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

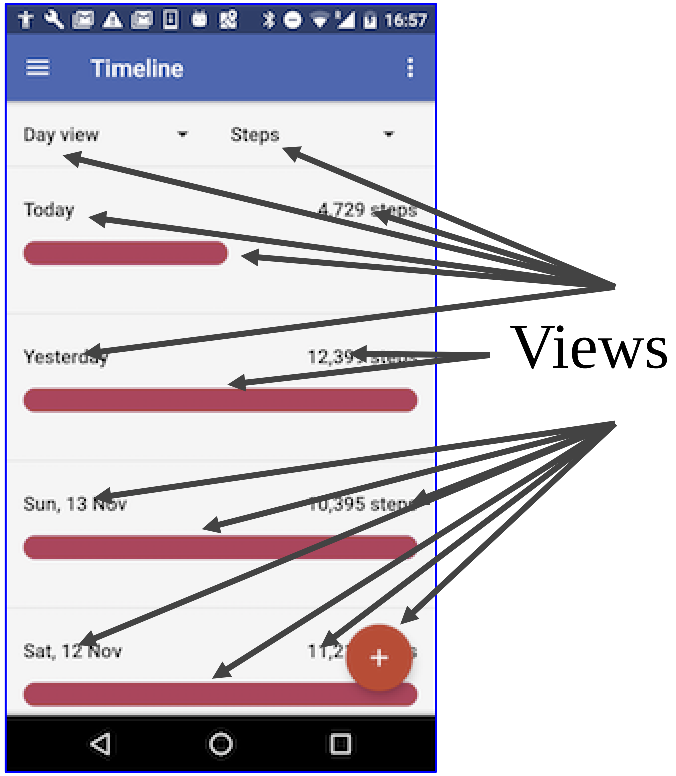
- In this lecture we examine additional fundamental building blocks of Android apps including the layouts, input/output controls, and buttons. We also explore resources and activity life cycle. App can build effectively with this fundamental knowledge. *The most fundamental knowledge is the most important 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

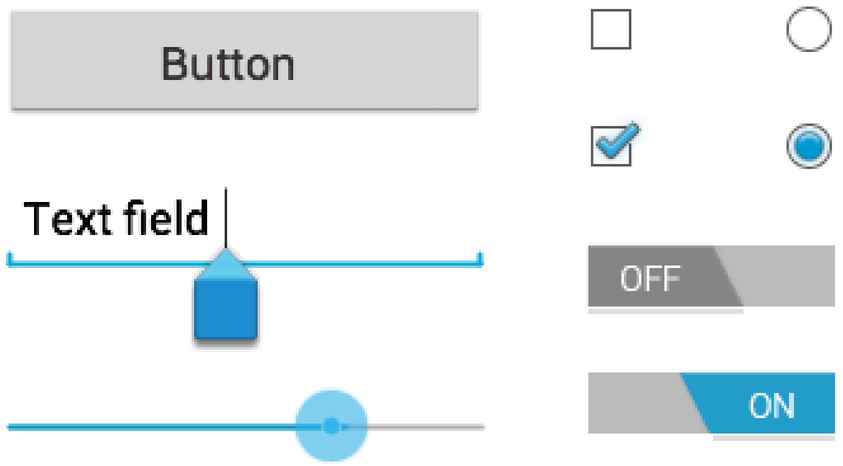
CheckBox RadioButton

Button

EditText

SeekBar

Toggle



Creating and laying out views

- Graphically using XML Design Editor
- XML Code Editor
- Programmatically
- Jetpack Compose (Not Covered)

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"

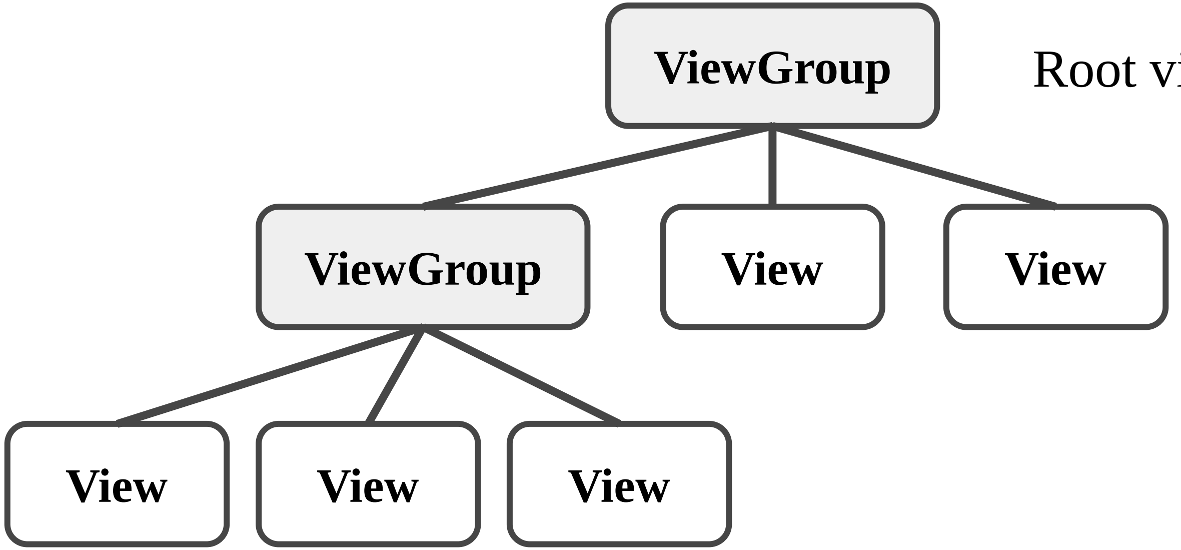
ViewGroup views

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

ViewGroup is the base class for layouts and view containers

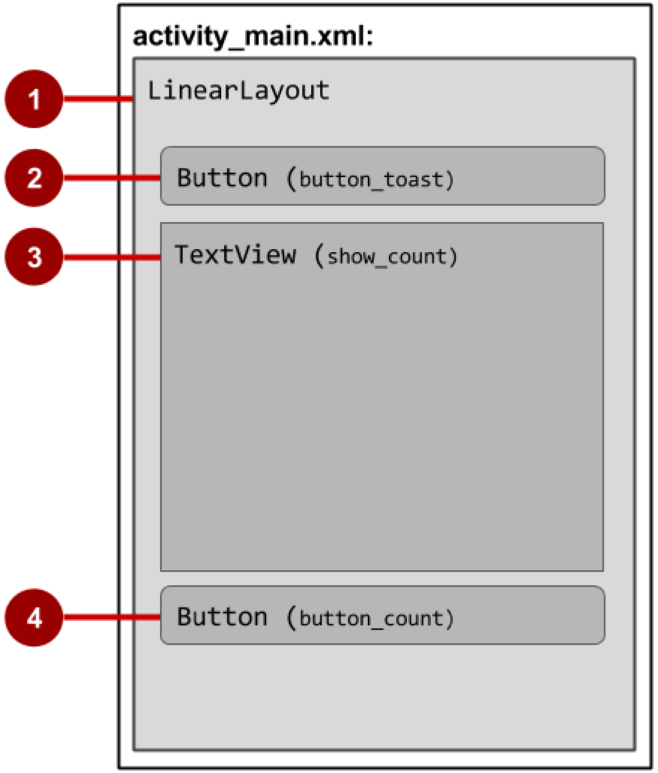
- ScrollView—scrollable view that contains one child view
- LinearLayout—arrange views in horizontal/vertical row
- RecyclerView—scrollable "list" of views or view groups

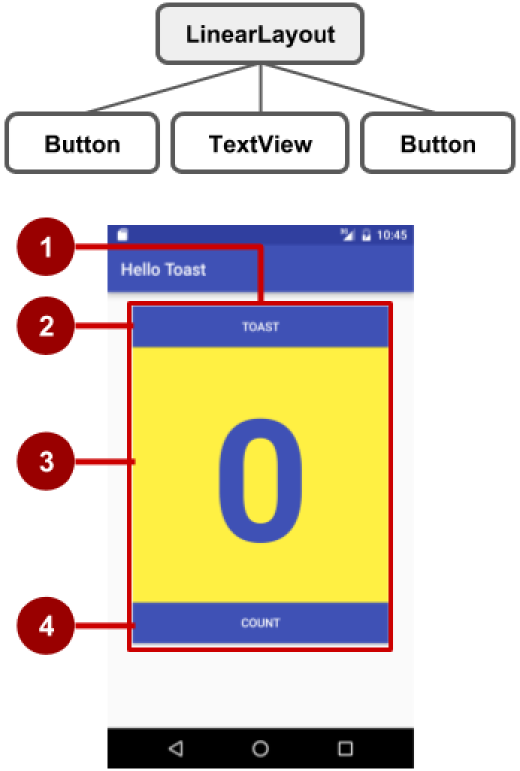
Hierarchy of view groups and views



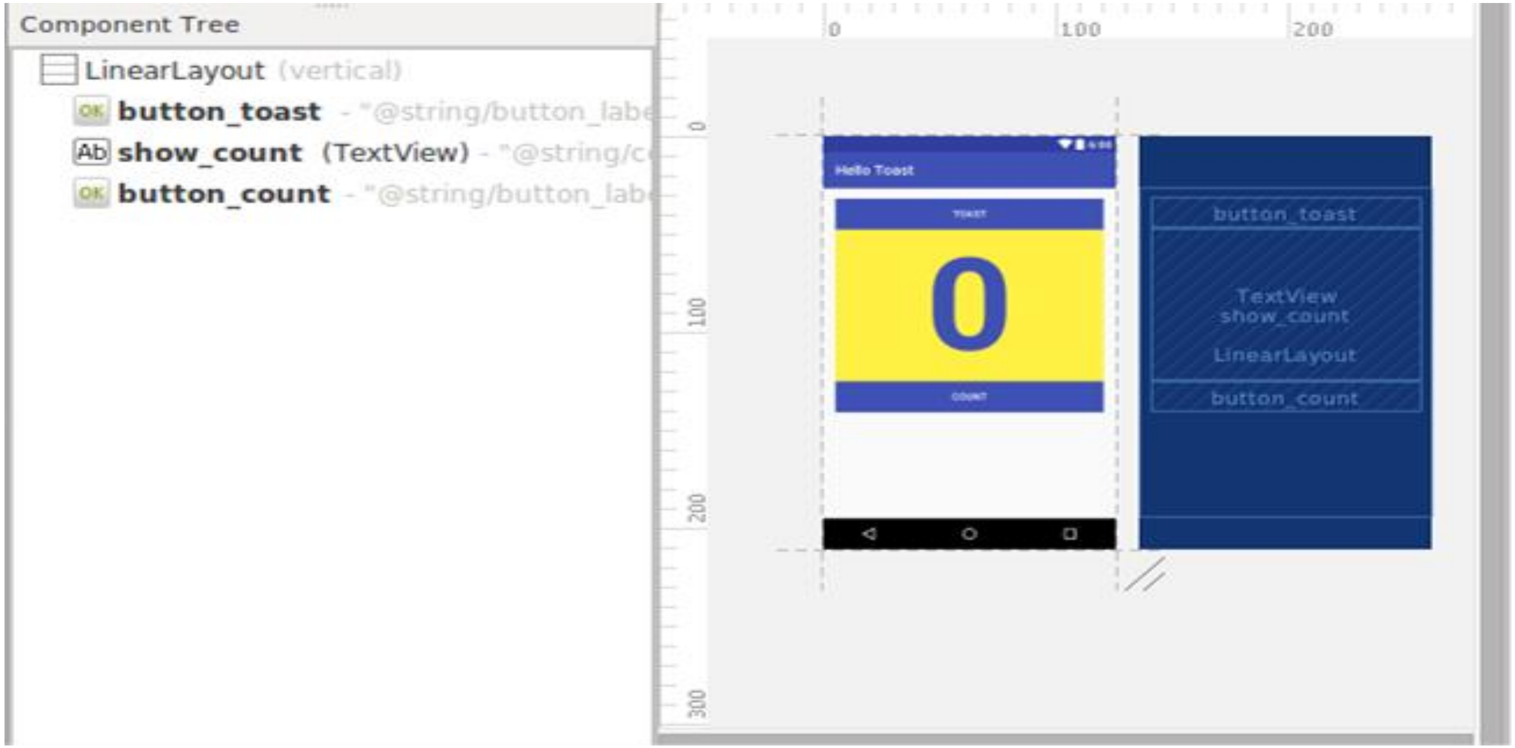
Root view is always a view group

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

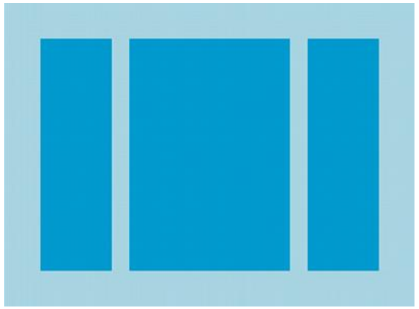
- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

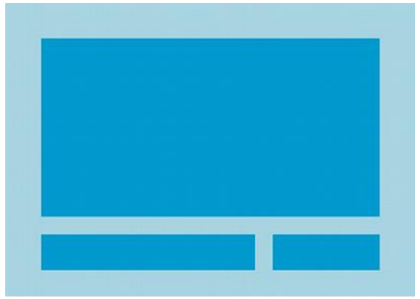
Layout Views

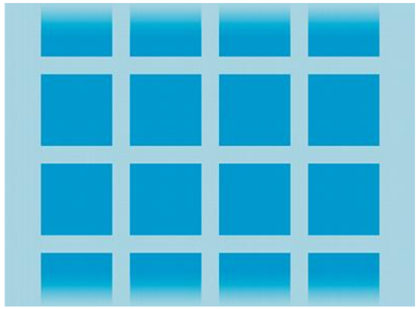
Layouts

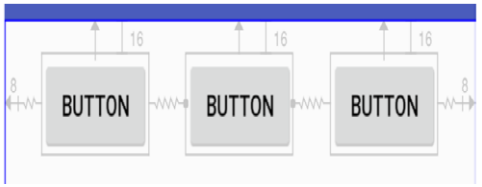
- are specific types of view groups
- are subclasses of ViewGroup
- contain child views
- can be in a row, column, grid, table, absolute

Common Layout Classes


LinearLayout


RelativeLayout


GridLayout



Constraint Layout

Common Layout Classes

- **LinearLayout** - horizontal or vertical row
- **RelativeLayout** - child views relative to each other
- **ConstraintLayout -** connect views with constraints
- **FrameLayout** - shows one child of a stack of children
- **GridView** - 2D scrollable grid

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
```

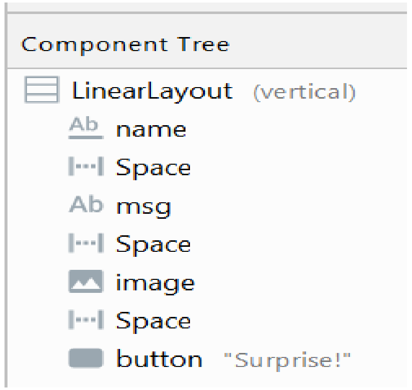
```
kotlin.Any
  ↳ android.view.View
      ↳ android.widget.TextView
          ↳ android.widget.Button
```

Java class-hierarchy

```
public class Button
extends TextView
```

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

Layout-hierarchy



```
Component Tree
LinearLayout (vertical)
    name
    Space
    msg
    Space
    image
    Space
    button "Surprise!"
```

- 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>
```

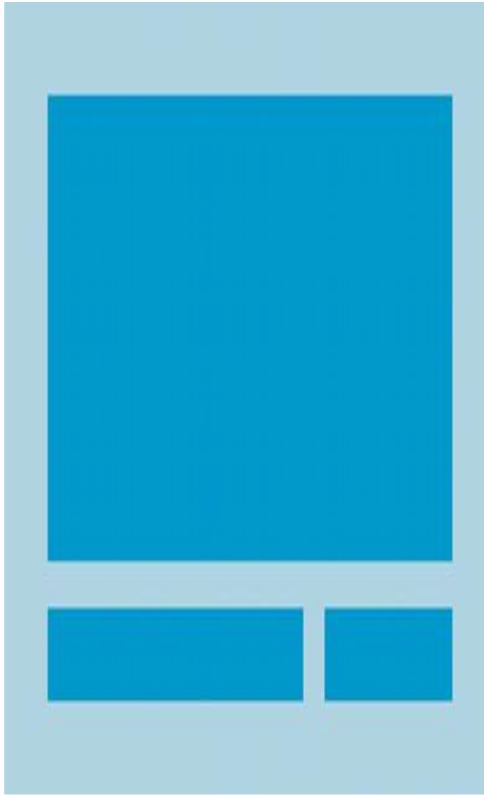
Here's an XML layout resource example of a `LinearLayout` set to the size of the screen, containing one `TextView` and a `Button` with the properties set to its height, width, text and id.

```
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

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.
  - It can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.



Constraint Layout

- ConstraintLayout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups).
- It's similar to RelativeLayout and more flexible
- All the power of ConstraintLayout is available directly from the Layout Editor's visual tools
- You can build your layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.

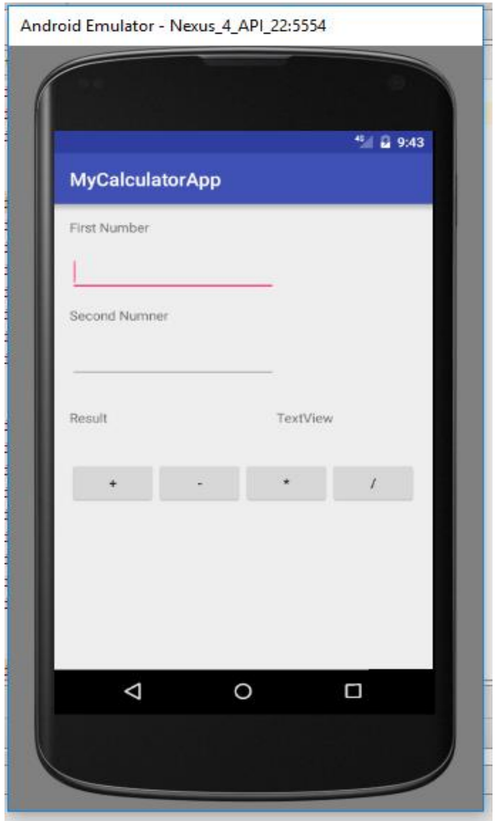
Hands on Example 1 – Simple Calculator using Relative and Linear Layouts

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 RelativeLayout and all buttons are combined using Linear Layout.

Refer Demo Code: Lesson4\CalculatorApp



Main Activity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import com.example.arcalculaterdemo.databinding.ActivityMainBinding

private lateinit var binding: ActivityMainBinding
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
```

Main Activity.kt

```
// click event implementation from xml  android:onClick="click"
  fun click(view: View) {
     // String s1 = et1.getString().toString()
     val num1 = binding.et1.text.toString() // to retrieve a text like getter - First Number
     val num2 = binding.et2.text.toString() // Second  Number
     when (view.id) { // Read the clicked Component --> view.getId()
        R.id.add -> {  // If View is add button
           val addition = num1.toInt() + num2.toInt()
         // Setting value to the Result text view like setter
           binding.tv4.text = addition.toString()
        }
```

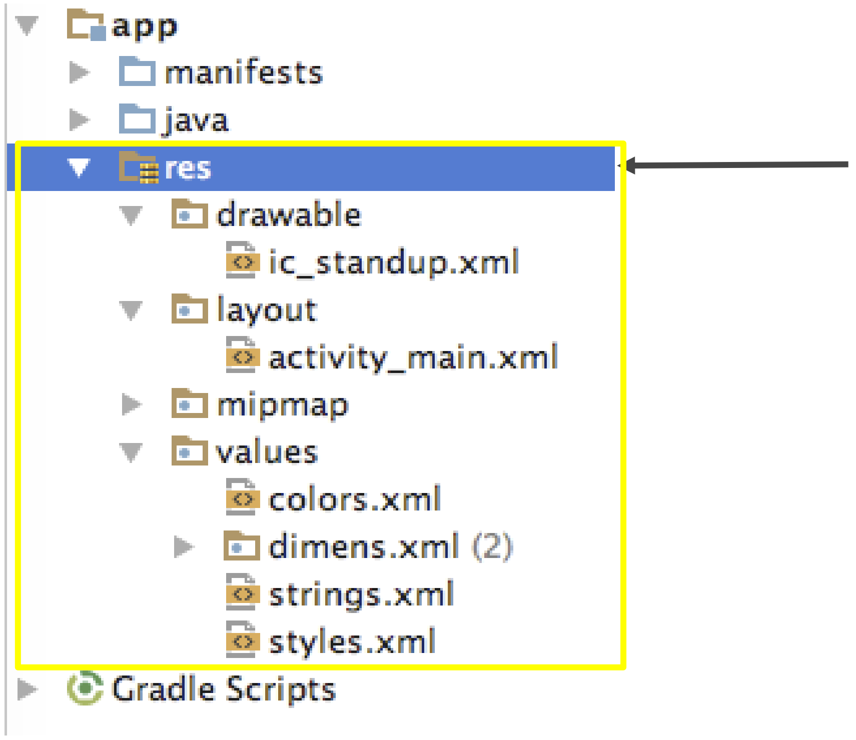
Main Activity.kt

```
        R.id.sub -> {   // If View is subtract button
            val minus = num1.toInt() - num2.toInt()
            binding.tv4.text = minus.toString()
         }
         R.id.mul -> {   // If View is multiplication button
            val mult = num1.toInt() * num2.toInt()
            binding.tv4. text = mult.toString()
         }
         R.id.div -> try { // If View is divide button
            val dvd = num1.toInt() / num2.toInt()
            binding.tv4.text = dvd.toString()
         }
         catch (e:ArithmeticException) {
            binding.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 2
TextView, ScrollView & Resources

- 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, ...

Hands on Example 2
TextView, ScrollView & Resources

- 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 level of nesting
  - Do not nest multiple scrolling views
  - Use HorizontalScrollView for horizontal scrolling
  - Use a RecyclerView for lists instead of ScrollViewto populate tabular data (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(scale-independent pixels)

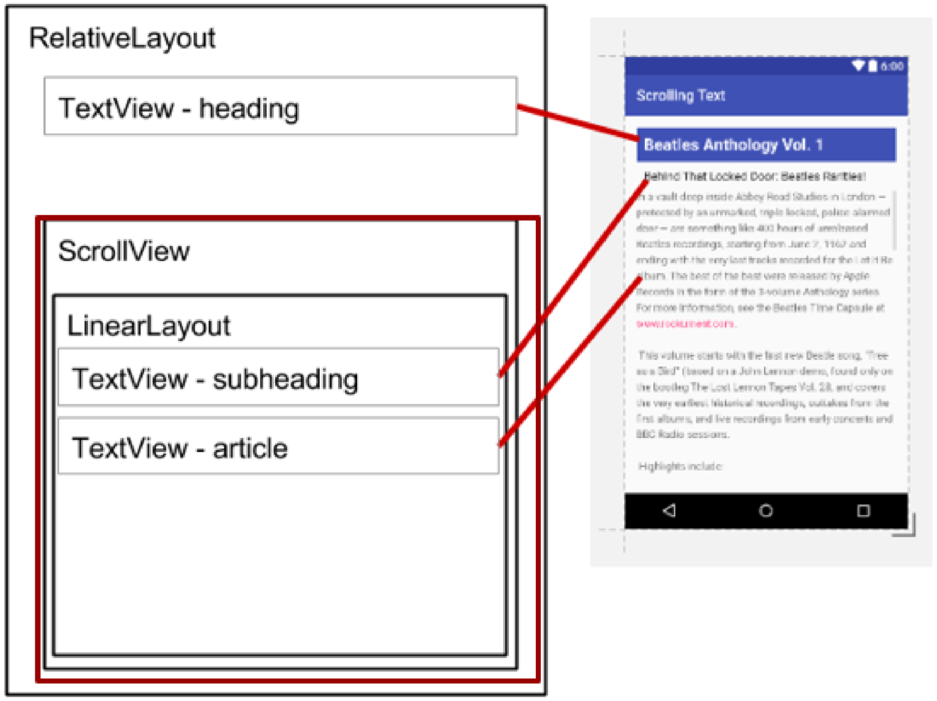
android:textStyle—normal, bold, italic, or bold|italic

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 7, 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:

Refer: ScrollViewApp

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)

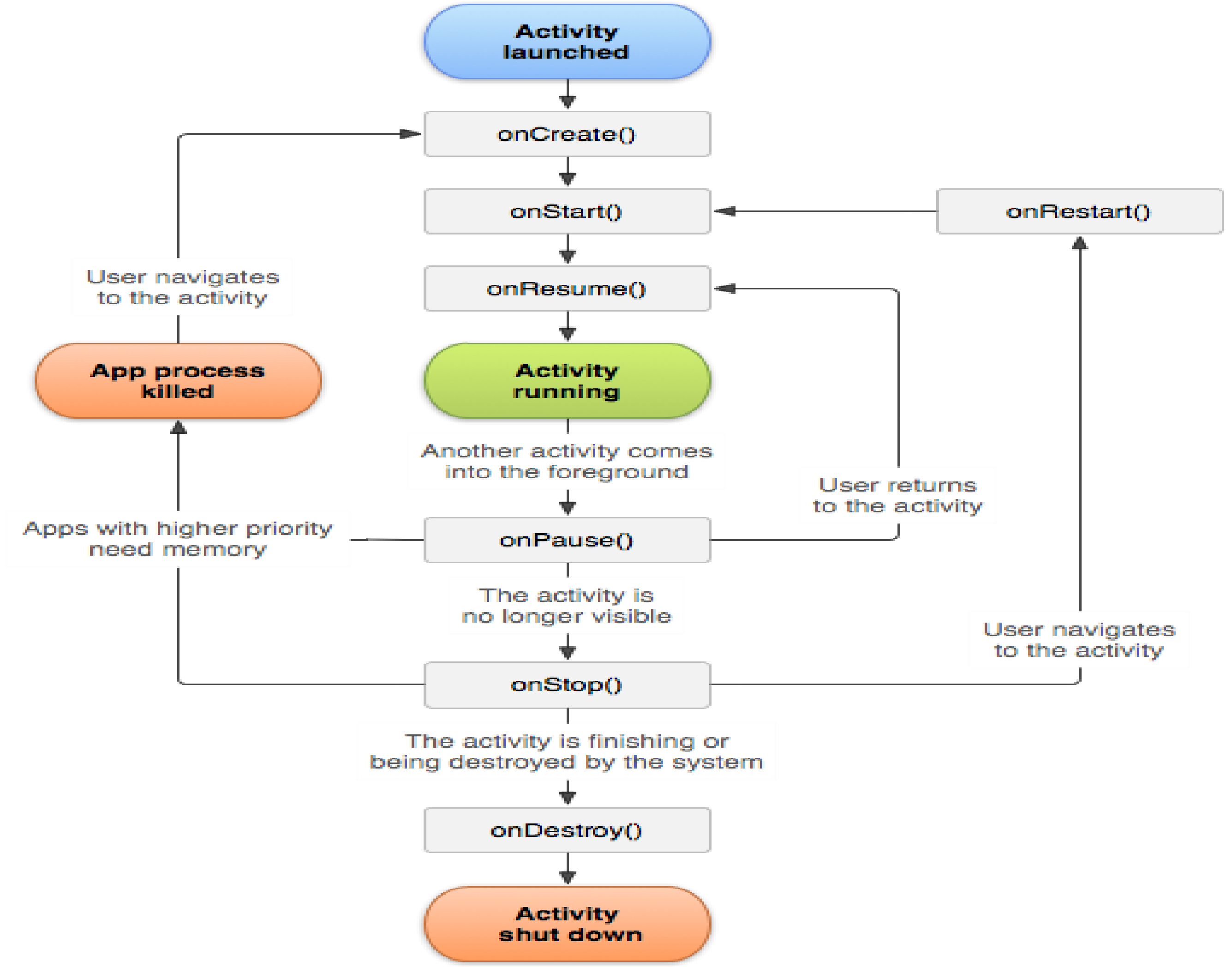
State changes are triggered by user action, configuration changes such as device rotation, or system action like changing language settings.

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


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

States of Activity Life cycle

1. Active State: [Activity in the Foreground/Running]
   - The activity is at the top of the Activity Stack, is the foreground task visible on the device screen, has focus and is currently interacting with the user.

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.
- Paused activities are held in memory, remain attached to the window manager, retain all state information and can quickly be restored to active status when moved to the top of the Activity Stack.

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 3: 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");
}
```

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")
}
```

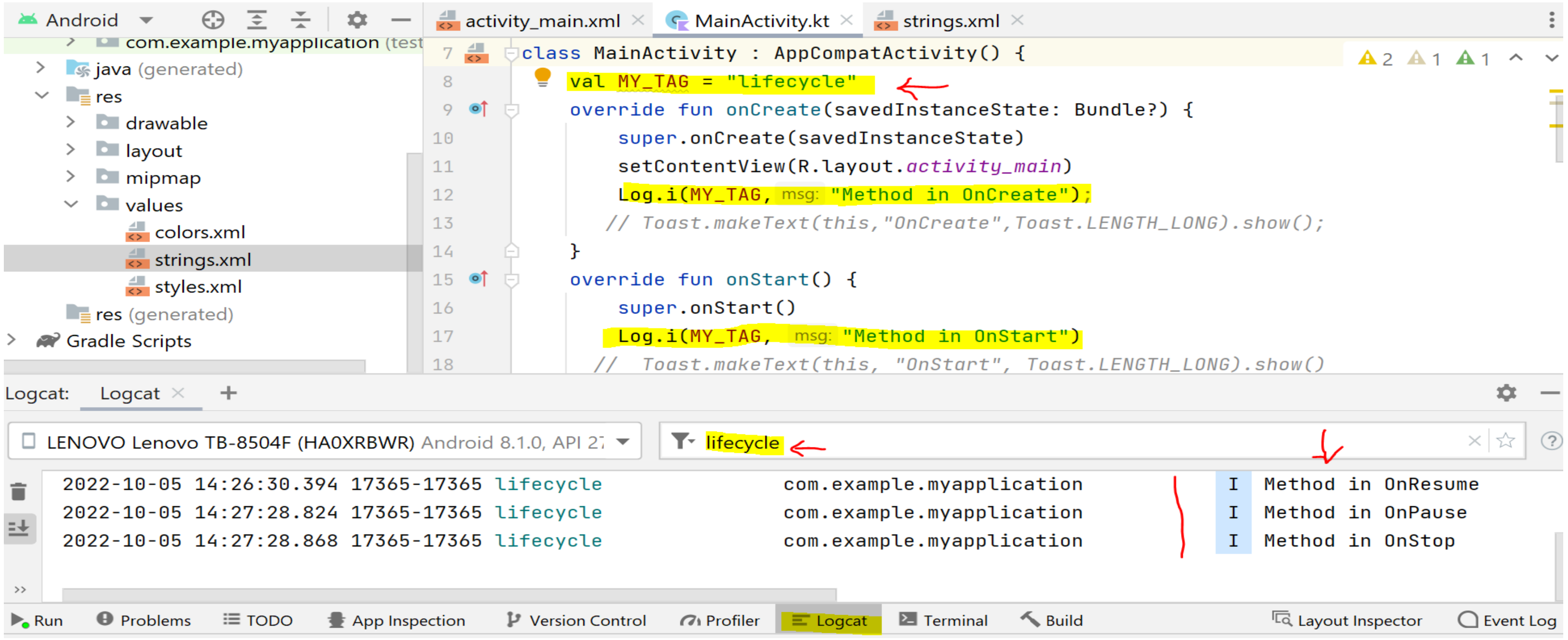
MainActivity.kt

```
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")
    }
}
```

How to See the Log.i(Message)

Click Logcat window at the bottom and type tag name or choose info on the list as highlights



```
class MainActivity : AppCompatActivity() {
    val MY_TAG = "lifecycle"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.i(MY_TAG, msg: "Method in OnCreate");
      // Toast.makeText(this,"OnCreate",Toast.LENGTH_LONG).show();
    }
    override fun onStart() {
        super.onStart()
        Log.i(MY_TAG,  msg: "Method in OnStart")
   //  Toast.makeText(this, "OnStart", Toast.LENGTH_LONG).show()
```

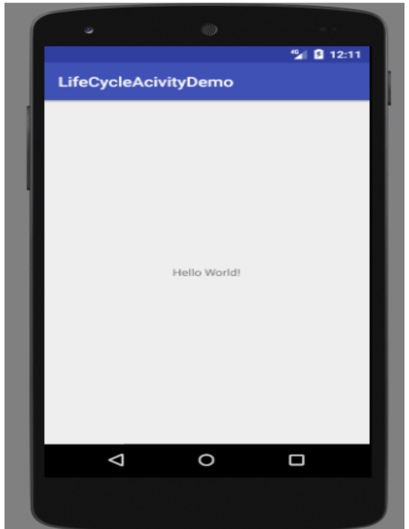
```
2022-10-05 14:26:30.394 17365-17365 lifecycle          com.example.myapplication          I  Method in OnResume
2022-10-05 14:27:28.824 17365-17365 lifecycle          com.example.myapplication          I  Method in OnPause
2022-10-05 14:27:28.868 17365-17365 lifecycle          com.example.myapplication          I  Method in OnStop
```

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.

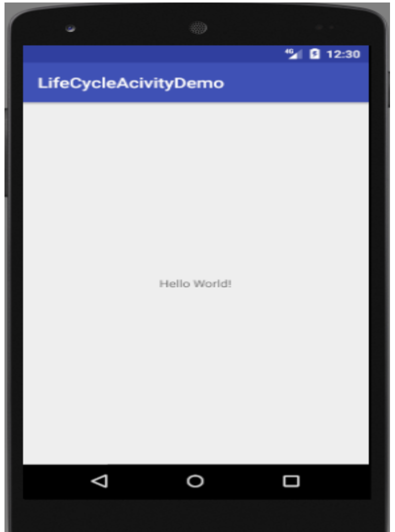
Android Emulator - Nexus_5_API_22:5554



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

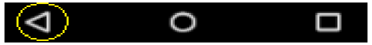
Activity Life Cycle – Screen Shots

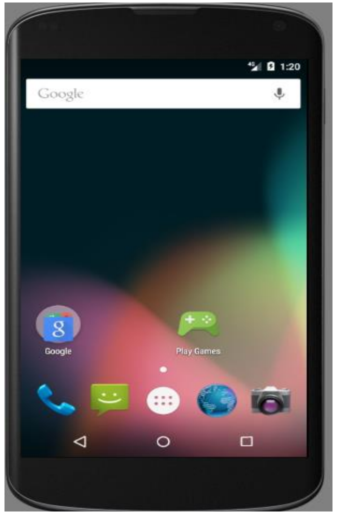
Screen 2 : Again start your application by clicking highlighted start button the from your Emulator and reload the same app. Now the Activity is restarted by invoking 1. OnRestart() 2. onStart() and 3. OnResume() itself. Now it is visible and not destroyed.



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

Activity Life Cycle – Screen Shots

Screen 3 : 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.

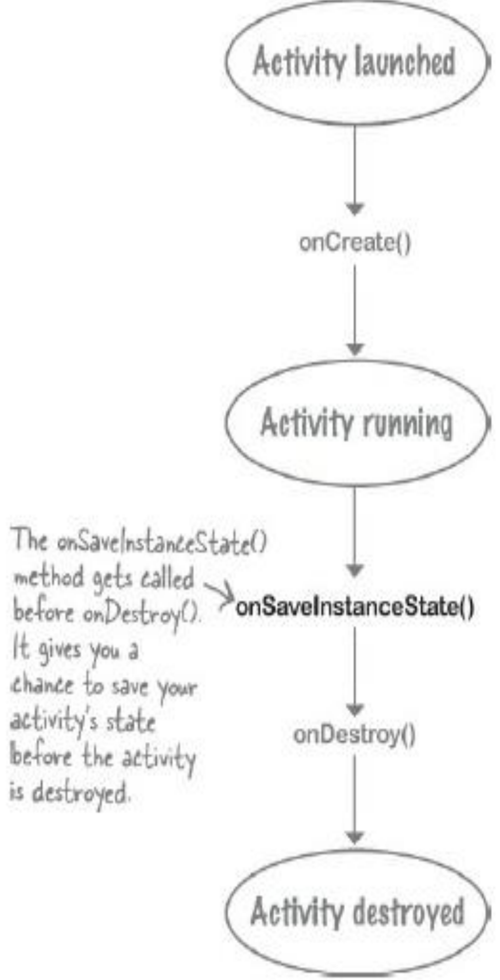


```
I/lifecycle: Method in OnPause
I/lifecycle: Method in OnStop
I/lifecycle: Method in OnDestroy
```

Refer : Lesson4/LifeCycleActivityDemo (Demo with single activity)

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 using onRestoreInstanceState()

```
override fun onRestoreInstanceState(savedInstanceState: Bundle?) {
        super.onRestoreInstanceState(savedInstanceState)
        if (savedInstanceState != null) {
            mCount = savedInstanceState.getInt("count")
            mTextViewCount?.text = mCount.toString()
         Toast.makeText(this,"OnReStoreInstanceState:"+"\n"+"mCount = " +
         mCount,Toast.LENGTH_LONG ).show()
        }
    }
```

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

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.

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.*

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Synthesis of parts for completeness of living

1. There are many ways to take Android user input just as Creative Intelligence synthesizes parts for completeness of living.
2. The many parts of an Android program function together in a cohesive whole as an Android app just as Creative Intelligence binds together delicate impulses of life.

---

3. **Transcendental Consciousness**: TC is the identity of each individual, located at the source of thought.
4. ***Impulses within the Transcendental field:*** *These impulses are perfectly balanced to create only the desired effect, no more and no less.*
5. ***Wholeness moving within Itself***: *In Unity Consciousness, one's unbounded identity is recognized to be the final truth about every object. All objects are seen to have the same ultimate identity, even though differences on the surface still remain.*