currently visible and focused Activity and all the user inputs are provided to it.
- In order to keep this Activity active, Android provides it with all the required resources and also terminates the previous activities if required.
- When another Activity becomes active, this Activity will be pushed to the paused state.

States of Activity Life cycle

2. Paused State:

- An Activity in this state is visible, but another Activity will have the focus and is present in the foreground.
- An Activity in the paused state is treated in the same way as it was treated when it was in the active state.
- The only difference is that it will not receive any user input.
- In serious cases, Android terminates a paused Activity to ensure availability of resources to the current Activity.
- An Activity is stopped when it becomes totally obscure.

States of Activity Life cycle

3. Stopped State: [Activity in the background]
   - In this state, the Activity is not visible. It is however, present in the memory with all the state information.
   - All such activities are now ready for termination when the system requires memory.
   - When an Activity is stopped, it’s data and UI information needs to be saved.
   - An Activity becomes inactive when it is closed or exited.
4. Inactive State: [Activity Doesn’t exist]
   - When the Activity is no longer in the memory, it is said to be in the inactive state.
   - An Activity goes to the inactive state when it is terminated.
   - All such activities need to be restarted before they are used again.

Hands on Example 4: Life Cycle Activity

Now we are going to Override all these methods to know the activity life cycle and always call up to superclass when implementing these methods. The we are displaying the Log message using Log.i(String TAG,String msg) by import android.util.Log;

```
class MainActivity : AppCompatActivity() {
    val MY_TAG = "lifecycle"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.i(MY_TAG,"Method in OnCreate");
}
```

- Refer: Lesson4_LogCat Filter Steps.pdf file to learn how to create logcat from Sakai Lectures/Lesson4 folder

MainActivity.kt

```
override fun onStart() {
      super.onStart()
      Log.i(MY_TAG, "Method in OnStart")
}
   override fun onResume() {
      super.onResume()
      Log.i(MY_TAG, "Method in OnResume")
}
   override fun onPause() {
      super.onPause()
      Log.i(MY_TAG, "Method in OnPause")
}
```

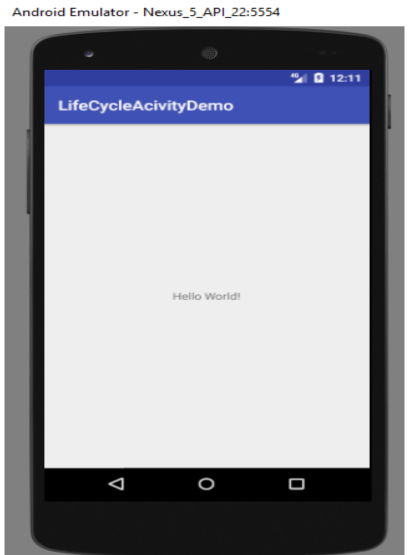
```
override fun onStop() {
      super.onStop()
      Log.i(MY_TAG, "Method in OnStop")
    }

   override fun onRestart() {
      super.onRestart()
      Log.i(MY_TAG, "Method in OnRestart")
    }
   override fun onDestroy() {
      super.onDestroy()
      Log.i(MY_TAG, "Method in OnDestroy")
   }
}
```

Activity Life Cycle – Screen Shots

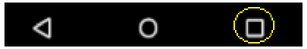
- Lifetime of an Activity is from onCreate() to onDestroy()
- Activity is Visible when onStart() to onStop()
- Activity is in Foreground onResume to onPause()

Screen 1 : After running the App, you will get the below screen and the Log message. The Activity is started by invoking 1. onCreate(), 2. onStart() and 3. onResume(). It is visible in the foreground.

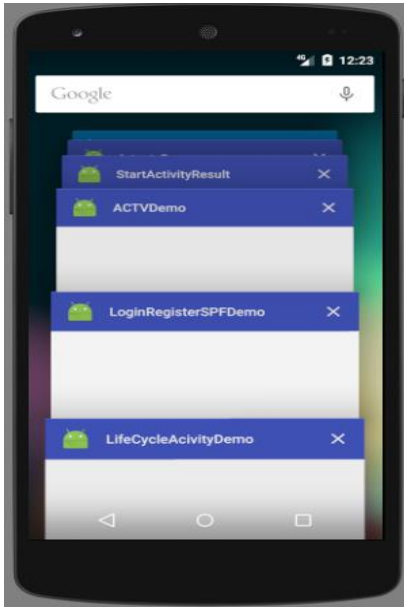


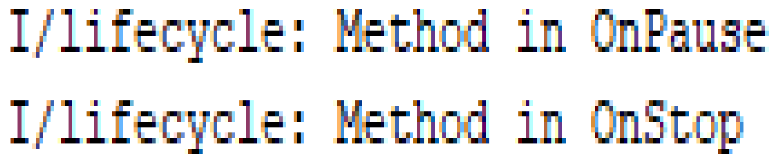
```
I/lifecycle: Method in OnCreate
I/lifecycle: Method in OnStart
I/lifecycle: Method in OnResume
```

Activity Life Cycle – Screen Shots

Screen 2 : Now press the highlighted  Pass button from your Emulator. The Activity is paused by invoking 1. OnPause() and 2. onStop().

Now it is not visible and not destroyed.

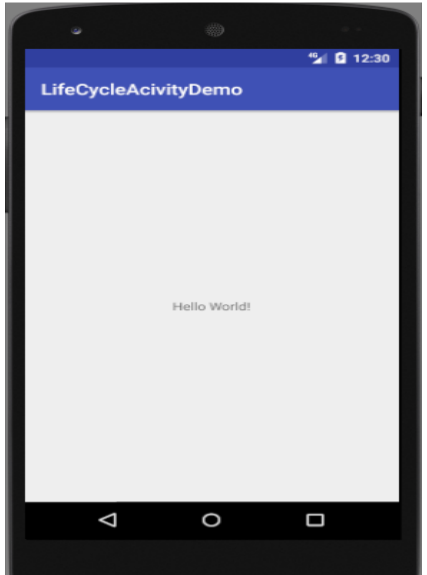




```
I/lifecycle: Method in OnPause
I/lifecycle: Method in OnStop
```

Activity Life Cycle – Screen Shots

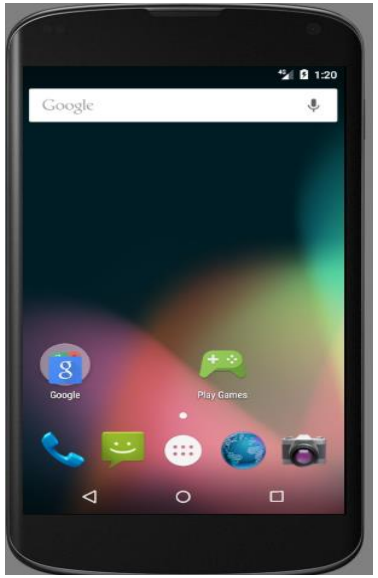
Screen 3 : Again start your application by clicking highlighted start button the from your Emulator and reload the same app. Now the Activity is restarted by invoking 1. OnRestart() 2. onStart() and 3. OnResume() itself. Now it is visible and not destroyed.



```
I/lifecycle: Method in OnRestart
I/lifecycle: Method in OnStart
I/lifecycle: Method in OnResume
```

Activity Life Cycle – Screen Shots

Screen 4 : If you press the highlighted back button from your Emulator your activity is destroyed by invoking 1. onPause() 2. OnStop() and 3. onDestroy(). The lifetime of the activity becomes over.

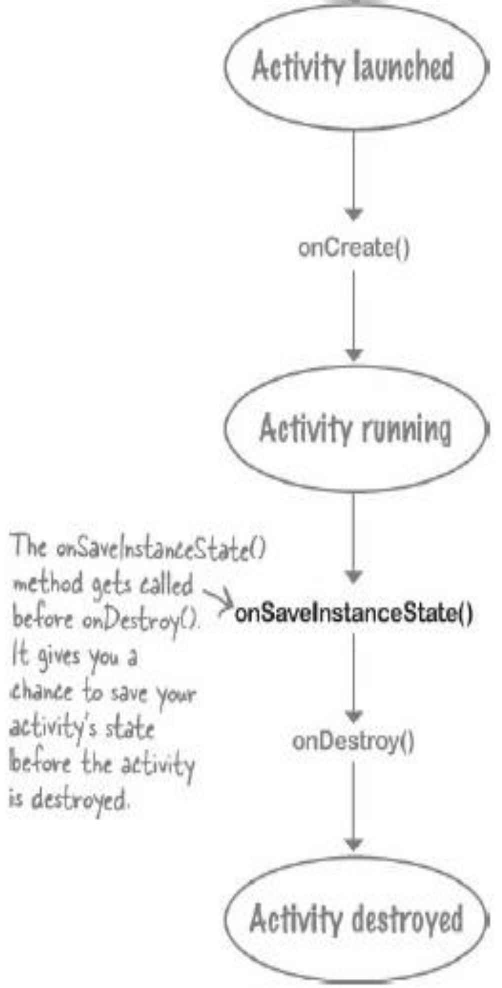


```
I/lifecycle: Method in OnPause
I/lifecycle: Method in OnStop
I/lifecycle: Method in OnDestroy
```

Refer : Lesson4/LifeCycleActivityDemo (Demo with single activity)
Refer : Lesson4/ActivityLifeCycleAPP (Demo with two activitities)

Saving instance state

- The better way of dealing with configuration changes (Rotate Device, changing the language settings) which you’ll use most often is to save the current state of the activity, and then reinstate it in the onCreate() method of the activity.
- To save the current state of the activity, you need to implement the onSaveInstanceState() method.
- The onSaveInstanceState() method gets called before the activity gets destroyed, which means you get an opportunity to save any values you want to retain before they get lost.
- The onSaveInstanceState() method takes one parameter, a Bundle. A Bundle allows you to gather different types of data into a single object
- The Bundle class provides a container for storing data using a key-value pair mechanism. The keys take the form of string values, while the values associated with those keys can be in the form of a primitive value or any object


Activity launched
onCreate()
Activity running
The onSaveInstanceState() method gets called before onDestroy(). It gives you a chance to save your activity's state before the activity is destroyed.
onSaveInstanceState()
onDestroy()
Activity destroyed

onSaveInstanceState(Bundle state)

```
//  To save the current state of the activity, you need to implement the onSaveInstanceState()
//method.
override fun onSaveInstanceState(outState: Bundle){
        super.onSaveInstanceState(outState)
        outState.putInt("count", mCount)
}
```

Refer : Lesson4\SaveStateDemo

To know more about Save State Information refer :
https://developer.android.com/topic/libraries/architecture/saving-states

Restoring instance state

Two ways to retrieve the saved Bundle

- in `onCreate(Bundle mySavedState)`
  Preferred, to ensure that your user interface, including any saved state, is back up and running as quickly as possible
- Implement callback (called after onStart())
  `onRestoreInstanceState(Bundle mySavedState)`

Restoring in onCreate()

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mTextViewCount = findViewById(R.id.text_view_count)
        if (savedInstanceState != null) {
            mCount = savedInstanceState.getInt("count")
            mTextViewCount?.text = mCount.toString()
        }
    }
```

Keyboard Shortcuts

Android Studio includes keyboard shortcuts for many common actions.

Refer : https://developer.android.com/studio/intro/keyboard-shortcuts

Main Point 3

During an Activity's runtime Lifecycle, an activity passes up and down through different states including onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart(), and onDestroy(). ***Science of Consciousness:*** *Similarly there are seven levels of consciousness, in the first three states waking consciousness; deep sleep and the dreaming state of sleep are known to every human being with a functional nervous system. The last four levels of consciousness; transcendental cosmic, god and unity consciousness are usually not available right away. These states become accessible only as one engages in regular practice of meditation.*

Summary

- The saving and restoration of dynamic state in an Android application is simply a matter of implementing the appropriate code in the appropriate lifecycle methods. For most user interface views, this is handled automatically by the Activity super class.
- In other instances, this typically consists of extracting values and settings within the *onSaveInstanceState()* method and saving the data as key-value pairs within the Bundle object passed through to the activity by the runtime system.
- State can be restored in either the *onCreate()* or the *onRestoreInstanceState()* methods of the activity by extracting values from the Bundle object and updating the activity based on the stored values.